Documentation 6.0

ZABBIX

03.07.2025

Contents

Zabbix 使用手册 **6**
1. 简介 . . . 6
1 手册结构 . . . 6
2 什么是 Zabbix . . . 6
3 Zabbix 功能 . . . 7
4 Zabbix 概述 . . . 8
5 Zabbix 6.0.0 新功能 . . . 9
6 Zabbix 6.0.1 新功能 . . . 19
7 Zabbix 6.0.2 新功能 . . . 19
8 Zabbix 6.0.3 新功能 . . . 19
9 Zabbix 6.0.4 新功能 . . . 20
10 Zabbix 6.0.5 新功能 . . . 20
11 Zabbix 6.0.6 新功能 . . . 21
12 Zabbix 6.0.7 新功能 . . . 22
13 Zabbix 6.0.8 新功能 . . . 22
14 Zabbix 6.0.9 新功能 . . . 23
15 Zabbix 6.0.10 新功能 . . . 23
16 Zabbix 6.0.11 新功能 . . . 23
17 Zabbix 6.0.12 新功能 . . . 23
18 Zabbix 6.0.13 新功能 . . . 24
19 Zabbix 6.0.14 新功能 . . . 25
20 What's new in Zabbix 6.0.15 . . . 25
21 What's new in Zabbix 6.0.16 . . . 26
22 What's new in Zabbix 6.0.17 . . . 26
23 What's new in Zabbix 6.0.18 . . . 26
24 What's new in Zabbix 6.0.19 . . . 27
25 What's new in Zabbix 6.0.20 . . . 27
26 What's new in Zabbix 6.0.21 . . . 28
27 What's new in Zabbix 6.0.22 . . . 28
28 What's new in Zabbix 6.0.23 . . . 29
29 What's new in Zabbix 6.0.24 . . . 29
30 What's new in Zabbix 6.0.25 . . . 30
31 What's new in Zabbix 6.0.26 . . . 30
32 What's new in Zabbix 6.0.27 . . . 31
33 What's new in Zabbix 6.0.28 . . . 31
34 What's new in Zabbix 6.0.29 . . . 32
35 Zabbix 6.0.30 新功能 . . . 32
36 Zabbix 6.0.31 新功能 . . . 33
37 Zabbix 6.0.32 新功能 . . . 33
38 What's new in Zabbix 6.0.33 . . . 33
39 What's new in Zabbix 6.0.34 . . . 34
40 What's new in Zabbix 6.0.35 . . . 34
41 What's new in Zabbix 6.0.36 . . . 35
42 What's new in Zabbix 6.0.37 . . . 35
43 What's new in Zabbix 6.0.38 . . . 35
44 What's new in Zabbix 6.0.39 . . . 35
45 What's new in Zabbix 6.0.40 . . . 36
46 What's new in Zabbix 6.0.41 . . . 36
2. 定义 . . . 36
3. Zabbix 进程 . . . 37

1 Server . . . 37
2 Agent . . . 44
3 Agent 2 . . . 47
4 Proxy . . . 49
5 Java gateway . . . 51
6 Sender . . . 55
7 Get . . . 56
8 JS . . . 56
9 Web service . . . 57
4. 安装 . . . 57
1 获取 Zabbix . . . 57
2 安装要求 . . . 58
3 从源代码安装 . . . 72
4 从二进制包安装 . . . 81
5 从容器中安装 . . . 96
6 Web 界面安装 . . . 117
7 升级步骤 . . . 124
8 已知问题 . . . 135
9 模版变更 . . . 142
10 6.0.0 更新说明 . . . 146
11 6.0.1 更新说明 . . . 148
12 6.0.2 更新说明 . . . 148
13 6.0.3 更新说明 . . . 148
14 6.0.4 更新说明 . . . 148
15 6.0.5 更新说明 . . . 148
16 6.0.6 更新说明 . . . 148
17 6.0.7 升级说明 . . . 149
18 6.0.8 更新说明 . . . 149
19 6.0.9 升级说明 . . . 149
20 6.0.10 升级说明 . . . 149
21 6.0.11 升级说明 . . . 149
22 6.0.12 升级说明 . . . 150
23 6.0.13 升级说明 . . . 150
24 Upgrade notes for 6.0.14 . . . 150
25 Upgrade notes for 6.0.15 . . . 150
26 Upgrade notes for 6.0.16 . . . 150
27 Upgrade notes for 6.0.17 . . . 151
28 Upgrade notes for 6.0.18 . . . 151
29 Upgrade notes for 6.0.19 . . . 151
30 Upgrade notes for 6.0.20 . . . 151
31 Upgrade notes for 6.0.21 . . . 151
32 Upgrade notes for 6.0.22 . . . 151
33 Upgrade notes for 6.0.23 . . . 152
34 Upgrade notes for 6.0.24 . . . 152
35 Upgrade notes for 6.0.25 . . . 152
36 Upgrade notes for 6.0.26 . . . 152
37 Upgrade notes for 6.0.27 . . . 152
38 Upgrade notes for 6.0.28 . . . 153
39 Upgrade notes for 6.0.29 . . . 153
40 6.0.30 版本升级说明 . . . 153
41 6.0.31 版本升级说明 . . . 153
42 Upgrade notes for 6.0.32 . . . 153
43 Upgrade notes for 6.0.33 . . . 153
44 Upgrade notes for 6.0.34 . . . 153
45 Upgrade notes for 6.0.35 . . . 154
46 Upgrade notes for 6.0.36 . . . 154
47 Upgrade notes for 6.0.37 . . . 154
48 Upgrade notes for 6.0.38 . . . 154
49 Upgrade notes for 6.0.39 . . . 154
50 Upgrade notes for 6.0.40 . . . 154
51 Upgrade notes for 6.0.41 . . . 155
5. 快速入门 . . . 155
1 登录和配置用户 . . . 155

2 新建主机 . . . 158
3 新增监控项 . . . 160
4 新建触发器 . . . 161
5 接收问题通知 . . . 163
6 新建模板 . . . 167
6. Zabbix Appliance . . . 169
7. 配置 . . . 172
1 主机和主机组 . . . 178
2 监控项 . . . 189
3 触发器 . . . 389
4 事件 . . . 418
5 事件关联 . . . 421
6 标记 . . . 427
7 可视化 . . . 430
8 模板 . . . 455
9 开箱即用的模板 . . . 455
10 事件通知 . . . 477
11 宏变量 . . . 519
12 用户和用户组 . . . 529
13 存储密钥 . . . 535
14 定时报表 . . . 537
8. 服务监控 . . . 540
1 服务树 . . . 541
2 服务动作 . . . 544
3 服务级别协议 SLA . . . 545
4 配置示例 . . . 547
9. Web 监控 . . . 552
1 Web 监控项 . . . 560
2 真实场景 . . . 562
10. 虚拟机监控 . . . 570
1 虚拟机自动发现相关键值字段 . . . 575
11. 维护期 . . . 578
12. 正则表达式 . . . 582
13 问题确认 . . . 587
14. 配置导出/导入 . . . 589
1 主机群组 . . . 590
2 模板 . . . 591
3 主机 . . . 612
4 网络拓扑图 . . . 631
5 媒介类型 . . . 640
15. 发现 . . . 648
1 网络发现 . . . 648
2 agent（主动模式）自动注册 . . . 656
3 底层自动发现 . . . 659
16. 分布式监控 . . . 706
1 proxy 代理 . . . 706
17. 加密 . . . 709
1 使用证书 . . . 716
2 使用预共享秘钥 . . . 724
3 故障排除 . . . 725
18. Web 界面 . . . 728
1 菜单 . . . 728
2 前端部件 . . . 730
3 用户设置 . . . 866
4 全局搜索 . . . 870
5 前端维护模式 . . . 872
6 页面参数 . . . 873
7 定义 . . . 874
8 定制主题风格 . . . 874
9 调试模式 . . . 875
10 Zabbix 使用的 cookie . . . 876
11 时区 . . . 876
13 Resetting password . . . 877

19. API . . . 877
方法参考 . . . 882
Zabbix API 在 6.0 中的变化 . . . 1477
附录 1. 参考说明 . . . 1479
附录 2. 从 5.4 到 6.0 的变更记录 . . . 1485
20. 模块 . . . 1488
21. 附录 . . . 1494
2 安装及配置 . . . 1494
3 后台进程配置 . . . 1532
4 协议 . . . 1589
5 监控项 . . . 1613
6 支持的函数 . . . 1644
7 宏 . . . 1678
8 单位符号 . . . 1704
9 时间段语法 . . . 1706
10 命令执行 . . . 1706
11 版本兼容性 . . . 1707
12 数据库错误处理 . . . 1708
13 Zabbix sender Windows 动态链接库 . . . 1708
14 Python library for Zabbix API . . . 1709
14 服务监控升级 . . . 1709
15 其他问题 . . . 1709
18 转义示例 . . . 1710
Zabbix agent 与 Zabbix agent 2 的比较 . . . 1712

Zabbix 手册页 **1713**
zabbix_agent2 . . . 1713
名字 . . . 1713
概要 . . . 1713
描述 . . . 1713
选项 . . . 1713
文件 . . . 1714
另请参见 . . . 1714
索引 . . . 1715
zabbix_agentd . . . 1715
名字 . . . 1715
概要 . . . 1715
描述 . . . 1715
选项 . . . 1715
文件 . . . 1716
另请参见 . . . 1716
索引 . . . 1716
zabbix_get . . . 1717
名字 . . . 1717
概要 . . . 1717
描述 . . . 1717
选项 . . . 1717
示例 . . . 1718
另请参见 . . . 1718
索引 . . . 1718
zabbix_js . . . 1718
名字 . . . 1719
概要 . . . 1719
描述 . . . 1719
OPTIONS . . . 1719
示例 . . . 1719
另请参见 . . . 1719
索引 . . . 1719
zabbix_proxy . . . 1720
名字 . . . 1720
概要 . . . 1720
描述 . . . 1720
OPTIONS . . . 1720

文件 . . . 1721
另请参见 . . . 1721
索引 . . . 1721
zabbix_sender . . . 1721
名字 . . . 1721
概要 . . . 1721
描述 . . . 1722
选项 . . . 1722
退出状态 . . . 1724
示例 . . . 1724
索引 . . . 1725
zabbix_server . . . 1725
名字 . . . 1725
概要 . . . 1725
描述 . . . 1725
选项 . . . 1725
文件 . . . 1726
另请参见 . . . 1726
索引 . . . 1727
zabbix_web_service . . . 1727
名称 . . . 1727
概要 . . . 1727
描述 . . . 1727
选项 . . . 1727
文件 . . . 1727
另请参见 . . . 1727
索引 . . . 1727

Zabbix 使用手册

欢迎查阅 Zabbix 用户使用手册。

Zabbix 产品手册由原厂 Zabbix 技术团队创建、Zabbix 中国——上海宏时数据系统有限公司组织开源社区志愿者翻译并维护。希望能帮助用户更好地使用 Zabbix，解决和管理日常 IT 运维监控遇到的各种问题。翻译虽然结束，优化并未停止，如有优化反馈、申请成为译者，欢迎联系小 Z 17502189550，market@grandage.cn。

版权声明

Zabbix 文档不在 GPL 许可下分发。利用 Zabbix 文档受以下条款约束：

您可以为了个人使用需要打印文档副本。在未以任何方式更改或编辑实际内容的前提下，允许转换成其他格式。你应该不得以任何形式或在任何媒体上传播此文档，除非您以类似 Zabbix 的方式传播文档（即以电子方式在 Zabbix 网站上下载）或 USB 或类似媒介上，但前提是该文档与软件在相同的媒介一起传播。任何其他用途，例如复制或使用该文档，使用全部或部分文档进行出版，需要事先获得 Zabbix 书面授权同意。Zabbix 对此文档以上未明确授权的所有文件保留所有权。

1. 简介

请使用侧边栏访问简介部分中的内容。

1 手册结构

结构

本手册的内容分为多个部分和小节，以便您轻松访问感兴趣的特定主题。

当您导航到各个部分时，请确保展开部分文件夹以显示子文件和各个页面中包含的全部内容。

手册尽可能提供相关内容页面之间的交叉链接，以确保用户不会错过相关信息。

章节

简介 提供有关当前 Zabbix 软件的一般信息。阅读本节应该为您提供选择 Zabbix 的一些充分理由。

Zabbix 概念 解释了 Zabbix 中使用的术语，并提供了 Zabbix 组件的详细信息。

安装 和快速入门 部分应该可以帮助您开始使用 Zabbix。

Zabbix 应用 是一种快速体验 Zabbix 使用的替代方案。

配置 是本手册中最大和最重要的部分之一。它包含大量关于如何设置 Zabbix 以监控你的环境的基本建议，从设置主机到获取基本数据到查看数据到配置通知和远程命令以在出现问题时执行。

IT 服务 部分详细介绍了如何使用 Zabbix 对监控环境进行高级概述。

Web 监控 可以帮助您了解如何监控网站的可用性。

虚拟机监控 提供了配置 VMware 环境监控的方法。

维护 ，正则表达式，事件确认 和XML 导入/导出 部分进一步说明如何使用 Zabbix 软件的各个方面。

发现 包含有关设置网络设备、主动模式的 agent（自动注册）、文件系统、网络接口等的自动发现的说明。

分布式监控 介绍在更大、更复杂的环境中使用 Zabbix 的可能性。

加密 帮助解释加密 Zabbix 组件之间的通信的可行性。

Web 界面 包含特定的使用 Zabbix 的 Web 界面的信息。

API 部分介绍了使用 Zabbix API 的详细信息。

可以通过附录 查看更详细的记述信息。还可以在此处找到常见问题解答部分。

2 什么是 **Zabbix**

概述

Zabbix 由 Alexei Vladishev 创建，目前由 Zabbix SIA 主导开发和支持。

Zabbix 是一个企业级的开源分布式监控解决方案。

Zabbix 是一款监控网络的众多参数以及服务器、虚拟机、应用程序、服务、数据库、网站、云等的健康和完整性的软件。Zabbix 使用灵活的通知机制，允许用户为几乎任何事件配置基于电子邮件的告警，以实现对服务器问题做出快速反应。Zabbix 基于存储的数据提供出色的报告和数据可视化功能。这使得 Zabbix 成为容量规划的理想选择。

Zabbix 支持轮询和 trapping。所有 Zabbix 报告和统计数据以及配置参数都可以通过基于 Web 的前端访问。基于 Web 的前端确保可以从任何位置评估您的网络状态和服务器的健康状况。如果配置得当，不管对于拥有少量服务器的小型组织还是拥有大量服务器的大公司来讲，Zabbix 都可以在监控 IT 基础设施方面发挥重要作用。

Zabbix 是免费的。Zabbix 是在 GPL 通用公共许可证第 2 版下编写和分发的。这意味着它的源代码是免费分发的，可供公众使用。

商业支持 由 Zabbix 公司及其世界各地的合作伙伴提供。

了解更多Zabbix 功能。

Zabbix 用户

世界各地许多不同规模的组织都依赖 Zabbix 作为主要监控平台。

3 Zabbix 功能

概述

Zabbix 是一个高度集成的网络监控解决方案，在一个软件包中提供了多种功能。

数据收集

- 可用性和性能检查
- 支持 SNMP（trapping 和 polling）、IPMI、JMX、VMware 监控
- 自定义检查
- 以自定义间隔收集所需数据
- 由 server/proxy 和 agents 执行

灵活的阈值定义

- 可以定义非常灵活的问题阈值，称为触发器，从后端数据库引用值

高度可配置的告警

- 可以针对升级计划、收件人、媒体类型自定义发送通知
- 使用宏可以使通知变得有意义和有用
- 自动化操作包括执行远程命令

实时图形

- 采集到的监控项值可以使用内置的绘图功能立即绘图

网络监控功能

- Zabbix 可以跟踪网站上的模拟鼠标点击路径并检查功能和响应时间

广泛的可视化选项

- 创建自定义图形的能力，可以将多个监控项组合成一个聚合图形
- 网络拓扑图
- 在仪表盘中显示幻灯片
- 报表
- 受监控资源的高级（业务）视图

历史数据存储

- 存储在数据库中的数据
- 可配置的历史（保留趋势）
- 内置管家程序

建议的配置

- 将受监控的设备添加为主机
- 一旦主机被数据库添加，就会开始进行数据采集
- 将模板应用于受监控的设备

模板的使用

- 在模板中分组检查
- 模板可以继承其他模板

网络发现

- 网络设备自动发现
- agent 自动注册
- 发现文件系统、网络接口和 SNMP OID

便捷的 **web** 界面

- 基于 web 的 PHP 前端
- 可从任何地方访问
- 可以通过你的方式点击（到任何页面）
- 审计日志

Zabbix API

- Zabbix API 为 Zabbix 提供可编程接口，用于大规模操作、第 3 方软件集成和其他用途。

权限系统

- 安全用户认证
- 某些用户可以被限制仅访问某些视图

全功能且易于扩展的 **agent**

- 部署在被监控目标上
- Linux 和 Windows 操作系统都适用于

二进制守护进程

- 用 C 编写，用于提高性能和减少内存占用
- 轻量级、便携

为复杂环境做好准备

- 使用 Zabbix proxy 轻松实现远程监控

4 Zabbix 概述

结构体系

Zabbix 由几个主要的软件组件组成。他们的职责概述如下。

Server

Zabbix server 是 agents 向其报告可用性和完整性信息和统计信息的中心组件。server 是存储所有配置、统计和操作数据的中央存储库。

数据存储

Zabbix 收集的所有配置信息以及数据都存储在数据库中。

Web 界面

为了从任何地方和任何平台轻松访问，Zabbix 提供了基于 Web 的界面。该接口是 Zabbix server 的一部分，通常（但不一定）与 server 运行在同一台设备上。

Proxy

Zabbix proxy 可以代替 Zabbix server 收集性能和可用性数据。proxy 是 Zabbix 部署的可选部分；但是对于分散单个 Zabbix server 的负载非常有用。

Agent

Zabbix agent 部署在被监控目标上，以主动监控本地资源和应用程序，并将收集到的数据报告给 Zabbix server。从 Zabbix 4.4 开始，有两种类型的 agent 可用：Zabbix agent （轻量级，在许多平台上支持，用 C 编写）和Zabbix agent 2 （非常灵活，易于使用插件扩展，用 Go 编写）。

数据流

此外，回顾一下 Zabbix 中的整体数据流也是很重要的。为了创建一个收集数据的监控项，必须首先创建一个主机。另一方面 Zabbix 必须首先拥有一个监控项来创建触发器。必须有触发器才能创建动作。因此，如果你想收到 服务器 *X* 上的 CPU 负载过高的警报，必须首先为

服务器 *X* 创建一个主机条目，然后创建一个用于监控其 CPU 的监控项，然后是一个触发器，如果 CPU 过高则触发动作，然后通过通过动作操作向您发送电子邮件。这可能看起来像很多步骤，其实使用模板并不需要。而且，由于这种设计，可以自定义创建非常灵活的设置。

5 Zabbix 6.0.0 新功能

Zabbix server 的高可用集群　新版本附带了针对 Zabbix server 的原生高可用解决方案。

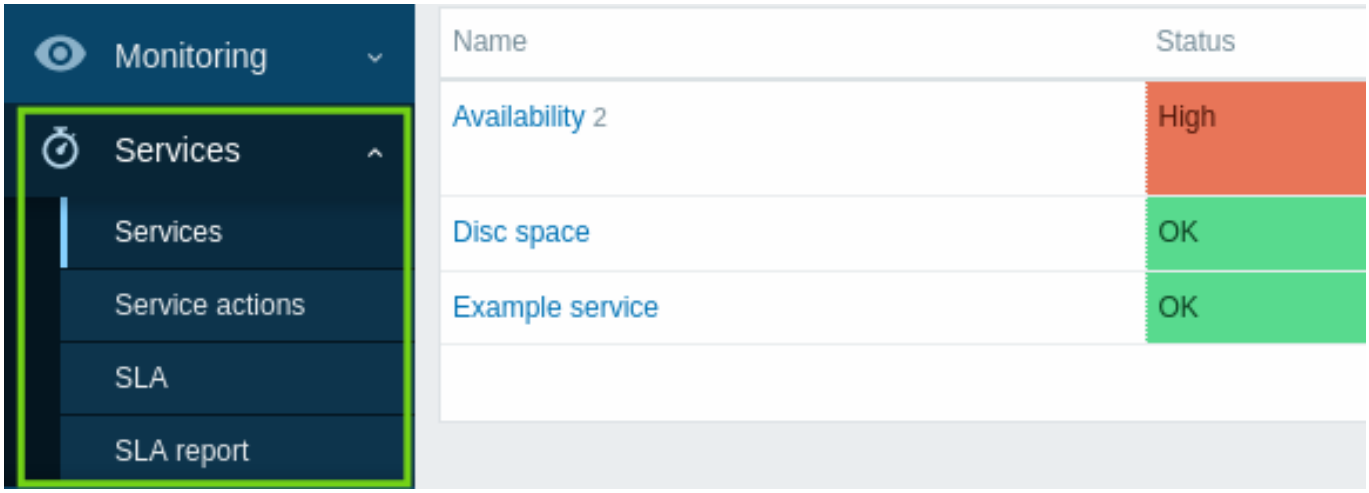
该解决方案由多个 zabbix_server 实例或节点组成，其中一次只能有一个节点处于活动状态（工作），而其他节点处于待机状态，准备好在当前节点停止或故障时接管。

另请参阅：高可用集群。

服务　对服务的监控进行了一些更新。服务监控提供了 Zabbix 中受监控基础设施的高级视图。

Zabbix 现在有一个新的服务菜单，包含四个菜单组件：

- 服务（Services） - 用于服务概述和服务配置（从 Monitoring 移动到 Services）
- 服务动作（Service actions） - 用于服务动作（新操作类型）
- SLA - 用于配置 SLA
- SLA 报表 - 用于 SLA 报表（也可用作仪表板小部件）



下面概述了对服务功能的其他主要改进。

基于标签的服务到问题的映射

以前版本中服务（services） 的可用性取决于触发器及其状态。在新版本中，它被相应服务的基于标签的问题映射所取代。

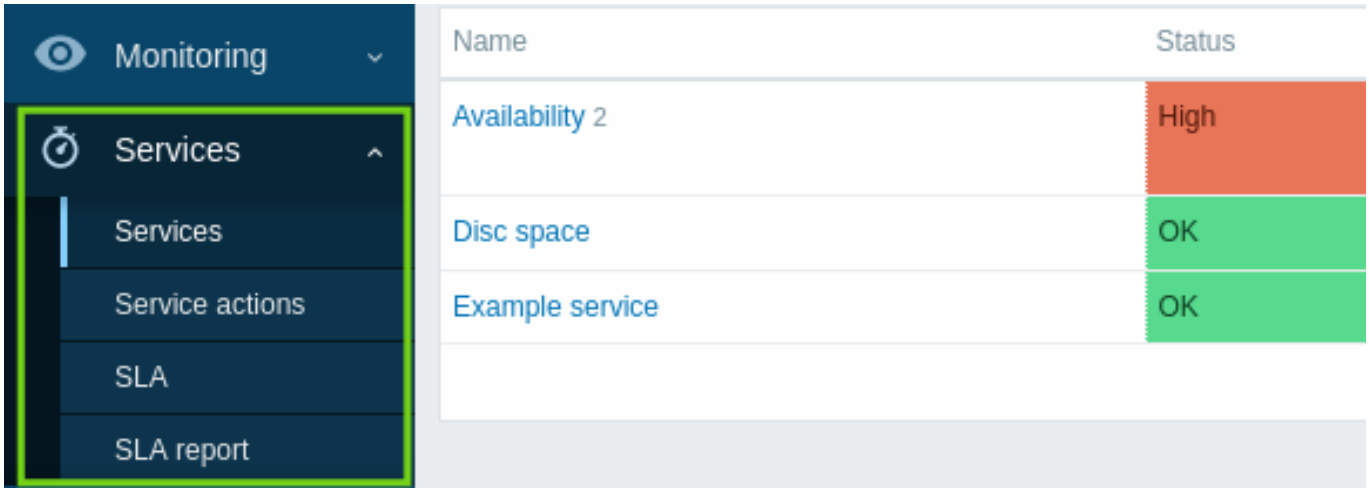
服务的配置和查看合并在 *Monitoring* → *Services* 中，并且在 *Configuration* → *Services* 中不再存在用于服务配置的单独部分。

在服务配置中，不再存在硬依赖和软依赖。相反，一个服务可以有多个父服务。

Services menu

There is now a new Services menu in Zabbix, with four menu sections:

- Services - for service overview and service configuration (moved from *Monitoring -> Services*)
- Service actions - for service actions (new action type)
- SLA - for configuring SLAs
- SLA report - for SLA reports (also available as dashboard widget)



状态计算和传播规则

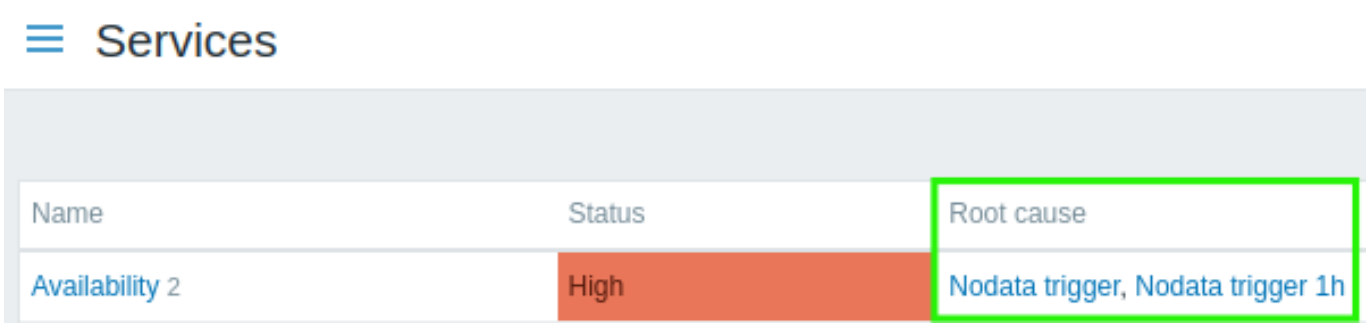
有新的状态计算规则和灵活的附加规则，用于基于直接子服务的状态和权重计算父服务的状态。现在还可以设置灵活的规则来将服务状态传播到父服务。

服务权限

在用户角色 级别实现了对服务的灵活权限。可以向所有、无或选定的服务授予读写或只读访问权限（基于名称或标签）。

根本原因分析

一个新的 根本原因（*Root cause*）列显示了直接或间接影响服务状态的潜在问题。



如果您单击问题名称，您可以在 *Monitoring* → *Problems* 中查看有关它的更多详细信息。

服务状态变化告警

现在可以接收有关服务状态更改的自动警报，类似于有关触发器状态更改的警报。

添加了一种新的服务动作 类型，类似于 Zabbix 中的其他动作。服务可能动作包括与服务相关的问题、恢复和更新动作的步骤。可以配置两种类型的动作：向指定的收件人发送消息和在 Zabbix server 上执行远程命令。与触发动作类似，服务动作支持问题升级场景。

新的消息模板 *Service*、*Service recovery* 和 *Service update* 已添加到媒体类型 中，应该定义为能够正确发送服务动作的通知。

服务克隆

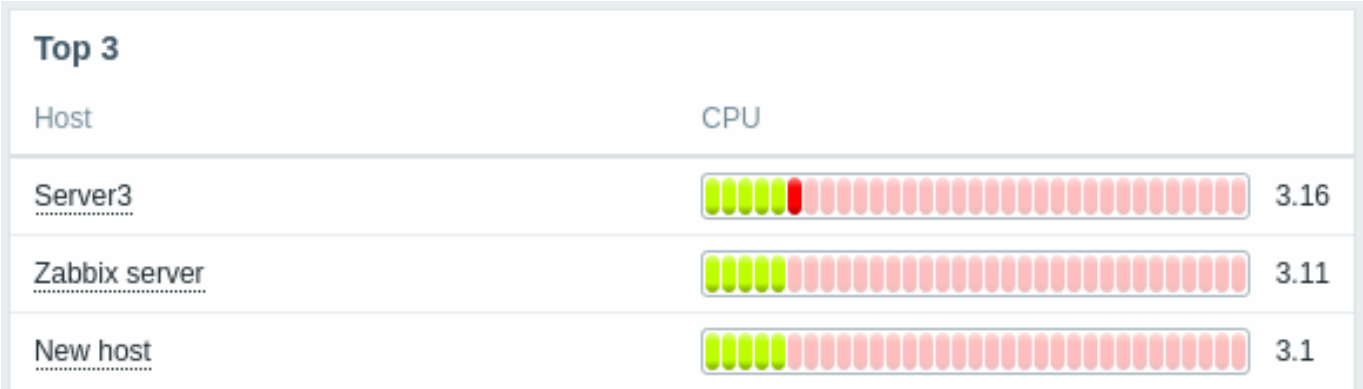
可以支持克隆服务。克隆按钮已添加到服务的配置表单中。克隆服务时，会保留其父链接，而不会保留子链接。

新增小部件　新版本中添加了以下几个仪表板小部件。

Top 主机

Top hosts 小部件被添加到仪表板小部件中。此小部件旨在替换弃用的 数据概览小部件。

Top hosts 小部件允许为数据概览创建自定义表格，这对用于容量规划的类似 *Top N* 类报告和条形进度报告很有用。

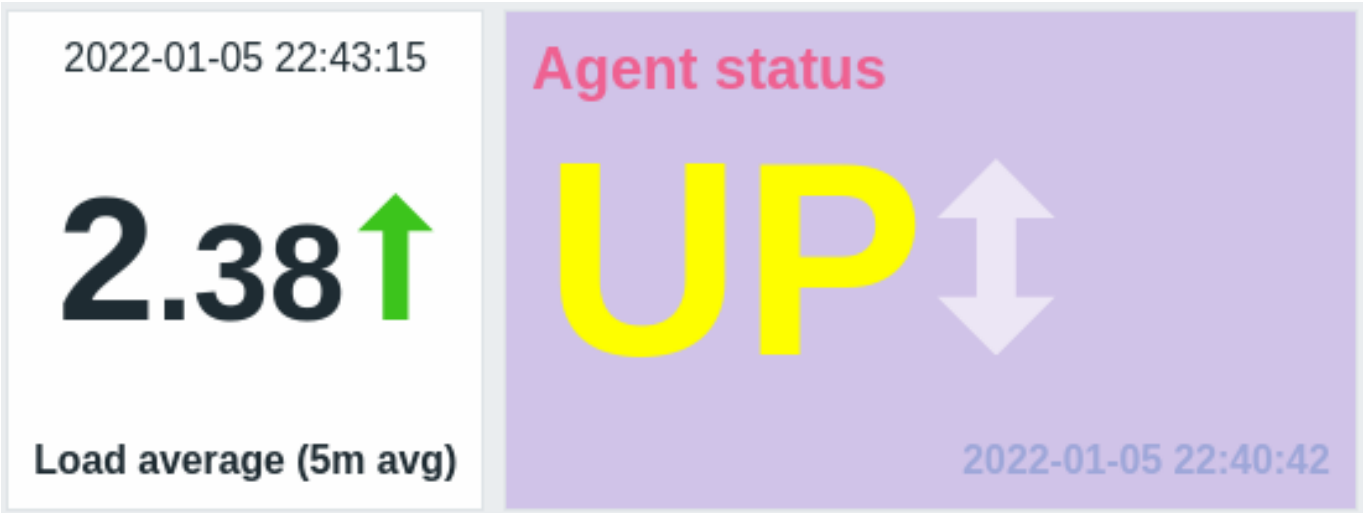


有关更多信息，请参阅Top 主机小部件。

监控项值

监控项值（*Item value*）小部件被添加到仪表盘小部件中。

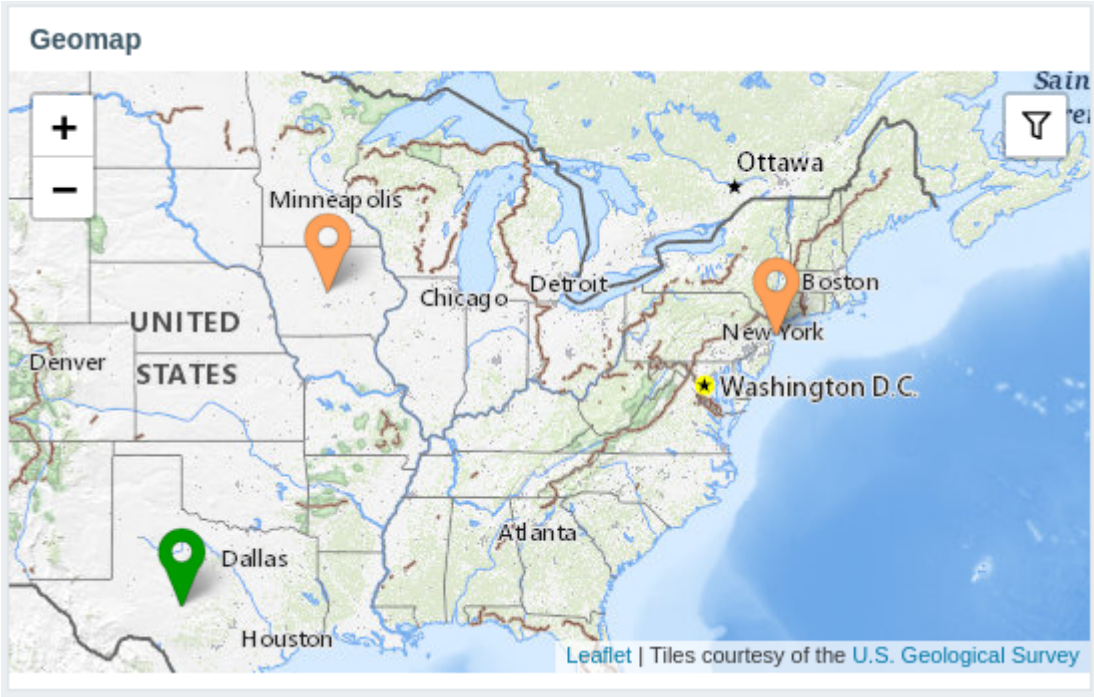
这种类型的小部件对于突出显示单个监控项的值很有用。不同的视觉风格的显示成为可能：



有关详细信息，请参阅Item value 小部件。

地理地图

为仪表板引入了一个新的地理地图小部件，提供了一种在地理地图上显示主机的方法。有关详细信息，请参阅 *Geomap* 仪表板小部件和地理地图。



函数　Prometheus 直方图的函数

在 Zabbix 中收集Prometheus 指标 已经有一段时间了，但有些指标很难使用。具体来说，直方图类型的指标可以在 Zabbix 中呈现为具有相同键值名称但参数不同的多个监控项。然而，即使这些监控项在逻辑上相关并代表相同的数据，如果没有专门的功能，很难分析收集到的数据。为了弥补新版本中的这一功能空白，添加了 **rate()** 和 **histogram_quantile()** 函数，产生与 PromQL 对应的相同结果。

补充此功能的其他新增功能是 **bucket_rate_foreach()** 和 **bucket_percentile()** 函数。有关更多信息，请参阅：

- 历史函数（参见 **rate()**）
- 聚合函数（参见 **histogram_quantile()**，**bucket_percentile()**）
- Foreach 函数（参见 **bucket_rate_foreach()**）

单调变化

现在可以使用新的 **monoinc()** 或 **monodec()** 历史函数 检查监控项值的单调增加或减少。

更改计数

添加了一个新的历史函数**changecount()** ，允许计算相邻值之间的更改次数。该函数支持三种不同的模式：计算所有变化，只减少，或者只增加。例如，它可用于跟踪用户数量的变化或系统正常运行时间的减少。

实体计数

添加了新的函数 来简化由foreach 函数 返回的特定主机、监控项或值的计数。

聚合函数：

- **count** - foreach 函数返回的数组中值的总数（返回一个整数）；
- **item_count** - 当前启用的符合过滤条件的项目总数（返回一个整数）。

Foreach 函数：

- **exists_foreach** - 当前启用的符合过滤条件的项目数（返回一个数组）。

基线监测

可用的基线监控选项集已通过两个新功能 **baselinedev** 和 **baselinewma** 进行了扩展。

- **baselinedev** - 将上一个数据周期与前一 season 的相同数据周期进行比较，返回偏差数；
- **baselinewma** - 通过使用加权移动平均算法对多个相等时间段 (’seasons’) 中同一时间范围内的数据进行平均来计算基线。

在这些函数的上下文中，术语’season’ 指的是可配置的时间范围，可以是几小时、几天、几周、几个月或几年。’season’ 长度和要分析的季节数量在函数参数中设置。

有关更多信息，请参阅历史函数 。

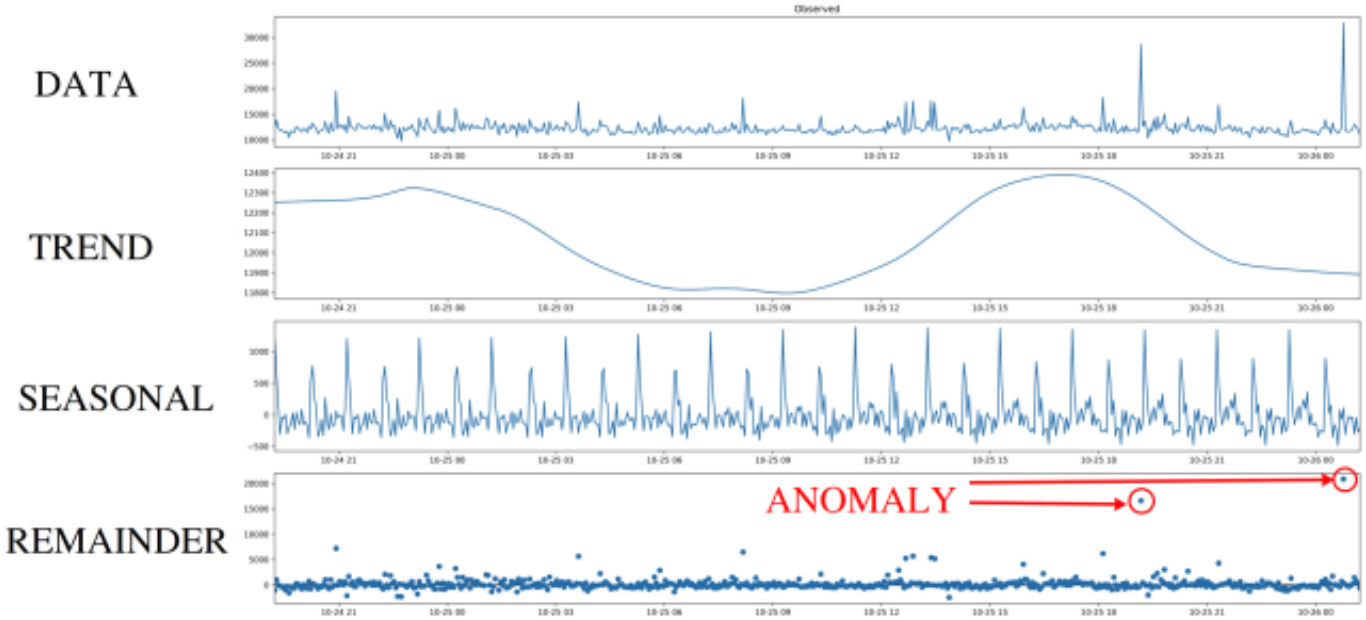
异常检测

Zabbix 5.2 引入了对基线监控有用的新趋势函数。但是，它们仍然需要定义相对阈值（例如，检查 2021 年 9 月的网络流量是否比 2020 年 9 月高出不到 2 倍）。存在难以定义此类阈值的用例。例如，一个新的但非常受欢迎的网站的网络流量可以在一年内自然增长很多倍，但增长速度是未知的。然而，无论自然流量增长如何，由于 DDOS 攻击导致的突然流量激增都必须生成警报。

异常检测算法正是这样做的——在其他值的上下文中查找看起来不正常的数据（异常值）。

添加了新的历史函数 trendstl()，它使用’ 分解’ 方法来计算异常率。它将单个时间序列拆分为其他三个序列：

- 仅包含原始数据发生较大变化的趋势序列（例如网站流量显示增长）
- 仅包含季节性变化的 season 序列（例如夏季网站流量较少，秋季较多）
- 仅包含不能被解释为趋势或 season 的一部分的剩余值的剩余序列



异常检测与余数序列一起工作，并检查是否存在与大多数余数相差太远的值。”远”表示余数序列的绝对值比标准差或平均差大 N 倍。

字符串函数

字符串函数 **concat** 允许连接两个以上的参数。它可用于以不同的组合组合字符串和值，或将两个或多个值相互附加。还支持数字数据类型。

监控项　自动类型选择

监控项配置表单会自动建议匹配的信息类型，如果选定的监控项键值仅返回特定类型的数据（例如，**log[]** 监控行需要信息类型：Log）。信息类型 * 参数现在位于监控项选项卡上的键值参数下，如果指定了至少一个预处理步骤，则在预处理选项卡上重复。如果 Zabbix 检测到所选信息类型和密钥可能不匹配，则会在 信息类型字段旁边显示一个警告图标。

新的和更新的监控项

Zabbix agent/agent 2 中添加了几个新监控项：

- **agent.hostmetadata** - 返回主机元数据
- **kernel.openfiles** - 返回打开文件描述符的数量
- **net.tcp.socket.count[]** - 返回匹配参数的 TCP 套接字数
- **net.udp.socket.count[]** - 返回匹配参数的 UDP 套接字数
- **vfs.dir.get[]** - 以 JSON 格式返回目录文件列表
- **vfs.file.get[]** - 以 JSON 格式返回有关文件的信息
- **vfs.file.owner[]** - 返回文件的所有权
- **vfs.file.permissions[]** - 返回一个 4 位字符串，其中包含具有 Unix 权限的八进制数

此外：

- **vfs.file.cksum[]** 支持第二个 mode 参数（*crc32*, *md5*, *sha256*）
- **vfs.file.size[]** 支持第二个 mode 参数（*bytes* or *lines*）
- **vfs.fs.discovery** 和 **vfs.fs.get** 在 Windows 上返回一个 {#FSLABEL} 宏（带有卷名）

有关详细信息，请参阅agent 监控项。

可计算监控项数据类型

可计算监控项现在不仅支持数字，还支持文本、日志和字符类型的信息。

Prometheus 指标的批量处理　在预处理队列中引入了依赖监控项的批量处理，以提高检索 Prometheus 指标的性能。

有关详细信息，请参阅Prometheus 检查 for more details。

JavaScript 方法　HTTP 方法 PATCH、HEAD、OPTIONS、TRACE、CONNECT 已添加到 JavaScript 引擎中。此外，引擎现在允许使用新的 JS 方法 HttpRequest.customRequest 发送自定义 HTTP 方法请求。

另请参阅：其他 JavaScript 对象。

Web 监控　Zabbix web 监控中增加了处理压缩内容的能力。支持 **libcurl** 支持的所有编码格式。

Preprocessing

Prometheus 查询　Zabbix Prometheus 预处理查询语言 支持两个额外的标签匹配运算符：

- **!=** -- 选择不等于提供的字符串的标签；
- **!~** -- 选择与提供的字符串不匹配的标签。

Prometheus 模式的结果处理

预处理中的 Prometheus 模式步骤会产生匹配多行的结果。为了处理这种情况，Prometheus 模式预处理步骤中添加了一个新的结果处理参数 ，该参数允许通过引入 sum、min、max、avg 和 count 等函数来聚合潜在的多个匹配行的数据。

宏　新增宏

新增用于触发器表达式调试和内部动作的宏。

表达式调试宏简化了触发器表达式的调试过程：

- {TRIGGER.EXPRESSION.EXPLAIN},{TRIGGER.EXPRESSION.RECOVERY.EXPLAIN} - 解析为部分评估的触发器或恢复表达式，其中仅应用基于监控项的函数；
- {FUNCTION.VALUE<1-9>}、{FUNCTION.RECOVERY.VALUE<1-9>} - 解析为事件发生时第 N 个基于监控项的函数的结果。

内部动作的宏包含监控项、LLD 规则或触发器变得不受支持的原因：

- {ITEM.STATE.ERROR} - 用于基于监控项的内部通知；
- {LLDRULE.STATE.ERROR} - 用于基于 LLD 规则的内部通知；
- {TRIGGER.STATE.ERROR} - 用于基于触发器的内部通知。

有关更多详细信息，请参阅支持的宏。

由表达式宏替换的简单宏

Zabbix 5.4 中引入了触发器和可计算监控项的新表达式语法。然而，旧的语法仍然在简单的宏中使用。在新版本中，简单宏的功能已转移到表达式宏中，并使用了新的表达式语法。有关更改的详细信息，请参见下面的比较：

Zabbix 6.0	Zabbix 6.0 之前
`{?avg(/host/key,1h)}`	`{host:key.avg(1h)}`
新版本中的表达式宏示例。	之前版本中的简单宏示例。

现有的简单宏将在升级过程中转换为表达式宏。表达式宏的范围与简单宏提供的范围相同。因此，表达式宏可用于：

- 问题通知和命令
- 问题更新通知和命令
- 地图元素标签
- 地图链接标签
- 地图形状标签
- 图形名称

不再支持位置宏

自 Zabbix 4.0 起已弃用的监控项名称 ($1, $2...$9) 中对位置宏的支持已被完全删除。

不再支持监控项名称中的用户宏

自 Zabbix 4.0 起已弃用的监控项名称（包括发现规则名称）中对用户宏的支持已被完全删除。

数据库　为了创造最佳的用户体验并确保在各种生产环境中获得最佳的 Zabbix 性能，已经放弃了对一些旧数据库版本的支持。这主要适用于接近使用寿命的数据库版本以及存在可能会干扰正常性能的未修复问题的版本。

从 Zabbix 6.0 开始，官方支持以下数据库 版本：

- *MySQL/Percona* 8.0.X
- *MariaDB* 10.5.X - 10.6.X
- *PostgreSQL* 13.X - 14.X
- *Oracle* 19c - 21c
- *TimescaleDB* 2.0.1-2.3
- *SQLite* 3.3.5-3.34.X

默认情况下，如果检测到不支持的数据库版本，Zabbix server 和 proxy 将不会启动。虽然不推荐，但现在可以通过修改server 或proxy 的 AllowUnsupportedDBVersions 配置参数来关闭数据库版本检查。

主键　主键现在用于新安装中的所有表，包括历史表。

现有安装不会自动升级主键。在现有安装中 手动升级历史表主键的说明适用于MySQL/MariaDB、PostgreSQL、TimescaleDB v1 和v2 以及Oracle。

对 MySQL 的 utf8mb4 支持

使用 MySQL/MariaDB 数据库的 Zabbix 安装现在支持使用 utf8mb4_bin 排序规则的 utf8mb4 编码。

以前只支持 utf8 编码，MySQL 代表 utf8mb3 编码，因此只支持正确 UTF-8 字符的子集。在新版本中，增加了对 utf8mb4 的支持，并支持 完整的 UTF-8 字符集。使用 utf8mb3 的旧安装保持不变，并且可以继续使用该编码。

Processes

无需重启 **agent** 即可重载用户参数　现在可以从配置文件重新加载用户参数，而无需重新启动 agent。为此，请运行新的 `userparameter_reload` 运行时控制选项，例如：

```
zabbix_agentd -R userparameter_reload
```

或

```
zabbix_agent2 -R userparameter_reload
```

UserParameter 是唯一将使用此命令重新加载的 agent 配置选项。

PCRE2 支持　已添加对 PCRE2 的支持，并且针对 RHEL/CentOS 7 及更高版本、SLES（所有版本）、Debian 9 及更高版本、Ubuntu 16.04 及更高版本的 Zabbix 安装包已更新为使用 PCRE2。

仍然支持 PCRE，但 Zabbix 只能使用 PCRE 或 PCRE2 库之一进行编译，两者不能同时使用。

Zabbix get 和 **Zabbix sender** 超时　Zabbix get 和 Zabbix sender 实用程序现在支持 `-t <seconds>` 或 `--timeout <seconds>` 超时参数。有效范围是：

- Zabbix get ：1-30 秒（默认：30 秒）
- Zabbix sender：1-300 秒（默认：60 秒）

扩展的 **SNMP** 网关功能　SNMP 网关现在可以提供有关处于问题状态的触发器的信息，并在触发器详细信息中显示主机信息。

此外，现在可以限制 SNMP 网关发送 SNMP traps 的速率。

支持的 OID 列表已扩展为新的 OID **.10**，用于以逗号分隔的触发器主机名列表。

SNMP 网关配置文件中添加了新参数：- *ProblemBaseOID* - 问题触发表的 OID；- *ProblemMinSeverity* - 最低严重性，不包括严重性较低的触发器；- *ProblemHideAck* - 如果指定，仅包含未确认问题的触发器；- *ProblemTagFilter* - 如果指定，仅包含具有指定标签名称的触发器；- *TrapTimer* - 如果设置，Zabbix 将在给定时间范围内发送不超过一个严重性最高的 trap。

有关详细信息，请参阅 Zabbix SNMP 网关。

单独处理 **ODBC** 检查　处理 ODBC 检查已从常规轮询器进程转移到单独的 server/proxy 进程 *ODBC pollers*。此更改允许限制轮询进程创建的与数据库的连接数。以前，ODBC 检查是由常规轮询器执行的，它也适用于 Zabbix agent 监控项、SSH 检查等。

Zabbix server 和proxy 配置文件中添加了一个新的配置参数 *StartODBCPollers* 。

您可以使用内部监控项 *zabbix[process,<type>]* 来监控 ODBC 轮询器负载。

运行时命令传输　Zabbix server 和 proxy 运行时命令现在通过套接字而不是 Unix 信号发送。此更改可以改善使用运行时控制选项的用户体验：

- 命令执行的结果会打印到控制台。
- 可以发送更长的输入参数，例如 HA 节点名称而不是节点编号。

基于 **BSD** 的操作系统上的运行时控制　以前，基于 BSD 的系统不支持 Zabbix server 和 Zabbix proxy 运行时控制选项。更改运行时命令传输方法已允许取消此限制。现在 *FreeBSD*、*NetBSD*、*OpenBSD* 和 **BSD* 系列的其他操作系统都支持大多数命令。有关确切列表，请参阅 Zabbix server 或proxy 的 运行时控制。

Zabbix agent 2 插件　单独的配置文件

每个 Zabbix agent 2 插件现在都有一个单独的配置文件。默认情况下，这些文件位于`./zabbix_agent2.d/plugins.d/` 目录中。该路径在 agent 2 配置文件的 `Include` 参数中指定，查看zabbix_agent2.conf 或zabbix_agent2.win.conf 文件相对路径。

外部插件加载器

以前，插件只能编译到 Zabbix agent 2 中，每次需要更改可用插件集时都需要重新编译 agent。现在，通过添加外部插件加载器，插件不必直接集成到 agent 2 中，并且可以作为单独的外部插件添加，从而使用于收集新监控指标的附加插件的创建过程更加容易。

外部插件的引入导致以下配置参数更改：- *Plugins.<PluginName>.Path* 参数已移至 *Plugins.<PluginName>.System.Path*。- *Plugins.<PluginName>.Capacity* 参数虽然仍受支持，但已被弃用，请改用 *Plugins.<PluginName>.System.Capacity* 。

模板　新的官方模板可用于监控：

Kubernetes

- 基于 *HTTP* 的 *Kubernetes* 节点
- 基于 *HTTP* 的 *Kubernetes* 集群状态
- 通过 *HTTP* 的 *Kubernetes API* 服务器
- 通过 *HTTP* 的 *Kubernetes* 控制器管理器
- 基于 *HTTP* 的 *Kubernetes* 调度程序
- 基于 *HTTP* 的 *Kubernetes kubelet*

要启用 Kubernetes 监控，您需要使用新工具 Zabbix Helm Chart，它会在 Kubernetes 集群中安装 Zabbix proxy 和 Zabbix agent。

要了解有关配置模板的更多信息，请参阅HTTP 模板操作。

Mikrotik

- *MikroTik <device model> SNMP* - 53 新型号特定模板，用于监控各种型号的 MikroTik 以太网路由器和交换机，请参阅 完整列表;https://git.zabbix.com/projects/ZBX/repos/zabbix/browse/templates/net/mikrotik)；
- *Mikrotik SNMP* - 用于监控 MikroTik 设备的通用模板。

可以获得以下模板：

- 在新装环境的 *Configuration → Templates* 中；
- 从以前的版本升级时，可以从 Zabbix Git 存储库 下载最新的模板，并在 *Configuration → Templates* 部分手动导入 Zabbix 。如果已存在同名模板，请在导入前选中删除缺失选项以实现干净导入。这样，已从更新模板中排除的监控项将被删除（请注意，已删除监控项的历史记录将丢失）。

Notifications

Webhook 集成　Webhook 集成允许使用webhook 媒体类型从 Zabbix 通知创建 Github 事件。

关于取消升级的通知

在配置动作操作时，可以通过取消选中相应选项的复选框来取消有关取消升级的通知。

前端

密码要求　现在可以为 Zabbix 内部身份验证方法 提供自定义密码复杂性要求。为了防止 Zabbix 用户设置弱密码，可以强制执行以下限制：

- 设置最小密码长度。
- 要求密码包含大小写字母、数字和/或特殊字符的组合。
- 禁止使用最常见且容易猜到的密码。

审计日志　记录

审计日志现在包含有关所有 Zabbix 对象的所有配置更改的记录，包括由于执行 LLD 规则、网络发现操作、自动注册操作或脚本执行而发生的更改。以前，从 Zabbix server 发起的配置更改，例如作为执行发现规则的结果，不会被记录。现在，此类对象修改将存储为归属于用户系统的审计记录。

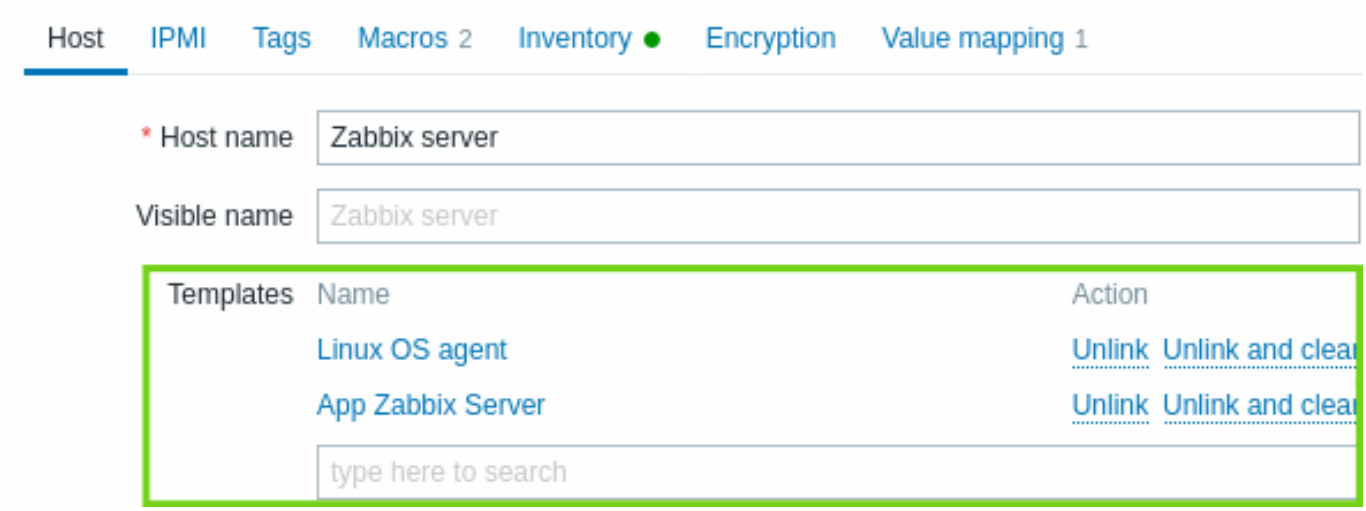
记录过滤器

添加了通过导致这些条目的前端操作过滤记录的功能。如果由于单个操作（例如链接/取消链接模板）而创建了多个日志记录，则这些记录将具有相同的 *Recordset ID*。

审计设置

新的 *Audit log* 部分 已添加到 *Administration→General* 菜单，允许启用或禁用审计日志。以前位于 *Housekeeper* 部分下的用于审核的管家设置也已移至新的 *Audit log* 部分。

模板链接更明显　为了使模板链接更加可见，现在将其放置在主机、主机原型和模板配置表单以及主机/模板批量更新表单的第一个选项卡中。



因此，已从所有相应表单中删除了用于模板链接的单独选项卡。

在相关的开发中，主机原型配置中的主机组/主机组原型选择的字段也已从单独的选项卡移动到第一个选项卡。

最新数据中的子过滤器

在 *Latest data* 部分中添加了一个子过滤器。子过滤器对于快速一键访问相关监控项组很有用。

子过滤器显示 可点击的链接，允许基于通用实体（主机、标签名称或标签值）过滤项目。单击实体后，立即过滤监控项。

有关详细信息，请参阅最新数据（latest data） 部分。

自定义图表的可用性改进

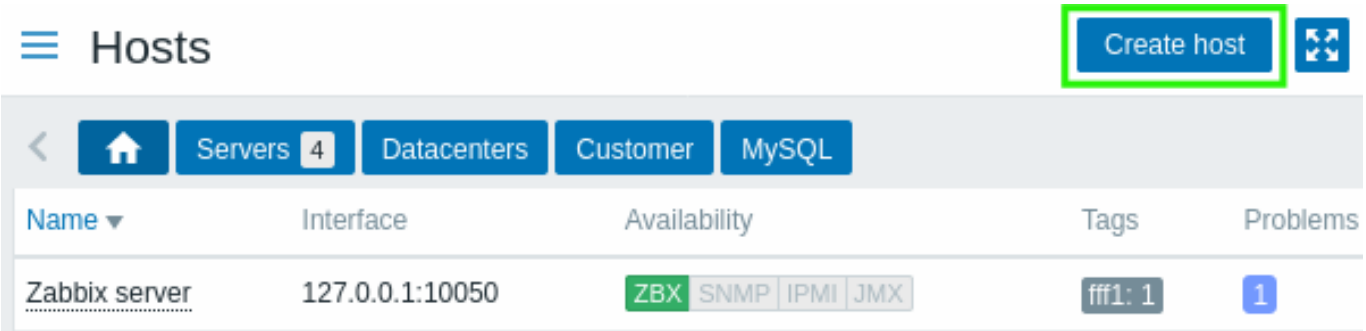
Monitoring → Hosts → Graphs 中的图表页面已经看到了一些可用性改进：

- 页面中不再有 20 个图形的限制
- 添加了一个子过滤器，允许基于公共标签或标签值快速选择相关图表组
- 主机的简单图表可以与自定义图表一起显示

有关更多详细信息，请参阅图表 页面。

从 Monitoring 创建主机

现在还可以从 *Monitoring → Hosts* 建新主机。



管理员和超级管理员用户可以使用 *Create host* 按钮。

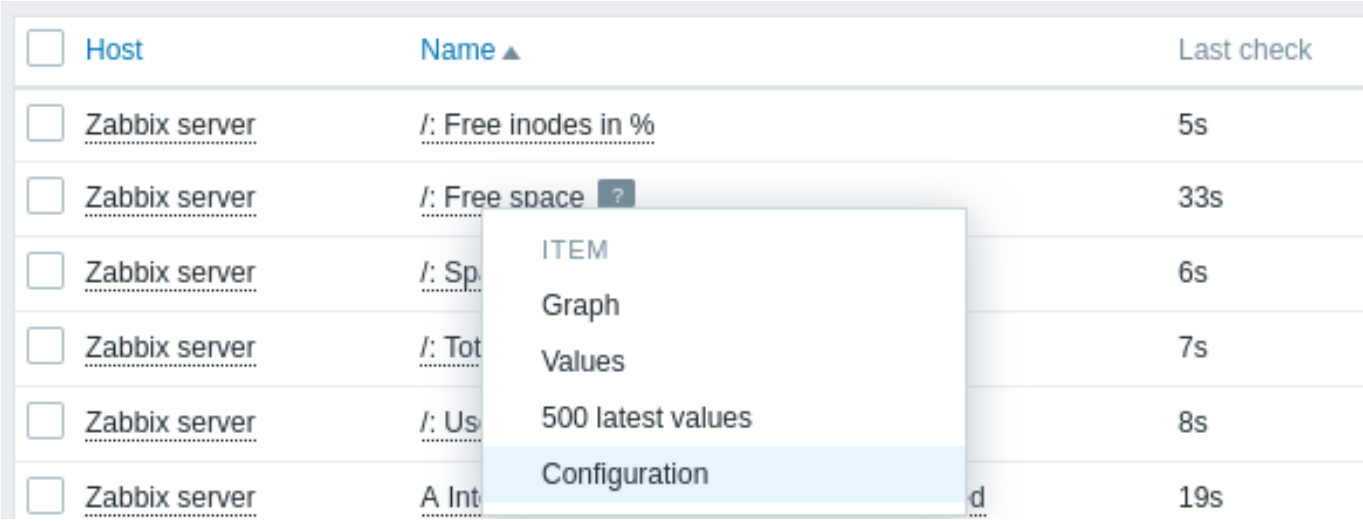
主机编辑作为弹窗

主机创建和编辑的表单可以在任何页面中通过点击 *Configuration → Hosts*、*Monitoring → Hosts* 通过（弹出）窗口模式打开，其中有主机菜单或其他直接链接到主机配置。

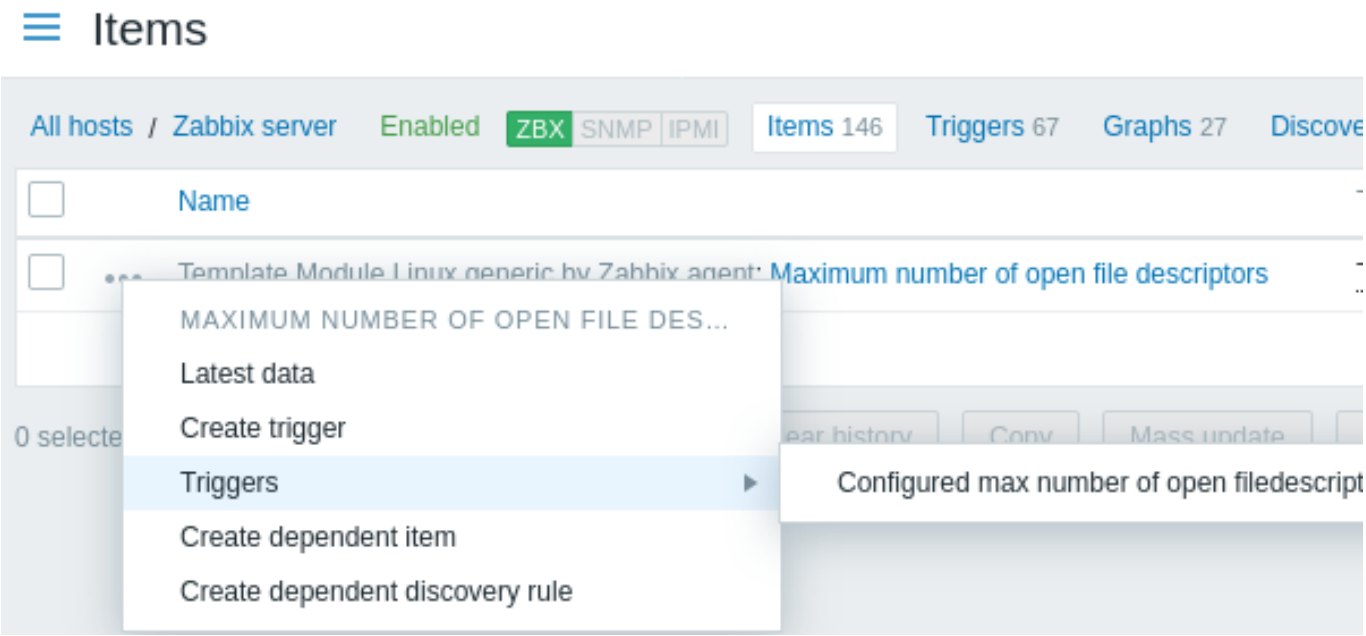
指向主机编辑页面的直接链接仍然有效，并且正在整页打开主机编辑页面。

在监控项配置和最新数据之间更好地导航

最新数据 中引入了新的监控项上下文菜单，允许访问监控项配置和可用图表：



相反，在配置菜单的监控项列表 中引入了一个新的上下文菜单，允许访问监控项的最新数据和其他有用的选项：



此菜单取代了以前版本中的向导选项。还为模板监控项和监控项原型 引入了类似的菜单。

Monitoring → Latest data 更新

对最新数据部分进行了一些改进：

- 显示自上次检查以来的时间（例如，1m 20s），而不是上次监控项执行时间。
- 将鼠标悬停在监控项的最后一个值上将显示未应用单位或值映射的原始值。
- 如果主机处于维护状态，主机名旁边会显示一个橙色扳手图标。

Monitoring → Overview 被移除

监控菜单中的概览（Overview）部分已被完全删除。使用数据概览（*Data overview*）和 触发器概览（Trigger overview）* 仪表板小部件仍然可以访问相同的功能。

杂项

- Zabbix Web 界面的默认语言已从英式英语更改为美式英语。已放弃对英式英语的支持。
- 主菜单中的 Share 链接已被 Integrations 链接取代，指向 Zabbix 网站上的 Integrations 页面。
- 如果 Zabbix Web 界面以 Zabbix 网站上可用的语言之一打开，单击集成链接将以适当的语言打开集成页面。对于所有其他语言，包括英语，集成页面将以英语打开。
- 用于计算条件的动作配置 中的自定义表达式现在最多可包含 1024 个字符（之前为 255 个）。
- *Monitoring->Hosts* 部分现在显示指向主机问题屏幕的链接，即使当前没有打开任何问题。

重大变化

审计日志　为了实现审计日志 功能的变化，以前存在的数据库结构必须重新设计。在升级过程中 `auditlog` ，`auditlog_details` 数据库表将被具有不同格式的新表 `auditlog` 替换。* 现有的审核日志记录将被删除 **。

支持的数据库版本检查　Zabbix server 和proxy 现在将在启动前检查数据库版本，如果版本超出支持范围，将不会启动。有关更多详细信息，请参阅数据库。

PCRE2 支持　Zabix 现在支持 PCRE 和 PCRE2。用于 RHEL/CentOS 7 及更高版本、SLES（所有版本）、Debian 9 及更高版本、Ubuntu 16.04 及更高版本的 Zabbix 软件包已更新为使用 PCRE2 而不是 PCRE 进行编译。从源代码编译时，用户可以选择指定”--with-libpcre” 或”--with-libpcre2” 标志。如果您要升级现有安装，将 PCRE 更改为 PCRE2 可能会导致某些正则表达式表现不同 - 请参阅已知问题了解详细信息。

6 Zabbix 6.0.1 新功能

Zabbix agent 2 监控项

- 添加了对监控项 **net.dns** 和 **net.dns.record** 的原生支持。这些监控项与 Zabbix agent 2 一起使用，现在支持并发检查处理。在 Windows 上，`ip` 参数中允许自定义 DNS IP 地址，`timeout` 和 `count` 参数不再被忽略。
- S.M.A.R.T. 插件支持的 *smart.disk.discovery* 和 *smart.attribute.discovery* 监控项已更新，现在以小写形式返回 {#DISKTYPE} 宏值。

发现禁用的 **systemd** 单元　现在还可以使用 *systemd.unit.discovery* 监控项的键发现禁用的 systemd 单元，由 Zabbix agent 2 支持。请注意，要从禁用的 systemd 单元的原型创建监控项和触发器，可能需要为 {#UNIT.ACTIVESTATE} 和 {#UNIT.UNITFILESTATE} 宏调整（或删除）禁止 LLD 过滤器。

有关更多详细信息，请参阅发现 systemd 服务。

SNI support in encrypted connections　Encrypted TCP connections between Zabbix agent and Zabbix server or proxy now support SNI.

LDAP 简单检查中的 **SourceIP** 支持　SourceIP 支持已添加到 LDAP 简单检查。注意对于 OpenLDAP，需要 2.6.1 或更高版本。

7 Zabbix 6.0.2 新功能

Zabbix agent 2 主动检查配置

Zabbix agent 2 中添加了一个新的可选配置参数 *ForceActiveChecksOnStart* 。将参数设置为 `ForceActiveChecksOnStart=1` 将确保在 Zabbix agent 重新启动时立即收集主动检查的监控项数据，具有 调度update interval的监控项除外。否则，agent 重新启动后的第一次数据收集将在随机时间发生，该时间小于监控项更新间隔，以防止资源使用高峰。

通过使用 *Plugins.<PluginName>.System.ForceActiveChecksOnStart*，也可以仅为特定插件设置此选项(例如，`Plugins.Uptime.System.For`
如果设置，插件级参数将覆盖全局设置。

JMX 监控　模板 *Generic Java JMX* 现在包含内存池和垃圾回收器的低级别自动发现规则。

键盘导航　已对前端的信息图标实施键盘控制。现在是这样可以专注于信息图标，并使用键盘打开提示。

8 Zabbix 6.0.3 新功能

PostgreSQL 指标

Zabbix agent 2 的 PostgreSQL 插件中添加了一个新的监控项。指标 **pgsql.queries** 用于监控查询执行时间。

模板

新模板 *OpenWeatherMap by HTTP* 现在可以通过 HTTP 监控 OpenWeatherMap。有关设置说明，请参阅HTTP 模板操作。

在现有模板中进行了以下更改：

- 在模板 *Windows services by Zabbix agent*，*Windows services by Zabbix agent active*，*Windows by Zabbix agent*，*Windows by Zabbix agent active* ，{$SERVICE.NAME.NOT_MATCHES} 宏值已更新以筛选出扩展的服务列表。
- 模板 *PostgreSQL by Zabbix agent 2* 现在将会检查慢查询的数量，如果数量超过阈值，则会产生问题。

您可以获得以下模板：

- 在新设备的 *Configuration* → *Templates*；
- 如果要从以前的版本升级，可以从 Zabbix Git repository下载新模板，或在下载的最新 Zabbix 版本的 `templates` 目录中找到它们。然后，在 *Configuration* → *Templates* 您可以手动将模板导入 Zabbix。

9 Zabbix 6.0.4 新功能

Top hosts 组件的文本数据　现在可以在Top hosts 组件中选择包含任何信息类型（包括字符、文本和 日志）的监控项。例如，现在可以使用此组件显示每台主机上运行的 Zabbix agent 版本。

OpenSSL 3.0 支持　现已支持 OpenSSL 3.0.x。请注意，此更改不会影响前端加密（使用自己的 openssl-php 包）和 Java 网关到监控目标的 JMX 加密连接（使用自己的 Java 加密库）。

模板　新模板可用：- *TrueNAS SNMP* - 通过 SNMP 监控 TrueNAS 存储操作系统 - *Proxmox VE by HTTP* - 请参阅HTTP 模板的设置说明

以下模板中添加了新宏，新宏定义用于虚拟文件系统监控文件系统利用率的告警和关键阈值，*HOST-RESOURCES-MIB storage SNMP*，*Linux by Prom*，*Linux filesystems SNMP*，*Linux filesystems by Zabbix agent active*，*Linux filesystems by Zabbix agent*，*Mellanox SNMP*，*PFSense SNMP*，*Windows filesystems by Zabbix agent active*，*Windows filesystems by Zabbix agent*。文件系统利用率触发器已更新为使用这些宏。

您可以获得以下模板：

- 在新设备的 *Configuration* → *Templates* ；
- 如果要从以前的版本升级，可以从 Zabbix Git repository下载新模板，或在下载的最新 Zabbix 版本的 `templates` 目录中找到它们。然后，在 *Configuration* → *Templates* 您可以手动将模板导入 Zabbix。

GLPi 集成　一个新的 GLPi 集成 允许使用webhook 媒体类型，用于根据 Zabbix 问题通知在 GLPi 协助部分创建问题。

S.M.A.R.T. 监控　Zabbix agent 2 支持的 *Smart plugin* 现在提供了更高效的磁盘发现，并允许返回有关特定磁盘的信息，而不是所有发现的磁盘。Zabbix agent 2 监控项 **smart.disk.discovery** 和 **smart.disk.get** 已更新。模板 *SMART by Zabbix agent 2* 和 *SMART by Zabbix agent 2 (active)* 已修改，以纳入新功能。

10 Zabbix 6.0.5 新功能

模板　新的模板已经可以使用:

- CockroachDB by HTTP
- Envoy Proxy by HTTP
- HashiCorp Consul Cluster by HTTP
- HashiCorp Consul Node by HTTP

有关设置说明，请参阅HTTP 模板.

你可以使用如下方式获取到新的模板:

- 在新安装的 配置 → 模板中;
- 如果要从以前的版本升级，可以从 Zabbix Git repository 下载新模板，或在下载的最新 Zabbix 版本的 `templates` 目录中找到它们。然后，在 配置 → 模板您可以手动将模板导入 Zabbix 。

Prometheus 预处理 **NaN** 的值　这里会有一个新的行为去处理 (跳过)NaN 值。因此，如果一个数据集由有效的数值和 NaN 组成，那么 NaN 值将被跳过，并且:

- 'avg', 'max', 'min', 'sum' 会使用有效值进行计算并且返回结果
- 'count' 会返回有效值的个数

如果数据集中的所有值都是 NaN 那么'avg', 'max', 'min', and 'sum' 会返回一个"no data (at least one value is required)" 的错误提示，并且'count' 会返回 0。

以前，如果值 NaN 是数据集中的第一个值，那么:

- 'avg', 'max', 'min', 'sum' 会返回"Value "NAN" of type "string" is not suitable for value type "Numeric (float)"" 的错误提示
- 'count' 会返回值的数量 (包括 NaN 值)

同样在之前，如果值 NaN 不是数据集中的第一个值，那么:

- 'avg', 'sum' 会返回"Value "NAN" of type "string" is not suitable for value type "Numeric (float)"" 的错误提示
- 'max' 返回在遇到第一个 NaN 之前的最大值
- 'min' 返回在遇到第一个 NaN 之前的最小值
- 'count' 返回值的数量 (包括 NaN 值)

主机的最新数据链接显示数量　在监控 -> 主机 中主机的最新数据链接现在显示带有最新数据的项目数量。

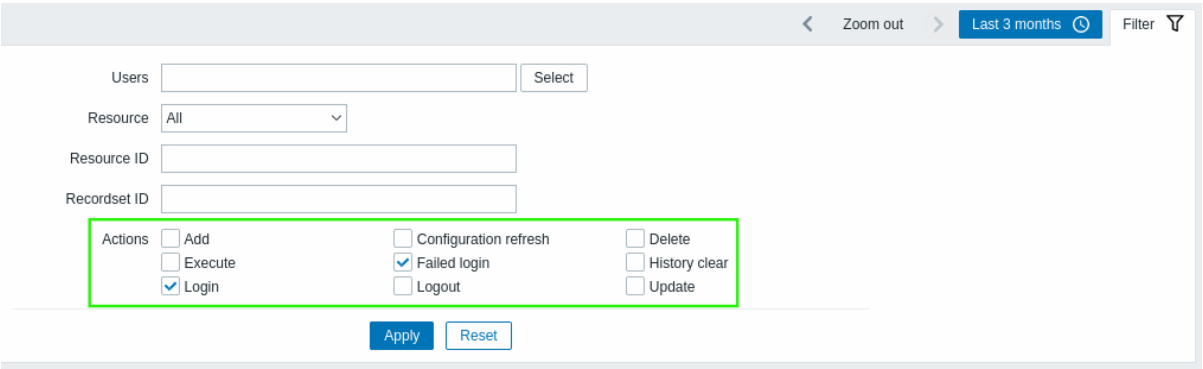
前端语言　现在在前端可以使用德语和越南语。

在最新的数据过滤器中的扩展列表　在latest data 的数据子过滤器找那个引入了可扩展列表:

- 对于每个条目组 (例如标签，主机)，最多可以显示 10 行记录。如果想显示更多的记录条数，通过点击最后显示的三个点图标这个列表可以扩展到最多 1000 个条目 (在frontend definitions 的 SUBFILTER_VALUES_PER_GROUP 的值) 。by clicking on a three-dot icon displayed at the end. 之前这个上限是 100，而且不可用扩展增加.
- 在 *Tag values* 列表中，现在显示了最多 10 行 Tag 名称。如果有更多带值的 Tag 名称，通过单击显示在底部的三个点图标，可以将该列表扩展到最多 200 个 Tag 名称。在以前，带 Tag 名称的列表条数上限是 20 行，而且不可以扩展。标记名的不可扩展的最多 20 行是限制。

对于每个 Tag 名称，最多显示 10 行的值 (可扩展到 1000 个条目 (在frontend definitions) 中的 SUBFILTER_VALUES_PER_GROUP 的值).

审计日志过滤器　现在可以在 报告 -> 审计的审计日志过滤器中选择多个操作:



这有助于在审计列表中查看所有相关操作 (例如，前端登录的成功和失败记录)。

11 Zabbix 6.0.6 新功能

PHP 8 支持　现在支持 PHP 8.0 和 8.1。

MariaDB 10.7 支持　MariaDB 支持的最高版本现在是 10.7.x。

扩展 **MongoDB** 插件　MongoDB plugin 插件不再是 Zabbix agent 2 的一部分，现在是一个可加载的插件. MongoDB 支持的版本列表已经扩展到 *2.6-5.3*.

插件功能和支持的items 没有改变.

模板　新的模板已经可以使用:

- HPE MSA 2040 Storage by HTTP
- HPE MSA 2060 Storage by HTTP
- HPE Primera by HTTP

有关设置说明，请参阅HTTP 模板。

你可以使用如下方式获取到新的模板:

- 在新安装的 配置 → 模板;
- 如果要从以前的版本升级，可以从 Zabbix Git repository 下载新模板，或在下载的最新 Zabbix 版本的 `templates` 目录中找到它们。然后，在 配置 → 模板您可以手动将模板导入 Zabbix 。

ExpressMS webhook API changed　API version changed to v4 in ExpressMS messenger webhook.

12 Zabbix 6.0.7 新功能

MariaDB 10.8 支持　MariaDB 支持的最高版本现在是 10.8.x。

TimescaleDB 2.6 支持　TimescaleDB 支持的最高版本现在是 2.6。

模板　新模板 *HPE Synergy by HTTP* 可用。

请参阅HTTP 模板 的设置说明。

您可以获得这些模板：

- ·在新安装的配置 → 模板；
- ·如果您是从以前的版本升级，您可以从 Zabbix Git 存储库 下载新的模板或在下载的最新 Zabbix 版本的 `templates` 目录。然后，在 配置 → 模板中，您可以手动将它们导入 Zabbix。

Updated templates

PostgreSQL Agent 2 template has been updated.

A trigger for detecting checksum failures has been added to the Dbstat item of the PostgreSQL Agent 2 template.

You can get this template:

- In *Configuration* → *Templates* in new installations;
- If you are upgrading from previous versions, you can download new templates from Zabbix Git repository or find them in the `templates` directory of the downloaded latest Zabbix version. Then, while in *Configuration* → *Templates* you can import them manually into Zabbix.

13 Zabbix 6.0.8 新功能

月份用大写字母缩写　“月”现在缩写为前端的大写字母“M”。以前它被缩写为小写的“m”，与分钟的缩写重叠。

TimescaleDB 2.7 支持　TimescaleDB 支持的最高版本现在是 2.7。

模板　一个新的模板 *OPNsense by SNMP* 是可用的。

你可以获得这个模板：

- ·在新安装的配置 → 模板；
- ·如果您是从以前的版本升级，您可以从 Zabbix Git 存储库 下载新模板或在下载的最新 `Zabbix` 版本的 `templates` 目录。然后，在 配置 → 模板中，您可以手动将它们导入 Zabbix。

RHEL 包重命名　RHEL 软件包已通过在名称中添加 “release” 一词进行重命名：

命名	包名
旧	`zabbix-agent-6.0.7-1.el9.x86_64.rpm`
新	`zabbix-agent-6.0.8-release1.el9.x86_64.rpm`

没有与此更改相关的功能更改。

这是准备提供次要版本（即 6.0.x）候选发布包的准备，预计从 6.0.9 开始。命名更改将确保对于在其系统上同时启用了稳定和不稳定存储库的人，将以正确的顺序接收存储库更新。此命名更改仅适用于 RHEL 包。

14 Zabbix 6.0.9 新功能

表达式宏　表达式宏 现在支持 {ITEM.KEY<1-9>} 宏。

包　SQL 脚本已从 “/usr/share/doc” 目录移动到 Zabbix 包中的 “/usr/share”。

15 Zabbix 6.0.10 新功能

记住过滤器设置

在几个监控页面（问题、主机、最新数据）中，当前的过滤器设置现在被记住在用户配置文件中。当用户再次打开页面时，过滤器设置将保持不变。

此外，更改（但未保存）收藏过滤器的标记现在是过滤器名称旁边的绿点，而不是斜体的过滤器名称。

TimescaleDB 2.8 支持　TimescaleDB 支持的最高版本现在是 2.8。

PostgreSQL 15 支持　现在支持 PostgreSQL 15。请注意，TimescaleDB 不支持 PostgreSQL 15。

可以离线构建 **Zabbix agent 2**　Zabbix agent 2 现在可以离线构建。源码 tarball 现在包含 `src/go/vendor` 目录，应该确保 golang 不会被强制自动下载依赖模块。仍然可以使用 `go mod tidy` 或 `go get` 命令手动更新到最新模块。

PostgreSQL 插件可加载　PostgreSQL 插件 现在可以在 Zabbix agent 2（以前内置）中加载。

另请参阅：PostgreSQL 可加载插件 存储库

前端　各种各样的

- ·如果历史或趋势表包含压缩块，但 *Override item history period* 或 *Override item trend period* 选项被禁用，现在会显示有关 TimescaleDB 不正确内部管理配置的警告。有关详细信息，请参阅TimescaleDB 设置。

16 Zabbix 6.0.11 新功能

报告零索引节点的文件系统　`vfs.fs.get` agent 监控项现在能够报告 inode 计数为零的文件系统，对于具有动态 inode 的文件系统（例如 btrfs）可能就是这种情况。

此外，在模式设置为 “pfree” 或 “pused” 的情况下，“vfs.fs.inode” 监控项现在不会变得不受支持。相反，此类文件系统的 pfree/pused 值将分别报告为 “100” 和 “0”。

优化的 **API** 查询　API 数据库查询，在搜索 *hosts* 和 *items* 表中的名称时创建，已经过优化，现在可以更有效地处理。

17 Zabbix 6.0.12 新功能

改进了历史同步器的性能　通过引入新的读写锁提高了历史同步器的性能。这通过在访问配置缓存时使用共享读锁减少了历史同步器、捕获器和代理轮询器之间的锁定。新锁只能由配置同步器执行配置缓存重新加载。

18 Zabbix 6.0.13 新功能

重大变化　可加载插件版本控制

Loadable plugins for Zabbix agent 2 现在使用与 Zabbix 本身相同的版本控制系统。进行了以下版本更改：

- MongoDB 1.2.0 -> MongoDB 6.0.13
- PostgreSQL 1.2.1 -> PostgreSQL 6.0.13

Zabbix 6.0 的任何次要版本都支持这些插件。请注意，每个插件的源代码存储库现在包含一个专用的 *release/6.0* 分支（以前只有 *master* 分支）。

MariaDB 10.10 支持　MariaDB 支持的最高版本现在是 10.10.x。

TimescaleDB 2.9 支持　TimescaleDB 支持的最高版本现在是 2.9。

配置导入　以前，导入过程会因可导入实体（主机组、监控项、图形等）的 UUID 不匹配而失败。例如，如果主机上已存在同名的主机组，则无法导入主机组。

在新版本中，导入不会因为 UUID 不匹配而失败；取而代之的是，实体将通过唯一性标准（例如实体 ID（名称））进行匹配。实体将被导入，UUID 将更新为导入实体的 UUID。

在另一个改进中，当通过导入的模板或主机删除模板链接（模板链接的删除缺失选项）时，不再删除未链接模板的继承实体（模板被取消链接，而不是取消链接和清除），除非导入文件中缺少这些实体并且特定实体的 删除缺失选项已标记。

由于此更改，标记模板链接的删除缺失选项时的警告消息将不再显示。

使用 **Zabbix agent 2** 查询 **Oracle** 数据库中的单独表空间　以下Zabbix agent 2 监控项，支持 Oracle 插件，现在有额外的可选参数：

- oracle.diskgroups.stats[<existingParameters>,**<diskgroup>**]
- oracle.archive.info[<existingParameters>,**<destination>**]
- oracle.cdb.info[<existingParameters>,**<database>**]
- oracle.pdb.info[<existingParameters>,**<database>**]
- oracle.ts.stats[<existingParameters>,**<tablespace>**,**<type>**]

这些参数允许查询单独的数据实例而不是所有数据，从而提高性能。

使用 `docker.container_info[]` 检索附加信息　`docker.container_info[]` Zabbix agent 2 item 现在支持检索有关 Docker 容器的部分（短）或完整底层信息的选项。

用于分析的运行时命令　用于分析的运行时命令已添加到 Zabbix 服务器和 Zabbix 代理。

- ·`prof_enable` - 启用分析
- ·`prof_disable` - 禁用分析

可以为每个服务器/代理进程启用分析。启用的分析按函数名称提供所有 rwlocks/mutexes 的详细信息。

也可以看看：

- ·Zabbix 服务器运行时命令
- ·Zabbix 代理运行时命令

JavaScript 的 **HMAC** 函数　JavaScript 引擎中添加了一个新函数，允许返回 HMAC 哈希：

- ·`hmac('<hash type>',key,string)`

这对于需要基于散列的消息身份验证代码 (HMAC) 来签署请求的情况很有用。支持 MD5 和 SHA256 哈希类型，例如：

·- ·`hmac('md5',key,string)` ·- ·`hmac('sha256',key,string)`

模板　新模板可用：

- AWS EC2 by HTTP
- AWS by HTTP
- AWS RDS instance by HTTP
- AWS S3 bucket by HTTP

请参阅HTTP 模板 的设置说明。

您可以获得这些模板：

- ·在新安装的配置 → 模板；
- ·如果您是从以前的版本升级，您可以从 Zabbix Git 存储库 下载新的模板或在下载的最新 Zabbix 版本的 `templates` 目录。然后，在 *Configuration* → *Templates* 中，您可以手动将它们导入 Zabbix。

模板 *Oracle by Zabbix agent 2* 已根据对多个Zabbix agent 2 监控项 所做的更改进行了更新（删除了多个静态监控项；添加了多个监控项原型）。

有关更新的更多信息，请参阅模板变更。

您可以获得这些模板：

- ·新安装的在配置 → 模板；
- ·如果您是从以前的版本升级，您可以从 Zabbix Git 存储库 下载新模板或在下载的最新 `Zabbix` 版本的 `templates` 目录。然后，在 配置 → 模板中，您可以手动将它们导入到 Zabbix。

Webhook 集成　一种新的媒介类型 *LINE* 现在可用，允许使用webhook 功能将有关 Zabbix 事件的通知发送到 *LINE* messenger。

前端语言　现在在前端可以使用加泰罗尼亚语和罗马尼亚语。

Windows 更新的 **Golang** 库　Zabbix agent 2 与 MongoDB 或 PostgreSQL 插件一起使用的 Golang 库现在是 `github.com/Microsoft/go-wini` 版本 0.6.0（以前是 `github.com/natefinch/npipe`）。另见Golang 库，MongoDB 插件依赖, 和PostgreSQL 插件依赖项。

Zabbix agent 2 打开文件描述符限制增加　Zabbix agent 2 包中附带的 systemd 服务文件现在声明打开文件描述符限制为 8196。以前，系统默认限制为 1024。新限制足以满足默认 Zabbix agent 2 配置。如果您有非 -standard agent 2 配置，例如，使用额外的插件或扩展功能，此限制可能需要手动进一步增加。在这种情况下，调整 systemd 单元文件中的 `LimitNOFILE` 参数。

19 Zabbix 6.0.14 新功能

可加载插件

加密的 **MongoDB** 插件连接　使用命名会话连接到 MongoDB 时，MongoDB 插件现在支持 TLS 加密。

从 Zabbix 6.0.14 开始，更新的插件（MongoDB 插件 1.2.1）包含在 Zabbix 官方包中。请注意，MongoDB 是一个可加载插件，可以从包或源中单独安装。该插件适用于任何次要版本的 Zabbix 6.0。有关详细信息，请参阅MongoDB 插件。

PHP support　The maximum supported version for PHP is now 8.2.

Limits for JavaScript objects in preprocessing　The following limits for JavaScript objects in preprocessing have been introduced:

- The total size of all messages that can be logged with the `Log()` method has been limited to 8 MB per script execution.
- The initialization of multiple `HttpRequest` objects has been limited to 10 per script execution.
- The total length of header fields that can be added to a single `HttpRequest` object with the `addHeader()` method has been limited to 128 Kbytes (special characters and header names included).

20 What's new in Zabbix 6.0.15

MariaDB 10.11 support　The maximum supported version for MariaDB is now 10.11.X.

TimescaleDB 2.10 support　The maximum supported version for TimescaleDB is now 2.10.

Connection options for Oracle plugin Oracle plugin, supported for Zabbix agent 2, now allows to specify `as sysdba`, `as sysoper`, or `as sysasm` login option. The option can be appended either to the user item key parameter or to the plugin configuration parameter Plugins.Oracle.Sessions.<SessionName>.User in the format `user as sysdba` (login option is case-insensitive; must not contain a trailing space).

Signing data using RS256 A new `sign(hash,key,data)` JavaScript function has been implemented allowing to use the RS256 encryption algorithm to calculate the signature.

For more details see: Additional JavaScript objects.

21 What's new in Zabbix 6.0.16

Configuration sync optimization for Oracle

For Zabbix installations with Oracle, it is now possible to manually change item and item preprocessing database field types from *nclob* to *nvarchar2* by applying a database patch.

Patch application may increase the speed of configuration sync in environments with large number of items and item preprocessing steps, but will reduce the maximum field size limit from 65535 bytes to 4000 bytes for some item parameters. See Known issues for details.

22 What's new in Zabbix 6.0.17

Webhook integrations

New webhook media type for pushing Zabbix notifications to Event-Driven Ansible has been added.

Mixing item key and session parameters in Zabbix agent 2 plugins Zabbix agent 2 now allows to override named session parameters by specifying new values in the item key parameters. Previously, users had to select if they prefer to provide connection string values in a named session or in an item key. If a named session has been used, related item key parameters had to be empty. Now, if using named sessions, only the first parameter (usually, a URI) has to be specified in the named session, whereas other parameters can be defined either in the named session or in the item key.

HTML support in Geomap attribution dropped The attribution text for the Geomap dashboard widget can now only contain plain text; HTML support has been dropped.

In Geographical maps settings in the Administration → General section, the field *Attribution* is now only visible when *Tile provider* is set to *Other*.

23 What's new in Zabbix 6.0.18

Items docker.container_stats

The docker.container_stats item on Zabbix agent 2 now also returns a `pids_stats` property with the current number of processes/threads on the container.

Cleaner configuration export YAML files generated during Zabbix entity configuration export no longer contain empty lines between entities in an array, which makes such files shorter and more convenient to work with. See Configuration export/import section for updated export examples.

UTF-8 BOM in configuration import Configuration import now supports files with a UTF-8 byte-order mark (BOM).

Cosmos DB monitoring The template *Azure by HTTP* now also works with Azure Cosmos DB for MongoDB.

You can get this template:

- In *Configuration* → *Templates* in new installations.

- If you are upgrading from previous versions, you can download this template from Zabbix Git repository or find it in the *zabbix/templates* directory of the downloaded latest Zabbix version. Then, while in *Configuration* → *Templates* you can import it manually into Zabbix.

Proxy history housekeeping The limitation on the amount of outdated information deleted from the proxy database per proxy history housekeeping cycle has been removed.

Previously the housekeeper deleted only no more than 4 times the `HousekeepingFrequency` hours of outdated information. For example, if `HousekeepingFrequency` was set to "1", no more than 4 hours of outdated information (starting from the oldest entry) was deleted. In cases when a proxy would constantly receive data older than set in `ProxyOfflineBuffer`, this could result in excessive data accumulation.

Now this limitation has been removed, providing a more effective proxy history housekeeping solution.

A new template *Google Cloud Platform by HTTP (GCP by HTTP)* is available.

Default values for Zabbix agent 2 Zabbix agent 2 plugins now allow to define default values for connecting to monitoring targets in the configuration file. If no value is specified in an item key or a named session, the plugin will use the value defined in the corresponding default parameter. New parameters have the structure `Plugins.<PluginName>.Default.<Parameter>` - for example, `Plugins.MongoDB.Default.Uri=tcp://localhost:27017`. See for more info:

- Configuring plugins
- Plugin configuration file parameters

24 What's new in Zabbix 6.0.19

Aggregate functions The **count_foreach** function now returns '0' for a matching item in the array, if no data are present for the item or the data do not match the filter. Previously such items would be ignored (no data added to the aggregation).

TimescaleDB 2.11 support Support for TimescaleDB version 2.11 is now available.

Configurable TLS and connection parameters in MQTT plugin The MQTT plugin for Zabbix agent 2 now provides additional configuration options, which can be defined in the plugin configuration file as named session or default parameters:

- Connection-related parameters: broker URL, topic, username, and password;
- TLS encryption parameters: location of the top-level CA(s) certificate, MQTT certificate or certificate chain, private key.

All of the new parameters are optional.

JavaScript preprocessing The heap limit for scripts has been upped from 64 to 512 megabytes.

Supported platforms Support for Debian 12 (Bookworm) has been added, and official packages are available for download on Zabbix website.

25 What's new in Zabbix 6.0.20

Templates A new template *AWS ECS Cluster by HTTP* (along with its Serverless Cluster version) is available.

You can get this template:

- In *Configuration* → *Templates* in new installations;
- If you are upgrading from previous versions, you can download new templates from Zabbix Git repository or find them in the *zabbix/templates* directory of the downloaded latest Zabbix version. Then, while in *Configuration* → *Templates* you can import them manually into Zabbix.

Frontend Spellcheck disabled in non-descriptive text areas

Spellcheck has been disabled for the text areas in which non-descriptive text is entered, such as scripts, expressions, macro values, etc.

Miscellaneous Database TLS connection for MySQL on SLES 12

The packages for server/proxy installation on SUSE Linux Enterprise Server version 12 are now built using MariaDB Connector/C library, thus enabling the encryption of connection to MySQL using the DBTLSConnect parameter. The supported encryption values are "required" and "verify_full".

26 What's new in Zabbix 6.0.21

MySQL 8.1 support

The maximum supported version for MySQL is now 8.1.X.

MariaDB 11.0 support

The maximum supported version for MariaDB is now 11.0.X.

Log file monitoring

For `log[]`, `logrt[]`, `log.count[]`, `logrt.count[]` items, regular expression runtime errors are now logged in the Zabbix agent log file. See more details.

Items New item for Zabbix agent 2

A new item has been added to MySQL plugin for Zabbix agent 2. This new item, mysql.custom.query, can be used for executing custom MySQL queries.

Templates New template is available:

- AWS Cost Explorer by HTTP

You can get this template:

- In *Configuration* → *Templates* in new installations;
- If you are upgrading from previous versions, you can download new templates from Zabbix Git repository or find them in the *zabbix/templates* directory of the downloaded latest Zabbix version. Then, while in *Configuration* → *Templates* you can import them manually into Zabbix.

Notifications Webhook integrations

New webhook media type for pushing Zabbix notifications to Mantis Bug Tracker has been added.

Installation Support for ARM64/AArch64

ARM64/AArch64 installation packages are now available for Debian, RHEL 8, 9 and its derivatives, as well as SLES/OpenSUSE Leap 15.

27 What's new in Zabbix 6.0.22

Aggregate functions

The **last_foreach** function is now also supported in the following aggregate functions: kurtosis, mad, skewness, stddevpop, stddevsamp, sumofsquares, varpop, and varsamp.

Return value limit

The return value limit for receiving data from external sources (such as scripts or other programs) has been raised to 16MB. This affects user parameters, remote commands, SSH checks, external checks, scripts, system.run[] item, and vfs.file.contents[] item.

Templates New template is available:

- MantisBT by HTTP

You can get this template:

- In *Configuration* → *Templates* in new installations;

- If you are upgrading from previous versions, you can download new templates from Zabbix Git repository or find them in the *zabbix/templates* directory of the downloaded latest Zabbix version. Then, while in *Configuration* → *Templates* you can import them manually into Zabbix.

28 What's new in Zabbix 6.0.23

Databases Supported versions

PostgreSQL 16 and **MariaDB 11.1** are now supported.

Plugins New item for PostgreSQL Zabbix agent 2 plugin

New item, **pgsql.version**, has been added to PostgreSQL Zabbix agent 2 plugin. This item is used for returning the PostgreSQL version.

Templates New templates

New template is available:

- Nextcloud by HTTP

You can get this template:

- In *Configuration* → *Templates* in new installations;
- If you are upgrading from previous versions, you can download new templates from Zabbix Git repository or find them in the *zabbix/templates* directory of the downloaded latest Zabbix version. Then, while in *Configuration* → *Templates* you can import them manually into Zabbix.

Updated templates

PostgreSQL by ODBC and PostgreSQL by Zabbix agent 2 templates now include the item and trigger for monitoring PostgreSQL version.

Frontend Miscellaneous

The *Clear history* button located in *Configuration* → *Hosts* → *Items* has been renamed *Clear history and trends* to more accurately describe its function, which is the same as the *Clear history and trends* button in the item configuration form.

In trigger action configuration, the condition type *Trigger name* has been renamed *Event name* to better describe its function. Note that by default, the event name matches the trigger name unless a custom event name is specified in trigger configuration.

29 What's new in Zabbix 6.0.24

Databases TimescaleDB 2.12 support

Support for TimescaleDB version 2.12 is now available.

Plugins New item in Zabbix agent 2 plugin

The item for returning the database server version is now available in MongoDB plugin (`mongodb.version`).

Items Content conversion to UTF-8

HTTP agent items, web scenarios, web checks and JavaScript items have been improved to convert to UTF-8 from the character set specified in the HTTP header or HTTP meta tag.

Templates New templates

New template is available:

- HPE iLO by HTTP

You can get this template:

- In *Configuration* → *Templates* in new installations;

- If you are upgrading from previous versions, you can download new templates from Zabbix Git repository or find them in the *zabbix/templates* directory of the downloaded latest Zabbix version. Then, while in *Configuration* → *Templates*, you can import them manually into Zabbix.

Updated templates

Integration with OpenShift has been added to Kubernetes cluster state by HTTP template.

30 What's new in Zabbix 6.0.25

This minor version has no functional changes.

TimescaleDB 2.13 support

Support for TimescaleDB version 2.13 is now available.

Items Additional ssh.run options

The **ssh.run[]** item has been updated and now allows passing additional SSH options as a part of the item key. These options are supported only using libssh of version 0.9.0 and higher or libssh2. Supported option keys and values depend on the SSH library. See SSH checks for details.

Plugins Cache mode parameter for PostgreSQL plugin

New parameters for controlling the cache mode by default or on session name level have been added to the PostgreSQL plugin configuration:

- `Plugins.PostgreSQL.Default.CacheMode`
- `Plugins.PostgreSQL.Sessions.<SessionName>.CacheMode`

The cache mode parameter may have one of two allowed values: *prepare* (default) or *describe*. Note that "describe" is primarily useful when the environment does not allow prepared statements such as when running a connection pooler like PgBouncer.

31 What's new in Zabbix 6.0.26

Templates New templates

The set of Azure by HTTP templates has been supplemented with the Azure Cost Management by HTTP template.

You can get this template:

- In *Configuration* → *Templates* in new installations;
- If you are upgrading from previous versions, you can download new templates from Zabbix Git repository or find them in the *zabbix/templates* directory of the downloaded latest Zabbix version. Then, while in *Configuration* → *Templates*, you can import them manually into Zabbix.

Updated templates

MSSQL by ODBC template:

- new item has been added - MSSQL DB '{#DBNAME}': Recovery model, which returns the database recovery model under the database discovery;
- new macros, namely, {$MSSQL.BACKUP_FULL.USED}, {$MSSQL.BACKUP_DIFF.USED}, {$MSSQL.BACKUP_LOG.USED}, have been added - those can be used for disabling backup age triggers for a certain database;
- LLD rules for quorum and quorum members discovery have been added;
- the type of the LLD rules has been changed from "Database monitor" to "Dependent item";
- items with `db.odbc.discovery` key have been turned into items dependent on `db.odbc.get` item.

Frontend PHP support

The maximum supported version for PHP is now 8.3.

X-Frame-Options HTTP header

The *X-Frame-Options header* parameter has been renamed to *Use X-Frame-Options header*, now consists of a checkbox and an input field (allowing you to disable the header by unmarking a checkbox instead of specifying "null" in the input field), and supports additional values.

Other security parameters now also follow the same structure. For more information, see the security parameters in *Administration → General*.

Databases MySQL 8.2 support

The maximum supported version for MySQL is now 8.2.X.

32 What's new in Zabbix 6.0.27

Databases MySQL 8.3 support

The maximum supported version for MySQL is now 8.3.X.

MariaDB 11.2 support

The maximum supported version for MariaDB is now 11.2.X.

Plugins MSSQL

A new plugin for direct monitoring of MSSQL by Zabbix agent 2 has been added.

For more information, see:

- MSSQL plugin readme
- Agent 2 items
- MSSQL plugin parameters

Templates New templates

A new template is available:

- YugabyteDB by HTTP, which includes the *YugabyteDB Cluster by HTTP* template for monitoring each YugabyteDB cluster.

You can get this template:

- In *Configuration → Templates* in new installations;
- If you are upgrading from previous versions, you can download new templates from the Zabbix Git repository or find them in the *zabbix/templates* directory of the latest Zabbix version you have downloaded. Then, while in *Configuration → Templates*, you can import them manually into Zabbix.

Platforms Zabbix agent 2 support on Windows

To prevent critical security vulnerabilities, the minimum Windows version for Zabbix agent 2 has been raised to Windows 10/Windows Server 2016. See note under Supported platforms for more information.

33 What's new in Zabbix 6.0.28

Databases TimescaleDB 2.14 support

The maximum supported version for TimescaleDB is now 2.14.X.

Templates New templates

A new template is available:

- MSSQL by Zabbix agent 2

You can get this template:

- In *Configuration → Templates* in new installations;

- If you are upgrading from previous versions, you can download new templates from the Zabbix Git repository or find them in the *zabbix/templates* directory of the latest Zabbix version you have downloaded. Then, while in *Configuration* → *Templates*, you can import them manually into Zabbix.

34 What's new in Zabbix 6.0.29

Databases MariaDB 11.3 support

The maximum supported version for MariaDB is now 11.3.X.

Templates New templates

A new template is available:

- Oracle Cloud by HTTP, a master template that discovers various Oracle Cloud Infrastructure (OCI) services and resources.

You can get this template:

- In *Configuration* → *Templates* in new installations;
- If you are upgrading from previous versions, you can download new templates from the Zabbix Git repository or find them in the *zabbix/templates* directory of the latest Zabbix version you have downloaded. Then, while in *Configuration* → *Templates*, you can import them manually into Zabbix.

Updated templates

- FortiGate by SNMP template has been supplemented with metrics regarding VPN, high availability (HA), wireless termination points (WTPs), SD-WAN health checks, and HW sensors.

35 Zabbix 6.0.30 新功能

前端 前端语言

现在前端已启用荷兰语、格鲁吉亚语和西班牙语。

插件 Ember+

已添加用于通过 Zabbix agent 2 直接监控 Ember+ 的新插件。

更多信息，请参见：

- Ember+ 插件自述文件
- Agent 2 监控项
- Ember+ 插件参数
- Agent 2 安装

模板 新模板

AWS ELB 模板集已补充了模板AWS ELB 网络负载均衡器（通过 HTTP）。

您可以通过以下方式获取此模板：

- 在新安装中，在配置 → 模板中；
- 如果您是从以前的版本进行升级，您可以从 Zabbix 的Git 仓库下载新模板，或者在您下载的最新 Zabbix 版本的 *zabbix/templates* 目录中找到它们。然后，在配置 → 模板中，您可以手动将它们导入 Zabbix。

更新的模板

OS 模板（基于 agent、SNMP 和 Prometheus）已更新挂载文件系统。在挂载文件系统发现中，“空间不足” 和 “空间严重不足” 触发器不再具有绝对阈值和突发条件。

宏 电子邮件媒体类型的宏支持

在配置电子邮件媒体类型时，现在可以在用户名和密码字段中使用宏。

36 Zabbix 6.0.31 新功能

Items More secure JavaScript preprocessing

The JavaScript function atob now returns an array of 8-bit unsigned integers instead of a decoded string.

前端 预处理测试结果的截断

在测试预处理步骤时，现在发送到前端的测试结果将被截断到最大 512KB 的大小。请注意，大于 512KB 的数据仍会由 Zabbix 服务器完全处理。

GSM modem validation for SMS media type

In SMS media type configuration, the GSM modem path is now validated to be a modem device or symlink to such.

模板 新模板

有一个新模板可供使用：

- Jira Data Center by JMX，用于监控 Jira Data Center 健康状况的模板。

您可以通过以下方式获取此模板：

- 在新安装中，进入配置 → 模板；
- 如果您是从以前的版本升级，可以从 Zabbix 的Git 仓库下载新模板，或者在您下载的最新 Zabbix 版本的 *zabbix/templates* 目录中找到它们。然后，在配置 → 模板中，您可以手动将它们导入到 Zabbix 中。

37 Zabbix 6.0.32 新功能

数据库 MySQL 8.4 支持

MySQL 的支持版本现在最高为 8.4.X。

MariaDB 11.4 支持

MariaDB 的支持版本现在最高为 11.4.X。

TimescaleDB 2.15 支持

TimescaleDB 的支持版本现在最高为 2.15.X。

模板 新模板

Azure by HTTP 模板集合中新增了 Azure VM Scale Set by HTTP 模板。

您可以获取此模板：

- 在新安装的 配置 → 模板中；
- 如果您是从以前的版本升级，您可以从 Zabbix Git 仓库 下载新模板或在下载的最新 Zabbix 版本的 *zabbix/templates* 目录中找到它们。然后，在 配置 → 模板中，您可以手动将它们导入到 Zabbix 中。

Updated templates

The templates Zabbix server health, Remote Zabbix server health, Zabbix proxy health, and Remote Zabbix proxy health have been updated for improved data visualization in item graphs by regrouping the displayed metrics.

38 What's new in Zabbix 6.0.33

Databases MySQL 9.0 support

The maximum supported version for MySQL is now 9.0.X.

Templates New templates

- AWS Lambda by HTTP, a template for monitoring AWS Lambda metrics.

You can get this template:

- In *Configuration* → *Templates* in new installations;
- If you are upgrading from previous versions, you can download new templates from Zabbix Git repository or find them in the *zabbix/templates* directory of the downloaded latest Zabbix version. Then, while in *Configuration* → *Templates*, you can import them manually into Zabbix.

39 What's new in Zabbix 6.0.34

Databases TimescaleDB 2.16 support

The maximum supported version for TimescaleDB is now 2.16.X.

MariaDB 11.5 support

The maximum supported version for MariaDB is now 11.5.X.

Notifications Webhook integrations

A new integration is available that allows to integrate Zabbix with the MS Teams workflow feature using the Zabbix webhook media type.

Items Special characters supported in ODBC check user password

Special characters are now supported when specifying the ODBC check user password in the frontend.

Templates New templates

- GitHub repository by HTTP, a template for monitoring GitHub repositories.

You can get this template:

- In *Configuration* → *Templates* in new installations;
- If you are upgrading from previous versions, you can download new templates from Zabbix Git repository or find them in the *zabbix/templates* directory of the downloaded latest Zabbix version. Then, while in *Configuration* → *Templates*, you can import them manually into Zabbix.

40 What's new in Zabbix 6.0.35

Items Internal item for monitoring high-availability manager

The `ha manager` (high-availability manager) process now can be monitored by the zabbix[process*] internal item.

Templates New templates

- Huawei OceanStor V6 by SNMP, a template for monitoring SAN Huawei OceanStor V6.

Updated templates

- In all APC templates, the discovery rule for external defective battery packs has been supplemented with a filter to avoid creating unsupported items.
- In the templates MSSQL by Zabbix agent 2 and MSSQL by ODBC, a timeout has been added and certain item keys have been adjusted to increase monitoring stability.

You can get these templates:

- In *Configuration* → *Templates* in new installations;
- If you are upgrading from previous versions, you can download new templates from Zabbix Git repository or find them in the *zabbix/templates* directory of the downloaded latest Zabbix version. Then, while in *Configuration* → *Templates*, you can import them manually into Zabbix.

41 What's new in Zabbix 6.0.36

TimescaleDB 2.17 support

The maximum supported version for TimescaleDB is now 2.17.X.

PostgreSQL 17 support

PostgreSQL 17 is now supported.

MSSQL instance names supported in agent 2 plugin

It is now possible to specify MSSQL instance names as part of the connection string in MSSQL agent 2 plugin items, for example:

```
mssql.custom.query[sqlserver://localhost/InstanceName,...]
```

Note that in this case **no port** must be specified. If port is specified, then the instance name will be ignored and the specified port will be used for connection.

Performance counter query updated in MSSQL agent 2 plugin

The performance counter query of the MSSQL agent 2 plugin has been updated, resolving issues in the data retrieval of unnamed (default) MSSQL instances.

New templates

- Nutanix Prism Element by HTTP, a template set that also includes the templates *Nutanix Cluster Prism Element by HTTP* and *Nutanix Host Prism Element by HTTP*.

You can get this template:

- In *Configuration* → *Templates* in new installations;
- If you are upgrading from previous versions, you can download new templates from Zabbix Git repository or find them in the *zabbix/templates* directory of the downloaded latest Zabbix version. Then, while in *Configuration* → *Templates*, you can import them manually into Zabbix.

Updated templates

The template *VMware Hypervisor* within the VMware and VMware FQDN template sets has been updated with the option to be used as a standalone template.

42 What's new in Zabbix 6.0.37

This minor version does not have any functional changes.

43 What's new in Zabbix 6.0.38

This minor version does not have any functional changes.

44 What's new in Zabbix 6.0.39

TimescaleDB 2.18 support

The maximum supported version for TimescaleDB is now 2.18.X.

New templates

- The Azure by HTTP template set has been supplemented with the template *Azure SQL Managed Instance by HTTP*.

You can get this template:

- In *Configuration* → *Templates* in new installations;

- When upgrading from previous versions, the latest templates can be downloaded from the Zabbix Git repository and manually imported into Zabbix in the *Configuration* → *Templates* section. If a template with the same name already exists, check the *Delete missing* option before importing to achieve a clean import. This way the items that have been excluded from the updated template will be removed (note that history of the deleted items will be lost).

Updated templates

- All Dell HTTP and SNMP templates have been updated with improvements, including fixes for items, macros, and scripts.

45 What's new in Zabbix 6.0.40

New templates

- The template set Zabbix server health has been supplemented with the templates *Zabbix server health by Zabbix agent*, *Zabbix server health by Zabbix agent active*, and Zabbix proxy health - with *Zabbix proxy health by Zabbix agent* and *Zabbix proxy health by Zabbix agent active*, enabling the monitoring of internal Zabbix metrics via Zabbix agent.

You can get this template:

- In *Configuration* → *Templates* in new installations;
- When upgrading from previous versions, the latest templates can be downloaded from the Zabbix Git repository and manually imported into Zabbix in the *Configuration* → *Templates* section. If a template with the same name already exists, check the *Delete missing* option before importing to achieve a clean import. This way the items that have been excluded from the updated template will be removed (note that history of the deleted items will be lost).

46 What's new in Zabbix 6.0.41

TimescaleDB 2.19 support

The maximum supported version for TimescaleDB is now 2.19.X.

2. 定义

概览　在本节中，您可以了解 Zabbix 中一些常用术语的含义。

定义　***host***（主机） - 要通过 *IP/DNS* 监控的联网设备。

host group（主机组） - 主机的逻辑分组；它可能包含主机和模板。主机组中的主机和模板没有以任何方式相互链接。在为不同用户组分配主机访问权限时使用主机组。

item（监控项） - 你想要接收的主机的特定数据，一个度量/指标数据。

value preprocessing（值预处理） - 在数据存入数据库之前转化/预处理接收到的指标数据。*

trigger（触发器） - 一个被用于定义问题阈值和" 评估" 控项接收到的数据的逻辑表达式。当接收到的数据高于阈值时，触发器从'Ok' 变成'Problem' 状态。当接收到的数据低于阈值时，触发器保留/返回'Ok' 的状态。

event（事件） - 一次发生的需要注意的事情，例如触发器状态改变、自动发现/*agent* 自动注册。

event tag（事件标签） - 预设的事件标记可以被用于事件关联，权限细化设置等。

event correlation（事件关联） - 一种灵活而精确地将问题与其解决方法联系起来的方法比如说，你可以定义触发器 A 告警的异常可以由触发器 B 解决，触发器 B 可能采用完全不同的数据采集方式。

problem（问题） - 一个处在" 问题" 状态的触发器。

problem update（问题更新） - *Zabbix* 提供的问题管理选项，例如添加评论、确认、更改严重性或手动关闭。

action（动作） - 对事件作出反应的预先定义的方法。一个动作由多个操作（例如发送通知）和条件（什么情况下执行操作）组成。

escalation（升级） - 用于在动作中执行操作的自定义场景；发送通知/执行远程命令的序列。

media（媒体） - 发送告警通知的渠道；传输媒介。

notification（通知） - 通过选定的媒体通道发送给用户的关于某个事件的消息。

remote command（远程命令） - 在某些条件下在受监控主机上自动执行的预定义命令。

template（模板） - 可以应用于一个或多个主机的一组实体集（包含监控项、触发器、图表、低级别自动发现规则、*web* 场景等）。模版的应用使得主机上的监控任务部署快捷方便；也可以使监控任务的批量修改更加简单。模版是直接关联到每台单独的主机上。

web scenario（***web*** 场景） - 检查一个网站的可用性的一个或多个 *HTTP* 请求。

frontend（前端） - *Zabbix* 的 *web* 界面。

dashboard（仪表盘） - *web* 界面的可定制部分，以可视化的单元（又叫小部件）显示重要信息的摘要和可视化。

widget（小部件） - 在仪表板中使用的显示某种类型和来源的信息（摘要、地图、图表、时钟等）的可视化单元。

Zabbix API - *Zabbix API* 允许您使用 *JSON RPC* 协议来创建、更新和获取 *Zabbix* 对象（如主机、监控项、图表等）或执行任何其他自定义任务。

Zabbix server - *Zabbix* 软件的中央进程，执行监控、与 *Zabbix proxy* 和 *agent* 交互、计算触发器、发送通知；数据的中央存储库。

Zabbix proxy - 一个可以代表 *Zabbix server* 收集数据的进程，减轻 *server* 的一些处理负载。

Zabbix agent - 部署在被监控目标上以主动监控本地资源和应用程序的进程。

Zabbix agent 2 - 新一代 *Zabbix agent*，主动监控本地资源和应用程序，允许使用自定义插件进行监控。

> **Attention:**
> 因为 Zabbix agent 2 与 Zabbix agent 共享许多功能，所以如果功能行为相同，文档中的术语"Zabbix agent" 同时代表 Zabbix agent 和 Zabbix agent 2。Zabbix agent 2 仅在其功能不同的地方特别命名。

encryption（加密） - 使用传输层安全 (TLS) 协议 支持 *Zabbix* 组件（*server*，*proxy*，*agent*，*zabbix_sender* 和 *zabbix_get* 实用程序）之间的加密通信。

network discovery（网络发现） - 网络设备的自动发现。

low-level discovery（低级别自动发现） - 自动发现特定设备上的底层实体（例如：文件系统、网络接口等）

low-level discovery rule（自动别自动发现规则） - 用于在设备上自动发现低级别实体的一组定义。

item prototype（监控项原型） - 带有某些参数作为变量的度量，为低级别自动发现做好了准备。在低级别自动发现之后，变量被自动替换为实际发现的参数，度量标准自动开始收集数据。

trigger prototype（触发器原型） - 以某些参数作为变量的触发器，为低级别自动发现做好准备。在低级别自动发现之后，变量被自动替换为实际发现的参数，触发器自动开始计算数据。

其他一些 Zabbix 实体的 原型也在低级别自动发现中使用 - 图形原型、主机原型、主机组原型。

agent autoregistration（***agent*** 自动注册） - *Zabbix agent* 自动注册为主机并开始监控的自动化过程。

3. Zabbix 进程

请使用侧边导航栏来访问此章节中的内容。

1 Server

概述

Zabbix server 是整个 Zabbix 软件的核心程序。

Zabbix Server 负责执行数据的主动轮询和被动获取，计算触发器条件，向用户发送通知。它是 Zabbix Agent 和 Proxy 报告系统可用性和完整性数据的核心组件。Server 自身可以通过简单服务远程检查网络服务（如 Web 服务器和邮件服务器）。

Zabbix Server 是所有配置、统计和操作数据的中央存储中心，也是 Zabbix 监控系统的告警中心。在监控的系统中出现任何异常，将发出通知给管理员。

基本的 Zabbix Server 的功能分解成为三个不同的组件。他们是：Zabbix server、Web 前端和数据库。

Zabbix 的所有配置信息都存储在 Server 和 Web 前端进行交互的数据库中。例如，当你通过 Web 前端（或者 API）新增一个监控项时，它会被添加到数据库的监控项表里。然后，Zabbix server 以每分钟一次的频率查询监控项表中的有效项，接着将它存储在 Zabbix server 中的缓存里。这就是为什么 Zabbix 前端所做的任何更改需要花费两分钟左右才能显示在最新的数据段的原因。

服务进程

通过二进制包安装的组件

Zabbix server 进程以守护进程（Deamon）运行。Zabbix server 的启动可以通过执行以下命令来完成：

```
shell> service zabbix-server start
```

上述命令在大多数的 GNU/Linux 系统下都可以正常完成。如果是其他系统，你可能要尝试以下命令来运行：

```
shell> /etc/init.d/zabbix-server start
```

类似的，停止、重启、查看状态，则需要执行以下命令：

```
shell> service zabbix-server stop
shell> service zabbix-server restart
shell> service zabbix-server status
```

手动启动

如果以上操作均无效，您可能需要手动启动，找到 Zabbix Server 二进制文件的路径并且执行：

```
shell> zabbix_server
```

您可以将以下命令行参数用于 Zabbix server：

```
-c --config <file>              配置文件路径（默认/usr/local/etc/zabbix_server.conf）
-R --runtime-control <option>   执行管理功能
-h --help                       帮助
-V --version                    显示版本号
```

使用命令行参数运行 Zabbix server 的示例:：

```
shell> zabbix_server -c /usr/local/etc/zabbix_server.conf
shell> zabbix_server --help
shell> zabbix_server -V
```

运行时控制

运行时控制选项：

选项	描述	目标
config_cache_reload	重新加载配置缓存。如果当前正在加载缓存，则忽略。	
diaginfo[=<**target**>]	在服务器日志文件中收集诊断信息。	**historycache** - 历史缓存统计 **valuecache** - 值缓存统计 **preprocessing** - 预处理管理器统计信息 **alerting** - 警报管理器统计信息 **lld** - LLD 管理器统计信息 **locks** - 互斥锁列表（在 **BSD 系统 * 上为空）
ha_status	记录高可用性 (HA) 集群状态。	
ha_remove_node=target	删除由其列出的编号指定的高可用性 (HA) 节点。 请注意，无法删除活动/备用节点。	**target** - 列表中的节点编号(可以通过运行 ha_status 获得)
ha_set_failover_delay=delay	设置高可用性 (HA) 故障转移延迟。 支持时间后缀，例如 10 秒，1 分钟。	
secrets_reload	从 Vault 重新加载机密。	
service_cache_reload	重新加载服务管理器缓存。	
snmp_cache_reload	重新加载 SNMP 缓存，清除所有主机的 SNMP 属性（引擎时间、引擎启动、引擎 ID、凭据）。	
housekeeper_execute	启动管家程序。如果当前正在进行管家处理程序，则忽略。	
trigger_housekeeper_execute	启动触发器管家处理程序。如果触发管家处理程序当前正在进行，则忽略。	

选项	描述	目标
log_level_increase[=<**target**>]	增加日志级别，如果未指定 target，则影响所有进程。 在 **BSD* 系统上不受支持。	进程类型 - 指定类型的所有进程（例如，poller） 查看所有服务器进程类型。 **process type,N** - 进程类型和编号（例如，poller，3） **pid** - 进程标识符（1 到 65535）。对于较大的值，请将目标指定为 “process type,N”。
log_level_decrease[=<**target**>]	降低日志级别，如果未指定 target，则影响所有进程。 在 **BSD* 系统上不支持。	
prof_enable[=<**target**>]	启用分析。 如果未指定目标，则影响所有进程。 启用的分析按函数名称提供所有 rwlocks/mutex 的详细信息。 自 Zabbix 6.0.13 起支持。	进程类型 - 指定类型的所有进程（例如历史同步器） 支持的进程类型作为分析目标：警报器、警报管理器、可用性管理器、配置同步器、发现器、escalator , history poller, history syncer, housekeeper, http poller, icmp pinger, ipmi manager, ipmi poller, java poller, lld manager, lld worker, odbc poller, poller, preprocessing manager, preprocessing worker, proxy poller, 自监控, 服务管理器, snmp trapper, task manager, timer, trapper, unreachable poller, vmware collector **process type,N** - 进程类型和数量 (e.g., history syncer,1) **pid** - 进程标识符（1 到 65535）。对于较大的值，将目标指定为 “process type,N”。 **scope** - `rwlock`、`mutex`、`processing` 可以与进程类型和数量一起使用（例如，history syncer,1,processing)
prof_disable[=<**target**>]	禁用分析。 如果未指定目标，则影响所有进程。 自 Zabbix 6.0.13 起受支持。	进程类型 - 指定类型的所有进程（例如 history syncer） 支持的进程类型作为分析目标：参见 `prof_enable` **process type,N** - 进程类型和数量（例如，history syncer,1） **pid** - 进程标识符（1 到 65535）。对于较大的值，将目标指定为 “process type,N”。

使用运行时控制重新加载服务器配置缓存的示例：

```
shell> zabbix_server -c /usr/local/etc/zabbix_server.conf -R config_cache_reload
```

使用运行时控制收集诊断信息的示例：

在服务器日志文件中收集所有可用的诊断信息：

```
shell> zabbix_server -R diaginfo
```

在服务器日志文件中收集历史缓存统计信息：

```
shell> zabbix_server -R diaginfo=historycache
```

使用运行时控制重新加载 SNMP 缓存的示例：

```
shell> zabbix_server -R snmp_cache_reload
```

使用运行时控制触发管家执行的示例：

```
shell> zabbix_server -c /usr/local/etc/zabbix_server.conf -R housekeeper_execute
```

使用运行时控制更改日志级别的示例：

增加所有进程的日志级别：

```
shell> zabbix_server -c /usr/local/etc/zabbix_server.conf -R log_level_increase
```

增加第二个轮询进程的日志级别：

```
shell> zabbix_server -c /usr/local/etc/zabbix_server.conf -R log_level_increase=poller,2
```

使用 PID 1234 增加进程的日志级别：

```
shell> zabbix_server -c /usr/local/etc/zabbix_server.conf -R log_level_increase=1234
```

降低所有 http poller 进程的日志级别：

```
shell> zabbix_server -c /usr/local/etc/zabbix_server.conf -R log_level_decrease="http poller"
```

将 HA 故障转移延迟设置为最短 10 秒的示例：

```
shell> zabbix_server -R ha_set_failover_delay=10s
```

进程用户

Zabbix server 允许使用非 root 用户运行。它将以任何非 root 用户的身份运行。因此，使用非 root 用户运行 server 是没有任何问题的.

如果你试图以 “root” 身份运行它，它将会切换到一个已经 “写死” 的 “zabbix” 用户，您可以参考安装 章节。按此相应地修改 Zabbix server 配置文件中的 “AllowRoot” 参数，则可以只以 “root” 身份运行 Zabbix server。

如果 Zabbix server 和agent 均运行在同一台服务器上，建议您使用不同的用户运行 server 和 agent 。否则，，如果两者都以相同的用户运行，Agent 可以访问 Server 的配置文件，任何 Zabbix 管理员级别的用户都可以很容易地检索到 Server 的信息。例如，数据库密码。

配置文件

有关配置 Zabbix server 的详细信息，请查阅配置文件 章节。

启动脚本

这些脚本用于在系统启动和关闭期间自动启动和停止 Zabbix 进程。此脚本位于 misc/init.d 目录下。

服务器进程类型

- alert manager - 警报队列管理器
- alert syncer - 警报数据库编写器
- alerter - - 发送通知的进程
- availability manager - 主机可用性更新进程
- configuration syncer -管理配置数据的内存缓存进程
- discoverer - 发现设备进程
- escalator - 升级操作进程
- history poller - 处理需要数据库连接的计算和内部检查进程
- history syncer - 历史数据库编写器
- housekeeper - 删除旧历史数据进程
- http poller - web 监控轮询器
- icmp pinger - 用于 icmp ping 检查的轮询器
- ipmi manager - IPMI 轮询管理器
- ipmi poller - IPMI 检查的轮询器
- java poller - 用于 Java 检查的轮询器
- lld manager - 低级别发现任务的管理进程
- lld worker - 低级别发现任务的工作进程
- odbc poller - 用于 ODBC 检查的轮询器
- poller - 用于被动检查的普通轮询器
- preprocessing manager - 预处理任务管理器
- preprocessing worker - 数据预处理进程
- problem housekeeper - 消除已删除触发器问题进程
- proxy poller - 被动代理轮询器

- `report manager`- 定时报表任务管理器
- `report writer` - 定时报表处理进程
- `self-monitoring` - 收集内部服务器统计数据的进程
- `snmp trapper` - SNMP 陷阱 trapper
- `task manager` - 远程执行其他组件请求的任务进程（例如关闭问题、确认问题、立即检查监控项值、远程命令功能）
- `timer` - 处理维护的计时器
- `trapper` - 用于主动检查、陷阱、代理通信的 trapper
- `unreachable poller` - 不可达设备的轮询器
- `vmware collector` - VMware 数据收集器，负责从 VMware 服务收集数据

服务器日志文件可用于观察这些进程类型。

可以使用 **zabbix[process,<type>,<mode>,<state>]** 内部监控项来监控各种类型的 Zabbix 服务器进程。

支持的平台

由于服务器操作的安全性要求和任务关键性，UNIX 是唯一能够始终如一地提供必要性能、容错和弹性的操作系统。Zabbix 以市场主流的操作系统版本运行。

经测试，Zabbix 可以运行在下列平台：

- Linux
- Solaris
- AIX
- HP-UX
- Mac OS X
- FreeBSD
- OpenBSD
- NetBSD
- SCO Open Server
- Tru64/OSF1

Note:
Zabbix 可以运行在其他类 Unix 操作系统上。

语言环境

值得注意的是，Zabbix server 需要 UTF-8 语言环境，以便可以正确解释某些文本项。大多数现代类 Unix 系统都默认使用 UTF-8 语言环境，但是，有些系统可能需要做特定的设置。

1 高可用

概述

通常高可用性 (HA) 需要在几乎不需要停机的关键基础设施中使用。为了在服务出现任何失败故障时进行故障转移，进行接管。

Zabbix 提供了一个本地的高可用性解决方案，方便设置，不需要任何 HA 专业知识也可以完成。本地 Zabbix HA 对于防止 Zabbix server 的软件/硬件故障或减少维护停机时间是有用的。

Zabbix 高可用模式下，多台 Zabbix server 作为集群中的节点运行。当集群中的一个 Zabbix server 处于 active 时，其他服务器处于 standy，随时准备在必要时接管。



切换到 Zabbix HA 还不确定。您可以在任何时候切换回 standalone 状态。

参见: 实施详细

启用高可用集群

作为集群节点启动 Zabbix server

在 server配置中需要两个参数来启动 Zabbix server 作为集群节点:

- **HANodeName** 集群中每个 zabbix server 节点名称必须要有且唯一。

这是将在 agent 和 proxy 配置中引用 server 的名称 (例如：zabbix-node-01)。如果您没有指定 HANodeName，那么服务器将以 standalone 模式启动。

- **NodeAddress** 参数必须为每个节点指定。

NodeAddress 参数 (address:port) 将被 Zabbix 前端用来连接到主 server 节点。NodeAddress 必须匹配相应 Zabbix server 的 IP 或 FQDN 名称。

在对配置文件进行更改后，重新启动所有 Zabbix server。- 它们现在将作为集群节点启动。- server 的新状态可以在 报表 → 系统信息 查看，也可以用以下命令查看:

```
zabbix_server -R ha_status
```

此运行时命令会将当前 HA 集群状态记录到 Zabbix 服务器日志中（并输出到标准输出）：

```
Failover delay: 60 seconds
Cluster status:
   #  ID                        Name                 Address                 Status      Last Access
   1. ckzxxqg7u0001lsropenyzh3m zabbix-node-01       64.227.66.193:10051     standby     0s
   2. ckzxyqo1k00013frpq539e1jp zabbix-node-02       64.227.74.25:10051      active      3s
```

准备前端

确保 Zabbix server 地址: 端口在前端配置中没有定义 (在 frontend 文件目录的 `conf/zabbix.conf.php` 中找到)。

```
// Uncomment and set to desired values to override Zabbix hostname/IP and port.
// $ZBX_SERVER                  = '';
// $ZBX_SERVER_PORT             = '';
```

Zabbix 前端将通过读取 Zabbix 数据库中节点表的设置来自动检测活动节点。使用主节点的节点地址作为 Zabbix server 地址。

代理配置

HA 集群节点（服务器）必须列在被动或主动 Zabbix 代理的配置中。

对于被动代理，节点名称必须列在代理的服务器参数 中，以逗号分隔。

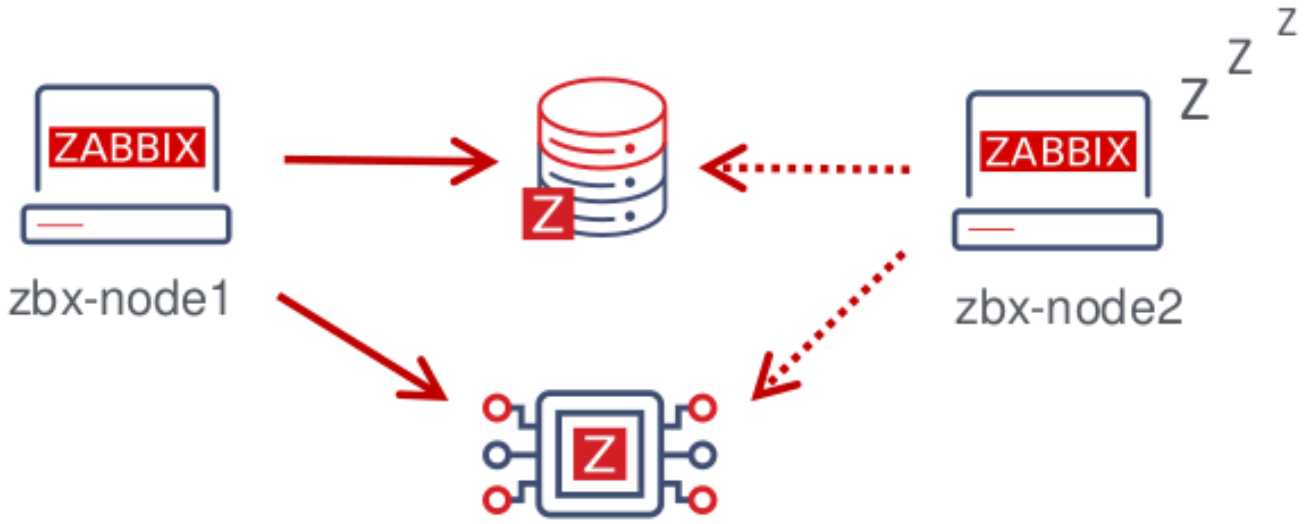
```
Server=zabbix-node-01,zabbix-node-02
```

对于主动代理，节点名称必须列在代理的服务器参数中，以分号分隔。

```
Server=zabbix-node-01;zabbix-node-02
```

Agent 配置

Zabbix agent 或者 Zabbix agent 2. 必须将 HA 集群节点列入配置文件。



启用被动检查 agent, 节点名称必须列入 Server参数, 使用逗号分隔.

```
Server=zabbix-node-01,zabbix-node-02
```

启用主动检查 agent, 节点名称必须列入 ServerActive参数. 注意，对于主动检查 agent，节点与其他 server 之间必须用逗号分隔，而节点本身必须用分号分隔，例如:

```
ServerActive=zabbix-node-01;zabbix-node-02
```

故障切换到备用节点

如果主节点停止，Zabbix 将自动故障转移到另一个节点。要发生故障转移，必须至少有一个节点处于备用状态。

故障转移会有多快? 所有节点每 5 秒更新一次他们的最后访问时间 (和状态，如果它被更改)。因此:

- 如果主节点关闭并成功报告其状态为 “stopped”，另一个节点将在 **5** 秒内接管。
- 如果主节点关闭/变得不可用，而无法更新其状态，备用节点将等待故障转移延迟 + 5 秒来接管。

故障转移延迟是可配置的，支持的范围在 10 秒到 15 分钟之间 (默认为 1 分钟)。要修改故障切换延迟，可以执行以下命令:

```
zabbix_server -R ha_set_failover_delay=5m
```

管理 HA 集群

可以使用专用的运行时控制 选项管理 HA 集群的当前状态：

- ·`ha_status` - 在 Zabbix 服务器日志中记录 HA 集群状态（并输出到标准输出）
- ·`ha_remove_node=target` - 删除由其 <target> 标识的 HA 节点 - 列表中节点的编号（该编号可以从运行 ha_status 的输出中获得），例如：

```
zabbix_server -R ha_remove_node=2
```

请注意，主备节点不能被删除。

- ·`ha_set_failover_delay=delay` - 设置 HA 故障转移延迟（10 秒到 15 分钟之间；支持时间后缀，例如 10s、1m）

可以监控节点状态：

- ·在报告 → 系统信息
- ·在系统信息仪表板小部件中
- ·使用服务器的 `ha_status` 运行时控制选项（见上文）。

`zabbix[cluster,discovery,nodes]` 内部监控项可用于节点发现，因为它返回具有高可用性节点信息的 JSON。

禁用 HA 集群

禁用 HA 高可用集群:

- 备份配置文件
- 停止 standby 节点
- 移除 HANodeName 参数从活跃的主 server 上
- 重启主 server (它将以 standalone 模式启动)

实施细节

Zabbix server 支持高可用性 (HA) 集群是一个可选择的解决方案。本地 HA 解决方案被设计为易于使用，它可以跨站点工作，并且对 Zabbix 识别的数据库没有特定的要求。用户可以自由地使用本地 Zabbix HA 解决方案或第三方 HA 解决方案，这取决于什么最适合其环境中的高可用性需求。

该解决方案由多个 zabbix_server 实例或节点组成。

每一个节点: - 单独配置 - 使用相同的数据库 - 可能有几种模式: active, standby, unavailable, stopped

一次只能有一个节点处于活动状态 (工作)。备节点只运行一个进程——HA 管理器。备用节点不进行数据收集、处理或其他常规 server 活动; 它不监听端口; 它们拥有最少的数据库连接。

主节点和备节点每 5 秒更新一次上次访问时间。每个备节点监控主节点的最后一次访问时间。如果主节点的最后一次访问时间超过了 “故障转移延迟” 秒，备用节点将自己切换为主节点，并将 “不可用” 状态分配给先前的主节点。

主节点监视自己的数据库连接—如果丢失超过 “故障转移延迟-5” 秒，它必须停止所有处理并切换到备用模式。主节点还监视备用节点的状态——如果备用节点的最后访问时间超过了 “故障转移延迟” 秒，备用节点将被分配为 “不可用” 状态。

这些节点被设计成跨较小的 Zabbix 版本兼容。

2 Agent

概述

Zabbix agent 部署在被监控目标上，以主动监控本地资源和应用程序（硬盘、内存、处理器统计信息等）。

Zabbix agent 收集本地的操作信息并将数据报告给 Zabbix server 用于进一步处理。一旦出现异常 (例如硬盘空间已满或者有崩溃的服务进程)，Zabbix server 会主动警告管理员指定机器上的异常。

Zabbix agents 的极高效率缘于它可以利用本地系统调用来完成统计数据的采集。

被动和主动检查

Zabbix agent 可以运行被动检查和主动检查。

在被动检查 模式中 agent 应答数据请求。Zabbix server（或 proxy）询求数据，例如 CPU load，然后 Zabbix agent 返还结果。

主动检查 处理过程将相对复杂。Agent 必须首先从 Zabbix sever 索取监控项列表以进行独立处理，然后会定期发送采集到的新值给 Zabbix server。

是否执行被动或主动检查是通过选择相应的监控项类型来配置的。Zabbix agent 处理 “Zabbix agent” 或 “Zabbix agent（active）” 类型的监控项。

支持的平台

Zabbix agent 支持 在以下平台上：

- Windows（自 XP 以来的所有桌面和服务器版本）
- Linux（也可在分发包）
- macOS
- IBM AIX
- FreeBSD
- OpenBSD
- Solaris

类 UNIX 系统上的 Agent

类 UNIX 系统上的 Zabbix agent 运行在被监控的主机上。

安装

有关通过二进制包安装 Zabbix agent 的详细信息，请查阅 以二进制包安装 章节。

此外，如果您不想使用二进制包，请查阅以源码包安装的说明。

> **Attention:**
> 通常，32 位 Zabbix agent 可以在 64 位系统上运行，但在某些情况下可能会失败。

通过二进制包安装的组件

Zabbix agent 进程以守护进程（Deamon）运行。Zabbix agent 的启动可以通过执行以下命令来完成：

shell> service zabbix-agent start

上述命令在大多数的 GNU/Linux 系统下都可以正常完成。如果是其他系统，你可能要尝试以下命令来运行：

shell> /etc/init.d/zabbix-agent start

类似的，停止、重启、查看状态，则需要执行以下命令：

shell> service zabbix-agent stop shell> service zabbix-agent restart shell> service zabbix-agent status

手动启动

如果以上操作均无效，您可能需要手动启动，找到 Zabbix agent 二进制文件的路径并且执行：

shell> zabbix_agentd

Windows 系统上的 Agent

Windows 系统上的 Zabbix agent 作为一个 Windows 服务运行。

准备

Zabbix agent 作为 zip 压缩文件分发。下载该文件后，您需要将其解压缩。选择任何文件夹来存储 Zabbix 代理和配置文件，例如：

C:\zabbix

复制二进制文件 bin\zabbix_agentd.exe 和配置文件 conf\zabbix_agentd.conf 到 c:\zabbix 下。

按需编辑 c:\zabbix\zabbix_agentd.conf 配置文件，确保指定了正确的”Hostname” 参数。

安装

完成此操作后，使用以下命令将 Zabbix agent 安装为 Windows 服务：

C:> c:\zabbix\zabbix_agentd.exe -c c:\zabbix\zabbix_agentd.conf -i

现在您可以像任何其他 Windows 服务一样配置 “Zabbix agent” 服务。

有关在 Windows 上安装和运行 Zabbix agent 的详细信息，请查阅于此。

其它 agent 选项

您可以在主机上运行单个或多个 Agent 实例。单个实例可以使用默认配置文件或命令行中指定的配置文件。如果是多个实例，则每个 Agent 程序实例必须具有自己的配置文件（其中一个实例可以使用默认配置文件）。

以下命令参数可以在 Zabbix agent 中使用：

参数	描述
UNIX and Windows agent	
-c --config <config-file>	配置文件的绝对路径。您可以使用此选项来制定配置文件，而不是使用默认文件。在 UNIX 上，默认的配置文件是 /usr/local/etc/zabbix_agentd.conf 或由编译时的 --sysconfdir 或 --prefix 变量来确定。在 Windows 上, 默认的配置文件是 c:\zabbix_agentd.conf
-p --print	输出已知的监控项并退出。注意：要返回 用户自定义参数 的结果，您必须指定配置文件（如果它不在默认路径下）。
-t --test <item key>	测试指定的监控项并退出。注意：要返回 用户自定义参数 的结果，您必须指定配置文件（如果它不在默认路径下）。
-h --help	显示帮助信息
-V --version	显示版本信息
仅 **UNIX agent**	
-R --runtime-control <option>	执行管理功能。请参阅 运行时机制的控制。
仅 **Windows agent**	
-m --multiple-agents	使用多个 Agent 实例（使用 -i、-d、-s、-x）。为了区分实例的服务名称，每项服务名都会包涵来自配置文件里的 Hostname 值。
** 仅 Windows agent （功能）**	
-i --install	以服务的形式安装 Zabbix Windows agent。
-d --uninstall	卸载 Zabbix Windows agent 服务。
-s --start	启动 Zabbix Windows agent 服务。
-x --stop	停止 Zabbix Windows agent 服务。

使用命令行参数的具体示例：

- 打印输出所有内置监控项和它们的值。
- 使用指定的配置文件中的 “mysql.ping” 键值来测试用户自定义参数。
- 在 Windows 下使用默认路径下的配置文件 c:\zabbix_agentd.conf 安装 Zabbix agent 服务。
- 使用位于与 agent 可执行文件同一文件夹中的配置文件 zabbix_agentd.conf 为 Windows 安装 “Zabbix Agent [Hostname]” 服务，并通过从配置文件中的唯一 Hostname 值扩来为命名。

shell> zabbix_agentd --print shell> zabbix_agentd -t ”mysql.ping” -c /etc/zabbix/zabbix_agentd.conf shell> zabbix_agentd.exe -i shell> zabbix_agentd.exe -i -m -c zabbix_agentd.conf

运行时控制

使用运行时控制选项，您可以更改 agent 进程的日志级别。

选项	描述	目标
log_level_increase[=<target>]	增加日志等级. 如果未指定目标，则所有进程都会受到影响。	目标可以指定为: **process type** - 指定类型的所有进程 (e.g., listener) 查看所有agent 进程类型。 **process type,N** - 进程类型和数量 (e.g., listener,3) **pid** - 进程 id (1 to 65535)。对于较大的值，请将目标指定为 “进程类型，N”。
log_level_decrease[=<target>]	降低日志级别 如果未指定目标，则所有进程都会受到影响。	
userparameter_reload	从当前配置文件重新加载用户参数 请注意，UserParameter 是将重新加载的唯一 agent 配置选项。	

例子：

- 给所有进程增加日志级别。
- 给第二个监听进程增加日志级别。
- 给 PID 号为 1234 的进程增加日志级别。
- 给所有主动检查进程降低日志级别。

```
shell> zabbix_agentd -R log_level_increase
shell> zabbix_agentd -R log_level_increase=listener,3
shell> zabbix_agentd -R log_level_increase=1234
shell> zabbix_agentd -R log_level_decrease="active checks"
```

Note:
运行时控制器不支持 OpenBSD, NetBSD andWindows.

Agent 进程类型

- `active checks` - 执行主动检查的进程
- `collector` - 执行数据收集的进程
- `listener` - 执行被动监控监听的进程

Agent 日志文件可用于观察这些进程类型。

进程用户

Zabbix agent 在 UNIX 上允许使用非 root 用户运行。它将以任何非 root 用户的身份运行。因此，使用非 root 用户运行 agent 是没有任何问题的.

如果你试图以 “root” 身份运行它，它将会切换到一个已经 “写死” 的 “zabbix” 用户，该用户必须存在于您的系统上。如果您只想以 “root” 用户运行 agent，您必须在 agent 配置文件里修改 ‘AllowRoot‘参数。

配置文件

有关配置 Zabbix agent 的详细信息，请查阅 zabbix_agentd 或 Windows agent 章节。

本地环境

值得注意的是，Zabbix agent 需要 UTF-8 语言环境，以便某些文本 Zabbix agent 监控项可以返回预期的内容。大多数现代类 Unix 系统都默认使用 UTF-8 语言环境，但是，有些系统可能需要特定的设置。

退出码

在 2.2 版本之前，Zabbix 代理在成功退出时返回 0，在失败时返回 255。从版本 2.2 和更高版本开始，Zabbix 代理在成功退出时返回 0，在失败时返回 1。

3 Agent 2

概述

Zabbix agent 2 是新一代的 Zabbix agent，可以替代 Zabbix agent 使用。Zabbix agent 2 已开发为：

- 减少 TCP 连接数量
- 提供改进的检查并发性
- 使用插件很容易扩展。一个插件应该能够:
  - 提供由几行简单代码组成的简单检查
  - 提供复杂的检查，包括长时间运行的脚本和独立的数据收集，并定期发回数据
- 做一个临时的替代品 Zabbix agent (因为它支持之前的所有功能)

Agent 2 是用 Go 语言编写 (复用了一些 Zabbix agent 中的 C 语言代码).。在构建 Zabbix agent 2 时，需要配置当前支持的 Go 环境 Go version 。

Agent 2 在 Linux 上没有内置的守护进程支持; 它可以作为 Windows service.

被动检查的工作类似于 Zabbix agent。主动检查支持调度/灵活间隔同时并发检查仅使用一个 active server.

Note:
默认情况下，Zabbix agent2 将在监控项更新间隔内以条件随机时间调度主动检查的第一次数据采集，避免资源使用率突增。为了在 agent 重启后立即做好非调度采集间隔 的主动检查，设置配置文件中 ForceActiveChecksOnStart 参数（全局）或者 Plugins.<Plugin name>.System.ForceActiveChecksOnStart（仅影响特定插件）。插件级别参数会覆盖全局参数。从 Zabbix6.0.2 版本开始，支持启动 agent 时强制执行主动检查。

检查并发

来自不同插件的检查可以同时执行。一个插件的并发检查次数受插件容量设置的限制。每个插件都可以有一个硬编码的容量设置 (默认为 100)，可以使用 `Plugins.<Plugin name>.Capacity=N` 设置 *Plugins* 配置 parameter.

参考: Plugin development guidelines.

支持的平台

以下平台支持 Zabbix agent 2：

- Windows（自 XP 以来的所有桌面和服务器版本；也可作为 预编译二进制文件)
- Linux（也可在分发包）

安装

Zabbix agent 2 在预编译的 Zabbix 包中可用。要从 源码 编译 Zabbix agent 2，您必须指定 `--enable-agent2` 配置选项。

操作

Zabbix agent 2 可以使用以下命令行参数:

参数	描述
-c --config <config-file>	配置文件的路径。 您可以使用此选项指定非默认的配置文件。 在 UNIX, 默认是 /usr/local/etc/zabbix_agent2.conf 或者编译时设置compile-time 变量*--sysconfdir* or *--prefix*
-f --foreground	在前台运行 Zabbix agent (默认: true).
-p --print	打印已知监控项并退出。 注意: 也返回用户参数结果, 必须指定配置文件 (如果它不在默认位置)。
-t --test <item key>	测试指定监控项并退出。 注意: 也返回用户参数 结果, 必须指定配置文件 (如果它不在默认位置)。
-h --help	打印帮助信息并退出。
-v --verbose	打印调试信息。将此选项与-p 和-t 选项一起使用。

参数	描述
-V --version	打印代理版本号并退出。
-R --runtime-control <option>	执行管理功能。查看runtime control.

特殊 例子使用命令行参数:

- 打印 agent 内置的所有监控项和值
- 使用指定配置文件中定义的 “mysql.ping” 键测试一个用户参数

```
shell> zabbix_agent2 --print
shell> zabbix_agent2 -t "mysql.ping" -c /etc/zabbix/zabbix_agentd.conf
```

运行时控制

运行时控制提供了一些远程控制选项。

操作	描述
log_level_increase	增加日志等级。
log_level_decrease	降低日志等级。
metrics	可用的指标列表。
version	显示 agent 版本。
userparameter_reload	从当前配置文件重新加载用户参数。 请注意 UserParameter 是将被重新加载的唯一 agent 配置选项。
help	显示运行时控制的帮助信息

例子:

- 为 agent 2 增加日志等级
- 打印运行时控制选项

```
shell> zabbix_agent2 -R log_level_increase
shell> zabbix_agent2 -R help
```

配置文件

agent 2 的配置参数大多与 Zabbix agent 兼容，但也有一些例外。

新的参数	描述
ControlSocket	实时控制套接字路径。Agent 2 使用控制套接字runtime commands.
EnablePersistentBuffer, *PersistentBufferFile*, *PersistentBufferPeriod*	这些参数用于在 agent 2 上为活动监控项配置持久存储.
ForceActiveChecksOnStart	确定 agent 是否应在重新启动后立即执行主动检查或随时间均匀分布。自 Zabbix 6.0.2 起支持。
Plugins	插件可能有自己的参数, 以 `Plugins.<Plugin name>.<Parameter>=<value>` 格式。常见的插件参数是 *Capacity*, 设置可以同时执行的检查的限制。
StatusPort	agent 2 将监听的端口，用于 HTTP 状态请求和显示已配置的插件列表以及一些内部参数
丢弃的参数	描述
AllowRoot, *User*	不支持，因为守护进程不支持。
LoadModule, *LoadModulePath*	不支持可加载模块。
StartAgents	此参数在 Zabbix agent 中用于增加被动检查并发性或禁用它们。在 Agent 2, 并发是在插件级别配置的，可以通过容量设置来限制。然而，当前不支持禁用被动检查。
HostInterface, *HostInterfaceItem*	不支持。

有关详细信息，请参阅配置文件选项zabbix_agent2.

退出码

从版本 4.4.8 开始，Zabbix agent 2 也可以用旧的 OpenSSL 版本 (1.0.1,1.0.2)。

在这种情况下，Zabbix 提供了在 OpenSSL 中锁定的互斥体。如果互斥锁锁定或解锁失败，则错误消息被打印到标准错误流 (STDERR)，Agent2 退出，返回代码分别为 2 或 3。

4 Proxy

概述

Zabbix proxy 是一个可以从一个或多个受监控设备采集监控数据并将信息发送到 Zabbix server 的进程，主要是代表 Zabbix server 工作。所有收集的数据都在本地缓存，然后传输到 proxy 所属的 Zabbix server。

部署 Zabbix proxy 是可选的，但可能非常有利于分担单个 Zabbix server 的负载。如果只有代理采集数据，则 Zabbix server 上会减少 CPU 和磁盘 I/O 的开销。

Zabbix proxy 是无需本地管理员即可集中监控远程位置、分支机构和网络的理想解决方案。

Zabbix proxy 需要使用独立的数据库。

> **Attention:**
> 值得注意的是，Zabbix proxy 支持 SQLite、MySQL 和 PostgreSQL 作为数据库。使用 Oracle 或 DB2 需要您承担一定的风险，例如，在自动发现规则中的遇到问题返回值 。

详见：在分布式环境中使用 Zabbix proxy。

运行 Proxy

如果通过包安装

Zabbix proxy 进程以守护进程（Deamon）运行。Zabbix proxy 的启动可以通过执行以下命令来完成：

```
shell> service zabbix-proxy start
```

上述命令在大多数的 GNU/Linux 系统下都可以正常完成。如果是其他系统，你可能要尝试以下命令来运行：

```
shell> /etc/init.d/zabbix-proxy start
```

类似的，Zabbix proxy 的停止、重启、查看状态，则需要执行以下命令：

```
shell> service zabbix-proxy stop
shell> service zabbix-proxy restart
shell> service zabbix-proxy status
```

手动启动

如果上述方法无效，则必须手动启动。找到路到 zabbix_proxy 二进制文件并执行：

```
shell> zabbix_proxy
```

您可以将以下命令行参数用于 Zabbix proxy:

```
-c --config <file>              配置文件的路径
-f --foreground                 运行Zabbix proxy在前台
-R --runtime-control <option>   启动管理员后台功能
-h --help                       帮助
-V --version                    显示版本信息
```

使用命令行参数运行 Zabbix proxy 的示例：

```
shell> zabbix_proxy -c /usr/local/etc/zabbix_proxy.conf
shell> zabbix_proxy --help
shell> zabbix_proxy -V
```

运行时控制

运行时控制选项:

选项	描述	目标
config_cache_reload	重新加载配置缓存。如果当前正在加载缓存则被忽略。 主动模式的 Zabbix proxy 连接到 Zabbix 服务器并请求配置数据。	
diaginfo[=<**target**>]	在代理日志文件中收集诊断信息。	**historycache** - 历史缓存状态 **preprocessing** - 预处理管理状态 **locks** - 互斥量列表 (在 **BSD* 系统中是空的)

选项	描述	目标
snmp_cache_reload	重新加载 SNMP 缓存，清除所有主机的 SNMP 属性 (引擎时间、引擎引导、引擎 id、凭据)。	
housekeeper_execute	启动管家处理过程。如果管家处理过程当前正在进行，则将被忽略。	
log_level_increase[=<**target**>]	增加日志级别，如果没有指定 target，将影响所有进程。 **BSD* 系统不支持。	**process type** - - 指定类型的所有进程 (e.g., poller) 另请参阅代理进程类型. **process type,N** - 指定进程类型和数量 (e.g., poller,3) **pid** - 进程标识符 (1 ~ 65535)。对于较大的值，将 target 指定为 “ ’process type,N”。
log_level_decrease[=<**target**>]	降低日志级别，如果没有指定 target，将影响所有进程。 **BSD* 系统不支持。	
prof_enable[=<**target**>]	启用分析。 如果未指定目标，则影响所有进程。 启用的分析按函数名称提供所有 rwlocks/mutex 的详细信息。 自 Zabbix 6.0.13 起支持。	进程类型 - 指定类型的所有进程（例如，history syncer）查看所有代理进程类型。 **’process type,N** - 进程类型和数量（例如，history syncer,1） **pid** - 进程标识符（1 到 65535）。对于较大的值，将目标指定为 “ ’process type,N”。 **scope** - `rwlock`、`mutex`、`processing` 可以与进程类型和数量（例如，history syncer ,1,processing）或所有类型的进程（例如，history syncer,rwlock）一起使用
prof_disable[=<**target**>]	禁用分析。 如果未指定目标，则影响所有进程。 自 Zabbix 6.0.13 起受支持。	进程类型 - 指定类型的所有进程（例如，history syncer）查看所有代理进程类型。 **process type,N** - 进程类型和数量（例如，history syncer,1） **pid** - 进程标识符（1 到 65535）。对于较大的值，将目标指定为 “process type,N”。

使用运行时控制重新加载代理配置缓存的示例：

```
shell> zabbix_proxy -c /usr/local/etc/zabbix_proxy.conf -R config_cache_reload
```

使用运行时控制收集诊断信息的示例:

```
在Proxy日志文件中收集所有可用的诊断信息：
shell> zabbix_proxy -R diaginfo

在Proxy日志文件中收集历史缓存统计信息：
shell> zabbix_proxy -R diaginfo=historycache
```

使用运行时控制重新加载 SNMP 缓存的例子:

```
shell> zabbix_proxy -R snmp_cache_reload
```

使用运行时控制触发管家执行的示例

```
shell> zabbix_proxy -c /usr/local/etc/zabbix_proxy.conf -R housekeeper_execute
```

使用运行时控制更改日志级别的示例:

```
增加所有进程的日志级别：
shell> zabbix_proxy -c /usr/local/etc/zabbix_proxy.conf -R log_level_increase

增加第二个轮询进程的日志级别：
shell> zabbix_proxy -c /usr/local/etc/zabbix_proxy.conf -R log_level_increase=poller,2
```

使用PID 1234增加进程的日志级别：
```
shell> zabbix_proxy -c /usr/local/etc/zabbix_proxy.conf -R log_level_increase=1234
```

降低所有http轮询器进程的日志级别：
```
shell> zabbix_proxy -c /usr/local/etc/zabbix_proxy.conf -R log_level_decrease="http poller"
```

进程用户

Zabbix proxy 被设计为以非 root 用户的身份运行。它将以任何非 root 用户的身份运行。因此，您可以作为任何非 root 用户运行代理而没有任何问题。

如果您尝试以 “root” 身份运行它，它将切换到硬编码的 “zabbix” 用户，这必须在您的系统上。如果你相应地修改了代理配置文件中的 AllowRoot 参数，那么你只能以 root 身份运行 proxy。

配置文件

另请参考configuration file 配置 zabbix_proxy 的详细信息。

Proxy 进程类型

- availability manager - 主机可用性更新的进程
- configuration syncer - 管理配置数据的内存缓存的进程
- data sender - proxy 输出发送进程
- discoverer - - 发现设备的进程程
- heartbeat sender - proxy 心跳信息发送进程
- history poller - 处理需要数据库连接的计算、聚合和内部检查的进程
- history syncer - 历史数据同步进程
- housekeeper - 删除旧历史数据的进程
- http poller - web 监控进程
- icmp pinger - icmpping 检查进程
- ipmi manager - IPMI 管理进程
- ipmi poller - IPMI 检查进程
- java poller - Java 检查进程
- odbc poller - ODBC 检查进程
- poller - 被动检查的普通轮询器
- preprocessing manager - 预处理任务的管理
- preprocessing worker - - 数据预处理的进程
- self-monitoring - 服务器内部统计信息收集进程
- snmp trapper - traper 用于 SNMP trap
- task manager - 远程执行其他组件请求的任务的过程 (例如，关闭问题、确认问题、现在检查项目值、远程命令功能)
- trapper - 陷阱主动检查，陷阱，代理通信
- unreachable poller - 针对不可达设备的轮询器
- vmware collector - VMware 数据收集器负责从 VMware 服务中收集数据

可以使用代理日志文件来观察这些进程类型。

可以使用 **zabbix[process,<type>,<mode>,<state>]** 内部监控项来监控各种类型的 Zabbix proxy 进程。

支持的平台

Zabbix proxy 和 Zabbix server 支持运行的平台 server#supported platforms 相同。

本地环境

请注意，代理需要 UTF-8 语言环境，以便能够正确解释某些文本项。大多数现代类 unix 系统默认使用 UTF-8 语言环境，然而，有些系统可能需要专门设置。

5 Java gateway

概述

从 Zabbix 2.0 开始，以 Zabbix 守护进程方式原生支持监控 JMX 应用程序就存在了，称之为 “Zabbix Java gateway”。Zabbix Java gateway 的守护进程是用 Java 编写。为了在特定主机上找到 JMX 计数器的值，Zabbix server 向 Zabbix Java gateway 发送请求，后者使用 JMX 管理 API 来远程查询相关的应用。该应用不需要安装额外的软件。只需要在启动时，命令行添加 -Dcom.sun.management.jmxremote 选项即可。

Java gateway 接受来自 Zabbix server 或 Zabbix proxy 的传入连接，并且只能用作 “被动 proxy”。与 Zabbix proxy 相反，它也可以从 Zabbix proxy （Zabbix proxy 不能被链接）调用。在 Zabbix server 或 Zabbix proxy 配置文件中，可以直接配置每个 Java gateway 的访问，因此每个 Zabbix server 或 Zabbix proxy 只能配置一个 Java gateway。如果主机将有 **JMX agent** 或其他类型的监控项，则只将 **JMX agent** 监控项传递给 Java gateway 进行检索。

当必须通过 Java gateway 更新监控项时，Zabbix server 或 proxy 将连接到 Java gateway 并请求该值，Java gateway 将检索该值并将其传递回 Zabbix server 或 Zabbix proxy。因此，Java gateway 不会缓存任何值。

Zabbix server 或 Zabbix proxy 具有连接到 Java gateway 的特定类型的进程，由 **StartJavaPollers** 选项控制。在内部，Java gateway 启动多个线程，由 **START_POLLERS** 选项控制。在服务器端，如果连接超过 **Timeout** 选项配置的秒数，它将被终止，但 Java gateway 可能仍在忙于从 JMX 计数器检索值。为了解决这个问题，从 Zabbix 2.0.15、Zabbix 2.2.10 和 Zabbix 2.4.5 开始，Java gateway 中有 **TIMEOUT** 选项，允许为 JMX 网络操作设置超时。

Zabbix server 或 proxy 尝试尽可能地将请求汇集到单个 JMX 目标（受监控项取值间隔影响），并在单个连接中将它们发送到 Java Gateway 以获得更好的性能。

此外，建议让 **StartJavaPollers** 选项的值小于或等于 **START_POLLERS**，否则可能会出现 Java gateway 中没有可用线程来为传入请求提供服务的情况。

以下部分描述了如何获取和运行 Zabbix Java gateway，如何配置 Zabbix server（或 Zabbix proxy）来使用 Zabbix Java gateway 进行 JMX 监控，以及如何在 Zabbix GUI 中配置与特定 JMX 计数器对应的 Zabbix 监控项。

When an item has to be updated over Java gateway, Zabbix server or proxy will connect to the Java gateway and request the value, which Java gateway in turn retrieves and passes back to the server or proxy. As such, Java gateway does not cache any values.

Zabbix server or proxy has a specific type of processes that connect to Java gateway, controlled by the option **StartJavaPollers**. Internally, Java gateway starts multiple threads, controlled by the **START_POLLERS** option. On the server side, if a connection takes more than **Timeout** seconds, it will be terminated, but Java gateway might still be busy retrieving value from the JMX counter. To solve this, there is the **TIMEOUT** option in Java gateway that allows to set timeout for JMX network operations.

Zabbix server or proxy will try to pool requests to a single JMX target together as much as possible (affected by item intervals) and send them to the Java gateway in a single connection for better performance.

It is suggested to have **StartJavaPollers** less than or equal to **START_POLLERS**, otherwise there might be situations when no threads are available in the Java gateway to service incoming requests; in such a case Java gateway uses ThreadPoolExecutor.CallerRunsPolicy, meaning that the main thread will service the incoming request and temporarilylabel will not accept any new requests.

If you are trying to monitor Wildfly-based Java applications with Zabbix Java gateway, please install the latest jboss-client.jar available on the Wildfly download page.

获取 java 网关

您可以从源代码或从下载的包中安装 Java 网关 Zabbix website.

使用下面的链接，您可以访问以下信息：如何获取和运行 Zabbix java 网关，如何配置 Zabbix server（或 Zabbix proxy）以使用 Zabbix Java 网关进行 JMX 监控，以及如何在 Zabbix 前端配置对应于特定 JMX 计数器的 Zabbix 监控项。

安装途径	安装说明	配置步骤说明
Sources	安装	配置
RHEL packages	安装	配置
Debian/Ubuntu packages	安装	配置

1 使用源码安装

概述

如果你是使用源码安装，那么下列信息会帮助你配置 Zabbix Java 网关。

文件概述

如果你从源码中获得 Java 网关，那么您最终会得到 $PREFIX/sbin/zabbix_java 下的 Shell 脚本、JAR 和配置文件的集合。这些文件的作用总结如下。

```
bin/zabbix-java-gateway-$VERSION.jar
```

Java 网关自身 JAR 文件。

```
lib/logback-core-0.9.27.jar
lib/logback-classic-0.9.27.jar
lib/slf4j-api-1.6.1.jar
```

`lib/android-json-4.3_r3.1.jar`

Java 网关依赖于: Logback, SLF4J, 和Android JSON 库。

```
lib/logback.xml
lib/logback-console.xml
```

Logback 的配置文件。

```
shutdown.sh
startup.sh
```

用于启动和停止 Java 网关的便捷脚本。

`settings.sh`

用于控制上述启动和停止脚本的配置文件。

配置和运行 Java 网关

Java 网关默认监听 10052 端口。如果你想运行 Java 网关在其他端口，你可以在 settings.sh 的脚本中指定其他端口号。详情可见Java 网关配置文件中描述的如何更改端口等其他信息。

> **Warning:**
> 10052 端口不是由 IANA 注册的.

当配置好，你可以使用启动脚本来运行 Java 网关:

```
$ ./startup.sh
```

同样，如果你不再使用 Java 网关了，可以运行关闭脚本将其停止:

```
$ ./shutdown.sh
```

请注意，Java 网关是轻量应用，不需要数据库，这一点与 Server 和 Proxy 不同。

配置 Zabbix Server 关联 Java 网关

当 Java 网关启动并运行后，你需要告诉 Zabbix server 去哪里找 Zabbix Java 网关。通过在server 配置文件中指定 JavaGateway 和 JavaGatewayPort 来完成这个操作。如果运行 JMX 应用程序的主机是由 Zabbix 代理监控的，则可以在proxy 配置文件中指定连接参数。

```
JavaGateway=192.168.3.14
JavaGatewayPort=10052
```

默认情况下，server 不会启动任何与 JMX 监控相关的进程。如果你希望用到它，则必须指定 Java pollers 的数量。此操作与配置常规 pollers 和 trappers 数量一样。

```
StartJavaPollers=5
```

配置完 server 或 proxy 后，一定不要忘记重启 server 或 proxy。

调试 Java 网关

为了防止在 Java gateway 出现任何问题或在 Zabbix 前端看不到详细的报错信息的情况下，你可以通过 Java gateway 日志文件来查看。

默认情况下，Java gateway 将其活动日志记录到日志级别为”info” 的 /tmp/zabbix_java.log 文件中。有时候，该日志信息可能不够详细，需要在日志级别为”debug” 中获取。为了提升日志级别，需要修改 lib/logback.xml 文件，并将 <root> 标记的日志等级属性更改为”debug” :

```
<root level="debug">
  <appender-ref ref="FILE" />
</root>
```

值得注意的是，与 Zabbix server 或 Zabbix proxy 不同，更改 logback.xml 文件并不需要重启 Zabbix Java gateway，它会自动提交。当完成调试后，可以将日志级别还原成 “info”。

如果希望将日志记录到其他文件或完全不同的介质，如数据库，那么只需要调整 logback.xml 文件。详见Logback 手册 获取更多信息。

有时为了调试，将 Java 网关用作控制台应用比以守护进程来启动更方便。因此可以在 settings.sh 中注释掉 PID_FILE 变量，这时运行 startup.sh 脚本启动 Java gateway 就会作为控制台应用来启动，并将 Logback 使用 lib/logback-console.xml 文件，这不仅会记录到控制台，还会启用日志级别 “debug”。

最后，请注意，由于 Java gateway 使用 SLF4J 来记录，您可以通过在 lib 目录放置合适的 JAR 文件来将 Logback 替换为您选中的框架。详见 SLF4J 手册 以获取更多信息。

JMX 监控

详见JMX 监控 页面以获取更多信息。

2 Setup from RHEL packages

Overview

If installed from RHEL packages, the following information will help you in setting up Zabbix Java gateway.

Configuring and running Java gateway

Configuration parameters of Zabbix Java gateway may be tuned in the file:

```
/etc/zabbix/zabbix_java_gateway.conf
```

For more details, see Zabbix Java gateway configuration parameters.

To start Zabbix Java gateway:

```
# service zabbix-java-gateway restart
```

To automatically start Zabbix Java gateway on boot:

RHEL 7 and later:

```
# systemctl enable zabbix-java-gateway
```

RHEL prior to 7:

```
# chkconfig --level 12345 zabbix-java-gateway on
```

Configuring server for use with Java gateway

With Java gateway up and running, you have to tell Zabbix server where to find Zabbix Java gateway. This is done by specifying JavaGateway and JavaGatewayPort parameters in the server configuration file. If the host on which JMX application is running is monitored by Zabbix proxy, then you specify the connection parameters in the proxy configuration file instead.

```
JavaGateway=192.168.3.14
JavaGatewayPort=10052
```

By default, server does not start any processes related to JMX monitoring. If you wish to use it, however, you have to specify the number of pre-forked instances of Java pollers. You do this in the same way you specify regular pollers and trappers.

```
StartJavaPollers=5
```

Do not forget to restart server or proxy, once you are done with configuring them.

Debugging Java gateway

Zabbix Java gateway log file is:

```
/var/log/zabbix/zabbix_java_gateway.log
```

If you like to increase the logging, edit the file:

```
/etc/zabbix/zabbix_java_gateway_logback.xml
```

and change `level="info"` to ”debug” or even ”trace” (for deep troubleshooting):

```
<configuration scan="true" scanPeriod="15 seconds">
[...]
      <root level="info">
              <appender-ref ref="FILE" />
      </root>

</configuration>
```

JMX monitoring

See JMX monitoring page for more details.

3 从 **Debian/Ubuntu** 包安装

概述

如果你是通过 Debian/Ubuntu 的包安装，那么下列信息会帮助你配置 Zabbix Java 网关。

配置和运行 Java 网关

Java 网关的配置参数可以通过如下文件进行调整：

```
/etc/zabbix/zabbix_java_gateway.conf
```

关于更多信息，详见 Zabbix Java geteway 配置参数.

通过以下命令启动 Zabbix Java 网关：

```
# service zabbix-java-gateway restart
```

通过以下命令配置 Zabbix Java 网关开机自启动：

```
# systemctl enable zabbix-java-gateway
```

配置 Zabbix Server 关联 Java 网关

当 Java 网关启动并运行后，你需要告诉 Zabbix server 去哪里找 Zabbix Java 网关。通过在server 配置文件中指定 JavaGateway 和 JavaGatewayPort 来完成这个操作。如果运行 JMX 应用程序的主机是由 Zabbix 代理监控的，则可以在proxy 配置文件中指定连接参数。

```
JavaGateway=192.168.3.14
JavaGatewayPort=10052
```

默认情况下，server 不会启动任何与 JMX 监控相关的进程。如果你希望用到它，则必须指定 Java pollers 的数量。此操作与配置常规 pollers 和 trappers 数量一样。

```
StartJavaPollers=5
```

配置完 server 或 proxy 后，一定不要忘记重启 server 或 proxy。

调试 Java 网关

Zabbix Java 网关的日志路径：

```
/var/log/zabbix/zabbix_java_gateway.log
```

如果要增加日志记录，编辑以下文件：

```
/etc/zabbix/zabbix_java_gateway_logback.xml
```

并将 `level="info"` 更改为”debug” 或”trace”（深度排错模式）

```
<configuration scan="true" scanPeriod="15 seconds">
[...]
      <root level="info">
              <appender-ref ref="FILE" />
      </root>

</configuration>
```

JMX 监控

详见JMX 监控 页面以获取更多信息。

6 Sender

概述

Zabbix sender 是一个用来推送性能数据给 Zabbix Server 处理的命令行应用程序。

该应用程序通常用在定期推送可用性和性能数据等在长耗时的用户脚本上。

要将监控数据直接保存在 Zabbix Server 或 Zabbix Proxy 上，必须将监控项类型配置为trapper 。

运行 Zabbix sender

一个运行 Zabbix UNIX sender 的例子：

```
shell> cd bin
shell> ./zabbix_sender -z zabbix -s "Linux DB3" -k db.connections -o 43
```

其中：

- z - Zabbix server 域名或者 IP 地址
- s - 被监控的主机（与 Zabbix 前端主机名对应）

- k - 监控项键
- o - 发送的值

> **Attention:**
> 包含空格的选项必须使用双引号引用。

Zabbix sender 可通过从输入文件发送多个值。详见Zabbix sender manpage

如果指定了配置文件，Zabbix sender 将使用 agent ServerActive 配置参数中定义的所有地址发送数据。如果发送到一个地址失败，发件人将尝试发送到其他地址。如果将批数据发送到一个地址失败，则以下批不会发送到此地址。

Zabbix sender 接受 UTF-8 编码的字符串（对于类 UNIX 系统和 Windows ），且在文件中没有字节顺序标记（BOM）。

Zabbix sender 同样可以在 Windows 上运行：

```
zabbix_sender.exe [options]
```

从 Zabbix 1.8.4 开始，zabbix_sender 实时发送方案已得到改进，可以连续接收多个传递给它的值，并通过单个连接将它们发送到 Zabbix Server。两个不超过 0.2 秒的值可以放在同一堆栈中，但最大 pooling 时间仍然是 1 秒。

> **Note:**
> Zabbix sender 如果指定的配置文件中存在无效（不遵循 *parameter=value* 注释）的参数条目，则 Zabbix sender 将终止。

Running Zabbix sender with low-level discovery

An example of running Zabbix sender for sending a JSON-formatted value for low-level discovery:

```
./zabbix_sender -z 192.168.1.113 -s "Zabbix server" -k trapper.discovery.item -o '[{"{#ID}":"/","{#THRESHO
```

For this to work, the low-level discovery rule must have a Zabbix trapper item type (in this example, with `trapper.discovery.item` key).

7 Get

概览

Zabbix get 是一个命令行实用程序，可用于与 Zabbix agent 通信并从 agent 检索所需信息。

该实用程序通常用于 Zabbix agent 的故障排除。

运行 Zabbix get

UNIX 下运行 Zabbix get 从 agent 获取处理器负载值的例子：

·shell> cd bin ·shell> ./zabbix_get -s 127.0.0.1 -p 10050 -k system.cpu.load[all,avg1]

运行 Zabbix get 以从网站捕获字符串的另一个示例：

·shell> cd bin ·shell> ./zabbix_get -s 192.168.1.1 -p 10050 -k ”web.page.regexp[www.example.com,,,”USA: ([a-zA-Z0-9.-]+)”,,\1]”

请注意，此处的监控项键包含一个空格，因此引号用于将监控项键标记到 shell。引号不是监控项键的一部分；它们将被 shell 修剪，不会传递给 Zabbix agent。

Zabbix get 接受以下命令行参数：

·-s --host <host name or IP> ·指定主机的主机名或 IP 地址。·-p --port <port number> ·指定在主机上运行的 agent 程序的端口号。默认值为 10050。·-I --source-address <IP address> 指定源 IP 地址。·-t --timeout <seconds> ·指定超时。有效范围：1-30 秒（默认值：30 秒）。·-k --key <item key> ·指定监控项的键以检索其值。·-h --help ·获取帮助信息。·-V --version ·显示版本号。

另请参阅Zabbix get 帮助页 了解更多信息。

Zabbix get on Windows 可以类似运行：

·zabbix_get.exe options

8 JS

概述

zabbix_js 是一个命令行实用程序，可用于嵌入脚本测试。

该程序可执行带有字符串参数的用户自定义脚本并打印结果。脚本的执行是由内嵌的 Zabbix 脚本引擎来完成的。

在编译或执行错误的情况下，zabbix_js 将在 stderr 中打印错误并以代码 1 退出。

用法

```
zabbix_js -s script-file -p input-param [-l log-level] [-t timeout]
zabbix_js -s script-file -i input-file [-l log-level] [-t timeout]
zabbix_js -h
zabbix_js -V
```

zabbix_js 可接收如下命令行参数：

```
  -s, --script script-file          指定待执行脚本的文件名。若 '-' 作为文件名时，脚本名由stdin输入。
  -i, --input input-file            指定输入参数的文件名。若 '-' 作为文件名时，脚本名由stdin输入。
  -p, --param input-param           指定输入参数。
  -l, --loglevel log-level          指定日志级别。
  -t, --timeout timeout             指定超时时间（单位：秒）。
  -h, --help                        显示帮助信息。
  -V, --version                     显示版本号。
```

例如：

```
zabbix_js -s script-file.js -p example
```

9 Web service

概述

Zabbix web service 是一个用来连接外部网站服务的进程。现在，Zabbix web service 用来收集和发送定时报告，并且计划未来添加更多功能。

Zabbix server 通过 HTTP(s) 连接到 web 服务。Zabbix web service 要求谷歌浏览器安装在同一台主机上；在某些发行版上，该服务也可以与 Chromium 一起使用（可查看known issues）。

安装

Zabbix web service 模块在预编译的 Zabbix 安装包中提供，可从Zabbix 网站下载。通过源码包中编译Zabbix web service时，需要指定`--enable-webservice` 配置参数。

另见：

- 配置文件选项zabbix_web_service;
- 配置定时报表

4. 安装

请使用侧边栏访问安装部分的内容。

1 获取 Zabbix

概述

获取 Zabbix 安装介质有四种方法：

- 从发行包 安装；
- 下载最新的归档源码包并编译它；
- 从容器 中安装；
- 下载Zabbix 应用。

请转到 Zabbix 下载页面 下载最新的源码包或应用，此页面提供最新版本的直接链接。如果要下载旧版本，请参阅以下稳定版本下载链接。

获取 Zabbix 源码

如何获取 Zabbix 源码有如下几种方式：

- 可从 Zabbix 官方网站获取发布的稳定版本 下载

- 可从 Zabbix 官方开发网站页面获取源架构 下载
- 可以从 Git 源代码中获取最新的开发版本存储系统：
  - 完整存储 Zabbix 源码的主要位置是 https://git.zabbix.com/scm/zbx/zabbix.git
  - 主版本和支持版本也镜像到 Github，网址为 https://github.com/zabbix/zabbix

必须安装 Git 客户端才能克隆存储源。官方命令行 Git 客户端包在发布版本中通常称为 **git**。示例：在 Debian/Ubuntu 上安装，请运行如下命令：

```
sudo apt-get update
sudo apt-get install git
```

若需获取所有 Zabbix 源代码，请更改相应文件来替换源码并执行如下命令：

```
git clone https://git.zabbix.com/scm/zbx/zabbix.git
```

2 安装要求

硬件

内存和磁盘

Zabbix 运行需要物理内存和磁盘空间。如果刚接触 Zabbix，128 MB 的物理内存和 256 MB 的可用磁盘空间可能是一个很好的起点。然而，所需的内存和磁盘空间显然取决于被监控的主机数量和配置参数。如果您计划调整参数以保留较长的历史数据，那么您应该考虑至少有几 GB 磁盘空间，以便有足够的磁盘空间将历史数据存储在数据库中。

每个 Zabbix 守护程序进程都需要与数据库服务器建立多个连接。为连接分配的内存量取决于数据库引擎的配置。

> **Note:**
> 您拥有的物理内存越多，数据库（以及 Zabbix ）的工作速度就越快！

CPU

Zabbix，尤其是 Zabbix 数据库可能需要大量 CPU 资源，该具体取决于被监控参数的数量和所选的数据库引擎。

其它硬件

如果需要启用短信（SMS）通知功能，需要串行通讯口（serial communication port）和串行 GSM 调制解调器（serial GSM modem）。USB 转串行转接器也同样可以工作。

硬件配置示例

该表提供了硬件配置示例，假设是 **Linux/BSD/Unix** 平台。

这些是开始时的大小和硬件配置示例。每个 Zabbix 安装都是独一无二的。确保在暂存或开发环境中对 Zabbix 系统的性能进行基准测试，以便在将 Zabbix 安装部署到其生产环境之前充分了解您的要求。

安装大小	监控指标[1]	CPU/vCPU 内核	内存 (GiB)	数据库	Amazon EC2[2]
小	1 000	2	8	MySQL 服务器，Percona 服务器，MariaDB 服务器，PostgreSQL	m6i.large/m6g.large
中	10 000	4	16	MySQL 服务器，Percona 服务器，MariaDB 服务器，PostgreSQL	m6i.xlarge/m6g.xlarge
大	100 000	16	64	MySQL 服务器，Percona 服务器，MariaDB 服务器，PostgreSQL，Oracle	m6i.4xlarge/m6g.4xlarge

安装大小	监控指标[1]	CPU/vCPU 内核	内存 (GiB)	数据库	Amazon EC2[2]
非常大	1 000 000	32	96	MySQL 服务器，Percona 服务器，MariaDB 服务器，PostgreSQL，Oracle	m6i.8xlarge/m6g.8xlarge

[1] 1 个指标 = 1 个项目 + 1 个触发器 + 1 个图表
 [2] Amazon 通用 EC2 实例示例，使用 ARM64 或 x86_64 架构，在 Zabbix 安装评估和测试期间应选择适当的实例类型，如计算/内存/存储优化，然后再安装到其生产环境中。

Note:
实际配置很大程度上取决于活动监控项的数量和刷新率（请参阅本节的数据库大小 部分详情页）。对于大型安装，强烈建议在单独的机器上运行数据库。

受支持的平台

由于服务器操作的安全性要求和任务关键性，UNIX 是唯一能够始终如一地提供必要性能、容错和弹性的操作系统。Zabbix 以市场主流的操作系统版本运行。

经测试，Zabbix 组件可以运行在下列平台：

平台	Server	Agent	Agent2
Linux	x	x	x
IBM AIX	x	x	-
FreeBSD	x	x	-
NetBSD	x	x	-
OpenBSD	x	x	-
HP-UX	x	x	-
Mac OS X	x	x	-
Solaris	x	x	-
Windows	-	x	x

Note:
Zabbix server/agent 也可以在其他类 Unix 操作系统上运行。自 XP 以来，所有 Windows desktop 和 server 版本都支持 Zabbix agent。

Attention:
如果使用加密编译，Zabbix 将禁用核心转储（Core dumps），如果系统不允许禁用核心转储，则 Zabbix 不会启动。

软件

Zabbix 是围绕现代 Web 服务器，领先的数据库引擎和 PHP 脚本语言构建的。

第三方外部软件支持

如果标记为必需，则所需的软件/库是绝对必要的。可选的软件/库用于支持某些特定功能。

软件	必需状态	支持的版本	注释
MySQL/Percona	其中之一	8.0.30-8.4.X	如果使用 MySQL（或 Percona）作为 Zabbix 后端数据库，则需要此软件。需要 InnoDB 引擎。 添加了对 MySQL 版本的支持： - 从 Zabbix 6.0.21 开始支持 8.1； - 从 Zabbix 6.0.26 开始支持 8.2； - 从 Zabbix 6.0.27 开始支持 8.3； - 从 Zabbix 6.0.32 开始支持 8.4。 我们建议使用 C API (lib-mysqlclient) 库来构建服务器/代理。 对于 MySQL 版本 8.0.0-8.0.28，另请参阅：可能的死锁。

软件	必需状态	支持的版本	注释
MariaDB		10.5.00-11.4.X	需要 InnoDB 引擎。 推荐版本为 10.5。 我们建议使用 MariaDB Connector/C 库来构建服务器/代理。 添加了对 MariaDB 版本的支持： - 从 Zabbix 6.0.6 开始支持 10.7.X； - 从 Zabbix 6.0.7 开始支持 10.8.X； - 从 Zabbix 6.0.13 开始支持 10.9.X 和 10.10.X； - 从 Zabbix 6.0.15 开始支持 10.11.X； - 从 Zabbix 6.0.21 开始支持 11.0.X； - 从 Zabbix 6.0.23 开始支持 11.1.X； - 从 Zabbix 6.0.27 开始支持 11.2.X； - 从 Zabbix 6.0.29 开始支持 11.3.X； - 从 Zabbix 6.0.32 开始支持 11.4.X。 另请参阅：可能的死锁。
Oracle		19c - 21c	如果使用 Oracle 作为 Zabbix 后端数据库，则需要此软件。

软件	必需状态	支持的版本	注释
PostgreSQL		13.0-16.X	如果使用 PostgreSQL 作为 Zabbix 后端数据库，则需要此软件。根据安装大小，可能需要增加 PostgreSQL *work_mem* 配置属性（默认值为 4MB），以便数据库用于特定操作的内存足够，查询执行不会花费太长时间。 添加了对 PostgreSQL 版本的支持： - 从 Zabbix 6.0.10 开始支持 15.X； - 从 Zabbix 6.0.23 开始支持 16.X。

软件	必需状态	支持的版本	注释
TimescaleDB for PostgreSQL		2.1.0-2.15.X	如果使用 TimescaleDB 作为 PostgreSQL 数据库扩展，则需要此软件。 请确保安装支持压缩的 TimescaleDB 社区版。 注意，从 TimescaleDB 2.10 开始支持 PostgreSQL 15。 您也可以参考官方文档 了解 PostgreSQL 和 TimescaleDB 版本兼容性的详细信息。 添加了对 TimescaleDB 版本的支持： - 从 Zabbix 6.0.7 开始支持 2.6； - 从 Zabbix 6.0.8 开始支持 2.7； - 从 Zabbix 6.0.10 开始支持 2.8； - 从 Zabbix 6.0.13 开始支持 2.9； - 从 Zabbix 6.0.15 开始支持 2.10； - 从 Zabbix 6.0.19 开始支持 2.11； - 从 Zabbix 6.0.24 开始支持 2.12； - 从 Zabbix 6.0.25 开始支持 2.13； - 从 Zabbix 6.0.28 开始支持 2.14； - 从 Zabbix 6.0.32 开始支持 2.15。

软件	必需状态	支持的版本	注释
SQLite	可选	3.3.5-3.34.X	SQLite 仅支持 Zabbix 代理。如果使用 SQLite 作为 Zabbix 代理数据库，则需要此软件。
smartmontools		7.1 或更高版本	需要 Zabbix 代理 2。
who			需要用户计数插件。
dpkg			需要 system.sw.packages 插件。
pkgtool			需要 system.sw.packages 插件。
rpm			需要 system.sw.packages 插件。
pacman			需要 system.sw.packages 插件。

Note:
尽管 Zabbix 可以与操作系统中可用的数据库一起使用，为了获得最佳体验，我们建议使用从官方数据库开发者仓库安装的数据库。

前端

Zabbix 前端支持的最小屏幕宽度为 1200px。

软件	版本	备注
Apache	1.3.12 或更高版本	
PHP	7.2.5 或更高版本	不支持 PHP 8.0。
PHP 扩展：		
gd	2.0.28 或更高版本	PHP GD 扩展必须支持 PNG (*--with-png-dir*)、JPEG (*--with-jpeg-dir*) 和 FreeType 2 (*--with-freetype-dir*)。
bcmath		php-bcmath (*--enable-bcmath*)
ctype		php-ctype (*--enable-ctype*)
libXML	2.6.15 或更高版本	php-xml，如果由分发者作为单独的包提供。
xmlreader		php-xmlreader，如果由分发者作为单独的包提供。
xmlwriter		php-xmlwriter，如果由分发者作为单独的包提供。
session		php-session，如果由分发者作为单独的包提供。
sockets		php-net-socket (*--enable-sockets*)。需要用户脚本支持。
mbstring		php-mbstring (*--enable-mbstring*)
gettext		php-gettext (*--with-gettext*)。Required for translations to work.
ldap		php-ldap. 仅当在前端使用 LDAP 身份验证时才需要。
openssl		php-openssl. 仅当在前端使用 SAML 身份验证时才需要。
mysqli		如果 MySQL 用作 Zabbix 后端数据库，则需要。

软件	版本	备注
oci8		如果使用 Oracle 作为 Zabbix 后端数据库，则需要。
pgsql		如果使用 PostgreSQL 作为 Zabbix 后端数据库，则需要。

Note:
Zabbix 也许可以在以前的 Apache、MySQL、Oracle 和 PostgreSQL 版本上运行。

Attention:
如果需要使用默认 DejaVu 以外的字体，可能会需要 PHP 的 imagerotate 函数。如果缺少，在 Zabbix 前端查看图形时可能会显示异常。该函数只有在使用捆绑的 GD 库编译 PHP 时才可用。在 Debian 和某些发行版本中，这个问题不存在。

客户端浏览器

浏览器必须启用 Cookies 和 Java Script 。

支持 Google Chrome，Mozilla Firefox，Microsoft Edge，Apple Safari 和 Opera 的最新稳定版本。

Warning:
为了执行 IFrame 的“同源政策”，意味着 Zabbix 不能放在不同域的 frames 中。

但是，如果放置在 frames 中的页面和 Zabbix 前端位于同一个域中，则置于 Zabbix frames 中的页面将可以访问 Zabbix 前端（通过 JavaScript）。像 `http://secure-zabbix.com/cms/page.html` 这样的页面，如果置于 `http://secure-zabbix.com/zabbix/` 的聚合图形或仪表盘上，将拥有对 Zabbix 的完整 JS 访问权限。

Server

强制性要求始终需要。可选要求在支持特定功能时需要。

需求	状态	描述
libpcre	强制	PCRE 库被 Perl 兼容正则表达式 (PCRE) 支持所需要。 命名可能因 GNU/Linux 发行版而异，例如'libpcre3' 或'libpcre1'。(Zabbix 6.0.0) 支持 PCRE v8.x 及 PCRE2 v10.x。
libevent		大量请求指标和 IPMI 监控需要。1.4 版本及以上。Zabbix proxy 该项可选；IPMI 监控支持必须。
libpthread		被互斥锁（mutex）和读写分离锁（read-write lock）支持所需要。
zlib		被压缩支持所需要。
OpenIPMI	可选	被 IPMI 支持所需要。
libssh2		被 SSH 支持所需要。版本 1.0 以上。
fping		被ICMP ping 监控项 所需要。
libcurl		被 web 监控，VMware 监控和 SMTP 认证所需要。如果是为了 SMTP 认证，需要 7.20.0 以上的版本，同时需要 Elasticsearch。
libiksemel		被 Jabber 支持所需要。
libxml2		被 VMware 监控所需要。
net-snmp		被 SNMP 支持所需要。

Agent

需求	状态	描述
libpcre	强制	PCRE 库被 Perl 兼容正则表达式 (PCRE) 支持所需要。 命名可能因 GNU/Linux 发行版而异，例如'libpcre3' 或'libpcre1'。(Zabbix 6.0.0) 支持 PCRE v8.x 及 PCRE2 v10.x。

需求	状态	描述
GnuTLS, *OpenSSL* or *LibreSSL*	可选	当使用加密 时需要。 在 Microsoft Windows 系统上需要 OpenSSL 1.1.1 及以上版本。

Note:
从 5.0.3 版本开始，Zabbix agent 将不再支持低于 6.1 TL07 / AIX 7.1 TL01 版本的 AIX 平台。

Agent 2

需求	状态	描述
libpcre	强制	PCRE 库被 Perl 兼容正则表达式 (PCRE) 支持所需要。 命名可能因 GNU/Linux 发行版而异，例如'libpcre3' 或'libpcre1'。(Zabbix 6.0.0) 支持 PCRE v8.x 及 PCRE2 v10.x。
OpenSSL	可选	当使用加密时需要。 UNIX 平台上需要 OpenSSL 1.0.1 或更高版本。 OpenSSL 库必须启用 PSK 支持。不支持 LibreSSL。 在 Microsoft Windows 系统上需要 OpenSSL 1.1.1 及以上版本。

Java 网关

如果您从源代码仓库或存档中获得了 Zabbix，则必要的依赖项已包含在源代码树中。

如果您从发行版的软件包中获得了 Zabbix，那么打包系统已经提供了必要的依赖项。

在上述两种情况下，即可准备部署软件了，而不需要下载额外的依赖包。

但是，如果您希望提供这些依赖关系的版本（例如，如果您正在为某些 Linux 发行版准备软件包），则下面是 Java 网关已知可以使用的库的版本列表。Zabbix 也许可以与这些库的其他版本一起使用。

下表列出了原始代码中当前与 Java 网关捆绑在一起的 JAR 文件：

库名	许可	网站	备注
logback-core-1.2.3.jar	EPL 1.0, LGPL 2.1	http://logback.qos.ch/	0.9.27, 1.0.13, 1.1.1 和 1.2.3 测试通过。
logback-classic-1.2.3.jar	EPL 1.0, LGPL 2.1	http://logback.qos.ch/	0.9.27, 1.0.13, 1.1.1 和 1.2.3 测试通过。
slf4j-api-1.7.30.jar	MIT License	http://www.slf4j.org/	1.6.1, 1.6.6, 1.7.6 和 1.7.30 测试通过。
android-json-4.3_r3.1.jar	Apache License 2.0	https://android.googlesource.com/platform/libcore/+/master/json	2.3.3_r1.1 和 4.3_r3.1 测试通过。关于创建 JAR 文件，详见 src/zabbix_java/lib/README 说明。

Java 网关可以使用 Oracle Java 或开源 OpenJDK（版本 1.6 或更高版本）构建。Zabbix 提供的软件包是使用 OpenJDK 编译的。下表提供了有关用于按发行版构建 Zabbix 软件包的 OpenJDK 版本的信息：

发行版	OpenJDK 版本
RHEL/CentOS 8	1.8.0
RHEL/CentOS 7	1.8.0
SLES 15	11.0.4
SLES 12	1.8.0
Debian 10	11.0.8
Ubuntu 20.04	11.0.8
Ubuntu 18.04	11.0.8

默认端口号

以下每个组件的开放端口列表适用于默认配置：

Zabbix 组件	端口号	协议	连接类型
Zabbix agent	10050	TCP	on demand
Zabbix agent 2	10050	TCP	on demand
Zabbix server	10051	TCP	on demand
Zabbix proxy	10051	TCP	on demand
Zabbix Java gateway	10052	TCP	on demand
Zabbix web service	10053	TCP	on demand
Zabbix frontend	80	HTTP	on demand
	443	HTTPS	on demand
Zabbix trapper	10051	TCP	on demand

Note:
端口号应在防火墙中打开以启用 Zabbix 通信。向外访问的 TCP 连接通常不需要明确的防火墙设置。

数据库大小

Zabbix 配置文件数据需要固定数量的磁盘空间，且增长不大。

Zabbix 数据库大小主要取决于这些变量，这些变量决定了存储的历史数据量：

- 每秒处理值的数量

这是 Zabbix server 每秒接收的新值的平均数。例如，如果有 3000 个监控项用于监控，取值间隔为 60 秒，则这个值的数量计算为 3000/60 = ** 50 **。

这意味着每秒有 50 个新值被添加到 Zabbix 数据库中。

- Housekeeper 的历史记录设置

Zabbix 将接收到的值保存一段固定的时间，通常为几周或几个月。每个新值都需要一定量的磁盘空间用于数据和索引。

所以，如果我们每秒收到 50 个值，且希望保留 30 天的历史数据，值的总数将大约在 (**30***24*3600)* **50** = 129.600.000，即大约 130M 个值。

根据所使用的数据库引擎，接收值的类型（浮点数、整数、字符串、日志文件等），单个值的磁盘空间可能在 40 字节到数百字节之间变化。通常，数值类型的每个值大约为 90 个字节[2]。在上面的例子中，这意味着 130M 个值需要占用 130M * 90 bytes = **10.9GB** 磁盘空间。

Note:
文本/日志类型的监控项值的大小是无法确定的，但可以以每个值大约 500 字节来计算。

- Housekeeper 的趋势记录设置

Zabbix 为表 **trends** 中的每个项目保留 1 小时的最大值 / 最小值 / 平均值 / 统计值。该数据用于趋势图形和历史数据图形。这一个小时的时间段是无法自定义。

Zabbix 数据库，根据数据库类型，每个值总共需要大约 90 个字节。假设我们希望将趋势数据保持 5 年。3000 个监控项的值每年需要占用 3000*24*365* **90** = **2.2GB**，或者 5 年需要占用 **11GB** 。

- Housekeeper 的事件记录设置

每个 Zabbix 事件需要大约 250 个字节的磁盘空间[1]。很难估计 Zabbix 每天生成的事件数量。在最坏的情况下，假设 Zabbix 每秒生成一个事件。

对于每个恢复的事件，将创建一个 event_recovery 记录。通常，大多数事件将被恢复，因此我们可以假设每个事件有一个 event_recovery 记录。这意味着每个事件额外 80 个字节。

(可选）事件可以具有标记，每个标记记录需要大约 100 字节的磁盘空间[1]。每个事件的标签数 (#tags) 取决于配置。因此，每个事件都需要额外的标签数 #tags * 100 字节的磁盘空间。

这意味着如果想要保留 3 年的事件，这将需要 3*365*24*3600* (250+80+ 标签数 #tags*100) = **~30GB**+ 标签数 #tags*100B 的磁盘空间[2]。

Note:
[1] 当具有非 ASCII 的事件名称、标记和值时，需要的空间会更多。
[2] 大小近似值基于 MySQL，对于其他数据库可能有所不同。

下表包含可用于计算 Zabbix 系统所需磁盘空间的公式：

类型	所需磁盘空间的公式 (字节)
Zabbix 配置	固定大小。通常为 10MB 或更少。
历史数据	days*(items/refresh rate)*24*3600*bytes items：监控项数量 days：保留历史记录的天数 refresh rate：监控项的平均刷新率 bytes：保留单个值所需要占用的字节数，依赖于数据库引擎，通常为 ~90 字节。
趋势数据	days*(items/3600)*24*3600*bytes items：监控项数量 days：保留历史记录的天数 bytes：保留单个趋势数据所需要占用的字节数，依赖于数据库引擎，通常为 ~90 字节。
事件数据	days*events*24*3600*bytes events：每秒产生的事件数量。假设最糟糕的情况下，每秒产生 1 个事件。 days：保留历史数据的天数。 bytes：保留单个趋势数据所需的字节数，取决于数据库引擎，通常为 ~330 + 每个事件的平均标签数 * 100 字节。

因此，所需要的磁盘总空间按下列方法计算：
Zabbix 配置 **+** 历史数据 **+** 趋势数据 **+** 事件数据
在安装 Zabbix 后不会立即使用磁盘空间。数据库大小取决于 housekeeper 设置，在某些时间点增长或停止增长。

时间同步

服务器上拥有精确的系统时间对 Zabbix 的运行非常重要。ntpd 是最流行的守护程序，它将主机的时间与其他计算机的时间同步。强烈建议在运行 Zabbix 组件的所有系统上保持系统时间同步。

安全设置 **Zabbix** 的最佳实践

概述

本章节内容提供了对于 Zabbix 来说，可采用更加安全的安装方式。

实现 Zabbix 的功能并不局限于该范例中所涉及的内容。但建议用户可通过参考该配置方式为 Zabbix 提供更好的系统安全性。

访问控制

最小权限原则

Zabbix 应该始终遵守最小权限原则。该原则意味着用户的账号（在 Zabbix 前端）或流程用户（对于 Zabbix server/proxy 或 agent）仅具有执行预期功能所必需的权限。换句话说，用户账号应始终以尽可能小的权限运行。

> **Attention:**
> 赋予 “zabbix” 用户提供额外权限将允许其访问配置文件，并执行可能损害基础架构整体安全性的操作。

最小权限原则，应考虑 Zabbix 前端用户类型。重要的是要理解，虽然 “管理员” 用户类型的权限低于 “超级管理员” 用户类型，但它具有允许管理配置和执行自定义脚本的管理权限。

> **Note:**
> 某些信息甚至适用于非特权用户。例如，虽然 *Administration → Scripts* 不适用于非超级管理员，但脚本本身可通过使用 Zabbix API 来进行检索。应使用限制脚本权限和不添加敏感信息（如访问凭据等）来避免暴露全局脚本中可用的敏感信息。

Zabbix agent 的安全用户

在默认配置中，Zabbix server 和 Zabbix agent 进程共享一个 “zabbix” 用户。若希望确保 Zabbix agent 无法访问 Zabbix server 配置中的敏感信息（例：数据库登录信息），则应以不同的用户身份运行 Zabbix agent：

1. 创建一个安全用户；“
2. 在 Zabbix agent 的configuration file 中指定此用户（修改’User’ 参数）；
3. 以管理员权限的用户重启 Zabbix agent。此后权限将赋予给先前指定的用户。

带有 OpenSSL 的在 Windows 上的 zabbix agent

使用 OpenSSL 编译的 Zabbix Windows agent 将尝试访问 c:\openssl-64bit 中访问 SSL 配置文件。磁盘 C 上的 “ openssl-64bit” 目录可以由非特权用户创建。

因此，为了加强安全性，需手动创建此目录并撤销非管理员用户的写访问权限。

请注意，在 Windows 的 32 位和 64 位版本上，目录名称将有所不同。

密码学

为 Zabbix 前端设置 SSL

在 RHEL/Centos 操作系统上，安装 mod_ssl 包：

```
yum install mod_ssl
```

为 SSL keys 创建目录：

```
mkdir -p /etc/httpd/ssl/private
chmod 700 /etc/httpd/ssl/private
```

创建 SSL 证书：

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/httpd/ssl/private/apache-selfsigned.key -
```

请用户根据需求适当填写下列提示内容。最重要的参数是请求 Common Name 参数。您需要输入要与服务器关联的域名。如果您没有域名，则可以输入公共 IP 地址。下面将使用 *example.com*。

```
Country Name (两个字母) [XX]:
State or Province Name (全名) []:
Locality Name (eg, city) [默认的城市]:
Organization Name (eg, company) [默认的公司名]:
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:example.com
Email Address []:
```

编辑 Apache SSL 配置：

```
/etc/httpd/conf.d/ssl.conf

DocumentRoot "/usr/share/zabbix"
ServerName example.com:443
SSLCertificateFile /etc/httpd/ssl/apache-selfsigned.crt
SSLCertificateKeyFile /etc/httpd/ssl/private/apache-selfsigned.key
```

重启 Apache 服务使以上修改的配置生效：

```
systemctl restart httpd.service
```

Web 服务器加固

在 URL 的根目录上启用 Zabbix

将虚拟主机添加到 Apache 配置，并将文档根目录的永久重定向设置为 Zabbix SSL URL。注意将 *example.com* 替换为服务器的实际名称。

```
/etc/httpd/conf/httpd.conf

#Add lines

<VirtualHost *:*>
    ServerName example.com
    Redirect permanent / http://example.com
</VirtualHost>
```

重启 Apache 服务使以上修改的配置生效：

```
systemctl restart httpd.service
```

在 Web 服务器上启用强制安全传输技术

为了保护 Zabbix 前端不受协议降级攻击，我们建议在 Web 服务器上启用HSTS策略

例如，要在 Apache 配置中为您的 Zabbix 前端启用 HSTS 策略：

/etc/httpd/conf/httpd.conf

将以下指令添加到虚拟主机的配置中：

```
<VirtualHost *:443>
   Header set Strict-Transport-Security "max-age=31536000"
</VirtualHost>
```

重新启动 Apache 服务以应用更改：

```
systemctl restart httpd.service
```

Enforcing Secure and SameSite Session Cookies in Zabbix

When configuring Zabbix, it is essential to enforce secure and SameSite attributes for session cookies to enhance security and prevent cross-site request forgery (CSRF) attacks. However, enforcing `SameSite=Strict` may cause issues in certain scenarios, such as:

- Dashboard URL widgets displaying "user not logged in" when embedding same-domain iframes.
- Users accessing the dashboard via HTTP instead of HTTPS may face login issues.
- Inability to share URLs to specific Zabbix menu sections or hosts.

To mitigate these issues, users should have a way to adjust the SameSite policy.

1. Secure Cookies

Setting the `secure` flag ensures that cookies are only transmitted over HTTPS, preventing exposure over unencrypted connections.

To enable secure cookies in Zabbix, add or modify the following setting in the web server configuration:

For Apache:

```
Header always edit Set-Cookie ^(.*)$ $1;Secure
```

For Nginx:

```
proxy_cookie_path / "/; Secure";
```

Ensure that your Zabbix frontend is accessed via HTTPS; otherwise, cookies with the `Secure` flag will not be sent.

2. Configuring the SameSite Attribute

Web server settings can also enforce the SameSite attribute:

For Apache:

```
<IfModule mod_headers.c>
    Header onsuccess edit Set-Cookie (.*) "$1; SameSite=Strict"
</IfModule>
```

For Nginx (version 1.19.3+):

```
proxy_cookie_flags ~ samesite=Strict; # Replace ~ with 'zbx_session' for specificity
```

Enabling Content Security Policy (CSP) on the web server

To protect Zabbix frontend against Cross Site Scripting (XSS), data injection, and similar attacks, we recommend enabling Content Security Policy on the web server. To do so, configure the web server to return the HTTP header.

> **Attention:**
> The following CSP header configuration is only for the default Zabbix frontend installation and for cases when all content originates from the site's domain (excluding subdomains). A different CSP header configuration may be required if you are, for example, configuring the *URL* widget to display content from the site's subdomains or external domains, switching from *OpenStreetMap* to another map engine, or adding external CSS or widgets.

To enable CSP for your Zabbix frontend in Apache configuration, follow these steps:

1. Locate your virtual host's configuration file:

- `/etc/httpd/conf/httpd.conf` on RHEL-based systems
- `/etc/apache2/sites-available/000-default.conf` on Debian/Ubuntu

2. Add the following directive to your virtual host's configuration file:

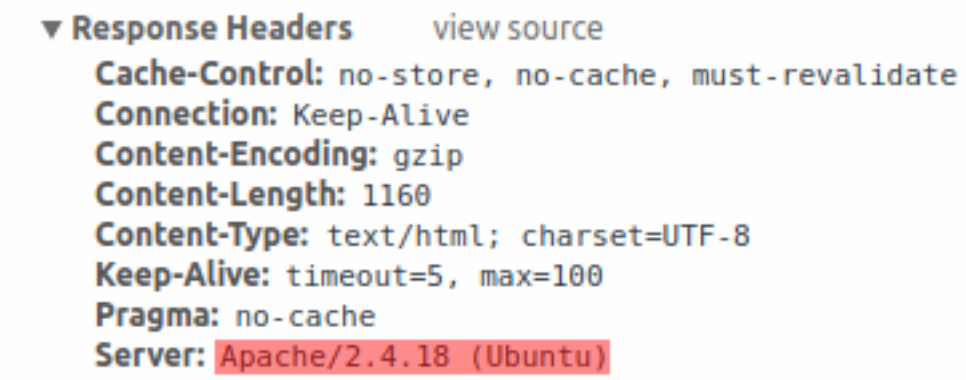
```
<VirtualHost *:*>
    Header set Content-Security-Policy: "default-src 'self' *.openstreetmap.org; script-src 'self' 'unsafe
</VirtualHost>
```

3. Restart the Apache service to apply the changes:

```
#### On RHEL-based systems:
systemctl restart httpd.service

#### On Debian/Ubuntu
systemctl restart apache2.service
```

禁用曝光的 Web 服务器信息

建议在 Web 服务器强化过程中禁用所有 Web 服务器签名。默认情况下，Web 服务器正在公开软件签名：



可以通过向 Apache（用作示例）配置文件添加两行来禁用签名：

```
ServerSignature Off
ServerTokens Prod
```

可以通过更改 php.ini 配置文件来禁用 PHP 签名（X-Powered-By HTTP header）（默认情况下禁用签名）：

```
expose_php = Off
```

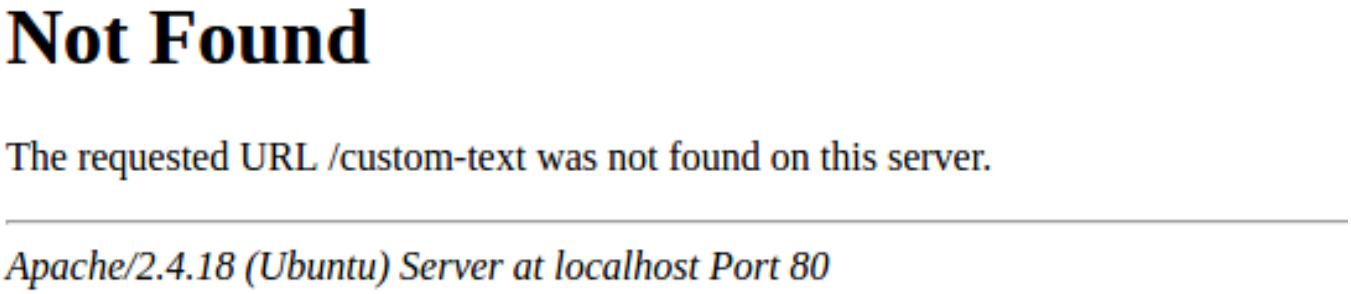
若使应用配置文件更改生效，需重新启动 Web 服务器。

通过在 Apache 中使用 mod_security（libapache2-mod-security2）可以实现额外的安全级别。mod_security 允许删除服务器签名，而不是仅仅从服务器签名中删除版本。通过在安装 mod_security 之后将 “SecServerSignature” 更改为任何所需的值，可以将签名更改为任何值。

请参阅 Web 服务器的文档以获取有关如何删除/更改软件签名的帮助。

禁用默认 Web 服务器错误页面

建议禁用默认错误页面以避免信息泄露。Web 服务器默认使用内置错误页面：



作为 Web 服务器强化过程的一部分，应替换/删除默认错误页面。“ErrorDocument” 指令可用于为 Apache Web 服务器定义自定义错误页面/文本（用作示例）。

请参阅您的 Web 服务器的文档以查找有关如何替换/删除默认错误页面的帮助。

删除 web 服务器测试页面

建议移除 web 服务器测试页面，以免信息泄露。默认情况下，Web 服务器 webroot 包含一个名为 index.html 的测试页面（以 Ubuntu 上的 Apache2 为例）：

[] (../../../../assets/en/manual/installation/requirements/test_page.png)

作为 web 服务器加固过程的一部分，测试页面应该被删除或设为不可用。

Zabbix 设置

在默认情况下，Zabbix 配置了 *X-Frame-Options HTTP* 响应头 设置为 SAMEORIGIN，这意味着内容只能加载到与页面本身具有相同来源的框架中。

从外部 URL 中提取内容的 Zabbix 前端元素（即 URL dashboard widget）在启用了所有沙盒限制的沙盒中显示检索到的内容。

这些设置增强了 Zabbix 前端的安全性，并提供针对 XSS 和点击劫持攻击的保护。超级管理员可以根据需要修改 *iframe sandboxing* 和 *X-Frame-Options HTTP response header* 参数。在更改默认设置之前，请仔细权衡风险和效果。不建议完全关闭沙盒及 X-Frame-Options。

使用普通密码隐藏文件

增加密码复杂度避免暴力破解密码，建议修改 Web 服务器配置限制 ui/data/top_passwords.txt 文件的访问。该文件包含最普通以及上下文特定的密码，防止用户设置这些密码，如果密码策略的 *Avoid easy-to-guess passwords* 参数被启用。

例如，使用 `location` 指令限制 NGINX 文件访问：

location = /data/top_passwords.txt { deny all; return 404; }

通过 `.htacess` 文件限制 Apache：

<Files ”top_passwords.txt”>
Order Allow,Deny Deny from all </Files>

UTF-8 编码

UTF-8 是 Zabbix 唯一支持的编码类型。众所周知，应用该类型编码可使 Zabbix 正常运转且没有任何安全漏洞。请用户注意，如果使用其它一些编码类型，我们发现会出现许多安全问题。

Windows 安装程序路径

使用 Windows 安装程序时，建议使用安装程序提供的默认路径，因为使用没有适当权限的自定义路径可能会危及安装的安全性。

Zabbix 安全公告和 CVE 数据库

请参阅Zabbix 安全公告和 CVE 数据库 (https://www.zabbix.com/security_advisories)。

3 从源代码安装

你可以通过从源代码编译获得最新的 Zabbix 版本。

这里教程提供了从源代码安装 Zabbix 的详细步骤。

1 安装 Zabbix 守护进程

1 下载源代码压缩包

前往Zabbix 下载页面下载源代码压缩包。并使用如下命令解压下载好的源代码：

```
$ tar -zxvf zabbix-6.0.0.tar.gz
```

> **Note:**
> 命令中的 Zabbix 版本必须输入正确。它必须与所下载的压缩包名称一致。

2 创建用户账户

所有的 Zabbix 守护进程都必须要一个非特权用户。如果一个非特权用户启动了一个 Zabbix 守护进程，它就会以这个用户运行。

然而，如果一个守护进程以 ‘root’ 账户运行，它会切换到一个 ‘zabbix’用户，这个用户是必须存在的。要创建这样一个用户（在它自己的 “zabbix” 组里），

在基于 RedHat 的系统里，运行：

```
groupadd --system zabbix
useradd --system -g zabbix -d /usr/lib/zabbix -s /sbin/nologin -c "Zabbix Monitoring System" zabbix
```

在基于 Debian 的系统里，运行：

```
addgroup --system --quiet zabbix
adduser --quiet --system --disabled-login --ingroup zabbix --home /var/lib/zabbix --no-create-home zabbix
```

Attention:
Zabbix 进程不许要 home 目录，因此我们不推荐创建它。然而，如果你将要使用的某些功能需要它（比如在 `$HOME/.my.cnf` 里存放 MySQL 凭证），你可以使用如下命令去创建 home 目录。

在基于 RedHat 的系统里，运行：
```
mkdir -m u=rwx,g=rwx,o= -p /usr/lib/zabbix
chown zabbix:zabbix /usr/lib/zabbix
```
在基于 Debian 的系统里，运行：
```
mkdir -m u=rwx,g=rwx,o= -p /var/lib/zabbix
chown zabbix:zabbix /var/lib/zabbix
```

安装 Zabbix 前端不许要单独的用户。

如果 Zabbix server和agent运行在同一台机器上，建议使用与 agent 不同的用户来运行。否则，如果两者使用相同的用户，agent 可以访问 server 的配置文件，Zabbix 里任何 Admin 级别的用户可以轻易地获取诸如数据可密码等信息。

Attention:
以 `root`、`bin` 或者其他任何有特殊权限的账户运行 Zabbix 都有安全风险。

3 创建 Zabbix 数据库

对 Zabbixserver和proxy守护进程，还有 Zabbix 前端，必须要有一个数据库。但运行 Zabbixagent不需要。

此处 SQL脚本用于创建数据库模式和插入数据集。Zabbix proxy 数据库只需要数据库模式，而 Zabbix server 数据库在数据库模式之上还需要数据集。

创建了 Zabbix 数据库之后，执行以下步骤来编译 Zabbix。

4 配置源代码

当为 Zabbix server 或 proxy 配置源码时，必须指定要使用的数据库类型。同一时间，只用一种数据库类型可以与 server 或 proxy 编译。

要查看所有支持的配置选项，在提取的 Zabbix 源代码目录运行：

```
./configure --help
```

要为 Zabbix server 和 agent 配置源代码，你可以执行类似如下命令：

```
./configure --enable-server --enable-agent --with-mysql --enable-ipv6 --with-net-snmp --with-libcurl --wit
```

要为 Zabbix server（和，比如 PostgreSQL）配置源代码，你可以执行：

```
./configure --enable-server --with-postgresql --with-net-snmp
```

要为 Zabbix server（和，比如 SQLite）配置源代码，你可以执行：

```
./configure --prefix=/usr --enable-proxy --with-net-snmp --with-sqlite3 --with-ssh2
```

要为 Zabbix agent 配置源代码，你可以执行：

```
./configure --enable-agent
```

或者 Zabbix agent 2：

```
./configure --enable-agent2
```

Note:
构建 Zabbix agent 2 需要一个用当前支持的Go 版本配置的 Go 环境。安装指导详见golang.org。

编译选项注意事项：

- 如果使用了--enable-agent 选项命令行实用程序 zabbix_get 和 zabbix_sender 会被编译。
- --with-libcurl 和 --with-libxml2 配置选项对虚拟机监控是必须的；--with-libcurl 对 SMTP 身份验证和 `web.page.*`Zabbix agent监控项. 也是必须的。请注意，cURL 7.20.0 或更高版本要求有 --with-libcurl 配置选项。
- Zabbix （从 3.4.0 版本开始）始终使用 PCRE 库进行编译；安装它不是可选的。--with-libpcre=[DIR] 只允许指向特定的基本安装目录，而不是搜索 libpcre 文件的一些常见位置。
- 你可以使用 --enable-static 标志来静态链接库。如果你计划在不同的服务器之间分发编译的二进制文件，你必须使用这个标志来使这些二进制文件在没有必须的库的情况下工作。请注意，--enable-static 在Solaris不同做
- 构建服务器时不建议使用 --enable-static 选项。为了静态构建服务器，你必须每个需要的外部库的静态版本。配置脚本中不会严格地检查这些。

- 将可选路径添加到 MySQL 配置文件 --with-mysql=/<path_to_the_file>/mysql_config，以便在需要使用不在默认位置的 MySQL 客户端库时选择所需的 MySQL 客户端。当在同一系统上安装了多个版本的 MySQL 或 MariaDB 与 MySQL 一起安装时，这个配置很有用。
- 使用 --with-oracle 标志指定 OCI API 的位置。

Attention:
如果./configure 由于缺少库或者其他条件而失败，请查看 `config.log` 文件获取错误的更多详细信息。例如，如果 `libssl` 缺失，即时错误信息可能具有误导性：

```
checking for main in -lmysqlclient... no
configure: error: Not found mysqlclient library
```

而 `config.log` 则有更多详细的描述：

```
/usr/bin/ld: cannot find -lssl
/usr/bin/ld: cannot find -lcrypto
```

另见：

- 用加密支持编译 Zabbix
- 已知问题在 HP-UX 上编译 Zabbix

5 Make 和 install 所有

Note:
如果从Zabbix Git repository安装，必须先执行：`$ make dbschema`

```
make install
```

这一步应该以具有足够权限的用户身份运行（通常是'root'，或使用 sudo）。

运行 `make install` 将默认在 /usr/local/sbin 目录安装守护进程二进制文件 (zabbix_server, zabbix_agentd, zabbix_proxy) ，在 /usr/local/bin ，目录安装客户端二进制文件（zabbix_get、zabbix_sender）。

Note:
要指定与 /usr/local 不同的位置，在之前配置源的步骤中使用 --prefix 键，例如 --prefix=/home/zabbix。在这种情况下，守护程序二进制文件将安装在 <prefix>/sbin 下，而实用程序则安装在 <prefix>/bin 下，手册页将安装在 <prefix>/share 下。

6 查看和编辑配置文件

- 编辑 Zabbix agent 配置文件 **/usr/local/etc/zabbix_agentd.conf**

你需要为每个安装了 zabbix_agentd 的主机配置此文件。

你必须在文件中指定 Zabbix 服务器 **IP** 地址，来自其他主机的连接将被拒绝。

- 编辑 Zabbix 服务器配置文件 **/usr/local/etc/zabbix_server.conf**

你必须指定数据库名称、用户和密码（如果使用了）。

如果你是小型安装（最多十台受监控的主机），其他参数的默认值将适合你。如果你想最大化 Zabbix server（或 proxy）的性能，你应该修改默认参数。更多信息详见性能调优 章节。

- 如果你安装了 Zabbix proxy，编辑 proxy 配置文件 **/usr/local/etc/zabbix_proxy.conf**

你必须指定服务器 IP 地址和 proxy 的主机名（服务器必须知道），以及数据库名称、用户和密码（如果使用了）。

Note:
使用 SQLite 必须指定数据库文件的完整路径；不需要数据库用户和密码。

7 启动守护进程

在服务器端运行 zabbix_server。

```
shell> zabbix_server
```

Note:
确保你的系统允许分配 36MB （或稍微多一点）的共享内存，否则 server 可能无法启动，你会在 server 的日志文件里看到 "不能为 <type of cache> 分配共享内存"。这可能在 FreeBSD 和 Solaris 8 上发生。
请参阅本页底部的"另请参阅" 部分，了解如何配置共享内存。

在所有被监控的机器上运行 zabbix_agentd。

```
shell> zabbix_agentd
```

> **Note:**
> 确保你的系统允许分配 2MB 的共享内存，否则 agent 可能无法启动，你会在 agent 的日志文件里看到 “无法为收集器分配共享内存。” 这可能在 Solaris 8 上发生。

如果你安装了 Zabbix proxy，运行 zabbix_proxy。

```
shell> zabbix_proxy
```

2 安装 Zabbix 网页界面

复制 PHP 文件

Zabbix 前端是 PHP 编写的，所以运行它需要 PHP 支持的网络服务器。安装只需简单的从 ui 目录复制 PHP 文件到网络服务器 HTML 文档目录。

Apache 网络服务器的 HTML 文档目录的常见位置包括：

- /usr/local/apache2/htdocs (从源代码安装 Apache 的默认目录)
- /srv/www/htdocs (OpenSUSE, SLES)
- /var/www/html (Debian, Ubuntu, Fedora, RHEL, CentOS)

建议使用子目录而非 HTML 根目录。要创建子目录并将 Zabbix 前端文件复制过去，请执行如下命令，以替换实际目录：

```
mkdir <htdocs>/zabbix
cd ui
cp -a . <htdocs>/zabbix
```

如果计划用英语之外的语言，请参考前端安装其他语言 。

安装前端

关于 Zabbix 前端的安装，请参考网页界面安装页面的信息。

3 安装 Java gateway

只有在你想监控 JMX 应用程序时，才需要安装 Java gateway。Java gateway 是轻量级的，不需要数据库。

从源代码安装，先下载 并解压源代码压缩包。

要编译 Java gateway，请带 `--enable-java` 选项执行 `./configure`。建议指定 `--prefix` 选项来请求默认的 /usr/local 以外的安装路径，因为安装 Java gateway 将创建一个完整的目录树，而不仅仅是一个可执行文件。

```
$ ./configure --enable-java --prefix=$PREFIX
```

要将 Java gateway 编译并打包到一个 JAR 文件中，执行 `make`。请注意，这一步你的路径中可能需 `javac` 和 `jar` 可执行文件。

```
$ make
```

现在你在 src/zabbix_java/bin 目录有一个 zabbix-java-gateway-$VERSION.jar 文件。如果你对从分配的目录中的 src/zabbix_java 运行 Java gateway 感到满意，那么你可以继续按指导配置和运行Java gateway。否则，请确保您有足够的权限并执行 `make install`。

```
$ make install
```

继续设置以获取更多关于配置和运行 Java gateway 的详细信息。

4 安装 Zabbix web 服务

只有当你想使用定时报表时，才需要安装 Zabbix web 服务。

要从源代码安装，请先下载 和解压源代码压缩包。

要编译 Zabbix web 服务，请带 `--enable-webservice` 选项执行 `./configure`。

> **Note:**
> 构建 Zabbix web 服务需要配置好的 Go 1.13 + 版本的环境。

在安装了 web 服务的机器上运行 zabbix_web_service：

```
shell> zabbix_web_service
```

可在web 服务 了解关于配置定时报表生成的更多信息。

在 **macOS** 上构建 **Zabbix agent**

概述

本节演示如何从包含或不包含 TLS 的源代码构建 Zabbix macOS agent 二进制文件。

必要条件

您将需要命令行开发人员工具（不需要 Xcode），Automake，pkg-config 和 PCRE（v8.x）或 PCRE2（v10.x）。如果要使用 TLS 构建 agent 二进制文件，则还需要 OpenSSL 或 GnuTLS。

要安装 Automake 和 pkg-config ，您将需要来自 https://brew.sh/ 的软件包管理器。要安装它，请打开终端并运行以下命令：

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

然后安装 Automake 和 pkg-config：

```
$ brew install automake
$ brew install pkg-config
```

如何准备 PCRE、OpenSSL 和 GnuTLS 库取决于它们如何链接到 agent。如果您打算在已具有这些库的 macOS 计算机上运行代理二进制文件，则可以使用 Homebrew 提供的预编译库。这些通常是 macOS 机器，它们使用 Homebrew 来构建 Zabbix agent 二进制文件或用于其他目的。

如果 agent 的二进制文件将在没有共享版本的库的 macOS 计算机上使用，则应从源代码编译静态库，并将 Zabbix agent 与它们链接。

使用共享库构建 agent 二进制文件

安装 PCRE2 (如果需要的话，在下面的命令中将 *pcre2* 替换为 *pcre*)：

```
$ brew install pcre2
```

使用 TLS 构建时，请安装 OpenSSL 和/或 GnuTLS：

```
$ brew install openssl
$ brew install gnutls
```

下载 Zabbix 源代码：

```
$ git clone https://git.zabbix.com/scm/zbx/zabbix.git
```

不使用 TLS 构建 agent：

```
$ cd zabbix
$ ./bootstrap.sh
$ ./configure --sysconfdir=/usr/local/etc/zabbix --enable-agent --enable-ipv6
$ make
$ make install
```

使用 OpenSSL 构建 agent：

```
$ cd zabbix
$ ./bootstrap.sh
$ ./configure --sysconfdir=/usr/local/etc/zabbix --enable-agent --enable-ipv6 --with-openssl=/usr/local/op
$ make
$ make install
```

使用 GnuTLS 构建 agent：

```
$ cd zabbix-source/
$ ./bootstrap.sh
$ ./configure --sysconfdir=/usr/local/etc/zabbix --enable-agent --enable-ipv6 --with-gnutls=/usr/local/opt
$ make
$ make install
```

使用不带 TLS 的静态库构建 agent 二进制文件

让我们假设 PCRE 静态库将安装在 $HOME/static-libs 中，我们将使用 PCRE2 10.39。

```
$ PCRE_PREFIX="$HOME/static-libs/pcre2-10.39"
```

下载并构建具有 Unicode 支持的 PCRE：

```
$ mkdir static-libs-source
$ cd static-libs-source
$ curl --remote-name https://github.com/PhilipHazel/pcre2/releases/download/pcre2-10.39/pcre2-10.39.tar.gz
```

```
$ tar xf pcre2-10.39.tar.gz
$ cd pcre2-10.39
$ ./configure --prefix="$PCRE_PREFIX" --disable-shared --enable-static --enable-unicode-properties
$ make
$ make check
$ make install
```

下载 Zabbix 源代码并构建 agent：

```
$ git clone https://git.zabbix.com/scm/zbx/zabbix.git
$ cd zabbix
$ ./bootstrap.sh
$ ./configure --sysconfdir=/usr/local/etc/zabbix --enable-agent --enable-ipv6 --with-libpcre2="$PCRE_PREFI
$ make
$ make install
```

使用 OpenSSL 构建带有静态库的 agent 二进制文件

构建 OpenSSL 时，建议在成功构建后运行 `make test`。即使构建成功，测试有时也会失败。在这种情况下，应进行研究并解决问题，然后再继续。

让我们假设 PCRE 和 OpenSSL 静态库将安装在 "$HOME/static-libs" 中。我们将使用 PCRE2 10.39 和 OpenSSL 1.1.1a。

```
$ PCRE_PREFIX="$HOME/static-libs/pcre2-10.39"
$ OPENSSL_PREFIX="$HOME/static-libs/openssl-1.1.1a"
```

让我们在 `static-libs-source` 中构建静态库：

```
$ mkdir static-libs-source
$ cd static-libs-source
```

下载并构建具有 Unicode 支持的 PCRE：

```
$ curl --remote-name https://github.com/PhilipHazel/pcre2/releases/download/pcre2-10.39/pcre2-10.39.tar.gz
$ tar xf pcre2-10.39.tar.gz
$ cd pcre2-10.39
$ ./configure --prefix="$PCRE_PREFIX" --disable-shared --enable-static --enable-unicode-properties
$ make
$ make check
$ make install
$ cd ..
```

下载并构建 OpenSSL：

```
$ curl --remote-name https://www.openssl.org/source/openssl-1.1.1a.tar.gz
$ tar xf openssl-1.1.1a.tar.gz
$ cd openssl-1.1.1a
$ ./Configure --prefix="$OPENSSL_PREFIX" --openssldir="$OPENSSL_PREFIX" --api=1.1.0 no-shared no-capieng n
$ make
$ make test
$ make install_sw
$ cd ..
```

下载 Zabbix 源代码并构建 agent：

```
$ git clone https://git.zabbix.com/scm/zbx/zabbix.git
$ cd zabbix
$ ./bootstrap.sh
$ ./configure --sysconfdir=/usr/local/etc/zabbix --enable-agent --enable-ipv6 --with-libpcre2="$PCRE_PREFI
$ make
$ make install
```

使用带有 GnuTLS 的静态库构建 agent 二进制文件

GnuTLS 依赖于 Nettle 加密后端和 GMP 算法库。本文将使用 Nettle 中包含的 mini-gmp，而不是使用完整的 GMP 库。

构建 GnuTLS 和 Nettle 时，建议在成功构建后运行 `make check`。即使构建成功，测试有时也会失败。在这种情况下，应进行研究并解决问题，然后再继续。

让我们假设 PCRE、Nettle 和 GnuTLS 静态库将被安装在 `$HOME/static-libs`。我们将使用 PCRE2 10.39、Nettle 3.4.1 和 GnuTLS 3.6.5。

```
$ PCRE_PREFIX="$HOME/static-libs/pcre2-10.39"
$ NETTLE_PREFIX="$HOME/static-libs/nettle-3.4.1"
$ GNUTLS_PREFIX="$HOME/static-libs/gnutls-3.6.5"
```

让我们在 static-libs-source 中构建静态库：

```
$ mkdir static-libs-source
$ cd static-libs-source
```

下载并构建 Nettle:

```
$ curl --remote-name https://ftp.gnu.org/gnu/nettle/nettle-3.4.1.tar.gz
$ tar xf nettle-3.4.1.tar.gz
$ cd nettle-3.4.1
$ ./configure --prefix="$NETTLE_PREFIX" --enable-static --disable-shared --disable-documentation --disable
$ make
$ make check
$ make install
$ cd ..
```

下载并构建 GnuTLS:

```
$ curl --remote-name https://www.gnupg.org/ftp/gcrypt/gnutls/v3.6/gnutls-3.6.5.tar.xz
$ tar xf gnutls-3.6.5.tar.xz
$ cd gnutls-3.6.5
$ PKG_CONFIG_PATH="$NETTLE_PREFIX/lib/pkgconfig" ./configure --prefix="$GNUTLS_PREFIX" --enable-static --d
$ make
$ make check
$ make install
$ cd ..
```

下载 Zabbix 源代码并构建 agent：

```
$ git clone https://git.zabbix.com/scm/zbx/zabbix.git
$ cd zabbix
$ ./bootstrap.sh
$ CFLAGS="-Wno-unused-command-line-argument -framework Foundation -framework Security" \
> LIBS="-lgnutls -lhogweed -lnettle" \
> LDFLAGS="-L$GNUTLS_PREFIX/lib -L$NETTLE_PREFIX/lib" \
> ./configure --sysconfdir=/usr/local/etc/zabbix --enable-agent --enable-ipv6 --with-libpcre2="$PCRE_PREFI
$ make
$ make install
```

在 **Windows** 上构建 **Zabbix agent**

概述

本节将演示如何从有或没有 TLS 的源构建 Zabbix Windows agent 二进制文件。

编译 OpenSSL

接下来的步骤将助你在 MS Windows 10（64 位）的源中编译 OpenSSL。

1. 编译 OpenSSL 你需要在 Windows 机器上安装：
    1. C compiler (e.g. VS 2017 RC),
    2. NASM (https://www.nasm.us/),
    3. Perl (e.g. Strawberry Perl from http://strawberryperl.com/),
    4. Perl module Text::Template (cpan Text::Template).
2. 获取 OpenSSL 源代码 https://www.openssl.org/。这里用 OpenSSL 1.1.1 版本。
3. 解压 OpenSSL 源，例如，解压在 E:\openssl-1.1.1。
4. 打开命令行窗口。例如，VS 2017 RC 的 x64 原生工具命令提示符。
5. 至 OpenSSL 源目录, 例如，E:\openssl-1.1.1。
    1. 验证 NASM 能被找到：`e:\openssl-1.1.1> nasm --version  NASM version 2.13.01 compiled on May  1 2017`
6. 配置 OpenSSL，例如：`e:\openssl-1.1.1> perl E:\openssl-1.1.1\Configure VC-WIN64A no-shared no-capieng no-srp no-gost no-dgram no-dtls1-method no-dtls1_2-method  --api=1.1.0 --prefix=C:\OpenSSL-Win64-111- --openssldir=C:\OpenSSL-Win64-111-static`

- 请注意选项'no-shared'：如果使用'no-shared' 那么 OpenSSL 静态库 libcrypto.lib 和 libssl.lib 将' 自给自足'，生成的 Zabbix 二进制文件本身将包含 OpenSSL ，不需要外部的 OpenSSL DLLs。优点：Zabbix 二进制文件可以复制到其他没有 OpenSSL 库的 Windows 机器上。缺点：当新的 OpenSSL bugfix 版本发布时，需要重新编译并安装 Zabbix agent。
- 如果不使用'no-shared'，那么静态库 libcrypto.lib 和 libssl.lib 会在运行时使用 OpenSSL DLLs 。
- 优点：当新的 OpenSSL bugfix 版本发布时，你可能只需要升级 OpenSSL DLLs 不用重新编译 Zabbix agent。
- 缺点：复制 Zabbix agent 到另一个机器时，需要同时复制 OpenSSL DLLs。

7. 编译 OpenSSL，运行测试，安装：`e:\openssl-1.1.1> nmake e:\openssl-1.1.1> nmake test ... All tests successful. Files=152, Tests=1152, 501 wallclock secs ( 0.67 usr + 0.61 sys = 1.28 CPU) Result: PASS e:\openssl-1.1.1> nmake install_sw`'install_sw' 仅安装软件组件（例如库，头文件，但不安装文档）。如果你希望安装所有的文件，请用"nmake install"。

编译 PCRE

1. 从 pcre.org (https://github.com/PhilipHazel/pcre2/releases/download/pcre2-10.39/pcre2-10.39.zip) 存储库中下载 PCRE 或 PCRE2 (Zabbix 6.0 以上支持) 库：
2. 提取到目录 *E:\pcre2-10.39*
3. 从https://cmake.org/download/ 安装 CMake，安装过程中选择：确保 cmake\bin 在你的路径的 (测试版本 3.9.4).
4. 创建一个新的空的构建目录，最好是源目录的子目录。例如 *E:\pcre2-10.39\build*.
5. 打开命令行窗口，例如 VS 2017 上的 x64 原生工具命令提示符，并且从该外部环境运行 cmake-gui。不要试图从窗口开始菜单启动 Cmake，因为这可能会导致错误。
6. 分别为源目录和构建目录输入 *E:\pcre2-10.39* 和 *E:\pcre2-10.39\build* 。
7. 点击"Configure" 按钮。
8. 为此项目指定生成器时，请选择"NMake Makefiles"。
9. 创建一个新的空的安装目录。例如，*E:\pcre2-10.39-install*。
10. GUI 将列出几个配置选项。确保选择了以下几个选项：
    - **PCRE_SUPPORT_UNICODE_PROPERTIES** ON
    - **PCRE_SUPPORT_UTF** ON
    - **CMAKE_INSTALL_PREFIX** *E:\pcre2-10.39-install*
11. 再次点击"Configure" 。相邻的"Generate" 按钮需要启用。
12. 点击"Generate".
13. 如果出现错误，建议在尝试重复构建 CMake 的过程中删除 CMake 缓存。在 CMake GUI 中，缓存可以通过选择"File > Delete Cache" 来删除。
14. 构建目录现在应该包含了一个可用的构建系统 - *Makefile*.
15. 打开命令行窗口，例如 VS 2017 上的 x64 原生工具命令提示符，导航到上面提到的 *Makefile* 。
16. 运行 NMake 命令：`E:\pcre2-10.39\build> nmake install`

编译 Zabbix

以下步骤将助你从 MS Windows 10 (64 位) 上的源码中编译 Zabbix。在编译有或没有 TLS 支持的 Zabbix 时，唯一显著的区别在步骤 4。

1. 在 Linux 机器上，检查 git 源代码：`$ git clone https://git.zabbix.com/scm/zbx/zabbix.git $ cd zabbix $ ./bootstrap.sh $ ./configure --enable-agent --enable-ipv6 --prefix=`pwd` $ make dbschema $ make dist`
2. 在 Windows 机器上复制和解压归档文件，例如 zabbix-4.4.0.tar.gz。
3. 我们假设源代码在 e:\zabbix-4.4.0 中。打开命令行窗口，例如 VS 2017 RC 中的 x64 原生工具命令提示符。转至 E:\zabbix-4.4.0\build\win32\project。
4. 编译 zabbix_get，zabbix_sender 和 zabbix_agent。
    - 无 TLS：`E:\zabbix-4.4.0\build\win32\project> nmake /K PCREINCDIR=E:\pcre2-10.39-install\include PCRELIBDIR=E:\pcre2-10.39-install\lib`
    - 有 TLS：`E:\zabbix-4.4.0\build\win32\project> nmake /K -f Makefile_get TLS=openssl TLSINCDIR=C:\Open TLSLIBDIR=C:\OpenSSL-Win64-111-static\lib PCREINCDIR=E:\pcre2-10.39-install\include PCRELIBDIR=E:\pcre2-10.39-install\lib E:\zabbix-4.4.0\build\win32\project> nmake /K -f Makefile_sender TLS=openssl TLSINCDIR="C:\OpenSSL-Win64-111-static\include TLSLIBDIR="C:\OpenS PCREINCDIR=E:\pcre2-10.39-install\include PCRELIBDIR=E:\pcre2-10.39-install\lib E:\zabbix-4.4 nmake /K -f Makefile_agent TLS=openssl TLSINCDIR=C:\OpenSSL-Win64-111-static\include TLSLIBDIR=C:\OpenSSL-Win64-111-static\lib PCREINCDIR=E:\pcre2-10.39-install\include PCRELIBDIR=E:\pcre2-10.39-install\lib`
5. 新的二进制文件位于 e:\zabbix-4.4.0\bin\win64。由于 OpenSSL 是用'no-shared' 选项编译的，Zabbix 二进制文件本身包含 OpenSSL，可以复制到其他没有 OpenSSL 的机器中。

用 LibreSSL 编译 Zabbix

该过程类似于使用 OpenSSL 编译，但是你需要对位于 `build\win32\project` 目录中的文件进行一些小的改变：

`* In ''Makefile_tls'' delete ''/DHAVE_OPENSSL_WITH_PSK''. i.e. find <code>`

CFLAGS = $(CFLAGS) /DHAVE_OPENSSL /DHAVE_OPENSSL_WITH_PSK</code> 然后用 `CFLAGS = $(CFLAGS) /DHAVE_OPENSSL`

进行替换

```
  * In ''Makefile_common.inc'' add ''/NODEFAULTLIB:LIBCMT'' i.e. find <code>
```

/MANIFESTUAC:”level=’asInvoker’ uiAccess=’false’” /DYNAMICBASE:NO /PDB:$(TARGETDIR)\$(TARGETNAME).pdb</code> 然后用 `/MANIFESTUAC:"level='asInvoker' uiAccess='false'" /DYNAMICBASE:NO /PDB:$(TARGETDIR)\$(TARGETNAME).pdb /NODEFAULTLIB:LIBCMT` 进行替换

在 **Windows** 中构建 **Zabbix agent 2**

概述

本节将演示如何从源代码构建 Zabbix agent 2 (Windows)。

安装 MinGW 编译器

1. 下载带有 SJLJ (设置跳转/长跳转) 异常处理和窗口线程的 MinGW-w64 (例如 *x86_64-8.1.0-release-win32-sjlj-rt_v6-rev0.7z*)
2. 提取并移动到 *c:\mingw*
3. 设置环境变量

```
@echo off
set PATH=%PATH%;c:\mingw\bin
cmd
```

编译时使用 Windows 提示符代替 MinGW 提供的 MSYS 终端。

编译 PCRE 开发库

以下说明将编译并安装 *c:\dev\pcre* 中的 64 位 PCRE 库和 *c:\dev\pcre32* 的 32 位库:

1. 从 pcre.org(http://ftp.pcre.org/pub/pcre/) 下载 PCRE 8.XX 版本库，然后提取
2. 打开 *cmd* 并导航到提取的源

构建 64 位 PCRE

1. 删除就配置/缓存（如果有）:

```
del CMakeCache.txt
rmdir /q /s CMakeFiles
```

2. 运行 cmake (CMake 可从这里安装https://cmake.org/download/):

```
cmake -G "MinGW Makefiles" -DCMAKE_C_COMPILER=gcc -DCMAKE_C_FLAGS="-O2 -g" -DCMAKE_CXX_FLAGS="-O2 -g" -DCM
```

3. 接下来，运行：

```
mingw32-make clean
mingw32-make install
```

构建 32 位 PCRE

1. 运行：

```
mingw32-make clean
```

2. 删除 *CMakeCache.txt*：

```
del CMakeCache.txt
rmdir /q /s CMakeFiles
```

3. 运行 cmake：

```
cmake -G "MinGW Makefiles" -DCMAKE_C_COMPILER=gcc -DCMAKE_C_FLAGS="-m32 -O2 -g" -DCMAKE_CXX_FLAGS="-m32 -O
```

4. 接下来，运行：

```
mingw32-make install
```

安装 OpenSSL 开发库

1. 从 https://curl.se/windows/ 下载 32 和 64 位版本
2. 相应地将文件提取到 *c:\dev\openssl32* 和 *c:\dev\openssl*。
3. 然后删除提取的 **.dll.a* (dll call wrapper libraries) ，因为 MinGW 在静态库前会优先考虑它们。

编译 Zabbix agent 2

32 位

打开 MinGW 环境（Windows 命令提示符）并导航至 Zabbix 源树中的 *build/mingw* 目录。

运行：

```
mingw32-make clean
mingw32-make ARCH=x86 PCRE=c:\dev\pcre32 OPENSSL=c:\dev\openssl32
```

64 位

打开 MinGW 环境（Windows 命令提示符）并导航至 Zabbix 源树目录中的 *build/mingw*。

运行：

```
mingw32-make clean
mingw32-make PCRE=c:\dev\pcre OPENSSL=c:\dev\openssl
```

> **Note:**
> 32 和 64 位版本都可以构建在 64 位的平台上，但是 32 位平台只能构建 32 位版本。在 32 位平台上运行时，请遵循 64 位版本在 64 位平台上运行的步骤。

4 从二进制包安装

使用 ZABBIX 官方存储库

Zabbix SIA 提供如下官方 RPM 和 DEB 包:

- Red Hat Enterprise Linux/CentOS
- Debian/Ubuntu/Raspbian
- SUSE Linux Enterprise Server

yum/dnf, apt 和 zypper 的各种操作系统发行版的软件包文件可以在repo.zabbix.com上找到。

请注意，尽管某些操作系统发行版（特别是基于 Debian 的发行版）提供了它们自己的 Zabbix 包，但 Zabbix 不支持这些包。第三方提供的 Zabbix 包可能已过时，缺乏最新的特性和 bug 修复。推荐只使用repo.zabbix.com上的官方软件包。如果之前用过非官方的 Zabbix 包，请参阅upgrading Zabbix packages from OS repositories的说明操作。

1 Red Hat 企业版 **Linux**

概述

适用于 Red Hat Enterprise Linux 和 Oracle Linux 的官方 Zabbix 6.0 LTS 软件包可在 Zabbix 网站。

软件包可用于 MySQL/PostgreSQL 数据库和 Apache/Nginx 网络服务器支持。

Zabbix agent 包和实用程序 *Zabbix get* 和 *Zabbix sender* 在 Zabbix 官方存储库上可用 RHEL 9, RHEL 8, RHEL 7, RHEL 6，以及 RHEL 5。

Zabbix 官方存储库也提供 *fping*、*iksemel* 和 *libssh2* 包。这些包位于 不支持 目录。

EL9 的 EPEL 存储库也提供了 Zabbix 包。如果同时安装了官方 Zabbix 存储库和 EPEL 存储库，那么 EPEL 中的 Zabbix 包必须通过在 `/etc/yum.repos.d/` 下的 EPEL 存储库配置文件中添加以下子句来排除：

·[epel] ·... ·excludepkgs=zabbix*

安装注意事项

请参阅下载页面中每个平台的安装说明：

- ·安装存储库
- ·安装服务器/代理/前端
- ·创建初始数据库，导入初始数据
- ·为 Zabbix 服务器配置数据库
- ·为 Zabbix 前端配置 PHP
- ·启动服务器/代理进程
- ·配置 Zabbix 前端

如果您想以 root 身份运行 Zabbix agent，请参阅Running agent as root。

Zabbix web 服务进程，用于 [计划报告生成]（/manual/web_interface/frontend_sections/reports/scheduled），需要谷歌浏览器。浏览器不包含在包中，必须手动安装。

使用时序数据库导入数据

使用 TimescaleDB，除了 PostgreSQL 的导入命令外，还需运行：

·# cat /usr/share/zabbix-sql-scripts/postgresql/timescaledb.sql | sudo -u zabbix psql zabbix

> **Warning:**
> 只有 Zabbix 服务器支持 TimescaleDB。

PHP 7.2

Zabbix 前端需要 **7.2** 或更新的 PHP 版本。

SELinux 配置

Zabbix 使用基于套接字的进程间通信。在启用 SELinux 的系统上，可能需要添加 SELinux 规则以允许 Zabbix 在 SocketDir 目录中创建/使用 UNIX 域套接字。当前套接字文件由服务器（告警器、预处理、IPMI）和代理（IPMI）使用。套接字文件是持久的，这意味着它们在进程运行时存在。

在强制模式下启用 SELinux 状态后，您需要执行以下命令以启用 Zabbix 前端和服务器之间的通信：

RHEL 7 及更高版本：

·# setsebool -P httpd_can_connect_zabbix on ·如果可以通过网络访问数据库（包括 PostgreSQL 中的 “localhost”），您也需要允许 Zabbix 前端连接到数据库 :·# setsebool -P httpd_can_network_connect_db on

7 之前的 RHEL：

·# setsebool -P httpd_can_network_connect on ·# setsebool -P zabbix_can_network on

前端和 SELinux 配置完成后，重启 Apache Web 服务器：

·# service httpd restart

此外，Zabbix 提供 zabbix-selinux-policy 包作为 RHEL 8 和 [RHEL 7] 源 RPM 包的一部分（http://repo.zabbix.com/zabbix/6.0/rhel/8/SRPMS/）。这个包为 SELinux 提供了一个基本的默认策略，并通过允许 Zabbix 创建和使用套接字并启用到 PostgreSQL 的 httpd 连接（由前端使用）使 zabbix 组件开箱即用。

源 *zabbix_policy.te* 文件包含以下规则 :·

·module zabbix_policy 1.2;

·require { ·type zabbix_t; ·type zabbix_port_t; ·type zabbix_var_run_t; ·type postgresql_port_t; ·type httpd_t; ·class tcp_socket name_connect; ·class sock_file { create unlink }; ·class unix_stream_socket connectto; ·} ··#============== zabbix_t ============== ·allow zabbix_t self:unix_stream_socket connectto; ·allow zabbix_t zabbix_port_t:tcp_socket name_connect; ·allow zabbix_t zabbix_var_run_t:sock_file create; ·allow zabbix_t zabbix_var_run_t:sock_file unlink; ·allow httpd_t zabbix_port_t:tcp_socket name_connect; ··#============== httpd_t ============== ·allow httpd_t postgresql_port_t:tcp_socket name_connect;

这个包的创建是为了防止用户因为配置复杂而关闭 SELinux。它包含足以加速 Zabbix 部署和配置的默认策略。为获得最高安全级别，建议设置自定义 SELinux 设置。

代理安装

添加所需的存储库后，您可以通过运行以下命令安装 Zabbix 代理：

·# dnf install zabbix-proxy-mysql zabbix-sql-scripts

将命令中的 “mysql” 替换为 “pgsql” 以使用 PostgreSQL，或替换为 “sqlite3” 以使用 SQLite3（仅限代理）。

“zabbix-sql-scripts” 包包含 Zabbix 服务器和 Zabbix 代理的所有支持的数据库管理系统的数据库模式，并将用于数据导入。

创建数据库

创建 Zabbix 代理的单独数据库。

Zabbix server 和 Zabbix proxy 不能使用同一个数据库。如果它们安装在同一台主机上，则代理数据库必须具有不同的名称。

导入数据

导入初始 schema：

·# cat /usr/share/zabbix-sql-scripts/mysql/proxy.sql | mysql -uzabbix -p zabbix

对于 PostgreSQL（或 SQLite）的代理：

·# cat /usr/share/zabbix-sql-scripts/postgresql/proxy.sql | sudo -u zabbix psql zabbix ·# cat /usr/share/zabbix-sql-scripts/sqlite3/proxy.sql | sqlite3 zabbix.db

为 Zabbix 代理配置数据库

编辑 zabbix_proxy.conf：

·# vi /etc/zabbix/zabbix_proxy.conf ·DBHost=localhost ·DBName=zabbix ·DBUser=zabbix ·DBPassword=<password>

在 Zabbix 代理的 DBName 中，使用与 Zabbix 服务器不同的数据库。

在 DBPassword 中使用 MySQL 的 Zabbix 数据库密码；PostgreSQL 的 PostgreSQL 用户密码。

在 PostgreSQL 中使用 DBHost=。您可能希望保留默认值设置 DBHost=localhost（或 IP 地址），但这会使 PostgreSQL 使用网络套接字连接到 Zabbix。有关说明，请参阅SELinux 配置 (/manual/installation/install_from_packages/rhel#selinux_configuration)。

启动 Zabbix 代理进程

要启动 Zabbix 代理进程并使其在系统启动时启动：

·# service zabbix-proxy start ·# systemctl enable zabbix-proxy

前端配置

Zabbix 代理没有前端；它只与 Zabbix 服务器通信。

Java 网关安装

只有在以下情况下才需要安装Java 网关 您想要监视 JMX 应用程序。Java 网关是轻量级的，不需要数据库。

添加所需的存储库后，您可以通过运行以下命令安装 Zabbix Java 网关：

·# dnf install zabbix-java-gateway

继续setup 了解有关配置和运行 Java 网关的更多详细信息。

安装调试信息包

> **Note:**
> Debuginfo 包目前可用于 RHE 版本 7、6 和 5。

要启用 debuginfo 存储库，请编辑*/etc/yum.repos.d/zabbix.repo* 文件。将 enabled=0 更改为 enabled=1 用于 zabbix-debuginfo 存储库。

·[zabbix-debuginfo] ·name=Zabbix Official Repository debuginfo - $basearch ·baseurl=http://repo.zabbix.com/zabbix/5.5/rhel/7/$basearch/deb ·enabled=0 ·gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-ZABBIX-A14FE591 ·gpgcheck=1

这将允许您安装 zabbix-debuginfo 包。

·# yum install zabbix-debuginfo

这个单一的包包含所有二进制 Zabbix 的调试信息组件。

2 Debian/Ubuntu/Raspbian

概述

官方 Zabbix 软件包可用于：

Debian 11 (Bullseye)	下载
Debian 10 (Buster)	下载
Debian 9 (Stretch)	下载
Ubuntu 20.04 (Focal Fossa) LTS	下载
Ubuntu 18.04 (Bionic Beaver) LTS	下载
Ubuntu 16.04 (Xenial Xerus) LTS	下载
Ubuntu 14.04 (Trusty Tahr) LTS	下载
Raspbian 11 (Bullseye)	下载
Raspbian 10 (Buster)	下载
Raspbian 9 (Stretch)	下载

软件包可用于 MySQL/PostgreSQL 数据库和 Apache/Nginx 网络服务器支持。

Zabbix 6.2 尚未发布。下载链接指向 6.2 之前的软件包。

安装注意事项

参见不同平台下载页面的安装说明 per platform in the download page for:

- 安装软件源
- 安装 server/agent/前端
- 创建初始数据库，导入初始数据
- 为 Zabbix server 配置数据库
- 为 Zabbix 前端配置 PHP
- 启动 server/agent 进程
- 配置 Zabbix 前端

如果要以 root 用户身份运行 Zabbix agent，请参见以 root 用户运行 agent。

Zabbix web service 进程用于生成定时报表，需要 Google 浏览器。在软件包中不包含浏览器，必须手动单独安装。

使用 Timescale DB 导入数据

使用 TimescaleDB，除了为 PostgreSQL 导入命令，还要执行：

```
# cat /usr/share/doc/zabbix-sql-scripts/postgresql/timescaledb.sql | sudo -u zabbix psql zabbix
```

Warning:
TimescaleDB 仅支持 Zabbix 服务器。

PHP 7.2

Zabbix 前端需要 PHP **7.2** 或更新的版本来启动 Zabbix 5.0。

在 Zabbix 前端安装低于 7.2 版本的 PHP 请参考instructions 。

配置 SELinux

参阅适用于 RHEL/CentOS 的SELinux 配置。

前端和 SELinux 配置好之后，重启 Apache 网络服务器：

```
# service apache2 restart
```

Proxy 安装

添加好所需软件源后，可通过执行以下命令来安装 Zabbix proxy：

```
# apt install zabbix-proxy-mysql
```

将命令中的'mysql' 替换为'pgsql' 以使用 PostgreSQL，或者替换为'sqlite3' 以使用 SQLite3 （仅 proxy 适用）。

创建数据库

为 Zabbix proxy 单独创建数据库 。

Zabbix server 和 Zabbix proxy 不能使用同一个数据库。如果他们是安装在同一个主机中的，则 proxy 数据库需要不同的命名。

导入数据

导入初始数据库模式

```
# cat /usr/share/doc/zabbix-sql-scripts/mysql/proxy.sql | mysql -uzabbix -p zabbix
```

对于使用 PostgreSQL（或 SQLite）的 proxy：

```
# cat /usr/share/doc/zabbix-sql-scripts/postgresql/proxy.sql | sudo -u zabbix psql zabbix
# cat /usr/share/doc/zabbix-sql-scripts/sqlite3/proxy.sql | sqlite3 zabbix.db
```

为 Zabbix proxy 配置数据库

编辑 zabbix_proxy.conf:

```
# vi /etc/zabbix/zabbix_proxy.conf
DBHost=localhost
DBName=zabbix
DBUser=zabbix
DBPassword=<password>
```

在 DBName 中为 Zabbix proxy 创建单独的数据库或重命名数据库。

在 DBPassword 中对 MySQL 使用 Zabbix 数据库密码；PosgreSQL 使用它自己的用户密码。

将 DBHost= 与 PostgreSQL 一起用，你可能需要保留默认设置 DBHost=localhost (或 1 个 IP 地址)，但这可能会使 PostgreSQL 通过网络套字连接到 Zabbix。参考 RHEL/CentOS 的Selinux 配置 获取说明。

启动 Zabbix proxy 进程

要启动 Zabbix proxy 进程并使其在系统启动时启动，请执行以下操作:

```
# systemctl restart zabbix-proxy
# systemctl enable zabbix-proxy
```

前端配置

Zabbix proxy 没有前端；它只与 Zabbix server 通信。

安装 Java gateway

只有当你想监控 JMX 应用程序时，才需要安装Java gateway 。Java gateway 是轻量级的不需要数据库。

添加了所需的软件源之后，就可执行如下命令安装 Zabbix Java gateway：

```
# apt install zabbix-java-gateway
```

了解更多关于配置和运行 Java gateway 的详细信息可跳转至java 设置 。

3 SUSE Linux 企业服务器

概述

官方 Zabbix 软件包可用于：

SUSE Linux 企业服务器 15	下载
SUSE Linux 企业服务器 12	下载

Note:
由于 MySQL 库较旧，验证 *Verify CA* 加密模式不适用于带有 MySQL 库的 SLES 12 （和所有次要操作系统版本）。

添加 Zabbix 软件源

安装软件源配置包。这个包里面有 yum（软件包管理器）配置文件。

SLES 15:

```
# rpm -Uvh --nosignature https://repo.zabbix.com/zabbix/6.0/sles/15/x86_64/zabbix-release-6.0-1.sles15.noa
# zypper --gpg-auto-import-keys refresh 'Zabbix Official Repository'
```

SLES 12:

```
# rpm -Uvh --nosignature https://repo.zabbix.com/zabbix/6.0/sles/12/x86_64/zabbix-release-6.0-1.sles12.noa
# zypper --gpg-auto-import-keys refresh 'Zabbix Official Repository'
```

请注意，Zabbix web 服务进程用于生成定时报表，需要安装 Google Chrome 浏览器。安装包内不含浏览器，需另外手动安装。

安装 Server/frontend/agent

安装支持 MySQL 的 Zabbix server/frontend/agent:

```
# zypper install zabbix-server-mysql zabbix-web-mysql zabbix-apache-conf zabbix-agent
```

如果将包用于 Nginx 网络服务器，请将命令中的’apache’ 替换为’nginx’ 。详见：在 SLES 12/15 上为 Zabbix 设置 Nginx。

若使用 Zabbix agen 2（仅 SLES 15 SP1+），需将命令中的’zabbix-agent’ 替换为’zabbix-agent2’ 。

安装支持 MySQL 的 Zabbix proxy：

```
# zypper install zabbix-proxy-mysql
```

将命令中的’mysql’ 替换为’pgsql’ 以使用 PostgreSQL。

创建数据库

Zabbix server 和proxy 守护进程需要数据库，运行 Zabbix agent不需要。

Warning:
Zabbix server 和 Zabbix proxy 不能使用同一个数据库，必须单独创建。因此，如果他们被安装在了同一个主机上，数据库要使用不同的名称创建！

使用提供的说明来创建数据库，MySQL 与PostgreSQL。

数据导入

使用 MySQL 导入 **server** 初始模式和数据：

```
# zcat /usr/share/doc/packages/zabbix-sql-scripts/mysql/create.sql.gz | mysql -uzabbix -p zabbix
```

系统将提示你输入新创建的数据库密码。

使用 PostgreSQL:

```
# zcat /usr/share/doc/packages/zabbix-sql-scripts/postgresql/create.sql.gz | sudo -u zabbix psql zabbix
```

使用 TimescaleDB，除了前面的命令，还要运行：

```
# zcat /usr/share/doc/packages/zabbix-sql-scripts/postgresql/timescaledb.sql.gz | sudo -u <username> psql
```

> **Warning:**
> 仅 Zabbix server 支持 TimescaleDB。

对于 proxy，导入初始模式：

```
# zcat /usr/share/doc/packages/zabbix-sql-scripts/mysql/schema.sql.gz | mysql -uzabbix -p zabbix
```

对于带有 PostgreSQL 的 proxy：

```
# zcat /usr/share/doc/packages/zabbix-sql-scripts/postgresql/schema.sql.gz | sudo -u zabbix psql zabbix
```

为 Zabbix server/proxy 配置数据库

编辑 /etc/zabbix/zabbix_server.conf 和 zabbix_proxy.conf 来使用它们各自的数据库。例如：

```
# vi /etc/zabbix/zabbix_server.conf
DBHost=localhost
DBName=zabbix
DBUser=zabbix
DBPassword=<password>
```

在 DBPassword 中对 MySQL 使用 Zabbix 数据库密码；在 PosgreSQL 中用 PosgreSQL 用户密码。

将 DBHost= 与 PostgreSQL 一起用，你可能希望保留默认值设置 DBHost=localhost (或一个 IP 地址)，但这会使 PostgreSQL 使用网络套接连接到 Zabbix。

Zabbix 前端配置

根据使用的网络服务器 (Apache/Nginx) 为 Zabbix 前端编辑相应配置文件：

- 对于 Apache，配置文件在 /etc/apache2/conf.d/zabbix.conf。一些 PHP 设置已经配置好了。但还是有必要取消"date.timezone" 设置的注释，设置正确的时区 。

```
php_value max_execution_time 300
php_value memory_limit 128M
php_value post_max_size 16M
php_value upload_max_filesize 2M
php_value max_input_time 300
php_value max_input_vars 10000
php_value always_populate_raw_post_data -1
# php_value date.timezone Europe/Riga
```

- zabbix-nginx-conf 包为 Zabbix 前端安装了单独的 Nginx server。它的配置文件位于/etc/nginx/conf.d/zabbix.conf。为了运行 Zabbix 前端，还是有必要取消注释并设置 listen 和/或 server_name 指令。

```
# listen 80;
# server_name example.com;
```

- Zabbix 为 Nginx 使用自己的专用 php-fpm 连接池：

它的配置文件位于/etc/php7/fpm/php-fpm.d/zabbix.conf。一些 PHP 设置已经设置好了。但你还是有必要正确设置 date.timezone。

```
php_value[max_execution_time] = 300
php_value[memory_limit] = 128M
php_value[post_max_size] = 16M
php_value[upload_max_filesize] = 2M
php_value[max_input_time] = 300
php_value[max_input_vars] = 10000
```

```
; php_value[date.timezone] = Europe/Riga
```

现在，你可以继续进行前端安装步骤 以访问新安装的 Zabbix。

请注意 Zabbix proxy 没有前端，只与 Zabbix server 通信。

启动 Zabbix server/agent 进程

启动 Zabbix server 和 agent 进程，并让其随系统启动而启动。

使用 Apache 网络服务器：

```
# systemctl restart zabbix-server zabbix-agent apache2 php-fpm
# systemctl enable zabbix-server zabbix-agent apache2 php-fpm
```

在 Nginx 网络服务器中将'apache2' 替换为'nginx'。

安装 debuginfo 软件包

为了启用 debuginfo 软件源，编辑 */etc/zypp/repos.d/zabbix.repo* 文件。请为 zabbix-debuginfo 软件源将 `enabled=0` 改为 `enabled=1`。

```
[zabbix-debuginfo]
name=Zabbix Official Repository debuginfo
type=rpm-md
baseurl=http://repo.zabbix.com/zabbix/4.5/sles/15/x86_64/debuginfo/
gpgcheck=1
gpgkey=http://repo.zabbix.com/zabbix/4.5/sles/15/x86_64/debuginfo/repodata/repomd.xml.key
enabled=0
update=1
```

然后就可安装 zabbix-***<component>***-debuginfo 包了。

4 从 **MSI** 安装 **Windows** 代理

概述

可以从 Windows MSI 安装包（32 位或 64 位）安装 Zabbix agent。download.

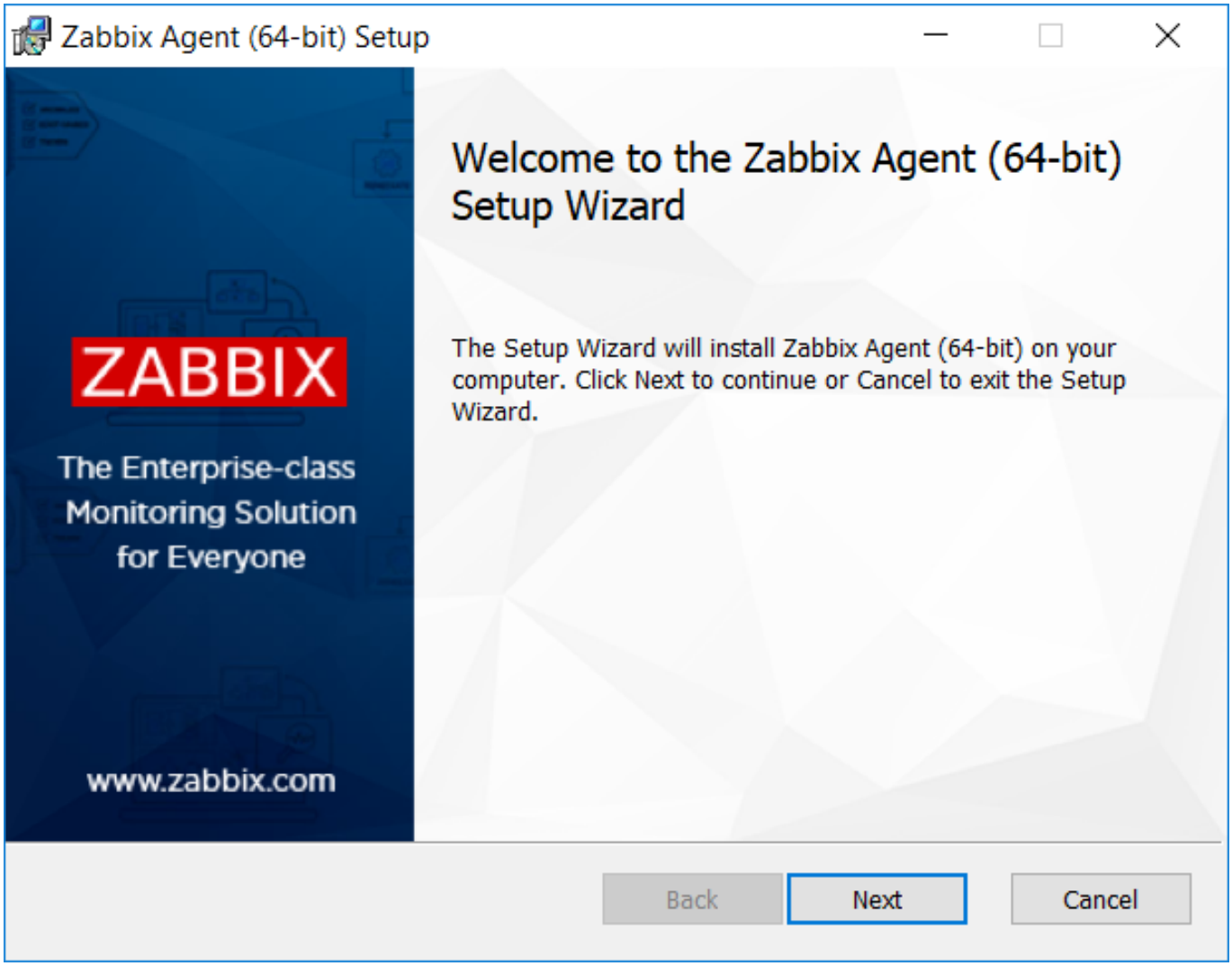
32 位包不能安装在 64 位 Windows 中。

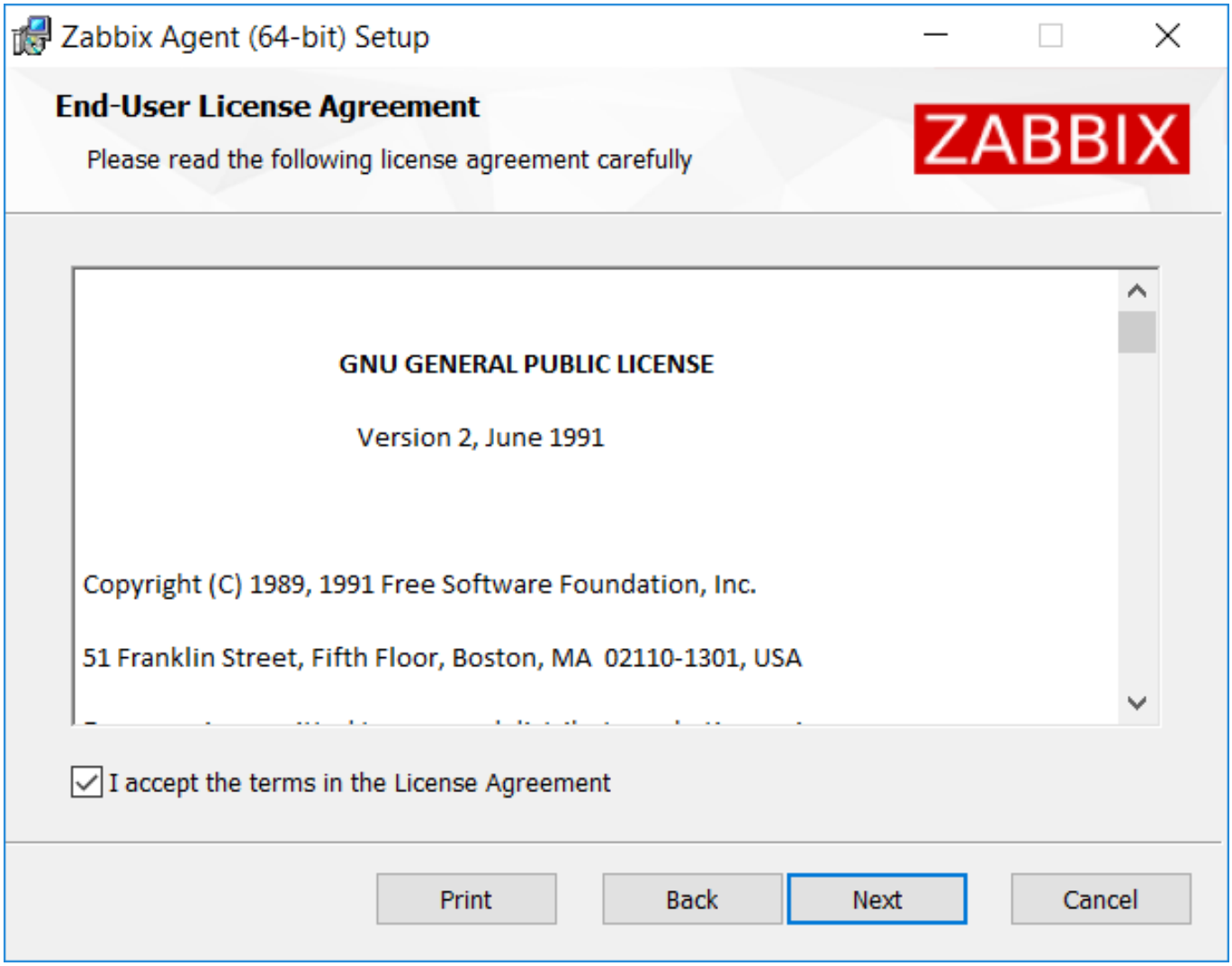
所有软件包都支持 TLS，然而，配置 TLS 是可选的。

支持 UI 和命令行的安装。

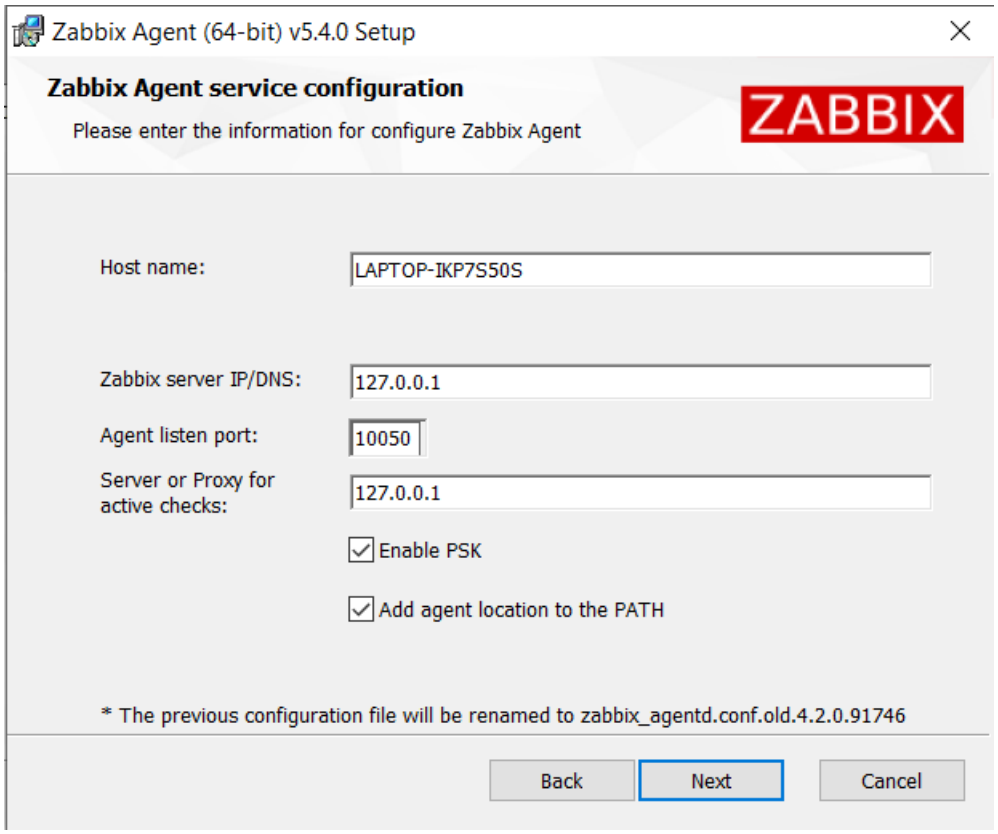
安装步骤

请双击已下载的 MSI 文件进行安装。


Zabbix Agent (64-bit) Setup
ZABBIX
The Enterprise-class Monitoring Solution for Everyone
www.zabbix.com
Welcome to the Zabbix Agent (64-bit) Setup Wizard
The Setup Wizard will install Zabbix Agent (64-bit) on your computer. Click Next to continue or Cancel to exit the Setup Wizard.
Back
Next
Cancel

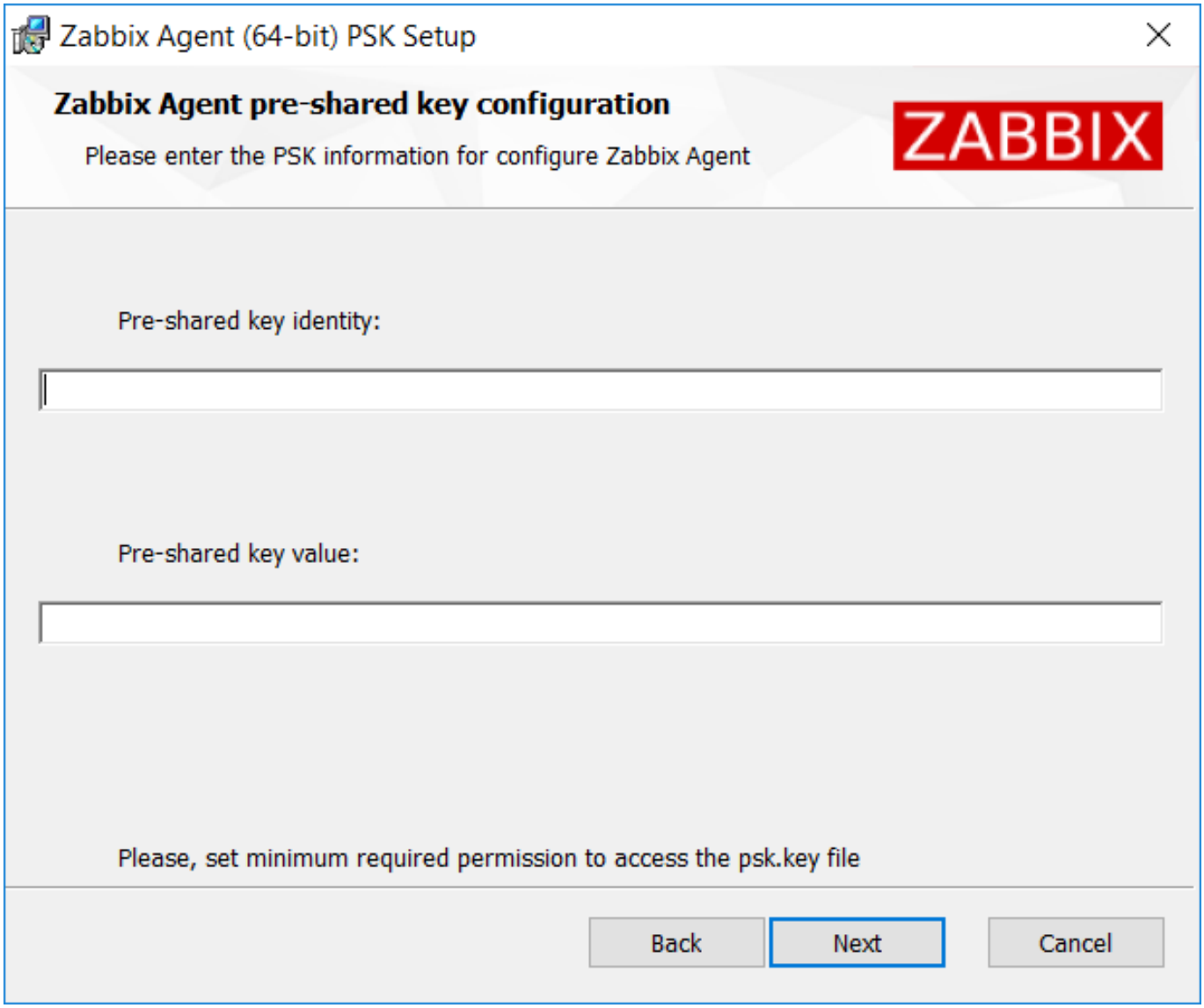


接受许可证已进入下一步。

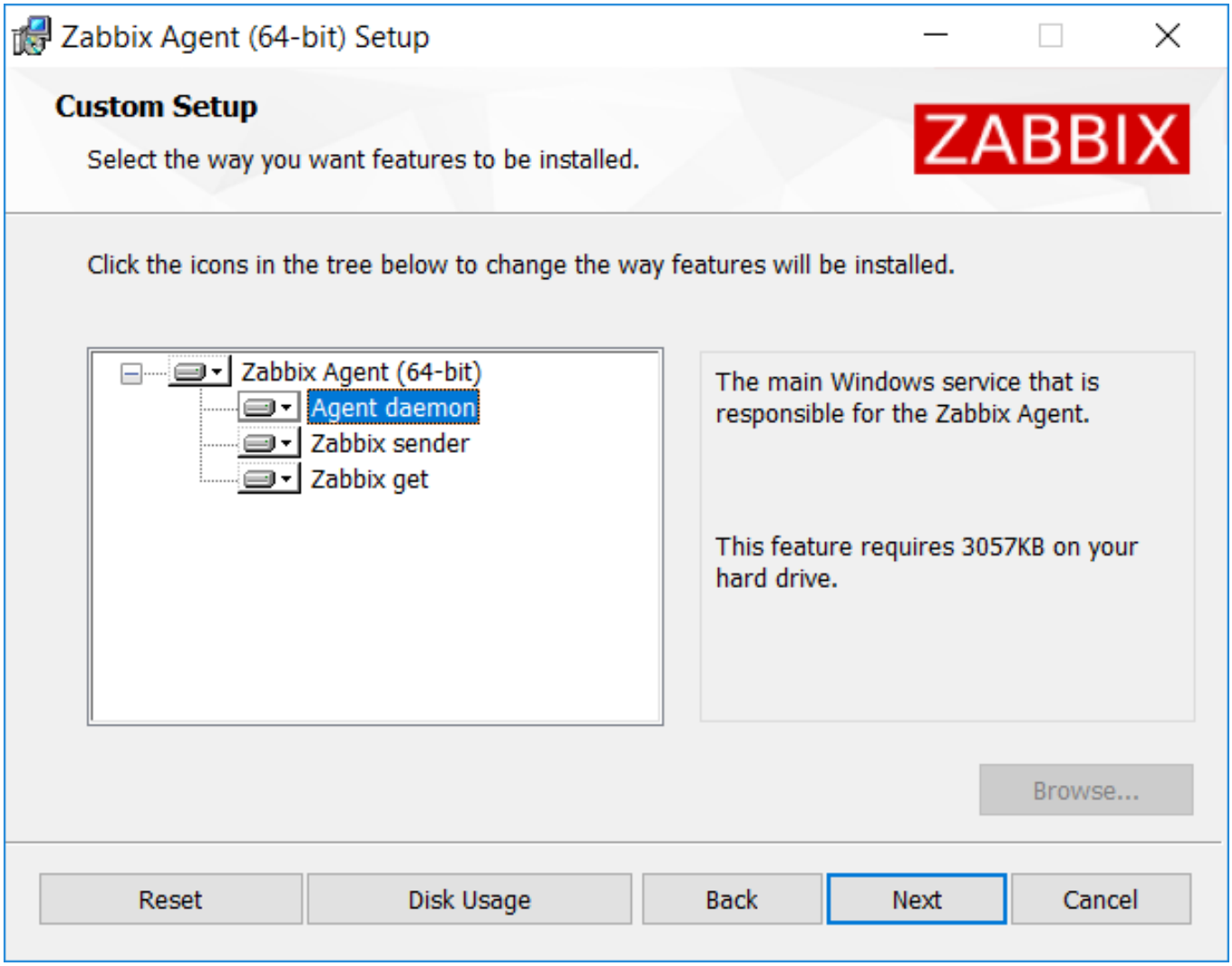


具体参数。

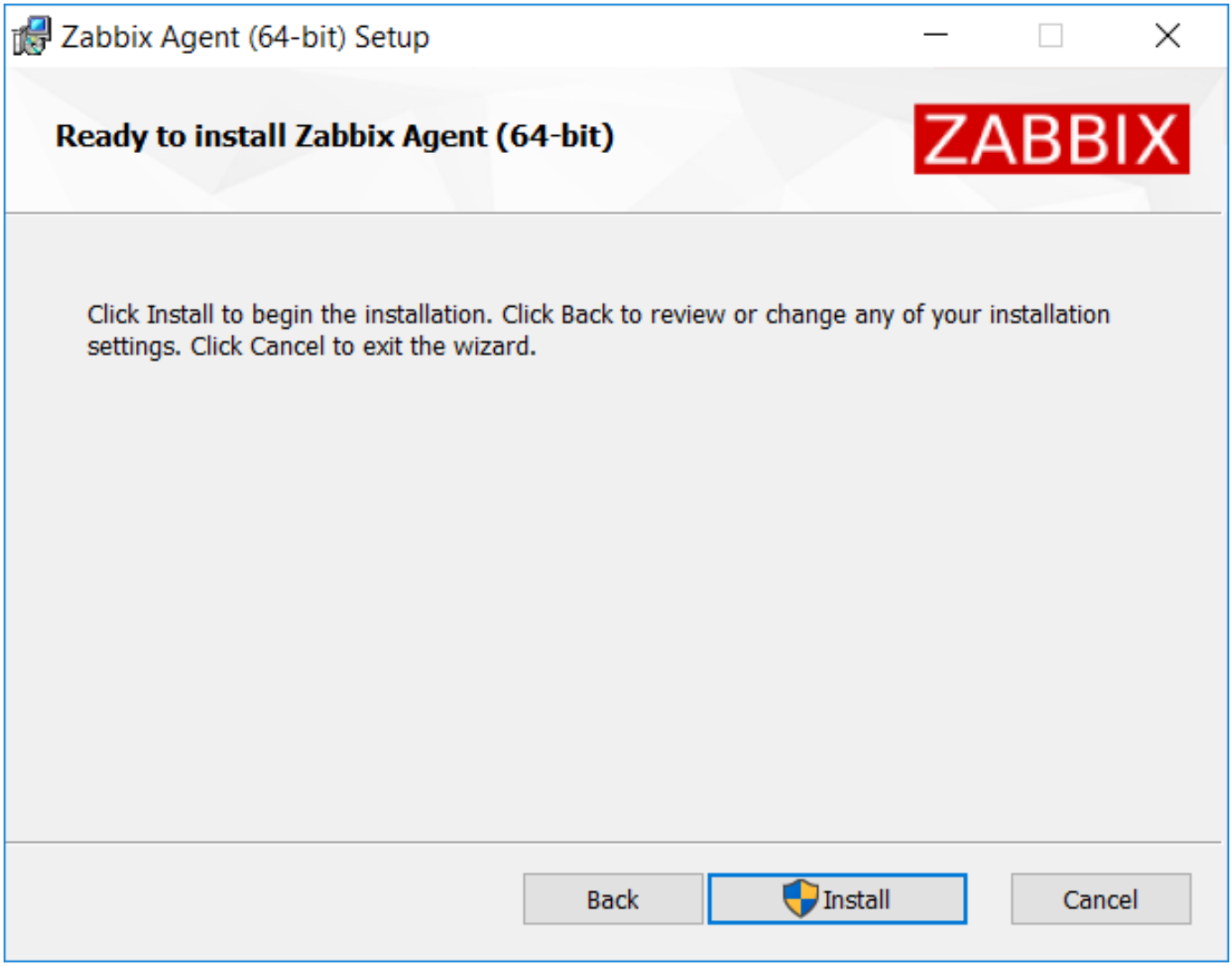
参数	描述
主机名	指定主机名。
Zabbix server IP/DNS	指定 Zabbix server 的 IP/DNS。
Agent 监听端口	指定代理监听端口 (默认为 10050)。
主动检查 *Server* 或 *Proxy*	为激活 agent 主动检查指定 Zabbix server/proxy 的 IP/DNS。
启用 *PSK*	选中校验框，通过预共享密钥激活 TLS 支持。
将 *agent* 位置添加到 *PATH*	将 agent 位置添加至 PATH 变量。



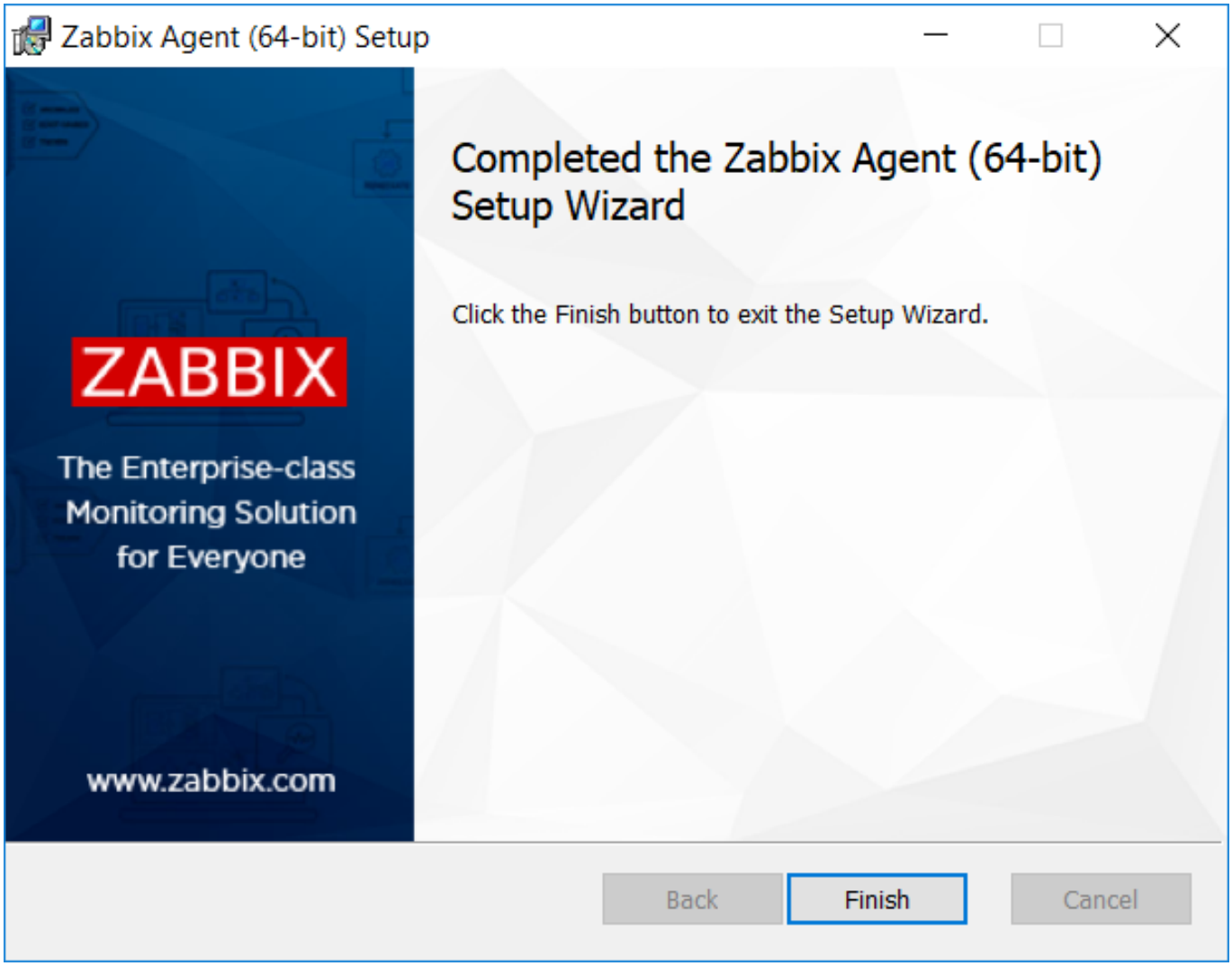
输入预共享密钥的标识和对应值。此步骤仅在上一步中选中 *Enable PSK* 之后才有用。



选择要安装的 Zabbix 组件- Zabbix agent daemon, Zabbix sender, Zabbix get.



Zabbix 组件和配置文件将安装在程序文件 *Zabbix Agent* 文件夹中。zabbix_agentd.exe 在 Windows 服务中将被设置为自动启动。



命令行安装

支持的参数

创建的 MSI 支持以下参数集：

序号	参数	说明
1	日志类型	
2	日志文件	
3	服务器	
4	监听	
5	服务器活动	
6	主机名	
7	超时	
8	TLS 连接	
9	TLS 接受	
10	TLSPSK 身份	
11	TLSPSK 文件	
12	TLSPSK 值	
13	TLSCA 文件	
14	TLSCRL 文件	
15	TLS 服务器证书颁发者	
16	TLS 服务器证书对象	
17	TLS 证书文件	
18	TLSKEY 文件	
19	LISTENIP	
20	主机界面	
21	主机元数据	
22	主机元数据项	

序号	参数	说明
23	端口状态	仅限 Zabbix agent2。
24	启用持久缓冲区	仅限 Zabbix agent2。
25	持续缓冲区	仅限 Zabbix agent2。
26	持续缓冲文件	仅限 Zabbix agent2。
27	安装文件夹	
28	启用路径	
29	跳过	SKIP=fw - 不安装防火墙规则
30	包含	由；分隔的序列
31	允许拒绝密钥	”AllowKey” 和”DenyKey” 参数 的序列，用；分隔，使用 \\; 转义分隔符。

安装，可运行：

```
SET INSTALLFOLDER=C:\Program Files\za

msiexec /l*v log.txt /i zabbix_agent-6.0.0-x86.msi /qn^
 LOGTYPE=file^
 LOGFILE="%INSTALLFOLDER%\za.log"^
 SERVER=192.168.6.76^
 LISTENPORT=12345^
 SERVERACTIVE=::1^
 HOSTNAME=myHost^
 TLSCONNECT=psk^
 TLSACCEPT=psk^
 TLSPSKIDENTITY=MyPSKID^
 TLSPSKFILE="%INSTALLFOLDER%\mykey.psk"^
 TLSCAFILE="c:\temp\f.txt1"^
 TLSCRLFILE="c:\temp\f.txt2"^
 TLSSERVERCERTISSUER="My CA"^
 TLSSERVERCERTSUBJECT="My Cert"^
 TLSCERTFILE="c:\temp\f.txt5"^
 TLSKEYFILE="c:\temp\f.txt6"^
 ENABLEPATH=1^
 INSTALLFOLDER="%INSTALLFOLDER%"^
 SKIP=fw^
 ALLOWDENYKEY="DenyKey=vfs.file.contents[/etc/passwd]"
```

or

```
msiexec /l*v log.txt /i zabbix_agent-6.0.0-x86.msi /qn^
 SERVER=192.168.6.76^
 TLSCONNECT=psk^
 TLSACCEPT=psk^
 TLSPSKIDENTITY=MyPSKID^
 TLSPSKVALUE=1f87b595725ac58dd977beef14b97461a7c1045b9a1c963065002c5473194952
```

5 从 **PKG** 安装 **Mac OS Agent**

概述

Zabbix Mac OS Agent 可以从 PKG 安装程序包安装。点此 下载。加密版本和不加密版本均可以下载。

安装 agent

可以使用图形用户界面安装代理，也可以从命令行。例如:

```
sudo installer -pkg zabbix_agent-5.4.0-macos-amd64-openssl.pkg -target /
```

确保在命令中使用正确的 Zabbix 包版本。它必须与下载的程序包的名称匹配。

运行 Agent

Agent 将在安装或重新启动后自动启动。

如有需要，你可以编辑位于/usr/local/etc/zabbix/zabbix_agentd.conf 的配置文件。

手动启动 agent，你可以执行:

```
sudo launchctl start com.zabbix.zabbix_agentd
```

手动停止 agent，你可以执行:

```
sudo launchctl stop com.zabbix.zabbix_agentd
```

在升级过程中，现有配置文件不会被覆盖。如有需要，它将被替换为一个新的 zabbix_agentd.conf.NEW 文件，用于查看和更新现有配置文件。请记得在手动更改配置文件后重新启动 agent 。

故障排除和删除代理

本节列出了一些有用的命令，可用于故障排除和删除 Zabbix 代理安装。

查看 Zabbix 代理是否正在运行：

```
ps aux| grep zabbix_agentd
```

查看是否已经从包中安装了 Zabbix 代理：

```
$ pkgutil --pkgs | grep zabbix
com.zabbix.pkg.ZabbixAgent
```

查看从安装程序包中安装的文件（请注意此视图中不显示初始的 / ）：

```
$ pkgutil --only-files --files com.zabbix.pkg.ZabbixAgent
库/LaunchDaemons/com.zabbix.zabbix_agentd.plist
usr/local/bin/zabbix_get
usr/local/bin/zabbix_sender
usr/local/etc/zabbix/zabbix_agentd/userparameter_examples.conf.NEW
usr/local/etc/zabbix/zabbix_agentd/userparameter_mysql.conf.NEW
usr/local/etc/zabbix/zabbix_agentd.conf.NEW
usr/local/sbin/zabbix_agentd
```

停止 Zabbix 代理，如果它是用 launchctl 启动的：

```
sudo launchctl unload /Library/LaunchDaemons/com.zabbix.zabbix_agentd.plist
```

删除随安装程序包一起安装的文件（包括配置和日志）：

```
sudo rm -f /Library/LaunchDaemons/com.zabbix.zabbix_agentd.plist
sudo rm -f /usr/local/sbin/zabbix_agentd
sudo rm -f /usr/local/bin/zabbix_get
sudo rm -f /usr/local/bin/zabbix_sender
sudo rm -rf /usr/local/etc/zabbix
sudo rm -rf /var/log/zabbix
```

移除 Zabbix agent 安装包：

```
sudo pkgutil --forget com.zabbix.pkg.ZabbixAgent
```

6 不稳定版本

概述

从 Zabbix 6.0.9 开始提供次要 Zabbix 版本（即 Zabbix 6.0.x）候选发布包。

以下说明用于启用不稳定的 Zabbix 发布存储库（默认情况下禁用）。

首先，安装或更新到最新的 zabbix-release 包。要在您的系统上启用 rc 包，请执行以下操作：

Red Hat 企业版 Linux

打开 /etc/yum.repos.d/zabbix.repo 文件并为 zabbix-unstable 仓库设置 enabled=1。

·[zabbix-unstable] ·name=Zabbix Official Repository (unstable) - $basearch ·baseurl=https://repo.zabbix.com/zabbix/5.5/rhel/8/$basearch/ ·enabled=1 ·gpgcheck=1 ·gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-ZABBIX-A14FE591

Debian/Ubuntu

打开/etc/apt/sources.list.d/zabbix.list 并取消注释 “Zabbix unstable repository”。

·# Zabbix 不稳定的仓库·deb https://repo.zabbix.com/zabbix/5.5/debian bullseye main ·deb-src https://repo.zabbix.com/zabbix/5.5/debian bullseye main

SUSE

打开 `/etc/zypp/repos.d/zabbix.repo` 文件并为 `zabbix-unstable` 仓库设置 enable=1。

·[zabbix-unstable] ·name=Zabbix Official Repository ·type=rpm-md ·baseurl=https://repo.zabbix.com/zabbix/5.5/sles/15/x86_64/ · gpgcheck=1 ·gpgkey=https://repo.zabbix.com/zabbix/5.5/sles/15/x86_64/repodata/repomd.xml.key ·enabled=1 ·update=1

5 从容器中安装

Docker Zabbix 为每个 Zabbix 组件提供 Docker image 作为可移植和自给自足的容器，以加快部署和更新过程。

Zabbix 组件支持 MySQL 和 PostgreSQL 数据库、Apache2 和 Nginx Web 服务器。这些 image 被分成多个不同的 image。

Docker 基础 images

Zabbix 组件在 Ubuntu、Alpine Linux 和 CentOS 基础 image 上提供:

Image	Version
alpine	3.12
ubuntu	20.04 (focal)
centos	8

所有 Zabbix image 已设置为在基础 image 更新时，重新生成最新的 Zabbix image。

Docker file 源

任何人都可以使用位于 github.com 上的 官方仓库 来追踪 Docker file 的变化。您可以 fork 项目，或根据官方 Docker file 制作自己的 image。

结构

所有 Zabbix 组件在以下 Docker 存储库中可用：

- Zabbix agent - zabbix/zabbix-agent
- Zabbix server
  - 支持 MySQL 数据库的 Zabbix server - zabbix/zabbix-server-mysql
  - 支持 PostgreSQL 数据库的 Zabbix server - zabbix/zabbix-server-pgsql
- Zabbix web 界面
  - 基于 Apache2 Web 服务器的 Zabbix web 界面，支持 MySQL 数据库 - zabbix/zabbix-web-apache-mysql
  - 基于 Apache2 Web 服务器的 Zabbix web 界面，支持 PostgreSQL 数据库 - zabbix/zabbix-web-apache-pgsql
  - 基于 Nginx Web 服务器的 Zabbix web 界面，支持 MySQL 数据库 - zabbix/zabbix-web-nginx-mysql
  - 基于 Nginx Web 服务器的 Zabbix web 界面，支持 PostgreSQL 数据库 - zabbix/zabbix-web-nginx-pgsql
- Zabbix proxy
  - Zabbix proxy ，支持 SQLite3 数据库 - zabbix/zabbix-proxy-sqlite3
  - Zabbix proxy，支持 MySQL 数据库 - zabbix/zabbix-proxy-mysql
- Zabbix Java 网关 - zabbix/zabbix-java-gateway

此外，还有 SNMP trap 支持。它仅作为基于 Ubuntu Trusty 的附加存储库 (zabbix/zabbix-snmptraps) 提供。它可以与 Zabbix server 和 Zabbix proxy 链接。

版本

Zabbix 组件的每个镜像仓库都包含了下列标签：

- `latest` - 基于 Alpine Linux 镜像的最新稳定版的 Zabbix 组件
- `alpine-latest` - 基于 Alpine Linux 镜像的最新稳定版的 Zabbix 组件
- `ubuntu-latest` - 基于 Ubuntu 镜像的最新稳定版的 Zabbix 组件
- `alpine-5.4-latest` - 基于 Alpine Linux 镜像的最新次要版本的 Zabbix 5.4 组件
- `ubuntu-5.4-latest` - 基于 Unbuntu 镜像的最新次要版本的 Zabbix 5.4 组件
- `alpine-5.4.*` - 基于 Alpine Linux 镜像的不同次要版本的 Zabbix 5.4 组件，其中 * 代表 Zabbix 组件的次要版本
- `ubuntu-5.4.*` - 基于 Ubuntu 镜像的不同次要版本的 Zabbix 5.4 组件，其中 * 代表 Zabbix 组件的次要版本

Initial configuration

After downloading the images, start the containers by executing `docker run` command followed by additional arguments to specify required environment variables and/or mount points. Some configuration examples are provided below.

Attention:
Zabbix must not be run as PID1/as an init process in containers.

环境变量

所有 Zabbix 组件 image 都提供环境变量来控制配置。这些环境变量在每个组件 image 仓库中列出。这些环境变量是 Zabbix 配置文件中的选项，但具有不同的命名方法。例如，ZBX_LOGSLOWQUERIES 等于来自 Zabbix server 和 Zabbix proxy 配置文件的 LogSlowQueries。

Attention:
一些配置选项是不允许更改的。例如，`PIDFile` 和 `LogType`。

其中，一些组件有特定的环境变量，而这些环境变量在官方 Zabbix 配置文件并不存在：

变量	组件	描述
DB_SERVER_HOST	Server Proxy Web 界面	这个变量指的是 MySQL 或 PostgreSQL 的 IP 或 DNS。 默认情况下，这个值根据 MySQL 和 PostgreSQL，分别为 mysql-server 或 postgres-server
DB_SERVER_PORT	Server Proxy Web 界面	这个变量指的是 MySQL 或 PostgreSQL 的端口。 默认情况下，这个值根据 MySQL 和 PostgreSQL，分别为’3306’ 或’5432’ 。
MYSQL_USER	Server Proxy Web 界面	MySQL 数据库用户。 默认情况下，这个值为’zabbix’。
MYSQL_PASSWORD	Server Proxy Web 界面	MySQL 数据库密码。 默认情况下，这个值为’zabbix’。
MYSQL_DATABASE	Server Proxy Web 界面	Zabbix 数据库库名。 默认情况下，这个值根据 Zabbix server 和 Zabbix proxy，分别为’zabbix’ 和’zabbix_proxy’ 。
POSTGRES_USER	Server Web 界面	PostgreSQL 数据库用户。 默认情况下，这个值为’zabbix’。
POSTGRES_PASSWORD	Server Web 界面	PostgreSQL 数据库密码。 默认情况下，这个值为’zabbix’。
POSTGRES_DB	Server Web 界面	Zabbix 数据库库名。 默认情况下，这个值根据 Zabbix server 和 Zabbix proxy，分别为’zabbix’ 和’zabbix_proxy’ 。
PHP_TZ	Web 界面	PHP 时区格式。支持时区的完整列表参照 php.net。 默认情况下，这个值为’Europe/Riga’ 。
ZBX_SERVER_NAME	Web 界面	Web 界面右上角显示的安装名称。 默认情况下，这个值为’Zabbix Docker’ 。
ZBX_JAVAGATEWAY_ENABLE	Server Proxy	是否启用 Zabbix Java 网关以采集与 Java 相关的检查数据。 默认情况下，这个值为”false” 。
ZBX_ENABLE_SNMP_TRAPS	Server Proxy	是否启用 SNMP trap 功能。这需要存在 zabbix-snmptraps 容器实例，并共享 /var/lib/zabbix/snmptraps volume 到 Zabbix server 或 proxy。

Volumes

Image 中允许使用一些挂载点。根据 Zabbix 组件类型，这些挂载点各不相同：

Volume	描述
Zabbix agent	

/etc/zabbix/zabbix_agentd.d	这个 volume 允许包含 **.conf* 文件并使用 `UserParameter` 扩展 Zabbix agent。
/var/lib/zabbix/modules	这个 volume 允许加载其它 module 并使用LoadModule 功能扩展 Zabbix agent。
/var/lib/zabbix/enc	这个 volume 用于存放 TLS 相关的文件。这些文件名使用 `ZBX_TLSCAFILE`, `ZBX_TLSCRLFILE`, `ZBX_TLSKEY_FILE` ，`ZBX_TLSPSKFILE` 等环境变量指定。
Zabbix server	
/usr/lib/zabbix/alertscripts	这个 volume 用于自定义告警脚本。即zabbix_server.conf 中的 `AlertScriptsPath` 参数。
/usr/lib/zabbix/externalscripts	这个 volume 用于外部检查。即zabbix_server.conf 中的 `ExternalScripts` 参数。
/var/lib/zabbix/modules	这个 volume 允许通过LoadModule 功能加载额外的模块以扩展 Zabbix server。
/var/lib/zabbix/enc	这个 volume 用于存放 TLS 相关的文件。这些文件名使用 `ZBX_TLSCAFILE`, `ZBX_TLSCRLFILE`, `ZBX_TLSKEY_FILE` ，`ZBX_TLSPSKFILE` 等环境变量指定。
/var/lib/zabbix/ssl/certs	这个 volume 用于存放客户端认证的 SSL 客户端认证文件。即 [zabbix_server.conf] 中的 `SSLCertLocation` 参数。
/var/lib/zabbix/ssl/keys	这个 volume 用于存放客户端认证的 SSL 私钥文件。即zabbix_server.conf 中的 `SSLKeyLocation` 参数。
/var/lib/zabbix/ssl/ssl_ca	这个 volume 用于存放 SSL 服务器证书认证的证书颁发机构 (CA) 文件。即zabbix_server.conf 中的 `SSLCALocation` 参数。
/var/lib/zabbix/snmptraps	这个 volume 用于存放 snmptraps.log 文件。它可由 zabbix-snmptraps 容器共享，并在创建 Zabbix server 新实例时使用 Docker 的 `--volumes-from` 选项继承。可以通过共享 volume ，并将 ZBX_ENABLE_SNMP_TRAPS 环境变量切换为'true' 以启用 SNMP trap 处理功能。
/var/lib/zabbix/mibs	这个 volume 允许添加新的 MIB 文件。它不支持子目录，所有的 MIB 文件必须位于 `/var/lib/zabbix/mibs` 下。
Zabbix proxy	
/usr/lib/zabbix/externalscripts	这个 volume 用于外部检查。即zabbix_proxy.conf 中的 `ExternalScripts` 参数。
/var/lib/zabbix/modules	这个 volume 允许通过LoadModule 功能加载额外的模块以扩展 Zabbix server。
/var/lib/zabbix/enc	这个 volume 用于存放 TLS 相关的文件。这些文件名使用 `ZBX_TLSCAFILE`, `ZBX_TLSCRLFILE`, `ZBX_TLSKEY_FILE` ，`ZBX_TLSPSKFILE` 等环境变量指定。
/var/lib/zabbix/ssl/certs	这个 volume 用于存放客户端认证的 SSL 客户端认证文件。即zabbix_proxy.conf 中的 `SSLCertLocation` 参数。
/var/lib/zabbix/ssl/keys	这个 volume 用于存放客户端认证的 SSL 私钥文件。即zabbix_proxy.conf 中的 `SSLKeyLocation` 参数。
/var/lib/zabbix/ssl/ssl_ca	这个 volume 用于存放 SSL 服务器证书认证的证书颁发机构 (CA) 文件。即zabbix_proxy.conf 中的 `SSLCALocation` 参数。
/var/lib/zabbix/snmptraps	这个 volume 用于存放 snmptraps.log 文件。它可由 zabbix-snmptraps 容器共享，并在创建 Zabbix server 新实例时使用 Docker 的 `--volumes-from` 选项继承。可以通过共享 volume ，并将 ZBX_ENABLE_SNMP_TRAPS 环境变量切换为'true' 以启用 SNMP trap 处理功能。
/var/lib/zabbix/mibs	这个 volume 允许添加新的 MIB 文件。它不支持子目录，所有的 MIB 文件必须位于 `/var/lib/zabbix/mibs` 下。
基于 **Apache2 web** 服务器的 **Zabbix web** 界面	
/etc/ssl/apache2	这个 volume 允许为 Zabbix Web 界面启用 HTTPS。这个 volume 必须包含为 Apache2 SSL 连接准备的 `ssl.crt` 和 `ssl.key` 两个文件。
基于 **Nginx web** 服务器的 **Zabbix web** 界面	
/etc/ssl/nginx	这个 volume 允许为 Zabbix Web 接口启用 HTTPS。这个 volume 必须包含为 Nginx SSL 连接装备的 `ssl.crt` 和 `ssl.key` 两个文件。
Zabbix snmptraps	
/var/lib/zabbix/snmptraps	这个 volume 包含了已接收到的 SNMP traps 命名的 `snmptraps.log` 日志文件。

/var/lib/zabbix/mibs	这个 volume 允许添加新的 MIB 文件。它不支持子目录，所有的 MIB 文件必须位于 `/var/lib/zabbix/mibs` 下。

关于更多的信息请在 Docker Hub 的 Zabbix 官方仓库查看。

使用示例

示例 **1**

该示例示范了如何运行 MySQL 数据库支持的 Zabbix Server 、基于 Nginx Web 服务器的 Zabbix Web 界面和 Zabbix Java 网关。

1. 创建专用于 Zabbix 组件容器的网络：

```
docker network create --subnet 172.20.0.0/16 --ip-range 172.20.240.0/20 zabbix-net
```

2. 启动空的 MySQL 服务器实例：

```
docker run --name mysql-server -t \
      -e MYSQL_DATABASE="zabbix" \
      -e MYSQL_USER="zabbix" \
      -e MYSQL_PASSWORD="zabbix_pwd" \
      -e MYSQL_ROOT_PASSWORD="root_pwd" \
      --network=zabbix-net \
      --restart unless-stopped \
      -d mysql:8.0 \
      --character-set-server=utf8 --collation-server=utf8_bin \
      --default-authentication-plugin=mysql_native_password
```

3. 启动 Zabbix Java 网关实例：

```
 docker run --name zabbix-java-gateway -t \
      --network=zabbix-net \
      --restart unless-stopped \
      -d zabbix/zabbix-java-gateway:alpine-5.4-latest
```

4. 启动 Zabbix server 实例，并将其关联到已创建的 MySQL server 实例：

```
docker run --name zabbix-server-mysql -t \
      -e DB_SERVER_HOST="mysql-server" \
      -e MYSQL_DATABASE="zabbix" \
      -e MYSQL_USER="zabbix" \
      -e MYSQL_PASSWORD="zabbix_pwd" \
      -e MYSQL_ROOT_PASSWORD="root_pwd" \
      -e ZBX_JAVAGATEWAY="zabbix-java-gateway" \
      --network=zabbix-net \
      -p 10051:10051 \
      --restart unless-stopped \
      -d zabbix/zabbix-server-mysql:alpine-5.4-latest
```

Note:
Zabbix server 实例将 10051/TCP 端口（Zabbix trapper）暴露给主机。

5. 启动 Zabbix Web 界面，并将其关联到已创建的 MySQL server 和 Zabbix server 实例：

```
docker run --name zabbix-web-nginx-mysql -t \
      -e ZBX_SERVER_HOST="zabbix-server-mysql" \
      -e DB_SERVER_HOST="mysql-server" \
      -e MYSQL_DATABASE="zabbix" \
      -e MYSQL_USER="zabbix" \
      -e MYSQL_PASSWORD="zabbix_pwd" \
      -e MYSQL_ROOT_PASSWORD="root_pwd" \
      --network=zabbix-net \
      -p 80:8080 \
      --restart unless-stopped \
      -d zabbix/zabbix-web-nginx-mysql:alpine-5.4-latest
```

Note:
Zabbix web 界面实例将 80/TCP 端口（HTTP）暴露给主机。

示例 **2**

该示例示范了如何运行 PostgreSQL 数据库支持的 Zabbix server、基于 Nginx Web 服务器的 Zabbix Web 界面和 SNMP trap 功能。

1. 创建专用于 Zabbix 组件容器的网络：

```
# docker network create --subnet 172.20.0.0/16 --ip-range 172.20.240.0/20 zabbix-net
```

2. 启动空的 PostgreSQL server 实例：

```
docker run --name postgres-server -t \
      -e POSTGRES_USER="zabbix" \
      -e POSTGRES_PASSWORD="zabbix_pwd" \
      -e POSTGRES_DB="zabbix" \
      --network=zabbix-net \
      --restart unless-stopped \
      -d postgres:latest
```

3. 启动 Zabbix snmptraps 实例：

```
docker run --name zabbix-snmptraps -t \
      -v /zbx_instance/snmptraps:/var/lib/zabbix/snmptraps:rw \
      -v /var/lib/zabbix/mibs:/usr/share/snmp/mibs:ro \
      --network=zabbix-net \
      -p 162:1162/udp \
      --restart unless-stopped \
      -d zabbix/zabbix-snmptraps:alpine-5.4-latest
```

Note:
Zabbix snmptrap 实例将 162/UDP 端口（SNMP traps）暴露给主机。

4. 启动 Zabbix server 实例，并将其关联到已创建的 PostgreSQL server 实例：

```
docker run --name zabbix-server-pgsql -t \
      -e DB_SERVER_HOST="postgres-server" \
      -e POSTGRES_USER="zabbix" \
      -e POSTGRES_PASSWORD="zabbix_pwd" \
      -e POSTGRES_DB="zabbix" \
      -e ZBX_ENABLE_SNMP_TRAPS="true" \
      --network=zabbix-net \
      -p 10051:10051 \
      --volumes-from zabbix-snmptraps \
      --restart unless-stopped \
      -d zabbix/zabbix-server-pgsql:alpine-5.4-latest
```

Note:
Zabbix server 实例将 10051/TCP 端口（Zabbix trapper）暴露给主机。

5. 启动 Zabbix Web 界面，并将其关联到已创建的 PostgreSQL server 和 Zabbix server 实例：

```
docker run --name zabbix-web-nginx-pgsql -t \
      -e ZBX_SERVER_HOST="zabbix-server-pgsql" \
      -e DB_SERVER_HOST="postgres-server" \
      -e POSTGRES_USER="zabbix" \
      -e POSTGRES_PASSWORD="zabbix_pwd" \
      -e POSTGRES_DB="zabbix" \
      --network=zabbix-net \
      -p 443:8443 \
      -p 80:8080 \
      -v /etc/ssl/nginx:/etc/ssl/nginx:ro \
      --restart unless-stopped \
      -d zabbix/zabbix-web-nginx-pgsql:alpine-5.4-latest
```

Note:
Zabbix web 界面实例将 443/TCP 端口（HTTPS）暴露给主机。
/etc/ssl/nginx 目录必须包含具有所需名称的证书。

示例 **3**

该示例示范了如何在 Red Hat 8 上使用 podman 运行 MySQL 数据库支持的 Zabbix Server 、基于 Nginx Web 服务器的 Zabbix Web 界面和 Zabbix Java 网关。

1. 创建一个名为 zabbix 的 pod 并暴露端口 (web 界面、Zabbix server trapper)：

```
podman pod create --name zabbix -p 80:8080 -p 10051:10051
```

2. （可选）在 zabbix pod 中启动 Zabbix agent 容器：

```
podman run --name zabbix-agent \
    -eZBX_SERVER_HOST="127.0.0.1,localhost" \
    --restart=always \
    --pod=zabbix \
    -d registry.connect.redhat.com/zabbix/zabbix-agent-50:latest
```

3. 在主机上创建 ./mysql/ 目录并启动 Oracle MySQL server 8.0:

```
podman run --name mysql-server -t \
      -e MYSQL_DATABASE="zabbix" \
      -e MYSQL_USER="zabbix" \
      -e MYSQL_PASSWORD="zabbix_pwd" \
      -e MYSQL_ROOT_PASSWORD="root_pwd" \
      -v ./mysql/:/var/lib/mysql/:Z \
      --restart=always \
      --pod=zabbix \
      -d mysql:8.0 \
      --character-set-server=utf8 --collation-server=utf8_bin \
      --default-authentication-plugin=mysql_native_password
```

3. 启动 Zabbix server 容器：

```
podman run --name zabbix-server-mysql -t \
                  -e DB_SERVER_HOST="127.0.0.1" \
                  -e MYSQL_DATABASE="zabbix" \
                  -e MYSQL_USER="zabbix" \
                  -e MYSQL_PASSWORD="zabbix_pwd" \
                  -e MYSQL_ROOT_PASSWORD="root_pwd" \
                  -e ZBX_JAVAGATEWAY="127.0.0.1" \
                  --restart=always \
                  --pod=zabbix \
                  -d registry.connect.redhat.com/zabbix/zabbix-server-mysql-50
```

4. 启动 Zabbix Java 网关容器：

```
podman run --name zabbix-java-gateway -t \
      --restart=always \
      --pod=zabbix \
      -d registry.connect.redhat.com/zabbix/zabbix-java-gateway-50
```

5. 启动 Zabbix web 界面容器：

```
podman run --name zabbix-web-mysql -t \
                  -e ZBX_SERVER_HOST="127.0.0.1" \
                  -e DB_SERVER_HOST="127.0.0.1" \
                  -e MYSQL_DATABASE="zabbix" \
                  -e MYSQL_USER="zabbix" \
                  -e MYSQL_PASSWORD="zabbix_pwd" \
                  -e MYSQL_ROOT_PASSWORD="root_pwd" \
                  --restart=always \
                  --pod=zabbix \
                  -d registry.connect.redhat.com/zabbix/zabbix-web-mysql-50
```

Note:
zabbix pod 从 zabbix-web-mysql 容器的 8080/TCP 向主机的 80/TCP port (HTTP) 暴露端口。

Docker Compose Zabbix 还提供了用于在 Docker 中定义和运行多容器 Zabbix 组件的 compose 文件。这些 compose 文件可以在 github.com: https://github.com/zabbix/zabbix-docker 上的 Zabbix docker 官方仓库中找到。这些 compose 文件作为示例添加，并支持广泛。例如，Zabbix proxy 支持 MySQL 和 SQLite3。

以下为几个不同版本的 compose 文件：

文件名	描述
docker-compose_v3_alpine_mysql_latest.yaml	该 compose 文件运行基于 Alpine Linux 的 Zabbix 5.4 最新版本的组件，支持 MySQL 数据库。
docker-compose_v3_alpine_mysql_local.yaml	该 compose 文件本地构建和运行基于 Alpine Linux 的 Zabbix 5.4 最新版本的组件，支持 MySQL 数据库。
docker-compose_v3_alpine_pgsql_latest.yaml	该 compose 文件运行基于 Alpine Linux 的 Zabbix 5.4 最新版本的组件，支持 PostgreSQL 数据库。
docker-compose_v3_alpine_pgsql_local.yaml	该 compose 文件本地构建和运行基于 Apline Linux 的 Zabbix 5.4 最新版本的组件，支持 PostgreSQL 数据库。
docker-compose_v3_centos_mysql_latest.yaml	该 compose 文件运行基于 CentOS 8 的 Zabbix 5.4 最新版本的组件，支持 MySQL 数据库。
docker-compose_v3_centos_mysql_local.yaml	该 compose 文件本地构建和运行基于 CentOS 8 的 Zabbix 5.4 最新版本的组件，支持 MySQL 数据库。
docker-compose_v3_centos_pgsql_latest.yaml	该 compose 文件运行基于 CentOS 8 的 Zabbix 5.4 最新版本的组件，支持 PostgreSQL 数据库。
docker-compose_v3_centos_pgsql_local.yaml	该 compose 文件本地构建和运行基于 CentOS 8 的 Zabbix 5.4 最新版本的组件，支持 PostgreSQL 数据库。
docker-compose_v3_ubuntu_mysql_latest.yaml	该 compose 文件运行基于 Ubuntu 20.04 的 Zabbix 5.4 最新版本的组件，支持 MySQL 数据库。
docker-compose_v3_ubuntu_mysql_local.yaml	该 compose 文件本地构建和运行基于 Ubuntu 20.04 的 Zabbix 5.4 最新版本的组件，支持 MySQL 数据库。
docker-compose_v3_ubuntu_pgsql_latest.yaml	该 compose 文件运行基于 Ubuntu 20.04 的 Zabbix 5.4 最新版本的组件，支持 PostgreSQL 数据库。
docker-compose_v3_ubuntu_pgsql_local.yaml	该 compose 文件本地构建和运行基于 Ubuntu 20.04 的 Zabbix 5.4 最新版本的组件，支持 PosegreSQL 数据库。

Attention:
Docker compose 文件支持 Docker Compose 3 版本。

存储

Compose 文件已经配置为支持主机上的存储。当你使用 Compose 文件运行 Zabbix 组件时，Docker Compose 将在其所在文件夹中创建一个 zbx_env 目录，该目录将包含于Volumes 章节所述相同的结构，以用于数据库存储。

此外，volume 下的文件 /etc/localtime 和 /etc/timezone 为只读模式。

环境变量文件

在 github.com 上与存放 compose 文件的同一目录中，您可以在 compose 文件中找到每个组件的默认环境变量文件，这些环境变量文件的命令与 .env_<type of component> 类似。

示例

示例 **1**

```
# git checkout 5.4
# docker-compose -f ./docker-compose_v3_alpine_mysql_latest.yaml up -d
```

这个命令将会为每个 Zabbix 组件下载最新的 Zabbix 5.4 image，并以 detach 模式运行。

Attention:
不要忘记从 github.com 的 Zabbix 官方镜像仓库下载 .env_<type of component> 文件和 compose 文件。

示例 **2**

```
# git checkout 5.4
# docker-compose -f ./docker-compose_v3_ubuntu_mysql_local.yaml up -d
```

这个命令将会下载基于 Ubuntu 20.04 的 image，并在本地构建 Zabbix 5.4 组件，以 detach 模式运行。

Installation with OpenShift

Overview

Zabbix helps you to do a real-time monitoring of millions of metrics collected from tens of thousands of servers, virtual machines and network devices. The Zabbix Operator allows users to easily deploy, manage, and maintain Zabbix deployments on OpenShift. By installing this integration you will be able to deploy Zabbix server/proxies and other components with a single command.

Supported features

Zabbix Operator comes with a few possible installation options:

- **Zabbix server** - a simple Zabbix installation with included Zabbix server, Zabbix web interface and Zabbix Java gateway with MySQL database support. The feature does not provide MySQL service and requires an external MySQL database.
- **Zabbix server (full)** - a Zabbix installation with included Zabbix server, Zabbix web interface, Zabbix Java gateway and MySQL server instance.
- **Zabbix proxy (SQLite3)** - a very simple way to gain power of Zabbix proxy. The feature has SQLite3 support for Zabbix proxies and allows to specify the amount of proxies.
- **Zabbix proxy (MySQL)** - another option of Zabbix proxy. This option supports and delivers a MySQL database. It is possible to use a built-in MySQL database instance or an external one.
- **Zabbix agent** - a Zabbix agent can be deployed on each available node for stability and performance monitoring on remote nodes. It allows to gather metrics with full automation!
- **Zabbix appliance** - a Zabbix appliance is a very simple way to test and check Zabbix features. This option provides all the core components in one solution. It includes Zabbix server, Zabbix Java gateway, Zabbix web interface and MySQL server in deployment. It is very useful for testing Zabbix features!

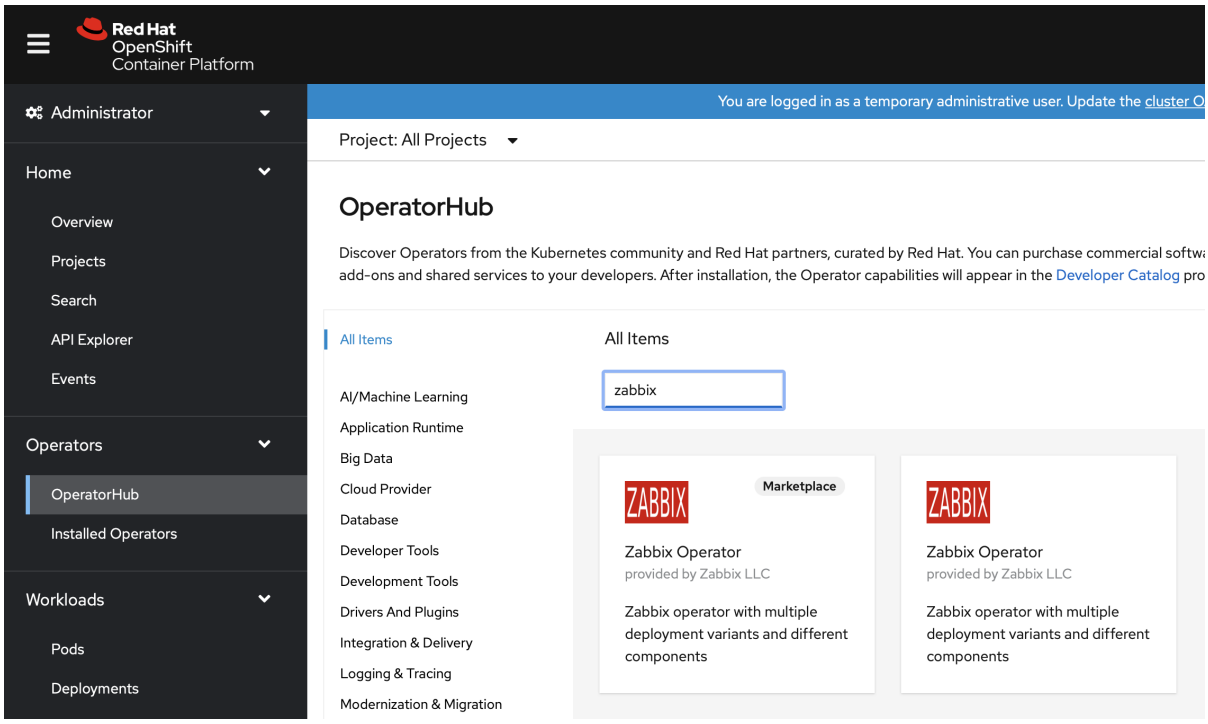
Currently Zabbix Operator is based on the Zabbix 6.0 LTS version and supports OpenShift 4.1, 4.2, 4.3, 4.4, 4.5, and 4.6.

Installing Zabbix Operator

Using RedHat Marketplace

> **Attention:**
> The installation of Zabbix Operator using Red Hat Marketplace requires the OpenShift cluster to be registered in the Marketplace Portal, including the roll out of the PullSecret in your cluster. Failure to do so will result in an image pull authentication failure with the Red Hat registry.

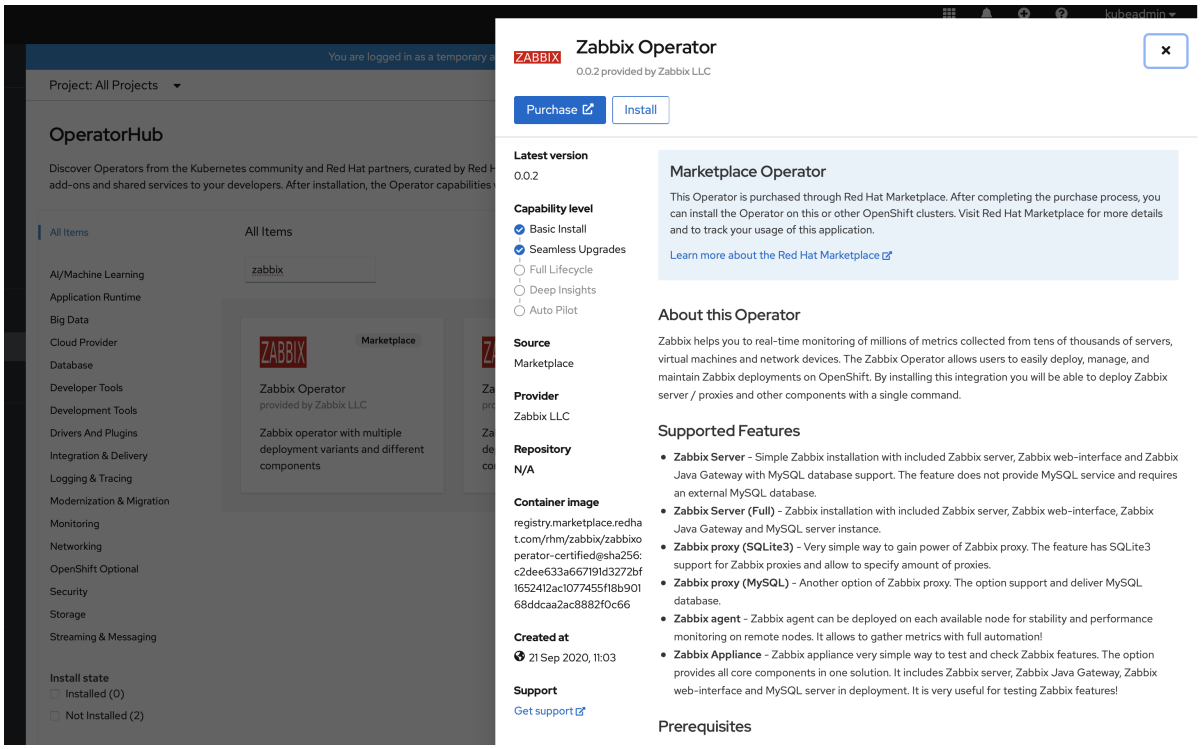
1. Select the OperatorHub from the Operators submenu and search for Zabbix.



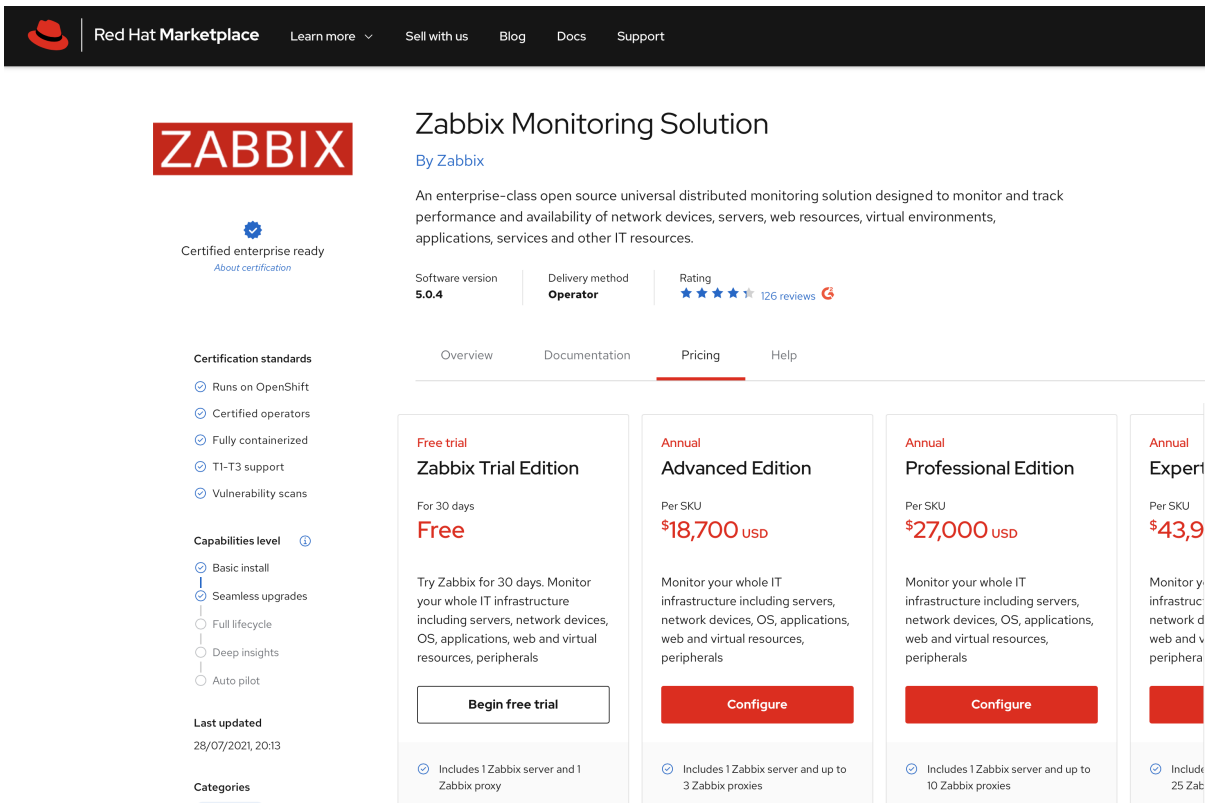
Choose the RedHat Marketplace option.

2. Select ”Zabbix Operator” and click on *Purchase*.

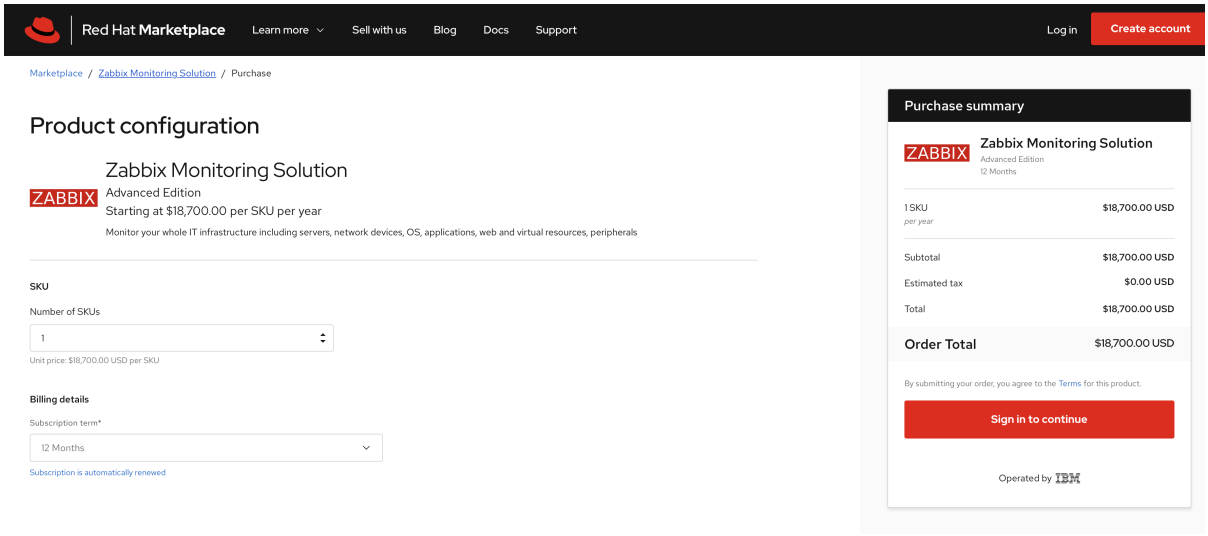
> **Attention:**
> Openshift needs to be registered with the Red Hat Marketplace portal.



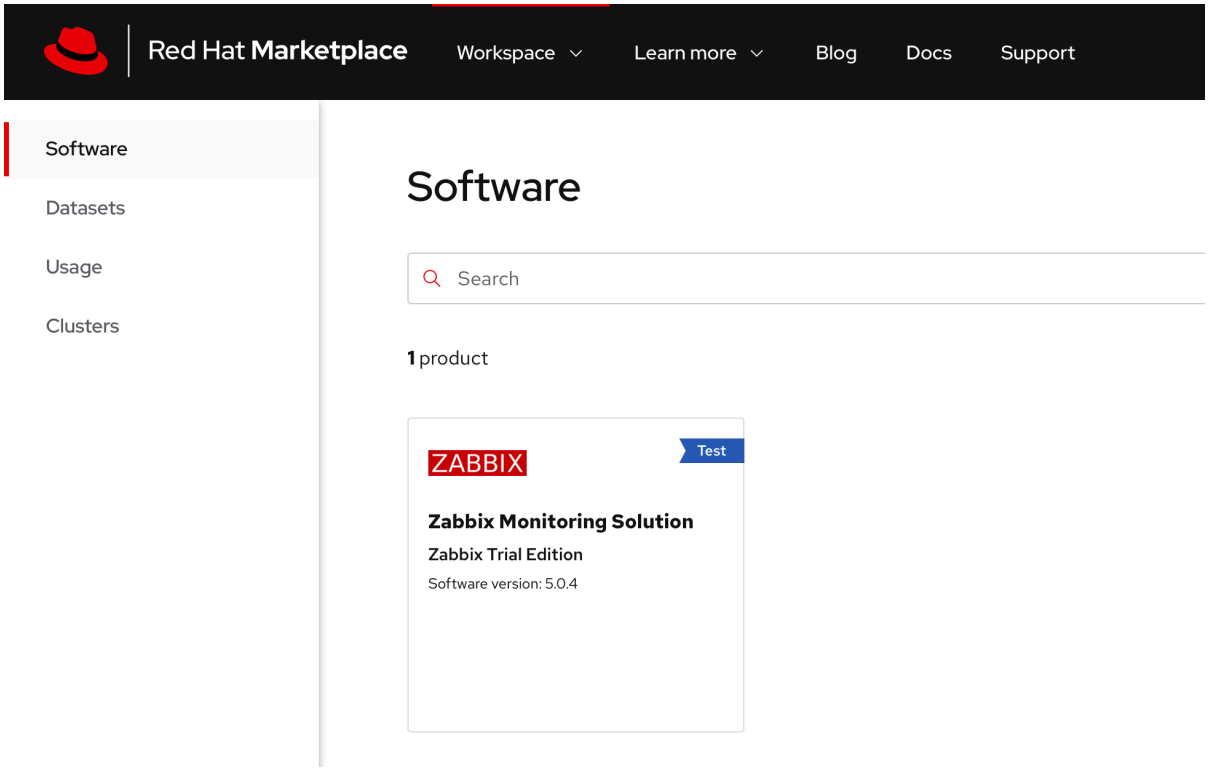
3. Select the most suitable install option.



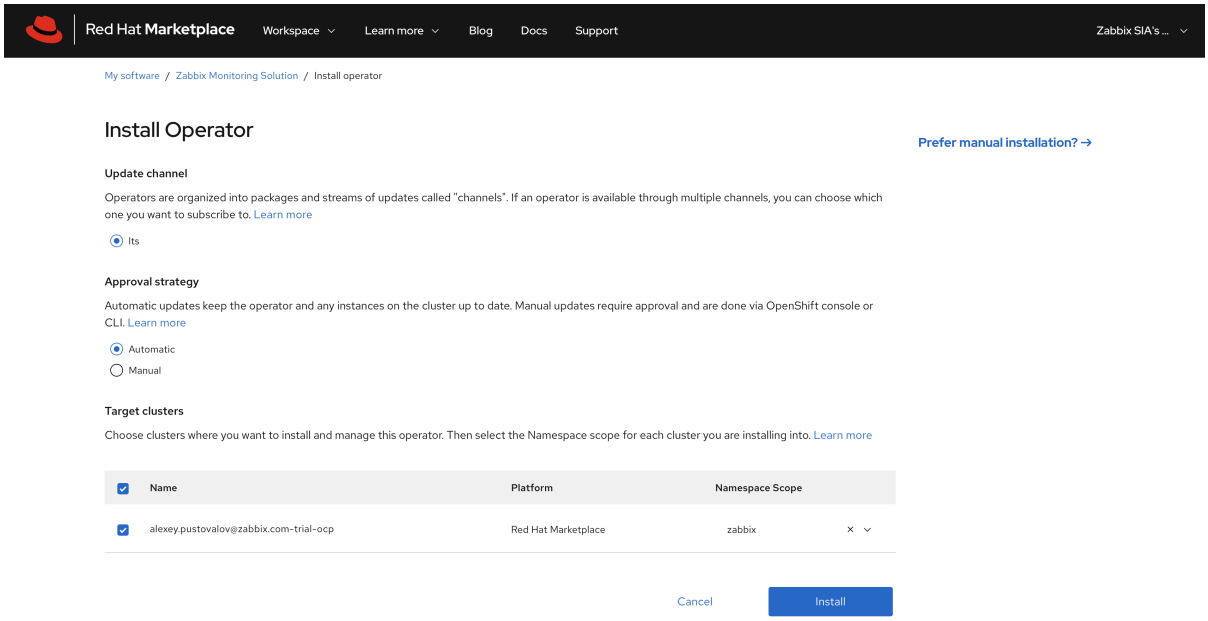
4. Specify the product configuration to fit your needs.



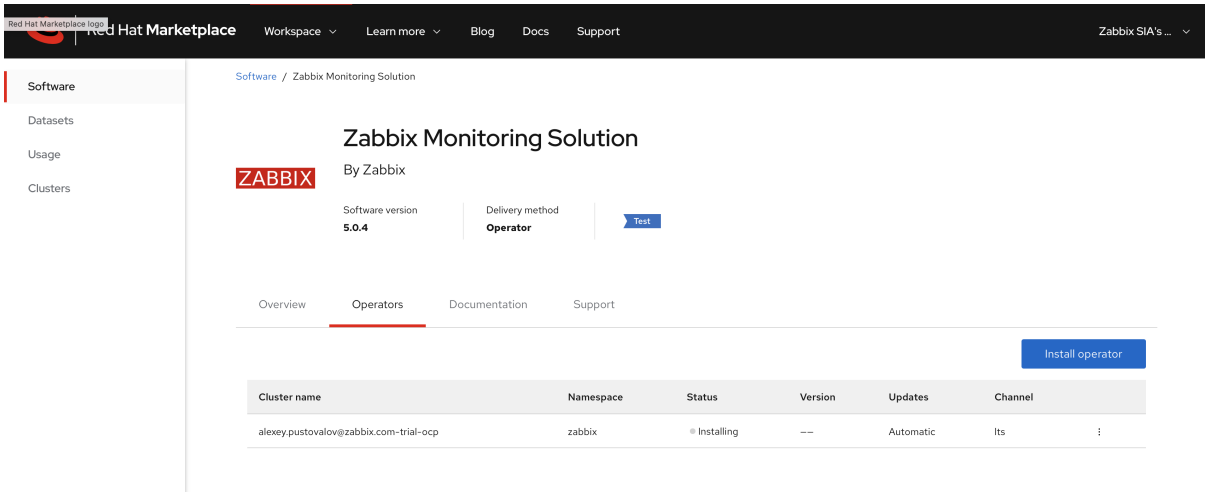
5. Navigate to your software within Red Hat Marketplace and install the Zabbix Operator software as specified in the image.



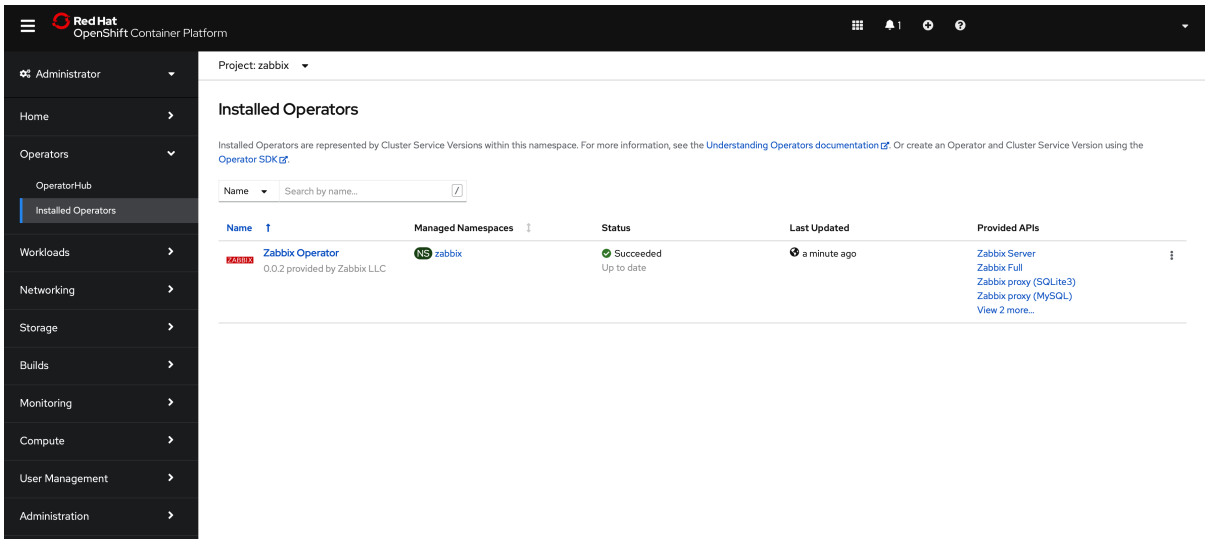
6. Install the Operator. Set the update approval strategy to *Automatic* to ensure that you always have the latest version of Zabbix components installed.



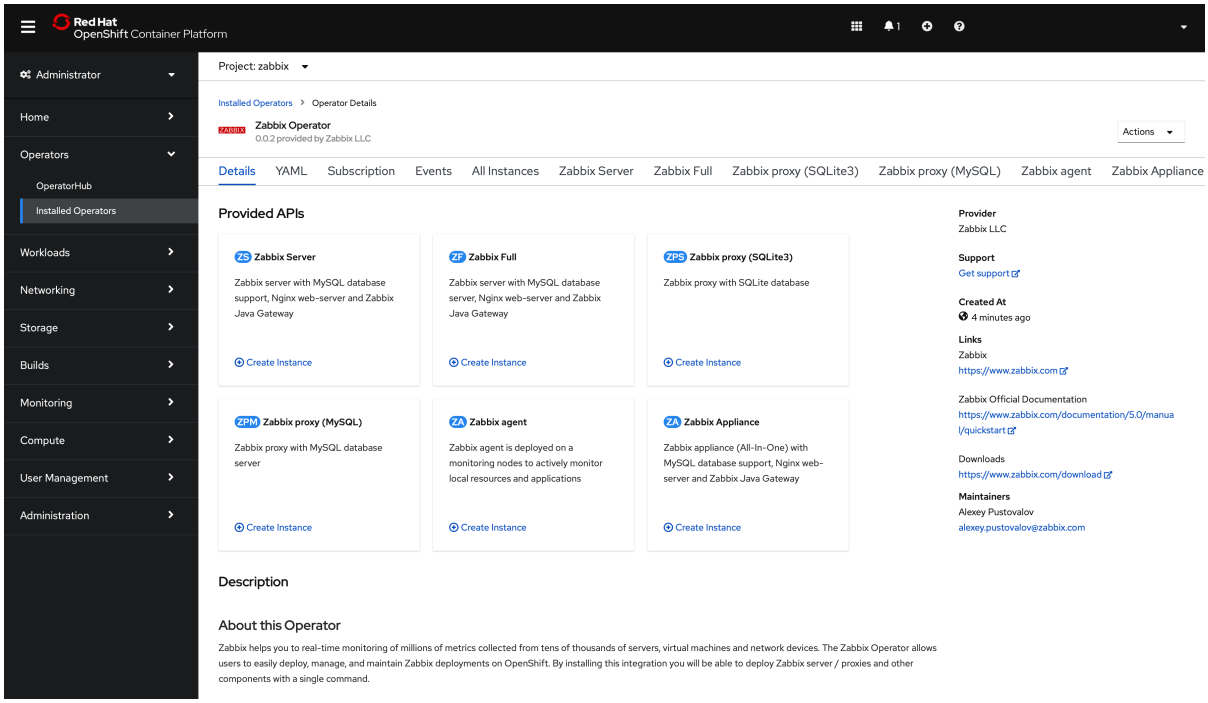
7. The Zabbix Operator is now installed into your specified cluster.



8. Go to Operators → Installed Operators.



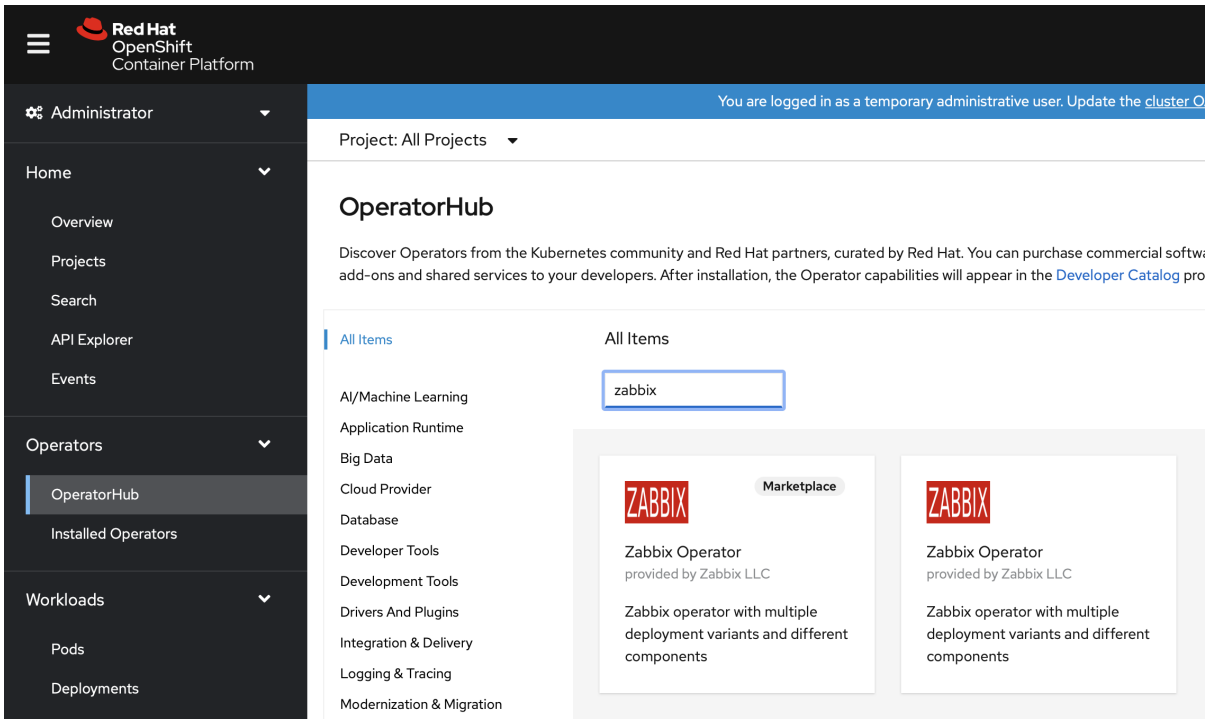
9. Open the ”Zabbix Operator” configuration page.



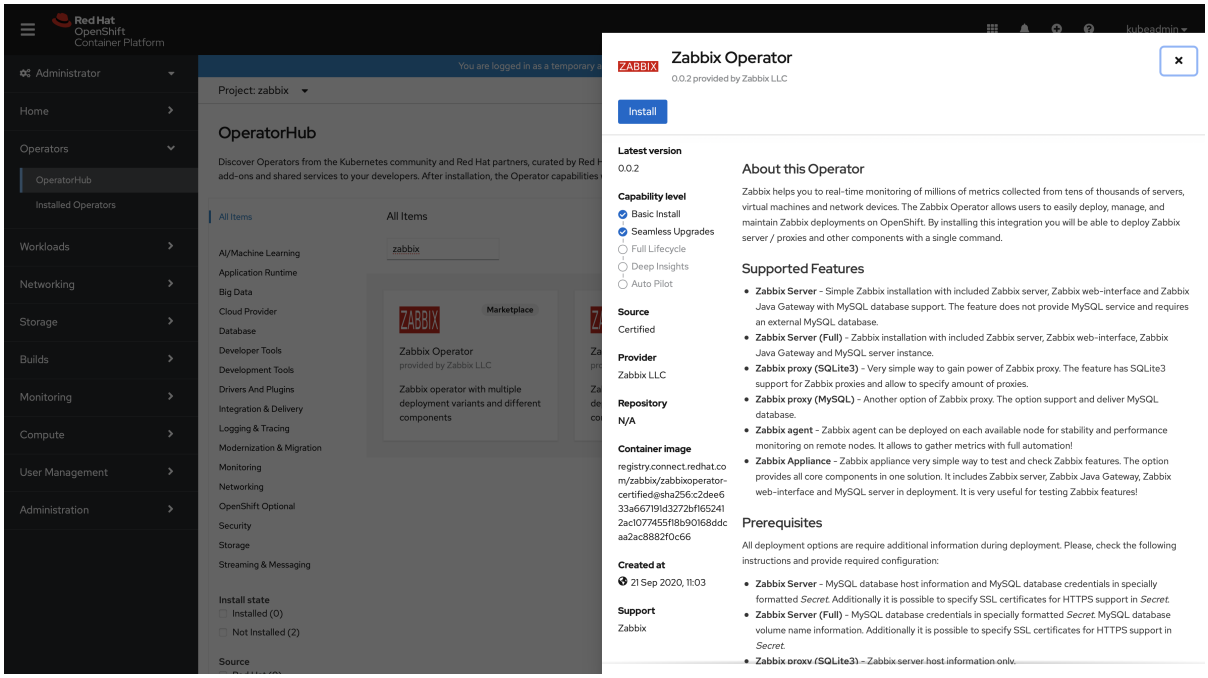
Using OperatorHub

> **Note:**
> If you have installed OpenShift in AWS ensure that the requisite ports are opened for the worker nodes' security group.

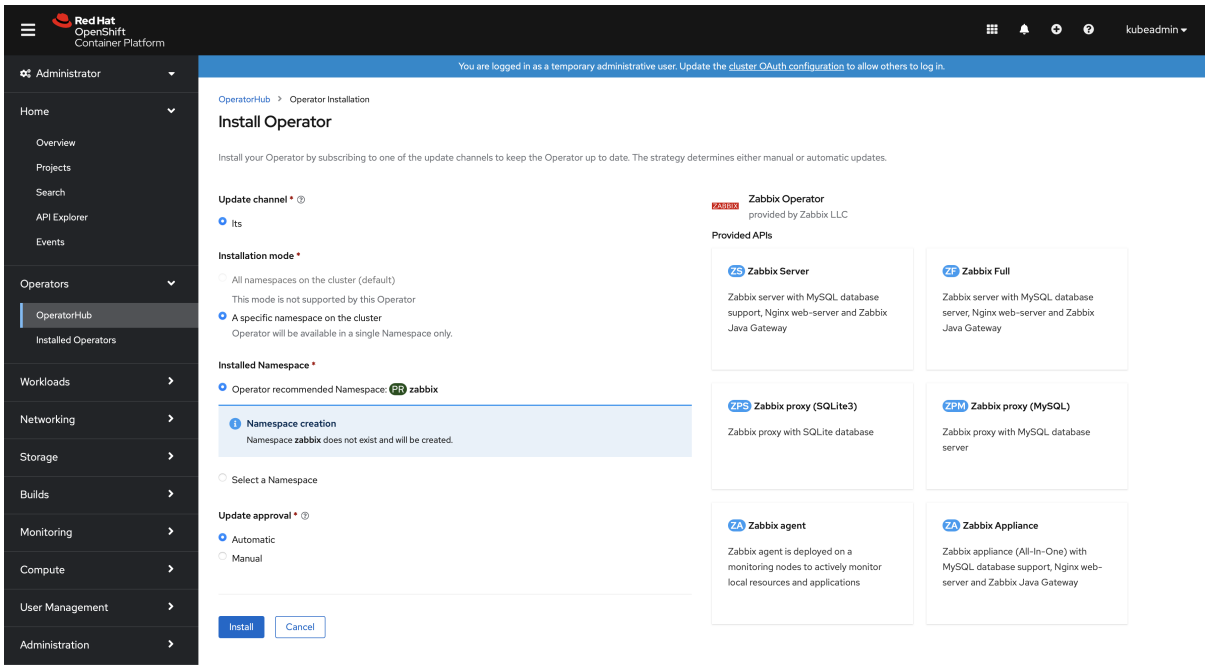
1. Select OperatorHub from the Operators submenu and search for Zabbix.



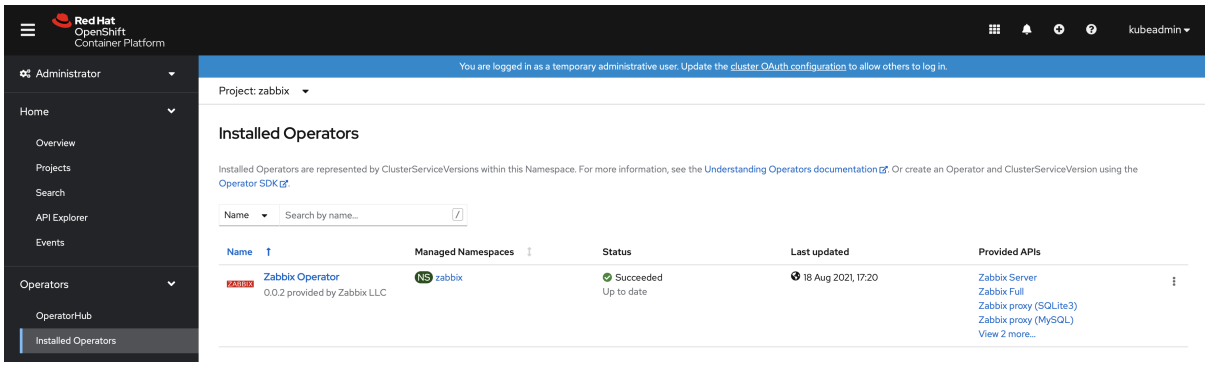
2. Select Zabbix Operator and click on Install.



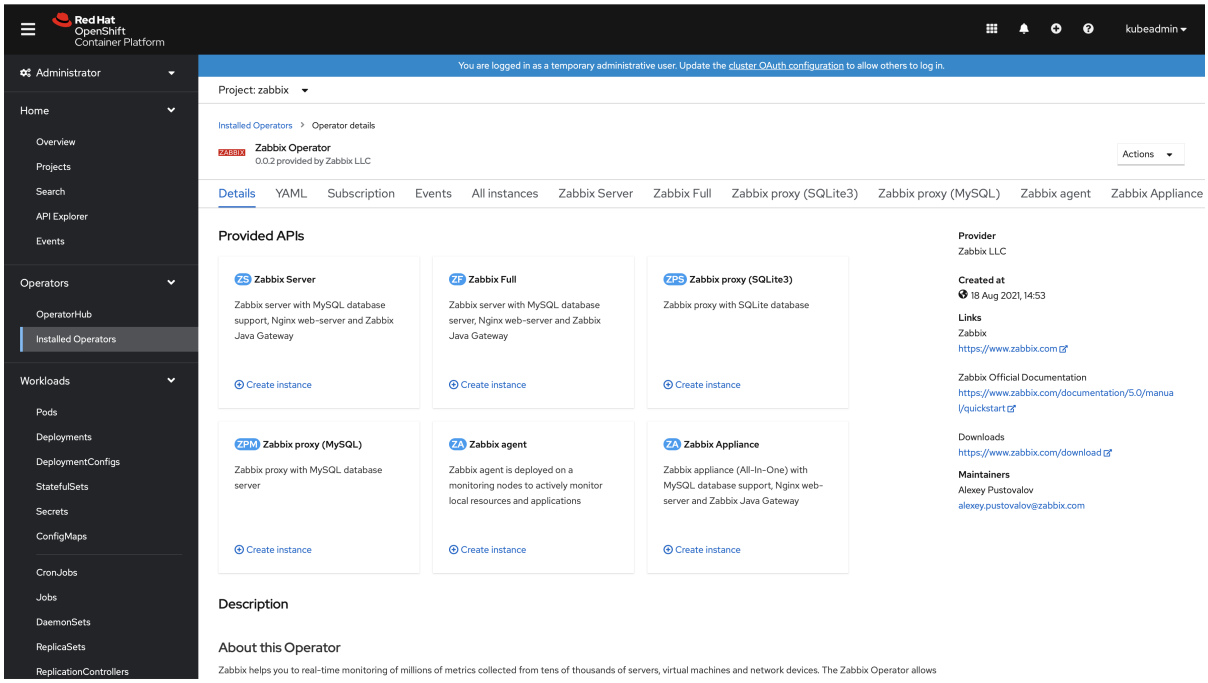
3. Select the installation options.



4. Go to Operators → Installed Operators.

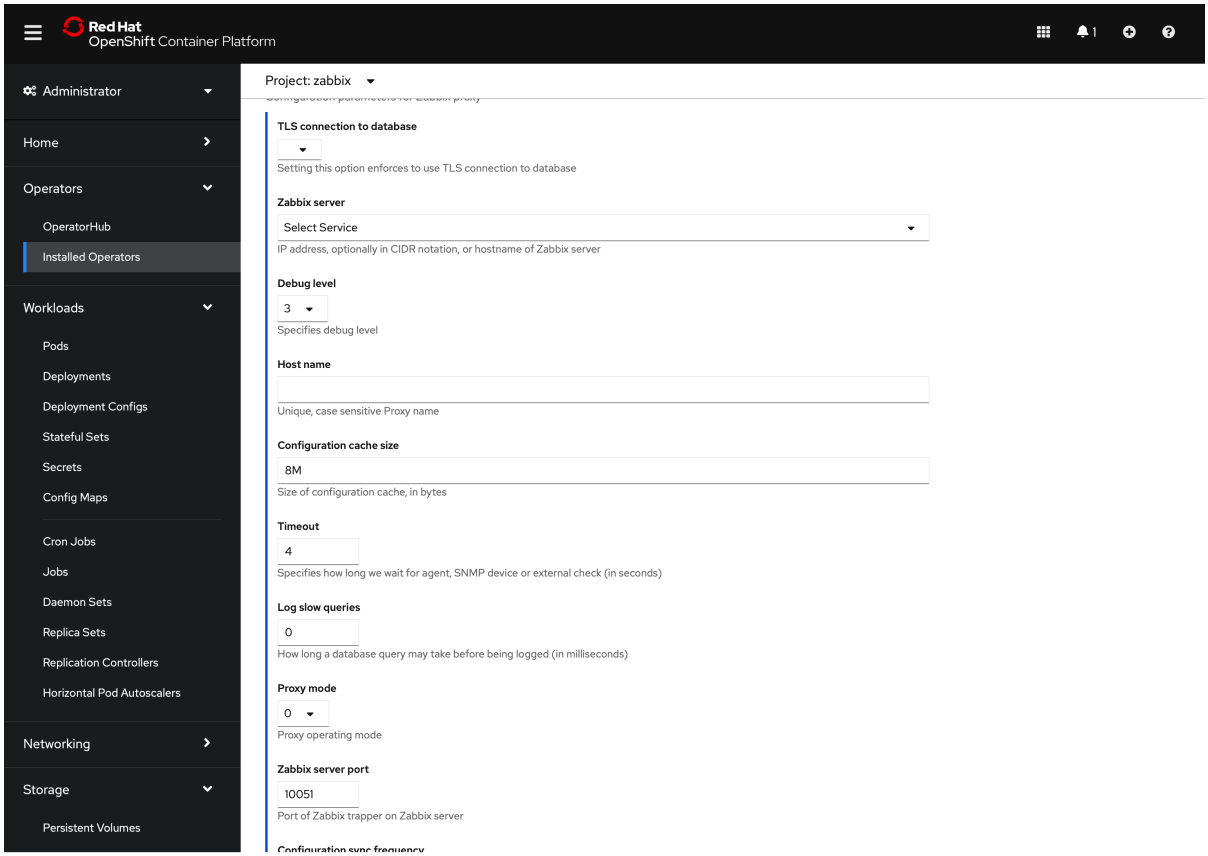


5. Open the ”Zabbix Operator” configuration page.

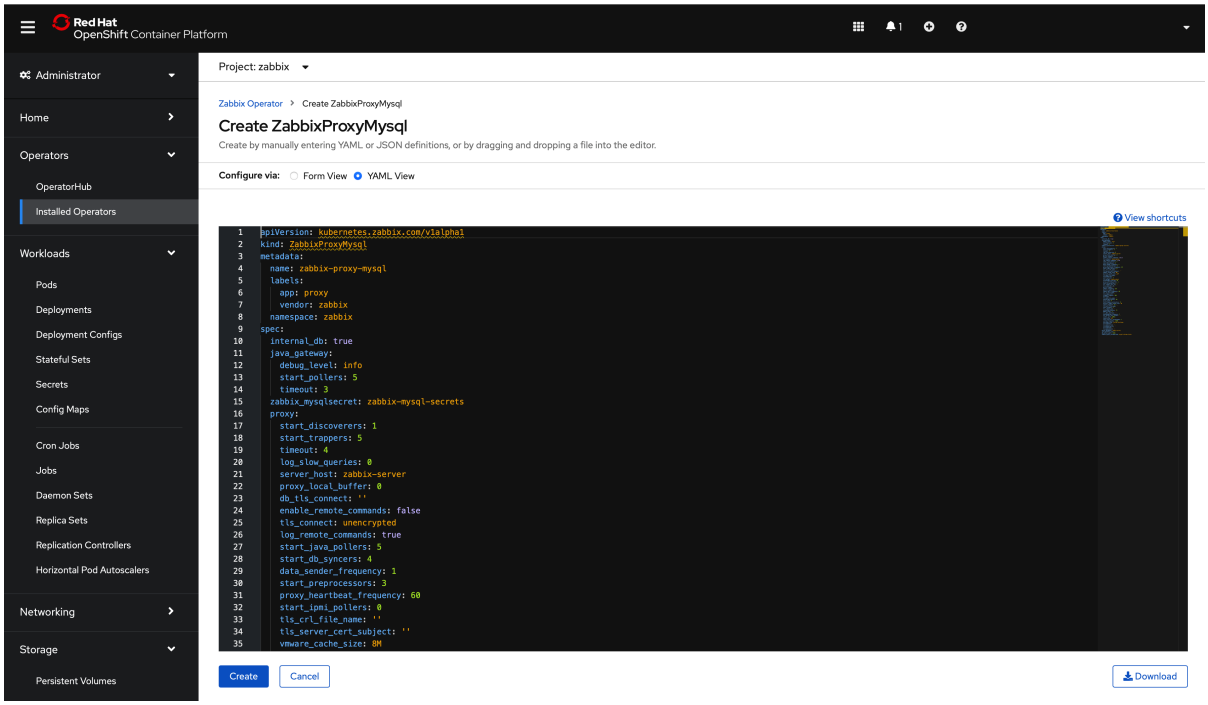


Configuration

Some of the operands (installation options) require additional resources to be created before. The following section describes these prerequisites. All possible configuration options are available during operand deployment. For example, **Zabbix proxy (MySQL)**:



The YAML section provides all available options with default values:



Zabbix server

This operand has a few prerequisites:

1. An existing MySQL database entry point - a MySQL database/cluster must be created before running the ”Zabbix Server” operand. For example, a standalone MySQL server with persistent volume:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
spec:
  selector:
    matchLabels:
      app: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - image: mysql:8.0
        name: mysql
        args:
          - mysqld
          - '--character-set-server=utf8'
          - '--collation-server=utf8_bin'
          - '--default-authentication-plugin=mysql_native_password'
        env:
```

```
          # Use secret in real usage
        - name: MYSQL_ROOT_PASSWORD
          value: Welcome1!
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: mysql-persistent-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: mysql-persistent-storage
        persistentVolumeClaim:
          claimName: mysql-pv-claim
---
apiVersion: v1
kind: Service
metadata:
  name: mysql
spec:
  ports:
  - port: 3306
  selector:
    app: mysql
  clusterIP: None
```

Please, note that Zabbix does not support a utf8_mb4 charset and default caching_sha2_password authentication plugin.

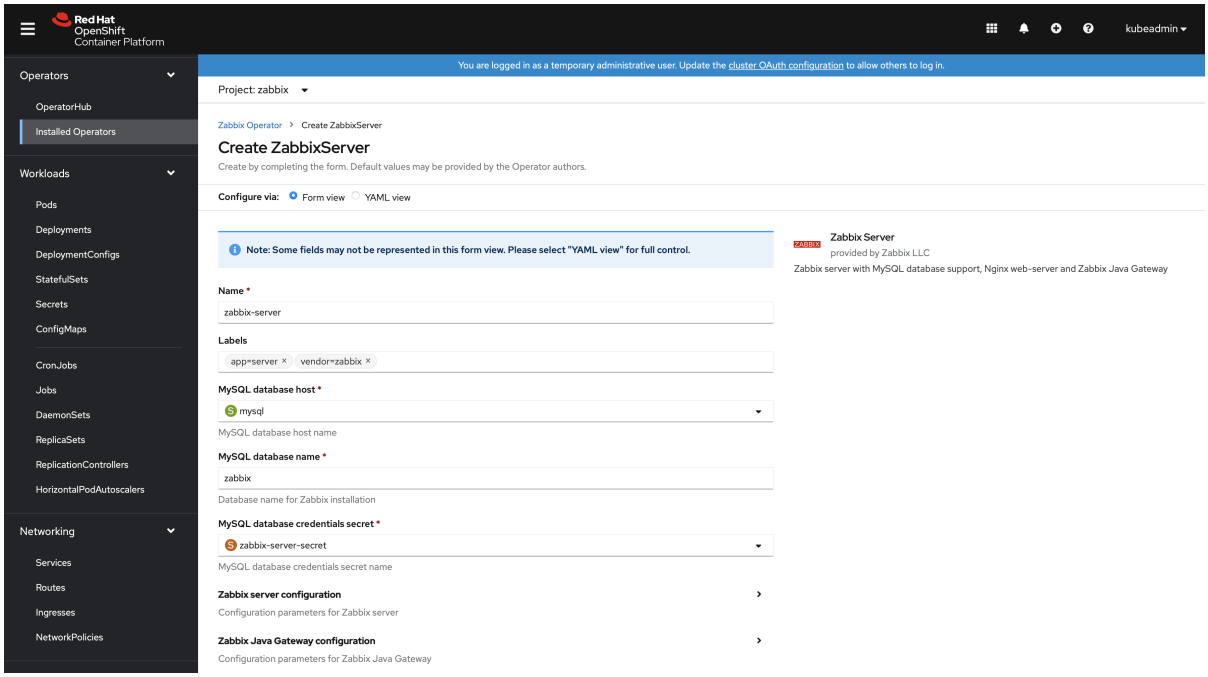
2. MySQL credentials using secret - must be secret with mysql_root_password, mysql_zabbix_username and mysql_zabbix_password data. For example:

```
kind: Secret
apiVersion: v1
metadata:
  name: zabbix-server-secrets
data:
  mysql_root_password: V2VsY29tZTEh
  mysql_zabbix_password: emFiYml4X3N1cGVyIQ==
  mysql_zabbix_username: emFiYml4
type: Opaque
```

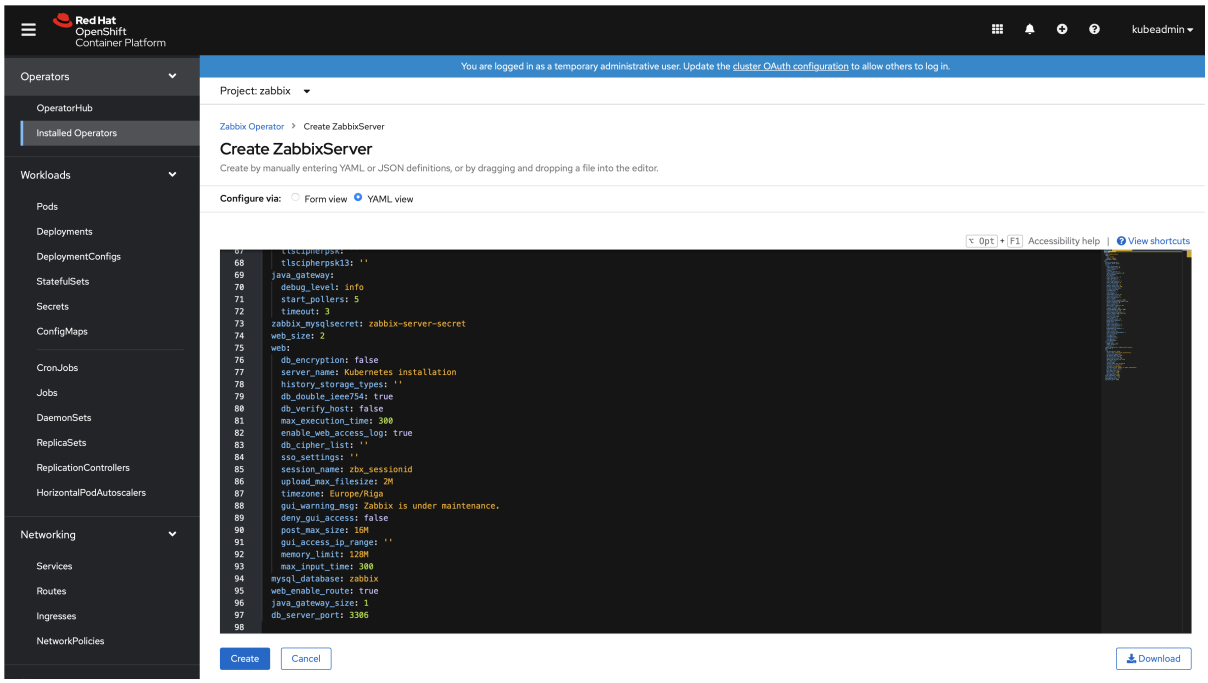
where all fields are encoded using base64. For example:

```
# echo -n "zabbix" | base64
emFiYml4Cg
```

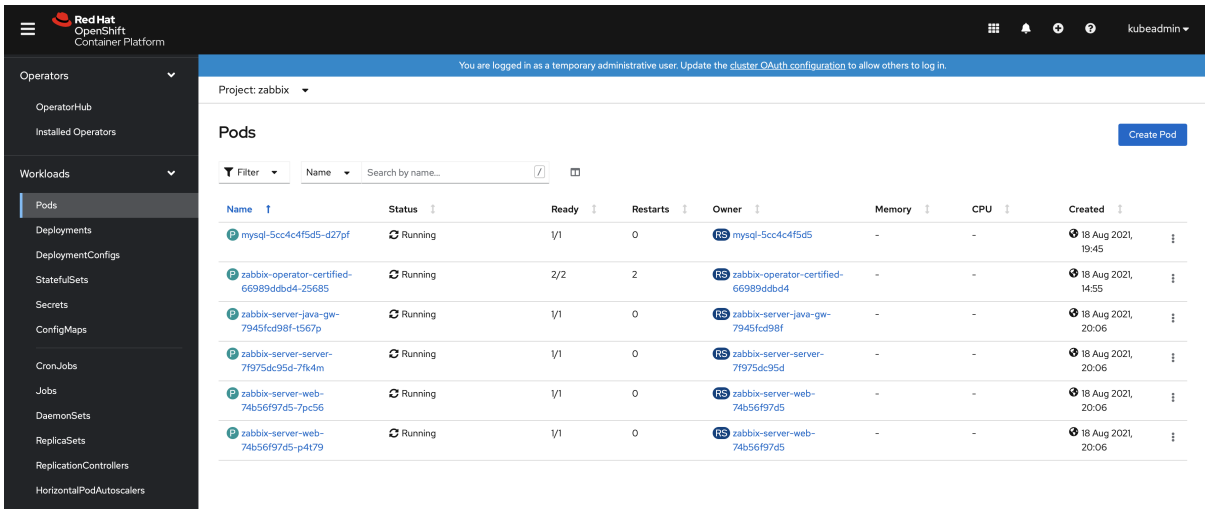
An example of ”Zabbix Server” operand configuration:



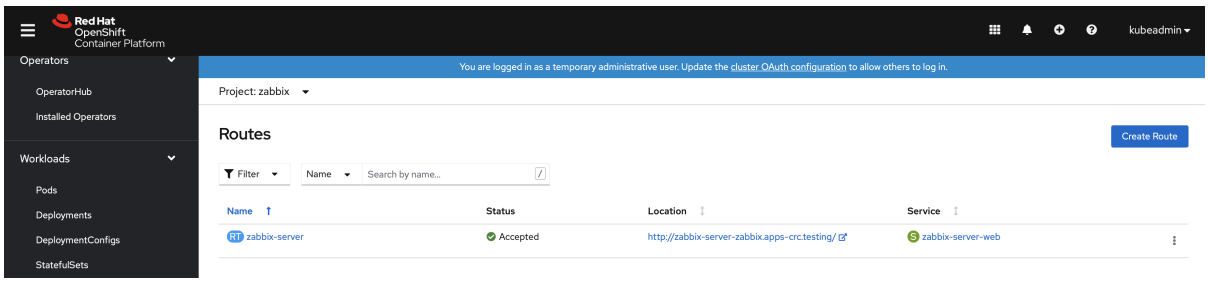
All configuration options are available using the form view, but it is possible to use the YAML view as well. For example:



Finally, the operand will create multiple pods. It is possible to examine them in the Workloads → Pods section:



The route for Zabbix web interface is located under Networking → Routes. The URL provides access to the Zabbix web interface. In the following example it is `http://zabbix-server-zabbix.apps-crc.testing/`:



Zabbix full

This operand has a few prerequisites:

1. MySQL volume claim - must be persistent volume claim. For example:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: zabbix-database
  namespace: zabbix
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 50Gi
```

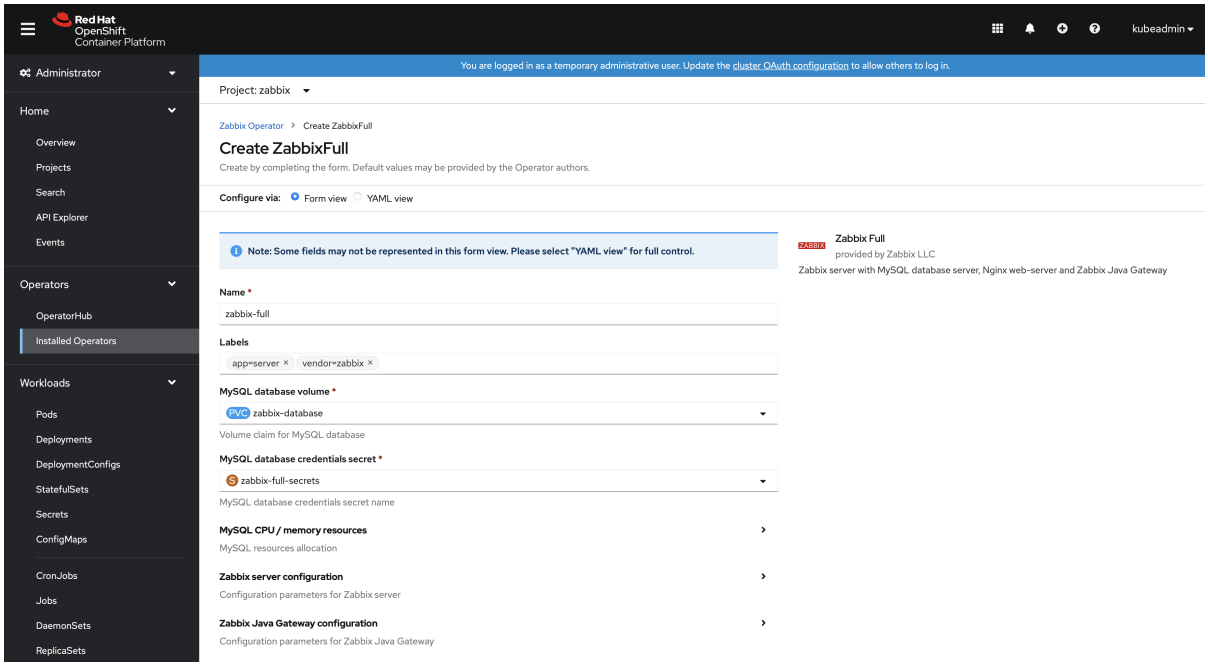
2. MySQL credentials using secret - must be secret with mysql_root_password, mysql_zabbix_username and mysql_zabbix_password data. For example:

```
kind: Secret
apiVersion: v1
metadata:
  name: zabbix-full-secrets
data:
  mysql_root_password: V2VsY29tZTEh
  mysql_zabbix_password: emFiYml4X3N1cGVyIQ==
  mysql_zabbix_username: emFiYml4
type: Opaque
```

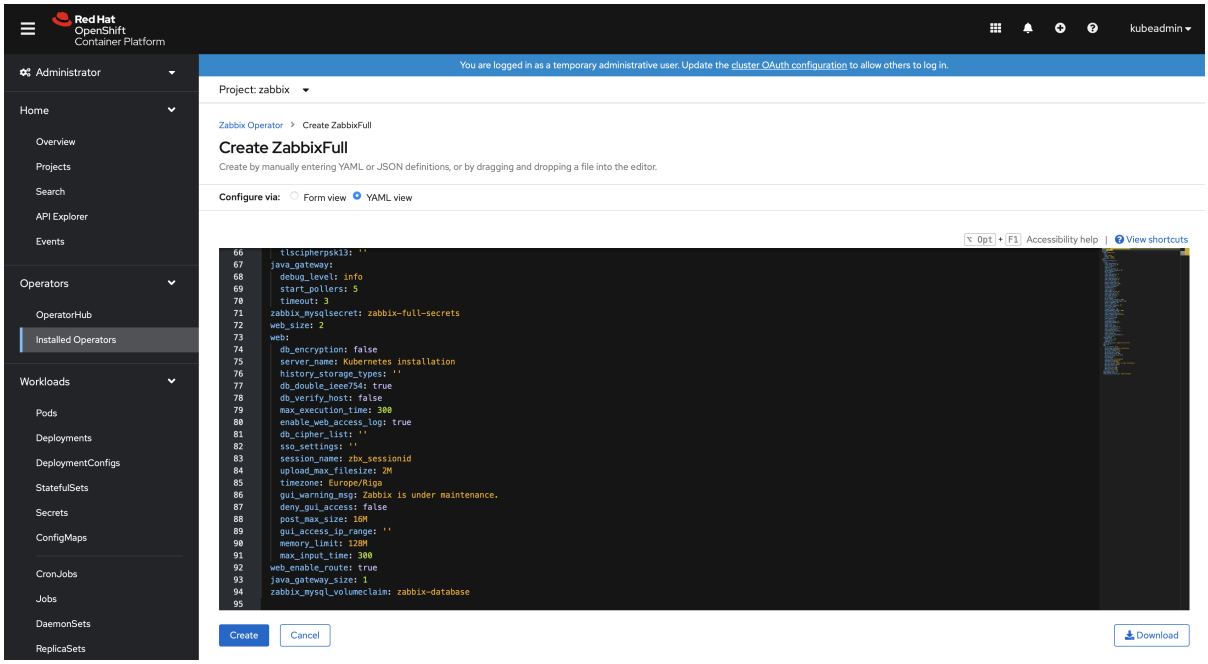
where all fields are encoded using base64. For example:

```
# echo -n "zabbix" | base64
emFiYml4Cg
```

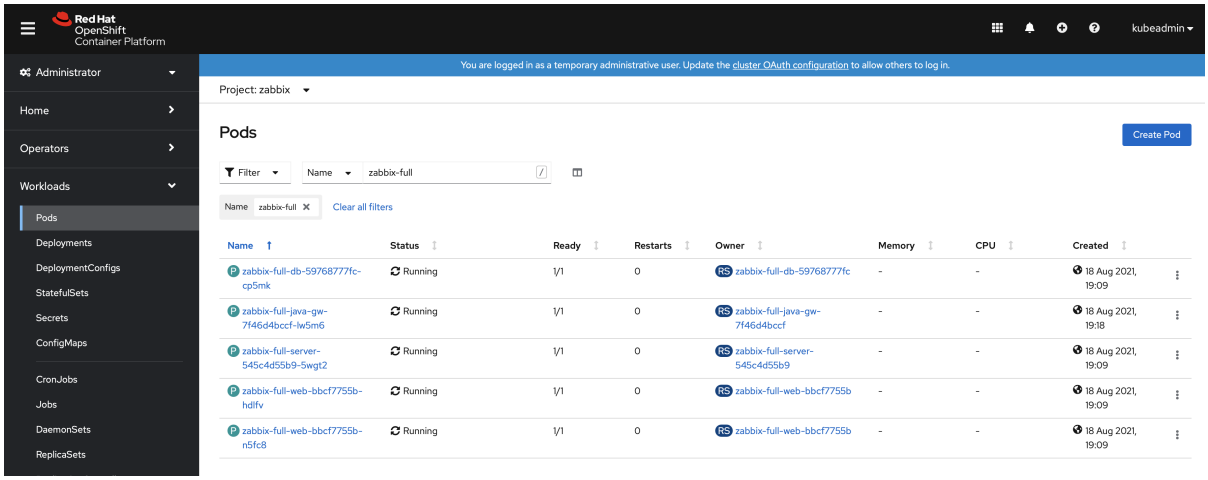
An example of ”Zabbix Full” operand configuration:



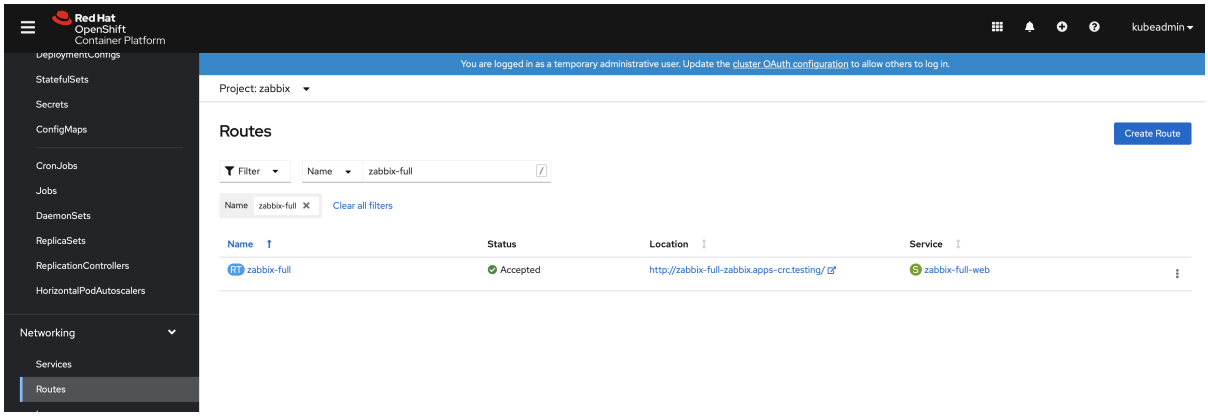
All configuration options are available using the form view, but it is possible to use the YAML view as well. For example:



Finally, the operand will create multiple pods. It is possible to examine them in the Workloads → Pods section:

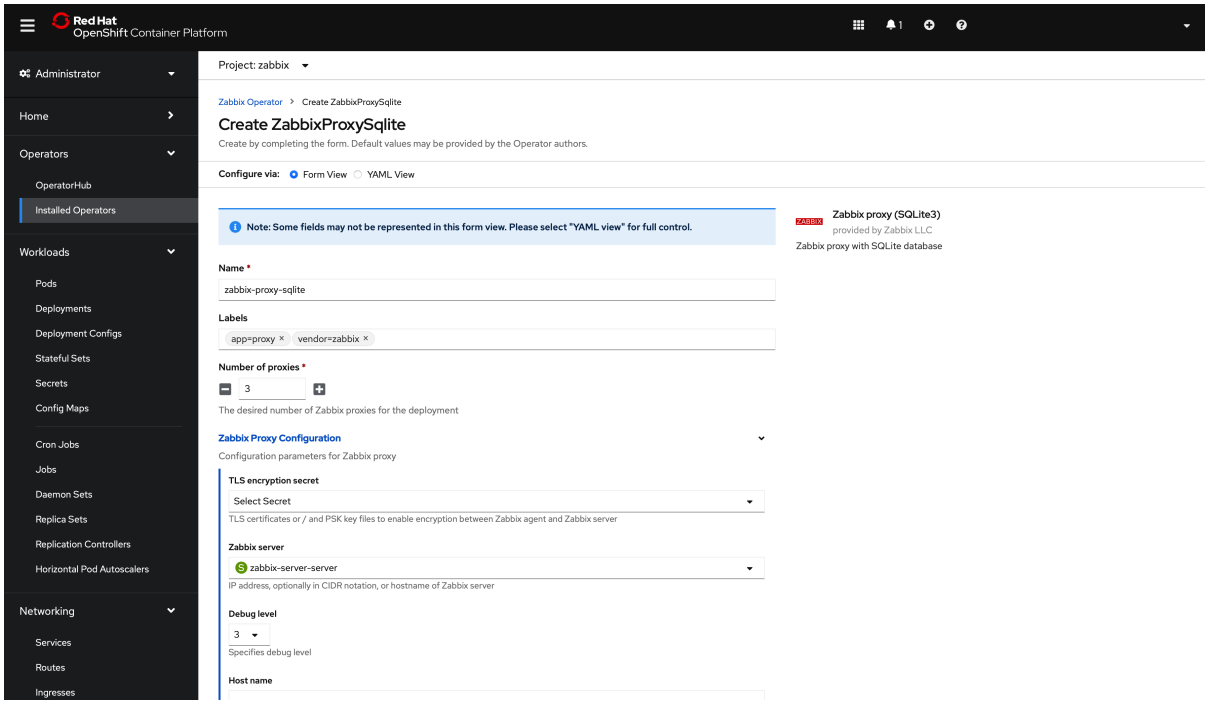


The route for Zabbix web interface is located under Networking → Routes. The URL provides access to the Zabbix web interface. In the following example it is `http://zabbix-full-zabbix.apps-crc.testing/`:

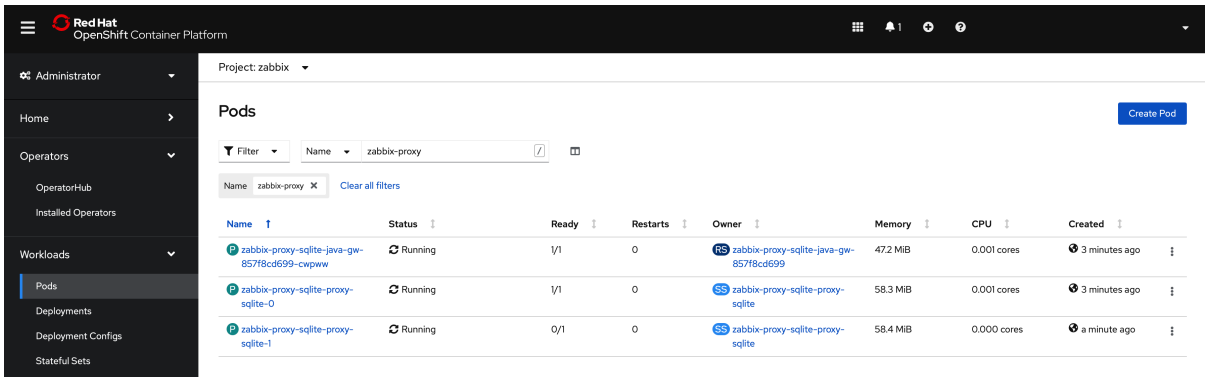


Zabbix proxy (SQLite3)

An example of ”Zabbix proxy (SQLite3)” operand configuration:



Finally, the operand will create multiple pods. It is possible to examine them in the Workloads → Pods section:

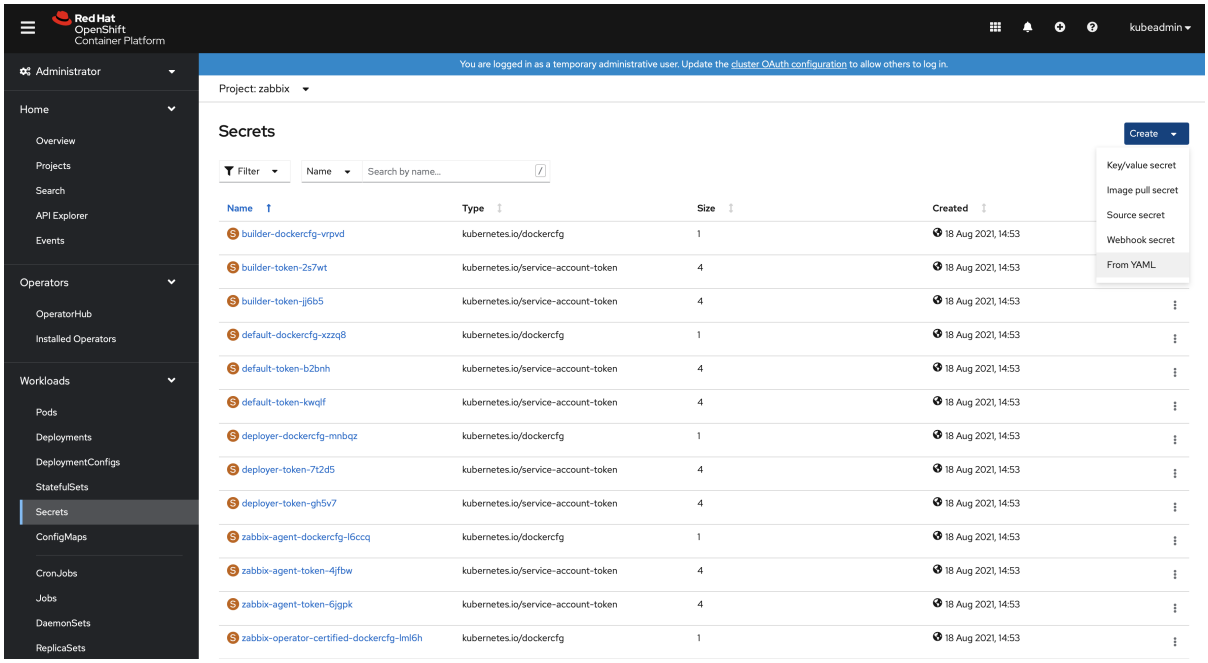


Additional information

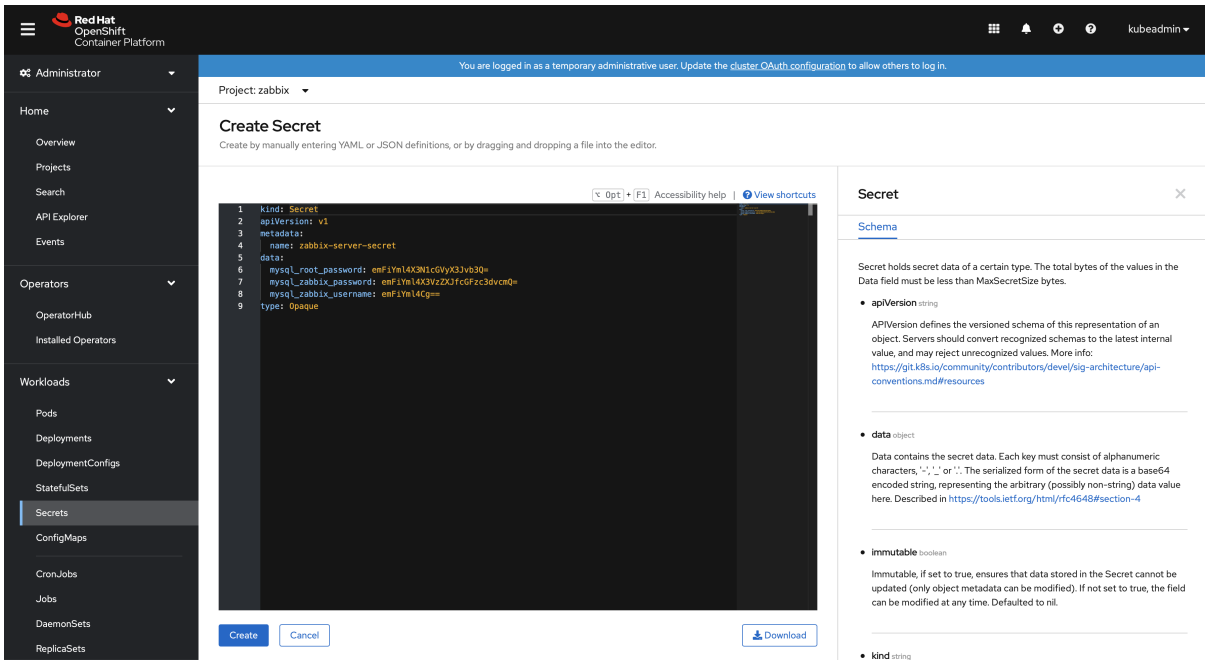
Creating new secret

The following procedure describes how to create a new secret using Openshift Console.

1. Open the Workloads → Secrets section and switch project to the Zabbix Operator project (by default, ”zabbix”).



2. Create a new secret using the *From YAML* option.



SSL certificates for HTTPS

It is possible to enable HTTPS directly in the Zabbix web interface pods. In this case create the following secret using the YAML option:

```
kind: Secret
apiVersion: v1
metadata:
  name: zabbix-web-sslsecret
data:
  ssl.crt: >-
   < ssl.crt data>
  ssl.key: >-
     < ssl.key data >
  dhparam.pem: >-
   <  dhparam.pem data >
```

The names of certificates and DH Parameters file are static. Please use the listed in the above example only!

MySQL database certificate base encryption

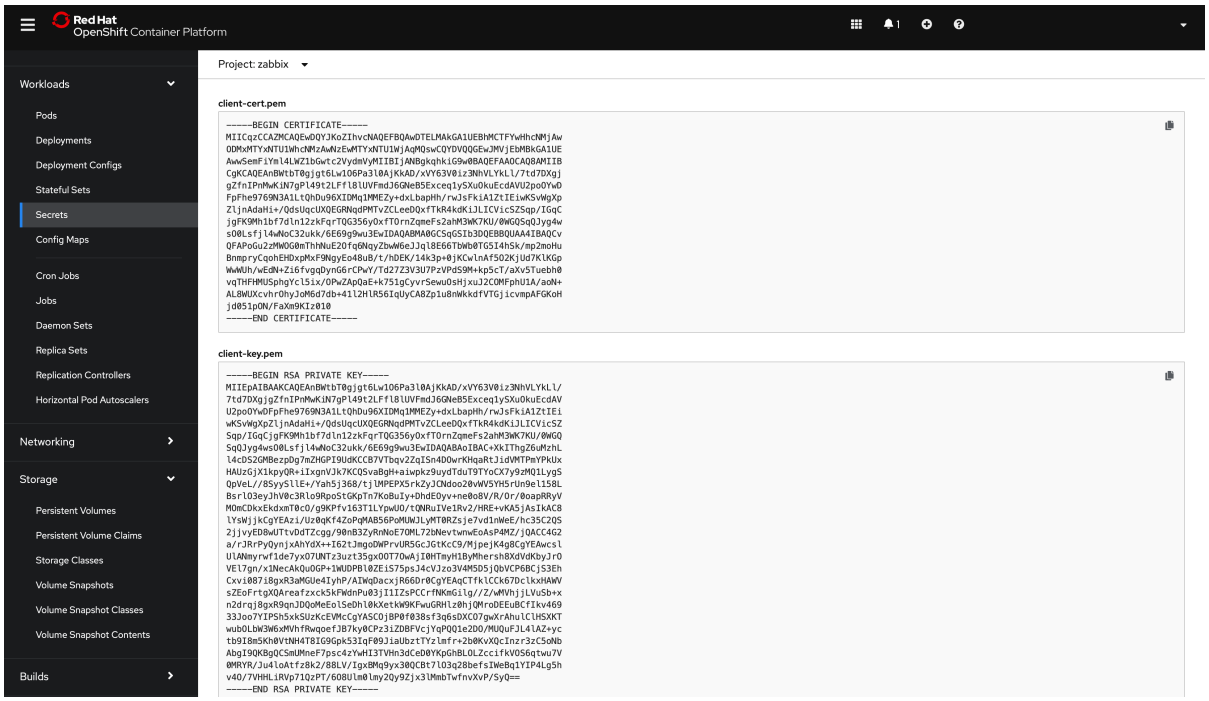
MySQL database side:

```
apiVersion: v1
data:
 root-ca.pem: >-
  < root-ca.pem data>
 server-cert.pem: >-
  < server-cert.pem data>
 server-key.pem: >-
  < server-key.pem data>
kind: Secret
metadata:
 name: zabbix-db-server-tls-secret
type: Opaque
```

Zabbix components side:

```
apiVersion: v1
data:
 client-cert.pem: >-
  < client-cert.pem data>
 client-key.pem: >-
  < client-key.pem data>
 root-ca.pem: >-
  < root-ca.pem data>
kind: Secret
metadata:
 name: zabbix-db-client-tls-secret
type: Opaque
```

Certificates must include "-----BEGIN RSA PRIVATE KEY-----" and "-----END RSA PRIVATE KEY-----". For example:



Then, during deployment, in the Zabbix component section and MySQL server (if using built-in server) choose the proper ”TLS connection to database” option value and the ”MySQL database certificates (client)” secret value.

Known issues

1. Zabbix agent does not have the possibility to determine proper node name. It always has dynamic hostname.

6 Web 界面安装

本章节提供有关 Zabbx Web 界面的部署步骤说明。Zabbix 前端是由 PHP 语言编写，所以其网页服务的运行需要支持 PHP 语言的网站服务器。

> **Note:**
> You can find out more about setting up SSL for Zabbix frontend by referring to these best practices.

欢迎主界面

在浏览器中输入 Zabbix 前端的 URL 来进入主界面。通过依赖包的方式对 Zabbix 进行安装，其 URL 的输入格式会略有不同，相关格式如下所示：

- 对于 Apache: *http://<server_ip_or_name>/zabbix*
- 对于 Nginx: *http://<server_ip_or_name>*

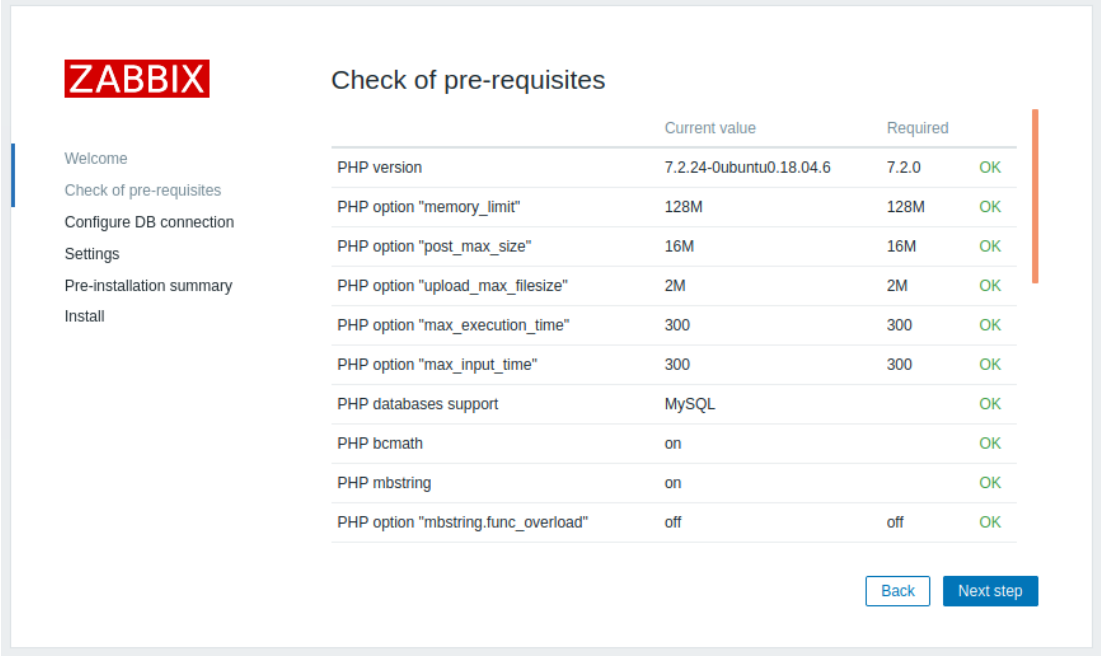
根据安装方式输入正确的 URL 后，您将会进入到前端安装的向导程序。

使用系统默认语言下拉菜单，更改系统默认语言，并以所选语言继续安装过程（非必选）。详细信息，请参考Installation of additional frontend languages.



先决条件检查

确保满足所有软件先决条件。



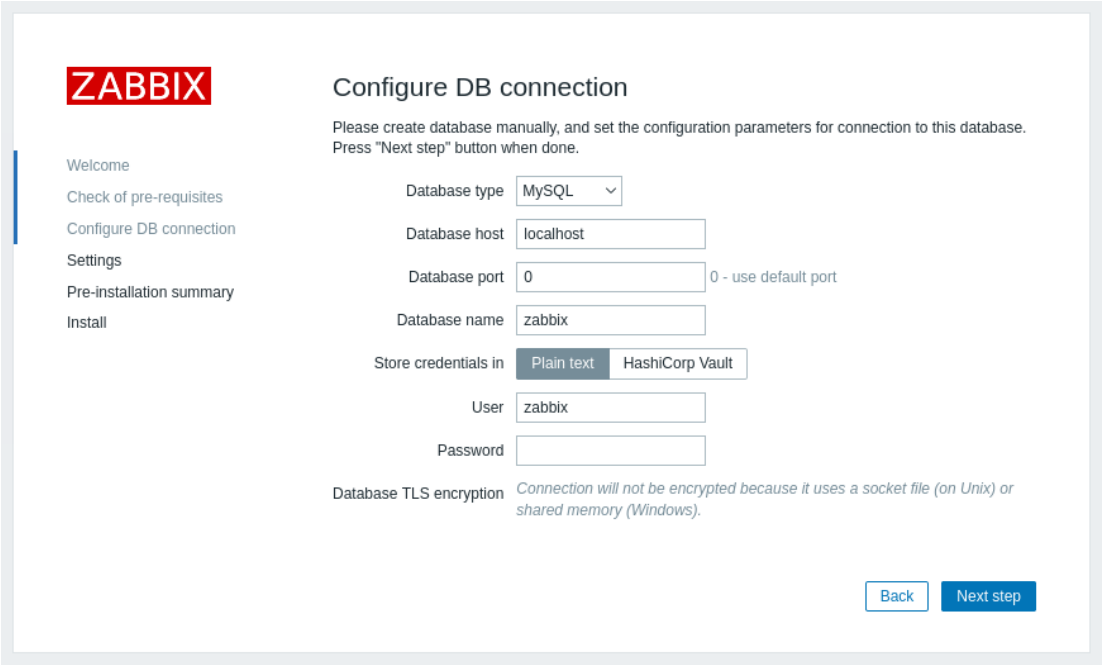
先决条件	最小值	描述
PHP 版本	7.2.5	
PHP memory_limit 选项	128MB	在 php.ini 中： memory_limit = 128M
PHP post_max_size 选项	16MB	在 php.ini 中： post_max_size = 16M
PHP upload_max_filesize 选项	2MB	在 php.ini 中： upload_max_filesize = 2M
PHP max_execution_time 选项	300 秒（允许值 0 和 -1）	在 php.ini 中： max_execution_time = 300
PHP max_input_time 选项	300 秒（允许值 0 和 -1）	在 php.ini 中： max_input_time = 300
PHP session.auto_start 选项	必须禁用	在 php.ini: session.auto_start = 0
数据库支持	其中之一：MySQL、Oracle、PostgreSQL。	必须安装以下模块之一： mysql、oci8、pgsql
bcmath		php-bcmath
mbstring		php-mbstring
PHP mbstring.func_overload 选项	必须禁用	在 php.ini: mbstring.func_overload = 0
sockets		php-net-socket. 需要用户脚本支持。
gd	2.0.28	php-gd. PHP GD 扩展必须支持 PNG 图像 (*--with-png-dir*)、JPEG (*--with-jpeg-dir*) 图像和 FreeType 2 (*--with-freetype-dir*)。
libxml	2.6.15	php-xml
xmlwriter		php-xmlwriter
xmlreader		php-xmlreader
ctype		php-ctype
session		php-session
gettext		php-gettext 自 Zabbix 2.2.1 起，PHP gettext 扩展不是安装 Zabbix 的强制要求。如果未安装 gettext，前端将照常工作，但是翻译将不可用。

可选的先决条件也会罗列在列表中。一个失败的可选先决条件会显示为橙色，并具有 *Warning* 的状态。如果可选先决条件不满足，安装程序也可以继续进行。

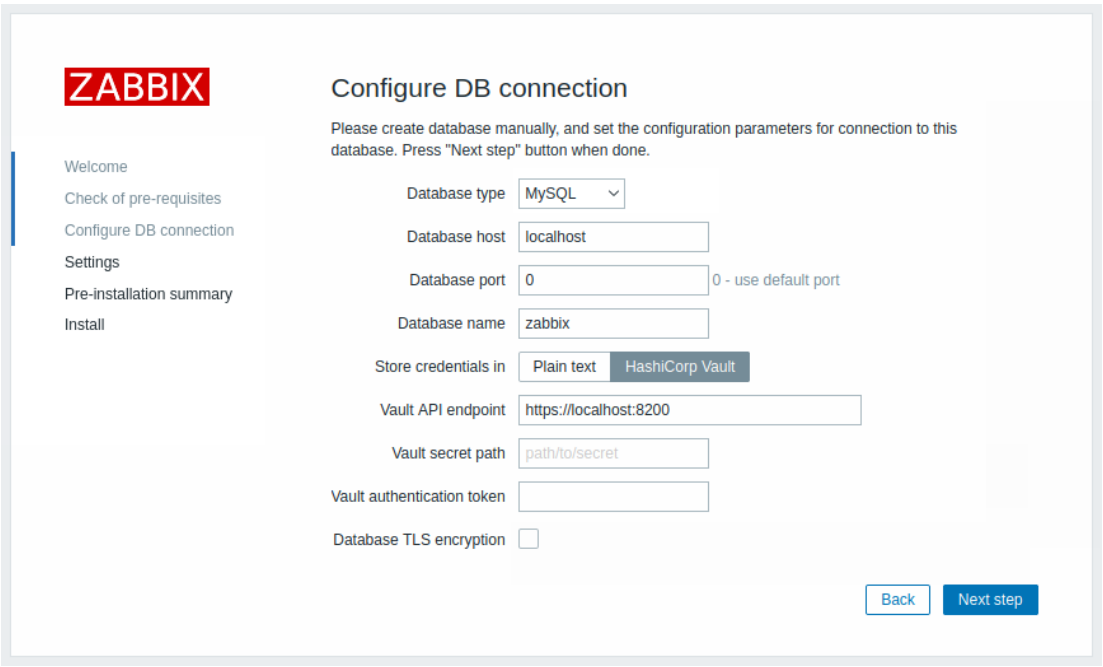
Attention:
若需要更改 Apache 的用户或用户组，则必须验证会话文件夹的权限。否则 Zabbix 的安装将无法继续。

配置数据库连通性

请在该页面输入连接到数据库所需的详细信息。在创建与数据库的连接前，Zabbix 数据库必须先被创建。



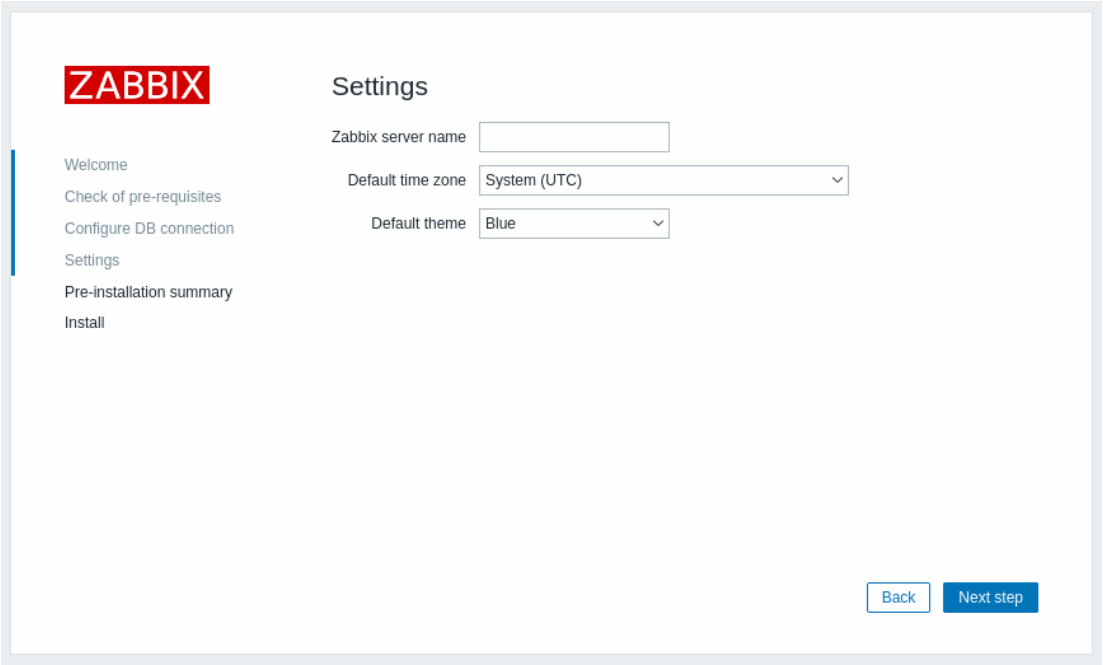
若选择 *Database TLS encryption* 选项，则需要在出现的信息栏中填写有关configuring the TLS connection的配置信息（该功能仅限数据库类型为 MySQL 或 PostgreSQL）。若选择 HashiCorp Vault 选项来进行凭据存储，请在附加的信息栏中输入相关信息，用以说明 Vault API 端点、隐藏路径以及身份验证令牌：



配置

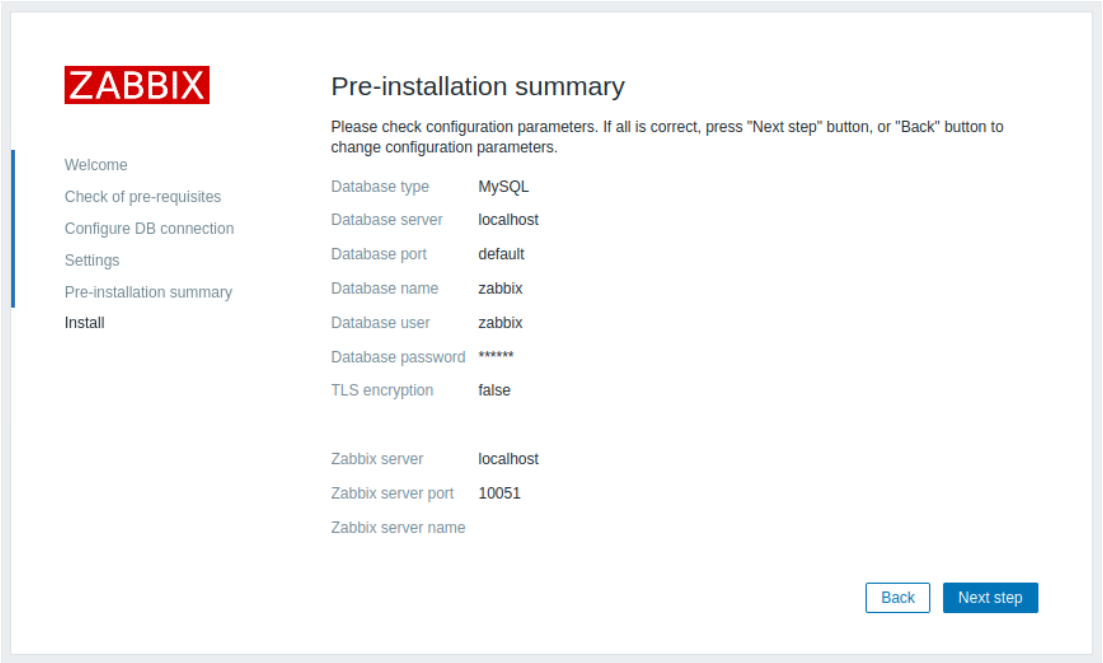
对 Zabbix 服务器进行命名的配置为可选配置。该配置一旦提交，设定的服务器名称就会显示在网页的菜单栏和页面标题中。

配置默认time zone和前端的主题。



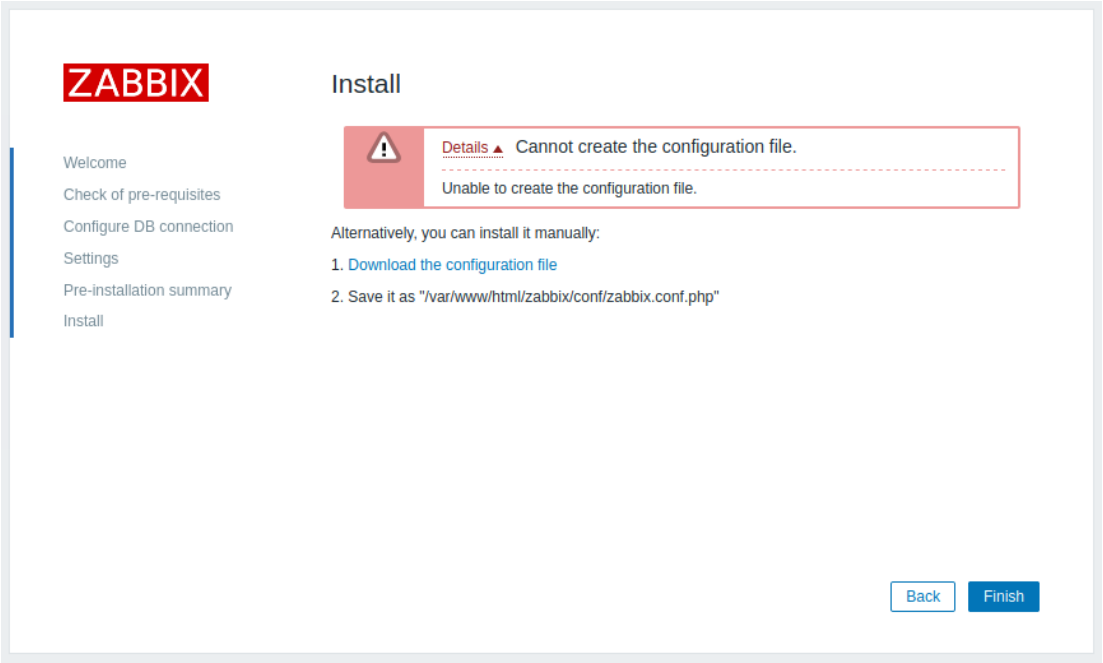
预安装总概

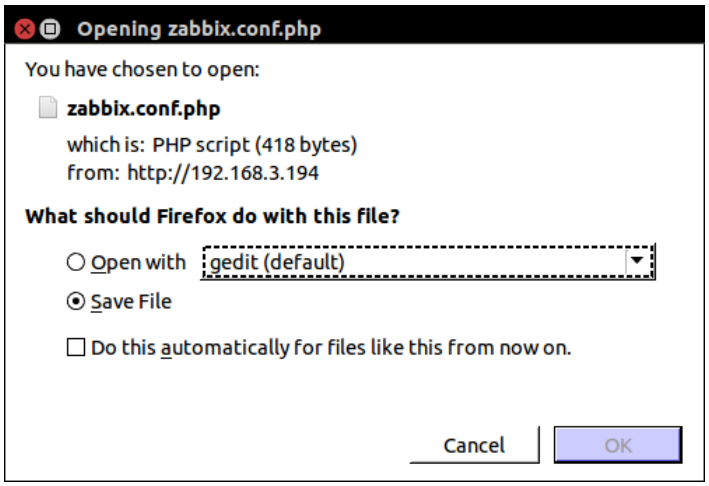
查看配置概要。



安装

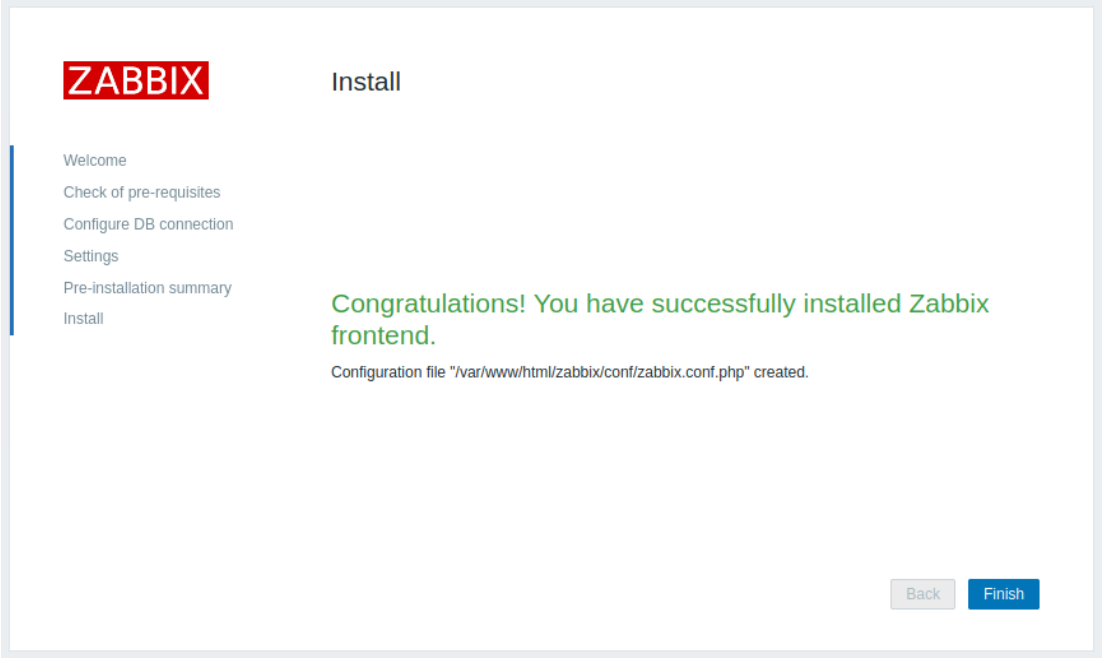
若采用从源代码安装 Zabbix，请下载配置文件并将其 Zabbix PHP 文件复制到所在网站服务器 HTML 文件子目录中的 conf/ 下。





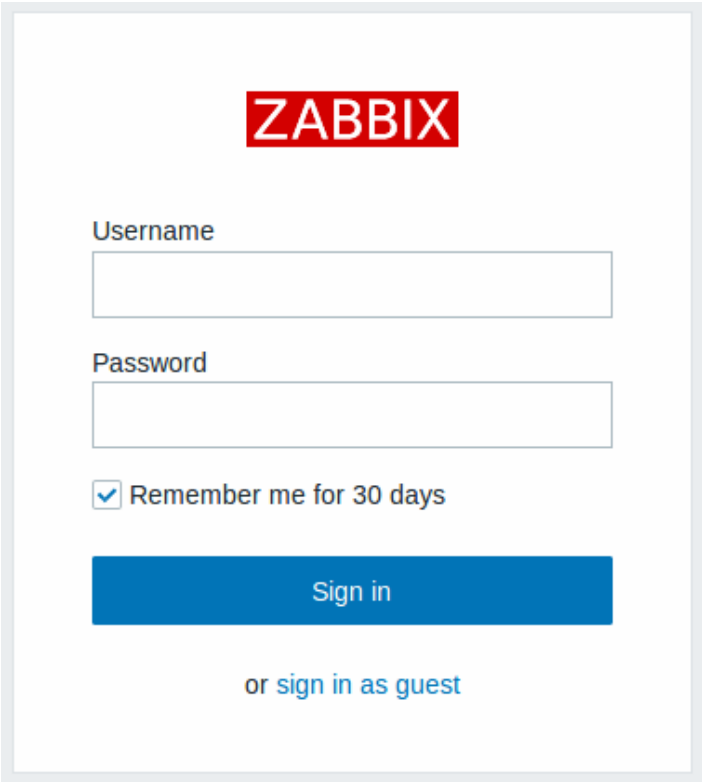
Note:
若网站服务器用户对 conf/ 目录具有写入权限，则配置文件将自动保存，并且可以立即执行下一步。

完成安装。



登录

Zabbix 前端已准备就绪！默认用户名是 **Admin**，密码 **zabbix**。



继续getting started with Zabbix.

Debian/Ubuntu 前端安装

概述

自 Zabbix 5.0 版本开始，Zabbix 前端需 PHP 7.2 及以上版本。不幸的是，老版本的 Debian 和 Ubuntu 只提供 PHP 低于 7.2 的版本。

发行版支持的 PHP 版本

发行版	PHP 版本
Debian 10 (buster)	7.3
Debian 9 (stretch)	7.0
Debian 8 (jessie)	5.6
Ubuntu 20.04 (focal)	7.4
Ubuntu 18.04 (bionic)	7.2
Ubuntu 16.04 (xenial)	7.0
Ubuntu 14.04 (trusty)	5.5
Raspbian 10 (buster)	7.3
Raspbian 8 (stretch)	7.0

在 *stretch*、*jessie*、*xenial* 和 *trusty* 发行版上，PHP 7.2 依赖项不可用，无法通过简单的方式安装 Zabbix 前端的 5.0 或者更高版本。考虑到这一点，在上述发行版中，`zabbix-frontend-php` 包已被替换为 `zabbix-frontend-php-deprecated` 包。
主要区别在于没有对任何 php 或网络服务器包有直接的依赖关系。因此，用户可以（并且必须）自己提供这些依赖项。换句话说，单独安装 `zabbix-frontend-php-deprecated` 包不会给你一个工作的前端。必须手动安装 Web 服务器及其模块的 PHP 7.2（使用 PPA/从源代码构建 PHP）。我们不赞同使用特殊的安装方式。

> **Note:**
> 在旧版本的 Debian/Ubuntu 上获取 PHP 7.2 或更高版本的官方方法是升级到 buster/bionic。

Zabbix 前端所需的 PHP 模块有 `php-gd`、`php-bcmath`、`php-mbstring`、`php-xml`、`php-ldap` 和 `php-json`。

7 升级步骤

概述

本章节提供了关于升级至 Zabbix **6.0** 的信息：

- 使用二进制包安装：
  - 对于Red Hat Enterprise Linux/CentOS
  - 对于Debian/Ubuntu
- 使用源码包sources

可以从 Zabbix **5.4**.x、**5.2**.x、**5.0**.x、**4.4**.x、**4.2**.x、**4.0**.x、**3.4**.x、**3.2**.x、**3.0**.x、**2.4**.x、**2.2**.x 和 * *2.0**.x 直接升级到 Zabbix 6.0.x。要从早期版本升级，请参阅 2.0 及更早版本的 Zabbix 文档。

Note:
请注意，在升级后，如果外部软件与升级后的 Zabbix 版本不兼容，Zabbix 中的某些第三方软件集成可能会受到影响。

由二进制包升级

概述

本章节提供使用 Zabbix 的 RPM 和 DEB 二进制包成功升级至 Zabbix 6.0 所需的步骤升级 ：

- 红帽企业 Linux/CentOS
- Debian/Ubuntu

由操作系统存储库中的 Zabbix 软件包进行升级

通常，发行版本的操作系统（特别是基于 Debian 的发行版）会提供基于自身系统的 Zabbix 软件包。
请注意，Zabbix 不支持这些软件包，它们通常已经过时并且缺乏最新的功能和错误修复。建议通过由官方支持的 repo.zabbix.com 软件安装包来完成安装。

若需从 OS 发行版提供的软件包进行升级（或在某时已安装），请按照以下步骤换到官方 Zabbix 软件包：

1. 首先，卸载旧版本的安装包。
2. 检查卸载后可能留下的残留文件。
3. 按照 Zabbix 提供的 installation instructions安装官方包。

切勿进行直接更新，因为这可能会导致安装中断。

1 Red Hat 企业版 Linux

概述

本节提供使用 Red Hat Enterprise Linux 的官方 Zabbix 软件包从 Zabbix **5.4**.x 成功升级 到 Zabbix **6.0**.x 所需的步骤。

虽然升级 Zabbix agent 不是强制性的（但推荐），但 Zabbix 服务器和代理必须是相同的主要版本。因此，在服务器-代理设置中，必须停止和升级 Zabbix 服务器和所有代理。在服务器升级期间保持代理运行不再带来任何好处，因为在代理升级期间，它们的旧数据将被丢弃，并且不会收集新数据直到代理配置与服务器同步。

请注意，对于代理上的 SQLite 数据库，升级前来自代理的历史数据将丢失，因为不支持 SQLite 数据库升级并且必须手动删除 SQLite 数据库文件。proxy 第一次启动时 SQLite 数据库文件丢失，proxy 会自动创建。

根据数据库大小，数据库升级到版本 6.0 可能需要很长时间。

Warning:
升级前请务必阅读相关的升级说明！

提供以下升级说明：

升级自	阅读完整的升级说明	版本之间最重要的变化
5.4.x ·	对于： Zabbix 6.0	所需的最低数据库版本已提高； 如果数据库过时，服务器/代理将不会启动； 审计日志记录丢失因为数据库结构改变。

升级自	阅读完整的升级说明	版本之间最重要的变化
5.2.x ·	对于： Zabbix 5.4 Zabbix 6.0	所需的最低数据库版本已提高； 将删除的监控项汇总为单独的类型。
5.0.x LTS	适用于： Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	最低要求的 PHP 版本从 7.2.0 升级到 7.2.5。
4.4.x ·	对于： Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	放弃对 IBM DB2 的支持； 所需的最低 PHP 版本从 5.4.0 提高到 7.2.0； 所需的最低数据库版本已提高； 更改了 Zabbix PHP 文件目录。
4.2.x ·	对于： Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	Jabber、Ez Texting 媒体类型已删除。
4.0.x LTS	对于： Zabbix 4.2 Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 >Zabbix 6.0	较旧的代理不再可以向升级后的服务器报告数据； 较新的代理将不再能够与较旧的 Zabbix 服务器一起工作。
3.4.x ·	对于： Zabbix 4.0 Zabbix 4.2 Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 >Zabbix 5.4 Zabbix 6.0	'libpthread' 和'zlib' 库现在是强制性的； 对纯文本协议的支持被删除，标头是强制性的； 1.4 之前版本的 Zabbix 代理不再受支持； 被动代理配置中的服务器参数现在是强制性的。
3.2.x ·	对于： Zabbix 3.4 Zabbix 4.0 Zabbix 4.2 Zabbix 4.4 Zabbix 5.0 >Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	SQLite 支持作为 Zabbix 服务器/前端的后端数据库； 支持 Perl 兼容正则表达式 (PCRE) 而不是 POSIX 扩展； 'libpcre' 和'libevent' 库对于 Zabbix 服务器是必需的； 为用户 p 添加了退出代码检查参数、远程命令和 system.run[] 项目没有'nowait' 标志以及 Zabbix 服务器执行的脚本； Zabbix Java 网关必须升级以支持新功能。
3.0.x LTS	适用于： Zabbix 3.2 Zabbix 3.4 Zabbix 4.0 Zabbix 4.2 Zabbix 4.4 >Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	数据库升级可能会很慢，具体取决于历史表的大小。

升级自	阅读完整的升级说明	版本之间最重要的变化
2.4.x ·	对于： Zabbix 3.0 Zabbix 3.2 Zabbix 3.4 Zabbix 4.0 Zabbix 4.2 >Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	最低要求的 PHP 版本从 5.3.0 升级到 5.4.0 LogFile 必须指定代理参数
2.2.x LTS	适用于： Zabbix 2.4 Zabbix 3.0 Zabbix 3.2 Zabbix 3.4 Zabbix 4.0 >Zabbix 4.2 Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	删除了基于节点的分布式监控
2.0.x ·	对于： Zabbix 2.2 Zabbix 2.4 Zabbix 3.0 Zabbix 3.2 Zabbix 3.4 >Zabbix 4.0 Zabbix 4.2 Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	所需的最低 PHP 版本从 5.1.6 升级到 5.3.0； 正确的服务器工作需要大小写敏感的 MySQL 数据库；字符集 utf8 和 utf8_bin 排序规则是 Zabbix 服务器与 MySQL 数据库正常工作所必需的。请参阅数据库创建脚本。 需要 “mysqli”PHP 扩展而不是 “mysql”

您可能还想查看 6.0 的要求。

Note:
在升级期间运行两个并行的 SSH 会话可能很方便，在一个会话中执行升级步骤并在另一个会话中监视服务器/代理日志。例如，在第二个 SSH 会话中运行 “tail -f zabbix_server.log” 或 “tail -f zabbix_proxy.log”，实时显示最新的日志文件条目和可能的错误。这对于生产实例可能至关重要。

升级过程

1 停止 Zabbix 进程

停止 Zabbix 服务器以确保没有新数据插入到数据库。

·# systemctl stop zabbix-server

如果升级代理，也停止代理。

·#systemctl stop zabbix-proxy

Attention:
不再可能启动升级后的服务器并让旧的和未升级的代理向较新的服务器报告数据。Zabbix 从未推荐或支持的这种方法现在已被正式禁用，因为服务器将忽略来自未升级代理的数据。

2 备份已有的 Zabbix 数据库

这是非常重要的一步。确保您有数据库的备份。如果升级过程失败（磁盘空间不足、电源关闭、任何意外问题），这将有所帮助。

3 备份配置文件、PHP 文件和 Zabbix 二进制文件

制作 Zabbix 二进制文件、配置文件和 PHP 文件目录的备份副本。

配置文件：

·# mkdir /opt/zabbix-backup/ ·# cp /etc/zabbix/zabbix_server.conf /opt/zabbix-backup/ ·# cp /etc/httpd/conf.d/zabbix.conf · /opt/zabbix-backup/

PHP 文件和 Zabbix 二进制文件：

·# cp -R /usr/share/zabbix/ /opt/zabbix-backup/ ·# cp -R /usr/share/zabbix-* /opt/zabbix-backup/

4 更新仓库配置包

要继续升级，必须更新您当前的存储库包。

·# rpm -Uvh https://repo.zabbix.com/zabbix/6.0/rhel/8/x86_64/zabbix-release-6.0-1.el8.noarch.rpm

5 升级 Zabbix 组件

要升级 Zabbix 组件，您可以运行如下命令：

·# yum upgrade zabbix-server-mysql zabbix-web-mysql zabbix-agent

如果使用 PostgreSQL，请在命令中将 `mysql` 替换为 `pgsql`。如果升级代理，请在命令中将 `server` 替换为 `proxy`。如果升级 agent 2，请在命令中将 `zabbix-agent` 替换为 `zabbix-agent2`。

要使用 Apache **on RHEL 8** 正确升级 Web 前端，还要运行：

·# yum install zabbix-apache-conf

6 查看组件配置参数

有关强制更改 的详细信息，请参阅升级说明。

7 启动 Zabbix 进程

启动更新后的 Zabbix 组件。

·# systemctl start zabbix-server ·# systemctl start zabbix-proxy ·# systemctl start zabbix-agent ·# systemctl start zabbix-agent2

8 清除网络浏览器 cookie 和缓存

升级后，您可能需要清除网络浏览器 cookie 和网络浏览器缓存，以便 Zabbix web 界面正常工作。

小版本之间升级

可以在 6.0.x 的次要版本之间升级（例如，从 6.0.1 到 6.0.3）。次要版本之间的升级很容易。

要执行 Zabbix 次要版本升级，需要运行：

·$ sudo yum upgrade ’zabbix-*’

要执行 Zabbix 服务器次要版本升级运行：

·$ sudo yum upgrade ’zabbix-server-*’

要执行 Zabbix agent 次要版本升级运行：

·$ sudo yum upgrade ’zabbix-agent-*’

或者，对于 Zabbix 代理 2：

·$ sudo yum upgrade ’zabbix-agent2-*’

请注意，您也可以在这些命令中使用 “更新” 而不是 “升级”。虽然 “升级” 将删除过时的包，但 “更新” 将保留它们。

2 Debian/Ubuntu

概述

本章节介绍如何使用由官方发布的、基于 Debian/Ubuntu 系统的 Zabbix 二进制升级包，将当前应用的 Zabbix **5.4**.x 版本升级到最新的 Zabbix **6.0**.x 版本。

虽然 Zabbix 官方并未强制要求需要及时升级 Zabbix agents（推荐升级），但在使用 Zabbix 的过程中，需要保持 Zabbix server 和 proxies 保持相同的版本。因此，在服务器和代理服务器升级过程中，Zabbix Server 和所有的 proxies 都必须暂停进程以完成升级。在升级过程中不推荐坚持所有 Zabbix proxy 的运行，完成升级后所有的旧数据都会被摒弃。并且除非 proxy 的配置已完成同步操作，否则不会有新的数据上传。

请注意，若您在 proxy 端应用 SQLite 数据库，那么 proxy 升级之前所存储的数据将全部被清除。因为 SQLite 数据库的升级操作是不被支持的并且 SQLite 数据库文档需要被手动清除。当 proxy 第一次启动时，发现 SQLite 文档不存在，那么 proxy 会自动生成数据库文档。

6.0 版本升级的时间长短取决于数据库的总体大小。

请务必在升级前仔细阅读相关的升级文档！

提供以下升级说明:

当前版本	阅读完整的升级说明	不同版本间重要变化
5.4.x	适用于: Zabbix 6.0	提高了最低要求的数据库版本； 若数据库版本不满足要求，Server/proxy 将不会启动； 由于架构发生变化，旧的审计日志将会被摒弃。
5.2.x	适用于: Zabbix 5.4 Zabbix 6.0	提高了对数据库最低版本的要求； 所有的聚合监控项将被视为单独的监控项直接删除。
5.0.x LTS	适用于: Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	PHP 组件最低版本组件由 7.2.0 升到 7.2.5。
4.4.x	适用于: Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	放弃对 IBM DB2 的支持； PHP 组件最低版本组件由 5.4.0 升到 7.2.5； 提高了对数据库最低版本的要求； 改变了 Zabbix PHP 文档目录。
4.2.x	适用于: Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	Jabber, Ez Texting 媒体类型删除。
4.0.x LTS	适用于: Zabbix 4.2 Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	旧版本的 Zabbix proxies 将不会再向升级过的 server 发送数据； 新版本的 agent 不再能够与旧的 Zabbix server 一同运作。
3.4.x	适用于: Zabbix 4.0 Zabbix 4.2 Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	’libpthread’ 与’zlib’ 库强制应用； 取消对纯文本协议的支持，并且强制要求标头配置； 不再支持 1.4 版本之前的 Zabbix agents ； 若启用 proxy 的被动模式，那么 Server 参数是必须要配置的参数。
3.2.x	适用于: Zabbix 3.4 Zabbix 4.0 Zabbix 4.2 Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	SQLite 类型的数据库不再作为 Zabbix server/frontend 的支持类型； Perl Compatible Regular Expressions (PCRE) 兼容正则表达式，替换了 POSIX； Zabbix server 强制要求应用’libpcre’ 与’libevent’ 库； 为用户参数添加了退出代码检查、远程命令和不标注’nowait’ 标志的 system.run[] 监控项以及 Zabbix server 执行的脚本； 要求升级 Zabbix Java gateway 以支持新版本功能。

当前版本	阅读完整的升级说明	不同版本间重要变化
3.0.x LTS	适用于: Zabbix 3.2 Zabbix 3.4 Zabbix 4.0 Zabbix 4.2 Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	根据数据库历史表的大小，数据库升级的进程可能会很长。
2.4.x	适用于: Zabbix 3.0 Zabbix 3.2 Zabbix 3.4 Zabbix 4.0 Zabbix 4.2 Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	PHP 组件的最低版本要求由 5.3.0 升级到 5.4.0； LogFile agent 参数必须进行配置说明
2.2.x LTS	适用于: Zabbix 2.4 Zabbix 3.0 Zabbix 3.2 Zabbix 3.4 Zabbix 4.0 Zabbix 4.2 Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	移除了基于节点的分布式监控
2.0.x	适用于: Zabbix 2.2 Zabbix 2.4 Zabbix 3.0 Zabbix 3.2 Zabbix 3.4 Zabbix 4.0 Zabbix 4.2 Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	对 PHP 要求的最低版本从 5.1.6 升到 5.3.0； 为了支持 server 的正常运作，需要支持大小写识别的 MySQL 数据库；为了使 Zabbix server 正常运作，需要对 MySQL 数据库进行字符设定，需设定字符为 utf8 和 utf8_bin 。请查阅数据库创建脚本. PHP 扩展更改’mysql’ 为现在的’mysqli’。

您也可以参考如下 6.0 版本的安装需求 。

对于正在运行中的项目，用户可以考虑在对 Zabbix 进行升级时，同时开启两路 SSH 协议登录，一路用于运行软件的升级操作，一路用于监控 server/proxy 的 log 文档。举例如下：运行 `tail -f zabbix_server.log` 或者 `tail -f zabbix_proxy.log` 在第二路 SSH 连接，方便用户通过最新的 log 文档对升级过程进行实时监控，发现可能存在的问题或错误。

升级程序

1 停止 Zabbix 进程

用户需要停止 Zabbix server 服务，以确保没有新数据写入数据库。

```
# service zabbix-server stop
```

若需要升级 Zabbix proxy，同样需要先停止 Zabbix proxy 进程。

```
# service zabbix-proxy stop
```

2 备份当前的数据库

请用户确认，在升级前备份了数据库，这是非常关键的一步。如果升级失败（因磁盘空间不足、断电或其他意外导致的升级失败），备份的数据库将大有帮助。

3 备份配置文件、PHP 文件和 Zabbix 二进制文件

请用户在升级前确认备份了 Zabbix 二进制文件、配置文件和 PHP 文件。

配置文件：

```
 # mkdir /opt/zabbix-backup/
 # cp /etc/zabbix/zabbix_server.conf /opt/zabbix-backup/
 # cp /etc/apache2/conf-enabled/zabbix.conf /opt/zabbix-backup/
```

PHP 文件和 Zabbix 二进制文件：

```
 # cp -R /usr/share/zabbix/ /opt/zabbix-backup/
 # cp -R /usr/share/doc/zabbix-* /opt/zabbix-backup/
```

4 更新存储配置包

要继续更新，必须卸载当前的存储库包。

```
# rm -Rf /etc/apt/sources.list.d/zabbix.list
```

然后安装新的存储库配置包。

在 **Debian 11** 运行:

```
# wget https://repo.zabbix.com/zabbix/6.0/debian/pool/main/z/zabbix-release/zabbix-release_6.0-1+debian11_
# dpkg -i zabbix-release_6.0-1+debian11_all.deb
```

在 **Debian 10** 运行:

```
# wget https://repo.zabbix.com/zabbix/6.0/debian/pool/main/z/zabbix-release/zabbix-release_6.0-1+debian10_
# dpkg -i zabbix-release_6.0-1+debian10_all.deb
```

在 **Debian 9** 运行:

```
# wget https://repo.zabbix.com/zabbix/6.0/debian/pool/main/z/zabbix-release/zabbix-release_6.0-1+debian9_a
# dpkg -i zabbix-release_6.0-1+debian9_all.deb
```

在 **Ubuntu 20.04** 运行:

```
# wget https://repo.zabbix.com/zabbix/6.0/ubuntu/pool/main/z/zabbix-release/zabbix-release_6.0-1+ubuntu20.
# dpkg -i zabbix-release_6.0-1+ubuntu20.04_all.deb
```

在 **Ubuntu 18.04** 运行:

```
# wget https://repo.zabbix.com/zabbix/6.0/ubuntu/pool/main/z/zabbix-release/zabbix-release_6.0-1+ubuntu18.
# dpkg -i zabbix-release_6.0-1+ubuntu18.04_all.deb
```

在 **Ubuntu 16.04** 运行:

```
# wget https://repo.zabbix.com/zabbix/6.0/ubuntu/pool/main/z/zabbix-release/zabbix-release_6.0-1+ubuntu16.
# dpkg -i zabbix-release_6.0-1+ubuntu16.04_all.deb
```

在 **Ubuntu 14.04** 运行:

```
# wget https://repo.zabbix.com/zabbix/6.0/ubuntu/pool/main/z/zabbix-release/zabbix-release_6.0-1+ubuntu14.
# dpkg -i zabbix-release_6.0-1+ubuntu14.04_all.deb
```

更新存储库信息

```
# apt-get update
```

5 升级 Zabbix 组件

升级 Zabbix 组件，可以运行以下命令：

```
  # apt-get install --only-upgrade zabbix-server-mysql zabbix-frontend-php zabbix-agent
```

若使用 PostgreSQL 数据库，请在命令中将 `mysql` 替换为 `pgsql`。若升级 proxy，请在命令中将 `server` 替换为 `proxy`。若升级 Zabbix agent 2，在命令中将 `zabbix-agent` 替换为 `zabbix-agent2`。

与此同时，要使得 Apache 能正常升级 Web 前端，还需运行如下命令：

```
    # apt-get install zabbix-apache-conf
```

发行版 **prior to Debian 10 (buster) / Ubuntu 18.04 (bionic) / Raspbian 10 (buster)** 不提供 PHP 7.2 或更高版本，而其对 Zabbix 前端 5.0 又是必要的。有关安装 Zabbix 前端旧发行版的信息，请查阅information。

6 检查 Zabbix 组件配置文件的参数

在新版本中，Zabbix 组件的配置文件发生了一些变化，详见升级说明mandatory changes。

关于新的选项参数，详见此章节What's new 。

7 启动 Zabbix 进程

启动升级后的 Zabbix 组件。

```
# service zabbix-server start
# service zabbix-proxy start
# service zabbix-agent start
# service zabbix-agent2 start
```

8 清除浏览器的 Cookies 和缓存

待升级完毕后，可能需要清除浏览器的 Cookies 和缓存，以便 Zabbix 的 Web 界面能正常工作。

Zabbix 次要版本之间的升级如果要升级 Zabbix 6.0.x 的次要版本（例如：从 6.0.1 升级到 6.0.3），是非常容易的。

在升级 Zabbix 所有组件的次要版本时，只需运行以下命令：

```
$ sudo apt install --only-upgrade 'zabbix.*'
```

在升级 Zabbix server 的次要版本时，只需运行以下命令：

```
$ sudo apt install --only-upgrade 'zabbix-server.*'
```

在升级 Zabbix agent 的次要版本时，只需运行以下命令：

```
$ sudo apt install --only-upgrade 'zabbix-agent.*'
```

在升级 Zabbix agent 2 的次要版本时，只需运行以下命令：

```
$ sudo apt install --only-upgrade 'zabbix-agent2.*'
```

从源码包升级

概述

本节提供使用官方 Zabbix 源码包从 Zabbix **5.4**.x 升级 到 Zabbix **6.0**.x 所需的步骤。

尽管升级 Zabbix agent 不是强制性的（但推荐），但 Zabbix server 和 proxy 必须是相同的主版本。因此，在 server-proxy 架构中，Zabbix server 和所有的 proxy 都必须停止并升级。保持 proxy 运行没有任何用处，因为在 proxy 升级期间，它们的旧数据将被丢弃，并且在 proxy 配置与 server 同步之前不会收集新数据。

> **Attention:**
> 使用较新的升级的 server 并让较旧但未升级的 proxy 向其发送数据不可能再成功启动。Zabbix 从未推荐或支持这种方法，现在已正式禁用，因为 server 将忽略来自未升级 proxy 的数据。

请注意，proxy 如果使用 SQLite 数据库，升级前来自 proxy 的历史数据将丢失，因为不支持 SQLite 数据库表结构升级并且必须手动删除 SQLite 数据库文件。初次启动 proxy 时如果缺少 SQLite 数据库文件，proxy 会自动创建它。

根据数据库大小，数据库升级到版本 6.0 可能需要很长时间。

> **Warning:**
> 升级前请务必阅读相关 升级说明！

提供以下升级说明：

从 X 版本升级	阅读完整的升级说明	版本之间最重要的变化
5.4.x ·	适用于： Zabbix 6.0	提高了最低要求的数据库版本； 如果数据库版本过低，Server/proxy 将无法启动； 由于数据库结构更改，审计日志记录丢失。
5.2.x ·	适用于： Zabbix 5.4 Zabbix 6.0	提高了最低要求的数据库版本； Aggregate 监控项作为单独类型将删除。

从 X 版本升级	阅读完整的升级说明	版本之间最重要的变化
5.0.x LTS	适用于： Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	最低要求的 PHP 版本从 7.2.0 提高到 7.2.5。
4.4.x ·	适用于： Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	不再支持 IBM DB2； 最低要求的 PHP 版本从 5.4.0 提高到 7.2.0； 提高了最低要求的数据库版本； 更改了 Zabbix PHP 文件目录。
4.2.x ·	适用于： Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	删除了 Jabber、Ez 短信媒体类型。
4.0.x LTS	适用于： Zabbix 4.2 Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	旧版本的 proxy 不再可以向升级的 server 报告数据； 高版本的 agent 将不再能够与较低版本的 Zabbix server 匹配。
3.4.x ·	适用于： Zabbix 4.0 Zabbix 4.2 Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	强制使用’libpthread’ 和’zlib’ 库； 对纯文本协议的支持被丢弃，并且头部是强制性的； 1.4 以前版本的 Zabbix agent 将不再支持； 被动 proxy 配置中的 Server 参数现在是必需的。
3.2.x ·	适用于： Zabbix 3.4 Zabbix 4.0 Zabbix 4.2 Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	Zabbix server/前端支持 SQLite 作为数据库; 支持 Perl 兼容正则表达式 (PCRE) 而不是 POSIX 扩展； Zabbix server 必须使用’libpcre’ 和’libevent’ 库； 为不带’nowait’ 标志的用户参数、远程命令和 system.run[] 监控项以及 Zabbix server 执行脚本添加了退出代码检查； Zabbix Java gateway 必须升级以支持新功能。
3.0.x LTS	适用于： Zabbix 3.2 Zabbix 3.4 Zabbix 4.0 Zabbix 4.2 Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	数据库升级可能会很慢，具体取决于历史表的大小。
2.4.x ·	适用于： Zabbix 3.0 Zabbix 3.2 Zabbix 3.4 Zabbix 4.0 Zabbix 4.2 Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	最低要求的 PHP 版本从 5.3.0 升级到 5.4.0； agent 配置文件的的 LogFile 参数必须指定。

从 X 版本升级	阅读完整的升级说明	版本之间最重要的变化
2.2.x LTS	适用于： Zabbix 2.4 Zabbix 3.0 Zabbix 3.2 Zabbix 3.4 Zabbix 4.0 Zabbix 4.2 Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	移除了基于节点的分布式监控。
2.0.x ·	适用于： Zabbix 2.2 Zabbix 2.4 Zabbix 3.0 Zabbix 3.2 Zabbix 3.4 Zabbix 4.0 Zabbix 4.2 Zabbix 4.4 Zabbix 5.0 Zabbix 5.2 Zabbix 5.4 Zabbix 6.0	最低要求的 PHP 版本从 5.1.6 提高到 5.3.0； server 正常工作需要区分大小写的 MySQL 数据库；Zabbix server 需要 utf8 和 utf8_bin 字符集排序规则才能与 MySQL 数据库正常工作。请参阅数据库创建脚本。 'mysqli' 需要 PHP 扩展而不是'mysql'。

你可能还想参考 6.0 的其他需求 。

Note:
在升级期间运行两个并行 SSH 会话可能会很方便，其中一个执行升级步骤，另一个监视 server/proxy 日志。例如，在第二个 SSH 会话中运行 `tail -f zabbix_server.log` 或 `tail -f zabbix_proxy.log` 实时显示最新的日志文件条目和可能的错误。这对于生产实例可能很关键。

Server 升级步骤

1 停止 Server 服务

停止 Zabbix server 服务并确认不再有新的数据写入数据库。

2 备份当前的 Zabbix 数据库

这是非常重要的一步。确保您有数据库的备份，如果升级过程失败（如磁盘空间不足、断电等任何意外问题），它将有所帮助。

3 备份配置文件、PHP 文件和 Zabbix 程序文件

备份 Zabbix 程序文件、配置文件和 PHP 文件目录。

4 安装新的 server 程序文件

查看从源代码编译 Zabbix Server 的说明 。

5 查看 server 配置参数

有关强制更改 的详细信息，请参阅升级说明。

有关新增的可选参数，请参阅6.0 新变化 。

6 启动新的 Zabbix 程序文件

启动新的程序文件。检查日志以确认程序文件是否成功启动。

Zabbix server 会自动升级数据库。服务启动时，Zabbix server 会报告当前的（包括强制和可选）和所需要的数据库版本。如果当前强制版本比要求的版本低，Zabbix Server 会自动执行所要求的数据库版本升级补丁。数据库升级进度（百分比）被写入 Zabbix server 的日志。当日志出现 "database upgrade fully completed" 表示数据库升级成功。如果有任何补丁升级失败，Zabbix server 将不会启动。如果当前强制数据库版本比要求的更新，Zabbix server 也不会启动。仅当当前强制数据库版本对应于所需数据库版本时，Zabbix server 才会启动。

8673:20161117:104750.259 current database version (mandatory/optional): 03040000/03040000 8673:20161117:104750.259 required mandatory version: 03040000

在启动 server 服务之前：

- 确保数据库用户有足够的权限（创建表、删除表、创建索引、删除索引）；
- 确保您有足够的可用磁盘空间。

7 安装新的 Zabbix web 界面

要求的最低 PHP 版本是 7.2.5。可参阅安装说明。

8 清除浏览器 cookie 和缓存

升级后，您可能需要清除 Web 浏览器的 cookie 和缓存信息，以使 Zabbix Web 界面正常工作。

Proxy 升级步骤

1 停止 proxy 服务

停止 proxy 服务

2 备份配置文件和 Zabbix Proxy 旧版本程序

备份配置文件和 Zabbix Proxy 旧版本程序。

3 安装新的 proxy 程序

参考从源码编译安装 Zabbix proxy 的说明 。

4 查看 proxy 配置参数

此版本没有对 proxy 的参数 做强制性变更。

5 启动新的 Zabbix proxy

启动新的 Zabbix proxy 服务。检查日志以确认是否升级成功。

Zabbix proxy 会自动升级数据库。数据库升级与启动Zabbix server 服务类似。

Agent 升级步骤

> **Attention:**
> 升级 agent 不是强制性的。只有在需要使用新功能时才需要升级 agent。

本节的升级过程适用于升级 Zabbix agent 和 Zabbix agent2。

1 停止 agent 服务

停掉 Zabbix agent 服务。

2 备份配置文件和 Zabbix agent 程序文件

备份配置文件和 Zabbix agent 程序文件。

3 安装新的 agent 程序文件

参考源码编译安装 Zabbix agent 的说明。

4 查看 agent 配置参数

在此版本中，agent 和agent2 的参数都没有强制更改部分。

5 启动新的 Zabbix agent

启动新的 Zabbix agent 服务。检查日志以确认启动成功。

次版本之间的升级

升级 6.0.x 的次版本（例如从 6.0.1 到 6.0.3）时，需要对 server/proxy/agent 执行与主版本升级步骤相同的操作。唯一的区别是，在次版本之间进行升级时，不会对数据库进行任何更改。

另请参阅：编译问题。

使用 MySQL 8.0.0-8.0.17 启动 Proxy

zabbix_proxy 在 8.0.0-8.0.17 版本的 MySQL 上启动失败并报错"access denied" ：

```
[Z3001]  connection to database 'zabbix' failed: [1227] Access denied; you need (at least one of) the SUPE
```

这是由于 MySQL 8.0.0 版本开始强制设置会话变量的特殊权限。然而，在 8.0.18 中删除了此行为：从 MySQL 8.0.18 开始，不再限制设置系统会话变量的操作

解决方法是向 `zabbix` 用户授予额外权限：

对于 MySQL 8.0.14 - 8.0.17: grant SESSION_VARIABLES_ADMIN on . to 'zabbix'@'localhost';

对于 MySQL 8.0.0 - 8.0.13: grant SYSTEM_VARIABLES_ADMIN on . to 'zabbix'@'localhost';

Timescale DB：大量分区导致高内存使用

PostgreSQL 版本 9.6-12 在更新具有大量分区的表时会使用过多的内存（查看问题报告）。当在具有 TimescaleDB 的系统上更新趋势时，如果趋势被分割成相对较小的（例如 1 天）块，这个问题就会显现出来。这导致在默认的管家设置下，趋势表中存在数百个块 - 这是 PostgreSQL 可能耗尽内存的情况。

这个问题自 Zabbix 5.0.1 起对于新的 TimescaleDB 安装已经解决，但如果 TimescaleDB 是在 Zabbix 之前设置的，请参见 ZBX-16347 查看迁移说明。

Timescale DB 2.5.0：压缩策略可能在包含整数的表上失败

当使用 TimescaleDB 2.5.0 时，这个问题会显现出来。自 TimescaleDB 2.5.1 起已经解决。

更多信息，请参见 TimescaleDB Issue #3773。

升级

升级成功的 SQL 模式设置

在 MySQL/MariaDB 中，必须设置"STRICT_TRANS_TABLES" 模式的 `sql_mode` 设置。如果缺少此设置，Zabbix 数据库升级将失败（另请参阅 ZBX-19435）。

从 MariaDB 10.2.1 以及之前的版本升级

如果数据库表是使用 MariaDB 10.2.1 及之前的版本创建的，升级 Zabbix 可能会失败，因为在这些版本中，默认的 ROW_FORMAT（即行格式，是指数据的记录即数据行在磁盘中的物理存储方式）是 compact。这个问题可以通过将 ROW_FORMAT 更改为 dynamic 来解决（另请参见 ZBX-17690）。

模板

双栈（IPv4/IPv6）环境中的模板兼容性

在双栈环境中（系统配置为支持 IPv4 和 IPv6），主机名 `localhost` 通常解析为 IPv4 和 IPv6 地址。由于许多操作系统和 DNS 解析器普遍优先考虑 IPv6 而不是 IPv4，如果被监控的服务仅配置为在 IPv4 上监听，Zabbix 模板可能无法正确工作。

未配置为在 IPv6 地址上监听的服务可能变得无法访问，导致监控失败。用户可能正确配置了 IPv4 的访问，但由于默认优先考虑 IPv6 的行为，仍然面临连接问题。

解决这个问题的方法是确保服务（Nginx、Apache、PostgreSQL 等）配置为同时监听 IPv4 和 IPv6 地址，并且 Zabbix server/agent 被允许通过 IPv6 访问。此外，在 Zabbix 模板和配置中，明确使用 `localhost` 而不是 `127.0.0.1` 以确保与 IPv4 和 IPv6 的兼容性。

例如，当使用 Zabbix agent 2 监控 PostgreSQL 模板时，您可能需要编辑 `pg_hba.conf` 文件以允许 `zbx_monitor` 用户的连接。如果双栈环境优先考虑 IPv6（系统将 localhost 解析为 `::1`），并且您配置了 `localhost` 但只添加了一个 IPv4 条目（`127.0.0.1/32`），连接将失败，因为没有匹配的 IPv6 条目。

以下 `pg_hba.conf` 文件示例确保 `zbx_monitor` 用户可以使用不同的认证方法，从本地机器通过 IPv4 和 IPv6 地址连接到任何数据库：

```
# TYPE      DATABASE      USER             ADDRESS          METHOD
  host      all           zbx_monitor      localhost        trust
  host      all           zbx_monitor      127.0.0.1/32     md5
  host      all           zbx_monitor      ::1/128          scram-sha-256
```

如果需要，您也可以在配置 Zabbix agent 2 监控 PostgreSQL 模板的连接字符串时，直接使用 IPv4 地址（`127.0.0.1`）。

意外安装 EPEL Zabbix 包的处理方法

如果安装并启用了 EPEL 仓库，并且从包中安装 Zabbix，那么将导致安装的是 EPEL Zabbix 包而不是官方 Zabbix 包。

在这种情况下，需要卸载 EPEL 中的 Zabbix 包，例如：

```
dnf remove zabbix-server-mysql
```

阻止从 EPEL 安装 Zabbix 包。在 `/etc/yum.conf` 文件中添加以下行：

```
exclude=zabbix6.0*
```

重新安装 Zabbix server：

```
dnf install zabbix-server-mysql
```

请注意，官方 Zabbix 包的版本字符串中包含单词 `release`：

```
6.0.25-release1.el8
```

适用于 Red Hat UBI 环境的 RHEL Zabbix 包

在 Red Hat Universal Base Image 环境中从 Red Hat Enterprise Linux (RHEL) 包安装 Zabbix 时，请确保可以访问所需的仓库和依赖项。Zabbix 包依赖于 `libOpenIPMI.so` 和 `libOpenIPMIposix.so` 库，这些库在 UBI 系统上启用的默认包管理器仓库中没有提供，这将导致安装失败。

`libOpenIPMI.so` 和 `libOpenIPMIposix.so` 库包含在 `OpenIPMI-libs` 包中，该包由 `redhat-#-for-<arch>-appstream-rpms` 仓库提供。对该仓库的访问由订阅进行管理，在 UBI 环境中，订阅通过将 RHEL 主机的仓库配置和密钥目录挂载到容器文件系统命名空间中来传播。

更多信息，请参阅 ZBX-24291。

Expired signing key for RHEL packages

When upgrading Zabbix on Red Hat Enterprise Linux, you may encounter an expired signing key issue for packages on Zabbix repository. When a signing key expires, attempts to verify package signatures will result in an error indicating that the certificate or key is no longer valid. For example:

```
error: Verifying a signature using certificate D9AA84C2B617479C6E4FCF4D19F2475308EFA7DD (Zabbix LLC (Jul 2
  1. Certificiate 19F2475308EFA7DD invalid: certificate is not alive
      because: The primary key is not live
      because: Expired on 2024-07-04T11:41:23Z
  2. Key 19F2475308EFA7DD invalid: key is not alive
      because: The primary key is not live
      because: Expired on 2024-07-04T11:41:23Z
```

To resolve such issues, manually reinstall the latest `zabbix-release` package for your specific variant of RHEL (replace the link below with the correct one from Zabbix repository).

For example, on **RHEL 9**, run:

```
rpm -Uvh https://repo.zabbix.com/zabbix/6.0/rhel/9/x86_64/zabbix-release-latest.el9.noarch.rpm
```

Then, update the repository information:

```
dnf update
```

For more information, see ZBX-24761.

与 MariaDB 的数据库 TLS 连接

如果使用 MariaDB 数据库，则 DBTLSConnect 参数 的 *verify_ca* 选项不支持数据库 TLS 连接。

MySQL/MariaDB 可能出现的死锁

当在高负载下运行，并且涉及多个 LLD worker 时，可能会遇到由与行锁定策略相关的 InnoDB 错误（参见 上游错误）。从 8.0.29 开始，该错误在 MySQL 中已修复，但在 MariaDB 中未修复。有关详细信息，请参阅 ZBX-21506。

全局事件关联

如果第一个事件和第二个事件之间的时间间隔非常小，即半秒或更短，事件可能无法正确关联。

PostgreSQL 11 及更早版本的数据库支持的浮点数类型范围

PostgreSQL 11 及更早版本仅支持大约 -1.34E-154 到 1.34E+154 的浮点数范围。

NetBSD 8.0 及更新版本

在 NetBSD 8.X 和 9.X 上，Zabbix 各种进程可能会在启动时随机崩溃。这是由于默认堆栈大小（4MB）太小，需要执行以下命令来增加：

`ulimit -s 10240`

点击 ZBX-18275 查看相关问题报告。

Zabbix agent 2 中的正则表达式限制

由于标准的 Go regexp 库的限制，Zabbix agent 2 不支持正向预查和反向预查功能。

IPMI 检查

用 Debian 9 (stretch) 和 Ubuntu 16.04 (xenial) 之前版本的标准 OpenIPMI 库包运行 IPMI 检查会无法正常运行。要解决此问题，请重新编译 OpenIPMI 库并启用 OpenSSL，参考 ZBX-6139。

SSH 检查

- 如果 libssh2 库是从软件包安装的，一些 Linux 发行版如 Debian、Ubuntu 不支持加密私钥 (带密码)。有关详细信息，请参阅 ZBX-4850。

- 在某些带有 OpenSSH 8 的 Linux 发行版上使用 libssh 0.9.x 时，SSH 检查可能偶尔会报告 “无法从 SSH 服务器读取数据”。这是由 libssh 问题 引起的（更详细的报告）. 该错误预计已由稳定的 libssh 0.9.5 版本修复。有关详细信息，另请参阅 ZBX-17756。

- 使用管道 “|” 在 SSH 脚本中可能会导致 “无法从 SSH 服务器读取数据” 错误。在这种情况下，建议升级 libssh 库版本。有关详细信息，另请参阅 ZBX-21337。

ODBC 检查

- MySQL unixODBC 驱动程序不应与 Zabbix server 或针对 MariaDB 连接器库编译的 Zabbix proxy 一起使用，反之亦然，如果可能，最好也避免使用与驱动程序相同的连接器，因为存在一个 上游错误。建议的设置：

  PostgreSQL、SQLite 或 Oracle 连接器 → MariaDB 或 MySQL unixODBC 驱动程序 MariaDB 连接器 → MariaDB unixODBC 驱动程序 MySQL 连接器 → MySQL unixODBC 驱动程序

  查看 ZBX-7665 获取更多信息和可用的解决方法。

- 从 Microsoft SQL Server 查询的 XML 数据可能在 Linux 和 UNIX 系统上以各种方式被截断。

- 观察到使用 ODBC 检查通过 Linux 的 Oracle Instant Client 的不同版本监控 Oracle 数据库会导致 Zabbix server 崩溃。另请参阅：ZBX-18402，ZBX-20803。

- 如果使用 FreeTDS UnixODBC 驱动程序，您需要在 SQL 查询前加上’SET NOCOUNT ON’ 语句（例如，`sql SET NOCOUNT ON DECLARE @strsql NVARCHAR(max) SET @strsql = ....`）。否则，Zabbix 中的数据库监控项将无法检索到信息，并出现错误”SQL 查询返回空结果”。查看 ZBX-19917 获取更多信息。

监控项中的请求方法参数不正确

在 HTTP 检查中使用的 request_method 参数可能被错误地设置为 “1”，监控项的非默认值是由于从 Zabbix 4.0 之前的版本升级造成的。点击 ZBX-19308 查看解决方法。

Web 监控和 HTTP agent

由于 上游 bug，在 Web 场景或 HTTP agent 中启用 “SSL verify peer” 时，Zabbix server 在 CentOS 6、CentOS 7 或其他 Linux 发行版上可能存在内存泄漏（leaks memory）的问题。参考 ZBX-10486 获取更多信息和解决方案。

简单检查

在 v3.10 之前的 **fping** 版本中存在错误处理重复的回显重放数据包的 bug。可能会导致 `icmpping`, `icmppingloss`, `icmppingsec` 监控项故障。建议使用最新版本的 **fping**。参考 ZBX-11726 。

在无根容器中执行 fping 时出现的错误

当容器以无根模式或在受限环境中运行时，执行 ICMP 检查时可能会遇到与 fping 执行相关的错误，例如 `fping: Operation not permitted` 或所有资源的所有数据包丢失。

要解决这个问题，请将 `--cap-add=net_raw` 添加到”docker run” 或”podman run” 命令中。

此外，在非根环境中执行 fping 可能需要修改 sysctl，例如：

```
sudo sysctl -w "net.ipv4.ping_group_range=0 1995"
```

这里的”1995” 是 zabbix GID。有关更多详细信息，请参阅 ZBX-22833。

SNMP 检查

对于 OpenBSD 操作系统，如果在 Zabbix server 配置文件中设置了 SourceIP 参数，则 5.7.3（及之前）版本中 Net-SNMP 库的 use-after-free bug 会导致 Zabbix server 崩溃。其中一种解决办法是不设置 SourceIP 参数。同样的问题也存在于 Linux，但它不会导致 Zabbix server 停止工作。OpenBSD 上的 net-snmp 软件包的本地补丁已启用，并将与 OpenBSD 6.3 一起发布。

SNMP 数据峰值

SNMP 监控数据中的峰值可能与某些物理因素有关，如电源中的电压峰值。详情点击 ZBX-14318。

SNMP trap

SNMP trap 所需的 “net-snmp-perl” 软件包已在 RHEL/CentOS 8.0-8.2 中删除，在 RHEL 8.3 中重新添加。

所以如果你使用的是 RHEL 8.0-8.2，最好的解决方案是升级到 RHEL 8.3；如果你使用的是 CentOS 8.0-8.2，您可以等待 CentOS 8.3 或使用 EPEL 源提供的软件包。

详情参阅 ZBX-17192 。

Alerter 进程在 RHEL 7 中崩溃

在 RHEL 7. 中遇到了 Zabbix server 的 alerter 进程崩溃的情况。有关详细信息，请参阅 ZBX-10461 。

升级 Zabbix Agent 2（6.0.5 或更旧版本）

在升级 Zabbix Agent 2（版本为 6.0.5 或更旧版本）时，可能会出现与插件相关的文件冲突错误。要解决此错误，请备份您的 Agent 2 配置（如果需要），然后卸载 Agent 2 并重新安装。

在基于 RHEL 的系统上运行以下命令：

```
dnf remove zabbix-agent2
dnf install zabbix-agent2
```

在基于 Debian 的系统上运行以下命令：

```
apt remove zabbix-agent2
apt install zabbix-agent2
```

有关更多信息，请参阅 ZBX-23250。

前端区域信息错乱

已经观察到，前端当地区域值可能会在没有明显逻辑的情况下错乱，例如：某些页面（或部分页面）以一种语言显示，其他页面（或部分页面）显示另一种语言。常出现在一些用户使用一个语言环境，而其他用户使用另一个语言环境的情况。

已知的解决方法是在 PHP 和 Apache 中禁用多线程。问题与如何 [在 PHP 中] (https://www.php.net/manualen/function.setlocale) 设置语言环境有关：语言环境信息是按进程维护的，而不是按线程维护的。因此，在多线程环境中，当有多个监控项由同一个 Apache 进程运行时，可能会在另一个线程中更改语言环境，从而改变 Zabbix 线程中处理数据的方式。

更多信息，详见相关问题报告: - ZBX-10911 (Problem with flipping frontend locales) - ZBX-16297 (Problem with number processing in graphs using the `bcdiv` function of BC Math functions)

PHP 7.3 opcache 配置

如果在 PHP 7.3 配置中启用了 “opcache”，第一次加载时 Zabbix 前端可能会显示一个空白屏幕。这是一个已注册的 PHP bug。要解决此问题，请在 PHP 配置（php.ini 文件）中将”opcache.optimization_level” 参数设置为 0x7FFFBFDF 。

图表

夏令时

夏令时（DST）的变更在显示 X 轴标签时会导致不规律现象（日期重复、日期缺失等）。

求和聚合

当在图表中使用求和聚合 对于不到一小时的时间段，如果数据来自趋势，图表会显示不正确（放大）的值。

文本重叠

对于一些前端语言（例如，日语），本地字体可能会导致图表图例中的文本重叠。为了避免这种情况，请使用 PHP GD 扩展的 2.3.0 版本(或更高版本)。

日志文件监控

如果文件系统达到 100% 并且日志正在追加，则 `log[]` 和 `logrt[]` 监控项会从头重读日志文件，(参阅 ZBX-10884 ）获取更多。

MySQL 查询缓慢

Zabbix server 在监控项的值不存在时会生成缓慢的 SELECT 查询。已知此问题会发生在 MySQL 5.6/5.7 版本中（有关更深入的讨论，请参阅ZBX-10652），并且在特定情况下，也可能出现在更高版本的 MySQL 中。解决这个问题的一种方法是在 MySQL 中禁用`index_condition_pushdown` 或`prefer_ordering_index` 优化器。但请注意，这种解决方法可能无法解决所有与慢查询相关的问题。

Oracle 数据库配置同步缓慢

在 Oracle 数据库中拥有大量监控项和预处理步骤的 Zabbix 6.0 安装中，配置同步可能会比较慢。这是由于 Oracle 数据库引擎处理 *nclob* 类型字段的速度较慢所致。

为了提高性能，您可以通过手动应用数据库补丁items_nvarchar_prepare.sql，将字段类型从 *nclob* 转换为 *nvarchar2*。请注意，此转换会将预处理参数和监控项参数（例如描述、脚本监控项的字段脚本、HTTP agent 监控项的字段请求正文和头部、数据库监控项的字段 *SQL* 查询）的最大字段大小限制从 65535 字节减少到 4000 字节。补丁中提供了查询，以确定在应用补丁之前需要删除的模板名称，作为注释提供。或者，如果设置了 MAX_STRING_SIZE，您可以在补丁查询中将 *nvarchar2(4000)* 更改为 *nvarchar2(32767)*，以设置 32767 字节的字段大小限制。

有关更深入的讨论，请参阅ZBX-22363。

API 登录

使用自定义脚本方式 进行 `user.login` 登录而不执行 `user.logout`，会创建大量打开的用户会话。

Persistent filter settings from links

When opening a link to Zabbix frontend page that contains filter settings, including the time selector, the filter is automatically saved in the database for the user, replacing the previously saved filter and/or time selector settings for that page. These settings remain active until the user manually updates or resets them.

SNMPv3 trap 中的 IPv6 地址问题

由于 net-snmp bug，在 SNMP trap 中使用 SNMPv3 时可能无法正确显示 IPv6 地址。有关更多详细信息和可能的解决方法，请参阅 ZBX-14541。

裁剪了登录失败信息中的长 IPv6 IP 地址

登录失败的消息将仅显示存储的 IP 地址的前 39 个字符，这是由于数据库字段中的字符限制。这意味着超过 39 个字符的 IPv6 IP 地址将不能完整显示。

Windows 的 Zabbix agent

Zabbix agent 配置文件 (zabbix_agentd.conf) 中设置非 DNS 性质的 `Server` 参数可能会增加 Windows 上 Zabbix agent 的响应时间。发生这种情况是因为 Windows DNS 缓存守护程序不会缓存 IPv4 地址的否定响应。但是会缓存 IPv6 地址的否定响应，因此可能的解决方法是在主机上禁用 IPv4。

YAML 导出/导入

YAML 导出/导入 存在一些已知问题：

- 错误消息不会被翻译；
- 有时会无法导入带有.yaml 文件扩展名的有效 JSON 文件；
- 日期如果不加引号会自动转换为 Unix 时间戳。

使用 NGINX 和 php-fpm 在 SUSE 上设置向导

在 SUSE 上使用 NGINX + php-fpm，前端设置向导将无法保存配置文件。这是由 /usr/lib/systemd/system/php-fpm.service 单元中的设置引起的，该设置阻止 Zabbix 写入 /etc。(在 PHP 7.4 中引入).。

有两种解决方法可供选择：

- 在 php-fpm systemd 单元中将 ProtectSystem 选项设置为'true' 而不是'full'。
- 手动保存 /etc/zabbix/web/zabbix.conf.php 文件。

在 Ubuntu 20 上使用 Chromium 运行 Zabbix web

尽管在大多数情况下，Zabbix Web 服务可以使用 Chromium 运行，但在 Ubuntu 20.04 上使用 Chromium 会导致以下错误：

```
Cannot fetch data: chrome failed to start:cmd_run.go:994:
WARNING: cannot create user data directory: cannot create "/var/lib/zabbix/snap/chromium/1564": mkdir /var
Sorry, home directories outside of /home are not currently supported. See https://forum.snapcraft.io/t/112
```

发生此错误是因为 `/var/lib/zabbix` 已经被用作用户'zabbix' 的主目录。

MySQL 自定义错误代码

如果在 Azure 上安装 Zabbix 和 MySQL，Zabbix 日志中可能会出现不明确的报错信息 *[9002] Some errors occurred* 。这种通用错误文本由数据库发送到 Zabbix server 或 proxy。若要获取有关错误原因的详细信息，请查看 Azure 日志。

切换到 PCRE2 后正则表达式无效

在 Zabbix 6.0 中增加了对 PCRE2 的支持。尽管 PCRE 仍然得到支持，但针对 RHEL 7 及更新版本、SLES（所有版本）、Debian 9 及更新版本、Ubuntu 16.04 及更新版本的 Zabbix 安装包已更新为使用 PCRE2。虽然带来了许多好处，但切换到 PCRE2 可能会导致某些现有的 PCRE 正则表达式模式变得无效或行为不同。特别是，这影响了模式 `*^[\\w-\\.]*`。为了在不影响语义的情况下使这个正则表达式再次有效，将表达式更改为 `*^[-\\w\\.]*`。这是因为 PCRE2 将连字符视为分隔符，在字符类内部创建范围。

以下 Zabbix 安装包已更新，现在使用 PCRE2：RHEL 7 及更新版本、SLES（所有版本）、Debian 9 及更新版本、Ubuntu 16.04 及更新版本。

Zabbix 6.0.0-6.0.2 中服务转换错误

在 Zabbix 6.0 中，引入了新的、更灵活的服务状态计算算法。

从 Zabbix 6.0 之前的版本升级到 Zabbix 6.0.0、6.0.1 或 6.0.2 后，“如果所有子服务都有问题则最严重” 和 “子服务中最严重的” 服务状态计算规则将会互换。在 Zabbix 6.0.0 及更新版本中创建的服务将具有正确的状态计算规则。

当从 6.0 之前的版本升级到 Zabbix 6.0.3 或更新版本时，Zabbix 将正确更新服务状态计算规则。从 6.0.x 升级到 6.0.3 对服务状态计算规则没有影响。

地图小部件错误

如果你使用较老的 Zabbix 版本（LNMP 结构）升级并且在升级期间未切换到新的 NGINX 配置文件，则 Geomap 小部件中的地图可能会出现无法正确加载的情况。

要解决此问题，可以丢弃旧的配置文件，使用 6.0 包中的配置文件并参照 下载说明 中的 *e.* 部分为 *Zabbix* 前端配置 *PHP* 的描述重新配置它。

或者，你可以手动编辑现有的 NGINX 配置文件（通常是 */etc/zabbix/nginx.conf*）。打开文件并找到以下配置模块：

```
location ~ /(api\/|conf[^\.]|include|locale|vendor) {
        deny            all;
        return          404;
}
```

然后，将此块替换为：

```
location ~ /(api\/|conf[^\.]|include|locale) {
        deny            all;
        return          404;
}

location /vendor {
        deny            all;
        return          404;
}
```

Incorrect information from nested host groups in maps

Information from nested host groups is incorrectly displayed in maps, for example:

- Host group label displays the problem summary not including all hosts in nested host groups;
- ”Host group elements” view does not display a separate map element for each host in the nested host groups.

Zabbix server 和 proxy 的日志轮转

logrotate 实用工具仅包含在 zabbix-agent、zabbix-agent2 和 zabbix-web-service 的包中，但需要单独为 Zabbix server 和 proxy 安装。从 Zabbix 6.4.4rc1 开始，logrotate 依赖项已添加到 RHEL 和 SUSE 的 server 和 proxy 包中。

Zabbix 6.0.11 中的问题

JSONPath 解析错误

JSONPath 解析错误发生在前导空格和空数组/对象的情况下。在 Zabbix 6.0.12 中修复。

LLD 过滤器中的 AND/OR 评估

在此版本中，底层发现过滤器/覆盖中的 AND/OR 表达式的评估可能会失败。在 Zabbix 6.0.12 中修复。

Windows agent 归档中缺少文件

Windows Zabbix agent 的下载 ZIP 文件在 6.0.0 至 6.0.27 版本中缺少 zabbix_sender.h 和 zabbix_sender.lib 文件，这些文件是 zabbix_sender.dll 所需的。

Use case with global variables shared across webhook calls

As global variables are shared across different webhook calls, the following code will result in the tag value counter gradually increasing:

```
try
{
   aa = aa + 1;
}
catch(e)
{
```

```
   aa = 0;
}

result = {
        'tags': {
            'endpoint': aa
        }
    };
return JSON.stringify(result);
```

Using local variables instead of global ones is recommended to make sure that each script operates on its own data and that there are no collisions between simultaneous calls.

Processor groups on Windows

Microsoft documentation states that systems with fewer than 64 logical processors always have a single processor group, Group 0. However, Zabbix users have reported a rare bug ZBX-20260, when there are two processor groups on systems with 64 or less logical processors. This resulted in having the "\Processor(n)" performance counters for only one processor group out of two. The actual root cause of this bug is not known. However, a similar case was described at stackoverflow.com, and the root cause there was in interoperation between BIOS and Windows.

Limits of filtering with utf8mb4 collations

Filters (e.g., in *Configuration* → *Maintenance*) may not function correctly when applied to entities containing certain Unicode characters (e.g., 𝓔, 𝔢). This issue arises due to how the default utf8mb4_bin collation for MySQL or MariaDB databases handles sorting and comparison of Unicode characters.

To address this limitation, users can change the collation of database columns to alternatives such as utf8mb4_0900_bin, utf8mb4_0900_ai_ci, or utf8mb4_unicode_520_ci. Note, however, that changing the collation may cause unexpected behavior in the handling of empty spaces, as well as sorting and filtering for other characters.

For more information on changing collations, see MySQL documentation or MariaDB documentation. For details on collation differences, see Unicode Character Sets in MySQL documentation.

MySQL 8.0 Group Replication in multi-primary mode

When using MySQL 8.0 Group Replication in multi-primary mode, you may encounter an error during transaction commits similar to the following:

```
1531697:20250128:064734.697 query [txnlev:1] [update alerts set status=1,retries=0,error='' where alertid=
1531697:20250128:064734.713 query [txnlev:1] [commit;]
1531697:20250128:064734.753 [Z3005] query failed: [3101] Plugin instructed the server to rollback the curr
```

This error appears to be triggered by issues with rollback operations involving foreign key constraints.

See also:

- ZBX-26060 for the related problem report.
- MySQL Bug #96758 "Rollbacks with Foreign Keys on single node" for the upstream issue.

1 Compilation issues

These are the known issues regarding Zabbix compilation from sources. For all other cases, see the Known issues page.

Compiling Zabbix agent on HP-UX

If you install the PCRE library from a popular HP-UX package site http://hpux.connect.org.uk, for example from file `pcre-8.42-ia64_64-11.3` you get only the 64-bit version of the library installed in the /usr/local/lib/hpux64 directory.

In this case, for successful agent compilation customized options need to be used for the "configure" script, e.g.:

```
CFLAGS="+DD64" ./configure --enable-agent --with-libpcre-include=/usr/local/include --with-libpcre-lib=/us
```

Library in a non-standard location

Zabbix allows you to specify a library located in a non-standard location. In the example below, Zabbix will run `curl-config` from the specified non-standard location and use its output to determine the correct libcurl to use.

```
$ ./configure --enable-server --with-mysql --with-libcurl=/usr/local/bin/curl-config
```

This will work if it is the only libcurl installed in the system, but might not if there is another libcurl installed in a standard location (by the package manager, for example). Such is the case when you need a newer version of the library for Zabbix and the older one for other applications.

Therefore, specifying a component in a non-standard location will not always work when the same component also exists in a standard location.

For example, if you use a newer libcurl installed in `/usr/local` with the libcurl package still installed, Zabbix might pick up the wrong one and compilation will fail:

```
usr/bin/ld: ../../src/libs/zbxhttp/libzbxhttp.a(http.o): in function 'zbx_http_convert_to_utf8':
/tmp/zabbix-master/src/libs/zbxhttp/http.c:957: undefined reference to 'curl_easy_header'
collect2: error: ld returned 1 exit status
```

Here, the function `curl_easy_header()` is not available in the older `/usr/lib/x86_64-linux-gnu/libcurl.so`, but is available in the newer `/usr/local/lib/libcurl.so`.

The problem lies with the order of linker flags, and one solution is to specify the full path to the library in an LDFLAGS variable:

```
$ LDFLAGS="-Wl,--no-as-needed /usr/local/lib/libcurl.so" ./configure --enable-server --with-mysql --with-l
```

Note the `-Wl,--no-as-needed` option which might be needed on some systems (see also: default linking options on Debian-based systems).

9 模版变更

此页面列出了 Zabbix 内置模板的所有变更。

注意，升级到最新的 Zabbix 版本不会自动升级所使用的模板。建议通过以下方式修改现有安装中的模板:

- 从Zabbix Git repository下载最新模版；
- 然后，在配置 → 模板你可以手动导入到 Zabbix。如果已经存在相同名称的模板，在导入时应该检查 *Delete missing* 选项以实现干净的导入。这样，更新后的模板中不再存在的旧项将被删除 (注意，这将意味着丢失这些旧项的历史记录)。

在 6.0.0 中的变化

Updated template format

As of Zabbix 6.0, all templates follow an updated format, which may impact the import of pre-6.0 templates.

The updated default template name no longer contains the initial denominator 'Template' or the target application, e.g. 'Net' or 'OS'. The target application is instead specified in the template file using tags.

For example, as of Zabbix 6.0, 'Template Net TP-LINK SNMP' is named 'TP-LINK by SNMP'; 'Template OS Windows SNMP' is named 'Windows by SNMP', whereby its YAML export file contains the following tags:

```
    tags:
      - tag: class
        value: os
```

In the new format, each template and its elements are also assigned a UUID property for looking up matching objects, for example:

```
groups:
  - uuid: 846977d1dfed4968bc5f8bdb363285bc
    name: 'Templates/Operating systems'
templates:
  - uuid: f9a59315c8944853bb91c0a9ec3056d7
    template: 'Windows by SNMP'
    name: 'Windows by SNMP'
```

Importing a pre-6.0 template in Zabbix 6.0 may therefore result in the old template being renamed and suggested to be merged with the updated template. To avoid any unwanted merges, consider implementing one of these options:

- Migrating your setup to the updated template format and moving customizations (if any) to separate templates.
- Renaming your pre-6.0 template to include a custom prefix or suffix, e.g. 'Template OS Windows SNMP *custom*'. In most cases, this can be done directly in the import file. If not, you can use the Zabbix frontend or API to rename and export the templates.
- Adding a `uuid` tag and custom value to the template import file.

To compare templates between different Zabbix versions, you can open a given template in the Zabbix Git repository and switch between the relevant release branches.

新模版

详情查阅 Zabbix 6.0.0 中的new templates

模版的变化

- {#FSLABEL} 宏已添加到对应的项目名称和描述在 *Windows by Zabbix agent* 和 *Windows by Zabbix agent active* 模板中
- vfs.file.cksum[/etc/passwd] 代理项目已经被更改成 vfs.file.cksum[/etc/passwd,sha256]
- A new check zabbix[process,odbc poller,avg,busy] has been added to *Zabbix server*, *Zabbix proxy*, *Remote Zabbix server* and *Remote Zabbix proxy* templates. The metric is used for monitoring the average time for which ODBC processes have been busy during the last minute (in percentage). 一个新的检查 zabbix[process,odbc poller,avg,busy] 已添加到 *Zabbix server*, *Zabbix proxy*, *Remote Zabbix server* 和 *Remote Zabbix proxy* 模板。该指标用于监视 ODBC 进程在最后一分钟内忙碌的平均时间 (百分比)。

在 6.0.2 中的变化

模板 *Generic Java JMX* 现在包含两个发现规则：- Garbage collector discovery - Memory pool discovery

在 6.0.3 中的变化

可使用新的模板 *OpenWeatherMap by HTTP*。

对现有模板进行了以下更改:

- 在模板 *Windows services by Zabbix agent*, *Windows services by Zabbix agent active*, *Windows by Zabbix agent*, *Windows by Zabbix agent active* 中 {$SERVICE.NAME.NOT_MATCHES} 宏值已被更新，以过滤掉扩展的服务列表。
- 模版 *PostgreSQL by Zabbix agent 2* 现在将检查慢速查询的数量，如果数量超过阈值将产生问题。

在 6.0.4 中的变化

可用的新模版：

- *TrueNAS SNMP* - 通过 SNMP 监控 TrueNAS 存储操作系统。
- *Proxmox VE by HTTP* - 参见HTTP 模板的设置说明。

模版 *SMART by Zabbix agent 2* and *SMART by Zabbix agent 2 (active)* 已更新：- *Attribute discovery* LLD 规则已被删除，而 *Disk discovery*LLD 规则现在将基于预定义的特定厂商的属性集发现磁盘；- **smart.disk.get** 项现在只能返回关于特定磁盘的信息，而不是所有磁盘的信息。

模板中添加了新的宏，用于定义用于监控虚拟文件系统的文件系统利用率的警告和临界阈值。*HOST-RESOURCES-MIB storage SNMP*, *Linux by Prom*, *Linux filesystems SNMP*, *Linux filesystems by Zabbix agent active*, *Linux filesystems by Zabbix agent*, *Mellanox SNMP*, *PFSense SNMP*, *Windows filesystems by Zabbix agent active*, *Windows filesystems by Zabbix agent*. 为使用这些宏文件系统利用率触发器已经更新。

6.0.5 中的变化

新模板可用：

- CockroachDB by HTTP
- Envoy Proxy by HTTP
- HashiCorp Consul Cluster by HTTP
- HashiCorp Consul Node by HTTP

请参阅HTTP 模板 的设置说明。

6.0.6 中的变化

新模板可用：

- HPE MSA 2040 Storage by HTTP
- HPE MSA 2060 Storage by HTTP
- HPE Primera by HTTP

请参阅HTTP 模板 的设置说明。

6.0.7 中的变化

新的模板 *HPE Synergy by HTTP* 可用。

模板 *HashiCorp Consul Node by HTTP* 和 *HashiCorp Consul Cluster by HTTP* 现在支持 Consul 命名空间。

PostgreSQL Agent 2 template has been updated:

A trigger for detecting checksum failures has been added to the Dbstat item of the PostgreSQL Agent 2 template. According to PostgreSQL documentation, you can use checksums on data pages to help detect corruption by the I/O system that would otherwise be silent.

6.0.8 中的变化

一个新的模板 *OPNsense by SNMP* 是可用的。

6.0.13 中的变化

新模板可用：

- AWS EC2 by HTTP
- AWS by HTTP
- AWS RDS instance by HTTP
- AWS S3 bucket by HTTP

请参阅HTTP 模板 的设置说明。

模板 *Oracle by Zabbix agent 2* 已经升级了：

- 已删除以下静态监控项，这些监控项在单个查询中请求所有现有相关数据库对象的数据：
  - “Oracle：获取归档日志信息”
  - “Oracle：获取 ASM 统计信息”
  - “Oracle：获取 CDB 和 No-CDB 信息”
  - “Oracle：获取 PDB 信息”
  - “Oracle：获取表空间统计信息”
- 以下 agent 监控项原型已添加到相应的发现规则中：
  - 归档日志发现规则：“Archivelog ’{#DEST_NAME}’：获取归档日志信息”
  - ASM 磁盘组发现：“ASM ’{#DGNAME}’：获取 ASM 统计信息”
  - 数据库发现：“Oracle 数据库 ‘{#DBNAME}’：获取 CDB 和非 CDB 信息”
  - PDB 发现：“Oracle 数据库 ‘{#DBNAME}’：获取 PDB 信息”
  - 表空间发现：“Oracle TBS ’{#TABLESPACE}’：获取表空间统计信息”

CHANGES IN 6.0.18

The template *Azure by HTTP* now also works with Azure Cosmos DB for MongoDB.

CHANGES IN 6.0.20

A new template *AWS ECS Cluster by HTTP* (along with its Serverless Cluster version) is available.

CHANGES IN 6.0.21

New template is available:

- AWS Cost Explorer by HTTP

CHANGES IN 6.0.22

New template is available:

- MantisBT by HTTP

CHANGES IN 6.0.23

New template is available:

- Nextcloud by HTTP

Updated templates

PostgreSQL by ODBC and PostgreSQL by Zabbix agent 2 templates now include the item and trigger for monitoring PostgreSQL version.

CHANGES IN 6.0.24

Updated templates

Integration with OpenShift has been added to Kubernetes cluster state by HTTP template.

CHANGES IN 6.0.26

Updated templates

MSSQL by ODBC template:

- new item has been added - MSSQL DB ’{#DBNAME}’: Recovery model, which returns the database recovery model under the database discovery;
- new macros, namely, {$MSSQL.BACKUP_FULL.USED}, {$MSSQL.BACKUP_DIFF.USED}, {$MSSQL.BACKUP_LOG.USED}, have been added - those can be used for disabling backup age triggers for a certain database;
- LLD rules for quorum and quorum members discovery have been added;
- the type of the LLD rules has been changed from “Database monitor” to “Dependent item”;
- items with `db.odbc.discovery` key have been turned into items dependent on `db.odbc.get` item.

CHANGES IN 6.0.27

New templates

A new template is available:

- YugabyteDB by HTTP, which includes the *YugabyteDB Cluster by HTTP* template for monitoring each YugabyteDB cluster.

CHANGES IN 6.0.28

New templates

A new template is available:

- MSSQL by Zabbix agent 2

CHANGES IN 6.0.29

Updated templates

- FortiGate by SNMP template has been supplemented with metrics regarding VPN, high availability (HA), wireless termination points (WTPs), SD-WAN health checks, and HW sensors.

CHANGES IN 6.0.30

New templates

The AWS ELB template set has been supplemented with the template AWS ELB Network Load Balancer by HTTP.

CHANGES IN 6.0.31

New templates

A new template is available:

- Jira Data Center by JMX, a template for monitoring Jira Data Center health.

CHANGES IN 6.0.32

New templates

The set of Azure by HTTP templates has been supplemented with the Azure VM Scale Set by HTTP template.

CHANGES IN 6.0.33

New templates

A new template is available:

- AWS Lambda by HTTP, a template for monitoring AWS Lambda metrics.

CHANGES IN 6.0.34

New templates

A new template is available:

- GitHub repository by HTTP, a template for monitoring GitHub repositories.

CHANGES IN 6.0.35

Updated templates

- In all APC templates, the discovery rule for external defective battery packs has been supplemented with a filter to avoid creating unsupported items.
- In the templates MSSQL by Zabbix agent 2 and MSSQL by ODBC, a timeout has been added and certain item keys have been adjusted to increase monitoring stability.

CHANGES IN 6.0.36

New templates

A new template is available:

- Nutanix Prism Element by HTTP, a template set that also includes the templates *Nutanix Cluster Prism Element by HTTP* and *Nutanix Host Prism Element by HTTP*.

Updated templates

The template *VMware Hypervisor* within the VMware and VMware FQDN template sets has been updated with the option to be used as a standalone template.

CHANGES IN 6.0.39

New templates

A new template is available:

- The Azure by HTTP template set has been supplemented with the template *Azure SQL Managed Instance by HTTP*.

CHANGES IN 6.0.40

New templates

A new template is available:

- The template set Zabbix server health has been supplemented with the templates *Zabbix server health by Zabbix agent*, *Zabbix server health by Zabbix agent active*, and Zabbix proxy health - with *Zabbix proxy health by Zabbix agent* and *Zabbix proxy health by Zabbix agent active*, enabling the monitoring of internal Zabbix metrics via Zabbix agent.

10 6.0.0 更新说明

这些说明用于从 Zabbix 5.4.x 升级到 Zabbix 6.0.0。所有笔记分为：

- `Critical` - 与升级过程和 Zabbix 功能变化相关的最关键信息
- `Informational` - 描述 Zabbix 功能变化的所有剩余信息

可以从 Zabbix 5.4.0 之前的版本升级到 Zabbix 6.0.0。有关从更早的 Zabbix 版本升级的所有相关信息，请参阅升级说明 。

至关重要的 数据库

为了创造最佳的用户体验并确保在各种生产环境中获得最佳的 Zabbix 性能，已经放弃了对一些旧版本数据库的支持。这主要适用于接近使用寿命的数据库版本以及存在可能会干扰正常性能的未修复问题的版本。

从 Zabbix 6.0 开始，官方支持以下数据库 版本：

- *MySQL/Percona* 8.0.X
- *MariaDB* 10.5.X - 10.6.X
- *PostgreSQL* 13.X
- *Oracle* 19c - 21c
- *TimescaleDB* 2.0.1-2.3
- *SQLite* 3.3.5-3.34.X

默认情况下，如果检测到不支持的数据库版本，Zabbix server 和 proxy 将不会启动。虽然不推荐，但现在可以通过修改server 或proxy 的 AllowUnsupportedDBVersions 配置参数来关闭数据库版本适配。

主键

主键现在用于新安装中的所有表，包括历史表。

现有安装不会自动升级主键。在现有安装中 手动升级历史表为主键的说明适用于一下数据库：MySQL/MariaDB，PostgreSQL，TimescaleDB v1 和v2 以及Oracle。

PCRE2 支持

添加了对 PCRE2 的支持。仍然支持 PCRE，但 Zabbix 只能使用 PCRE 或 PCRE2 库之一进行编译，两者不能同时使用。

以下 Zabbix 安装包已更新，现在使用 PCRE2：- RHEL/CentOS 7 及更高版本 - SLES（所有版本）- Debian 9 及更高版本 - Ubuntu 16.04 及更高版本

注意，切换到 PCRE2 后，您可能需要更新一些正则表达式。特别是 `^[\w-\.]` 需要更改模式 `^[-\\w\\.]` 以继续正常工作 - 查看已知问题 了解详细说明。

单独处理 ODBC 检查

ODBC 检查处理现在由单独的 server/proxy 进程 *odbc pollers* 执行。以前，ODBC 检查是由常规轮询器执行的，它也适用于 Zabbix 代理项、SSH 检查等。

Zabbix server 和proxy 配置文件中添加了一个新的配置参数 *StartODBCPollers* 默认值为 *1*。此参数可能需要根据 server 或 proxy 执行的 ODBC 检查次数进行调整。你也可以相应地减少 *StartPollers* 参数设置的常规轮询器的数量。

内部监控项 *zabbix[process,<type>]* 可用于监控 ODBC 轮询器负载。

审计日志

为了改进 Zabbix 中的审计日志，并使审计日志完整可靠，必须重新设计以前存在的数据库结构。在升级数据库表期间 `auditlog` 、`auditlog_details` 将被具有不同格式的新表 `auditlog` 取代。旧的审计记录不会被保留。

新的 *Audit log*（审计日志） 单元已添加到 *Administration*（管理）→*General*（常规）菜单中，允许启用（默认）或禁用审计日志。以前位于 *Housekeeper*（管家）菜单下的用于审计的管家设置也已移至新的 *Audit log*（审计日志）单元。现有的管家设置将被保存。

API 更改

参阅 Zabbix 6.0.0 中的API changes 。

移除 Monitoring（监控）→ Overview（概览）

监控菜单中的概览部分已被完全移除。使用仪表板小部件 *Data overview*（数据概览） and *Trigger overview*（触发器概览）可以实现相同的功能。

禁用更改继承触发器的依赖项

禁用了更改从模板继承的触发器的依赖关系的功能。

原因是在更新模板的触发器依赖关系时，继承触发器的依赖关系被覆盖。

因此，始终仅在根模板级别设置触发器依赖更可靠。

宏

不再支持位置宏

自 Zabbix 4.0 起已弃用的监控项名称 ($1, $2...$9) 中对位置宏的支持已被完全删除。

不再支持监控项名称中的用户宏

自 Zabbix 4.0 起已弃用的监控项名称（包括发现规则名称）中对用户宏的支持已被完全删除。

表达式宏取代简单宏

简单宏的功能已转移到表达式宏。现有的简单宏将在升级过程中转换为表达式宏。在不超过长度限制的情况下无法转换的宏将不会被转换，这会在日志中输出警告。

一般性的　弃用历史/趋势相关的内部监控项

以下内部监控项现已弃用，并将在未来的主版本中删除：

- `zabbix[history]`
- `zabbix[history_log]`
- `zabbix[history_str]`
- `zabbix[history_text]`
- `zabbix[history_uint]`
- `zabbix[trends]`
- `zabbix[trends_uint]`

Zabbix agent 2 插件

每个 Zabbix agent 2 插件都有独立的配置文件。默认这些文件位于 `./zabbix_agent2.d/plugins.d/` 路径下。也可以由 agent 2 配置文件的参数 `Include` 来指定。配置文件参见zabbix_agent2.conf 或zabbix_agent2.win.conf。

用户密码

以前，在用户配置表单和登录表单中，用户密码中的空格已被自动修剪。引入可配置密码复杂度要求 后，密码中的空格不再修剪。因此，密码中有空格的用户将无法像往常一样登录，而必须输入没有空格的’ 旧’ 密码。要继续使用带空格的密码，他们需要重新创建密码。

Prometheus 指标的批量处理

由于在 Prometheus 指标的预处理队列中引入了依赖监控项的批量处理，依赖监控项将不再并行处理，这可能会影响它们的处理速度。

运行时命令的传输

Zabbix server 和 proxy 运行时命令现在通过套接字而不是 Unix 信号发送。此更改可以改善使用运行时控制选项的用户体验：

- 命令执行的结果会打印到控制台。
- 可以发送更长的输入参数，例如 HA 节点名称而不是节点编号。

不再支持收藏的自定义图表

不再可以在 *Monitoring*（监控） -> *Hosts*（主机） -> *Graphs*（图形）中将自定义图表添加到收藏夹。升级后，任何现有的自定义图表都将从收藏夹中删除。

服务监控

进行了与服务监控功能相关的几项重大更新 。升级期间将通过以下方式更改现有服务树配置：

- 问题和服务之间基于触发器的依赖关系被基于标签的服务到问题的映射所取代。已链接到服务的触发器将获得一个新标签 `ServiceLink：<trigger ID>:<trigger name>`（标签值将被截断为 32 个字符）。链接的服务将获得相同的问题标签。
- 硬依赖和软依赖不再存在。相反，一个服务将有多个父服务。
- ’Status calculation algorithm（状态计算算法）’ 将使用以下规则升级：
  - Do not calculate（不计算）→ 将状态设置为 OK
  - 问题，如果至少一个子节点有问题 → 最关键的子节点
  - 问题，如果所有子节点都有问题 → 如果所有子节点都有问题到达临界（最严重）
- SLA 不再是服务属性，而是可以分配给多个服务的单独实体。在升级期间，相同的 SLA 将被分组，并且每个组将创建一个 SLA。服务将获得一个新的服务标签 `SLA:<ID>` 用于匹配。

也可以参看：

- 服务监控升级详解；
- 服务配置。

11 6.0.1 更新说明

配置同步器

配置同步器的性能得到了改进。如果有大量模板，建议增加 server/proxy 上的 CacheSize。同时建议删除未使用的模板。

请注意，server/proxy 上的默认 CacheSize 已增加到 32M。

监控项变化

Zabbix agent 2 中添加了对监控项 **net.dns** 和 **net.dns.record** 的原生支持。在 Zabbix agent 2 for Windows 中，这些监控项现在允许在 `ip` 参数中自定义 DNS IP 地址，并且不再忽略 `timeout` 和 `count` 参数。

12 6.0.2 更新说明

此小版本没有任何升级说明。

13 6.0.3 更新说明

此小版本没有任何升级说明。

14 6.0.4 更新说明

此小版本没有任何升级说明。

15 6.0.5 更新说明

此小版本没有任何升级说明。

16 6.0.6 更新说明

外部 **MongoDB** 插件　MongoDB 插件 现已可以作为可加载的插件使用了，而不再是 Zabbix agent 2 的一部分。支持的 MongoDB 版本列表已扩展至 *2.6-5.3*。

插件功能及支持的监控项 没有改变。

17 6.0.7 升级说明

符号链接名称扩展　符号链接名称和符号链接的完整路径现在在 `vfs.dir.get[]` 和 `vfs.file.get[]` 监控项中返回，而不是解析为符号链接目标。

18 6.0.8 更新说明

此小版本没有任何升级说明。

19 6.0.9 升级说明

此小版本没有任何升级说明。

20 6.0.10 升级说明

重大变化

PostgreSQL 插件移动到可加载插件　PostgreSQL 插件 不再内置在 Zabbix agent 2 中。相反，它现在是 agent 2 中的可加载插件。

此更改可能/将破坏 Ansible、Chef 等的自动化，因为不再可能直接拉取插件存储库。

另请参阅：PostgreSQL 可加载插件 存储库

21 6.0.11 升级说明

重大变化

数据库模式更新　新列 *name_upper* 已添加到数据库表 **items**。该列包含监控项名称或 LLD 规则名称的大写版本。已实施此更改以允许使用不区分大小写的索引搜索 API 查询优化。

优化的 **API** 查询　在 *hosts* 和 *items* 表中搜索名称时创建的 API 数据库查询已经过优化，现在可以更有效地处理。由于此更改，需要在升级期间创建确定性触发器。

在 MySQL 和 MariaDB 上，如果启用了二进制日志记录并且没有超级用户权限并且 MySQL 配置文件中未设置 log_bin_trust_function_creators = 1，则需要设置 GLOBAL log_bin_trust_function_creators = 1。要使用 MySQL 控制台设置变量，请运行：

·mysql> SET GLOBAL log_bin_trust_function_creators = 1;

升级成功完成后，可以禁用 log_bin_trust_function_creators：

·mysql> SET GLOBAL log_bin_trust_function_creators = 0;

还为 PostgreSQL 和 Oracle 数据库创建了触发器。

JSONPath 解析错误　在前导空格和空数组/对象的情况下，此版本中会出现 JSONPath 解析错误。在 Zabbix 6.0.12 中修复。

AND/OR 评估 **LLD** 过滤器　在此版本中，底层发现过滤器/覆盖中的 AND/OR 表达式的评估可能会失败。在 Zabbix 6.0.12 中修复。

22 6.0.12 升级说明

改进了历史同步器的性能　通过引入新的读写锁提高了历史同步器的性能。这通过在访问配置缓存时使用共享读锁减少了历史同步器、捕获器和代理轮询器之间的锁定。新锁只能由配置同步器执行配置缓存重新加载。

23 6.0.13 升级说明

重大变化　可加载插件版本控制

Loadable plugins for Zabbix agent 2 现在使用与 Zabbix 本身相同的版本控制系统。进行了以下版本更改：

- MongoDB 1.2.0 -> MongoDB 6.0.13
- PostgreSQL 1.2.1 -> PostgreSQL 6.0.13

Zabbix 6.0 的任何次要版本都支持这些插件。请注意，每个插件的源代码存储库现在包含一个专用的 *release/6.0* 分支（以前只有 *master* 分支）。

配置导入　查看配置导入 过程中的更改。

使用 **Zabbix agent 2** 查询 **Oracle** 数据库中的单独表空间　以下Zabbix agent 2 监控项，支持 Oracle 插件，现在有额外的可选参数：

- oracle.diskgroups.stats[<existingParameters>,**<diskgroup>**]
- oracle.archive.info[<existingParameters>,**<destination>**]
- oracle.cdb.info[<existingParameters>,**<database>**]
- oracle.pdb.info[<existingParameters>,**<database>**]
- oracle.ts.stats[<existingParameters>,**<tablespace>**,**<type>**]

这些参数允许查询单独的数据实例而不是所有数据，从而提高性能。

Zabbix agent 2 打开文件描述符限制增加　Zabbix agent 2 包中提供的 systemd 服务文件现在声明打开文件描述符限制为 8196。以前，系统默认限制为 1024。新限制足以满足默认的 Zabbix agent 2 配置。如果您有非标准 agent 2 配置，例如，使用额外的插件或扩展功能，则可能需要手动进一步增加此限制。在这种情况下，调整 systemd 单元文件中的 `LimitNOFILE` 参数。

24 Upgrade notes for 6.0.14

Limits for JavaScript objects in preprocessing　The following limits for JavaScript objects in preprocessing have been introduced:

- The total size of all messages that can be logged with the `Log()` method has been limited to 8 MB per script execution.
- The initialization of multiple `HttpRequest` objects has been limited to 10 per script execution.
- The total length of header fields that can be added to a single `HttpRequest` object with the `addHeader()` method has been limited to 128 Kbytes (special characters and header names included).

25 Upgrade notes for 6.0.15

This minor version doesn’t have any upgrade notes.

26 Upgrade notes for 6.0.16

This minor version doesn’t have any upgrade notes.

27 Upgrade notes for 6.0.17

HTML support in Geomap attribution dropped The attribution text for the Geomap dashboard widget can now only contain plain text; HTML support has been dropped. If this field already contains HTML, it will be rendered as plain text after the upgrade.

In Geographical maps settings in the Administration → General section, the field *Attribution* is now only visible when *Tile provider* is set to *Other*.

28 Upgrade notes for 6.0.18

Proxy history housekeeping The limitation on the amount of outdated information deleted from the proxy database per proxy history housekeeping cycle has been removed.

Previously the housekeeper deleted only no more than 4 times the `HousekeepingFrequency` hours of outdated information. For example, if `HousekeepingFrequency` was set to "1", no more than 4 hours of outdated information (starting from the oldest entry) was deleted. In cases when a proxy would constantly receive data older than set in `ProxyOfflineBuffer`, this could result in excessive data accumulation.

Now this limitation has been removed, providing a more effective proxy history housekeeping solution.

29 Upgrade notes for 6.0.19

Aggregate functions The **count_foreach** function now returns '0' for a matching item in the array, if no data are present for the item or the data do not match the filter. Previously such items would be ignored (no data added to the aggregation).

30 Upgrade notes for 6.0.20

This minor version doesn't have any upgrade notes.

31 Upgrade notes for 6.0.21

Maximum JSON Depth

The maximum allowed JSON depth has been set to 64 to improve security and performance during JSON parsing.

Macro functions

The range of the `fmtnum` macro function is now limited to 0-20.

32 Upgrade notes for 6.0.22

Autoregistration table cleared from orphaned records

Previously the `autoreg_host` table was never cleared. Now orphaned records that reference neither an autoregistration event nor an existing host will be removed by the Housekeeper.

PostgreSQL plugin parameters**

The following PostgreSQL plugin parameters are no longer mandatory if Plugins.PostgreSQL.Sessions.<SessionName>.TLSConnect is set to `verify_ca` or `verify_full`:

- Plugins.PostgreSQL.Sessions.<SessionName>.TLSCertFile
- Plugins.PostgreSQL.Sessions.<SessionName>.TLSKeyFile

33 Upgrade notes for 6.0.23

Breaking changes Agent 2 plugins

The correct master host is now returned if specified in the `mysql.replication.get_slave_status[connString,<username>,<passwo` item "masterHost" parameter. Previously the "masterHost" parameter was ignored and always the first master host was returned.

If this parameter is not specified, all hosts are returned.

MySQL plugin parameters

The following MySQL plugin parameters are no longer mandatory if Plugins.Mysql.Sessions.<SessionName>.TLSConnect is set to `verify_ca` or `verify_full`:

- Plugins.Mysql.Sessions.<SessionName>.TLSCertFile
- Plugins.Mysql.Sessions.<SessionName>.TLSKeyFile

PostgreSQL-TimescaleDB combination no longer checked

Zabbix no longer checks for the supported PostgreSQL-TimescaleDB combination. As before, Zabbix does check for the supported PostgreSQL version or TimescaleDB version separately.

34 Upgrade notes for 6.0.24

This minor version doesn't have any upgrade notes.

MongoDB plugin parameters

The following MongoDB plugin parameters are no longer mandatory if Plugins.MongoDB.Sessions.<SessionName>.TLSConnect is set to `verify_ca` or `verify_full`:

- Plugins.MongoDB.Sessions.<SessionName>.TLSCertFile
- Plugins.MongoDB.Sessions.<SessionName>.TLSKeyFile

35 Upgrade notes for 6.0.25

This minor version doesn't have any upgrade notes.

Consistency introduced in sha256 checksums of vfs.file.cksum item

In earlier Zabbix versions, the Zabbix agent item `vfs.file.cksum` produced different sha256 sums for the same file depending on the platform (processor architecture) – sha256 sums on AIX, HP-UX (Itanium) and Solaris (SPARC) differed from those produced on Intel x86-64 platforms.

The issue has now been fixed; however, after upgrading, monitoring sha256 sums of files on AIX, HP-UX, or Solaris SPARC may result in false positives of files having been modified.

36 Upgrade notes for 6.0.26

This minor version doesn't have any upgrade notes.

37 Upgrade notes for 6.0.27

This minor version doesn't have any upgrade notes.

Zabbix agent 2 support on Windows

To prevent critical security vulnerabilities, the minimum Windows version for Zabbix agent 2 has been raised to Windows 10/Server 2016. See note under Supported platforms for more information.

38 Upgrade notes for 6.0.28

Guest user authorization

Automatic login for the guest user has been removed. After this change, the guest user will need to log in like any other user. Previously, a guest could immediately get to almost any monitoring or reporting page without going through authorization.

Invalid regular expression in proc.* items

proc.* agent items will now become 'not supported' if an invalid regular expression is supplied.

39 Upgrade notes for 6.0.29

This minor version doesn't have any upgrade notes.

40 6.0.30 版本升级说明

此次次要版本没有任何升级说明。

41 6.0.31 版本升级说明

此次要发布的次要版本没有任何升级说明。

42 Upgrade notes for 6.0.32

New index on auditlog table

A new index has been added to the `auditlog` table to improve database and frontend response times when filtering records by *Recordset ID* in the Audit log.

Note that users with large audit logs may experience extended upgrade times due to the database size.

Server to stop with read-only database

A standalone Zabbix server will now stop if the database becomes read-only.

43 Upgrade notes for 6.0.33

This minor version does not have any upgrade notes.

44 Upgrade notes for 6.0.34

Databases Database changes

A slow template cloning issue has been resolved by adding indexes for the uuid field. This change may cause long upgrade times on large datasets.

Processes Problems no longer cached without enabled services

The service manager no longer caches problems if there are no services enabled.

The service manager will check for enabled services in configuration cache during startup and cache problems only if there are enabled services; it will also detect when the first service is added/last service removed and will initialize/free the problem cache.

45 Upgrade notes for 6.0.35

This minor version does not have any upgrade notes.

Agent 2 package support for RHEL 6 dropped

Since Zabbix 6.0.35, Zabbix agent 2 packages are no longer available for RHEL 6 due to Go 1.22 (and newer) being incompatible with GCC 4.4.7 provided by RHEL 6, which results in compilation errors. Note, however, that Zabbix agent packages are still supported.

46 Upgrade notes for 6.0.36

This minor version does not have any upgrade notes.

47 Upgrade notes for 6.0.37

This minor version does not have any upgrade notes.

48 Upgrade notes for 6.0.38

Breaking changes Java 11 required for Java gateway

Zabbix Java gateway now requires Java 11 for runtime (building from source is still possible with Java 8), due to updated minimum logback library versions:

Library	New minimum version	Old minimum version
logback-classic	**1.5.16**	1.2.9
logback-core	**1.5.16**	1.2.9
slf4j-api	**2.0.16**	1.7.32

49 Upgrade notes for 6.0.39

Breaking changes MSSQL Zabbix agent 2 plugin update

The template MSSQL by Zabbix agent 2 has been updated with filters to include or exclude discovered quorum members by name as well as a service filter that allows filtering by cluster name to exclude empty clusters. To have the template work without errors, the MSSQL Zabbix agent 2 plugin must be updated to a version equal to or above 6.0.39.

50 Upgrade notes for 6.0.40

This minor version does not have any upgrade notes.

51 Upgrade notes for 6.0.41

This minor version does not have any upgrade notes.

Minimum supported Go version

The minimum supported Go version has been raised from 1.21 to 1.23.

If you previously built Zabbix agent 2, agent 2 loadable plugins, or web service from source using a Go version older than 1.23, it is recommended to rebuild these components using a supported Go version to receive the latest security updates and bug fixes. Components built with an older Go version will continue to work, but upgrading them will require a newer Go environment.

5. 快速入门

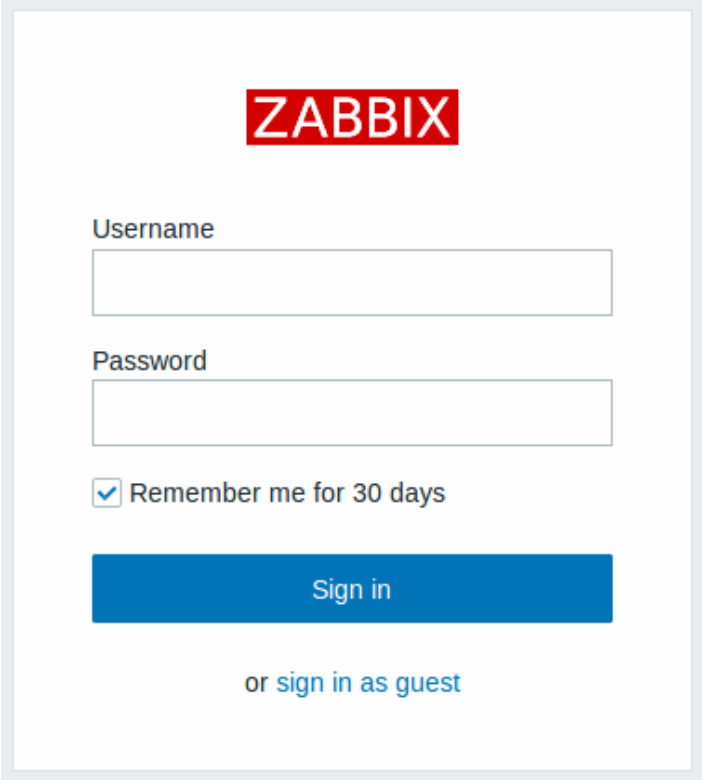
请使用侧边栏访问快速入门部分的内容。

1 登录和配置用户

概述

本章你会学习如何登录 Zabbix，以及在 Zabbix 内创建一个系统用户。

登录



这是 Zabbix 的 “欢迎” 界面。输入用户名 **Admin** 以及密码 **zabbix** 以作为Zabbix 超级用户登录。将授予访问配置和管理菜单的权限。

防止暴力破解

为了防止暴力破解和词典攻击，如果发生连续五次尝试登录失败，Zabbix 界面将暂停 30 秒。

在下次成功登录后，将会在界面上尝试登录失败的 IP 地址。

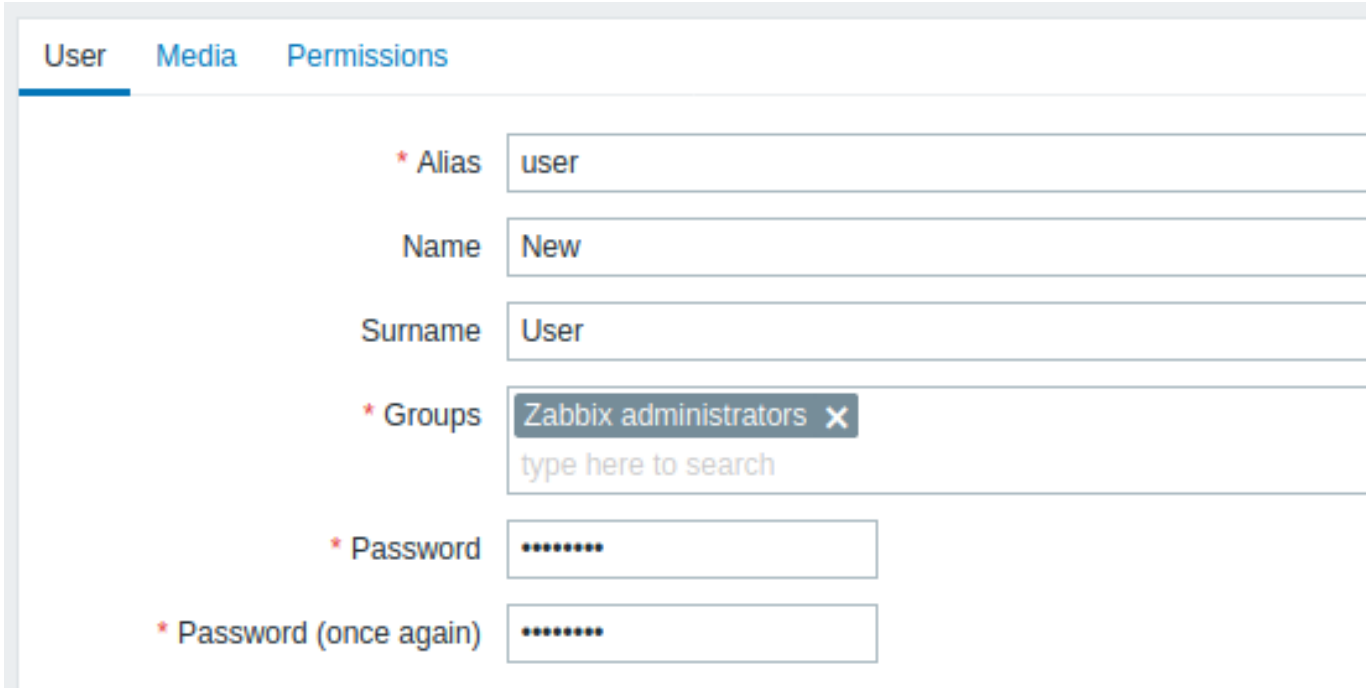
创建用户

可以在管理（*Administration*）→ 用户（*Users*）下查看用户信息。



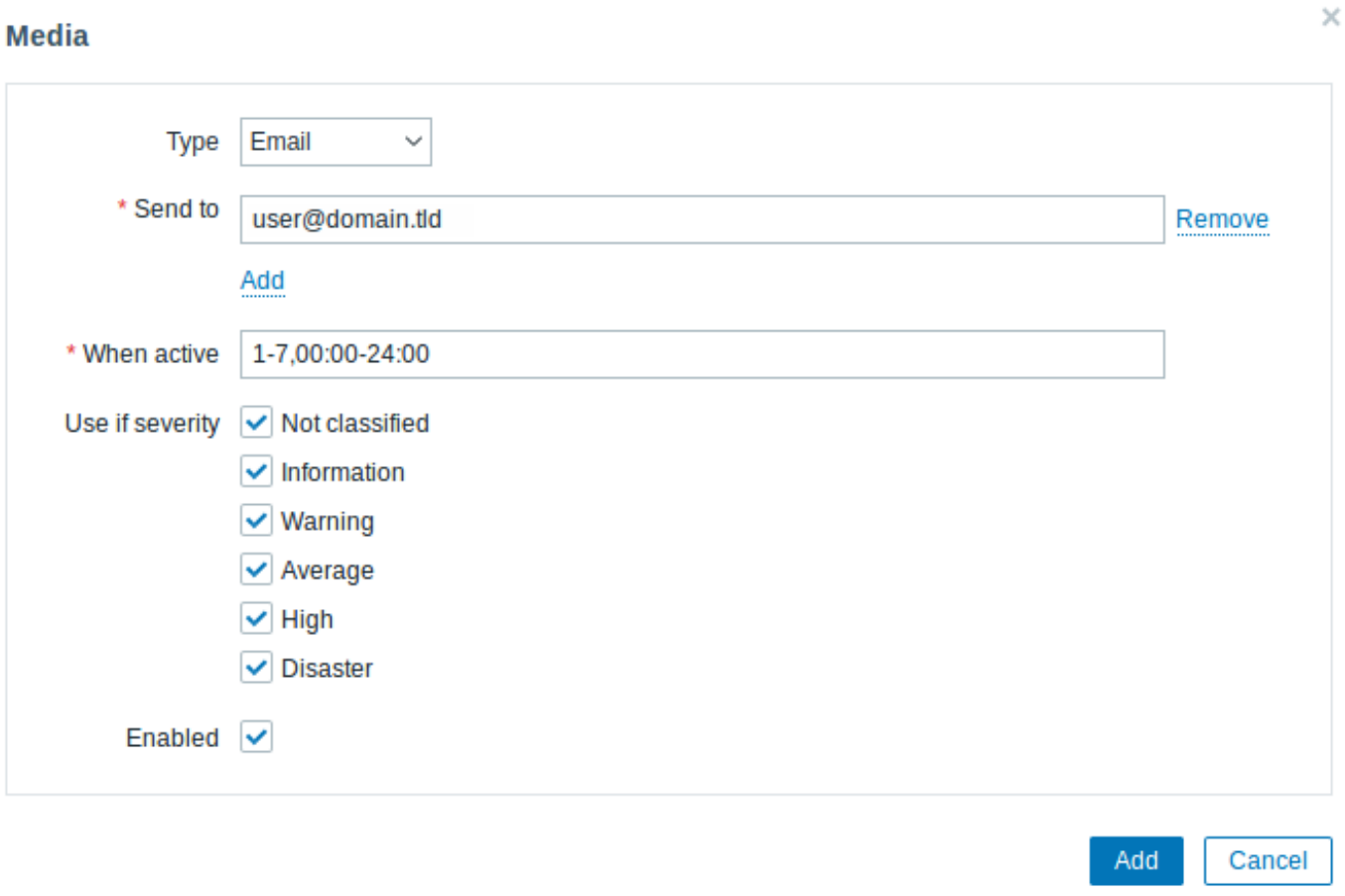
点击 创建用户添加一个新用户。

在创建的用户表单中，请确保将你的用户添加到现有的用户组, 例如'Zabbix administrators'.



所有必填输入字段均标有红色星号。

默认情况下，新用户没有为其定义媒介（通知传递方法）。如果要创建，请转到“媒介”选项卡并单击添加。

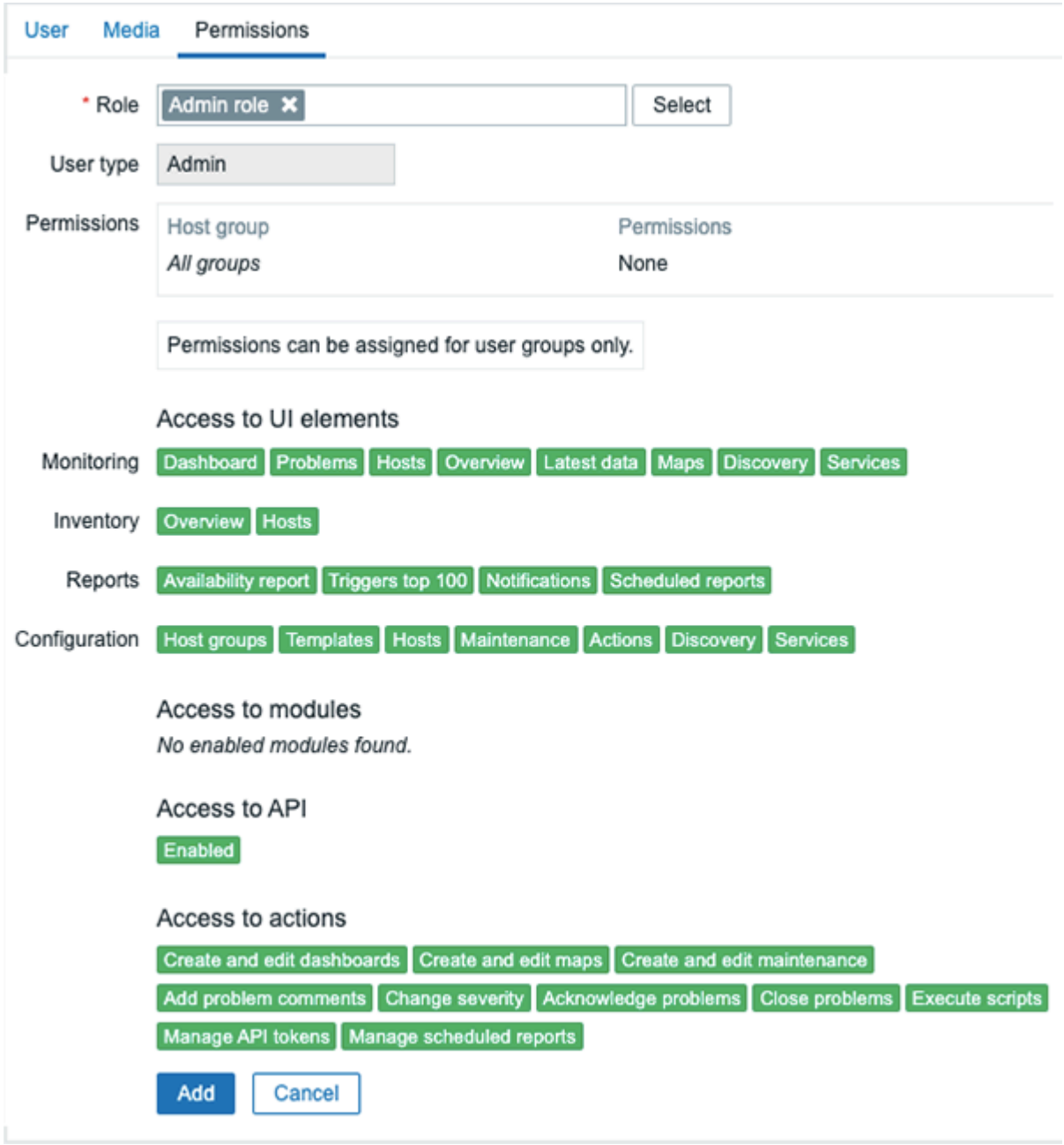


在此弹出窗口中，输入用户的电子邮件地址。

你可以为媒介指定一个时间活动周期 (参考时间周期说明 页面，查看该字段格式的描述)，默认情况下，媒介一直是活动的。你也可以通过自定义触发器严重等级来激活媒介，但暂时保持所有级别的都处于启用状态。

点击添加保存媒介，然后转到 “权限” 选项卡。

权限选项卡有一个必填字段角色。该角色决定用户可以查看哪些前端元素，以及允许用户执行哪些操作。点击选择，然后从列表中选择一个角色。例如，选择 *Admin role* 以允许访问除管理之外的所有 Zabbix 前端部分。稍后，你可以修改权限或创建更多用户角色。选择角色后，权限将显示在同一选项卡中：

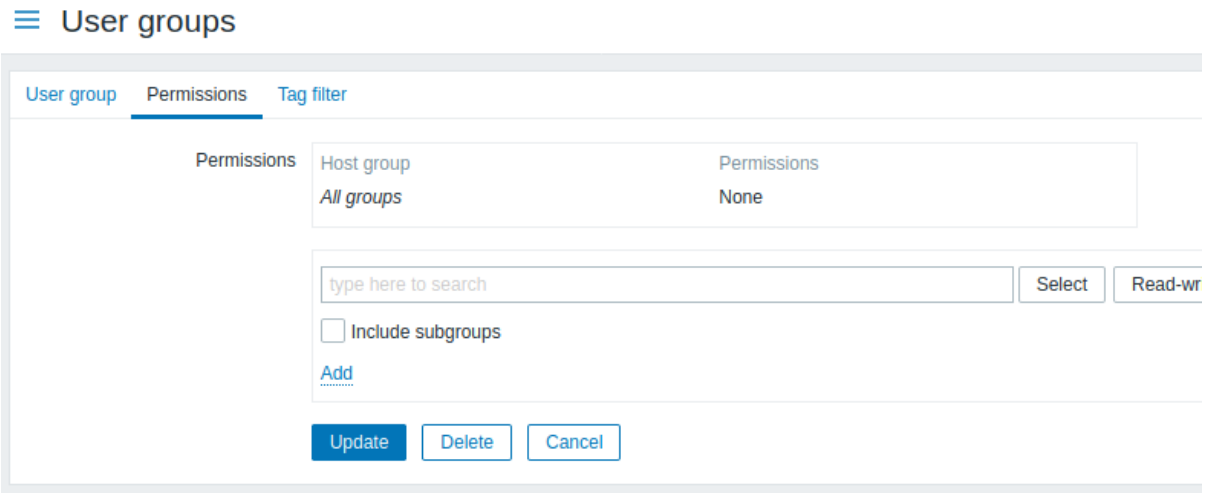


在用户窗口中点击添加以保存用户。新用户将出现在用户列表中。

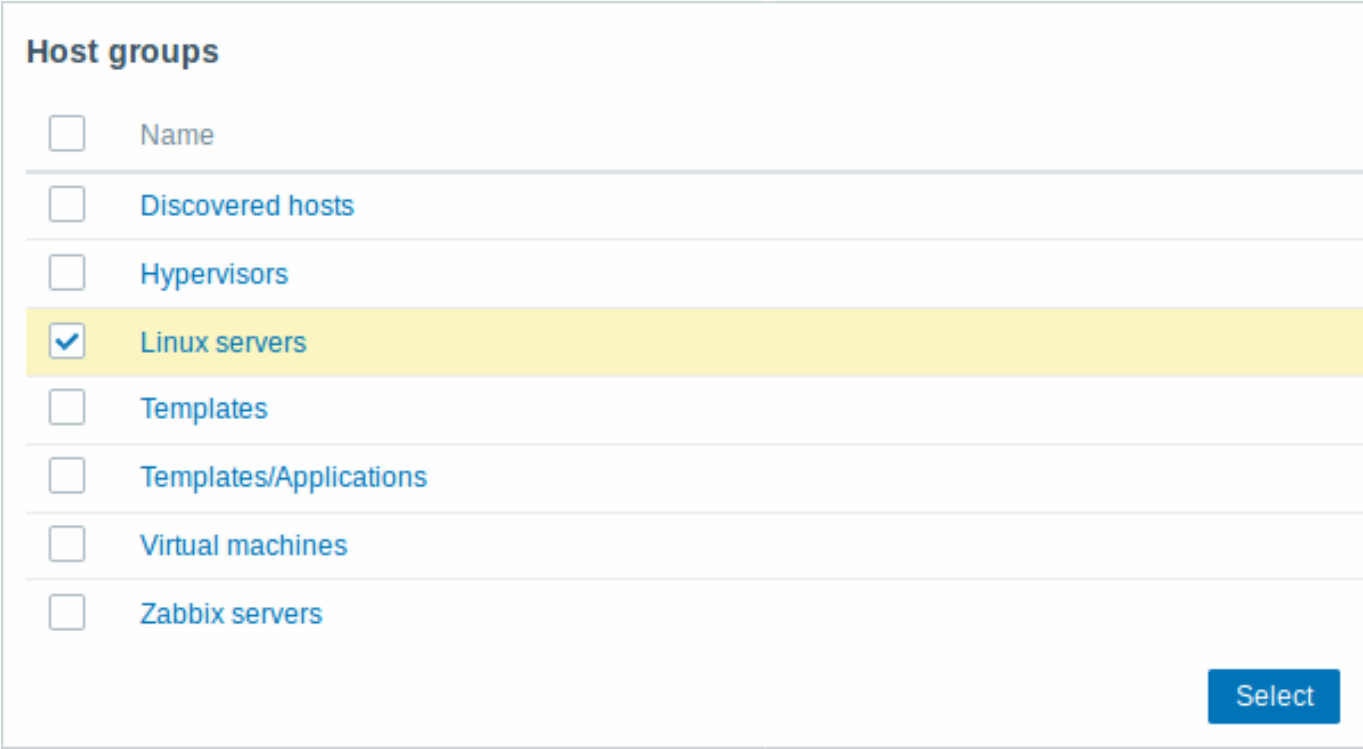


添加权限

默认情况下，一个新的用户没有权限访问主机。要授予用户权限，请单击组列中的用户组（在本例中为 “Zabbix administrators”）。在 “用户群组” 窗口中，转到 “权限” 选项卡。



该用户需要以只读方式访问 *Linux Server* 主机组，因此请单击用户组选择字段旁边的选择。



在此弹出框中，选中在 “Linux servers” 旁边的复选框，然后单击选择。*Linux servers* 就会显示在选择清单中。单击 “Read” 按钮设置权限级别，然后添加到权限列表中。在用户组属性表单中，单点击更新。

Attention:
在 Zabbix 中，主机的访问权限被分配给用户组, 而不是单独的用户。

权限设置完成了！您可以尝试使用新用户的凭据登录。

2 新建主机

概述

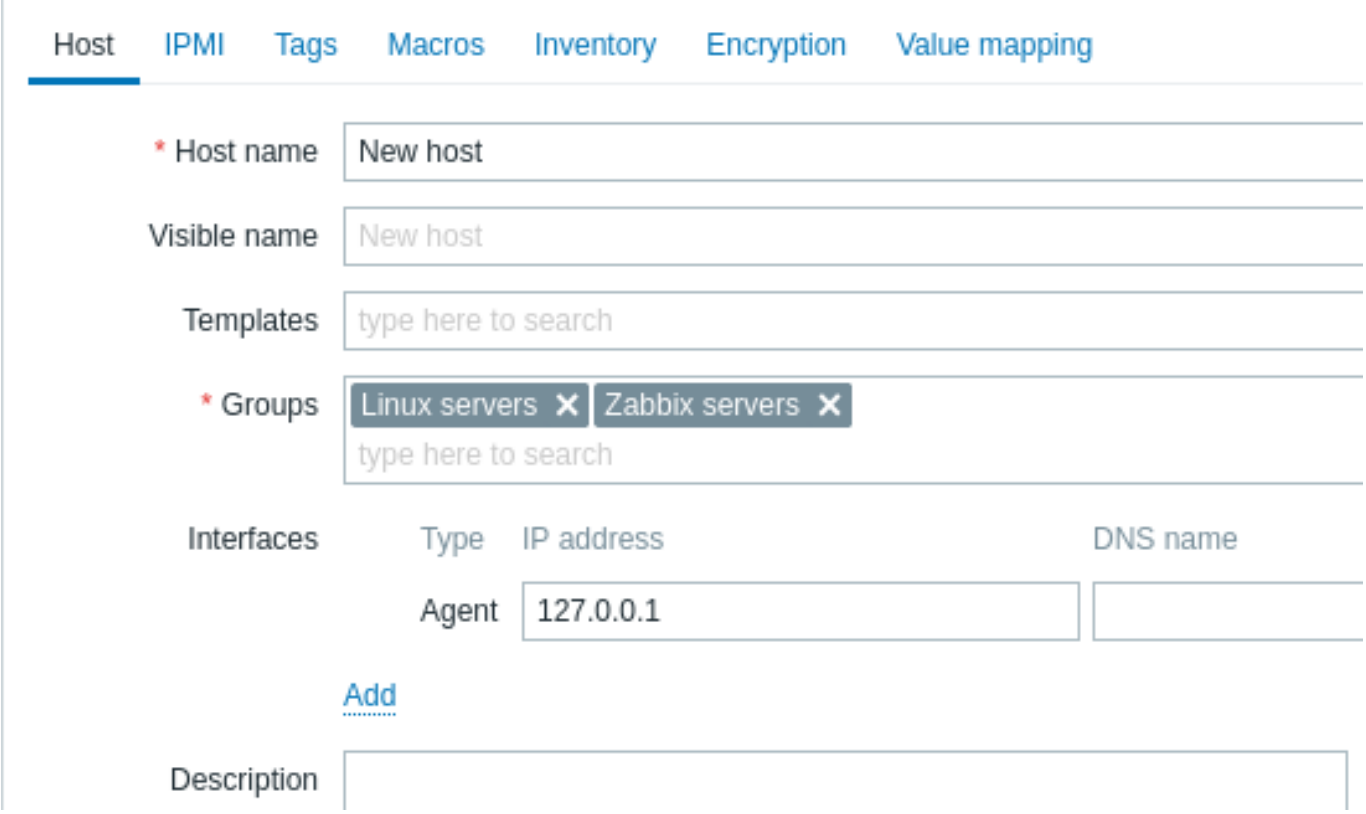
通过本节，你将会学习到如何创建一个新的主机。

Zabbix 中的主机是一个你想要监控的网络实体（物理的，或者虚拟的）。Zabbix 中，对于主机的定义非常灵活。它可以是一台物理服务器，一个网络交换机，一个虚拟机或者某些应用程序。

添加主机

Zabbix 中，可以通过配置 → 主机或者监测 → 主机，查看已配置的主机信息。默认已有一个名为’Zabbix server’ 的预先定义好的主机。但我们需要学习如何添加另一个。

点击创建主机来新增一台主机。将会展示出一个主机配置表。



所有必填字段均标有红色星号。

至少需要提供以下信息：

主机名

- 输入一个主机名。允许使用大小写字母、数字、空格、点、破折号和下划线。

主机组

- 通过单击选择按钮选择一个或多个现有组，或输入不存在的主机组名以创建新组。

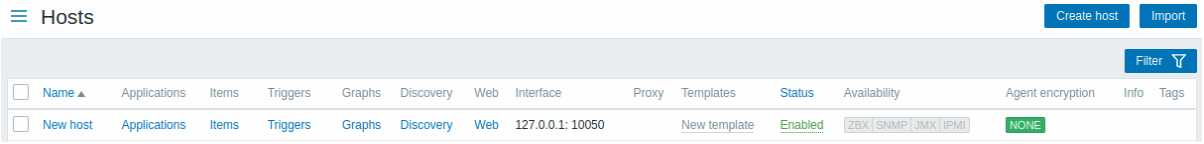
Note:
所有访问权限都是分配给主机组的，而不是单个主机。这就是为什么一个主机必须至少属于一个主机组。

接口***: IP*** 地址

- 虽然技术上它不是必填字段，但您可能希望输入主机的 IP 地址。请注意，如果这是 Zabbix server 的 IP 地址，必须在 Zabbix agent 配置文件中指定'Server' 参数值。

其他选项 我们暂时使用默认值。

当完成后，点击添加。你可以在主机列表中看到你新添加的主机。



可用性列包含每个接口的主机可用性指标。我们已经定义了 Zabbix 代理接口，因此我们可以使用代理可用性图标 (上面有 'ZBX') 来判断主机可用性：

- ZBX - 表示主机状态尚未建立，尚未发生监控指标检查
- ZBX - 表示主机可用，监控指标检查已成功
- ZBX - 表示主机不可用，监控指标检查失败（将鼠标光标移动到图标上以查看错误消息）。可能是由于接口凭证不正确造成了通信问题。检查 zabbix server 是否正在运行，并稍后尝试刷新页面。

3 新增监控项

概述

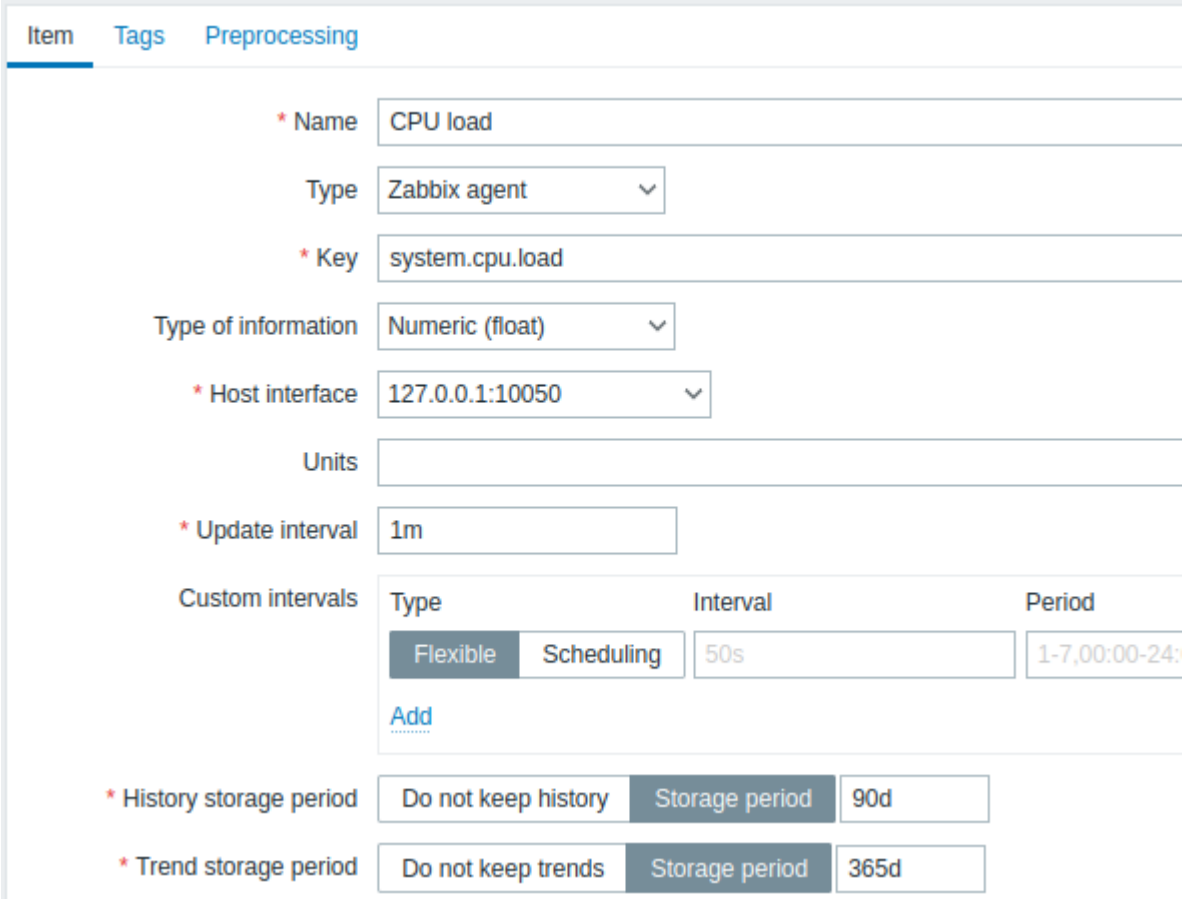
通过本节，你将会学习到如何创建一个新的监控项。

监控项是 Zabbix 中采集数据的基础。没有监控项，就没有数据——因为一个主机中只有监控项定义了单一的指标或者需要获得的数据。

添加监控项

所有的监控项都是依赖于主机的。这就是为什么我们要配置一个监控项时，先要进入配置 → 主机页面查找到新建的主机。

在" 新主机" 行中，点击监控项这个链接，然后点击创建监控项，将会展示一个监控项配置表。



所有必填字段均标有红色星号。

对于我们的示例，需要提供以下信息:

名称

- 输入 *CPU load* 作为值. 在列表和其他地方，都会显示这个值作为监控项名称。

键值

- 手动输入 system.cpu.load 作为值。这是监控项的一个技术上的名称，用于识别获取信息的类型。这个特定值需要是 Zabbix agent预定义键值的其中一种。

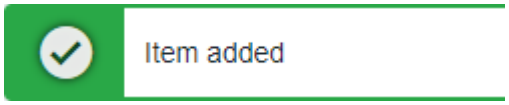
信息类型

- 此属性定义预期数据的格式。对于键值 *system.cpu.load* , 这个字段会自动设置成 浮点数.

Note:
您可能还希望减少监控项历史数据的保留天数为 7 天或 14 天。这是一种很好的做法，可以避免数据库保存大量的历史数据。

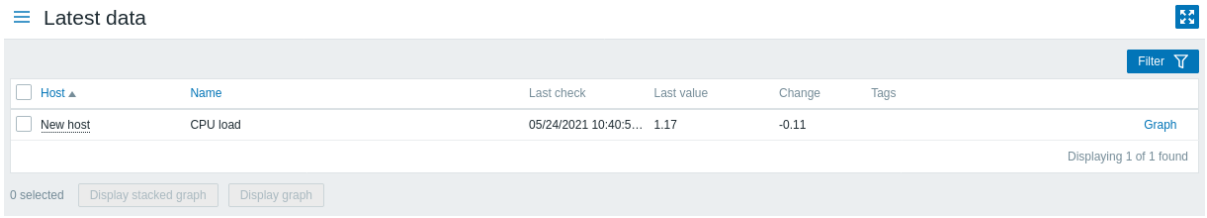
其他选项 我们暂时使用默认值。

当完成后，点击添加。新的监控项将出现在监控项列表中。点击列表中的详细以查看具体细节。



查看数据

当一个监控项创建完成后，你可能好奇它具体获得了什么值。前往 *Monitoring* → *Latest data*，在过滤器中选择刚才新建的主机，然后点击 *Apply*。



同时，第一次获得的监控项值最多可能需要 60 秒。默认情况下，这是服务器读取变化后的配置文件，获取并执行新的监控项的频率。

如果你在” 更改 “列中没有看到值，可能到目前为止只获得了一次值。等待 30 秒以获得新的监控项值。

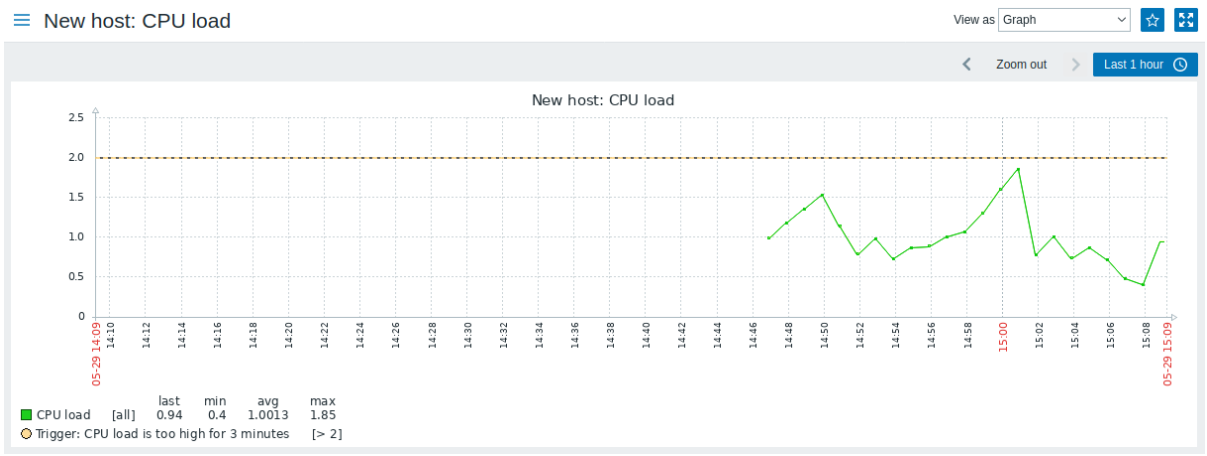
如果你在没有看到类似截图中的监控项信息，请确认：

- 你输入的监控项” 键值 “和” 信息类型 “同截图中的一致
- agent 和 server 都处于运行状态
- 主机状态为 “已启用” 并且它的可用性图标是绿色的
- 在主机的下拉菜单中已经选择了对应主机，且监控项处于已启用状态

图表

当监控项运行了一段时间后，可以查看可视化图表。简单图表 适用于任何信息类型为数值型的监控项，且不需要额外的配置。这些图表会在运行时生成。

前往 *Monitoring* → *Latest data*，然后点击监控项后的” 图表 “ ’ 链接来查看。



4 新建触发器

概述

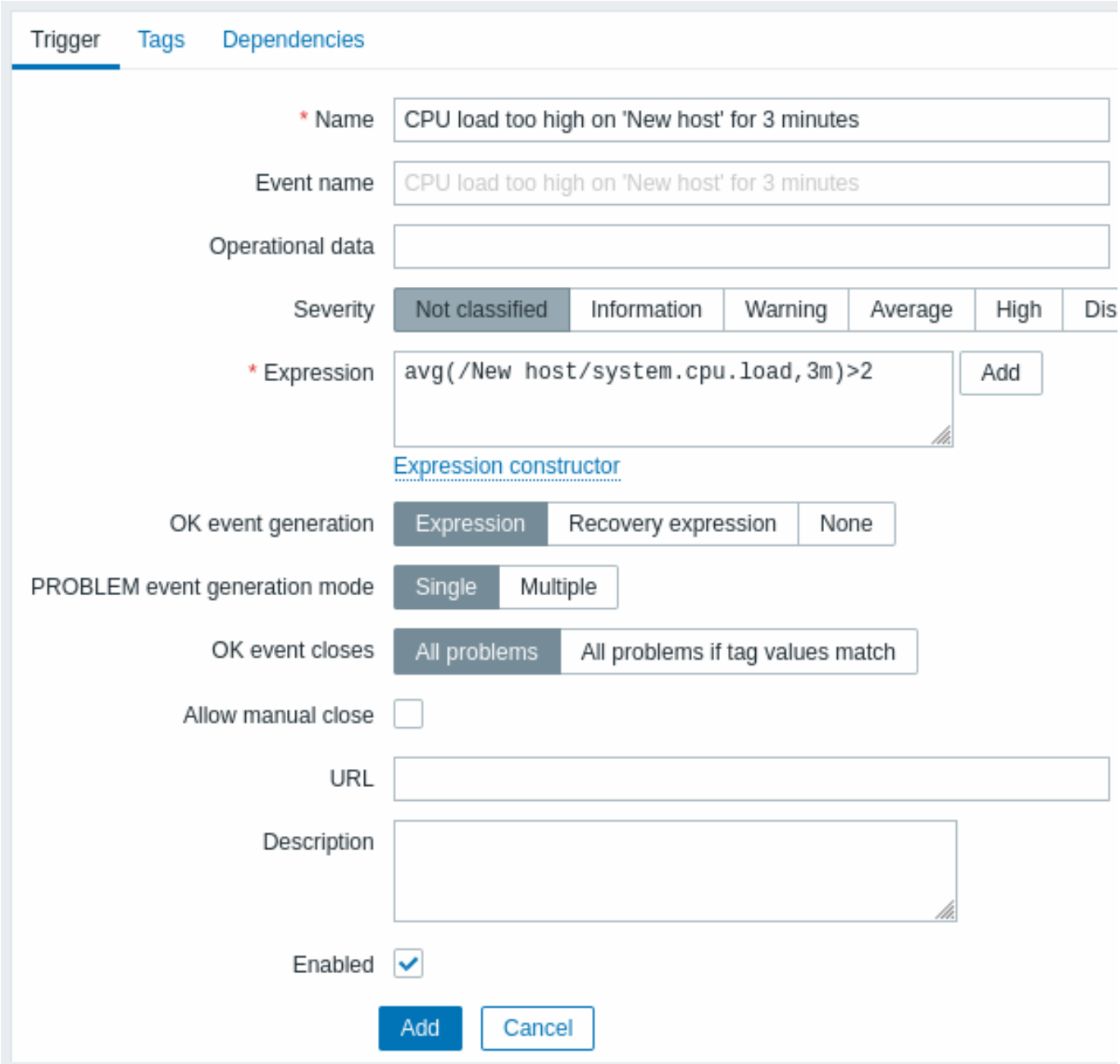
通过本节，你将会学习到如何创建一个新的触发器。

监控项只是用于收集数据. 如果需要自动评估收到的数据，我们需要定义触发器。触发器包含了一个表达式，这个表达式定义了数据的可接受的阈值级别。

如果收到的数据超过了这个定义好的级别，触发器将被 “触发”，或者进入 “问题” 状态——从而引起我们的注意，让我们知道有问题发生。如果数据再次恢复到合理的范围，触发器将返回 “ok” 状态。

添加触发器

为监控项配置触发器，前往配置 → 主机，找到” 新增的主机 “，点击旁边的触发器，然后点击创建触发器。将会展示一个触发器配置表。



对于我们的触发器，需要提供以下信息：

名称

- 输入 *CPU load too high on 'New host' for 3 minutes* 作为值。在列表和其他地方，都会显示这个值作为触发器名称。

表达式

- 输入: avg(/New host/system.cpu.load,3m)>2

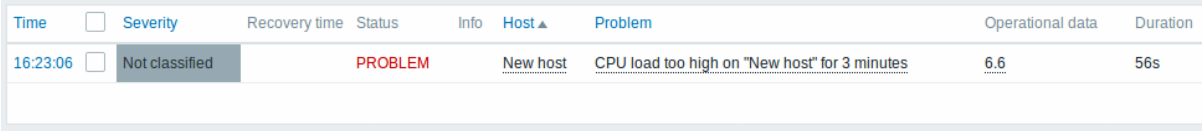
这个是触发器的表达式。确认这个表达式输入正确，直到最后一个符号。这个监控项键值 (system.cpu.load) 用于指出具体的监控项。这个特定的表达式大致是说如果 3 分钟内，CPU 负载的平均值超过 2，那么就触发了问题的阈值。你可以查看更多的触发器表达式语法。

完成后，点击添加。新的触发器将会显示在触发器列表中。

显示触发器状态

当一个触发器定义后，你可能想查看它的状态。

如果 CPU 负载超过了你在触发器中定义的阈值，这个问题将显示在监测 → 问题中。



状态栏中的闪烁表示最近触发状态的变化，即过去 30 分钟内发生的变化。

5 接收问题通知

概述

通过本节，你将会学习到如何在 Zabbix 中设置告警通知。

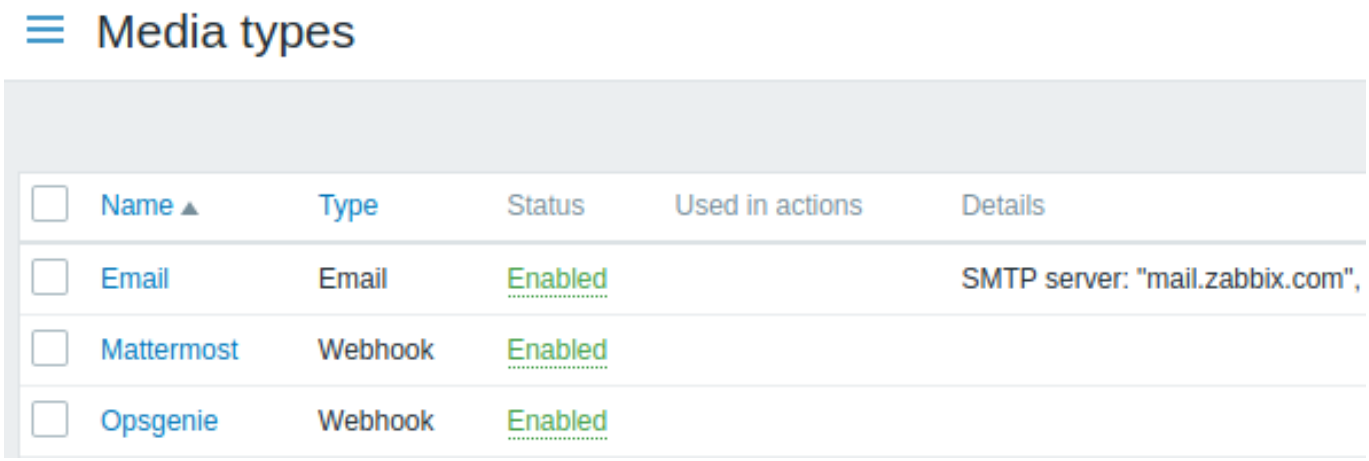
当监控项收集数据并且触发器处于问题的状态下，在系统中设定告警机制也很有用，这将使我们不需要盯着 Zabbix 前端也能收到一些重要的事件通知。

这就是通知的功能。电子邮件是最受欢迎的问题通知方式，我们将会学习如何设置电子邮件通知。

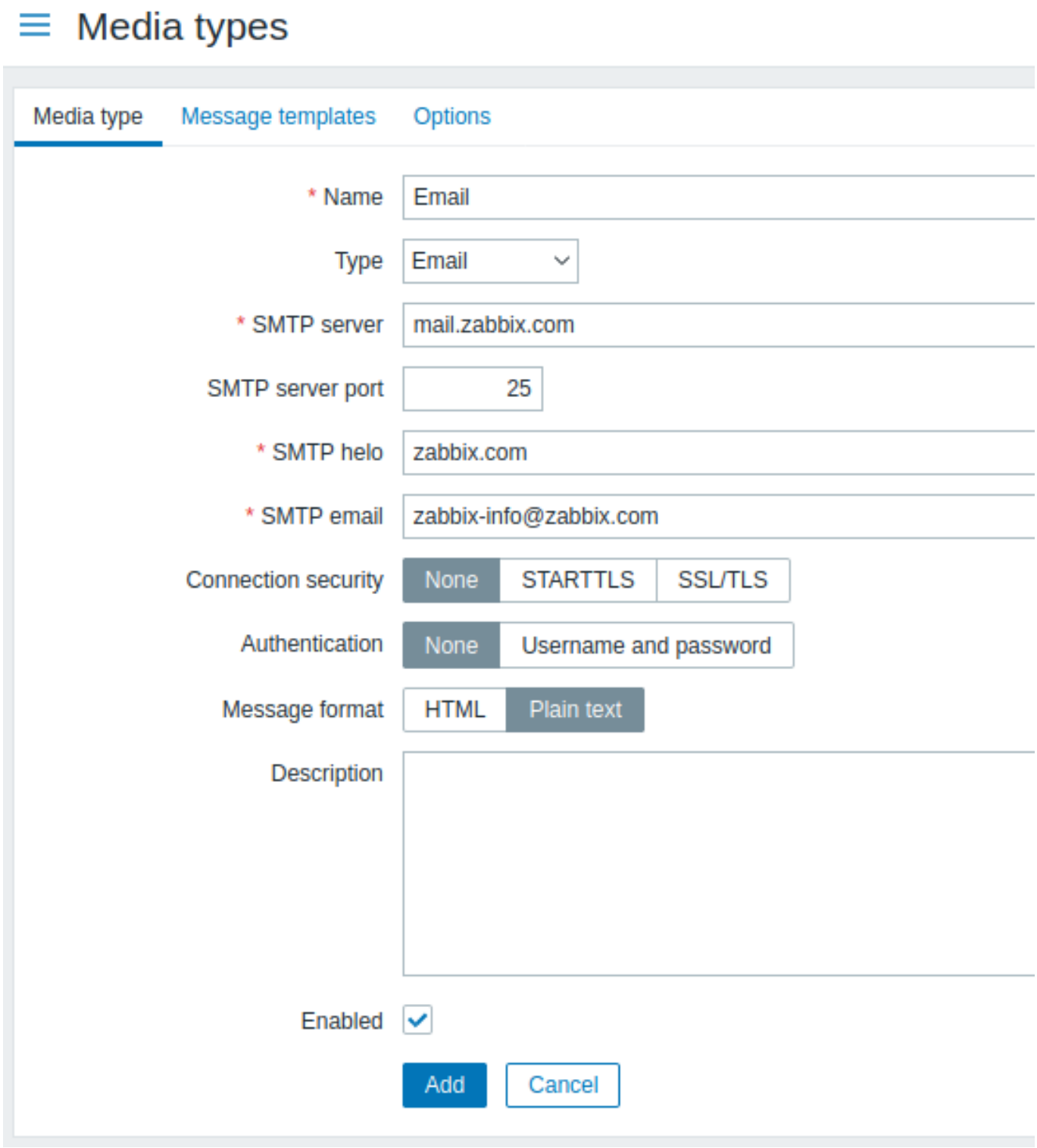
电子邮件设置

Zabbix 中最开始预定义了一些通知发送方式。电子邮件是其中的一种。

前往管理 → 媒介类型，点击预定义媒介类型列表中的 *Email*，以配置电子邮件。



这将为我们展示电子邮件设置定义表单。



所有必填字段均标有红色星号。

根据你的环境，设置 SMTP 服务器，SMTP HELO 以及 SMTP 电子邮件的值。

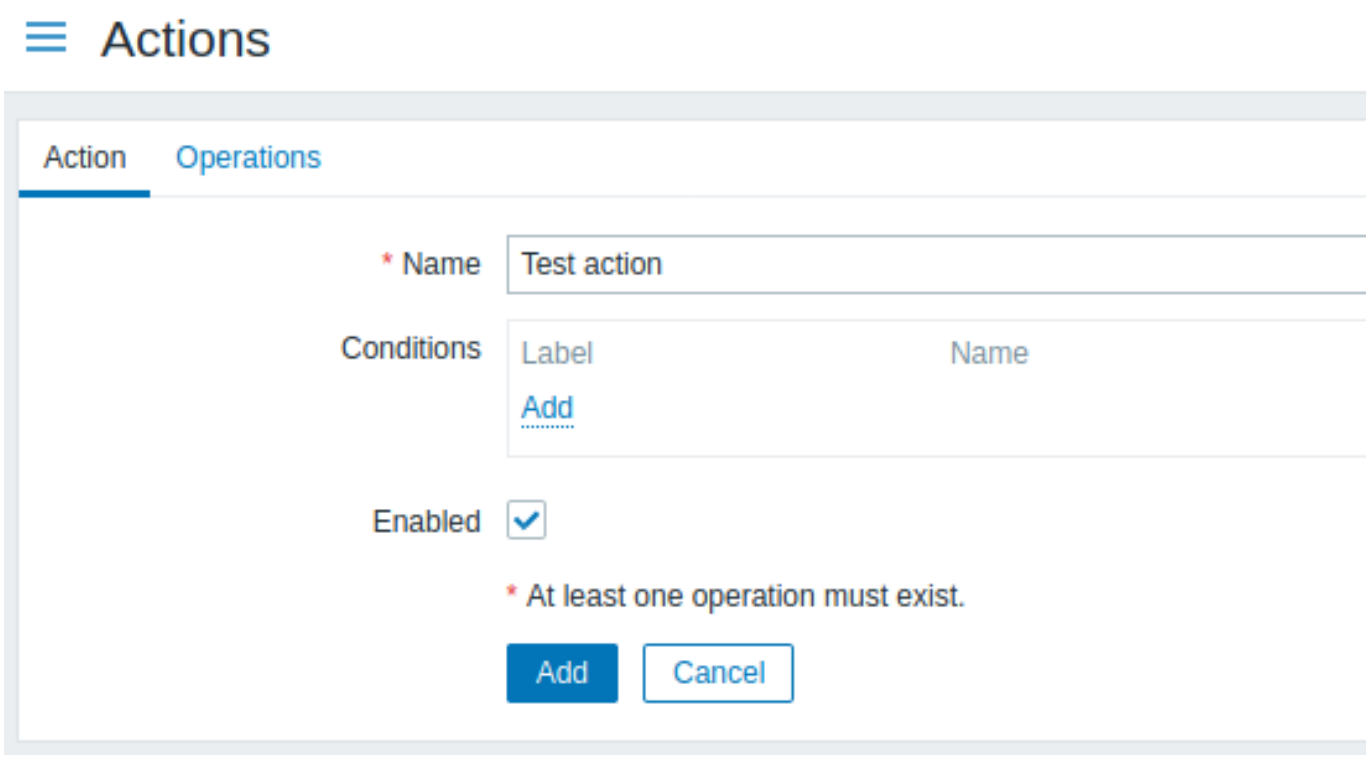
Note:
”SMTP 电子邮件 “ 将作为 Zabbix 通知的 “发件人” 地址。

一切就绪后，点击更新。

现在你已经配置了” 电子邮件 “作为一种可用的媒介类型。一个媒介类型必须通过发送地址来关联用户 (如同我们在配置一个新用户) 中做的，否则它将无法生效。

新建动作

发送通知是 Zabbix 中动作执行的操作之一。因此，为了建立一个通知，前往配置 → 动作，然后点击创建动作。

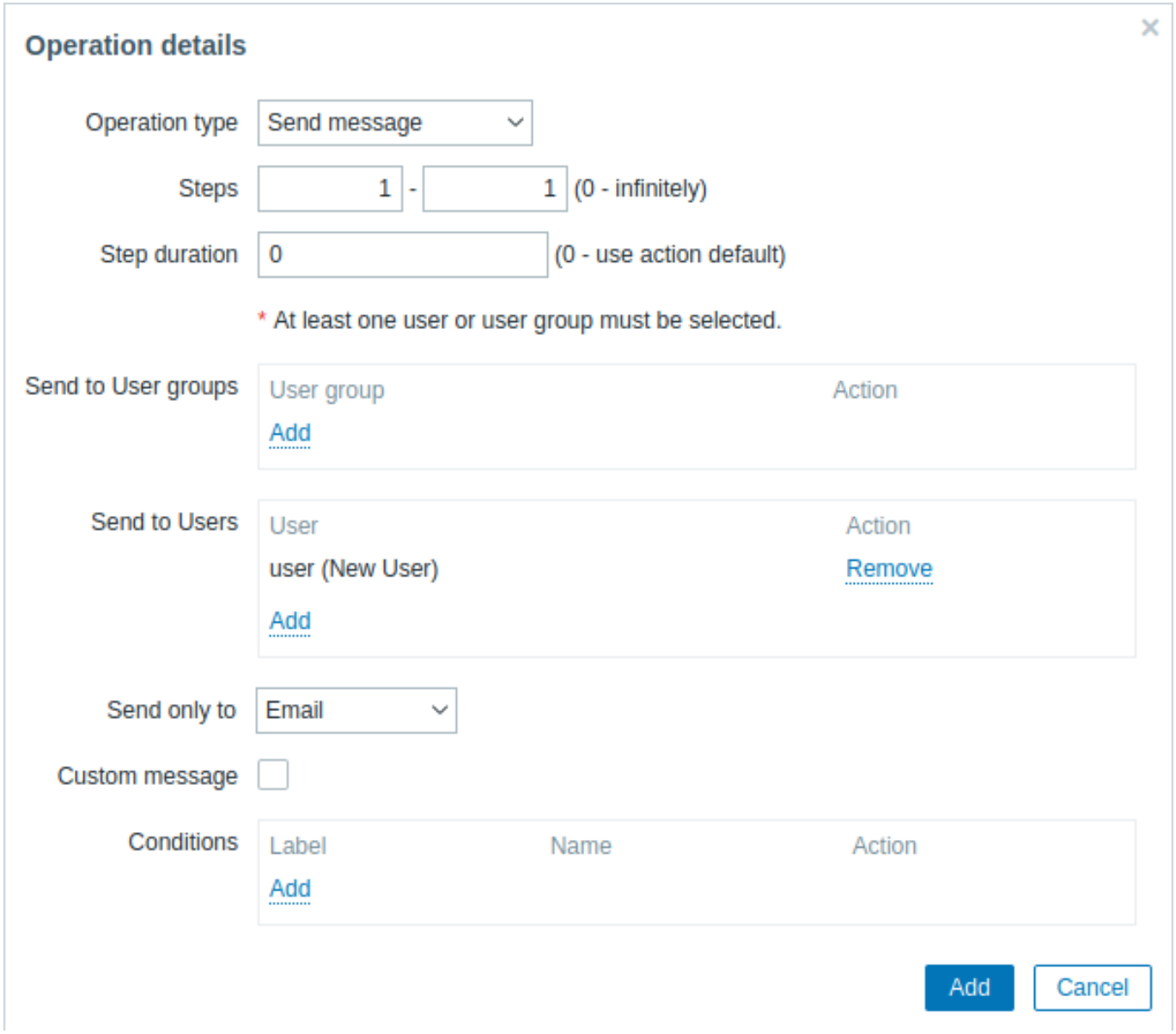


所有必填字段均标有红色星号。

在这个表单中，为动作输入一个名称。

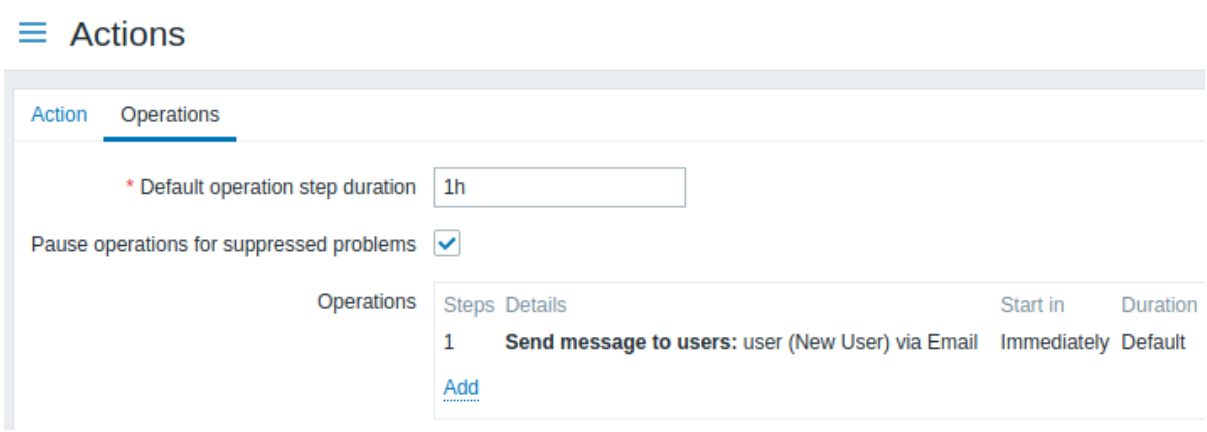
在最简单的情况下，如果我们不添加任何更具体的条件，动作将在从 “Ok” 到 “Problem” 的任何触发器状态更改时被执行。

我们还需要定义这个动作具体要做什么 —— 即在 操作标签页中执行的操作。点击操作块中的添加，将会打开一个操作表单。



所有必填字段均标有红色星号。

这里，在 *Send to users* 块中点击添加并选择我们之前定义的用户 ('user')。在仅送到中选择 *Email*。在完成后，点击添加，这个操作将会被添加：



这就是一个简单的动作配置，最后点击动作表单中的添加。

接收通知

现在，在配置了发送通知的情况下，实际接收一个通知会很有趣。为了实现这个目的，我们可能会故意增加主机上的负载——这样我们的触发器才会被触发，然后我们会收到问题通知。

打开主机的控制台，并运行：

```
cat /dev/urandom | md5sum
```

你可能需要运行一个或者多个 这样的进程.

现在，前往监测 → 最新数据，查看 “CPU Load” 的值是否已经增长。记住，为了使我们的触发器被触发，“CPU Load” 的值需要在在 3 分钟运行的过程中超过 2。一旦满足这个条件：

- 在监测 → 问题中，你可以看到闪烁 “问题” 状态的触发器。
- 你的电子邮件中，会收到一个问题通知。

Attention:
如果通知功能没有正常工作：
- 再次验证电子邮件设置和动作设置已经被正确配置
- 确认你创建的用户对生成事件的主机至少拥有读权限。正如添加用户 步骤中提到的。“Zabbix 管理员” 用户组中的用户必须对’Linux servers’ 主机组（该主机所属组）至少拥有读权限。
- 另外，你可以在报表 → 动作日志中检查动作日志。

6 新建模板

概述

通过本节，你将会学习到如何创建一个新的模板。

之前我们学习了如何创建监控项、触发器以及如何获取主机的问题通知。

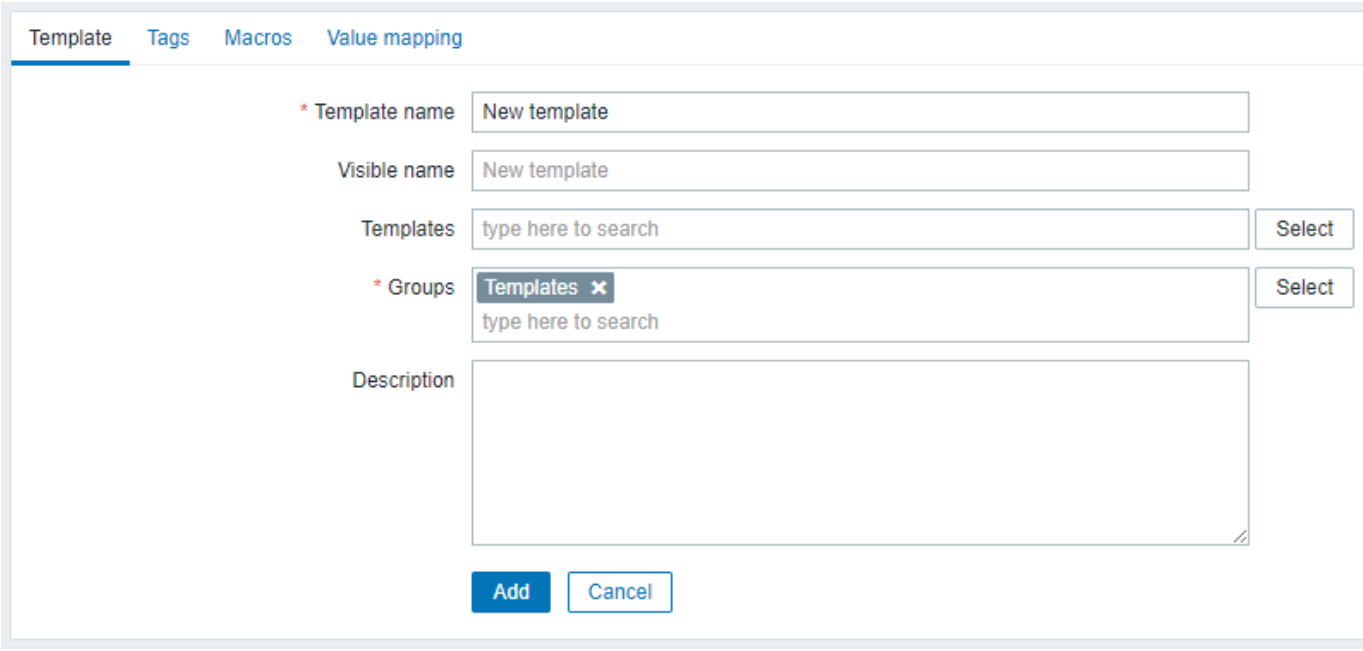
虽然这些步骤提供了很大的灵活性，但仍然需要很多步骤才能完成。如果我们需要配置上千台主机，一些自动化操作会带来更多便利性。

模版功能可以实现这一点。模版允许对有用的监控项、触发器和其他对象进行分组，只需要一步就可以对监控主机应用模版，以达到重复使用的目的。

当一个模版链接到一个主机后，主机会继承这个模版中的所有对象。简单而言，一组预先定义好的检查会被快速应用到主机上。

添加模板

在开始使用模版之前，你必须先创建一个。在配置 → 模版中，点击创建模版。将会展示出一个模版配置表。



所有必填字段均标有红色星号。

此处需要输入的参数包括：

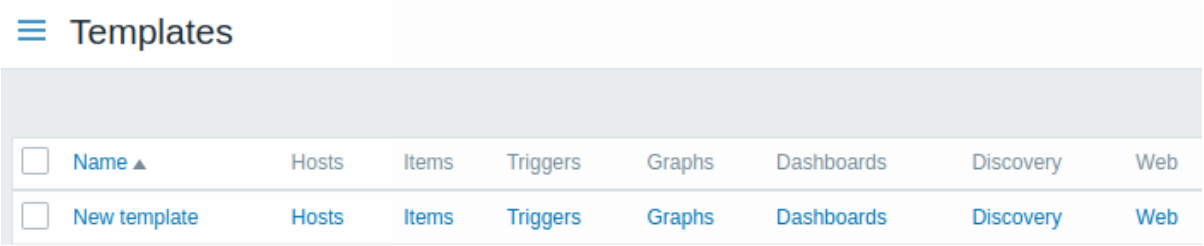
模板名称

- 输入一个模板名称。允许使用大小写字母、数字、空格和下划线。

主机组

- 通过单击选择按钮选择一个或多个现有组。模板必须属于某个主机组。

完成后，点击添加。你可以在模板列表中看到你新添加的模板。



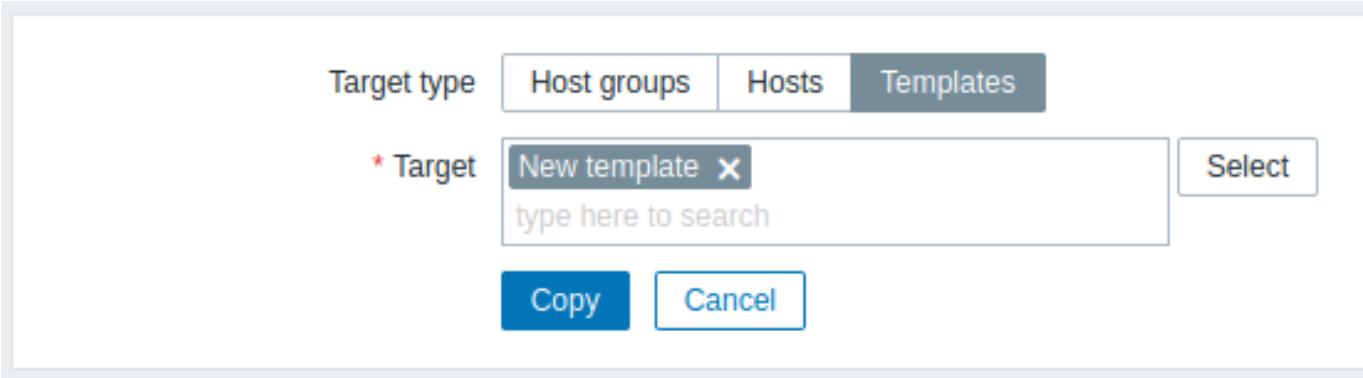
你可以在这看到模版信息，但里面什么都没有——没有监控项、触发器或其他其他对象。

在模版中添加监控项

在模版中添加监控项，前往“新建主机“的监控项列表。在配置 → 主机，点击“新建主机“旁边的监控项。

然后：

- 选中列表中“CPU 负载” 监控项的选择框
- 点击列表下方的复制
- 选择要复制这个监控项的目标模版



所有必填字段均标有红色星号。

- 点击复制

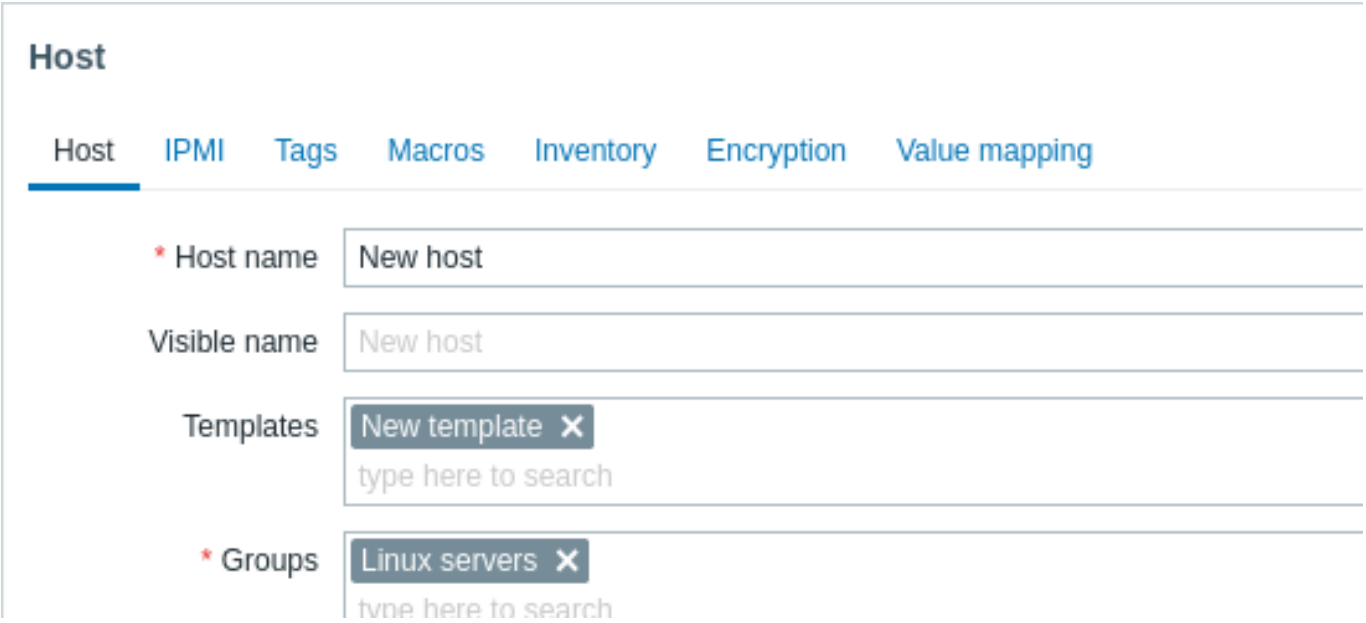
你现在可以前往配置 → 模版，” 新模板” 模版中会有一个新的监控项。

我们目前只创建了一个监控项，但你可以用同样的方法在模版中添加其他的监控项，触发器以及其他对象，直到满足特定需求（如监控操作系统，监控单个应用）的完整的对象组合。

链接模版到主机

准备好一个模版后，将它链接到一个主机。前往配置 → 主机，点击“新建主机” 打开其属性表单，并找到模板字段。

开始在模板字段中键入新建模板。我们创建的模板名称应该出现在下拉列表中。向下滚动选择。查看它是否出现在模板字段中。



单击表单中的更新保存更改。现在，模板和它所持有的所有对象都被添加到主机中。

正如您可能已经猜到的，这种方法也可以应用于任何其他主机。在模板级别对监控项、触发器和其他对象的任何更改都将传播到模板链接到的主机。

链接预定义模版到主机

正如您可能已经注意到的，Zabbix 为各种操作系统、设备和应用程序提供了一组预定义的模板。要快速开始监控，您可以将适当的模板链接到主机，但要注意，这些模板需要根据您的环境进行微调。一些检查可能不需要，轮询间隔可能过于频繁。

可参考该链接，查看更多关于模板的信息。

6. Zabbix Appliance

概述　作为手动设置或重用现有 Zabbix 服务器的替代方法，用户可以下载 Zabbix appliance 或 Zabbix appliance 安装 CD image。

Zabbix appliance 及安装 CD 基于 AlmaLinux 8 (x86_64)。

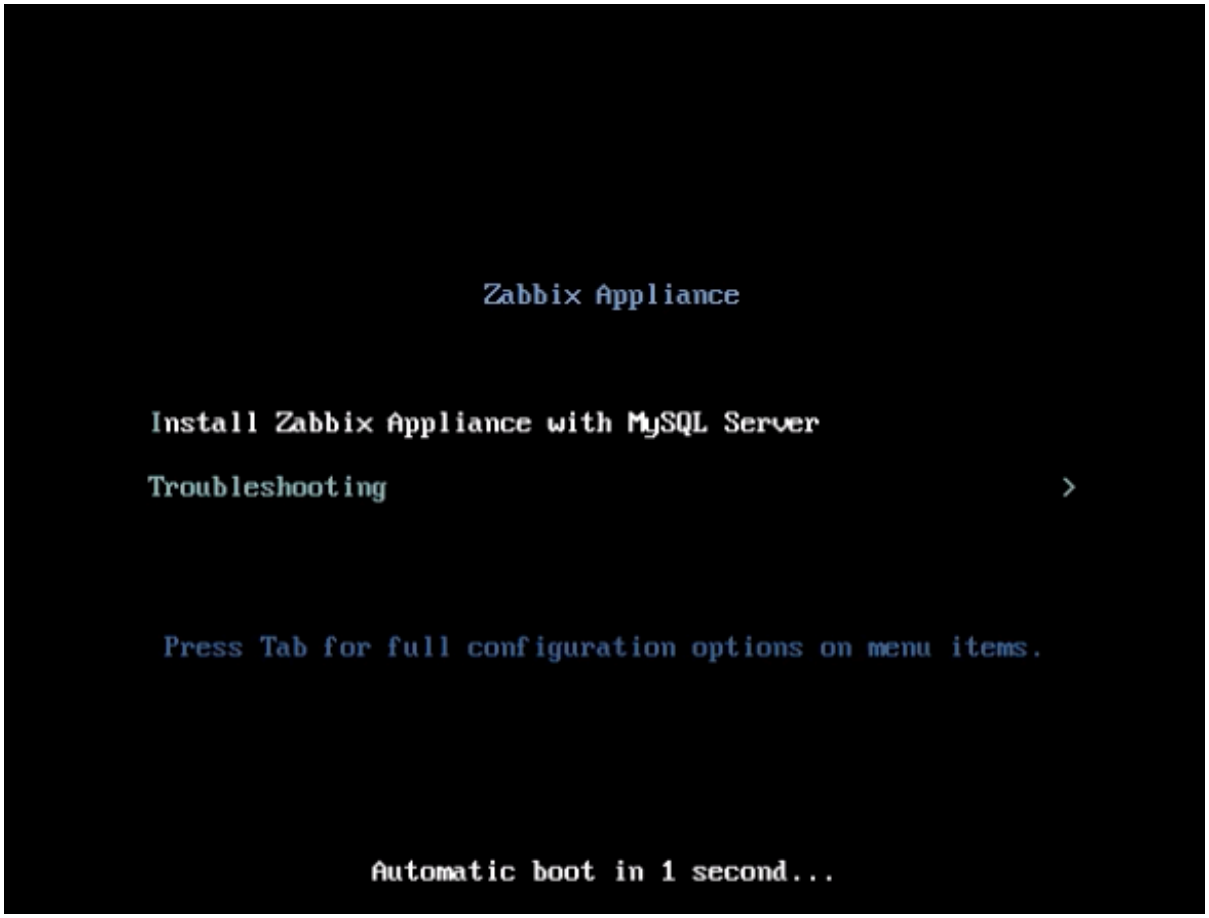
Zabbix appliance 安装 CD 可用于 Zabbix server (MySQL) 的即时部署。

> **Attention:**
> 您可以使用 Appliance 来评估 Zabbix。这个 Appliance 不为重要的生产用途设计。

系统需求：

- *RAM*: 1.5 GB
- 磁盘空间: 应至少为虚拟机分配 8 GB。

Zabbix 安装 CD/DVD 的 boot 菜单：



Zabbix Appliance 包含一个 Zabbix server (已配置并在 MySQL 上运行) 及一个前端。

Zabbix Appliance 提供如下格式的虚拟机 image：

- VMWare (.vmx)
- Open virtualization format (.ovf)

- Microsoft Hyper-V 2012 (.vhdx)
- Microsoft Hyper-V 2008 (.vhd)
- KVM, Parallels, QEMU, USB stick, VirtualBox, Xen (.raw)
- KVM, QEMU (.qcow2)

要开始使用，请 boot Appliance 并通过浏览器访问 Appliance 通过 DHCP 接收的 IP。

Attention:
必须在主机上启用 DHCP。

在虚拟机内部查看 IP 地址，可以执行：

```
ip addr show
```

要访问 Zabbix 前端，可以访问 **http://<host_ip>** （应在虚机网络设置中启用桥接模式以便从主机的浏览器访问）。

Note:
如果 Appliance 在 Hyper-V 上启动失败，你可以按下 `Ctrl+Alt+F2` 以切换 tty session.

1 对 **AlmaLinux 8** 配置的更改　Appliance 基于 AlmaLinux 8。有一些配置与基本的 AlmaLinux 设置有一定区别。

1.1 仓库

官方的 Zabbix 仓库 已经被添加到 */etc/yum.repos.d*：

```
[zabbix]
name=Zabbix Official Repository - $basearch
baseurl=http://repo.zabbix.com/zabbix/6.0/rhel/8/$basearch/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-ZABBIX-A14FE591
```

1.2 防火墙配置

Appliance 使用 iptables 防火墙预定义了一些规则：

- 开启了 SSH 端口 (22 TCP)；
- 开启了 Zabbix agent (10050 TCP) 及 Zabbix trapper (10051 TCP) 端口；
- 开启了 HTTP (80 TCP) 及 HTTPS (443 TCP) 端口；
- 开启了 SNMP trap 端口 (162 UDP)；
- 开启了连接到 NTP 的端口 (53 UDP)；
- ICMP 数据包限制为每秒 5 个；
- 所有其他传入连接都将断开。

1.3 使用静态 IP 地址

默认情况下，Appliance 使用 DHCP 来获取 IP 地址。如果要指定一个静态 IP 地址：

- 使用 root 用户登录；
- 打开 */etc/sysconfig/network-scripts/ifcfg-eth0* 文件；
- 将 *BOOTPROTO=dhcp* 替换为 *BOOTPROTO=none* ；
- 添加如下行：
  - *IPADDR=<Appliance* 的 *IP* 地址 *>*
  - *PREFIX=<CIDR* 前缀 *>*
  - *GATEWAY=<* 网关 *IP* 地址 *>*
  - *DNS1=<DNS* 服务器 *IP* 地址 *>*
- 执行 **systemctl restart network** 命令。

如果需要的话，查询 Red Hat 官方文档

1.4 更改时区

应用默认使用 UTC 作为系统时钟。如需更改时区，那么从 */usr/share/zoneinfo* 中复制合适的文件到 */etc/localtime* 中，例如：

```
cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
```

2 Zabbix 配置　Zabbix Appliance 安装过程中使用了下列密码和配置：

2.1 凭证信息 (用户名: 密码)

系统：

- root:zabbix

Zabbix 前端：

- Admin:zabbix

数据库：

- root:<random>
- zabbix:<random>

> **Note:**
> 数据库密码是在安装过程中随机生成的。
> Root 密码存储在 /root/.my.cnf 文件中。不需要在”root” 用户下输入密码。

要更改数据库用户密码，必须在以下位置同时改变配置：

- MySQL;
- /etc/zabbix/zabbix_server.conf;
- /etc/zabbix/web/zabbix.conf.php.

> **Note:**
> 分别为 Zabbix Server 和 Zabbix 前端定义了单独的用户 `zabbix_srv` 及 `zabbix_web`。

2.2 文件路径

- 配置文件位于 **/etc/zabbix**.
- Zabbix server，proxy 及 agent 的日志文件位于 **/var/log/zabbix**。
- Zabbix 前端位于 **/usr/share/zabbix**。
- **zabbix** 用户的 home 目录位于 **/var/lib/zabbix**。

2.3 对 Zabbix 配置的更改

- 前端时区已被设置为 Europe/Riga （可以在 **/etc/php-fpm.d/zabbix.conf** 中修改）；

3 访问前端　默认情况下，允许从任何位置访问前端。

前端可以从 *http://<host>* 进行访问。

可在 **/etc/nginx/conf.d/zabbix.conf** 中修改此设置。在修改此文件后必须重启 Nginx。为此，请使用 SSH 以 **root** 用户身份登录并执行：

```
systemctl restart nginx
```

4 防火墙　默认情况下，只有配置更改 上列举的端口是开放的。要添加额外的端口，可编辑”*/etc/sysconfig/iptables*” 文件并重新加载防火墙规则。

```
systemctl reload iptables
```

5 升级　Zabbix Appliance 的包可以升级。为此，可以执行：

```
dnf update zabbix*
```

6 系统服务　在 Systemd 中列举 Zabbix 的 service：

```
systemctl list-units zabbix*
```

7 特定 **image** 的说明　7.1 VMware

vmdk 格式的 image 可直接在 VMware Player、Server 和 Workstation 产品中使用。要在 ESX,、ESXi 和 vSphere 中使用，它们必须使用 VMware converter 进行转换。

7.2 HDD/flash image (raw)

```
dd if=./zabbix_appliance_5.2.0.raw of=/dev/sdc bs=4k conv=fdatasync
```

将 */dev/sdc* 替换为你的 Flash/HDD 磁盘设备。

7. 配置

请使用左侧导航栏来访问 “配置” 这一章节的内容。

1 配置模板

概述

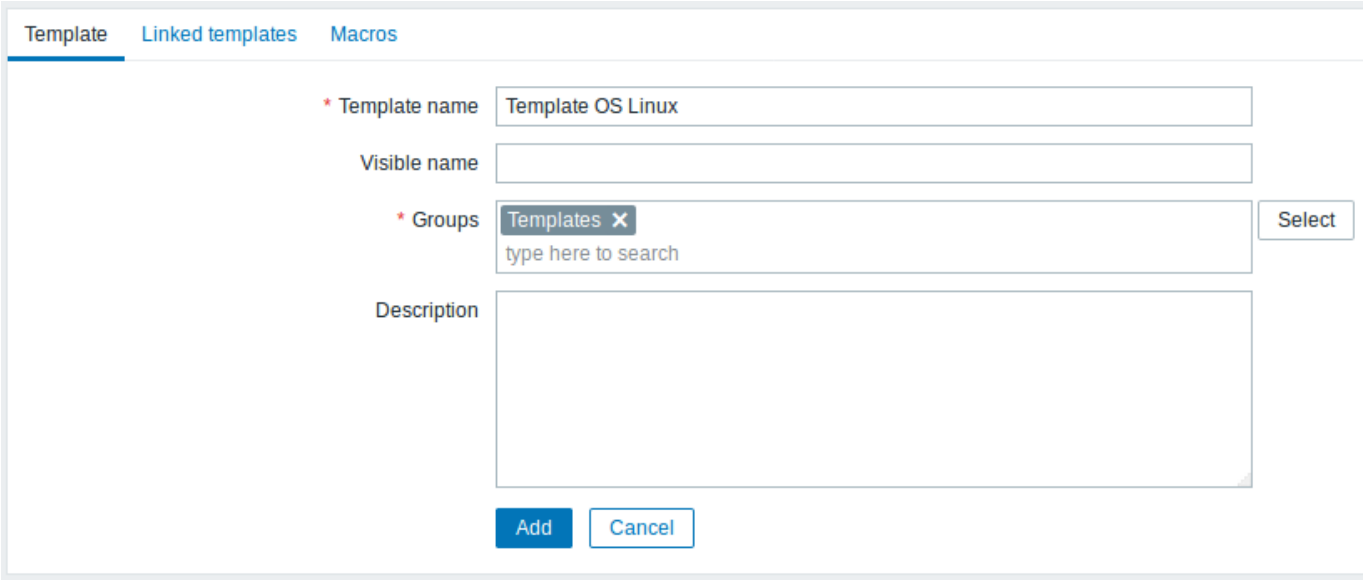
配置模板需要首先通过定义一些常规参数来创建模板，然后添加实体（监控项，触发器，图表等）。

创建模板

要创建模板，请执行以下操作：

- 转到配置 → 模板
- 点击创建模板
- 编辑模板属性

模板选项卡包含常规模板属性。

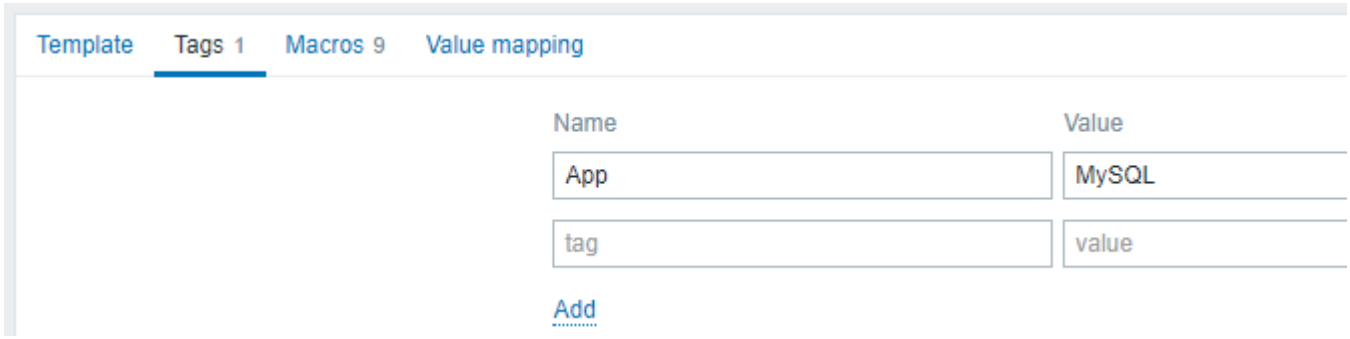


所有必填字段都标有红色星号。

模板属性：

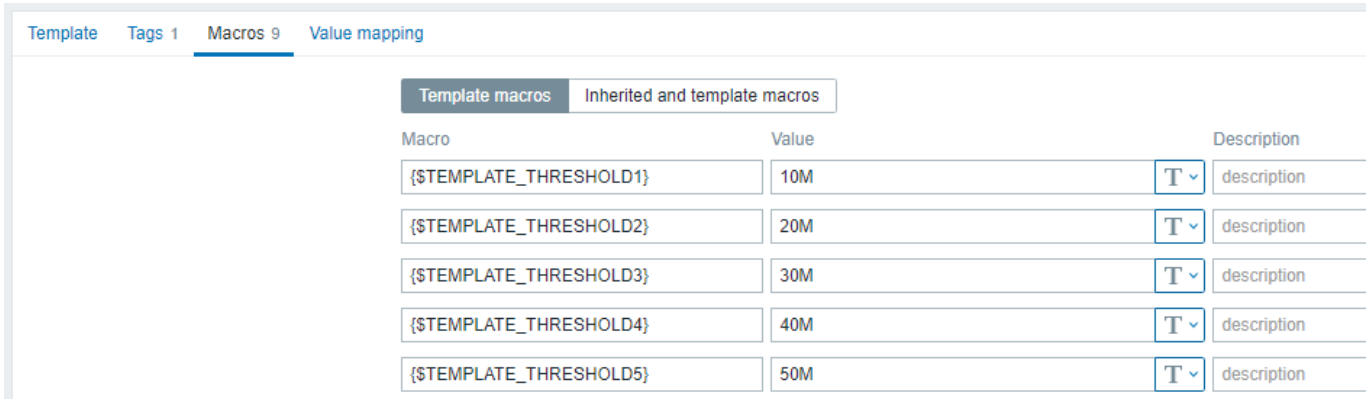
参数	描述
模板名称	唯一的模板名称。允许使用字母数字、空格、点、破折号和下划线。但是，空格是不允许在首尾使用。
可见名称	如果你设置了此名称，那么它将在列表，地图等中可见的。
模板	将一个或多个 “嵌套” 模板链接到此模板。所有实体（项目、触发器、图表等）都将从链接的模板中继承。 要链接新模板，请开始在链接新模板字段中输入模板名称。将出现匹配模板列表；向下滚动以选择。或者，您可以单击字段旁边的 选择并从弹出窗口的列表中选择模板。在保存或更新模板配置表单时，在 链接新模字段中选择的模板将被链接。 要取消链接模板，请使用 链接模板块中的两个选项之一: 取消链接 - 取消链接模板，但保留其监控项、触发器和图形 取消链接并清除 - 取消链接模板并删除其所有监控项、触发器和图形
群组	模板所属的主机/模板组。
描述	输入模板说明。

标签选项卡允许您定义模板级别标签。链接了此模板的所有主机的所有问题，将使用此处输入的标签。

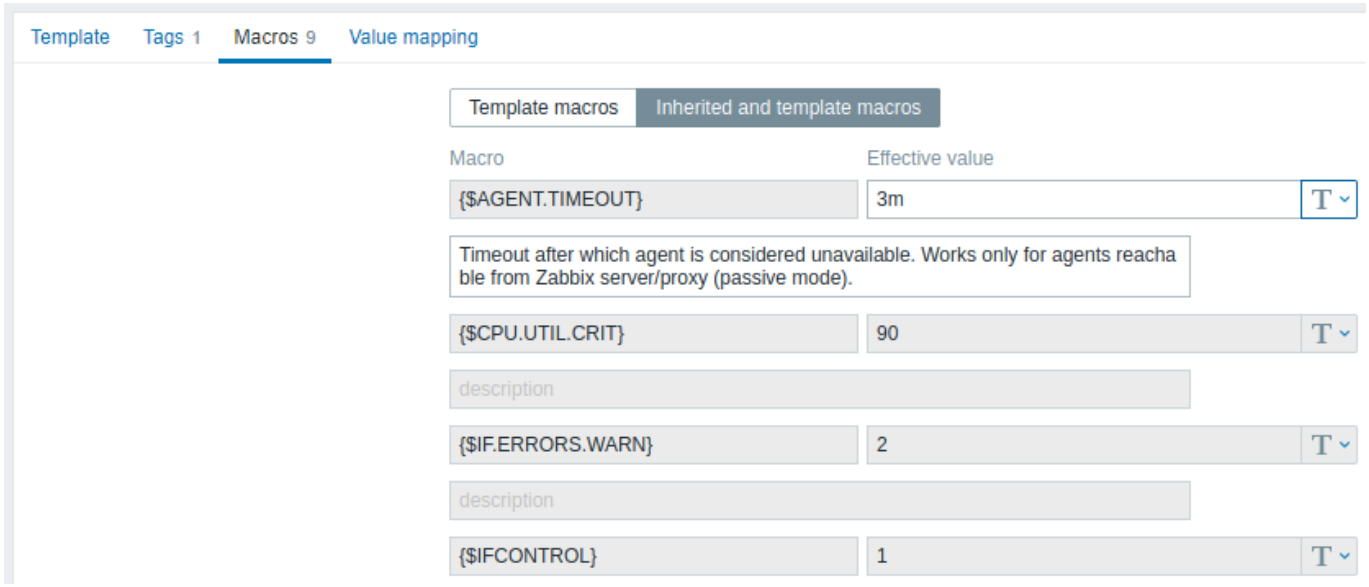


标签支持用户宏、{INVENTORY.*} 宏，{HOST.HOST}，{HOST.NAME},{HOST.CONN}，{HOST.DNS}，{HOST.IP}，{HOST.PORT} 和 {HOST.ID} 宏。

宏选项卡允许您将模板级用户宏 定义为名称-值键值对。请注意，宏的值可以保存为纯文本、机密文本或 Vault 秘钥。还支持添加描述。



如果选择了继承模板的宏选项，则还可以从链接的模板和全局宏中查看宏。在这里，模板的所有定义的用户宏都显示了它们所决定的值以及它们的起源。



为方便起见，提供了相应模板和全局宏配置的链接。也可以在模板级别上编辑嵌套模板/全局宏，有效地创建模板上宏的副本。

值映射选项卡允许在值映射中配置监控项数据的人性化展示。

按钮:

Add	添加模板。添加的模板应该出现在列表中。
Update	更新现有模板的属性。
Clone	根据当前模板的属性创建另一个模板，包括从链接模板继承的实体（监控项，触发器等）

Full clone	基于当前模板的属性创建另一个模板，包括从链接的模板继承并直接附加到当前模板的实体（监控项，触发器等
Delete	删除模板; 模板（监控项，触发器等）的实体与链接的主机保留。
Delete and clear	从链接的主机中删除模板及其所有实体。
Cancel	取消编辑模板属性。

创建一个模板，可以开始添加一些实体。

> **Attention:**
> 监控项必须首先添加到模板中。如果没有相应的监控项，则无法添加触发器和图形。

添加监控项，触发器，图形

要向模板添监控项，请执行以下操作：

- 转到配置 → 主机（或模板）
- 单击所需主机/模板行中的监控项
- 标记要添加到模板的监控项的复选框
- 点击监控项列表下面的复制
- 选择要复制的监控项的模板（或模板组），然后单击复制

所有选定的监控项都应该被复制到模板中。

添加触发器和图形以类似的方式完成（分别从触发器和图形列表），请记住，只有在首先添加所需监控项时，才能添加它们。

添加仪表盘

要在配置 → 模板中添加仪表盘，请执行以下操作：

- 点击模板行中的仪表盘
- 按照通常的配置仪表盘的方法配置仪表盘

> **Attention:**
> 可以包含在模板仪表板中的小部件有：经典图形、图形原型、时钟、纯文本、URL。

> **Note:**
> 有关访问从模板仪表板创建的主机仪表板的详细信息，请参阅主机仪表盘。

配置低级别自动发现规则

请参阅手册的低级别自动发现部分。

添加 Web 场景

在模板的配置 → 模板中添加 Web 场景，请执行以下操作：

- 点击模板行中的 *Web*
- 按照配置 Web 场景的通常方式配置 Web 场景

2 链接/取消链接

概述

链接是将模板应用于主机的过程，而取消链接将从主机中删除与模板的关联。

> **Attention:**
> 模板直接链接到各个主机，而不是主机组。只需将模板添加到主机组就不会链接到主机组。主机组仅用于主机和模板的逻辑分组。

链接模板

要将模板链接到主机，请执行以下操作：

- 转到配置 → 主机
- 单击所需的主机
- 开始在 模板字段中输入模板名称。将出现匹配的模板列表；向下滚动以选择。
- 或者，您可以单击旁边的 选择字段，并从弹出窗口的列表中选择一个或多个模板，在弹出窗口中选择一个或多个模板
- 单击主机属性表单中的添加/更新

主机现在将拥有模板的所有实体（监控项，触发器，图形等）。

Attention:
如果在这些模板中具有相同监控项键的监控项，则将多个模板链接到同一主机将失败。并且作为触发器和图形使用的监控项，如果使用相同的监控项键，它们也不能从多个模板链接到单个主机。

当从模板添加实体（监控项，触发器，图表等）时：

- 主机上先前存在的相同实体被更新为模板的实体，并且任何现有实体的主机级自定义都将丢失
- 添加模板中的实体
- 在模板连接之前，只存在于主机上的任何直接链接的实体保持不变

在列表中，模板中的所有实体都以模板名称为前缀，表示这些属于特定模板。模板名称本身（灰色文本）是一个链接，允许在模板级别访问这些实体的列表。

如果某个实体（监控项，触发器，图表等）没有以模板名称为前缀，则意味着它之前存在于主机上并且不是由模板添加的。

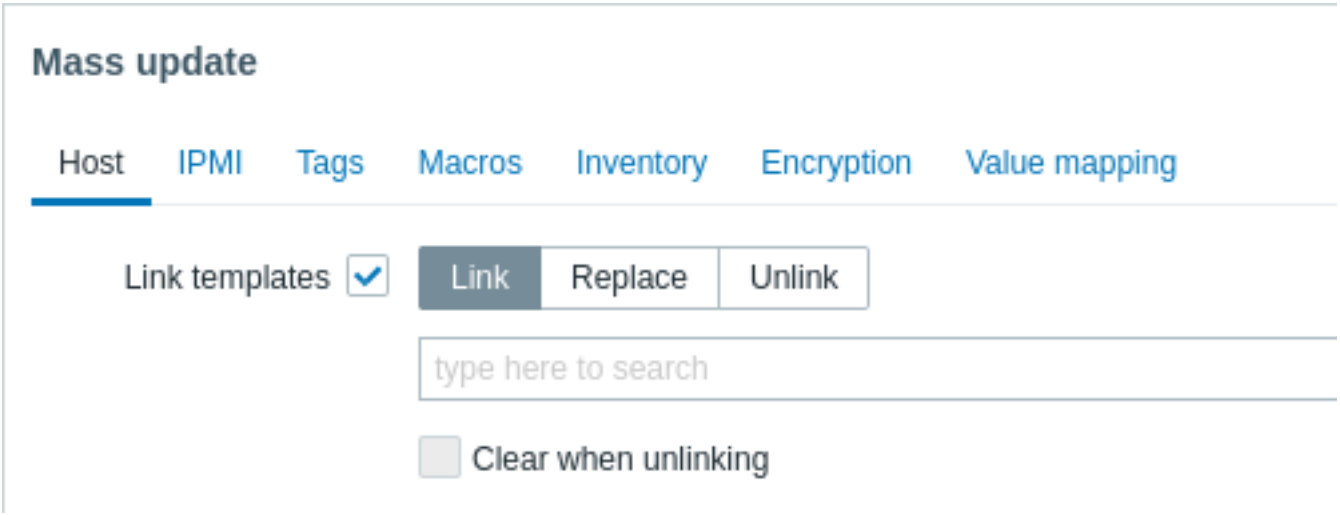
实体唯一性标准

从模板中添加实体（监控项，触发器，图表等）时，重要的是要知道这些实体已经存在于主机上并需要更新，哪些实体有所不同。决定同一性/差异的唯一性标准是：

- 用于监控项 - 监控项键
- 用于触发器 - 触发器名称和表达式
- 用于自定义图形 - 图形名称及其监控项

将模板链接到多个主机

要更新许多主机的模板链接，在配置 → 主机中通过标记其复选框来选择一些主机，然后单击列表下方的 批量更新，然后选择链接模板：



要链接其他模板，在自动完成字段中开始输入模板名称，直到出现一个提供匹配模板的下拉列表。只需向下滚动即可选择要链接的模板。

替换选项将允许取消关联之前被连接到主机的任何模板的同时，链接一个新的模板。取消链接选项将允许取消链接指定的模板。取消链接并清理选项不仅允许取消链接任何以前链接的模板，也从中移除（监控项，触发器等）继承的所有元素。

Note:
Zabbix 提供了大量预定义的模板。您可以使用这些作为参考，但请注意在生产中原封不动地使用它们，因为它们可能包含太多监控项，并且太频繁地轮询数据。如果您想使用它们，请对其进行微调以适合您的需要。

编辑链接实体

如果您尝试编辑从模板链接的监控项或触发器，您可能会发现许多关键选项无法进行编辑。这是有道理的，因为模板的想法是在模板级别以一键式方式编辑内容。但是，您仍然可以，例如，在单个主机上启用/禁用监控项并设置更新间隔、历史长度和一些其他参数。

Attention:
对在模板级别实现的实体的任何自定义将覆盖先前在主机级别对实体的自定义。

如果你想完全编辑实体，你必须在模板级别编辑它（模板级别的快捷方式显示在表单名称中），请记住这些更改将影响所有链接到此模板的主机。

取消链接模板

要从主机中取消链接模板，请执行以下操作：

- 转到 配置 → 主机
- 单击所需的主机并切换到模板选项卡
- 单击模板旁边的取消链接或取消链接并清理以取消链接
- 单击主机属性窗体中的更新

选择取消链接选项将简单地删除与模板的关联，同时将其所有实体（监控项，触发器，图形等）与主机保持一致。

选择取消链接并清理选项将删除与模板及其所有实体（监控项，触发器，图表等）的关联。

3 嵌套

概述

嵌套是一个模板包含一个或多个其他模板的一种方式。

由于在各种服务、应用程序等的各个模板上分离实体是有意义的，因此您最终可能会得到相当多的模板，所有这些模板都可能需要链接到相当多的主机。为了简化图片，可以将一些模板链接到一个模板中。

嵌套的好处是您只需将一个模板（“嵌套”，父模板）链接到主机，主机将自动继承链接模板（“嵌套”，子模板）的所有实体。例如，如果我们将模板 T1 和 T2 链接到模板 T3，我们就用来自 T1 和 T2 的实体补充 T3，而不是相反。如果我们将模板 A 链接到模板 B 和 C，我们用 A 中的实体补充 B 和 C。

配置嵌套模板

要链接模板，您需要使用现有模板或新模板，然后：

- 打开模板配置表单
- 找到模板字段
- 点击选择打开模板弹出窗口
- 在弹出窗口中，选择所需的模板，然后单击选择将模板添加到列表中
- 点击模板配置表单中的添加或更新

因此，父模板的所有实体以及链接模板的所有实体（例如监控项、触发器、图形等）现在都将出现在模板配置中，但链接模板仪表板除外，它仍然会被主机继承。

要取消链接任何链接的模板，请在同一表格中使用 取消链接或 取消链接并清理按钮，然后单击 更新。

选择 取消链接选项只会删除与链接模板的关联，而不会删除其所有实体（监控项、触发器、图表等）。

选择 取消链接并清理选项将删除与链接模板及其所有实体（监控项、触发器、图形等）的关联。

4 批量更新

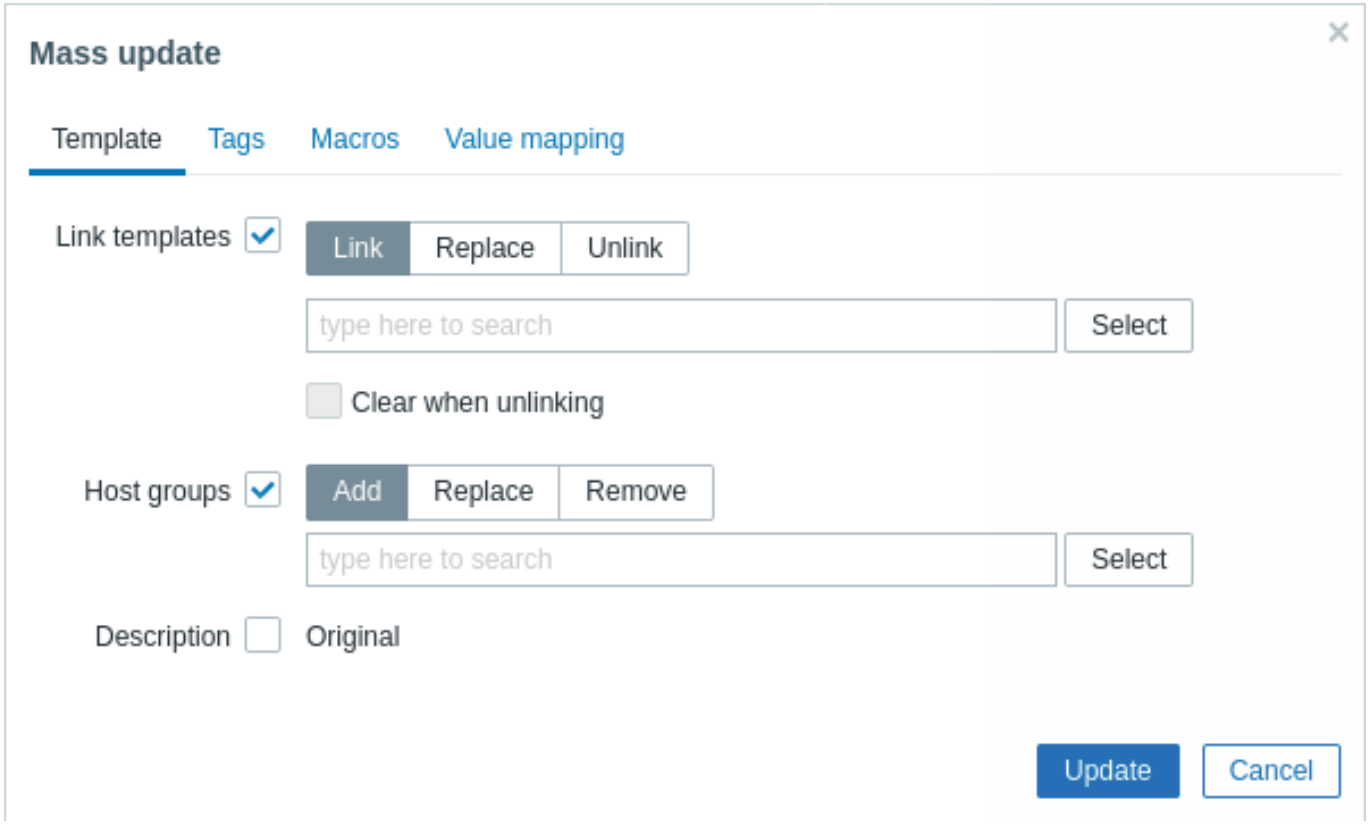
概览

有时您可能想要同时更改多个模板的某些属性。您可以使用批量更新功能，无需打开每个模板进行独立编辑。

使用批量更新

批量更新一些模板，操作如下:

- 在模板列表 中，标记要更新模板前的复选框
- 点击下方 更新按钮
- 页面跳转到带有所需属性 (模板, 标记, 宏或 值映射) 的选项卡
- 标记要更新的任何属性的复选框，并为它们输入一个新值



当选择更新 模板链接复选框时，将有以下选项可以选择:

- 链接 - 指定要链接的附加模板。
- 替换 - 指定要链接的模板，同时取消链接之前链接到模板的任何模板。
- 取消链接 - 指定要取消链接的模板。

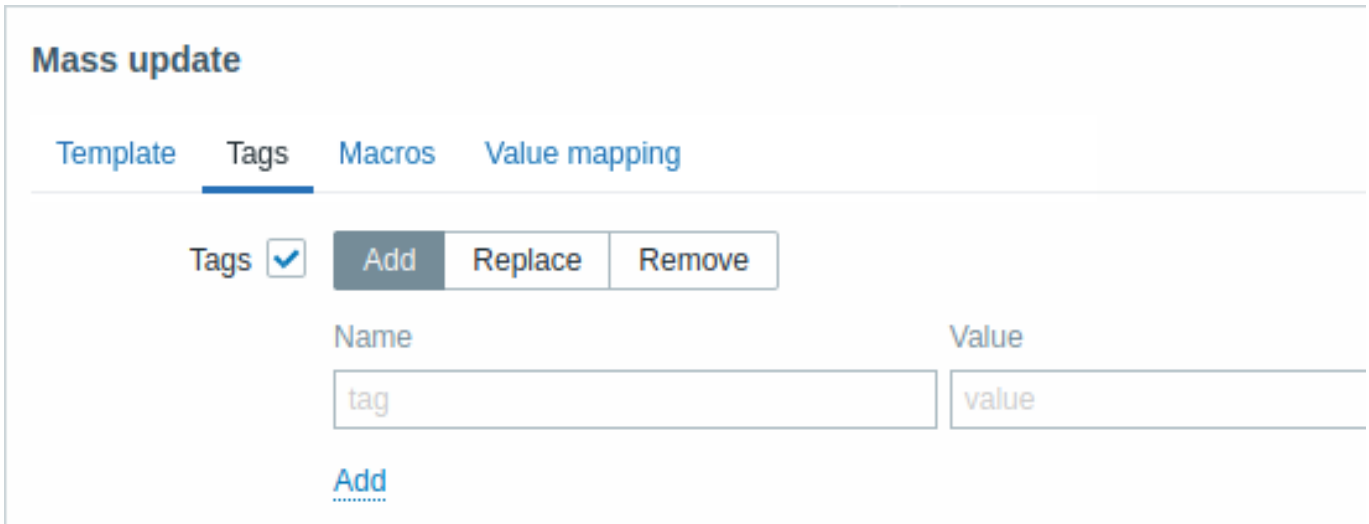
快捷搜索 -在搜索框输入模板名关键字后，将出现一个提供匹配模板的下拉菜单。只需向下滚动选择要 _ 链接/取消链接 _ 的模板即可。

当无链接时清除选项不仅允许取消任何先前关联的模板的关联，而且还可以删除从它们继承的所有元素 (监控项、触发器等)。

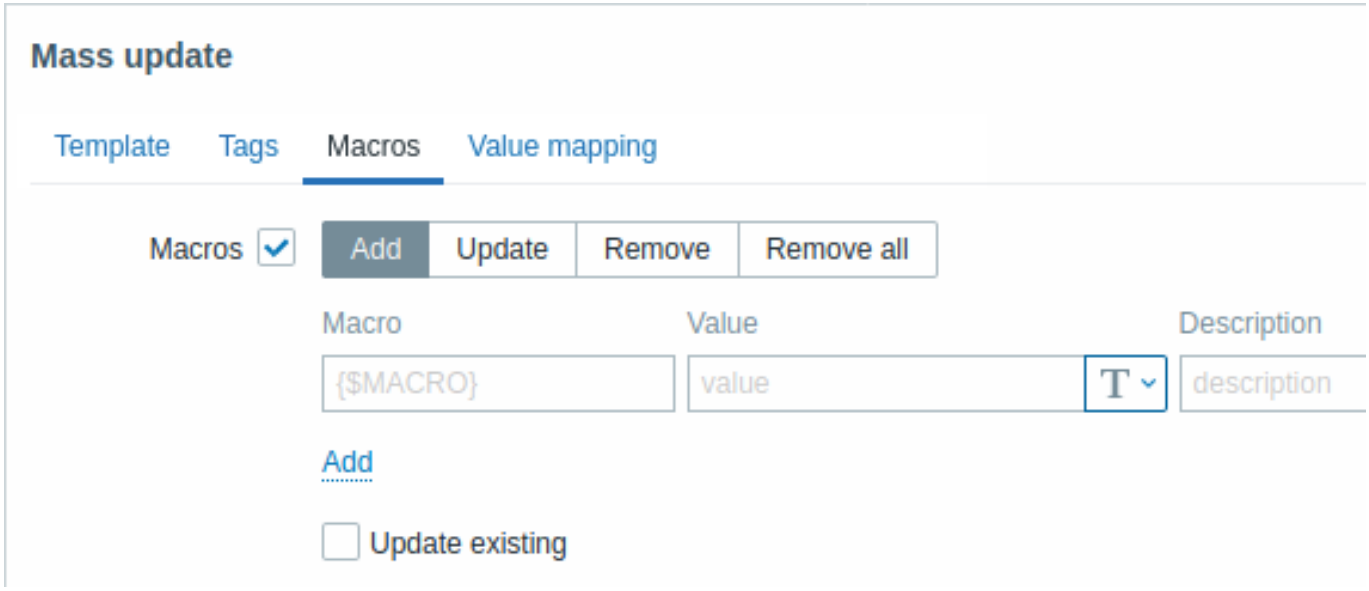
当选择更新 主机群组复选框时，将有以下选项可以选择:

- 添加 - 允许从现有的主机组中指定额外的主机组或为模板输入全新的主机组。
- 替换 - 将从任何现有主机组中删除模板，并将它们替换为在此字段中指定的模板 (现有或新的主机组)。
- 移除 - 将从模板中删除特定的主机组。

这些字段是自动完成的 - 在搜索框输入关键字后，会自动提供一个匹配主机组的下拉列表。如果主机组是新创建的，它也会出现在下拉列表中，并且在字符串后面用新 *(new)* 表示。数量较多时，只需向下滚动选择即可。

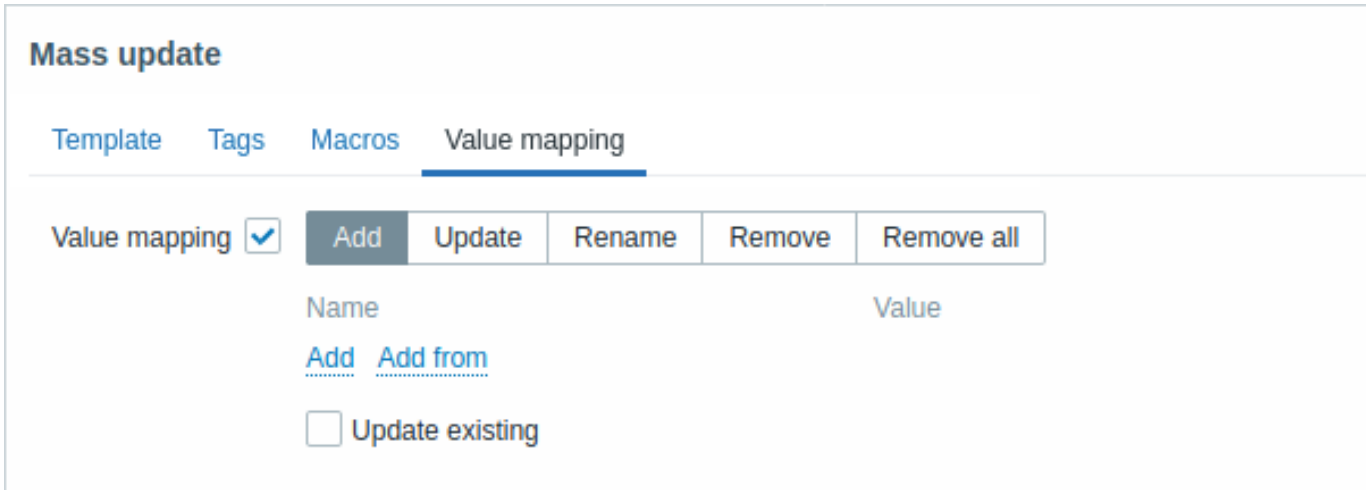


用户宏，{INVENTORY.*}，{HOST.HOST}, {HOST.NAME}, {HOST.CONN}, {HOST.DNS}, {HOST.IP}, {HOST.PORT} 和 {HOST.ID} 在标签中被支持。具有相同名称但不同值的标签不被认为是“重复”，可以添加到相同的模板中。



当选择更新 宏复选框时，将有以下选项可以选择:

- 添加- 允许为模板指定额外的用户宏。如果选中更新现有的复选框，则只更新指定宏名称的值、类型和描述。反之，如果模板中已经存在同名的宏，则不会更新该宏。
- 更新 - 将替换列表中指定的宏的值、类型和描述。如果选中添加缺失复选框，则模板上先前不存在的宏将被添加为新的宏。反之，则只有模板中已经存在的宏才会被更新。
- 移除 - 将从模板中删除指定的宏。如果选中除选中复选框，则除列表中指定的宏之外的所有宏都将被删除。如果未选中，则只删除列表中指定的宏。
- 全部移除 - 将从模板中删除所有的宏。如果未选中我确认移除所有宏复选框，将弹出需要选择该选项的警告提示框。



当选择更新 值映射复选框时，将有以下选项可以选择:

- 添加 - 添加值映射到模板。如果选中更新现有的复选框，只更新指定值映射中的所有属性。反之，如果值映射已存在，将不会更新。
- 更新 - 更新已存在值映射。如果选中添加缺失复选框，则模板上先前不存在的值映射将被添加为新的值映射。反之，则只有模板中已经存在的值映射才会被更新。
- 重命名 - 为已存在的值映射更换新的名称。
- 移除 - 将从模板中删除指定的值映射。如果选中除选中复选框，则除列表中指定的值映射之外的所有宏都将被删除。
- 全部移除 - 将从模板中删除所有的值映射。如果未选中我确认移除所有值映射复选框，将弹出需要选择该选项的警告提示框。

完成所有需要的更改后，点击更新按钮。所有选择模板的属性都会相应地更新。

1 主机和主机组

什么是 “主机”？

通常来讲，Zabbix 主机是指你想要监控的设备（例如服务器、工作站、交换机等）。

创建主机是使用 Zabbix 监控的首要任务之一。例如，如果你想监控一台服务器 “X” 上的某些指标，你必须首先创建这台名为 “服务器 X” 的主机，然后才能添加监控项。

主机组是由主机组成的。

前往创建和配置主机。

1 配置主机

概述

按照以下步骤可以在 Zabbix 前端页面创建主机：

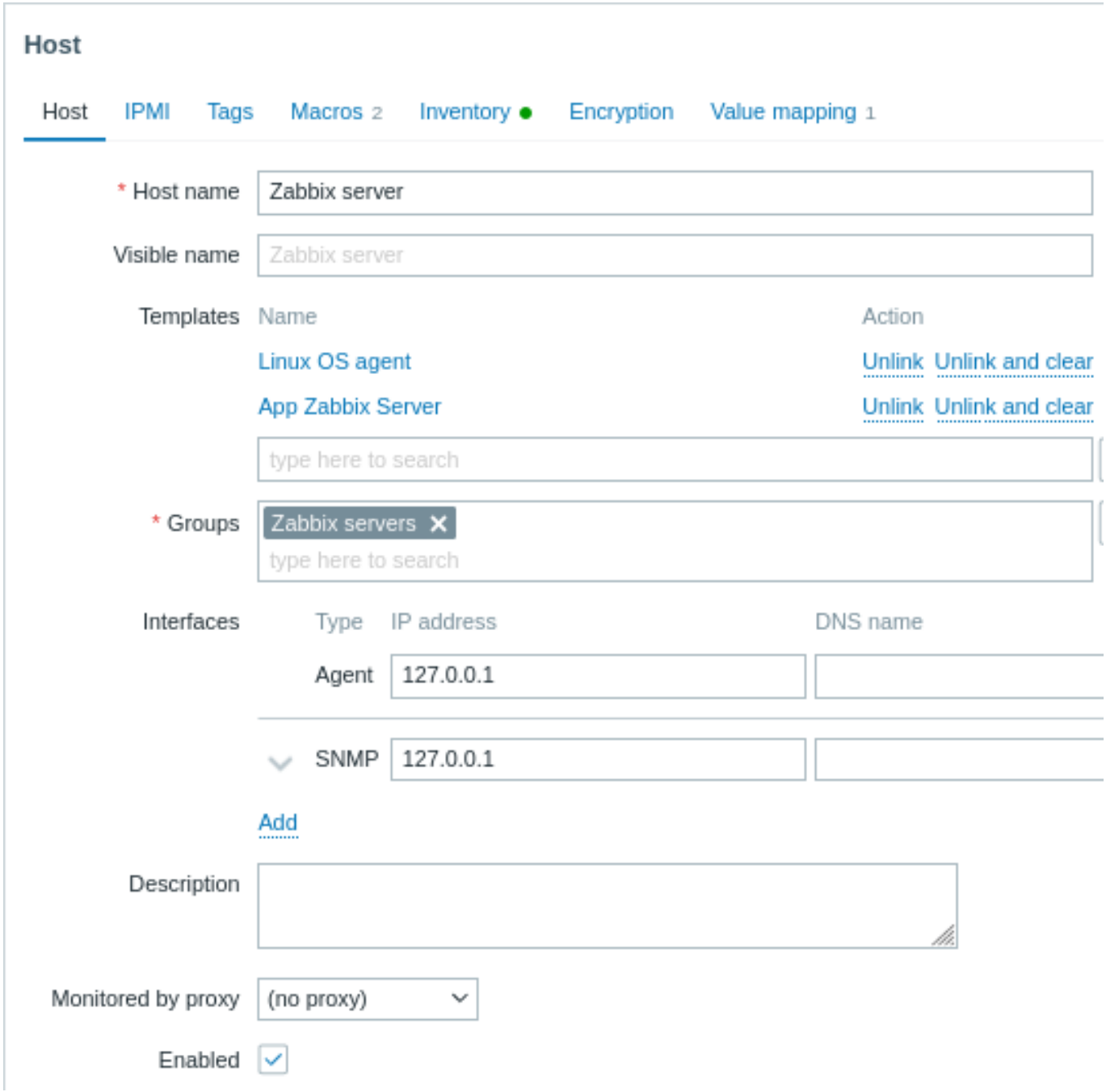
- 点击: *Configuration*（配置）→ *Hosts*（主机）或 *Monitoring*（监测）→ *Hosts*（主机）
- 在页面右侧点击 *Create host*（创建主机）（点击主机名编辑一个已有的主机）
- 在表单中输入主机的相关参数

你也可以在已经存在的主机上使用 *Clone*（克隆）和 *Full clone*（全克隆）按钮创建一台新的主机。点击 *Clone*（克隆）将保留所有的主机参数和模板链接（保留这些模版中的所有实体），*Full clone*（全克隆）将额外保留直接附加的实体（应用集、监控项、触发器、视图、自动发现规则和 Web 定制场景）。

注意: 当主机被克隆时，它将保留最初在模板上的所有模板实体。在现有主机级别上对这些实体所做的任何更改（例如更改的监控间隔、修改正则表达式或添加原型到底层发现规则）都不会复制到新主机；相反，它们将与最初模板上的保持一致。

配置

主机选项卡包含常规主机属性：



所有标有红色星号的为必填字段。

参数	描述
主机名	输入唯一的主机名。允许使用字母、数字、空格、点、破折号和下划线。开头和结尾不允许空格。 注意：配置正在运行的 Zabbix agent 主机时，agent 配置文件 的 *Hostname* 参数必须与此处输入的主机名相同。主动检查 需要依据该参数的值。
主机可见名	输入主机的唯一标识名称。如果你设置了这个名称，主机在列表、拓扑图等场景中将显示为此名称而不是主机名字段值。此属性支持 UTF-8 。

参数		描述
模板		将模板 链接到主机。所有实体（监控项、触发器、图表等）都将从模板继承。 要关联新模板，请开始在 链接新模板字段中输入模板名称进行模糊匹配。此时系统将会跳出匹配的模板列表；向下滚动以便选择。或者，您可以单击字段旁边的 选择，然后从弹出窗口的列表中选择模板。链接新模板字段中选择的模板将在保存或更新主机配置表单时链接到主机。 取消关联模板，请使用 链接的模板框中的两个选项之一： 取消链接 - 取消链接模板，但保留其监控项、触发器和图表 取消链接并清理 - 取消链接模板并删除其所有监控项、触发器和图表 列出的模板名可以点击跳转到模板配置表单。
主机组		选择主机所属的主机组。一台主机必须至少属于一个主机组。通过添加一个不存在的组名，可以创建一个新组并将主机关联到主机组。
主机接口		主机支持多种主机接口类型：*Agent*, *SNMP*, *JMX* 和 *IPMI*。 默认没有定义主机接口。要增加一个新接口，在 主机接口区域点击 添加并输入 *IP/DNS*, *Connect to* 和 *Port* 信息。 注意：用在任何监控项的接口都不能被移除，并且 移除链接是灰色的。 有关配置 SNMP 接口（v1、v2 和 v3）的更多详细信息，请参阅配置 SNMP 监控 。
	IP 地址	主机的 IP 地址（可选）。
	DNS 名称	主机的 DNS 名称（可选）。
	连接到	点击对应的按钮告诉 Zabbix server 采用哪种模式从 agent 端获取数据： **IP** - 连接到主机的 IP 地址（推荐） **DNS** - 连接到主机的 DNS 名称
	端口	TCP/UDP 端口。默认端口：Zabbix agent 是 10050，SNMP agent 是 161, JMX 是 12345，IPMI 是 623。
	默认	选择单选按钮设置默认接口。
描述		填写主机描述信息。
由 *agent* 代理程序监测		主机可以被 Zabbix server 或者 Zabbix proxy 监控： **(no proxy)** - 主机被 Zabbix server 监控 **Proxy name** - 主机被 Zabbix proxy “Proxy 名称” 监控
已启用		选中此项激活主机，准备接受监控。如果没选中，表示主机未激活，不能被监控。

IPMI 选项卡包含 IPMI 管理属性。

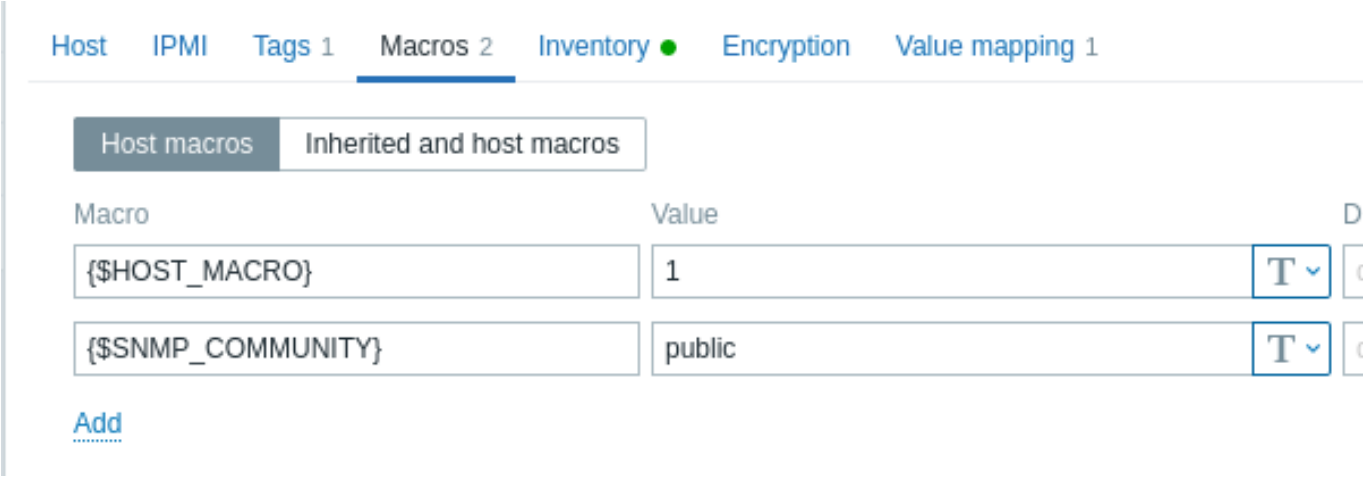
参数	描述
认证算法	选择认证算法。
权限级别	选择权限级别。
用户名	认证用户名。支持使用宏
密码	认证用户密码。支持使用宏

Tags（标签）选项卡允许定义主机级别的标签。该主机的所有问题都将使用此处输入的值进行标记。

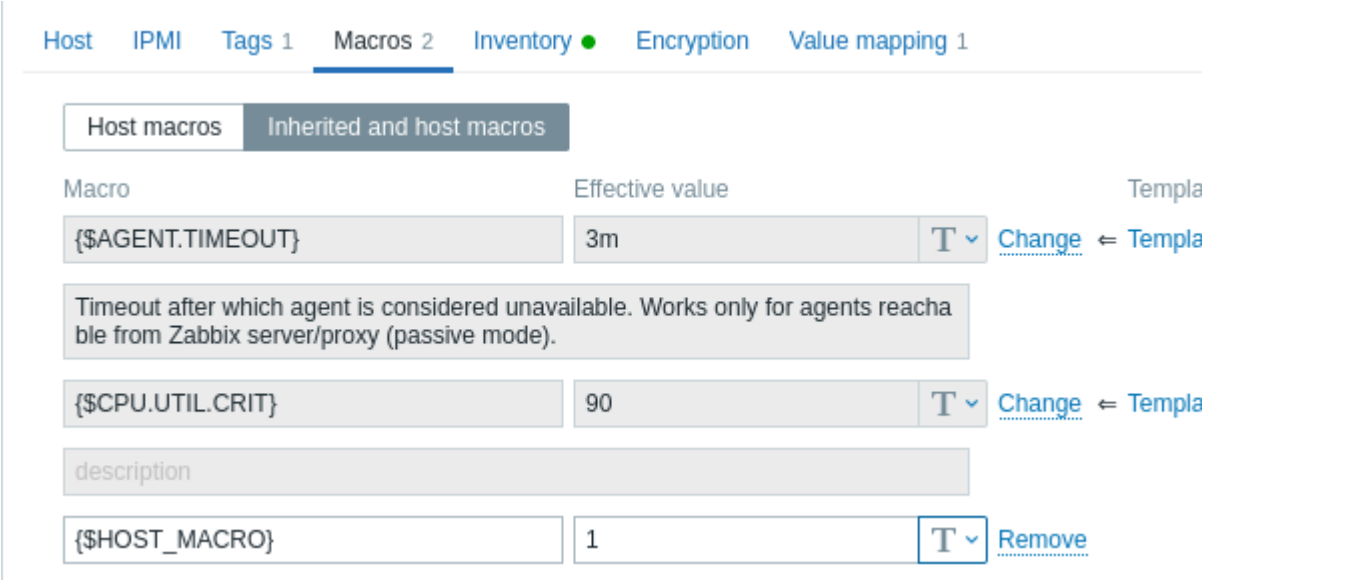


标签中支持以下宏：用户宏、{INVENTORY.*} 宏、{HOST.HOST}，{HOST.NAME}，{HOST.CONN}，{HOST.DNS}，{HOST.IP}，{HOST.PORT} 和 {HOST.ID}。

宏选项卡允许您将主机级别的用户宏 定义为键值对。请注意，宏值可以保存为纯文本、加密文本或 Vault 机密。也支持添加描述。

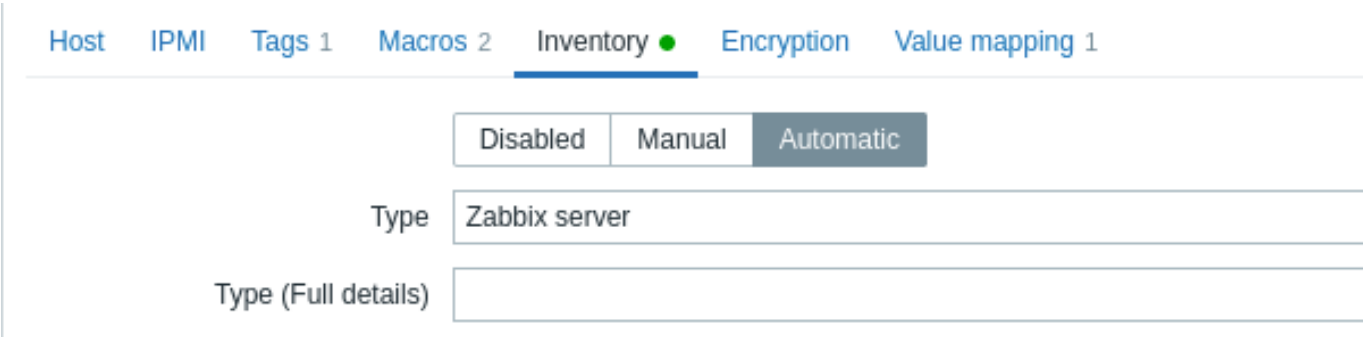


如果选择 继承以及主机宏选项，还可以在此处查看模板和全局级别的用户宏。所有为该主机定义的用户宏都显示为他们的来源以及对应的值。



为方便起见，页面提供了指向各个模板和全局宏配置的链接。还可以在主机级别编辑模板/全局宏，从而有效地在主机上创建宏的副本。

主机资产记录选项卡允许你手动为主机输入资产 信息。你还可以选择启用 自动资产信息填充，或者禁用此主机的资产信息填充。



如果启用了资产记录（手动或自动），则选项卡上方会显示绿点标示。

加密

加密选项卡允许建立与主机的加密 连接。

选项	描述
连接到主机	Zabbix server 或 proxy 如何与 Zabbix agent 连接：不加密（默认），使用 PSK（预共享密钥）或证书。
从主机连接	选择允许来自主机的连接类型（即来自 Zabbix agent 和 Zabbix sender）。可以同时选择多种连接类型（用于测试和切换到其他连接类型）。默认为 “不加密”。
发行者	许可的证书颁发者。证书首先通过 CA （证书颁发机构）进行验证。如果它是由 CA 签名的有效证书，则 *Issuer* 字段可用于进一步限制允许的 CA。如果 Zabbix 安装使用来自多个 CA 的证书，则使用此字段。如果此字段为空，则接受任何 CA。
主体	允许的证书主体。证书首先通过 CA （证书颁发机构）进行验证。如果它是由 CA 签名的有效证书，则 主体字段只能用于允许 主体字符串的一个值。如果此字段为空，则接受配置的 CA 签名的任何有效证书。
共享密钥身份	预共享密钥标识字符串。 不要将敏感信息放在此处，它会在网络上以未加密方式传输，以通知接收者使用哪个 PSK。
共享密钥（*PSK*）	预共享密钥（十六进制字符串）。最大长度：如果 Zabbix 使用 GnuTLS 或 OpenSSL 库，则为 512 个十六进制数字（256 字节 PSK），如果 Zabbix 使用 mbed TLS（PolarSSL）库，则为 64 个十六进制数字（32 字节 PSK）。示例： 1f87b595725ac58dd977beef14b97461a7c1045b9a1c963065002c5473194952。

值映射

Value mapping（值映射）选项卡允许在value mappings（值映射） 中配置监控项数据的人性化展示。

2 Configuring a host group

Overview

Host groups are used for the logical grouping of hosts and assigning user permissions to them.

Each host must have at least one host group assigned. A host may belong to multiple host groups, and each host group may contain multiple hosts.

Note that in Zabbix, all permissions are based on user groups and host/template groups. So, even if a single user needs access to a single host, it is granted by adding the user to a user group that has permission to access the host group containing that host.

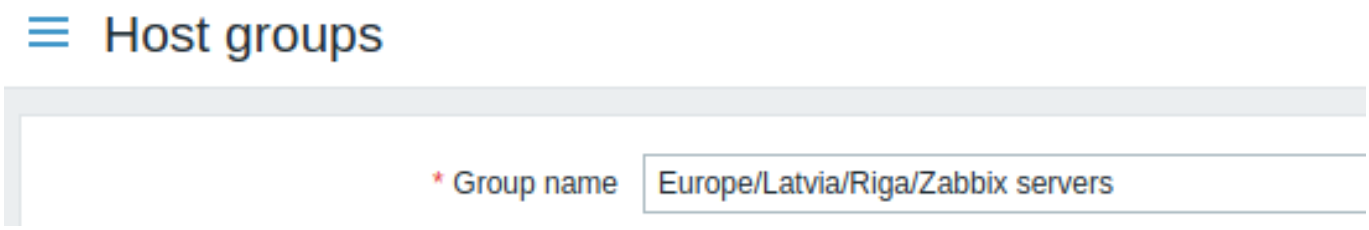
Configuration

> **Attention:**
> Only Super admin users can create host groups.

There are two options of creating a host group in Zabbix frontend.

Option one:

- Go to: *Configuration* → *Host groups*
- Click on *Create Group* in the upper right corner of the screen
- Enter the group name in the form



Option two: when configuring a host, enter a non-existing group name in the *Groups* input field.

Once the host group is created, you can click on the group name in the list under *Configuration* → *Host groups* to edit the group name, clone the group, or delete the group.

Deleting a host group only deletes the logical group, not the hosts in the group. It is not possible to delete a host group that is the only group for any existing host.

Creating host subgroups

A host subgroup (or nested host group) is a child of the parent host group that contains it. Nested representation of host groups is supported since Zabbix 3.2.0.

A subgroup is created by using the forward slash '/' in the group name input field to denote its relation to the parent group(s). For example:

- inputting `Europe/Latvia` creates the `Europe/Latvia` subgroup of the parent group `Europe`.
- inputting `Europe/Latvia/Riga/Zabbix servers` creates the respective subgroup within the nested parent groups `Europe`, `Europe/Latvia`, `Europe/Latvia/Riga`.

When creating a subgroup, using leading or trailing slashes, or several slashes in a row is not allowed. Escaping of '/' is not supported.

It is not required to create any parent host group(s) before creating a subgroup. You can choose whether to start by creating a subgroup (for example, `Europe/Latvia`) or any parent host group(s) (in our example, `Europe`). If you start by creating a subgroup, parent host group(s) will **not** be created automatically.

Permissions to host groups

- When creating a subgroup to an existing parent host group (for example, creating `Europe/Latvia` when `Europe` already exists), user group permissions to the subgroup are inherited from the parent.
- When creating a parent host group to an existing subgroup (for example, creating `Europe` when `Europe/Latvia` already exists), no permissions to the parent are set.

When editing any host group, you can also set an additional option, *Apply permissions and tag filters to all subgroups*. Marking this checkbox and clicking on *Update* will apply the same level of permissions and tag filters to all current and future subgroups of the host group being edited. This option is supported since Zabbix 3.4.0.

So, if any user groups have been given varying permissions to the subgroups of the host group being edited, marking the checkbox will grant all current and future subgroups the same user permissions and tag-based permissions as the group being edited.

Note that this option is not saved in the database and will override existing permissions. Any changes made through this option can be reverted only manually.

2 资产管理

概述

你可以将联网设备的资产信息保存在 Zabbix 里。

Zabbix 管理页面有一个特殊的 *Inventory*（资产记录）菜单。但你一开始不会看到任何数据，这里你也不能输入任何资产相关的信息。资产信息是在配置主机时人工录入建立的资产数据，或者通过使用某些自动填充选项完成的录入。

构建资产库

手动模式

当配置主机时，在 *Host inventory*（资产记录）选项卡中，你可以输入设备类型、序列号、位置、负责人等详细信息 - 这些数据将填充资产信息。

如果主机资产信息中包含 URL，并以'http' 或'https' 开头，则会在 *Inventory* 中呈现为可点击的链接。

自动模式

主机资产也可以自动填充。为了使自动填充功能生效，配置主机时 资产记录选项卡中的清单模式必须设置为 自动。

然后，在监控项配置中你可以通过配置主机监控项用任意指示目的字段相应属性的值（即 填充主机资产字段的监控项）来配置对应的主机资产字段。

以下是对资产信息自动采集非常有用的监控项:

- system.hw.chassis[full|type|vendor|model|serial] - 默认是 [full]，需要 root 权限
- system.hw.cpu[all|cpunum,full|maxfreq|vendor|model|curfreq] - 默认是 [all,full]
- system.hw.devices[pci|usb] - 默认是 [pci]
- system.hw.macaddr[interface,short|full] - 默认是 [all,full]，interface 支持正则表达式
- system.sw.arch
- system.sw.os[name|short|full] - 默认是 [name]
- system.sw.packages[package,manager,short|full] - 默认是 [all,all,full]，package 支持正则表达式

资产模式选择

可以在主机配置表单中选择资产模式。

默认情况下，新主机的资产模式是根据 *Administration*（管理）→ *General*（一般）→ *Other*（其他）中的 *Default host inventory mode*（默认主机资产记录模式）设置选择的。

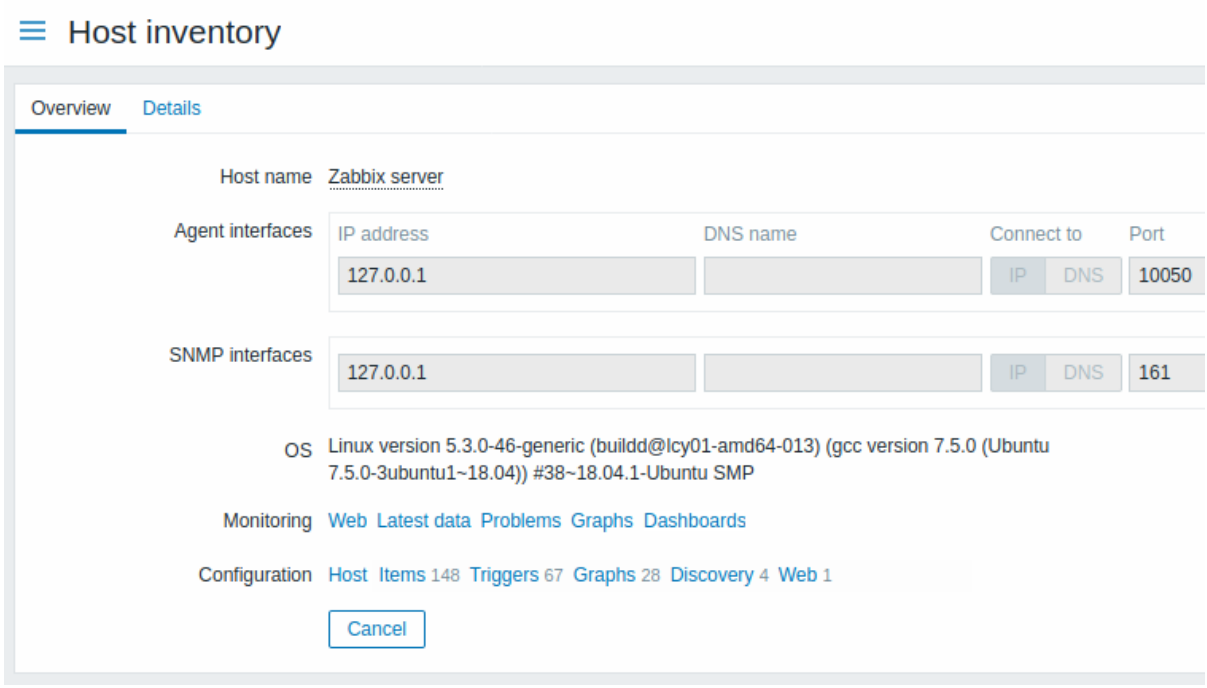
对于通过网络发现或自动注册操作添加的主机，可以定义 *Set host inventory mode*（设置主机资产记录模式）操作，选择手动或自动模式。此操作将覆盖 *Default host inventory mode*（默认主机资产记录模式）设置。

资产清单概述

资产记录菜单中提供了所有现有资产数据的详细信息。

在 资产记录 → 概览你可以通过资产的各个字段获取主机数。

在 资产记录 → 主机你可以看到所有具有资产信息的主机。单击主机名将以表单显示资产明细。



概览标签内容如下:

参数	描述
主机名	主机名。 单击该名称将打开一个菜单，其中包含为主机定义的脚本。 如果主机处于维护状态，则主机名将显示为橙色图标。
可见名称	主机的可见名称（如果已定义）。
主机（*Agent*、*SNMP*、*JMX*、*IPMI*） 接口	此块提供为主机配置的接口的详细信息。
操作系统	主机的操作系统清单字段（如果已定义）。
硬件	主机硬件清单字段（如果已定义）。
软件	主机软件清单字段（如果已定义）。
描述	主机描述。
监控	链接到包含此主机数据的监控部分：*Web*、最新数据、问题、图表、仪表板。
配置	此主机配置部分的链接：主机、应用程序、监控项、触发器、图形、发现、*Web*。 配置的实体数量是在每个链接后的括号中列出。

详细信息选项卡显示所有已填充（非空）的资产清单字段。

资产清单宏

有可用于通知的主机资产清单宏 {INVENTORY.*} ，例如：

”{INVENTORY.LOCATION1} 的服务器出现问题，负责人是 {INVENTORY.CONTACT1}，联系电话 {INVENTORY.POC.PRIMARY.PHONE.A1}。“

更多详细信息，请参阅supported macro（支持的宏）页面。

3 批量更新

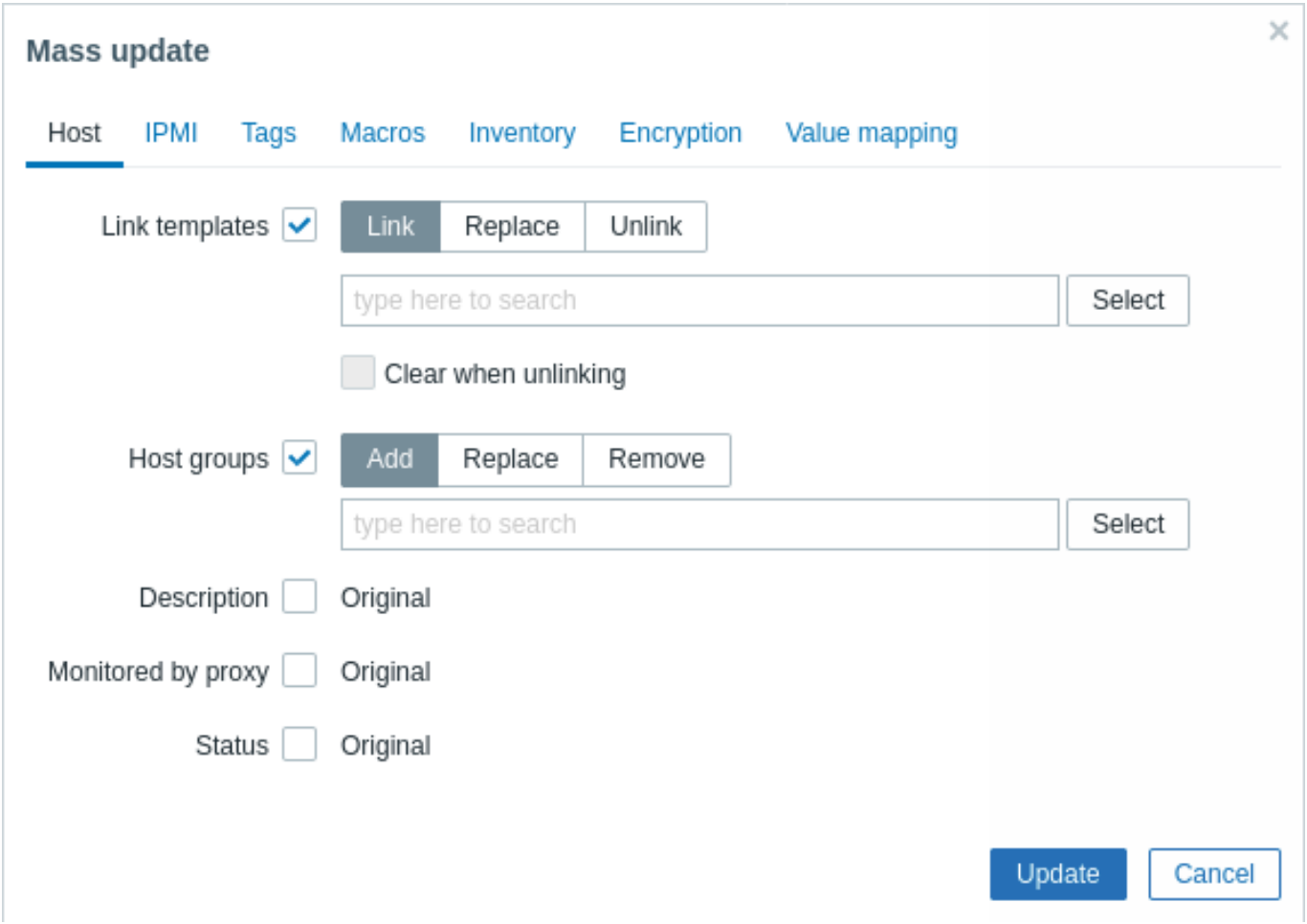
概述

有时你可能想要一次更改多个主机的某些属性。那么你可以使用批量更新功能来实现，而不是打开每个主机页面进行编辑。

使用批量更新

要批量更新某些主机，请按以下操作执行:

- 主机列表中，在要更新的主机之前选中复选框
- 点击下方的 *Mass update*（批量更新）按钮
- 跳转到属性所属的选项卡（*Host*（主机）、*IPMI*、*Tags*（标签）、*Macros*（宏）、*Inventory*（资产记录）、*Encryption*（加密）或 *Value mapping*（值映射））
- 选中要更新的任何属性的复选框，并输入新值



在选择 **template**（模板）链接更新按钮时，如下选项可供选择：

- *Link*（链接） - 指定需要链接的模板
- *Replace*（替换） - 指定需要链接的模板，并取消任意之前已经链接到该主机且不需要的模板链接
- *Unlink*（取消链接） - 指定需要取消链接的模板

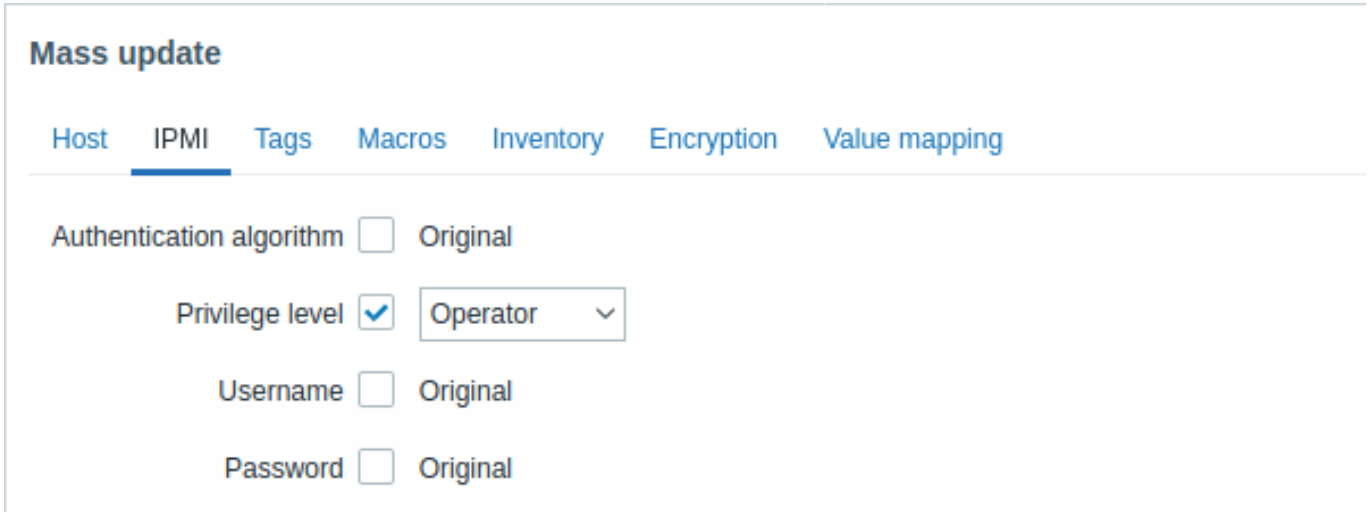
要指定链接/取消链接的模板，请在搜索框（在此输入搜索）字段中输入模板名，直到出现一个提供匹配模板的下拉菜单。只需向下滚动即可选择所需的模板。

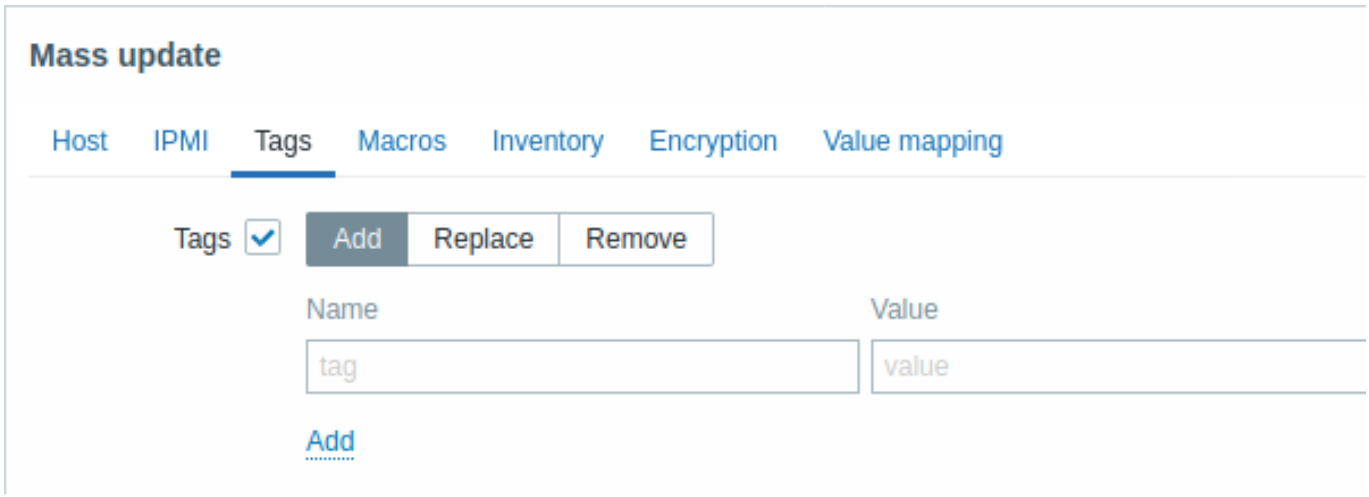
在选中 *Clear when unlinking*（取消链接时清除）选项时，在取消与任何以前链接的模板的关联的同时还会删除所有继承自该模板的元素（监控项、触发器等）。

在选择主机组更新对应的按钮时，如下选项可供选择：

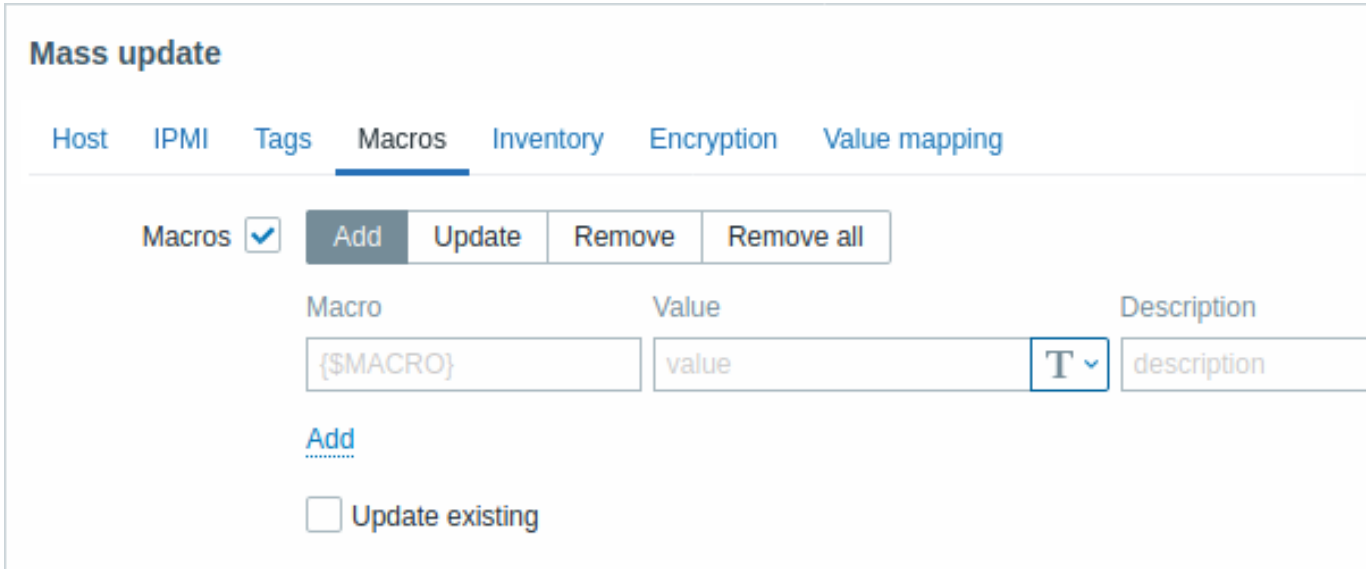
- *Add*（添加）- 从现有主机组指定要添加的主机组，或为所选主机输入全新的主机组
- *Replace*（替换）- 从任何现有主机组中删除所选主机，并替换为此字段中指定的主机组（现有的或新的主机组）
- *Remove*（移除）- 从所选主机中移除特定主机组

这些字段都是可以自动填充的 - 开始输入时会提供匹配的主机组的下拉列表。新的主机组也会出现在下拉列表中，并在字符串后用 *(new)* 表示。只需向下滚动即可选择。



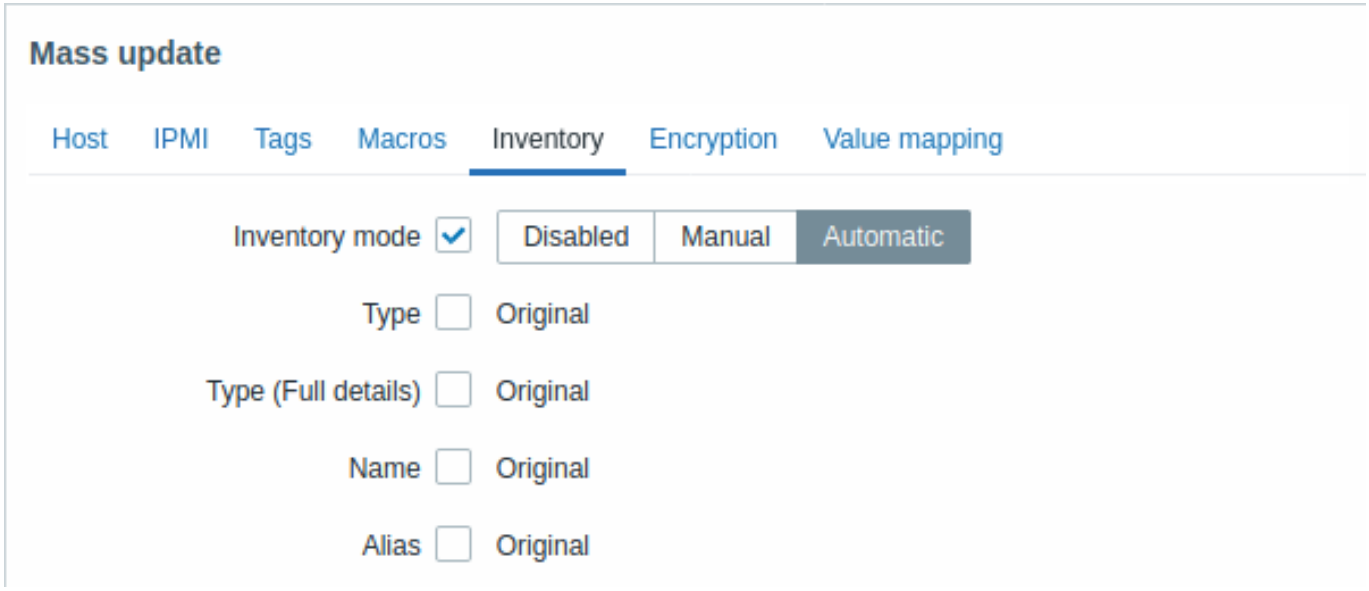


标签支持以下宏：用户宏、{INVENTORY.*} 宏、{HOST.HOST}、{HOST.NAME}、{HOST.CONN}、{HOST.DNS}、{HOST.IP}、{HOST.PORT} 和 {HOST.ID} 宏。注意：名称相同但值不同的标签是不被当成重复标签的，可以添加到同一个主机。

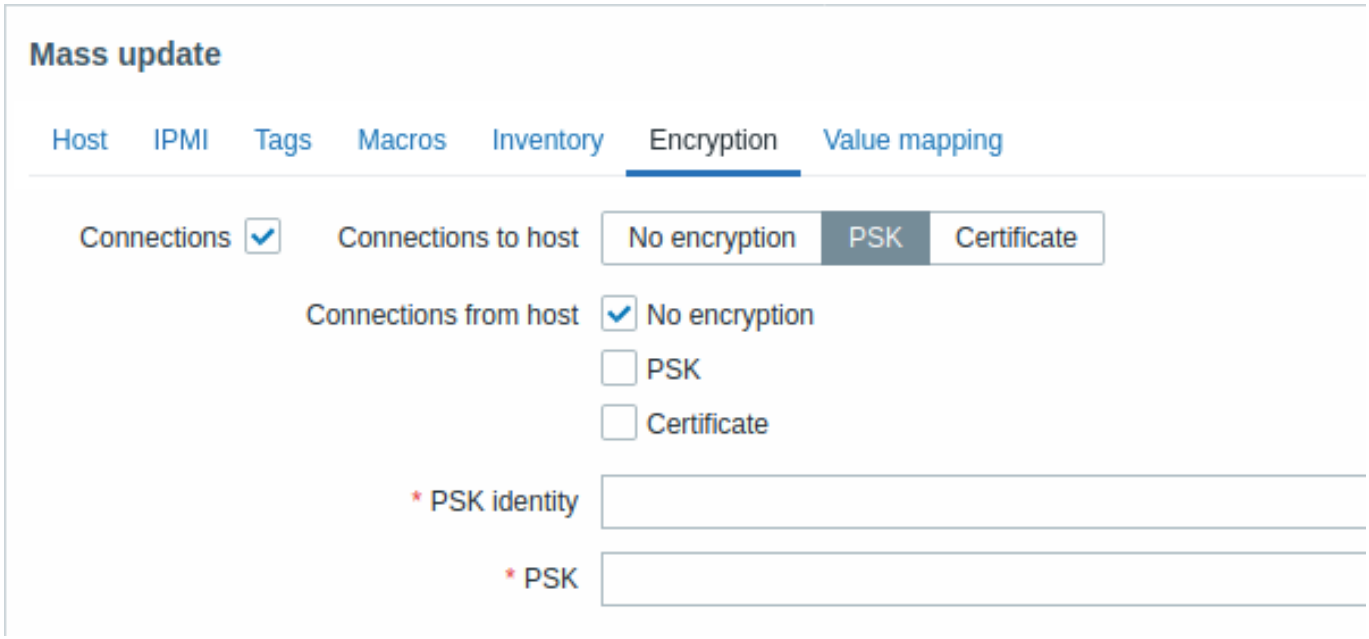


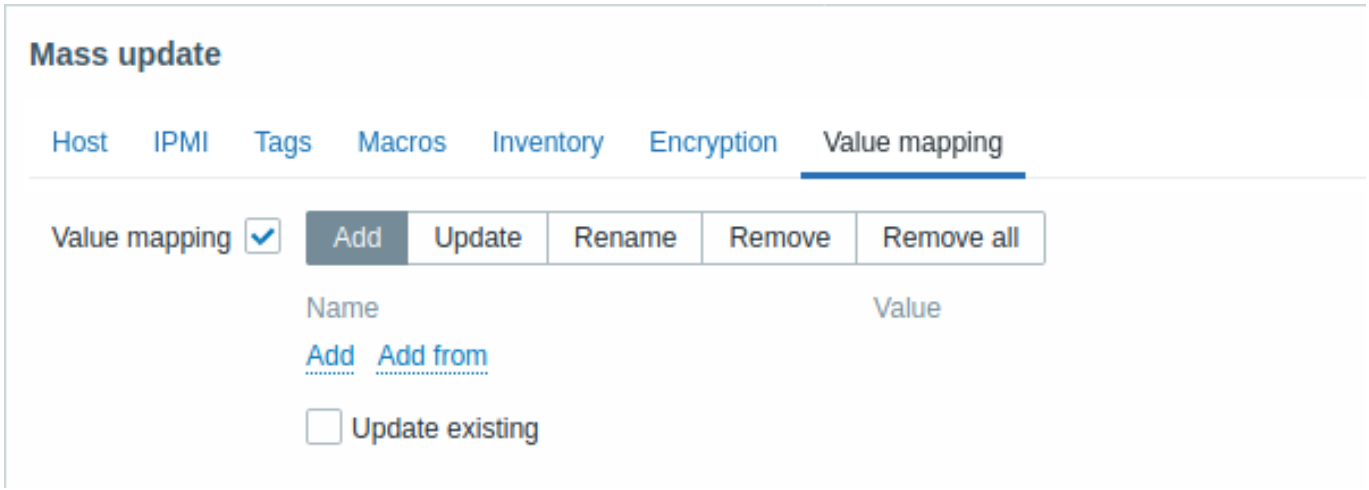
在选择宏更新按钮时，如下选项可供选择：

- *Add*（添加）- 允许为主机指定额外的用户宏。如果选中了 *Update existing*（更新现有的）复选框，则将更新指定宏名称的值、类型和描述。不选的话，如果主机上已存在具有该名称的宏，则不会对其进行更新。
- *Update*（更新）- 将替换此列表中指定的宏的值、类型和描述。如果选中 *Add missing*（添加所缺的）复选框，则主机上以前不存在的宏将被添加为新宏。如果不选，则仅更新主机上已存在的宏。
- *Remove*（移除）- 从主机中删除指定的宏。如果选中了 *Except selected*（除选定对象之外）复选框，则除列表中指定的宏之外的所有宏将被删除。如果不选，则仅删除列表中指定的宏。
- *Remove all*（全部删除）- 从主机上删除所有用户宏。如果没选中 *I confirm to remove all macros*（我确认删除所有的宏）复选框，将弹出一个要求确认删除所有宏的窗口。



为了能够批量更新资产记录，*Inventory mode*（资产记录模式）应该设置为’ 手动’ 或’ 自动’。





在选择值映射按钮时，如下选项可供选择：

- *Add*（增加）- 增加主机的值映射。如果选中了 *Update existing*（更新已有的），该名称的值映射下的所有属性都会被更新。否则，若该名称的值映射已存在，则不会被更新。
- *Update*（更新）- 更新已有的值映射。如果选中了 *Add missing*（添加所缺的），则原来主机中不存在的值映射会被加入到主机中。否则仅已经存在的值映射会被更新。
- *Rename*（重命名）- 重命名已存在的值映射。

- *Remove*（移除）- 从主机中删除指定的值映射。如果选中了 *Except selected*（除选定对象之外），则除了选定的值映射外都会被删除。
- *Remove all*（全部删除）- 删除主机的全部值映射配置。如果没选中 *I confirm to remove all value maps*（我确认删除所有的值映射）复选框，将弹出一个要求确认删除所有值映射的窗口。

完成所有必要的更改后，单击 *Update*（更新）。所有选定主机的属性将相应更新。

2 监控项

概述

监控项用来从主机收集数据。

配置主机后，你需要添加一些监控项以开始获取实际数据。

一个监控项是一个独立的指标。快速添加多个监控项的一种方法是将一个预定义的模板附加到主机。然而，为了优化系统性能，您可能需要对模板进行微调，使其只有必要的监控项被以必要的频率监控。

在单个监控项中，你可以指定从主机收集哪些数据。

为此，你需要使用监控项键值（key）。从而，具有键值名称为 **system.cpu.load** 的监控项将收集处理器负载的数据，而键值名为 **net.if.in** 的监控项将收集传入流量信息。

要在监控项键值中指定更多的参数，请在键值名称后的方括号中添加这些参数。例如，system.cpu.load**[avg5]** 将返回最近 5 分钟的处理器负载平均值，而 net.if.in**[eth0]** 将显示接口 eth0 的传入流量。

> **Note:**
> 所有支持的监控项类型和监控项键值的信息，请参阅监控项类型的各个部分。

继续创建和配置监控项。

1 创建监控项

概述

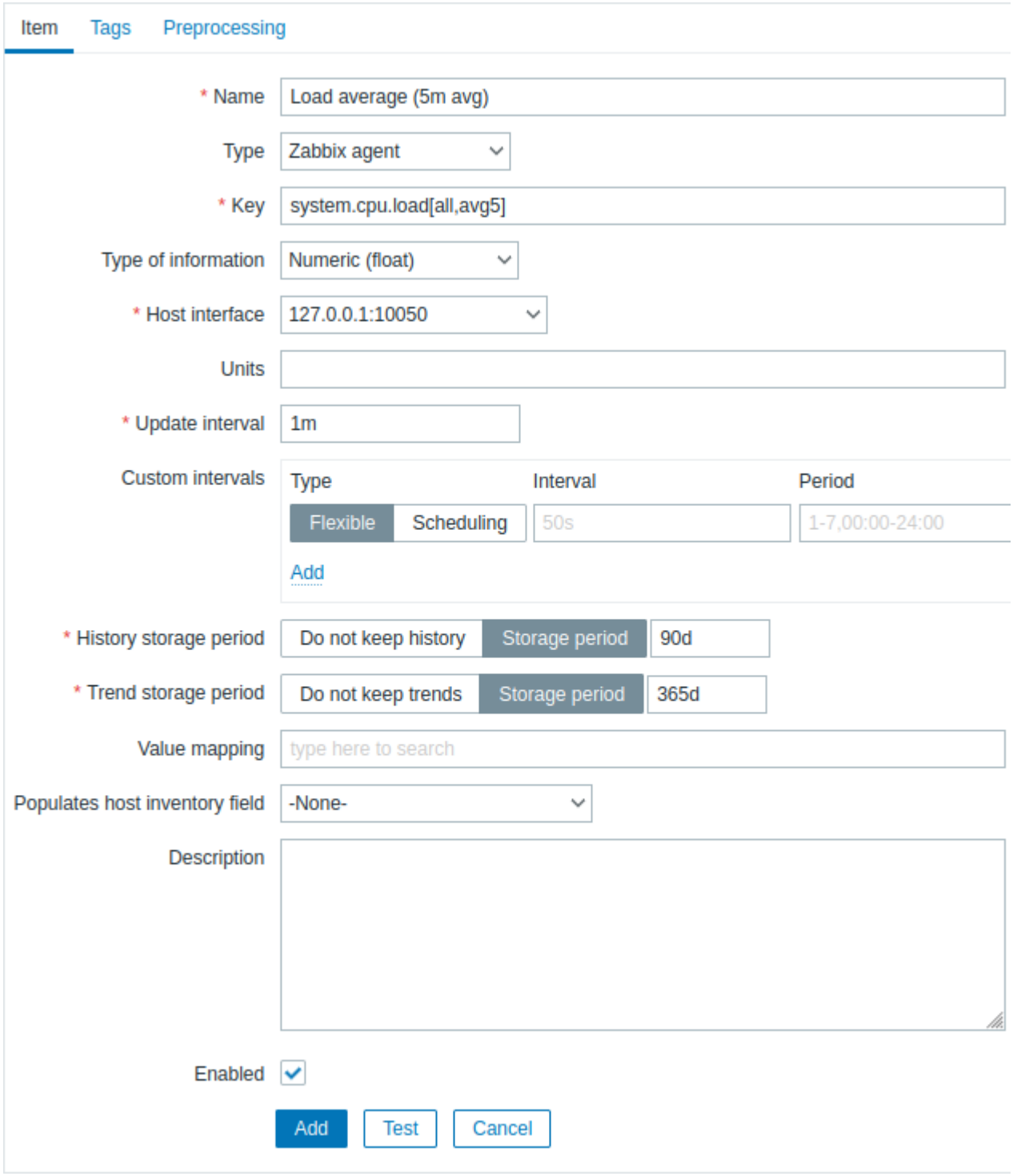
要在 Zabbix 管理页面创建一个监控项，请执行以下操作：

- 进入到: 配置 → 主机
- 在主机所在的行单击 监控项
- 点击屏幕右上角的创建监控项
- 在表单中输入监控项的参数

你也可以如此创建监控项：打开一个已有的监控项，按克隆按钮，然后以不同的名称保存。

配置

监控项选项卡包含了常规监控项属性。



所有必填字段都用红色星号标记。

参数	描述
名称	监控项名称。
类型	监控项类型。参考单独监控项类型 章节。
键值	监控项键值 (最多 2048 个字符)。 支持的监控项键值 能够在各个监控项类型中找到。 键值在单个主机中必须是唯一的。 如果键值类型是'Zabbix agent', 'Zabbix agent (主动)' or ' 简单检查', 则这个键值必须被 Zabbix agent 或 Zabbix server 支持。 也可以查看: 标准的键值格式.

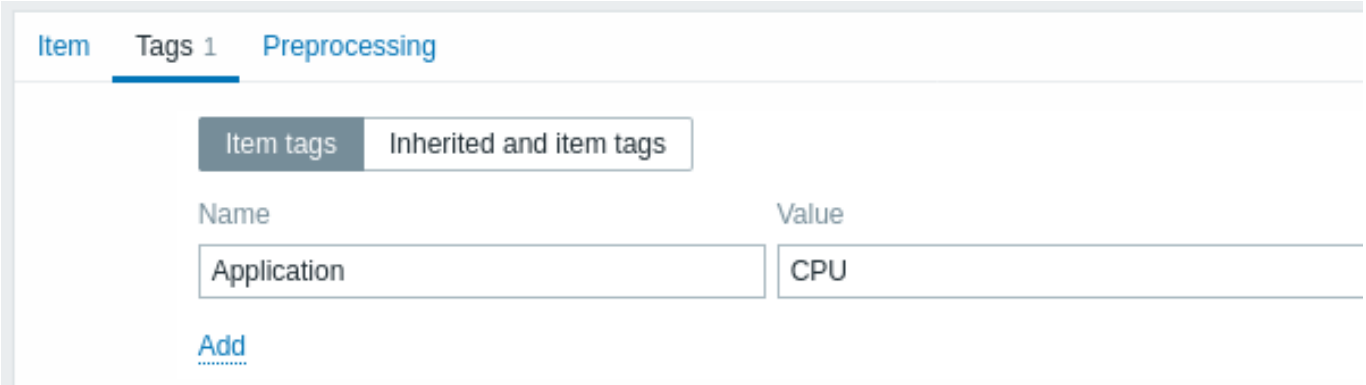
参数	描述
信息类型	执行转换后存储在数据库中的数据类型（如果有） **Numeric (unsigned)** - 64 位无符号整型 **Numeric (float)** - 64 位浮点数 允许大约 15 位的精度，范围从-1.79E+308 到 1.79E+308 (PostgreSQL 11 和更早的版本除外) 也支持用科学计数法接收值。例如:1.23E+7，1e308，1.1E-4 **Character** - 短文本数据 **Log** - 具有可选日志相关属性 (timestamp（时间戳），source（源），severity（严重性），logeventid（日志事件 id）) 的长文本数据 **Text** - 长文本数据。可参见文本数据限制 针对仅返回一种指定格式数据的监控项，匹配的信息类型可以自动选择。
主机接口	选择主机接口，此字段在编辑主机级别的监控项时可用
单位	如果设置了单位，Zabbix 在接收到数据后会进行处理，使其匹配设置的单位 默认情况下，如果原始值超过 1000，则除以 1000 再显示。例如，如果将单位设置为 _bps_ 并接收到值 881764，它将显示为 881.76 Kbps。 JEDEC 存储器标准, 用于处理 B(字节)，Bps(字节/秒) 单位，它除以 1024。因此，如果单位设置为 B 或 Bps, Zabbix 将显示: 1 为 1B/1Bps 1024 为 1KB/1KBps 1536 为 1.5KB/1.5KBps 如果使用以下与时间相关的单位，则使用特殊处理： **unixtime** - 转换成”yyyy.mm.dd hh:mm:ss”. 想要正确转换，接收的值必须是 *Numeric (unsigned)* 类型的信息。 **uptime** - 转换成”hh:mm:ss” 或”N days, hh:mm:ss” 例如，如果你收到的值为 881764（秒），则显示为 “10 天，04:56:04” **s** - 转换成”yyy mmm ddd hhh mmm sss ms”; 参数被视为秒数。 例如，如果您收到的值为 881764（秒），则显示为 “10d 4h 56m” 只显示 3 个主要单位，如 “1m 15d 5h” 或 “2h 4m 46s”。如果没有显示天数，则仅显示两个级别 - “1m 5h”（不显示分钟，秒或毫秒)。如果该值小于 0.001，将被转换成 “<1 ms”。 注意如果单位带有前缀 “!”，单元前缀/处理不会应用到监控项的值。请参阅单位转换.
更新间隔	每 N 秒获取一次这个监控项的新值。允许的最大更新间隔为 86400 秒（1 天） 支持时间后缀，例如 30s, 1m, 2h, 1d. 支持用户宏 。 单个宏必须填充整个字段。不支持字段中的多个宏或文本混合的宏。 注意: 仅当自定义间隔存在非零值时，更新间隔才能设置为 “ 0”。如果设置为 “ 0”，并且存在自定义间隔（灵活的或计划的）且具有非零值，则将在自定义间隔持续时间内轮询该监控项。 注意监控项成为活动后或更新间隔更改后的第一个监控项轮询可能发生在配置值之前。 可以通过点击 立即执行按钮查询现有被动监控项的值。
自定义时间间隔	你可以为检查监控项创建自定义的规则： 灵活 - 为更新间隔 (不同频率的间隔) 创建一个特例 调度 - 创建自定义轮询时间表。 详细信息请查看自定义时间间隔。 间隔字段支持时间后缀 例如，30s, 1m, 2h, 1d。 支持用户宏 。 单个宏必须填充整个字段。不支持字段中的多个宏或文本混合的宏。 从 Zabix 3.0.0 开始支持调度。 注意: 不适用于主动类型的 Zabbix agent 监控项。

参数	描述
历史数据存储周期	以下可供选择： 不存储历史数据 - 不存储监控项历史数据。如果只有依赖项需要保留历史记录，则对主监控项很有用。 此设置不能被全局管家设置. 存储周期 - 指定将详细历史数据保留在数据库中的时长 (1 小时至 25 年)。过期数据将被 housekeeper 管家删除。按秒存储。 支持时间后缀 例如，2h, 1d. 支持用户宏 存储周期的值可以被 *Administration*（管理）→ *General*（通用）→ 管家覆盖。 如果存在全局设置，则显示绿色告警信息  。当鼠标移到上面时会提示告警信息，例如：被全局管家设置 *(1d)* 覆盖。 建议保留最小可能天数的历史数据，以减轻数据库存储压力。可以保留更长的趋势数据来替代历史数据。 更多内容请参考历史与趋势.
趋势存储时间	以下可供选择： 不存储趋势数据 - 将不会存储趋势数据。 此设置不能被全局管家设置覆盖。 存储周期 - 指定在数据库中保存聚合（每小时，最大，平均，计数）历史数据的时长（1 天至 25 年）。管家将删除较旧的数据。按秒存储。 支持时间后缀例如 2h，1d。支持用户宏。 存储周期可以被 *Administration*（管理）→ *General*（通用）→ 管家中的设置覆盖。 如果存在全局设置，将显示一条绿色的警告消息  ，当鼠标移到上面时会提示告警信息，例如：被全局管家设置 *(7d)* 覆盖。 注意*:* 保持趋势不适用于非数字数据 - 字符、日志和文本。 更多信息请参考历史与趋势.
值映射	将值映射应用于此监控项。值映射 不会改变收到的值，仅用于显示数据。 适用于 数值型 *(*无符号*)*, 数值型 *(*浮点型*)* 和 字符型监控项。 例如, ”Windows service states”.
日志时间格式	仅适用于日志类型的监控项。支持占位符: * **y**: 年 *(1970-2038)* * **M**: 月 *(01-12)* * **d**: 日 *(01-31)* * **h**: 小时 *(00-23)* * **m**: 分钟 *(00-59)* * **s**: 秒 *(00-59)* 如果留空，则不会解析时间戳。 例如，从 Zabbix Agent 日志文件中考虑以下几行: ” 23480:20100328:154718.045 Zabbix agent started. Zabbix 1.8.2 (revision 11211).” 它以 PID 的六个字符位置开始，后跟日期，时间和行的其余部分。 该行的日志时间格式为”ppppppp:yyyyMMdd:hhmmss”. 注意，”p” 和”:” 字符只是占位符，只能是”yMdhms”.
填充主机资产记录字段	你可以选择监控项的值填充的主机资产字段，如果你为主机启用了自动发现模式资产管理 。 该字段在信息类型设置为’ 日志’ 时不可用。
描述	输入监控项描述。
启用	选中复选框以启用该监控项，以便对其进行处理。
最新数据	点击链接查看监控项的最新数据。 链接仅在编辑一个已存在的监控项时可用。

Note:
监控项类型的特定字段在对应页面会有描述。

Note:
当编辑主机级别上的现有模板 级别的监控项时，多个字段是只读的。你可以使用表单标题中的链接并转到模板级别并在其中进行编辑，但请记住，模板级别上的更改将更改模板链接到的所有主机的监控项。

标签选项卡允许定义监控项级别的标签。



监控项值预处理

预处理选项卡允许为接收到的值定义转换规则 。

测试

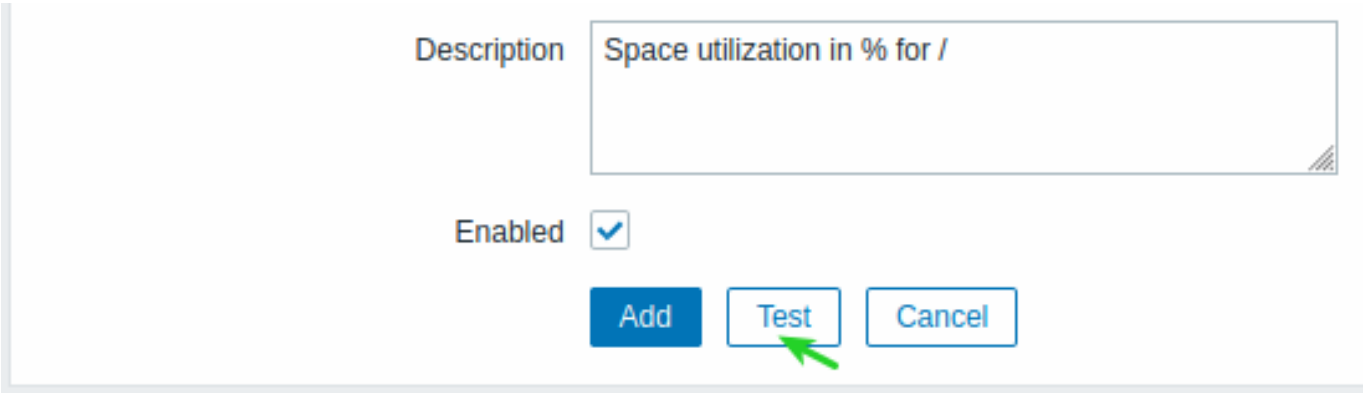
可以对监控项进行测试，如果配置正确，则可以返回实际值。测试甚至可以在保存监控项之前进行。

可以对主机和模板的监控项、监控项原型和低级别自动发现规则进行测试。agent（主动式）类型的监控项不能被测试。

监控项测试可用于以下被动式监控项：

- Zabbix agent
- SNMP agent (v1, v2, v3)
- IPMI agent
- SSH checks
- Telnet checks
- JMX agent
- Simple checks（除了监控项 `icmpping*`, `vmware.*`）
- Zabbix internal
- Calculated items
- External checks
- Database monitor
- HTTP agent
- Script

要测试监控项，请单击监控项配置表单底部的测试按钮。请注意，对于无法测试的监控项（例如简单检查之外的其它主动检查），测试按钮将被禁用。

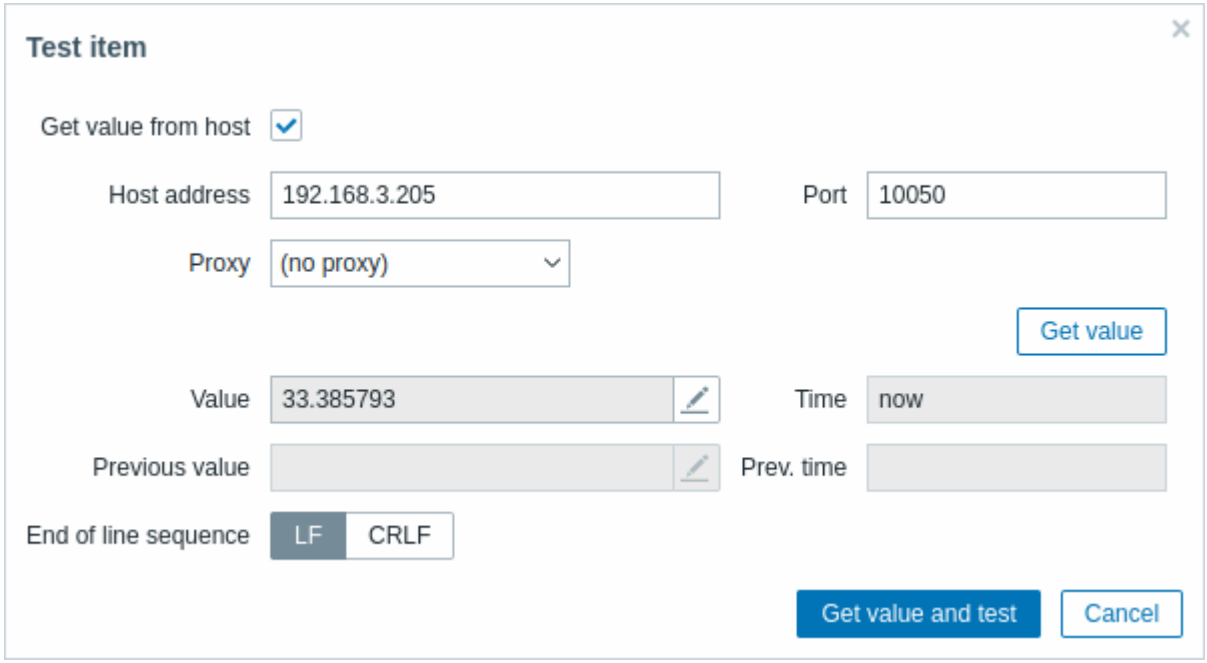


监控项测试表单包含主机参数（主机 IP 地址、端口、proxy 名称/无 proxy）和监控项相关的详情（如 SNMPv2 的团体名或 SNMPv3 的安全凭据）所需的字段。这些字段是上下文相关的：

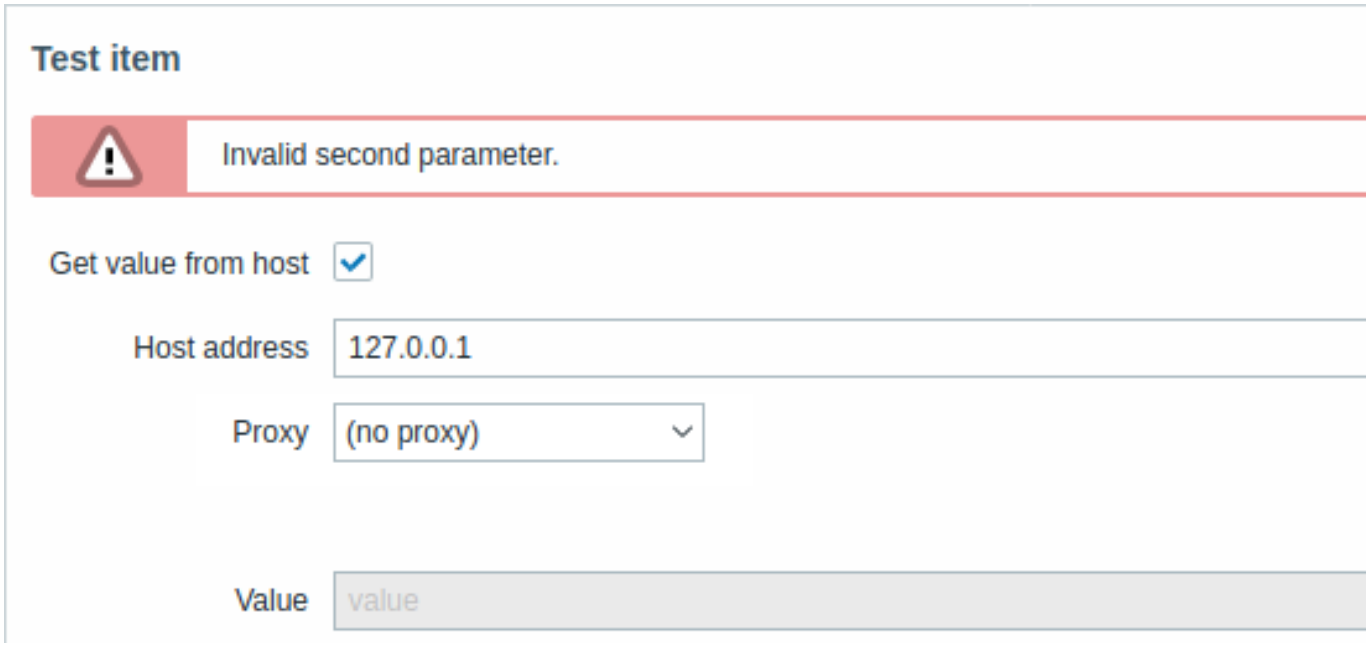
- 所需的参数值会尽可能预填充，例如，对于需要 agent 的监控项，所需信息就会从主机下所选的 agent 接口传递过来
- 模板内的监控项的相关参数需要手动填充
- 纯文本宏的值会被解析
- 值（或部分值）为密码或 Vault 宏的字段为空，必须手动输入。如果监控项参数中存在加密宏值，则会显示如下警告信息：“监控项包含用户定义的带有密码的宏。这些宏的值应该手动输入。”
- 在特定的监控项类型上下文中，不需要的字段会被禁用（例如，在可计算类型的监控项中主机地址字段和 proxy 字段被禁用）

要测试监控项，点击 获取值。如果成功获取到值，它会自动填充进对应的 *Value*（值）字段，将当前值（如果有的话）移动到 *Previous value*（前一次值）字段，同时计算 *Prev. time*（前一次）字段，即两个值（两次测试）之间的时间差，并尝试检测 EOL 序列，若在接收

到的值中检测到 “\n\r” 时切换到 CRLF。



如果配置不正确，则返回错误提示，并描述可能的原因。



从主机接收成功的值也可以用于测试预处理步骤。

表单按钮

表单底部的按钮允许执行多种操作。

Add	添加监控项。此按钮仅适用于新监控项。
Update	更新监控项的属性。
Clone	根据当前监控项的属性创建另一个监控项。
Execute now	立即执行新监控项值的检查。仅支持 被动检查 (参见更多详细信息)。 注意：当立即检查值时，配置缓存不会更新，因此该值不会反映监控项配置在最近极短时间内的最新更改。

Test	通过获取一个值来验证监控项配置是否正确。
Delete and clear	删除监控项历史记录和趋势数据。
Delete	删除监控项。
Cancel	取消对监控项属性的编辑。

文本数据限制

文本数据限制取决于后端数据库。在将文本值存储到数据库之前，它们会被截断以符合数据库值类型限制：

数据库	信息类型		
	Character(字符)	**Log**（日志）	**Text**（文本）
MySQL	255 字符	65536 字节	65536 字节
PostgreSQL	255 字符	65536 字符	65536 字符
Oracle	255 字符	65536 字符	65536 字符

自定义脚本限制

可用的自定义脚本长度取决于使用的数据库：

数据库	字符限制	字节限制
MySQL	65535	65535
Oracle 数据库	2048	4000
PostgreSQL	65535	无限制
SQLite（仅 **Zabbix proxy**）	65535	无限制

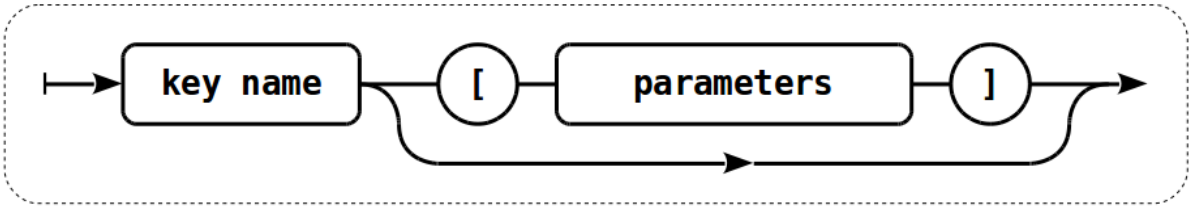
不支持的监控项

如果由于某种原因无法检索到值，则该监控项会变为不受支持的。但是该监控项仍会以设定的更新间隔重新检索。

不支持的监控项会被报告为 “不支持（NOT SUPPORTED）” 状态。

1 监控项键值的格式

监控项键值的格式（包括键值的参数）必须遵循语法规则。以下插图描述了支持的语法。可以通过跟随箭头来确定每个点上允许的元素和字符 - 如果可以通过线到达某个块，则允许，否则 - 不允许。



要构建有效的监控项键值，首先要指定键值的名称，然后选择是否具有参数，后面的两行描述了这一点。

键值名称

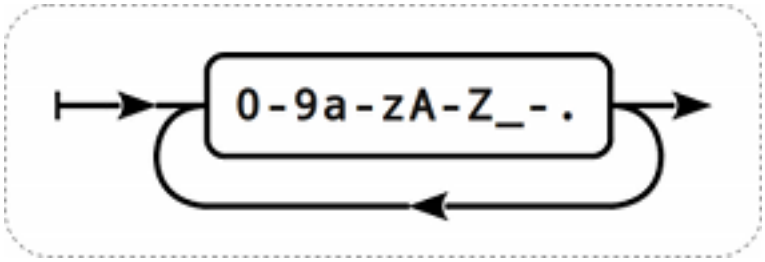
Key 名本身具有有限的允许字符范围，允许的字符是：

`0-9a-zA-Z_-.`

即:

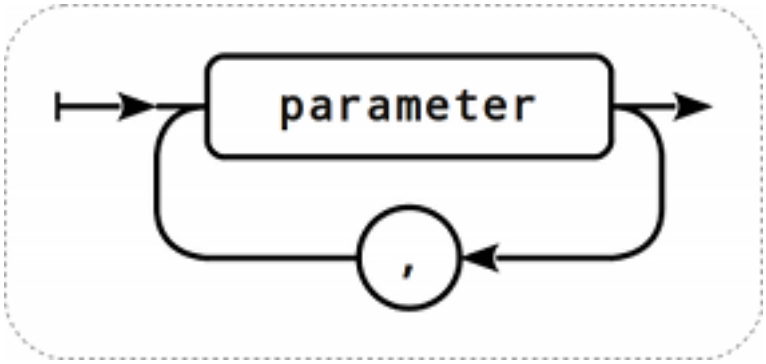
- 所有的数字;
- 所有的小写字母;
- 所有大写字母;
- 下划线;

- 减号;
- 点。

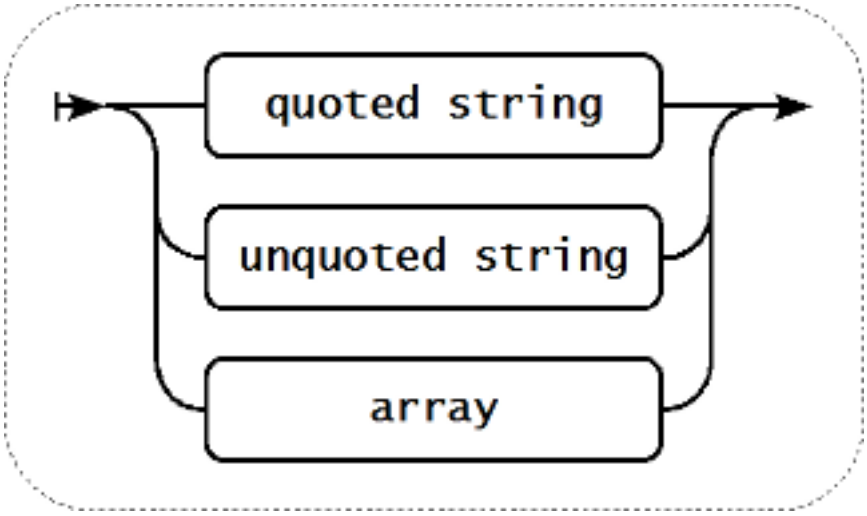


键值的参数

监控项的键值可以有多个逗号分隔的参数。



每个 key 参数可以是带引号、无引号的字符串或数组。



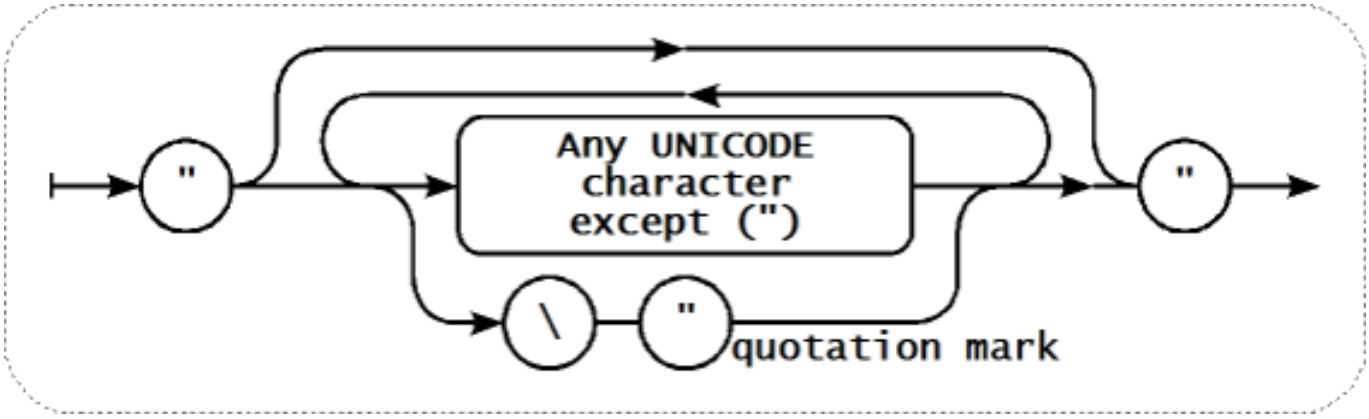
参数也可以为空，此时使用默认值。在这种情况下，如果指定了其它参数，则必须添加对应数量的逗号。例如，键值 **icmp-ping[„200„500]** 将指定每 ping 一次的时间间隔为 200 毫秒，超时时间为 500 毫秒，其它所有参数为默认值。

参数 - 带引号的字符串

如果键值参数为带引号的字符串，则允许任何 Unicode 字符。

如果键值参数的字符串中包含逗号，则该参数必须用引号引起来。

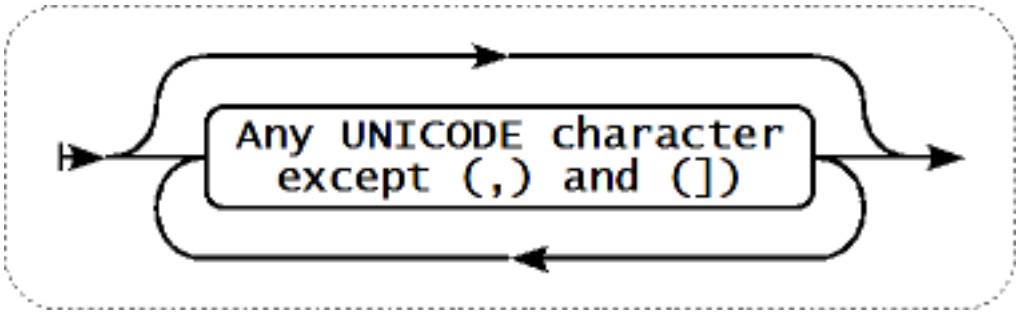
如果键值参数的字符串包含引号，则该参数必须用引号括起来，并且作为参数字符串一部分的每个引号都必须用反斜杠 () 进行转义。



Warning:
要引用监控项键值参数，只能使用双引号，不支持单引号。

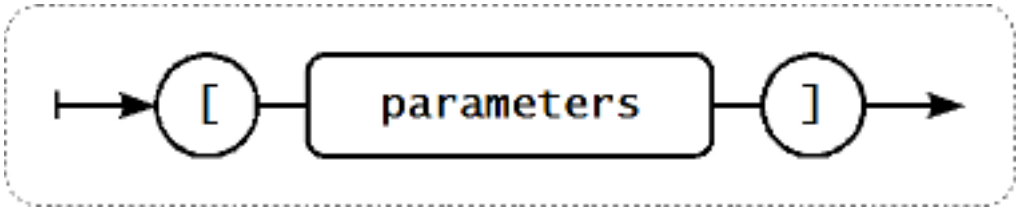
参数 - 不带引号的字符串

如果键值的参数是一个不带引号的字符串，可使用除逗号和右方括号（]）之外的任何 Unicode 字符。不带引号的参数不能以左方括号（[）开头。



参数 - 数组

如果 key 参数是一个数组，它需要包含在方括号中，其中各个参数需要列为一行且符合多个参数的规则和语法。



Attention:
多级参数数组，例如 `[a,[b,[c,d]],e]`，是不受支持的。

2 自定义间隔

概述

有两种方法可以创建检查监控项的时间的自定义规则。灵活间隔，允许重新定义默认更新间隔；调度，可以在特定时间或时间序列执行监控项检查。

灵活间隔

灵活间隔允许重定义特定时间段的默认更新间隔。灵活的间隔用间隔和期间进行定义，其中：

- 间隔 – 在指定时间段内的更新间隔
- 期间 – 灵活间隔有效的时间段（周期格式请参阅详细说明时间期间）

可以定义多达七种灵活的时间间隔。如果多个灵活间隔设置有冲突，则在冲突周期中使用其中最小的间隔值。请注意，如果灵活间隔的最小值为 “0”，则不会进行轮询。在灵活间隔之外，使用默认更新间隔。

请注意，如果灵活间隔等于周期的长度，则该监控项将被精确检查一次。如果灵活间隔大于周期，则可能会检查该监控项一次，或者完全不检查该监控项（因此不建议这样配置）。如果灵活间隔小于周期，监控项将至少被检查一次。

如果灵活间隔设置为 “0”，则在灵活间隔期间不轮询监控项，并在周期结束后根据默认更新间隔恢复轮询。示例：

间隔	周期	描述
10	1-5,09:00-18:00	监控项将在工作时间内每 10 秒检查一次。
0	1-7,00:00-7:00	监控项不会在夜间检查。
0	7-7,00:00-24:00	监控项不会在星期日检查。
60	1-7,12:00-12:01	监控项将在每天 12:00 点检查。请注意，这被用作计划检查的变通方法，且从 Zabbix 3.0 开始建议使用调度间隔来实现。

调度

调度用于在特定时间检查监控项。虽然默认的自定义时间间隔是灵活，但是调度常用于指定独立执行的检查计划。

调度定义为: `md<filter>wd<filter>h<filter>m<filter>s<filter>` 其中:

- **md** - month days
- **wd** - week days
- **h** - hours
- **m** - minutes
- **s** – seconds

`<filter>` 用于指定其前缀的值 (日, 时, 分, 秒) 并被定义为:: `[<from>[-<to>]][/<step>][,<filter>]` 其中:

- `<from>` 和 `<to>` 定义匹配值的范围（包括）。如果忽略 `<to>` 则过滤器匹配 `<from> - <from>` 范围。如果 `<from>` 也被省略，则过滤器匹配所有可能的值。
- `<step>` 通过该范围定义数字值的跳过。默认情况下，`<step>` 的值为 `1`，这意味着所有定义范围的值都匹配。

<step> 通过该范围定义数字值的跳过。默认情况下，<step> 的值为 1，这意味着所有定义范围的值都匹配。虽然过滤器定义是可选的，但必须至少使用一个过滤器。过滤器必须有一个范围或定义的 <step> 值。

如果没有定义低级别过滤器，则一个空的 filter 既与 “0” 匹配，又匹配所有可能的值。例如，如果省略小时过滤器，仅当分钟和秒的过滤器也被省略则只有 “0” 小时将匹配，否则空的小时过滤器将匹配所有小时值。

过滤器前缀的有效 <from> 和 <to> 值分别为：

前缀	描述	*<from>*	*<to>*
md	Month days	1-31	1-31
wd	Week days	1-7	1-7
h	Hours	0-23	0-23
m	Minutes	0-59	0-59
s	Seconds	0-59	0-59

<from> 值必须小于或等于 <to> 值。<step> 值必须大于或等于 1 且小于或等于 <to> - <from>。

单个数字月份、小时、分钟和秒值可以前缀为 0。例如 md01-31 和 h/02 是有效间隔，但 md01-031 和 wd01-07 无效。

在 Zabbix 前端，多个调度间隔以单独的行输入。在 Zabbix API 中，它们连接成单个字符串，以分号; 作为分隔符。

如果同一个时间匹配了几个间隔，则只执行一次。例如，wd1h9;h9 将在星期一上午 9 点执行一次。

示例:

间隔	描述
m0-59	每分钟执行一次
h9-17/2	从 9:00 开始每 2 小时执行一次（9:00，11:00 ...）
m0,30 or m/30	在每小时的 hh:00 和 hh:30 执行
m0,5,10,15,20,25,30,35,40,45,50,55 or m/5	每 5 分钟执行
wd1-5h9	每周一至周五 9:00
wd1-5h9-18	每个星期一到星期五在 9：00,10：00，...，18:00
h9,10,11 or h9-11	每天上午 9:00，10:00 和 11:00
md1h9m30	每个月的第一天在 9:30
md1wd1h9m30	如果是星期一，每个月的第一天在 9:30 执行
h9m/30	在 9:00，9:30 执行
h9m0-59/30	在 9:00，9:30 执行
h9,10m/30	在 9:00，9:30，10:00，10:30 执行
h9-10m30	在 9:30，10:30 执行
h9m10-40/30	在 9:10，9:40 执行
h9,10m10-40/30	在 9:10，9:40，10:10，10:40 执行
h9-10m10-40/30	在 9:10，9:40，10:10，10:40 执行
h9m10-40	在 9:10，9:11，9:12，... 9:40 执行
h9m10-40/1	在 9:10，9:11，9:12，... 9:40 执行
h9-12,15	在 9:00，10:00，11:00，12:00，15:00 执行
h9-12,15m0	在 9:00，10:00，11:00，12:00，15:00 执行
h9-12,15m0s30	在上午 9 时 30 分，上午 10 时 30 分，11 时 30 分，12 时 30 分，15 时 30 分执行
h9-12s30	在 9:00:30，9:01:30，9:02:30 ... 12:58:30，12:59:30 执行
h9m/30;h10 (*API*-指定语法)	在 9:00，9:30，10:00 执行
h9m/30 h10 (在前端的另一行添加)	在 9:00，9:30，10:00 执行

Aligning time zones for proxies and agent 2

Note that Zabbix proxies and agent 2 use their local time zones when processing scheduling intervals.

For this reason, when scheduling intervals are applied to items monitored by Zabbix proxy or agent 2 active items, it is recommended to set the time zone of the respective proxies or agent 2 the same as Zabbix server, otherwise the queue may report item delays incorrectly.

The time zone for Zabbix proxy or agent 2 can be set using the environment variable `TZ` in the `systemd` unit file:

```
[Service]
...
Environment="TZ=Europe/Amsterdam"
```

2 监控项值预处理

概述

预处理允许为接收到的监控项值定义转换规则。在将值保存到数据库之前，可以进行一次或多次转换。

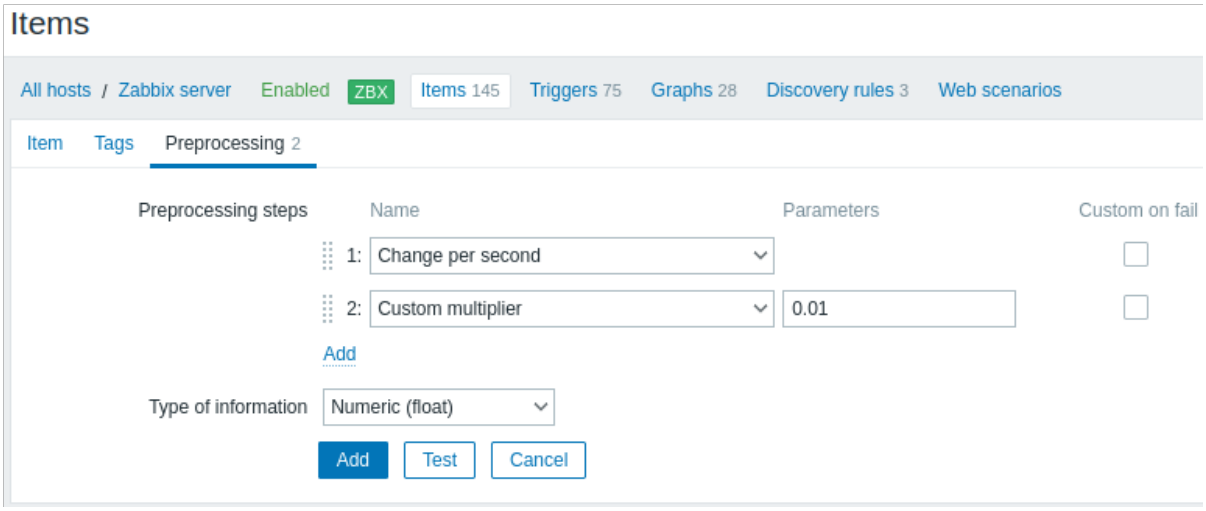
转换按照它们定义的顺序执行。预处理是由 Zabbix server 或者 Zabbix proxy (如果监控项通过 proxy 进行监控) 执行。

请注意，转换成所需要的值类型是在预处理步骤的后期执行的，如果有需要，转换也可以进行相应的预处理步骤。详见preprocessing details了解更多技术细节。

还可以参考: Usage examples

配置

预处理规则在监控项的 预处理标签中进行配置 。



Attention:
任何预处理步骤失败都将导致监控项变成不支持的，除非使用自定义失败选项（*Custom on fail*）指定自定义错误处理办法。

对于日志监控项，日志元数据 (没有值) 将总是重置监控项的不支持的状态，使监控项再次受到支持，即使在从 agent 接收到日志值后发生了初始错误。

用户宏 在监控项值预处理参数中支持带有上下文的用户宏，包含 JavaScript 代码。

Note:
当宏被它的值替换时，上下文将被忽略。宏值将直接插入到代码中，在将值放入 JavaScript 代码之前不能添加额外的转义。请注意，在某些情况下，这可能会导致 JavaScript 错误。

类型		
	转换动作	描述
文本	正则表达式	将值与 <pattern> 正则表达式匹配，并将值替换为 <output>。正则表达式支持使用 \N 序列提取最多 10 个捕获的组。 输入值不匹配将使该监控项不受支持。 参数: **pattern** - 正则表达式 **output** - 输出格式模板。\N（其中 N=1...9）转义序列将替换为第 N 个匹配组。\0 转义序列将替换为匹配的文本。 请参考正则表达式 部分现有示例。 如果你勾选了 自定义失败复选框, 则在预处理步骤失败时，该监控项将不受支持，并且可以指定自定义错误处理选项：丢弃值、设置指定值或设置指定错误消息。

类型		
	替换	找到搜索字符串并将其替换为另一个（或没有）。将替换所有出现的搜索字符串。 参数: **search string** - 要查找和替换的字符串，区分大小写（必需） **replacement** - 用于替换搜索字符串的字符串。替换字符串也可以为空，有效地允许在找到时删除搜索字符串。 可以使用转义序列来搜索或替换换行符、回车符、制表符和空格 “\n \r \ t \s”; 反斜杠可以转义为 “\\”，转义序列可以转义为 “\\n”。在底层自动发现期间自动完成换行符、回车、制表符的转义。
	截取	从值的开头和结尾删除指定的字符。
	右截取	从值的末尾删除指定的字符。
	左截取	从值的开头删除指定的字符。
结构化数据		

	XML XPath	使用 XPath 功能从 XML 数据中提取值或片段。
		要使此选项起作用，必须在编译 Zabbix server 时使用 libxml 参数。
		示例:
		`number(/document/item/value)`
		将从
		`<document><item><value>10</value></item>`
		提取 10
		`number(/document/item/@attribute)`
		将从 `<document><item attribute="10"></item></document>`
		提取 10
		`/document/item` will extract
		`<item><value>10</value></item>`
		from
		`<document><item><value>10</value></item>`
		注意，不支持命名空间。
		如果您勾选了 自定义失败复选框，则在预处理步骤失败的情况下，该监控项将不会变得不受支持，并且可以指定自定义错误处理选项：丢弃值、设置指定值或设置指定错误信息。

	JSON Path	使用JSONPath 功能 从 JSON 数据中提取值或片段。 如果您勾选了 自定义失败复选框，则在预处理步骤失败的情况下，该监控项将不会变得不受支持，并且可以指定自定义错误处理选项：丢弃值、设置指定值或设置指定错误信息。
	CSV to JSON	将 CSV 文件数据转换为 JSON 格式。 有关更多信息，请参见: CSV to JSON 预处理.
	XML to JSON	将 XML 格式的数据转换为 JSON。 有关更多信息，请参见: 序列化规则。 如果您勾选了 自定义失败复选框，则在预处理步骤失败的情况下，该监控项将不会变得不受支持，并且可以指定自定义错误处理选项：丢弃值、设置指定值或设置指定错误信息。
算术		

类型		
	自定义乘数	将该值乘以指定的整数或浮点值。 使用此选项将以 KB、MBps 等形式接收的值转换为 B、Bps。否则 Zabbix 无法正确设置 (K, M, G 等) 后缀 (K, M, G etc). 注意如果监控项类型是 数字 *(*无符号*)*, 在传入自定义乘法器之前带有小数部分的传入值将被裁剪 ('0.9' 将变成'0')。 支持: 科学记数法, 例如, 1e+70 (从 2.2 开始); 用户宏和 LLD 宏 (从 4.0 版本开始); 包含宏的字符串, 例如, `{#MACRO}e+10`, `{$MACRO1}e+{$MACRO2}` (从 5.2.3 版本开始) 宏必须解析为整数或浮点数。 如果你使用 自定义失败复选框, 在预处理步骤失败的情况下，该项不会变得不受支持，并且可以指定自定义错误处理选项: 丢弃该值、设置指定值或设置指定的错误消息。
更改		
	简单更改	计算当前值与先前值的差值。 计算为 **value**-**prev_value**，其中 *value* - 当前值；*prev_value* - 以前收到的值 此设置可用于衡量不断增长的值。如果当前值小于前一个值，Zabbix 会丢弃该差异（不存储任何内容）并等待另一个值。 每个监控项只允许进行一次更改操作。 如果您选中 *Custom on fail* 复选框，如果预处理步骤失败，该项目将不会变得不受支持，并且可以指定自定义错误处理选项：丢弃值、设置指定值或设置指定错误消息。
	每秒更改	计算每秒速度的值变化（当前值与先前值之间的差异）。 计算为 (**value**-**prev_value**)/(**time**-**prev_time**), 其中 *value* - 当前值；*prev_value* - 先前收到的值；*time* - 当前时间戳；*prev_time* - 前一个值的时间戳。 此设置对于获取不断增长的值的每秒速度非常有用。如果当前值小于前一个值，Zabbix 会丢弃该差异（不存储任何内容）并等待另一个值。这有助于正确处理，例如，32 位 SNMP 计数器的包装（溢出）。 注意：由于此计算可能产生浮点数，建议设置 "信息类型" 到 *Numeric (*浮点型*)*，即使传入的原始值是整数。这对于小数部分很重要的小数尤其重要。如果浮点值很大并且可能超过'float' 字段长度，在这种情况下整个值可能会丢失，实际上建议使用 *Numeric (unsigned)* 并且只修剪小数部分。 每个监控项只允许进行一次更改操作。 如果您选中 *Custom on fail* 复选框，则在预处理步骤失败的情况下，该监控项将不会变得不受支持，并且可以指定自定义错误处理选项：要么丢弃 value，设置一个指定的值，或者设置一个指定的错误信息。

类型		
数字系统		
	布尔到十进制	将值从布尔格式转换为十进制。文本表示被转换为 0 或 1。因此，“TRUE” 存储为 1，“FALSE” 存储为 0。所有值都以不区分大小写的方式匹配。当前识别的值为： *TRUE* - true, t, yes, y, on, up, running, enabled, available, ok, master *FALSE* - false, f, no, n, off, down, unused, disabled, unavailable, err, slave 此外，任何非零数值都被视为 TRUE，而零被视为 FALSE。 如果您标记 *Custom on fail* 复选框，如果预处理步骤失败，该监控项将不会变得不受支持，并且可以指定自定义错误处理选项：丢弃值、设置指定值或设置指定错误消息。
	八进制转十进制	将值从八进制格式转换为十进制。 如果您选中 *Custom on fail* 复选框，则在预处理步骤失败的情况下，该监控项将不会变得不受支持，并且可以指定自定义错误处理选项：要么丢弃 value，设置一个指定的值，或者设置一个指定的错误信息。
	十六进制转十进制	将值从十六进制格式转换为十进制。 如果您选中 *Custom on fail* 复选框，则在预处理步骤失败的情况下，该监控项将不会变得不受支持，并且可以指定自定义错误处理选项：要么丢弃 value，设置一个指定的值，或者设置一个指定的错误信息。
自定义脚本		
	JavaScript	在单击参数字段或铅笔图标时出现的块中输入 JavaScript 代码。 注意, 可用的 JavaScript 长度取决于所使用的使用的数据库. 有关更多信息，请参见:Javascript 预处理.
验证		
	在范围内	通过指定最小/最大值（包括）来定义值应在的范围。 接受数值（包括任意位数、可选的小数部分和可选的指数部分、负值）。可以使用用户宏和底层自动发现宏。最小值应小于最大值。 必须至少存在一个值。 如果您选中 *Custom on fail* 复选框，则在预处理步骤失败的情况下，该监控项不会变得不受支持，并且有可能指定自定义错误处理选项：丢弃值、设置指定值或设置指定错误消息。
	匹配正则表达式	指定一个值必须匹配的正则表达式。 如果您选中 *Custom on fail* 复选框，则在预处理步骤失败的情况下，该监控项将不会变得不受支持，并且可以指定自定义错误处理选项：要么丢弃值，设置一个指定的值，或者设置一个指定的错误信息。

类型		
	不匹配正则表达式	指定一个值不能匹配的正则表达式。 如果您选中 *Custom on fail* 复选框，则在预处理步骤失败的情况下，该监控项将不会变得不受支持，并且可以指定自定义错误处理选项：要么丢弃该值，设置一个指定的值，或设置一个指定的错误消息。
	检查 *JSON* 中的错误	检查位于 JSONpath 的应用程序级错误消息。如果成功并且消息不为空则停止处理；否则，使用此预处理步骤之前的值继续处理。请注意，这些外部服务错误会按原样报告给用户，而不添加预处理步骤信息。 如果无法解析无效 JSON，则不会报告错误。 如果您选中 *Custom on fail* 复选框，如果预处理步骤失败，该监控项将不会变得不受支持，并且可以指定自定义错误处理选项：丢弃值、设置指定值或设置指定错误消息。
	检查 *XML* 中的错误	检查位于 XPath 的应用程序级错误消息。如果成功并且消息不为空则停止处理；否则，使用此预处理步骤之前的值继续处理。请注意，这些外部服务错误将按原样报告给用户，而不添加预处理步骤信息。 如果无法解析无效 XML，则不会报告错误。 如果您选中 *Custom on fail* 复选框，如果预处理步骤失败，该监控项将不会变得不受支持，并且可以指定自定义错误处理选项：丢弃值、设置指定值或设置指定错误消息。
	使用正则表达式检查错误	使用正则表达式检查应用程序级错误消息。如果成功并且消息不为空则停止处理；否则，使用此预处理步骤之前的值继续处理。请注意，这些外部服务错误按原样报告给用户，无需添加预处理步骤信息。 参数： **pattern** - 正则表达式 **output** - 输出格式模板。\N（其中 N=1…9）转义序列将替换为第 N 个匹配组。\0 转义序列将替换为匹配的文本。 如果您选中 *Custom on fail* 复选框，则在预处理步骤失败的情况下，该监控项将不会变得不受支持，并且可以指定自定义错误处理选项：要么丢弃该值，设置一个指定的值，或者设置一个指定的错误消息。
	检查不支持的值	检查检索监控项值是否有错误。通常这会导致监控项变得不受支持，但您可以通过指定 *Custom on fail* 错误处理选项来修改该行为：丢弃值，设置指定值（在这种情况下，监控项将保持支持并且 value 可以在触发器中使用）或设置指定的错误消息。请注意，对于此预处理步骤，失败时自定义复选框灰显并始终标记。 此步骤始终作为第一个预处理步骤执行，并在保存对监控项的更改后置于所有其他步骤之上。它只能使用一次。 从 5.2.0 开始支持。
节流		

类型		
	丢弃不变	如果一个值没有改变，则丢弃它。 如果一个值被丢弃，它不会保存在数据库中，并且 Zabbix 服务器不知道该值已被接收。不会评估触发器表达式，因此不会创建/解决相关触发器的问题。函数仅基于实际保存在数据库中的数据起作用。由于趋势是基于数据库中的数据构建的，如果一个小时内没有保存任何值，则该小时也不会有趋势数据。 只能为一个监控项指定一个限制选项。 * 注意 * Zabbix 代理监控的监控项有可能非常小的值差异（小于 0.000001）不会被代理正确丢弃，但如果 Zabbix 服务器数据库 尚未升级，则作为相同的值存储在历史记录中
	丢弃不变的心跳	如果值在定义的时间段内（以秒为单位）未更改，则丢弃该值。 支持正整数值来指定秒数（最小值 - 1 秒）。该字段可以使用时间后缀（例如 30s、1m、2h、1d）。在该字段中可以使用用户宏和底层自动发现宏。 如果一个值被丢弃，它不会保存在数据库中，并且 Zabbix 服务器不知道该值已被接收。不会评估触发器表达式，因此不会创建/解决相关触发器的问题。函数仅基于实际保存在数据库中的数据起作用。由于趋势是基于数据库中的数据构建的，如果一个小时内没有保存任何值，则该小时也不会有趋势数据。 只能为一个监控项指定一个限制选项。 注意 Zabbix 代理监控的监控项有可能非常小的值差异（小于 0.000001）不会被代理正确丢弃，但如果 Zabbix 服务器数据库 尚未升级，则作为相同的值存储在历史记录中。
Prometheus		
	Prometheus pattern	使用以下查询从 Prometheus 指标中提取所需数据。 有关更多信息，请参考：Prometheus checks
	Prometheus to JSON	将所需的 Prometheus 指标转换为 JSON。 有关更多信息，请参考：Prometheus checks

Attention:
对于更改和限制预处理步骤，需要使用以前的值来计算/比较新值。以前的值由预处理管理器处理，预处理步骤配置在进行更改或 Zabbix server/proxy 重新启动时重置。由于先前的值重置:

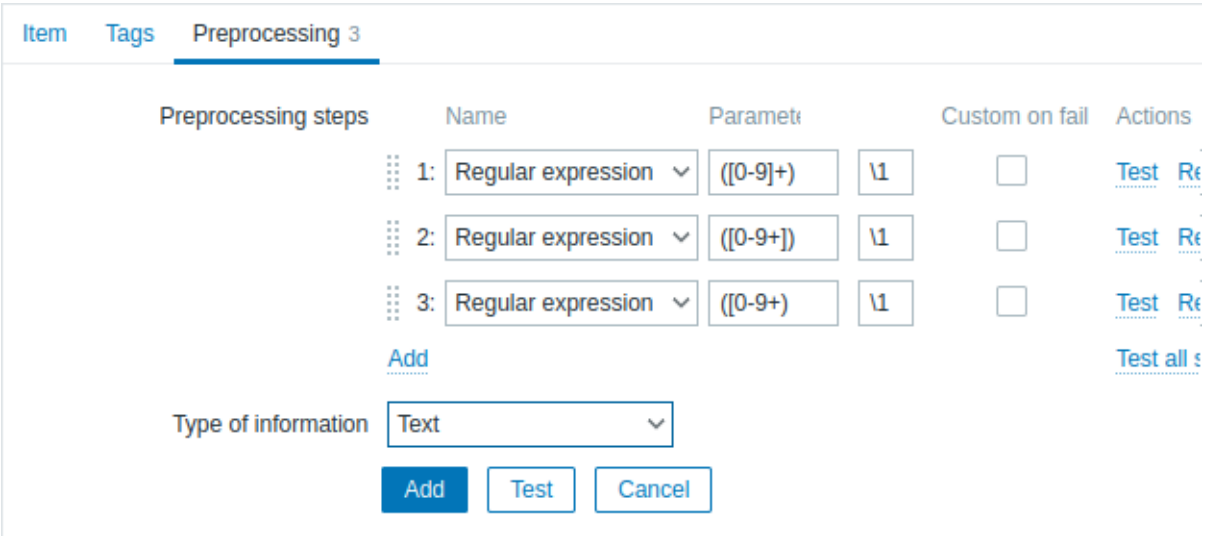
- 对于简单改变, 每秒更改步骤 - 下一个值将被忽略，因为没有先前的值来计算更改;
- 对于 丢弃没有改变的数据. 带心跳检查丢弃不变化的数据步骤 - 下一个值永远不会被丢弃, 即使它应该因为丢弃规则而被丢弃，也不会丢弃。

当定义了至少一个预处理步骤时，监控项的信息类型参数显示在选项卡的底部。如果需要，可以在不离开 *Preprocessing* 选项卡的情况下更改信息类型。详细参数说明见创建监控项 。

Note:
如果您对信息类型设置为 *Numeric (unsigned)* 的监控项使用自定义乘数或将值存储为 *Change per second* 并且得到的计算值实际上是一个浮点数，通过修剪小数部分并将值存储为整数，计算值仍然被接受为正确的值。

Testing

测试预处理步骤有助于确保复杂的预处理管道产生预期的结果，而无需等待接收和预处理监控项值。

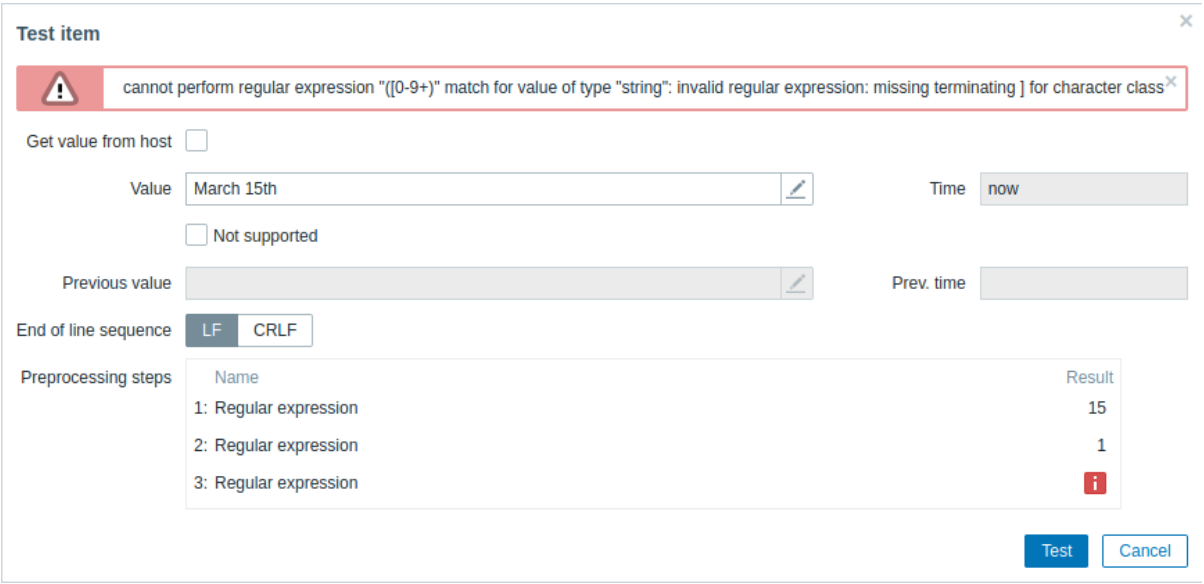


可以测试：

- 与假设值相比
- 与主机的实际值相比较

每个预处理步骤都可以单独测试，也可以一起测试所有步骤。当您分别单击动作块中的 *Test* 或 *Test all steps* 按钮时，将打开一个测试窗口。

检验假设值



参数	描述
从主机获取值	如果您想测试假设值，请不要选中此复选框。 另请参阅：测试真实值.
值	输入要测试的输入值。 单击参数字段或查看/编辑按钮将打开一个文本区域窗口，用于输入值或代码块。
不支持的	选中此复选框可测试不受支持的值。 此选项可用于测试 检查不受支持的值预处理步骤。
时间	显示输入值的时间：now（只读）。
以前的值	输入之前的输入值进行比较。 仅适用于 更改和 节流预处理步骤。
以前的时间点	输入之前的输入值时间进行比较。 仅适用于更改和 节流预处理步骤。 默认值基于监控项的“更新间隔”字段值（如果为“1m”，然后这个字段用 now-1m 填充）。如果未指定任何内容或用户无权访问主机，则默认为 now-30s。

参数	描述
宏	如果使用了任何宏，它们将与它们的值一起列出。出于测试目的，这些值是可编辑的，但更改只会保存在测试上下文中。
行尾序列	为多行输入值选择行尾序列： **LF** - LF（换行）序列 **CRLF** - CRLF（回车换行）序列。
预处理步骤	列出了预处理步骤；单击 *Test* 按钮后，将显示每个步骤的测试结果。如果该步骤测试失败，则会显示错误图标。鼠标悬停时会显示错误描述。如果为步骤指定了“失败时自定义”并执行了该操作，则会在预处理测试步骤行之后立即出现一个新行，显示已执行的操作和结果产生（错误或值）。
结果	测试预处理步骤的最终结果在所有步骤一起测试时显示在所有情况下（当您单击测试所有步骤按钮时）。 还显示转换为监控项的值类型的类型，例如“结果转换为数字（无符号）”。

单击 测试以查看每个预处理步骤后的结果。

测试值存储在单个步骤或所有步骤的测试会话之间，允许用户更改预处理步骤或监控项配置，然后返回到测试窗口，而无需重新输入信息。但是，页面刷新时值会丢失。

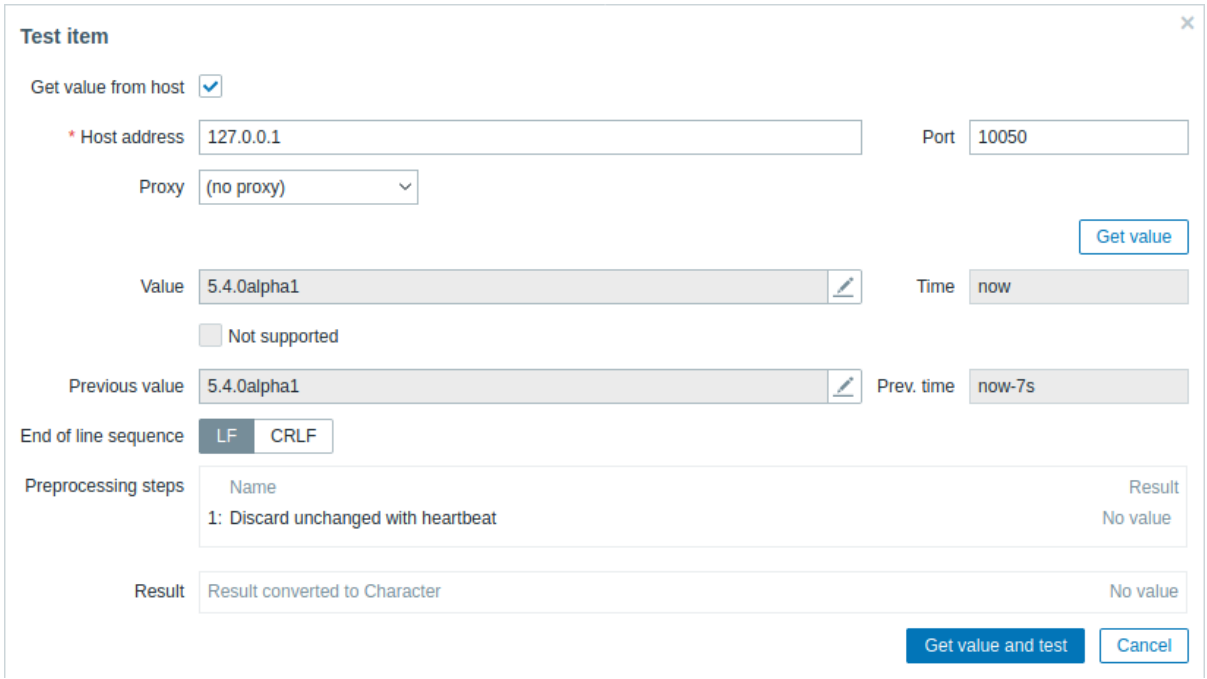
测试由 Zabbix server 完成。前端向服务器发送相应的请求并等待结果。该请求包含输入值和预处理步骤（带有扩展的用户宏）。对于 更改和 节流步骤，可以指定可选的先前值和时间。服务器响应每个预处理步骤的结果。

所有技术错误或输入验证错误都显示在测试窗口顶部的错误框中。

测试真实值

根据实际值测试预处理:

- 标记 从主机获取值复选框
- 输入或验证主机参数（主机地址、端口、代理名称/无代理）和监控项特定的详细信息（例如 SNMPv2 社区或 SNMPv3 安全凭证）。这些字段是上下文感知的：
  - 这些值在可能的情况下预先填写，即对于需要 agent 的监控项，通过从主机的选定 agent 接口获取信息
  - 模板监控项的值必须手动填写
  - 解析纯文本宏值
  - 值（或部分值）为机密或 Vault 宏的字段为空，必须手动输入。如果任何监控项参数包含秘密宏值，则会显示以下警告消息：“监控项包含具有秘密值的用户定义的宏。应手动输入这些宏的值。”
  - 在监控项类型的上下文中不需要时禁用这些字段（例如，计算监控项的主机地址和代理字段被禁用）
- 点击 获取值并测试来测试预处理



如果您在监控项配置表单（“显示值”字段）中指定了值映射，监控项测试对话框将在最终结果之后显示另一行，名为“应用了值映射的结果”。

特定于从主机获取实际值的参数:

参数	描述
从主机获取值	标记此复选框以从主机获取实际值。
主机地址	输入主机地址。 此字段由监控项主机接口的地址自动填充。
端口	输入主机端口。 此字段由监控项主机接口的端口自动填充。
SNMP 接口的附加字段 * (SNMP* 版本、*SNMP* 社区、上下文名称等)	有关配置 SNMP 接口（v1、v2 和 v3）的更多详细信息。请查看配置 SNMP 监控。 这些字段会从监控项主机接口自动填充。
代理	如果主机由代理监控，请指定代理。 此字段由主机的代理（如果有）自动填充。

有关其余参数，请参考Testing hypothetical value 。

1 使用示例

概述

本节介绍使用预处理步骤完成一些实际任务的示例。

过滤 VMware 事件日志记录

使用正则表达式预处理过滤 VMWare 事件日志的不必要事件。

1. 在工作的 VMWare Hypervisor 主机上，检查事件日志项 `vmware.eventlog[<url>,<mode>]` 是否存在并且工作正常。请注意，如果在主机创建期间已链接 *Template VM VMWare* 模板，则事件日志项可能已经存在于管理程序上。

2. 在 VMWare Hypervisor 主机上创建一个 “日志” 类型的依赖项 并将事件日志项设置为其主项。

在依赖项的 “预处理” 选项卡中，选择 “匹配正则表达式” 验证选项和填充模式，例如：

```
".* logged in .*" - 过滤事件日志中的所有日志事件
"\bUser\s+\K\S+" -  只筛选事件日志中带有用户名的行
```

Attention:
如果正则表达式不匹配，则依赖项变得不受支持，并显示相应的错误消息。为了避免这种情况，请选中 “失败时自定义” 复选框并选择丢弃不匹配的值。

另一种允许使用匹配组和输出控制的方法是在 “预处理” 选项卡中选择 “正则表达式” 选项并设置参数，例如:

```
pattern: ".*logged in.*", output: "\0" - 过滤事件日志中的每行日志事件
pattern "User (.*?)(?=\ )", output: "\1" - 仅截取事件日志中筛选用户名信息
```

Checking retrieved value type

This example uses the Custom multiplier preprocessing step to check if the retrieved item value type is numeric.

In the *Preprocessing* tab of an item, select the ”Custom multiplier” preprocessing step and set the following parameter:

```
##### Multiplies the retrieved value by 1:
number: 1
```

Attention:
If preprocessing fails (e.g., input is not numeric), then the item becomes unsupported with a corresponding error message. To avoid this, mark the ”Custom on fail” checkbox and select an option such as discarding the value or setting a custom one.

Checking for not supported value

This example uses the Check for not supported value preprocessing step to check if the item value could not be retrieved.

When a Zabbix server/proxy poller process attempts to collect an item value, it may:

- Return a valid result.
- Return a result that initially seems valid but may become unsupported later (e.g., due to a value type mismatch after preprocessing).

- Return an error of collecting the value, causing the item to become unsupported. Common causes include:
  - Unknown item key (for Zabbix agent, Simple check, or Zabbix internal items)
  - Unknown OID (SNMP agent), unknown sensor (IPMI agent), or no JMX metric (JMX agent)
  - Cannot read trap file (SNMP trap)
  - Script not found (External check)
  - No such URL (HTTP agent)
  - Login failed (SSH agent, TELNET agent)
  - Invalid formula syntax (Calculated), JavaScript syntax error (Script), or invalid SQL (Database monitor)

To detect and handle errors of collecting item values, you can use the ”Check for not supported value” preprocessing step. Note that this step is always executed first and only detects errors that occur before preprocessing begins.

In the *Preprocessing* tab of an item, select the ”Check for not supported value” preprocessing step.

Then, use the *Custom on fail* option to discard the value (in this case, the error), set a custom value, or return a custom error message. Please note that discarded values are not stored in the database; as a result, triggers are not evaluated and trend data is not generated.

2 预处理细节

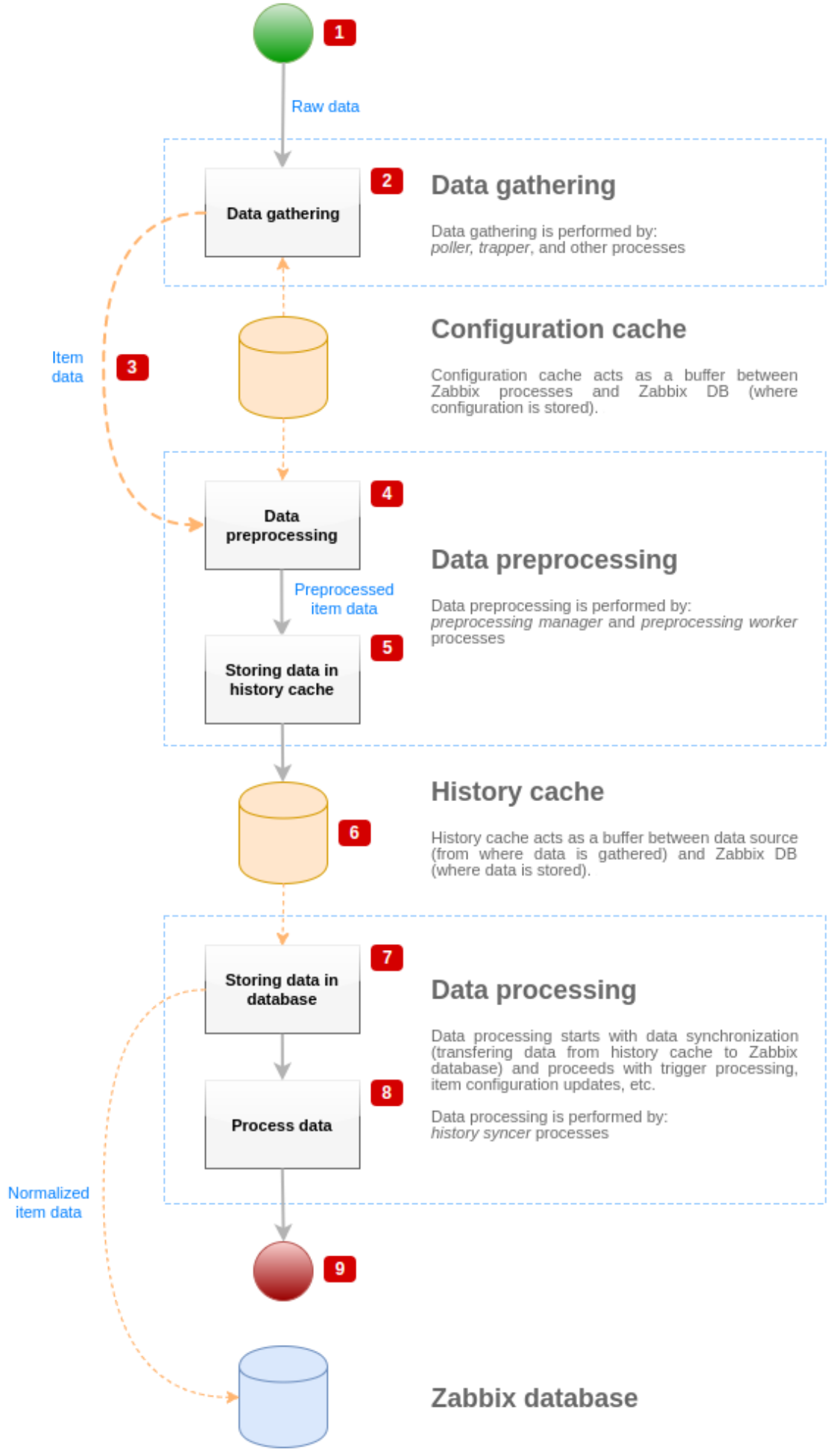
概述

本节提供监控项值预处理详细信息。监控项值预处理允许为接收到的监控项值定义和执行转换规则 。

预处理由预处理管理进程管理，该进程在 Zabbix 3.4 中添加，用于执行预处理步骤。所有来自不同数据收集器的值（无论是否经过预处理）都会在添加到历史缓存之前通过预处理管理器。基于套接字的 IPC 通信作用于数据收集器（轮询器、捕获器等）和预处理进程之间。Zabbix server 或 Zabbix proxy（由代理监控的监控项）负责执行预处理步骤。

监控项值预处理

为了可视化从数据源到 Zabbix 数据库的数据流，我们可以使用下面的简化图：


1
Raw data
Data gathering
2
Data gathering
Data gathering is performed by:
poller, trapper, and other processes
Item data
3
Configuration cache
Configuration cache acts as a buffer between Zabbix processes and Zabbix DB (where configuration is stored).
Data preprocessing
4
Data preprocessing
Data preprocessing is performed by:
preprocessing manager and preprocessing worker processes
Preprocessed item data
Storing data in history cache
5
History cache
6
History cache acts as a buffer between data source (from where data is gathered) and Zabbix DB (where data is stored).
Storing data in database
7
Data processing
Data processing starts with data synchronization (transfering data from history cache to Zabbix database) and proceeds with trigger processing, item configuration updates, etc.
Process data
8
Data processing is performed by:
history syncer processes
Normalized item data
9
Zabbix database

上图仅以简化形式显示了与监控项值处理相关的流程、对象和操作。该图没有显示有条件的方向变化、错误处理或循环。预处理管理器的本地数据缓存也没有显示，因为它不直接影响数据流。此图的目的是显示监控项价值处理中涉及的流程及其交互方式。

- 数据收集从数据源的原始数据开始。此时数据只包含 ID、时间戳和值（也可以是多个值）
- 无论使用哪种类型的数据收集器，对于主动或被动检查、陷阱监控项等的想法都是相同的，因为它只更改数据格式和通信启动器（任何一个数据收集器都在等待连接和数据，或数据收集器发起通信并请求数据）。验证原始数据，从配置缓存中检索监控项配置（使用配置数据丰富数据）。
- 基于套接字的 IPC 机制用于将数据从数据收集器传递到预处理管理器。此时数据收集器继续收集数据，无需等待预处理管理器的响应。
- 执行数据预处理。这包括执行预处理步骤和依赖项处理。

Note:
如果任何预处理步骤失败，则在执行预处理时，监控项可以将其状态更改为不支持。

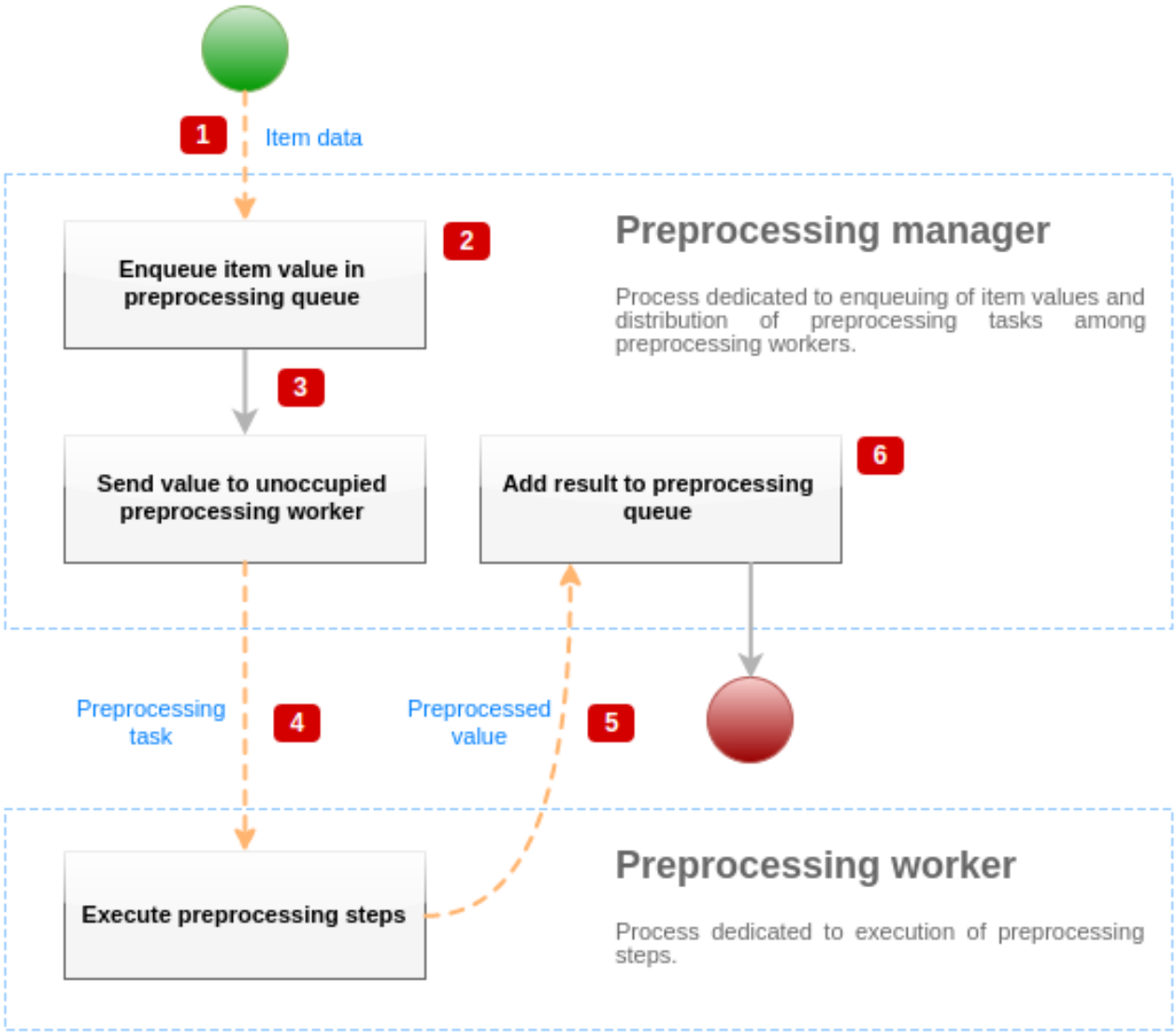
- 来自预处理管理器的本地数据缓存的历史数据正在刷新到历史缓存中。
- 此时数据流停止，直到历史缓存的下一次同步（当历史同步器进程执行数据同步时）。
- 同步过程从数据规范化开始，将数据存储在 Zabbix 数据库中。数据规范化执行到所需监控项类型（监控项配置中定义的类型）的转换，包括基于这些类型允许的预定义大小截断文本数据（HISTORY_STR_VALUE_LEN 用于字符串，HISTORY_TEXT_VALUE_LEN 用于文本和 HISTORY_LOG_VALUE_LEN 用于日志值）。规范化完成后，数据正在发送到 Zabbix 数据库。

Note:
如果数据规范化失败（例如，当文本值无法转换为数字时），监控项可以将其状态更改为不支持。

- 正在处理收集的数据 - 检查触发器，如果监控项不支持，则更新监控项配置等。
- 从监控项值处理的角度来看，这被认为是数据流的结束。

监控项值预处理

为了可视化数据预处理过程，我们使用下面的简化图来展示：



上图仅以简化形式显示了与监控项值预处理相关的流程、对象和主要操作。该图没有显示有条件的方向变化、错误处理或循环。这张图只显示了一个预处理进程（在现实生活中可以使用多个预处理进程），只处理一个监控项值，我们假设该监控项需要执行至少一个预处理步骤。此图的目的是展示监控项值预处理管道背后的理念。

- 使用基于套接字的 IPC 机制将监控项数据和监控项值传递给预处理管理器。
- 监控项被放置在预处理队列中。

Note:
监控项可以放在预处理队列的末尾或开头。Zabbix 内部监控项总是放在预处理队列的开头，而其他监控项类型在最后排队。

- 此时数据流停止，直到至少有一个未占用（即不执行任何任务）预处理进程。
- 当预处理进程可用时，将向它发送预处理任务。
- 预处理完成后（预处理步骤执行失败或成功），预处理值被传递回预处理管理器。
- 预处理管理器将结果转换为所需格式（由监控项值类型定义）并将结果放入预处理队列。如果当前监控项有依赖项，则依赖项也将添加到预处理队列中。依赖项在主要监控项之后的预处理队列中排队，但仅适用于有值设置且不处于不支持状态的主要监控项。

监控项值处理流水线

监控项值处理分多个步骤（或阶段）在多个进程中执行。这可能导致：

- 依赖项可以接收值，而主要监控项不能。这可以通过使用以下用例来实现：
  - 主要监控项具有值类型 `UINT`，(可以使用陷阱监控项)，依赖项具有值类型 `TEXT`。
  - 主要监控项和依赖项都不需要预处理步骤。
  - 文本值（如 “abc”）应传递给主要监控项。
  - 由于没有要执行的预处理步骤，预处理管理器检查主项是否处于不支持状态以及是否设置了值（两者都为真）并将具有与主要监控项相同的值的依赖项排入队列（因为没有预处理步骤）。
  - 当主要监控项和依赖项都达到历史同步阶段时，主要监控项变为不支持，因为值转换错误（文本数据无法转换为无符号整数）。

因此，依赖项收到一个值，而主要监控项状态将变为不支持。

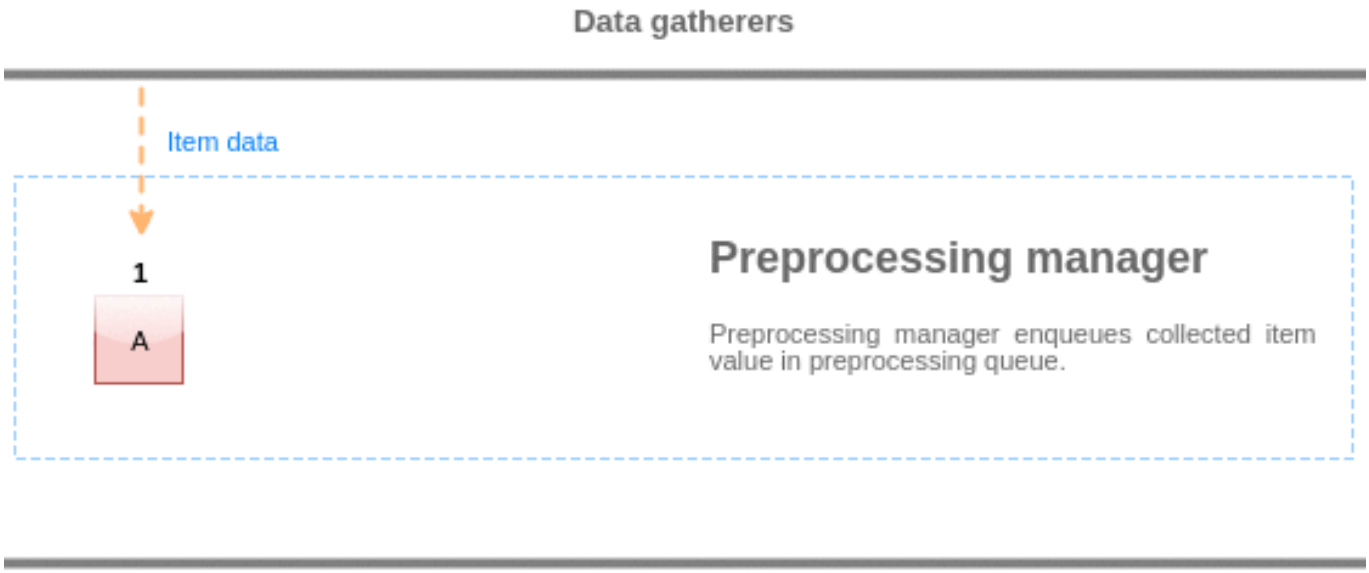
- 依赖项接收主要监控项历史记录中不存在的值。用例与前一个非常相似，除了主要监控项类型。例如，如果主要监控项使用 CHAR 类型，则主要监控项值将在历史同步阶段被截断，而依赖项将从主要监控项的初始（未截断）值接收它们的值。

预处理队列

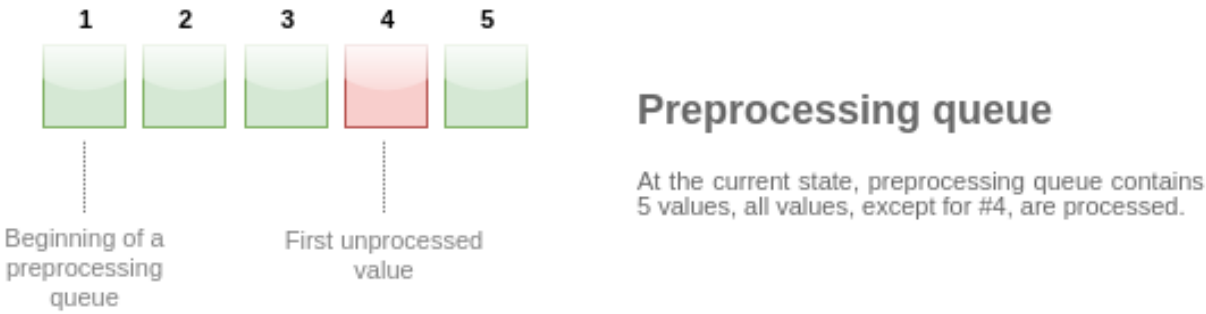
预处理队列是一种 FIFO 数据结构，它存储值，保留值由预处理管理器检索的顺序。FIFO 逻辑有多个例外：

- 内部监控项在队列的开头排队
- 依赖项始终排在主要监控项之后

我们使用下面的简化图来可视化展示预处理队列的逻辑：



预处理队列中的值从队列的开头刷新到第一个未处理的值。因此，例如，预处理管理器将刷新值 1、2 和 3，但不会刷新值 5，因为值 4 尚未处理：



刷新后队列中只剩下两个值（4 和 5），将值添加到预处理管理器的本地数据缓存中，然后将值从本地缓存传输到历史缓存。预处理管理器可以在单项模式或批量模式下从本地数据缓存中刷新值（用于依赖项和批量接收的值）。

预处理进程

Zabbix server 配置文件允许用户设置预处理工作进程的数量。StartPreprocessors 配置参数应用于设置预处理进程的预分配实例数。预处理进程的最佳数量可以由许多因素决定，包括“可预处理”监控项（需要执行任何预处理步骤的监控项）的数量、数据收集过程的数量、监控项预处理的平均步骤数等。

但是假设没有像解析大型 XML/JSON 块这样的繁重的预处理操作，预处理进程的数量可以匹配数据收集器的总数。这样，大多数情况下（除了来自收集器的数据大量进入的情况）至少有一个空闲的预处理进程来处理收集的数据。

Warning:
太多的数据收集进程（轮询器、无法访问的轮询器、ODBC 轮询器、HTTP 轮询器、Java 轮询器、pingers、陷阱器、代理轮询器）连同 IPMI 管理器、SNMP 陷阱器和预处理进程可能会耗尽预处理管理器的每个进程的文件描述符限制。这将导致 Zabbix server 停止（通常在启动后不久，但有时可能需要更多时间）。应修改配置文件或提高限制以避免出现这种情况。

3 JSONPath 功能

概述

本部分提供监控项值预处理步骤中支持的 JSONPath 功能的详细信息。

JSONPath 由用点分隔的段组成。段可以是一个简单的词，如 JSON 值名称、*，也可以是括在方括号 [ ] 中的更复杂的构造。括号段前的分隔点是可选的，可以省略。例如：

路径	描述
`$.object.name`	返回 object.name 的内容。
`$.object['name']`	返回 object.name 的内容。
`$.object.['name']`	返回 object.name 的内容。
`$["object"]['name']`	返回 object.name 的内容。
`$.['object'].["name"]`	返回 object.name 的内容。
`$.object.history.length()`	返回 object.history 数组元素的个数。
`$[?(@.name == 'Object')].price.first()`	返回第一个名为'Object' 的对象的价格字段。
`$[?(@.name == 'Object')].history.first().length()`	返回第一个名为'Object' 的对象的历史数组元素个数。
`$[?(@.price > 10)].length()`	返回 price 大于 10 的对象个数。

参考: 从 JSONPath 中的 LLD 宏值中转义特殊字符.

支持的段

段	描述
`<name>`	按名称匹配对象属性。
`*`	匹配所有对象属性。
`['<name>']`	按名称匹配对象属性。
`['<name>', '<name>', ...]`	通过任何列出的名称匹配对象属性。
`[<index>]`	按索引匹配数组元素。
`[<number>, <number>, ...]`	通过任何列出的索引匹配数组元素。
`[*]`	匹配所有对象属性或数组元素。
`[<start>:<end>]`	按定义的范围匹配数组元素： **<start>** - 要匹配的第一个索引（包括）。如果未指定，则匹配从头开始的所有数组元素。如果为负数，则指定从数组末尾开始的偏移量。 **<end>** - 要匹配的最后一个索引（不包括）。如果未指定，则匹配所有数组元素到最后。如果为负数，则指定从数组末尾开始的偏移量。
`[?(< 表达式 >)]`	通过应用过滤表达式匹配对象/数组元素。

要查找忽略其根节点的匹配段（单独的段），它必须以'..' 为前缀，例如 `$..name` 或 `$..['name']` 返回所有'name' 属性的值。

可以通过在 JSONPath 中添加 ~ 后缀来提取匹配的元素名称。它返回匹配对象的名称或匹配数组项的字符串格式的索引。输出格式遵循与其他 JSONPath 查询相同的规则 - 确定路径结果按 "原样" 返回，不确定路径结果以数组形式返回。但是，提取与明确路径匹配的元素的名称并没有多大意义——它是已知的。

过滤表达式

过滤表达式是固定符号的一种表达式。

支持的操作：

操作	描述	示例
"<text>"	文本常量。	'value: \'1\''
'<text>'		"value: '1'"
<number>	支持科学计数法的数值常数。	123
< 以 $ 开头的 jsonpath>	来自输入文档根节点的 JSONPath 引用的值；只支持明确的路径。	$.object.name
< 以 @ 开头的 jsonpath>	当前对象/元素的 JSONPath 引用的值；只支持明确的路径。	@.name

支持的操作：

操作	类型	描述	结果
-	二进制	减法	数字
+	二进制	加法	数字
/	二进制	除法	数字
*	二进制	乘法	数字
==	二进制	等于	布尔值（1 或 0）
!=	二进制	不等于	布尔值（1 或 0）
	二进制	小于	布尔值（1 或 0）
<=	二进制	小于或等于	布尔值（1 或 0）
>	二进制	大于	布尔值（1 或 0）
>=	二进制	大于或等于	布尔值（1 或 0）
=~	二进制	匹配正则表达式	布尔值（1 或 0）
!	一元	非布尔值	布尔值（1 或 0）
||	二进制	布尔值或	布尔值（1 或 0。
&&	二进制	布尔值和	布尔值（1 或 0）

函数

函数可以用在 JSONPath 的末尾。如果前面的函数返回后面函数接受的值，则可以链接多个函数。

支持的函数：

函数	描述	输入	输出
avg	输入数组中数字的平均值。	数字数组。	数字。
min	输入数组中数字的最小值。	数字数组。	数字。
max	输入数组中数字的最大值。	数字数组。	数字。
sum	输入数组中数字的总和。	数字数组。	数字。
length	输入数组中的元素数量。	数组。	数字。
first	第一个数组元素。	数组。	取决于输入数组内容的 JSON 构造（对象、数组、值）。

JSONPath 聚合函数接受带引号的数值。这意味着如果需要聚合，则将值从字符串类型转换为数字。

不兼容的输入会导致函数产生错误。

输出值

JSONPaths 可以分为确定路径和不确定路径。确定路径只能返回 null 或单个匹配项。不定路径可以返回多个匹配项，基本上是带有分离的、多个名称/索引列表、数组切片或表达式段的 JSONPath。但是，当使用函数时，JSONPath 变得明确，因为函数总是输出单个值。

确定路径返回它所引用的对象/数组/值，而不定路径返回匹配的对象/数组/值的数组。

空格

空格（空格、制表符）可以在括号符号段和表达式中自由使用，例如，`$[ 'a' ][ 0 ][ ?( $.b == 'c' ) ][ : -1 ].first( ).`

字符串

字符串应该用单' 或双" 引号括起来。在字符串内，单引号或双引号（取决于使用哪个引号括起来）和反斜杠 \ 用反斜杠 \ 字符转义。

示例

输入数据

```
{
  "books": [
    {
      "category": "reference",
      "author": "Nigel Rees",
      "title": "Sayings of the Century",
      "price": 8.95,
      "id": 1
    },
    {
      "category": "fiction",
      "author": "Evelyn Waugh",
      "title": "Sword of Honour",
      "price": 12.99,
      "id": 2
    },
    {
      "category": "fiction",
      "author": "Herman Melville",
      "title": "Moby Dick",
      "isbn": "0-553-21311-3",
      "price": 8.99,
      "id": 3
    },
    {
      "category": "fiction",
      "author": "J. R. R. Tolkien",
      "title": "The Lord of the Rings",
      "isbn": "0-395-19395-8",
      "price": 22.99,
      "id": 4
    }
  ],
  "services": {
    "delivery": {
      "servicegroup": 1000,
      "description": "Next day delivery in local town",
      "active": true,
      "price": 5
    },
    "bookbinding": {
      "servicegroup": 1001,
      "description": "Printing and assembling book in A5 format",
      "active": true,
      "price": 154.99
    },
    "restoration": {
      "servicegroup": 1002,
      "description": "Various restoration methods",
      "active": false,
      "methods": [
        {
          "description": "Checmical cleaning",
          "price": 46
        },
        {
          "description": "Pressing pages damaged by moisture",
          "price": 24.5
        },
        {
          "description": "Rebinding torn book",
          "price": 99.49
```

```
        }
      ]
    }
  },
  "filters": {
    "price": 10,
    "category": "fiction",
    "no filters": "no \"filters\""
  },
  "closed message": "Store is closed",
  "tags": [
    "a",
    "b",
    "c",
    "d",
    "e"
  ]
}
```

JSONPath	类型	结果	注释
`$.filters.price`	definite	10	
`$.filters.category`	definite	fiction	
`$.filters['no filters']`	definite	no ”filters”	
`$.filters`	definite	{ ”price”: 10, ”category”: ”fiction”, ”no filters”: ”no \”filters\”” }	
`$.books[1].title`	definite	Sword of Honour	
`$.books[-1].author`	definite	J. R. R. Tolkien	
`$.books.length()`	definite	4	
`$.tags[:]`	indefinite	[”a”, ”b”, ”c”, ”d”, ”e”]	
`$.tags[2:]`	indefinite	[”c”, ”d”, ”e”]	
`$.tags[:3]`	indefinite	[”a”, ”b”, ”c”]	
`$.tags[1:4]`	indefinite	[”b”, ”c”, ”d”]	
`$.tags[-2:]`	indefinite	[”d”, ”e”]	
`$.tags[:-3]`	indefinite	[”a”, ”b”]	
`$.tags[:-3].length()`	definite	2	
`$.books[0, 2].title`	indefinite	[”Sayings of the Century”, ”Moby Dick”]	
`$.books[1]['author', "title"]`	indefinite	[”Evelyn Waugh”, ”Sword of Honour”]	
`$..id`	indefinite	[1, 2, 3, 4]	
`$.services..price`	indefinite	[5, 154.99, 46, 24.5, 99.49]	
`$.books[?(@.id == 4 - 0.4 * 5)].title`	indefinite	[”Sword of Honour”]	此查询表明可以在查询中使用算术运算。当然这个查询可以简化为 `$.books[?(@.id == 2)].title`
`$.books[?(@.id == 2 || @.id == 4)].title`	indefinite	[”Sword of Honour”, ”The Lord of the Rings”]	
`$.books[?(!(@.id == 2))].title`	indefinite	[”Sayings of the Century”, ”Moby Dick”, ”The Lord of the Rings”]	
`$.books[?(@.id != 2)].title`	indefinite	[”Sayings of the Century”, ”Moby Dick”, ”The Lord of the Rings”]	
`$.books[?(@.title =~ " of ")].title`	indefinite	[”Sayings of the Century”, ”Sword of Honour”, ”The Lord of the Rings”]	

JSONPath	类型	结果	注释
`$.books[?(@.price > 12.99)].title`	indefinite	[”The Lord of the Rings”]	
`$.books[?(@.author > "Herman Melville")].title`	indefinite	[”Sayings of the Century”, ”The Lord of the Rings”]	
`$.books[?(@.price > $.filters.price)].title`	indefinite	[”Sword of Honour”, ”The Lord of the Rings”]	
`$.books[?(@.category == $.filters.category)].title`	indefinite	[”Sword of Honour”,”Moby Dick”,”The Lord of the Rings”]	
`$..[?(@.id)]`	indefinite	[{ ”category”: ”reference”, ”author”: ”Nigel Rees”, ”title”: ”Sayings of the Century”, ”price”: 8.95, ”id”: 1 }, { ”category”: ”fiction”, ”author”: ”Evelyn Waugh”, ”title”: ”Sword of Honour”, ”price”: 12.99, ”id”: 2 }, { ”category”: ”fiction”, ”author”: ”Herman Melville”, ”title”: ”Moby Dick”, ”isbn”: ”0-553-21311-3”, ”price”: 8.99, ”id”: 3 }, { ”category”: ”fiction”, ”author”: ”J. R. R. Tolkien”, ”title”: ”The Lord of the Rings”, ”isbn”: ”0-395-19395-8”, ”price”: 22.99, ”id”: 4 }]	
`$.services..[?(@.price > 50)].description`	indefinite	’[”Printing and assembling book in A5 format”, ”Rebinding torn book”]	
`$..id.length()`	definite	4	
`$.books[?(@.id == 2)].title.first()`	definite	Sword of Honour	
`$..tags.first().length()`	definite	5	$..tags 是不明确的路径，所以它返回一个匹配元素的数组 - [[”a”, ”b”, ”c”, ”d”, ”e”]], first() 返回第一个元素 - [”a”, ”b”, ”c”, ”d”, ”e”] 和最后 ength() 计算它的长度 - 5.

JSONPath	类型	结果	注释
`$.books[*].price.min()`	definite	8.95	
`$..price.max()`	definite	154.99	
`$.books[?(@.category == "fiction")].price.avg()`	definite	14.99	
`$.books[?(@.category == $.filters.xyz)].title`	indefinite		不匹配的查询对于明确和不明确的路径返回 NULL。
`$.services[?(@.active=="true")].servicegroup`	indefinite	[1000,1001]	在布尔值比较中必须使用文本常量。
`$.services[?(@.active=="false")].servicegroup`	indefinite	[1002]	在布尔值比较中必须使用文本常量。
`$.services[?(@.servicegroup==1000)]~.first()`	definite	恢复	

从 JSONPath 中的 LLD 宏值中转义特殊字符

当在 JSONPath 预处理中使用底层自动发现宏并解析其值时，将应用以下特殊字符转义规则：

- 只考虑转义反斜杠 (\) 和双引号 (”) 字符；
- 如果解析的宏值包含这些字符，则每个字符都用反斜杠转义；
- 如果已使用反斜杠转义，则不会将其视为转义，需要再次使用反斜杠进行转义。

例子:

JSONPath	LLD 宏值	替换后
$.[?(@.value == ”{#MACRO}”)]	special ”value”	$.[?(@.value == ”special \”value\””)]
	c:\temp	$.[?(@.value == ”c:\\temp”)]
	a\\b	$.[?(@.value == ”a\\\\b”)]

在表达式中使用时，可能有特殊字符的宏应该用双引号括起来:

JSONPath	LLD 宏值	替换后	结果
$.[?(@.value == ”{#MACRO}”)]	special ”value”	$.[?(@.value == ”special \”value\””)]	OK
$.[?(@.value == {#MACRO})]		$.[?(@.value == special \”value\”)]	**Bad JSONPath expression**

在路径中使用时，可能包含特殊字符的宏应括在方括号 和双引号中：

JSONPath	LLD 宏值	替换后	结果
$.[”{#MACRO}”].value	c:\temp	$.[”c:\\temp”].value	OK
$.{#MACRO}.value		$.c:\\temp.value	**Bad JSONPath expression**

4 JavaScript 预处理

概述

本节提供 JavaScript 预处理的详细信息。

JavaScript 预处理

JavaScript 预处理是通过调用具有单个参数 “值” 和用户提供的函数体的 JavaScript 函数来完成的。预处理步骤的结果是从这个函数返回的值，例如，要执行华氏到摄氏度的转换，用户必须输入：

```
return (value - 32) * 5 / 9
```

在 JavaScript 预处理参数中，将被服务器包装成一个 JavaScript 函数：

```
function (value)
{
   return (value - 32) * 5 / 9
}
```

输入参数 “值” 始终作为字符串传递。返回值通过 ToString() 方法自动强制转换为字符串（如果失败，则错误作为字符串值返回），但有一些例外：

- 返回未定义的值将导致错误
- 返回空值将导致输入值被丢弃，很像 “Custom on fail” 操作中的 “丢弃值” 预处理。

可以通过抛出值/对象（通常是字符串或错误对象）来返回错误。

例子:

```
if (value == 0)
    throw "Zero input value"
return 1/value
```

每个脚本都有 10 秒的执行超时（根据脚本的不同，触发超时可能需要更长的时间）；超过它会返回错误。强制执行 64 兆字节的堆限制。

JavaScript 预处理步骤字节码被缓存并在下次应用该步骤时重用。对监控项预处理步骤的任何更改都将导致缓存的脚本被重置并稍后重新编译。

连续运行时失败（连续 3 次）将导致引擎重新初始化，以减少一个脚本破坏下一个脚本的执行环境的可能性（此操作使用 DebugLevel 4 及更高级别记录）。

JavaScript 预处理是用 Duktape 实现的 (https://duktape.org/) JavaScript 引擎。

参考: 另外的 JavaScript 对象和全局函数

在脚本中使用宏

可以在 JavaScript 代码中使用用户宏。如果脚本包含用户宏，则这些宏在执行特定预处理步骤之前由服务器/代理解析。注意，在前端测试预处理步骤时，宏值不会被拉取，需要手动输入。

> **Note:**
> 将宏替换为其值时将忽略上下文。宏值按原样插入代码中，在将值放入 JavaScript 代码之前无法添加额外的转义。请注意，这可能在某些情况下会导致 JavaScript 错误。

在下面的示例中，如果接收到的值超过 {$THRESHOLD} 宏值，则将返回阈值（如果存在）：

```
var threshold = '{$THRESHOLD}';
return (!isNaN(threshold) && value > threshold) ? threshold : value;
```

其他 JavaScript 对象

概述

本节介绍 Zabbix 对 JavaScript 语言的支持，该语言使用 Duktape 和受支持的全局 JavaScript 函数实现。

内置对象

Zabbix

Zabbix 对象提供与 Zabbix 内部功能的交互。

方法	描述
`log(loglevel, message)`	使用 <loglevel> 日志级别将 <message> 写入 Zabbix 日志（参见配置文件 DebugLevel 参数）。

示例:

```
Zabbix.log(3, "this is a log entry written with 'Warning' log level")
```

你可以使用以下别名：

别名	原义
console.log(object)	Zabbix.log(4, JSON.stringify(object))
console.warn(object)	Zabbix.log(3, JSON.stringify(object))
console.error(object)	Zabbix.log(2, JSON.stringify(object))

方法	描述
`sleep(delay)`	将 JavaScript 执行延迟 `delay` 毫秒。

示例（延迟执行 15 秒）：

```
Zabbix.sleep(15000)
```

HttpRequest

该对象封装了 cURL 句柄，允许发送简单的 HTTP 请求。错误将作为异常抛出。

Attention:
HttpRequest 是自 Zabbix 5.4 以来此对象的新名称。以前它曾经被命名为 CurlHttpRequest。Zabbix 5.4 中的方法名称也已更改。旧的对象/方法名称现在已弃用，并且在 Zabbix 6.0 之后将停止对它们的支持。

方法	描述
`addHeader(name, value)`	添加 HTTP 标头字段。此字段用于所有后续请求，直到使用 clearHeader() 方法清除。
`clearHeader()`	清除 HTTP 标头。如果没有设置头字段，HttpRequest 会将 Content-Type 设置为 application/json 如果发布的数据是 JSON 格式的；否则文本/纯文本。
`connect(url)`	向 URL 发送 HTTP CONNECT 请求并返回响应。
`customRequest(method, url, data)`	允许在第一个参数中指定任何 HTTP 方法。
`delete(url, data)`	使用可选的 *data* 负载向 URL 发送 HTTP DELETE 请求并返回响应。
`getHeaders(<asArray>)`	返回接收到的 HTTP 标头字段的对象。 `asArray` 参数可以设置为 “true”（例如 `getHeaders(true)`）、“false” 或未定义。如果设置为 “true”，接收到的 HTTP 标头字段值将作为数组返回；这应该用于检索多个同名标头的字段值。 如果未设置或设置为 “false”，则接收到的 HTTP 标头字段值将作为字符串返回。
`get(url, data)`	发送 HTTP GET 请求到带有可选 *data* 负载的 URL 并返回响应。
`head(url)`	向 URL 发送 HTTP HEAD 请求并返回响应。
`options(url)`	向 URL 发送 HTTP OPTIONS 请求并返回响应。
`patch(url, data)`	将 HTTP PATCH 请求发送到带有可选 *data* 负载的 URL 并返回响应。
`put(url, data)`	将 HTTP PUT 请求发送到带有可选 *data* 负载的 URL 并返回响应。
`post(url, data)`	发送 HTTP POST 请求到带有可选 *data* 负载的 URL 并返回响应。
`getStatus()`	返回最后一个 HTTP 请求的状态码。
`setProxy(proxy)`	将 HTTP 代理设置为 “代理” 值。如果此参数为空，则不使用代理。

`setHttpAuth(bitmask, username, password)`	在’bitmask’ 参数中设置启用的 HTTP 身份验证方法 (HTTPAUTH_BASIC, HTTPAUTH_DIGEST, HTTPAUTH_NEGOTIATE, HTTPAUTH_NTLM, HTTPAUTH_NONE)
	HTTPAUTH _NONE 标志允许禁用 HTTP 身份验证。
	示例：
	`request.setHttpAuth(HTTPAUTH_NTLM | HTTPAUTH_BASIC, username, password)`
	`request.setHttpAuth(HTTPAUTH_NONE)`

`trace(url, data)`	发送 HTTP TRACE 请求到带有可选 *data* 负载的 URL 并返回响应。

示例:

```
try {
    Zabbix.log(4, 'jira webhook script value='+value);

    var result = {
        'tags': {
            'endpoint': 'jira'
        }
```

```
    },
    params = JSON.parse(value),
    req = new HttpRequest(),
    fields = {},
    resp;

    req.addHeader('Content-Type: application/json');
    req.addHeader('Authorization: Basic '+params.authentication);

    fields.summary = params.summary;
    fields.description = params.description;
    fields.project = {"key": params.project_key};
    fields.issuetype = {"id": params.issue_id};
    resp = req.post('https://tsupport.zabbix.lan/rest/api/2/issue/',
        JSON.stringify({"fields": fields})
    );

    if (req.getStatus() != 201) {
        throw 'Response code: '+req.getStatus();
    }

    resp = JSON.parse(resp);
    result.tags.issue_id = resp.id;
    result.tags.issue_key = resp.key;
} catch (error) {
    Zabbix.log(4, 'jira issue creation failed json : '+JSON.stringify({"fields": fields}));
    Zabbix.log(4, 'jira issue creation failed : '+error);

    result = {};
}

return JSON.stringify(result);
```

XML

XML 对象允许处理监控项中的 XML 数据以及底层自动发现预处理和 Webhook。

Attention:
要使用 XML 对象，必须使用支持 libxml2 编译的 Server 或 proxy。

方法	描述
XML.query(expression, data)	使用 XPath 检索节点内容。如果未找到节点，则返回 null。 **expression** - XPath 表达式； **data** - XML 数据作为字符串。
XML.toJson(data)	将 XML 格式的数据转换为 JSON。
XML.fromJson(object)	将 JSON 格式的数据转换为 XML。

示例:

输入:

```
<menu>
    <food type = "breakfast">
        <name>Chocolate</name>
        <price>$5.95</price>
        <description></description>
        <calories>650</calories>
    </food>
</menu>
```

输出:

```
{
    "menu": {
        "food": {
            "@type": "breakfast",
            "name": "Chocolate",
            "price": "$5.95",
            "description": null,
            "calories": "650"
        }
    }
}
```

序列化规则

XML 到 JSON 的转换将根据以下规则进行处理（对于 JSON 到 XML 的转换，应用相反的规则）：

1. XML 属性将被转换为名称前面带有 “@” 的键。

示例:

输入:

```
 <xml foo="FOO">
   <bar>
     <baz>BAZ</baz>
   </bar>
 </xml>
```

输出:

```
 {
   "xml": {
     "@foo": "FOO",
     "bar": {
       "baz": "BAZ"
     }
   }
 }
```

2. 自闭合元素 (<foo/>) 将被转换为具有’null’ 值。

示例:

输入:

```
<xml>
  <foo/>
</xml>
```

输出:

```
{
  "xml": {
    "foo": null
  }
}
```

3. 空属性（具有”” 值）将被转换为具有空字符串 (”) 值。

示例:

输入:

```
<xml>
  <foo bar="" />
</xml>
```

输出:

```
{
  "xml": {
    "foo": {
```

```
      "@bar": ""
    }
  }
}
```

4. 具有相同元素名称的多个子节点将被转换为具有值数组作为其值的单个键。

示例:

输入:

```
<xml>
  <foo>BAR</foo>
  <foo>BAZ</foo>
  <foo>QUX</foo>
</xml>
```

输出:

```
{
  "xml": {
    "foo": ["BAR", "BAZ", "QUX"]
  }
}
```

5. 如果文本元素没有属性也没有子元素，它将被转换为字符串。

示例:

输入:

```
<xml>
    <foo>BAZ</foo>
</xml>
```

输出:

```
{
  "xml": {
    "foo": "BAZ"
   }
}
```

6. 如果一个文本元素没有子元素，但有属性：文本内容将被转换为一个元素，键为'#text'，内容为值；属性将按照序列化规则 1 中的说明进行转换。

示例:

输入:

```
<xml>
  <foo bar="BAR">
    BAZ
  </foo>
</xml>
```

输出:

```
{
  "xml": {
    "foo": {
      "@bar": "BAR",
      "#text": "BAZ"
    }
  }
}
```

全局 JavaScript 函数

使用 Duktape 实现了额外的全局 JavaScript 函数：

- btoa(string) - 将字符串编码为 base64 字符串

- atob(base64_string) - 对 base64 字符串进行解码

```
try {
    b64 = btoa("utf8 string");
    utf8 = atob(b64);
}
catch (error) {
    return {'error.name' : error.name, 'error.message' : error.message}
}
```

- md5(string) - 计算字符串的 MD5 哈希
- sha256(string) - 计算字符串的 SHA256 哈希
- hmac('<hash type>',key,string) - 返回 HMAC 哈希作为十六进制格式的字符串。支持 MD5 和 SHA256 哈希类型。键和字符串参数支持二进制数据。例如：
- `hmac('md5',key,string)`
- `hmac('sha256',key,string)`

5 CSV 转换成 JSON 预处理

概述

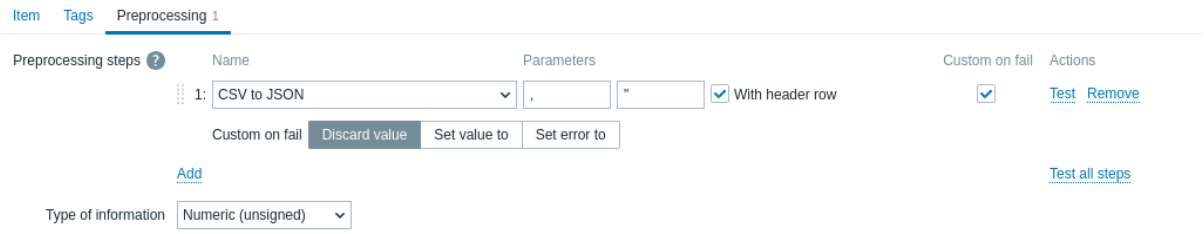
在预处理的步骤中，可以将 CSV 文件数据转换为 JSON 格式，支持项目如下：

- 监控项 (监控项原型)
- 低级别自动发现

配置

配置 CSV 到 JSON 的预处理步骤:

- 转到预处理选项监控项/自动发现规则 配置
- 点击 添加
- 选择 *CSV to JSON* 选项



第一个参数允许设置自定义分隔符。请注意，如果 CSV 输入的第一行以 “Sep=” 开头，后跟一个 UTF-8 字符，则在未设置第一个参数的情况下，该字符将用作分隔符。如果未设置第一个参数，且未从 “Sep=” 行检索到分隔符，则使用逗号作为分隔符。

第二个可选参数允许设置引号符号。

如果勾选 *With header row* ，标题行值将被解释为列名 (详见标题预处理 了解更多信息)。

如果勾选 *Custom on fail* ，那么在预处理步骤失败的情况下，该监控项项将不会不受支持。另外，可以设置自定义错误处理选项: 丢弃该值，设置指定值或设置指定的错误消息。

标题预处理

CSV 文件标题行可以用两种不同的方式处理:

- 如果勾选 *With header row* - 标题行值被解释为列名。在这种情况下，列名必须是唯一的，数据行不应该包含比标题行更多的列；
- 如果勾选 *With header row* - 标题行解释为数据。自动生成列名 (1,2,3,4...)

CSV 文件示例:

```
Nr,Item name,Key,Qty
1,active agent item,agent.hostname,33
"2","passive agent item","agent.version","44"
3,"active,passive agent items",agent.ping,55
```

Note:
输入中的引号字段中的引号字符必须在其前面加上另一个引号字符进行转义。

有标题行预处理

有标题行预处理时期望 JSON 的输出:

```
[
   {
      "Nr":"1",
      "Item name":"active agent item",
      "Key":"agent.hostname",
      "Qty":"33"
   },
   {
      "Nr":"2",
      "Item name":"passive agent item",
      "Key":"agent.version",
      "Qty":"44"
   },
   {
      "Nr":"3",
      "Item name":"active,passive agent items",
      "Key":"agent.ping",
      "Qty":"55"
   }
]
```

无标题行预处理

无标题行预处理时期望 JSON 的输出:

```
[
   {
      "1":"Nr",
      "2":"Item name",
      "3":"Key"
      "4":"Qty"
   },
   {
      "1":"1",
      "2":"active agent item",
      "3":"agent.hostname"
      "4":"33"
   },
   {
      "1":"2",
      "2":"passive agent item",
      "3":"agent.version"
      "4":"44"
   },
   {
      "1":"3",
      "2":"active,passive agent items",
      "3":"agent.ping"
      "4":"55"
   }
]
```

3 监控项类型

概述

监控项类型包含从系统获取数据的多种方式。每个监控项类型都有一组自己支持的监控项 key 和所需的参数。

以下监控项类型由 Zabbix 提供：

- Zabbix agent 检查
- SNMP agent 检查
- SNMP traps
- IPMI 检查
- 简单检查
  - VMware 监控
- 日志文件监控
- 可计算监控项
  - 聚合计算
- Zabbix 内部检查
- SSH 检查
- Telnet 检查
- 外部检查
- 采集器监控项
- JMX 监控
- ODBC 检查
- 相关监控项
- HTTP 检查
- Prometheus 检查
- 脚本监控

所有监控项类型的详细描述都包含在本章的各个小节中。即使监控项类型提供了大量的数据收集的方式，你还可以通过用户参数 or 可加载模块进一步扩展数据收集方式。

一些监控检查由 Zabbix Server 执行（称作无代理监控），而其它监控检查则需要 Zabbix agent 或者 Zabbix Java 网关（使用 JMX 监视）执行。

Attention:
如果特定的监控项类型需要特定的接口（如 IPMI 检查需要主机上的 IPMI 接口），该接口必须存在于主机定义中。

可以在主机定义中设置多个接口：Zabbix agent，SNMP agent，JMX 和 IPMI。如果一个监控项使用多个接口，它将搜索可用的主机接口（按照以下顺序：Agent→SNMP→JMX→IPMI）直到找到连接的第一个匹配的接口。

返回文本的所有监控项（字符，日志，文本信息类型）都可以返回空格（如适用）和值设置为空的字符串。（2.0 版本后支持）

1 Zabbix agent

概述

这些检查与 Zabbix agent 进行通信，实现数据的采集

有被动和主动两种 agent 检查模式，在配置监控项时，你可以选择所需的类型:

- *Zabbix agent* - 被动检查
- *Zabbix agent(*主动*)* - 主动检查

支持的监控项

下表提供了可用的 Zabbix agent 监控项键值的详细信息。

另请参阅

- 各平台支持的监控项
- Zabbix agent 2 支持的监控键值
- Windows 客户端指定的监控键值
- 最低权限级别 Windows 客户端监控项

强制和可选参数

没有尖括号的参数是强制性的。用尖括号 **<** **>** 标记的参数是可选的。

与命令行实用程序一起使用

请注意，在命令行中使用 zabbix_agentd 或 zabbix_get 测试或使用项目键时，您也应该考虑 shell 语法。

例如，如果键的某个参数必须用双引号括起来，则必须显式转义双引号，否则它们将被 shell 修剪为特殊字符，并且不会传递给 Zabbix 实用程序。

例如:

```
$ zabbix_agentd -t 'vfs.dir.count[/var/log,,,"file,dir",,0]'

$ zabbix_agentd -t vfs.dir.count[/var/log,,,\"file,dir\",,0]
```

内核数据

监控项键值			
描述	返回值	参数	注释
kernel.maxfiles			
OS 支持的最大打开文件数。	整型		
kernel.maxproc			
OS 支持的最大进程数	整型		
kernel.openfiles			
返回当前打开文件句柄数	整型		自 Zabbix 6.0 开始支持该参数

日志数据

更多信息参考日志监控.

键值			
描述	返回值	参数	注释
log[file,<regexp>,<encoding>,<maxlines>,<mode>,<output>,<maxdelay>,<options>,<persistent_dir>]			

键值			
监控日志文件。	日志	**file** - 日志文件的全路径 **regexp** - 正则表达式 所需要的匹配的模式 **encoding** - 编码标识符 **maxlines** - 客户端将发送到 Zabbix server 或 proxy 的每秒最大行数。此参数覆盖zabbix_agentd.conf中的'MaxLinesPerSecond' 值 **mode** (自版本 2.0 以后)- 可能的值有: *all* (默认), *skip* - 跳过旧数据的处理 (仅影响新创建的监控项)。 **output** (自版本 2.2 以后) - 一个可选的输出格式模板，**\0** 转义序列被替换为文本的匹配部分 (匹配从开始的第一个字符到结束的字符) 而 **\N** (其中 N=1...9) 转义序列被替换为第 N 个匹配组 (或如果 N 超过捕获的组数，则为空字符串)。 **maxdelay** (自版本 3.2 后) - 最大延迟，以秒为单位。类型: 浮点型，值: 0 - (默认) 从不忽略日志文件行; > 0.0 - 忽略旧行，以便在 "maxdelay" 秒内分析最新行。使用前请阅读maxdelay 说明！ **options** (自版本 4.4.7 以后) - 附件选项: *mtime-noreread* - 非唯一记录, 仅在文件大小更改时重读 (忽略修改时间更改). (从 5.0.2 开始不推荐使用此参数，因为现在忽略了 mtime。) **persistent_dir** (从 5.0.18、5.4.9 版本开始，仅在 Unix 系统上的 zabbix_agentd 中；在 Agent2 中不支持) - 存储持久性的目录的绝对路径名文件。另请参阅有关持久文件附加说明	该监控项必须配置为主动检查。 如果文件丢失或权限不允许访问，监控项变为不受支持。 如果 `output` 留空 - 返回包含匹配文本的整行。请注意，除'Result is TRUE' 之外的所有全局正则表达式类型始终返回整个匹配行，并且 `output` 忽略该参数。 在 agent 上使用 `output` 参数进行内容提取。 例如: => log[/var/log/syslog] => log[/var/log/syslog,error] => log[/home/zabbix/logs/logfile„,100] 使用 *`output`* 参数中从日志记录中提取一个数字*:* => log[/app1/app.log,"task run [0-9.]+ sec, processed ([0-9]+) records, [0-9]+ errors"„„\1] → 将匹配日志记录"2015-11-13 10:08:26 task run 6.08 sec, processed 6080 records, 0 errors" 并只发送'6080' 到服务器。因为发送的是一个数值，所以该监控项的" 信息类型" 可以设置为" 数字 (无符号)" 该值可用于图形，触发器等。 使用 *`output`* 参数在发送到服务器之前重写日志记录*:* => log[/app1/app.log,"([0-9:-]+) task run ([0-9.]+) sec, processed ([0-9]+) records, ([0-9]+) errors"„„"\1 RECORDS: \3, ERRORS: \4, DURATION: \2"] → 将匹配日志记录"2015-11-13 10:08:26 task run 6.08 sec, processed 6080 records, 0 errors" 并且向服务端发送修改后的记录"2015-11-13 10:08:26 RECORDS: 6080, ERRORS: 0, DURATION: 6.08" 。 另请参阅日志监控其他信息。

log.count[file,<regexp>,<encoding>,<maxproclines>,<mode>,<maxdelay>,<options>,<persistent_dir>]

键值			
监控日志文件中匹配的行数量	整型	**file** - 含全路径的日志文件名称 **regexp** - 正则表达式匹配的内容模式 **encoding** - 编码标识符 **maxproclines** - agent 每秒分析新增的最大行数 (不能超过 10000)。在zabbix_agentd.conf中设置的默认值是 10*'MaxLinesPerSecond' **mode** - 可能的值: *all* (默认), *skip* - 跳过旧数据的处理 (仅影响新创建的监控项)。 **maxdelay** - 最大延迟秒数。类型: 浮点型。值: 0 - (默认) 从不忽略日志行数; > 0.0 - 忽略旧行以便在 "maxdelay" 秒内分析最新行。使用前请阅读maxdelay 说明! **options** (自 4.4.7 版后) - 附加选项: *mtime-noreread* - 非唯一记录, 仅在文件大小发生更改时重新读取 (忽略修改时间更改). (从 5.0.2 开始不推荐使用此参数, 因为现在忽略了 mtime。) **persistent_dir** (从 5.0.18、5.4.9 版本开始，仅在 Unix 系统上的 zabbix_agentd 中；在 Agent2 中不支持) - 存储持久性的目录的绝对路径名文件。另请参阅有关持久文件附加说明	该监控项必须配置为主动检查. 如果文件缺失或权限不允许访问，则监控项不受支持。 另请参阅有关日志监控的其他信息 该监控项不支持 Windows 的事件日志。 自 Zabbix 3.2.0 版本后开始支持。

logrt[file_regexp,<regexp>,<encoding>,<maxlines>,<mode>,<output>,<maxdelay>,<options>,<persistent_dir>]

键值			
监控轮换的日志文件。	日志	**file_regexp** - 文件的绝对路径和正则表达式匹配的文件名模式 请注意只有文件名是正则表达式 **regexp** - 描述正则表达式匹配的文件内容模式 **encoding** - 编码标识符 **maxlines** - agent 每秒将发送到 Zabbix 服务器或代理的最大新行数。此参数覆盖zabbix_agentd.conf的'MaxLinesPerSecond'的值 **mode** (自版本 2.0 后) - 可能值: *all* (默认), *skip* - 跳过旧数据的处理 (仅影响新创建的监控项)。 **output** (自版本 2.2 后) - 一个可选的输出格式模板。**\0** 转义序列被替换文本的匹配部分 (匹配从开始的第一个字符到结束的字符) 而 **\N** (其中 N=1...9) 转义序列被替换为第 N 个匹配组 (或如果 N 超过捕获的组数，则为空字符串)。 **maxdelay** (自版本 3.2 后) -最大的时延，以秒为单位。类型: 浮点型。值: 0 - (默认) 从不忽略日志文件行数; > 0.0 - 忽略旧行，以便在 "maxdelay" 秒内分析最新行。使用前请阅读maxdelay 说明! **options** (版本 4.0 和版本 4.4.7 后; *mtime-reread*, 和 *mtime-noreread* 选项) - 日志文件轮换类型和其他选项。可能的值: *rotate* (默认), *copytruncate* - 注意的是 *copytruncate* 不能和 *maxdelay* 一起使用，在这种情况下 *maxdelay* 必须为 0 或未指定; 参阅copytruncate 文档。 *mtime-reread* - 非唯一记录，如果修改时间或大小更改则重新读取 (默认), *mtime-noreread* -非唯一记录，仅在大小更改时重新读取 (忽略修改时间更改)。 **persistent_dir** (从 5.0.18, 5.4.9 版本开始, 仅在 Unix 系统 zabbix_agentd 中; 不支持 Agent2) - 存储持久文件的目录的绝对路径名。另请参阅有关持久文件的附加说明。	该监控项必须配置为主动检查. 日志轮换基于文件的最后修改时间。 请注意，logrt 旨在处理一个当前活动的日志文件，并轮换几个其他匹配的非活动文件。例如，如果一个目录有许多活动日志文件，则应为每个文件创建一个单独的 logrt 项。否则，如果一个 logrt 项获取太多文件，可能会导致内存耗尽和监控崩溃。 如果 `output` 留空 - 返回包含匹配文本的整行。请注意，除'Result is TRUE' 之外的所有全局正则表达式类型始终返回整个匹配行，并且 `output` 忽略该参数. 在代理上使用 `output` 参数进行内容提取。 例如: => logrt["/home/zabbix/logs/^logfile[0-9]{1,3}$",,,100] → 将匹配文件名如"logfile1"(不会匹配".logfile1") => logrt["/home/user/^logfile_.*_[0-9]{1,3}$","pattern_to_match","UTF-8",100] → 将从文件名如"logfile_abc_1" 或"logfile__001" 中采集数据 使用 *`output`* 参数从日志记录中提取数字*:* => logrt[/app1/^test.*log$,"task run [0-9.]+ sec, processed ([0-9]+) records, [0-9]+ errors",,,\1] → 将匹配一条日志记录"2015-11-13 10:08:26 task run 6.08 sec, processed 6080 records, 0 errors" 并发送'6080' 到服务端。因为发送的数值，所以此监控项" 信息类型" 可以设置为" 数字 (无符号)" 这个值可以用于图形，触发器等。 使用 *`output`* 参数可以在发送到服务端前重写日志记录*:* => logrt[/app1/^test.*log$,"([0-9 :-]+) task run ([0-9.]+) sec, processed ([0-9]+) records, ([0-9]+) errors",,,"\1 RECORDS: \3, ERRORS: \4, DURATION: \2"] → 将匹配一条日志记录"2015-11-13 10:08:26 task run 6.08 sec,

键值				
logrt.count[file_regexp,<regexp>,<encoding>,<maxproclines>,<mode>,<maxdelay>,<options>,<persistent_dir>] 轮换的监控日志文件中的匹配行数	整型	**file_regexp** - 文件绝对路径和正则表达式 所需匹配的文件名模式 **regexp** - 正则表达式 所需的内容匹配模式 **encoding** - 编码标识符 **maxproclines** - agent 每秒分析新增的最大行数 (不能超过 10000)。在zabbix_agentd.conf中设置的默认值是 10*'MaxLinesPerSecond'。 **mode** - 可能值: *all* (默认), *skip* - 跳过旧数据的处理 (仅影响新创建的监控项). **maxdelay** - 最大时延，以秒为单位。类型: 浮点型。值: 0 - (默认) 从不忽略日志文件行; > 0.0 - 以便在 "maxdelay" 秒内分析最新行。使用前请阅读maxdelay说明! **options** (自版本 4.0 和 4.4.7 后; *mtime-reread* 和 *mtime-noreread* 选项) - 日志文件轮换类型和其他选项，可能的值: *rotate* (默认), *copytruncate* - 注意 *copytruncate* 不能与 *maxdelay* 一起使用，这种情况下 *maxdelay* 必须为 0 或未指定; 参阅copytruncate 说明, *mtime-reread* - 非唯一记录, 如果修改时间或大小改变重新读 (默认), *mtime-noreread* - 非唯一记录, 仅当大小改变时重新读 (忽略修改时间的改变)。 **persistent_dir** (自版本 5.0.18 和 5.4.9 后, 仅在 Unix 系统上的 zabbix_agentd 中; 不支持 Agent2) - 存储持久文件的目录的绝对路径名。另请参阅有关持久文件附加说明。	该监控项必须配置为主动检查. 日志轮换基于文件的最后修改时间。 另请参阅有关日志监控其他信息 此监控项不支持 Windows 事件日志 自 Zabbix 3.2.0 后开始支持

Modbus 数据

键值			
描述	返回值	参数	注释
modbus.get[endpoint,<slave id>,<function>,<address>,<count>,<type>,<endianness>,<offset>]			

键值			
读取 Modbus 数据	JSON 对象	**endpoint** - 终端定义为 `protocol://connection_string` **slave id** - 从机 ID **function** - Modbus 函数 **address** - 第一个注册地址, 线圈或输入 **count** - 读取的记录数量 **type** - 数据类型 **endianness** - 字节序配置 **offset** - 寄存器数, 从' 地址' 开始, 其结果将被丢弃。 参阅参数的详细描述 。	自 Zabbix 5.2.0 后开始支持

网络数据

键值			
描述	返回值	参数	注释
net.dns[<ip>,name,<type>,<timeout>,<count>,<protocol>]			
检查 DNS 服务是否启动	0 - DNS 已关闭 (服务器并没有响应或 DNS 解析失败) 1 - DNS 已启动	**ip** - DNS 服务器 IP 地址 (默认 DNS 服务器上的 IP 留空, 在 Windows 上忽略) **name** -要查询的 DNS 名称 **type** - 要查询的记录类型 (默认为 *SOA*) **timeout** (在 Windows 上忽略) -请求超时秒数 (默认是 1 秒) **count** (在在 Windows 上忽略) -请求的尝试次数 (默认是 2) **protocol** (自版本 3.0 后) - 用于执行 DNS 查询的协议: *udp* (默认) 或 *tcp*	例如: => net.dns[8.8.8.8,example.com,MX,2,1] 可能的值 type 是: *ANY*, *A*, *NS*, *CNAME*, *MB*, *MG*, *MR*, *PTR*, *MD*, *MF*, *MX*, *SOA*, *NULL*, *WKS* (Windows 除外), *HINFO*, *MINFO*, *TXT*, *SRV* 不支持国际化域名，请改用 IDNA 编码名称。 从 Zabbix 1.8.6 (Unix) 和 2.0.0 (Windows) 开始支持 SRV 记录类型. Zabbix 2.0 之前的命名（仍然支持）：*net.tcp.dns*
net.dns.record[<ip>,name,<type>,<timeout>,<count>,<protocol>]			
执行 DNS 查询。	具有所需信息类型的字符串	**ip** - DNS 服务器的 IP 地址 (默认 DNS 服务器留空，在 Windows 上忽略) **name** - 要查询的 DNS 名称 **type** - 要查询的记录类型 (默认为 *SOA*) **timeout** (在 Windows 上忽略) - 请求超时秒数 (默认是 1 秒) **count** (在 Windows 上忽略) - 请求的尝试次数 (默认是 2) **protocol**(自 3.0 版起) - 用于执行 DNS 查询的协议: *udp* (默认) 或 *tcp*	例如: => net.dns.record[8.8.8.8,example.com,MX,2, 可能的值 type 是: *ANY*, *A*, *NS*, *CNAME*, *MB*, *MG*, *MR*, *PTR*, *MD*, *MF*, *MX*, *SOA*, *NULL*, *WKS* (Windows 除外), *HINFO*, *MINFO*, *TXT*, *SRV* 不支持国际化域名，请改用 IDNA 编码名称。 从 Zabbix 1.8.6 (Unix) 和 2.0.0 (Windows) 开始支持 SRV 记录类型。 Zabbix 2.0 之前的命名（仍然支持）：*net.tcp.dns.query*
net.if.collisions[if]			
Number of out-of-window collisions.	整型	**if** - 网络接口名称	

键值			
net.if.discovery 网络接口列表. 用于低级别的发现。	JSON 对象		自 Zabbix 2.0 后支持 在 FreeBSD, OpenBSD 和 NetBSD 上从 Zabbix 2.2 后开始支持 某些 Windows 版本 (如 Server 2008) 可能需要安装最新更新以支持接口名称中的非 ASCII 字符。
net.if.in[if,<mode>] 网络接口的传入流量统计	整型	**if** - 网络接口名称 (Unix); 网络接口完整描述或 IPv4 地址；或者，如果在大括号内，网络接口 GUID（窗口） **mode** - 可能的值: *bytes* - 字节数 (默认) *packets* - 包数 *errors* - 错误数 *dropped* - 丢弃的数据包数量 *overruns (fifo)* - FIFO 缓冲错误数 *frame* - 数据包帧错误数 *compressed* - 设备驱动程序发送或接收的压缩数据包数 *multicast* - 设备驱动程序接收的多播帧数	在 Windows 上, 该监控项从 64 位计数器中获取值（如果可用），Windows Vista 和 Windows Server 2008 中引入了 64 位接口统计计数器。如果 64 位计数器不可用，agent 将使用 32 位计数器。 从 Zabbix 1.8.6 开始支持 Windows 上的多字节接口名称 例如: => net.if.in[eth0,errors] => net.if.in[eth0] 您可以使用 net.if.discovery 或 net.if.list 项获取 Windows 上的网络接口描述。 您可以将此键与预处理每秒更改步骤一起使用，以获取每秒字节数统计信息。
net.if.out[if,<mode>] 网络接口的传出流量统计	整型	**if** - 网络接口名称 (Unix); 网络接口完整描述或 IPv4 地址；或者，如果在大括号内，网络接口 GUID（窗口） **mode** - 可能的值: *bytes* - 字节数 (默认) *packets* -包数 *errors* - 错误数 *dropped* - 丢弃的包的数量 *overruns (fifo)* - FIFO 缓冲错误数 *collisions (colls)* - 接口上检测到的冲突数 *carrier* - 在设备驱动程序检测到的载体丢失数 *compressed* -设备驱动程序传输的压缩数据包数	在 Windows 上，该监控项从 64 位计数器中获取值（如果可用）。Windows Vista 和 Windows Server 2008 中引入了 64 位接口统计计数器。如果 64 位计数器不可用，agent 将使用 32 位计数器。 从 Zabbix agent 1.8.6 版本开始支持 Windows 上的多字节接口名称。 例如: => net.if.out[eth0,errors] => net.if.out[eth0] 您可以使用 net.if.discovery 或 net.if.list 项获取 Windows 上的网络接口描述。 您可以将此键与预处理每秒更改步骤一起使用，以获取每秒字节数统计信息。
net.if.total[if,<mode>]			

键值			
网络接口上传入和传出流量统计的总和。	整型	**if** - 网络接口名称 (Unix); 网络接口完整描述或 IPv4 地址；或者，如果在大括号内，网络接口 GUID（窗口） **mode** -可能值: *bytes* - 字节数 (默认) *packets* - 包数 *errors* - 错误数 *dropped* - 丢弃的数据包数量 *overruns (fifo)* - FIFO 缓冲错误数 *compressed* - 设备驱动程序发送或接收的压缩数据包数	在 Windows 上，该监控项从 64 位计数器中获取值（如果可用）。Windows Vista 和 Windows Server 2008 中引入了 64 位接口统计计数器。如果 64 位计数器不可用，agent 将使用 32 位计数器。 例如: => net.if.total[eth0,errors] => net.if.total[eth0] Y 您可以使用 net.if.discovery 或 net.if.list 项获取 Windows 上的网络接口描述。 可以将此键与预处理每秒更改步骤一起使用，以获取每秒字节数统计信息。 请注意，仅当 net.if.in 和 net.if.out 都在您平台上的丢弃数据包时，才支持丢弃数据包。
net.tcp.listen[port] 检查 TCP 端口是否处于监听状态	0 - 它不处于监听状态 1 - 它处于监听状态	**port** - TCP 端口号	例如: => net.tcp.listen[80] 自 Zabbix 1.8.4 后在 Linux 上支持 自 Zabbix 3.0.0 后, 在 Linux 内核 2.6.14 及以上, 如果可能，从内核的 NETLINK 接口获取有关侦听 TCP 套接字的信息. 否则，将从 /proc/net/tcp 和 /proc/net/tcp6 文件中检索信息。
net.tcp.port[<ip>,port] 检查是否可以与指定端口建立 TCP 连接。	0 -不能连接 1 - 能连接	**ip** - IP 或 DNS 名称 (默认是 127.0.0.1) **port** - 端口号	例如: => net.tcp.port[,80] → 可用于测试在端口 80 上运行 Web 服务的可用性 对于简单的 TCP 性能测试，使用 net.tcp.service.perf[tcp,<ip>,<port>] 请注意，这些检查可能会导致增加系统守护程序日志文件中的额外信息（通常会记录 SMTP 和 SSH 会话）。
net.tcp.service[service,<ip>,<port>]			

键值			
检查服务是否正在运行并接受 TCP 连接。	0 - 服务关闭 1 - 服务正在运行	**service** - 任意一个: *ssh*, *ldap*, *smtp*, *ftp*, *http*, *pop*, *nntp*, *imap*, *tcp*, *https*, *telnet* (另见详情) **ip** - IP 地址 (默认是 127.0.0.1) **port** - 端口号 (默认为标准服务使用端口号)	例如: => net.tcp.service[ftp„45] → 用于测试在 tcp 端口 45 上的 FTP 服务的可用性. 请注意，这些检查可能会导致系统守护程序日志文件中出现额外消息（SMTP 和 SSH 会话正在通常记录）。 目前不支持检查加密协议（如端口 993 上的 IMAP 或端口 995 上的 POP）。作为一种解决方法，请使用 net.tcp.port 进行此类检查。 仅 Zabbix agent 2 支持在 Windows 上检查 LDAP 和 HTTPS。 请注意，telnet 检查会查找登录提示（末尾为 “：”）。 请你参阅已知问题用于检查 HTTPS 服务。 从 Zabbix2.0 后开始支持 *https* 和 *telnet* 服务。
net.tcp.service.perf[service,<ip>,<port>]			
检查 TCP 服务的性能。	0 - 服务已关闭 seconds - 连接到服务花费的时间（秒）	**service** -以下任意一个: *ssh*, *ldap*, *smtp*, *ftp*, *http*, *pop*, *nntp*, *imap*, *tcp*, *https*, *telnet* (见详细) **ip** - IP 地址 (默认是 127.0.0.1) **port** - 端口号 (默认为标准服务使用端口号)	例如: => net.tcp.service.perf[ssh] → 可用于测试来自 SSH 服务器的初始响应速度。 目前不支持检查加密协议（如端口 993 上的 IMAP 或端口 995 上的 POP）。作为一种解决方法，请使用 net.tcp.service.perf[tcp,<ip>,<port>] 进行此类检查。 仅 Zabbix 代理 2 支持在 Windows 上检查 LDAP 和 HTTPS。 请注意，telnet 检查会查找登录提示（末尾为 “：”）。 请你参阅已知问题用于检查 HTTPS 服务。 从 Zabbix2.0 后开始支持 *https* 和 *telnet* 服务。
net.tcp.socket.count[<laddr>,<lport>,<raddr>,<rport>,<state>]			

键值			
返回与参数匹配的套接字数	整型	**laddr** - 本地 IPv4/6 地址或 CIDR 子网 **lport** - 本地的端口号或服务名称 **raddr** - 远程 IPv4/6 地址或 CIDR 子网 **rport** - 远程端口号或服务名称 **state** - 连接状态 (*established*, *syn_sent*, *syn_recv*, *fin_wait1*, *fin_wait2*, *time_wait*, *close*, *close_wait*, *last_ack*, *listen*, *closing*)	Zabbix agent 和 Zabbix agent 2 均在 Linux 上支持此项，64 位 Windows 在 Zabbix agent 2 上支持此项 例如: => net.tcp.socket.count[,80,,,established] → 检查本地 TCP 端口 80 是否处于 “已建立” 状态 从 Zabbix 6.0 后开始支持此项
net.udp.listen[port]			
检查 UDP 端口是否处于监听状态	0 - 它不处于监听状态 1 -它处于监听状态	**port** - UDP 端口号	例如: => net.udp.listen[68] 在 Linux 上自 Zabbix 1.8.4 后支持
net.udp.service[service,<ip>,<port>]			
检查服务是否正在运行并响应 UDP 请求。	0 - 服务已关闭 1 - 服务正在运行	**service** - *ntp* (见详情) **ip** -IP 地址 (默认是 127.0.0.1) **port** - 端口号 (默认为标准服务使用端口号)	例: => net.udp.service[ntp,,45] → 可用于测试 UDP 45 端口 NTP 服务的可用性。 Zabbix 3.0.0 后开始支持此项, 但 *ntp* 服务可用于以前版本的 net.tcp.service[] 监控项中
net.udp.service.perf[service,<ip>,<port>]			
检查 UDP 服务性能。	0 - 服务已关闭 seconds - 等待服务响应秒数	**service** - *ntp* (见详情) **ip** - IP 地址 (默认是 127.0.0.1) **port** - 端口号 (默认为标准服务使用端口号)	例: => net.udp.service.perf[ntp] → 可用于测试来自 NTP 服务的响应时间。 自 Zabbix 3.0.0 后开始支持该项，但 *ntp* 服务可用于以前版本的 net.tcp.service[] 监控项中
net.udp.socket.count[<laddr>,<lport>,<raddr>,<rport>,<state>]			
返回与参数匹配的 TCP 套接字数。	整型	**laddr** -本地 IPv4/6 地址或 CIDR 子网地址 **lport** -本地端口号或服务名称 **raddr** - 远程 IPv4/6 地址或 CIDR 子网 **rport** - 远程端口号或服务名称 **state** - 连接状态 (*established*, *unconn*)	Zabbix agent 和 Zabbix agent 2 均在 Linux 上支持此项，64 位 Windows 在 Zabbix agent 2 上支持此项。 Example: => net.udp.socket.count[,,,listening] → 检查是否有任何 UDP 套接字处于 “监听” 状态 从 Zabbix 6.0 开始支持此项

进程数据

键值			
描述	返回值	参数	注释
proc.cpu.util[<name>,<user>,<type>,<cmdline>,<mode>,<zone>]			

键值			
进程 CPU 使用百分比.	浮点型	**name** -进程名 (默认是 *all processes*) **user** - 用户名称 (默认是 *all users*) **type** -CPU 使用率类型: *total* (默认), *user*, *system* **cmdline** - 可按命令行过滤 (它是一个正则表达式) **mode** - 数据搜集模式: *avg1* (默认), *avg5*, *avg15* **zone** - 目标区域: *current* (默认), *all*. 这个参数仅在 Solaris 平台上支持	例: => proc.cpu.util[,root] → 在 root 用户下运行的所有进程的 CPU 使用率 => proc.cpu.util[zabbix_server,zabbix] → 在 zabbix 用户下运行的所有 zabbix_server 进程的 CPU 使用率 T 返回值基于单个 CPU 核心利用率百分比。例如，完全使用两个内核的进程的 CPU 利用率为 200%。 进程 CPU 利用率数据由支持最多 1024 个唯一（按名称、用户和命令行）查询的收集器收集。在过去 24 小时内未访问的查询将从收集器中删除。 请注意将 zone 参数设置为 *current* （或默认值）如果 agent 已在不支持区域的 Solaris 上编译，但在支持区域的较新 Solaris 上运行，则 agent 将返回 NOTSUPPORTED（agent 不能将结果限制为仅当前区域）。但是，在这种情况下，*all* 都支持。 自 Zabbix 3.0.0 起支持此键，并且可在多个平台上使用 (请参阅平台支持的监控项)。
proc.mem[<name>,<user>,<mode>,<cmdline>,<memtype>]			

键值			
进程使用的内存（以字节为单位）	整型 - 当 mode 为 *max*, *min*, *sum* 时 浮点数 - 当 mode 为 *avg* 时	**name** -进程名 (默认是 *all processes*) **user** - 用户名 (默认是 *all users*) **mode** - 可能的值: *avg*, *max*, *min*, *sum* (默认) **cmdline** - 按命令行过滤 (它是一个正则表达式) **memtype** - 进程使用的内存类型	例如: => proc.mem[,root] → 运行在 root 用户下的所有进程使用的内存 => proc.mem[zabbix_server,zabbix] → 运行在 zabbix 用户下的所有 zabbix_server 进程使用的内存 => proc.mem[,oracle,max,oracleZABBIX] → 在 oracle 用户下，包含有 oracleZABBIX 命令行运行的所有内存最多的内存 需要注意: 当多个进程使用共享内存时，进程使用的内存总和可能会导致大到不切实际。 参考说明 关于选择进程 name 和 cmdline 参数 (指定为 Linux). 当此监控项从命令行调用并包含命令行参数时 (例如使用 agent 测试模式: zabbix_agentd -t proc.mem[,,,apache2]), 一个额外的进程将被计算在内，因为 agent 将计算自己。 从 Zabbix3.0.0 起，memtype 参数开始支持多个平台
proc.num[<name>,<user>,<state>,<cmdline>,<zone>]			

键值			
进程数量	整型	**name** - 进程名称 (默认为 *all processes*) **user** - 用户名 (默认为 *all users*) **state** (*disk* 和 *trace* 选项从 3.4.0 起) - 可能值: *all* (默认), *disk* - 不间断睡眠, *run* - 运行, *sleep* - 可中断的睡眠, *trace* - 停止, *zomb* - 僵尸 **cmdline** - 按命令行过滤 (它是一个正则表达式) **zone** -目标区域: *current* (默认), *all*. 此参数仅在 Solaris 上受支持。	例如: => proc.num[,mysql] → 在 mysql 用户下运行的进程数 => proc.num[apache2,www-data] → www-data 用户下运行的 apache2 进程数 => proc.num[,oracle,sleep,oracleZABBIX] → 在 oracleZABBIX 命令行下的 oracle 用户运行的睡眠状态进程数 说明 关于选择进程 `name` 和 `cmdline` 参数 (指定于 Linux)。 在 Windows 上仅支持 `name` 和 `user` 参数 当此监控项从命令行调用并包含命令行参数时 (例如使用 agent 测试模式: `zabbix_agentd -t proc.num[,,,apache2]`), 一个额外的进程将被计算在内，因为 agent 将计算自己。 请注意将 zone 参数设置为 *current* （或默认值）如果 agent 已在不支持区域的 Solaris 上编译，但在支持区域的较新 Solaris 上运行，则 agent 将返回 NOTSUPPORTED（agent 不能将结果限制为仅当前区域)。但是，在这种情况下，*all* 都支持。

传感器数据

键值			
描述	返回值	参数	注释
sensor[device,sensor,<mode>]			
硬件传感器读数	浮点型	**device** - 设备名称 **sensor** - 传感器名称 **mode** - 可能的值: *avg*, *max*, *min* (如果省略此参数，设备和传感器将逐字处理).	在 linux2.4 上读取 /proc/sys/dev/sensors 例如: => sensor[w83781d-i2c-0-2d,temp1] 在 Zabbix 1.8.4 之前, *sensor[temp1]* 格式被使用。在 Linux2.6+ 上读取 /sys/class/hwmon 请参阅 Linux sensor监控项中更详细的说明

键值			
			读取 OpenBSD 上的 *hw.sensors* MIB 例如: => sensor[cpu0,temp0] → 一个 CPU 的温度 => sensor[”cpu[0-2]$”,temp,avg] → 前三个 CPU 温度的平均值 自 Zabbix 1.8.4 后开始支持 OpenBSD

系统数据

键值			
描述	返回值	参数	注释
system.boottime			
系统启动时间。	整型 (Unix 时间戳)		
system.cpu.discovery			
检测到的 CPU/CPUs 的内核列表，用于底层自动发现。	JSON 对象		自 2.4.0 后，支持所有平台
system.cpu.intr			
设备中断。	整型		
system.cpu.load[<cpu>,<mode>]			
CPU 负载.	浮点型	**cpu** - possible values: *all* (default), *percpu* (从版本 2.0 开始; 总负载除以在线 CPU 数) **mode** - 可能的值：*avg1* (一分钟的平均值，默认), *avg5*, *avg15*	Example: => system.cpu.load[,avg5].
system.cpu.num[<type>]			
CPU 数量。	整型	**type** - 可能的值：*online* (default), *max*	Example: => system.cpu.num
system.cpu.switches			
上下文切换次数。	整型		
system.cpu.util[<cpu>,<type>,<mode>,<logical_or_physical>]			
CPU 利用率百分比。	浮点型	**cpu** - *<CPU number>* or *all* (default) **type** - 可能的值：*user* (default), *idle*, *nice*, *system* （Windows 默认），*iowait*, *interrupt*, *softirq*, *steal*, *guest* (Linux 内核版本 2.6.24 及以上), *guest_nice* (Linux 内核版本 2.6.33 及以上). 可在平台特性 查看更多此参数的细节。 **mode** - 可能的值：*avg1* (一分钟的平均值，默认), *avg5*, *avg15* **logical_or_physical** (从 5.0.3 版本开始; 仅支持 AIX) - 可能的值：*logical* （默认），*physical*. 此参数仅支持 AIX。	Example: => system.cpu.util[0,user,avg5] 曾用名: *system.cpu.idleX*, *system.cpu.niceX*, *system.cpu.systemX*, *system.cpu.userX*
system.hostname[<type>, <transform>]			

键值			
系统主机名	字符串	**type** (在版本 5.4.7 前仅支持 Windows) - 可能的值：*netbios* （Windows 默认），*host* (Linux 默认), *shorthost* (从 5.4.7 版本开始; 返回第一个"." 前的主机名或在没有"." 时返回完整的主机名). **transform** (从 5.4.7 版本开始) - 可能的值：*none* （默认），*lower* (转换到小写)	这个值通过 Windows 的函数 GetComputerName() (**netbios**) 或者 gethostname() (**host**) 和其他操作系统的"hostname" 命令获取。 返回值示例： 在 *Linux* 上: => system.hostname → linux-w7x1 => system.hostname → example.com => system.hostname[shorthost] → example *on Windows*: => system.hostname → WIN-SERV2008-I6 => system.hostname[host] → Win-Serv2008-I6LonG => system.hostname[host,lower] → win-serv2008-i6long 可查看更详细的描述.
system.hw.chassis[<info>] 板卡信息。	字符串	**info** - one of full （默认），model, serial, type or vendor	Example: system.hw.chassis[full] Hewlett-Packard HP Pro 3010 Small Form Factor PC CZXXXXXXXX Desktop] 这个键依赖于SMBIOS 表的可用性。 会尝试从 sysfs 读取 DMI 表，如果访问 sysfs 失败，会尝试直接读取内存。 需要 **Root** 权限因为这个值是通过读取 sysfs 或 memory 获得的。 从 Zabbix 2.0 开始支持。
system.hw.cpu[<cpu>,<info>] CPU 信息。	字符串或整型	**cpu** - *<CPU number>* or *all* (默认) **info** - 可能的值：*full* （默认），*curfreq*, *maxfreq*, *model* or *vendor*	Example: => system.hw.cpu[0,vendor] → AuthenticAMD 从/proc/cpuinfo 和 /sys/devices/system/cpu/[cpunum]/cpufreq 获取信息。 如果设置了 CPU 数和 *curfreq* 或 *maxfreq*，会返回数字值 (Hz)。 从 Zabbix 2.0 开始支持。
system.hw.devices[<type>]			

键值			
PCI 或 USB 设备列表。	文本	**type** (从版本 2.0 开始) - *pci* (默认) or *usb*	Example: => system.hw.devices[pci] → 00:00.0 Host bridge: Advanced Micro Devices [AMD] RS780 Host Bridge [..] 返回 lspci 或 lsusb 程序的结果（不带参数执行）。
system.hw.macaddr[<interface>,<format>]			
MAC 地址列表。	字符串	**interface** - *all* （默认）或正则表达式 **format** - *full* （默认）或 *short*	列出接口名称匹配接口正则表达式的 MAC 地址列表 (*all* 列出所有接口). 示例： => system.hw.macaddr[”eth0$”,full] → [eth0] 00:11:22:33:44:55 If 如果 `format` 被设为 *short*，接口名称和重复的 MAC 地址将不会列出。 从 Zabbix 2.0 开始支持。
system.localtime[<type>]			
系统时间。	整型 - `type` 是 *utc* 时 字符串 - `type` 是 *local* 时	**type** (从版本 2.0 开始) - 可能的值： *utc* - （默认）从 (00:00:00 UTC, January 1, 1970) 开始计算的时间, 单位是秒。 *local* - 格式为’yyyy-mm-dd,hh:mm:ss.nnn,+hh:mm’ 的时间	只能用于被动检查。 示例： => system.localtime[local] → 用这个键创建一个监控项用于在 *Clock* 仪表盘组件展示主机时间。
system.run[command,<mode>]			
在主机上运行特定的命令。	命令的文本结果 1 - `mode` 为 *nowait* 时 (无论命令结果如何)	**command** - 待执行的命令 **mode** - 可能的值： *wait* - 等待执行结束（默认）， *nowait* - 不等待	包括被截取的空白符，最大可以返回 512KB 的数据。 命令的输出必须是文本才能正常处理。 示例： => system.run[ls -l /] → 根目录的详细文件列表。 *Note*: system.run 监控项默认被禁用。在启用监控项学习如何启用。 这个监控项的返回值是命令产生的标准输出和标准错误，未检查退出码。 从 Zabbix 2.4.0 开始允许空结果。 也可查看命令执行。
system.stat[resource,<type>]			

键值		
系统统计数据。	整型或浮点型	**ent** - 该区域可获取的处理器单元数量（浮点型） **kthr,<type>** - 关于内核线程状态的信息: *r* - 可运行的内核线程的平均数（浮点型） *b* - 虚拟内存管理器等待队列中内核线程的平均数（浮点型） **memory,<type>** - 虚拟和实际内存的使用信息： *avm* - 活跃的虚拟页 (整型) *fre* - 可用列表的大小 (整型) **page,<type>** - 页错误和分页活动的信息： *fi* - 每秒文件入页（浮点型） *fo* - 每秒文件出页（浮点型） *pi* - 从分页空间入页（浮点型） *po* - 向分页空间出页（浮点型） *fr* - 页释放（页替换）(浮点型) *sr* - 通过页替换算法扫描到的页（浮点型） **faults,<type>** - 陷阱和中断率： *in* - 设备中断（浮点型） *sy* - 系统调用（浮点型） *cs* - 内核线程上下文切换（浮点型） **cpu,<type>** - 处理器时间的分类百分比： *us* - 用户时间（浮点型） *sy* - 系统时间（浮点型） *id* - 空闲时间（浮点型） *wa* - 系统在磁盘或网络文件系统 I/O 请求多时的空闲时间 (浮点型) *pc* - 物理处理器消耗的时间 (浮点型) *ec* - 标称计算能力的消耗的百分比（浮点型） *lbusy* - 在用户级和系统级运行时，逻辑处理器的使用百分比（浮点型） *app* - 共享池中可用物理处理器数量（浮点型） **disk,<type>** - 磁盘统计信息： *bps* - 磁盘驱动器传输的数据量，单位是字节/秒（整型） *tps* - 发送给物理磁盘/磁带的每秒传输次数（浮点型），从 Zabbix 1.8.1 开始，此监控项仅被 AIX支持。 请注意这些监控项的以下限制： => system.stat[cpu,app] - 仅支持”Shared” 类型的 AIX LPAR => system.stat[cpu,ec] - 仅支持”Shared” 和”Dedicated” 类型的 AIX LPAR (”Dedicated” 一直返回 100 (百分比)) => system.stat[cpu,lbusy] - 仅支持”Shared” 类型的 AIX LPAR

键值			
system.sw.arch 软件架构信息。	字符串		示例： => system.sw.arch → i686 通过 uname() 函数获取信息。 从 Zabbix 2.0 开始支持。
system.sw.os[<info>] 操作系统信息。	字符串	**info** - 可能的值： *full* （默认），*short* or *name*	示例： => system.sw.os[short]→ Ubuntu 2.6.35-28.50-generic 2.6.35.11 信息通过以下方式（请注意并非所有发行版本都存在所有文件和选项）获取： /proc/version (*full*) /proc/version_signature (*short*) PRETTY_NAME 参数在支持的系统从/etc/os-release 获取，或/etc/issue.net (*name*) 从 Zabbix 2.0 开始支持。
system.sw.packages[<package>,<manager>,<format>] 已安装的包列表。	文本	**package** - *all* （默认）或正则表达式 **manager** - *all* （默认）或包管理器 **format** - *full* （默认）或 *short*	按首字母顺序，列出匹配 package正则表达式的已安装的包名 (*all* 列出全部). 示例： => system.sw.packages[mini,dpkg,short] → python-minimal, python2.6-minimal, ubuntu-minimal 支持的包管理器（执行的命令）： dpkg (dpkg --get-selections) pkgtool (ls /var/log/packages) rpm (rpm -qa) pacman (pacman -Q) 如果 `format` 设为 *full*，包按照包管理器分组（每个包管理器在单独的行，开头是方括号中的包管理器名称）。 如果 `format` 设为 *short*，包不会分组，会列在一行中。 从 Zabbix 2.0 开始支持。
system.swap.in[<device>,<type>]			

键值			
交换入（从设备到内存）统计	整型	**device** - 用于交换的设备（默认是 *all*） **type** - 可能的值：*count* （交换入的数量），*sectors* （交换入的分区），*pages* （交换入的页）. 也可在平台特性查看此参数的详情。	示例： => system.swap.in[,pages] 此信息的来源是： /proc/swaps, /proc/partitions, /proc/stat (Linux 2.4) /proc/swaps, /proc/diskstats, /proc/vmstat (Linux 2.6)
system.swap.out[<device>,<type>] 交换出（从内存到设备）统计。	整型	**device** - 用于交换的设备（默认是 *all*） **type** - 可能的值：*count* （交换出的数量），*sectors* （交换出的分区），*pages* （交换出的页）。 也可在平台特性查看此参数的详情。	示例： => system.swap.out[,pages] 此信息的来源是： /proc/swaps, /proc/partitions, /proc/stat (Linux 2.4) /proc/swaps, /proc/diskstats, /proc/vmstat (Linux 2.6)
system.swap.size[<device>,<type>] 交换区空间大小，字节数或总量的百分比。	整型 - 字节数的类型 浮点型 - 百分比的类型	**device** - 用于交换的设备 (默认是 *all*) **type** - 可能的值：*free* (空闲交换空间，默认), *pfree* (空闲交换空间的百分比), *pused* (已用交换空间的百分比), *total* (总交换空间), *used* (已用交换空间) 请注意 *pfree*, *pused* 在交换空间为 0 时不支持 Windows。 也可在平台特性查看此参数的详情。	示例： => system.swap.size[,pfree] → 空闲交换空间的百分比 如果未设置 *device* 的值，Zabbix agent 仅会考虑账户交换设备（文件），忽略物理内存。比如，在 Solaris 系统 *swap -s* 命令包括物理内存的一部分以及交换设备（不同于 *swap -l*）。 请注意此键在虚拟化（VMware ESXi, VirtualBox）Windows 平台可能报告错误的交换空间大小/百分比。在这种情况下你可使用 `perf_counter[\700(_Total)\702]` 键来获取正确的交换空间百分比。
system.uname			

键值			
系统标识	字符串		返回值实例（Unix）：FreeBSD localhost 4.2-RELEASE FreeBSD 4.2-RELEASE #0: Mon Nov i386 返回值实例（Windows）：Windows ZABBIX-WIN 6.0.6001 Microsoft® Windows Server® 2008 Standard Service Pack 1 x86 在 Unix，从 Zabbix 2.2.0 开始，此值通过 uname() 系统调用获取，在此之前通过调用”uname -a” 获取。此监控项的值可能和”uname -a” 的输出不同，并且不包括”uname -a” 打印的从其他来源获取的额外信息。 在 Windows 系统，从 Zabbix 3.0 开始，此值通过 Win32_OperatingSystem 和 Win32_Processor WMI 类获取。在此之前通过不稳定的 Windows APIs 和无稳定的注册表键获取。OS 名（包括版本）可能被翻译到了用户的显示语言。在 Windows 的某些版本，OS 名包括商标符号和额外的空格。 请注意在 Windows，此监控项返回 OS 架构，而在 Unix 返回 CPU 架构。
system.uptime 系统运行时间，单位是秒。	整型		在监控项配置中，使用 **s** or **uptime** 单位获取可读的值。
system.users.num 登录用户数	整型		**who** 命令是用于在 agent 处获取值的。

虚拟文件系统数据

键值			
描述	返回值	参数	注释
vfs.dev.discovery 块设备和类型的列表，用于底层自动发现。	JSON 对象		此监控项仅支持 Linux 平台。 从 Zabbix 4.4.0 开始支持。
vfs.dev.read[<device>,<type>,<mode>]			

键值			
磁盘读取统计信息。	整型 - 当 `type` 在 *sectors*, *operations*, *bytes* 中时 浮点型 - 当 `type` 在 *sps*, *ops*, *bps* 中时 注意: 如果更新间隔在 3 小时及以上时[2], 会一直返回'0'	**device** - 磁盘设备（默认是 *all* [3]） **type** - 可能的值：*sectors*, *operations*, *bytes*, *sps*, *ops*, *bps* 请注意'type' 在各平台的支持和默认情况，在平台特性查看详情。 *sps*, *ops*, *bps* 分别代表：sectors, operations, bytes 每秒的值。 **mode** - 可能的值：*avg1* (一分钟的平均值，默认), *avg5*, *avg15*. 此参数仅支持 `type` 在 sps, ops, bps 中的情况。	你也可使用相关设备名称（如 `sda`）加 /dev/ 前缀（如`/dev/sda`）。 支持 LVM 逻辑卷。 不同操作系统的'type' 默认值： AIX - operations FreeBSD - bps Linux - sps OpenBSD - operations Solaris - bytes 示例： => vfs.dev.read[,operations] *sps*, *ops* and *bps* 在支持的平台曾经支持 8 个设备（7 个单独的和一个 *all*）。从 Zabbix 2.0.1 开始这个限制是 1024 个设备（1023 个单独的和一个 *all*）。
vfs.dev.write[<device>,<type>,<mode>]			
磁盘写入统计信息。	整型 - 当 `type` 在 *sectors*, *operations*, *bytes* 中时 浮点型 - 当 `type` 在 *sps*, *ops*, *bps* 中时 注意: 如果更新间隔在 3 小时及以上时[2], 会一直返回'0'	**device** - 磁盘设备（默认是 *all* [3]） **type** - 可能的值：*sectors*, *operations*, *bytes*, *sps*, *ops*, *bps* 请注意'type' 在各平台的支持和默认情况，在平台特性查看详情。 *sps*, *ops*, *bps* 分别代表：sectors, operations, bytes 每秒的值。 **mode** - 可能的值：*avg1* (一分钟的平均值，默认), *avg5*, *avg15*. 此参数仅支持 `type` 在 sps, ops, bps 中的情况。	你也可使用相关设备名称（如 `sda`）加 /dev/ 前缀（如`/dev/sda`）。 支持 LVM 逻辑卷。 不同操作系统的'type' 默认值： AIX - operations FreeBSD - bps Linux - sps OpenBSD - operations Solaris - bytes 示例： => vfs.dev.read[,operations] *sps*, *ops* and *bps* 在支持的平台曾经支持 8 个设备（7 个单独的和一个 *all*）。从 Zabbix 2.0.1 开始这个限制是 1024 个设备（1023 个单独的和一个 *all*）。
vfs.dir.count[dir,<regex_incl>,<regex_excl>,<types_incl>,<types_excl>,<max_depth>,<min_size>,<max_size>,<min_age>,<max_ag			

键值			
目录条目计数。	整型	**dir** - 目录的绝对路径 **regex_incl** - 描述包含的 (文件, 目录, 符号链接) 的名称模式的正则表达式；如果为空，则包含所有 (默认值) **regex_excl** - 描述了排除 (文件, 目录, 符号链接) 的名称模式正则表达式；如果为空，不排除任何实体 (默认) **types_incl** - 要计数的目录条目类型, 可能的值：*file* - 常规文件, *dir* - 子目录, *sym* - 符号链接, *sock* - 套接字, *bdev* - 块设备, *cdev* - 字符设备, *fifo* - FIFO, *dev* - "bdev,cdev" 同义词, *all* - 所有类型（默认），即"file,dir,sym,sock,bdev,cdev,fifo"。多种类型时需在双引号中用逗号隔开。 **types_excl** - 不需要计数的目录条目类型 (见 <types_incl>)。如果某些条目类型同时位于 <types_incl> 和 <types_excl> 中, 则不计算此类型的目录条目。 **max_depth** - 遍历的子目录最大深度 **-1** （默认）- 无限制, **0** -不涉及子目录。 **min_size** -要计数文件的最小大小（以字节为单位）。小于此的文件不会被计算在内，内存后缀 可以被使用。 **max_size** - 要计数文件的最大大小（以字节为单位），大于此的文件将不会被计算在内。内存后缀 可以被使用 **min_age** - 要计数的目录条目最小创建时间（以秒为单位）。最新的条目不进行计数。时间后缀 可以被使用。 **max_age** - 要计数的目录条目最大创建时间（以秒为单位）。比此旧的条目将不被计算在内 (修改时间)。时间后缀 can 可以被使用. **regex_excl_dir** - 描述要排除的目录的名称模式的正则表达式，目录的所有内容都将被排除 (与 regex_excl 相比)	不支持的环境变量, 如%APP_HOME%, $HOME 和%TEMP% 。 伪目录"." 和".." 不会被计算在内。 目录遍历不会跟踪符号链接。 在 Windows 上，目录符号链接被跳过，硬链接只计算一次。 在计算条目大小时，`regex_incl` 和 `regex_excl` 都被应用于文件和目录，但在选择要遍历的目录时被忽略 (如果 regex_incl 是 "(?i)^.+\.zip$" 并且 max_depth 未设置, 那么所有子目录都将是遍历，但只会计算 zip 类型的文件)。 执行时间受限于代理配置 (3 秒) 的默认时间设置，由于大目录遍历可能需要更长的时间，因此不会返回任何数据并且该监控项项将变为不受支持。部分计数将不予返回 当按照大小过滤时，只有常规文件的大小具有意义，在 Linux 和 BSD 下，目录也有非零大小 (通常只有几 KB)。设备大小为 0 如 **/dev/sda1** 的大小不反映相应分区的大小。因此，在使用 `<min_size>` 和 `<max_size>`, 建议指定 `<types_incl>` 为"*file*", 防止出现意外。 例如: ⇒ vfs.dir.count[/dev] - 监控 /dev (Linux) 中的设备数量 ⇒ vfs.dir.count["C:\Users\ADMINI~1\AppData - 监控临时目录下的文件数量 (Windows) 自 Zabbix 4.0.0 后支持。

vfs.dir.get[dir,<regex_incl>,<regex_excl>,<types_incl>,<types_excl>,<max_depth>,<min_size>,<max_size>,<min_age>,<max_age>

键值			
目录条目列表	JSON	**dir** - 目录的绝对路径 **regex_incl** - 描述包含的 (文件, 目录, 符号链接) 的名称模式的正则表达式；如果为空，则包含所有 (默认值) **regex_excl** - 描述了排除 (文件, 目录, 符号链接) 的名称模式正则表达式；如果为空，不排除任何实体 (默认) **types_incl** - 要计数的目录条目类型, 可能的值：*file* - 常规文件, *dir* - 子目录, *sym* - 符号链接, *sock* - 套接字, *bdev* - 块设备, *cdev* - 字符设备, *fifo* - FIFO, *dev* - ”bdev,cdev” 同义词, *all* - 所有类型（默认），即”file,dir,sym,sock,bdev,cdev,fifo”。多种类型时需在双引号中用逗号隔开。 **types_excl** - 不需要计数的目录条目类型 (见 <types_incl>)。如果某些条目类型同时位于 <types_incl> 和 <types_excl> 中, 则不计算此类型的目录条目。 **max_depth** - 遍历的子目录最大深度 **-1** （默认）- 无限制, **0** -不涉及子目录。 **min_size** -要计数文件的最小大小（以字节为单位）。小于此的文件不会被计算在内，内存后缀 可以被使用。 **max_size** - 要计数文件的最大大小（以字节为单位），大于此的文件将不会被计算在内。内存后缀 可以被使用 **min_age** - 要计数的目录条目最小创建时间（以秒为单位）。最新的条目不进行计数。时间后缀 可以被使用。 **max_age** - 要计数的目录条目最大创建时间（以秒为单位）。比此旧的条目将不被计算在内 (修改时间)。时间后缀 can 可以被使用. **regex_excl_dir** - 描述要排除的目录的名称模式的正则表达式，目录的所有内容都将被排除 (与 regex_excl 相比)	不支持的环境变量, 如%APP_HOME%, $HOME 和%TEMP% 。 伪目录”.” 和”..” 不会被计算在内。 目录遍历不会跟踪符号链接。 在 Windows 上，目录符号链接被跳过，硬链接只计算一次。 在计算条目大小时，`regex_incl` 和 `regex_excl` 都被应用于文件和目录，但在选择要遍历的子目录时被忽略 (如果 regex_incl 是 “(?i)^.+\.zip$” 并且 max_depth 未设置, 那么所有子目录都将是遍历，但只会计算 zip 类型的文件)。 执行时间受限于代理配置 (3 秒) 的默认时间设置，由于大目录遍历可能需要更长的时间，因此不会返回任何数据并且该监控项项将变为不受支持。部分计数将不予返回 当按照大小过滤时，只有常规文件的大小具有意义，在 Linux 和 BSD 下，目录也有非零大小 (通常只有几 KB)。设备大小为 0 如 **/dev/sda1** 的大小不反映相应分区的大小。因此，在使用 `<min_size>` 和 `<max_size>`, 建议指定 `<types_incl>` 为”*file*”, 防止出现意外。 例如: ⇒ vfs.dir.count[/dev] - 监控 /dev (Linux) 中的设备数量 ⇒ vfs.dir.count[”C:\Users\ADMINI~1\AppData - 监控临时目录下的文件数量 (Windows) 自 Zabbix 6.0.0 后支持。

vfs.dir.size[dir,<regex_incl>,<regex_excl>,<mode>,<max_depth>,<regex_excl_dir>]

键值			
目录大小 (以字节为单位).	整型	**dir** - 目录的绝对路径 **regex_incl** - 描述包含的 (文件, 目录, 符号链接) 的名称模式的正则表达式；如果为空，则包含所有 (默认值) **regex_excl** - 描述了排除 (文件, 目录, 符号链接) 的名称模式正则表达式；如果为空，不排除任何实体 (默认) **mode** - 可能的值：*apparent* （默认）- 获取明确文件大小而不是磁盘使用情况 (`du -sb dir`), *disk* - 获取磁盘使用情况 (`du -s -B1 dir`)。与 du 命令不同, vfs.dir.size 监控项在计算目录大小会考虑隐藏文件 (`du -sb .[^.]* *` 在目录中起作用). **max_depth** -目录遍历的最大深度。**-1** （默认）- 无限制, **0** -不下钻到子目录 **regex_excl_dir** - 描述要排除的目录的名称模式的正则表达式，目录的所有内容都将被排除 (与 regex_excl 相比)	只计算 *zabbix* 用户至少可读权限的目录 在 Windows 上任何软链接被跳过，硬链接只计算一次 对于大型目录或慢速驱动器，此监控项可能会根据agent和server/proxy 配置文件中的超时设置而超时，可以根据需要设置超时值。 示例: ⇒ vfs.dir.size[/tmp,log] - 计算 /tmp 目录中包含'log' 的所有文件大小； ⇒ vfs.dir.size[/tmp,log,^.+\.old$] -计算 /tmp 目录中包含'log' 的所有文件大小，排除文件中包含'.old' 的文件。 文件大小限制取决于大文件支持。 自 Zabbix 3.4.0 后开始支持
vfs.file.cksum[file,<mode>] 文件校验和，由 UNIX cksum 算法计算得出。	整型 -`mode` 作为 *crc32* 字符串 - `mode` 作为 *md5*, *sha256*	**file** - 文件模式的全路径 **mode** - *crc32* （默认），*md5*, *sha256*	示例： => vfs.file.cksum[/etc/passwd] 返回值示例 (分别为 crc32/md5/sha256): 675436101 9845acf68b73991eb7fd7ee0ded23c44 ae67546e4aac995e5c921042d0cf0f1f7147 文件大小限制取决于大文件支持。 自 Zabbix 6.0 后开始支持 `mode` 参数
vfs.file.contents[file,<encoding>] 检索文件的内容。	文本	**file** - 文件的全路径 **encoding** - 编码标识符	如果文件为空或仅包含 LF/CR 字符，则返回空字符串。 字节顺序标记 (BOM) 从输出中排除。 示例： => vfs.file.contents[/etc/passwd] 此监控项仅限于不大于 64 KB 的文件。 从 Zabbix 2.0 开始支持。
vfs.file.exists[file,<types_incl>,<types_excl>]			

键值			
检查文件是否存在。	0 -没有找到 1 - 指定文件类型存在	**file** - 文件的完整路径。 **types_incl** - 包含的文件类型列表, 可能的值：*file* (常规文件，默认 (如果 types_excl 未设置)), *dir* (目录), *sym* (符号链接), *sock* (套接字), *bdev* (块设备), *cdev* (字符设备), *fifo* (FIFO), *dev* (与”bdev,cdev” 同义), *all* (所有提到的类型，默认为如果 types_excl 设置的值)。 **types_excl** - 排除的文件类型列表, 参阅 types_incl 获取可能的值 (默认情况下不排除任何类型)	多个类型必须用逗号分隔，整个集合用引号”” 括起来. 在 Windows 上，当使用命令行调用 zabbix_get.exe 或 agent2 时，双引号必须用反斜杠’\’ 转义，并且整个项目键用双引号括起来。 如果 <types_incl> 和 <types_excl> 中的类型相同, 则排除此类型的文件 例如: => vfs.file.exists[/tmp/application.pid] => vfs.file.exists[/tmp/application.pid,”file,dir, => vfs.file.exists[/tmp/application_dir,dir] 文件大小取决于大文件支持。 请注意，如果在不存在的目录中搜索目录，例如 vfs.file.exists[C:\no\dir,dir] (其中不存在’no’) 则该监控项可能会在 Windows 上变为不受支持
vfs.file.get[file] 返回有关文件的信息。	JSON 对象	**file** - 文件的全路径	类 UNIX 系统上支持的文件类型: 常规文件、目录、符号链接、套接字、块设备、字符设备、FIFO Windows 上支持的文件类型：常规文件、目录、符号链接 示例： => vfs.file.get[/etc/passwd] → 返回一个 JSON，其中包含有关 /etc/passwd 文件的信息 (类型、用户、权限、SID、uid 等) 自 Zabbix 6.0 后开始支持
vfs.file.md5sum[file] 文件的 MD5 校验和。	字符串 (MD5 的哈希文件)	**file** - 文件的全路径	示例： => vfs.file.md5sum[/usr/local/etc/zabbix_agen 示例的返回值: b5052decb577e0fffd622d6ddc017e82 版本 1.8.6 中删除了此项目的文件大小限制 (64 MB)。 文件大小限制取决于大文件支持。
vfs.file.owner[file,<ownertype>,<resulttype>]			

键值			
检索文件的所有者	字符串	**file** - 文件的全路径 **ownertype** - *user* （默认）或 *group* (仅 Unix) **resulttype** - *name* （默认）或 *id*; 对于 id - 在 Unix 上返回 uid/gid，Windows 上返回 SID	示例： => vfs.file.owner[/tmp/zabbix_server.log] → 返回/tmp/zabbix_server.log 文件所有者 => vfs.file.owner[/tmp/zabbix_server.log,,id] → 返回 /tmp/zabbix_server.log 文件所有者 id 自 Zabbix 6.0 后开始支持。
vfs.file.permissions[file] 返回一个 4 位字符串，其中包含具有 Unix 权限的八进制数。	字符串	**file** - 文件的全路径	在 Windows 上不支持 示例： => vfs.file.permissions[/etc/passwd] → 返回/etc/passwd 的文件权限, 例如, '0644' 自 Zabbix 6.0 后开始支持
vfs.file.regexp[file,regexp,<encoding>,<start line>,<end line>,<output>] 在文件中查找字符串。	包含匹配字符串的行，或由可选 `output` 参数指定的行	**file** - 文件的全路径 **regexp** - 描述所需模式编码的正则表达式 **encoding** - 编码标识符 **start line** - 要搜索的第一行 (默认为文件的第一行). **end line** - 要搜索的最后一行的编号 (默认为文件的最后一行)。 **output** - 可选的输出格式模板 **\0** 转义序列被替换为匹配的文本部分 (从匹配开始的第一个字符到匹配结束的字符) 而 **\N** (其中 N=1...9) 转义序列被替换为第 N 个匹配组 (如果 N 超过捕获组的数量，则为空字符串)。	仅返回第一个匹配行。 如果没有行匹配表达式，则返回一个空字符串。 字节顺序标记 (BOM) 从输出中排除。 使用该参数的内容提取 `output` 发生在代理上。 从 2.2 版本开始支持 `start line`, `end line` and `output` 参数。 例如: => vfs.file.regexp[/etc/passwd,zabbix] => vfs.file.regexp[/path/to/some/file,"([0-9]+)$",,3,5,\1] => vfs.file.regexp[/etc/passwd,"^zabbix:.:([0-9]+)",,,\1] → 获取 *zabbix* 用户 ID
vfs.file.regmatch[file,regexp,<encoding>,<start line>,<end line>] 在文件中查找字符串。	0 - 未找到匹配 1 - 找到	**file** - 文件全路径 **regexp** - 描述所需模式编码的正则表达式 **encoding** - 编码标识符 **start line** - 要搜索的第一行 (默认为文件的第一行). **end line** - 要搜索的最后一行的编号 (默认为文件的最后一行)。	字节顺序标记 (BOM) 被忽略。 从 2.2 开始支持 `start line` 和 `end line` 参数。 示例： => vfs.file.regmatch[/var/log/app.log,error]
vfs.file.size[file,<mode>]			

键值			
文件大小 (以字节为单位).	整型	**file** - 文件全路径 **mode** - 可能的值： *bytes* （默认）或 *lines* (也计算空行)	该文件必须对用户 *zabbix* 具有读取权限。 示例： => vfs.file.size[/var/log/syslog] 文件大小限制取决于大文件支持。 自 Zabbix6.0 起支持 mode 参数。
vfs.file.time[file,<mode>] 文件时间信息。	整型 (Unix 时间戳)	**file** - 文件全路径 **mode** - 可能的值： *modify* （默认）- 最后修改文件内容的时间 *access* - 最后读文件的时间。 *change* - 最后更改文件属性的时间。	示例： => vfs.file.time[/etc/passwd,modify] 文件大小限制取决于大文件支持.
vfs.fs.discovery 已安装文件系统及其类型的列表。用于底层自动发现.	JSON 对象		从 Zabbix 2.0 开始支持。 自 Zabbix3.0 起，Windows 支持 {#FSDRIVETYPE} 宏。 自 Zabbix6.0 起，Windows 支持 {#FSLABEL} 宏。
vfs.fs.get 挂载的文件系统列表、它们的类型、磁盘空间和 inode 统计信息。可用于底层自动发现.	JSON 对象		自 Zabbix 4.4.5 后开始支持。 自 Zabbix 6.0 起，Windows 上支持 {#FSLABEL} 宏
vfs.fs.inode[fs,<mode>] inode 数量或百分比	整型 - 表示数量 浮点型 - 用于百分比	**fs** - 文件系统 **mode** - 可能的值： *total* （默认），*free*, *used*, //pfree // (空闲, 百分比), *pused* (使用, 百分比)	示例： => vfs.fs.inode[/,pfree]
vfs.fs.size[fs,<mode>] 磁盘空间（以字节为单位）或占总数的百分比。	整型 - 字节 浮点型 - 百分比	**fs** - 文件系统 **mode** - 可能的值： *total* (默认), *free*, *used*, *pfree* (空闲, 百分比), *pused* (使用, 百分比)	如果是已挂载的卷，则返回本地文件系统的磁盘空间。 示例： => vfs.fs.size[/tmp,free] 当使用 *free* 模式时，文件系统的保留空间不考虑在内.

虚拟机数据

键值			
描述	返回值	参数	注释
vm.memory.size[<mode>]			

键值			
内存大小（以字节为单位）或占总数的百分比。	整型 - 字节 浮点型 -百分比	**mode** - 可能的值：*total* (默认), *active*, *anon*, *buffers*, *cached*, *exec*, *file*, *free*, *inactive*, *pinned*, *shared*, *slab*, *wired*, *used*, *pused* (used, percentage), *available*, *pavailable* (available, percentage) 参阅平台特性 支持和此参数的其他详细信息。	此监控项接受三类参数： 1) *total* - 内存总量; 2) 特定于平台的内存类型: *active*, *anon*, *buffers*, *cached*, *exec*, *file*, *free*, *inactive*, *pinned*, *shared*, *slab*, *wired*; 3) 用户级估计有多少内存已使用和可用: *used*, *pused*, *available*, *pavailable*。

网页监控数据

键值			
描述	返回值	参数	注释
web.page.get[host,<path>,<port>]			
获取网页内容。	作为文本的网页源（包括标题）	**host** - 主机名或 URL (如 `scheme://host:port/path`, 只有 *host* 是强制的). 允许的 URL 方案: *http*, *https*[4]. 缺少的方案将被使用 *http*。如果 URL 指定了 `path` 和 `port` 必须为空。连接服务需要验证时需指定用户名和密码，示例：`http://user:password@www.example.com` 只有 curl[4] 才有可能。 主机名支持 Punycode。 **path** - HTML 文档路径 (默认为 /) **port** - 端口号 (默认是 http 80)	如果指定的资源在 `host` 中不存在或不可用，则此监控项将不受支持。 `host` 可以是主机名、域名、IPv4 或 IPv6 地址。但是对于 IPv6 地址，Zabbix agent 必须在启用 IPv6 支持的情况下编译。 示例： => web.page.get[www.example.com,index.ph => web.page.get[https://www.example.com] => web.page.get[https://blog.example.com/?s => web.page.get[localhost:80] => web.page.get[”[::1]/server-status”]
web.page.perf[host,<path>,<port>]			
整个网页的加载时间（以秒为单位）。	浮点型	**host** - 主机名或 URL (如 `scheme://host:port/path`, 只有 *host* 是强制的). 允许的 URL 方案: *http*, *https*[4]. 缺少的方案将被使用 *http*。如果 URL 指定了 `path` 和 `port` 必须为空。连接服务需要验证时需指定用户名和密码，示例：`http://user:password@www.example.com` 只有 curl[4] 才有可能。 主机名支持 Punycode。 **path** - HTML 文档路径 (默认为 /) **port** - 端口号 (默认是 http 80)	如果指定的资源在 `host` 中不存在或不可用，则此监控项将不受支持。 `host` 可以是主机名、域名、IPv4 或 IPv6 地址。但是对于 IPv6 地址，Zabbix agent 必须在启用 IPv6 支持的情况下编译。 示例： => web.page.perf[www.example.com,index.ph => web.page.perf[https://www.example.com]
web.page.regexp[host,<path>,<port>,regexp,<length>,<output>]			

键值			
在网页上查找字符串。	匹配的字符串, 或由可选 `output` 参数指定。	**host** - 主机名或 URL (如 `scheme://host:port/path`, 只有 *host* 是强制的). 允许的 URL 方案: *http*, *https*[4]. 缺少的方案将被使用 *http*。如果 URL 指定了 `path` 和 `port` 必须为空。连接服务需要验证时需指定用户名和密码，示例：`http://user:password@www.example.com` 只有 curl[4] 才有可能。 主机名支持 Punycode。 **path** - HTML 文档路径 (默认为 /) **port** - 端口号 (默认是 http 80) **regexp** - 描述了所需模式的正则表达式 **length** -返回最大的字符数 **output** - 可选的输出格式模板，**\0** 转义序列被替换为文本的匹配部分 (从从匹配开始的第一个字符到匹配结束的字符），而 **\N** (其中 N=1...9) 转义序列被替换为第 N 个匹配组 (如果 N 超过捕获的组数，则为空字符串)。	如果指定的资源在 `host` 中不存在或不可用，则此监控项将不受支持。 `host` 可以是主机名、域名、IPv4 或 IPv6 地址。但是对于 IPv6 地址，Zabbix agent 必须在启用 IPv6 支持的情况下编译。 使用该参数的内容提取 `output` 发生在 agent 上。 从 2.2 版本开始支持 `output` 参数。 示例： => web.page.regexp[www.example.com,index => web.page.regexp[https://www.example.cor

Zabbix 指标

项目键			
说明	返回值	参数	注释
agent.hostmetadata 代理主机元数据。	字符串		返回 HostMetadata 或 HostMetadataItem 参数的值，如果未定义则返回空字符串。 自 Zabbix 6.0 起受支持。
agent.hostname 代理主机名。	字符串		返回： 作为被动检查 - 代理配置文件的主机名参数中列出的第一个主机的名称； 作为主动检查 - 当前主机名的名称。
agent.ping 代理可用性检查。	无 - 不可用 1 - 可用		使用 **nodata()** 触发器函数检查主机不可用性。
agent.variant Zabbix 代理的变体（Zabbix 代理或 Zabbix 代理 2）。	整数		返回值示例： 1 - Zabbix agent 2 - Zabbix agent 2
agent.version Zabbix agent 的版本。	字符串		返回值示例： 6.0.3
zabbix.stats[<ip>,<port>]			

项目键			
远程返回一组 Zabbix 服务器或代理内部指标。	JSON 对象	**ip** - 要远程查询的服务器/代理的 IP/DNS/网络掩码列表（默认为 127.0.0.1） ** 端口 ** - 要远程查询的服务器/代理端口（默认为 10051）	请注意，统计请求只会从“StatsAllowedIP” 中列出的地址接受server/proxy 目标实例上的参数。 此监控项返回一组选定的内部指标。详情参见Zabbix stats 的远程监控。
zabbix.stats[<ip>,<port>,queue,<from>,<to>] 返回队列中在 Zabbix 服务器或远程代理上延迟的监控项数。	JSON 对象	**ip** - 要远程查询的服务器/代理的 IP/DNS/网络掩码列表（默认为 127.0.0.1 ） **port** - 要远程查询的服务器/代理端口（默认为 10051） **queue** - 常量（按原样使用） **from** - 至少延迟（默认为 6 秒） 到 - 最多延迟（默认为无限）	请注意，统计请求只会从“StatsAllowedIP” 中列出的地址接受server/proxy 目标实例上的参数。

脚注

[1] Linux 相关。Zabbix agent 必须对文件系统*/proc* 有只读权限。www.grsecurity.org 的内核补丁限制了未授权用户的访问权限。

[2] `vfs.dev.read[]`, `vfs.dev.write[]`: 如果监控项值超过 3 小时无法获取，Zabbix agent 会终止” 过时” 的设备连接。这可能在系统设备动态变化或者手动移除时发生。请注意这些监控项的更新间隔在 3 小时以上时，会永远返回’0’。

[3] `vfs.dev.read[]`, `vfs.dev.write[]`: 如果将默认的 *all* 作为第一个参数，此键会返回汇总统计数据，包括所有的块设备如 sda，sbd 及其分区（sda1, sda2, sdb3...）和基于这些块设备/分区的多个设备（MD raid）和基于这些块设备/分区的逻辑设备（LVM）。在这些情况下返回值应只考虑相对值（随时间动态变化）而不是绝对值。

[4] SSL (HTTPS) 仅在 agent 在支持 cURL 的情况下编译时被支持。否则此监控项会返回不受支持。

编码设置

要确保获取的数据不被损坏，你可能需要在 `encoding` 参数中指定正确的编码来处理检查（例如’vfs.file.contents’）。受支持的的编码（代码页标识符）可在libiconv (GNU 项目) 的文档查找，或在微软 Windows SDK 的文档查找”Code Page Identifiers”。

如果未在 `encoding` 参数中指定编码，会应用以下解决方案策略：

- 如果 encoding 未指定 (或者为空字符串) 被认为是 UTF-8，数据原样处理；
- BOM 分析 - 在监控项’vfs.file.contents’, ’vfs.file.regexp’, ’vfs.file.regmatch’ 可应用。尝试使用文件开头的字节顺序标记（BOM）来决定正确的编码。如果没有 BOM - 改为应用标准解决方案（见上）。

agent 监控项故障排除

- 如果使用被动 agent，server 配置的 *Timeout* 值可能需要比 agent 配置的 *Timeout* 值高。否则此监控项可能不会获取到任何值，因为 server 到 agent 的请求先超时了。

Windows 特定的监控项键值

监控项键值

该表提供了 Zabbix Windows Agent 可用的监控项键值的详细描述.

也可参考: 最低权限级别的 Windows agent 监控项

Key			
描述	返回值	参数	注释
eventlog[name,<regexp>,<severity>,<source>,<eventid>,<maxlines>,<mode>]			

Key			
事件日志监控	日志	**name** - 事件日志名称 **regexp** - 所需匹配的正则表达式 **severity** - 正则表达式的严重级别 此参数接受以下值: "Information", "Warning", "Error", "Critical", "Verbose" (从 Zabbix 2.2.0 开始支持，在 Windows Vista 或更高版本上运行) **source** - 描述源标识符的正则表达式（自 Zabbix 2.2.0 起支持正则表达式） **eventid** - 描述事件标识符的正则表达式 **maxlines** - 代理每秒发送到 Zabbix 服务器或代理的最大新行数。此参数覆盖zabbix_agentd.win.conf 中'MaxLinesPerSecond' 的值. **mode** - 可能的值: *all* (默认), *skip* - 跳过处理旧数据（仅影响新创建的监控项）。	此监控项必须配置为主动模式. 例如: => eventlog[Application] => eventlog[Security,,"Failure Audit",,^(529|680)$] => eventlog[System,,"Warning\|Error"] => eventlog[System,,,,^1$] => eventlog[System,,,,@TWOSHORT] - 这里引用了自定义正则表达式 名为 `TWOSHORT` (定义了 结果为真的类型, 正则表达式本身为 `^1$\|^70$`)。 请注意代理无法从 "转发事件" 日志中发送事件。 `mode` 参数从 Zabbix 2.0.0 后开始支持。 "Windows Eventing 6.0" 从 Zabbix 2.2.0 后开始支持。 请注意选择的监控项非日志类型信息类型 将导致丢失本地时间戳, 源信息和日志级别也会丢失。 同样也可参考日志监控有关信息。
net.if.list 网络接口列表 (包含接口类型，状态，IPv4 地址，描述)。	文本		从 Zabbix agent 版本 1.8.1 后支持多字节网卡名称，从 Zabbix agent 版本 1.8.6 后，禁用的网卡列表并不显示 需要注意的是, 开启/关闭一些组件可能会更改 windows 界面的显示顺序. 某些 windows 版本 (如, Server 2008) 可能需要安装最新更新以支持接口名称中的非 ASCII 字符。
perf_counter[counter,<interval>] 任意 Windows 性能计数值。	整型, 浮点型, 字符串或文本 (取决于请求)	**counter** - 计数器的路径 **interval** - 用于存储平均值的最后 N 秒。`interval` 必须在 1~900 秒 (包含) 默认值为 1。	性能监视器可以用于获取可用的计数器列表。在 1.6 之前，计数器使用一个是简单的样本 (如 \System\Threads) 并返回返回正确的值。对于需要多个样本的计数器（例如 CPU 利用率），它不会按预期工作，1.6 之后，使用了 `interval`，因此每次检查返回最后 "间隔" 秒的平均值。 另请参阅: Windows 性能计数器。
perf_counter_en[counter,<interval>] 任意 Windows 性能计数英文值	整型, 浮点型, 字符串或文本 (取决于请求)	**counter** - 以英文表示的计数器路径 **interval** - 用于存储平均值的最后 N 秒。The `interval` 必须在 1~900 秒 (包含) 默认值为 1。	此监控项仅支持 **Windows Server 2008/Vista** 或以上版本。 你可以通过以下注册表信息找到英文列表: `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windo NT\CurrentVersion\Perflib\009`。 从 Zabbix agent 版本 4.0.13 和 4.2.7 后开始支持
perf_instance.discovery[object] Windows 性能计数器的对象实例列表，用于底层自动发现.	JSON 对象	**object** - 对象名称 (本地化)	自 Zabbix agent 版本 5.0.1 后开始支持

Key			
perf_instance_en.discovery[object] Windows 性能计数器的对象实例列表，使用英文对象名称发现时，用于底层自动发现。	JSON 对象	**object** - 对象名称 (英文)	自 Zabbix agent 版本 5.0.1 后开始支持。
proc_info[process,<attribute>,<type>] 有关特定进程的各种信息	浮点型	**process** - 进程名 **attribute** - 请求的进程属性 **type** - 表示类型（当存在多个同名进程时有意义）	以下 `attributes` 被支持: *vmsize* (默认) - 进程占用的虚拟内存大小, 单位为千字节 *wkset* - 进程工作集大小 (进程使用的物理内存量)，以千字节为单位 *pf* - 页面错误数 *ktime* - 以毫秒为单位的进程内核时间 *utime* - 以毫秒为单位的进程用户时间 *io_read_b* - 在 I/O 操作中进程读取的字节数 *io_read_op* - 进程读操作数量 *io_write_b* - 在 I/O 操作中进程写入的字节数 *io_write_op* - 进程写操作数量 *io_other_b* - 操作期间，在读操作和写操作之外由进程传送的字节数 *io_other_op* - 通过进程执行的 I/O 操作数量，而不是读取和写入操作 *gdiobj* - 进程使用的 GDI 对象数 *userobj* - 进程使用的 USER 对象数 有效的 `types` 是: *avg* (默认) - 所有进程名为 <process> 的平均值 *min* - 所有进程名为 <process> 中的最小值 *max* - 所有进程名为 <process> 中的最大值 *sum* - 所有进程名为 <process> 的求和 例如: => proc_info[iexplore.exe,wkset,sum] - 获取所有 Internet Explorer 进程占用的物理内存总量 => proc_info[iexplore.exe,pf,avg] - 获取 Internet Explorer 进程的平均页面错误数 请注意，在 64 位系统上，需要 64 位 Zabbix 代理才能正常工作。 注意: *io_**, *gdiobj* 和 *userobj* 属性仅在 Windows 2000 及更高版本的 Windows 上可用，不支持 Windows NT 4.0。
service.discovery Windows 服务列表，用于底层自动发现.	JSON 对象		自 Zabbix agent 版本 3.0 开始支持
service.info[service,<param>]			

Key			
关于服务的信息。	整型- `param` 为 *state*, *startup* 字符串 - `param` 为 *displayname*, *path*, *user* 文本 - `param` 为 *description* 关于 *state*: 0 - 运行, 1 - 暂停, 2 - 开始等待, 3 - 暂停等待, 4 - 继续等待, 5 - 停止等待, 6 - 停止, 7 - 位置, 255 - 没有这样的服务 关于 *startup*: 0 - 自动的, 1 - 自动延迟, 2 - 手动, 3 - 关闭, 4 - 位置, 5 - 自动触发启动, 6 - 自动延迟触发启动, 7 - 手动触发启动	**service** - 真实的服务名称或在 MMC 中显示的名称 **param** - *state* (默认), *displayname*, *path*, *user*, *startup* or *description*	例如: => service.info[SNMPTRAP] - SNMPTRAP 服务的状态 => service.info[SNMP Trap] - 同一的服务状态，但是指定了显示的名称 => service.info[EventLog,startup] - EventLog 服务的启动类型 监控项 service.info[service,state] 和 service.info[service] 将返回相同的信息 注意的是仅当参数为 *state* 时，监控项才返回不存在的服务（255） 该监控项自 Zabbix 3.0.0 后开始支持，它将用于替代废弃的 service_state[service] 监控项。
services[<type>,<state>,<exclude>]			
服务列表。	0 - 如果为空 文本- 以换行符分隔的服务列表	**type** - *all* (默认), *automatic*, *manual* 或 *disabled* **state** - *all* (默认), *stopped*, *started*, *start_pending*, *stop_pending*, *running*, *continue_pending*, *pause_pending* 或 *paused* **exclude** - 从结果中排除服务。排除的服务应该在双引号中列出，用逗号分隔，没有空格	例如: => services[,started] - 列出启动的服务 => services[automatic, stopped] - 应该运行但停止的服务 => services[automatic, stopped, ”service1,service2,service3”] - 应该运行但停止的服务，除了服务 service1, service2 and service3 以外 `exclude` 参数自 Zabbix 1.8.1 后开始支持.
wmi.get[<namespace>,<query>]			
执行 WMI 查询返回的第一个对象	整型, 浮点型, 字符串和文本 (取决于请求)	**namespace** - WMI 命名空间 **query** - WMI 查询返回单个的对象	WMI 查询使用 WQL执行。 例如: => wmi.get[root\cimv2,select status from Win32_DiskDrive where Name like ‘%PHYSICALDRIVE0%’] - 返回第一个物理磁盘的状态信息 这个监控键值自 Zabbix 2.2.0 后开始支持.
wmi.getall[<namespace>,<query>]			

Key			
执行 WMI 查询并返回全部的响应内容。 可以用于低级别自动发现。	JSON 对象	**namespace** - WMI 命名空间 **query** - WMI 查询	WMI 查询使用 WQL执行。 例如: => wmi.getall[root\cimv2,select * from Win32_DiskDrive where Name like '%PHYSICALDRIVE%'] - 返回物理磁盘状态信息 JSON 路径预处理 用于指向返回的 JSON 串中更具体的值 该监控项自 Zabbix 4.4.0 开始支持。
vm.vmemory.size[<type>] 虚拟内存大小(以字节为单位）或占总数的百分比	整型- 字节数 浮点数- 百分比	**type** - 可能的值: *available* (可用的虚拟内存), *pavailable* (可用的虚拟内存百分比), *pused* (使用到虚拟内存百分比), *total* (总虚拟内存大小, 默认), *used* (使用的虚拟内存大小)	例如: => vm.vmemory.size[pavailable] → 可用的虚拟内存百分比 虚拟的内存统计监控基于: * Windows 虚拟内存总大小 (物理内存总大小 + 分页文件大小); * Zabbix agent 可以提交的最大内存量; * 系统或 Zabbix agent 当前的提交内存限制，以较小者为准。 该键值自 Zabbix 3.0.7 和 3.2.3 后开始支持。

监控 Windows 服务

本教程提供了设置 Windows 服务监控的分步说明。假设 Zabbix server 和 agent 已经配置好并且可以运行。

第一步

获取服务名称

您可以通过转到 MMC 服务管理单元并调出服务的属性来获得该名称。在 “常规” 选项卡中，您应该会看到一个名为 “服务名称” 的字段。后面的值是您在设置监控项时将使用的名称。

例如，如果您想监控 “工作站” 服务，那么您的服务可能是：**lanmanworkstation**。

第二步

为监控服务配置监控项 。

该监控项 service.info[service,<param>] 检索有关特定服务的信息。根据您需要的信息, 指定接受以下的 *param* 选项: *displayname*, *state*, *path*, *user*, *startup* or *description*.。如果 *param* 未指定 (service.info[service])，则默认的值为 *state*

返回值的类型取决于选择的 *param*：对于 state 和 startup 是整型，displayname,path 和 user 为字符串类型，description 为文本型

例子:

- 键: service.info[lanmanworkstation]
- 值类型: 数字 (无符号)
- 显示值: 选择 *Windows* 服务状态值映射

两个值映射可用 _Windows service state_ 和 *Windows service startup type* 将数值映射到前端中的文本表示。

Windows 服务的发现

低级别自动发现 提供了一种在计算机上为不同实体自动创建项目、触发器和图形的方法。Zabbix 可以自动开始监控机器上的 Windows 服务，无需知道服务的确切名称，也可以手动创建每个服务的项目。过滤器可用于仅为感兴趣的服务生成实际监控项、触发器和图形。

Zabbix agent 2 特定的监控项键值

Zabbix agent 2 支持 Zabbix agent 在Unix and Windows 上支持的所有监控项键值。这个页面提供了额外的监控项键值的详细信息，或者你也可以按它们所属的插件分组只使用 Zabbix agent 2。

可以查看: Plugins supplied out-of-the-box ::: noteclassic 使用尖括号 **< >** 标记的参数是可选参数，没有使用尖括号标记的参数是必选参数。:::

Ceph

键值			
描述	返回值	参数	注释
ceph.df.details[connString,<user>,<apikey>]			
集群数据在 pool 中的使用和分布。	JSON 对象	**connString** - URI 或会话名称。 **user, password** - Ceph 登录凭据。	
ceph.osd.stats[connString,<user>,<apikey>]			
聚合和每个 OSD 统计信息。	JSON 对象	**connString** - URI 或会话名称。 **user, password** - Ceph 登录凭据。	
ceph.osd.discovery[connString,<user>,<apikey>]			
已发现的 OSD 列表。用于底层自动发现。	JSON 对象	**connString** - URI 或会话名称。 **user, password** - Ceph 登录凭据.	
ceph.osd.dump[connString,<user>,<apikey>]			
OSD 的使用阈值和状态。	JSON 对象	**connString** - URI 或会话名称。 用户、密码 - Ceph 登录凭据。	
ceph.ping[connString,<user>,<apikey>]			
测试是否可以建立与 Ceph 的连接。	0 - 连接断开（如果出现任何错误，包括认证和配置问题） 1 - 连接成功。	**connString** - URI 或会话名称。 **user, password** - Ceph 登录凭据。	
ceph.pool.discovery[connString,<user>,<apikey>]			
已发现的 pool 列表。用于底层自动发现。	JSON 对象	**connString** - URI 或会话名称。 **user, password** - Ceph 登录凭据.	
ceph.status[connString,<user>,<apikey>]			
整个集群的状态。	JSON 对象	**connString** - URI 或会话名称。 **user, password** - Ceph 登录凭据。	

Docker

键值			
描述	返回值	参数	注释
docker.container_info[<ID>,<info>]			
关于容器的底层信息。	ContainerInspect API 调用的输出序列化为 JSON 格式	**ID** - 容器的 ID 或名称。 **info** - 返回的信息量。支持的值：*short*（默认）或 *full*。	必须将 Agent2 用户（'zabbix'）添加到'docker' 组 以获得足够的权限。否则检查将失败。
docker.container_stats[<ID>]			
容器资源使用统计。	ContainerStats API 调用的输出和序列化为 JSON 的 CPU 使用百分比	** ID** - 容器的 ID 或名称。	必须将 Agent2 用户（"zabbix"）添加到"docker"组 以获得足够的权限。否则检查将失败。
docker.containers			
容器列表。	ContainerList API 调用的输出序列化为 JSON 格式	-	必须将 Agent2 用户（"zabbix"）添加到"docker"组 以获得足够的权限。否则检查将失败。
docker.containers.discovery[<选项 >]			

键值			
容器列表。用于底层自动发现。	JSON 对象	**options** - 指定是否应发现所有或仅运行的容器。支持的值： *true* - 返回所有容器； *false* - 仅返回正在运行的容器（默认）。	必须将 Agent2 用户（“zabbix”）添加到“docker”组 以获得足够的权限。否则检查将失败。
docker.data_usage			
有关当前数据使用情况的信息。	SystemDataUsage API 调用的输出序列化为 JSON 格式		必须将 Agent2 用户（“zabbix”）添加到“docker”组 以获得足够的权限。否则检查将失败。
docker.images			
图像列表。	ImageList API 调用的输出序列化为 JSON 格式	-	必须将 Agent2 用户（“zabbix”）添加到“docker”组 以获得足够的权限。否则检查将失败。
docker.images.discovery			
图像列表。用于底层自动发现。	JSON 对象	-	必须将 Agent2 用户（“zabbix”）添加到“docker”组 以获得足够的权限。否则检查将失败。
docker.info			
系统信息。	SystemInfo API 调用的输出序列化为 JSON 格式	-	必须将 Agent2 用户（“zabbix”）添加到“docker”组 以获得足够的权限。否则检查将失败。
docker.ping			
测试 Docker 守护进程是否处于活动状态。	1 - 连接处于活动状态 0 - 连接已断开	-	必须将 Agent2 用户（“zabbix”）添加到“docker”组 以获得足够的权限。否则检查将失败。

Ember+

Key			
Description	**Return value**	**Parameters**	**Comments**
ember.get[<uri>,<path>]			
Returns the result of the required device.	JSON object	**uri** - Ember+ device URI. Default: 127.0.0.1:9998 **path** - OID path to device. Empty by default, returns root collection data.	

Memchached

键值			
描述	回值	参数	注释
memcached.ping[connString,<user>,<password>]			
测试连接是否存活.	1 - 连接存活 0 - 连接断开 (如果出现包括认证和配置问题的错误提示)	**connString** - URI 或者 session name.	
memcached.stats[connString,<user>,<password>,<type>]			
获取 Stats 命令的输出.	JSON - 输出序列化为 JSON 格式	**connString** - URI 或者 session name. **user, password** - Memchached 登陆凭证. **type** - 返回的类型: *items*, *sizes*, *slabs* 或者 *settings* (默认为空, 返回常规统计数据).	

MongoDB

Key			
描述	返回值	参数	注释
mongodb.collection.stats[connString,<user>,<password>,<database>,collection]			
返回一个给定的集合，包含各种存储的统计信息.	JSON 对象	**connString** - URI 或者 session name. **user, password** - MongoDB 登陆凭证. **database** - 数据库名称 (缺省值: admin). **collection** — 集合名称.	
mongodb.collections.discovery[connString,<user>,<password>]			
用于底层自动发现 返回发现的集合列表.	JSON 对象	**connString** - URI 或者 session name. **user, password** - MongoDB 登陆凭证.	
mongodb.collections.usage[connString,<user>,<password>]			
返回集合的使用统计信息.	JSON 对象	**connString** - URI 或者 session name. **user, password** - MongoDB 登陆凭证.	
mongodb.connpool.stats[connString,<user>,<password>]			
返回有关从当前数据库实例到分片集群或副本集的其他成员的开放外向连接的信息。.	JSON 对象	**connString** - URI 或者 session name. **user, password** - MongoDB 登陆凭证.	
mongodb.db.stats[connString,<user>,<password>,<database>]			
返回给定数据库系统状态的统计信息.	JSON 对象	**connString** - URI 或者 session name. **user, password** - MongoDB 登陆凭证. **database** - database name (default: admin).	
mongodb.db.discovery[connString,<user>,<password>]			
用于底层自动发现 返回一个发现的数据库列表.	JSON 对象	**connString** - URI or session name. **user, password** - MongoDB 登陆凭证.	
mongodb.jumbo_chunks.count[connString,<user>,<password>]			
返回 jumbo chunks 的计数.	JSON 对象	**connString** - URI 或者 session name. **user, password** - MongoDB 登陆凭证.	
mongodb.oplog.stats[connString,<user>,<password>]			
使用从 oplog 轮询的数据返回副本集的状态.	JSON 对象	**connString** - URI 或者 session name. **user, password** - MongoDB 登陆凭证.	
mongodb.ping[connString,<user>,<password>]			
测试连接是否存活.	1 - 连接存活 0 - 连接断开 (如果出现包括认证和配置问题的错误提示).	**connString** - URI or session name. **user, password** - MongoDB 登陆凭证.	
mongodb.rs.config[connString,<user>,<password>]			
返回当前副本集的配置.	JSON 对象	**connString** - URI 或者 session name. **user, password** - MongoDB 登陆凭证.	
mongodb.rs.status[connString,<user>,<password>]			
从运行方法的 member 的角度返回副本集状态.	JSON 对象	**connString** - URI 或者 session name. **user, password** - MongoDB 登陆凭证.	

Key		
mongodb.server.status[connString,<user>,<password>]		
返回数据库的状态.	JSON 对象	**connString** - URI 或者 session name. **user, password** - MongoDB 登陆凭证.
mongodb.sh.discovery[connString,<user>,<password>]		
返回集群中已发现的分片的列表.	JSON 对象	**connString** - URI 或者 session name. **user, password** - MongoDB 登陆凭证.

MQTT

Key			
描述	返回值	参数	注释
mqtt.get[<broker_url>,topic,<username>,<password>]			
可以使用通配符订阅所提供代理的一个或多个特定 topic，并等待发布.	依赖 topic 内容. 如果使用通配符返回的 topic 内容是 JSON 格式.	**broker_url** - MQTT broker URL (如果为空, *localhost* 会使用 1883 端口). **topic** - MQTT topic (必选参数). 支持通配符 (+,#) . **username,password** - 身份认证凭据 (如果需要)	该项必须配置为active check ('Zabbix agent (主动)' 监控项类型). TLS 加密证书可通过将其保存到默认位置来使用 (比如 Ubuntu 系统使用 `/etc/ssl/certs/` 目录). 对于 TLS, 使用 `tls://` scheme.

MSSQL

Key			
Description	**Return value**	**Parameters**	**Comments**
mssql.availability.group.get[URI,<user>,<password>]			
Returns availability groups.	JSON object	**URI** - MSSQL server URI (the only supported schema is *sqlserver://*). Embedded credentials will be ignored; **user, password** - username, password to send to protected MSSQL server.	Supported since Zabbix 6.0.27. For more information see the MSSQL plugin readme.
mssql.custom.query[URI,<user>,<password>,queryName,<args...>]			
Returns the result of a custom query.	JSON object	**URI** - MSSQL server URI (the only supported schema is *sqlserver://*). Embedded credentials will be ignored; **user, password** - username, password to send to protected MSSQL server; **queryName** - name of a custom query configured in `Plugins.MSSQL.CustomQueriesDir` without the .sql extension; **args** - one or several comma-separated arguments to pass to a query.	Supported since Zabbix 6.0.27. For more information see the MSSQL plugin readme.
mssql.db.get			
Returns all available MSSQL databases.	JSON object		Supported since Zabbix 6.0.27. For more information see the MSSQL plugin readme.
mssql.job.status.get			
Returns the status of jobs.	JSON object		Supported since Zabbix 6.0.27. For more information see the MSSQL plugin readme.

Key		
mssql.last.backup.get Returns the last backup time for all databases.	JSON object	Supported since Zabbix 6.0.27. For more information see the MSSQL plugin readme.
mssql.local.db.get Returns databases that are participating in an Always On availability group and replica (primary or secondary) and are located on the server that the connection was established to.	JSON object	Supported since Zabbix 6.0.27. For more information see the MSSQL plugin readme.
mssql.mirroring.get Returns mirroring info.	JSON object	Supported since Zabbix 6.0.27. For more information see the MSSQL plugin readme.
mssql.nonlocal.db.get Returns databases that are participating in an Always On availability group and replica (primary or secondary) located on other servers (the database is not local to the SQL Server instance that the connection was established to).	JSON object	Supported since Zabbix 6.0.27. For more information see the MSSQL plugin readme.
mssql.perfcounter.get Returns the performance counters.	JSON object	Supported since Zabbix 6.0.27. For more information see the MSSQL plugin readme.
mssql.ping Ping the database. Test if connection is correctly configured.	1 - alive, 0 - not alive	Supported since Zabbix 6.0.27. For more information see the MSSQL plugin readme.

Key			
mssql.quorum.get Returns the quorum info.	JSON object		Supported since Zabbix 6.0.27. For more information see the MSSQL plugin readme.
mssql.quorum.members.get Returns the quorum members.	JSON object		Supported since Zabbix 6.0.27. For more information see the MSSQL plugin readme.
mssql.replica.get Returns the replicas.	JSON object		Supported since Zabbix 6.0.27. For more information see the MSSQL plugin readme.
mssql.version Returns the MSSQL version.	String		Supported since Zabbix 6.0.27. For more information see the MSSQL plugin readme.

MySQL

键值			
描述	返回值	参数	备注
mysql.db.discovery[connString, <username>, <password>] 用于底层自动发现给出 MySQL 数据库列表.	使用”show databases” SQL 语句在 LLD 查询的结果并且以 JSON 格式输出.	**connString** - URI 或者 session name. **username, password** - MySQL 登陆凭证.	
mysql.db.size[connString, <username>, <password>,dbName] 数据库大小，单位是 byte.	使用”select coalesce(sum(data_length + index_length),0) as size from information_schema.tables where table_schema=?” SQL 语句查询特定的数据大小的结果，单位是 byte.	**connString** - URI 或者 session name. **username, password** - MySQL 登陆凭证. **dbName** - 数据库名称.	
mysql.get_status_variables[connString, <username>, <password>] 全局状态的变量的值.	使用”show global status” SQL 语句查询的结果并且以 JSON 格式输出.	**connString** - URI 或者 session name. **username, password** - MySQL 登陆凭证.	
mysql.ping[connString, <username>, <password>] 测试连接是否存活.	1 - 连接存活 0 - 连接断开 (如果出现包括认证和配置问题的错误提示).	**connString** - URI 或者 session name. **username, password** - MySQL 登陆凭证.	
mysql.replication.discovery[connString, <username>, <password>] 用于底层自动发现列出 MySQL 的 Replication 列表.	使用”show slave status” SQL 语句在 LLD 中查询的结果并且以 JSON 格式输出.	**connString** - URI 或者 session name. **username, password** - MySQL 登陆凭证.	
mysql.replication.get_slave_status[connString, <username>, <password>, <masterHost>]			

键值		
Replication 状态.	使用”show slave status” SQL 语句查询的结果并且以 JSON 格式输出.	**connString** - URI 或者 session name. **username, password** - MySQL 登陆凭证. **masterHost** - Replication 中 master 的主机名.
mysql.version[connString, <username>, <password>]		
MySQL 版本.	以字符串的形式输出 MySQL 实例的版本信息.	**connString** - URI 或者 session name. **username, password** - MySQL 登陆凭证.

Oracle

Key			
描述	返回值	参数	注释
oracle.diskgroups.stats[connString,<user>,<password>,<service>]			
自动存储管理 (ASM) 磁盘组统计信息。	JSON 对象	**connString** - URI 或者 session name. **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。 **diskgroup** - 要查询的 ASM 磁盘组的名称。	
oracle.diskgroups.discovery[connString,<user>,<password>,<service>]			
用于底层自动发现 列出 ASM 磁盘组列表.	JSON 对象	**connString** - URI 或者 session name. **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。	
oracle.archive.info[connString,<user>,<password>,<service>]			
归档日志的统计数据.	JSON 对象	**connString** - URI 或者 session name. **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。 **destination** - 要查询的目的地的名称。	
oracle.cdb.info[connString,<user>,<password>,<service>]			
CDB 信息.	JSON 对象	**connString** - URI 或者 session name. **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。 **database** - 要查询的数据库名称。	
oracle.custom.query[connString,<user>,<password>,<service>, queryName, <args...>]			
自定义查询的结果.	JSON 对象	**connString** - URI 或者会话名称。 **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。 **queryName** — 自定义查询的名称（必须等于不带扩展名的 sql 文件的名称）。 **args...** — 要传递给查询的一个或多个逗号分隔参数。	
oracle.datafiles.stats[connString,<user>,<password>,<service>]			

Key		
数据文件的统计数据. oracle.db.discovery[connString,<user>,<password>,<service>]	JSON 对象	**connString** - URI 或者会话名称。 **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。
用于底层自动发现列出数据库列表. oracle.fra.stats[connString,<user>,<password>,<service>]	JSON 对象	**connString** - URI 或者 session name. **user, password** - Oracle 登陆凭证。 **service** - Oracle 服务名。
快速恢复区 (FRA) 统计信息。 oracle.instance.info[connString,<user>,<password>,<service>]	JSON 对象	**connString** - URI 或者 session name. **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。
实例统计数据. oracle.pdb.info[connString,<user>,<password>,<service>]	JSON 对象	**connString** - URI 或者会话名称。 **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。
可插拔数据库 (PDB) 信息。 oracle.pdb.discovery[connString,<user>,<password>,<service>]	JSON 对象	**connString** - URI 或者会话名称。 **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。
用于底层自动发现列出 PDB 列表. oracle.pga.stats[connString,<user>,<password>,<service>]	JSON 对象	**connString** - URI 或者 session name. **user, password** - Oracle 登陆凭证. **service** - Oracle service name.
PGA 统计数据. oracle.ping[connString,<user>,<password>,<service>]	JSON 对象	**connString** - URI 或者会话名称。 **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。
测试是否可以建立到 Oracle 的连接. oracle.proc.stats[connString,<user>,<password>,<service>]	0 - 连接失败 (如果出现包括认证和配置问题的错误提示) 1 - 连接成功.	**connString** - URI 或者会话名称。 **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。
进程统计数据. oracle.redolog.info[connString,<user>,<password>,<service>]	JSON 对象	**connString** - URI 或者会话名称。 **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。
来自控制文件的日志文件信息. oracle.sga.stats[connString,<user>,<password>,<service>]	JSON 对象	**connString** - URI 或者会话名称。 **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。

Key		
系统全局区 (SGA) 统计信息。	JSON 对象	**connString** - URI 或者会话名称。 **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。
oracle.sessions.stats[connString,<user>,<password>,<service>,<lockMaxTime>]		
会话统计数据.	JSON 对象	**connString** - URI 或者会话名称。 **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。 **lockMaxTime** - 一个长期 Session 被锁定的最长 Session 锁定时间，单位是秒. 缺省值: 600 seconds.
oracle.sys.metrics[connString,<user>,<password>,<service>,<duration>]		
一组系统的指标值.	JSON 对象	**connString** - URI 或者会话名称。 **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。 **duration** - 获取系统指标值的间隔，单位是秒. 可能的值: *60* — 长的间隔 (缺省值), *15* — 短的间隔.
oracle.sys.params[connString,<user>,<password>,<service>]		
一组系统参数值.	JSON 对象	**connString** - URI 或者会话名称。 **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。
oracle.ts.stats[connString,<user>,<password>,<service>]		
表空间统计数据.	JSON 对象	**connString** - URI 或者会话名称。 **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。
oracle.ts.discovery[connString,<user>,<password>,<service>]		
用于底层自动发现列出表空间列表。	JSON 对象	**connString** - URI 或者会话名称。 **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。
oracle.user.info[connString,<user>,<password>,<service>,<username>]		
用户信息.	JSON 对象	**connString** - URI 或者会话名称。 **user, password** - Oracle 登陆凭证. **service** - Oracle 服务名。 **username** - 一个用户名，这些信息是必要的. 用户名不支持小写. 缺省值: 当前用户.

PostgreSQL

Key			
描述	返回值	参数	注释
pgsql.autovacuum.count[uri,<username>,<password>,<dbName>]			

Key			
Autovacuum worker 的数量.	SQL 查询结果.	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	
pgsql.archive[uri,<username>,<password>,<dbName>] 归档文件的信息.	JSON 格式的 SQL 查询结果.	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	返回的数据由相关的监控项处理: **pgsql.archive.count_archived_files** - 成功归档的 WAL 文件的次数. **pgsql.archive.failed_trying_to_archive** - WAL 文件归档失败次数. **pgsql.archive.count_files_to_archive** - 归档的文件数量. **pgsql.archive.size_files_to_archive** - 归档的文件大小.
pgsql.bgwriter[uri,<username>,<password>,<dbName>] 数据库集群的检查点组合数量，按检查点类型细分.	JSON 格式的 SQL 查询结果.	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	返回的数据由相关的监控项处理: **pgsql.bgwriter.buffers_alloc** - 已分配的 buffer 数量. **pgsql.bgwriter.buffers_backend** - 由后端直接写入的 buffer 的数量. **pgsql.bgwriter.maxwritten_clean** - 后台写入器停止清理扫描的次数，因为它写入了太多的 buffer. **pgsql.bgwriter.buffers_backend_fsync** - 后台必须执行自己的 fsync 调用而不是后台写入器的次数. **pgsql.bgwriter.buffers_clean** - 后台写入器写入的 buffer 数量. **pgsql.bgwriter.buffers_checkpoint** - 检查点期间写入的 buffer 数量. **pgsql.bgwriter.checkpoints_timed** - 已执行的预定检查点的数量. **pgsql.bgwriter.checkpoints_req** - 已执行的请求检查点的数量. **pgsql.bgwriter.checkpoint_write_time** - 检查点处理中将文件写入磁盘的部分所花费的总时间，以毫秒为单位. **pgsql.bgwriter.sync_time** - 检查点处理中文件与磁盘同步的部分所花费的总时间.
pgsql.cache.hit[uri,<username>,<password>,<dbName>] PostgreSQL buffer cache 命中率.	百分比形式的 SQL 查询结果.	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	
pgsql.connections[uri,<username>,<password>,<dbName>]			

Key			
连接的类型.	JSON 对象.	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	返回的数据由相关的监控项处理: **pgsql.connections.active** - 后端正在执行查询. **pgsql.connections.fastpath_function_c** - 后端正在执行一个 fast-path 函数. **pgsql.connections.idle** - 后端正在等待一个新的客户端命令. **pgsql.connections.idle_in_transaction** - 后端位于事务中，但当前未执行查询. **pgsql.connections.prepared** - 准备的连接数. **pgsql.connections.total** - 总的连接数. **pgsql.connections.total_pct** - 在 PostgreSQL 服务器的 'max_connections' 设置中，总连接数的百分比. **pgsql.connections.waiting** - 查询中的连接数. **pgsql.connections.idle_in_transaction_** - 后端位于事务中的一条语句导致了错误，但当前未执行查询.
pgsql.dbstat[uri,<username>,<password>,dbName]			

Key			
用于底层自动发现采集每个数据库的统计数据.	JSON 格式的 SQL 查询结果.	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	返回的数据由相关的监控项处理: **pgsql.dbstat.numbackends[”{#DBNA** - 后端读取数据库中的数据文件块所花费的时间, 单位是 milliseconds. **pgsql.dbstat.sum.blk_read_time[”{#D** - 在这个数据库的后端读取数据文件块所花费的时间, 单位是 milliseconds. **pgsql.dbstat.sum.blk_write_time[”{#** - 该数据库的后端写入数据文件块所花费的时间, 单位是 milliseconds. **pgsql.dbstat.sum.checksum_failures[** - 检测到的数据页校验和失败的次数 (或在共享对象上), 如果数据校验和未启用，则为 NULL.(只在 PostgreSQL 12 版本中) **pgsql.dbstat.blks_read.rate[”{#DBNA** - 在这个数据库中读取的磁盘块的数量. **pgsql.dbstat.deadlocks.rate[”{#DBNA** - 该数据库中检测到的死锁数量. **pgsql.dbstat.blks_hit.rate[”{#DBNAM** - 磁盘块已经在缓存中被发现的次数，因此不需要读取 (这只包括在命中 PostgreSQL Pro buffer cache, 而不是操作系统的文件系统 cache 内). **pgsql.dbstat.xact_rollback.rate[”{#DB** - 该数据库中已回滚的事务数. **pgsql.dbstat.xact_commit.rate[”{#DB** - 该数据库中已提交的事务数. **pgsql.dbstat.tup_updated.rate[”{#DB** - 该数据库中由查询更新的行数. **pgsql.dbstat.tup_returned.rate[”{#DB** - 该数据库中查询返回的行数. **pgsql.dbstat.tup_inserted.rate[”{#DB** - 查询在该数据库中插入的行数. **pgsql.dbstat.tup_fetched.rate[”{#DB** - 该数据库中查询获取的行数. **pgsql.dbstat.tup_deleted.rate[”{#DB** - 该数据库中查询删除的行数. **pgsql.dbstat.conflicts.rate[”{#DBNAM** - 由于与此该数据库中的恢复冲突而取消的查询数 (冲突仅发生在备用服务器上). **pgsql.dbstat.temp_files.rate[”{#DBNA** - 由该数据库中的查询创建的临时文件的数量. 无论日志_temp_files 设置和创建临时文件的原因是什么，所有的临时文件都被计算 (例如, 排序或散列). **pgsql.dbstat.temp_bytes.rate[”{#DB** - 该数据库中查询写入临时文件的数据总量。包括来自所有临时文件的数据，无论日志_temp_files 设置和创建临时文件的原因 (例如, 排序或散

Key
pgsql.dbstat.sum[uri,<username>,<password>, <dbName>]

Key			
集群中所有数据库的汇总数据.	JSON 格式的 SQL 查询结果.	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	返回的数据由相关的监控项处理: **pgsql.dbstat.numbackends** - 当前连接到该数据库的后端数量. **pgsql.dbstat.sum.blk_read_time** - 在这个数据库的后端读取数据文件块所花费的时间，单位是 milliseconds. **pgsql.dbstat.sum.blk_write_time** - 该数据库的后端写入数据文件块所花费的时间, 单位是 milliseconds. **pgsql.dbstat.sum.checksum_failures** - 检测到的数据页校验和失败的次数 (或在共享对象上), 如果数据校验和未启用，则为 NULL (只在 PostgreSQL 12 版本中). **pgsql.dbstat.sum.xact_commit** - 该数据库中已提交的事务数. **pgsql.dbstat.sum.conflicts** - 由于与备用服务器上的恢复冲突导致的查询取消的数据库的统计信息. **pgsql.dbstat.sum.deadlocks** - 在该数据库中检测到的死锁数量. **pgsql.dbstat.sum.blks_read** - 在这个数据库中读取的磁盘块的数量. **pgsql.dbstat.sum.blks_hit** - 已经在缓冲区缓存中找到磁盘块的次数，因此不需要读取 (只有在 PostgreSQL Pro 缓存缓存命中被包含). **pgsql.dbstat.sum.temp_bytes** - 数据库中查询写入临时文件的数据总量。包括来自所有临时文件的数据，无论日志_temp_files 设置和创建临时文件的原因 (例如, 排序或散列). **pgsql.dbstat.sum.temp_files** - 由此数据库中的查询创建的临时文件的数量。无论日志_temp_files 设置和创建临时文件的原因是什么，所有的临时文件都被计算在内 (例如, 排序或散列). **pgsql.dbstat.sum.xact_rollback** - 该数据库中回滚事务的数量. **pgsql.dbstat.sum.tup_deleted** - 数据库中查询删除的行数. **pgsql.dbstat.sum.tup_fetched** - 该数据库中查询获取的行数. **pgsql.dbstat.sum.tup_inserted** - 查询在该数据库中插入的行数. **pgsql.dbstat.sum.tup_returned** - 该数据库中查询返回的行数. **pgsql.dbstat.sum.tup_updated** - 该数据库中由查询更新的行数.

Key			
pgsql.db.age[uri,<username>,<password>, dbName] 数据库中最旧的 FrozenXID 的年龄. 用于底层自动发现 发现数据库中最古老的 FrozenXID 的年龄.	SQL query for specific database in transactions.	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	
pgsql.db.bloating_tables[uri,<username>,<password>, <dbName>] 用于底层自动发现每个数据库膨胀的表的数量.	SQL 查询结果.	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	
pgsql.db.discovery[uri,<username>,<password>, <dbName>] 用于底层自动发现列出 PostgreSQL 的数据库列表.	LLD 中的 SQL 查询结果，以 JSON 格式输出.	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	
pgsql.db.size[uri,<username>,<password>, dbName] 用于底层自动发现列出数据库的大小，单位是 byte.	特殊数据库的 SQL 查询结果，单位是 byte.	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	
pgsql.locks[uri,<username>,<password>, <dbName>] 用于底层自动发现发现每个数据库授予的锁的信息.	JSON 格式的 SQL 查询结果.	**uri** - URI or session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	返回的数据由相关的监控项处理: **pgsql.locks.shareupdateexclusive[”{#** - 共享更新排他锁的数量. **pgsql.locks.accessexclusive[”{#DBNA** - 访问排他锁的数量. **pgsql.locks.accessshare[”{#DBNAME** - 访问共享锁的个数. **pgsql.locks.exclusive[”{#DBNAME}”]** - 排他锁的数量. **pgsql.locks.rowexclusive[”{#DBNAME** - 行排他锁的数量. **pgsql.locks.rowshare[”{#DBNAME}”]** - 行共享锁的数量. **pgsql.locks.share[”{#DBNAME}”]** - 共享锁的数量. **pgsql.locks.sharerowexclusive[”{#DB** - 共享行排他锁的数量.
pgsql.oldest.xid[uri,<username>,<password>, <dbName>] 最古老 XID 的年龄.	SQL 查询结果.	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	
pgsql.ping[uri,<username>,<password>, <dbName>] 测试连接是否存活.	1 - 连接存活 0 - 连接断开 (如果出现包括认证和配置问题的错误提示).	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	
pgsql.replication.count[uri,<username>,<password>, <dbName>] standby 服务器的数量.	SQL 查询结果.	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	

Key			
pgsql.replication.recovery_role[uri,<username>,<password>,<dbName>]			
Recovery 状态.	0 - master 模式 1 - recovery 在进行中 (standby 模式)	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	
pgsql.replication.status[uri,<username>,<password>,<dbName>]			
Replication 的状态.	0 - streaming 处于 down 状态 1 - streaming 处于 up 状态 2 - master 模式	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	
pgsql.replication_lag.b[uri,<username>,<password>,<dbName>]			
Replication lag，单位是 byte.	SQL 查询结果，单位是 byte.	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	
pgsql.replication_lag.sec[uri,<username>,<password>,<dbName>]			
Replication lag ，单位是 seconds.	SQL 查询结果，单位是 seconds.	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	
pgsql.uptime[uri,<username>,<password>,<dbName>]			
PostgreSQL 启动时间，单位是 milliseconds.	SQL 查询结果，单位是 milliseconds.	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	
pgsql.wal.stat[uri,<username>,<password>,<dbName>]			
WAL 统计数据.	JSON 格式的 SQL 查询结果.	**uri** - URI 或者 session name. **username, password** - PostgreSQL 登陆凭证. **dbName** - 数据库名称.	返回的数据由相关的监控项处理: **pgsql.wal.count** — WAL 文件的数量. **pgsql.wal.write** - WAL lsn 使用的 (单位是 byte).

Redis

Key			
描述	返回值	参数	注释
redis.config[connString,<password>,<pattern>]			
获取与 Pattern 匹配的 Redis 实例的配置参数.	JSON - 如果使用了 glob-style pattern 返回 JSON 格式 single value - 如果 pattern 不包含任何的通配符返回单个值	**connString** - URI 或者 session name. **password** - Redis 密码. **pattern** - glob-style pattern (* 是缺省值).	
redis.info[connString,<password>,<section>]			
获取 INFO 命令的输出.	JSON - 输出序列化为 JSON 格式	**connString** - URI 或者 session name. **password** - Redis 密码. **section** - section 信息 (*default* 是缺省值).	
redis.ping[connString,<password>]			

Key		
测试连接是否存活.	1 - 连接存活 0 - 连接断开 (如果出现包括认证和配置问题的错误提示)	**connString** - URI 或者 session name. **password** - Redis 密码.
redis.slowlog.count[connString,<password>] Redis 启动后慢日志的数量.	Integer 整数	**connString** - URI 或者 session name. **password** - Redis 密码.

S.M.A.R.T.

Key			
描述	返回值	参数	注释
smart.attribute.discovery 返回 S.M.A.R.T. 设备属性列表.	JSON 对象		将返回以下宏以及对应的值: {#NAME}, {#DISKTYPE}, {#ID}, {#ATTRNAME}, {#THRESH}. 支持 HDD, SSD 和 NVME 设备类型. 设备可以单独或组合在一个 RAID 中. {#NAME} 会有一个 RAID 组件, 比如: {”{#NAME}”: ”/dev/sda cciss,2”}
smart.disk.discovery 返回 S.M.A.R.T. 设备列表.	JSON 对象		将返回以下宏以及对应的值: {#NAME}, {#DISKTYPE}, {#MODEL}, {#SN}. 支持 HDD, SSD 和 NVME 设备类型. 设备可以单独或组合在一个 RAID 中. {#NAME} 会有一个 RAID 组件, 比如: {”{#NAME}”: ”/dev/sda cciss,2”}
smart.disk.get 返回 S.M.A.R.T. 设备的所有可用属性.	JSON 对象		支持 HDD, SSD 和 NVME 设备类型. 设备可以单独或组合在一个 RAID 中. 该数据包含 smartctl 版本和调用参数，以及附加字段: *disk_name* - 包含 RAID 发现所需的附加组件的名称, 比如: {”disk_name”: ”/dev/sda cciss,2”} *disk_type* - 包含磁盘类型 HDD, SSD, 或者 NVME, 比如: {”disk_type”: ”ssd”})

Systemd

Key			
描述	返回值	参数	注释
systemd.unit.get[unit name,<interface>]			

Key			
返回 SystemD Unit 的所有属性.	JSON 对象	**unit name** - Unit 名称 (你可能希望使用监控项原型中的 {#UNIT.NAME} 宏来发现 Unit 名称) **interface** - Unit 接口类型, 可能的值: *Unit* (缺省值), *Service*, *Socket*, *Device*, *Mount*, *Automount*, *Swap*, *Target*, *Path*	这些监控项只支持 Linux 平台. Unit 接口 LoadState, ActiveState 和 UnitFileState 会返回 text 文本和 integer 整形: `"ActiveState":{"state":1,"text":`
systemd.unit.info[unit name,<property>,<interface>] SystemD unit 信息.	String	**unit name** - Unit 名称 (你可能希望使用监控项原型中的 {#UNIT.NAME} 宏来发现 Unit 名称) **property** - Unit property (比如. ActiveState (缺省值), LoadState, Description) **interface** - Unit 接口类型 (比如 Unit (缺省值), Socket, Service)	这个监控项允许从特定的接口类型 dbus API 中检索特定的属性. 这些监控项只支持 Linux 平台. 示例: => sys-temd.unit.info[”{#UNIT.NAME}”] - 采集发现的 SystemD Uint 的活动状态信息 (active, reloading, inactive, failed, activating, deactivating) => sys-temd.unit.info[”{#UNIT.NAME}”,LoadState - 采集所发现的 SystemD Uint 的负载状态信息 => sys-temd.unit.info[mysqld.service,Id] - 检索指定的 Service technical 名称 (mysqld.service) => sys-temd.unit.info[mysqld.service,Description] - 检索指定的 Service 描述 (MySQL Server) => sys-temd.unit.info[mysqld.service,ActiveEnter - 检索 Service 最后一次进入活动状态的时间 (1562565036283903) => sys-temd.unit.info[dbus.socket,NConnections,S - 从这个 socket unit 采集连接数
systemd.unit.discovery[<type>] 用于底层自动发现 列出 SystemD Unit 的列表及其详细信息.	JSON 对象	**type** - 可能的值: *all*, *automount*, *device*, *mount*, *path*, *service* (缺省值), *socket*, *swap*, *target*	这些监控项只支持 Linux 平台.

Web certificate

Key			
描述	返回值	参数	注释
web.certificate.get[hostname,<port>,<address>]			

Key			
验证证书并返回证书详细信息。	JSON 对象	**hostname** - 可以是 IP 或者 DNS. 可能包含 URL scheme (只有 *https* 的时候), path (可能会被忽略), 端口. 如果第一个和第二个参数都提供了端口，则它们的值必须匹配. 如果第三个参数指定了 address，主机名仅用于 SNI 和主机名验证. **port** - 端口 (HTTPS 缺省是 443 端口). **address** - 可以是 IP 或者 DNS. 如果指定了 address 参数，它将被用于连接，第一个参数的主机名将被用于 SNI 和主机验证. 如果第 1 个参数是 IP，第 3 个参数是 DNS，第 1 个参数用于连接，第 3 个参数用于 SNI 和主机验证.	这个监控项不支持这几种情况，在 `host` 中指定的资源不存在或不可用，或者 TLS 握手失败，除了无效的证书之外还有其他错误. 目前不支持 AIA (Authority Information Access) X.509 扩展, CRLs 和 OCSP (包括 OCSP stapling), 证书透明度, 自定义 CA 信任库.

2 SNMP 客户端

概述

您可能希望在打印机、网络交换机、路由器或 UPS 等设备上使用 SNMP 监控，这些设备通常启用 SNMP，并且在这些设备上尝试设置完整的操作系统和 Zabbix 代理是不切实际的。

为了能够在这些设备上检索 SNMP agent 提供的数据，Zabbix 服务器必须初始配置 通过指定 `--with-net-snmp` 支持 SNMP 标志。

SNMP 检查仅通过 UDP 协议执行。

Zabbix 服务器和代理守护进程在单个请求中查询 SNMP 设备的多个值。这会影响所有类型的 SNMP 监控项（常规 SNMP 监控项、具有动态索引的 SNMP 监控项和 SNMP 底层自动发现）并且应该使 SNMP 处理更加高效。请参阅bulk processing 部分，了解有关其内部工作方式的技术细节。使用每个接口的 “使用批量请求” 设置，也可以为无法正确处理批量请求的设备禁用批量请求。

如果 Zabbix 服务器和代理守护进程收到不正确的 SNMP 响应，日志行将类似于以下内容：

·来自主机 “网关” 的 SNMP 响应不包含所有请求的变量绑定

虽然它们没有涵盖所有有问题的情况，但它们对于识别应该禁用批量请求的单个 SNMP 设备很有用。

Zabbix server/proxy 总是会在查询尝试失败后至少重试一次：通过 SNMP 库的重试机制或通过内部bulk processing 机制。

Warning:
如果监控 SNMPv3 设备，请确保 msgAuthoritativeEngineID（也称为 snmpEngineID 或 “引擎 ID”）永远不会被两个设备共享。根据 RFC 2571（第 3.1.1.1 节），每个设备都必须是唯一的。

Warning:
RFC3414 要求 SNMPv3 设备保留其 engineBoots。一些设备不这样做，这导致它们的 SNMP 消息在重新启动后被丢弃为过时的。在这种情况下，需要在服务器/代理上手动清除 SNMP 缓存（通过使用-R snmp_cache_reload）或者需要重新启动服务器/代理.

配置 SNMP 监控

要通过 SNMP 开始监控设备，必须执行以下步骤：

步骤 1

找出您要监控的项目的 SNMP 字符串（或 OID）。

要获取 SNMP 字符串列表，请使用 **snmpwalk** 命令（net-snmp的部分软件应该在 Zabbix 安装时同时安装）或等效工具：

```
shell> snmpwalk -v 2c -c public <主机 IP> 。
```

此处的 “2c” 代表 SNMP 版本，您也可以将其替换为 “1”，表示设备上的 SNMP 版本 v1。

它会返回给你一个 SNMP 字符串及其最后一个值的列表。如果不是，那么 SNMP ’community’ 可能与标准的’public’ 不同，在这种情况下，请找出它是什么。

然后，您可以遍历列表，直到找到要监控的字符串，例如：如果要监视通过端口 3 进入交换机的字节，你将使用此行中的 IF-MIB :: ifInOctets.3 字符串：

```
IF-MIB::ifInOctets.3 = Counter32: 3409739121
```

你现在可以使用 snmpget 命令找出’IF-MIB :: ifInOctets.3’ 的数字 OID：

```
shell> snmpget -v 2c -c public -On 10.62.1.22 IF-MIB::ifInOctets.3
```

请注意，字符串中的最后一个数字是您需要监控的端口号。另请参阅：动态索引。效果如下：

.1.3.6.1.2.1.2.2.1.10.3 = Counter32: 3472126941

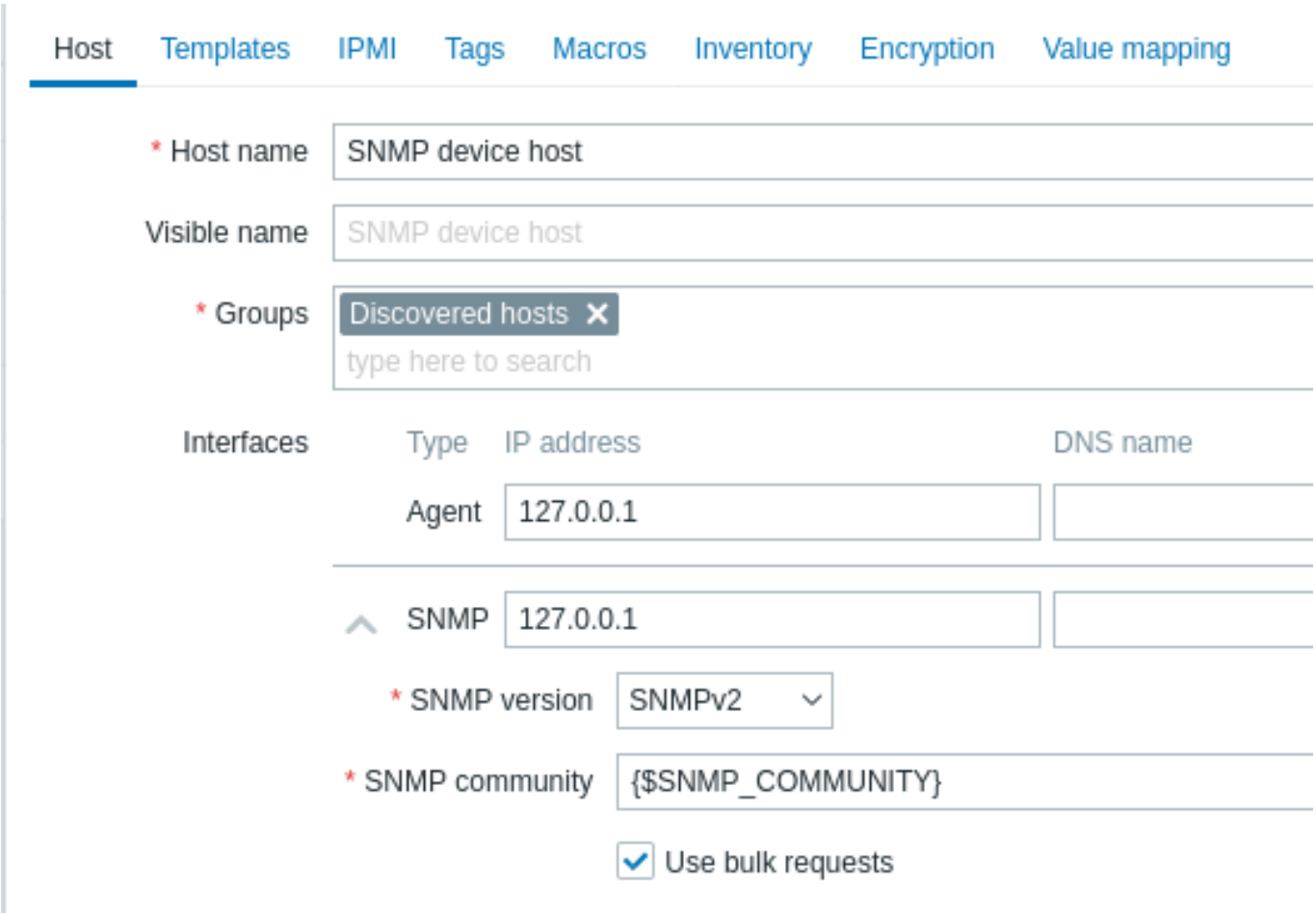
同样，OID 中的最后一个数字是端口号。

Note:
一些最常用的 SNMP OID，Zabbix 将自动转换为数字表示。

在上面的例子中，值类型是 “Counter32” 它在内部对应于 ASN_COUNTER 类型。完整的支持类型包括 ASN_COUNTER, ASN_COUNTER64, ASN_UINTEGER, ASN_UNSIGNED64, ASN_INTEGER, ASN_INTEGER64, ASN_FLOAT, ASN_DOUBLE, ASN_TIMETICKS, ASN_GAUGE, ASN_IPADDRESS, ASN_OCTET_STR 和 ASN_OBJECT_ID (从 2.2.8, 2.4.3 之后). 这些类型大致对应于 snmpget 输出的”Counter32”, ”Counter64”, ”UInteger32”, ”INTEGER”, ”Float”, ”Double”, ”Timeticks”, ”Gauge32”, ”IpAddress”, ”OCTET STRING”, ”OBJECT IDENTIFIER”, 但也有可能显示为”STRING”, ”Hex-STRING”, ”OID” 或者其它, 这取这取决于显示提示的表达方式。

步骤 2

为设备创建一台主机 。



为主机添加 SNMP 接口：

- 输入 IP 地址/DNS 名称和端口号
- 从下拉列表中选择 *SNMP* 版本
- 根据所选 SNMP 版本添加接口凭据：

- SNMPv1、v2 只需要 community 凭据（默认值是"'public'"）
- SNMPv3 需要更具体的选项（见下文）

- 保留 使用批量请求复选框标记以允许批量处理 SNMP 请求

SNMPv3 参数	说明
上下文名称	输入上下文名称以识别 SNMP 子网上的监控项。 自 Zabbix 2.2 起，SNMPv3 监控项支持上下文名称。 在此字段中解析用户宏。
安全名称	输入安全名称。 用户宏在此字段中解析。
安全级别	选择安全级别： **noAuthNoPriv** - 不使用身份验证或隐私协议 **AuthNoPriv** - 使用身份验证协议，不使用隐私协议 **AuthPriv** - 使用身份验证和隐私协议
身份验证协议	选择身份验证协议 - *MD5*、*SHA1*、*SHA224*、*SHA256*、*SHA384* 或 *SHA512*。
身份验证密码	输入身份验证密码。 用户宏在此字段中解析。
隐私协议	选择隐私协议 - *DES*、*AES128*、*AES192*、*AES256*、*AES192C* (Cisco) 或 *AES256C* (Cisco)。
隐私密码	输入隐私密码。 用户宏在此字段中解析。

如果 SNMPv3 凭据（安全名称，验证协议/口令，隐私协议）错误：

- ·Zabbix 从 net-snmp 接收到一个 ERROR，除了错误的 *Privacy passphrase* 在这种情况下 Zabbix 从 net-snmp 接收到一个 TIMEOUT 错误；
- ·（自 Zabbix 6.0.13 起）SNMP 接口可用性将切换为红色（不可用）。

Warning:
在不更改安全名称的情况下，对验证协议、验证口令、隐私协议或私钥的更改只有在手动清除服务器/代理上的缓存（通过使用-R snmp_cache_reload）或重新启动服务器/代理后才会生效。如果安全名称也已更改，则所有参数将立即更新。

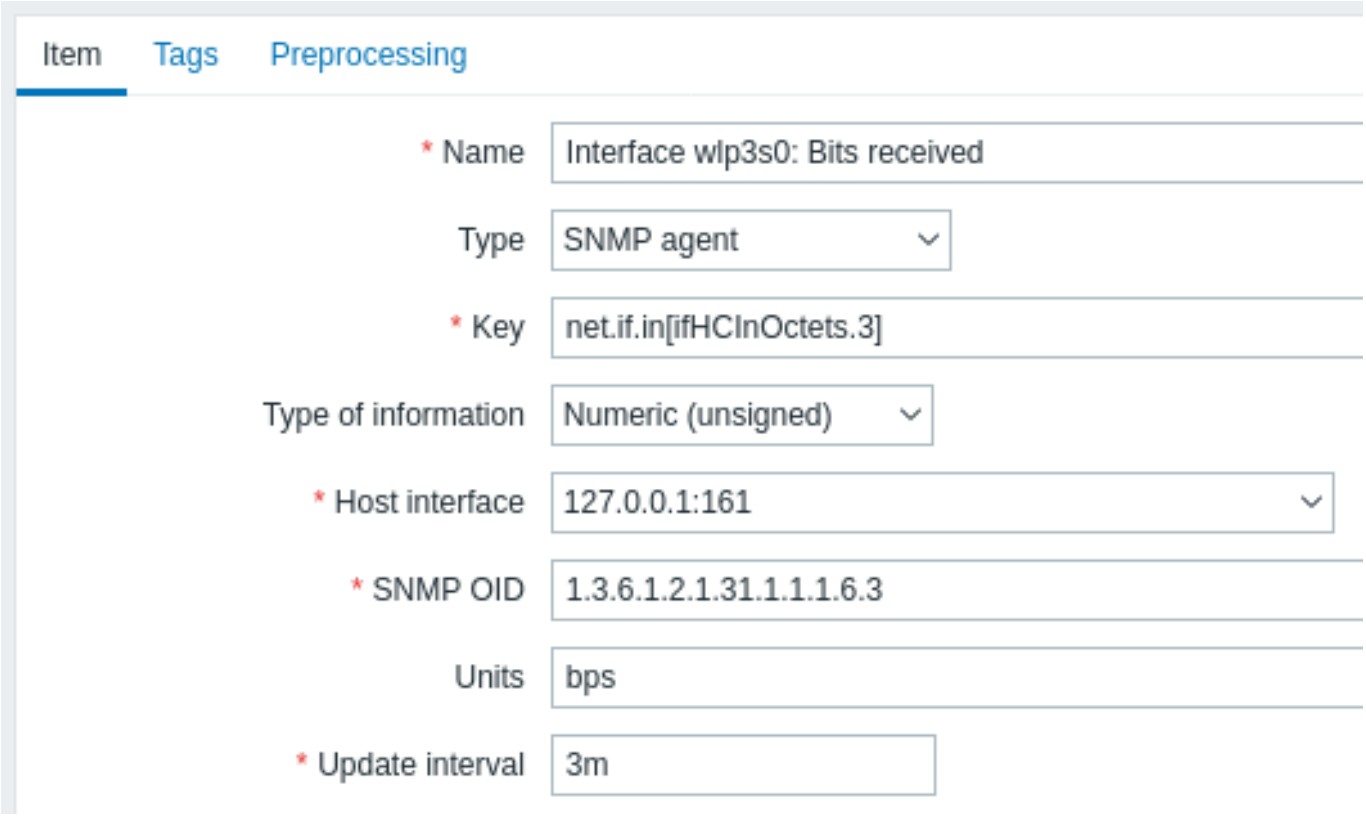
您可以使用 zabbix 提供的任意 SNMP 模板（*SNMP* 设备模板和其他模板），该模板将自动添加监控项。但是，那模板可能与主机不兼容。点击 "添加" 保存主机。

步骤 3

创建一个监控项。

所以现在回到 Zabbix 并点击前面创建的 SNMP 主机的监控项。如果你在创建主机时选择使用模板，你将拥有与主机相关联的 SNMP 监控项列表。我们假设你要使用 snmpwalk 和 snmpget 采集的信息创建监控项，单击创建监控项。在新的监控项表单中：

- 输入监控项" 名称"。
- 将" 类型" 字段更改为"SNMP 客户端"
- 输入 "键值" 为有意义的内容，例如，SNMP-InOctets-Bps。
- 确保 "主机接口" 字段中有你的交换机/路由器
- 将你之前检索到的文本或数字 OID 输入到'SNMP OID' 字段中，例如：.1.3.6.1.2.1.2.2.1.10.3
- 将 "信息类型" 设置为浮点数
- 如果你希望 "更新间隔" 和 "历史数据保留时长" 与默认值不同，请选择一个自定义乘数（如果需要），并输入数值
- 在进程预处理选项卡中，添加 *Change per second* 步骤（重要！，否则您将从 SNMP 设备获取累积值，而不是差异值）。



所有必填字段都标有红色星号。

现在保存监控项，进入监测中 → 最新数据来获取你的 SNMP 数据!

示例 1

通用示例：

参数	描述
OID	1.2.3.45.6.7.8.0 (or .1.2.3.45.6.7.8.0)
键值	用作触发器引用的唯一字符串 > 例如, ”my_param”.

请注意，OID 可以以数字或字符串形式给出。但是，在某些情况下，必须将字符串 OID 转换为数字表示。实用程序 snmpget 可用于此目的：

shell> snmpget -On localhost public enterprises.ucdavis.memory.memTotalSwap.0

示例 2

监控正常运行时间：

参数	说明
OID	MIB::sysUpTime.0
Key	router.uptime
Value type	浮点数
Units	uptime
Multiplier	0.01
Preprocessing step: Custom multiplier	0.01

批量处理的内部工作原理

从 Zabbix server 和 proxy 2.2.3 版本开始查询 SNMP 设备开始在一个请求获取多个值。这会影响几种类型的 SNMP 监控项：

- 常规 SNMP 监控项
- SNMP 监控项带动态索引
- SNMP 低级别自动发现规则

具有相同参数的单个接口上的所有 SNMP 监控项都将同时进行查询。前两种类型的监控项由轮询器分批采集，最多 128 个监控项，而低级发现规则如前所述单独处理。

在较低级别上，执行查询值的操作有两种：获取多个指定对象和游历 OID 树。

对于”getting”，GetRequest-PDU 最多使用 128 个变量绑定。对于”walking”，GetNextRequest-PDU 用于 SNMPv1 和 GetBulkRequest，”max-repetitions” 字段最多 128 个用于 SNMPv2 和 SNMPv3。

因此，每个 SNMP 监控项类型的批量处理的优势如下：

- 常规 SNMP 监控项受益于 “getting” 的改进;
- 具有动态索引的 SNMP 监控项受益于 “getting” 和 “walking” 改进：“getting” 用于索引验证，“walking” 用于构建缓存;
- SNMP 低级发现规则受益于 “walking” 的改进。然而，有一个技术问题，并非所有设备都能够根据请求返回 128 个值。有些总是给出正确的回应，其它情况则会以 “tooBig（1）” 错误做出回应，或者一旦潜在的回应超过了一定的限度，则一律不回应。

为了找到最佳数量的对象来查询给定的设备，Zabbix 使用以下策略。它在请求中查询 “值 1” 时谨慎开始。如果成功，它会在请求中查询 “值 2”。如果再次成功，则查询请求中的 “值 3”，并通过将查询对象的数量乘以 1.5 来继续，导致以下请求大小的顺序：1, 2, 3, 4, 6, 9, 13, 19, 28, 42, 63, 94, 128。

然而，一旦设备拒绝给出适当的响应（例如，对于 42 个变量），Zabbix 会做两件事情。

首先，对于当前批量监控项，它将单个请求中的对象数减半，并查询 21 个变量。如果设备处于活动状态，那么查询应该在绝大多数情况下都有效，因为已知 28 个变量可以工作，21 个变量明显少于此。但是，如果仍然失败，那么 Zabbix 会逐渐回到查询值。如果此时仍然失败，那么设备肯定没有响应，请求大小也不是问题。

Zabbix 为后续批量监控项做的第二件事是它从最后成功的变量数量开始（在我们的示例中为 28），并继续将请求大小递增 1，直到达到限制。例如，假设最大响应大小为 32 个变量，后续请求的大小为 29,30,31,32 和 33。最后一个请求将失败，Zabbix 将永远不再发出大小为 33 的请求。从那时起，Zabbix 将为该设备查询最多 32 个变量。

如果大型查询因此数量的变量而失败，则可能意味着两件事之一。设备用于限制响应大小的确切标准无法知晓，但我们尝试使用变量数来近似。因此，第一种可能性是，在一般情况下，此数量的变量大约是设备的实际响应大小限制：有时响应小于限制，有时它大于限制。第二种可能性是任何方向的 UDP 数据包都丢失了。由于这些原因，如果 Zabbix 查询失败，它会减少最大数量的变量以尝试深入到设备的舒适范围，但（从 2.2.8 开始）最多只能达到两次。

在上面的示例中，如果包含 32 个变量的查询失败，Zabbix 会将计数减少到 31. 如果发生这种情况也会失败，Zabbix 也会将计数减少到 30。但是，Zabbix 不会将计数减少到 30 以下，因为它会假设进一步的失败是由于 UDP 数据包丢失，而不是设备的限制。

但是，如果设备由于其他原因无法正确处理批量请求，并且上述启发式方法不起作用，Zabbix 2.4 版本之后每个接口都有 “使用批量请求” 设置，允许禁用该设备的批量请求。

1 动态索引

概述

虽然你可能会在 SNMP OID 中找到所需的索引号（例如网络接口），但有时你不能完全依赖不变的索引号。

索引号可能是动态的 -它们可能会随时间而改变，因此你的监控项可能会停止工作。

为了避免这种情况，可以定义一个考虑到索引号改变的可能性的 OID。

例如，如果需要检索索引值以匹配 Cisco 设备上的 **GigabitEthernet0/1** 接口的 **ifInOctets** ，请使用以下 OID：

```
ifInOctets["index","ifDescr","GigabitEthernet0/1"]
```

语法

使用 OID 的特殊语法：

<OID of data>[”index”,”<base OID of index>”,”<string to search for>”]

参数	描述
OID of data	主 OID 用于监控项上的数据检索。
index	处理方法。目前支持一种方法： **index** – 搜索索引，并将其附加到数据 OID
base OID of index	该 OID 将被搜索以获取与该字符串对应的索引值。
string to search for	用于在进行查找时与值精确匹配的字符串。区分大小写。

示例

获取 *apache* 进程的内存使用率。

如果使用这种 OID 语法:

```
HOST-RESOURCES-MIB::hrSWRunPerfMem["index","HOST-RESOURCES-MIB::hrSWRunPath", "/usr/sbin/apache2"]
```

索引号将在这里查找:

```
...
HOST-RESOURCES-MIB::hrSWRunPath.5376 = STRING: "/sbin/getty"
HOST-RESOURCES-MIB::hrSWRunPath.5377 = STRING: "/sbin/getty"
HOST-RESOURCES-MIB::hrSWRunPath.5388 = STRING: "/usr/sbin/apache2"
HOST-RESOURCES-MIB::hrSWRunPath.5389 = STRING: "/sbin/sshd"
...
```

现在我们有索引，5388. 索引将附加到此数据 OID，以便接收我们需要的值：

```
HOST-RESOURCES-MIB::hrSWRunPerfMem.5388 = INTEGER: 31468 KBytes
```

索引查找缓存

当请求动态索引项时，Zabbix 检索并缓存基础 OID 下的整个 SNMP 表用于索引（即使早发现了匹配）。这是为了在另一个监控项稍后引用相同的基础 OID -Zabbix 将在缓存中查找索引，而不是再次查询被监视的主机。请注意，每个轮询器进程使用单独的缓存。

在所有随后的值检索操作中，仅验证找到的索引。如果没有改变，则请求结果值；如果已更改，则会重建高速缓存- 遇到已更改索引的每个轮询器再次建立 SNMP 索引表。

2 特殊 OIDs

一些最常用的 SNMP OIDs 会自动转换为 Zabbix 的数字表示。例如，**ifIndex** 被解析成 **1.3.6.1.2.1.2.2.1.1**，**ifIndex.0** 被解析成 **1.3.6.1.2.1.2.2.1.1.0**。

该表包含特殊 OIDs 的列表。

特殊 OID	标识符	描述
ifIndex	1.3.6.1.2.1.2.2.1.1	每个接口的唯一值。
ifDescr	1.3.6.1.2.1.2.2.1.2	包含接口信息的文本字符串。该字符串应包括制造商名称、产品名称和硬件接口版本。
ifType	1.3.6.1.2.1.2.2.1.3	接口类型，根据协议栈中网络层 “下方” 的物理/链路协议进行区分。
ifMtu	1.3.6.1.2.1.2.2.1.4	可以在接口上发送/接收的最大数据报的大小，以八位字节指定。
ifSpeed	1.3.6.1.2.1.2.2.1.5	接口当前带宽的估计值，以比特/秒为单位。
ifPhysAddress	1.3.6.1.2.1.2.2.1.6	协议层的接口地址紧挨着协议栈中网络层的 “下方”。
ifAdminStatus	1.3.6.1.2.1.2.2.1.7	接口的当前管理状态。
ifOperStatus	1.3.6.1.2.1.2.2.1.8	接口的当前操作状态。
ifInOctets	1.3.6.1.2.1.2.2.1.10	接口接收到的八位字节总数，包括帧字符。
ifInUcastPkts	1.3.6.1.2.1.2.2.1.11	传送到更高层协议的子网单播数据包的数量。
ifInNUcastPkts	1.3.6.1.2.1.2.2.1.12	传送到更高层协议的非单播（即子网广播或子网多播）数据包的数量。
ifInDiscards	1.3.6.1.2.1.2.2.1.13	被选择丢弃的入站数据包的数量，即使没有检测到错误以阻止它们被传递到更高层的协议。丢弃此类数据包的一个可能原因是释放缓冲区空间。
ifInErrors	1.3.6.1.2.1.2.2.1.14	包含错误的入站数据包的数量，这些错误阻止它们被传递到更高层的协议。
ifInUnknownProtos	1.3.6.1.2.1.2.2.1.15	通过接口接收到的由于未知或不受支持的协议而被丢弃的数据包的数量。
ifOutOctets	1.3.6.1.2.1.2.2.1.16	从接口传出的八位字节总数，包括帧字符。
ifOutUcastPkts	1.3.6.1.2.1.2.2.1.17	高层协议请求传输的数据包的总数，这些数据包没有发送到该子层的多播或广播地址，包括那些被丢弃或未发送。
ifOutNUcastPkts	1.3.6.1.2.1.2.2.1.18	高层协议请求传输的包的总数，在这个子层被寻址到多播或广播地址，包括那些被丢弃或未被丢弃的包已发送。

特殊 OID	标识符	描述
ifOutDiscards	1.3.6.1.2.1.2.2.1.19	被选择丢弃的出站数据包的数量，即使没有检测到错误以阻止它们被传输。丢弃此类数据包的一个可能原因是释放缓冲区空间。
ifOutErrors	1.3.6.1.2.1.2.2.1.20	由于错误而无法传输的出站数据包数。
ifOutQLen	1.3.6.1.2.1.2.2.1.21	输出包队列的长度（以包为单位）。

3 MIB 文件

介绍

MIB 代表管理信息库。MIB 文件允许您使用 OID（对象标识符）的文本表示。

例如，

`ifHCOutOctets`

是 OID 的文本表示

`1.3.6.1.2.1.31.1.1.1.10`

在使用 Zabbix 监控 SNMP 设备时，您可以使用其中任何一种，但如果您使用文本表示时感觉更舒服，你必须安装 MIB 文件。

安装 MIB 文件

在基于 Debian 的系统上：

```
# apt install snmp-mibs-downloader
# download-mibs
```

在基于 RedHat 的系统上：

```
# dnf install net-snmp-libs
```

启用 MIB 文件

在基于 RedHat 的系统上，默认情况下应该启用 mib 文件。在基于 Debian 的系统，您必须编辑文件 `/etc/snmp/snmp.conf` 和注释掉 `mibs :` 的行

```
# 由于许可原因，snmp 软件包没有 MIB 文件，默认情况下禁用 MIB 加载。如果您添加了 MIB，您可以重新启用
# 通过注释掉以下行来加载它们。
#mibs :
```

测试 MIB 文件

可以使用 `snmpwalk` 实用程序来测试 snmp MIB。如果你没有安装，请使用以下说明。

在基于 Debian 的系统上：

```
# apt install snmp
```

在基于 RedHat 的系统上：

```
# dnf install net-snmp-utils
```

之后，当您查询网络设备时，以下命令一定不会出错：

```
$ snmpwalk -v 2c -c public <NETWORK DEVICE IP> ifInOctets
IF-MIB::ifInOctets.1 = Counter32: 176137634
IF-MIB::ifInOctets.2 = Counter32: 0
IF-MIB::ifInOctets.3 = Counter32: 240375057
IF-MIB::ifInOctets.4 = Counter32: 220893420
[...]
```

在 Zabbix 中使用 MIB

最重要的是要记住 Zabbix 进程不会得到通知对 MIB 文件所做的更改。所以每次改变后你必须重新启动 Zabbix server 或 Zabbix proxy，例如：

```
# service zabbix-server restart
```

重启 zabbix 服务后，对 MIB 文件所做的更改生效。

使用自定义 MIB 文件

每个 GNU/Linux 发行版都有标准的 MIB 文件。但是一些设备供应商提供他们自己的。

假设您想使用CISCO-SMI MIB 文件。这以下说明将下载并安装它：

```
# wget ftp://ftp.cisco.com/pub/mibs/v2/CISCO-SMI.my -P /tmp
# mkdir -p /usr/local/share/snmp/mibs
# grep -q '^mibdirs +/usr/local/share/snmp/mibs' /etc/snmp/snmp.conf 2>/dev/null || echo "mibdirs +/usr/lo
# cp /tmp/CISCO-SMI.my /usr/local/share/snmp/mibs
```

现在你应该可以使用它了。试着翻译一下名字对象 *ciscoProducts* 从 MIB 文件到 OID：

```
# snmptranslate -IR -On CISCO-SMI::ciscoProducts
.1.3.6.1.4.1.9.1
```

如果您收到错误而不是 OID，请确保之前的所有命令没有返回任何错误。

对象名称翻译成功，您可以使用自定义 MIB 文件。请注意查询中使用的 MIB 名称前缀 (*CISCO-SMI::*)。你在使用命令行工具以及 Zabbix 时将需要这个。

Zabbix 使用这个 MIB 文件之前不要忘记重启 Zabbix server/proxy 服务。

> **Attention:**
> 请记住，MIB 文件可以具有依赖关系。也就是说，一个 MIB 可能需要另一个 MIB。为了满足这些您必须安装所有受影响的 MIB 的依赖项文件。

3 SNMP 陷阱

概述

接收 SNMP 陷阱与查询启用 SNMP 的设备相反。

在这种情况下，信息是从支持 SNMP 的设备发送的，并且是被 Zabbix 收集或 “捕获”。

通常，在某些条件更改时会发送陷阱，并且客户端的 162 端口连接到服务器（与用于查询客户端的端口 161 不同）。使用陷阱可能会检测到在查询间隔中发生的一些短期问题，并且可能会遗漏查询数据。

在 Zabbix 中接收 SNMP 陷阱要与 **snmptrapd** 一起使用，以及将 snmp 陷阱传递给 Zabbix 的机制之一，Bash 或 Perl 脚本或 SNMPTT 。

> **Note:**
> 配置 Zabbix 后设置 trap 监控最简单的方法是使用 Bash 脚本方案，因为 Perl 和 SNMPTT 在现代发行版中经常缺失，需要更复杂的配置。但是，该方案使用配置为 `traphandle` 的脚本。为了在生产系统上获得更好的性能，请使用嵌入式 Perl 解决方案（带有 “do perl” 选项的脚本或 SNMPTT）。

接收陷阱的工作流程：

1. `snmptrapd` 收到陷阱
2. `snmptrapd` 将陷阱传递给接收器脚本（Bash、Perl）或 SNMPTT
3. 接收方解析、格式化并将 trap 写入文件
4. Zabbix SNMP trapper 读取并解析 trap 文件
5. 对于每个陷阱，Zabbix 会找到主机接口与接收到的陷阱地址匹配的所有 “SNMP trapper” 监控项。请注意，在匹配过程中，只会使用在主机界面中选择的 “IP” 或 “DNS”。
6. 对于每个找到的项目，将陷阱与 `snmptrap[regexp]` 中的正则表达式进行比较。陷阱设置为 **all** 的匹配的监控项值。如果没有找到匹配的监控项并且有 “snmptrap.fallback” 监控项，陷阱设置为那个值。
7. 如果陷阱没有设置为任何项的值，Zabbix 默认记录不匹配的陷阱。（这是通过管理 → 一般 → 其他中的 “记录不匹配的 SNMP 陷阱” 配置的。）

1 配置 SNMP 陷阱

在前端配置以下字段是特定于该监控项类型：

- 您的主机必须有 SNMP 接口

在 配置 → 主机中，在 主机接口字段中填入正确的 IP 或 DNS 地址设置 SNMP 接口。将每个接收到的陷阱的地址与所有 SNMP 接口的 IP 和 DNS 地址进行比较，以找到相应的主机。

- 配置监控项

在 键值字段中，使用 SNMP 陷阱 key 之一：

键		
描述	返回值	注释
snmptrap[regexp] 捕获与 **regexp** 中指定的正则表达式 匹配的所有 SNMP 陷阱。如果未指定 regexp，则捕获任何陷阱。	SNMP 陷阱	此监控项只能为 SNMP 接口设置。 此监控项键的参数支持用户宏和全局正则表达式。
snmptrap.fallback 捕获所有未被该接口的任何 snmptrap[] 监控项捕获的 SNMP 陷阱。	SNMP 陷阱	只能为 SNMP 接口设置此监控项。

Note:
此处不支持多行正则表达式匹配。

将 信息类型设置为 “Log” 以获取要解析的时间戳。请注意，其他格式（例如 “数字”）也是可以接受的，但可能需要自定义陷阱处理程序。

Note:
要使 SNMP 陷阱监控工作，它必须首先正确设置（见下文）。

2 设置 SNMP trap 监控

配置 Zabbix Server/Proxy

要读取 trap，必须将 Zabbix Server/Proxy 配置为启动 SNMPtrap 进程，并指向由 SNMPTT 或 perl trap 接收器写入的 trap 文件。为此，编辑配置文件 (zabbix_server.conf 或zabbix_proxy.conf)：

StartSNMPTrapper=1 SNMPTrapperFile=[TRAP FILE]

Warning:
如果使用 systemd 参数 **PrivateTmp**，则该文件不太可能在*/tmp* 下工作。

配置 Bash 陷阱接收器

要求：只有 snmptrapd。

Bash 陷阱接收器 脚本 可用于将陷阱直接从 snmptrapd 传递给 Zabbix 服务器。要配置它，请将 `traphandle` 选项添加到 snmptrapd 配置文件 (`snmptrapd.conf`)，参见示例(https://raw.githubusercontent.com/zabbix/zabbix-docker/6.2/Dockerfiles/snmptraps/alpine/conf/etc/snmp/snmptrapd.conf)。

配置 Perl trap 接收器

要求：Perl，使用 --enable-embedded-perl 编译的 Net-SNMP（自 Net-SNMP 5.4 起默认完成）

Perl 陷阱接收器（查找 misc/snmptrap/zabbix_trap_receiver.pl）可用于将陷阱直接从 snmptrapd 传递到 Zabbix 服务器。配置它：

- ·将 Perl 脚本添加到 snmptrapd 配置文件（snmptrapd.conf）中，例如：

  ·perl do ”[FULL PATH TO PERL RECEIVER SCRIPT]”;

- ·配置接收器，例如：

·$SNMPTrapperFile = ’[TRAP FILE]’; ·$DateTimeFormat = ’[DATE TIME FORMAT]’;

Note:
如果没有引用脚本名称，snmptrapd 将拒绝启动并显示消息，类似于这些：
·Regexp modifiers ”/l” and ”/a” are mutually exclusive at (eval 2) line 1, at end of line ·Regexp modifier ”/l” may not appear twice at (eval 2) line 1, at end of line

配置 SNMPTT

首先，snmptrapd 应该配置为使用 SNMPTT。

Note:
为获得最佳性能，SNMPTT 应配置使用 **snmptthandler-embedded** 将陷阱传递给它的守护进程。请参阅 配置 SNMPTT 的说明。

当 SNMPTT 配置为接收陷阱时，配置 `snmptt.ini`：

1. 启用 NET-SNMP 包中的 Perl 模块：
   net_snmp_perl_enable = 1
2. 将陷阱记录到 Zabbix 将读取的陷阱文件中：:
   log_enable = 1
   log_file = [TRAP FILE]
3. 设置日期时间格式：
   ·date_time_format = %H:%M:%S %Y/%m/%d = [DATE TIME FORMAT]

Warning:
“net-snmp-perl” 包已在 RHEL 8.0-8.2 被移除；但在 RHEL 8.3 中重新添加。有关详细信息，请参阅已知问题。

现在格式化陷阱以便 Zabbix 识别它们（编辑 snmptt.conf）：

1. 每个 FORMAT 格式化语句应以 “ZBXTRAP [address]” 开头，其中 [address] 将与 Zabbix 上 SNMP 接口的 IP 和 DNS 地址进行比较。例如：
   EVENT coldStart .1.3.6.1.6.3.1.1.5.1 ”Status Events” Normal
   FORMAT ZBXTRAP $aA Device reinitialized (coldStart)
2. 在下面查看更多关于 SNMP 陷阱格式的信息。

Attention:
不要使用未知的陷阱 - Zabbix 将无法识别它们。可以通过在 snmptt.conf 中定义一般事件来处理未知陷阱：
EVENT general .* ”General event” Normal

SNMP 陷阱格式

所有自定义的 Perl 陷阱接收器和 SNMPTT 陷阱配置必须按以下方式格式化陷阱：

[timestamp] [the trap, part1] ZBXTRAP [address] [the trap, part 2]

在哪里

- ·[timestamp] - 用于日志监控项的时间戳
- ·ZBXTRAP - 头表示一个新的陷阱从这一行开始
- ·[address] - 用于查找此 trap 的主机的 IP 地址

请注意，“ZBXTRAP” 和 “[address]” 将在处理过程中从消息中删除。如果陷阱以其他方式格式化，Zabbix 可能会意外地解析陷阱。

trap 示例：

·11:30:15 2011/07/27 .1.3.6.1.6.3.1.1.5.3 Normal ”Status Events” ·localhost - ZBXTRAP 192.168.1.1 Link down on interface 2. Admin state: 1. Operational state: 2

这将导致 IP=192.168.1.1 的 SNMP 接口出现以下陷阱：

·11:30:15 2011/07/27 .1.3.6.1.6.3.1.1.5.3 Normal ”Status Events” ·localhost - Link down on interface 2. Admin state: 1. Operational state: 2

3 系统要求

大文件支持

Zabbix 为 SNMP trap 文件提供了 “大文件支持”。Zabbix 可以读取的最大文件大小为 2^63（8 EiB）。请注意，文件系统可能会对文件大小添加下限。

日志轮询

Zabbix 不提供任何日志轮询系统（它应由用户处理）。日志轮询应该首先重命名旧文件，然后才能将其删除，以免丢失 trap：

1. Zabbix 在最后一个已知位置打开 trap 文件，并转到步骤 3
2. Zabbix 通过比较 inode 号和定义 trap 文件的 inode 号，检查当前打开的文件是否已经轮换。如果没有打开的文件，Zabbix 将重置最后一个位置并转到步骤 1。
3. Zabbix 从当前打开的文件中读取数据并设置新的位置。
4. 新数据被解析。如果这是轮询的文件，文件将关闭并返回到步骤 2。

5. 如果没有新的数据，Zabbix 等待 1 秒钟后，然后回到步骤 2。

文件系统

由于 Trap 文件的执行，Zabbix 需要文件系统支持 inode 来区分文件（该信息由 stat() 调用获取）。

使用不同 SNMP 协议版本的设置示例

此示例使用 snmptrapd 和 Bash 接收器脚本将陷阱传递到 Zabbix 服务器。

设置：

1. 配置 Zabbix 启动 SNMP trapper 并设置 trap 文件。添加到 `zabbix_server.conf`：

StartSNMPTrapper=1 SNMPTrapperFile=/tmp/my_zabbix_traps.tmp

2. 下载 Bash 脚本到`/usr/sbin/zabbix_trap_handler.sh`：

curl -o /usr/sbin/zabbix_trap_handler.sh https://raw.githubusercontent.com/zabbix/zabbix-docker/6.2/Dockerfiles/snmptraps/alpine/conf/usr/sb

如有必要，调整脚本中的 ZABBIX_TRAPS_FILE 变量。要使用默认值，首先创建父目录：

mkdir -p /var/lib/zabbix/snmptraps

3. 将以下内容添加到 `snmtrapd.conf`(参考工作示例(https://raw.githubusercontent.com/zabbix/zabbix-docker/6.2/Dockerfiles/snmptraps/alpi
.conf))

traphandle default /bin/bash /usr/sbin/zabbix_trap_handler.sh

4. 创建一个 SNMP 测试监控项:

Host SNMP interface IP: 127.0.0.1 Key: `snmptrap["linkup"]` Log time format: yyyyMMdd.hhmmss

5. 接下来我们将为我们选择的 SNMP 协议版本配置 `snmptrapd`，并使用 `snmptrap` 实用程序发送测试陷阱。

SNMPv1，SNMPv2

SNMPv1 和 SNMPv2 协议依赖于 “团体名” 身份验证。在下面的示例中，我们将使用 “secret” 作为团体名。它必须在 SNMP 陷阱发送器上设置为相同的值。

请注意，虽然 SNMPv2 仍在生产环境中广泛使用，但它不提供任何加密和真实发件人身份验证。数据以纯文本形式发送，因此这些协议版本只能在安全环境（例如专用网络）中使用，绝不能在任何公共或第三方网络上使用。

SNMPv1 目前并没有真正被使用，因为它不支持 64 位计数器并且被认为是一个遗留协议。

要启用接受 SNMPv1 或 SNMPv2 陷阱，您应该将以下行添加到 `snmptrapd.conf`。将 “secret” 替换为在 SNMP 陷阱发件人上配置的 SNMP 团体字：

authCommunity log,execute,net secret

接下来我们可以使用 snmptrap 发送测试陷阱。我们将在此示例中使用通用的 “link up”OID：

snmptrap -v 2c -c secret localhost 0 linkUp.0

SNMPv3

SNMPv3 解决了 SNMPv1/v2 安全问题并提供身份验证和加密。您可以使用 SHA 或 MD5 作为身份验证方法，使用 AES 或 DES 作为密码。

要启用接受 SNMPv3，请将以下行添加到 `snmptrapd.conf`：

createUser -e 0x8000000001020304 traptest SHA mypassword AES authuser log,execute traptest

> **Attention:**
> 请注意允许为此用户安全模型执行脚本的 “执行” 关键字。

snmptrap -v 3 -n "" -a SHA -A mypassword -x AES -X mypassword -l authPriv -u traptest -e 0x8000000001020304 localhost 0 linkUp.0

> **Warning:**
> 如果您希望使用 AES192 或 AES256 等强加密方法，从 5.8 版本开始请使用 net-snmp。您可能必须使用 “configure” 选项重新编译它：“--enable-blumenthal-aes”。旧版本的 net-snmp 不支持 AES192/AES256。另请参阅：http://www.net-snmp.org/wiki/index.php/Strong_Authentication_or_Encryption

确认

在这两个示例中，您将在 “/var/lib/zabbix/snmptraps/snmptraps.log” 中看到类似的行：

20220805.102235 ZBXTRAP 127.0.0.1 UDP: [127.0.0.1]:35736->[127.0.0.1]:162 DISMAN-EVENT-MIB::sysUpTimeInstance = 0:0:00:00.00 SNMPv2-MIB::snmpTrapOID.0 = IF-MIB::linkUp.0

Zabbix 中的监控项值将是：

2022-08-05 10:54:43→2022-08-05 10:54:41→ 20220805.105441 UDP: [127.0.0.1]:44262->[127.0.0.1]:162 DISMAN-EVENT-MIB::sysUpTimeInstance = 0:0:00:00.00 SNMPv2-MIB::snmpTrapOID.0 = IF-MIB::linkUp.0

请参阅

- 关于 SNMP traps 的 Zabbix 博客文章
- 配置 snmptrapd（net-snmp 官方文档）
- 配置 snmptrapd 接收 SNMPv3 通知（net-snmp 官方文档）

4 IPMI 检查

概述

你可以在 Zabbix 中监控智能平台管理接口（IPMI）设备的运行状况和可用性。要执行 IPMI 检查，Zabbix 服务器必须首先配置IPMI 支持。

IPMI 是计算机系统的远程” 关闭” 或 “带外” 管理的标准接口。它可以独立于操作系统直接从所谓的 “带外” 管理卡监视硬件状态。

Zabbix IPMI 监控仅适用于支持 IPMI 的设备 (HP iLO, DELL DRAC, IBM RSA, Sun SSP, 等等).

从 Zabbix 3.4 开始，添加了一个新的 IPMI 管理器进程来安排 IPMI 轮询器进行 IPMI 检查。现在，主机始终只由一个 IPMI 轮询器轮询，从而减少了与 BMC 控制器的打开连接数。通过这些更改，可以安全地增加 IPMI 轮询器的数量，而无需担心 BMC 控制器过载。启动至少一个 IPMI 轮询器时，将自动启动 IPMI 管理器进程。

也可以参考 IPMI 检查的已知问题。

配置

主机配置

主机必须配置为处理 IPMI 检查。必须添加 IPMI 接口，必须定义相应的 IP 和端口号，并且必须定义 IPMI 认证参数。

更多细节请查看主机配置 。

服务器配置

默认情况下，Zabbix 服务器未配置为启动任何 IPMI 轮询，因此任何添加的 IPMI 监控项将无法正常工作。要更改此选项，请以 root 身份打开 Zabbix 服务器配置文件 (zabbix_server.conf) 并查找以下行:

```
# StartIPMIPollers=0
```

取消注释，并设置 IPMI Poller 计数为 3，如下:

```
StartIPMIPollers=3
```

保存文件，然后重启 zabbix_server 。

监控项配置

配置主机级别的监控项:

- 选择’IPMI agent’ 作为 类型
- 在主机中输入唯一的监控项键值 (比如, ipmi.fan.rpm)
- 对于 主机接口选择 IPMI 接口 (IP 和端口). 注意 IPMI 接口在主机上必须存在.
- 指定 *IPMI* 传感器 (比如在 Dell Poweredge 型号服务器上的’FAN MOD 1A RPM’) 用于检索对应的指标。默认情况下，应指定传感器 ID。也可以在值之前使用前缀:
  - `id:` - 指定传感器 ID;
  - `name:` - 指定传感器全名。这在传感器只能通过指定全名来区分的情况下非常有用.
- 选择相应的信息类型 (’ 浮点数’ 在这个例子中；对于离散传感器使用’ 数字 (无正负)’), units (类似’rpm’) 和任何其它必需监控项属性

支持的检查

下表描述了 IPMI 代理检查中支持的内置监控项。

监控项键			
▲	描述	返回值	注释
ipmi.get			

监控项键			
	IPMI 传感器相关信息。	JSON 对象	此监控项可用于IPMI 传感器的发现。 从 Zabbix 5.0.0 开始支持.

关于 IPMI 离散传感器的注意事项

要在主机上找到传感器启动 Zabbix 服务器，启用 **DebugLevel=4**。等待几分钟，并在 Zabbix 服务器日志文件中查找传感器发现记录：

```
$ grep 'Added sensor' zabbix_server.log
8358:20130318:111122.170 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:7 id:'CATERR' reading_type:
8358:20130318:111122.170 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:15 id:'CPU Therm Trip' read
8358:20130318:111122.171 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:17 id:'System Event Log' re
8358:20130318:111122.171 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:17 id:'PhysicalSecurity' re
8358:20130318:111122.171 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:14 id:'IPMI Watchdog' readi
8358:20130318:111122.171 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:16 id:'Power Unit Stat' rea
8358:20130318:111122.171 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:16 id:'P1 Therm Ctrl %' rea
8358:20130318:111122.172 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:16 id:'P1 Therm Margin' rea
8358:20130318:111122.172 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:13 id:'System Fan 2' readin
8358:20130318:111122.172 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:13 id:'System Fan 3' readin
8358:20130318:111122.172 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:14 id:'P1 Mem Margin' readi
8358:20130318:111122.172 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:17 id:'Front Panel Temp' re
8358:20130318:111122.173 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:15 id:'Baseboard Temp' read
8358:20130318:111122.173 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:9 id:'BB +5.0V' reading_typ
8358:20130318:111122.173 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:14 id:'BB +3.3V STBY' readi
8358:20130318:111122.173 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:9 id:'BB +3.3V' reading_typ
8358:20130318:111122.173 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:17 id:'BB +1.5V P1 DDR3' re
8358:20130318:111122.173 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:17 id:'BB +1.1V P1 Vccp' re
8358:20130318:111122.174 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:14 id:'BB +1.05V PCH' readi
```

要解码 IPMI 传感器类型和状态，请在 http:*www.intel.com/content/www/us/en/servers/ipmi/ipmi-specifications.html (*在撰写本文时，最新的文件是 *http:*www.intel.com/content/dam/www/public/us/en/documents/product-briefs/second-gen-interface-spec-v2.pdf) 获取 IPMI2.0 规范的副本

开始的第一个参数是 “reading_type”。从规范中使用 “表 42-1，事件/读取类型代码范围” 来解码 “reading_type” 代码。我们示例中的大多数传感器都有 “reading_type：0x1”，这意味着是 “threshold” 传感器。“表 42-3，传感器类型代码” 表示：“类型：0x1” 表示温度传感器；“类型：0x2” -电压传感器；“类型：0x4” -风扇等阈值传感器有时称为 “模拟” 传感器，因为它们测量连续参数，如温度，电压，每分钟转数。

另一个例子 -一个带有 “read_type：0x3” 的传感器。“表 42-1，事件/读取类型代码范围” 表示读取类型代码 02h-0Ch 表示 “通用离散” 传感器。离散传感器具有多达 15 个可能的状态（换句话说-最多 15 个有意义的位）。例如，对于具有 “type：0x7” 的传感器 “CATERR”，“表 42-3，传感器类型代码” 表示此类型 “处理器”，各个位的含义是：00h（最低有效位）-IERR ；01h - 散热等。

在我们的示例中有几个传感器具有 “reading_type：0x6f”。对于这些传感器，“表 42-1，事件/读取类型代码范围” 建议使用 “表 42-3，传感器类型代码” 来解码位的含义。例如，传感器 “Power Unit Stat” 的类型为 “0x9”，表示 “Power Unit”。Offset 00h 表示 “PowerOff / Power Down”。换句话说，如果最低有效位为 1，则服务器断电。为了测试这个位，可以使用 **band** 与掩码 1 的功能。触发表达式可能就像

```
{www.zabbix.com:Power Unit Stat.band(#1,1)}=1
```

警告服务器关机。

关于 IPMI 离散传感器的注意事项

要在主机上找到传感器记录需要在启动 Zabbix 服务器上配置 **DebugLevel=4**。等待几分钟，并在 Zabbix 服务器日志文件中查找传感器发现记录:

```
$ grep 'Added sensor' zabbix_server.log
8358:20130318:111122.170 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:7 id:'CATERR' reading_type:
8358:20130318:111122.170 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:15 id:'CPU Therm Trip' read
8358:20130318:111122.171 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:17 id:'System Event Log' re
8358:20130318:111122.171 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:17 id:'PhysicalSecurity' re
8358:20130318:111122.171 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:14 id:'IPMI Watchdog' readi
8358:20130318:111122.171 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:16 id:'Power Unit Stat' rea
8358:20130318:111122.171 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:16 id:'P1 Therm Ctrl %' rea
8358:20130318:111122.172 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:16 id:'P1 Therm Margin' rea
8358:20130318:111122.172 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:13 id:'System Fan 2' readin
8358:20130318:111122.172 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:13 id:'System Fan 3' readin
```

```
8358:20130318:111122.172 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:14 id:'P1 Mem Margin' readi
8358:20130318:111122.172 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:17 id:'Front Panel Temp' re
8358:20130318:111122.173 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:15 id:'Baseboard Temp' read
8358:20130318:111122.173 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:9 id:'BB +5.0V' reading_typ
8358:20130318:111122.173 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:14 id:'BB +3.3V STBY' readi
8358:20130318:111122.173 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:9 id:'BB +3.3V' reading_typ
8358:20130318:111122.173 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:17 id:'BB +1.5V P1 DDR3' re
8358:20130318:111122.173 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:17 id:'BB +1.1V P1 Vccp' re
8358:20130318:111122.174 Added sensor: host:'192.168.1.12:623' id_type:0 id_sz:14 id:'BB +1.05V PCH' readi
```

要解码 IPMI 传感器类型和状态请在http://www.intel.com/content/www/us/en/servers/ipmi/ipmi-specifications.html 获取 IPMI 2.0 规范的副本 (在撰写本文时最新的文件是http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/second-gen-interface-spec-v2.pdf)。

开始的第一个参数是 “reading_type”。使用规范中的 “Table 42-1，事件/读数类型代码范围” 来解码 “reading_type” 代码。我们示例中的大多数传感器都有 “reading_type:0x1”，这意味着 “阈值” 传感器。“Table 42-3，传感器类型代码” 显示 “类型：0x1” 表示温度传感器，“类型：0x2” - 电压传感器，“类型：0x4” - 风扇等。阈值传感器有时被称为 “模拟” 传感器，因为它们测量连续参数，如温度、电压、每分钟转数。

另一个例子——带有 “reading_type:0x3” 的传感器。“Table 42-1，事件/读取类型代码范围” 表示读取类型代码 02h-0Ch 表示 “通用离散” 传感器。离散传感器有多达 15 种可能的状态（换句话说 - 多达 15 个有意义的位）。例如，对于 “类型：0x7” 的传感器 “CATERR”，“Table 42-3，传感器类型代码” 显示此类型表示 “处理器”，各个位的含义为：00h（最低有效位）- IERR，01h - 热跳闸等

在我们的示例中，几乎没有带有 “reading_type:0x6f” 的传感器。对于这些传感器，“Table 42-1，事件/读数类型代码范围” 建议使用 “Table 42-3，传感器类型代码” 来解码位的含义。例如，传感器 “Power Unit Stat” 的类型为 “type:0x9”，意思是 “Power Unit”。偏移量 00h 表示 “PowerOff/Power Down”。换句话说，如果最低有效位为 1，则服务器关闭。要测试该位，可以使用掩码为 “1” 的 **bitand** function。触发表达式可能像：

```
bitand(last(/www.example.com/Power Unit Stat,#1),1)=1
```

to warn about a server power off.

关于 OpenIPMI-2.0.16,2.0.17,2.0.18 和 2.0.19 中离散传感器名称的注释

OpenIPMI-2.0.16,2.0.17 和 2.0.18 中的离散传感器的名称通常在附近附加一个额外的”0” （或其它数字或字母）。例如，当 `ipmitool` 和 OpenIPMI-2.0.19 将传感器名称显示为”`PhysicalSecurity`” 或”`CATERR`” 时，在 OpenIPMI-2.0.16,2.0.17 和 2.0.18 中，名称分别为”`PhysicalSecurity0`” 或”`CATERR0`”。

当使用 OpenIPMI-2.0.16, 2.0.17 和 2.0.18 配置 IPMI 监控项时，请在 IPMI 代理监控项的 *IPMI* 传感器字段中使用以 “0” 结尾的名称。当你的 Zabbix 服务器升级到使用 OpenIPMI-2.0.19（或更高版本）的新 Linux 发行版时，具有这些 IPMI 离散传感器的监控项将变为 “不支持”。你必须更改其 *IPMI* 传感器名称（最后删除 “0”），并等待一段时间才能再次转为”Enabled”。

关于阈值和离散传感器同时可用的注意事项

一些 IPMI Agent 提供了相同名称的阈值传感器和离散传感器。在 2.2.8 和 2.4.3 之前的 Zabbix 版本中，选择了第一个提供的传感器。从 2.2.8 和 2.4.3 版本以后，偏向于阈值传感器。

连接终止注意事项

如果不执行 IPMI 检查（由于任何原因：所有主机 IPMI 监控项禁用/不支持、主机已禁用/已删除、主机维护等），IPMI 连接将从 Zabbix server 或 proxy 终止 3 到 4 小时，具体时间取决于 Zabbix server/proxy 何时启动。

5 简单检查

概述

简单检查通常用于检查远程未安装 Zabbix agent 的服务。

请注意，简单检查不需要 Zabbix agent，由 Zabbix server 和 Zabbix proxy 来负责处理（例如创建外部连接等）。

简单检查使用示例:

```
net.tcp.service[ftp,,155]
net.tcp.service[http]
net.tcp.service.perf[http,,8080]
net.udp.service.perf[ntp]
```

> **Note:**
> 在简单检查项的配置中，*User name* 和 *Password* 字段用于 Vmware 的监控项；非 VMware 监控项则可忽略。

支持的简单检查

Zabbix 支持的简单检查列表:

另请参考:

- VMware 监控项键值

键值	描述	返回值	参数	注释
icmpping[<target>,<packets>,<interval>,<size>,<timeout>]	通过 ICMP ping 检测主机的可访问性.	0 - ICMP ping 失败 1 - ICMP ping 成功	**target** - 主机 IP 或者域名 **packets** - 数据包数量 **interval** - 连续两个数据包之间的时间间隔，单位是毫秒 **size** - 数据包大小，单位是字节 **timeout** - 超时时间，单位是毫秒	示例: => icmpping[,4] → 4 个数据包中只要一个有返回，那么该项返回值为 1. 另请参考: 默认值表格。
icmppingloss[<target>,<packets>,<interval>,<size>,<timeout>]	发出的数据包丢包的百分比.	浮点数	**target** - 主机 IP 或者域名 **packets** - 数据包数量 **interval** - 连续两个数据包之间的时间间隔，单位是毫秒 **size** - 数据包大小，单位是字节 **timeout** - 超时时间，单位是毫秒	另请参考: 默认值表格。
icmppingsec[<target>,<packets>,<interval>,<size>,<timeout>,<mode>]	ICMP ping 响应时间，单位是秒.	浮点数	**target** - 主机 IP 或者域名 **packets** - 数据包数量 **interval** - 连续两个数据包之间的时间间隔，单位是毫秒 **size** - 数据包大小，单位是字节 **timeout** - 超时时间，单位是毫秒 **mode** - 可能的值: *min*, *max*, *avg* (缺省值)	传输过程中丢失或者超时的数据包不会被计算在内. 如果主机不可用或者超时，这个监控项会返回 0. 如果返回值小于 0.0001 秒，那么这个值会被置为 0.0001 秒. 另请参考: 默认值表格。
net.tcp.service[service,<ip>,<port>]				

键值				
	检查服务是否在运行并且接受 TCP 连接。	0 - 服务停止 1 - 服务正在运行	**service** - 可能的值: *ssh*, *ldap*, *smtp*, *ftp*, *http*, *pop*, *nntp*, *imap*, *tcp*, *https*, *telnet* (另见详情) **ip** - 主机 IP 或者域名 (默认使用主机 IP/DNS **port** - 端口号 (默认使用标准服务端口).	示例: => net.tcp.service[ftp,,45] → 通过检测 TCP 的 45 端口来检测 FTP 服务器的可用性. 请注意：使用 *tcp* 服务必须指定端口. 这些检测可能会在系统守护进程日志文件中产生额外的信息 (通常会记录 SMTP 和 SSH 会话). 目前不支持检测加密协议 (比如 993 端口的 IMAP 或者 995 端口的 POP) . 可以使用 net.tcp.service[tcp,<ip>,port] 作为一种检测方式进行检测. 从 Zabbix 2.0 以后开始支持 *https* 和 *telnet* 服务。
net.tcp.service.perf[service,<ip>,<port>]	检测 TCP 服务性能.	浮点数 0.000000 - 服务停止 seconds - 连接到服务花费的时间，单位是秒	**service** - 可能的值: *ssh*, *ldap*, *smtp*, *ftp*, *http*, *pop*, *nntp*, *imap*, *tcp*, *https*, *telnet* (另见详情) **ip** - IP 地址或者域名 (默认使用 IP/DNS) **port** - 端口号 (默认使用标准服务端口).	示例: => net.tcp.service.perf[ssh] → 可以用来测试 SSH 服务器的初始响应速度. 使用 *tcp* 服务必须指定端口. 目前不支持检测加密协议 (比如 993 端口的 IMAP 或者 995 端口的 POP). 可以使用 net.tcp.service.perf[tcp,<ip>, 作为一种检测方式进行检测. 从 Zabbix 2.0 以后开始支持 *https* 和 *telnet* 服务. 在 Zabbix 2.0 之前调用的是 tcp_perf 。
net.udp.service[service,<ip>,<port>]				

键值					
		检测服务是否正在运行并响应 UDP 请求.	0 - 服务停止 1 - 服务正在运行	**service** - 可能的值: *ntp* (另见详情) **ip** - IP 地址或者域名 (默认使用 IP/DNS) **port** - 端口号 (默认使用标准服务端口).	示例: => net.udp.service[ntp,,45] → 可用于测试 UDP 45 端口上 NTP 服务的可用性. 从 Zabbix 3.0 以后开始支持这个监控项, 但在之前的版本中 *ntp* 服务可用 net.tcp.service[] 监控项来监控。
net.udp.service.perf[service,<ip>,<port>]		检测 UDP 服务的性能.	浮点数 0.000000 - 服务停止 seconds - 等待服务响应的时间，单位是秒	**service** - 可能的值: *ntp* (另见详情) **ip** - IP 地址或者域名 (默认使用 IP/DNS) **port** - 端口号 (默认使用标准服务端口).	示例: => net.udp.service.perf[ntp] → 可用于测试 NTP 服务的响应时间. 从 Zabbix 3.0 以后开始支持这个监控项, 但在之前的版本中 *ntp* 服务可用 net.tcp.service[] 监控项来监控。

·::: noteimportant 对于 LDAP 简单检查（例如 `net.tcp.service[ldap]`）中的 SourceIP 支持，需要 OpenLDAP 2.6.1 或更高版本。从 Zabbix 6.0.1 开始，LDAP 简单检查支持 SourceIP。:::

超时处理

如果简单检查时间超过了 Zabbix server 或是 proxy 配置文件中设置的超时时间，Zabbix 将不会做处理。

ICMP pings

Zabbix 使用外部程序 **fping** 来处理 ICMP pings。

Fping 不包含在 Zabbix 的发行版中，您需要另外安装。如果程序未安装、程序权限错误或者程序路径与配置文件中 ('FpingLocation' 参数) 定义的不匹配，则不会处理 ICMP ping (**icmpping**, **icmppingloss**, **icmppingsec**)。

另请参考: 已知问题

fping 必须可以被 Zabbix 守护进程以 root 身份执行，需要设置 setuid 权限。为设置正确的权限，请以 **root** 身份来执行这些命令:

```
shell> chown root:zabbix /usr/sbin/fping
shell> chmod 4710 /usr/sbin/fping
```

执行以上两句命令，然后检查 **fping** 的权限。在某些情况下，可以通过执行 chmod 命令来重置所有权。

还要检查一下，如果用户 zabbix 属于 zabbix 组，则运行:

```
shell> groups zabbix
```

如果没有添加成功，通过如下命令解决:

```
shell> usermod -a -G zabbix zabbix
```

ICMP 检测参数的默认值、限制和以及数值的描述：

参数	单位	描述	Fping 的参数	Fping 默认设置	Zabbix 允许的限制	
				fping	**Zabbix min**	**max**

参数	单位	描述	Fping 的参数	Fping 默认设置	Zabbix 允许的限制	
packets	计数	发送到目标的请求报文数	-C	3	1	10000
interval	毫秒	连续数据包之间的间隔时间	-p	1000	20	不限制
size	byte	用字节描述的包大小 x86 架构是 56 byte, x86_64 架构是 68 byte	-b	56 or 68	24	65507
timeout	毫秒	**fping v3.x** - 上次发送数据包后等待超时时间，影响 *-C* 参数 **fping v4.x** - 每个包的单独超时时间	-t	**fping v3.x** - 500 **fping v4.x** - 继承 *-p* 参数的值，但是不能超过 2000	50	不限制

此外，Zabbix 使用 fping 参数 *-i interval ms* (不要和上边表格中的监控项参数 *interval* 混淆，它对应 fping 的 *-p* 参数) 和 *-S source IP address* (或者旧版本的 *-I* i)。这些参数通过使用不同的参数组合运行自动检测。Zabbix 尝试检测 fping 允许对*-i* 参数一起使用的最小值(以毫秒为单位)，尝试 3 个值: 0, 1 和 10。第一个成功的值将用于后续的 ICMP 检查. 这个过程是由每个ICMP pinger 进程单独完成的。

从 Zabbix 5.0.4 版本开始，fping 自动检测的参数每小时都会失效，并且在下一次尝试执行 ICMP 检查时再次加载。设置 DebugLevel>=4 可以在服务器或代理日志文件中查看该进程的详细信息。

Warning:
警告: 根据平台和版本的不同，fping 的默认值也会有所不同 - 如有疑问, 请参考 fping 文档。

Zabbix 将三个 *icmpping** 键值中任何一个 IP 地址写入一个临时文件中，然后传递给 **fping**。如果监控项有不同的键值参数，则只有具有相同键值参数的监控项 IP 才会被写入相同的单个文件。
所有写入到单个文件的 IP 地址将被 fping 并行检查，因此 Zabbix icmp pinger 进程将花费固定的时间来处理监控项，而不管文件中的 IP 地址数量。

监控 VMware 的监控项键值

监控项键值

该表格提供了用于监控VMware 环境 的简单检查的详细说明。

键值	描述	返回值	参数	注释
vmware.cl.perfcounter[<url>,<id>,<path>,<instance>]	VMware 集群性能指标计数.	整型	**url** - VMware 服务 URL **id** - VMware 集群 ID **path** - 性能计数路径 **instance** - 性能计数实例	`id` 可以从 vmware.cluster.discovery[] 接收到的作为 {#CLUSTER.ID}
vmware.cluster.discovery[<url>]	发现 VMware 集群.	JSON 对象	**url** - VMware 服务 URL	
vmware.cluster.status[<url>, <name>]				

键值				
	VMware 集群状态.	整型: 0 - gray; 1 - green; 2 - yellow; 3 - red	**url** - VMware 服务 URL **name** - VMware 集群名称	
vmware.datastore.discovery[<url>]				
	发现 VMware datastore.	JSON 对象	**url** - VMware 服务 URL	
vmware.datastore.hv.list[<url>,<datastore>]				
	Datastore hypervisors 列表.	JSON 对象	**url** - VMware 服务 URL **datastore** - datastore 名称	
vmware.datastore.read[<url>,<datastore>,<mode>]				
	从 datastore 中读取操作的时间(单位是 millisecond).	整型 [2]	**url** - VMware 服务 URL **datastore** - datastore 名称 **mode** - latency (默认是平均值), maxlatency (最大值)	
vmware.datastore.size[<url>,<datastore>,<mode>]				
	VMware datastore 存储空间单位是 byte 或者占总的百分比。	整型 - 单位 byte 浮点型 - 单位是百分比	**url** - VMware 服务 URL **datastore** - datastore 名称 **mode** - 可能的值: total (缺省值), free, pfree (空余的百分比), uncommitted	
vmware.datastore.write[<url>,<datastore>,<mode>]				
	从 datastore 中写入操作的时间(单位是 millisecond).	整型 [2]	**url** - VMware 服务 URL **datastore** - datastore 名称 **mode** - latency (默认是平均值), maxlatency (最大值)	
vmware.dc.discovery[<url>]				
	发现 VMware datacenters.	JSON 对象	**url** - VMware 服务 URL	
vmware.eventlog[<url>,<mode>]				
	VMware 事件日志.	Log	**url** - VMware 服务 URL **mode** - *all* (缺省值), *skip* - 跳过对旧数据的处理	这必须是一个 vmware.eventlog[] 每个 URL 监控项键值. 另请参考: example of filtering VMware 事件日志记录.
vmware.fullname[<url>]				
	VMware 服务完整的名称.	字符串	**url** - VMware 服务 URL	
vmware.hv.cluster.name[<url>,<uuid>]				

键值			
	VMware hypervisor 集群名称.	字符串	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称
vmware.hv.cpu.usage[<url>,<uuid>]			
	VMware hypervisor CPU 使用 (Hz).	整型	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称
vmware.hv.cpu.usage.perf[<url>,<uuid>]			
	VMware hypervisor 在此间隔内的处理器使用百分比.	浮点型	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称
vmware.hv.cpu.utilization[<url>,<uuid>]			
	VMware hypervisor 在此间隔内的处理器使用百分比, 取决于电源管理或者 HT.	浮点型	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称
vmware.hv.datacenter.name[<url>,<uuid>]			
	VMware hypervisor datacenter 名称.	字符串	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称
vmware.hv.datastore.discovery[<url>,<uuid>]			
	发现 VMware hypervisor datastore.	JSON 对象	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称
vmware.hv.datastore.list[<url>,<uuid>]			
	VMware hypervisor datastore 列表.	JSON 对象	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称
vmware.hv.datastore.multipath[<url>,<uuid>,<datastore>,<partitionid>]			
	可用的 datastore 路径数量.	整型	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称 **datastore** - datastore 名称 **partitionid** - 从 vmware.hv.datastore.discovery 发现的物理设备内部 ID
vmware.hv.datastore.read[<url>,<uuid>,<datastore>,<mode>]			

键值			
	从 datastore 中读取操作的平均时间 (单位是 millisecond).	整型 [2]	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称 **datastore** - datastore 名称 **mode** - latency (缺省值)
vmware.hv.datastore.size[<url>,<uuid>,<datastore>,<mode>]			
	VMware datastore 空间 (单位是 byte) 或者占总体的百分比.	整型 - 单位 byte 浮点型 - 单位是百分比	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称 **datastore** - datastore 名称 **mode** - 可能的值: total (缺省值), free, pfree (空闲的百分比), uncommitted
vmware.hv.datastore.write[<url>,<uuid>,<datastore>,<mode>]			
	从 datastore 中写入操作的平均时间 (单位是 millisecond).	整型 [2]	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称 **datastore** - datastore 名称 **mode** - latency (缺省值)
vmware.hv.discovery[<url>]			
	发现 VMware hypervisors.	JSON 对象	**url** - VMware 服务 URL
vmware.hv.fullname[<url>,<uuid>]			
	VMware hypervisor 名称.	字符串	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称
vmware.hv.hw.cpu.freq[<url>,<uuid>]			
	VMware hypervisor CPU 频率 (Hz).	整型	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称
vmware.hv.hw.cpu.model[<url>,<uuid>]			
	VMware hypervisor CPU model.	字符串	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称
vmware.hv.hw.cpu.num[<url>,<uuid>]			
	VMware hypervisor CPU 核数.	整型	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称
vmware.hv.hw.cpu.threads[<url>,<uuid>]			

键值				
	VMware hypervisor CPU 线程数.	整型	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称	
vmware.hv.hw.memory[<url>,<uuid>]				
	VMware hypervisor 总内存大小 (byte).	整型	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称	
vmware.hv.hw.model[<url>,<uuid>]				
	VMware hypervisor model.	字符串	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称	
vmware.hv.hw.uuid[<url>,<uuid>]				
	VMware hypervisor BIOS UUID.	字符串	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称	
vmware.hv.hw.vendor[<url>,<uuid>]				
	VMware hypervisor 供应商名称.	字符串	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称	
vmware.hv.maintenance[<url>,<uuid>]				
	VMware hypervisor 维护状态.	整型	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称	Returns '0' - 不在维护中 or '1' - 维护中
vmware.hv.memory.size.ballooned[<url>,<uuid>]				
	VMware hypervisor 膨胀内存大小 (byte).	整型	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称	
vmware.hv.memory.used[<url>,<uuid>]				
	VMware hypervisor 已用内存大小 (byte).	整型	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称	
vmware.hv.network.in[<url>,<uuid>,<mode>]				
	VMware hypervisor 网络入方向统计数据 (byte/s).	整型 [2]	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称 **mode** - bps (缺省值)	
vmware.hv.network.out[<url>,<uuid>,<mode>]				

键值				
	VMware hypervisor 网络出方向统计数据 (byte/s).	整型 [2]	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称 **mode** - bps (缺省值)	
vmware.hv.perfcounter[<url>,<uuid>,<path>,<instance>]				
	VMware hypervisor 性能计数器的值.	整型 [2]	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称 **path** - 性能计数路径 [1] **instance** - 性能计数实例. 默认情况下对聚合值使用空实例	
vmware.hv.power[<url>,<uuid>,<max>]				
	VMware hypervisor 用电功率 (W).	整型	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称 **max** - 允许的最大用电功率	
vmware.hv.sensor.health.state[<url>,<uuid>]				
	VMware hypervisor 健康状态汇总传感器.	整型: 0 - gray; 1 - green; 2 - yellow; 3 - red	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称	
vmware.hv.sensors.get[<url>,<uuid>]				
	VMware hypervisor 硬件供应商状态传感器.	JSON	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称	
vmware.hv.status[<url>,<uuid>]				
	VMware hypervisor 状态.	整型: 0 - gray; 1 - green; 2 - yellow; 3 - red	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称	使用主机系统整体状态属性.
vmware.hv.uptime[<url>,<uuid>]				
	VMware hypervisor 启动时间 (seconds).	整型	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称	
vmware.hv.version[<url>,<uuid>]				
	VMware hypervisor 版本.	字符串	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称	
vmware.hv.vm.num[<url>,<uuid>]				

键值			
	VMware hypervisor 上的虚拟机数量.	整型	**url** - VMware 服务 URL **uuid** - VMware hypervisor 主机名称
vmware.version[<url>]			
	VMware 服务版本.	字符串	**url** - VMware 服务 URL
vmware.vm.cluster.name[<url>,<uuid>]			
	VMware 虚拟机主机名.	字符串	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.cpu.latency[<url>,<uuid>]			
	由于争夺对物理 CPU 的访问，虚拟机无法运行的时间百分比).	浮点型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.cpu.num[<url>,<uuid>]			
	VMware 虚拟机的 CPU 数量.	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.cpu.readiness[<url>,<uuid>,<instance>]			
	虚拟机已就绪，但无法调度到物理 CPU 上运行的时间百分比.	浮点型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名 **instance** - CPU 实例
vmware.vm.cpu.ready[<url>,<uuid>]			
	虚拟机准备就绪，但无法调度到物理 CPU 上运行的时间 (以毫秒为单位). CPU 就绪时间取决于主机上的虚拟机数量及其 CPU 负载 (%).	整型 [2]	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.cpu.swapwait[<url>,<uuid>,<instance>]			
	等待 swap-in 的 CPU 时间百分比.	浮点型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名 **instance** - CPU 实例
vmware.vm.cpu.usage[<url>,<uuid>]			
	VMware 虚拟机 CPU 使用量 (Hz).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.cpu.usage.perf[<url>,<uuid>]			
	VMware 在此间隔内虚拟机 CPU 使用率的百分比.	浮点型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.datacenter.name[<url>,<uuid>]			
	VMware 虚拟机 datacenter 名称.	字符串	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名

键值			
vmware.vm.discovery[<url>]			
	发现 VMware 虚拟机.	JSON 对象	**url** - VMware 服务 URL
vmware.vm.guest.memory.size.swapped[<url>,<uuid>]			
	交换到交换空间的客户物理内存数量 (KB).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.guest.osuptime[<url>,<uuid>]			
	自上次启动操作系统以来所经过的总时间 (单位 second).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.hv.name[<url>,<uuid>]			
	VMware 虚拟机 hypervisor 名称.	字符串	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.memory.size[<url>,<uuid>]			
	VMware 虚拟机总内存 (byte).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.memory.size.ballooned[<url>,<uuid>]			
	VMware 虚拟机膨胀内存 (byte).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.memory.size.compressed[<url>,<uuid>]			
	VMware 虚拟机压缩内存大小 (byte).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.memory.size.consumed[<url>,<uuid>]			
	用于备份客户物理内存页面的主机物理内存使用量 (KB).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.memory.size.private[<url>,<uuid>]			
	VMware 虚拟机 private 内存大小 (byte).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.memory.size.shared[<url>,<uuid>]			
	VMware 虚拟机 shared 内存大小 (byte).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.memory.size.swapped[<url>,<uuid>]			
	VMware 虚拟机 swapped 内存大小 (byte).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.memory.size.usage.guest[<url>,<uuid>]			
	VMware 虚拟机 guest 内存使用量 (byte).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.memory.size.usage.host[<url>,<uuid>]			

键值			
	VMware 虚拟机主机内存使用量 (bytes).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.memory.usage[<url>,<uuid>]			
	主机物理内存消耗百分比.	浮点型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.net.if.discovery[<url>,<uuid>]			
	发现 VMware 虚拟机网络接口.	JSON 对象	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.net.if.in[<url>,<uuid>,<instance>,<mode>]			
	VMware 虚拟机网络接口入方向统计数据 (bytes/packets per second).	整型 [2]	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名 **instance** - 网络接口实例 **mode** - bps (缺省值)/pps - bytes/packets per second
vmware.vm.net.if.out[<url>,<uuid>,<instance>,<mode>]			
	VMware 虚拟机网络接口出方向统计数据 (bytes/packets per second).	整型 [2]	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名 **instance** - 网络接口实例 **mode** - bps (缺省值)/pps - bytes/packets per second
vmware.vm.net.if.usage[<url>,<uuid>,<instance>]			
	VMware 两个间隔之间的虚拟机网络使用率 (综合发送速率和接收速率) (KBps).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名 **instance** - 网络接口实例
vmware.vm.perfcounter[<url>,<uuid>,<path>,<instance>]			
	VMware 虚拟机性能计数器的值.	整型 [2]	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名 **path** - 性能计数的路径 [1] **instance** - 性能计数的实例. 聚合值默认使用空实例
vmware.vm.powerstate[<url>,<uuid>]			
	VMware 虚拟机电源状态.	整型: 0 - 电源关闭; 1 - 电源打开; 2 - 挂起中	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.storage.committed[<url>,<uuid>]			

键值			
	VMware 虚拟机 committed 的存储空间 (bytes).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.storage.readoio[<url>,<uuid>,<instance>]			
	在收集间隔内，对虚拟磁盘的平均未完成读请求数.	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名 **instance** - 磁盘设备实例 (必选)
vmware.vm.storage.totalreadlatency[<url>,<uuid>,<instance>]			
	从虚拟磁盘读取数据所需的平均时间 (millisecond).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名 **instance** - 磁盘设备实例 (必选)
vmware.vm.storage.totalwritelatency[<url>,<uuid>,<instance>]			
	从虚拟磁盘写入数据所需的平均时间 (millisecond).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名 **instance** - 磁盘设备实例 (必选)
vmware.vm.storage.uncommitted[<url>,<uuid>]			
	VMware 虚拟机 uncommitted 存储空间 (byte).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.storage.unshared[<url>,<uuid>]			
	VMware 虚拟机 unshared 存储空间 (byte).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.storage.writeoio[<url>,<uuid>,<instance>]			
	在采集间隔内，对虚拟磁盘未完成的写请求的平均次数.	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名 **instance** - 磁盘设备实例 (必选)
vmware.vm.uptime[<url>,<uuid>]			
	VMware 虚拟机启动时间 (second).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.vfs.dev.discovery[<url>,<uuid>]			
	发现 VMware 虚拟机磁盘.	JSON 对象	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名
vmware.vm.vfs.dev.read[<url>,<uuid>,<instance>,<mode>]			

键值				
	VMware 虚拟机磁盘读取统计数据 (bytes/operations per second).	整型 [2]	**url** - VMware 服务 URL **uuid** - VMware virtual 虚拟机主机名 **instance** - 磁盘实例 **mode** - bps (default)/ops - bytes/operations per second	
vmware.vm.vfs.dev.write[<url>,<uuid>,<instance>,<mode>]				
	VMware 虚拟机磁盘写入统计数据 (bytes/operations per second).	整型 [2]	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名 **instance** - 磁盘实例 **mode** - bps (default)/ops - bytes/operations per second	
vmware.vm.vfs.fs.discovery[<url>,<uuid>]				
	发现 VMware 虚拟机文件系统.	JSON 对象	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名	客户虚拟机需要安装 VMware Tools.
vmware.vm.vfs.fs.size[<url>,<uuid>,<fsname>,<mode>]				
	VMware 虚拟机文件系统统计数据 (bytes/percentages).	整型	**url** - VMware 服务 URL **uuid** - VMware 虚拟机主机名 **fsname** - 文件系统名称 **mode** - total/free/used/pfree/pused	客户虚拟机需要安装 VMware Tools.

脚注

[1] VMware 性能计数器路径具有 `group/counter[rollup]` 格式，其中：

- `group` - 性能计数器组，例如 *cpu*
- `counter` - 性能计数器名称，例如 *usagemhz*
- `rollup` - 性能计数器汇总类型，例如 *average*

所以上面的例子会给出以下计数器路径：`cpu/usagemhz [average]`

性能计数器组描述、计数器名称和汇总类型可以在VMware 文档找到。

另请参阅：为 VMware 创建自定义性能计数器名称。

[2] 这些监控项的值取自 VMware 性能计数器和 VMwarePerfFrequency参数 用于刷新他们在 Zabbix VMware 缓存中的数据：

- vmware.hv.datastore.read
- vmware.hv.datastore.write
- vmware.hv.network.in
- vmware.hv.network.out
- vmware.hv.perfcounter
- vmware.vm.cpu.ready
- vmware.vm.net.if.in
- vmware.vm.net.if.out
- vmware.vm.perfcounter
- vmware.vm.vfs.dev.read
- vmware.vm.vfs.dev.write

更多信息

如何配置 Zabbix 监控 VMware 环境可以参考Virtual machine monitoring 的详细信息。

6 日志文件监控

概述

Zabbix 可用于有/无日志轮询支持的日志文件的集中监控和分析。

当日志文件包含某些字符串或字符串模式时，可以使用通知来警告用户。

要监控日志文件，必须具有如下：

- 主机上已运行 Zabbix agent
- 设置日志监控项

Attention:
被监控日志文件的大小限制取决于大文件支持。

配置

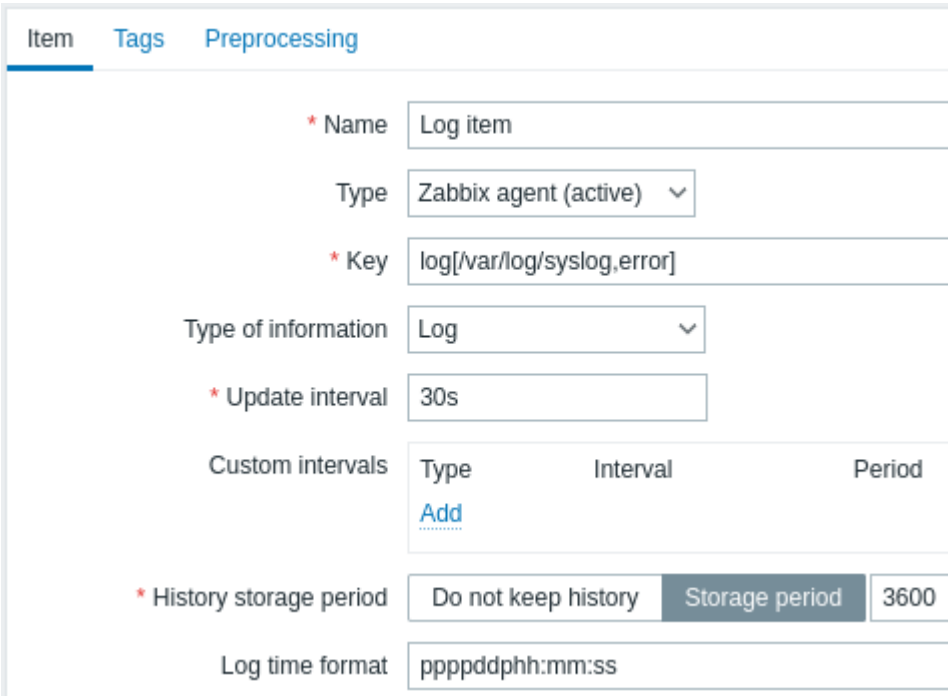
验证代理参数

确保在代理配置文件 中已设置：

- 'Hostname' 参数与前端的主机名一致
- 'ServerActive' 参数中的服务器被指定用于处理主动检查

监控项配置

配置一个日志监控项



所有标有红色星号的为必填字段。

具体日志监控项的输入：

类型	这里选择 **Zabbix agent (主动)**

键值	使用以下项键之一： **log[]** 或 **logrt[]**： 这两个监控项键允许监控日志和过滤日志内容正则表达式的条目(如果存在)。 例如：`log[/var/log/syslog,error]`。确保文件对“zabbix”用户具有读取权限，否则监控项状态将设置为“不支持”。 **log.count[]** 或 **logrt.count[]**： 这两个监控项键只允许返回匹配的行数。 有关使用这些监控项键及其参数的详细信息，请参阅支持的Zabbix agent 监控项 键部分。
信息类型	自动预填： 对于 log[] 或 logrt[] 项 - Log； 对于 log.count[] 或 logrt.count[] items - `Numeric (unsigned)`。 如果可选地使用 `output` 参数，您可以手动选择除 Log 之外的适当类型的信息。 请注意，选择非 Log 类型的信息将导致本地时间戳丢失。
更新间隔 *(秒)*	该参数定义了 Zabbix agent 检查日志文件中任何更改的频率。 将其设置为 1 秒将确保你能尽快的获得新记录。
日志时间格式	在此字段中，您可以选择指定解析日志行的时间戳的模式。 如果留空，则不会解析时间戳。 支持的占位符: * **y**: 年 *(0001-9999)* * **M**: 月 *(01-12)* * **d**: 日 *(01-31)* * **h**: 小时 *(00-23)* * **m**: 分 *(00-59)* * **s**: 秒 *(00-59)* 例如, 从 Zabbix agent 日志文件中查看以下行: ” 23480:20100328:154718.045 Zabbix agent started. Zabbix 1.8.2 (revision 11211).” 它以 PID 的六个字符位置开始，后面跟日期、时间和行的其余部分。 此行的日志时间格式为”ppppppp:yyyyMMdd:hhmmss”。 注意，“p” 和 “：” 字符只是占位符，而且只能是 “yMdhms”

注意事项

- 服务器和代理将监视日志的大小和最后修改时间 (对于 logrt) 的跟踪保存在两个计数器中. 此外:
  - 代理还在内部使用 inode 编号（在 UNIX/GNU/Linux 上）、文件索引（在 Microsoft Windows 上）和前 512 个日志文件字节的 MD5 的求和，以便在日志文件被截断和轮询时改进决策。
  - 在 UNIX/GNU/Linux 系统中，假定日志文件存储的文件系统是报告 inode 号，可以用来跟踪文件。
  - 在 Microsoft Windows 系统上，Zabbix agent 确定日志文件所在的文件系统类型，并使用:
    - 在 NTFS 文件系统上 64 位文件索引。
    - 在 ReFS 文件系统上 (仅从 Microsoft Windows Server 2012 开始支持)128 位文件 ID。
    - 在文件索引发生变化 (例如 FAT32、exFAT) 的文件系统中，当日志文件轮询导致多个日志文件在相同的最后修改时间内出现时，使用回退算法在不确定的条件下采取合理的方法。
  - inode 号，文件索引和 MD5 总和由 Zabbix 代理在内部收集。它们不传输到 Zabbix 服务器，并且在 Zabbix 代理停止时丢失。
  - 不要使用 “touch” 程序修改日志文件的最后修改时间，不要在以后恢复原始名称的情况下复制日志文件（这将更改文件 inode 号）。在这两种情况下，文件将被视为不同的，将从头开始进行分析，这可能会导致重复的告警。
  - 如果 `logrt[]` 监控项有几个匹配的日志文件，并且 Zabbix agent 跟随其中最新的日志文件，并删除了最近的日志文件会出现一条警告消息`"there are no files matching "<regexp mask>" in "<directory>"`。Zabbix 代理将忽略修改时间小于代理看到的 `logrt[]` 被检查监控项的最近修改时间的日志文件。
- 代理从上次停止的点开始读取日志文件。
- 在代理刚刚启动或已收到以前被禁用或不支持的监控项的情况下，已经分析的字节数（大小计数器）和最后修改时间（时间计数器）存储在 Zabbix 数据库中并发送到代理，以确保代理从此开始读取日志文件。但是，如果代理从服务器接受非零大小的计数器, 而 logrt[] 和 logrt.count[] 监控项没有找到，也没找到匹配的文件, 若文件稍后出现，大小计数器将重置为 0，将从头开始分析。
- 每当日志文件变得小于代理已知的日志大小计数器时，计数器将重置为零，代理从开始位置读取日志文件，将时间计数器考虑在内。
- 如果目录中存在多个匹配文件，且最后修改时间相同，则代理会尝试以相同的修改时间对所有日志文件进行正确分析，并避免跳过数据或分析相同的数据两次（尽管有时不能保证）。代理不承担任何特定的日志文件轮询方案。当提供具有相同修改时间的多个日志文件时，代理将以字典顺序降序处理它们。因此，对于某些轮询方案，日志文件将按原始顺序进行分析。对于其它轮询方案，原始

日志文件顺序将不会被执行，这可能导致以更改顺序报告匹配的日志文件记录（如果日志文件的上次修改时间不同，则不会发生问题）。

- Zabbix agent 每 更新间隔秒处理一次日志文件的新记录。
- Zabbix agent 不会每秒发送超过日志文件的 最大值。该限制可防止网络和 CPU 资源的过载，并会覆盖代理配置文件中 **MaxLinesPerSecond** 参数提供的默认值。
- 要找到所需的字符串，Zabbix 将处理比 MaxLinesPerSecond 中设置的多 10 倍的新行。例如，如果 `log[]` 或 `logrt[]` 监控项的 更新间隔为 1 秒，那么在默认情况下，代理将分析不超过 200 条日志文件记录，并在一次检查中向 Zabbix server 发送不超过 20 条的匹配记录。通过在代理配置文件中增加 **MaxLinesPerSecond**，或在监控项键中设置 **maxlines** 参数，一次检查可将分析日志文件记录和发送到 Zabbix 服务器的 1000 条匹配记录增加到 10000 条。如果 更新间隔设置为 2 秒，那么一次检查的限制将被设置为高于更新间隔 1 秒的 2 倍
- 此外，即使其中没有非日志值，日志和日志计数值始终限为代理发送缓冲区大小的 50%。因此，为了在一个连接（而不是几个连接）中发送 **maxlines** 值，代理的缓冲区大小 参数必须至少为 maxlines x 2。
- 在没有日志项的情况下，所有代理缓冲区大小都用于非日志值。当日志值出现时，它们会根据需要替换旧的非日志值，最多可替换为指定的 50%.
- 对于大于 256kB 的日志文件记录，只有前 256kB 与正则表达式匹配，而其余部分将被忽略。但是，如果 Zabbix agent 在处理长记录时停止，则会丢失代理的内部状态，并且在重新启动代理之后，可以对长记录进行不同的分析。
- 特别注意 “ ” 路径分隔符: 如果 file_format 是”file.log”, 则不应该有 “file” 目录, 因为不可能明确地定义 “.” 是转义的还是文件名的第一个符号。
- 仅在文件名中支持 `logrt` 的正则表达式，不支持目录正则表达式匹配。
- 在 UNIX 平台上，如果要找的日志文件的目录不存在，则 `logrt[]` 监控项将变为 NOTSUPPORTED。
- 在 Microsoft Windows 上，如果目录不存在，则监控项将不会变为 NOTSUPPORTED（例如，目录在监控项键中拼写错误）。
- 没有用于 `logrt[]` 监控项的日志文件不会使其 NOTSUPPORTED。读取 `logrt[]` 监控项的日志文件的错误将作为告警记录到 Zabbix agent 日志文件中，但不会使监控项变成 NOTSUPPORTED。
- Zabbix agent 日志文件可以帮助你找出为什么 `log[]` 或 `logrt[]` 监控项会成为 NOTSUPPORTED。Zabbix 可以监视其代理日志文件，除了在 DebugLevel=4 时。

提取正则表达式的匹配部分

有时我们可能只想从目标文件中提取感兴趣的值，而不是在找到正则表达式匹配时返回整行。

自 Zabbix2.2.0 以后，日志监控项能够从匹配的行中提取所需的值。这是在通过 `log` 和 `logrt` 监控项中附加 **output** 参数来实现的。

使用 “output” 参数可以指示我们可能感兴趣的匹配的子组。

例如

```
log[/path/to/the/file,"large result buffer allocation.*Entries: ([0-9]+)",,,,\1]
```

应该可以返回在以下内容中找到的条目数：

```
Fr Feb 07 2014 11:07:36.6690 */ Thread Id 1400 (GLEWF) large result
buffer allocation - /Length: 437136/Entries: 5948/Client Ver: >=10/RPC
ID: 41726453/User: AUser/Form: CFG:ServiceLevelAgreement
```

Zabbix 只返回数字的原因是因为**\1** 定义的，指的是第一个也是唯一的想要的子组：**([0-9]+)**

而且，通过提取和返回数字的能力，该值可用于定义触发器。

使用 maxdelay 参数

日志监控项中的 “maxdelay” 参数允许忽略日志文件中的一些较旧的行，以便在 “maxdelay” 秒内获取最近分析的行。

> **Warning:**
> 指定’maxdelay’>0 可能导致忽略重要的日志文件记录和错过的报警，只有在必要时才使用否则后果自负。

默认情况下，日志监控项将跟踪出现在日志文件中的所有新行。但是，有些应用程序在某些情况下开始在其日志文件中写入大量的消息。例如，如果数据库或 DNS 服务器不可用，则此类应用程序会向日志文件中注入数千条几乎相同错误消息，直到恢复正常为止。默认情况下，所有这些消息将被完全分析，并将匹配的行发送到配置为 `log` 和 `logrt` 监控项的服务器上。

内置的过载保护包括一个可配置的 “maxlines” 参数（保护服务器免受过多传入匹配日志行的影响）和 10*“maxlines” 限制（保护主机 CPU 和 I/O 免于 agent 在一次检查中过载）. 尽管如此，内置保护仍存在两个问题。首先，大量可能不那么有用的消息被报告给服务器并占用数据库空间。其次，由于每秒分析的行数有限，agent 可能会落后于最新的日志记录数小时。很可能，您可能更愿意尽快了解日志文件中的当前情况，而不是花几个小时在旧记录中爬行。

这两个问题的解决方案是使用 “maxdelay” 参数。如果指定’maxdelay’ > 0，则在每次检查处理的字节数时，测量剩余字节数和处理时间。agent 根据这些数字计算估计的延迟——分析日志文件中所有剩余记录需要多少秒。

如果延迟不超过 “maxdelay”，那么 agent 将像往常一样继续分析日志文件。

如果延迟大于 “maxdelay”，那么 agent 将通过 **“跳转”** 到一个新的估计位置来忽略日志文件的一个块，以便在 “maxdelay” 秒内分析剩下的行。

请注意，agent 甚至不会将忽略的行读入缓冲区，而是计算要在文件中跳转的大致位置。

跳过日志文件行的事实记录在代理日志文件中，如下所示：

```
14287:20160602:174344.206 item:"logrt["/home/zabbix32/test[0-9].log",ERROR,,1000,,,120.0]"
logfile:"/home/zabbix32/test1.log" skipping 679858 bytes
(from byte 75653115 to byte 76332973) to meet maxdelay
```

“to byte” 数字是近似的，因为在 “跳转” 之后，agent 将文件中的位置调整到日志行开头，日志行可能在文件中更远或更早。

根据增长速度与分析日志文件的速度的不同，你可能会看到没有” 跳转”、少有或经常” 跳转”、大或小的” 跳转”，甚至每次检查中的” 跳转” 都很小。系统负载和网络延迟的波动也会影响延迟的计算，因此” 跳转” 可以跟上 “maxdelay” 参数。

不推荐设置’maxdelay’ <’update interval’（这可能会导致频繁的”jumps”）

处理 “copytruncate” 日志文件轮换的注意事项

带有 `copytruncate` 选项的 `logrt` 假定不同的日志文件有不同的记录 (至少它们的时间戳不同)，因此初始块的 MD5(最多 512 字节) 将不同。两个具有相同的 MD5 初始块和的文件意味着其中一个是原始块，另一个是副本。

使用 `copytruncate` 选项的 `logrt` 将努力正确处理日志文件副本，而不报告副本。但是，与 logrt[] 监控项更新间隔相比，生成具有相同时间戳的多个日志文件副本、日志文件轮询频率更高、不建议频繁重新启动代理。代理试图合理地处理所有这些情况，但是在所有情况下都不能保证良好的结果。

关于持久文件的 log*[] 监控项的注释

持久文件的目的

当 Zabbix agent 启动，它会从 Zabbix server 或 proxy 接收活动检查列表。对于 log*[] 指标，它接收处理后的日志大小和修改时间，以查找从哪里开始日志文件监控。根据文件系统报告的实际日志文件大小和修改时间，代理决定从已处理的日志大小继续监控日志文件还是从头重新分析日志文件。

正在运行的代理维护大的属性集，用于在检查之间跟踪所有受监视的日志文件。当代理停止时，这种内存状态会丢失。

新的可选参数 **persistent_dir** 指定了一个目录，用于在文件中存储 log[], log.count[], logrt[] or logrt.count[] 监控项的状态。Zabbix agent 重启后，从持久化文件中恢复日志监控项的状态。

主要的用户场景是监控位于镜像文件系统中的日志文件，直到某个时刻，日志文件被写入两个镜像。然后镜像被分割。在活动副本上，日志文件仍在增长，获取新记录。Zabbix agent 对其进行分析并将处理后的日志大小和修改时间发送到服务端。在被动副本上，日志文件保持不变，远远落后于活动副本。稍后操作系统和 Zabbix agent 从被动副本重新启动。Zabbix agent 从服务器接收到的已处理日志大小和修改时间可能对被动副本的情况无效。要从文件系统镜像拆分时代理停止的位置继续进行日志文件监控，代理会从持久文件恢复其状态。

持久文件的代理操作

在启动时 Zabbix agent 对持久文件一无所知。只有在收到来自 Zabbix server (proxy) 的活动检查列表后，代理才会看到某些日志项应该由指定目录下的持久文件支持。

在代理操作期间，持久文件被打开以进行写入（使用 fopen(filename, ”w”)）并用最新数据覆盖。如果同时发生覆盖和文件系统镜像拆分，丢失持久文件数据的机会非常小，无需特殊处理。写入持久文件后不会强制同步到存储介质（不调用 (fsync())。

在成功向 Zabbix server 报告匹配的日志文件记录或元数据 (处理的日志大小和修改时间) 后，使用最新数据覆盖。这可能与每个监控项检查日志文件是否不断变化一样频繁。

代理关闭期间没有特殊操作。

在收到活动检查列表后，代理会将过时的持久文件标记为删除。如果出现以下情况，则持久文件将过时：1) 不再监视相应的日志项，2) 使用与以前不同的 **persistent_dir** 位置重新配置日志项

删除会延迟 24 小时完成，因为处于 NOTSUPPORTED 状态的日志文件不包含在活动检查列表中，但它们可能会在稍后变为 SUPPORTED，并且它们的持久文件将很有用。

如果代理在 24 小时到期之前停止，则不会删除过时的文件，因为 Zabbix agent 不再从 Zabbix server 获取有关其位置的信息。

在代理停止时，将日志项 **persistent_dir** 重新配置回旧的 **persistent_dir** 位置，用户没有删除就的持久文件 - 将导致从旧的持久化文件中恢复代理状态，从而导致丢失消息或错误告警。

持久文件的命名和位置

Zabbix agent 通过他们的键来区分活动检查。例如：logrt[/home/zabbix/test.log] 和 logrt[/home/zabbix/test.log,] 是不同的监控项。将前端的项 logrt[/home/zabbix/test.log„,10] 修改为 logrt[/home/zabbix/test.log„,20] 会导致 logrt[/home/zabbix/test.log„,10] 项从代理的活动检查清单中删除并创建 logrt[/home/zabbix/test.log„,20] 监控项 (某些属性在前端/服务器中进行修改，而不是在代理中)。

文件名由监控项密钥的 MD5 和加上监控项密钥长度组成，以减少冲突的可能性。例如，logrt[/home/zabbix50/test.log„„„„/home/zabbix50/agent_pr
监控项的状态将保存在持久文件 c963ade4008054813bbc0a650bb8e09266 中。

多个日志项可以使用相同的值 **persistent_dir**。

persistent_dir 是通过考虑特定文件系统布局、挂载点和挂载选项以及存储镜像配置来指定的 - 持久文件应该与受监控的日志文件位于同一镜像文件系统上。

如果 **persistent_dir** 目录无法创建或不存在，或者 Zabbix agent 的访问权限不允许创建/写入/读取/删除文件，则日志项将变为 NOTSUPPORTED。

如果在代理操作期间删除了对持久存储文件的访问权限或发生其他错误（例如磁盘已满），则错误将记录到代理日志文件中，但日志项不会变为 NOTSUPPORTED。

I/O 负载

每一批数据 (包含项目的数据) 成功发送到服务器后，项目的持久文件就会更新，例如, 默认的'BufferSize' 是 100。如果一个日志项找到 70 条匹配记录，那么前 50 条记录将分批发送，持久化文件将被更新，然后剩余 20 条记录将被发送 (可能会有一些延迟，更多数据积累) 在第二批中，并且将再次更新持久文件。

代理和服务器之间通信失败时的操作

来自 `log[]` 和 `logrt[]` 监控项的每个匹配行以及每个 `log.count[]` 和 `logrt.count[]` 监控项检查的结果都需要代理发送缓冲区中指定的 50 % 区域中的空闲时隙。缓冲区元素定期发送到服务器（或代理服务器），缓冲区可以再次释放。

虽然代理发送缓冲区中的指定日志区域中有空闲时隙，并且代理和服务器（或代理服务器）之间的通信失败，但是日志监控结果在发送缓冲区中累积。这有助于缓解短暂的通信故障。

在较长的通信失败期间，所有日志槽都被占用，并采取以下操作：

- `log[]` 和 `logrt[]` 监控项检查已停止。当通信恢复并且缓冲器中的空闲插槽可用时，从先前的位置恢复检查。若没有匹配的行丢失，稍后再报告.
- 如果 `maxdelay = 0`（默认），则 `log.count[]` 和 `logrt.count[]` 监控被停止。这种行为类似于上述的 `log[]` 和 `logrt[]` 监控项。请注意，这可能会影响 `log.count[]]` 和 `logrt.count[]` 结果：例如，一次检查计算出日志文件中有 100 个匹配行，但是由于缓冲区中没有空闲插槽，因此停止检查。当通信恢复时，代理将计数相同的 100 条匹配行，还有 70 条新的匹配行。代理会发送 count=170，就像它们在一次检查中发现的一样。
- `log.count[]` 和 `logrt.count[]]` 检查与 `maxdelay > 0`：如果在检查期间没有 "跳转"，则行为类似于上述。如果在日志文件行上发生 "跳转"，则保留 "跳转" 之后的位置，同时计算结果被丢弃。因此，即使在通信失败的情况下，代理也试图跟上日志文件的增长速度。

Handling of regular expression compilation and runtime errors

If a regular expression used in `log[]`, `logrt[]`, `log.count[]` or `logrt.count[]` item cannot be compiled by PCRE or PCRE2 library then the item goes into NOTSUPPORTED state with an error message. To continue monitoring the log item, the regular expression should be fixed.

If the regular expression compiles successfully, but fails at runtime (on some or on all log records), then the log item remains supported and monitoring continues. The runtime error is logged in the Zabbix agent log file (without the log file record).

Note that the logging of regular expression runtime errors is supported since Zabbix 6.0.21.

The logging rate is limited to one runtime error per check to allow Zabbix agent to monitor its own log file. For example, if 10 records are analyzed and 3 records fail with a regexp runtime error, one record is produced in the agent log.

Exception: if MaxLinesPerSecond=1 and update interval=1 (only 1 record is allowed to analyze per check) then regexp runtime errors are not logged.

zabbix_agentd logs the item key in case of a runtime error, zabbix_agent2 logs the item ID to help identify which log item has runtime errors. It is recommended to redesign the regular expression in case of runtime errors.

7 可计算监控项

概述

使用可计算监控项，可以根据其它监控项的值计算。

计算可以同时使用：

- 单个监控项的单一值
- 用于选择多个监控项进行聚合的复杂过滤器 (有关详细信息，参阅聚合计算)

因此，可计算监控项是创建虚拟数据源的一种方式。所有计算仅由 Zabbix server 完成。这些值是根据所使用的算术表达式定期计算的。

结果数据与任何其它监控项一样存储在 Zabbix 数据库中；历史和趋势值都被存储并且可以生成图形。

> **Note:**
> 如果计算结果是浮点值，如果计算的信息项类型为 *Numeric (unsigned)*，则将其修剪为整数。

可计算监控项与触发器表达式共享它们的语法。在可计算监控项中允许与字符串进行比较。可计算监控项可以由宏或与任何其它项类型相同的其它实体引用。

要使用可计算监控项，请选择项目类型 **Calculated**。

可配置字段

key 是唯一的监控项标识符 (每个主机)。您可以使用支持的符号创建任何键名。

应在 公式字段中输入计算定义。公式和密钥之间几乎没有任何联系。公式中不以任何方式使用关键参数。

一个简单公式的语法是：

```
function(/host/key,<parameter1>,<parameter2>,...)
```

这里:

function	支持的函数之一: last, min, max, avg, count, 等
host	用于计算的主机监控项 当前主机可以省略 (如下所示 `function(//key,parameter,...)`).
key	用于计算的监控项的键
parameter(s)	函数的参数，如果需要。

Attention:
如果用于引用函数参数或常量，公式中的用户宏将被扩展，如果引用函数、主机名、监控项键、监控项键参数或运算符，则不会扩展用户宏。

更复杂的公式可以使用函数、运算符和括号的组合。您可以使用触发器表达式中支持的所有函数和运算符。逻辑和运算符优先级完全相同。

与触发器表达式不同，Zabbix 根据监控项更新间隔处理可计算监控项，而不是在接收到新值时。

可计算监控项公式中历史函数引用的所有监控项都必须存在并且正在收集数据。此外，如果您更改引用监控项的键，则必须手动更新任何使用该键的公式。

在以下几种情况下，可计算监控项可能会变得不受支持：

- 引用的监控项
  - 没有找到
  - 被禁用
  - 属于禁用的主机
  - 不支持 (除了 nodata() 函数和具有未知值的运算符)
- 没有数据来计算函数
- 被 0 除
- 使用了不正确的语法

用法示例

示例 1

计算'/' 上可用磁盘空间的百分比。

使用函数 **last**:

```
100*last(//vfs.fs.size[/,free])/last(//vfs.fs.size[/,total])
```

Zabbix 将获取可用和总磁盘空间的最新值，并根据给定的公式计算百分比。

示例 2

计算 Zabbix 处理的值的 10 分钟平均值。

使用函数 **avg**:

```
avg(/Zabbix Server/zabbix[wcache,values],10m)
```

请注意，可计算监控项选取长时间段的数据会影响 Zabbix 服务器的性能。

示例 3

计算 eth0 的总带宽

两个函数之和:

```
last(//net.if.in[eth0,bytes])+last(//net.if.out[eth0,bytes])
```

示例 4

计算入站流量的百分比

更为复杂的表达式：

```
100*last(//net.if.in[eth0,bytes])/(last(//net.if.in[eth0,bytes])+last(//net.if.out[eth0,bytes]))
```

另请参阅: 聚合计算示例

聚合计算

概述

聚合计算是一种计算监控项 类型，允许 Zabbix 服务器从多个项目中收集信息，然后根据使用的聚合函数计算聚合。

聚合计算项仅支持无符号整数和浮点值（信息类型）。

聚合计算不需要在被监控的主机上运行任何 agent。

- 项目历史:

```
aggregate_function(function(/host/item,parameter),function(/host2/item2,parameter),...)
```
 - 一个 foreach 函数作为唯一参数: `aggregate_function(foreach_function(/*/key?[group="host group"],timeperiod))`

where:
- `聚合函数`是支持的聚合函数之一: avg，max，min，sum，等等.
- 函数是受支持的 foreach 函数之一: avg _ foreach、 count _ foreach 等

Foreach 函数使用项筛选器处理多个项的历史记录，并返回一个值数组——每个项一个值。

注意: 如果聚合项类型的信息是 Numeric (无符号)，则聚合结果生成的浮点值将被裁剪为整数。. 如果出现以下情况，则可能不支持聚合计算: - 没有找到任何引用的项 (如果项目键不正确、项目不存在或所有包含的组都不正确，则可能发生这种情况) - 没有数据来计算函数

使用示例

用于聚合计算的键的示例

示例 1

主机组 “ MySQL Servers” 的总磁盘空间

```
sum(last_foreach(/*/vfs.fs.size[/,total]?[group="MySQL Servers"]))
```

示例 2

主机上与 net.if.in[*] 匹配的所有项的最新值之和。

```
sum(last_foreach(/host/net.if.in[*]))
```

示例 3

主机组 “MySQL Servers” 的平均处理器负载。

```
avg(last_foreach(/*/system.cpu.load[,avg1]?[group="MySQL Servers"]))
```

示例 4

主机组 “MySQL Servers” 每秒平均查询次数 5 分钟平均值。

```
avg(avg_foreach(/*/mysql.qps?[group="MySQL Servers"],5m))
```

示例 5

具有特定标记的多个主机组中所有主机的平均 CPU 负载。

```
avg(last_foreach(/*/system.cpu.load?[(group="Servers A" or group="Servers B" or group="Servers C") and (ta
```

示例 6

用于计算整个主机组的监控项最新值总和。

```
sum(last_foreach(/*/net.if.out[eth0,bytes]?[group="video"])) / sum(last_foreach(/*/nginx_stat.sh[active]?[
```

示例 7

主机组 “Zabbix servers” 中不支持的监控项总数。

```
sum(last_foreach(/*/zabbix[host,,items_unsupported]?[group="Zabbix servers"]))
```

正确/错误的语法示例

表达式（包括函数调用）不能用作历史、趋势或循环函数 参数。但是，这些函数本身可以在其他（非历史）函数参数中使用。

表达式	示例
有效的	`avg(last(/host/key1),last(/host/key2)*10,last(/host/key1)*100)` `max(avg(avg_foreach(/*/system.cpu.load?[group="Servers A"],5m)),avg(avg_foreach(/*/system.cpu.load?[group="Servers B"],5m)),avg(avg_foreach(/*/system.cpu.load?[group="Servers C"],5m)))`

无效的	`sum(/host/key,10+2)`
	`sum(/host/key, avg(10,2))`
	`sum(/host/key,last(/host/key2))`

请注意，在如下表达式中：

·sum(sum_foreach(//resptime[*],5m))/sum(count_foreach(//resptime[*],5m))

不能保证等式的两个部分始终具有相同的一组值。在计算表达式的一部分时，可能会收到请求期间的新值，然后表达式的另一部分将具有一组不同的值。

8 内部检查

概述

内部检查可以监控 Zabbix 的内部进程。换句话说，可以监控 Zabbix server 或者 Zabbix proxy 的运行情况。

被计算的内部检查:

- 在 Zabbix server 上 - 主机是否被服务端监控
- 在 Zabbix proxy 上 - 主机是否被代理端监控

内部检查在 server 或者 proxy 上执行，无论主机是否处于维护状态。

要使用这个监控项，请选择 **Zabbix internal** 监控项类型。

Note:
内部检查是由 Zabbix 轮询进程（poller）处理。

性能

使用某些内部监控项可能会对性能产生负面影响。这些监控项是：

- ·`zabbix[host,,items]`
- ·`zabbix[host,,items_unsupported]`
- ·`zabbix[hosts]`
- ·`zabbix[items]`
- ·`zabbix[items_unsupported]`
- ·`zabbix[queue]`
- ·`zabbix[required_performance]`
- ·`zabbix[stats,,,queue]`
- ·`zabbix[triggers]`

系统信息 和队列 前端部分也会受到影响。

支持的检查

- 没有尖括号的参数为常量 - 比如, `zabbix[host,<type>,available]` 中的'host' 和'available'。在监控项键值 原样使用它们。
- "代理端不支持" 的监控项和监控项参数的值只能在主机被服务端监控的情况下收集。反之亦然，"服务端不支持" 的值只能在主机被代理端监控的情况下收集。

键值			
▲	描述	返回值	注释
zabbix[boottime]			

键值			
	Zabbix server 或者 Zabbix proxy 进程启动的时间，单位是秒.	整数.	
zabbix[cluster,discovery,nodes]	发现高可用集群 节点.	JSON.	这个监控项可以用在 LLD.
zabbix[history]	存储在 HISTORY 表中的值的数量.	整数.	这个监控项从 Zabbix 6.0 开始已经废弃. 如果使用了 MySQL InnoDB, Oracle 或者 PostgreSQL 不要使用这个监控项! *(代理端不支持)*
zabbix[history_log]	存储在 HISTORY_LOG 表中的值的数量.	整数.	这个监控项从 Zabbix 6.0 开始已经废弃. 如果使用了 MySQL InnoDB, Oracle 或者 PostgreSQL 不要使用这个监控项! *(代理端不支持)*
zabbix[history_str]	存储在 HISTORY_STR 表中的值的数量.	整数.	这个监控项从 Zabbix 6.0 开始已经废弃. 如果使用了 MySQL InnoDB, Oracle 或者 PostgreSQL 不要使用这个监控项! *(代理端不支持)*
zabbix[history_text]			

键值			
	存储在 HISTORY_TEXT 表中的值的数量.	整数.	这个监控项从 Zabbix 6.0 开始已经废弃. 如果使用了 MySQL InnoDB, Oracle 或者 PostgreSQL 不要使用这个监控项! *(代理端不支持)*
zabbix[history_uint]			
	存储在 HISTORY_UINT 表中的值的数量.	整数.	这个监控项从 Zabbix 6.0 开始已经废弃. 如果使用了 MySQL InnoDB, Oracle 或者 PostgreSQL 不要使用这个监控项! 这个监控项从 Zabbix 1.8.3 开始支持. *(代理端不支持)*
zabbix[host,,items]			
	主机上启用的监控项数量(包含支持的和不支持的) .	整数.	这个监控项从 Zabbix 3.0.0 开始支持.
zabbix[host,,items_unsupported]			
	主机上启用的不支持的监控项数量.	整数.	这个监控项从 Zabbix 3.0.0 开始支持.*
zabbix[host,,maintenance]			
	当前主机的维护状态.	0 - 主机处于正常状态, 1 - 主机处于维护状态，并且在采集数据, 2 - 主机处于维护状态，但是不采集数据.	这个监控项总是由 Zabbix 服务端处理，而不管主机当前所处位置(在服务端或代理端上). 代理端将不会接收这个带有配置数据的监控项. 第二个参数必须为空，并保留以备将来使用.
zabbix[host,discovery,interfaces]			

键值			
	在 Zabbix 前端中主机所有配置接口的详细信息.	JSON object.	这个监控项可以被用在low-level discovery. 这个监控项从 Zabbix 3.4.0 开始支持. *(代理端不支持)*
zabbix[host,<type>,available]	主机上特定类型检查的主接口的可用性.	0 - 不可用, 1 - 可用, 2 - 未知.	有效 类型是: *agent*, *snmp*, *ipmi*, *jmx* 监控项的值根据有关主机的不可达/不可用配置参数计算. 这个监控项从 Zabbix 2.0.0 开始支持.
zabbix[hosts]	已监控的主机数量.	整数.	
zabbix[items]	已启用的监控项数量 (支持的和不支持的).	整数.	
zabbix[items_unsupported]	不支持的监控项数量.	整数.	
zabbix[java,,<param>]	关于 Zabbix Java 网关的信息.	如果 <param> 是 *ping*, 则返回”1” . 可以使用 nodata() 触发器函数检查 Java 网关可用性. 如果 <param> 是 *version*, 则返回 Java 网关的版本. 例如: ”2.0.0”.	**param** 的有效值是: *ping*, *version* 第二个参数必须为空，并保留以备将来使用.
zabbix[lld_queue]	在 LLD 队列中排队的值.	整数.	这个监控项用来监控 LLD 的队列长度. 这个监控项从 Zabbix 4.2.0 开始支持.

键值			
zabbix[preprocessing_queue]	在预处理队列中排队的值的数量.	整数.	这个监控项用来监控预处理队列的长度. 这个监控项从Zabbix 3.4.0开始支持.
zabbix[process,<type>,<mode>,<state>]			

键值				
		一个特定 Zabbix 进程或一组进程 (由 <type> 和 <mode> 标记) 在 <state> 的时间，单位是百分比. 它只计算最后一分钟. 如果 <mode> 是未运行的 Zabbix 进程号 (例如, <mode> 被指定为 6 的时候运行 5 个 poller), 这样的监控项将变成不支持的状态. 最小值和最大值是指单个进程的使用百分比. 因此，如果在一组 3 个 poller 中，每个进程的使用率分别为 2、18 和 66, 最小值将返回 2, 最大值将返回 66. 进程报告它们在共享内存中所做的事情，并且自我监控进程则每秒钟都会汇总这些数据. 状态改变 (busy/idle) 是在变化时记录的 - 因此，一个进程成为 busy 状态时会被记录之后不会改变或更新状态，直到它变成 idle 状态. 这将确保即使完全挂起的进程也将正确地记录为 100% busy. 目前，”busy” 意味着”not sleeping”，但在未来可能会引入额外的状态 - 等待	时间的百分比. 浮点值.	server processes支持的 类型: *alert manager*, *alert syncer*, *alerter*, *availability manager*, *configuration syncer*, *discoverer*, *escalator*, *history poller*, *history syncer*, *housekeeper*, *http poller*, *icmp pinger*, *ipmi manager*, *ipmi poller*, *java poller*, *lld manager*, *lld worker*, *odbc poller*, *poller*, *preprocessing manager*, *preprocessing worker*, *proxy poller*, *self-monitoring*, *snmp trapper*, *task manager*, *timer*, *trapper*, *unreachable poller*, *vmware collector* proxy processes支持的 类型: *availability manager*, *configuration syncer*, *data sender*, *discoverer*, *heartbeat sender*, *history poller*, *history syncer*, *housekeeper*, *http poller*, *icmp pinger*, *ipmi manager*,

键值			
zabbix[proxy,<name>,<param>]	Zabbix 代理端的信息.	整数.	**name**: 代理端名称 **param** 的有效值有: *lastaccess* - 从代理端上收到的最后一次心跳消息的时间戳 *delay* - 采集的值多长时间未发送, 计算方式为” 代理端延迟” (当前代理端的时间与代理端上最老的未发送值的时间戳之间的差值) + (” 当前服务端时间” - ” 代理端最后一次访问时间”) 比如: => zabbix[proxy,”Germany”,lastacc `fuzzytime()` 函数 可以用来检测代理端的可用性. 这个监控项始终由 Zabbix 服务端处理与主机位置无关 (在服务端或者代理端).
zabbix[proxy_history]	代理端 history 表中等待发送到服务端的值的数量.	整数.	(服务端不支持)
zabbix[queue,<from>,<to>]	大于 <from> 的秒数但是小于 <to> 秒数的延迟的队列中被监控的监控项数量.	整数.	**from** - 缺省值: 6 秒 **to** - default: 无穷，不限制 Time-unit symbols 支持 s,m,h,d,w 这些参数.
zabbix[rcache,<cache>,<mode>]			

键值			
	Zabbix 配置缓存的可用性统计信息.	整数 (大小); float (百分比).	**cache**: *buffer* 有效的 **modes** 有: *total* - 总 buffer 大小 *free* - 空闲 buffer 大小 *pfree* - 空闲 buffer 的百分比 *used* - 使用的 buffer 大小 *pused* - 使用的 buffer 百分比 *pused* 模式从 Zabbix 4.0.0 开始支持.
zabbix[requiredperformance]	Zabbix 服务端或者 Zabbix 代理端所需的性能, 以每秒新增的值计算.	浮点数.	大约与 报告中的” 服务端所需性能，每秒新值” 相关 → *System information*.
zabbix[stats,<ip>,<port>]	远程 Zabbix 服务端或者代理端内部监控指标.	JSON 对象.	**ip** - 远程查询服务端/代理端的 IP/DNS/网络掩码列表 (缺省是 127.0.0.1) **port** - 远程查询服务端/代理端的端口号 (缺省是 10051) 注意，stats 请求将只接受目标实例上’StatsAllowedIP’server/pro参数中列出的地址. 此监控项返回一组选定的内部指标. 更多信息参见Zabbix 远程监控的统计数据. 从 4.2.0 开始支持.

键值			
zabbix[stats,<ip>,<port>,queue,<from>,<to>]	远程 Zabbix 服务端或者代理端内部队列指标 (另见 zabbix[queue,<from>,<to>]).	JSON 对象.	**ip** - 远程查询服务端/代理端的 IP/DNS/网络掩码列表 (缺省是 127.0.0.1) **port** - 远程查询服务端/代理端的端口号 (缺省是 10051) **from** - 最少的延迟 (缺省是 6 秒) **to** - 最多的延迟 (缺省是 infinity) 注意，stats 请求将只接受目标实例上'StatsAllowedIP' server/proxy 参数中列出的地址. 从 4.2.0 开始支持.
zabbix[tcache,cache,<parameter>]			

键值			
	Zabbix 趋势功能缓存的有效性统计.	整数 (大小); 浮点数 (百分比).	有效 **parameters** 是: *all* - 总缓存请求 (缺省值) *hits* - 缓存命中 *phits* - 缓存命中的百分比 *misses* - 缓存未命中 *pmisses* - 缓存未命中的百分比 *items* - 缓存的监控项数量 *requests* - 缓存请求的监控项数量 *pitems* - 缓存监控项/(缓存监控项 + 请求) 的百分比. 较低的百分比很可能意味着可以减少缓存大小. 从 5.4.0 开始支持. *(*代理端不支持*)*
zabbix[trends]	存储在 TRENDS 表中的值的数量.	整数.	该监控项自 Zabbix 6.0 以来已弃用. 如果使用了 MySQL InnoDB, Oracle 或者 PostgreSQL 不要使用这个监控项! *(*代理端不支持*)*
zabbix[trends_uint]	存储在 TRENDS_UINT 表中的值的数量.	整数.	该监控项自 Zabbix 6.0 以来已弃用. 如果使用了 MySQL InnoDB, Oracle 或者 PostgreSQL 不要使用这个监控项! 这个监控项从 Zabbix 1.8.3 开始支持. *(*代理端不支持*)*

键值			
zabbix[triggers]			
	Zabbix 数据库中已启用触发器的数量，在已启用的主机上启用所有监控项.	整数.	*(*代理端不支持*)*
zabbix[uptime]			
	Zabbix 服务端或者 Zabbix 代理端进程启动的时间，单位是秒.	整数.	
zabbix[vcache,buffer,<mode>]			
	Zabbix 值缓存的可用性统计信息.	整数 (大小); 浮点数 (百分比).	有效 **modes** 是: *total* - buffer 总大小 *free* - 空闲 buffer 的大小 *pfree* - 空闲 buffer 的百分比 *used* - 已用的 buffer *pused* - 已用的 buffer 百分比 *(*代理端不支持*)*
zabbix[vcache,cache,<parameter>]			

键值				
		Zabbix 值缓存的有效性统计.	整数. *mode* 参数有: 0 - 正常模式, 1 - 低内存模式	**parameter** 有效值有: *requests* - 请求的总数 *hits* - 缓存命中数量 (从缓存中获取的历史值) *misses* - 缓存未命中数量 (从数据库中获取的历史值) *mode* - 缓存操作模式的值 这个监控项从 Zabbix 2.2.0 开始支持，*mode* 参数从 Zabbix 3.0.0. *(*代理端不支持*)* 一旦开启低内存模式，缓存值将保持这种状态 24 小时，即使触发此模式的问题被更快地解决. 您可以将此键值与 *Change per second* 预处理步骤一起使用，以便获得每秒统计值.
zabbix[version]		Zabbix 服务端或者代理端的版本.	字符串.	这个监控项从 Zabbix 5.0.0 开始支持. 比如返回这样的值: 5.0.0beta1
zabbix[vmware,buffer,<mode>]				

键值					
	Zabbix vmware 缓存可用性统计信息.			整数 (大小); 浮点数 (百分比).	**modes** 有效值: *total* - buffer 的总大小 *free* - 空闲 buffer 的大小 *pfree* - 空闲 buffer 的百分比 *used* - 已用的 buffer 大小 *pused* - 已用的 buffer 百分比
zabbix[wcache,<cache>,<mode>]					
	Zabbix 写缓存的可用性和有效性统计.				<cache> 必须指定.
	Cache	**Mode**			
	values	all *(缺省值)*	Zabbix 服务端或者代理端处理的值的总数，不支持的监控项除外.	整数.	计数器. 你可以将此键值与 *Change per second* 预处理步骤一起使用，以便获得每秒统计值.
		float	处理的浮点值的数量.	整数.	计数器.
		uint	处理的无符号整数值的数量.	整数.	计数器.
		str	处理的字符/字符串的数量.	整数.	计数器.
		log	处理的日志值的数量.	整数.	计数器.
		text	处理的文本值的数量.	整数.	计数器.
		not supported	监控项处理导致监控项变得不受支持或保持该状态的次数.	整数.	计数器.
	history	pfree *(缺省值)*	可用历史 buffer 的百分比.	浮点数.	用于存储历史缓存的监控项值. 如果数值较低，则表明数据库端存在性能问题.
		free	空闲历史 buffer 大小.	整数.	
		total	总的历史 buffer 大小.	整数.	
		used	已用的历史 buffer 大小.	整数.	
		pused	已用的历史 buffer 百分比.	浮点数.	*pused* 模式从 Zabbix 4.0.0 开始支持.

键值					
	index	pfree *(缺省值)*	可用历史索引缓冲区的百分比.	浮点数.	历史索引缓存用于对存储在历史缓存中的值进行索引. *Index* 缓存从 Zabbix 3.0.0 开始支持.
		free	空闲历史索引历史缓冲区的大小.	整数.	
		total	总的历史索引历史缓冲区的大小.	整数.	
		used	已经使用的历史索引历史缓冲区的大小.	整数.	
		pused	已经使用的历史索引历史缓冲区的百分比.	浮点数.	*pused* 模式从 Zabbix 4.0.0 开始支持.
	trend	pfree *(缺省值)*	可用趋势缓存的百分比.	浮点数.	趋势缓存存储当前小时内所有接收数据的监控项的聚合. *(代理端不支持)*
		free	可用趋势缓存的大小.	整数.	*(代理端不支持)*
		total	总的趋势缓存的大小.	整数.	*(代理端不支持)*
		used	已用趋势缓存的大小.	整数.	*(代理端不支持)*
		pused	已用趋势缓存的百分比.	浮点数.	*(代理端不支持)* *pused* 模式从 Zabbix 4.0.0 开始支持.

9 SSH 检查

概述

SSH 检查作为一种 agent-less 监控，不依赖于 Zabbix agent 执行。

要执行 SSH 检查，Zabbix 服务器必须首先配置 SSH2 支持 (libssh2 或 libssh)。参考:要求。

Attention:
从 RHEL 8 开始只支持 libssh.

配置

密码验证

SSH 检查提供两种身份验证方法，用户/密码对和基于密钥文件。

如果您正在构建从源码 Zabbix，且不打算使用密钥文件，除了将 libssh2/libssh 链接到 Zabbix，无需额外配置。

密钥文件认证

SSH 监控项基于密钥的身份验证，需要对服务器配置进行某些更改。

使用 `root` 身份打开 Zabbix 服务器配置文件 (zabbix_server.conf) 并查找以下行：

```
# SSHKeyLocation=
```

取消注释，配置公钥和私钥所在文件夹的完整路径，样例如下：

```
SSHKeyLocation=/home/zabbix/.ssh
```

保存文件并重启 zabbix_server 服务。

这里的 */home/zabbix* 是 *zabbix* 用户的家目录，*.ssh* 是其中的一个目录。默认情况下，ssh-keygen命令生成的公钥和私钥将保存在这个目录中。

不同发行版操作系统的 zabbix-server 安装程序，会在不太明显的地方（与系统账户一样）创建一个带有主目录的 zabbix 用户账户。例如，*/var/lib/zabbix* 。

在生成秘钥前可以考虑指定密钥保存的路径到更常见的位置，该路径需要跟 zabbix server 配置中的 *SSHKeyLocation* 保持一致.

如果已经根据安装部分手动添加了 *zabbix* 账号，这些步骤可以跳过。因为在这种情况下，很可能主目录已经位于*/home/zabbix*。

要更改 *zabbix* 用户的设置，必须先停止 zabbix 所有的进程：

service zabbix-agent stop # service zabbix-server stop

要更改和移动一个已存在的目录应该执行以下命令:

```
# usermod -m -d /home/zabbix zabbix
```

在旧版本中不存在 zabbix 根目录，需要手动创建，可以执行以下命令:

```
# test -d /home/zabbix || mkdir /home/zabbix
```

为了保证安全，可以执行以下命令以设置跟目录的权限：

```
# chown zabbix:zabbix /home/zabbix
# chmod 700 /home/zabbix
```

当前进程如果是停止的，可以立即启动：

```
# service zabbix-agent start
# service zabbix-server start
```

生成公钥和私钥的命令和步骤如下：

```
# sudo -u zabbix ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/zabbix/.ssh/id_rsa):
Created directory '/home/zabbix/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/zabbix/.ssh/id_rsa.
Your public key has been saved in /home/zabbix/.ssh/id_rsa.pub.
The key fingerprint is:
90:af:e4:c7:e3:f0:2e:5a:8d:ab:48:a2:0c:92:30:b9 zabbix@it0
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|       .         |
|      o          |
| .     o         |
|+     . S        |
|.+   o =         |
|E .   * =        |
|=o . ..* .       |
|... oo.o+        |
+-----------------+
```

注意: 公钥 (*id_rsa.pub*) 和私钥 (*id_rsa*) 默认生成在*/home/zabbix/.ssh* 路径下，该路径对应 Zabbix server 的 *SSHKeyLocation* 配置参数

Attention:
ssh-keygen 命令和 SSH 服务可能支持 “rsa” 以外的密钥类型，但 Zabbix 使用的 libssh2 可能不支持.

Shell 配置表单

对于每一个将被 SSH 监控的主机。这个步骤应该只执行一次。

通过下面的命令，公钥文件可以安装在远程主机 *10.10.10.10* 上，这样就可以使用 *root* 帐户执行 SSH 检查:

```
# sudo -u zabbix ssh-copy-id root@10.10.10.10
The authenticity of host '10.10.10.10 (10.10.10.10)' can't be established.
RSA key fingerprint is 38:ba:f2:a4:b5:d9:8f:52:00:09:f7:1f:75:cc:0b:46.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.10.10.10' (RSA) to the list of known hosts.
root@10.10.10.10's password:
Now try logging into the machine, with "ssh 'root@10.10.10.10'", and check in:
  .ssh/authorized_keys
to make sure we haven't added extra keys that you weren't expecting.
```

现在 *zabbix* 用户可以使用默认私钥 (*/home/zabbix/.ssh/id_rsa*) ，对目标主机进行 SSH 免密登录检查

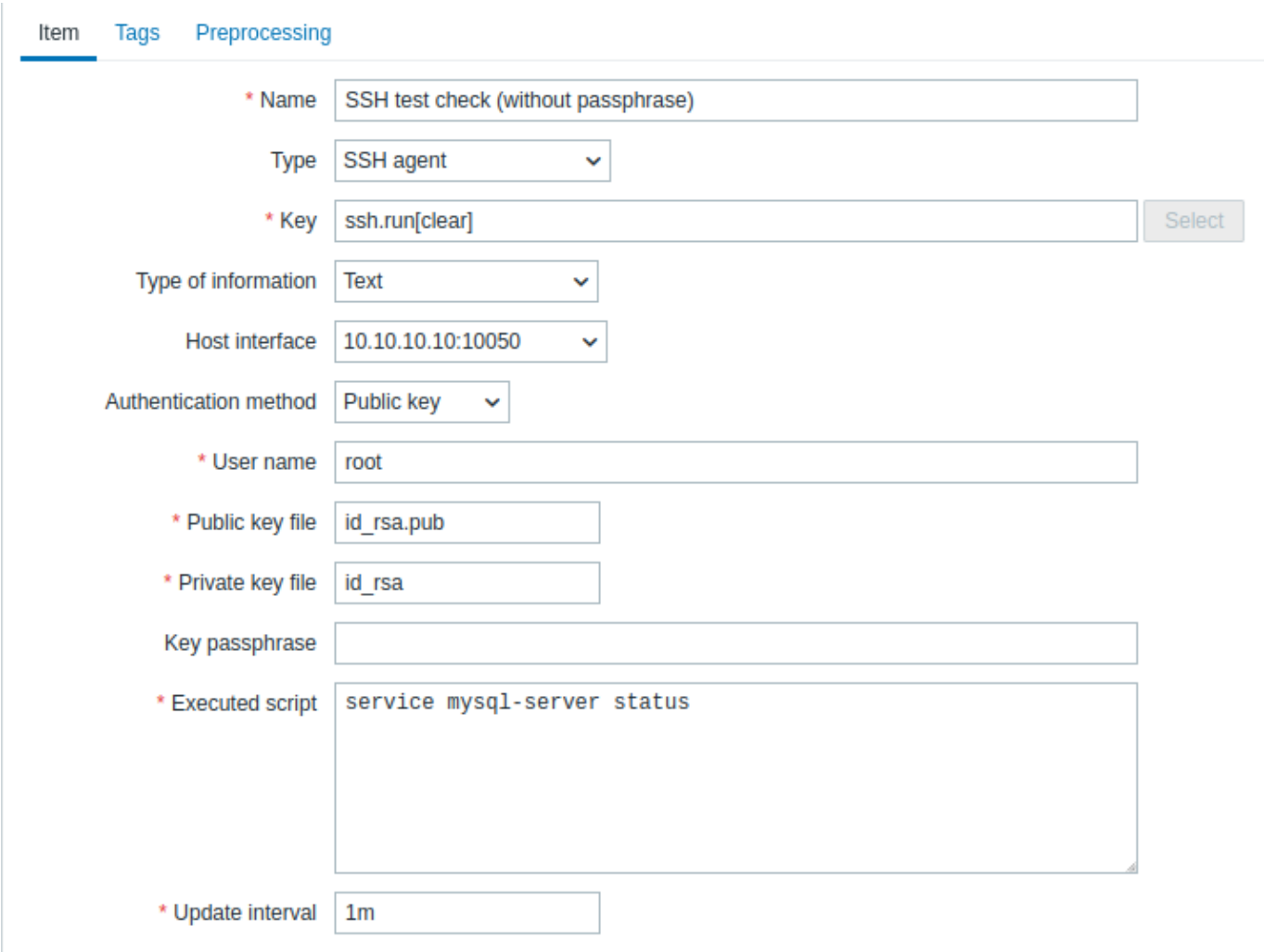
```
# sudo -u zabbix ssh root@10.10.10.10
```

如果登录成功, 表示 shell 中的配置部分已经完成，可以关闭远程 SSH 会话。

监控项配置

要执行的命令必须放在监控项配置中的 “执行的脚本” 字段中。

通过将多个命令放在一行上，可以遍历执行多个命令。在这种情况下，返回的值也将被格式化为多行。



所有必填字段都标有红色星号。

SSH 监控项需要特定信息的字段：

参数	说明	备注
类型	在此处选择 **SSH** 客户端。	

参数	说明	备注
Key	唯一（每个主机）监控项键，格式为 **ssh.run[<unique short description>,<ip>,<port>,<encoding>]**	<unique short description> 是必需的并且对于每个主机的所有 SSH 监控项应该是唯一的 默认端口是 22，而不是在分配此监控项的接口中指定的端口
认证方式	“密码” 或 “公钥” 选一个	
用户名	在远程主机上进行身份验证的用户名。 必填	
公钥文件	如果 *Authentication method* 是 “Public key”，则公钥的文件名。必需	示例：*id_rsa.pub* - 命令生成的默认公钥文件名 ssh-keygen
私钥文件	如果 *Authentication method* 是 “Public key”，则私钥的文件名。必需	示例：*id_rsa* - 默认私钥文件名
Password or 密钥密码	验证密码或 Passphrase 如果它用于私钥	如果没有使用密码，则将 *Key passphrase* 字段留空 另请参阅已知问题 关于密码短语的使用
执行的脚本	使用 SSH 远程会话执行的 shell 命令	示例： *date +%s* *service mysql-server status* *ps auxww | grep httpd | wc -l*

Attention:
libssh2 库可能会将可执行脚本截断为 ~32kB.

10 Telnet 检查

概述

Telnet 检查不需要安装 Zabbix agent，可对未安装代理的主机进行监控。

可配置字段

要执行的实际命令必须放在监控项配置中的 执行的脚本字段中。如果要执行多条命令，一行写一条，命令将逐条执行。这种情况下，返回值也将为多行显示。

支持的 shell 提示符结尾的字符：

- $
- #
-
- %

Note:
以上结束符的 telnet 提示行将从返回值中删除，但仅针对命令列表中的第一个命令，即仅在 telnet 会话开始处。

键值	描述
telnet.run[<unique short description>,<ip>,<port>,<encoding>]	使用 telnet 连接在远程设备上运行命令

Attention:
如果 telnet 检查返回一个非 ascii 字符和非 utf8 编码的值，那应该正确的指定 <encoding> 参数。有关详细信息，请参阅编码返回值 页面。

11 外部检查

概述

外部检查是 Zabbix server 通过运行 shell 脚本或二进制执行的检查。但是当主机被 Zabbix proxy 接管时，外部检查则由 Zabbix proxy 执行。

外部检查不需要在被监控的主机上运行任何代理

监控项键的语法是：

script[<parameter1>,<parameter2>,...]

参数：

字段	描述
script	shell 脚本或二进制文件的名称.
parameter(s)	可选的命令行参数.

如果您不想将任何参数传递给脚本, 您可以使用：

script[] or script

Zabbix Server 将查找外部脚本位置的目录 (Zabbix Server 配置文件中的 “ExternalScripts” 参数定义的位置) 并执行该命令。该命令将以 Zabbix Server 服务运行的用户身份执行 *script*，任何访问权限或环境变量都应该在封装在脚本中处理，该命令的权限应该只允许该用户在指定的目录下执行它。

Warning:
不要过度使用外部检查！因为每个脚本都需要 Zabbix Server 启动一个 fork 出来的进程，所以运行很多脚本会大幅度的降低 Zabbix 的性能。

用法示例

使用第一个参数执行脚本 **check_oracle.sh** '-h'。第二个参数将替换为 IP 地址或 DNS 名称，取决于主机属性中的选择。

```
check_oracle.sh["-h","{HOST.CONN}"]
```

假设主机配置为使用 IP 地址，Zabbix 将执行：

```
check_oracle.sh '-h' '192.168.1.4'
```

外部检查结果

检查的返回值是标准输出和标准错误（从 zabbix 2.0 开始，返回完整输出，并去掉了末尾的空格）。

Attention:
在标准错误输出的情况下，文本（字符、日志或文本类型的信息）项将不会转变为不被支持的监控项状态。

如果没有找到所请求的脚本，或者 Zabbix server 没有执行该脚本的权限，则不支持该监控项，并将设置相应的错误消息。在超时的情况下，监控项也将被标记为不受支持，并显示相应的错误消息，脚本 fork 出的进程将被杀死。

12 采集器监控项

概述

采集器监控项接收传入的数据，它不会去主动采集数据。

它接受任何形式的推送到 zabbix server 的数据。

要使用采集器监控项，你需要：

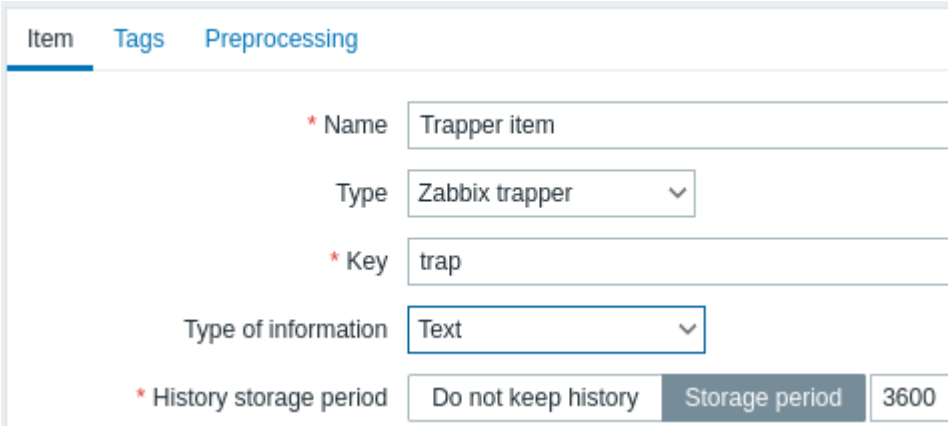
- 在 Zabbix 中建立一个采集器监控项
- 将数据发送到 Zabbix server

配置

监控项配置

要配置采集器监控项：

- 转到：配置 → 主机
- 点击主机所在行的监控项
- 点击创建监控项
- 在表单中输入监控项的参数



所有标有红色星号的都是必填字段。

需要填写采集器特定信息的字段是：

类型	在此处选择 **Zabbix** 采集器。
键值	输入发送数据时用于识别监控项的键。
信息类型	选择与将要发送的数据格式相对应的信息类型。
允许的主机	以逗号分隔的 IP 地址列表或主机名，可选择以 CIDR 表示法。 如果指定，则仅接受来自此处列出的主机的传入连接。 如果启用 IPv6 支持，则为'127.0.0.1'、'::127.0.0.1'、'::ffff:127.0.0.1' 是等价的，'::/0' 将允许任何 IPv4 或 IPv6 地址。'0.0.0.0/0' 可以用于允许任何 IPv4 地址. 请注意，" 与 IPv4 兼容的 IPv6 地址"（0000::/96 前缀）受支持，但已被 RFC4291不推荐使用。 示例：Server=127.0.0.1, 192.168.1.0/24, 192.168.3.1-255, 192.168.1-10.1-255, ::1,2001:db8::/ 32、zabbix.domain 从 Zabbix 2.2.0 开始, 该字段允许使用空格和用户宏. 自 Zabbix 4.0.2 起，主机宏：{HOST.HOST}, {HOST.NAME}、{HOST.IP}、{HOST.DNS}、{HOST.CONN} 允许在此字段中。

Note:
在保存该监控项后，您可能需要等待 60 秒，直到服务器的缓存配置更新到最新的内容后，才能发送监控值。

数据发送

在最简单的情况下，我们可以使用zabbix_sender程序来发送一些 "测试值"：

```
zabbix_sender -z <server IP address> -p 10051 -s "New host" -k trap -o "test value"
```

我们使用下列这些关键字来指定发送值

-z - 指定 Zabbix server 的 IP 地址

-p - 指定 Zabbix server 的端口（默认为 10051）

-s -指定主机（请确保在此使用 "主机名称" 的主机名 ，而不是 "可见的名称" 名称）

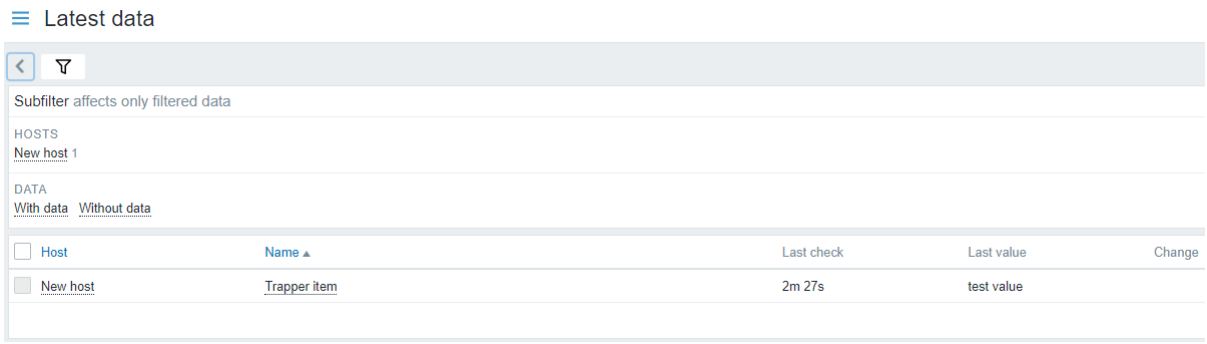
-k - 指定我们之前定义的监控项的键值

-o - 指定要发送的实际值

Attention:
Zabbix trapper 进程不会扩展监控项键值中使用的宏，以检查目标主机对应的监控项键值是否存在。

展示

这是 监控 → 最新数据 中的结果：



请注意，如果传入一个数值，数据图将在该值的时间点的左边和右边显示一条水平线。

13 JMX 监控

概述

JMX 监控端可用于监控 Java 应用程序的 JMX 计数器。

从 zabbix 2.0 开始，JMX 监控端以 Zabbix 守护进程的形式运行，称为 “Zabbix Java gateway”。

为了检索主机上特定 JMX 计数器的值，Zabbix 服务器查询 Zabbix **Java gateway**，网关使用JMX 管理 API远程查询指定的应用程序，将结果返回给 zabbix server。

有关更多细节和设置，请参考Zabbix Java gateway。

Warning:
Java gateway 和 JMX 应用程序之间的通信应该在防火墙上放行。

启用远程 JMX 监控 java 应用

Java 应用程序不需要安装任何额外的软件，但是需要使用下面指定的命令行选项来启动它，以使应用程序进程支持远程 JMX 监控。

如果你只是希望开始在本地主机上监控一个简单的 Java 应用程序，没有安全性选项，请参考添加以下选项启动它：

```
 java \
-Dcom.sun.management.jmxremote \
-Dcom.sun.management.jmxremote.port=12345\
-Dcom.sun.management.jmxremote.authenticate=false\
-Dcom.sun.management.jmxremote.ssl=false\
-Dcom.sun.management.jmxremote.registry.ssl=false\
-jar /usr/share/doc/openjdk-6-jre-headless/demo/jfc/Notepad/Notepad.jar
```

这个 Java 程序监听来自本地端口 12345 上的接入 JMX 连接（来自仅限本地主机），并不需要身份验证或 SSL。

如果要允许其他主机的连接，请设置 -Djava.rmi.server.hostname 参数为该接口的 IP。

如果您希望在安全性方面更加严格，还有许多其他 Java 选项可供选择。例如，下一个示例使用一组更通用的选项启动应用程序，并允许更多的网段连接，而不仅仅是本地主机。

```
 java \
-Djava.rmi.server.hostname=192.168.3.14\
-Dcom.sun.management.jmxremote \
-Dcom.sun.management.jmxremote.port=12345\
-Dcom.sun.management.jmxremote.authenticate=true \
-Dcom.sun.management.jmxremote.password.file=/etc/java-6-openjdk/management/jmxremote.password \
-Dcom.sun.management.jmxremote.access.file=/etc/java-6-openjdk/management/jmxremote.access \
-Dcom.sun.management.jmxremote.ssl=true\
-Dcom.sun.management.jmxremote.registry.ssl=true\
-Djavax.net.ssl.keyStore=$YOUR_KEY_STORE \
-Djavax.net.ssl.keyStorePassword=$YOUR_KEY_STORE_PASSWORD \
-Djavax.net.ssl.trustStore=$YOUR_TRUST_STORE \
-Djavax.net.ssl.trustStorePassword=$YOUR_TRUST_STORE_PASSWORD \
-Dcom.sun.management.jmxremote.ssl.need.client.auth=true \
-jar /usr/share/doc/openjdk-6-jre-headless/demo/jfc/Notepad/Notepad.jar
```

这些设置中的大部分可以在/etc/java-6-openjdk/management/management 中指定（或者读取你系统上的配置文件）

请注意，如果您希望使用 SSL，您必须修改 startup.sh 脚本，为 Java 网关添加 “-Djavax.net.ssl.*” 选项，以便它知道在哪里找到密钥和信任存储库

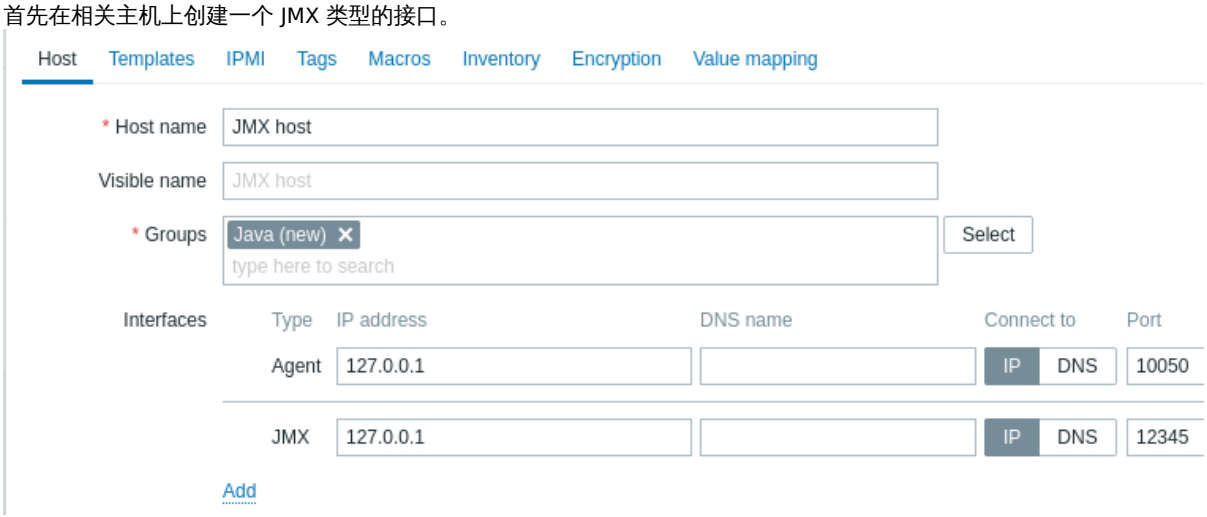
参见使用 JMX 监控和管理获得详细的描述。

在 Zabbix web 管理页面上配置 JMX 接口和监控项

Java 网关在运行时，服务器会主动连接它，Java 应用程序启用了远程 JMX 监视，现在可以在 Zabbix GUI 中配置接口和监控项了。

配置 JMX 接口

首先在相关主机上创建一个 JMX 类型的接口。

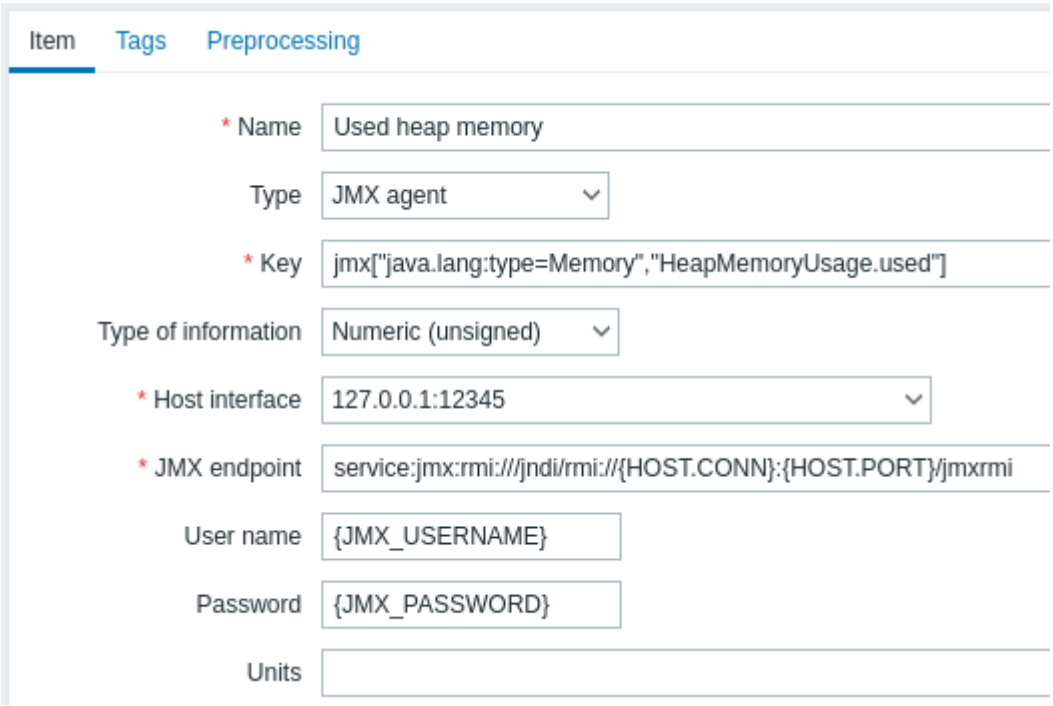


标有红色星号的为必填项。

添加 JMX 代理监控项

对于您感兴趣的每个 JMX 计数器，您添加附加到该接口的 **JMX** 代理监控项。

下面屏幕截图中的键表示 `jmx["java.lang:type=Memory","HeapMemoryUsage.used"]`。



所有必填输入字段都标有红色星号。

JMX 监控项需要的特定信息字段是：

.	.
类型	在这里设置 **JMX** 代理。

.	.
键值	jmx[] 监控项键包含三个参数： 对象名称 - MBean 的对象名称 属性名称 - 一个 MBean 属性名称，带有由点分隔的可选复合数据字段名称 唯一简短描述 - 允许主机上具有相同对象名称和属性名称的多个 JMX 监控项的唯一描述（可选） 有关 JMX 监控项的更多详细信息，请参见下文 从 Zabbix 3.4 开始，您可以使用 jmx.discovery[] 低级发现 项来发现 MBean 和 MBean 属性。
JMX 端点	您可以指定自定义 JMX 端点。确保 JMX 端点连接参数与 JMX 接口匹配。这可以通过在默认 JMX 端点中使用 {HOST.*} 宏来实现。 此字段自 3.4.0 起受支持。{HOST.*} macros 和用户宏被支持。
用户名	如果您在 Java 应用程序上配置了身份验证，请指定用户名。 支持用户宏。
密码	指定密码，如果您已在 Java 应用程序上配置身份验证。 支持用户宏。

如果您希望监视 “真” 或 “假” 的布尔计数器，则将信息类型指定为 “数字（无符号）” 并在 “预处理” 选项卡中选择 “布尔到十进制” 预处理步骤。服务器将布尔值分别存储为 1 或 0。

JMX 监控项详细信息

简单属性

MBean 对象名只不过是您在 Java 应用程序中定义的字符串。另一方面，属性名可能更复杂。如果一个属性返回原始数据类型 (整数，字符串等)，请不要担心，键参考以下例子：

```
jmx[com.example:Type=Hello,weight]
```

在这个例子中，一个对象的名称是”com.example:Type=Hello”，属性名是”weight”，可能返回值类型应该是”Numeric (float)”。

属性返回复合数据

当您的属性返回复合数据时，它变得更加复杂。例如：您的属性名称是”apple”，它返回一个表示其参数的哈希，如 “重量”、“颜色” 等。您的密钥可能看起来像这样：

```
jmx[com.example:Type=Hello,apple.weight]
```

这就是使用”.” 符号将属性名称和哈希键是如何分隔开。同样，如果一个属性返回嵌套的复合数据，那部分由”.” 分隔：

```
jmx[com.example:Type=Hello,fruits.apple.weight]
```

返回表格数据的属性

表格数据属性由一个或多个复合属性组成。如果在属性名参数中指定了这样一个属性，那么这个监控项值将以 JSON 格式返回该属性的完整结构。表格数据属性中的单个元素值可以使用预处理来检索。

表格数据属性示例:

```
 jmx[com.example:type=Hello,foodinfo]
```

监控项值示例:

```
[
  {
    "a": "apple",
    "b": "banana",
    "c": "cherry"
  },
  {
    "a": "potato",
    "b": "lettuce",
    "c": "onion"
  }
]
```

关于点的问题

当属性名或哈希键包含点，下面就是个例子:

```
jmx[com.example:Type=Hello,all.fruits.apple.weight]
```

如何告诉 Zabbix 属性名是”all.fruits”，而不只是 “all” 呢？如何区分作为属性名称一部分的点与分隔属性名和哈希键的点呢？这是一个问题。

在 **2.0.4** 版本之前，Zabbix Java Gateway 是无法处理此类情况的，在监控项里，用户只能留下 UNSUPPORTED 项了。从 2.0.4 开始解决了此问题，你所需要做的就是用反斜杠来转义名字的一部分点：

```
jmx[com.example:Type=Hello,all\.fruits.apple.weight]
```

同样，如果哈希键包含一个点，你也可以转义它：

```
jmx[com.example:Type=Hello,all\.fruits.apple.total\.weight]
```

其他问题

属性名中的反斜杠字符特应该被转义：

```
jmx[com.example:type=Hello,c:\\documents]
```

有关处理 JMX 监控项键值中的其他特殊字符，请参见this section。这就是全部了，希望可以快乐的使用 jMX 监控!

非基本数据类型

从 Zabbix 4.0.0 开始，可以使用返回的自定义 MBean 覆盖 **toString()** 方法的非基本数据类型。

在 JBoss EAP 6.4 中使用自定义锚点

自定义锚点允许使用默认 RMI 以外的不同传输协议。

为了说明这种可能性，让我们以配置 JBoss EAP 6.4 监控为例。首先，让我们做一些假设：

- 您已经安装了 Zabbix Java 网关。如果没有，那么你可以按照文档 执行。
- Zabbix server 和 Java gateway 都安装了前缀 /usr/local/
- JBoss 已经安装在 /opt/jboss-eap-6.4/ 并且正在运行独立模式
- 我们假设所有这些组件都在同一个主机上工作
- 防火墙和 SELinux 被禁用（或相应配置）

让我们在 zabbix_server.conf 中做一些简单的设置：

```
JavaGateway=127.0.0.1
StartJavaPollers=5
```

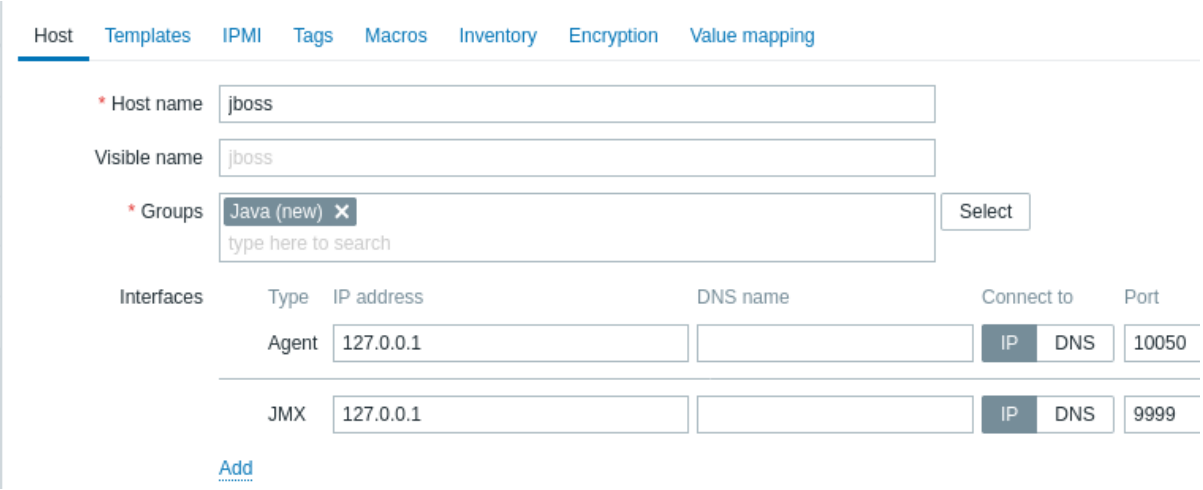
并在 zabbix_java/settings.sh 配置文件中（或 zabbix_java_gateway.conf）：

```
START_POLLES=5
```

检查 JBoss 是否监听其标准管理端口：

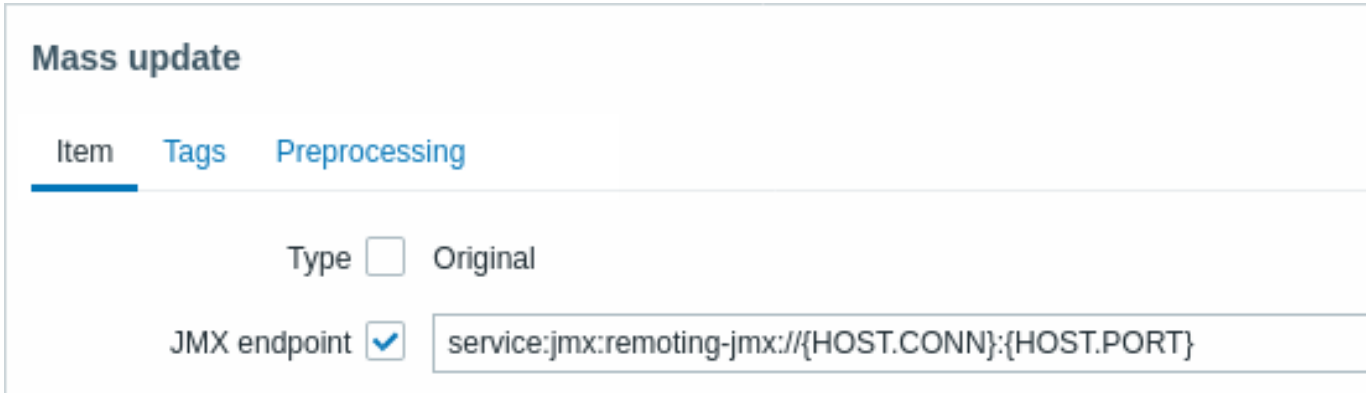
```
$ netstat -natp | grep 9999
tcp 0 0 127.0.0.1:9999 0.0.0.0:* LISTEN 10148/java
```

现在让我们在 Zabbix 中创建一个 JMX 接口为 127.0.0.1:9999 的主机。



我们知道这个版本的 JBoss 使用了 JBoss Remoting 协议而不是 RMI，我们可以为 JMX 模板中的监控项批量更新 JMX 锚点参数：

service:jmx:remoting-jmx://{HOST.CONN}:{HOST.PORT}



让我们更新配置缓存：

```
$ /usr/local/sbin/zabbix_server -R config_cache_reload
```

请注意，您可能首先会遇到错误。

```
3. mc [root@centos7-dev]:/home/vagrant/zabbix-3.2.6/src/zabbix_java (ssh)
com.zabbix.gateway.ZabbixException: java.net.MalformedURLException: Unsupported protocol: remoting-jmx
        at com.zabbix.gateway.JMXItemChecker.getValues(JMXItemChecker.java:97) ~[zabbix-java-gateway-3.4.2.jar:na]
        at com.zabbix.gateway.SocketProcessor.run(SocketProcessor.java:63) ~[zabbix-java-gateway-3.4.2.jar:na]
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149) [na:1.8.0_144]
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624) [na:1.8.0_144]
        at java.lang.Thread.run(Thread.java:748) [na:1.8.0_144]
Caused by: java.net.MalformedURLException: Unsupported protocol: remoting-jmx
        at javax.management.remote.JMXConnectorFactory.newJMXConnector(JMXConnectorFactory.java:359) ~[na:1.8.0_144]
        at javax.management.remote.JMXConnectorFactory.connect(JMXConnectorFactory.java:269) ~[na:1.8.0_144]
        at com.zabbix.gateway.ZabbixJMXConnectorFactory$1.run(ZabbixJMXConnectorFactory.java:76) ~[zabbix-java-gatewa
-3.4.2.jar:na]
        at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511) ~[na:1.8.0_144]
        at java.util.concurrent.FutureTask.run(FutureTask.java:266) ~[na:1.8.0_144]
        ... 3 common frames omitted
2017-11-07 13:52:12.644 [pool-1-thread-1] WARN  com.zabbix.gateway.SocketProcessor - error processing request
com.zabbix.gateway.ZabbixException: java.net.MalformedURLException: Unsupported protocol: remoting-jmx
        at com.zabbix.gateway.JMXItemChecker.getValues(JMXItemChecker.java:97) ~[zabbix-java-gateway-3.4.2.jar:na]
        at com.zabbix.gateway.SocketProcessor.run(SocketProcessor.java:63) ~[zabbix-java-gateway-3.4.2.jar:na]
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149) [na:1.8.0_144]
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624) [na:1.8.0_144]
        at java.lang.Thread.run(Thread.java:748) [na:1.8.0_144]
Caused by: java.net.MalformedURLException: Unsupported protocol: remoting-jmx
        at javax.management.remote.JMXConnectorFactory.newJMXConnector(JMXConnectorFactory.java:359) ~[na:1.8.0_144]
        at javax.management.remote.JMXConnectorFactory.connect(JMXConnectorFactory.java:269) ~[na:1.8.0_144]
        at com.zabbix.gateway.ZabbixJMXConnectorFactory$1.run(ZabbixJMXConnectorFactory.java:76) ~[zabbix-java-gatewa
-3.4.2.jar:na]
        at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511) ~[na:1.8.0_144]
        at java.util.concurrent.FutureTask.run(FutureTask.java:266) ~[na:1.8.0_144]
        ... 3 common frames omitted
2017-11-07 13:52:14.889 [Thread-0] INFO  com.zabbix.gateway.JavaGateway - Zabbix Java Gateway 3.4.2 (revision 72885)
as stopped
2017-11-07 13:52:26.167 [main] INFO  com.zabbix.gateway.JavaGateway - Zabbix Java Gateway 3.4.2 (revision 72885) has
tarted
```

” 不支持的协议: remoting-jmx” 表示 Java 网关不支持使用指定的协议。这可以通过创建具有以下内容的 ~/needed_modules.txt 文件：

```
jboss-as-remoting
jboss-logging
jboss-logmanager
jboss-marshalling
jboss-remoting
jboss-sasl
jcl-over-slf4j
jul-to-slf4j-stub
log4j-jboss-logmanager
remoting-jmx
slf4j-api
xnio-api
xnio-nio</pre>
```

然后执行命令：

```
$ for i in $(cat ~/needed_modules.txt); do find /opt/jboss-eap-6.4 -iname ${i}*.jar -exec cp {} /usr/local
```

于是，Java 网关将拥有所有必要的模块来使用 jmx 远程处理。剩下的就是重启 Java 网关，稍等一下，如果你做的一切都正确，将看到 JMX 监控数据开始出现在 Zabbix（另见：最新数据）。

14 ODBC 监控

概述

ODBC 监控对应于 Zabbix 前端中的 数据库监控监控项类型。

ODBC 是 C 语言编写的中间件 API，用于访问数据库管理系统 (DBMS)。ODBC 是由 Microsoft 开发的，后来被移植到了其它平台。

Zabbix 可以查询任何支持 ODBC 的数据库。实现这个目标，Zabbix 不直接连接数据库，而是使用 ODBC 接口并在 ODBC 中设置的驱动程序。该功能允许出于多种目的，更加有效地监视不同的数据库。例如，检测特定的数据库队列、使用统计信息等。Zabbix 支持 unixODBC，是最常用的开源 ODBC API 实现之一。

Attention:
更多信息请查看 ODBC 检查的已知问题 。

安装 unixODBC

安装 unixODBC 的建议方法是使用 Linux 操作系统默认包库，在主流的 Linux 发行版中，默认情况下 unixODBC 包含在镜像库中。如果找不到，可以在 unixODBC 主页上获取：http://www.unixodbc.org/download.html.

在 RedHat/Fedora 系统上使用 *yum* 包管理器安装 unixODBC：

```
shell> dnf -y install unixODBC unixODBC-devel
```

使用 *zypper* 包管理器在基于 SUSE 的系统上安装 unixODBC：

```
# zypper in unixODBC-devel
```

Note:
需要 unixODBC-devel 包来编译支持 unixODBC 的 Zabbix。

安装 unixODBC 驱动程序

应该为将被监控的数据库安装一个 unixODBC 数据库驱动程序。unixODBC 有一个支持的数据库和驱动程序列表：http://www.unixodbc.org/drivers.html。在某些 Linux 发行版中，数据库驱动程序包含在软件包存储库中。在 RedHat/Fedora 的系统上使用 *yum* 包管理器安装 MySQL 数据库驱动程序：

shell> dnf install mysql-connector-odbc

在 SUSE 的系统上使用 *zypper* 包管理器安装 MySQL 数据库驱动程序：

zypper in MyODBC-unixODBC

配置 unixODBC

ODBC 配置是通过编辑 **odbcinst.ini** 和 **odbc.ini** 文件完成的，要验证配置文件的位置，输入：

```
shell> odbcinst -j
```

odbcinst.ini 用于列出已安装的 ODBC 数据库驱动程序：

mysql Description = ODBC for MySQL Driver ·= /usr/lib/libmyodbc5.so

参数详情：

属性	描述
mysql	数据库驱动名
Description	数据库驱动描述
Driver	数据库驱动程序库文件位置

odbc.ini 用于自定义数据源：

[test] Description = MySQL test database Driver = mysql Server = 127.0.0.1 User = root Password = Port = 3306 Database = zabbix

参数详情：

属性	描述
test	数据源名称 (DSN)
Description	数据源描述
Driver	数据库驱动程序名称 - 在 odbcinst.ini 中指定
Server	数据库服务器 IP/DNS

属性	描述
User	用于连接的数据库用户
Password	数据库用户密码
Port	数据库连接端口
Database	数据库名称

要验证 ODBC 连接是否正常工作，应该测试到数据库的连接，这可以通过 isql 实用程序 (包含在 unixODBC 包中) 完成：

```
shell> isql test
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
SQL>
```

编译支持 ODBC 的 Zabbix

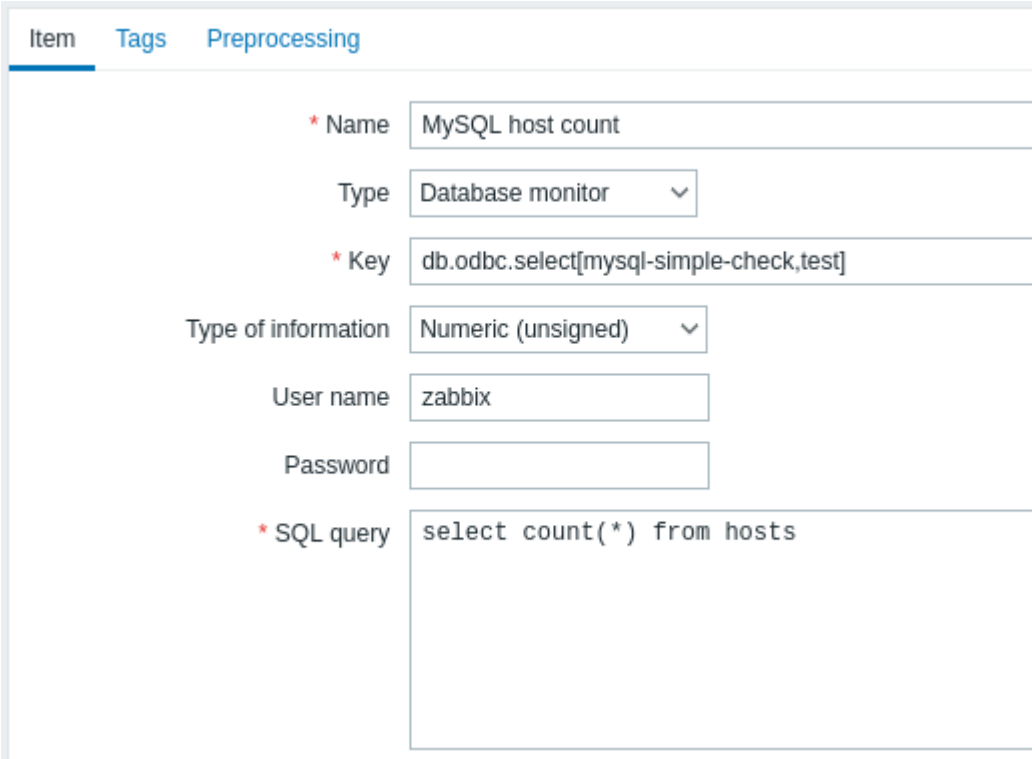
要启用 ODBC 支持，Zabbix 应该使用以下标志进行编译：

```
  --with-unixodbc[=ARG]   use odbc driver against unixODBC package
```

Note:
从源代码中了解有关 Zabbix 安装的更多信息。

Zabbix 前端的监控项配置

配置数据库监控监控项。



所有必填字段都标有红色星号

特别是对于数据库监控项，您必须输入：

类型	在此处选择数据库监控

键值	输入两个受支持的监控项键之一： **db.odbc.select**[<unique short description>,<dsn>,<connection string>] - 此监控项旨在返回一个值，即 SQL 查询结果第一行的第一列如果查询返回多列，则仅读取第一列如果查询返回多行，则仅读取第一行 **db.odbc.get**[<unique short description>,<dsn>,<connection string>] - 此项为能够以 JSON 格式返回多行/列。因此，它可以用作在一个系统调用中收集所有数据的主项，而 JSONPath 预处理可以用于从属项中以提取单个值有关详细信息，请参阅底层自动发现中使用的返回格式的示例。从 Zabbix 4.4 开始支持此监控项 唯一的描述将用于识别触发器等中的该监控项 虽然 `dsn` 和 `connection string` 是可选参数，但至少应该存在其中一个如果同时定义了数据源名称 (DSN) 和连接字符串，则 DSN 将被忽略 如果使用数据源名称，则必须按照 odbc.ini 中的指定设置 连接字符串可能包含驱动程序 - 特定参数 示例（MySQL ODBC 驱动程序 5 的连接）： => db.odbc.get[MySQL example,,"Driver=/usr/local/lib/libmyodbc5a.so;Database=master;Server=127.0
用户名	输入数据库用户名 如果在 odbc.ini 中指定了用户，则此参数是可选的 如果使用连接字符串，并且用户名字段不为空，则附加到连接字符串为 `UID=<user>`
密码	输入数据库用户密码 如果在 odbc.ini 中指定了密码，则此参数是可选的 如果使用连接字符串，并且 *Password* 字段不为空，则附加到连接中字符串为 `PWD=< 密码 >`
SQL 查询	输入 SQL 查询 请注意，对于 `db.odbc.select[]` 监控项，查询必须只返回一个值
信息类型	了解查询将返回什么类型的信息很重要，这样才能在此处正确选择信息类型，如果信息类型不正确，监控项将不受支持

注意事项

- Zabbix 不限制查询执行时间。用户可以选择在合理时间内执行的查询。
- Zabbix server 的Timeout参数值也用作于 ODBC 登陆超时时间（请注意，根据 ODBC 驱动，登录超时设置可能会被忽略）。
- 查询只能返回一个值。
- 如果查询返回多个列，则只读取第一列。
- 如果查询返回多行，则只读取第一行。
- SQL 命令必须以 `select` 开头。
- SQL 命令不能包含任何换行符。
- 另请参阅 ODBC 检查的已知问题

错误信息

ODBC 错误消息被构造成字段, 以提供详细信息. 例如：

```
Cannot execute ODBC query: [SQL_ERROR]:[42601][7][ERROR: syntax error at or near ";"; Error while executin

                                                  Native error code              Native error message
                                              SQLState
              Zabbix message     ODBC return code
```

请注意，错误消息长度限制为 2048 字节，因此信息可以被截断。如果有多个 ODBC 诊断记录，只要长度允许，Zabbix 将尝试把它们连接起来（用 丨 分隔）。

1 MySQL 推荐的 UnixODBC 设置

安装

- **Red Hat Enterprise Linux**:

  # dnf install mysql-connector-odbc

- **Debian/Ubuntu**:

请参考MySQL 文档下载相应平台所需的数据库驱动程序.

有关其他一些信息, 请参阅:安装 unixODBC.

配置

通过编辑 **odbcinstn.ini** 和 **odbcst.ini** 文件来完成 ODBC 配置。这些配置文件可以在*/etc* 文件夹下找到。可能缺少 **odbcinstn .ini** 文件，在这种情况下，需要手动创建它。

odbcinst.ini

```
[mysql]
Description = General ODBC for MySQL
Driver      = /usr/lib64/libmyodbc5.so
Setup       = /usr/lib64/libodbcmyS.so
FileUsage   = 1
```

请考虑以下 **odbc.ini** 配置示例参数。

- 通过 IP 连接的示例：

```
[TEST_MYSQL]
Description = MySQL database 1
Driver  = mysql
Port = 3306
Server = 127.0.0.1
```

- 通过 IP 连接并使用凭据，默认使用 Zabbix 数据库:

```
[TEST_MYSQL_FILLED_CRED]
Description = MySQL database 2
Driver  = mysql
User = root
Port = 3306
Password = zabbix
Database = zabbix
Server = 127.0.0.1
```

- 通过 socket 连接并使用凭据，默认使用 Zabbix 数据库：

```
[TEST_MYSQL_FILLED_CRED_SOCK]
Description = MySQL database 3
Driver  = mysql
User = root
Password = zabbix
Socket = /var/run/mysqld/mysqld.sock
Database = zabbix
```

其他配置和参数都可以参考MySQL 官方文档

2 PostgreSQL 数据库推荐的 UnixODBC 设置

安装

- **Red Hat Enterprise Linux**:

dnf install postgresql-odbc

- **Debian/Ubuntu**:

请参阅PostgreSQL 文档 为相应的平台下载必要的数据库驱动程序。

如需其他相关信息，请参考安装 unixODBC。

配置

通过编辑 **odbcinst.ini** 和 **odbc.ini** 文件来完成 ODBC 的配置。这些配置文件可以在 */etc* 文件夹中找到。**odbcinst.ini** 文件可能不存在，这时我们需要手动来创建它。

请考虑以下 **odbc.ini** 配置参数的示例：

odbcinst.ini

```
[postgresql]
Description = General ODBC for PostgreSQL
Driver      = /usr/lib64/libodbcpsql.so
Setup       = /usr/lib64/libodbcpsqlS.so
FileUsage   = 1
# Since 1.6 if the driver manager was built with thread support you may add another entry to each driver e
# This entry alters the default thread serialization level.
Threading   = 2
```

odbc.ini

```
[TEST_PSQL]
Description = PostgreSQL database 1
Driver  = postgresql
#CommLog = /tmp/sql.log
Username = zbx_test
Password = zabbix
# Name of Server. IP or DNS
Servername = 127.0.0.1
# Database name
Database = zabbix
# Postmaster listening port
Port = 5432
# Database is read only
# Whether the datasource will allow updates.
ReadOnly = No
# PostgreSQL backend protocol
# Note that when using SSL connections this setting is ignored.
# 7.4+: Use the 7.4(V3) protocol. This is only compatible with 7.4 and higher backends.
Protocol = 7.4+
# Includes the OID in SQLColumns
ShowOidColumn = No
# Fakes a unique index on OID
FakeOidIndex  = No
# Row Versioning
# Allows applications to detect whether data has been modified by other users
# while you are attempting to update a row.
# It also speeds the update process since every single column does not need to be specified in the where c
RowVersioning = No
# Show SystemTables
# The driver will treat system tables as regular tables in SQLTables. This is good for Access so you can s
ShowSystemTables = No
# If true, the driver automatically uses declare cursor/fetch to handle SELECT statements and keeps 100 ro
Fetch = Yes
# Bools as Char
# Bools are mapped to SQL_CHAR, otherwise to SQL_BIT.
BoolsAsChar = Yes
# SSL mode
SSLmode = Yes
# Send tobackend on connection
ConnSettings =
```

3 Oracle 推荐的 UnixODBC 设置

安装

请访问 Oracle 文档来获取详细说明。

如需其他相关信息，请参阅: 安装 unixODBC.

4 MSSQL 数据库推荐的 UnixODBC 设置

安装

- **Red Hat Enterprise Linux**:

·# dnf -y install freetds unixODBC

- **Debian/Ubuntu**:

请参阅FreeTDS 用户向导 来下载相应平台必要的数据库驱动。

如需其他相关信息，请参阅安装 unixODBC。

配置

通过编辑 **odbcinst.ini** 和 **odbc.ini** 文件来完成 ODBC 的配置。这些配置文件可以在 */etc* 文件夹中找到。**odbcinst.ini** 文件可能不存在，这时我们需要手动来创建它。

请考虑以下 **odbc.ini** 配置参数的示例：

odbcinst.ini

```
$ vi /etc/odbcinst.ini
[FreeTDS]
Driver = /usr/lib64/libtdsodbc.so.0
```

odbc.ini

```
$ vi /etc/odbc.ini
[sql1]
Driver = FreeTDS
Server = <SQL server 1 IP>
PORT = 1433
TDS_Version = 8.0
```

15 从属监控项

概述

有时一个监控项一次会收集多个指标，或者同时收集相关的指标会更有意义，例如：

- 单个内核的 CPU 利用率
- 输入/输出的总网络流量

为了允许进行批量指标收集和同时使用几个相关监控项，Zabbix 支持从属监控项。从属监控项依赖于主要监控项包含从属监控项的一次查询，主要监控项的新值会用于自动填充从属监控项的值。从属监控项的更新间隔与主要监控项相同且不可变。

Zabbix 预处理选项可用于从主要监控项的数据中，提取从属监控项所需部分的值。

预处理是由一个 “预处理管理器” 进程管理的，它被添加于 Zabbix 3.4 版本中，与 worker 进程一起执行预处理步骤。所有来自不同数据收集器的值（不管是否有预处理步骤），在添加到历史缓存之前，都要经过预处理管理器。基于套接字的 IPC 连接用于数据收集器 (pollers, trappers 等) 和预处理进程之间的通讯。

只有 Zabbix server 或 Zabbix proxy (如果主机被 Zabbix proxy 监控) 执行预处理步骤，并处理从属监控项。

任何类型的监控项，甚至是从属监控项，都可以设置为主要监控项。额外级别的从属监控项可以被用来从现有的从属监控项中提取出更小的值。

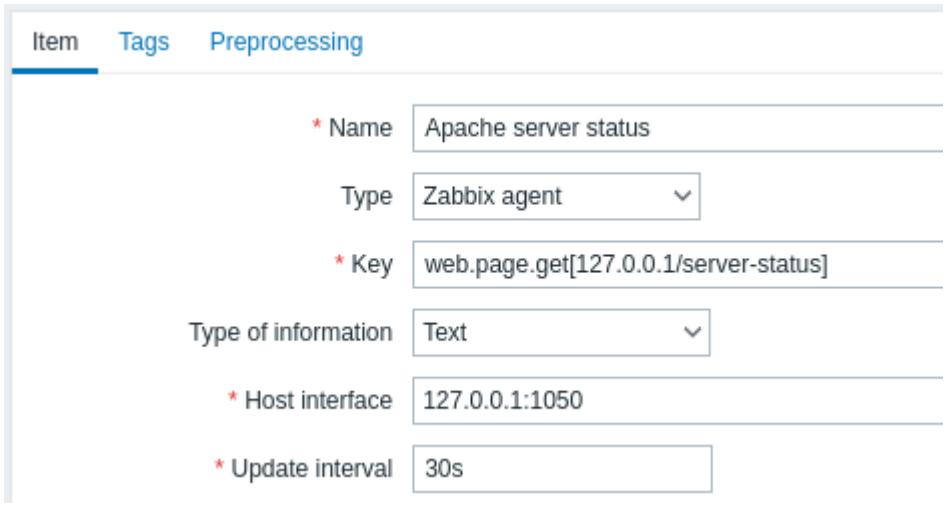
局限性

- 只允许相同的主机（模板）从属项
- 监控项原型可以依赖于来自同一主机的另一个监控项原型或常规监控项
- 主要监控项的从属项最大计数被限制为 29999（不考虑从属级别的数量）
- 最大允许 3 个从属级别
- 当从属监控项使用主机上来自模板的监控项作为主要监控项时，不允许导出为 XML

监控项配置

相关监控项依赖于主要监控项的数据，这就是为什么必须首先配置 主要监控项 (或主要监控项已存在)。

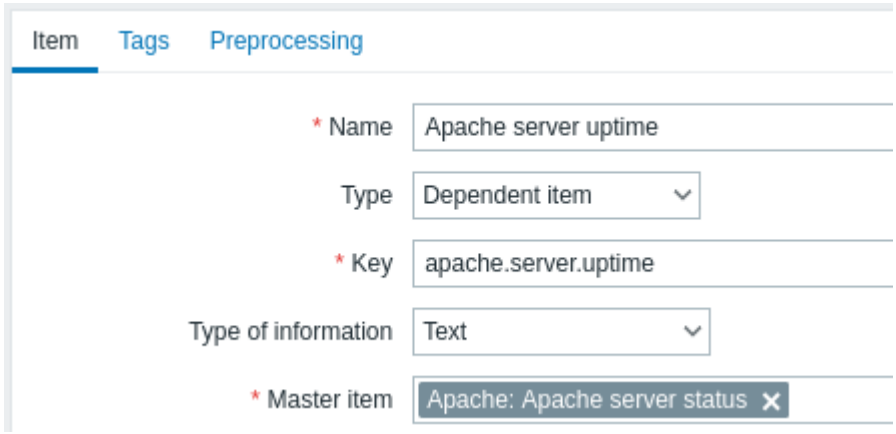
- 进入: 配置 → 主机
- 在主机那一行点击 监控项
- 点击 创建监控项
- 下表中输入监控项的参数



所有标有红色星号的为必填字段。

点击 添加保存主监控项。

接着，你可以配置 从属监控项。

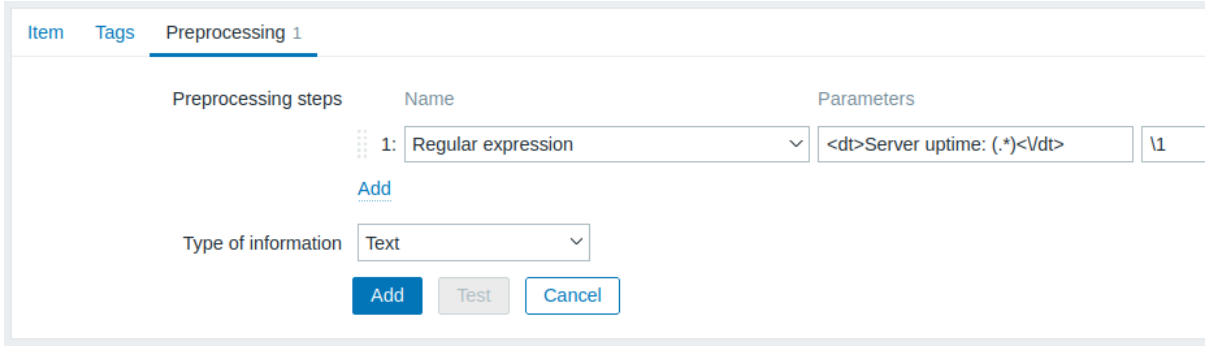


所有标有红色星号的为必填字段。

必要的相关监控项的特定信息的字段是：

类型	这里选择 **Dependent item** 。
键值	输入一个用于识别监控项的键。
主要项	选择主要监控项。主要监控项的值将用于填充从属监控项的值。
信息类型	选择与将要存储的数据格式相对应的信息类型

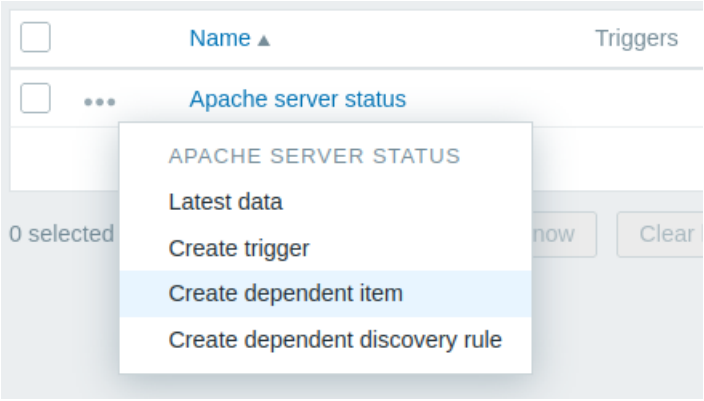
你可以使用监控项值的预处理 从主要监控项提取数据。



如果不进行预处理，相关监控项的值将与主要监控项的值完全相同。

点击 添加保存相关监控项。

单击 ••• 按钮可以访问更快地创建相关项的快捷方式，并在监控项列表中选择创建相关监控项。



展示

在监控项列表中，从属监控项以其主要监控项的名称作为前缀显示。

如果主监控项被删除，那么它的所有从属监控项也将会被删除。

16 HTTP 代理

概述

此监控项类型允许使用 HTTP/HTTPS 协议进行数据轮询。使用 Zabbix sender 或 Zabbix sender 协议也可以进行捕获。

HTTP 监控项检查由 Zabbix 服务器执行。但是，当主机由 Zabbix proxy 监控时，HTTP 项检查由 proxy 执行。

HTTP 监控项检查不需要任何 agent 运行在被监控的主机上。

HTTP agent 同时支持 HTTP 和 HTTPS。Zabbix 可以选择跟随重定向（参考下文 *Follow redirects* 的选项）。最大重定向数硬编码为 10（用 cURL 的参数 CURLOPT_MAXREDIRS）

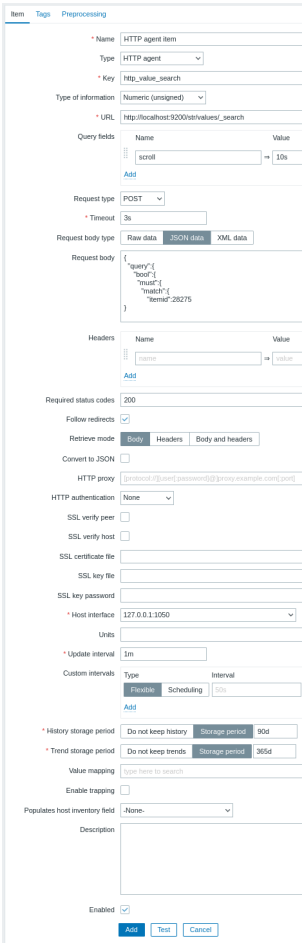
了解何时使用 HTTPS 协议，另请参阅已知问题

> **Attention:**
> Zabbix server/proxy 必须首先配置 cURL(libcurl) 支持。

配置

配置 HTTP 监控项：

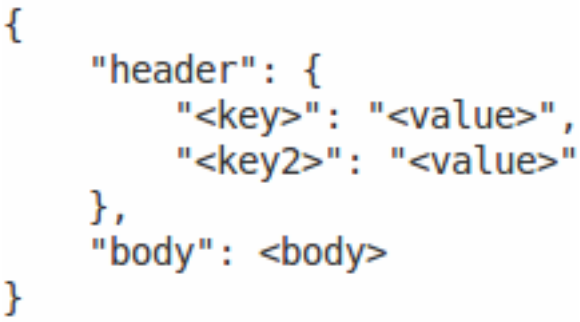
- 进入: 配置 → 主机
- 在主机的那行点击 监控项
- 点击 创建监控项
- 在表格中输入监控项的参数



所有标有红色星号的为必填字段。

需要的 HTTP 监控项特定信息的字段是：

参数	描述
类型	此处选择 **HTTP** 代理。
键值	输入用于识别监控项唯一性的键.
URL	连接和检索数据的 URL. 例如: https://www.example.com http://www.example.com/download 可以用 Unicode 字符指定域名。在执行 web 场景步骤时，它们将自动转换为 ASCII。 分析可以使用分析按钮将可选查询字段 (比如?name=Admin&password=mypassword) 与 URL 分离，将属性和值移动到查询字段中，以便自动 URL 编码. 限制在 2048 个字符。 支持的宏: {HOST.IP}, {HOST.CONN}, {HOST.DNS}, {HOST.HOST}, {HOST.NAME}, {ITEM.ID}, {ITEM.KEY}, {ITEM.KEY.ORIG}, 用户宏, 底层自动发现宏 这是设置CURLOPT_URL cURL 选项
查询字段	URL 的变量 (参见上文). 指定为属性和值对。 值是自动的 URL 编码。 从宏中解析值，然后自动编码 url 支持的宏: {HOST.IP}, {HOST.CONN}, {HOST.DNS}, {HOST.HOST}, {HOST.NAME}, {ITEM.ID}, {ITEM.KEY}, {ITEM.KEY.ORIG}, 用户宏, 底层自动发现宏. 设置 cURL 选项 CURLOPT_URL。
请求类型	选择请求的类型：*GET*, *POST*, *PUT* 或 *HEAD*
超时	Zabbix 不会花超过设定的时间来处理 URL (1~60 秒)。实际上，这个参数定义了连接 URL 的最大时间和执行 HTTP 请求的最大时间。因此，Zabbix 不会在一次检查中花费超过 2 倍的超时时间。 支持时间后缀, 例如 30s, 1m 支持的宏: 用户宏, 底层自动发现宏。 设置 cURL 选项 CURLOPT_TIMEOUT

参数	描述
请求体类型	设定请求体的类型 原始数据 - 自定义 HTTP 请求体，替换宏，但不执行编码。 **JSON** 数据 - HTTP 请求体是 JSON 格式的，宏可以用作字符串、数字、真和假; 用作字符串的宏必须包含在双引号中。从宏中解析值，然后自动转义。如果没有指定 header，那么服务器将把默认的 header 值设置为”Content-Type: application/json” **XML** 数据 - HTTP 请求体的 XML 格式。宏可以用作文本节点、属性或 CDATA 部分。从宏中解析值，然后在文本节点和属性中自动转义。如果没有指定 header ”Content-Type” 的值，那么服务器将把默认的 header 值设置为”Content-Type: application/xml” 注意选择 *XML* 数据，需要 libxml2 的支持。
请求体	输入请求体 支持的宏: {HOST.IP}, {HOST.CONN}, {HOST.DNS}, {HOST.HOST}, {HOST.NAME}, {ITEM.ID}, {ITEM.KEY}, {ITEM.KEY.ORIG}, 用户宏, 底层自动发现宏。
头	执行请求时将发送的自定义 HTTP 头。 指定为属性和值对。 支持的宏: {HOST.IP}, {HOST.CONN}, {HOST.DNS}, {HOST.HOST}, {HOST.NAME}, {ITEM.ID}, {ITEM.KEY}, {ITEM.KEY.ORIG}, 用户宏, 底层自动发现宏。 设置 CURLOPT_HTTPHEADER cURL 选项.
期望的状态码	期望的 HTTP 状态码的列表。如果 Zabbix 得到不在列表中的代码，那么这个监控项将变为不受支持状态。如果为空，则不执行检查。 例如: 200,201,210-299 列表里支持的宏: 用户宏, 底层自动发现宏。 这使用了 CURLINFO_RESPONSE_CODE cURL 选项。
跟随跳转	标记单选框使监控项跟随 HTTP 重定向。 这使用了 CURLOPT_FOLLOWLOCATION cURL 选项。
恢复模式	选择响应中必须检索的部分: **Body** - 仅主体 **Headers** - 仅头部 **Body and headers** - 主体和头部
转换为 *JSON*	头文件作为属性和值对保存在”header” 键下。 如果遇到’Content-Type: application/json’ 主体被保存为对象，否则它被存储为 string, 例如: 
HTTP 代理	可以使用格式 `[protocol://][username[:password]@]proxy.mycompany.com[:port]` 指定要使用的 HTTP 代理。 其中 `protocol://` 部分作为协议前缀可以用于指定代理协议 (例如 https, socks4, socks5; 查看 文档; 协议前缀的支持被添加于 cURL 7.21.7)。 如果不指定代理协议，将会视为使用 HTTP 代理。如果您指定了错误的协议，那么连接将失败且监控项将不受支持。由于没有指定协议，代理将被视为 HTTP 代理。 默认将使用 1080 端口。 如果指定本参数，代理信息将覆盖与代理相关的环境变量，如 http_proxy, HTTPS_PROXY 等。 如果本参数未指定，代理信息将不会覆盖与代理相关的环境变量。相关的值将会以 “原样” 传入，不会做健全性检查。 注意 HTTP 代理只支持简单的身份验证。 支持的宏: {HOST.IP}, {HOST.CONN}, {HOST.DNS}, {HOST.HOST}, {HOST.NAME}, {ITEM.ID}, {ITEM.KEY}, {ITEM.KEY.ORIG}, 用户宏, 底层自动发现宏。 设置 CURLOPT_PROXY cURL 选项。

参数	描述
HTTP 认证	验证类型: **None** - 不使用身份验证。 **Basic authentication** - 使用基本身份验证。 **NTLM authentication** - 使用 NTLM (Windows NT LAN Manager) 验证。 **Kerberos** - 使用 Kerberos 验证。详见为 Zabbix 配置 Kerberos **Digest** - 使用 Digest 验证。 选择身份验证方法后，展示输入用户名和密码提供两个额外字段的输入框，这里支持使用用户宏和底层自动发现宏。 设置 CURLOPT_HTTPAUTH cURL 选项。
SSL 验证对端	勾选复选框以开启配置需要验证 web 服务器的 ssl 证书。 服务器证书将自动从系统范围的证书颁发机构 (CA) 位置获取。可以使用 Zabbix 服务器或代理配置参数 SSLCALocation 重写 CA 文件的位置。 这设置的是 cURL 选项 CURLOPT_SSL_VERIFYPEER 。
SSL 验证主机	勾选复选框以开启验证 web 服务器证书的通用名称字段或主体备用名称字段是否匹配。 这设置的是 cURL 选项CURLOPT_SSL_VERIFYHOST 。
*SSL 证书文件 **	用于客户端认证的 SSL 证书文件名称。这个证书文件必须是 PEM[1] 格式的。 如果证书文件内包含私钥，则 SSL key file 字段请留空。如果密钥是加密的，需要设置 SSL key password 字段配置加密密钥的密码。检索文件的目录，需要在 Zabbix Server 或 Zabbix Proxy 的配置文件中指定 SSLCertLocation 参数。 支持的宏: {HOST.IP}, {HOST.CONN}, {HOST.DNS}, {HOST.HOST}, {HOST.NAME}, {ITEM.ID}, {ITEM.KEY}, {ITEM.KEY.ORIG}, 用户宏，底层自动发现宏。 这配置的是 cURL 选项CURLOPT_SSLCERT 。
SSL 密钥文件	用于客户端认证的 SSL 私钥文件名称。这个证书文件必须是 PEM[1] 格式的。检索文件的目录，需要在 Zabbix Server 或 Zabbix Proxy 的配置文件中指定 SSLCertLocation 参数。 支持的宏: {HOST.IP}, {HOST.CONN}, {HOST.DNS}, {HOST.HOST}, {HOST.NAME}, {ITEM.ID}, {ITEM.KEY}, {ITEM.KEY.ORIG}, 用户宏，底层自动发现宏。 这配置的是 cURL 选项 CURLOPT_SSLKEY 。
SSL 密钥密码	SSL 私钥文件密码。 支持的宏: 用户宏，底层自动发现宏。 这配置的是 cURL 选项 CURLOPT_KEYPASSWD 。
启用 *trapping*	勾选此复选框，此监控项也可以被作为trapper 监控项 ，也就是可以接受数据从 Zabbix sender 工具发送或使用 Zabbix sender 协议上报的数据。
允许的主机	仅在勾选启用 *trapping* 复选框后可见。 以逗号分隔的 IP 地址列表，也可使用 CIDR 表示法或主机名 如果本字段被指定，只有来自被列出的主机的入站连接才会被接受。 如果 IPv6 支持被打开'127.0.0.1', '::127.0.0.1', '::ffff:127.0.0.1' 会被等同处理，同时'::/0' 允许任何 IPv4 或 IPv6 地址。 '0.0.0.0/0' 可以被用来允许任意 IPv4 地址 注意，IPv4 兼容的 IPv6 地址 (0000::/96 prefix) 根据RFC4291已经被废弃。 示例： Server=127.0.0.1, 192.168.1.0/24, 192.168.3.1-255, 192.168.1-10.1-255, ::1,2001:db8::/32, zabbix.domain 空格和用户宏在此字段可以使用。 下列主机宏可以用于此字段: {HOST.HOST}, {HOST.NAME}, {HOST.IP}, {HOST.DNS}, {HOST.CONN} 。

Note:
如果 *HTTP* 代理字段留空，环境变量中设定的其他 HTTP 代理相关设置将被应用
对于 HTTP 访问 - 为 Zabbix server 进程用户设定 `http_proxy` 环境变量。如：
`http_proxy=http://proxy_ip:proxy_port`。
对与 HTTPS 访问 - 设定环境变量 `HTTPS_PROXY` 。如：
`HTTPS_PROXY=http://proxy_ip:proxy_port`。更多参考信息请运行 shell 命令：*# man curl* 。

Attention:
[1] Zabbix 仅支持 PEM 格式的证书与私钥文件。如果你拥有的是 PKCS#12 格式的密钥数据文件（通常使用扩展名 *.p12 或 *.pfx）你需要使用下列命令生成 PEM 格式文件：

```
openssl pkcs12 -in ssl-cert.p12 -clcerts -nokeys -out ssl-cert.pem
openssl pkcs12 -in ssl-cert.p12 -nocerts -nodes  -out ssl-cert.key
```

示例

示例 1

发送一个简单 GET 请求，以从其他服务获取数据。案例为 Elasticsearch：

- 创建一个从 URL `localhost:9200/?pretty` GET 的监控项
- 注意获得的相应内容：

```
{
  "name" : "YQ2VAY-",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "kH4CYqh5QfqgeTsjh2F9zg",
  "version" : {
    "number" : "6.1.3",
    "build_hash" : "af51318",
    "build_date" : "2018-01-26T18:22:55.523Z",
    "build_snapshot" : false,
    "lucene_version" : "7.1.0",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You know, for search"
}
```

- 现在可以使用 JSONPath 预处理步骤获取版本号信息：`$.version.number`

示例 2

发送一个简单 POST 请求，从其他服务抓取数据。以 Elasticsearch 为例：

- 创建一个 POST 监控项使用此 URL: `http://localhost:9200/str/values/_search?scroll=10s`
- 配置以下的 POST 请求体获取处理器负载（1 分钟平均值/核心）

```
{
    "query": {
        "bool": {
            "must": [{
                "match": {
                    "itemid": 28275
                }
            }],
            "filter": [{
                "range": {
                    "clock": {
                        "gt": 1517565836,
                        "lte": 1517566137
                    }
                }
            }]
        }
    }
}
```

- 得到响应:

```
{
    "_scroll_id": "DnF1ZXJ5VGhlbkZldGNoBQAAAAAAAAAkFllRMlZBWS1UU1pxTmdEeGVwQjRBTFEAAAAAAAAAJRZZUTJWQVktVFN
    "took": 18,
    "timed_out": false,
    "_shards": {
        "total": 5,
        "successful": 5,
        "skipped": 0,
        "failed": 0
    },
    "hits": {
        "total": 1,
```

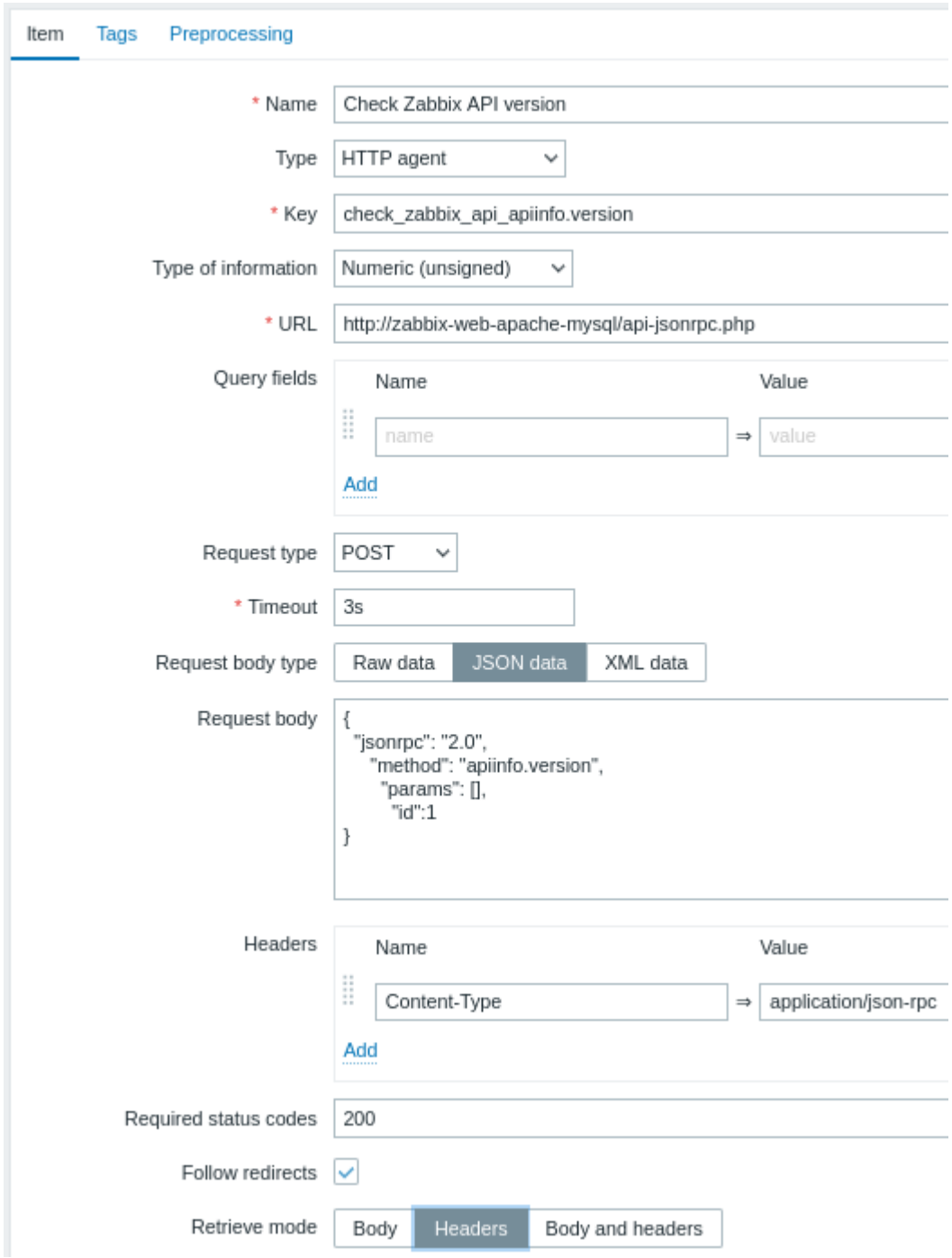
```
        "max_score": 1.0,
        "hits": [{
            "_index": "dbl",
            "_type": "values",
            "_id": "dqX9VWEBV6sEKSMyk6sw",
            "_score": 1.0,
            "_source": {
                "itemid": 28275,
                "value": "0.138750",
                "clock": 1517566136,
                "ns": 25388713,
                "ttl": 604800
            }
        }]
    }
}
```

- 现在可以使用 JSONPath 预处理步骤获取监控项的值：`$.hits.hits[0]._source.value`

示例 3

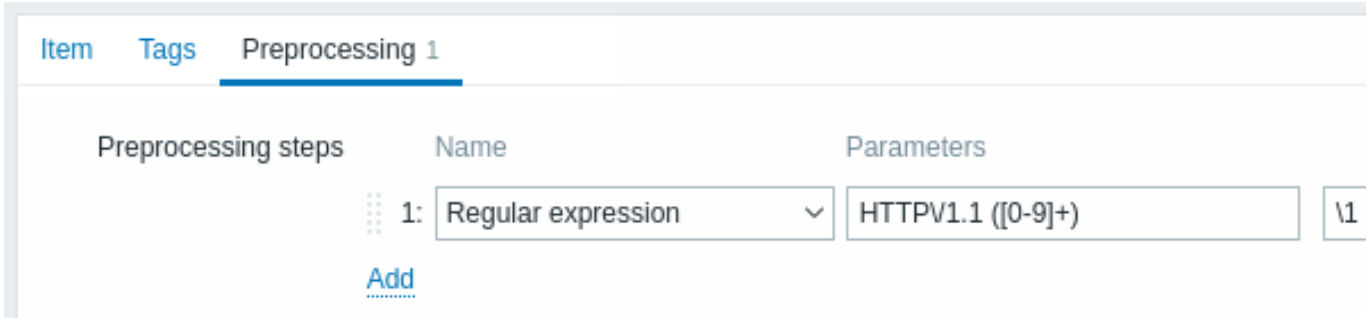
检查 Zabbix API 是否可用，可以用apiinfo.version 这个接口

- 监控项配置:

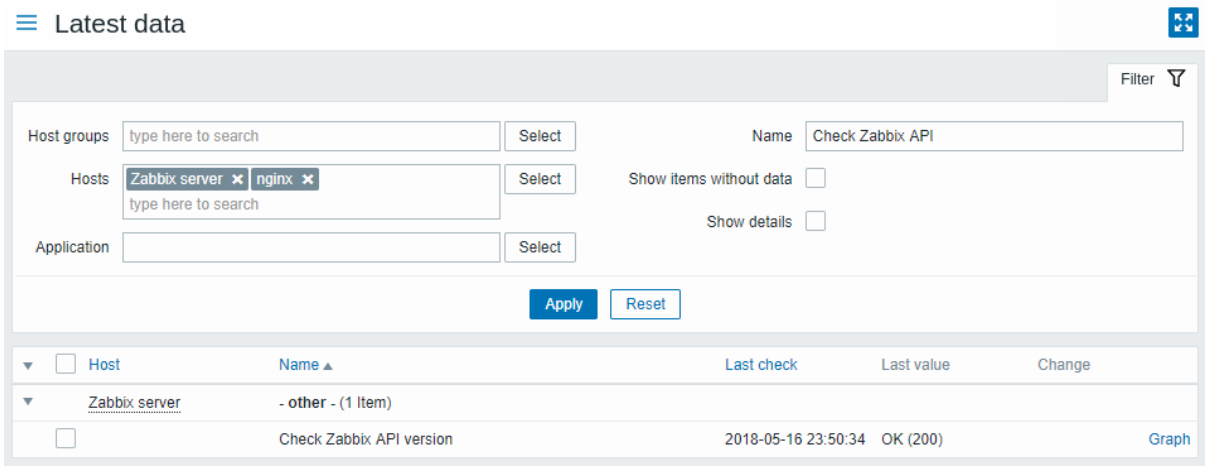


注意使用 POST 方法传输 JSON 数据，设定请求头并仅要求检索头信息：

- 配置监控项值的预处理功能，使用正则表达式方法获取 HTTP 响应代码：



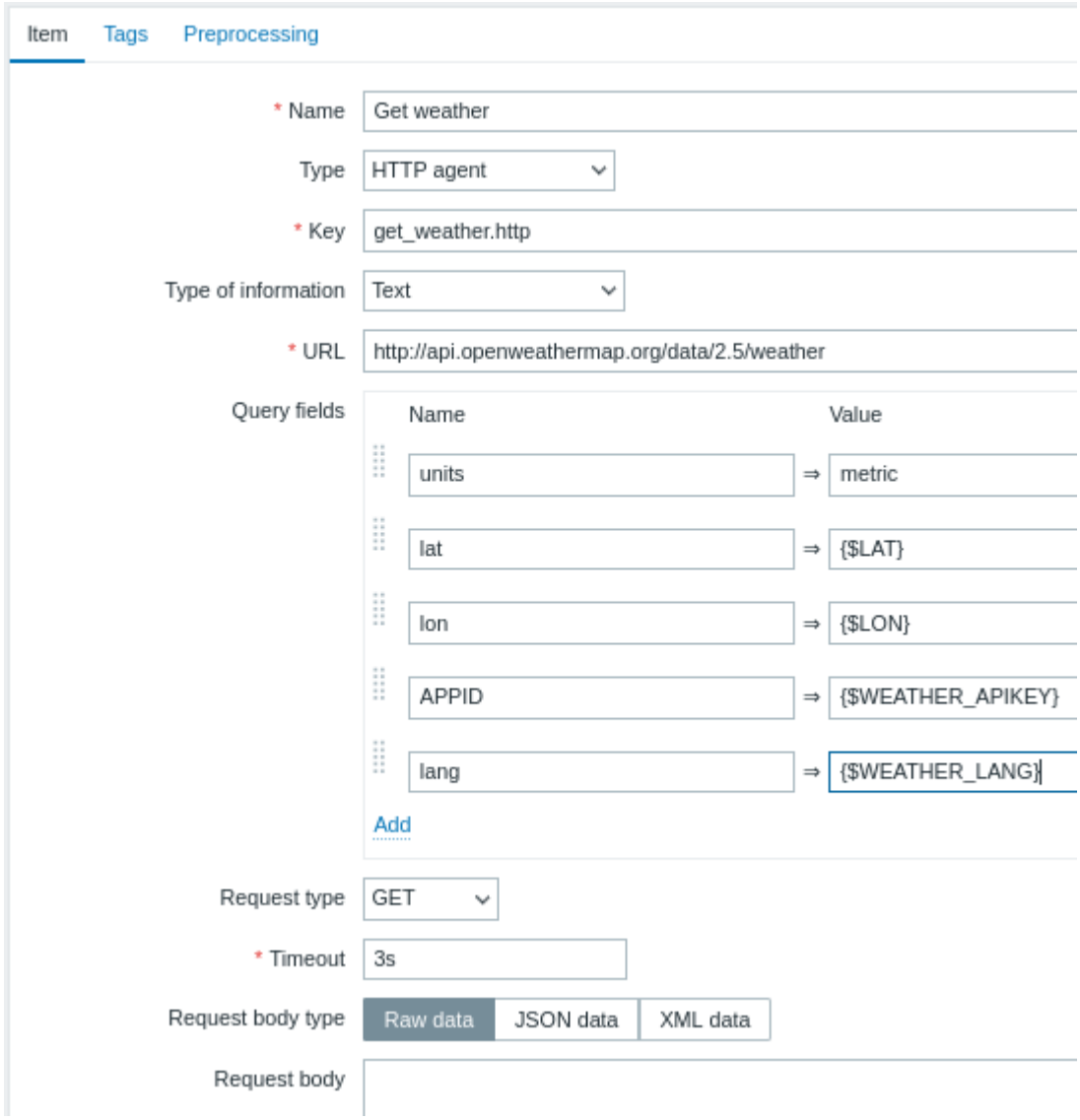
- 在 最新数据中检查最新获取的数据：



示例 4

通过连接到 Openweathermap 检索天气信息公共服务。

- 配置主要监控项将大量数据收集到一个简单 JSON 对象：



注意在 query 的字段中使用宏。参考 Openweathermap API 文档 来了解如何填写它们。

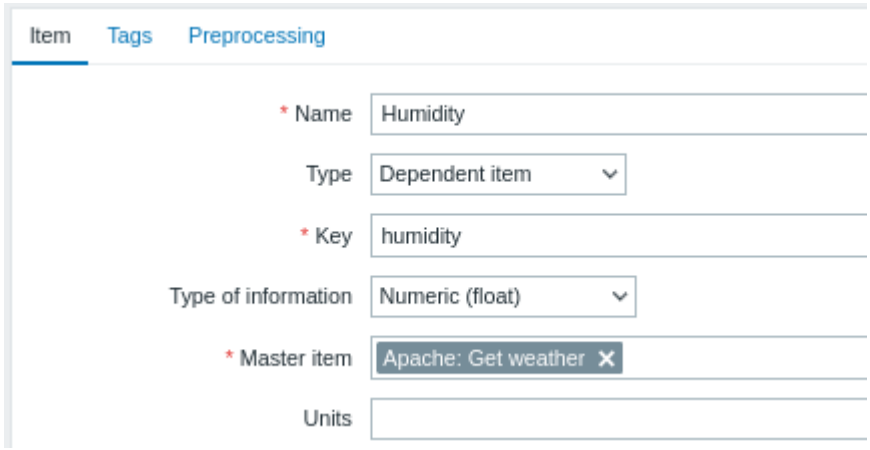
一个简单 JSON 对象会被返回给 HTTPS agent：

```
{
    "body": {
        "coord": {
            "lon": 40.01,
```

```
            "lat": 56.11
        },
        "weather": [{
            "id": 801,
            "main": "Clouds",
            "description": "few clouds",
            "icon": "02n"
        }],
        "base": "stations",
        "main": {
            "temp": 15.14,
            "pressure": 1012.6,
            "humidity": 66,
            "temp_min": 15.14,
            "temp_max": 15.14,
            "sea_level": 1030.91,
            "grnd_level": 1012.6
        },
        "wind": {
            "speed": 1.86,
            "deg": 246.001
        },
        "clouds": {
            "all": 20
        },
        "dt": 1526509427,
        "sys": {
            "message": 0.0035,
            "country": "RU",
            "sunrise": 1526432608,
            "sunset": 1526491828
        },
        "id": 487837,
        "name": "Stavrovo",
        "cod": 200
    }
}
```

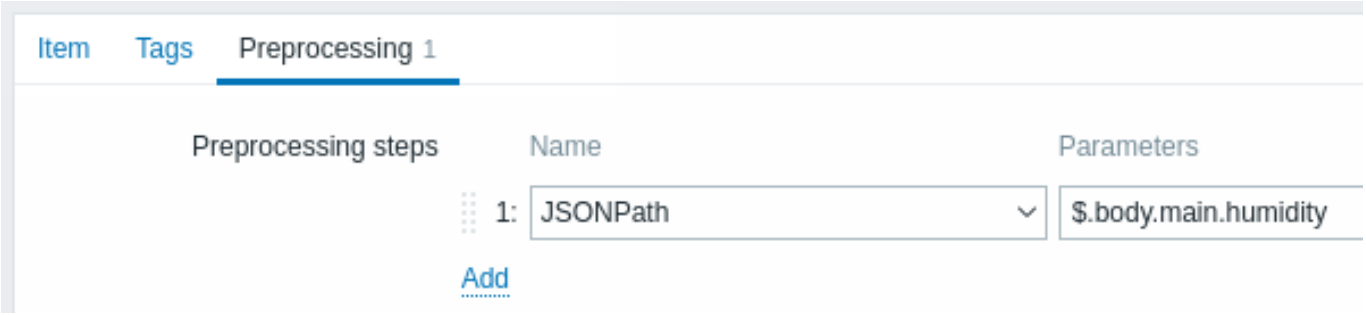

下一个任务是配置从属监控项从这个 JSON 对象中导出必要的数据。

- 配置一个简单的从属监控项获取湿度数据：

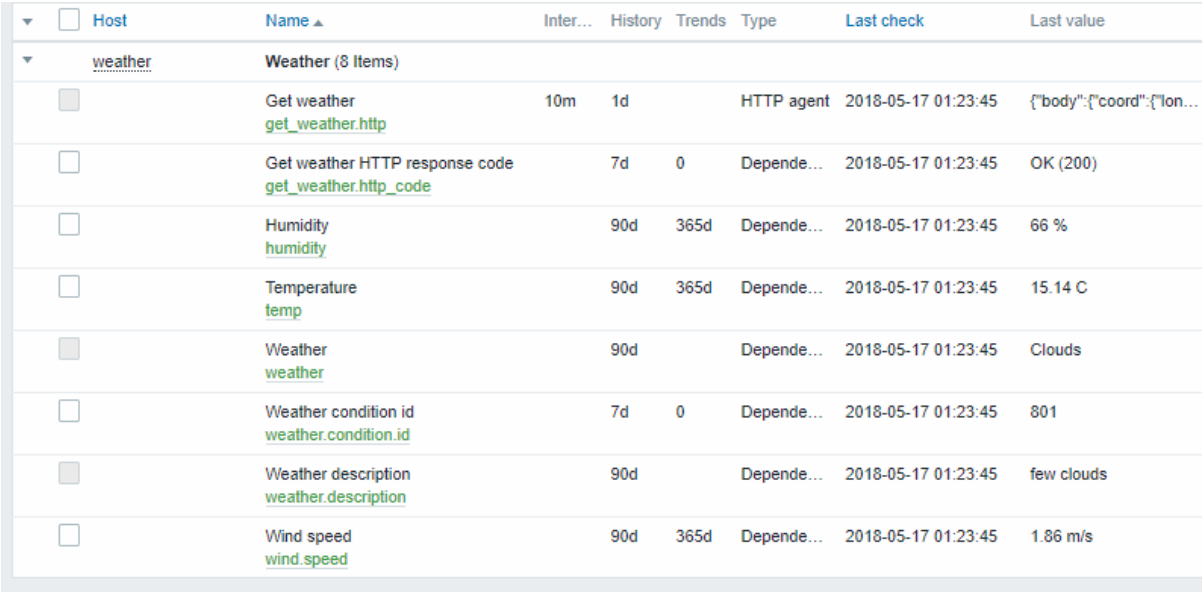


其他天气指标，例如 “温度” 可以以相同方式添加。

- 使用 JSONPath 来配置简单的从属监控项的值预处理步骤：



- 在 最新数据中检查天气数据的结果：

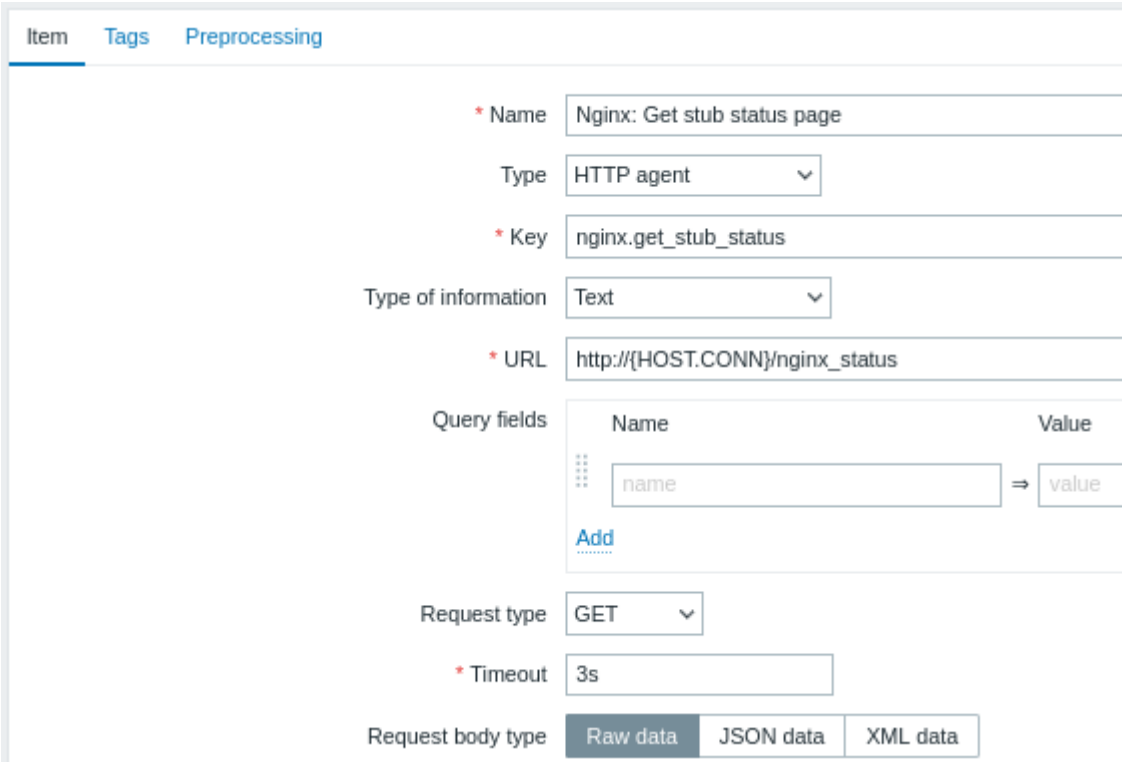


Host	Name ▲	Inter...	History	Trends	Type	Last check	Last value
weather	**Weather** (8 Items)						
	Get weather get_weather.http	10m	1d		HTTP agent	2018-05-17 01:23:45	{"body":{"coord":{"lon...
	Get weather HTTP response code get_weather.http_code		7d	0	Depende...	2018-05-17 01:23:45	OK (200)
	Humidity humidity		90d	365d	Depende...	2018-05-17 01:23:45	66 %
	Temperature temp		90d	365d	Depende...	2018-05-17 01:23:45	15.14 C
	Weather weather		90d		Depende...	2018-05-17 01:23:45	Clouds
	Weather condition id weather.condition.id		7d	0	Depende...	2018-05-17 01:23:45	801
	Weather description weather.description		90d		Depende...	2018-05-17 01:23:45	few clouds
	Wind speed wind.speed		90d	365d	Depende...	2018-05-17 01:23:45	1.86 m/s

示例 5

链接 Nginx 状态页面，批量获取它的指标。

- 根据 官方指南配置 NGINX。
- 配置一个主要监控项批量收集数据:

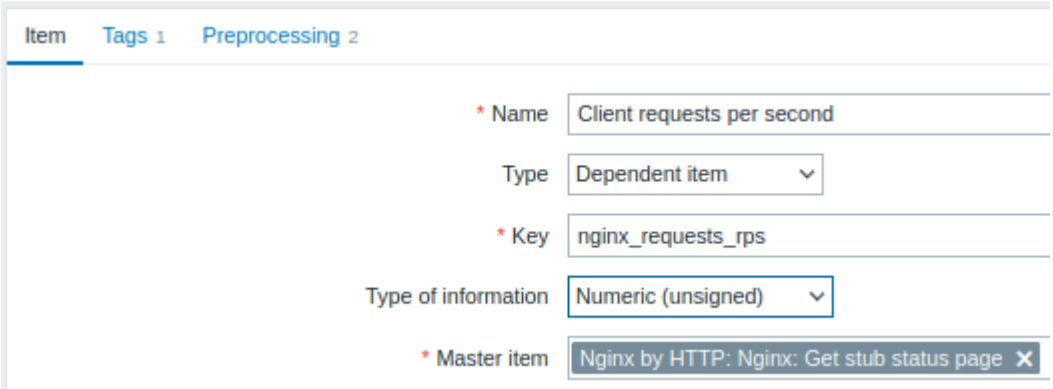


简单的 Nginx 状态也输出如下：

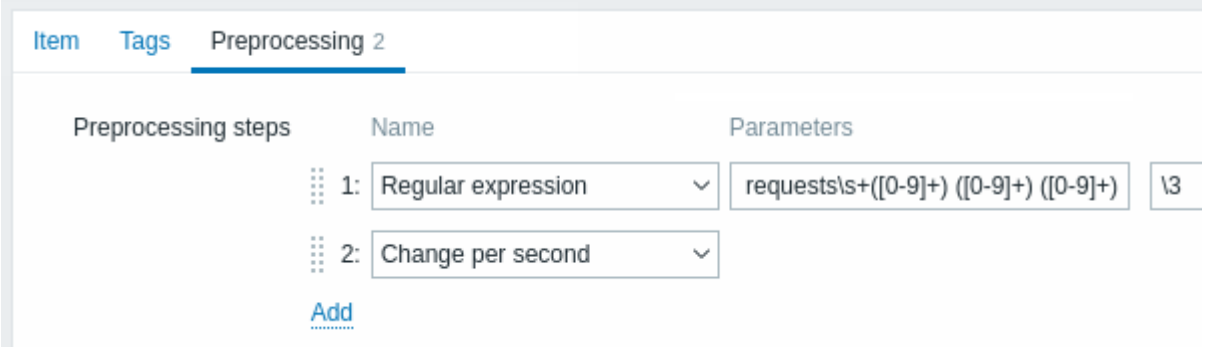
```
Active connections: 1 Active connections:
server accepts handled requests
 52 52 52
Reading: 0 Writing: 1 Waiting: 0
```

下一个任务是任务是配置从属监控项提取数据。

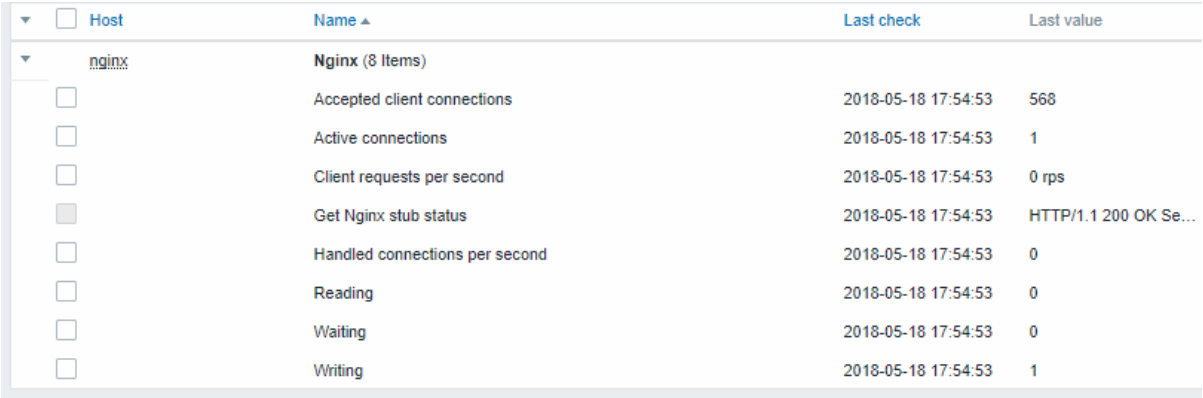
- 配置简单从属监控项名以采集每秒请求数数据：



- 这个简单从属监控项，需要配置正则表达式预处理步骤 `server accepts handled requests\s+([0-9]+) ([0-9]+) ([0-9]+)`:



- 对所有数据都配置好了从属监控项后，在 最新数据中，可以查看到来自 stub 模块的完整结果



17 Prometheus 数据处理

概述

Zabbix 可以采集符合 Prometheus 行协议格式的监控数据。

采集数据需要配置以下两步：

- 创建一个作为主要监控项的HTTP 监控项 指向合适的站点批量采集数据。例如：`https://<prometheus host>/metrics`
- 配置使用 Prometheus 预处理选项的从属监控项，来从主要监控项中获取需要的数据

Zabbix 有两个 Prometheus 预处理选项：

- *Prometheus* 正则匹配 - 可以应用于一般监控项中，用来从查询 Prometheus 行协议中提取数据

- *Prometheus* 转 *JSON* - 可以应用于一般监控项或低级别自动发现监控项。使用这个预处理步骤，将会将采集到的 Prometheus 数据转化为 JSON 格式返回给监控项

批量处理

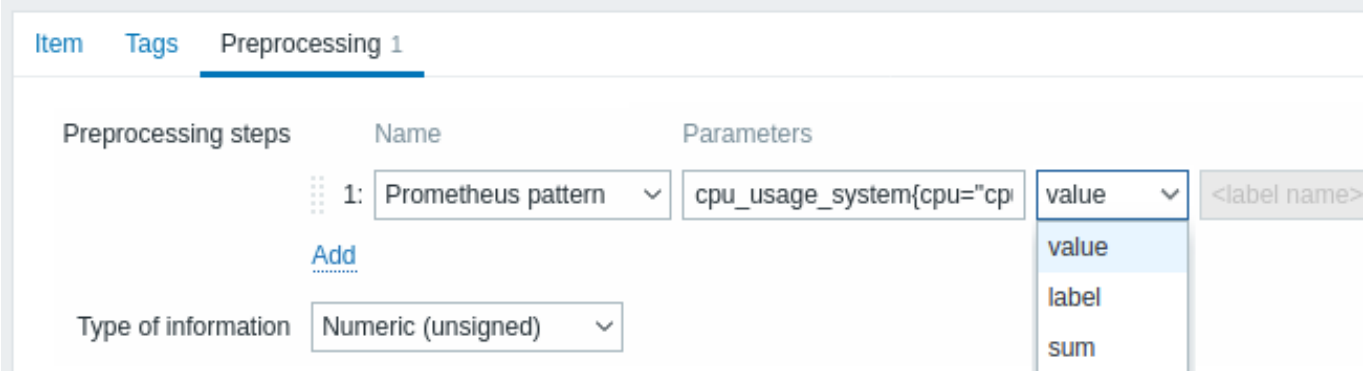
从属监控项支持批量处理。为了启用缓存和索引，是 *Prometheus* 正则匹配步骤必须是预处理配置的第一个步骤。当 *Prometheus* 正则匹配是预处理的第一步时，经过这个预处理步骤配置的第一组 `<label>==<value>` 条件解析后的 Prometheus 数据会被缓存并建立索引。这个缓存可以在其他从属监控项的其他后续预处理过程中得到重用。为了更好的优化性能，第一个作为条件的 label，应当匹配尽可能多的不同值。

如果有额外的预处理步骤需要在第一步之前执行，正确做法是将该步骤放在主要监控项或者新建一个从属监控项，并将这个监控项作为处理 Prometheus 数据处理监控项的主监控项。

配置

已经事先正确配置了 HTTP 代理类型的主要监控项，你需要创建一个相关监控项 来应用 Prometheus 预处理步骤：

- 在配置表单中填入一般从属监控项的参数
- 前往预处理选项卡
- 选择一个 **Prometheus** 预处理选项（*Prometheus* 正则匹配 or *Prometheus* 转 *JSON*）



以下参数特定于 *Prometheus* 正则匹配预处理选项：

参数	描述	示例
Pattern	要定义所需数据的正则匹配规则，可以使用与 PromQL（Prometheus 查询语言）十分类似的检索语言 (详见 [对比表]](#query_language_comparison)), 举个例子： <metric name> - 指定指标名称 {__name__=~"<regex>"} - 根据正则表达式，匹配指标名称 {<label name>="<label value>",...} - 指定标签名称 {<label name>=~"<regex>",...} - 根据正则表达式，匹配标签名称 {__name__=~".*"}==<value> - 指定指标值 或者将以上的条件任意组合： <metric name>{<label1 name>="<label1 value>",<label2 name>=~"<regex>",...}==<value> 标签名称可以是任何 UTF-8 字符的序列，但对于反斜杠、双引号和换行符必须使用转义，就像 \\，\"，\n；其他字符不应被转义。	*wmi_os_physical_memory_free_bytes* *cpu_usage_system{cpu="cpu-total"}* *cpu_usage_system{cpu=~".*"}* *cpu_usage_system{cpu="cpu-total",host=~".*"}* *wmi_service_state{name="dhcp"}==1* *wmi_os_timezone{timezone=~".*"}==1*

参数	描述	示例
Result processing	定义如何返回处理结果，提供了值，标签或应用适当的函数（如正则匹配的到了多行结果，需要对这些结果进行简单聚合）： 值 - 返回指标的值（当匹配到多行记录时，会引发错误） 标签 - 在标签字段中指定的标签返回值（当匹配到多个指标时，会引发错误） 总和 - 返回值的总和 最小值 - 返回最小值 **max** - 返回最大值 平均值 - 返回平均值 计数 - 返回值的计数 此参数字段仅适用于 **Prometheus** 正则匹配预处理步骤。	实际用例见表后。
Output	定义标签名称（可选）。在这种情况下，返回与标签名称对应的值。 如果在 *Result processing* 字段中选择了“Label”，则此字段仅适用于 *Prometheus pattern* 选项。	

Prometheus 转 JSON

来自 Prometheus 的数据可以被低级别自动发现使用。要这样使用数据，必须要转化为 JSON 格式。Zabbix 提供了 *Prometheus* 转 *JSON* 作为预处理步骤的选项，这可以直接按照格式要求返回数据。

详情参见在低级别自动发现中，使用 Prometheus 数据。

查询语言对照表

下表列出了 PromQL 与 Zabbix Prometheus 预处理查询语言之间的异同。

	PromQL 瞬时向量选择器	Zabbix Prometheus 预处理
不同点		
查询目标	Prometheus 服务器	Prometheus 行协议中的纯文本
返回值	瞬时向量	指标或标签的值（Prometheus 正则匹配） JSON 格式的包含单个值的指标构成的数组（Prometheus 转 JSON）
标签匹配操作符	**=**, **!=**, **=~**, **!~**	**=**, **!=**, **=~**, **!~**
匹配标签或指标名称的正则表达式	RE2	PCRE
比较操作符	详见清单	只有 **==** (相等) 用于支持值过滤
相似点		
依据指标名相同的字符串筛选	<metric name> or {__name__=”<metric name>”}	<metric name> or {__name__=”<metric name>”}
依据匹配正则表达式的指标名筛选	{__name__=~”<regex>”}	{__name__=~”<regex>”}
依据相同字符串的标签名称值筛选	{<label name>=”<label value>”,...}	{<label name>=”<label value>”,...}
依据匹配正则表达式的标签名称值筛选	{<label name>=~”<regex>”,...}	{<label name>=~”<regex>”,...}
根据相同的字符串的值筛选	{__name__=~”.*”} == <value>	{__name__=~”.*”} == <value>

18 脚本监控项

概览

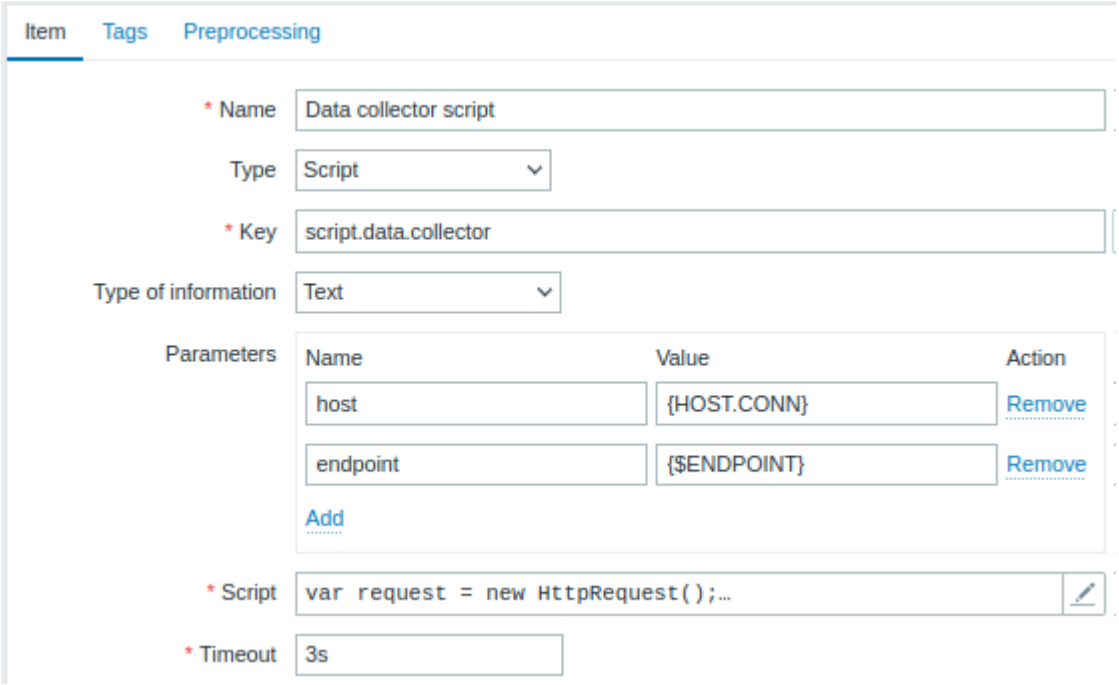
脚本监控项可以通过执行用户自定义的 JavaScript 代码，检索 HTTP/HTTPS 的方式来收集数据。除了脚本外，可以指定一个可选的参数列表（一些键值对）并配置超时限制。

此监控项类型在有收集数据过程中需要多个步骤或复杂逻辑的场景非常有用。举个例子，一个脚本监控项可以被配置为执行一个 HTTP 调用，然后经过某些方式处理从第一步调用得到的数据，并将转换后的数值传递给第二个 HTTP 调用。

脚本监控项由 Zabbix server 或 Zabbix proxy 的轮询器处理。

配置

在监控项配置表单的 类型字段中选择“脚本”，然后填写必填的字段。



所有必须输入字段都用红色星号标记。

脚本监控项需要特定信息的字段是:

字段	描述
键值	输入一个用于标识监控项的唯一值。
参数	指定要作为属性和键值对传递给脚本的变量。 宏变量 {HOST.CONN}, {HOST.DNS}, {HOST.HOST}, {HOST.IP}, {HOST.NAME}, {ITEM.ID}, {ITEM.KEY}, {ITEM.KEY.ORIG} 和用户宏 是支持的。
脚本	在单击 “脚本” 参数字段时出现的块中输入 JavaScript 代码 (或在它旁边的查看/编辑按钮上)。此代码必须提供返回指标值的逻辑。 这段代码可以访问所有的参数，它可以执行 HTTP GET, POST, PUT 和 DELETE 请求，并控制 HTTP 头和请求体. 另请参考：额外的 JavaScript 对象, JavaScript 手册.
超时	JavaScript 执行超时 (1-60s, 缺省值是 3s)；超过该值将返回错误. 支持时间后缀，比如 30s, 1m。 根据脚本的不同，触发超时可能需要更长的时间。

示例

简单的数据采集

从 *https://www.example.com/release_notes* 页面收集内容:

- 创建一个监控项，类型选择” 脚本 “。
- 在脚本字段填写下面的代码：

```
var request = new HttpRequest();
return request.get("https://www.example.com/release_notes");
```

带参数收集数据

使用宏 {HOST.CONN} 作为参数值并获取响应：

- 创建一个监控项，类型选择” 脚本 “。
- 创建一个参数：
  Name： host
  Value: {HOST.CONN}
- 在脚本字段填写下面的代码：

```
var request = new HttpRequest();
return request.post("https://postman-echo.com/post", JSON.parse(value));
```

多个 HTTP 请求

同时从 *https://www.example.com* 和 *https://www.example.com/release_notes* 两个站点收集数据:

- 创建一个监控项，类型选择” 脚本 “。
- 在脚本字段填写下面的代码：

```
var request = new HttpRequest();
return request.get("https://www.example.com") + request.get("https://www.example.com/release_notes");
```

记录日志

添加”Log test” 到 Zabbix server 日志并返回数值”1” 给监控项：

- 创建一个监控项，类型选择” 脚本 “。
- 在脚本字段填写下面的代码：

```
Zabbix.log(3, 'Log test');
return 1;
```

4 历史数据与趋势数据

概述

历史数据（history）和趋势数据（trends）是 Zabbix 中存储收集到的数据的两种方式。

历史数据：每一个收集到的监控数据，趋势数据：按小时统计计算的平均值数据。

历史数据的留存

通过设置历史数据保留时长，可以指定历史数据留存的时长。在以下位置，你可以找到相关的输入框：

- 监控项属性表单
- 批量更新监控项
- 管家配置页

任何过旧的历史数据会被管家从数据库中删除。

一般来讲，强烈建议将历史数据保留时长设置得尽可能的小。这么做可以让数据库不会因存储了大量的历史数据，导致超负荷运行。

可以选择长时间的保留趋势数据，来替代长期需要的历史数据。例如：设置成保留 14 天历史数据和 5 年的趋势数据。

参考数据库空间大小页，来了解历史数据和趋势数据各自需要的数据库空间。

当设置了较短的历史数据保留时间，图形会使用趋势数据值显示旧数据，因此依旧可以通过图形查看旧数据。

> **Attention:**
> 如果历史数据保留时长被设置为 “0”，只有相关项目和资产记录会被更新。不会计算触发器函数，因为触发器计算是基于历史数据的。

> **Note:**
> 作为保存历史数据的替代方法，考虑使用可加载模块导出历史数据 功能。

趋势数据的留存

趋势数据是一种内建的历史数据压缩机制，可以用来存储数字类型监控项的每小时的最小值、最大值、平均值和记录数量。

通过设置趋势存储时间，可以指定趋势数据留存的时长。在以下位置，你可以找到相关的输入框：

- 监控项属性表单
- 批量更新监控项
- 管家配置页

通常趋势数据设置的的留存时间应当比历史数据留存时间设置的长。任何过旧的趋势数据会被管家从数据库删除。

Zabbix server 在运行时会在趋势缓存中积累趋势数据，因为有数据流入。在这些情况下，Zabbix Server 会将每个监控项的前一个小时趋势数据写入数据库 (在前端可以看到):

- 服务器接收到监控项首个当前一小时的值
- 在还差 5 分钟或更少时间达到一小时，仍然没有该监控项当前一小时的值
- 服务器停止

要查看图表上的趋势，你需要至少等待到下一个小时的开始 (如果监控项经常更新)，最多等待到下一个小时的结束 (如果监控项很少更新)，最多 2 个小时。

当服务器刷新趋势缓存时，如果数据库中已经有这一小时的趋势 (例如，服务器在这一小时中已经重新启动)，服务器需要使用更新语句，而不是简单的插入。因此，在大型环境的安装中，如果需要重新启动，最好在一个小时结束时停止服务器，在下一个小时开始时开始，以避免趋势数据重叠。

趋势数据生成和历史表没有关系。

Attention:
如果趋势存储时间被设置为 “0”，Zabbix server 将不再计算或存储该监控项的趋势数据

Note:
趋势数据的计算和存储将会使用与原值相同的数据类型。无符号数字（unsigned Numeric）数据类型的值，平均值计算的结果小数点后会被舍去，所以记录值之间的间隔越小，计算结果结果将会精确度越低。例如：如果监控项的得到了得到了两个值，分别是 “0” 和 “1”，那么平均值的计算结果将会是 “0”，而不是 “0.5”。
此外，重启服务器可能会导致当前小时无符号数字类型的数据，平均值计算的精度损失。

5 用户自定义参数

概述

用户定义参数可以用来帮助用户实现通过 Zabbix agent 执行非 Zabbix 原生的 agent check。

你可以编写一个命令来检索所需的数据，并将其包含在用户自定义参数agent 配置文件中 (’UserParameter’ 参数配置)。

用户自定义参数配置语法如下：

```
UserParameter=<key>,<command>
```

如你所见，一条用户自定义参数除了命令部分，还包括一个 key。这个 key 将在配置监控项时使用。输入你觉得容易引用的 key（key 在一台主机中必须是唯一的）。

重启 agent 或使用 agent 新的在线控制选项，例如：

```
zabbix_agentd -R userparameter_reload
```

然后，在配置一个监控项时，输入指定的 key 来找到需要执行的用户参数的命令行。

用户自定义参数是由 Zabbix agent 来执行命令的。在监控项预处理步骤前，最多可以返回 512KB 的数据。但是，请注意，最终可以存储在数据库中的文本值，在 MySQL 上的限制为 64KB（其他数据库的信息请参阅数据表）。

/bin/sh 作为在 UNIX 操作系统中的命令行解释器使用。用户自定义参数参照 agent check 超时；如果超时时间到了，那么执行用户自定义参数的子进程将会被中止。

参见:

- 分布教程 配置用户自定参数
- 命令行执行

用户自定义参数用例

一个简单的命令：

```
UserParameter=ping,echo 1
```

agent 将始终为使用 “ping” 为 key 的监控项返回 “1”。

一个复杂一些的例子：

```
UserParameter=mysql.ping,mysqladmin -uroot ping | grep -c alive
```

如果 Mysql 服务器是活动状态，agent 将返回 “1”，否则会返回 “0”。

灵活的用户自定义参数

灵活的用户自定义参数可以从 key 中接受参数。这是一种使用一个用户自定义参数创建多个监控项的方式。

灵活的用户自定义参数的语法如下：

```
UserParameter=key[*],command
```

参数	描述
Key	唯一的监控项 key。[*] 用于定义该 key 接受括号内的参数。 参数需在配置监控项时给出
Command	命令在执行时，引用 key 中指定的值 只对灵活的用户参数有效: 你可以在命令中使用位置引用 $1 ... $9 来引用监控项 Key 中的相应参数。 Zabbix 解析监控项 Key 的 [] 中包含的参数，并相应地替换 $1，...，$9。 $0 会替换为完整的原始命令（在对 $0，...，$9 执行替换之前的命令）运行。 不管位置参数（$0,...,$9) 是用双引号 (") 还是单引号 (') 括起来，都会解析位置引用。 要使用位置引用解析，请指定双美元符号（$）- 例如，" awk '{print $$2}' " 。在这种情况下，执行命令时，$$2 实际上会变成 $2 。

Attention:
仅对灵活的用户自定义参数进行带有 $ 符号的搜索，并由 Zabbix agent 解析替换。对于简单的用户自定义参数，这种引用会被跳过，因此不需要任何 $ 符号引用。

Attention:
默认情况下，用户自定义参数中不允许使用某些特殊符号。有关完整列表，请参阅不安全的用户参数文档。

示例 1

先来一个简单的：

```
UserParameter=ping[*],echo $1
```

我们可以定义无数个监控项来监控所有形如 ping[something] 格式的东西。

- ping[0] - 将总是返回 ' 0 '
- ping[aaa] - 将总是返回 'aaa'

示例 2

让我们更进一步！

```
UserParameter=mysql.ping[*],mysqladmin -u$1 -p$2 ping | grep -c alive
```

这个用户自定义参数可以用来监控 MySQL 数据库的可用性状态。我们可以传入用户名和密码：

```
mysql.ping[zabbix,our_password]
```

示例 3

一个文件中有多少行匹配正则表达式？

```
UserParameter=wc[*],grep -c "$2" $1
```

这个用户自定义参数能用来计算一个文件中有多少行匹配相应的表达式。就像下面一样：

```
wc[/etc/passwd,root]
wc[/etc/services,zabbix]
```

命令执行结果

命令的返回值是标准输出和标准错误。

> **Attention:**
> 标准错误情况下，不支持文本（字符、日志或是文本类型的信息）的监控项

返回文本的用户自定义参数（字符，日志，文本信息类型）可以返回空格。如果结果不可用，那么这个监控项会变为不支持状态。

1 扩展 **Zabbix Agents**

本教程提供了如何使用用户自定义参数扩展 Zabbix agent用户自定义参数功能的详细步骤。

第一步

写一个脚本或命令行以检测所需的参数。

举个例子，我们编辑了下面的命令以获取 MySQL Server 执行的查询总数：

```
mysqladmin -uroot status | cut -f4 -d":" | cut -f1 -d"S"
```

这个命令执行后，将恢复返回 SQL 查询的总数。

第二步

添加命令到 zabbix_agentd.conf:

```
UserParameter=mysql.questions,mysqladmin -uroot status | cut -f4 -d":" | cut -f1 -d"S"
```

mysql.questions 作为 key 需要是唯一标识符。可以是任何有效的字符，比如 *queries*。

通过使用带有 '-t' 标识的 zabbix_agentd 命令测试此用户自定义参数的执行。(如果是以 root 用户运行，请注意 agent 与作为守护进程执行时的权限差异)：

```
zabbix_agentd -t mysql.questions
```

第三步

通过如下方式重新加载用户自定义参数:

```
zabbix_agentd -R userparameter_reload
```

除了执行命令行，也可以重启 agent 来实现。

使用zabbix_get 程序测试该用户自定义参数。

第四步

在被监控主机中添加使用 key 值为 'mysql.questions' 的新监控项。监控项类型必须使用 Zabbix Agent 或 Zabbix Agent（Active)。

注意在 Zabbix Server 上。必须设置正确的返回值类型，否则 Zabbix 无法处理。

6 可加载模块

1 概览

可加载模块提供了一个侧重性能的选项，来扩展 Zabbix 的功能。

目前已经有以下的功能来扩展 Zabbix 功能:

- 用户自定义变量 (Agent 指标)
- 扩展检查 (无 Agent 监控)
- `system.run[]` Zabbix Agent 监控项.

这些功能工作的十分优秀，但是存在一个重要的缺陷，名字叫 fork()。在每次处理用户指标的时候都必须创建一个新的子进程，这样不会有优秀的性能表现。通常这并不是个大问题，然而这会在监控嵌入式系统、拥有大量监控参数或运行具有逻辑繁多或启动时间长的脚本的情况下成为一个严重的问题。

可加载模块提供了在不额外消耗性能的情况下，扩展 Zabbix Agent、Server 和 Proxy。

可加载模块是基于在 Zabbix 守护进程启动时加载的共享库。这个库包含了一些功能，以便 Zabbix 可以检测到该文件确实是一个可以被加载和使用的模块。

可加载模块具有许多优点。良好的性能和容易实现逻辑的能力很重要，但最重要的能力是开发、使用和共享 Zabbi 模块。可加载模块有助于实现无故障维护，有助于更轻松的提供新功能并且不依赖于 Zabbix 核心代码库。

二进制形式的模块的授权和分发应在 GPL 许可证的许可下管理（模块运行时连接到 Zabbix 并且使用 Zabbix 的头文件；目前 ZABBIX 的代码根据 GPL 许可证进行授权）。ZABBIX 不保证二进制兼容性。

在一个 ZABBIX LTS(长期支持)版本支持周期内保证 API 模块的稳定性。ZABBIX API 的稳定性无法保证（从技术上讲，可以从模块调用 ZABBIX 内部函数，但不能保证这些模块可以工作）。

2 模块 API

为了将共享库视作 ZABBIX 模块，它应该实现并导出一些函数。目前，ZABBIX 模块 API 中由六个函数，其中一个是强制性的，另外五个是可选的。

3 强制接口

唯一的强制函数是 **zbx_module_api_version()**:

```
int zbx_module_api_version(void);
```

此函数应该返回实现这个模块的 API 版本，并且为了模块能被加载，这个版本必须与 ZABBIX 支持的模块 API 版本相匹配。Zabbix 支持的模块 API 的版本为 ZBX_MODULE_API_VERSION。所以这个函数应该返回这个常量。为了保持源代码兼容性，现在将旧常量 ZBX_MODULE_API_VERSION_ONE，替换成 ZBX_MODULE_API_VERSION，但不建议使用它。

3.1 可选接口

可选的函数是 **zbx_module_init()**, **zbx_module_item_list()**, **zbx_module_item_timeout()**, **zbx_module_history_write_cbs()** and **zbx_module_uninit()**:

```
int zbx_module_init(void);
```

这个函数应该对模块的执行进行必要的初始化（如果有的话）。如果成功，则返回 ZBX_MODULE_OK。否则它应该返回 ZBX_MODULE_FAIL。若为后一种情况，ZABBIX 将无法启动。

```
ZBX_METRIC  *zbx_module_item_list(void);
```

此函数应当返回模块支持的监控项列表。每个监控项被定义为 ZBX_METRIC 的结构下，详细信息请见后文。这个列表应以 “key” 字段为 NULL 作为 ZBX_METRIC 结构的终止。

```
void    zbx_module_item_timeout(int timeout);
```

如果模块导出 **zbx_module_item_list()**，此函数用于定义 Zabbix 配置文件中的超时设置，基于这个模块的监控项检查应遵守这个设置。这边，“timeout” 参数以秒为单位。

```
ZBX_HISTORY_WRITE_CBS   zbx_module_history_write_cbs(void);
```

这个函数应当返回 ZABBIX 服务器将用于导出不同数据类型历史记录的回调函数。回调函数应以 ZBX_HISTORY_WRITE_CBS 结构的字段提供，如果模块对于某种类型的历史纪录不感兴趣，则字段可以为 NULL 。

```
int zbx_module_uninit(void);
```

这个函数应当执行必要的反初始化（如果有的话），如释放分配的资源、关闭文件描述符等。

所有的函数会在 ZABBIX 启动的时候加载模块时，除了 zbx_module_uninit() 都将被调用一次。在卸载模块时，zbx_module_uninit() 会被 ZABBIX 调用。

3.2 定义监控项

每个监控项都应当被定义在 ZBX_METRIC 结构中：

```
typedef struct
{
    char        *key;
    unsigned    flags;
    int     (*function)();
    char        *test_param;
}
ZBX_METRIC;
```

这里的 **key** 指的时监控项的 key（例如：“ dummy.random”），**flags** 可以是 CF_HAVEPARAMS 或 0 (取决于监控项是否接受参数)，**function** 是实现该监控项的 C 函数（例如：”zbx_module_dummy_random”），最后 **test_param** 是使用” -P ” 标志启动 ZABBIX Agent 时使用的参数里列表（例如：”1,1000”, 可以是 NULL)。下面是一个具体示例：

```
static ZBX_METRIC keys[] =
{
    { "dummy.random", CF_HAVEPARAMS, zbx_module_dummy_random, "1,1000" },
    { NULL }
}
```

每个实现一个监控项的函数应该接受俩哥哥指针参数函数，第一个是一种 AGENT_REQUEST 类型，第二个是一种 AGENT_RESULT 类型：

```
int zbx_module_dummy_random(AGENT_REQUEST *request, AGENT_RESULT *result)
{
    ...

    SET_UI64_RESULT(result, from + rand() % (to - from + 1));

    return SYSINFO_RET_OK;
}
```

如果这个监控项的值被成功获取，这些函数应当返回 SYSINFO_RET_OK。否则，应当返回 SYSINFO_RET_FAIL。关于如何从 AGENT_REQUEST 获取信息以及如何设定 AGENT_RESULT 的详情，请参阅示例 “dummy” 模块。

3.3 提供历史记录输出的回调

从 ZABBIX 4.0.0 开始，不再支持通过 ZABBIX Proxy 模块输出历史记录:::

模块可以按来行指定输出历史数据的函数：数字（浮点）、数字（无符号）、字符串、文本和日志：

```
typedef struct
{
    void    (*history_float_cb)(const ZBX_HISTORY_FLOAT *history, int history_num);
    void    (*history_integer_cb)(const ZBX_HISTORY_INTEGER *history, int history_num);
    void    (*history_string_cb)(const ZBX_HISTORY_STRING *history, int history_num);
    void    (*history_text_cb)(const ZBX_HISTORY_TEXT *history, int history_num);
    void    (*history_log_cb)(const ZBX_HISTORY_LOG *history, int history_num);
}
ZBX_HISTORY_WRITE_CBS;
```

每个输出历史纪录的函数都应当把”history_num” 元素作为 “history” 数组的参数。依据需要输出的历史记录类型，“history” 分别是以下结构的数组：

```
typedef struct
{
    zbx_uint64_t    itemid;
    int     clock;
    int     ns;
```

```
    double      value;
}
ZBX_HISTORY_FLOAT;

typedef struct
{
    zbx_uint64_t    itemid;
    int     clock;
    int     ns;
    zbx_uint64_t    value;
}
ZBX_HISTORY_INTEGER;

typedef struct
{
    zbx_uint64_t    itemid;
    int     clock;
    int     ns;
    const char  *value;
}
ZBX_HISTORY_STRING;

typedef struct
{
    zbx_uint64_t    itemid;
    int     clock;
    int     ns;
    const char  *value;
}
ZBX_HISTORY_TEXT;

typedef struct
{
    zbx_uint64_t    itemid;
    int     clock;
    int     ns;
    const char  *value;
    const char  *source;
    int     timestamp;
    int     logeventid;
    int     severity;
}
ZBX_HISTORY_LOG;
```

回调会在 ZABBIX server 的历史记录同步进程完成历史记录同步操作，数据被写入 ZABBIX 数据库并将值保存在值缓存中后执行。

3.4 构建模块

目前，模块应当再 ZABBIX 源代码树中构建，因为模块 API 依赖于一些 ZABBIX 头文件中定义的一些数据结构。

对可加载模块来说，最重要的头是 **include/module.h**，它定义了这些住居结构。另一个很有用的头文件 **include/sysinc.h** ，它的执行会包含必要的系统头文件，这有助于 include/module.h 的正常工作。

为了 include/module.h 和 include/sysinc.h 被导入，应在 ZABBIX 源代码树的根目录下执行**./configure** 命令。这将创建 **include/config.h** 文件，其中包含了 include/sysinc.h 依赖。(如果你获得的 ZABBIX 源代码来自子版本存储库，则./configure 脚本尚不存在，应首先运行 **./bootstrap.sh** 脚本来生成它。)

记住这些信息，一切都准备好了去构建模块。该模块应包含 **sysinc.h** 和 **module.h**，构建脚本应确保这两个文件包含于路径中。有关详细信息，参见下文 “dummy” 模块。

其它有用的头文件 **include/log.h**，它定义了 **zabbix_log()** 函数，可用于记录和调试目的。

4 配置参数

ZABBIX Agent, Server 和 Proxy 支持两个参数来处理模块：

- LoadModulePath – 可加载模块所在的完整路径
- LoadModule – 启动时加载的模块。这些模块必须位于 LoadModulePath 制定的目录中。允许包含多个 LoadModule 参数

例如：要扩展 ZABBIX Agent 我们可以添加以下参数：

```
LoadModulePath=/usr/local/lib/zabbix/agent/
LoadModule=mariadb.so
LoadModule=apache.so
LoadModule=kernel.so
LoadModule=dummy.so
```

在启动 Agent 时，它将从/usr/local/lib/zabbix/agent/目录加载 mariadb.so,apache.so, kernel.so 和/usr/local/lib/zabbix 目录下的 dummy.so 模块。如果发生缺少模块、权限错误或该共享库文件不是 ZABBIX 模块，那么 Agent 的启动将失败。

5 前端配置

ZABBIX Agent、Server 和 Proxy 支持可加载模块。因此 ZABBIX 前端中的监控项类型依据模块在哪里被加载。如果模块在 Agent 端被加载那么监控项类型应当设置为 “Agent 检查” 或 “Agent 检查 (主动) ”。如果在 Server 端或 Proxy 端被加载，那么响应的类型应当为 “简单检查”。

通过 ZABBIX 模块历史记录输出不需要进行前端配置。如果模块成功加载并提供 **zbx_module_history_write_cbs()** 函数且该函数应至少返回一个非 NULL 回调方法，则将自动启动历史记录输出。

6 Dummy 模块

ZABBIX 包含一个用 C 语言编写的示例模块。该模块位于” src/modules/dummy”：

```
alex@alex:~trunk/src/modules/dummy$ ls -l
-rw-rw-r-- 1 alex alex 9019 Apr 24 17:54 dummy.c
-rw-rw-r-- 1 alex alex   67 Apr 24 17:54 Makefile
-rw-rw-r-- 1 alex alex  245 Apr 24 17:54 README
```

这个模块由详细的文档，可以作为您编写自己的模块的模板。

如上所述，在 ZABBIX 源代码根目录下运行./configure 命令后，至于要运行 **make** 即可构建 **dummy.so**.

```
/*
** Zabbix
** Copyright (C) 2001-2016 Zabbix SIA
**
** This program is free software; you can redistribute it and/or modify
** it under the terms of the GNU General Public License as published by
** the Free Software Foundation; either version 2 of the License, or
** (at your option) any later version.
**
** This program is distributed in the hope that it will be useful,
** but WITHOUT ANY WARRANTY; without even the implied warranty of
** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
** GNU General Public License for more details.
**
** You should have received a copy of the GNU General Public License
** along with this program; if not, write to the Free Software
** Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.
**/

####include "sysinc.h"
####include "module.h"

/* the variable keeps timeout setting for item processing */
static int  item_timeout = 0;

/* module SHOULD define internal functions as static and use a naming pattern different from Zabbix intern
/* symbols (zbx_*) and loadable module API functions (zbx_module_*) to avoid conflicts
static int  dummy_ping(AGENT_REQUEST *request, AGENT_RESULT *result);
```

```c
static int  dummy_echo(AGENT_REQUEST *request, AGENT_RESULT *result);
static int  dummy_random(AGENT_REQUEST *request, AGENT_RESULT *result);

static ZBX_METRIC keys[] =
/*  KEY         FLAG        FUNCTION    TEST PARAMETERS */
{
    {"dummy.ping",      0,      dummy_ping, NULL},
    {"dummy.echo",      CF_HAVEPARAMS,  dummy_echo, "a message"},
    {"dummy.random",    CF_HAVEPARAMS,  dummy_random,   "1,1000"},
    {NULL}
};

/******************************************************************************
 *                                                                            *
 * Function: zbx_module_api_version                                           *
 *                                                                            *
 * Purpose: returns version number of the module interface                    *
 *                                                                            *
 * Return value: ZBX_MODULE_API_VERSION - version of module.h module is       *
 *               compiled with, in order to load module successfully Zabbix   *
 *               MUST be compiled with the same version of this header file   *
 *                                                                            *
 ******************************************************************************/
int zbx_module_api_version(void)
{
    return ZBX_MODULE_API_VERSION;
}

/******************************************************************************
 *                                                                            *
 * Function: zbx_module_item_timeout                                          *
 *                                                                            *
 * Purpose: set timeout value for processing of items                         *
 *                                                                            *
 * Parameters: timeout - timeout in seconds, 0 - no timeout set               *
 *                                                                            *
 ******************************************************************************/
void    zbx_module_item_timeout(int timeout)
{
    item_timeout = timeout;
}

/******************************************************************************
 *                                                                            *
 * Function: zbx_module_item_list                                             *
 *                                                                            *
 * Purpose: returns list of item keys supported by the module                 *
 *                                                                            *
 * Return value: list of item keys                                            *
 *                                                                            *
 ******************************************************************************/
ZBX_METRIC  *zbx_module_item_list(void)
{
    return keys;
}

static int  dummy_ping(AGENT_REQUEST *request, AGENT_RESULT *result)
{
    SET_UI64_RESULT(result, 1);

    return SYSINFO_RET_OK;
}
```

```c
static int  dummy_echo(AGENT_REQUEST *request, AGENT_RESULT *result)
{
    char    *param;

    if (1 != request->nparam)
    {
        /* set optional error message */
        SET_MSG_RESULT(result, strdup("Invalid number of parameters."));
        return SYSINFO_RET_FAIL;
    }

    param = get_rparam(request, 0);

    SET_STR_RESULT(result, strdup(param));

    return SYSINFO_RET_OK;
}

/******************************************************************************
 *                                                                            *
 * Function: dummy_random                                                     *
 *                                                                            *
 * Purpose: a main entry point for processing of an item                      *
 *                                                                            *
 * Parameters: request - structure that contains item key and parameters      *
 *              request->key - item key without parameters                    *
 *              request->nparam - number of parameters                        *
 *              request->timeout - processing should not take longer than     *
 *                                 this number of seconds                     *
 *              request->params[N-1] - pointers to item key parameters        *
 *                                                                            *
 *             result - structure that will contain result                    *
 *                                                                            *
 * Return value: SYSINFO_RET_FAIL - function failed, item will be marked      *
 *                                  as not supported by zabbix                *
 *               SYSINFO_RET_OK - success                                     *
 *                                                                            *
 * Comment: get_rparam(request, N-1) can be used to get a pointer to the Nth  *
 *          parameter starting from 0 (first parameter). Make sure it exists  *
 *          by checking value of request->nparam.                             *
 *                                                                            *
 ******************************************************************************/
static int  dummy_random(AGENT_REQUEST *request, AGENT_RESULT *result)
{
    char    *param1, *param2;
    int from, to;

    if (2 != request->nparam)
    {
        /* set optional error message */
        SET_MSG_RESULT(result, strdup("Invalid number of parameters."));
        return SYSINFO_RET_FAIL;
    }

    param1 = get_rparam(request, 0);
    param2 = get_rparam(request, 1);

    /* there is no strict validation of parameters for simplicity sake */
    from = atoi(param1);
    to = atoi(param2);

    if (from > to)
```

```
    {
        SET_MSG_RESULT(result, strdup("Invalid range specified."));
        return SYSINFO_RET_FAIL;
    }

    SET_UI64_RESULT(result, from + rand() % (to - from + 1));

    return SYSINFO_RET_OK;
}

/******************************************************************************
 *                                                                            *
 * Function: zbx_module_init                                                  *
 *                                                                            *
 * Purpose: the function is called on agent startup                           *
 *          It should be used to call any initialization routines             *
 *                                                                            *
 * Return value: ZBX_MODULE_OK - success                                      *
 *               ZBX_MODULE_FAIL - module initialization failed               *
 *                                                                            *
 * Comment: the module won't be loaded in case of ZBX_MODULE_FAIL             *
 *                                                                            *
 ******************************************************************************/
int zbx_module_init(void)
{
    /* initialization for dummy.random */
    srand(time(NULL));

    return ZBX_MODULE_OK;
}

/******************************************************************************
 *                                                                            *
 * Function: zbx_module_uninit                                                *
 *                                                                            *
 * Purpose: the function is called on agent shutdown                          *
 *          It should be used to cleanup used resources if there are any      *
 *                                                                            *
 * Return value: ZBX_MODULE_OK - success                                      *
 *               ZBX_MODULE_FAIL - function failed                            *
 *                                                                            *
 ******************************************************************************/
int zbx_module_uninit(void)
{
    return ZBX_MODULE_OK;
}

/******************************************************************************
 *                                                                            *
 * Functions: dummy_history_float_cb                                          *
 *            dummy_history_integer_cb                                        *
 *            dummy_history_string_cb                                         *
 *            dummy_history_text_cb                                           *
 *            dummy_history_log_cb                                            *
 *                                                                            *
 * Purpose: callback functions for storing historical data of types float,    *
 *          integer, string, text and log respectively in external storage    *
 *                                                                            *
 * Parameters: history     - array of historical data                         *
 *             history_num - number of elements in history array              *
 *                                                                            *
 ******************************************************************************/
```

```c
static void dummy_history_float_cb(const ZBX_HISTORY_FLOAT *history, int history_num)
{
    int i;

    for (i = 0; i < history_num; i++)
    {
        /* do something with history[i].itemid, history[i].clock, history[i].ns, history[i].value, ... */
    }
}

static void dummy_history_integer_cb(const ZBX_HISTORY_INTEGER *history, int history_num)
{
    int i;

    for (i = 0; i < history_num; i++)
    {
        /* do something with history[i].itemid, history[i].clock, history[i].ns, history[i].value, ... */
    }
}

static void dummy_history_string_cb(const ZBX_HISTORY_STRING *history, int history_num)
{
    int i;

    for (i = 0; i < history_num; i++)
    {
        /* do something with history[i].itemid, history[i].clock, history[i].ns, history[i].value, ... */
    }
}

static void dummy_history_text_cb(const ZBX_HISTORY_TEXT *history, int history_num)
{
    int i;

    for (i = 0; i < history_num; i++)
    {
        /* do something with history[i].itemid, history[i].clock, history[i].ns, history[i].value, ... */
    }
}

static void dummy_history_log_cb(const ZBX_HISTORY_LOG *history, int history_num)
{
    int i;

    for (i = 0; i < history_num; i++)
    {
        /* do something with history[i].itemid, history[i].clock, history[i].ns, history[i].value, ... */
    }
}

/******************************************************************************
 *                                                                            *
 * Function: zbx_module_history_write_cbs                                     *
 *                                                                            *
 * Purpose: returns a set of module functions Zabbix will call to export      *
 *          different types of historical data                                *
 *                                                                            *
 * Return value: structure with callback function pointers (can be NULL if    *
 *               module is not interested in data of certain types)           *
 *                                                                            *
 ******************************************************************************/
ZBX_HISTORY_WRITE_CBS   zbx_module_history_write_cbs(void)
```

```
{
    static ZBX_HISTORY_WRITE_CBS    dummy_callbacks =
    {
        dummy_history_float_cb,
        dummy_history_integer_cb,
        dummy_history_string_cb,
        dummy_history_text_cb,
        dummy_history_log_cb,
    };

    return dummy_callbacks;
}
```

这个模块导出三个新的监控项类型：

- dummy.ping - 总是返回'1'
- dummy.echo[param1] - 总是返回第一个参数，例如 dummy.echo[ABC] 将返回" ABC "
- dummy.random[param1, param2] - 返回 param1 与 param2 范围内的随机数，例如, dummy.random[1,1000000]

7 限制

仅对类 Unix 平台实现了可加载模块的支持。这意味着它不适用于 Windows 平台的 Agent。

某些情况下，模块可能要从 *zabbix_agentd.conf* 读取与模块相关的配置参数。目前不支持这么操作。如果您需要模块使用某些配置参数，则应该实现特定与模块的配置文件解析。

7 Windows 性能计数器

概览

你可以使用 *perf_counter[]* 这个 key 有效的监控 Windows 性能计数器。

例如：

```
perf_counter["\Processor(0)\Interrupts/sec"]
```

或

```
perf_counter["\Processor(0)\Interrupts/sec", 10]
```

有关使用此 key 的更多信息请参阅Windows 专用监控项。

为了获取可用于监控的新能计数器完整列表，你可以运行：

```
typeperf -qx
```

可以通过低级别发现来发现 windows 性能计数器的多对象实例, 并自动创建多实例对象的性能计数器监控项。

数字表示

由于性能计数器的命名在不同的 Windows 服务器上可能不同，这取决于服务器的地区设置。因此，在创建用于监控具有不同地区设置的多台 Windows 设备的模板时，会引发一定的问题。

同时，每个新能计数器也可以通过其数字形式来引用，无论如何，数字形式都是唯一的，因此你可以使用数字表示而不是字符串。

为了找到同义的数字，需要运行 **regedit**，然后找到 *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib\009* 这个注册表。

注册表中包含形如下面所示的信息：

```
1
1847
2
System
4
Memory
6
```

· % Processor Time
· 10
· File Read Operations/sec
· 12
· File Write Operations/sec
· 14
· File Control Operations/sec
· 16
· File Read Bytes/sec
· 18
· File Write Bytes/sec
· ....

这样你就可以找到性能计数器每个字符串对应的数字，例如：

· System → 2
· % Processor Time → 6

然后你就可以使用这些数字来表示性能计数器路径：

· \2\6

性能计数器参数

你可以部署一些 PerfCounter 参数，来完成通过 Windows 性能计数器监控。

例如，你可以将下面的内容添加到 ZABBIX 代理配置文件中：

```
PerfCounter=UserPerfCounter1,"\Memory\Page Reads/sec",30
或
PerfCounter=UserPerfCounter2,"\4\24",30
```

配置了这些参数后，你就可以简单的使用 *UserPerfCounter1* 或 *UserPerfCounter2* 作为 key 来创建相应的监控项。

当然，别忘了在更改配置文件后重新启动 ZABBIX Agent。

8 批量更新

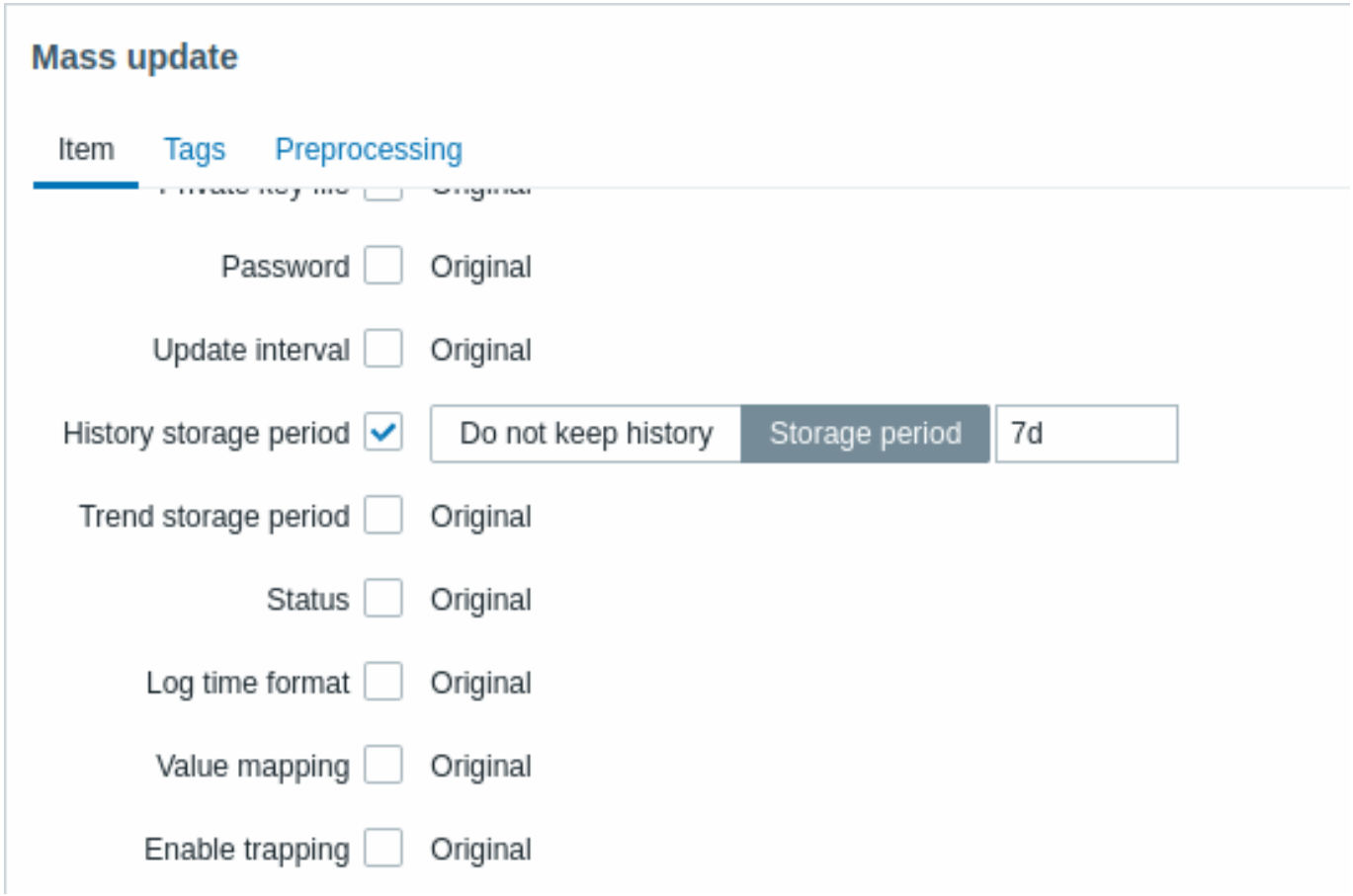
概览

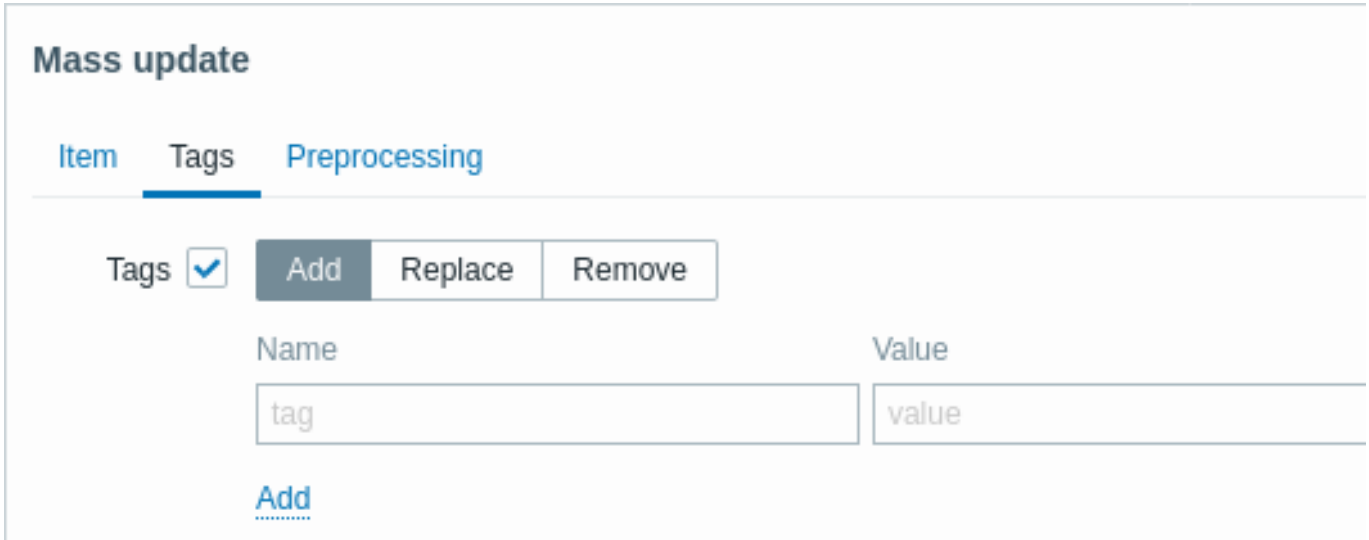
有时你可能想要一次更改多个监控项的某些属性。你可以使用批量更新功能，而不是打开每个独立的监控项进行编辑。

使用批量更新

要批量更新某些监控项，请按如下步骤操作：

- 在监控项列表，标记想要更新的监控项的复选框
- 点击列表下方的 批量更新按钮
- 导航到所需属性的选项卡 (监控项, 标签或者 预处理)
- 标记想要更新的属性的复选框
- 键入该属性的新值

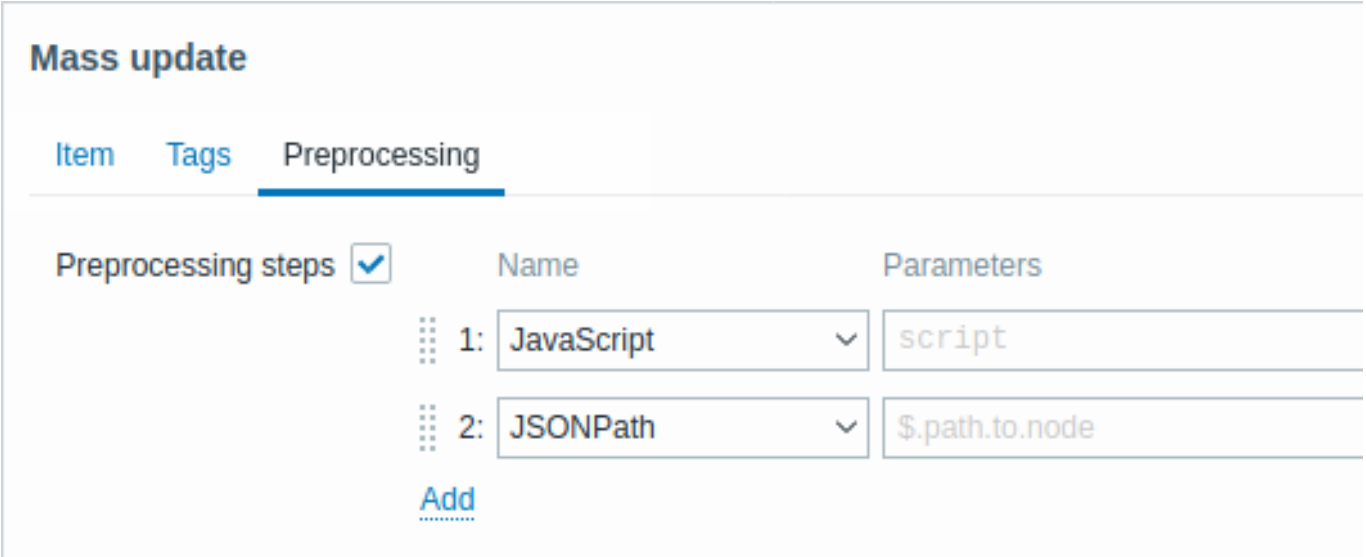




标签选项允许:

- 增加 - 给监控项增加指定的标签 (带有相同名字但是不同值的标签不会被认为是' 重复' ，且可以添加给同一主机).
- 替换 - 删除指定的标签，然后增加一个新的标签
- 删除 - 从监控项中删除一个指定的标签

用户宏，标签支持 {INVENTORY.*} 宏，{HOST.HOST}，{HOST.NAME},{HOST.CONN}，{HOST.DNS}，{HOST.IP}，{HOST.PORT} 和 {HOST.ID} 等用户宏.



当修改完成后，点击 更新按钮。

9 值映射

概览

为了接收到的值能更 “人性化” 的表示，你可以使用包含表示数值/字符值和字符串之间映射的值映射。

值映射也可以用在 ZABBIX 的前端和告警通知媒介中。

举个例子，一个监控项有值 ‘0’ 和 ‘1’ 能通过值映射，以认可读的形式表示值：

- ’0’ => ’ 不可用’
- ’1’ => ’ 可用’

或者, 一组备份关系的值映射可以是:

- ’F’ → ’ 全量备份’
- ’D’ → ’ 差异备份’
- ’I’ → ’ 增量备份’

另一个例子, 电压值的范围可以映射为:

- ’<=209’ => ’ 低’
- ’210-230’ => ’ 正常’
- ’>=231’ => ’ 高’

值映射可以在模板级或主机级进行定义。一旦定义好后，就对相应的模板或主机的监控项都可用。值映射没有继承属性 - 主机上的监控项模板仍然使用模板里定义的值映射；将具有值映射的模板链接到主机上，并不会使主机继承值映射。

在配置监控项 时你可以使用值映射来以” 人性化” 的方式展现监控项的值，通过索引事先在值映射字段定义好的值映射即可实现。

> **Note:**
> 值映射能被用来替换 数字（无符号），数字（浮点）和 字符类型的监控项信息。

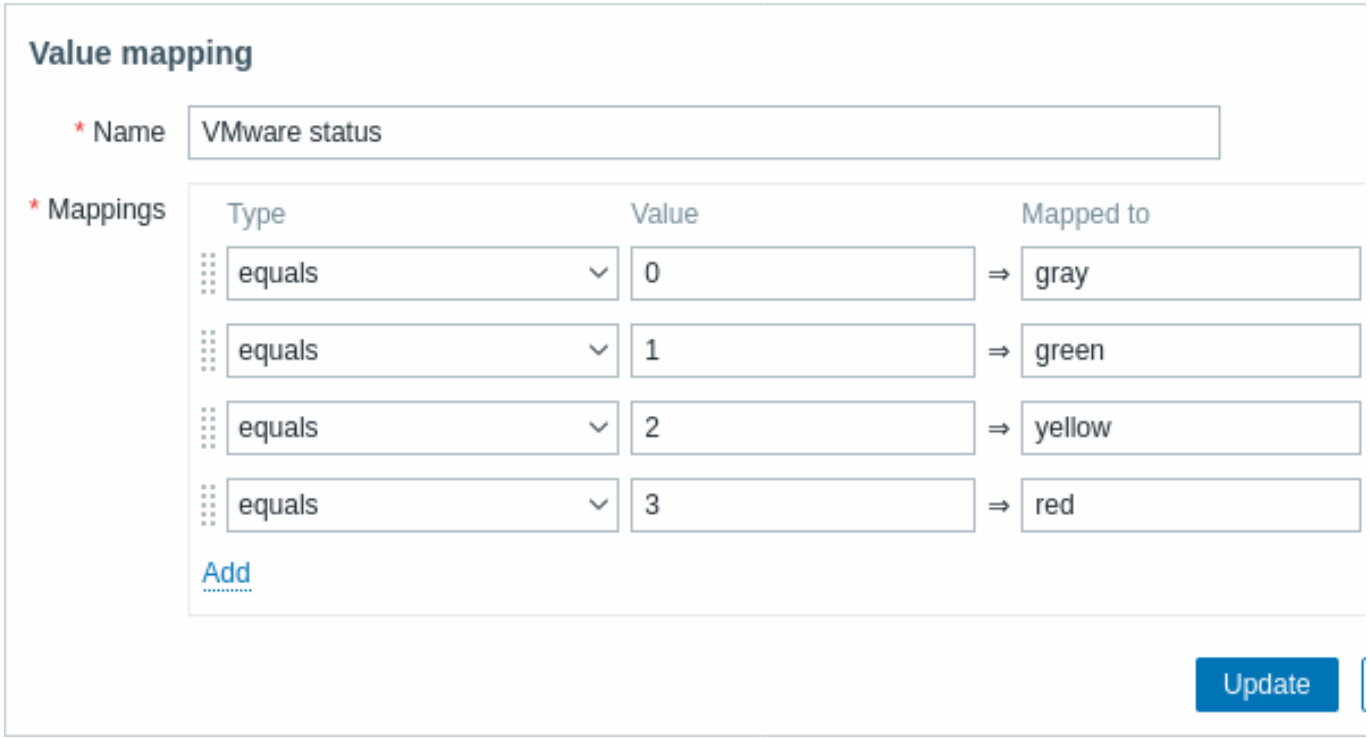
值映射可以根据对应的模板或主机导出/导入。

值映射可以批量更新。主机 和模板 的批量更新表单都有一个值映射的标签用来批量更新值映射。

配置

要定义一个值映射:

- 打开主机或者模板配置表单
- 前往 值映射标签
- 点击 增加来增加一个新映射
- 点击一个已存在的值映射名字来进行编辑

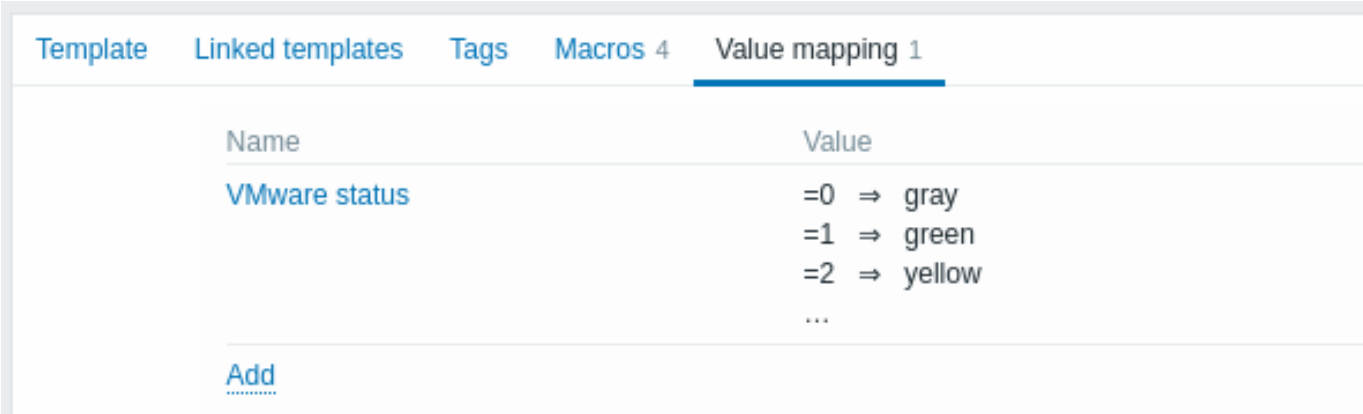


值映射参数:

参数		描述
名称		值映射集的唯一名称。
映射		数值/字符串值映射到字符串的映射规则。 值映射是根据映射规则的顺序来应用的。可以通过拖拽来重新排序映射。 只有数值型的映射类型支持诸如 (大于等于, 小于等于, 在范围内) 的映射范围。
	类型	映射类型: 等于 - 映射相等的值 大于等于 - 映射大于等于的值 小于等于 - 映射小于等于的值 在范围内 - 映射范围内的值; 范围通过 <number1>-<number2>, 或 <number> 的方位说明。也支持多个范围 (例如 1-10,101-110,201) 正则 - 映射正则表达式 相关的值 (不支持全局正则表达式) 缺省 - 所有未完成的值都将被映射，除了那些具有特定映射的值
	值	传入值。 取决于映射类型, 可能包含一个范围或一个正则表达式。
	映射为	传入值被映射的字符串。

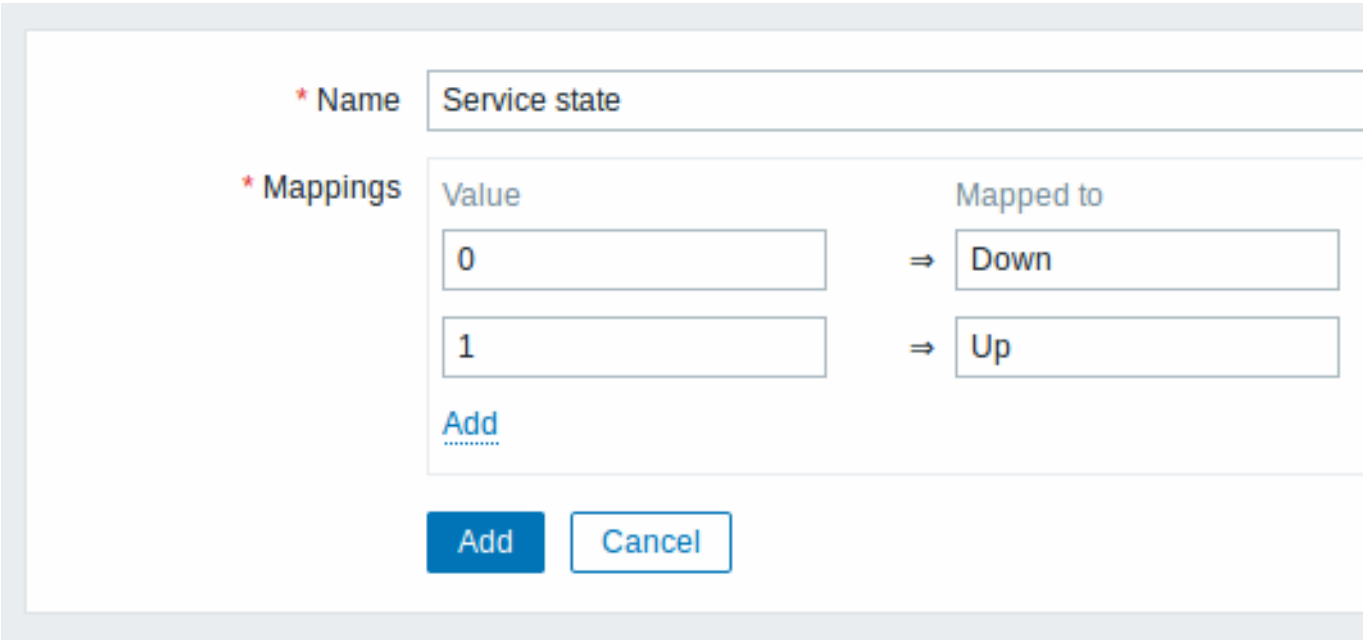
所有强制输入字段都用红色星号标记。

当值映射显示在列表中时，只显示前 3 个值它的映射是可见的，而三个点表示更多的映射存在。



值映射如何工作的

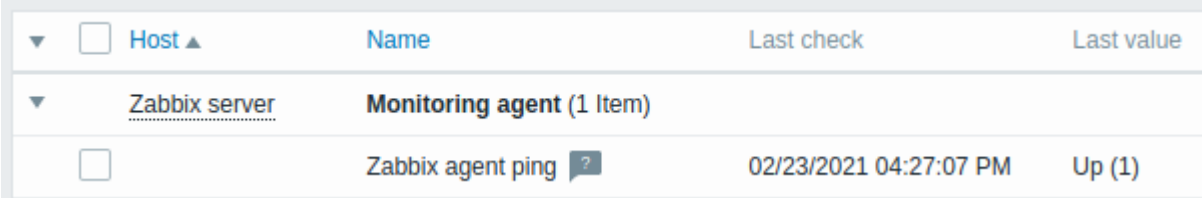
举个例子，有一个预定义的 Agent 监控项'Ping to the server ' 使用了一个已经存在的模板级值映射名字叫'Service state' ，来显示其值。



在监控项的配置页面，你可以从显示值字段看到对此值映射的引用。



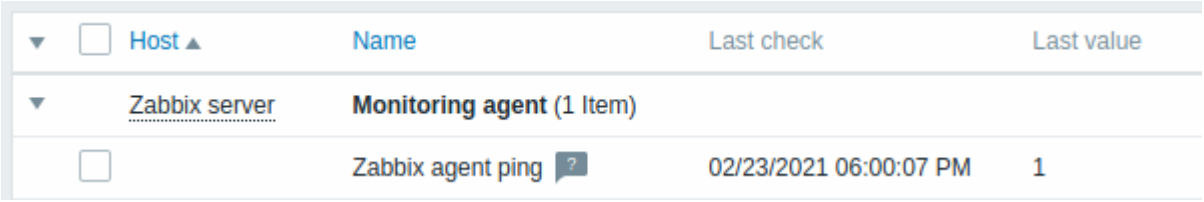
这样配置以后，在监控中 → 最新数据会以映射的值 "Up" 显示（括号中显示的时原始值）。



在最新数据部分中，显示的值缩短为 20 个符号，如果使用值映射，则此缩短规则不会应用于映射值，而是仅应用于原始值（显示在括号中）。

Note:
在接收通知时，以人类可读的形式显示的值也更容易理解。

如果没有预定义的值映射，你只能看到：



这样的情况下，要么猜测 “1” 是什么意思，要么去搜索文档以找到答案。

10 队列

概览

队列显示正在等待刷新的监控项。队列只是数据的一种 逻辑上表现。ZABBIX 中并没有 IPC 队列或者其它任何队列的机制。

由 Proxy 们监控的监控项也会被包含在队列中 -这些监控项将按 Proxy 历史数据更新周期被计数为队列。

只有具有刷新时间计划的监控项才会记录在队列中。也就是说队列中将不包含以下的监控项类型：

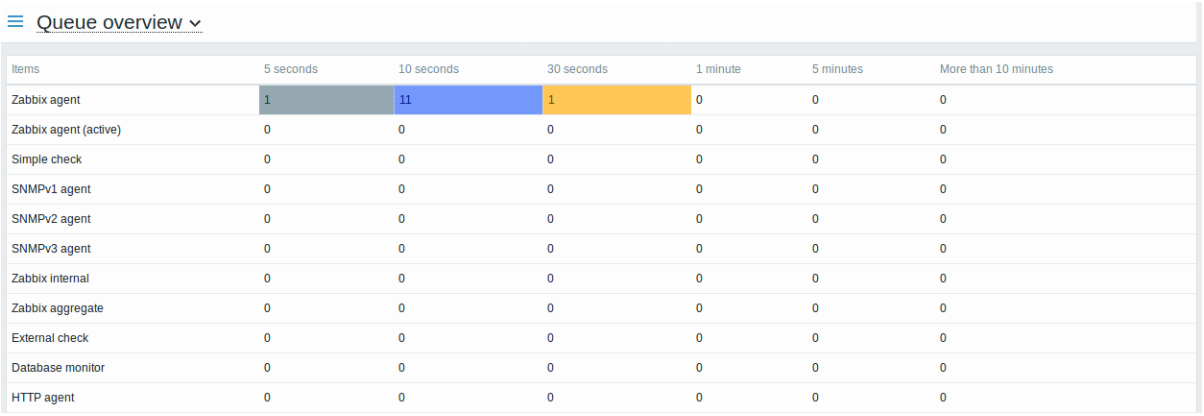
- log, logrt 和 event log (Active Agent)
- SNMP trap 监控项
- trapper 监控项
- web 监控监控项
- 有依赖的监控项

队列显示的统计信息是 ZABBIX Server 是否健康的指标。

使用 JSON 协议直接从 ZABBIX Server 检索队列。这个页面的信息只在 ZABBIX Server 运行时可用。

查看队列

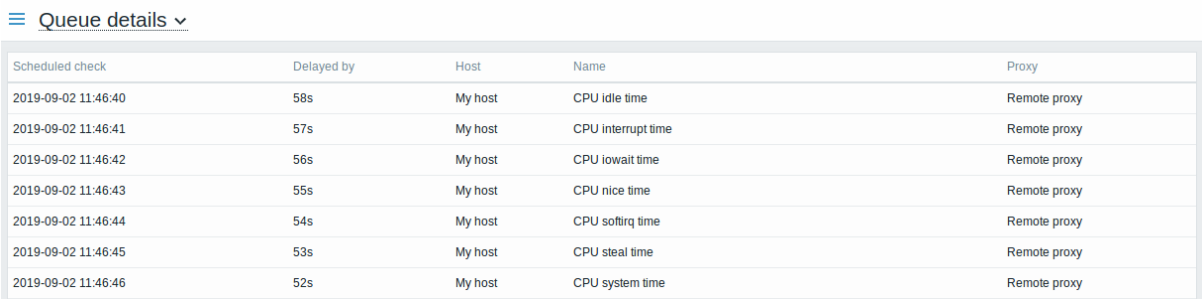
要查看队列，请跳转管理 → 队列。在右侧的下达菜单中选择概览。



这个图片通常是”ok“的话我们可以认为服务器运行时正常的。

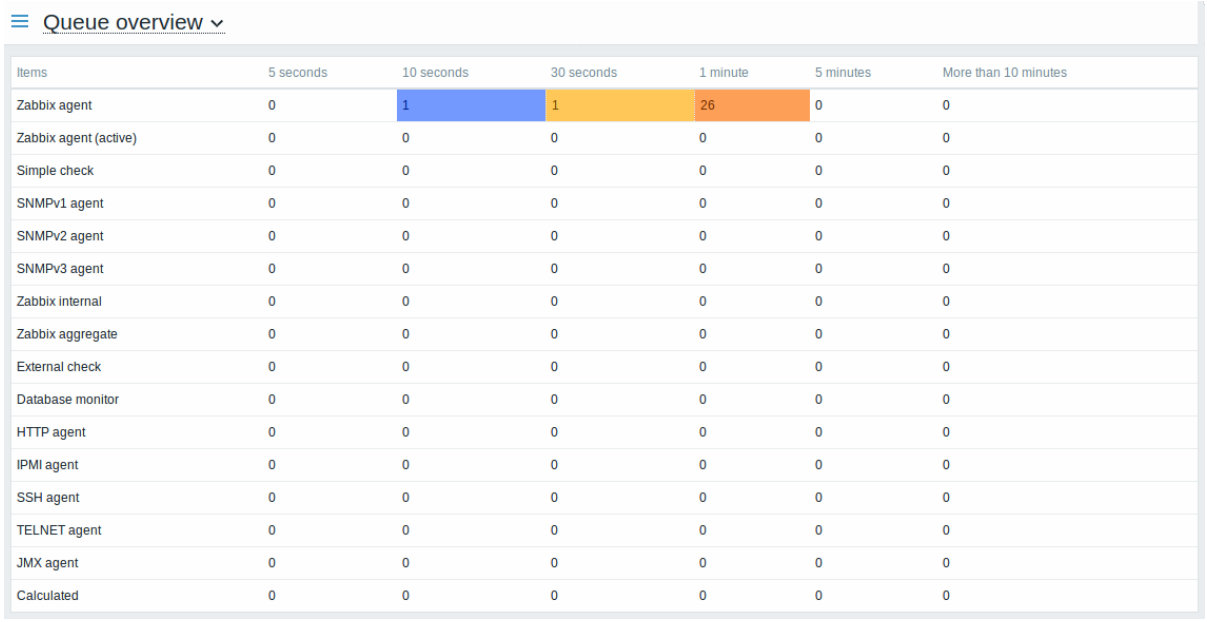
队列里显示有一些监控项已经等待了有 30 秒。需要对这些都是哪些监控项引起重视。

可以在下拉菜单中选择队列细节来实现，然后你就可以看到这些延迟监控项的列表。



通过这些细节信息，可以找出这些监控项发生延迟的原因。

有一两个延迟监控项，不要慌张。它们有可能在一秒内被更新。但是如果你看到了一大堆延迟很久的监控项，这可能导致严重的问题。


Queue overview

Items	5 seconds	10 seconds	30 seconds	1 minute	5 minutes	More than 10 minutes
Zabbix agent	0	1	1	26	0	0
Zabbix agent (active)	0	0	0	0	0	0
Simple check	0	0	0	0	0	0
SNMPv1 agent	0	0	0	0	0	0
SNMPv2 agent	0	0	0	0	0	0
SNMPv3 agent	0	0	0	0	0	0
Zabbix internal	0	0	0	0	0	0
Zabbix aggregate	0	0	0	0	0	0
External check	0	0	0	0	0	0
Database monitor	0	0	0	0	0	0
HTTP agent	0	0	0	0	0	0
IPMI agent	0	0	0	0	0	0
SSH agent	0	0	0	0	0	0
TELNET agent	0	0	0	0	0	0
JMX agent	0	0	0	0	0	0
Calculated	0	0	0	0	0	0

队列监控项

有一个特别的内部监控项 **zabbix[queue,<from>,<to>]** 可以用于监控 ZABBIX 中队列的健康状态。他会返回指定时间区间的监控项数目。有关更多信息请参阅内部监控项。

11 值缓存

概览

为了更快地计算触发器表达式、计算或聚合类型监控项和一些宏。ZABBIX Server 支持值缓存选项。

这个内存缓存，可以用于访问历史数据，而不需要对数据库直接执行 SQL 调用。如果请求的历史值不在缓存中，则会从数据库请求缺失的数据，并相应地更新缓存。

要启用值缓存功能，Zabbix 服务器配置支持可选的 **ValueCacheSize** 参数。

有两个内部的监控项来监控值缓存：**zabbix[vcache，buffer，<mode>]** 和 **zabbix[vcache，cache，<parameter>]**。查看更多细节，请参阅内部监控项。

12 立刻检查

概览

在 ZABBIX 中，检查一个新监控项的值是基于配置更新间隔的循环过程。虽然对于大多数监控项来说间隔非常短，但是有些监控项 (包括低级别的发现规则) 更新间隔会很长。因此在实际情况下，可能需要很快的检查新的值。-例如，发现资源的变化。为了满足这种需求，可以改成被动检查并立刻检索新的值。

这个功能仅支持被动检查。支持以下监控项的类型：

- Zabbix agent (被动)
- SNMPv1/v2/v3 agent
- IPMI agent
- 简单检查
- Zabbix 内部
- 外部检查
- 数据库监控
- JMX agent

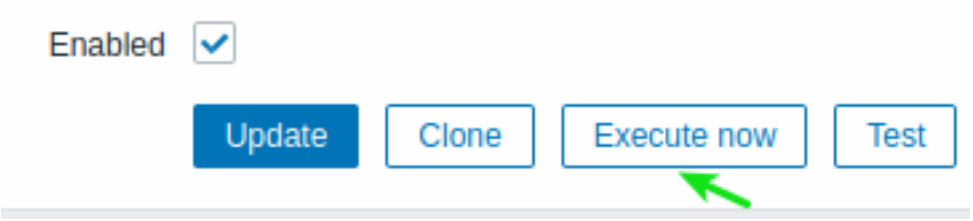
- SSH agent
- Telnet
- 计算
- HTTP agent
- 脚本

Attention:
检查必须存在于配置缓存中才能执行。有关详细信息请参阅缓存更新频率。
在执行检查前，若配置缓存没有更新，那么将不会检索最近更改配置的监控项/自动发现规则。同样也无法检查刚刚创建的监控项/规则的最新值，可以在配置监控项时通过测试选项来验证下。

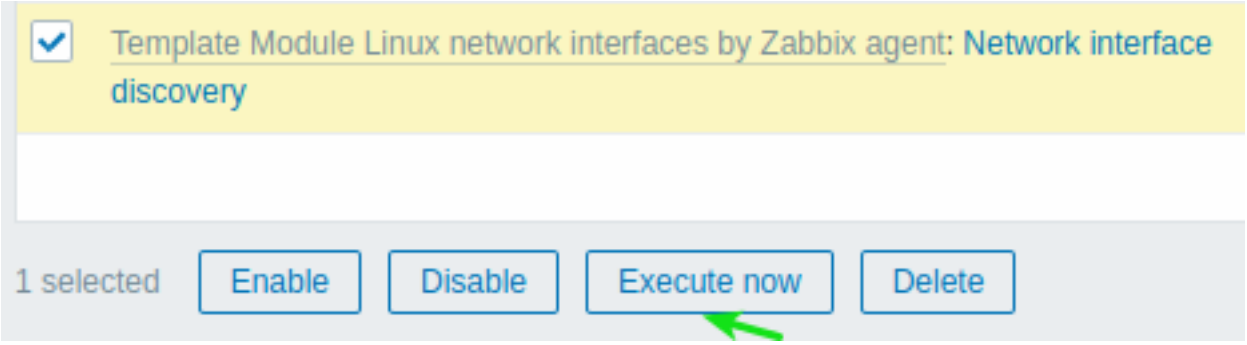
配置

要立刻执行被动检查：

- 在已存在的监控项（或自动发现规则）配置表单中点击立即检查:



- 在监控项（或发现规则列表）中，选定监控项/规则后，单击立即检查:



在第二种情况下，可以选择几个项目/规则，一次性对它们进行“立刻检查”。

13 限制代理检查

概览

可以通过创建监控项黑名单、白名单或白名单/黑名单的组合来限制 agent 的检查。

为此，请结合使用两个 agent配置 参数:

- `AllowKey=<pattern>` - 允许哪些检查；<pattern> 使用通配符 (*) 表达式指定
- `DenyKey=<pattern>` - 拒绝哪些检查；<pattern> 使用通配符 (*) 表达式指定

请注意:

- 缺省情况下，所有 `system.run[*]` 监控项 (远程命令，脚本) 是禁用的，即便没有指定拒绝健；
- 从 Zabbix 5.0.2 开始，代理参数 EnableRemoteCommands :

* `Zabbix agent`已 弃 用
* `Zabbix agent2`不 再 支 持

因此，要允许所有远程命令，请指定 AllowKey=system.run[*] 参数。要允许部分远程命令，创建 system.run[] 命令指定的白名单。要禁用指定的远程命令行，在 AllowKey=system.run[*] 参数前使用 system.run[] 命令增加 DenyKey 参数。

重要规则

- 没有拒绝规则的白名单只允许 system.run[*] 监控项。对于所有其它监控项，如果没有 DenyKey 参数，则不允许使用 AllowKey 参数；这种只有 AllowKey 参数情况下，Zabbix agent 将不会启动。
- 顺序很重要。指定参数在配置文件中按其出现顺序逐一检查:

  - 一旦监控项键值与允许/拒绝规则匹配，该项即被允许或拒绝；并且规则检查停止。因此，如果一个项同时满足允许规则和拒绝规则，那么结果取决于最先匹配到的那个规则。
  - 顺序还影响 EnableRemoteCommands 参数（如果使用）。
- 支持无限数量的 AllowKey/DenyKey 参数。
- - AllowKey、DenyKey 规则不影响 HostnameItem、HostMetadataItem、HostInterfaceItem 配置参数。
- - 键模式是一个通配符表达式，其中通配符 (*) 与特定位置的任意数量的任意字符匹配。它可以用于关键字名称和参数。
- 如果在 agent 配置中不允许某个特定的监控项，则该监控项将被报告为不受支持（没有给出有关原因的提示）；
- Zabbix agent 命令行使用带--print（-p）选项将不显示配置为不允许的键值;
- Zabbix agent 命令行使用有--test（-t）选项将会显示配置不允许的键值为”Unsupported item key” 状态;
- 拒绝的远程命令不会记录在 agent 日志中（如果 LogRemoteCommands=1）。

用例

拒绝特定检查

- 使用 DenyKey 参数将特定检查列入黑名单。匹配到的键值将被禁止。所有未匹配到的键值都将被允许，除了 system.run[] 监控项。

例如：

```
# 拒绝安全数据访问
DenyKey=vfs.file.contents[/etc/passwd,*]
```

Attention:
黑名单可能不是一个好的选择，因为新的 Zabbix 版本中，可能存在没有在当前配置中明确限制的键值。这可能会导致安全缺陷。

拒绝特定命令，允许其他命令

- 使用 DenyKey 参数将特定命令列入黑名单。使用 AllowKey 参数将其他命令全部列入白名单。

```
# 禁止特定命令
DenyKey=system.run[ls -l /]

# 允许其他脚本
AllowKey=system.run[*]
```

允许特定检查，拒绝其他检查

- 使用 AllowKey 参数将特定检查列入白名单，使用 DenyKey=* 拒绝其他检查 DenyKey=*

例如：

```
# 允许阅读日志：
AllowKey=vfs.file.*[/var/log/*]

# 允许本地时间检查
AllowKey=system.localtime[*]

# 拒绝所有其他键
DenyKey=*
```

模式示例

模式	描述	匹配	不匹配
*	匹配所有可能的带或不带参数的键。	任何	无
vfs.file.contents	匹配不带参数的 `vfs.file.contents`。	vfs.file.contents	vfs.file.contents[/etc/passwd]
vfs.file.contents[]	匹配带有空参数的 `vfs.file.contents`。	vfs.file.contents[]	vfs.file.contents
vfs.file.contents[*]	匹配 `vfs.file.contents` 和任何参数；不匹配没有方括号的 `vfs.file.contents`。	vfs.file.contents[] vfs.file.contents[/path/to/file]	vfs.file.contents
vfs.file.contents[/etc/passwd]	匹配 `vfs.file.contents` 与第一个参数匹配 /etc/passwd 和所有其他参数具有任何值（也可以为空）。	vfs .file.contents[/etc/passwd,] vfs.file.contents[/etc/passwd,utf8]	vfs.file.contents[/etc/passwd] vfs. file.contents[/var/log/zabbix_server.log] vfs.file.contents[]

模式	描述	匹配	不匹配
*vfs.file.contents[*passwd*]*	匹配 `vfs.file.contents`，第一个参数匹配 *passwd* 而没有其他参数。	vfs.file.contents[/etc/passwd] vfs.file.contents[/etc/passwd,utf8]	vfs.file.contents[/etc/passwd,]
*vfs.file.contents[*passwd*,*]*	匹配 `vfs.file.contents`，只有第一个参数匹配 *passwd* 和所有后续参数具有任何值（也可以为空）。	vfs.file.contents[/etc/passwd,] vfs.file.contents[/etc/passwd,utf8]	vfs.file.contents[/etc/passwd] vfs.file.contents[/tmp/test]
vfs.file.contents[/var/log/zabbix_server.log,,abc]*	匹配 `vfs.file.contents` 与第一个参数匹配 /var/log/zabbix_server.log，第三个参数匹配'abc' 和任何（也可以为空）第二个参数。	vfs.file.contents[/var/log/zabbix_server.log,,abc] vfs.file.contents[/var/log/zabbix_server.log,utf8,abc]	vfs.file.contents[/var/log/zabbix_server.log]
vfs.file.contents[/etc/passwd,utf8]	匹配 `vfs.file.contents`，第一个参数匹配 /etc/passwd，第二个参数匹配'utf8'，没有其他参数。	vfs.file.contents[/etc/passwd,utf8]	vfs.file.contents[/etc/passwd,] vfs.file.contents[/etc/passwd,utf16]
*vfs.file.**	匹配任何以 `vfs.file.` 开头且不带任何参数的键。	vfs.file.contents vfs.file.size	vfs.file.contents[] vfs.file.size[/var/log/zabbix_server.log]
vfs.file.[*]*	匹配任何以 `vfs.file.` 开头的键和任何参数。	vfs.file.size.bytes[] vfs.file. 大小 [/var/log/zabbix_server.log, utf8]	vfs.file.size.bytes
vfs..contents*	匹配任何以 `vfs.` 开头并以 `.contents` 结尾且不带任何参数的键。	vfs.mount.point.file.contents vfs..contents	vfs.contents

system.run 和 AllowKey

假设有一个 "myscript.sh" 这样脚本，它可以通过 Zabbix 代理以多种方式在主机上执行：

1. 作为被动或主动检查中的项目键，例如：

- system.run[myscript.sh]
- system.run[myscript.sh,wait]
- system.run[myscript.sh.nowait]

在这里，用户可以添加 "wait"、"nowait" 或省略第二个参数，使用 system.run[] 的默认值。

2. 作为全局脚本（由用户在前端或 API 中启动）。

用户在 *Administration* → *Scripts* 中配置此脚本，设置 "在 Zabbix agent 上执行" 并将 "myscript.sh" 放入脚本的 "命令" 输入字段。当从前端或 API 调用时，由 Zabbix server 发送到代理上：

- system.run[myscript.sh,wait] - 最高到 Zabbix 5.0.4
- system.run[myscript.sh] - 从 5.0.5 开始

这里用户不控制 "wait"/"nowait" 参数。

3. 作为动作的远程命令。Zabbix server 发送到 agent：

- system.run[myscript.sh,nowait]

在这里，用户再次不控制 "wait"/"nowait" 参数。

这意味着如果我们将 AllowKey 设置为：

```
AllowKey=system.run[myscript.sh]
```

然后

- system.run[myscript.sh] - 将被允许
- system.run[myscript.sh,wait], system.run[myscript.sh,nowait] 将不允许 - 如果作为行动步骤

要允许所有描述的变体，您可以添加：

```
AllowKey=system.run[myscript.sh,*]
DenyKey=system.run[*]
```

到 agent/agent2 参数。

14 插件

概览

插件提供了扩展 Zabbix 监控功能的选项。插件是用 Go 语言编写的，并且只有 Zabbix agent2 支持。

插件提供了可加载模块 (由 C 编写)，以及其他扩展 Zabbix 功能的方法，例如用户参数(agent 指标),外部检查 (无代理监控) 和 `system.run[]` Zabbix agent 监控项。

以下功能特定于 agent 2 及其插件：

- 被动、主动检查都支持定期和灵活采集间隔；
- 与调度和任务并发相关的任务队列管理；
- 插件级超时；
- Zabbix agent 2 及其插件在启动时的兼容性检查。

从 Zabbix 6.0.0 开始，插件不必直接集成到 agent2 中，并且可以作为单独的外部插件添加，从而使得创建额外的插件来收集新的监控指标更加容易。

本页列出了 Zabbix 原生插件，并从用户角度描述了插件配置原则。有关编写自己的插件的说明，请参阅插件开发指南。

配置插件

本节提供常用的插件配置原则和最佳实践。

所有插件都使用 *Plugins.** 参数配置，可以是 Zabbix agent 2 配置文件的一部分或插件自己的配置文件。如果插件使用单独的配置文件，该文件的路径应该在 Zabbix agent 2 配置文件的 Include 参数中指定。

每个插件参数应具有以下结构：

Plugins.<PluginName>.<Parameter>=<Value>

参数名称应符合以下要求：

- 建议您的插件名称大写；
- 参数应大写；
- 不允许有特殊字符;
- 嵌套不受最高层数限制；
- 参数数量不受限制。

硬编码默认值

如果监控项键或命名会话参数中未提供身份验证所需的参数，则插件将使用硬编码默认值。

命名会话

命名会话代表附加级别的插件参数，可以用于为每个被监视的实例定义单独的身份验证参数集。每个命名的会话参数应具有以下结构：

Plugins.<PluginName>.Sessions.<SessionName>.<Parameter>=<Value>

会话名称可以当作连接字符串来配置监控项键值参数，而不是分别指定 URI、用户名和密码。在监控项键中，第一个参数可以是连接字符串或 Uri。如果第一个键参数匹配配置文件中指定的会话名称，检查将使用命名会话参数执行。如果第一个 key 参数没有匹配到会话名称，它将被视为 Uri。

注意：

- 在键值参数中使用连接字符串（会话名称）时，键值参数的用户名和密码必须为空；
- 不支持传递嵌入式 URI 凭据，请考虑使用命名会话；
- 如果没有为命名会话指定身份验证参数，将使用写死的默认值。

可用的命名会话参数列表取决于插件，详情参阅独立插件配置文件 。

示例：监控 “MySQL1” 和 “MySQL2” 两个实例可以采用如下配置方式：

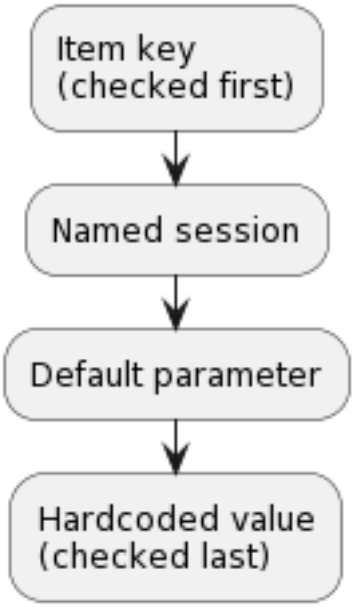
```
Plugins.Mysql.Sessions.MySQL1.Uri=tcp://127.0.0.1:3306
Plugins.Mysql.Sessions.MySQL1.User=<UsernameForMySQL1>
Plugins.Mysql.Sessions.MySQL1.Password=<PasswordForMySQL1>
Plugins.Mysql.Sessions.MySQL2.Uri=tcp://127.0.0.1:3307
Plugins.Mysql.Sessions.MySQL2.User=<UsernameForMySQL2>
Plugins.Mysql.Sessions.MySQL2.Password=<PasswordForMySQL2>
```

现在，这些名称可以作为连接字符串用于键值中，而不是 URIs：

```
mysql.ping[MySQL1]
mysql.ping[MySQL2]
```

Parameter priority

Since version 6.4.3, Zabbix agent 2 plugins search for connection-related parameter values in the following order:



1. The first item key parameter is compared to session names. If no match is found it is treated as an actual value; in this case, step 3 will be skipped. If a match is found, the parameter value (usually, a URI) must be defined in the named session.
2. Other parameters will be taken from the item key if defined.
3. If an item key parameter (for example, password) is empty, plugin will look for the corresponding named session parameter.
4. If the session parameter is also not specified, the value defined in the corresponding default parameter will be used.
5. If all else fails, the plugin will use the hardcoded default value.

Example 1

Monitoring of two instances “MySQL1” and “MySQL2”.

Configuration parameters:

```
Plugins.Mysql.Sessions.MySQL1.Uri=tcp://127.0.0.1:3306
Plugins.Mysql.Sessions.MySQL1.User=mysql1_user
Plugins.Mysql.Sessions.MySQL1.Password=unique_password
Plugins.Mysql.Sessions.MySQL2.Uri=tcp://192.0.2.0:3306
Plugins.Mysql.Sessions.MySQL2.User=mysql2_user
Plugins.Mysql.Sessions.MySQL2.Password=different_password
```

Item keys: `mysql.ping[MySQL1]`, `mysql.ping[MySQL2]`

Example 2

Providing some of the parameters in the item key (supported since Zabbix 6.0.17).

Configuration parameters:

```
Plugins.Postgres.Sessions.Session1.Uri=tcp://192.0.2.234:5432
Plugins.Postgres.Sessions.Session1.User=old_username
Plugins.Postgres.Sessions.Session1.Password=session_password
```

Item key: `pgsql.ping[session1,new_username,,postgres]`

As a result of this configuration, the agent will connect to PostgreSQL using the following parameters:

- URI from session parameter: *192.0.2.234:5432*
- Username from the item key: *new_username*
- Password from session parameter (since it is omitted in the item key): *session_password*
- Database name from the item key: *postgres*

Example 3

Collecting a metric using default configuration parameters.

Configuration parameters:

```
Plugins.Postgres.Default.Uri=tcp://192.0.2.234:5432
Plugins.Postgres.Default.User=zabbix
Plugins.Postgres.Default.Password=password
```

Item key: `pgsql.ping[,,,postgres]`

As a result of this configuration, the agent will connect to PostgreSQL using the parameters:

- Default URI: *192.0.2.234:5432*
- Default username: *zabbix*
- Default password: *password*
- Database name from the item key: *postgres*

连接

一些插件支持同时从多个实例收集指标。可以监控本地和远程实例。并支持 TCP 和 Unix-socket 连接。

建议配置插件以保持与实例的连接处于打开状态。好处是减少了网络拥塞、延迟、CPU 和内存使用率，因为连接数较少。客户端库负责处理这些。

Note:
未使用的连接应保持打开的时间段可由 *Plugins.<PluginName>.KeepAlive* 参数确定。示例：*Plugins.Memcached.KeepAlive*

插件

Zabbix agent 2 支持的所有指标都由插件收集。

内置

Zabbix agent 2 的以下插件开箱即用。单击插件名称以转到包含其他信息的插件存储库。

插件名称	描述	支持的监控项键	注释
Agent	正在使用的 Zabbix agent 的指标。	agent.hostname、agent.ping、agent.version	支持的键与 Zabbix agent key 具有相同的参数。
Ceph	Ceph 监控。	ceph.df.details，ceph。osd.stats、ceph.osd.discovery、ceph.osd.dump、ceph.ping、ceph.pool.discovery、ceph.status	
CPU	系统 CPU 监控（CPU 数/CPU 核数，发现的 CPU 数，CPU 利用率).	system.cpu.discovery, system.cpu.num, system.cpu.util	支持的键与 Zabbix agent keys 具有相同的参数。
Docker	Docker 容器的监控。	docker.container_info, docker.container _stats、docker.containers、docker.containers.discovery、docker.data_usage、docker.images、docker.images.discovery、docker.info、docker.ping	另请参阅：配置参数
File	文件指标集合。	vfs.file.cksum、vfs.file.contents、vfs.file.exists、vfs.file.md5sum、vfs.file.regexp、vfs.file.regmatch、vfs.file .size, vfs.file.time	支持的键与 Zabbix agent keys 具有相同的参数。
Kernel	内核监控。	kernel.maxfiles, kernel.maxproc	支持的键有与 Zabbix agent keys 具有相同的参数。

插件名称	描述	支持的监控项键	注释
Log	日志文件监控。	log, log.count, logrt, logrt.count	支持的键与 Zabbix agent keys 具有相同的参数。 另请参阅： 插件配置参数 (Unix/Windows)
Memcached	Memcached 服务器监控。	memcached.ping, memcached.stats	
Modbus	读取 Modbus 数据。	modbus.get	支持的键与 Zabbix agent keys 具有相同的参数。
MQTT	接收 MQTT 主题的发布值。	mqtt.get	
MySQL	MySQL 及其分支的监控。	mysql.db。discovery, mysql.db.size, mysql.get_status_variables, mysql.ping, mysql.replication.discovery, mysql.replication.get_slave_status, mysql.version	配置数据库的加密连接，使用命名会话 并在 agent 配置文件中为命名会话指定 TLS 参数。目前，TLS 参数不能作为监控项键参数传递。
NetIf	网络接口监控。	net.if.collisions, net.if.discovery, net.if.in, net.if.out, net.if.total	支持的键与 Zabbix agent keys 具有相同的参数。
Oracle	Oracle 数据库监控。	oracle.diskgroups.stats, oracle .diskgroups.discovery, oracle.archive.info, oracle.archive.discovery, oracle.cdb.info, oracle.custom.query, oracle.datafiles.stats, oracle.db.discovery, oracle.fra .stats、oracle.instance.info、oracle.pdb.info、oracle.pdb.discovery、oracle.pga.stats、oracle.ping、oracle.proc.stats、oracle.redolog.info、oracle .sga.stats、oracle.sessions.stats、oracle.sys.metrics、oracle.sys.params、oracle.ts.stats、oracle.ts.discovery、oracle.user.info	在使用插件之前安装 Oracle Instant Client。
Proc	进程 CPU 使用率。	proc.cpu.util	支持的键与 Zabbix agent key 具有相同的参数。
Redis	Redis 服务器监控。	redis.config, redis.info ，redis.ping ，redis.slowlog.count	
Smart	S.M.A.R.T. monitoring.	smart.attribute.discovery, smart.disk.discovery, smart.disk.get	执行 Zabbix agent 2 的用户需要对 smartctl 的 Sudo/root 访问权限。所需的最低 smartctl 版本为 7.1。 < br> 支持的keys 只能在 Linux/Windows 上与 Zabbix agent 2 一起使用，作为被动和主动检查。 另请参阅： 配置参数
Swap	以字节/百分比为单位的交换空间大小。	system.swap.size	支持的键与 Zabbix agent keys 具有相同的参数。

插件名称	描述	支持的监控项键	注释
SystemRun	运行指定命令。	system.run	支持的键与 Zabbix agent keys 具有相同的参数。 另请参阅： 插件配置参数 (Unix/Windows)
Systemd	systemd 服务的监控。	systemd.unit.discovery、systemd.unit.get、systemd.unit.info	
TCP	TCP 连接可用性检查。	net.tcp.port	支持的键与 Zabbix agent key 具有相同的参数。
UDP	UDP 服务可用性和性能的监控。	net.udp.service, net.udp.service.perf	支持的键与 Zabbix agent keys 具有相同的参数。
Uname	检索有关系统的信息。	system.hostname, system. sw.arch, system.uname	支持的键与 Zabbix agent keys 具有相同的参数。
Uptime	系统正常运行时间指标收集。	system.uptime	支持的键与 Zabbix agent keys 具有相同的参数。
VFSDev	VFS 指标收集。	vfs.dev.discovery, vfs.dev .read, vfs.dev.write	支持的键与 Zabbix agent keys 具有相同的参数。
WebCertificate	TLS/SSL 网站证书监控。	web.certificate.get	
WebPage	网页监控.	web.page.get, web.page .perf, web.page.regexp	支持的键与 Zabbix agent keys 具有相同的参数。
ZabbixAsync	异步指标收集。	net.tcp.listen, net.udp.listen, sensor, system.boottime, system.cpu.intr, system.cpu.load、system.cpu.switches、system.hw.cpu、system.hw.macaddr、system.localtime、system.sw。os, system.swap.in, system.swap.out, vfs.fs.discovery	支持的键与 Zabbix agent keys 具有相同的参数。
ZabbixStats	Zabbix server/proxy 内部指标或队列中延迟监控项的数量.	zabbix.stats	支持的键与 Zabbix agent keys 具有相同的参数。
ZabbixSync	同步指标收集。	net.dns, net.dns.record , net.tcp.service, net.tcp.service.perf, proc.mem, proc.num, system.hw.chassis, system.hw.devices, system.sw.packages, system.users .num、vfs.dir.count、vfs.dir.size、vfs.fs.get、vfs.fs.inode、vfs.fs.size、vm.memory.size。	支持的键与 Zabbix agent keys具有相同的参数。

可加载

Note:
可加载插件，启动时：
 - *-V --version* - 打印插件版本和许可信息；
 - *-h --help* - 打印帮助信息。

单击插件名称以转到包含其他信息的插件存储库。

插件名称	描述	支持的监控项键	注释
MongoDB	MongoDB 服务器和集群（基于文档的分布式数据库）的监控。	mongodb.collection.stats, mongodb.collections.discovery, mongodb.collections.usage, mongodb.connpool.stats, mongodb.db.stats, mongodb.db.discovery, mongodb.jumbo_chunks.count, mongodb.oplog.stats, mongodb.ping, mongodb.rs.config, mongodb.rs.status, mongodb.server.status, mongodb.sh.discovery	此插件自 Zabbix 6.0.6（之前内置）起可加载。 要配置与数据库的加密连接，请使用命名会话并在 agent 配置文件中为命名会话指定 TLS 参数。 在插件版本 1.2.1、6.0.13 和更新版本中受支持[1]。 目前 TLS 参数不能作为监控项键参数传递。 另见 [MongoDB 插件配置文件](/manual/appendix/config/zabbix_agent2_plugins/mongo 配置文件）。
PostgreSQL	PostgreSQL 及其分支的监控。	pgsql.autovacuum.count, pgsql.archive, pgsql.bgwriter, pgsql.cache.hit, pgsql.connections, pgsql.custom.query, pgsql.dbstat, pgsql.dbstat.sum, pgsql.db.age, pgsql.db.bloating_tables, pgsql.db.discovery, pgsql.db.size, pgsql.locks, pgsql.oldest.xid, pgsql.ping, pgsql.queries, pgsql.replication.count, pgsql.replication.process, pgsql.replication.process.discovery, pgsql.replication.recovery_role, pgsql.replication.status, pgsql.replication_lag.b, pgsql.replication_lag.sec, pgsql.uptime, pgsql.wal.stat	此插件自 Zabbix 6.0.10（以前内置）起可加载。 要配置与数据库的加密连接，请使用命名会话并在 agent 配置文件中为命名会话指定 TLS 参数。目前，TLS 参数不能作为监控项键参数传递。

另请参阅：构建可加载插件。

脚注

[1] - 从 Zabbix 6.0.13 开始，可加载插件开始使用与 Zabbix 本身相同的版本控制系统。因此，MongoDB 插件版本从 1.2.1 更改为 6.0.13。

1 构建可加载插件

概览

此页面提供了从源码构建可加载插件二进制文件所需的步骤。

如果下载了源码压缩包，则可以离线构建插件，例如，没有互联网连接。

以 PostgreSQL 插件为例。可以用类似的方式构建其他可加载插件。

步骤

1. 从Zabbix Cloud Images and Appliances下载插件源码。官方下载页面即将上线。

2. 将压缩包传输到您要构建插件的机器上。

3. 解压压缩包，例如:

tar xvf zabbix-agent2-plugin-postgresql-6.0.13.tar.gz

确保将 “zabbix-agent2-plugin-postgresql-6.0.13.tar.gz” 替换为下载存档的名称。

4. 进入解压后的目录：

cd <path to directory>

5. 运行:

make

6. 插件可执行文件可以放置在任何地方，只要它可以被 Zabbix agent 2 加载。在插件配置文件中指定插件二进制文件的路径，例如在 PostgreSQL 插件的 postgresql.conf 中：

Plugins.PostgreSQL.System.Path=/path/to/executable/zabbix-agent2-plugin-postgresql

7. 必须在 Zabbix agent 2 配置文件的 Include 参数中指定插件配置文件的路径：

Include=/path/to/plugin/configuration/file/postgresql.conf

生成文件目标

Zabbix 提供的可加载插件具有具有以下目标的简单 makefile：

目标	描述
make	构建插件。
make clean	删除通常通过构建插件创建的所有文件。
make check	进行自检。需要一个真正的 PostgreSQL 数据库。
make style	使用 “golangci-lint” 检查 Go 代码风格。
make format	使用 “go fmt” 格式化 Go 代码。
make dist	创建一个存档，其中包含构建插件及其自测所需的所有包的插件源和源。

3 触发器

概述

触发器是 “评估” 监控项采集的数据和表示当前系统状况的逻辑表达式。

当我们使用监控项用于采集系统的数据时，一种不切合实际的情况是，我们无法始终等待一个警告或值得关注的条件。综上 “评估” 监控项数据的工作，我们可以留给触发器表达式。

触发器表达式允许定义一个什么状况的数据条件是 “可接受” 的阈值。因此，如果接收的数据超过了可接受的状态条件，则触发器会被触发 - 或将状态更改为异常。

一个触发器可以拥有下面几种状态：

值	述
OK	这是一个正常的触发器状态。在旧版本的 Zabbix 中称为 FALSE。
PROBLEM	通常意味着发生了某些事情。例如，CPU 负载较高。在旧版本的 Zabbix 中称为 TRUE。

在基本触发器配置中，我们可能希望为某些监控数据的五分钟平均值设置告警阈值，例如 CPU 负载。这是通过定义一个触发器表达式来完成的，其中：

- 我们将 “avg” 函数应用于监控项键中收到的值
- 我们使用五分钟的时间进行评估
- 我们将阈值设置为 “2”

如果五分钟平均值超过 2，此触发器将 “触发”（成为问题 PROBLEM）。

在更复杂的触发器中，表达式可能包含组合具有多种功能和多种阈值。也可以看看：Trigger expression.

大多数触发函数的评估基于history数据，而一些用于长期分析的触发函数，例如 **trendavg()**，**trendcount()** 等，使用趋势数据。

> **Note:**
> 启用触发器后（将其配置状态从 禁用更改为 启用），触发器表达式会在其中的监控项收到值或处理基于时间的函数的时间到来时立即求值。

大多数触发函数都是根据监控项值进行评估的历史 数据，而一些用于长期分析的触发功能，例如 趋势平均值 **()**,**trendcount()** 等，使用趋势数据。

计算时间

每次 Zabbix 服务器收到作为表达式一部分的新值时，都会重新计算触发器。当接收到新值时，表达式中包含的每个函数都会重新计算（不仅仅是接收到新值的函数）。

此外，如果在表达式中使用了基于时间的函数，则每次接收到新值和每 30 秒重新计算一次触发器。

基于时间的函数是 **nodata()**、**date()**、**dayofmonth()**、**dayofweek()**、**time()**、**now()**；Zabbix 历史同步器进程每 30 秒重新计算一次。

引用趋势函数的触发器 **only** 在表达式中的每个最小时间段评估一次。另见趋势函数。

评估周期

在引用监控项历史的函数中使用评估周期。它允许指定我们感兴趣的间隔。它可以是指定为时间段 (30s, 10m, 1h) 或值范围 (#5 - for 五个最近值）。

评估周期测量到 “now(现在)” - 其中 “now(现在)” 是触发器的最新重新计算时间（参见Calculation time 上面）；“now(现在)” 不是现在的 “now(现在)” 服务器时间。

评估周期指定：

- 考虑 “now-time period（现在时间周期）” 和 “now(现在)” 之间的所有值（或者，与时间偏移量，在 “now-time shift-time period(现在时间偏移点时间区间)” 和 “now-time_shift(现在\偏移量)”）
- 考虑不超过过去的值的数量，向上到 “now(现在)”
  - 如果时间周期或计数有 0 个可用值指定 - 然后是使用它的触发器或计算项功能变得不受支持.

注意：

- 如果在触发器中只使用一个函数 (引用监控数据历史)，那么 “now（现在）” 总是最新收到的值。例如，如果最后一个值是在一小时前收到的，那么评估期将被视为一小时前的最新值。
- 当第一个监控值被接收时，一个新的触发器被计算出来 (history functions 历史函数); 基于时间的函数将在 30 秒内计算。因此，即使设置的评估周期 (例如一个小时) 自触发器创建以来还没有过去，也将计算触发器。触发器也将在第一个值之后计算，即使计算范围被设置为 10 个最新的值。

未知状态

在以下情况下，触发器表达式中可能会出现未知操作数：

- ·使用了不受支持的监控项
- ·支持监控项的功能评估导致错误

在这种情况下，触发器通常评估为 “未知”（尽管有一些例外）。有关详细信息，请参阅具有未知操作数的表达式。

可以在未知触发器上 [得到通知] (/manual/config/events/sources#internal-events)。

1 配置一个触发器

概述

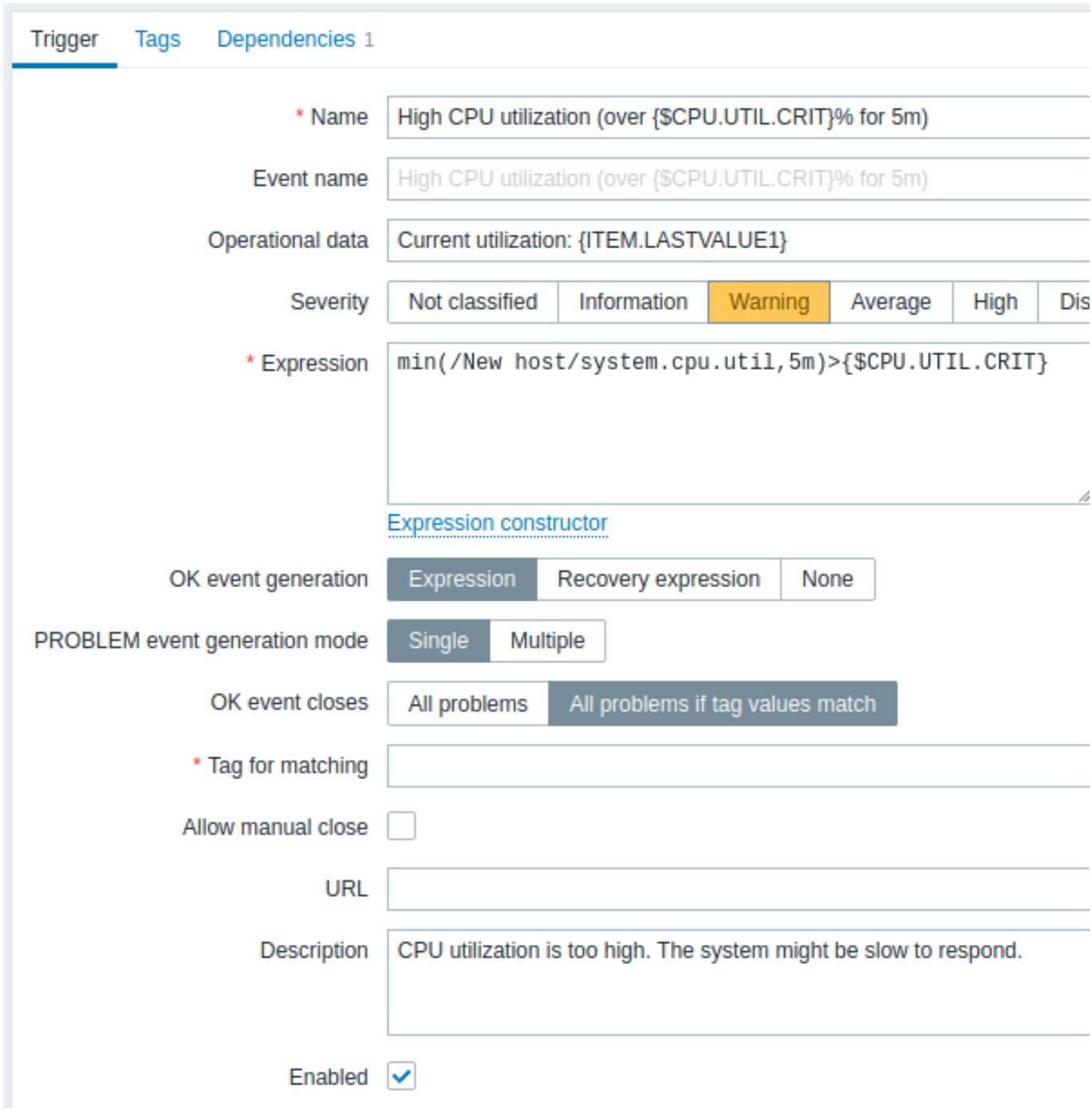
配置一个触发器，按以下步骤操作：

- 进入：配置 → 主机
- 点击主机一行的 触发器
- 点击右上角的 创建触发器（或者点击触发器名称去修改一个已存在的触发器）
- 在窗口中输入触发器的参数

另请参阅有关触发器及其计算时间的一般信息。

配置

Trigger(触发器) 标签页包含了所有必要的触发器属性。



所有强制输入字段都用红色星号标记。

参数	描述
名称	触发器名称. 支持的macros (宏): {HOST.HOST}, {HOST.NAME}, {HOST.PORT}, {HOST.CONN}, {HOST.DNS}, {HOST.IP}, {ITEM.VALUE}, {ITEM.LASTVALUE {ITEM.LOG.*} and {$MACRO}。 **$1, $2...$9** 宏可以用来指第一, 第二... 第九表达式的常量。 备注: 如果引用了相对简单的常量或明确的表达式, $1-$9

参数	描述
事件名称	如果定义了这个名称，将使用这个名称来创建问题事件名称，而不是触发器名称。事件名称可用于构建包含问题数据的有意义的警报(查看示例)。支持与触发器名

参数	描述
操作数据	操作数据允许定义任意字符串和宏。这些宏会在 *Monitoring* (监控) → *Problems* (问题) 中动态地解析实时数据。触发器名称中的宏将在问题发生时解析为它们的值，并成

参数	描述
严重性	通过点击对应的按钮来设置所需的触发器severity。

参数	描述
表达式	定义问题条件的逻辑表达式。 当表达式中包含的所有条件都满足时，就会产生问题，即表达式的计算结果为TRUE。 除非在恢复表达式中指定了额外的恢复条件，否

参数	描述
生成 *OK* 事件	OK 事件生成选项： **Expression**(表达式) - OK 事件基于与问题事件相同的表达式生成; **Recovery expression**(恢复表达式) - 如果问题表达式计算为 false，恢复表达式计算为 true，则生成 OK 事件;

参数	描述
恢复表达式	逻辑expression]用于定义问题解决的条件。只有在表达式表达式计算为FALSE之后才对恢复表达式进行评估。恢复表达式有利于配置触发器滞后。如果问题条件仍然存在，则

参数	描述
问题事件生成模式	生成问题事件的模式： **Single**（单个）- 当触发器第一次进入‘Problem‘状态时，生成一条单个事件。 **Multiple**（多重）- 每一个触发器“Problem”评估都将生产一条事件。

参数	描述
事件成功关闭	选择OK事件是否关闭：**All problems**(所有问题) - 此触发器的所有问题 **All problems if tag values match**(所有问题如果标签值匹配) - 只有那些匹配事件标签值引发的问题。从Zabbix 3.2.0

参数	描述
匹配标记	输入事件标签名称以用于事件关联。如果在事件成功关闭中选择了‘所有问题如果标签值匹配’，在这种情况下是强制性的。从 Zabbix 3.2.0 开始支持。

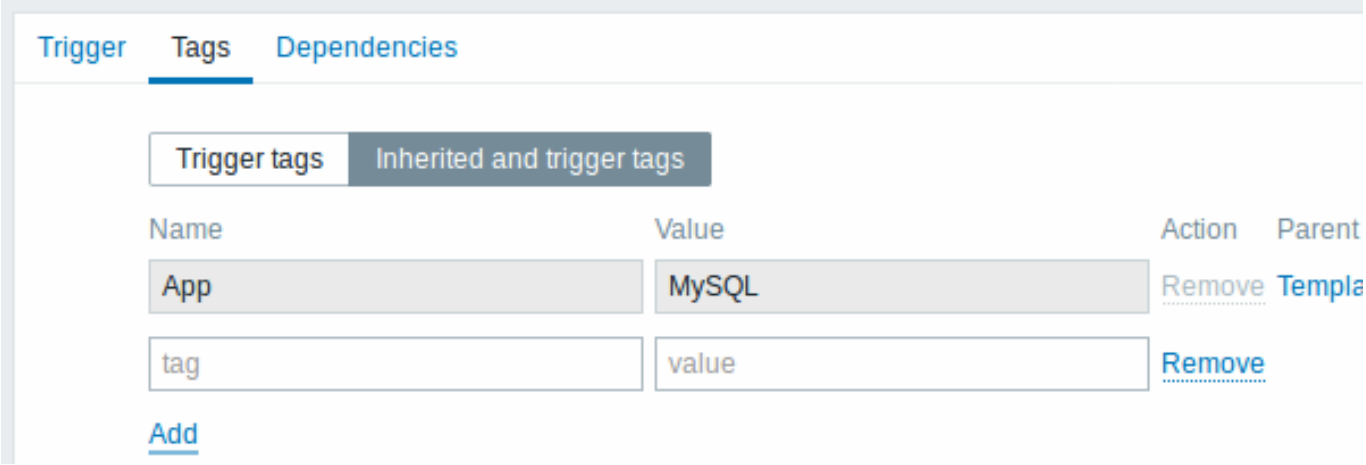
参数	描述
允许手动关闭	检查是否允许手动关闭由该触发器生成的问题事件。 在确认问题事件时，手动关闭是可能的。 从 Zabbix 3.2.0 开始支持。

参数	描述
URL	如果不为空，在监测→问题和问题仪表板中点击触发器名称，这里输入的URL可以作为链接。(*URL*选项在触发器上下文菜单) 可以在触发器URL字段中使用宏，例如

参数	描述
描述	文本字段用于提供有关此触发器的更多信息。可能包含解决具体问题的指令、负责人员的联系细节等。从 *Zabbix 2.2* 开始，描述可以包含与触发器名称相同的宏集

参数	描述
已启用	不选中该框将禁用触发器。问题事件对应的触发器被禁用，则前端不显示该事件但不会删除。

Tags（标签）选项卡允许您定义触发级tags。此触发器的所有问题都将使用此处输入的值进行标记。



此外，*Inherited and trigger tags*（继承和触发器标记）选项允许查看在模板级定义的标记，如果触发器来自该模板。如果有多个模板使用相同的标记，这些标记只显示一次，并且用逗号分隔模板名。触发器不“继承”和显示主机级标记。

Parameter	Description
Name（名称）*/Value*（值）	设置自定义标记来标记触发事件。 标签是一对标签名和值。只能使用名称或将其与值配对。一个触发器可能有几个名称相同但值不同的标记。 用户宏，用户宏上下文，低级别发现宏和宏functions(函数)带有'{{ITEM.VALUE}},{{ITEM.LASTVALUE}} 和低级别发现宏。低级别发现宏可以在宏上下文中使用。 {TRIGGER.ID} 宏。它可能有助于识别从触发器原型创建的触发器，例如，在维护期间抑制这些触发器产生的问题。当扩展值的总长度超过 255 时，将被截断为 255 个字符。 查看所有支持事件标签的macros（宏）。 Event tags（事件标签）可以用于事件关联，在动作条件中，也可以在 *Monitoring→Problems* 或 *Problems* widget 中看到

Dependencies（依赖项）选项卡包含触发器的所有dependencies。

单击 *Add* 添加新的依赖项。

您还可以通过打开现有触发器来配置触发器，按下 *Clone* 按钮，然后保存在不同的位置名称。

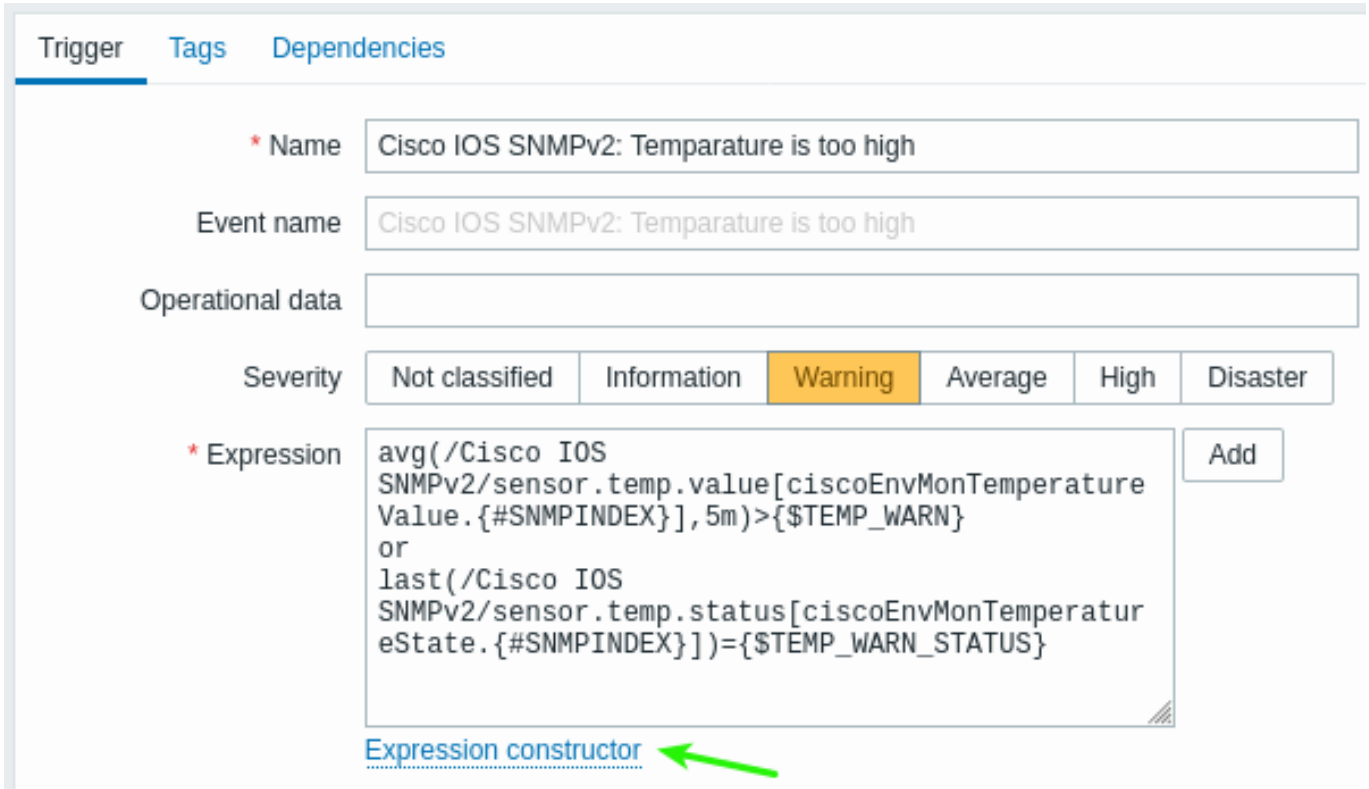
测试表达式

可以根据接收的值测试配置的触发器表达式，以确定表达式结果。

以官方模板的表达为例：

```
avg(/Cisco IOS SNMPv2/sensor.temp.value[ciscoEnvMonTemperatureValue.{#SNMPINDEX}],5m)>{$TEMP_WARN}
or
last(/Cisco IOS SNMPv2/sensor.temp.status[ciscoEnvMonTemperatureState.{#SNMPINDEX}])={$TEMP_WARN_STATUS}
```

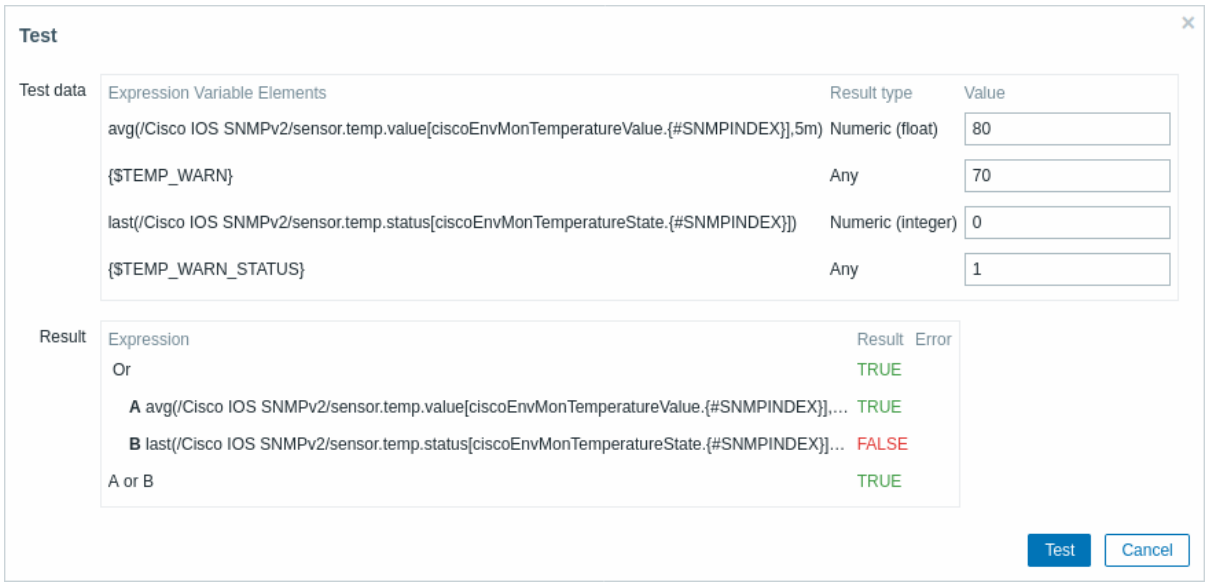
若要测试表达式，请点击表达式字段下的 *Expression constructor*（表达式构造器）。



在表达式构造器列出了所有单个表达式。打开测试窗口，点击在表达式列表下方 *Test(*测试*)*。



在测试窗口中，您可以输入示例值（在这个示例中为 “80, 70, 0，1”），然后点击 测试按钮查看表达式结果。



可以看到每个表达式的结果以及整个表达式的结果。

“TRUE” 结果意味着指定的表达式是正确的。在这个特定的情况 A 下，“80” 大于 {$TEMP_WARN} 指定值 “70”，出现 “TRUE” 结果。

“FALSE” 结果表示指定的表达式不正确。在这个特定的情况 B 下，在这个例子中 {$TEMP_WARN_STATUS} 是 “1”，需要与指定的 “0” 值相等，这是错误的。出现 “FALSE” 结果。

选择的表达式类型是 “OR”。如果指定条件中的至少一个（在这种情况下为 A 或 B）是真的，那么最终结果也是 TRUE。意味着，当前值超过了警告值，出现了异常。

2 触发器表达式

概览

triggers(触发器) 中使用的表达式非常灵活。您可以使用它们来创建复杂的逻辑测试监控统计。

一个简单的表达式使用 **function**（函数）应用于监控项的一些参数。该函数返回一个与阈值比较后的结果，使用运算符和常数。

一个简单有用的表达式的语法是 `function(/host/key,parameter)<operator><constant>`.

例如:

```
min(/Zabbix server/net.if.in[eth0,bytes],5m)>100K
```

如果在最后五分钟期间接收到的网络字节数最小值超过 100 KB 将触发。

虽然语法完全相同，但从功能的角度来看有两种类型的触发器表达式：

- 问题表达 - 定义问题的条件
- 恢复表达式（可选）- 定义附加条件问题解决

当单独定义问题表达式时，该表达式将同时用作问题阈值和问题恢复阈值。只要问题表达式的计算结果为 TRUE，就会出现问题。只要问题表达式的计算结果为 FALSE，问题就解决了。

当定义问题表达式和补充恢复表达式时，问题解决将变得更加复杂: 不仅问题表达式必须为 FALSE，而且恢复表达式也必须为 TRUE。这对于创建hysteresis 滞后和避免触发器振荡非常有用。

函数

函数允许计算收集的值（average 平均值、minimum 最小值、maximum 最大值、sum 总和）、查找字符串、参考当前时间和其他因素。

支持的函数可用的完整列表。

通常，函数会返回数值以进行比较。返回字符串时，可以使用 **=** 和 **<>** 运算符进行比较（参见示例）。

函数参数

函数参数允许选项：

- 主机和监控项（仅引用主机项目历史的功能）
- 功能特定的参数
- 其他表达式（不适用于引用主机的函数监控项，历史数据，请参阅other expressions（其他表达式） 例子）

主机和监控项键可以指定为/host/key。被引用的监控项必须处于支持的状态（**nodata()** 无数据函数除外，这也是为不受支持的监控项计算的）。

而作为函数参数的其他触发器表达式仅限于触发器中的非历史函数，此限制不适用于calculated items（计算监控项）。

特定函数参数

特定函数的参数放置在监控项之后，并且是用逗号与监控项键隔开。请参阅supported functions（支持的函数）获取这些的完整列表参数。

Preceded by a hashtag, the parameter has a different meaning: 大多数数值函数都接受时间作为参数。您可以使用 seconds 或time suffixes（时间后缀） 来表示时间。前置 # 号的参数有不同的含义：

表达式	描述
sum(/host/key,**10m)**	最近 10 分钟的数值总和.
sum(/host/key,**#10)**	最后十个值的总和.

前置 # 号的参数用于 **last** 函数含义不同 - 它们表示第 N 个先前的值，因此给定值 3、7、2、6、5（从最近到最近）：(从最近到最远):

- last(/host/key,#2) 返回'7'
- last(/host/key,#5) 返回'5'

时间偏移

支持可选的时间偏移，时间或值计数作为函数参数。此参数允许从一个引用过去一段时间的数据。

时移以 now 开头 - 指定当前时间，并且是后跟 +N<time unit> 或 -N<time unit> - 加或减 N 时间单位

例如，avg(/host/key,1h:now-1d) 将返回一天前的一个小时的平均值。

具有绝对时间段的时移

时移参数支持绝对时间段，例如午夜，一天到午夜，一周的周一到周日，第一天到最后一天，一个月到下一个月。

绝对时间段的时移以 now 开始 - 指定当前时间，然后是任意数量的时间操作：/<time unit> - 定义时间单位的开始和结束，例如，从午夜到一天的午夜，+N<time unit> 或 -N<time unit> - 添加或减去 N 个时间单位.

请注意时移的值可以大于或等于 0，而时间段最小值为 1。

参数	描述
1d:now/d	昨天
1d:now/d+1d	今天
2d:now/d+1d	过去 2 天
1w:now/w	上星期
1w:now/w+1w	这星期

其他表达式

函数参数可能包含其他表达式，如下所示语法:

min(min(/host/key,1h),min(/host2/key2,1h)*10)

请注意，如果函数引用，则可能不会使用其他表达式监控项历史。例如，不允许使用以下语法：

~~min(/host/key,#5*10)~~

运算符

触发器支持下列运算符 （在执行中优先级递减）

优先级	运算符	定义	注意**unknown values**（未知值）	强制转换操作为浮点数 [1]
1	**-**	负数	**-**Unknown → Unknown	Yes
2	**not**	逻辑非	**not** Unknown → Unknown	Yes
3	*	乘法	0 * Unknown → Unknown (yes, Unknown, not 0 - to not lose 算术运算中的未知数) 1.2 * Unknown → Unknown	Yes

优先级	运算符	定义	注意**unknown values**（未知值）	强制转换操作为浮点数[1]
	/	除法	Unknown **/** 0 → error Unknown **/** 1.2 → Unknown 0.0 **/** Unknown → Unknown	Yes
4	**+**	算术加	1.2 **+** Unknown → Unknown	Yes
	-	算术减	1.2 **-** Unknown → Unknown	Yes
5	**<**	少于. 运算符定义为: A<B ⇔ (A<B-0.000001)	1.2 **<** Unknown → Unknown	Yes
	<=	小于或等于。 运算符定义为: A<=B ⇔ (A≤B+0.000001)	Unknown **<=** Unknown → Unknown	Yes
	>	多于。 运算符定义为： A>B ⇔ (A>B+0.000001)		Yes
	>=	More than or equal to. The operator is defined as: A>=B ⇔ (A≥B-0.000001)		Yes
6	**=**	等于。 运算符定义为： A=B ⇔ (A≥B-0.000001) and (A≤B+0.000001)		No [1]

优先级	运算符	定义	注意**unknown values**（未知值）	强制转换操作为浮点数 [1]
	<>	不等于。运算符定义为: A<>B ⇔ (A<B-0.000001) or (A>B+0.000001)		No [1]
7	**and**	逻辑与	0 **and** Unknown → 0 1 **and** Unknown → Unknown Unknown **and** Unknown → Unknown	Yes
8	**or**	逻辑或	1 **or** Unknown → 1 0 **or** Unknown → Unknown Unknown **or** Unknown → Unknown	Yes

[1] 如果字符串操作数仍然转换为数字:

- 另一个操作数是数字
- 运算符以外的 **=** or **<>** 在操作数上使用

(如果转换失败 - 数字操作数被转换为字符串操作数并且两个操作数都作为字符串进行比较。

not、**and** 和 **or** 运算符区分大小写，并且必须在小写。它们还必须用空格或括号括起来。

除 **-** 和 **not** 外，所有运算符都从左到右关联性。**-** 和 **not** 是非关联的（意思是 **-(-1)** 和 **not (not 1)** 应该用来代替 **--1** 和 不是不是 **1**)。

评价结果：

- **<**, **<=**, **>**, **>=**, **=**, **<>** 如果指定的运算符应在触发器表达式中产生 “1” 关系为真，如果为假，则为 “0”。如果至少有一个操作数是未知结果为未知；
- **and** 对于已知操作数，如果它的两个操作数都将产生 “1” 则不等于’0’；否则，它产生 “0”；对于未知仅当一个操作数比较等于时，操作数 **and** 才会产生 “0” ’0’; 否则，它会产生 “未知”.
- **or** 对于已知的操作数，如果它的任何一个操作数都将产生 “1” 则不等于’0’；否则，它产生 “0”；对于未知仅当一个操作数比较不等于 0 时，操作数 **or** 才会产生 “1”; 否则，它会产生 “未知”；
- 逻辑否定运算符 **not** 的结果如果其操作数的值比较不等于 “0”，则操作数为 “0”；如果其操作数的值比较等于 “0”，则为 “1”。对于未知操作数 **not** 产生 “未知”。

缓存值

触发器评估所需的值由 Zabbix server 缓存。由于此触发器评估在服务器重新启动后一段时间导致较高的数据库负载。当监控项历史数据被移除（手动或 housekeeper）时，缓存值不会被清除，因此服务器将使用缓存的值，直到它们比触发器函数中定义的时间段或服务器重启的时间长。

触发器示例

示例 1

Zabbix server 上的处理器负载太高。

```
last(/Zabbix server/system.cpu.load[all,avg1])>5
```

’/Zabbix server/system.cpu.load[all,avg1]’ 给出了被监控参数的简短名称。它指定了服务器是 “Zabbix server”，监控项的键值是 “system.cpu.load[all,avg1]”。通过使用函数 “last()” 获取最新的值。最后，“>5” 意味着当 Zabbix server 最新获取的处理器负载值大于 5 时触发器就会处于异常状态。

示例 2

www.example.com 已超载。

```
last(/www.example.com/system.cpu.load[all,avg1])>5 or min(/www.example.com/system.cpu.load[all,avg1],10m)>
```

当前处理器负载超过 5 或过去 10 分钟内 CPU 负载都超过 2。

示例 3

/etc/passwd 文件被修改

```
(last(/www.example.com/vfs.file.cksum[/etc/passwd],#1)<>last(/www.example.com/vfs.file.cksum[/etc/passwd],
```

当 /etc/passwd 校验和的前一个值与最近的值不同时，表达式为真。

类似的表达式可能有助于监控重要的文件变化，例如 /etc/passwd、/etc/inetd.conf、/kernel 等。

示例 4

服务器网卡从 Internet 下载一个大文件。min 函数的使用：

```
min(/www.example.com/net.if.in[eth0,bytes],5m)>100K
```

在过去 5 分钟内，eth0 上接收字节数大于 100kb 时，表达式为 true。

示例 5

SMTP 服务群集的两个节点都停止。注意在一个表达式中使用两个不同的主机:

```
last(/smtp1.example.com/net.tcp.service[smtp])=0 and last(/smtp2.example.com/net.tcp.service[smtp])=0
```

当两个 SMTP 服务器（smtp1.example.com 和 smtp2.example.com）的 smtp 服务关闭时为真。

示例 6

Zabbix 代理需要升级。函数 find() 的使用：

```
find(/example.example.com/agent.version,,"like","beta8")=1
```

如果 Zabbix 代理的版本为 beta8，则表达式为真。

示例 7

服务器无法访问。

```
count(/example.example.com/icmpping,30m,,"0")>5
```

如果主机 “example.example.com” 在过去 30 分钟内超过 5 次无法访问，则表达式为真。

示例 8

最近 3 分钟内没有心跳。函数 nodata() 的使用：

```
nodata(/example.example.com/tick,3m)=1
```

要使用这个触发器，’tick’ 必须定义成一个trapper 陷阱器监控项。主机应该使用 zabbix_sender 定期发送这个监控项的数据。如果在 180 秒内没有接收到数据，则触发值变为异常状态。

注释 ‘nodata’ 可以在任何类型的监控项中使用。

示例 9

夜间的 CPU 负载

使用 time() 函数:

```
min(/Zabbix server/system.cpu.load[all,avg1],5m)>2 and time()>000000 and time()<060000
```

触发器只能在晚上 (00:00-06:00) 将其状态更改为 true。

示例 10

CPU 活动随时异常。

使用函数 time() 和 **not** 运算符：

·min(/zabbix/system.cpu.load[all,avg1],5m)>2 ·and not (dayofweek()=7 and time()>230000) ·and not (dayofweek()=1 and time()<010000)

触发器可以随时将其状态更改为真，每周更改 2 小时（星期日，23:00 - 星期一，01:00）除外。

示例 10

检查客户端本地时间是否与 Zabbix 服务器时间同步。使用 fuzzytime() 函数:

```
fuzzytime(/MySQL_DB/system.localtime,10s)=0
```

当 MySQL_DB 服务器的本地时间与 Zabbix server 之间的时间相差超过 10 秒，触发器将变为异常状态。注意’system.localtime’ 必须配置为passive check 被动检查.

示例 11

比较今天的平均负载和昨天同一时间的平均负载（使用时移作为 `now-1d`）。

```
avg(/server/system.cpu.load,1h)/avg(/server/system.cpu.load,1h:now-1d)>2
```

如果最后一小时的平均负载超过昨天同一小时的平均负载两倍，触发器将触发。

示例 12

使用了另一个监控项的值来获得触发器的阈值：

```
last(/Template PfSense/hrStorageFree[{#SNMPVALUE}])<last(/Template PfSense/hrStorageSize[{#SNMPVALUE}])*0.
```

如果可用存储量低于 10%，触发器将触发。

示例 13

使用evaluation result 评估结果获取超过阈值的触发器数量:

```
(last(/server1/system.cpu.load[all,avg1])>5) + (last(/server2/system.cpu.load[all,avg1])>5) + (last(/serve
```

如果表达式中至少有两个触发器大于 5，触发器将触发。

示例 14

比较两个监控项的字符串值 - 这里的操作数是返回字符串的函数。

问题：如果两台主机 Ubuntu 版本不同，则产生告警。

```
last(/NY Zabbix server/vfs.file.contents[/etc/os-release])<>last(/LA Zabbix server/vfs.file.contents[/etc/
```

示例 15

比较两个字符串值 - 操作数是：

- 返回字符串的函数
- 宏和字符串的组合

问题：检测 DNS 查询的变化

监控项键是：

```
net.dns.record[8.8.8.8,{$WEBSITE_NAME},{$DNS_RESOURCE_RECORD_TYPE},2,1]
```

宏定义为

```
{$WEBSITE_NAME} = example.com
{$DNS_RESOURCE_RECORD_TYPE} = MX
```

并且通常返回：

```
example.com             MX       0 mail.example.com
```

检测 DNS 查询结果是否与预期的结果有偏差的触发器表达式如下：

```
last(/Zabbix server/net.dns.record[8.8.8.8,{$WEBSITE_NAME},{$DNS_RESOURCE_RECORD_TYPE},2,1])<>"{$WEBSITE_N
```

注意第二个操作数的引号。

示例 16

比较两个字符串值 - 操作数是：

- 返回字符串的函数
- 带有特殊字符 \ 和" 的字符串常量

问题：检测/tmp/hello 文件内容是否等于：

```
\" //hello ?\"
```

选项 1）直接写字符串

```
last(/Zabbix server/vfs.file.contents[/tmp/hello])="\\\" //hello ?\\\""
```

请注意在比较字符串时如何转义 \ 和" 字符。

选项 2) 使用宏

```
{$HELLO_MACRO} = \" //hello ?\"
```

在表达式中：

```
last(/Zabbix server/vfs.file.contents[/tmp/hello])={$HELLO_MACRO}
```

示例 17

比较长期周期。

问题：上个月 Exchange 服务器的负载增加了 10% 以上

trendavg(/Exchange/system.cpu.load,1M:now/M)>1.1*trendavg(/Exchange/system.cpu.load,1M:now/M-1M)

您也可以在触发器配置中使用 Event name（事件名称），构建有意义的警报消息，例如如下内容。

```
"Load of Exchange server increased by 24% in July (0.69) comparing to June (0.56)"
```

事件名称必须定义为：

```
Load of {HOST.HOST} server increased by {{?100*trendavg(//system.cpu.load,1M:now/M)/trendavg(//system.cpu.
```

这种情况在触发器配置中允许手动关闭也很有用。

滞后

有时，问题和恢复状态之间需要一个间隔，而不是一个简单的阈值。例如，如果我们想定义一个触发器，它在服务器机房温度高于 20°C 时报告问题，并且我们想让它在温度低于 15°C 之前保持问题状态，那么简单的触发器阈值为 20°C 是不够的。

相反，我们需要首先为问题事件定义一个触发器表达式 (温度高于 20°C)。然后我们需要定义一个额外的恢复条件 (温度低于 15°C)。这是通过在defining 定义触发器时定义一个额外的 *Recovery expression* 恢复表达式参数来实现的。

在这种情况下，问题恢复将分两步进行：

- 首先，问题表达式（温度高于 20°C）评估为 FALSE
- 其次，恢复表达式（温度低于 15°C）评估为 TRUE

仅当问题事件为先解决。

Warning:
当问题表达式为 TRUE 时，即使恢复表达式为 TRUE 也不会恢复。

示例 1

机房温度过高。

问题表达式:

```
last(/server/temp)>20
```

恢复表达式：

```
last(/server/temp)<=15
```

示例 2

磁盘剩余空间过低。

问题表达式: 最后 5 分钟小于 10GB

```
max(/server/vfs.fs.size[/,free],5m)<10G
```

恢复表达式：最近 10 分钟超过 40GB

```
min(/server/vfs.fs.size[/,free],10m)>40G
```

不支持项的表达式和未知的值

Zabbix3.2 之前的版本对触发器表达式中不支持的监控项非常严格。表达式中的任何不支持的监控项都会立即将触发器值呈现为 “未知”。

从 Zabbix3.2 开始通过将未知值引入到表达式评估中，对不受支持的项有更灵活的方法：

- 对于 nodata() 函数，监控项是否支持不影响。即使引用不支持的监控项，函数也会评估。
- 具有 OR 和 AND 的逻辑表达式可以计算为已知值无论未知操作数如何，在两种情况下：
  - ”1 or 不支持的监控项函数 1 or 不支持的监控项函数 2 or ...” 可以被评估为’1’ (True)，
  - ”0 and 不支持的监控项函数 1 and 不支持的监控项函数 2 and ...” 可以被评估为’0’ (False)，
    Zabbix 试图评估不支持的监控项作为 `Unknown` 值的逻辑表达式。在上述两种情况下，将产生一个已知值；在其他情况下，触发值将是 `Unknown`。
- 如果对受支持监控项的函数评估导致错误，则函数值为 `Unknown`，参与进一步表达评估。

请注意，未知值可能仅在逻辑表达式中 “消失” 为如上所述。在算术表达式中，未知值总是导致结果 “未知”（除以 0 除外）

如果具有多个不受支持的监控项的触发器表达式计算为 `Unknown` ，前端的错误信息指向最后一个评估不支持的监控项。

3 触发器依赖关系

概述

有时候一台主机的可用性依赖于另一台主机。如果一台路由器宕机，则路由器后端的服务器将变得不可用。如果这两者都设置了触发器，你可能会收到关于两个主机宕机的通知，然而只有路由器是真正故障的。

这就是主机之间某些依赖关系可能有用的地方，设置依赖关系的通知可能会被抑制，并且只发送根本问题的通知。

虽然 Zabbix 不支持主机之间的直接依赖关系，但是它们可以定义另外一种更加灵活的方式 - 触发器依赖关系。一个触发器可以有一个或多个依赖的触发器。

因此在我们简单示例中，我们打开服务器触发器配置的窗口，并设置它依赖于路由器的相应触发器。有了这样的依赖性，只要它所依赖的触发器处于“异常”状态，服务器触发器就不会改变状态，因此不会执行依赖的动作，也不会发送通知。

如果服务器和路由器都宕机且有依赖关系，Zabbix 将不执行依赖触发器的动作。

依赖触发器上的动作不会被执行，如果触发器依赖于:

- 状态从'PROBLEM' 修改为'UNKNOWN'
- 通过关联或者基于时间功能的手工关闭
- 被非依赖触发器的监控项值恢复
- 已禁用，已禁用监控项或禁用项目主机

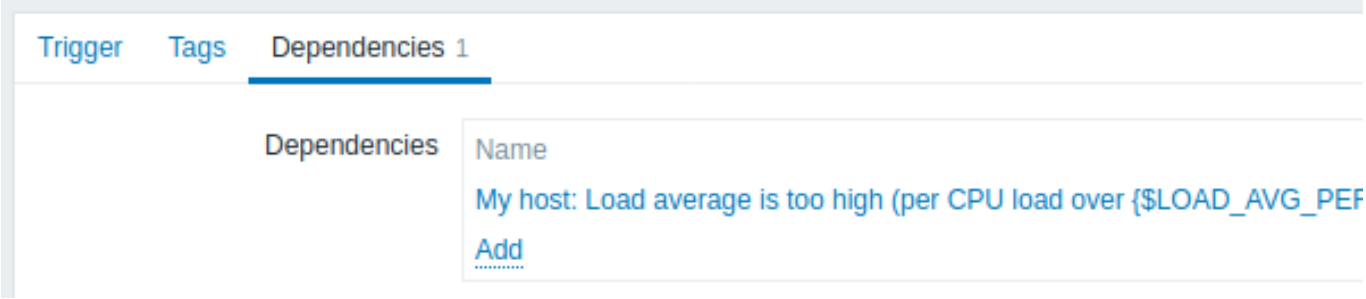
请注意，上述情况下的“次要”（依赖）触发器不会立即更新。当父触发器处于 PROBLEM 状态时，它的依赖项报告的值可能我们无法信任。因此，依赖触发器只会被重新评估，并改变它的状态，只有在父触发器处于 OK 状态并且我们收到了可信赖的指标之后。

另外:

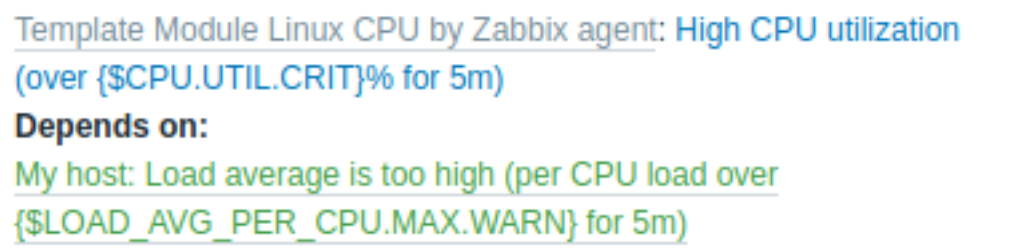
- 触发器依赖可以从任何主机触发器添加到任何其他主机触发器，只要它不会导致循环依赖。
- 触发器依赖可以从一个模板添加到另一个模板，如果模板 A 的触发器依赖于模板 B 的触发器，模板 A 只能与模板 B 一起链接到主机（或其他模板），但是模板 B 可以单独链接到主机（或其他模板）。
- 触发器依赖可以从模板触发器添加到主机触发器。在这种情况下，例如, 有一个触发器依赖于路由器（主机）触发器的模板。链接到这个模板的所有主机都将依赖于特定的路由器。
- 可以不添加从主机触发器到模板触发器的触发器依赖性。
- 触发器依赖可以从一个触发器原型添加到另一个触发器原型 (在同一个低级别自动发现规则中) 或真实触发器中。触发器原型可以不依赖来自不同 LLD 规则的触发器原型或者触发器原型中创建的一个触发器。主机触发器原型不能依赖于模板中的触发器。

配置

若要定义依赖关系，在触发器配置表格打开依赖关系标签。单击“依赖关系”块中的 添加，并选择触发器将依赖的一个或多个触发器。



点击更新。现在列表中触发器有了依赖性标示。



几个依赖关系的示例

例如，主机位于路由器 2 后面，路由器 2 在路由器 1 后面。

```
Zabbix - 路由器1 - 路由器2 - 主机
```

如果路由器 1 宕机，显然主机和路由器 2 也不可达，然而我们不想收到主机、路由器 1 和路由器 2 都宕机的 3 条通知。

因此，在这种情况下我们定义了两个依赖关系：

'主机宕机' 触发器依赖于 '路由器2宕机' 触发器
'路由器2宕机' 触发器依赖于 '路由器1宕机' 触发器

在改变' 主机宕机' 触发器的状态之前，Zabbix 将会检查相应触发器的依赖关系，如果找到，并且一个触发器处于 “异常” 状态，则触发器状态不会发生改变，因此不会执行动作，也不会发送通知。

Zabbix 递归执行此检查，如果路由器 1 或路由器 2 是不可达的状态，那么主机触发器则不会更新。

4 触发器严重性

触发器严重性定义了触发器的重要程度。Zabbix 支持以下的触发器严重性：

严重性	定义	颜色
未分类	严重程度未知	灰色
信息	仅供参考	浅蓝
警告	被警告	黄色
一般严重	一般问题	橘色
严重	发生了重要的问题	浅红色
灾难	灾难. 经济损失等	红色

严重性用于：

- 触发器的直观表示，不同的颜色代表不同的严重程度。
- 全局报警音频。不同的音频代表不同的严重程度。
- 用户媒介，不同的用户媒介（通知渠道）代表不同的严重程度。例如，SMS - 高严重性，email - 其他。
- 通过触发器严重程度的条件来限制动作。

可以自定义触发器严重性的名称和颜色.

5 自定义触发器严重性

可以在 管理 → 常规 → 触发器严重性中配置触发器严重性名称和严重性颜色相关的 GUI 主题。颜色在所有 GUI 主题之间共享。

翻译自定义严重性的名称

> **Attention:**
> 如果使用 Zabbix 前端翻译, 自定义严重性名称将会覆盖默认翻译名称。

默认触发器严重性名称可用于在所有语言环境中进行翻译。如果更改严重性名称，则会在所有区域设置中使用自定义名称，并且需要额外的手动翻译。

自定义严重性名称的翻译步骤:

- 设置自定义严重性名称，例如' 重要'
- 编辑 <frontend_dir>/locale/<required_locale>/LC_MESSAGES/frontend.po
- 添加如下 2 行:

```
msgid "Important"
msgstr "<translation string>"
```

保存文件。

- 在 <frontend_dir>/locale/README 创建.mo 文件作为描述

这里 **msgid** 应该匹配新的自定义严重性名称，**msgstr** 应该是用特定语言翻译的。

此过程应在每个严重性名称更改之后执行。

6 批量更新

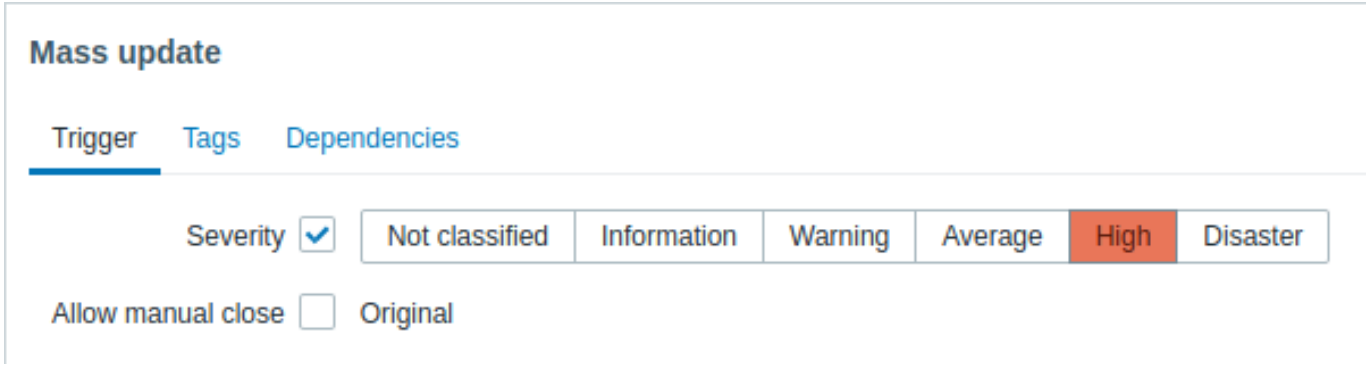
概述

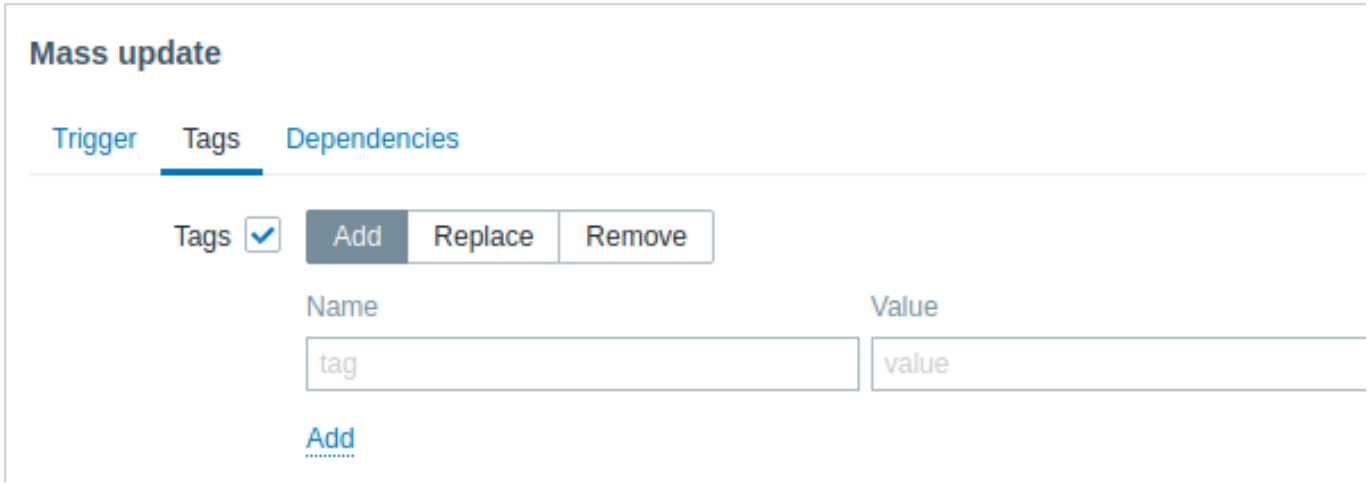
通过批量更新，您可以一次更改多个触发器的某些属性，从而无需打开每个单独的触发器进行编辑。

使用批量更新

要批量更新某些触发器，请执行以下操作：

- 在列表中标记要更新的触发器的复选框
- 点击列表下方的批量更新
- 导航到具有所需属性 (触发器, 标签 or 依赖项) 的选项卡
- 标记要更新的任何属性的复选框

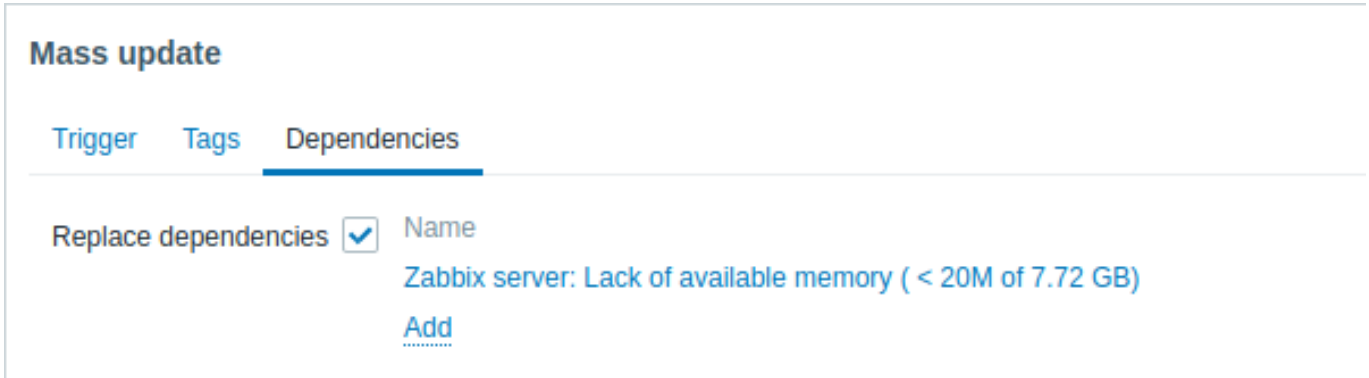




选择相应的标签更新按钮时，可以使用以下选项：

- 添加 - 允许为触发器添加新标签;
- 替换 - 将从触发器中删除任何现有标签，并用下面指定的标签替换它们；
- 删除 - 将从触发器中删除指定的标签。

请注意，具有相同名称但不同值的标签不被视为’ 重复’ ，可以添加到同一个触发器中。



替换依赖项 - 将从触发器中删除任何现有依赖项并将它们替换为指定的依赖项。

单击 更新来应用更改。

7 预测触发功能

概述

有时会有即将出现问题的迹象。可以通过这些迹象，来提前采取措施防止或至少将问题的影响降至最低。

Zabbix 有可以根据历史数据预测被监控系统未来行为的工具。这些工具是通过预测触发功能实现的。

1 功能

我们需要知道的两件事是如何定义问题状态和采取行动需要多少时间。这里有两种方法可以设置一个关于潜在不必要情况的触发信号。第一：当系统在“采取行动时间”之后预期处于问题状态时，触发器必须触发。第二：当系统将在“采取行动的时间”之内处于问题状态时，触发器必须触发。要使用的相应触发函数是 **forecast** and **timeleft**。请注意，这两个函数的基础统计分析基本相同。您可以按照自己喜欢的方式设置触发器，并获得类似的结果。

2 参数

这两个功能使用几乎相同的参数集。请参考支持的函数 列表。

2.1 时间间隔

首先，你应该指定 Zabbix 需要分析的历史时间来提出预测。你可以通过 `time period` 参数和可选时间偏移以熟悉的方式执行此操作，就像使用 **avg**, **count**, **delta**, **max**, **min** and **sum** 函数一样。

2.2 预测范围

(仅限 **forecast**)
参数 `time` Zabbix 应该在多长时间内推断它在历史数据中找到的依赖关系。不管你是否使用 `time_shift` ，`time` 总是从当前时刻开始计算。

2.3 达到阈值

(仅限 **timeleft**)
参数 `threshold` 指定分析监控项必须达到的值，不管从上还是从下达到。一旦确定了 f(t) (见下文) ，我们应该求解方程 f(t) = `threshold` 并返回更靠近现在和从现在开始向右的根，如果没有这样的根，则返回 999999999999.9999。

Note:
当监控项值接近阈值然后越过它时, **timeleft** 假定交叉点已经过去，因此切换 `threshold` 水平的, 水平的下一个交叉点（如果有）。最佳实践应该是使用预测作为普通问题诊断的补充，而不是替代。[a]

[a] 根据 声明 这些是芯片引脚上的电压，一般来说可能需要缩放。

2.4 拟合函数

默认 `fit` 是线性函数. 但是，如果你的监控系统更复杂，可以有更多选项。

`fit`	x = f(t)
线性	$x = a + b*t$
多项式[1]	$x = a_0 + a_1*t + a_2*t^2 + ... + a_n*t^n$
指数	$x = a*exp(b*t)$
对数	$x = a + b*log(t)$
幂	$x = a*t^b$

2.5 模式

(仅限 **forecast**)
每次评估触发函数时，它都会从指定的历史期间获取数据，并将指定的函数拟合到数据中。因此，如果数据略有不同，拟合函数也会略有不同。如果我们只是简单地计算未来某个特定时间拟合函数的值，那么将一无所获对于分析监控项在现在和未来那个时刻之间的预期行为。对于某些 `fit` 选项 (例如 *polynomial*) 未来的简单值可能会产生误导。

模式	**forecast** 结果
值	f(now + `time`)
最大值	$max_{now <= t <= now + time}$ f(t)
最小值	$min_{now <= t <= now + time}$ f(t)
增量	*max - min*
平均值	average of f(t) (now <= t <= now + `time`) 参考定义

3 细节

为避免计算量很大，我们将指定周期内第一个值的时间戳加上 1 ns 视为新的零时间 (当前纪元时间为 10^9, 纪元平方为 10^{18}, 双精度约为 10^{-16})。添加 1 ns 以提供涉及计算 log(t) 的对数 and 幂 _ 拟合的所有正时间值。时移不会影响 线性, 多项式, 指数 (除了更简单和更精确的计算) 但会改变对数和 幂函数的形状。

[1] `Secure` 表示 cookie 只能通过来自客户端的安全 HTTPS 连接传输。当设置为 “true” 时，只有存在安全连接时才会设置 cookie。

4 潜在错误

在如下情况，函数返回 -1:

- 指定的评估期不包含数据；
- 数学运算的结果未定义[2]；
- 数值复杂性（不幸的是，对于某些输入数据集的双精度浮点格式的范围和精度变得不够）[3]。

Note:
如果选择的拟合不能很好地描述所提供的数据，或者无法准确预测的数据太少，则不会标记警告或错误。

5 示例和错误处理

要在主机上的可用磁盘空间即将用完时收到警告，可以使用如下触发器表达式：

```
timeleft(/host/vfs.fs.size[/,free],1h,0)}<1h
```

但是，错误代码 -1 可能会发挥作用并使你的触发器处于问题状态。一般来说，你会收到警告说明你的预测无法正常工作，你应该更彻底地查看它们以找出原因。但有时它很糟糕，因为 -1 可能只是意味着没有关于在过去一小时内获得的主机可用磁盘空间的数据。如果您收到太多误报警报，请考虑使用更复杂的触发器表达式[4]：

```
timeleft(/host/vfs.fs.size[/,free],1h,0)<1h and timeleft(/host/vfs.fs.size[/,free],1h,0)<>-1
```

forecast 情况有点困难。首先，-1 可能会或可能不会将触发器置于问题状态，具体取决于您是否有表达式像 `forecast(/host/item,(...))<...` 或像 `forecast(/host/item,(...))>...`

此外，如果监控项值为负是正常的，则 -1 可能是有效的预测。但是在现实世界中这种情况的概率可以忽略不计 (看看 operator =如何工作)。因此，如果您想要或不想分别将 -1 视为问题，可以添加 `...` 或 `forecast(/host/item,(...))=-1` 或 `... and forecast(/host/item,(...))<>-1`

4 事件

概述

Zabbix 中生成的事件有以下几种类型：

- 触发器事件—无论何时一个触发器的状态发生改变（*OK*→ 问题 →*OK*）
- 服务事件—无论何时服务的状态发生改变（*OK*→ 问题 →*OK*）
- 发现事件—当检测到主机或服务时
- 自动注册事件—当主动 agent 被服务器自动注册时
- 内部事件—当一个监控项目或低级别自动发现规则变得不受支持或一个触发器进入未知状态时

Note:
从 Zabbix 2.2 开始支持内部事件。

事件是有时间戳的，可以作为动作的基础，例如发送通知电子邮件等。

要在前端查看事件的详细信息，点击监控 → 问题。在那里你可以单击事件的日期和时间查看事件的详细信息。

更多信息请查看：

- 触发器事件
- 其他事件源

1 触发器事件生成

概述

触发器状态的变化是事件最常见和最重要的来源。每次触发器的状态改变时，都会生成一个事件。该事件包含了触发器状态变更的详细信息、发生时间以及触发器的新状态。

[2] 例如，拟合 指数或 幂函数涉及计算监控项值的 log() 。如果数据包含零或负数，你将收到错误，因为 log() 仅针对正值定义。

[3] 对于线性, 指数, 对数和 幂拟合，所有必要的计算都可以明确写出来。对于多项式无需任何额外步骤即可计算仅值。计算 平均涉及计算多项式反导数（分析）。计算最大, 最小和 增量涉及计算多项式导数 (解析) 并找到它的根 (数值). 求解 f(t) = 0 涉及找到多项式根（数值）。

[4] 但在这种情况下，-1 可能会导致您的触发器从问题状态中恢复。要完全保护使用：`timeleft(/host/vfs.fs.size[/,free],1h,0)<1h and ({TRIGGER.VALUE}=0 and timeleft(/host/vfs.fs.size[/,free],1h,0)<>-1 or {TRIGGER.VALUE}=1)`

触发器会创建两种类型的事件：问题和正常。

问题事件

以下情况，会创建一个问题事件：

- 当触发器状态正常，触发器表达式评估结果为 TRUE 时；
- 当触发器启用了多重问题事件生成，每次触发器表达式评估结果为 TRUE 时。

正常事件

一个正常事件会关闭关联的问题事件，可能由以下 3 个组件生成：

- 触发器——基于 “正常事件生成” 和 “正常事件关闭”
- 事件关联
- 任务管理器——当一个事件手动关闭时

触发器

触发器有 “正常事件生成” 的设置用来控制如何生成正常事件：

- 表达式——当触发器在问题状态其表达式评估为 FALSE 时，会生成一个正常事件。这是最简单的设置，默认启用。
- 恢复表达式——当触发器在问题状态其表达式评估为 FALSE 且恢复表达式评估为 TRUE。如果触发器恢复标准与问题标准不同可以使用此设置。
- 无——不生成正常事件，可以与多个问题事件生成组合使用，当某事件发生时可以简单地发送通知。

此外，触发器有 “正常事件关闭” 的设置用来控制关闭问题事件：

- 所有问题 ——正常事件会关闭该触发器打开的所有问题事件。
- 标签值匹配的所有问题 ——正常事件会关闭该触发器打开的所有问题事件并且至少有一个匹配的标签值。此标签由 “匹配标签 “的触发器设置。如果没有要关闭的问题事件就不会生成正常事件。这通常称为触发器级别事件关联。

事件关联

事件关联（也叫做全局事件关联）是一种设置自定义事件关闭（导致正常事件生成）的规则。

这个规则定义了新的问题事件如何匹配已有的问题事件，并允许通过生成对应的正常事件来关闭新的事件或匹配的事件。

但是，配置事件关联必须非常地谨慎，因为它可能对事件处理性能造成负面影响，如果配置错误会关闭比预期更多的事件（最坏的情况会关闭所有的事件）。下面是几个配置的提示：

1. 总是通过为控制事件（匹配了旧事件的事件）设置唯一的的标签来缩小关联范围，并且使用 “新事件标签” 关联条件
2. 当使用 “关闭旧事件” 操作时不要忘记添加基于旧事件的条件，否则所有已有的问题事件都会关闭
3. 避免使用不同关联配置所使用的通用标签

任务管理器

如果触发器启用了 “允许手动关闭” 的设置，就可能手动关闭触发器生成的问题事件。这在问题升级中由前端完成。事件不是直接关闭的——相反，会生成一个 “关闭事件” 的任务，很快会由任务管理器处理。问题事件会被关闭，任务管理器会生成对应的正常事件。

2 其他事件来源

服务事件

只有在为这些事件启用了服务动作时才会生成服务事件。在这种情况下，每一个服务状态的改变都会生成一个新事件：

- 问题事件——当服务状态从 OK 变为问题时
- OK 事件——当服务状态从问题变为 OK 时

事件包含服务状态改变的详细信息——何时发生以及新状态是什么。

自动发现事件

Zabbix 定期扫描网络发现规则中定义的 IP 范围。每个规则可以单独配置检查频率。一旦发现主机或服务，就会生成发现事件（或多个事件）。

Zabbix 生成以下事件：

事件	何时生成
服务启动	每当 Zabbix 检测到活跃的服务。
服务停止	每当 Zabbix 不能检测到服务。
主机启动	如果一个 IP 至少有一个服务 UP 的。
主机宕机	如果所有服务都没有反应。
发现服务	如果服务停止后恢复或第一次发现服务。

事件	何时生成
服务丢失	如果服务启动后丢失。
发现主机	如果主机宕机后恢复或第一次发现主机。
主机丢失	如果主机启动后丢失。

主动 agent 自动发现事件

主动 agent 自动注册会在 Zabbix 生成事件。

如果配置了，当以前未知的主动 agent 请求检查或者主机的元数据改变时，主动 agent 自动注册会生成事件。Zabbix 服务器使用接收到的 agent 的 IP 和端口添加一个新的自动注册的主机。

更多信息，请查看主动 agent 自动注册页面。

内部事件

内部事件发生在以下情况：

- 一个监控项的状态从 “正常” 改变为 “不支持”
- 一个监控项的状态从 “不支持” 改变为 “正常”
- 一个底层发现规则的状态从 “正常” 改变为 “不支持”
- 一个底层发现规则的状态从 “不支持” 改变为 “正常”
- 一个触发器的状态从 “正常” 改变为 “未知”
- 一个触发器的状态从 “未知” 改变为 “正常”

自从 Zabbix 2.2 开始支持内部事件。引入内部事件的目的是允许用户在发生任何内部事件时都收到通知发生，例如，一个监控项变得不支持并且停止收集数据。

只有启用了内部动作才会生成内部事件。要停止生成内部事件（例如，当监控项变得不支持），请在 “配置”→“动作”→“内部事件” 里为内部事件禁用所有动作。

Note:
如果禁用了内部动作，当一个对象处于 “不支持” 状态时，这个对象的恢复事件仍会创建。

如果启用了内部动作，当一个对象处于 “不支持” 状态时，这个对象的恢复事件仍会创建，即使该对象没有创建过 “问题事件”。

另请参阅：接收关于不支持的监控项的通知

3 问题的手动关闭

概述

通常在触发器状态由 “问题” 变为 “OK” 时，问题事件会自动解决，但在某些情况下，可能很难确定问题是否已经通过触发器表达式得到解决。在这种情况下，问题需要手动解决。

例如，*Syslog* 可能会报告为了获得最佳性能需要调整一些内核参数。在这种情况下，故障会报告给 Linux 管理员，他们会修复故障并手动关闭。

只有触发器启用了允许手动关闭选项，问题才可以手动关闭。

当一个问题 “手动关闭” 时，Zabbix 会为 Zabbix 服务器生成一个新的内部任务。然后任务管理器进程执行这个任务，并在问题事件关闭后生成一个 OK 事件。

手动关闭一个问题并不意味着底层的触发器再也不会进入” 问题 “状态。触发表达式会重新评估并可能产生一个问题，如：

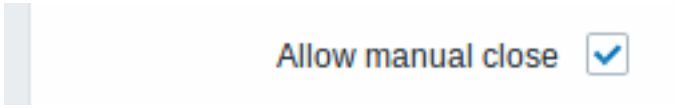
- 当任何包含触发器表达式的监控项收到新数据时（请注意，节流预处理步骤丢弃的值不被视为已接收，也不会导致重新评估触发表达式）；
- 当表达式中使用基于时间的函数时。所有基于时间的函数的列表可以在触发器页面找到。

配置

手动关闭问题需要两步。

触发器配置

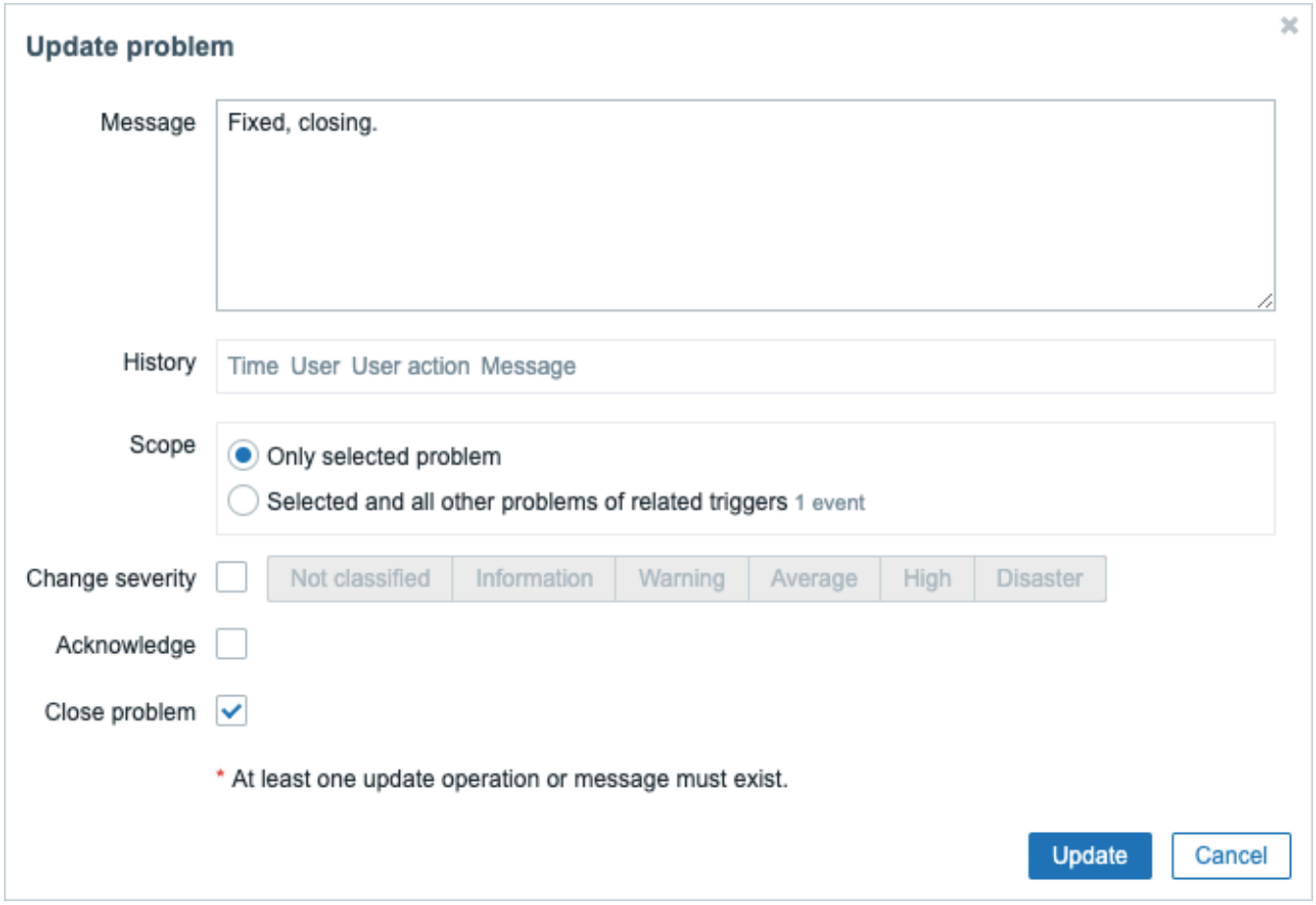
在触发器配置中，启用允许手动关闭选项。



问题更新窗口

如果触发器配置了手动关闭标识的问题出现，你可以打开这个问题的问题更新弹窗手动关闭。

要关闭问题，勾选列表中的关闭问题选项并单击更新。



所有必填输入字段都标有红色星号。

请求由 Zabbix 服务器处理。一般关闭问题只需要几秒钟。在关闭中的过程问题的状态显示在监控 → 问题里。

验证

可以通过以下方式确认一个问题已经被手动关闭了：

- 在事件详细信息里，通过监控 → 问题可以查看；
- 在通知消息里使用宏 {EVENT.UPDATE.HISTORY} 会提供该信息。

5 事件关联

概述

事件关联允许以非常精确灵活地方式关联问题事件和它们的解决方法。

事件关联可以被定义为：

- 触发器级别——不同的问题和解决方法可以关联同一个触发器
- 全局——使用全局关联规则可以通过不同的触发器轮询方法将问题和它们的解决方法关联起来

1 基于触发器的事件关联

概述

基于触发器的事件关联允许关联一个触发器产生的不同问题。

通常，在 Zabbix 中正常事件会关闭一个触发器生成的所有问题事件，但在某些情况下需要更加细致的方法。例如，当监控日志文件时，在日志文件中想要发现某些问题，并将它们单独关闭，而不是一起关闭。

当触发器配置页面的多重问题事件生成选项为启用的情况下，通常适用于日志监控、采集（trap）处理等。

可以在 Zabbix 里关联基于标签的问题事件。标签用于提取值并为问题事件创建标识。利用这一点，也可以根据匹配标签单独关闭问题。

换言之，相同的触发器可以创建由事件标签标识的的不同事件。因此，可以一个一个单独地标识问题事件，并基于事件标签地标识单独关闭。

工作原理

在日志监控中，可能会遇到下面类似地输出：

```
Line1: 应用1停止

Line2: 应用2停止

Line3: 应用1重启

Line4: 应用2重启
```

事件关联地想法是将从 “Line1” 的问题事件到 “Line3” 的恢复事件，从 “Line2” 的问题事件到 “Line4” 的恢复事件相匹配，并能逐个关闭这些问题：

```
Line1: 应用1停止

Line3: 应用1重启   #问题来自于Line1关闭

Line2: 应用2停止

Line4: 应用2重启   #问题来自于Line2关闭
```

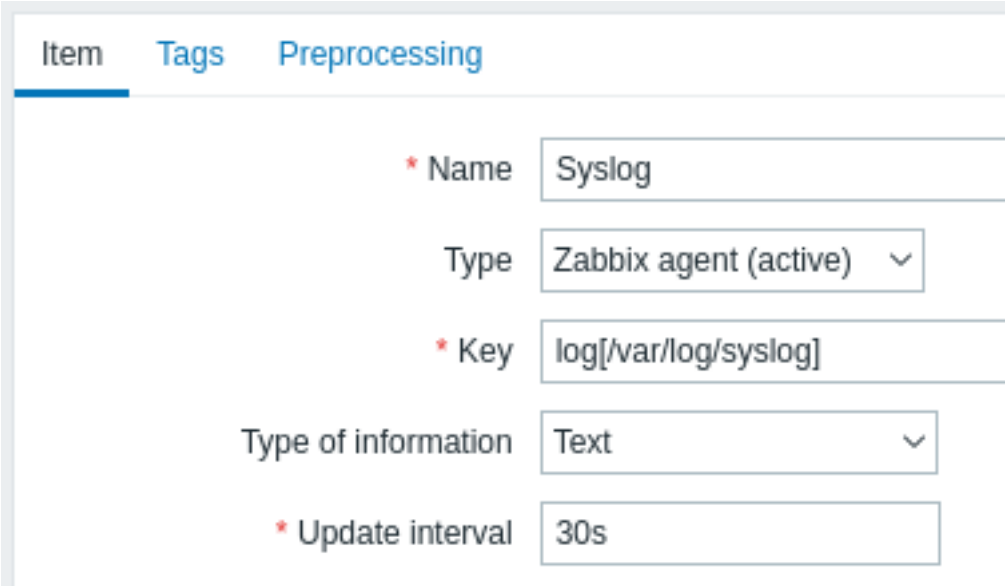
为此，需要通过标签将这些事件相关联，例如，可以标识为 “应用 1” 和 “应用 2”。这个过程也可以将正则表达式应用于日志中来提取标签的值。然后，当事件创建时，他们分别给标识为 “应用 1” 和 “应用 2”，并且问题可以与解决方法相匹配。

配置

监控项

首先，你可能想设置一个监控日志文件的监控项，例如：

```
log[/var/log/syslog]
```



设置监控项后，等待一分钟，等待配置更改，然后去最新数据 确认该监控项已经开始收集数据。

触发器

要让监控项工作，你需要配置触发器。重要的是要决定日志文件中哪些条目值得注意。例如，以下触发表达式将搜索像 “Stopping” 这样的字符串来发现潜在问题的信号：

```
find(/My host/log[/var/log/syslog],,"regexp","Stopping")=1
```

Attention:
为了确保包含字符串 “Stopping” 的每一行都被视为问题，还要在触发器配置中将问题事件生成模式设置为 “多种”。

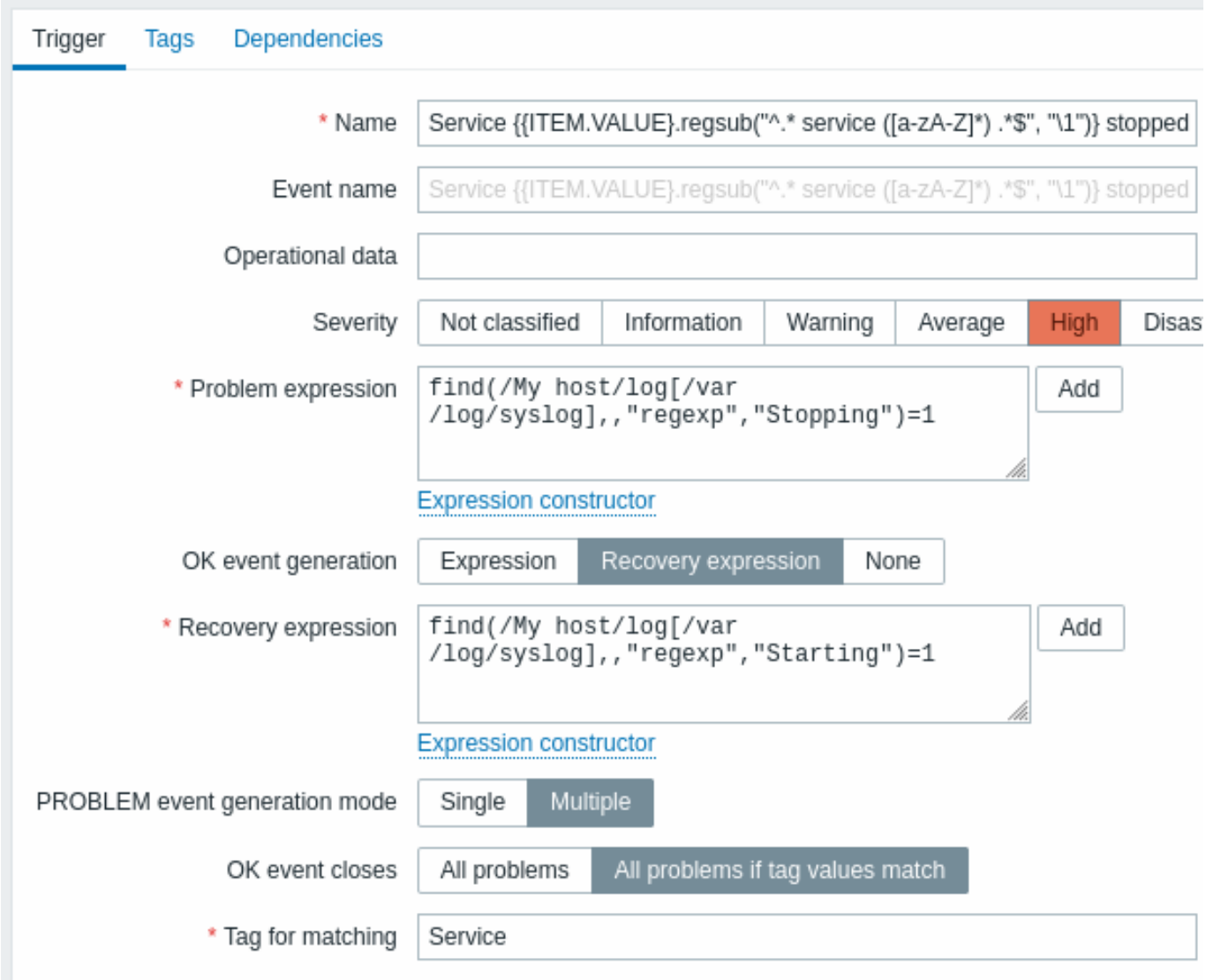
然后定义一个恢复表达式。如果发现包含字符串 “starting” 的日志行，以下恢复表达式将解决所有问题：

```
find(/My host/log[/var/log/syslog],,"regexp","Starting")=1
```

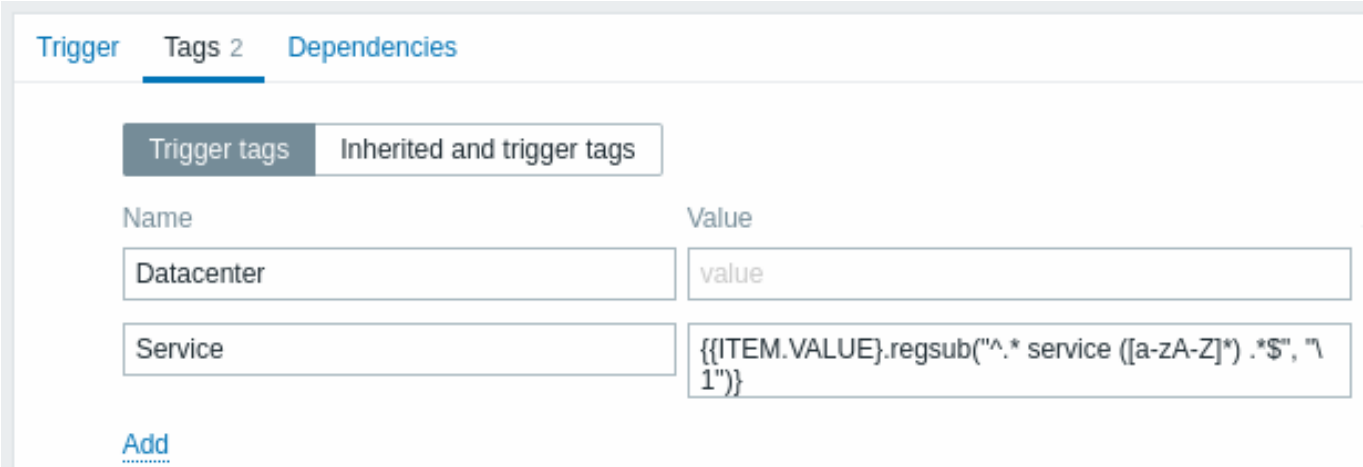
由于以某种方式确保相应的根因问题得到解决很重要，我们不希望仅仅是解决所有问题。这就是标签可以提供帮助的地方。

问题和解决方案可以通过在触发器配置中指定标签来匹配。必须进行以下设置：

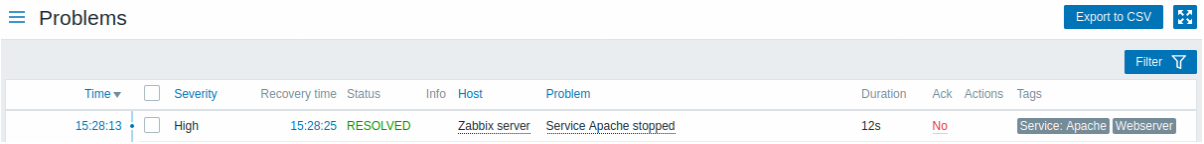
- 问题事件生成模式：多种
- 正常事件关闭：如果标签匹配所有事件
- 输入事件匹配的标签名称



- 配置标签从日志行中获取标签值



如果配置成功，你将在监控 → 问题里看到问题事件被应用程序标识并匹配它们的解决方案



Warning:
因为可能配置错误，会为不关联的问题创建相似的事件标签，请阅读下面列出的案例！

- 两个应用程序向同一个日志文件写入错误和恢复消息时，用户可能决定为同一个触发器使用两个标签值不同的应用标签，在标签值中使用不同的正则表达式从 {ITEM.VALUE} 宏获取应用名称 A 和 B（例如，当消息格式不同时）。然而，如果没有与正则表达式匹配，这可能无法按计划工作。不匹配的正则表达式会产生空的标签值，同一个空标签值足以将问题事件和正常事件关联起来。所以，一个来自应用 A 的恢复消息会意外地关闭一个来自应用 B 的错误消息。
- 实际标签和标签值仅在触发器触发时可见。如果正则表达式使用不正确，会不做提示的自动替换为一个 *UNKNOWN* 字符串。如果最初的有 *UNKNOWN* 标签值的问题事件被错过了，随后可能会出现一个有相同 *UNKNOWN* 标签值的正常事件来关闭本不该关闭的问题事件。
- 如果用户使用 {ITEM.VALUE} 宏而没有将宏函数设为标签值，255 个字符的限制会生效。当日志消息很长并且前 255 个字符不是特定的，这也可能导致非关联问题拥有相似的事件标签。

2 全局事件关联

概述

全局事件关联允许覆盖 Zabbix 监控的所有指标并创建关联性。

可以关联由完全不同的触发器创建的事件，并对它们应用相同的操作。通过创建智能关联规则，实际上可以避免数以千计的重复通知，并专注于问题的根本原因！

全局事件关联是一种强大的机制，它可以让您从基于单个触发的问题和解决逻辑中解开自己。到目前为止，单个问题事件是由一个触发器创建的，我们依赖于相同的问题解决触发器。我们无法用另一个触发器解决一个触发器创建的问题。但是基于事件标记的事件关联，我们可以。

例如，日志触发器可以报告应用程序问题，而轮询触发器可以报告应用程序启动并运行。利用事件标记，您可以将日志触发器标记为状态：*Down*，而将轮询触发器标记为状态：*Up*。然后，在全局关联规则中，您可以关联这些触发器并为此关联分配适当的操作，例如关闭旧事件。

在另一种用途中，全局关联可以识别类似的触发器并对它们应用相同的操作。如果我们每个网络端口问题只能获得一个问题报告怎么办？无需全部报告。通过全局事件关联也是能够实现的。

全局事件关联在关联规则中配置。关联规则定义新问题事件如何与现有问题事件配对以及在匹配情况下要执行的操作（关闭新事件，通过生成相应的 OK 事件来关闭匹配的旧事件）。如果问题被全局关联关闭，则会在监控 - > 问题的消息列中报告。

配置全局关联规则仅适用于 Zabbix 超级管理员级别用户。

必须非常仔细地配置事件关联，因为它会对事件处理性能产生负面影响，或者如果配置错误，会关闭比预期更多的事件（在最坏的情况下，甚至可以关闭所有问题事件）。:::

要安全地配置全局关联，请遵循以下重要提示：

- 减少相关范围。始终为与旧事件配对的新事件设置唯一标记，并使用新事件标记关联条件;

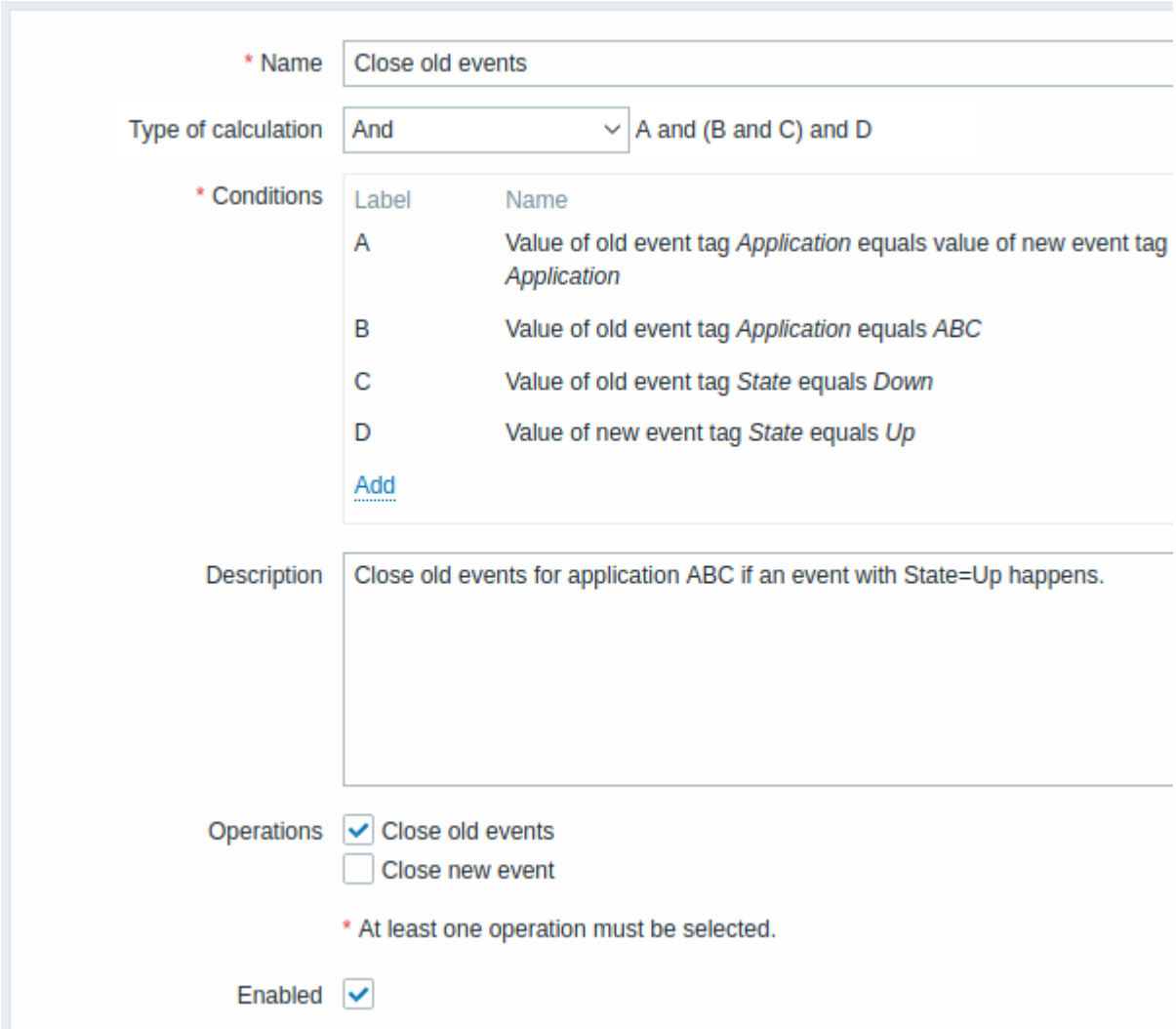
- 使用关闭旧事件操作时，根据旧事件添加条件（或者可以关闭所有现有问题）；
- 避免使用可能最终被不同关联配置使用的常见标记名称;
- 保持关联规则的数量仅限于您真正需要的数量。

可参考: known issues.

配置

要全局配置事件关联规则：

- 去 配置 → 关联事件
- 在右侧点击新建关联（或点击关联名称编辑已有规则）
- 在表单中输入关联规则的参数

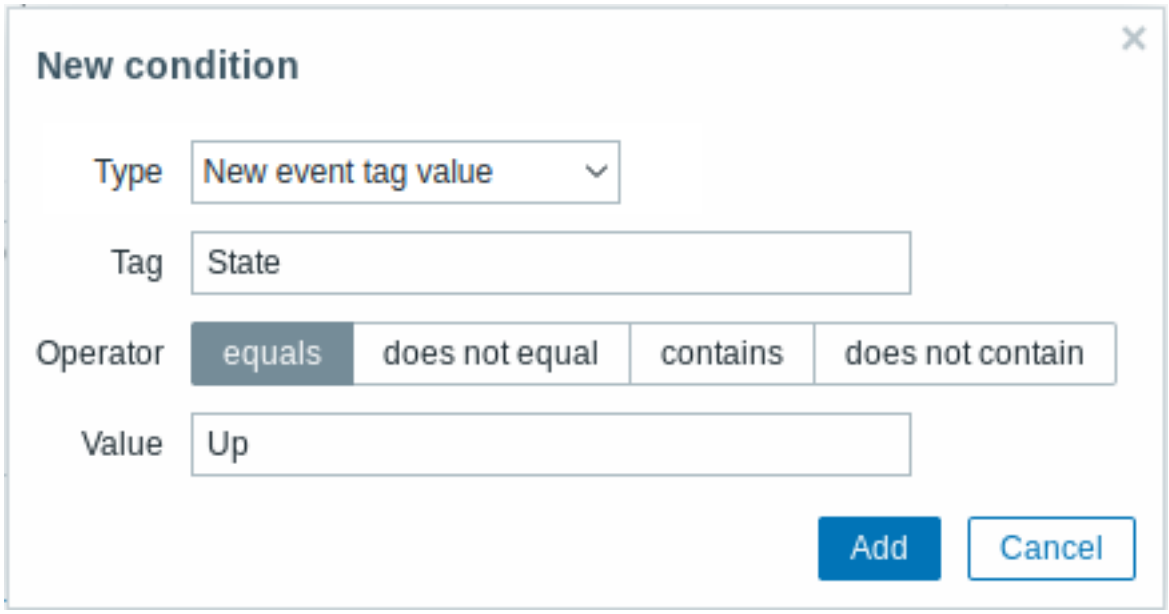


所有必填输入字段都标有红色星号。

参数	描述
名称	唯一的关联规则名称。
计算类型	有以下计算条件的选项可供选择： **And** - 必须满足所有条件 **Or** - 如果满足一个条件就足够了 **And/Or** - AND 具有不同的条件类型和 OR 具有同一条件类型 自定义表达式 - 用于评估动作条件的用户定义的计算公式。它必须包括所有条件（以大写字母 A、B、C、... 表示），并可能包括空格、制表符、括号 ()、**and**（区分大小写）、**or**（区分大小写）、**not**（区分大小写）。
条件	条件列表。有关配置条件的详细信息，请参阅下文。
描述	关联规则的描述。
操作	在事件关联时标记要执行的操作的复选框。可以使用以下操作： 关闭旧事件 - 发生新事件时关闭旧事件。使用关闭旧事件操作时，始终根据旧事件添加一个条件，否则现有的所有得问题都会关闭。 关闭新事件 - 新事件发生时关闭它。

参数	描述
已启用	如果您标记此复选框，将启用关联规则。

要配置新条件的详细信息，请在条件模块里单击 Add 。将打开一个弹出窗口，您可以在其中编辑条件细节。



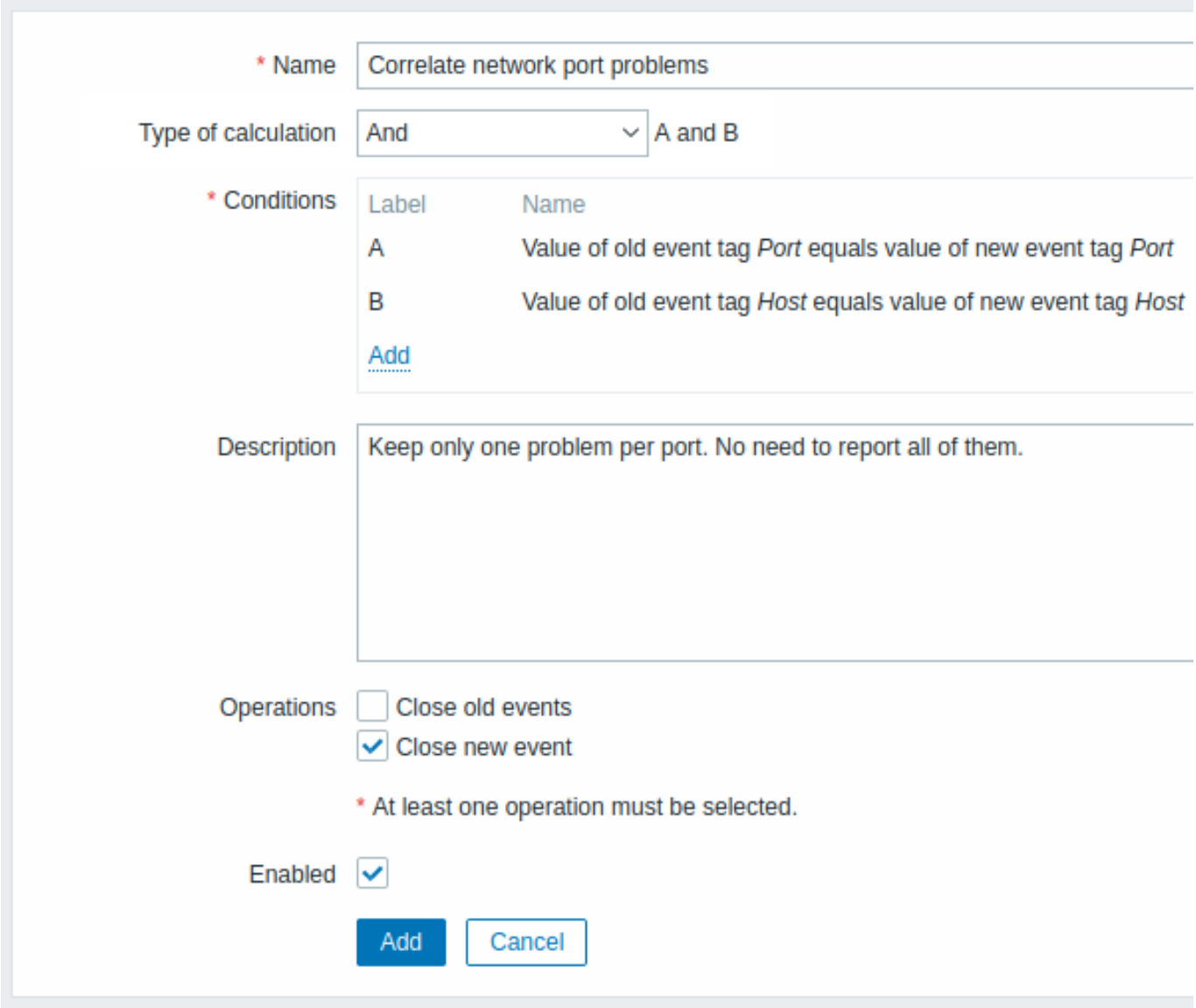
参数	描述
新条件	选择关联事件的条件。 注意如果没有指定旧事件的条件，所有旧事件都可以被匹配和关闭。同样，如果没有指定新事件的条件，所有新事件都可以被匹配和关闭。 有以下条件选项可供选择： 旧事件标签 - 指定用于匹配旧事件的标签。 新事件标签 - 指定用于匹配新事件的标签。 新事件主机组 - 指定要匹配新事件的主机组。 事件标签对 - 指定新事件的标签和旧事件的标签以进行匹配。在这种情况下，如果两个事件的标签的值匹配，事件也将匹配。标签名称不需要匹配。 此选项可用于匹配运行时值，该值在配置时可能不知道（另见示例 1）。 旧事件标签值 - 指定用于匹配的旧事件标签名称和值，使用以下运算符： 等于 - 有旧事件标签值 不等于 - 没有旧事件标签值 包含 - 有旧事件标签值中的字符串 不包含 - 没有旧事件标签值中的字符串 新事件标签标签值 - 使用以下运算符指定用于匹配的新事件标签名称和值： 等于 - 有新事件的标签值 不等于 - 没有新事件的标签值 包含 - 有新事件标签值中的字符串 不包含 - 没有新事件标签值中的字符串

Warning:
由于可能存在配置错误，当可能为不相关问题创建类似事件标签时，请查看下面列出的案例！

- ·实际标签和标签值仅在触发器触发时才可见。如果使用的正则表达式无效，它会自动替换为 *UNKNOWN* 字符串。如果带有 *UNKNOWN* 标记值的初始问题事件被遗漏，可能会出现具有相同 *UNKNOWN* 标记值的后续正常事件，这些事件可能会关闭它们不应该关闭的问题事件。
- 如果用户使用 {ITEM.VALUE} 宏，而不使用宏函数作为标签值，则适用 255 个字符的限制。日志信息时很长，且前 255 个字符是非特定的，这可能还会导致无关问题有类似的事件标签。

示例 1

停止来自同一网络端口的重复问题事件。



如果触发器上存在主机和端口的标签值与最初的事件的相同，这个全局关联规则会关联问题。

这个操作将会关闭同一网络端口的新的问题事件，只保留最初的问题事件。

6 标记

概览

Zabbix 中有一个选项可以标记各种实体。标签可以定义为：

- 模板
- 主机
- 监控项
- 网络场景
- 触发器
- 模板项目和触发器
- 主机、监控项和触发器原型

标签有多种用途，最明显的是标记事件。如果实体被标记，相应的新事件也会被标记：

- 带有标记的模板 - 所有由此模板相关实例 (监控项、触发器等) 创建的主机问题都会被标记。
- 带有标记的主机 - 主机的所有问题都将被标记
- 带有标记的项目、网络场景 - 该监控项的所有数据/问题或 web 场景将被标记
- 带有标记的触发器 - 此触发器的所有问题都将被标记

一个问题事件继承了整个模板链中的所有标签，主机、监控项、Web 场景、触发器。当标记事件时，完全相同的 `tag:value` 组合（在解析宏之后）合并为一个而不是被复制。

拥有自定义事件标签可以提供更大的灵活性。特别是事件可以基于相关事件标签。在其他用途中，可以根据标记定义操作事件。监控项问题可以根据标签进行分组。

标记实现为一对 *tag name* 和 *value*。您可以仅使用名称或将其与值配对：

MySQL, Service:MySQL, Services, Services:Customer, Applications, Application:Java, Priority:High

一个实体（模板、主机、监控项、Web 场景、触发器或事件）可能是使用相同名称但值不同的标签 - 这些标签不会被认为是“重复”。同样，一个没有值的标签和同一个标签值可以同时使用。

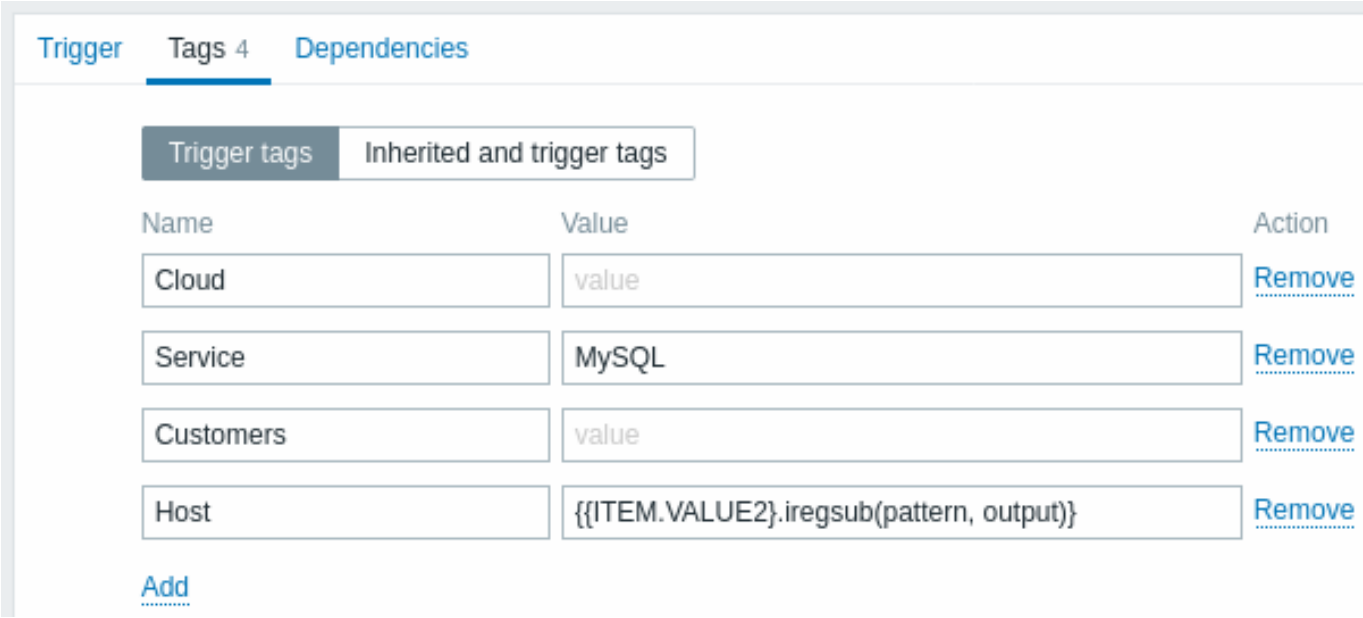
用例

此功能的一些用例如下：

1. 在前端标记触发事件
    - 在触发器级别定义标签;
    - 在监控 → 问题中查看所有触发问题是如何用这些标签标记的.
2. 标记所有模板继承的问题
    - 在模板级别定义标签，例如’App=MySQL’;
    - 查在监控 →* 问题 * 中，可以看到由该模板中的触发器创建的那些主机问题是如何被标记为这些标记的。
3. 标记所有主机问题
    - 在主机级别定义标签，例如’Service=JIRA’;
    - 在监控中的标签 → 问题中查看主机触发器的所有问题是如何用这些标记的。
4. 相关监控项组
    - 在监控项级别定义标签，例如’MySQL’;
    - 使用 最新数据的监控项标签过滤器查看所有标记为’MySQL’。
5. 识别日志文件中的问题并分别关闭
    - 在日志触发器中定义标签，这些标签将通过提取宏 {{ITEM.VALUE<N>}.regsub()} 的值来标记事件;
    - 在触发器配置中，使用多个问题事件生成模式;
    - 在触发器配置中，使用event 相关性: 选择 OK 事件仅关闭匹配事件并选择匹配标签;
    - 查看使用标签创建并单独关闭的问题事件。
6. 用来过滤通知
    - 在触发级别上定义标签，以便通过不同的标签标记事件;
    - 在动作条件中使用标签过滤，仅接收与标签数据匹配的事件通知。
7. 使用从监控项值中提取的信息作为标签值 - 在标签值中使用 `{{ITEM.VALUE<N>}.regsub()}` 宏; - 在 监控 → 问题中查看从监控项值中提取的数据作为标签值。
8. 在通知中更好地识别问题
    - 在触发级别定义标签;
    - 在问题通知中使用 {EVENT.TAGS} 宏;
    - 更容易识别通知所属的应用程序/服务。
9. 使用模板级别的标签简化配置任务
    - 在模板触发级别定义标签;
    - 查看从模板触发器创建的所有触发器上的这些标记.
10. 使用来自低级别发现 (LLD) 的标签创建触发器
    - 在触发器原型上定义标签;
    - 在标签名称或值中使用 LLD 宏;
    - 查看从触发器原型创建的所有触发器上的这些标签。

配置

可以在专用选项卡中输入标签，例如在触发器配置：



宏支持

以下宏可用于触发标记：

- {ITEM.VALUE}、{ITEM.LASTVALUE}、{HOST.HOST}、{HOST.NAME}、{HOST.CONN}、{HOST.DNS}、{HOST.IP}、{HOST.PORT} 和 {HOST.ID} 宏可用于填充标签名称或标签值
- {INVENTORY.*}宏 可以在主机触发器表达式中引用一个或多个主机资产值
- 标签名称/值支持用户宏（/manual/config/macros/user_macros）和用户宏上下文。用户宏上下文可能包括低级发现宏
- 在触发器原型中，可以使用低级别的发现宏来发现标记名/值

以下宏可用于基于触发器的通知：

- {EVENT.TAGS} 和 {EVENT.RECOVERY.TAGS} 宏将解析为逗号分隔的事件标签或恢复事件标签列表
- {EVENT.TAGSJSON} 和 {EVENT.RECOVERY.TAGSJSON} 宏将解析到包含事件标签的 JSON 数组对象 或恢复事件标记对象

以下宏可用于模板、主机、监控项和网页场景标签：

- {HOST.HOST}、{HOST.NAME}、{HOST.CONN}、{HOST.DNS}、{HOST.IP}、{HOST.PORT} 和 {HOST.ID} 宏
- {INVENTORY.*} 宏
- 用户宏
- 低级发现宏可以在监控项原型标签中使用

以下宏可用于主机原型标签：

- {HOST.HOST}、{HOST.NAME}、{HOST.CONN}、{HOST.DNS}、{HOST.IP}、{HOST.PORT} 和 {HOST.ID} 宏
- {INVENTORY.*} 宏
- 用户宏
- 在发现过程中将解析低级发现宏 ，然后添加到发现主机

触发器标签中提取子字符串

支持提取子字符串以填充标签名称或标签值，使用宏function -将正则表达式应用于 {ITEM.VALUE}，{ITEM.LASTVALUE} 宏或低级发现宏所获得的值。例如：

```
{{ITEM.VALUE}.regsub（pattern，output）}
{{ITEM.VALUE}.iregsub（pattern，output）}

{{#LLDMACRO}.regsub（pattern，output）}
{{#LLDMACRO}.iregsub（pattern，output）}
```

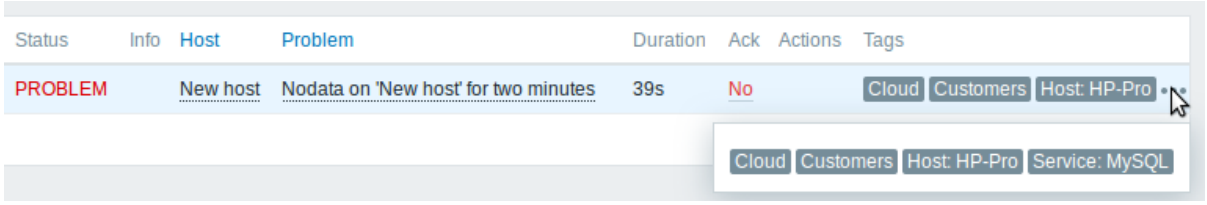
如果标签名和标签值的宏被解析后长度超过 255 个字符，标记名和值将被切割为 255 个字符。

另请参阅：在低级发现宏使用宏函数来实现事件标记。

查看事件标签

标记（如果已定义）可以在以下新事件中看到：

- 监控 → 问题
- 监控 → 问题 → 事件详情
- 监控 → 仪表板 → 问题小部件（在弹出窗口中将鼠标悬停在问题名称上时打开 |



仅显示前三个标签对。如果有超过三个标签对，由三个点表示。如果你把鼠标移动到这三个点上，所有标签对都会显示在弹出窗口中。

请注意，在监控 → 问题或问题仪表板小部件中，标签的显示顺序受标签过滤器和过滤器中的标签显示优先级影响。

7 可视化

1 图表

概览

随着大量数据流入 Zabbix，如果可以以图形化查看正在发生的事情，而不仅是数字，那么对于用户来说，这将变得更加容易。。

这就是图表的用武之地。图表可以在一定程度上掌握数据流一目了然，关联问题，发现某事何时开始或者展示某事何时可能变成问题。

Zabbix 为用户提供：

- 监控项内置的简单图形
- 创建复杂自定义图表的能力
- 在ad-hoc 图表中快速访问多个项目的比较
- 可定制化的矢量图表

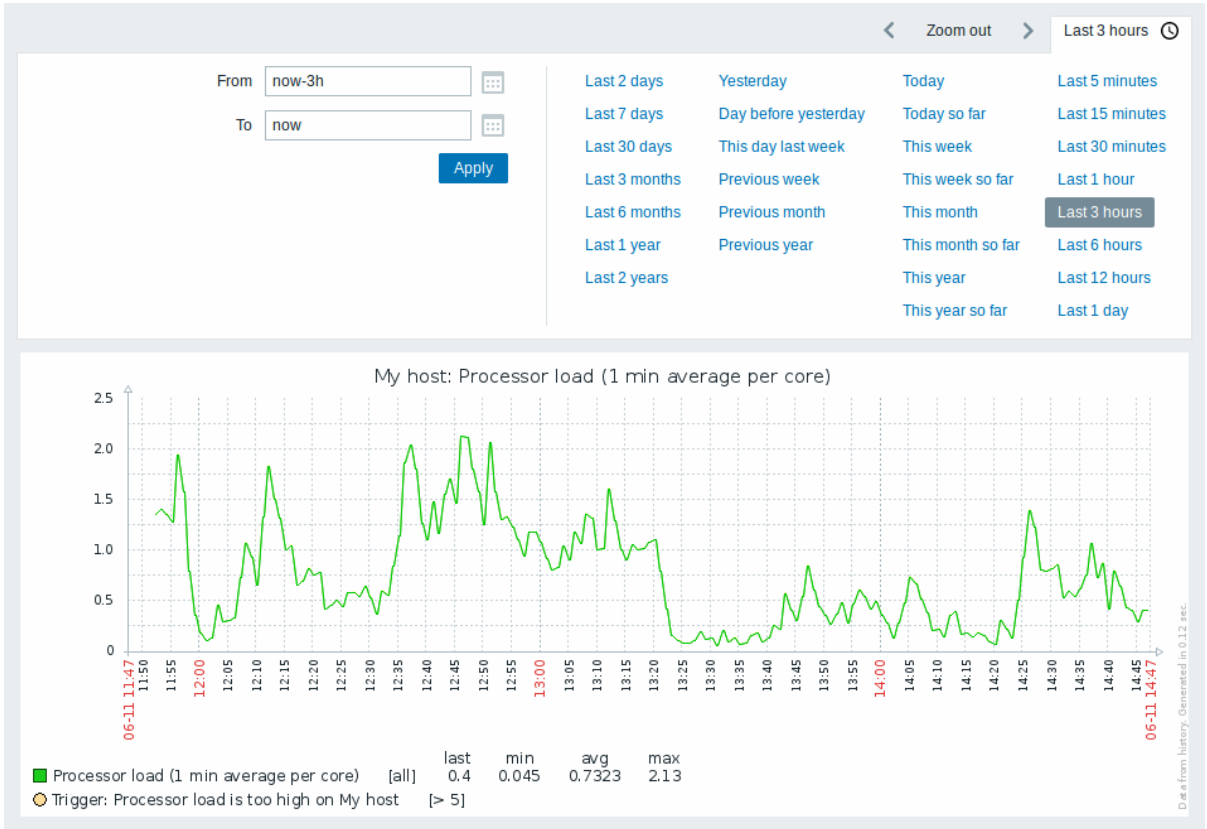
1 简单的图表

概览

提供了简单的图表，用于可视化监控项收集的数据。

用户部分无需配置工作即可查看简单图表。它们由 Zabbix 免费提供。

只需转到 监控 → 最新数据并单击图表链接将显示相应的监控项和图表。



Note:
为所有数值型监控项提供了简单的图表。为了文本监控项，历史链接在 监控 → 最新数据中是可用的。

时间段选择器

请注意图表上方的时间段选择器。它允许一键选择经常需要的时间段。

请注意，今天、本周、本月、本年显示整个期间，包括未来的小时/天。今天到目前为止，相比之下，只显示过去的小时数。

一旦选择了一个周期，它可以在时间上来回移动点击  箭头按钮。缩小按钮允许缩小第二个周期次或在每个方向上增加 50%。缩小也可以通过双击图表。整个时间段选择器可以是通过单击包含所选期间的选项卡标签折叠。

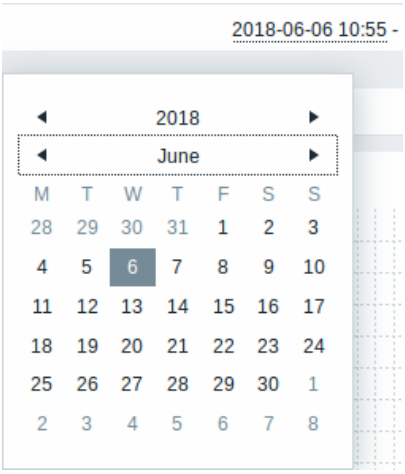
From/To 字段以以下任一方式显示所选期间：

- 格式为 “Y-m-d H:i:s” 的绝对时间语法
- 相对时间语法，例如：`now-1d`

相对格式的日期可以包含一个或多个数学操作（- 或 +），例如 `now-1d` 或 `now-1d-2h+5m`。对于相对时间支持以下缩写：

- 现在
- s（秒）
- 米（分钟）
- h（小时）
- d（天）
- w（周）
- M（月）
- y（年）

可以通过单击选择特定的开始/结束日期 *From/To* 字段旁边的日历图标。在这种情况下，日期选择器弹出窗口将打开。



在日期选择器中，可以在使用 Tab 和 Shift+Tab 的年/月/日。键盘箭头或箭头按钮允许选择所需的值。按 Enter 键（或单击所需的值）激活选择。

控制显示时间的另一种方法是突出显示用鼠标左键绘制图形。图表将放大到松开鼠标左键后突出显示的区域。

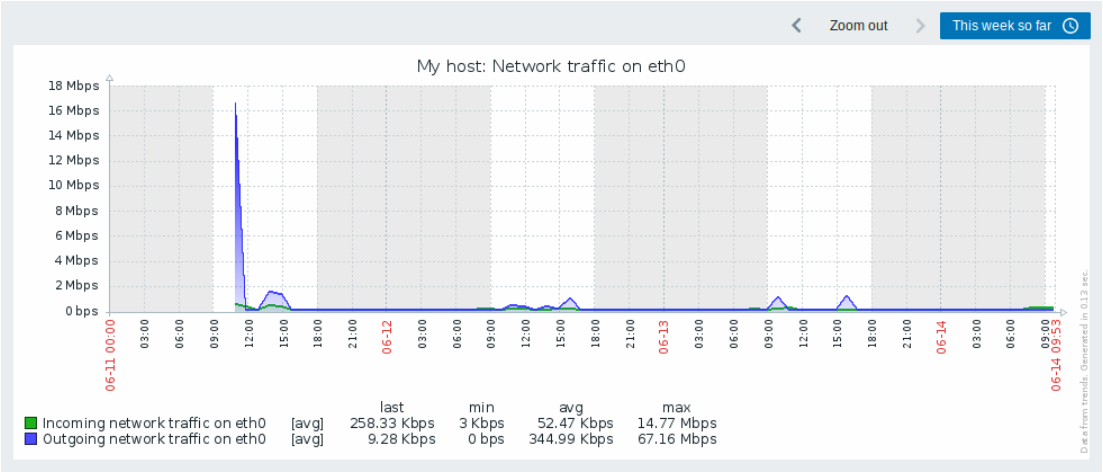
如果未指定时间值或字段留空，则时间值将设置为 “00:00:00”。这不适用于今天的日期选择：在这种情况下，时间将设置为当前值。

最近的数据与更长的时期

对于最近的数据，会绘制一条单线连接每个接收到的数据价值。只要至少有一条线，就画一条线水平像素可用于一个值。

对于显示较长周期的数据绘制三条线 - 深色绿色表示平均值，而浅粉色和浅绿色线显示该时间点的最大值和最小值。空间高点和低点之间用黄色背景填充。

工作时间（工作日）在图表中显示为白色背景，而非工作时间以灰色显示（默认前端主题使用原始蓝色）。

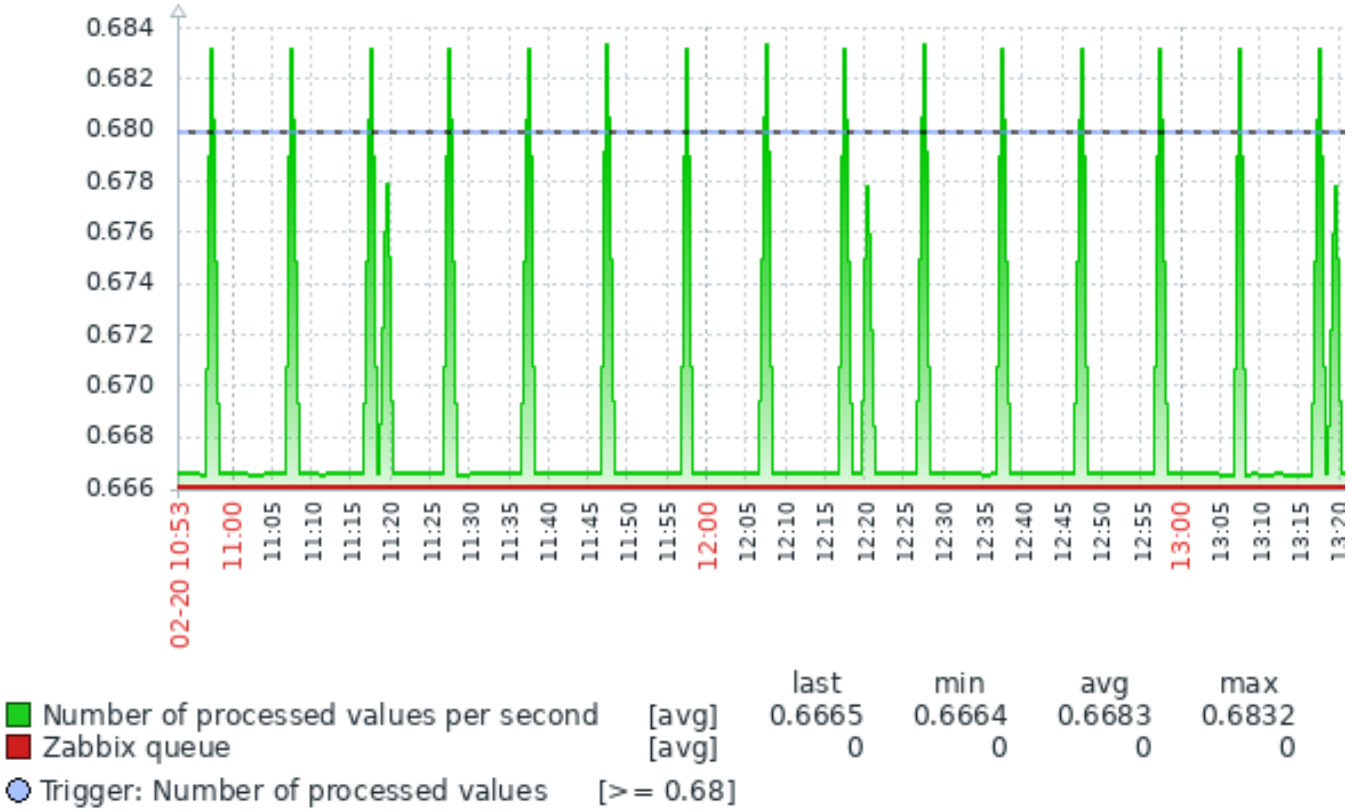


在简单的图表中工作时间总会显示，而在自定义图表 中可以按照用户偏好显示。

如果图表显示超过 3 个月，则不显示工作时间。

触发器线

单一触发器显示成一条带有触发器严重性颜色的黑色虚线——注意图表上的蓝线和图例中显示的触发信息。最多 3 条触发器线可以显示在图表上；如果有更多触发器，那么严重性较低的触发器优先展示。触发器总是会显示在单一图表中，而在自定义图表 中可按照用户偏好显示。



从历史/趋势生成

可以根据监控项历史或趋势生成图表。

对于前端调试模式 已激活的用户，在图表的右下角显示一个灰色的垂直标题，指示数据来源。

影响趋势历史使用的几个因素：

- 监控项历史的寿命。例如，项目历史可以保留 14 天。在这种情况下，任何超过十四天的数据都将不被保存。
- 图表中的数据拥塞。如果要显示的秒数水平图形像素超过 3600/16，显示趋势数据（即使监控项历史在同一时期仍然可用）。
- 如果趋势被禁用，监控项历史将用于图表构建（如果可用于该时期）。从 Zabbix 2.2.1 开始支持此功能（之前，禁用趋势意味着期间的图表是空的，即使监控项历史可用）。

数据缺失

对于具有定期更新间隔的监控项，在如果未收集项目数据，则图表展示为空。

但是，对于 trapper 监控项和具有计划更新间隔的项（但是定期更新间隔设置为 0），在第一个收集的值和最后一个收集的值之间是一条直线；这条线分别位于第一个/最后一个值的级别上。

切换到原始值

右上角的下拉菜单允许从简单图形切换到值*/500* 个最新值列表。这对于查看组成图形的数值很有用。

这里表示的是原始值，即没有单位或使用处理后的值。但是，会显示应用了值映射。

已知的问题

有关图表，请参阅已知问题。

2 自定义图表

概述

顾名思义，自定义图表提供自定义功能。

虽然简单的图表有利于查看单个项目的数据，但它们不支持配置功能。

因此，如果您想更改图形样式或线条的显示方式或比较几个监控项，例如，单个图表展示接收和转发的流量，您需要一个自定义图表。

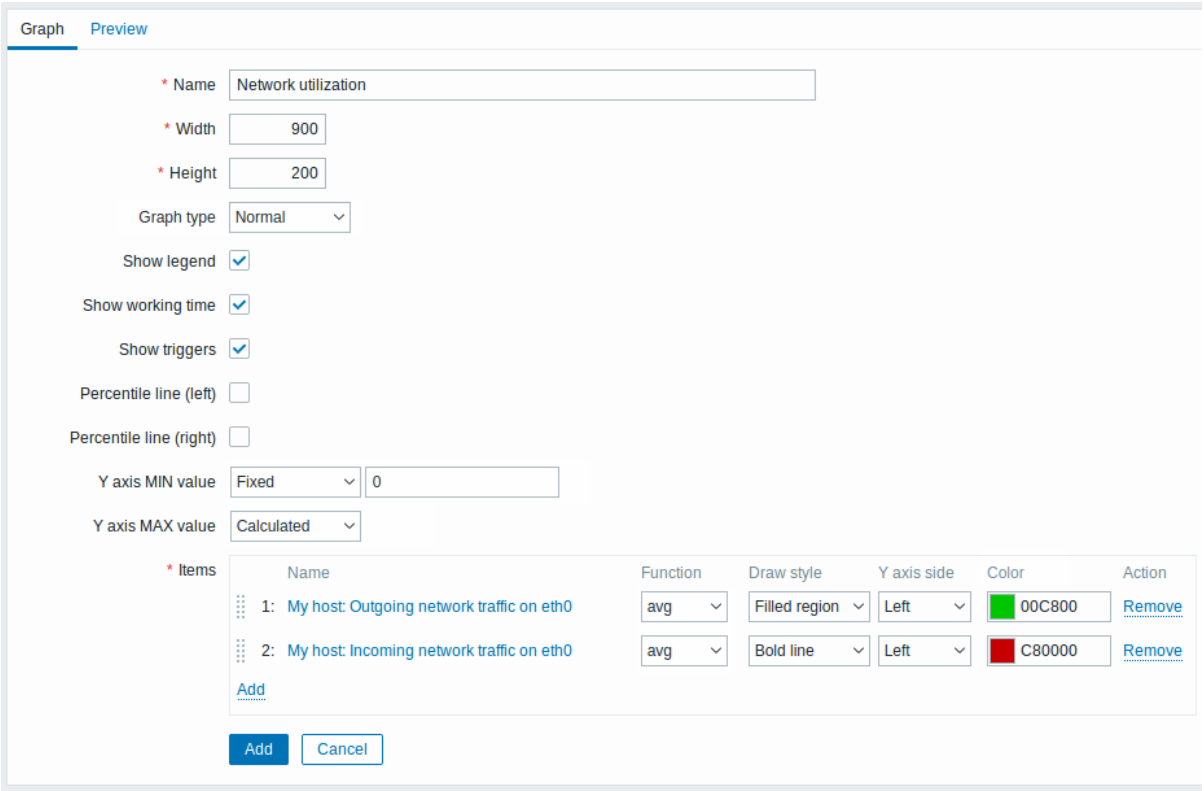
自定义图表是手动配置的。

它们可以为一台主机或多台主机或单个主机创建模板。

配置自定义图表

要创建自定义图表，请执行以下操作：

- 转到配置 → 主机（或模板）
- 单击所需主机或模板旁边行中的 图表
- 在图表屏幕中点击创建图表
- 编辑图表属性



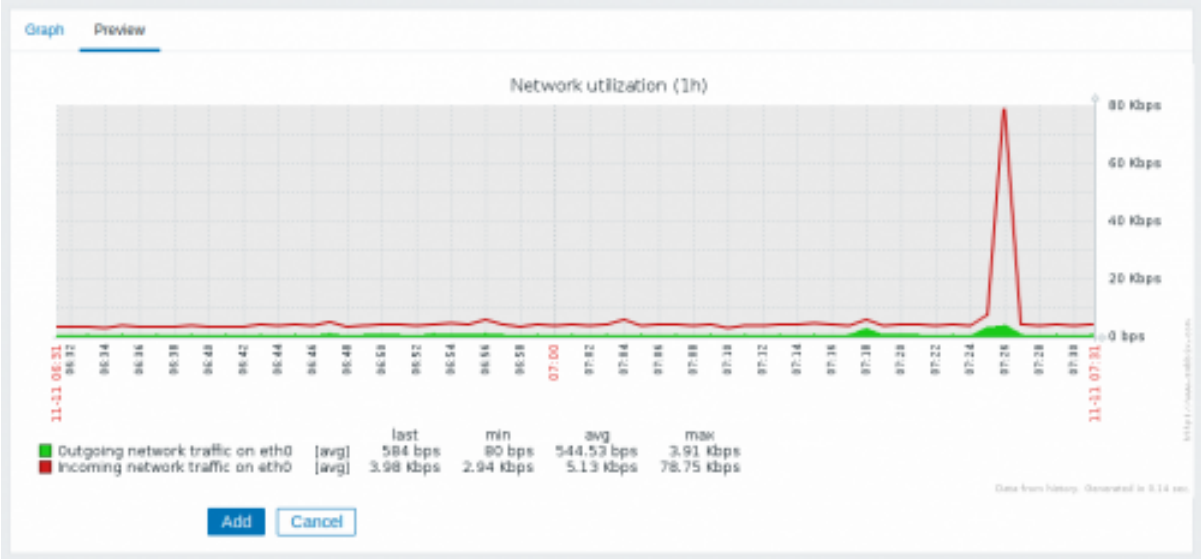
所有必填字段都标有红色星号。

图表属性：

参数		说明
名称		唯一的图形名称。 此字段支持表达式宏，但仅限于 avg、last、min 和 max 函数，以时间为参数（例如，{?avg(/host/key,1h)}）。 支持在此宏中使用 {HOST.HOST<1-9>} 宏，引用图中的第一个、第二个、第三个等主机，例如 {?avg(/{HOST.HOST2}/key,1h)}。 请注意，使用此宏引用第一个主机是多余的，因为可以隐式引用第一个主机，例如 {?avg(//key,1h)}。
宽度		以像素为单位的图形宽度（仅用于预览和饼图/分解图）。
高度		以像素为单位的图形高度。
图表类型		图表类型： 正常 - 正常图表，值显示为线条 堆叠 - 堆叠图表，显示填充区域 饼图 - 饼图 展开 - “展开” 饼图，部分显示为饼图的 “切出”
显示图例		选中此框将设置为显示图例。
显示工作时间		如果选择，非工作时间将以灰色背景显示。不适用于饼图和分解饼图。
显示触发器		如果选中，简单触发器将显示为带有黑色短划线的线条，上面有触发器严重性颜色。不适用于饼图和分解饼图。
百分位线（左）		显示左 Y 轴的百分位。例如，如果设置了 95% 的百分位，则百分位线将位于 95% 的值所在的水平。显示为亮绿色线。仅适用于普通图。
百分位线（右）		显示右侧 Y 轴的百分位。例如，如果设置了 95% 的百分位，则百分位线将位于 95% 的值所在的水平。显示为一条鲜红的线。仅适用于普通图。
Y 轴最小值		Y 轴最小值： 计算 - Y 轴最小值将自动计算 固定 - Y 轴固定最小值. 不适用于饼图和分解饼图。 监控项 - 所选项目的最后一个值将是最小值
Y 轴最大值		Y 轴最大值： 计算 - Y 轴最大值将自动计算 固定 - Y 轴最大值固定. 不适用于饼图和分解饼图。 监控项 - 所选项目的最后一个值将是最大值
3D 视图		启用 3D 样式。仅适用于饼图和分解饼图。
监控项		要在此图中显示的数据的监控项。单击 增加以选择项目。您还可以选择各种显示选项（功能、绘制样式、左/右轴显示、颜色）。
	排序顺序 (*0→100*)	绘制顺序。0 将首先被处理。可用于在另一个后面（或前面）绘制线条或区域。 您可以通过线条开头的箭头拖放项目以设置排序顺序或哪个项目显示在另一个前面.
	名称	所选项目的名称显示为链接。单击该链接可打开其他可用项目的列表。
	类型	类型（仅适用于饼图和分解饼图）： 简单 - 项目的值在饼图中按比例表示 图表求和 -监控项值代表整个饼图 请注意，“图形总和” 项目的着色仅在 “比例” 项目不占用的范围内可见。
	功能	选择当一个项目的每个垂直图形像素存在多个值时将显示哪些值： **all** - 在图形中显示所有可能的值（最小值、最大值、平均值）. 请注意，对于较短的时间段，此设置无效；仅在较长时期内，当垂直图像素中的数据拥塞增加时，“全部” 开始显示最小值、最大值和平均值。此功能仅适用于 *Normal* 图形类型。另请参阅：生成图表从历史/趋势。 **avg** - 显示平均值 **last** - 显示最新值。此功能仅适用于饼图/爆炸的饼图作为图形类型。 ** max ** - 显示最大值 ** min ** - 显示最小值
	绘制样式	选择绘制样式（仅适用于普通图；堆叠图始终使用填充区域）应用于项目数据 - 线条、粗线、填充区域、点，虚线，渐变线。
	Y 轴侧	选择 Y 轴侧显示项目数据-左，右。
	颜色	选择要应用到项目数据的颜色。

图表预览

在 预览选项卡中，将显示图形的预览，以便您可以立即查看您正在创建的内容。



请注意，预览不会显示模板项的任何数据。



在此示例中，请注意显示触发级别的粗虚线和图例中显示的触发信息。

Note:
最多可显示 3 条触发线。如果有更多的触发器，然后具有较低严重性的触发器被优先用于展示。

如果图形高度设置为小于 120 像素，则不会触发显示在图例中。

3 Ad-hoc 图表

概览

虽然简单图表非常适合访问单个监控项的数据，而自定义图表 提供定制化选择，但是这两种图表都没法很轻松的快速创建多个监控项的比较图表。

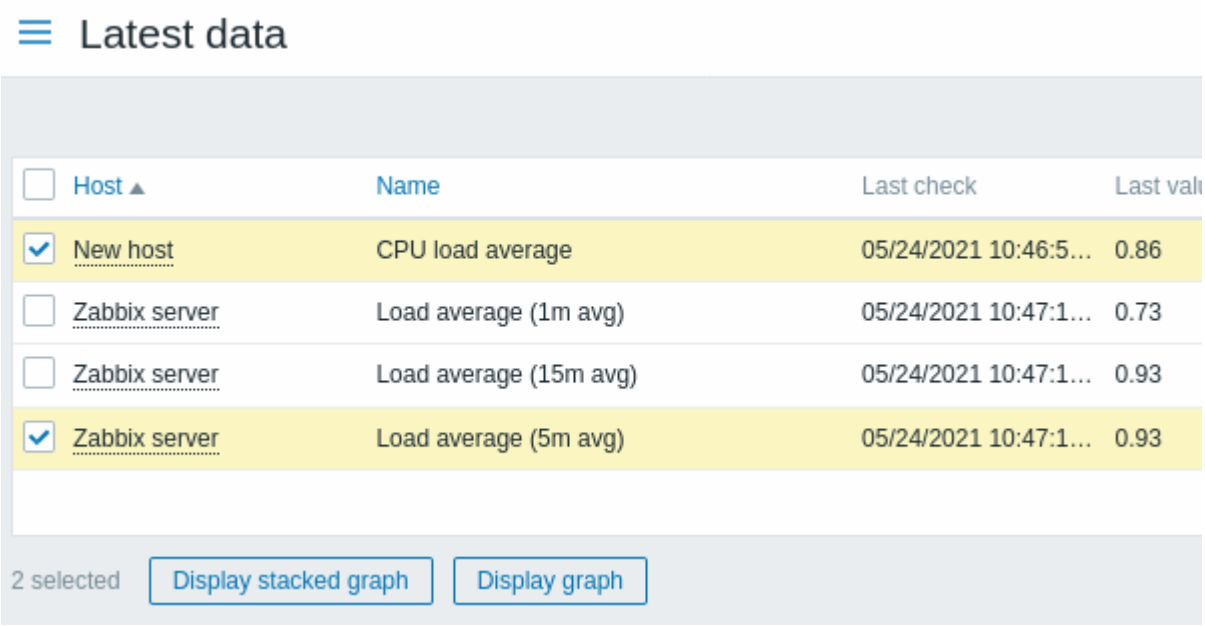
为了解决这个问题，从 Zabbix 2.4 开始可以创建 ad-hoc 以非常快速的方式绘制多个监控项的图表。

配置

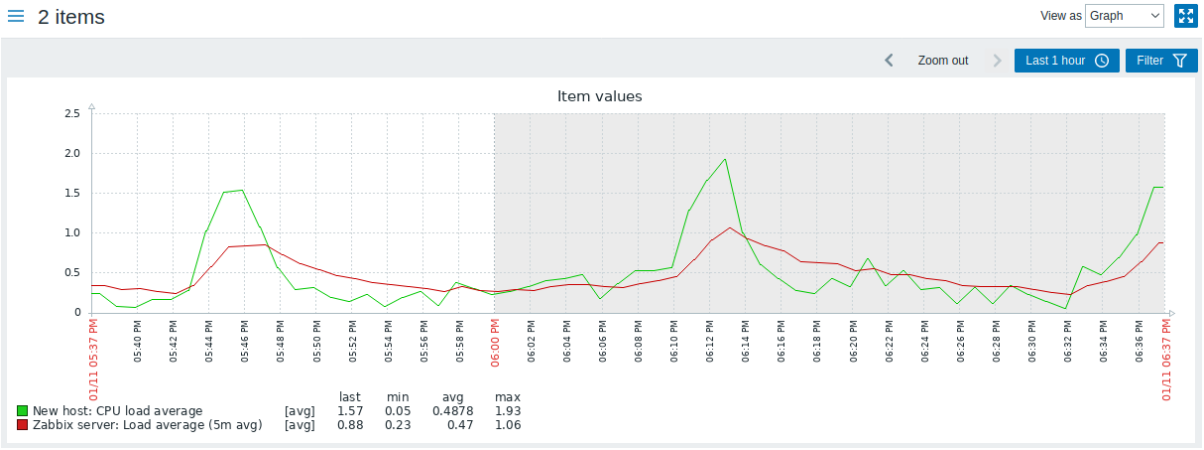
要创建 Ad-hoc 图表，请执行以下操作：

- 转到监控 → 最新数据
- 使用过滤器显示您想要的监控项
- 标记要绘制的监控项的复选框

- 点击显示堆叠数据图或显示数据图按钮

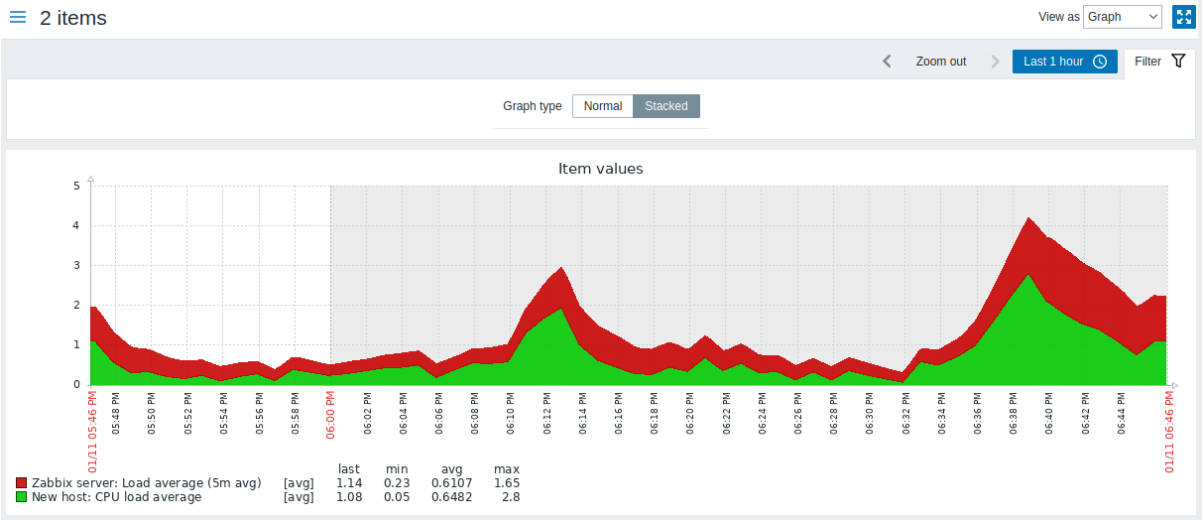


您的图表会立即创建：



请注意，为避免在图表中显示过多的线，只会显示每个监控项的平均值（不显示最小值/最大值行显示）。触发器和触发器信息不显示在图形。

在创建的图形窗口中，您可以使用时间段选择器，也可以从“正常”折线图切换到堆叠图（可来回切换）。



4 聚合图形

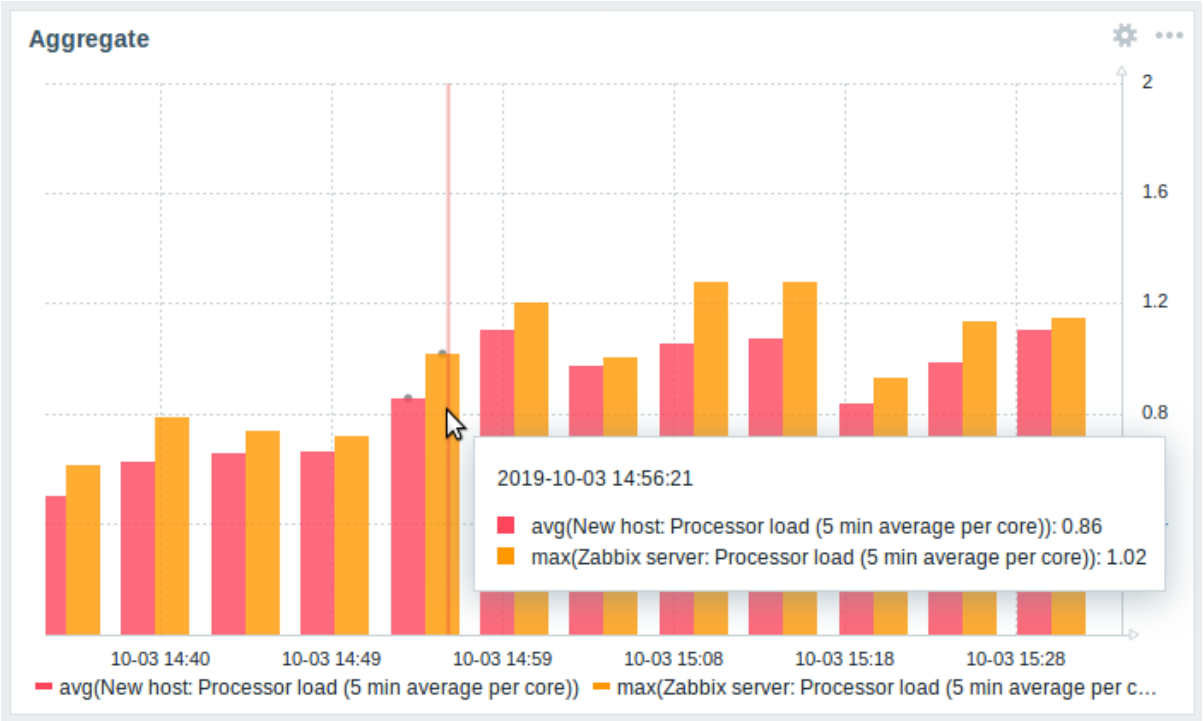
概述

在仪表板中的图形组件中可用的聚合函数，允许显示所选间隔的聚合值（5 分钟、一小时、一天），而不是所有值。

聚合选项如下：

- 最小
- 最大
- 平均
- 计数
- 总和
- 第一个（显示的第一个值）
- 最后一个（显示的最后一个值）

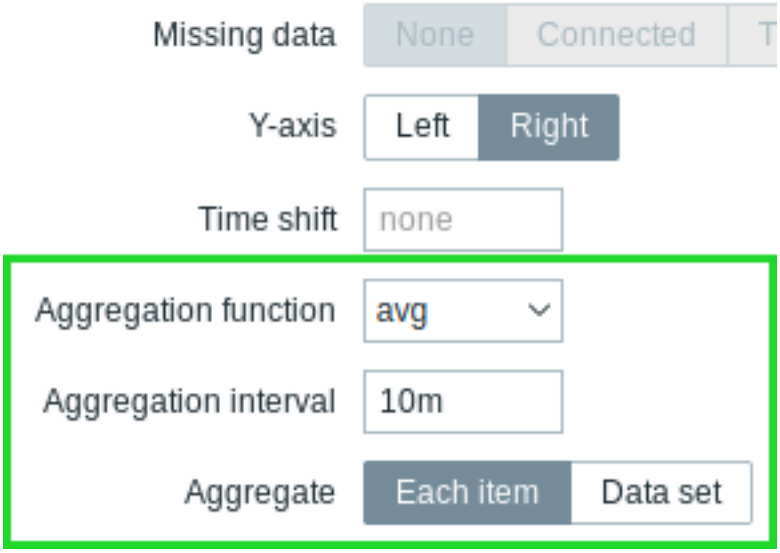
数据聚合最令人兴奋的用途是可以对一段时间内的并行数据进行比较:：



将鼠标悬停在图表中的某个时间点上时，会显示日期和时间，监控项及其聚合值。监控项是显示在括号中，括号前面是对应的聚合函数。请注意，这是图中的日期和时间，而不是实际时间。

配置

在配置图形小部件时，可以在数据集设置中使用聚合选项。[图形部件](/manual/web_interface/frontend_sections/monitoring/dashboard/widgets#g



您可以选择聚合函数和时间间隔。作为数据集可能包含多个监控项，还有另一个选项允许分别显示每个监控项或将所有数据集监控项的聚合数据作为一个聚合值显示。

用例

Nginx 服务器的平均请求数

查看每天每秒对 Nginx 服务器的平均请求数：

- 将每秒项的请求计数添加到数据集中
- 选择聚合函数 “avg” 并指定区间 “1d”
- 显示一个条形图，其中每个条形图代表平均值每天每秒的请求数

集群间每周最小磁盘空间

查看一周内集群中最低的磁盘空间。

- 添加到数据集：主机 `cluster*`, ”Free disk space on /data” 键
- 选择聚合函数 `min` 并指定间隔 `1w`
- 显示一个条形图，其中每个条形代表每个 /data 卷的每周的最小磁盘集群空间

2 网络拓扑图

概述

当你在维护一个网络的时候，你可能想看下整体基础设施的网络拓扑结构。这种情况下，你可以在 Zabbix 中创建一个拓扑图来实现，可以是网络和任何相关节点。

所有用户都可以配置网络拓扑图. 拓扑图可以是公开的 (所有用户可见) 或者私有的 (部分用户可见)。

请参照配置网络拓扑。

1 配置网络地图

概述

在 Zabbix 中配置地图首先需要定义一些常规参数来创建地图，然后用元素和元素链接填充实际的地图。

您可以使用主机、主机组、触发器、图像或其他地图来填充地图。

图标用于表示地图元素。您可以定义和图标一起显示的信息并设置以特殊的方式显示最近的问题。您可以链接图标并定义要在链接上显示的信息。

您可以通过单击图标添加可访问的自定义 URL。因此您可以将主机图标链接到主机属性或将地图图标链接到另一个地图。

地图在 监控中管理 →地图，其中它们可以被配置、管理和查看。在监控视图中，您可以单击图标并利用它链接某些脚本和网址。

自 Zabbix 3.4 起，网络地图基于矢量图形 (SVG)。

公共和私有地图

Zabbix 中的所有用户（包括非管理员用户）都可以创建网络地图。地图有一个所有者——创建它们的用户。地图可以公开或私有。

- 公共地图对所有用户可见，要查看它，用户必须具有对至少一个 map 元素的读访问权。如果用户/用户组对公共映射具有读写权限，并且至少对对应映射的所有元素 (包括链接中的触发器) 具有读权限，则可以对公共映射进行编辑。
- 私有地图仅对其所有者和地图所有者所共享的用户/用户组可见地图可以被分组为共享于所有者。普通（非超级管理员）用户只能和他所属的组和用户进行分享。管理员级别的用户可以查看私有地图，无论是所有者还是属于共享用户列表的私有地图。私有地图可由地图所有者编辑，用户/用户组对其具有读写权限，至少有读取相应地图的所有元素的权限，包括链接中的触发器。

用户没有读取权限的地图元素是以灰色图标显示，元素的所有文本信息被隐藏。然而，触发标签是可见的，即使用户没有触发器的权限。

要将元素添加到地图，用户还必须至少有读权限。

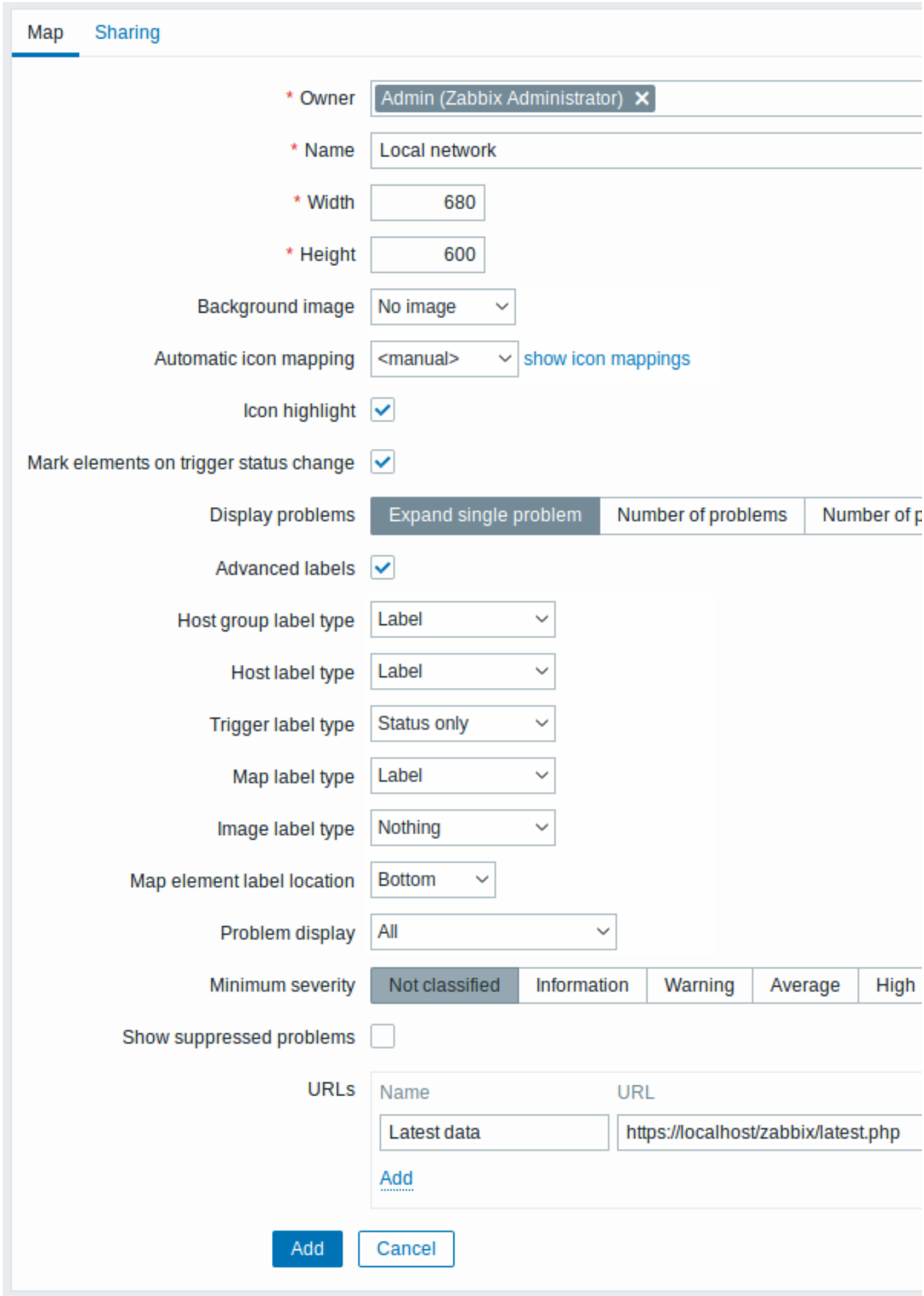
创建地图

要创建地图，请执行以下操作：

- 转到监控 → 地图
- 转到所有地图的视图
- 点击创建地图

您也可以使用 *Clone* 和 *Full clone* 按钮基于现有地图的配置形式以创建新地图。点击克隆将保留原始地图的总体布局属性，但没有元素。完整克隆将保留常规布局属性以及原始地图的所有元素。

Map 选项卡包含一般地图属性：



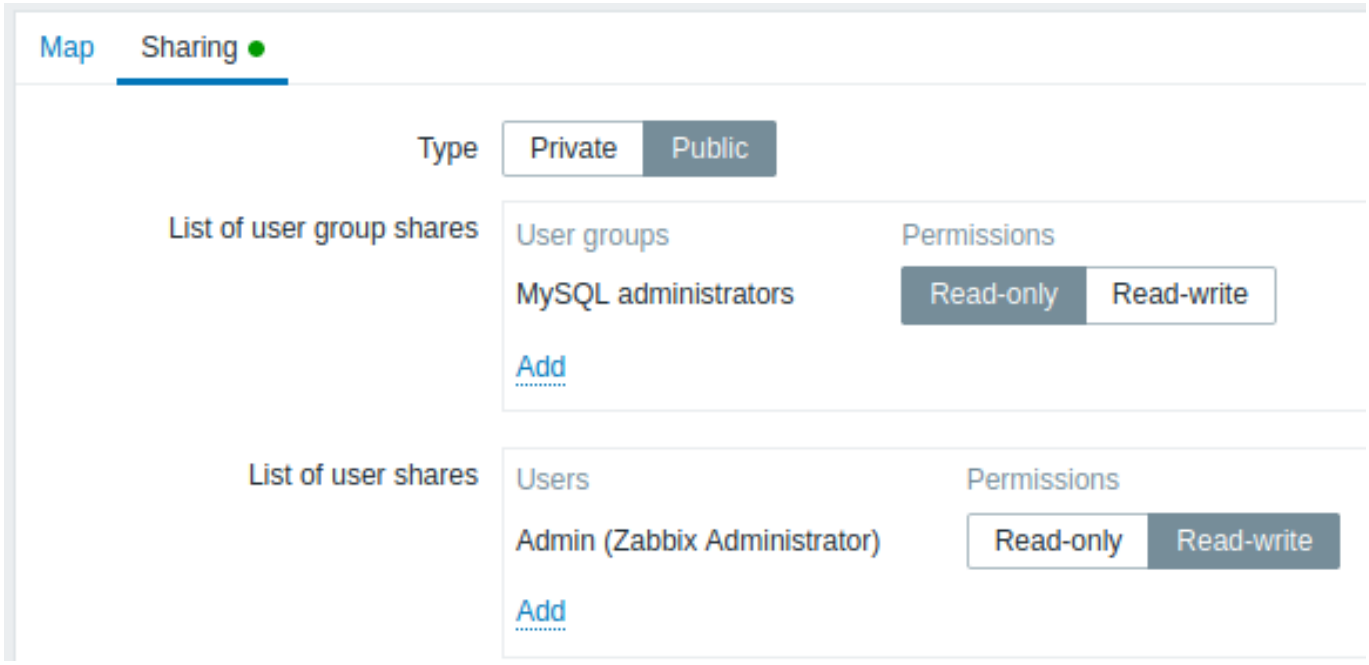
所有标有红色星号的是必填字段。

通用地图属性：

参数	说明
所有者	地图所有者的姓名。
名称	唯一的地图名称。
宽度	以像素为单位的地图宽度。
高度	以像素为单位的地图高度。
背景图片	使用背景图片： 无图片 - 无背景图片（白色背景） 图片 - 选择的图片用作背景图片。不执行缩放。您可以使用地理地图或任何其他图像来增强您的地图。
自动图标映射	您可以设置使用自动图标映射，在 *Administration → General → Icon mapping* 中配置。图标映射允许将某些图标映射到某些主机清单字段。
图标突出显示	如果您选中此框，地图元素将突出显示。 具有活动触发器的元素将收到圆形背景，与最严重的触发器颜色相同。此外，如果确认所有问题，圆圈周围会显示一条粗绿线。 “已禁用” 或 “维护中” 状态的元素将分别获得方形背景、灰色和橙色。 另请参阅：查看地图
在触发状态更改时标记元素	最近触发状态的更改（最近的问题或已解决的问题）将在元素图标的三个方向用没有标签标记（向内的红色三角形）突出显示。标记显示 30 分钟。
显示问题	使用地图元素选择问题的显示方式： 展开单个问题 - 如果只有一个问题，则显示问题名称。否则，显示问题总数。 问题数 - 显示问题总数 问题数并展开最关键的一个 - 最关键的名称问题并显示问题总数。 “最关键” 是根据问题严重性确定的，如果相等，则根据问题事件 ID（ID 较高或最新的问题首先显示）确定。对于触发地图元素，它基于问题严重性，如果相等，则触发列表中的触发位置。如果同一个触发器出现多个问题，将显示最近的一个。
高级标签	如果您选中此框，您将能够为单独的元素类型定义单独的标签类型。
地图元素标签类型	用于地图元素的标签类型： 标签 - 地图元素标签 **IP** 地址 - IP 地址 元素名称 - 元素名称（例如，主机名） 仅状态 - 仅状态（OK 或 PROBLEM） **Nothing** - 不显示标签
地图元素标签位置	相对于地图元素的标签位置： 底部 - 地图元素下方 左 - 左侧 右 - 右侧 **Top** - 地图元素上方
问题显示	将问题计数显示为： 全部 - 将显示完整的问题计数 分隔 - 未确认的问题计数将单独显示为总问题计数的数字 **Unacknowledged only** - 只显示未确认的问题计数
最低触发严重性	低于所选最低严重性级别的问题将不会显示在地图上。 例如，选择警告，信息和未分类级别触发器的更改将不会反映在地图中。 从 Zabbix 2.2 开始支持此参数。
显示抑制的问题	标记复选框以显示由于主机维护而被抑制（未显示）的问题。
URLs	可以定义每种元素类型的 URL（带有标签）。当用户在地图查看模式下单击元素时，这些将显示为链接。 宏可用于地图 URL 名称和值。有关完整列表，请参阅支持的宏 并搜索 “映射 URL 名称和值”。

分享

分享选项卡包含地图类型以及私有地图的分析享选项（用户组，用户）：



参数	说明
Type	选择地图类型： **Private** - 地图仅对选定的用户组和用户可见 **Public** - 地图对所有人可见
用户组共享列表	选择地图可访问的用户组。 您可以允许只读或读写访问。
用户共享列表	选择可以访问地图的用户。 您可以允许只读或读写访问。

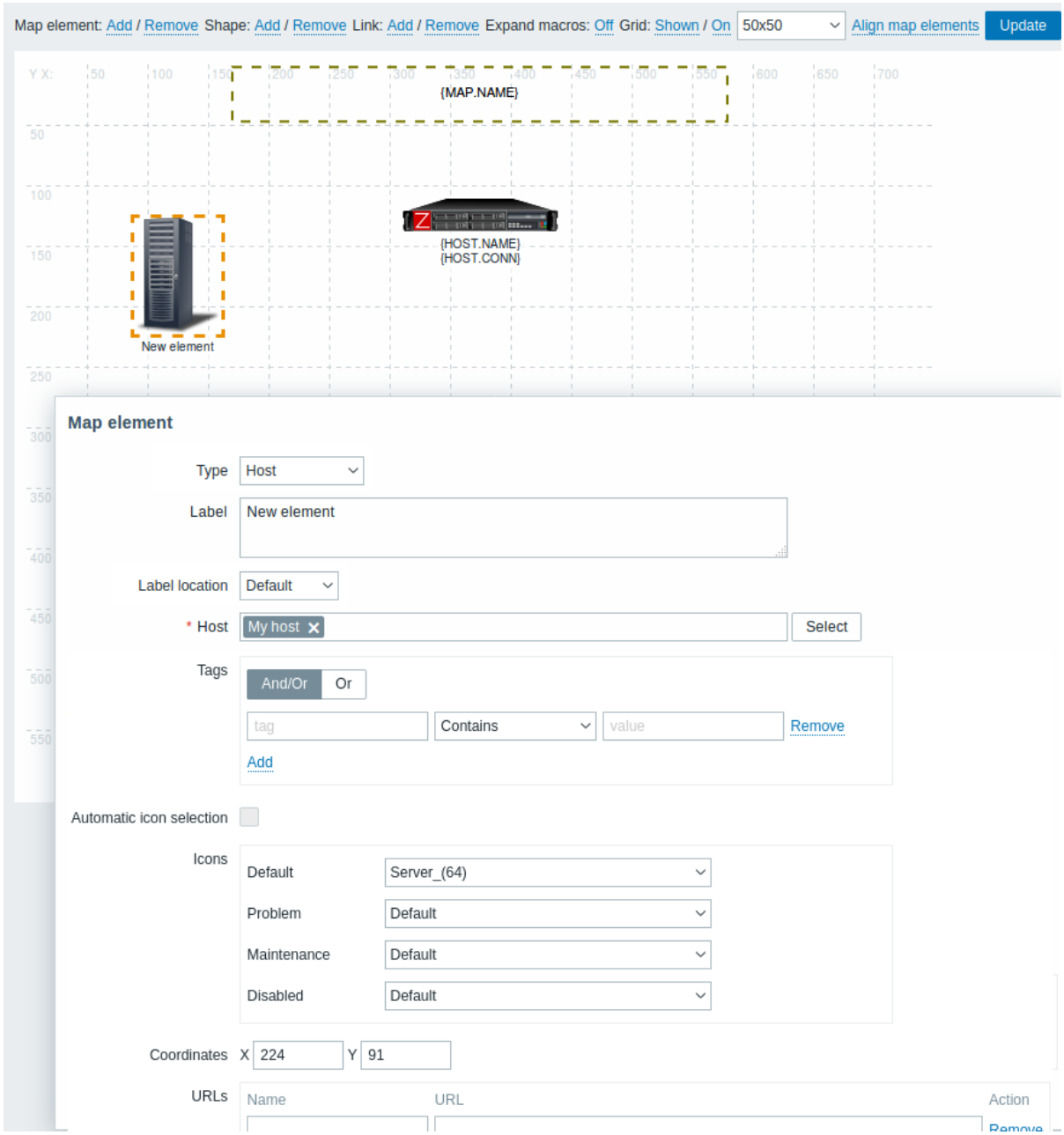
当您单击 *Add* 保存此地图时，您已创建了一个具有名称、尺寸和某些偏好的空地图。现在您需要添加一些元素。为此，请单击地图列表中的 *Constructor* 以打开可编辑区域。

添加元素

要添加元素，请单击 “地图元素” 旁边的添加。新元素将出现在地图的左上角。拖放到任何你喜欢的地方。

请注意，使用 Grid 选项 “On”，元素将始终与网格（您可以从下拉列表中选择各种网格大小，也可以隐藏/显示网格）。如果您想将元素放置在任何位置而不对齐，请转 “关闭” 选项。（随机元素稍后可以再次对齐到带有对齐地图元素按钮的网格。）

现在你已经有了一些元素，你可能想要开始通过给出名称等来区分它们。通过单击元素显示表单，您可以设置元素类型，命名，选择不同的图标等



地图元素属性：

参数	说明
类型	元素类型： **Host** - 表示所选主机的所有触发器状态的图标 **Map** - 表示地图所有元素状态的图标 触发器 - 表示一个或多个触发器状态的图标 主机组 - 表示属于选定组 图片 - 一个图标，未链接到任何资源
标签	图标标签，任何字符串。 可以使用宏和多行字符串。 该字段支持表达式宏，但只能与 `avg`、`last`、`min` 和 `max` 函数，以时间作为参数（例如，`{?avg(/host/key,1h)}`）。 查看支持的完整列表宏，请参阅支持的宏 并搜索 “地图元素标签”。
标签位置	与图标相关的标签位置： 默认 - 地图的默认标签位置 底部 - 图标下方 左 - 到左侧 右侧 - 右侧 顶部 - 图标上方

参数	说明
主机	如果元素类型为 “Host”，请输入主机。此字段是自动完成的，因此开始输入主机名将提供匹配主机的下拉列表。向下滚动以选择。单击 “x” 以删除所选内容。
地图	选择地图，如果元素类型是’Map’。
触发器	如果元素类型是 “触发器”，请在下面的 *New triggers* 字段中选择一个或多个触发器，然后单击 *Add*。 可以更改所选触发器的顺序，但只能在触发的严重程度相同。多个触发器选择还会影响构造模式和视图模式下的 {HOST.*} 宏分辨率。 // 1 在构造模式中// 第一个显示的 {HOST.*} 宏将根据第一个触发器进行解析在列表中（基于触发严重性）。 // 2 查看模式// 取决于通用地图属性中的显示问题 参数。 * 如果选择 *Expand single problem* 模式，第一个显示的 {HOST.*} 宏将根据最新检测到的问题触发器（与严重性无关）或列表中的第一个触发器（如果未检测到问题）得到解决); * 如果选择了 *Number of questions and expand most critical one* 模式，第一个显示的 {HOST.*} 宏将根据触发器的严重性得到解决。
主机组	如果元素类型为 “主机组”，请输入主机组。此字段是自动完成的，因此开始输入组的名称将提供匹配组的下拉列表。向下滚动以选择。单击 “x” 以删除所选内容。
标签	指定标签以限制小部件中显示的问题数量。可以包括和排除特定的标签和标签值。可以设置几个条件。标签名称匹配始终区分大小写。 每个条件都有多个可用的运算符： **Exists** - 包括指定的标签名称 **Equals** - 包括指定的标签名称和值（区分大小写） 包含 - 包含指定的标签名称，其中标签值包含输入的字符串（子字符串匹配，不区分大小写） 不存在 - 排除指定的标签名称 不等于 - 排除指定的标签名称和值（区分大小写） 不包含 - 排除标签值包含输入字符串的指定标签名称（子字符串匹配，不区分大小写） 条件有两种计算类型： **And/Or** - 必须满足所有条件，标签名相同的条件将按 Or 条件分组 或 - 如果满足一个条件就足够了 此字段可用于主机和主机组元素类型。
自动图标选择	在这种情况下，将使用图标映射来确定要显示的图标。
图标	在这些情况下，您可以选择为元素显示不同的图标：默认、问题、维护、禁用。
坐标 *X*	地图元素的 X 坐标。
坐标 *Y*	地图元素的 Y 坐标。
URLs	可以为元素设置特定于元素的 URL。当用户在地图查看模式下单击元素时，这些将显示为链接。如果元素有自己的 URL，并且定义了其类型的地图级 URL，它们将组合在同一个菜单中。 宏可用于地图元素名称和值。有关完整列表，请参阅支持的宏 并搜索 “映射 URL 名称和值”。

添加的元素不会自动保存。如果你离开页面，所有更改都可能丢失。

因此，最好单击顶部的 **Update** 按钮右上角。单击后，无论您使用什么，更改都会保存在以下弹出窗口中选择。

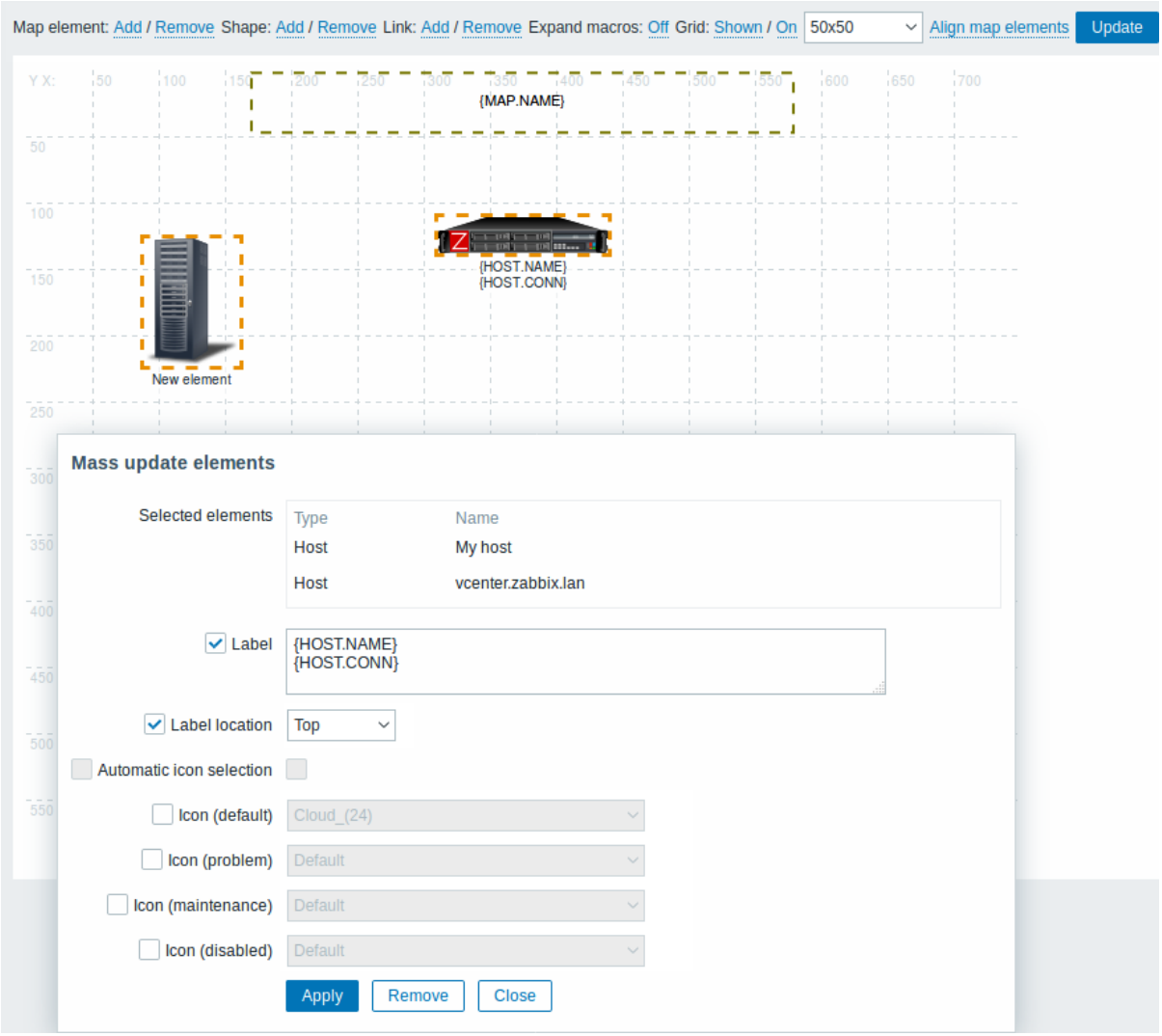
选定的网格选项也与每个地图一起保存。

选择元素

要选择元素，请选择一个，然后按住 *Ctrl* 选择其他。

您还可以通过在可编辑区域并选择其中的所有元素。

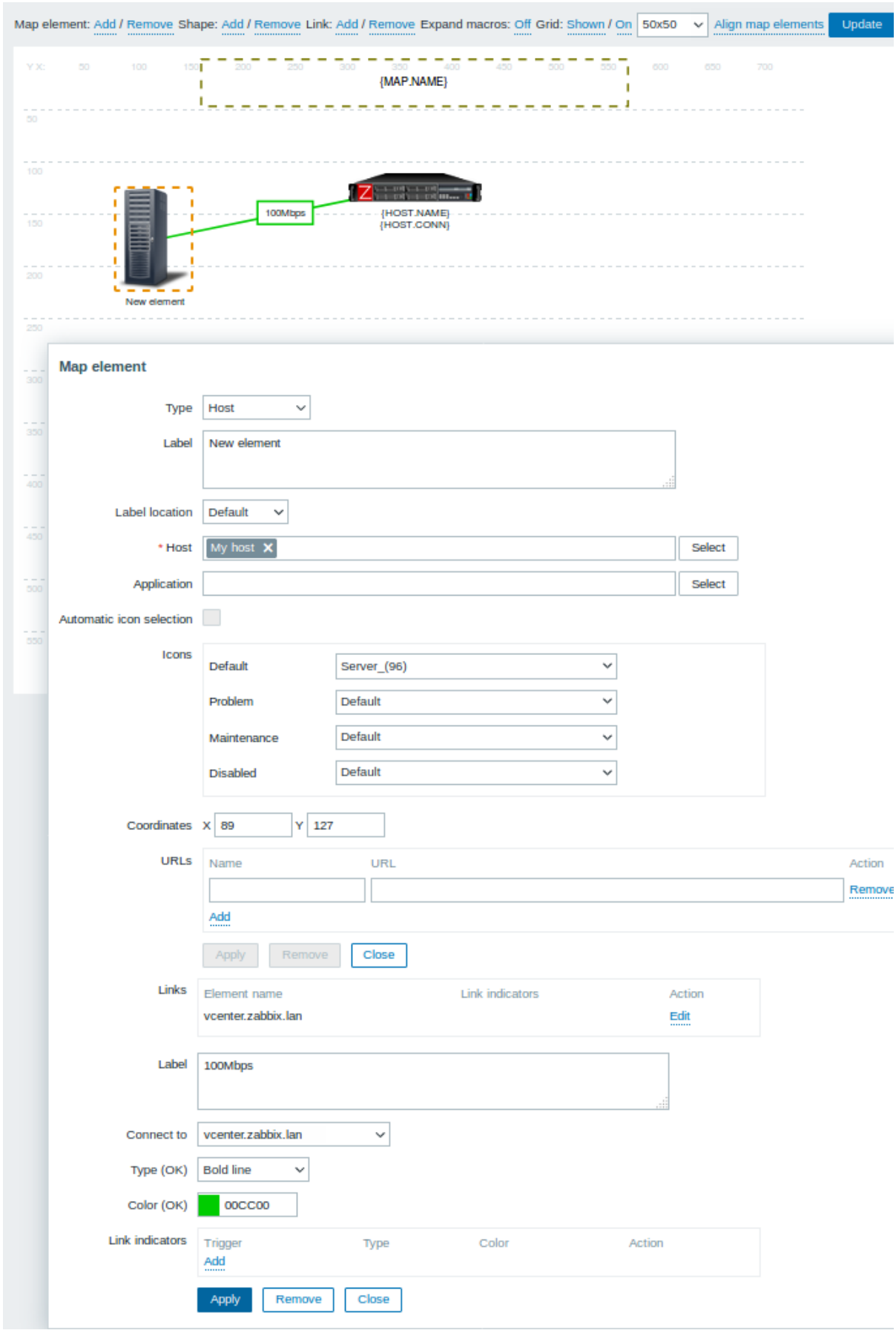
一旦选择了多个元素，元素属性形式就会发生变化到批量更新模式，以便您可以更改选定的属性元素一气呵成。为此，请使用复选框标记属性，然后为其输入一个新值。您可以在此处使用宏（例如，{HOST.NAME} 用于元素标签）。



链接元素

一旦你在地图上放置了一些元素，就该开始链接了他们。要链接两个元素，您必须首先选择它们。随着元素选中，单击链接旁边的添加。

创建链接后，单元素表单现在包含一个附加的链接部分。单击 *Edit* 以编辑链接属性。


Map element: Add / Remove Shape: Add / Remove Link: Add / Remove Expand macros: Off Grid: Shown / On 50x50 Align map elements Update
{MAP.NAME}
100Mbps
{HOST.NAME}
{HOST.CONN}
New element
Map element
Type Host
Label New element
Label location Default
* Host My host Select
Application Select
Automatic icon selection
Icons
Default Server_(96)
Problem Default
Maintenance Default
Disabled Default
Coordinates X 89 Y 127
URLs Name URL Action
Remove
Add
Apply Remove Close
Links Element name Link indicators Action
vcenter.zabbix.lan Edit
Label 100Mbps
Connect to vcenter.zabbix.lan
Type (OK) Bold line
Color (OK) 00CC00
Link indicators Trigger Type Color Action
Add
Apply Remove Close

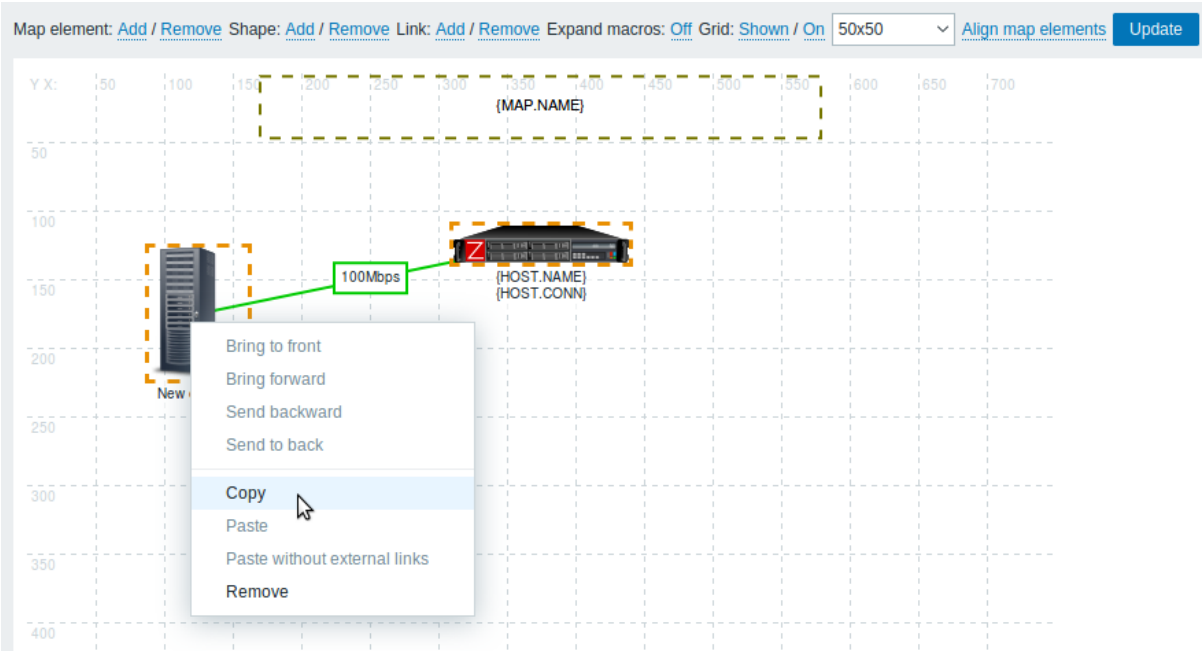
链接属性：

参数	说明
标签	将在链接顶部呈现的标签。 此字段支持表达式宏，但仅限于 `avg`、`last`、`min` 和 `max` 函数，以时间为参数（例如，`{?avg(/host/key,1h)}`）。
连接到	链接连接到的元素。
类型 *(OK)*	默认链接样式： **Line** - 单行 粗线 - 粗线 **Dot** - 点 虚线 - 虚线
颜色（*OK*）	默认链接颜色。
链接指示器	链接到链接的触发器列表。如果触发器的状态为 PROBLEM，则其样式将应用于链接。

移动和复制粘贴元素

几个选定的元素可以移动到地图中的另一个位置通过单击选定元素之一，按住鼠标按钮，然后将光标移动到所需位置。

一个或多个元素可以通过选择元素来复制，然后用鼠标右键单击选定的元素并选择从菜单中复制。



要粘贴元素，请使用鼠标右键单击地图区域并从菜单中选择粘贴。粘贴没有外部链接选项将粘贴元素，只保留它们之间的链接选定的元素。

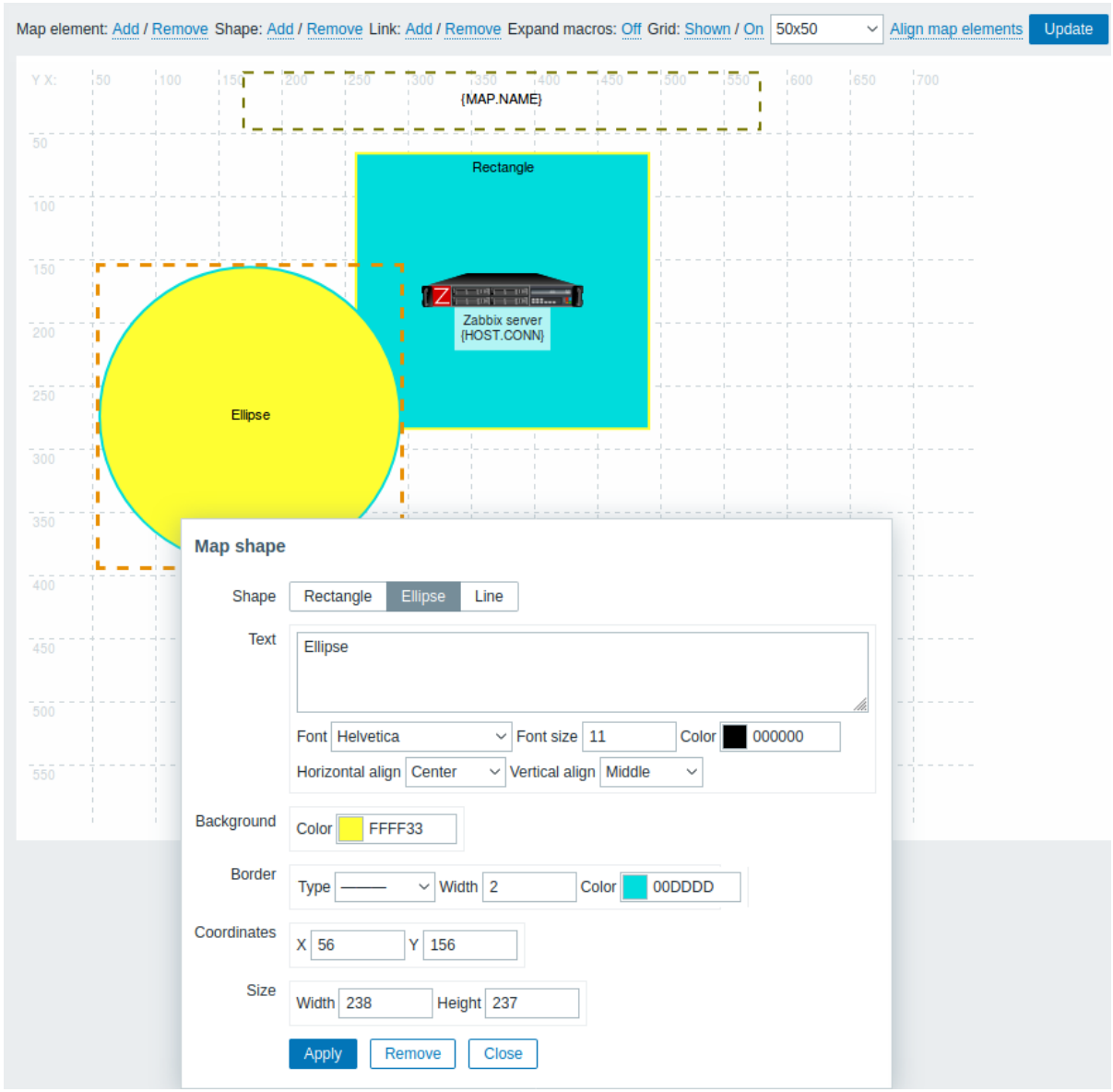
复制粘贴在同一浏览器窗口中工作。不支持键盘快捷键。

添加形状

除了地图元素，还可以添加一些形状。形状不是地图元素；它们只是一种视觉表现。例如，可以使用矩形作为背景来分组一些主机。可以添加矩形和椭圆形。

要添加形状，请单击形状旁边的添加。新形状将出现在地图的左上角。将其拖放到您喜欢的任何位置。

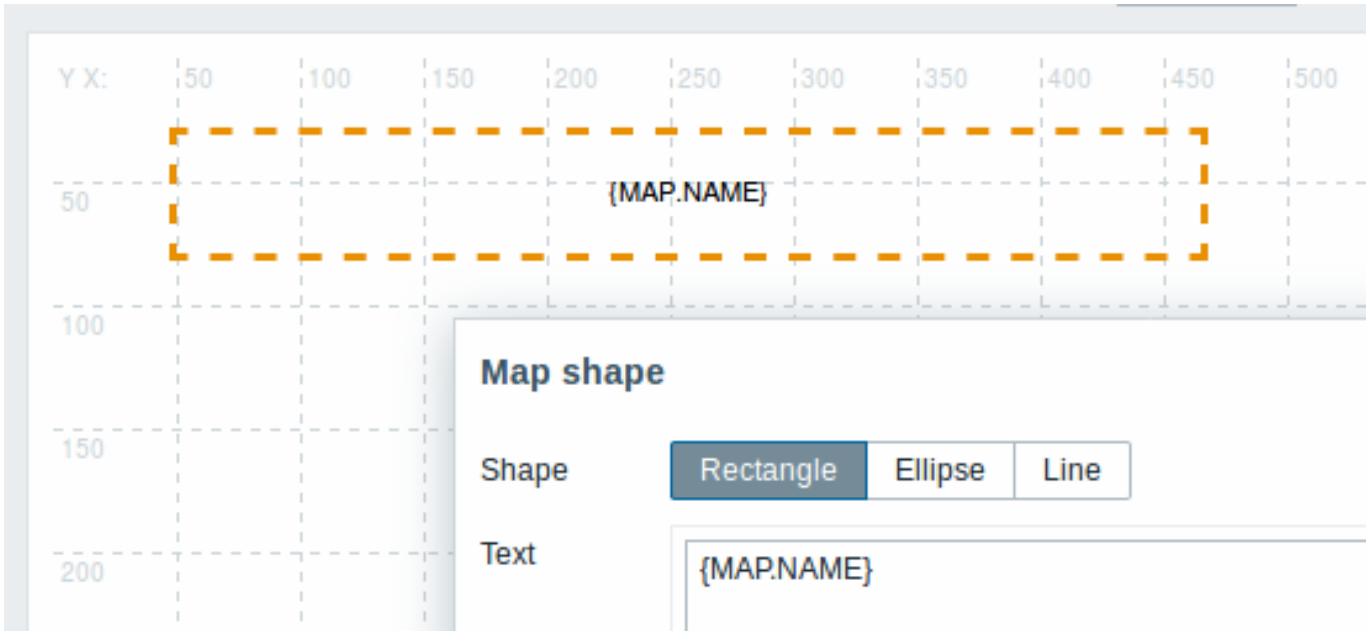
添加了具有默认颜色的新形状。通过单击形状，窗体显示，您可以自定义形状的外观，添加文本，等等。



要选择形状，请选择一个，然后按住 *Ctrl* 选择其他。选择几种形状后，常见的属性可以是质量更新，与元素类似。

可以在形状中添加文本。表达宏 支持文本，但只有 `avg`、`last`、`min` 和 `max` 函数，有时间作为参数（例如，`{?avg(/host/key,1h)}`）。

要仅显示文本，可以通过删除形状边框（在 边框字段中选择 “无”）。例如，取请注意上面屏幕截图中可见的 `{MAP.NAME}` 宏是如何实现的实际上是一个带有文本的矩形形状，单击时可以看到宏：



{MAP.NAME} 在查看地图时解析为配置的地图名称。

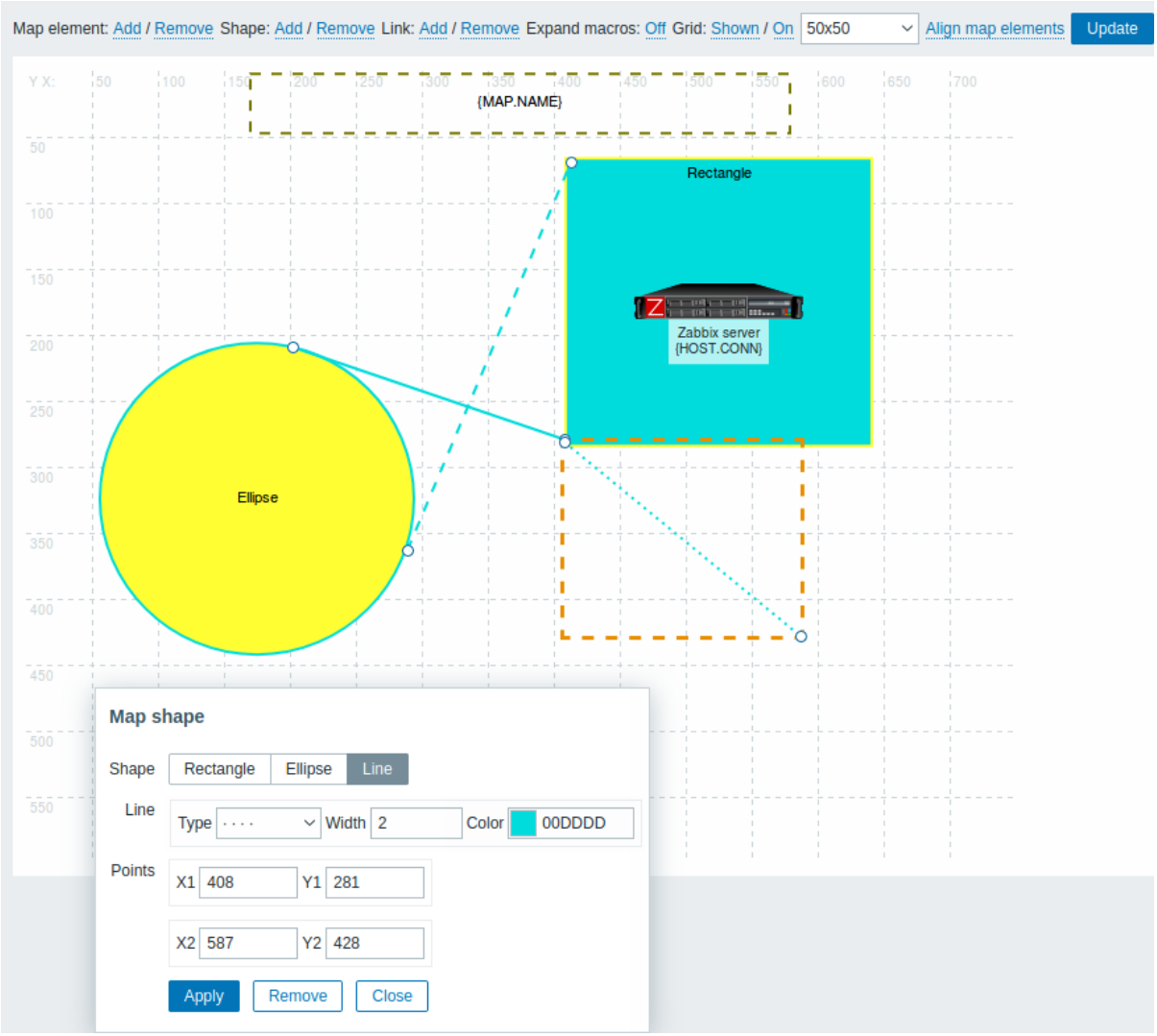
如果文本中使用了超链接，则它们在查看时变为可点击的地图。

文本的换行始终在形状内“开启”。然而，在一椭圆，线条被包裹起来，好像椭圆是一个矩形。没有实现自动换行，所以长词（不适合的词形状）没有被包裹，而是被屏蔽（构造器页面）或被剪裁（其他页面有地图）。

添加行

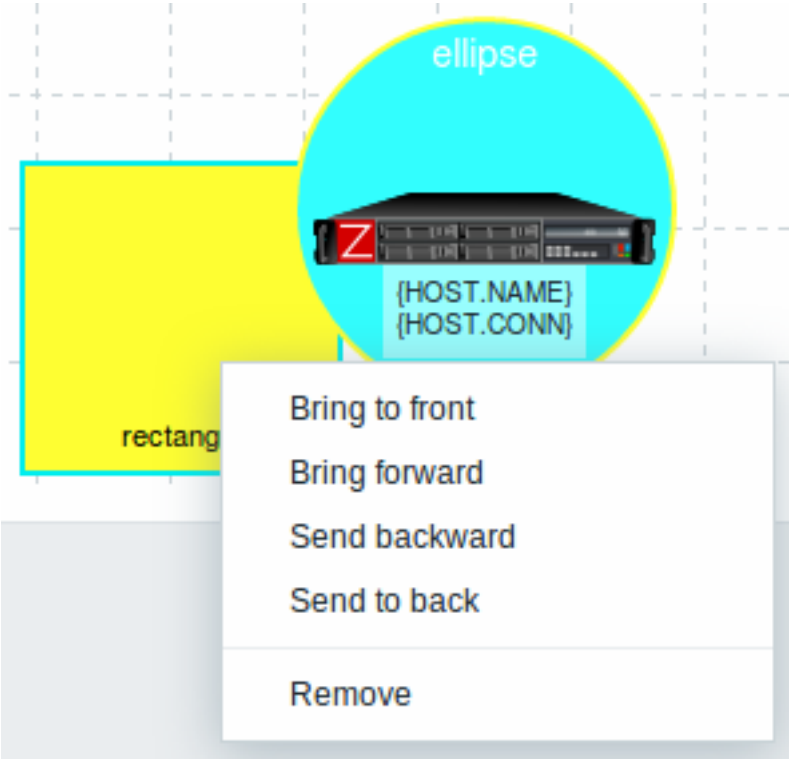
除了形状之外，还可以添加一些线。线可以用于链接地图中的元素或形状。

要添加线，请单击形状旁边的添加。一个新的形状将出现在地图的左上角。选择它并点击线编辑表格以将形状更改为线条。然后调整线属性，例如线型、宽度、颜色等。



排序形状和线条

要将一个形状放在另一个形状前面（反之亦然），请用鼠标单击右键调出地图形状菜单。

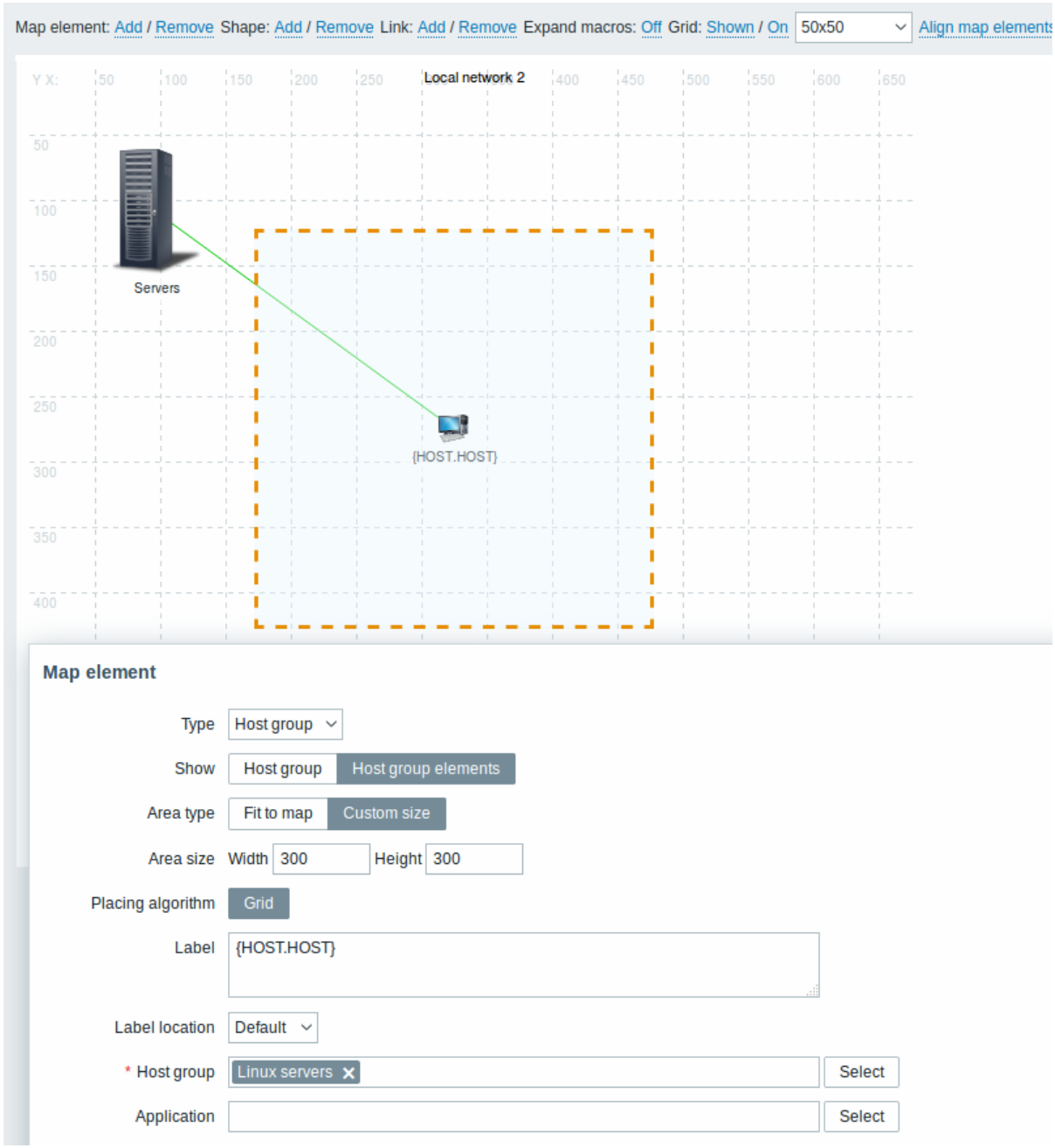


2 主机组元素

概述

本节说明如何在配置网络地图时添加“ Host group”类型的元素。

配置



所有必填字段都标有红色星号。

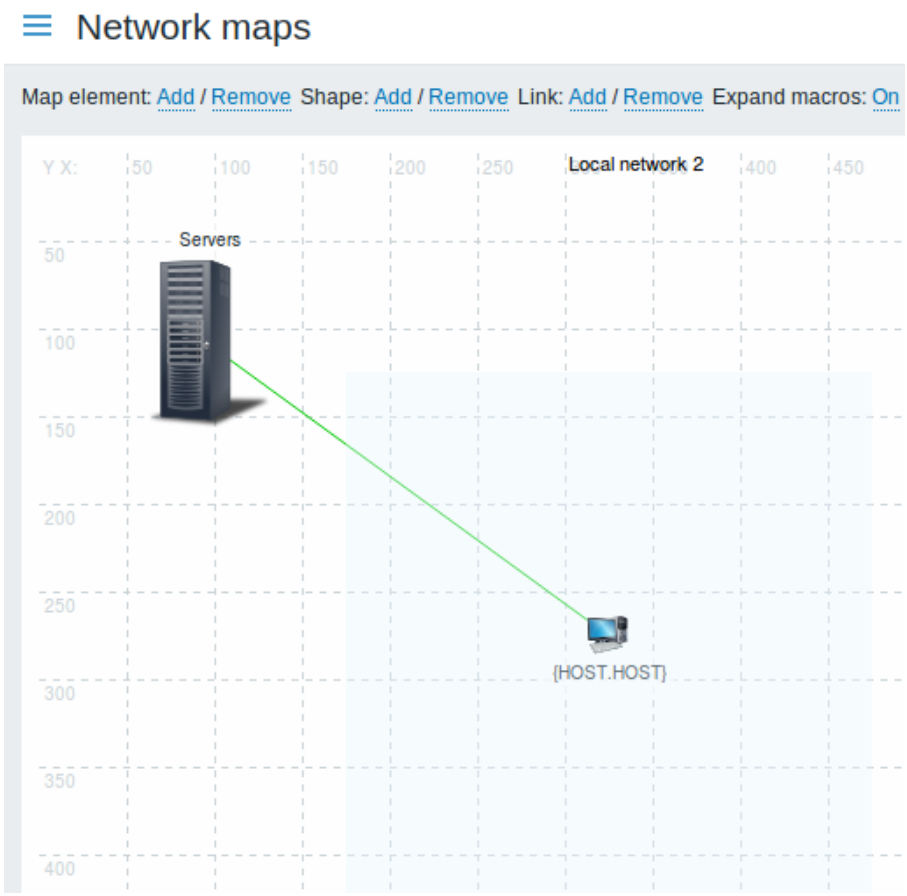
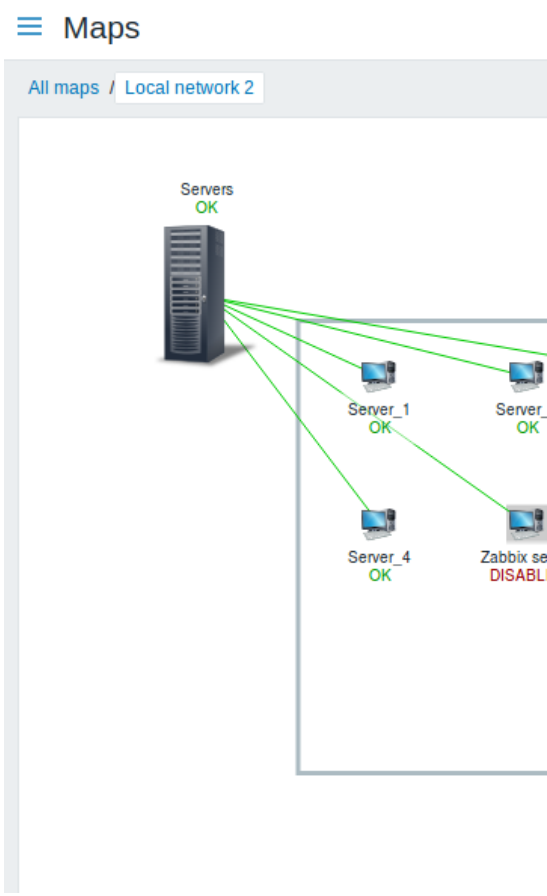
此表包含 *Host group* 元素类型的典型参数：

参数	说明
类型	选择元素类型: 主机组 - 表示属于所选组的所有主机的所有触发器状态的图标
显示	显示选项： 主机组 - 选择此选项将导致单个图标显示有关特定主机组的相应信息 主机组元素 - 选择此选项将结果是多个图标显示有关某个主机组的每个元素（主机）的相应信息
区域类型	如果选择了“主机组元素”参数，则此设置可用： 适合地图 - 所有主机组元素均等放置在地图中 ** 自定义大小 * * - 手动设置要显示的所有主机组元素的地图区域
区域大小	如果选择了“主机组元素”参数和“区域类型”参数，则此设置可用： 宽度 - 用于指定地图区域宽度的数值 高度 - 要输入以指定地图区域高度的数值
放置算法	网格 – 显示所有主机组元素的唯一可用选项

参数	说明
标签	图标标签，任意字符串。 标签中可以使用宏和多行字符串。 如果地图元素的类型是“主机组””指定某些宏会对显示每个主机的相应信息的地图视图产生影响。例如，如果使用 {HOST.IP} 宏，则编辑地图视图将仅显示宏 {HOST.IP} 本身，而地图视图将包含并显示每个主机的唯一 IP 地址

查看主机组元素

如果“主机组元素”显示选项为选择。选择“主机组元素”作为 * show * 选项时，您起初只会看到主机组的一个图标。然而，当你保存地图，然后转到地图视图，你会看到地图包括特定主机组的所有元素（主机）：

地图编辑视图	地图视图
	

请注意如何使用 {HOST.NAME} 宏。在地图编辑中，宏名称是不解析的，而在查看地图时，所有唯一名称的主机都被显示。

3 链接指标

概述

你可以将一些触发器链接网络地图中的元素。当这些触发器进入问题状态时，该链接可以查看相关内容。

在配置链接时，可以设置默认链接类型和颜色。当你将触发器关联到链接时，你可以根据这些触发器来分配不同的链接类型和颜色。

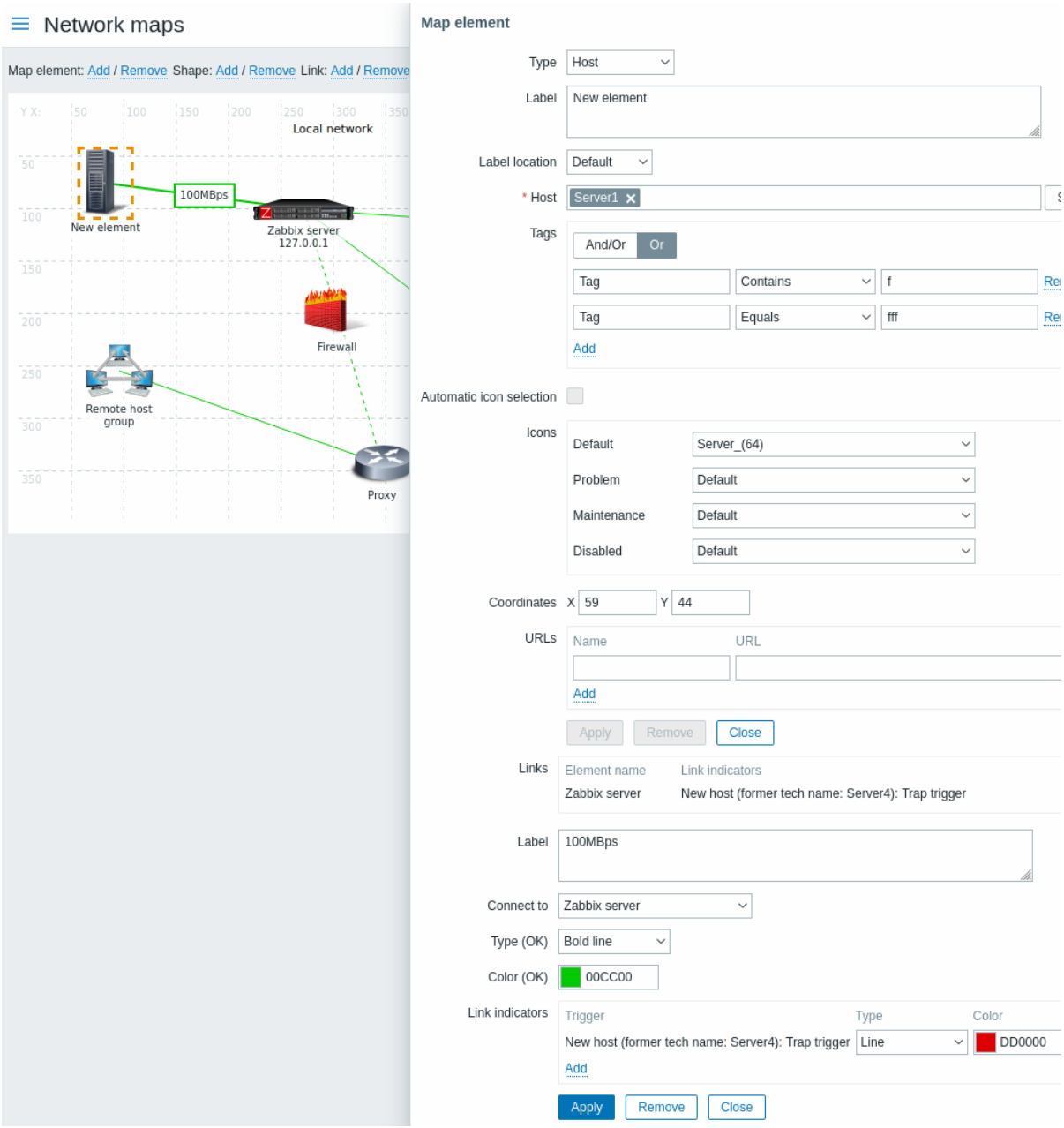
如果这些触发器中的任何一个进入问题状态，它们的连接样式和颜色将显示在链接上。所以你的默认链接可能是一条绿线。现在，由于触发器处于问题状态，你的链接可能变为粗体红色（如果你这样定义的话）。

配置

要将触发器分配为链接指示器，请执行以下操作：

- 选择地图元素
- 单击 链接部分中的 编辑以获取相应的链接

- 点击链接指标块中的添加并选择一个或多个触发器



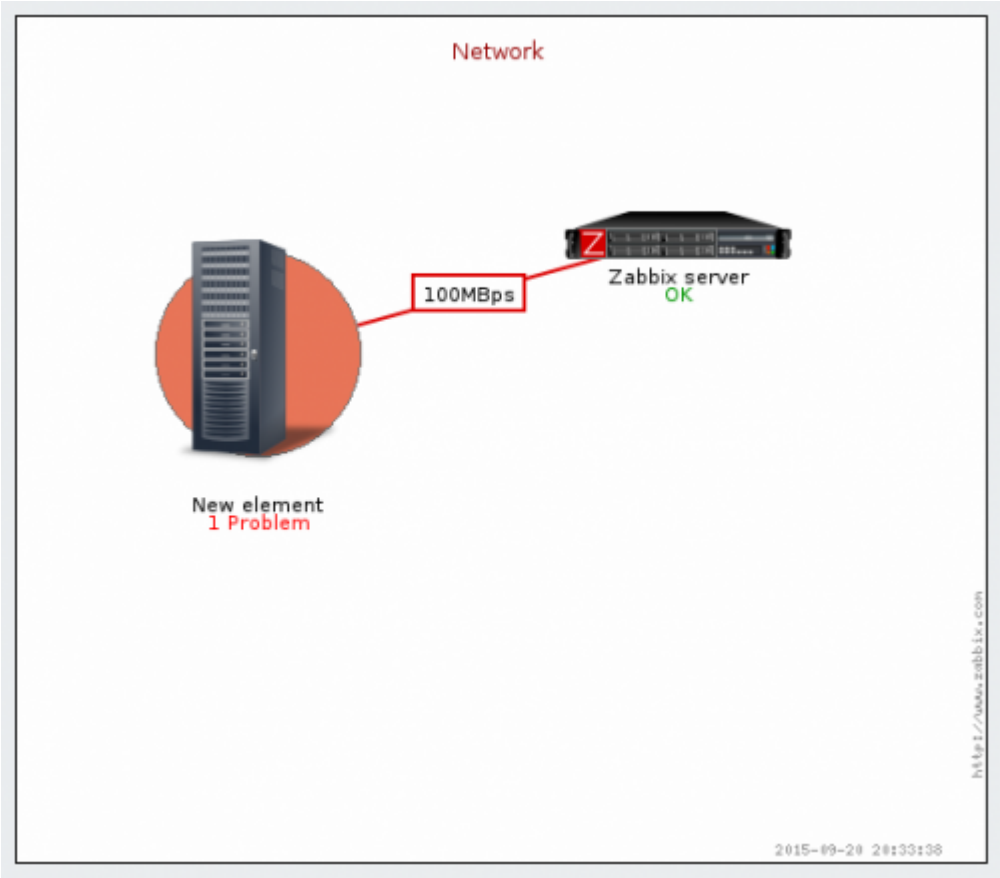
所有必填字段都标有红色星号。

在链接指标列表中可以看到添加的触发器。

你可以直接从列表中设置每个触发器的链接类型和颜色。完成后，点击 应用，关闭表格并点击 更新保存地图更改。

展示

在 监控 → 地图中，在触发器进入问题状态时相应的颜色的链接将显示。



Note:
如果多个触发器进入问题状态，则问题具有最高严重性的将确定链接样式和颜色。如果具有相同严重性的多个触发器分配给同一个映射链接，ID 最低的优先。另请注意：

1. 最低严重性触发和显示抑制问题设置从地图配置影响考虑哪些问题。
2. 在触发器有多个问题的情况下（发生了多个问题），每个问题的严重性可能与触发器不同严重性（手动更改），可能有不同的标签（由于宏），并且可能被压制。

3 仪表板

仪表板及其小部件提供了一个强大的可视化平台，其中包括现代图表、地图、幻灯片显示等工具。



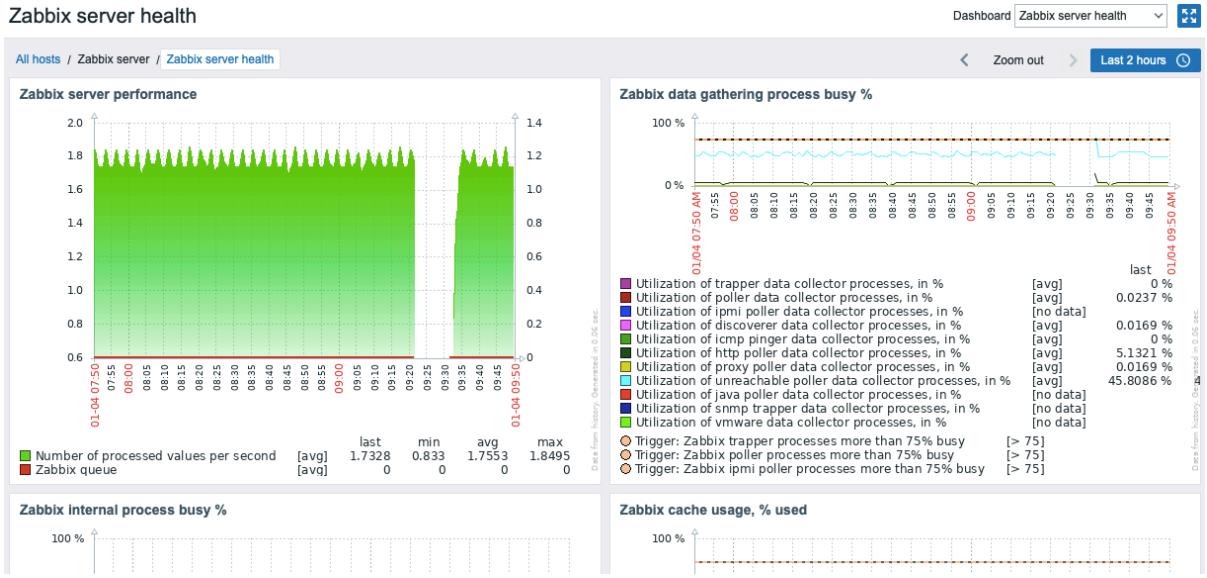
4 主机仪表盘

概述

主机仪表盘与全局仪表盘类似，仅显示主机相关的数据，没有所有者。

主机仪表盘在链接到主机的模板 级进行配置，然后根据主机生成仪表盘。主机仪表盘的部件仅可以复制给相同模板的主机。全局仪表盘的部件不能拷贝到主机仪表盘。

主机仪表盘 不能通过监控 →仪表盘部分进行配置和访问, 这种方式是全局仪表盘的配置和访问方式。主机仪表盘的访问方式通过如下部分。



可以通过在配置仪表盘右上角的下拉菜单进行切换来查看主机仪表盘。通过监控 → 主机部分，选择左上角仪表盘名称导航链接下的所有主机进行切换。

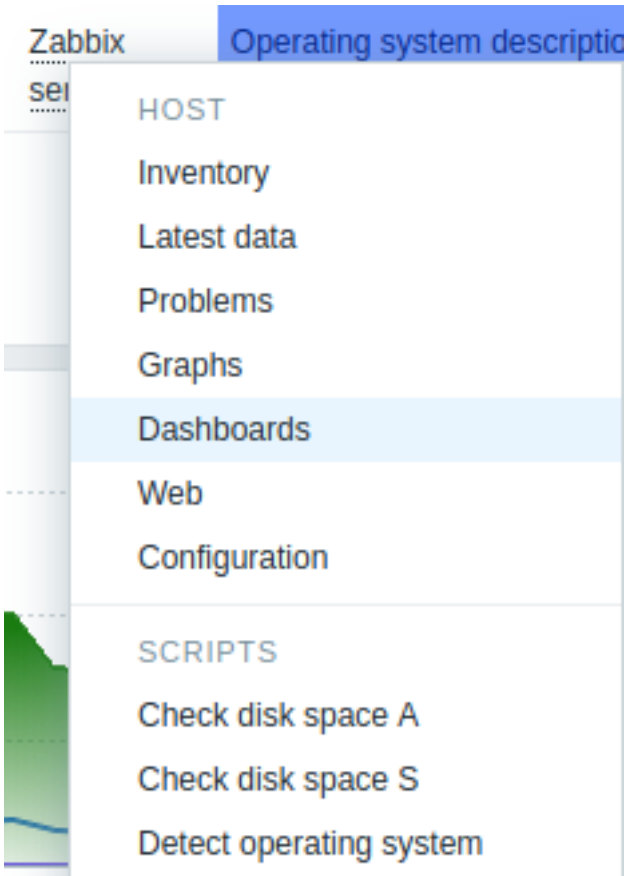
主机仪表盘的小部件不可编辑。

注意，主机仪表盘在 Zabbix 5.2 之前是被用作主机聚合图形之中的。所以在导入早期版本的包含聚合图形的模板时，聚合图形会被忽略。

访问主机仪表盘

主机仪表盘的访问方式如下:

- 通过主机菜单（前端的很多部分都可以看到）：
  - 点击主机名称，然后在下拉菜单中选择 仪表盘



- 当在全局搜索中搜索主机名称时:

- 点击搜索结果中的 仪表盘链接

- 点击资产 →主机的主机名称时:
  - 点击 仪表盘

8 模板

概览

模板是可以方便地应用于多个主机的一组实体。实体可以是：

- 监控项
- 触发器
- 图表
- 仪表盘
- 低级别自动发现规则
- web 场景

由于现实生活中的许多主机是相同或类似的，所以，您为一个主机创建的一组实体（监控项，触发器，图表，...）可能对许多主机有用。当然，您可以将它们复制到每个新的主机上，但需要费很大功夫。相反，使用模板，您可以将它们复制到一个模板，然后根据需要将模板应用于尽可能多的主机。

当模板链接到主机时，模板的所有实体（监控项，触发器，图表，...）都将添加到主机。模板直接分配给每个单独的主机（而不是主机组）。

模板通常用于为特定服务或应用程序（如 Apache，MySQL，PostgreSQL，Postfix ...）分组实体，然后应用于运行这些服务的主机。

使用模板的另一个好处是当所有主机都需要更改时。只需要在模板上更改某些内容将会将更改应用到所有链接的主机。

因此，使用模板是减少工作量并简化 Zabbix 配置的好方法。

在创建和配置模板中继续。

9 开箱即用的模板

概述

Zabbix 致力于提供越来越多有用的开箱即用模板列表。开箱即用的模板已预先配置，这是加速监控作业部署的有效方法。

模板获取来源：

- 在新安装的 Zabbix 中 - 前往配置 → 模板;
- 如果是从旧版本升级的 Zabbix，你可以在下载的最新版本的 Zabbix 的 `templates` 目录中找到模板文件。在经过手工导入这些模板文件后，可以在配置 → 模板页找到模板。
- 可以从Zabbix git repository下载模板文件。(请确认模板文件与您的 Zabbix 兼容)

请使用侧边栏目录访问特定类型模板及操作须知。

参考资料:

- 导入模板
- 链接模板

HTTP 模板操作

使用HTTP agent方式收集模板指标数据的正确操作步骤:

1. 在 Zabbix 中创建一个主机，指定监控目标的 IP 地址或 DNS 名称为主接口。这需要宏 {HOST.CONN} 在模板项中正确的解析。
2. 将模板链接 到步骤 1 中创建的主机 (如果模板在 Zabbix 安装中不可用，您可能需要首先导入模板的 `.xml` 文件 - 参见开箱即用的模板说明).
3. 根据需要调整必配宏的值。
4. 配置被监视的主机实例，以允许与 Zabbix 共享数据 - 请参阅 附加步骤/注释列 3 中的说明。

> **Note:**
> 此页仅包含正确模板操作所需的最小宏的集和设置步骤。在模板的 `Readme.md` 文件中（可通过单击模板名称进行访问），可查看模板的详细描述，包括宏、监控项和触发器的完整列表。

模板	必配宏	附加步骤/注释

Apache by HTTP	**{$APACHE.STATUS.HOST}** - - Apache 状态页的主机名或 IP 地址 (默认: 127.0.0.1).
	{$APACHE.STATUS.PATH} -URL 路径 (默认: server-status?auto).
	{$APACHE.STATUS.PORT} - Apache 状态页的端口号 (默认: 80).
	{$APACHE.STATUS.SCHEME} -请求协议. 支持: http (默认), https.
		Apache 模块 mod_status 需要设置 (详情参见 Apache 文档).
		可用性检查, 执行:
		`httpd -M 2>/dev/null | grep status_module`
		
		Apache 配置样例:
		`<Location "/server-status">`
		`SetHandler server-status`
		`Require host example.com`
		`</Location>`

Asterisk by HTTP	**{$AMI.PORT}** - 检测服务可用性的 AMI 端口号 (默认：8088). **{$AMI.SECRET}** - Asterisk Manager 的 secret(默认: zabbix). **{$AMI.URL}** - Asterisk Manager 的 API URL 格式 `<scheme>://<host>:<port>/<prefix>/rawman` (默认: http://asterisk:8088/asterisk/rawman). **{$AMI.USERNAME}** - the Asterisk Manager 用户名.	1. 启用 mini-HTTP Server. 2. 添加选项 `webenabled=yes` 到配置文件 *manager.conf*. 3. 在 Asterisk 实例中创建 *Asterisk Manager* 用户.
ClickHouse by HTTP	**{$CLICKHOUSE.PORT}** - ClickHouse HTTP 端点的端口号 (默认：8123). **{$CLICKHOUSE.SCHEME}** - 请求协议. 支持: http (默认), https. **{$CLICKHOUSE.USER}**, **{$CLICKHOUSE.PASSWORD}** - ClickHouse 登陆凭据 (默认用户名: zabbix, 密码: zabbix_pass). 如果不需要身份验证，请从监控项的 HTTP agent 类型配置项的请求头中删除.	通过'web' 配置文件创建 ClickHouse 用户，并赋予数据库查询权限 (详情参见 ClickHouse 文档). 请参阅模板的 Readme.md 文件，了解 zabbix.xml 的文件配置.
Cloudflare by HTTP	**{$CLOUDFLARE.API.TOKEN}** - Cloudflare API token (默认: '<change>'). **{$CLOUDFLARE.ZONE_ID}** - Cloudflare 站点 Zone ID (默认: '<change>').	在 Cloudflare 账户下我的个人资料 → *API Tokens* 中获取 Cloudflare API Tokens. 在 Cloudflare 账户下账户主页 → 站点中获取 Zone ID.
DELL PowerEdge R720 by HTTP, DELL PowerEdge R740 by HTTP, DELL PowerEdge R820 by HTTP, DELL PowerEdge R840 by HTTP	**{$API.URL}** - Dell iDRAC Redfish API URL 格式 `<scheme>://<host>:<port>` (默认: < 输入您的 URL 地址 >) **{$API.USER}**, **{$API.PASSWORD}** - Dell iDRAC 登陆凭证 (默认: 不设置).	Dell 服务器上的 iDRAC 接口: 1. 启用 Redfish API . 2. 创建一个用户监控用户，赋予只读权限.
Elasticsearch Cluster by HTTP	**{$ELASTICSEARCH.PORT}** - Elasticsearch 主机的端口号 (默认: 9200). **{$ELASTICSEARCH.SCHEME}** - 请求协议. 支持: http (默认), https. **{$ELASTICSEARCH.USERNAME}**, **{$ELASTICSEARCH.PASSWORD}** - 登陆凭证, 仅当用于 Elasticsearch 身份验证时才需要.	

模板	必配宏	附加步骤/注释

Etcd by HTTP	**{$ETCD.PORT}**- Etcd API 端点的端口号 (默认: 2379).
	{$ETCD.SCHEME} - 请求协议. 支持: http (默认), https.
	{$ETCD.USER}**, **{$ETCD.PASSWORD} - 登陆凭证, 仅当用于 Etcd 身份验证时才需要.
		端点的接口路径/metrics 采集指标; 指定 API 端点的接口路径，请使用 `--listen-metrics-urls` 参数 (详情参见 Etcd 文档).
		
		本地验证 Etcd 指标采集，执行:
		`curl -L http://localhost:2379/metrics`
		
		检查 Etcd 指标可以从 Zabbix proxy 或 Zabbix server 采集，执行:
		`curl -L http:%%//<etcd_node_adress>:2379/metrics%%`
		
		每个 Etcd 的节点都需要链接这个模板.

GitLab by HTTP	**{$GITLAB.PORT}** - GitLab Web 端点的端口号 (默认: 80) **{$GITLAB.URL}** - GitLab 实例 URL (默认: localhost)	该模板只适用与本地 GitLab 实例; 指标采集路径为 `/metrics` . 需要设置客户 IP 白名单，才能访问指标 (详情参见 GitLab 文档). 请注意，某些指标可能不适用于特定的 GitLab 实例版本和配置.
Hadoop by HTTP	**{$HADOOP.NAMENODE.HOST}** - Hadoop NameNode 节点的主机 IP 或 FQDN (默认: NameNode). **{$HADOOP.NAMENODE.PORT}** - Hadoop NameNode 节点的 web-UI 端口号 (默认: 9870). **{$HADOOP.RESOURCEMANAGER.HOST}** - Hadoop ResourceManager 节点的 IP 地址或 FQDN (默认: ResourceManager). **{$HADOOP.RESOURCEMANAGER.PORT}** - Hadoop ResourceManager 的 web-UI 端口号 (默认: 8088).	采用 HTTP agent 和 JSONPath 预处理方式远程轮询 Hadoop API 来收集指标数据。Zabbix 服务器（或代理）完成对 ResourceManager、NodeManager、NameNode、DataNodes API 的直接请求处理.
HAProxy by HTTP	**{$HAPROXY.STATS.PATH}** - HAProxy 状态页的路径 (默认: stats). **{$HAPROXY.STATS.PORT}** - HAProxy 状态页主机或容器的端口号 (默认: 8404). **{$HAPROXY.STATS.SCHEME}** - 请求协议. 支持: http (默认), https.	需设置 HAProxy 状态页 (详情参见 HAProxy 博客文档 或模板的 Readme.md 配置样例).
HashiCorp Vault by HTTP	**{$VAULT.API.PORT}** - Vault API 请求的监听端口 (默认: 8200). **{$VAULT.API.SCHEME}** - Vault API 请求的协议. 支持: http (默认), https. **{$VAULT.HOST}** - Vault 主机名 (默认: < 填写您的 Vaultu 主机名 >). **{$VAULT.TOKEN}** - Vault 认证令牌 (默认: < 填写您的 TOKEN>).	1. 配置 Vault API (详情参见 官方文档). 2. 创建 Vault 服务的 token, 并将其赋值于宏 {$VAULT.TOKEN}.
Hikvision camera by HTTP	**{$HIKVISION_ISAPI_PORT}** - 设备上的 ISAPI 端口 (默认: 80). **{$USER}**, **{$PASSWORD}** - camera 登陆凭证 (默认用户: admin, 密码: 1234).	
InfluxDB by HTTP	**{$INFLUXDB.API.TOKEN}** - InfluxDB API 授权 token (默认: ‘’). **{$INFLUXDB.URL}** - InfluxDB 实例 URL 格式 `<scheme>://<host>:<port>` (默认: http://localhost:8086).	此模板从本地 InfluxDB 实例/metrics 路径中采集 InfluxDB 服务指标. 详情参见 InfluxDB 文档.

模板	必配宏	附加步骤/注释
Jenkins by HTTP	**{$JENKINS.API.KEY}** - 访问 Metrics Servlet 的 API 秘钥; 采集通用指标时必须配置 (默认: “). **{$JENKINS.API.TOKEN}** - 用于 HTTP BASIC 授权的 API token; 监控执行节点和构建过程所需 (默认: “). **{$JENKINS.URL}** - Jenkins URL 格式 <scheme>://<host>:<port>; 监控执行节点和构建过程所需 (默认: “). **{$JENKINS.USER}** - HTTP BASIC 授权用户; 监控执行节点和构建过程所需 (默认: zabbix).	通过请求 Metrics API 采集指标. 通用指标: 安装配置 Metrics 插件，配置参数参考 官方文档. 发行一个 API 秘钥用于接入 Metrics Servlet，并将其赋值于宏 {$JENKINS.API.KEY}. 监控执行节点和构建过程所需: 创建一个 API token 用于 Jenkins 监控用户，并将其赋值于宏 {$JENKINS.API.TOKEN}. 详情参见 Jenkins 文档.
Kubernetes API server by HTTP	**{$KUBE.API.SERVER.URL}** - 实例 URL (默认: http://localhost:8086/metrics). **{$KUBE.API.TOKEN}** - API 授权 token (默认: “).	模板要求 Kubernetes 集群中必须安装 Zabbix Helm Chart. 内部指标将从/metrics 中采集. 使用 Bearer API token 授权方式. 详情参见 Kubernetes 文档.
Kubernetes Controller manager by HTTP	**{$KUBE.CONTROLLER.SERVER.URL}** - 实例 URL (默认: http://localhost:10252/metrics). **{$KUBE.CONTROLLER.TOKEN}** - API authorization token (默认: “).	模板要求 Kubernetes 集群中必须安装 Zabbix Helm Chart . 内部指标将从/metrics 中采集. 使用 Bearer API token 授权方式. 详情参见 Kubernetes 文档.
Kubernetes kubelet by HTTP	**{$KUBE.KUBELET.URL}** - instance URL (默认: https://localhost:10250). **{$KUBE.API.TOKEN}** - API 授权 token (默认: “).	模板要求 Kubernetes 集群中必须安装 Zabbix Helm Chart. 内部指标将从/metrics 中采集. 使用 Bearer API token 授权方式. 详情参见 Kubernetes 文档.
Kubernetes nodes by HTTP	**{$KUBE.API.ENDPOINT}** - Kubernetes API 访问地址格式 <scheme>://<host>:<port>/api (默认: 不设置). **{$KUBE.API.TOKEN}** - API 授权 token (默认: “).	模板要求 Kubernetes 集群中必须安装 Zabbix Helm Chart. 生成服务账户 token, 执行: `kubectl get secret zabbix-service-account -n zabbix -o jsonpath={.data.token} | base64 -d` 详情参见 Kubernetes 文档. 模板中包含有一些额外的宏，可用于过滤被发现的工作节点的某些指标.
Kubernetes Scheduler by HTTP	**{$KUBE.SCHEDULER.SERVER.URL}** - 实例 URL (默认: http://localhost:10251/metrics). **{$KUBE.SCHEDULER.TOKEN}** - Scheduler API 授权 token (默认: “).	模板要求 Kubernetes 集群中必须安装 Zabbix Helm Chart. 内部指标将从/metrics 中采集. 使用 Bearer API token 授权方式. 详情参见 Kubernetes 文档.
Kubernetes cluster state by HTTP	**{$KUBE.API.HOST}** - Kubernetes API 服务器 (默认: 不设置). **{$KUBE.API.PORT}**- Kubernetes API 端口 (默认:6443). **{$KUBE.API.TOKEN}** - API 授权 token (默认: “).	模板要求 Kubernetes 集群中必须安装 Zabbix Helm Chart. 内部服务指标将从 kube-state-metrics 端点中采集. 使用 Bearer API token 授权方式. 详情参见 Kubernetes 文档. 模板中包含有一些额外的宏，可用于过滤被发现的工作节点的某些指标.
Microsoft SharePoint by HTTP	**{$SHAREPOINT.URL}** - 门户页面 URL, 例如 *http://sharepoint.companyname.local/* (默认: “). **{$SHAREPOINT.ROOT}** - 根目录; 只监控根目录及其所有子文件夹 (默认: /Shared Documents) **{$SHAREPOINT.USER}**, **{$SHAREPOINT.PASSWORD}** - SharePoint 登陆凭证 (默认: 不设置).	模板中包含有一些额外的宏，可用于在 LLD 中过滤掉某些字典和类型 (可用于过滤的宏的说明请参见模板的 Readme.md).

模板	必配宏	附加步骤/注释
NetApp AFF A700 by HTTP	**{$URL}** - AFF700 集群 URL 地址 (默认: ' ') **{$USERNAME}**, **{$PASSWORD}** - AFF700 登陆凭证 (默认: 不设置).	创建 AFF A700 主机，Zabbix 客户端接口配置集群管理 IP.
NGINX by HTTP	**{$NGINX.STUB_STATUS.HOST}** - NGINX stub_status 主机或者容器的主机名或 IP 地址 (默认: localhost). **{$NGINX.STUB_STATUS.PATH}** - NGINX stub_status 页面访问路径 (默认: basic_status). **{$NGINX.STUB_STATUS.PORT}** - NGINX stub_status 主机或容器监听端口 (默认: 80). **{$NGINX.STUB_STATUS.SCHEME}** - 请求协议. 支持: http (默认), https.	'ngx_http_stub_status_module 需要被设置 (配置样例详情参见 NGINX 文档 或模板的 Readme.md 文件). 可用性检查, 执行: `nginx -V 2>&1 | grep -o with-http_stub_status_module`
NGINX Plus by HTTP	**{$NGINX.API.ENDPOINT}** - NGINX Plus API URL 访问格式 `<scheme>://<host>:<port>/<location>/` (默认: ' ').	1. 启用 NGINX Plus API (详情参见 NGINX 文档). 2. 设置宏 {$NGINX.API.ENDPOINT} 3. 如果需要, 使用其他模板的宏过滤自动发现操作，仅需配置 zones 和 upstreams.
OpenWeatherMap by HTTP	**{$OPENWEATHERMAP.API.ENDPOINT}** - OpenWeatherMap API endpoint (默认: api.openweathermap.org/data/2.5/weather) **{$OPENWEATHERMAP.API.TOKEN}** - OpenWeatherMap API key (默认:' ') **{$LOCATION}** - 要检索其指标的位置 (默认: Riga)	有关获取 API 密钥的说明，参考 OpenWeatherMap 文档。 **{$LOCATION}** 宏支持如下格式： *geocoordinates* - 例如，56.95,24.0833 *location name* - 例如，Chicago *OpenWeatherMap location ID* - 下载 ID 列表 *zip/postal code with a country code* - 例如，94040,us 使用 | 分隔符来指定多个 location。 例如： 43.81821,7.76115|Riga|2643743|94040,us

PHP-FPM by HTTP	**{$PHP_FPM.HOST}** - PHP-FPM 主机或容器的主机名或 IP 地址 (默认: localhost).
	{$PHP_FPM.PING.PAGE} - PHP-FPM ping 页面访问路径 (默认:ping).
	{$PHP_FPM.PORT} - PHP-FPM 主机或容器的端口号 (默认: 80).
	{$PHP_FPM.PROCESS_NAME} - PHP-FPM 进程名 (默认: php-fpm).
	{$PHP_FPM.SCHEME} - 请求协议. 支持: http (默认), https.
	{$PHP_FPM.STATUS.PAGE} - PHP-FPM 状态页面访问路径 (默认:status).
		1. 打开配置文件并启用状态页:
		`pm.status_path = /status`
		`ping.path = /ping`
		
		2. 语法验证: `$ php-fpm7 -t`
		
		3. 重新加载 php-fpm 服务.
		
		4. 在 Nginx 配置文件 Server 配置块 (虚拟主机) 中, 添加 (带注释的扩展样例参见模板的 *Readme.md*):
		`location ~ ^/(status|ping)\$ {`
		`access_log off;`
		`fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;`
		`fastcgi_index index.php;`
		`include fastcgi_params;`
		`fastcgi_pass 127.0.0.1:9000;`
		`}`
		
		5. 语法检查: `$ nginx -t`
		
		6. 重新加载 Nginx
		
		7. 检查状态页: `curl -L 127.0.0.1/status`

Proxmox VE by HTTP	**{$PVE.TOKEN.ID}** - API token 允许对 REST API 的大多数部分进行无状态访问 (默认: 未设置)。
	{$PVE.TOKEN.SECRET} - secret key (默认: 未设置).
	{$PVE.URL.PORT} - 服务器侦听的端口 (默认：8006)。
		创建一个单独的用户进行监控，然后为此用户生成 API token。
		向 token 和用户授予以下访问级别:
		`Check: ["perm","/",["Sys.Audit"]]`
		`Check: ["perm","/nodes/{node}",["Sys.Audit"]]`
		`Check: ["perm","/vms/{vmid}",["VM.Audit"]]`

模板	必配宏	附加步骤/注释

RabbitMQ cluster by HTTP	**{$RABBITMQ.API.CLUSTER_HOST}** - RabbitMQ 集群 API 端点的主机名或 IP 地址 (默认: 127.0.0.1).
	{$RABBITMQ.API.SCHEME} - 请求协议. 支持: http (默认), https.
	{$RABBITMQ.API.USER}**, **{$RABBITMQ.API.PASSWORD} - RabbitMQ 登陆凭证 (默认用户: zbx_monitor, 密码: zabbix).
		启用 RabbitMQ 管理插件 (参见RabbitMQ 文档).
		
		创建一个 RabbitMQ 监控用户，赋予必要的权限, 执行:
		" rabbitmqctl add_user zbx_monitor <PASSWORD> "
		`rabbitmqctl set_permissions -p / zbx_monitor %% "" "" ".*"%%`
		`rabbitmqctl set_user_tags zbx_monitor monitoring`
		
		单节点安装时, 可将集群模板分配给已安装节点模板的主机。如果群集由多个节点组成，建议将群集模板分配给一个单独的负载主机使用.

TiDB by HTTP	**{$TIDB.PORT}** - TiDB 服务器指标信息 web 端点的端口 (默认: 10080) **{$TIDB.URL}** - TiDB 服务器 UR (默认: localhost).	模板适用于 PingCAP TiDB 集群的 TiDB 服务器. 内部指标将从 TiDB /metrics 和 TiDB 监控 API 中采集.
TiDB PD by HTTP	**{$TIDB.PORT}** - TiDB 服务器指标信息 web 端点的端口 (默认: 2379) **{$TIDB.URL}** - TiDB 服务器 URL (默认: localhost).	模板适用于 PingCAP TiDB 集群的 PD 服务器. 内部指标将从 PD /metrics 和 TiDB 监控 API 中采集.
TiDB TiKV by HTTP	**{$TIDB.PORT}** - TiDB 服务器指标信息 web 端点的端口 (默认: 20180) **{$TIDB.URL}** - TiDB 服务器 UR (默认: localhost).	模板适用于 PingCAP TiDB 集群的 TiKV 服务器. 内部指标将从 TiKV /metrics 中采集.
Travis CI by HTTP	**{$TRAVIS.API.TOKEN}** - Travis API Token (默认: 不设置) **{$TRAVIS.API.URL}** - Travis API URL (默认: api.travis-ci.com).	可在 用于 → 设置 → *API* 授权查找 Travis API token. {$TRAVIS.API.URL} 私有项目格式为 *api.travis-ci.com*. {$TRAVIS.API.URL} 企业项目格式为 *api.example.com* (将 example.com 替换成 Travis CI 所在的域名).
VMWare SD-WAN VeloCloud by HTTP	**{$VELOCLOUD.TOKEN}** - VMware SD-WAN Orchestrator API Token (默认: "). **{$VELOCLOUD.URL}** - VMware SD-WAN Orchestrator URL, 例如, *velocloud.net* (默认: ").	必须在 VMware SD-WAN Orchestrator 中创建 API token (详情参见VMware 文档).
ZooKeeper by HTTP	**{$ZOOKEEPER.COMMAND_URL}** - admin.commandURL; 相对根 URL，该 URL 用于列出和发出命令 (默认: commands). **{$ZOOKEEPER.PORT}** - admin.serverPort; 内置 Jetty 服务的侦听端口号 (默认: 8080). **{$ZOOKEEPER.SCHEME}** - 请求协议. 支持: http (默认), https.	通过对 AdminServer 发出请求，可从每个 Zookeep 节点采集指标 (默认启用). 启用或配置 AdminServer 参见 ZooKeeper 文档.

IPMI 模板操作

IPMI 模板不需要任何特定的设置. 要开始监控，请将模板链接 到目标主机。(如果 Zabbix 安装中没有该模板，则可能需要先导入该模板的.xml 文件 - 有关说明，请参见开箱即用的模板 部分)。

> **Note:**
> 页面仅包含最小的一组宏和正确的模板操作所需的设置步骤。在模板的 Readme.md 文件中提供了模板的详细说明，包括宏，项和触发器的完整列表（可通过单击模板名称访问）。

模板	必配宏	附加步骤/注释
Chassis by IPMI	**{$IPMI.USER}**, **{$IPMI.PASSWORD}** - 访问 BMC 的凭据 (默认: none)	-

JMX 模板操作

确保通过JMX收集模板指标的正确步骤:

1. 确保已正确安装和设置 Zabbix Java 网关 。
2. 链接 模板到目标主机。主机应设置 JMX 接口。
如果模板在您的 Zabbix 中不可用，您可能需要先导入摸板文件.xml - 查看开箱即用的模板 说明部分。3. 根据需要调整必配宏的值。
4. 配置要监控的实例允许与 Zabbix 共享数据- 请参阅 附加步骤/注释字段。

> **Note:**
> 该页面仅包含最小的一组宏和正确的模板操作所需的设置步骤。在模板的 Readme.md 文件中提供了模板的详细说明，包括宏，项和触发器的完整列表（可通过单击模板名称访问）。

模板	必配宏	附加步骤/注释
Apache ActiveMQ by JMX	**{$ACTIVEMQ.PORT}** - JMX 端口 (默认: 1099). **{$ACTIVEMQ.USERNAME}**, **{$ACTIVEMQ.PASSWORD}** - JMX 登录凭证 (默认用户名: admin, 密码: activemq).	根据 官方文档 启用和配置 JMX 访问 Apache ActiveMQ.
Apache Cassandra by JMX	**{$CASSANDRA.USER}**, **{$CASSANDRA.PASSWORD}** - Apache Cassandra 登录凭据 (默认用户名: zabbix, 密码: zabbix)	根据 官方文档 启用和配置 JMX 访问 Apache Cassandra.
Apache Kafka by JMX	**{$KAFKA.USER}**, **{$KAFKA.PASSWORD}** - Apache Kafka 登录凭据 (默认用户名: zabbix, 密码: zabbix)	根据 官方文档 启用和配置 JMX 访问 Apache Kafka .
Apache Tomcat by JMX	**{$TOMCAT.USER}**, **{$TOMCAT.PASSWORD}** - Apache Tomcat 登录凭据; leave blank if Tomcat installation does not require authentication (默认: 不设置).	根据 官方文档 (请选择正确的版本) 启用和配置 JMX 访问 Apache Tomcat.
GridGain by JMX	**{$GRIDGAIN.USER}**, **{$GRIDGAIN.PASSWORD}** - GridGain 登录凭据 (默认用户名: zabbix, 密码: <secret>).	根据 文档 启用和配置 JMX 访问 GridGain In-Memory Computing Platform.
Ignite by JMX	**{$IGNITE.USER}**, **{$IGNITE.PASSWORD}** - Apache Ignite 登录凭据 (默认用户名: zabbix, 密码: <secret>).	启动和配置 JMX 访问 Apache Ignite. 默认情况下，JMX 树层次结构包含类加载器. 添加下列 Java 虚拟机参数`-DIGNITE_MBEAN_APPEND_CLASS_LOADER_ID=false` 将排除一个具有类加载器名称的级别. Cache 和 Data Region 指标需配置 - 详情参见 Ignite 文档. See also: Monitoring and Management Using JMX Technology
WildFly Domain by JMX	**{$WILDFLY.JMX.PROTOCOL}** - JMX 协议 (默认: remote+http) **{$WILDFLY.USER}**, **{$WILDFLY.PASSWORD}** - WildFly 登录凭据 (默认用户名: zabbix, 密码: zabbix).	1. 根据 官方文档 启用和配置 JMX 访问 WildFly. 2. 从`/(wildfly,EAP,Jboss,AS)/bin/client` 中复制 *jboss-client.jar* 到 `/usr/share/zabbix-java-gateway/lib` 目录中. 3. 重启 Zabbix Java 网关.
WildFly Server by JMX	**{$WILDFLY.JMX.PROTOCOL}** - JMX 协议 (默认: remote+http) **{$WILDFLY.USER}**, **{$WILDFLY.PASSWORD}** - WildFly 登录凭据 (默认用户名: zabbix, 密码: zabbix).	1. 根据 官方文档启用和配置 JMX 访问 WildFly. 2. Copy 从`/(wildfly,EAP,Jboss,AS)/bin/client` 复制 *jboss-client.jar* 到 `/usr/share/zabbix-java-gateway/lib` 目录中. 3. 重启 Zabbix Java 网关.

ODBC 模板操作

确保通过ODBC monitoring收集模板指标的正确步骤:

1. 确保在 Zabbix 服务器或代理上安装了所需的 ODBC 驱动程序。
2. 链接 模板到目标主机（如果模板在您的 Zabbix 安装中不可用，您可能需要先导入模板的.xml 文件 - 请参阅开箱即用的模板（/manual/config/templates_out_of_the_box）部分的说明）。3. 根据需要调整必需宏的值。如果宏值中的密码包含分号（;），则应该用大括号括起来，参见 [ODBC 监控]（https://www.zabbix.com/documentation/6.0/en/manual/config/items/itemtypes/odbc_checks?hl=ODBC%2Cmonitoring) 了解详情。
3. 配置被监控的实例以允许与 Zabbix 共享数据 - 请参阅 其他步骤/注释列中的说明。

> **Note:**
> 此页面仅包含正确模板操作所需的最小宏集和设置步骤。模板的 Readme.md 文件中提供了模板的详细说明，包括宏、监控项和触发器的完整列表（可通过单击模板名称访问）。

| 模板 | 强制宏 | 附加步骤/注释 | |--------|----------------|-------------------- --| |MSSQL by ODBC|**{$MSSQL.DSN}** - 系统数据源名称（默认：< 填写你的 DSN>）
{$MSSQL.PORT} - Microsoft SQL Server 的 TCP 端口（默认：1433）
{$MSSQL.USER }, {$MSSQL.PASSWORD} - Microsoft SQL 登录凭据（默认值：未设置）| 创建用于监控的 Microsoft SQL 用户并授予用户以下权限：查看服务器状态；查看任何定义 (请参阅 Microsoft SQL 文档 以获得详细信息)。

“服务的 TCP 端口状态” 监控项使用 {HOST.CONN} 和 {$MSSQL.PORT} 宏来检查 Microsoft SQL 实例的可用性。| |MySQL by ODBC|**{$MYSQL.DSN}** - 系统数据源名称（默认：< 填写你的 DSN>）
{$MYSQL.USER}, {$MYSQL.PASSWORD} - MySQL 登录凭据；密码可以为空（默认值：未设置）| 要向将用于监控的 MySQL 用户授予所需的权限，请运行：
GRANT USAGE,REPLICATION CLIENT,PROCESS,SHOW DATABASES,SHOW VIEW ON %% *.* TO '<username>'@'%';%%

 详见MYSQL 文档 .| |Oracle by ODBC|**{$ORACLE.DSN}** - 系统数据源名称（默认：<Put your DSN here>）
{$ORACLE.PORT} - Oracle DB 的 TCP 端口（默认：1521）
{$ORACLE.USER}, {$ORACLE.PASSWORD} - Oracle 登录凭据（默认值：未设置）|1。要创建用于监控的 Oracle 用户，请运行：
CREATE USER zabbix_mon IDENTIFIED BY <PASSWORD>;
-- Grant access to the zabbix_mon user.
GRANT CONNECT, CREATE SESSION TO zabbix_mon;
GRANT SELECT ON V_$instance TO zabbix_mon;
GRANT SELECT ON V_$database TO zabbix_mon;
GRANT SELECT ON v_$sysmetric TO zabbix_mon;
GRANT SELECT ON v$recovery_file_dest TO zabbix_mon;
GRANT SELECT ON v$active_session_history TO zabbix_mon;
GRANT SELECT ON v$osstat TO zabbix_mon;
GRANT SELECT ON v$restore_point TO zabbix_mon;< br>GRANT SELECT ON v$process TO zabbix_mon;
GRANT SELECT ON v$datafile TO zabbix_mon;
G SELECT ON v$pgastat TO zabbix_mon;
GRANT SELECT ON v$ sgastat 到 zabbix_mon;
GRANT SELECT ON v$log 到 zabbix_mon;
GRANT SELECT ON v$archive_dest 到 zabbix_mon;
GRANT SELECT ON v$asm_diskgrou 到 zabbix_mon;
 GRANT SELECT ON sys.dba_data_files TO zabbix_mon;
GRANT SELECT ON DBA_TABLESPACES TO zabbix_mon;
GRAN T SELECT ON DBA_TABLESPACE_USAGE_METRICS TO zabbix_mon;
GRANT SELECT ON DBA_USERS TO zabbix_mon;

2. 确保 ODBC 使用会话参数连接到 Oracle NLS_NUMERIC_CHARACTERS= '.,'

3. 向 odbc.ini 添加一条新记录：
[$ORACLE.DSN]
Driver = Oracle 19 ODBC driver
Servername = $ORACLE.DSN
DSN = $ORACLE .DSN

4. 通过 isql 检查连接：
isql $TNS_NAME $DB_USER $DB_PASSWORD
<b 为 Oracle ENV 使用配置 Zabbix 服务器或 Zabbix 代理。编辑或添加一个新文件：*/etc/sysconfig/zabbix-server*，或者对于代理：*/etc/sysconfig/zabbix-proxy*。然后将以下行添加到文件中：
export ORACLE_HOME=/usr/lib/oracle/19.6/client64
export PATH=$PATH:$ORACLE_HOME/bin
export LD_LIBRARY_PATH=$ ORACLE_HOME/lib:/usr/lib64:/usr/lib:$ORACLE_HOME/b TNS_ADMIN=$ORACLE_HOME/network/admin

6. 重新启动 Zabbix 服务器或代理。

Zabbix agent 2 模板操作

确保通过Zabbix agent 2收集模板指标的正确步骤:

1. 确保主机上安装了 Zabbix agent 2，并且该版本包含所需的插件。有时候，您需要先升级 agent 2。
2. 将模板链接 到目标主机 (如果模板在您的 Zabbix 中不可用，您可能需要先导入模板文件.xml 文件 - 查看开箱即用的模板 说明部分)。
3. 根据需要调整必配宏的值。注意，用户宏可用于覆盖配置参数。
4. 配置要监控的实例允许与 Zabbix 共享数据- 请参阅 附加步骤/注释字段

> **Attention:**
> Zabbix agent 2 模板与插件一起工作。基本配置可以通过简单地调整用户宏来完成，更深层次的定制可以参考插件配置本身实现。例如，如果插件支持命名会话，则可以通过在配置文件中为每个实体指定具有自己的 URI、用户名和密码的命名会话，监视多个相同类型的实体（例如 MySQL1 和 MySQL2）。

> **Note:**
> 该页面仅包含最小的一组宏和正确的模板操作所需的设置步骤。在模板的 Readme.md 文件中提供了模板的详细说明，包括宏，项和触发器的完整列表（可通过单击模板名称访问）。

模板名称	必配宏	附加步骤/注释
Ceph by Zabbix agent 2	**{$CEPH.API.KEY}** - Ceph API 秘钥 (默认: zabbix_pass). 如果 {$CEPH.CONNSTRING} 为 URI，秘钥必须设置。 如果 {$CEPH.CONNSTRING} 为会话名，则值为空. **{$CEPH.CONNSTRING}** - 连接字符串; 会话名或 URI 的格式: < 协议(主机: 端口)>. URI 只支持 HTTPS 协议. 例如: Prod, https://localhost:8003 (默认) **{$CEPH.USER}** - Ceph 监控用户 (默认:zabbix). 如果 {$CEPH.CONNSTRING} 为 URI，用户必须设置.<br 如果 {$CEPH.CONNSTRING} 为会话名，则值为空.	与 *Ceph* 插件一起使用; 支持给会话命名. 1. Ceph RESTful 模块配置，参见文档. 2. 确保 RESTful API 端点连接的可用性.
Docker	-	与 *Docker* 插件一起使用; 不支持给会话命名. 设置 Docker API 端点路径, 在 agent 2 的配置文件 中修改参数 `Plugins.Docker.Endpoint` (默认: `Plugins.Docker.Endpoint=unix:///var/run/docker.sock` 可用性测试, 执行: `zabbix_get -s docker-host -k docker.info`
Memcached	**{$MEMCACHED.CONN.URI}** - URI 连接字符串格式; 端口可选; 密码未使用. 如果不设置, 插件默认值: tcp://localhost:11211. 例子: tcp://127.0.0.1:11211, tcp://localhost, unix:/var/run/memcached.sock.	与 *Memcached* 插件一起使用; 支持给会话命名. 可用性测试, 执行: `zabbix_get -s memcached-host -k memcached.ping`
MongoDB cluster by Zabbix agent 2	**{$MONGODB.CONNSTRING}** - URI 连接字符串格式; 密码未使用 (默认: tcp://localhost:27017). 支持会话名称或以下格式定义的 URI: %% <protocol(host:port)>%% URI 仅支持 TCP 协议. 例子: MongoDB1, tcp://172.16.0.10 **{$MONGODB.USER}**, **{$MONGODB.PASSWORD}** - MongoDB 凭证 (默认: none). 如果未设置，同时 {$MONGODB.CONNSTRING} 为 URI, 配置文件的参数将被使用. 如果 {$MONGODB.CONNSTRING} 为会话名，必须为空.	与 *MongoDB* 插件一起使用; 支持给会话命名. MongoDB 配置说明, 参见插件. 可用性测试, 执行: `zabbix_get -s mongos.node -k 'mongodb.ping["{\$MONGODB.CONNSTRING}","{\$MONGODB.`

模板名称	必配宏	附加步骤/注释
MongoDB node by Zabbix agent 2	**{$MONGODB.CONNSTRING}** - URI 连接字符串格式; 密码未使用 (默认: tcp://localhost:27017). 支持会话名称或以下格式定义的 URI: %% <protocol(host:port)>%% URI 仅支持 TCP 协议. 例子: MongoDB1, tcp://172.16.0.10 **{$MONGODB.USER}**, **{$MONGODB.PASSWORD}** - MongoDB 凭证 (默认: none). 如果未设置，同时 {$MONGODB.CONNSTRING} 为 URI, 配置文件的参数将被使用. 如果 {$MONGODB.CONNSTRING} 为会话名，必须为空.	与 *MongoDB* 插件一起使用; 支持给会话命名. MongoDB 配置说明, 参见参见. 可用性测试, 执行: `zabbix_get -s mongodb.node -k 'mongodb.ping["{\$MONGODB.CONNSTRING}","{\$MONGODB.`
MySQL by Zabbix agent 2	**{$MYSQL.DSN}** - MySQL 实例的系统数据源名称 (默认: < 输入您的 DSN>). 支持会话名称或以下格式定义的 URI: %% <protocol(host:port or /path/to/socket)/>%% URI 支持 TCP 和 Unix 协议. 例子: MySQL1, tcp://localhost:3306, tcp://172.16.0.10, unix:/var/run/mysql.sock **{$MYSQL.USER}**, **{$MYSQL.PASSWORD}** - MySQL 凭证 (默认: none). 如果 {$MYSQL.DSN} 为 URI，必须设置. 如果 {$MYSQL.DSN} 为会话名，必须为空.	与 *MySQL* 插件一起使用; 支持给会话命名. 赋予 MySQL 监控用户必要的权限, 执行: `GRANT USAGE,REPLICATION CLIENT,PROCESS,SHOW DATABASES,SHOW VIEW ON *.* TO '<username>'@'%';` 参见 MySQL 文档查看关于 用户权限 和Unix sockets信息.

模板名称	必配宏	附加步骤/注释

Oracle by Zabbix agent 2	**{$ORACLE.CONNSTRING}** - 连接串; 支持会话名称或以下格式定义的 URI: <protocol(host:port or /path/to/socket)/>
	URI 仅支持 TCP 协议.
	例子: Oracle1, tcp://localhost:1521
	
	**{$ORACLE.SERVICE}** - Oracle 服务名 (默认: ORA). 如果 {$ORACLE.CONNSTRING} 为 URI，必须设置.
	如果 {$ORACLE.CONNSTRING} 为会话名，必须为空.
	
	{$ORACLE.USER}**, **{$ORACLE.PASSWORD} - Oracle 凭证 (默认用户名: zabbix, 密码: zabbix_password). 如果 {$ORACLE.CONNSTRING} 为 URI，必须设置.
	如果 {$ORACLE.CONNSTRING} 为会话名，必须为空.
		与 *Oracle* 插件一起使用; 支持给会话命名.
		
		安装Oracle Instant Client.
		创建并赋予 Oracle 用户必要的权限, 执行:
		`CREATE USER zabbix_mon IDENTIFIED BY <PASSWORD>;`
		`-- Grant access to the zabbix_mon user.`
		`GRANT CONNECT, CREATE SESSION TO zabbix_mon;`
		`GRANT SELECT ON DBA_TABLESPACE_USAGE_METRICS TO zabbix_mon;`
		`GRANT SELECT ON DBA_TABLESPACES TO zabbix_mon;`
		`GRANT SELECT ON DBA_USERS TO zabbix_mon;`
		`GRANT SELECT ON SYS.DBA_DATA_FILES TO zabbix_mon;`
		`GRANT SELECT ON V\$ACTIVE_SESSION_HISTORY TO zabbix_mon;`
		`GRANT SELECT ON V\$ARCHIVE_DEST TO zabbix_mon;`
		`GRANT SELECT ON V\$ASM_DISKGROUP TO zabbix_mon;`
		`GRANT SELECT ON V\$DATABASE TO zabbix_mon;`
		`GRANT SELECT ON V\$DATAFILE TO zabbix_mon;`
		`GRANT SELECT ON V\$INSTANCE TO zabbix_mon;`
		`GRANT SELECT ON V\$LOG TO zabbix_mon;`
		`GRANT SELECT ON V\$OSSTAT TO zabbix_mon;`
		`GRANT SELECT ON V\$PGASTAT TO zabbix_mon;`
		`GRANT SELECT ON V\$PROCESS TO zabbix_mon;`
		`GRANT SELECT ON V\$RECOVERY_FILE_DEST TO zabbix_mon;`
		`GRANT SELECT ON V\$RESTORE_POINT TO zabbix_mon;`
		`GRANT SELECT ON V\$SESSION TO zabbix_mon;`
		`GRANT SELECT ON V\$SGASTAT TO zabbix_mon;`
		`GRANT SELECT ON V\$SYSMETRIC TO zabbix_mon;`
		`GRANT SELECT ON V\$SYSTEM_PARAMETER TO zabbix_mon;`

PostgreSQL Agent 2	**{$PG.URI}** - 连接串; 支持会话名称或以下格式定义的 URI: %% <protocol(host:port or /path/to/socket)/>%%. URI 支持 TCP 和 Unix 协议.
	例子: Postgres1, tcp://localhost:5432, tcp://172.16.0.10
	{$PG.USER}**, **{$PG.PASSWORD} - PostgreSQL 凭证 (默认用户: postgres, 密码:postgres).
	如果 {$PG.URI} 为 URI，必须设置.
	如果 {$PG.URI} 为会话名，必须为空.
		与 *PostgreSQL* 插件一起使用; 支持给会话命名.
		
		针对 PostgreSQL 10 或更新的版本创建并赋予用户必要的权限, 执行:
		`CREATE USER 'zbx_monitor' IDENTIFIED BY '<password>';`
		`GRANT EXECUTE ON FUNCTION pg_catalog.pg_ls_dir(text) TO zbx_monitor;\\GRANT EXECUTE ON FUNCTION pg_catalog.pg_stat_file(text) TO zbx_monitor;`
		
		编辑 pg_hba.conf 设置 Zabbix agent 白名单 (详情参见PostgreSQL 文档).

模板名称	必配宏	附加步骤/注释
Redis	**{$REDIS.CONN.URI}** - URI 连接串格式; 端口为可选项; 密码未使用. 若未设置, 插件默认值: tcp://localhost:6379	与 *Redis* 插件一起使用; 支持给会话命名. 可用性测试, 执行: `zabbix_get -s redis-master -k redis.ping`
SMART by Zabbix agent 2 / SMART by Zabbix agent 2 active	-	执行 Zabbix agent 2 的用户对 smartctl 命令必须能 Sudo/root 提权. 支持 smartctl 的最小版本是 7.1. 磁盘发现 LLD 规则查找所有开启 S.M.A.R.T. 的 HDD、SSD、NVMe 磁盘. 属性发现 LLD 规则查找每个磁盘的所有供应商特定属性. 要跳过某些属性，请在主机级别使用 {$SMART.DISK.NAME.MATCHES} 中的磁盘名称和 {$SMART.ATTRIBUTE.ID.MATCHES} 中的属性 ID 设置正则表达式.
Systemd by Zabbix agent 2	-	无需特定配置.
Website certificate by Zabbix agent 2	**{$CERT.WEBSITE.HOSTNAME}** - 连接 Web 站点的 DNS 名称 (默认: < 输入您的 DNS 名称 >).	与 *WebCertificate* 插件一起使用; 支持给会话命名. 可用性测试, 执行: `zabbix_get -s <zabbix_agent_addr> -k web.certificat e.get[<website_DNS_name>]` 使用 Zabbix 代理接口为 TLS/SSL 证书创建单独的主机并将模板链接到该主机.

Zabbix agent 模板操作

确保通过Zabbix agent收集模板指标的正确步骤:

1. 确保主机上安装了 Zabbix agent。对于主动模式, 还需要确认 agent 配置文件已添加 `ServerActive` 参数。
2. 将模板链接 到目标主机 (如果模板在您的 Zabbix 中不可用，您可能需要先导入模板文件.xml 文件 - 查看开箱即用的模板 说明部分)。
3. 根据需要调整必配宏的值。
4. 配置要监控的实例允许与 Zabbix 共享数据- 请参阅 附加步骤/注释字段。

Note:
该页面仅包含最小的一组宏和正确的模板操作所需的设置步骤。在模板的 Readme.md 文件中提供了模板的详细说明，包括宏，项和触发器的完整列表（可通过单击模板名称访问）。

模板名称	必配宏	附加步骤/注释

Apache by Zabbix agent	**{$APACHE.STATUS.HOST}** - Apache 状态页面的主机名或 IP 地址 (默认: 127.0.0.1)
	{$APACHE.STATUS.PATH} - URL 路径 (默认: server-status?auto)
	{$APACHE.STATUS.PORT} - Apache 状态页面的端口 (默认: 80)
		Apache 模块 `mod_status` 应该设置 (详情参见 Apache 文档).
		可用性检查, 执行:
		`httpd -M 2>/dev/null | grep status_module`
		
		Apache 配置样例:
		`<Location "/server-status">`
		`SetHandler server-status`
		`Require host example.com`
		`</Location>`

HAProxy by Zabbix agent	**{$HAPROXY.STATS.PATH}** - HAProxy 统计页面的路径 (默认: stats) **{$HAPROXY.STATS.PORT}** - HAProxy 的端口统计主机或容器 (默认: 8404) **{$HAPROXY.STATS.SCHEME}** - HAProxy 状态页面支持的协议. 支持: http (默认), https	HAProxy 状态页需要设置 (配置案例详情参见 HAProxy博文 或模板的 Readme.md 文件).

模板名称	必配宏	附加步骤/注释
IIS by Zabbix agent / IIS by Zabbix agent active	**{$IIS.PORT}** - IIS Server 监听端口 (默认: 80) **{$IIS.SERVICE}** - 端口检查服务 (默认: http). 详情参见net.tcp.service部分.	IIS 服务器应具有以下角色: Web Server IIS Management Scripts and Tools 详情参见 IIS 文档.
Microsoft Exchange Server 2016 by Zabbix agent/Microsoft Exchange Server 2016 by Zabbix agent active		请注意，该模板不提供有关 Windows 服务状态的信息。建议与 *OS Windows by Zabbix agent* 或 *OS Windows by Zabbix agent active* 模板一起使用。
Nginx by Zabbix agent	**{$NGINX.STUB_STATUS.HOST}** - NGINX stub_status 主机或者容器的主机名或 IP 地址 (默认: localhost) **{$NGINX.STUB_STATUS.PATH}** - NGINX stub_status 页面访问路径 (默认: basic_status) **{$NGINX.STUB_STATUS.PORT}** - NGINX stub_status 主机或容器监听端口 (默认: 80)	ngx_http_stub_status_module 需要被设置 (配置样例详情参见 Nginx 文档或模板的 Readme.md 文件). 可用性检查, 执行: `nginx -V 2>&1 | grep -o with-http_stub_status_module`

PHP-FPM by Zabbix agent	**{$PHP_FPM.HOST}** - PHP-FPM 主机或容器的主机名或 IP 地址 (默认: `localhost`)
	{$PHP_FPM.PING.PAGE} - PHP-FPM ping 页面访问路径 (默认:`ping`)
	{$PHP_FPM.PORT} - PHP-FPM 主机或容器的端口号 (默认: 80)
	{$PHP_FPM.PROCESS_NAME} - PHP-FPM 进程名 (默认:`php-fpm`)
	{$PHP_FPM.STATUS.PAGE} - PHP-FPM 状态页面访问路径 (默认:`status`)
		1. 打开配置文件并启用状态页:
		`pm.status_path = /status`
		`ping.path = /ping`
		
		2. 语法验证: `\$ php-fpm7 -t`
		
		3. 重新加载 php-fpm 服务.
		
		4. 在 Nginx 配置文件 Server 配置块 (虚拟主机) 中, 添加 (带注释的扩展样例参见模板的 *Readme.md*):
		`location ~ ^/(status|ping)\$ {`
		`access_log off;`
		`fastcgi_param SCRIPT_FILENAME \$document_root\$fastcgi_script_name;`
		`fastcgi_index index.php;`
		`include fastcgi_params;`
		`fastcgi_pass 127.0.0.1:9000;`
		`}`
		
		5. 语法检查: `\$ nginx -t`
		
		6. 重新加载 Nginx
		
		7. 检查状态页: `curl -L 127.0.0.1/status`

RabbitMQ cluster by Zabbix agent	**{$RABBITMQ.API.CLUSTER_HOST}** - RabbitMQ 集群 API 端点的主机名或 IP 地址 (默认:`127.0.0.1`)
	{$RABBITMQ.API.USER}**, **{$RABBITMQ.API.PASSWORD} - RabbitMQ 登陆凭证 (默认用户: zbx_monitor, 密码: zabbix)
		启用 RabbitMQ 管理插件 (参见 RabbitMQ 文档).
		
		创建一个 RabbitMQ 监控用户，赋予必要的权限, 执行:
		`" rabbitmqctl add_user zbx_monitor <PASSWORD> "`
		`rabbitmqctl set_permissions -p / zbx_monitor %% "" "" ".*"%%`
		`rabbitmqctl set_user_tags zbx_monitor monitoring`
		
		单节点安装时, 可将集群模板分配给已安装节点模板的主机。如果群集由多个节点组成，建议将群集模板分配给一个单独的负载主机使用.

模板名称	必配宏	附加步骤/注释

MySQL by Zabbix agent	**{$MYSQL.HOST}** - MySQL 主机或容器的主机名或 IP 地址 (默认: localhost)
	{$MYSQL.PORT} - 数据库服务端口 (默认: 3306)
		1. 如有必要，将 mysql 和 mysqladmin 的路径添加到全局环境变量 PATH.
		
		2. 将 Zabbix *templates* 目录下的 `template_db_mysql.conf` 文件复制到 Zabbix agent 子配置文件目录中 (默认为/etc/zabbix/zabbix_agentd.d/)，并重启 Zabbix agent.
		
		3. 创建 MySQL 用户 `zbx_monitor`. 赋予该用户必要的权限, 执行:
		`GRANT USAGE,REPLICATION CLIENT,PROCESS,SHOW DATABASES,SHOW VIEW ON %% *.* TO '<username>'@'%';%%`
		(详情参见 MYSQL 文档).
		
		4. 在 Linux 主机中，Zabbix agent 需要在 home 目录下创建 `.my.cnf` (默认为/var/lib/zabbix) 或在 Windows 主机的 c:\中创建 `my.cnf`. 该文件必须包含三行字符串:
		`[client]`
		”user=’zbx_monitor’ ”
		”password=’<password>’ ”

PostgreSQL	**{$PG.DB}** - 连接到服务器的数据库名称 (默认: postgres)
	{$PG.HOST} - 数据库服务器主机或套接字目录 (默认:127.0.0.1)
	{$PG.PORT} - 数据库服务器监听端口 (默认: 5432)
	{$PG.USER} - 数据库用户名 (默认: zbx_monitor)
		1. 创建一个只读用户 `zbx_monitor` 以访问 PostgreSQL 服务器。对于 PostgreSQL 10 和更新版本，执行:
		`CREATE USER zbx_monitor WITH PASSWORD '<PASSWORD>' INHERIT;`
		`GRANT pg_monitor TO zbx_monitor;`
		对于老的 PostgreSQL 版本，执行:
		`CREATE USER zbx_monitor WITH PASSWORD '<PASSWORD>';`
		`GRANT SELECT ON pg_stat_database TO zbx_monitor;`
		
		2. 复制 `postgresql/` 至 Zabbix agent home 目录 (/var/lib/zabbix/).
		
		3. 将 Zabbix *templates* 目录下的 `template_db_postgresql.conf` 文件复制到 Zabbix agent 子配置文件目录中 (/etc/zabbix/zabbix_agentd.d/) 并重启 Zabbix agent.
		
		4. 编辑 `pg_hba.conf` 为 Zabbix agent 设置白名单 (详情参见 PostgreSQL 文档).
		例子:
		`host all zbx_monitor 127.0.0.1/32 trust`
		`host all zbx_monitor 0.0.0.0/0 md5`
		`host all zbx_monitor ::0/0 md5`
		
		5. 监控一个远程服务器, 在创建 Zabbix agent home 目录 (/var/lib/zabbix/) 中创建 `.pgpass` 文件，并在每一行中添加实例、端口、数据库、用户、密码信息 (详情参见 PostgreSQL 文档).
		例子:
		`<REMOTE_HOST1>:5432:postgres:zbx_monitor:<PASSWORD>`
		`*:5432:postgres:zbx_monitor:<PASSWORD>`

网络设备的标准化模板

概述

为了提供交换机和路由器等网络设备的监控，我们创建了两个所谓的模型：网络设备本身（基本上是它的机框）和网络接口

从 Zabbix 3.4 开始提供了许多网络设备系列模板。所有模板都覆盖（尽可能从设备中获取这些监控项）：

- 机框故障监控（电源，风扇和温度，总体状态）
- 机框性能监控（CPU 和内存监控项）
- 机框资产收集（序列号，型号名称，固件版本）
- 使用 IF-MIB 和 EtherLike-MIB 进行网络接口监控（接口状态，接口流量负载，以太网的双工状态）

这些模板获取来源：

- 在新安装的 Zabbix 中 - 前往配置 → 模板;
- 如果是从旧版本升级的 Zabbix，你可以在下载的最新版本的 Zabbix 的 `templates` 目录中找到模板文件。在经过手工导入这些模板文件后，可以在配置 → 模板页找到模板。

如果要导入新的开箱即用模板，您可能还需要将 `@Network` 自动发现接口全局正则表达式更新为：

```
Result is FALSE: ^Software Loopback Interface
Result is FALSE: ^(In)?[lL]oop[bB]ack[0-9._]*$
Result is FALSE: ^NULL[0-9.]*$
Result is FALSE: ^[lL]o[0-9.]*$
Result is FALSE: ^[sS]ystem$
Result is FALSE: ^Nu[0-9.]*$
```

更新后，会过滤掉在大多数系统上环回和空接口。

设备

可用模板的设备系列列表：

模板名称	厂商	设备系列	已知模型	操作系统	使用的 MIB 库	标签
Alcatel Timetra TiMOS SNMP	Alcatel	Alcatel Timetra	ALCATEL SR 7750	TiMOS	TIMETRA-SYSTEM-MIB,TIMETRA-CHASSIS-MIB	Certified
Brocade FC SNMP	Brocade	Brocade FC switches	Brocade 300 SAN Switch-	-	SW-MIB,ENTITY-MIB	Performance, Fault
Brocade_Foundry Stackable SNMP	Brocade	Brocade ICX	Brocade ICX6610, Brocade ICX7250-48, Brocade ICX7450-48F		FOUNDRY-SN-AGENT-MIB, FOUNDRY-SN-STACKING-MIB	Certified
Brocade_Foundry Nonstack-able SNMP	Brocade Foundry	Brocade MLX, Foundry	Brocade MLXe, Foundry FLS648, Foundry FWSX424		FOUNDRY-SN-AGENT-MIB	Performance, Fault

模板名称	厂商	设备系列	已知模型	操作系统	使用的 MIB 库	标签
Cisco Catalyst 3750<device model> SNMP	Cisco	Cisco Catalyst 3750	Cisco Catalyst 3750V2-24FS, Cisco Catalyst 3750V2-24PS, Cisco Catalyst 3750V2-24TS, Cisco Catalyst SNMP, Cisco Catalyst SNMP		CISCO-MEMORY-POOL-MIB, IF-MIB, EtherLike-MIB, SNMPv2-MIB, CISCO-PROCESS-MIB, CISCO-ENVMON-MIB, ENTITY-MIB	Certified
Cisco IOS SNMP	Cisco	Cisco IOS ver > 12.2 3.5	Cisco C2950	IOS	CISCO-PROCESS-MIB,CISCO-MEMORY-POOL-MIB,CISCO-ENVMON-MIB	Certified
Cisco IOS versions 12.0_3_T-12.2_3.5 SNMP	Cisco	Cisco IOS > 12.0 3 T and 12.2 3.5	-	IOS	CISCO-PROCESS-MIB,CISCO-MEMORY-POOL-MIB,CISCO-ENVMON-MIB	Certified
Cisco IOS prior to 12.0_3_T SNMP	Cisco	Cisco IOS 12.0 3 T	-	IOS	OLD-CISCO-CPU-MIB,CISCO-MEMORY-POOL-MIB	Certified
D-Link DES_DGS Switch SNMP	D-Link	DES/DGX switches	D-Link DES-xxxx/DGS-xxxx,DLINK DGS-3420-26SC	-	DLINK-AGENT-MIB,EQUIPMENT-MIB,ENTITY-MIB	Certified
D-Link DES 7200 SNMP	D-Link	DES-7xxx	D-Link DES 7206	-	ENTITY-MIB,MY-SYSTEM-MIB,MY-PROCESS-MIB,MY-MEMORY-MIB	Performance Fault Interfaces

模板名称	厂商	设备系列	已知模型	操作系统	使用的 MIB 库	标签
Dell Force S-Series SNMP	Dell	Dell Force S-Series	S4810		F10-S-SERIES-CHASSIS-MIB	Certified
Extreme Exos SNMP	Extreme	Extreme EXOS	X670V-48x	EXOS	EXTREME-SYSTEM-MIB,EXTREME-SOFTWARE-MONITOR-MIB	Certified
Huawei VRP SNMP	Huawei	Huawei VRP	S2352P-EI	-	ENTITY-MIB,HUAWEI-ENTITY-EXTENT-MIB	Certified
Intel_Qlogic Infiniband SNMP	Intel/QLogic	Intel/QLogic Infiniband devices	Infiniband 12300		ICS-CHASSIS-MIB	Fault Inventory
Juniper SNMP	Juniper	MX,SRX,EX models	Juniper MX240, Juniper EX4200-24F	JunOS	JUNIPER-MIB	Certified
Mellanox SNMP	Mellanox	Mellanox Infiniband devices	SX1036	MLNX-OS	HOST-RESOURCES-MIB,ENTITY-MIB,ENTITY-SENSOR-MIB,MELLANOX-MIB	Certified

模板名称	厂商	设备系列	已知模型	操作系统	使用的 MIB 库	标签
MikroTik CCR<device model> SNMP	MikroTik	MikroTik Cloud Core Routers (CCR series)	Separate dedicated templates are available for MikroTik CCR1009-7G-1C-1S+, MikroTik CCR1009-7G-1C-1S+PC, MikroTik CCR1009-7G-1C-PC, MikroTik CCR1016-12G, MikroTik CCR1016-12S-1S+, MikroTik CCR1036-12G-4S-EM, MikroTik CCR1036-12G-4S, MikroTik CCR1036-8G-2S+, MikroTik CCR1036-8G-2S+EM, MikroTik CCR1072-1G-8S+, MikroTik CCR2004-16G-2S+, MikroTik CCR2004-1G-12S+2XS	RouterOS	MIKROTIK-MIB,HOST-RESOURCES-MIB	Certified

模板名称	厂商	设备系列	已知模型	操作系统	使用的 MIB 库	标签
MikroTik CRS<device model> SNMP	MikroTik	MikroTik Cloud Router Switches (CRS series)	Separate dedicated templates are available for MikroTik CRS106-1C-5S, MikroTik CRS109-8G-1S-2HnD-IN, MikroTik CRS112-8G-4S-IN, MikroTik CRS112-8P-4S-IN, MikroTik CRS125-24G-1S-2HnD-IN, MikroTik CRS212-1G-10S-1S+IN, MikroTik CRS305-1G-4S+IN, MikroTik CRS309-1G-8S+IN, MikroTik CRS312-4C+8XG-RM, MikroTik CRS317-1G-16S+RM, MikroTik CRS326-24G-2S+IN, MikroTik CRS326-24G-2S+RM, MikroTik CRS326-24S+2Q+RM, MikroTik CRS328-24P-4S+RM, MikroTik CRS328-4C-20S-4S+RM,	RouterOS, SwitchOS	MIKROTIK-MIB,HOST-RESOURCES-MIB	Certified

模板名称	厂商	设备系列	已知模型	操作系统	使用的 MIB 库	标签
MikroTik CSS<device model> SNMP	MikroTik	MikroTik Cloud Smart Switches (CSS series)	Separate dedicated templates are available for MikroTik CSS326-24G-2S+RM, MikroTik CSS610-8G-2S+IN	RouterOS	MIKROTIK-MIB,HOST-RESOURCES-MIB	Certified
MikroTik FiberBox SNMP	MikroTik	MikroTik FiberBox	MikroTik FiberBox	RouterOS	MIKROTIK-MIB,HOST-RESOURCES-MIB	Certified
MikroTik hEX <device model> SNMP	MikroTik	MikroTik hEX	Separate dedicated templates are available for MikroTik hEX, MikroTik hEX lite, MikroTik hEX PoE, MikroTik hEX PoE lite, MikroTik hEX S	RouterOS	MIKROTIK-MIB,HOST-RESOURCES-MIB	Certified
MikroTik netPower <device model> SNMP	MikroTik	MikroTik netPower	Separate dedicated templates are available for MikroTik netPower 15FR, MikroTik netPower 16P SNMP, MikroTik netPower Lite 7R	RouterOS, SwitchOS Lite	MIKROTIK-MIB,HOST-RESOURCES-MIB	Certified

模板名称	厂商	设备系列	已知模型	操作系统	使用的 MIB 库	标签
MikroTik PowerBox <device model> SNMP	MikroTik	MikroTik PowerBox	Separate dedicated templates are available for MikroTik PowerBox, MikroTik PowerBox Pro	RouterOS	MIKROTIK-MIB,HOST-RESOURCES-MIB	Certified
MikroTik RB<device model> SNMP	MikroTik	MikroTik RB series routers	Separate dedicated templates are available for MikroTik RB1100AHx4, MikroTik RB1100AHx4 Dude Edition, MikroTik RB2011iL-IN, MikroTik RB2011iL-RM, MikroTik RB2011iLS-IN, MikroTik RB2011UiAS-IN, MikroTik RB2011UiAS-RM, MikroTik RB260GS, MikroTik RB3011UiAS-RM, MikroTik RB4011iGS+RM, MikroTik RB5009UG+S+IN	RouterOS	MIKROTIK-MIB,HOST-RESOURCES-MIB	Certified

模板名称	厂商	设备系列	已知模型	操作系统	使用的 MIB 库	标签
MikroTik SNMP	MikroTik	MikroTik RouterOS devices	MikroTik CCR1016-12G, MikroTik RB2011UAS-2HnD, MikroTik 912UAG-5HPnD, MikroTik 941-2nD, MikroTik 951G-2HnD, MikroTik 1100AHx2	RouterOS	MIKROTIK-MIB,HOST-RESOURCES-MIB	Certified
QTech QSW SNMP	QTech	Qtech devices	Qtech QSW-2800-28T	-	QTECH-MIB,ENTITY-MIB	Performance Inventory
Ubiquiti AirOS SNMP	Ubiquiti	Ubiquiti AirOS wireless devices	NanoBridge,NanoStation,Loco	AirOS	IEEE802dot11-MIB,HOST-RESOURCES-MIB	Performance
HP Comware HH3C SNMP	HP	HP (H3C) Comware	HP A5500-24G-4SFP HI Switch	-	HH3C-ENTITY-EXT-MIB,ENTITY-MIB	Certified
HP Enterprise Switch SNMP	HP	HP Enterprise Switch	HP ProCurve J4900B Switch 2626, HP J9728A 2920-48G Switch	-	STATISTICS-MIB,NETSWITCH-MIB,HP-ICF-CHASSIS,ENTITY-MIB,SEMI-MIB	Certified
TP-LINK SNMP	TP-LINK	TP-LINK	T2600G-28TS v2.0	-	TPLINK-SYSMONITOR-MIB,TPLINK-SYSINFO-MIB	Performance Inventory
Netgear Fastpath SNMP	Netgear	Netgear Fastpath	M5300-28G	-	FASTPATH-SWITCHING-MIB,FASTPATH-BOXSERVICES-PRIVATE-MIB	Fault Inventory

模板设计

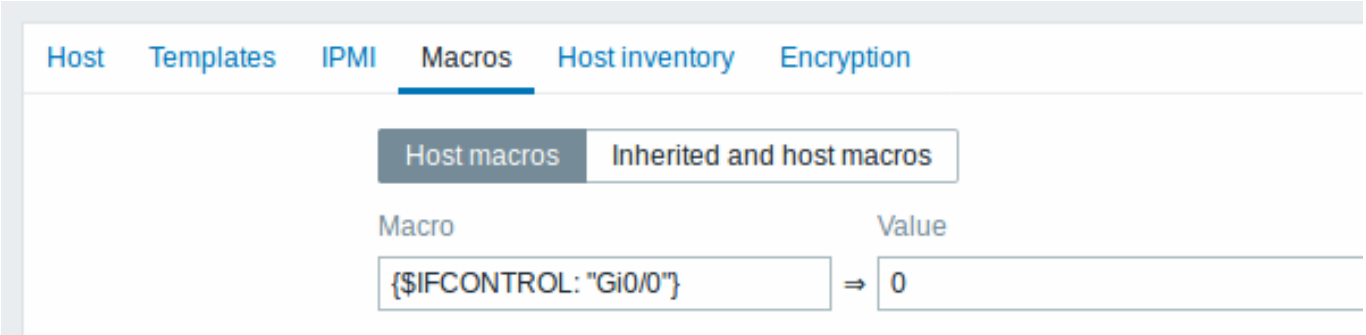
模板的设计考虑了以下几点：

- 尽可能多的使用用户宏，以便用户可以调整触发器；
- 尽可能使用底层自动发现，以尽量减少不受支持的监控项的数量；
- 所有模板都依赖于模板 ICMP Ping，因此所有设备也由 ICMP 检查；
- 监控项不使用任何 MIB - SNMP OID 用于监控项和底层自动发现。因此，无需将任何 MIB 加载到 Zabbix 中即可使模板正常工作；
- 环回网络接口在发现时被过滤以及 ifAdminStatus = down(2) 的接口;
- 尽可能使用 IF-MIB::ifXTable 中的 64 位计数器。如果不支持，则使用默认的 32 位计数器。

所有发现的网络接口都有一个触发器来监控其运行状态（链接），例如：

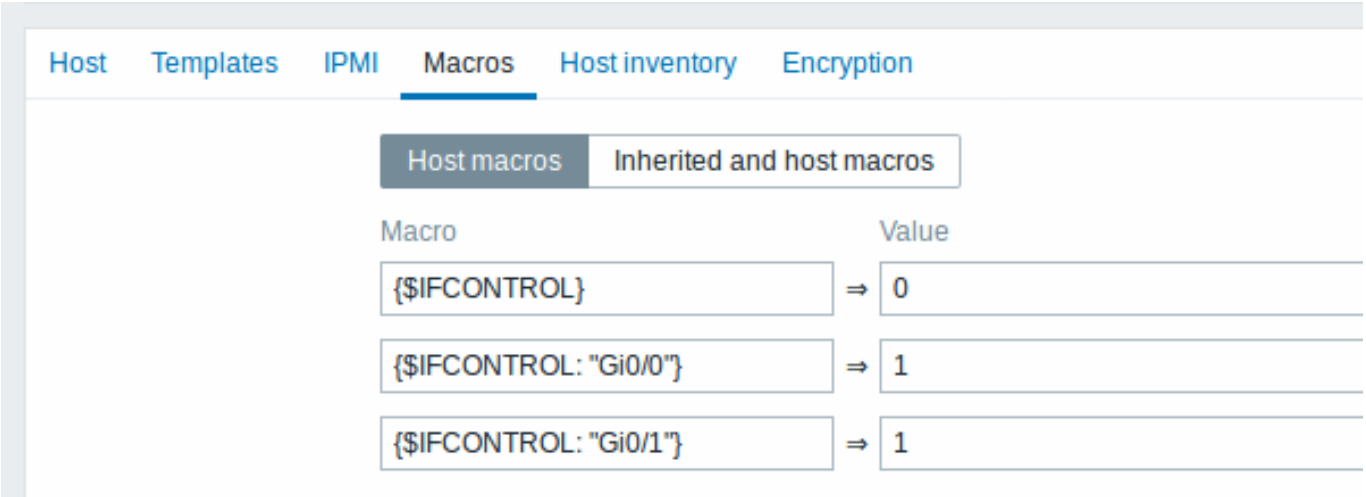
```
{$IFCONTROL:"{#IFNAME}"}=1 and last(/Alcatel Timetra TiMOS SNMP/net.if.status[ifOperStatus.{#SNMPINDEX}])=
```

- 如果您不想监控特定接口的这种情况，请创建一个上下文值为 0 的用户宏。例如：



其中 Gi0/0 是 {#IFNAME}。这样触发器就不再用于此特定接口。

- 您还可以更改所有触发器的默认行为，使其不触发并仅对有限数量的接口（如上行链路）激活此触发器：



标签

- Performance – 设备系列 MIB 提供了一种监控 CPU 和内存监控项的方法;
- Fault - 设备系列 MIB 提供监控至少一个温度传感器的方法;
- Inventory – 设备系列 MIB 提供了至少收集设备序列号和型号名称的方法;
- Certified – 涵盖上述所有三个主要类别。

10 事件通知

概述

在配置了诸多监控项和触发器，并且在触发器状态发生变化的情况下，用户已经接收到了一些告警信息，那么接下来就要考虑通过配置动作（actions）来响应事件的发生。

值得强调的是，用户不可能一直观察触发器或者事件的状态变化。更好的解决办法是，当发生明显的变化时（例如出现紧急问题）用户可以接收到系统发送的通知。当然，问题发生时，所有相关人员都可以接收到该事件通知是最好的。

可以说，事件通知是 Zabbix 首要的动作配置之一。该配置可以细致到对特定事件所要通知的对象和时间进行定义。

开启 Zabbix 事件通知的发送和接收功能，您需要做到：

- 定义媒介
- 配置动作 向指定的定义媒介发送消息

标准的动作由 条件和 操作两个元素构成。基本上，每当设定的条件达成时，就会执行相对应的设定操作。最重要的两个操作分别为：发送消息（事件提醒）和执行远程命令。

对于发现和自动注册所创建的事件，用户可以配置一些额外的操作。这些操作包括添加或删除主机，链接一个监控模板等。

1 媒介类型

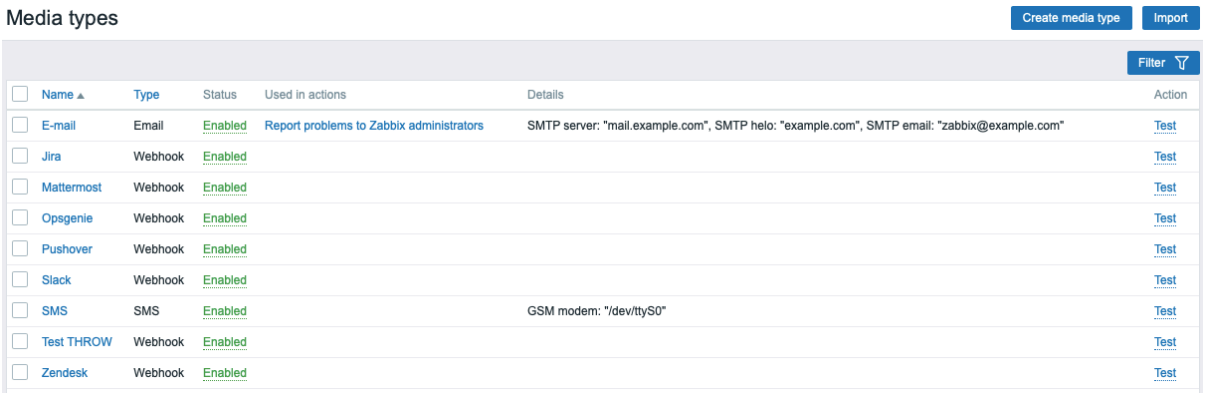
概述

媒介的定义确定了 Zabbix 发送通知和告警的渠道。

您可以配置多种媒介类型：

- 电子邮箱
- 短信
- 自定义报警脚本
- Webhook

媒介类型的配置位于 管理 → 媒介类型。



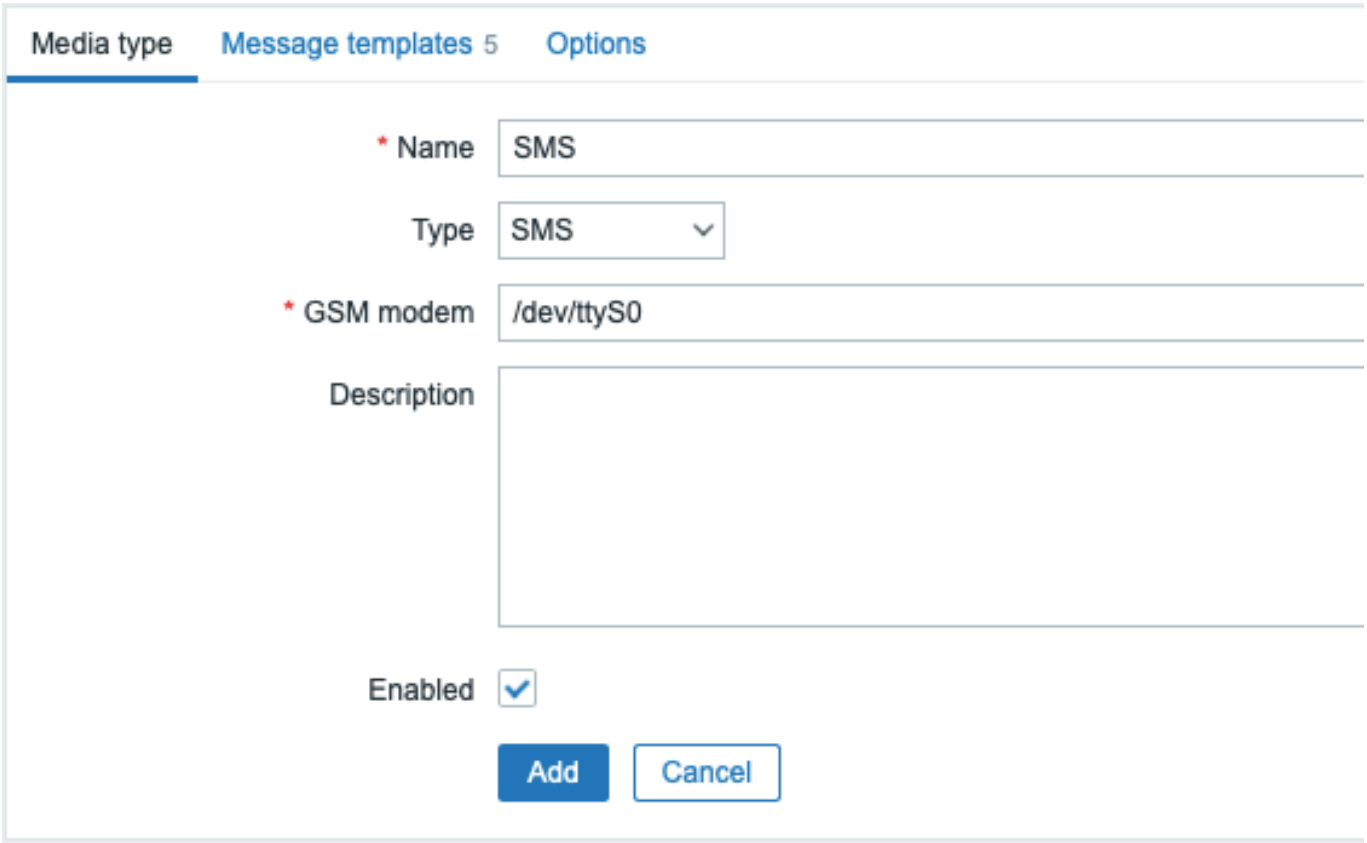
某些媒介类型是在默认数据中完成了自定义的。您只需对其配置参数进行一些微调操作，即可正常运行。

媒介类型配置的正确与否，是可以通过点击对应媒介类型中最后一行的 测试按钮进行测试的 (请参考媒介类型测试)。

创建一个新的媒介类型，可以通过点击 创建媒介类型按钮来完成。点击该按钮后，即可打开媒介类型的配置表单。

通用参数

某些参数是媒介类型的通用参数。



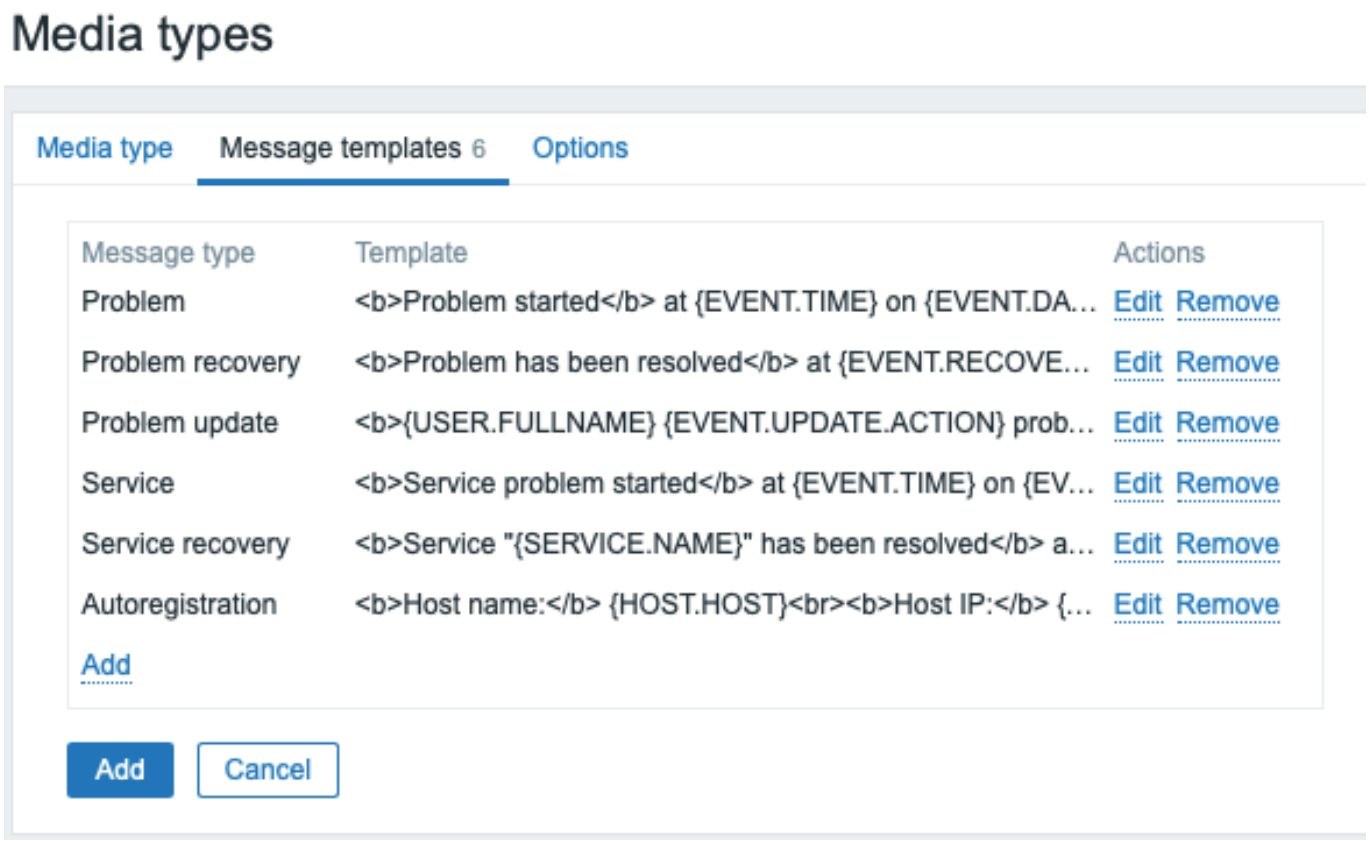
位于 媒介类型选项卡中的常见通用属性包括：

参数	说明
名称	媒介类型的名称。
类型	选择媒介的类型。
描述	对该媒介类型的描述。
启用	通过勾选复选框来启用该媒介类型。

有关媒介的具体参数，请参考各个媒体类型的说明页面。

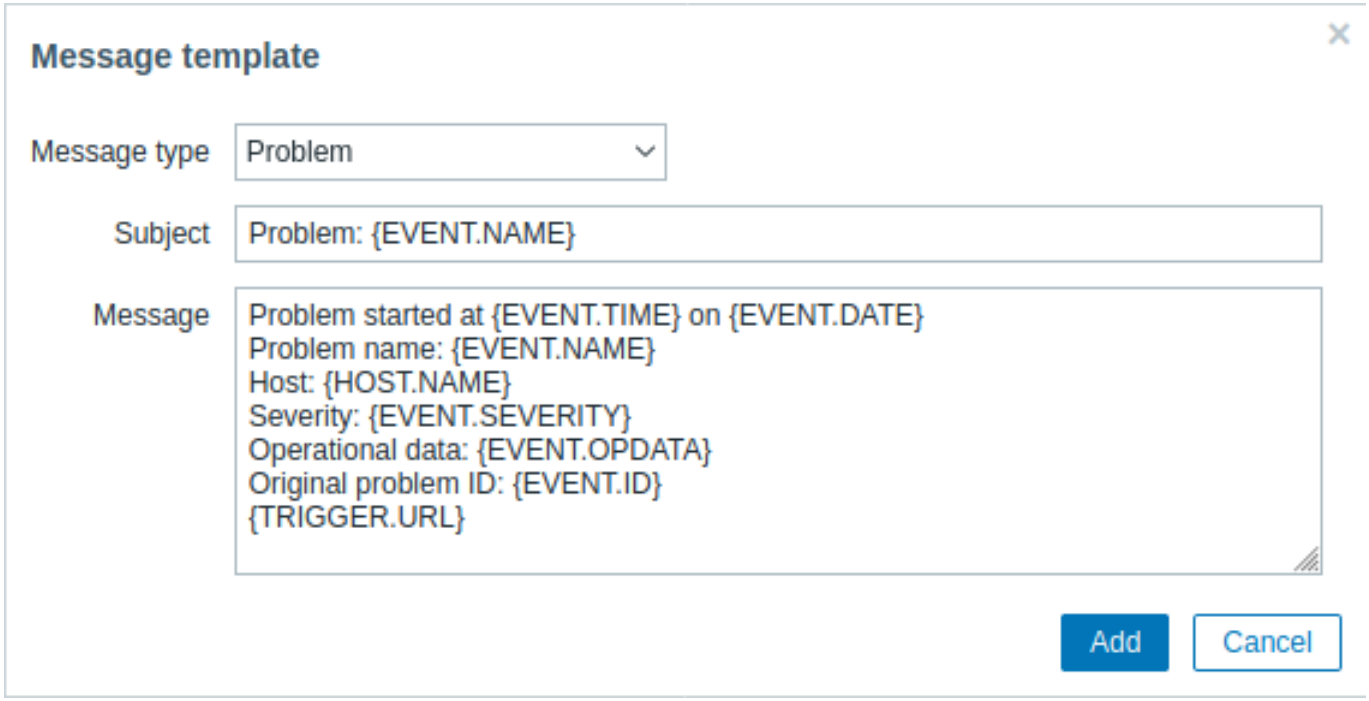
用户可以在消息模板选项卡中，对下列所有或者某些事件类型进行相关的默认通知消息配置：

- 问题
- 问题恢复
- 问题更新
- 服务
- 服务恢复
- 服务更新
- 自动发现
- 自动注册
- 内部问题
- 内部问题恢复



自定义消息模板需要：

- 在 消息模板选项卡中，点击 Add：将打开 消息模板弹出窗口。
- 选择需要的 消息类型，编辑 主题和添加 消息文本内容。
- 点击 添加按钮来保存该消息模板。



消息模板的配置参数：

参数	说明
消息类型	选择事件类型，该参数定义默认消息类型。 每种事件类型只支持定义一个默认消息。
主题	默认的消息主题。该主题可以包含宏参数。主题的长度限定在 255 个字符。 SMS 类型的媒介并不支持配置主题参数。
消息	默认的消息内容。根据数据库类型，消息内容被限制在一定的字符数量以内 (更多信息，请参考消息发送 中的内容)。 消息内可以包含支持的宏。 在问题和问题更新消息类型中，均支持宏表达式的应用 (举例来说，{?avg(/host/key,1h)})。

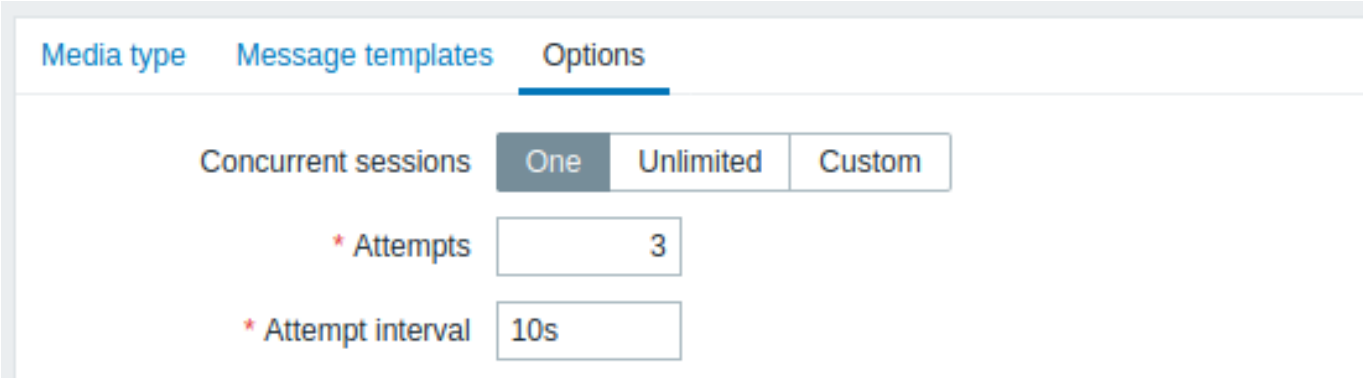
对现有的消息模板进行修改，需要：在 动作一行中点击 Edit 来编辑模板或者点击 Remove 来删除消息模板。

用户也可以根据需求对一个自定义的消息模板定义一个特定的动作（相关具体内容，请参考动作操作)。动作配置中的用户自定义消息会覆盖默认媒介类型的消息模板。

Warning:
消息模板的定义是必要且强制性的，其配置内容包括配置不使用默认消息通知的 webhooks 或者应用自定义报警脚本。举例来说，若未定义 Pushover webhook 的具体问题消息，那么针对动作 “对 Pushover webhook 发送消息” 将不会有任何通知发送。

选项选项卡包含了报警处理设置。每种媒介类型都可以配置相同的选项集。

所有媒介类型都是并行进行处理的。尽管，对每种媒介类型的最大并存会话数是可以进行配置的，但是在服务器上的报警进程总数是由参数 StartAlerters 参数进行限制的。由同一个触发器生成的报警信息会按照次序依次进行处理。从这点来看，只有配置多个触发器才可能同时生成多个报警。



参数	说明
并发会话	选择媒介类型的并行报警会话数量： 壹 - 单一会话 无限制 - 无限制的会话数量 自定义 - 对会话数量进行自定义设置 无限制/自定义下的大数值设置，意味着发送通知会同时拥有多个会话并且会话数量也会增长。 无限制/自定义下的大数值设置，应在有大量通知需要同时发送的情境下使用。 如果需要发送的通知数量多于并发会话所支持的数量，那么剩余的通知则会排队等待；余下的通知并不会被系统丢弃。
尝试次数	尝试发送通知的次数。该参数最大可配置数值为 100；该参数默认数值为 '3'。如果该参数被设定为 '1'，那么 Zabbix 只会发送一次通知，即便发送失败也不会尝试再次发送。
尝试间隔	当发送通知失败时，重新尝试发送通知的频率，单位为秒（0-3600）。 若该参数设定为 '0'，那么在发送通知失败后，Zabbix 会立即重发。 支持定义时间后缀，例如：5s，3m，1h。

媒介类型测试

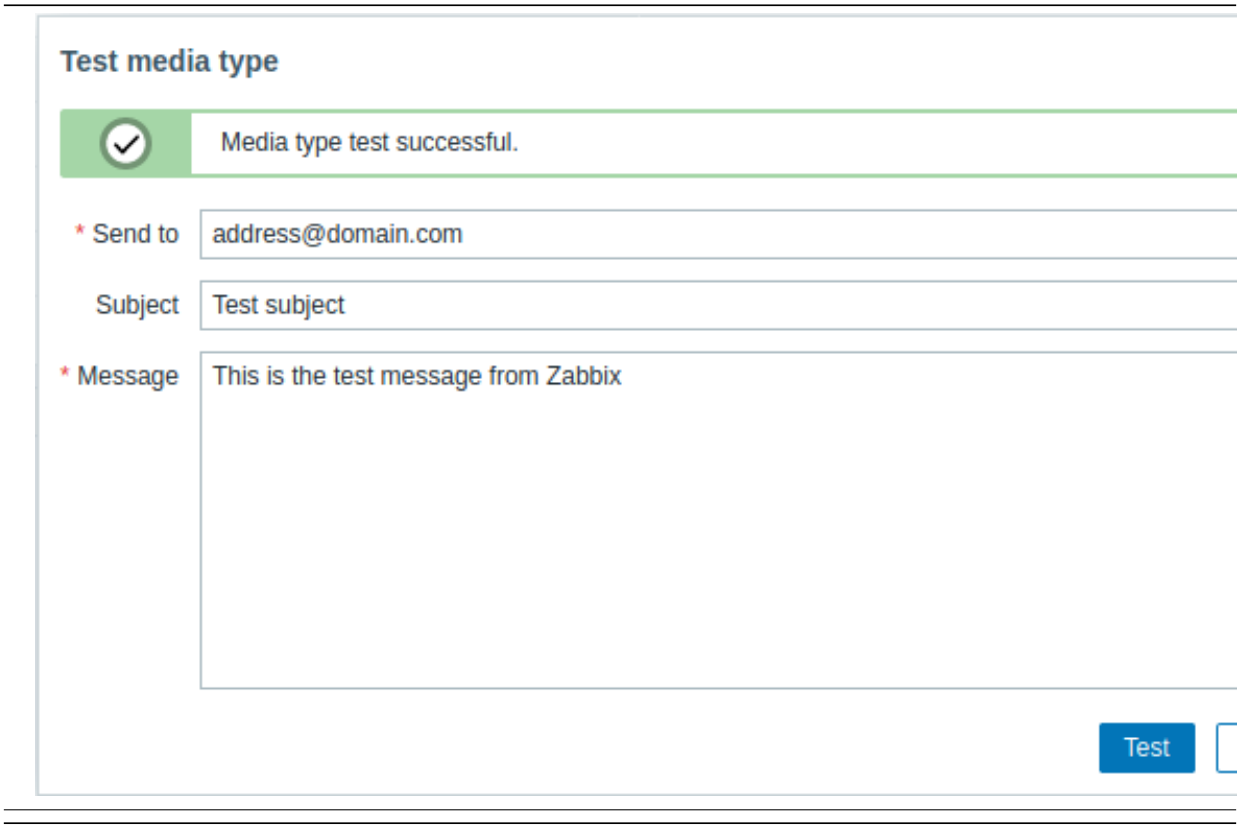
完成配置的媒介类型可以对其进行测试，确认是否可以正常工作。

电子邮件

举例来说，对一个电子邮件的媒介类型进行测试：

- 在媒介类型列表 中确认相关联的电子邮件
- 点击位于列表最后一栏的 测试按钮（测试窗口会自动打开）
- 在 发送至栏中填写收件人地址，同时填写发送的具体内容并依自身需求选择性填写主题信息
- 通过点击 测试按钮发送测试信息

测试的成功与否都会通过消息窗口进行展示：

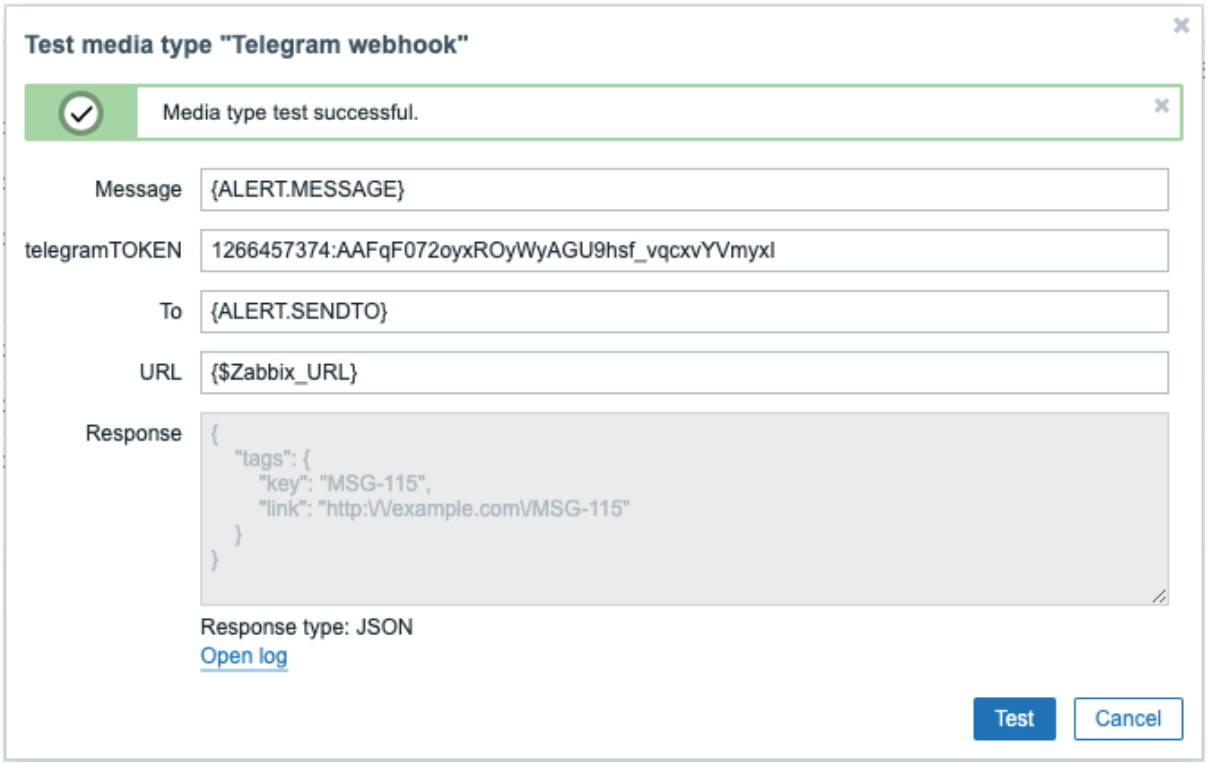


Webhook

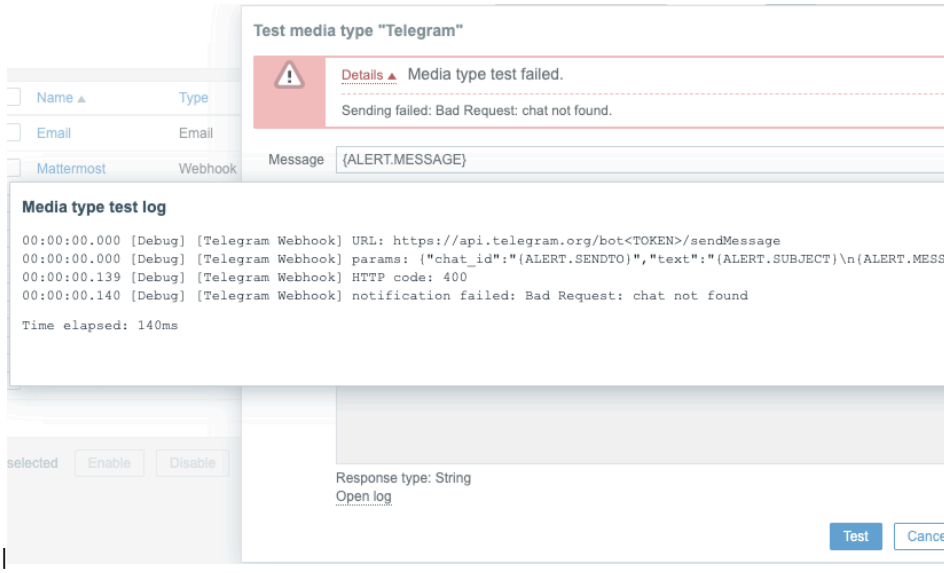
测试 webhook 媒介类型：

- 在媒介类型列表 中找到相关联的 webhook
- 点击位于列表最后一栏的 测试按钮（测试窗口会自动打开）
- 根据需求，编辑 webhook 参数值
- 点击 测试

默认情况下，webhook 测试使用配置期间输入的参数执行。但是，可以更改属性值以进行测试。在测试窗口进行替换或者删除的操作只会影响测试流程，webhook 的实际属性数值不会发生改变。



不关闭测试窗口查看媒介类型测试日志：



- 点击 打开日志(自动弹出新的窗口展示日志)。	<

当 webhook 类型测试成功

- 界面将显示*"Media type test successful."*
- 服务器响应会提示在 *Response* 灰色信息栏中
- *Response* 信息栏下方会详细标注响应类型（JSON 或 String）

当 webhook 类型测试失败

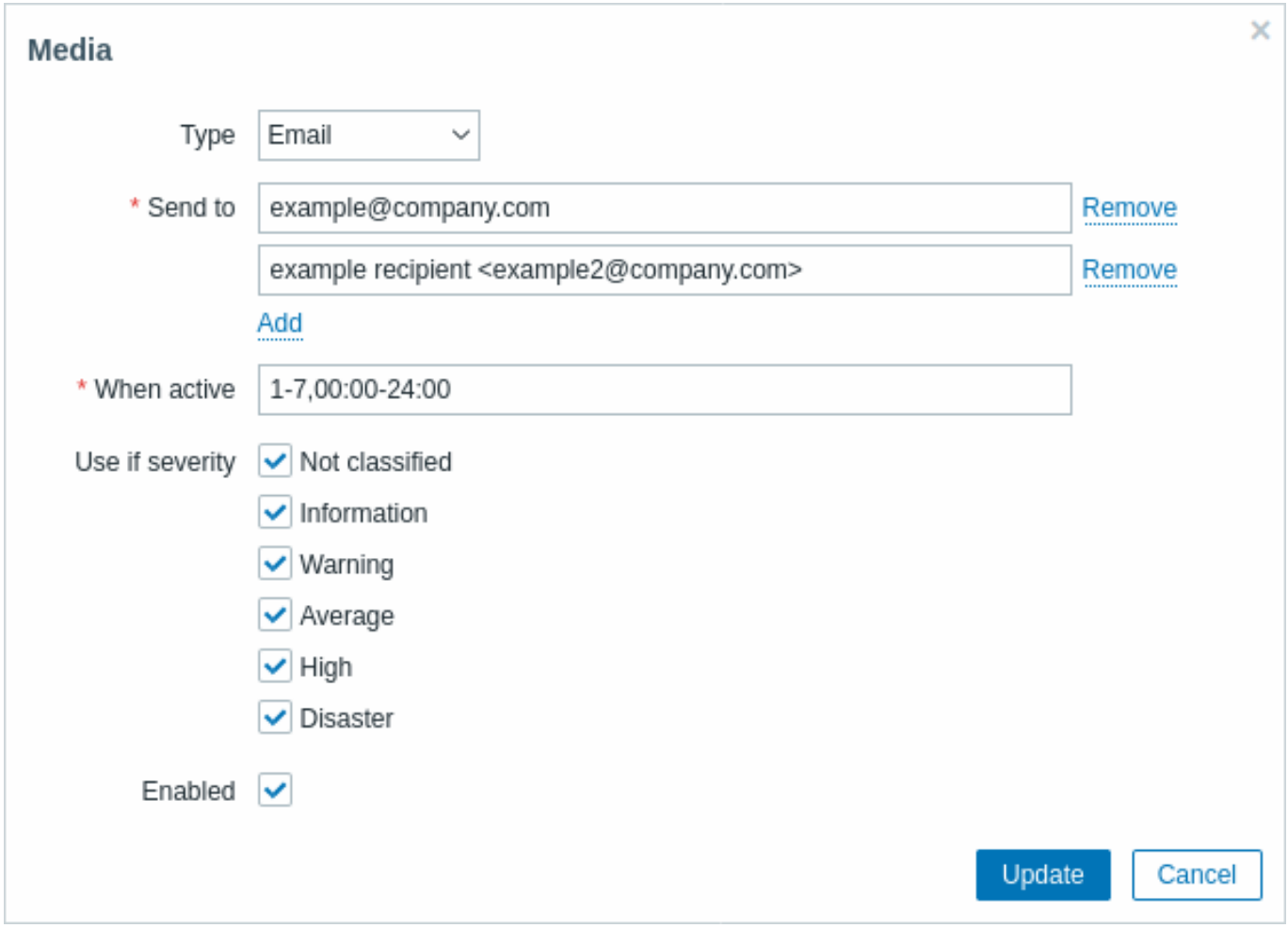
- 操作界面会提示信息*"Media type test failed."* ，有关测试失败的详细信息也会同时展示。

用户媒介

要接收媒介类型的通知，必须在用户配置文件中定义此媒介类型的媒介（电子邮件地址/电话号码/webhook user ID 等）。例如，如果用户配置文件中未定义 webhook"X" 介质，则使用 webhook"X" 向用户 "Admin" 发送消息的操作将始终无法发送任何内容。

定义用户媒介：

- 转到您的用户配置文件，或转到管理 → 用户并打开用户属性表单
- 在 "媒介" 选项卡中，单击 Add



用户媒介属性：

参数	说明
类型	下拉列表包含所有已配置媒介类型的名称。
发送至	提供发送消息所需的联系信息。 对于电子邮件媒介类型，可以通过单击电子邮件栏下方 [](../../../assets/en/manual/config/add_link.png) 来添加多个地址。在这种情况下，通知将发送到所有提供的电子邮件地址。也可以在电子邮件收件人的 发送至字段中指定收件人姓名，格式为 “收件人姓名 <address1@company.com>”。请注意，如果提供了收件人姓名，则电子邮件地址应括在尖括号 (<>) 中。支持名称中的 UTF-8 字符，不支持引号对和注释。例如：*John Abercroft <manager@nycdatacenter.com>* 和 *manager@nycdatacenter.com* 都是有效格式。不正确的例子：*John Doe zabbix@company.com, %%”Zabbix\@\<H(comment)Q\>” zabbix@company.com %%*.
何时发送	您可以限制发送消息的时间，例如，仅设置工作日 (1-5,09:00-18:00)。请注意，此限制基于用户时区。如果用户时区更改并且与系统时区不同，则可能需要相应地调整此限制，以免错过重要消息。 请参阅时间段规范页面格式的描述。
告警级别设置	标记您想要接收通知的触发器严重性的复选框。 请注意如果您想接收由非触发器引起的事件通知，请 一定要勾选默认程度（‘未分级’）。 保存后，选择的触发级别将以相应的级别颜色显示，而未选择的将显示为灰色。
状态	用户媒介的状态。 启用 - 正在使用中。 禁用 - 未使用。

1 电子邮箱

概述

若要使用电子邮件作为消息发送的通道，那么您需要选择电子邮件作为媒介类型，同时为接收消息的用户指定具体的邮箱。

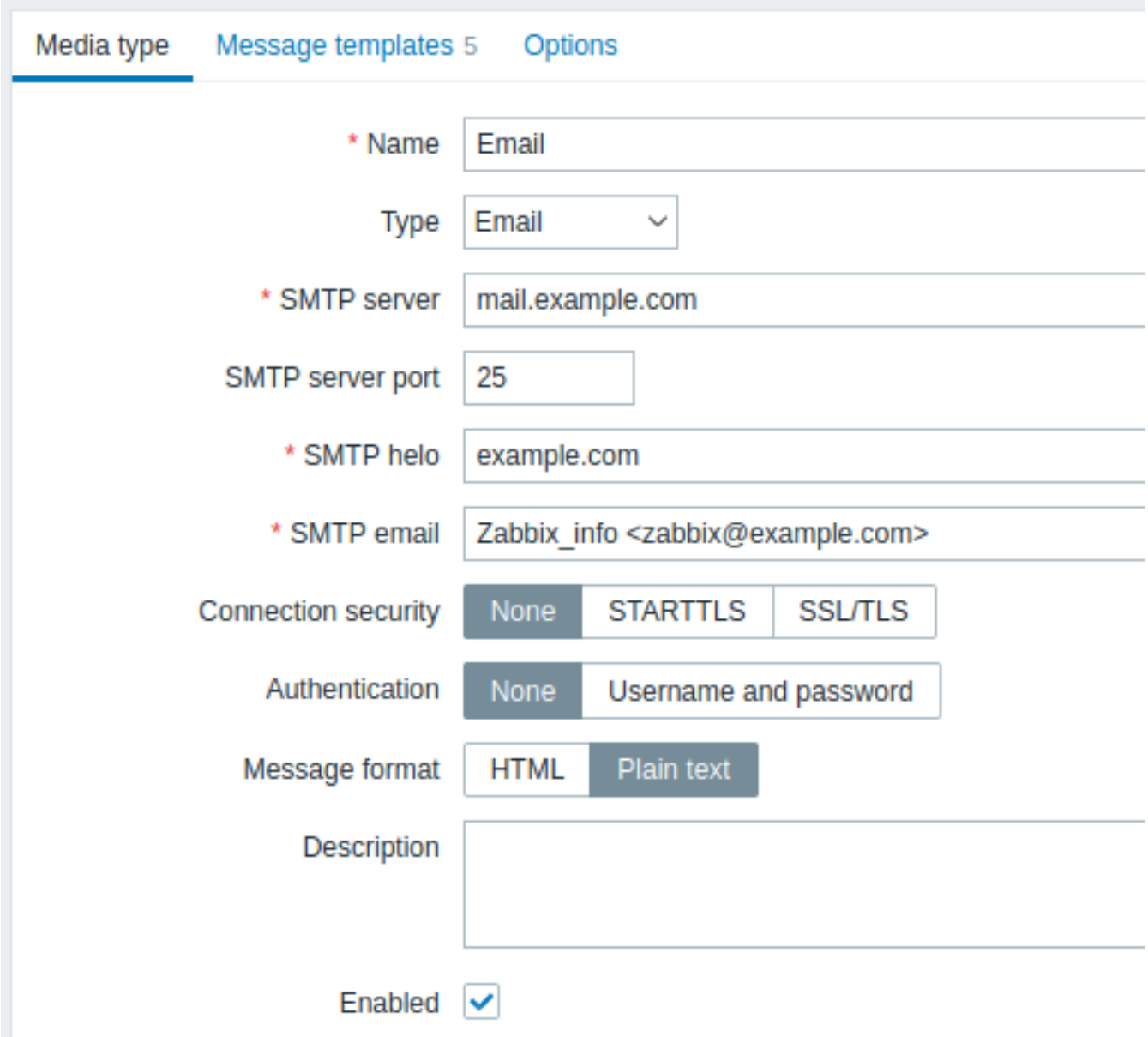
Note:
同一事件的多个通知会由相同的邮件线程处理。

配置

将电子邮件配置为媒体类型：

- 转到管理 → 媒体类型
- 点击创建媒体类型（或点击预定义媒体类型列表中的电子邮件）。

媒体类型选项卡包含一般媒体类型属性：



所有必填输入字段都标有红色星号。

以下参数特定于电子邮件媒体类型：

参数	说明
SMTP 服务器	设置一个 SMTP 服务器来处理外发消息。
SMTP 服务器端口	设置 SMTP 服务器端口以处理传出消息。 从 *Zabbix 3.0* 开始支持此选项。
SMTP helo	设置正确的 SMTP helo 值，通常是域名。

参数	说明
SMTP 电子邮件	此处输入的地址将用作发送邮件的发件人地址。 自 Zabbix 2.2 版本起，支持为发件人实际邮箱地址添加显示名称（如上述截图中 *Zabbix_info <zabbix@* 中的 *"Zabbix_info" company.com>*）。 与 RFC 5322 允许的相比，Zabbix 电子邮件中的显示名称有一些限制，如示例： 有效示例： *zabbix@company.com*（仅电子邮件地址，无需使用尖括号） *Zabbix_info <zabbix@company.com>*（有显示名称并且使用尖括号包含邮箱地址） *∑Ω-monitoring <zabbix@company.com>*（显示名称中有使用 UTF-8 字符） 无效示例： *Zabbix HQ zabbix@company。com*（有显示名称，但没有使用尖括号包含电子邮件地址） *"Zabbix\@\<H(comment)Q\>" <zabbix@company.com>*（尽管 RFC 5322 有效，但 Zabbix 电子邮件中不支持括号和注释）
连接安全	选择连接安全级别： 无 - 不使用 CURLOPT_USE_SSL 选项 **STARTTLS** - 使用带有 CURLUSSSL_ALL 值的 CURLOPT_USE_SSL 选项 **SSL/TLS** - CURLOPT_USE_SSL 的使用是可选的 从 *Zabbix 3.0* 开始支持该选项。
SSL verify peer	勾选复选框以验证 SMTP 服务器的 SSL 证书。 "SSLCALocation" 服务器配置指令的值应放入CURLOPT_CAPATH用于证书验证。 设置 cURL 选项，请参考CURLOPT_SSL_VERIFYPEER。 > 从 *Zabbix 3.0* 开始支持该选项。
SSL verify host	选中该复选框以验证 SMTP 服务器证书的 *Common Name* 字段或 *Subject Alternate Name* 字段是否匹配。 设置 cURL 选项，请参考CURLOPT_SSL_VERIFYHOST。 从 *Zabbix 3.0* 开始支持该选项。
身份验证	选择身份验证级别： 无 - 未设置 cURL 选项 （自 3.4.2 版本起）用户名和密码 - 验证机制由 cURL 完成而非"AUTH=*" （直到 3.4.2 版本）普通密码 - CURLOPT_LOGIN_OPTIONS参数设置为 "AUTH=PLAIN" 从 *Zabbix 3.0* 开始支持该选项。
用户名	用于身份验证的用户名。 设置CURLOPT_USERNAME参数的值。 从 *Zabbix 3.0* 开始支持该选项。
密码	用于身份验证的密码。 设置 CURLOPT_PASSWORD参数的值。 从 *Zabbix 3.0* 开始支持该选项。
消息格式	选择消息格式： **HTML** - 以 HTML 格式发送 纯文本 - 以纯文本格式发送

Attention:
要使 SMTP 身份验证选项可用，Zabbix server 编译时添加--with-libcurl 选项 (编译)，cURL 要求 7.20.0 及以上版本。

有关如何配置默认消息和告警处理选项的详细信息，另请参阅通用媒体类型参数。

用户媒介

当完成电子邮件媒介类型的配置，请前往 管理 → 用户配置栏，对用户属性中的电子邮件媒介进行配置。用户媒介的设定步骤，该设定适用于所有媒介类型，请参考媒介类型 页面。

2 短信

概述

Zabbix 支持使用连接到 Zabbix server 串行口的串行 GSM 调制解调器发送短信。

请确保满足以下条件：

- 串行设备的速率（Linux 下通常为 /dev/ttyS0 ）与 GSM 调制解调器的速度一致。Zabbix 没有设置串行链路的速度。它使用默认设置。
- 'zabbix' 用户对串行设备具有读/写访问权限。执行 ls -l /dev/ttyS0 命令来查看当前串口设备的权限。

- GSM 调制解调器已经输入了 PIN 码，并且在电源复位后会将其保留。或者，您可以禁用 SIM 卡上的 PIN 码。可以通过在终端软件（如 Unix minicom 或 Windows HyperTerminal）中发出命令 AT + CPIN ="NNNN" 来输入 PIN 码（NNNN 是您的 PIN 码，且必须放在引号中）。

Zabbix 已通过以下 GSM 调制解调器的测试：

- Siemens MC35
- Teltonika ModemCOM/G10

配置短信作为消息的传送通道时，需要将短信配置为媒介类型，并输入相应用户的电话号码。

配置

将 SMS 配置为媒介类型：

- 进入 管理 → 媒介类型
- 点击 创建媒介类型 (或者点击预定义的媒介类型列表中的 SMS)。

如下参数仅适用于 SMS 媒介类型：

参数	说明
GSM modem	设置 GSM 调制解调器的串行设备名称。

有关如何配置默认消息和告警处理的详细内容，请参见通用媒介类型参数 。请注意，SMS 通知无法并行发送。

用户媒介

配置完 SMS 媒体类型后，需进入 管理 → 用户为用户配置 SMS 媒介。配置用户媒介的步骤和配置其它媒介类型的方式类似，可以参见媒介类型

3 自定义告警脚本

概述

如果当前的告警媒介类型无法满足您的要求，您可以创建一个自定义的脚本对告警通知进行处理。

告警脚本在 Zabbix 服务器上执行。这些脚本放置于服务器配置文件configuration file中定义的 **AlertScriptsPath** 目录下。

以下是一个自定义告警脚本的示例:

```
#####!/bin/bash

to=$1
subject=$2
body=$3

cat <<EOF | mail -s "$subject" "$to"
$body
EOF
```

Attention:
从 3.4 版本开始，Zabbix 会检查执行的命令和脚本的退出代码。任何不为 **0** 的退出代码都将被视为command execution 错误. 在这种情况下，Zabbix 会尝试重复执行。

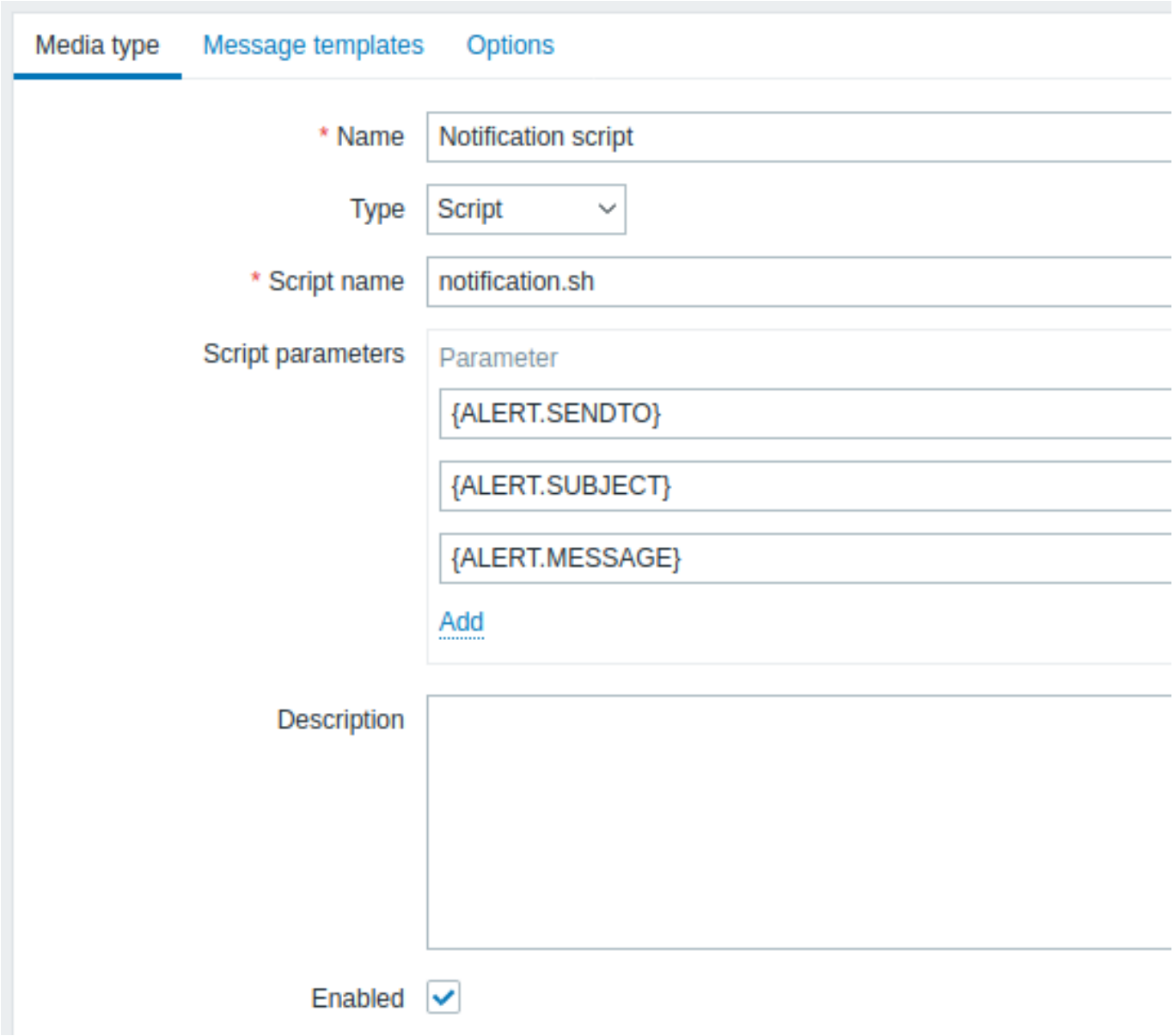
环境变量不会为脚本保留或创建，因此它们应该被明确处理。

配置

配置自定义告警脚本为媒介类型：

- 进入 管理 → 媒介类型
- 点击 创建媒介类型

媒介类型页包含一些通用的媒介类型属性如下：



标红星的为必填字段。

下列参数只适用于脚本媒介类型：

参数	说明
脚本名称	输入脚本的名称
脚本参数	添加脚本的命令行参数 脚本参数中支持 {ALERT.SENDTO}, {ALERT.SUBJECT} 和 {ALERT.MESSAGE} 宏 从 Zabbix 3.0 版本开始，支持自定义脚本参数.

关于如何配置默认消息及告警处理的详细内容，请参见通用媒介类型参数

Warning:
即使告警脚本没有使用默认消息，此媒介类型使用的操作类型的消息模板仍必须定义，否则通知将无法发送。

Attention:
从 Zabbix 3.4.0 开始实现了多个告警媒介并行处理，所以需要注意的是，当配置了多个告警脚本时，这些脚本是可以被告警进程并行处理的。告警进程的进程数可以通过 Zabbix Server 的配置项 StartAlerters参数进行限制。

用户媒介

配置完媒介类型后，需进入管理 → 用户为用户配置相应的媒介。配置用户媒介的步骤和配置其它媒介类型的方式类似，可以参见媒介类型页.

注意，在定义用户媒介的时候，*Send to* 字段不能为空。如果该字段在告警脚本中不会被使用，可以输入任意支持的字段以跳过该校验。

4 Webhook

概述

webhook 媒体类型对于使用自定义的 JavaScript 代码进行 HTTP 调用非常有用，它可以直接与外部软件（如 Helpdesk 系统、聊天工具或信使）进行集成。您可以选择使用 Zabbix 提供的集成方式或创建一个自定义集成方式。

集成

以下集成方式允许使用预定义的 webhook 媒介类型来推送 Zabbix 通知:

- brevis.one
- Discord
- Express.ms messenger
- Github issues
- iLert
- iTop
- Jira
- Jira Service Desk
- ManageEngine ServiceDesk
- Mattermost
- Microsoft Teams
- Opsgenie
- OTRS
- Pagerduty
- Pushover
- Redmine
- Rocket.Chat
- ServiceNow
- SIGNL4
- Slack
- SolarWinds
- SysAid
- Telegram
- TOPdesk
- VictorOps
- Zammad
- Zendesk

> **Note:**
> 除了这里列出的服务外，Zabbix 还可以集成 **Spiceworks** (无需 webhook)。要把 Zabbix 通知转换成 Spiceworks 单据，需先创建一个电子邮件媒介类型 ，并在指定的 Zabbix 用户配置设置中输入 Spiceworks helpdesk 的邮件地址 (例如 help@zabbix.on.spiceworks.com)。

配置

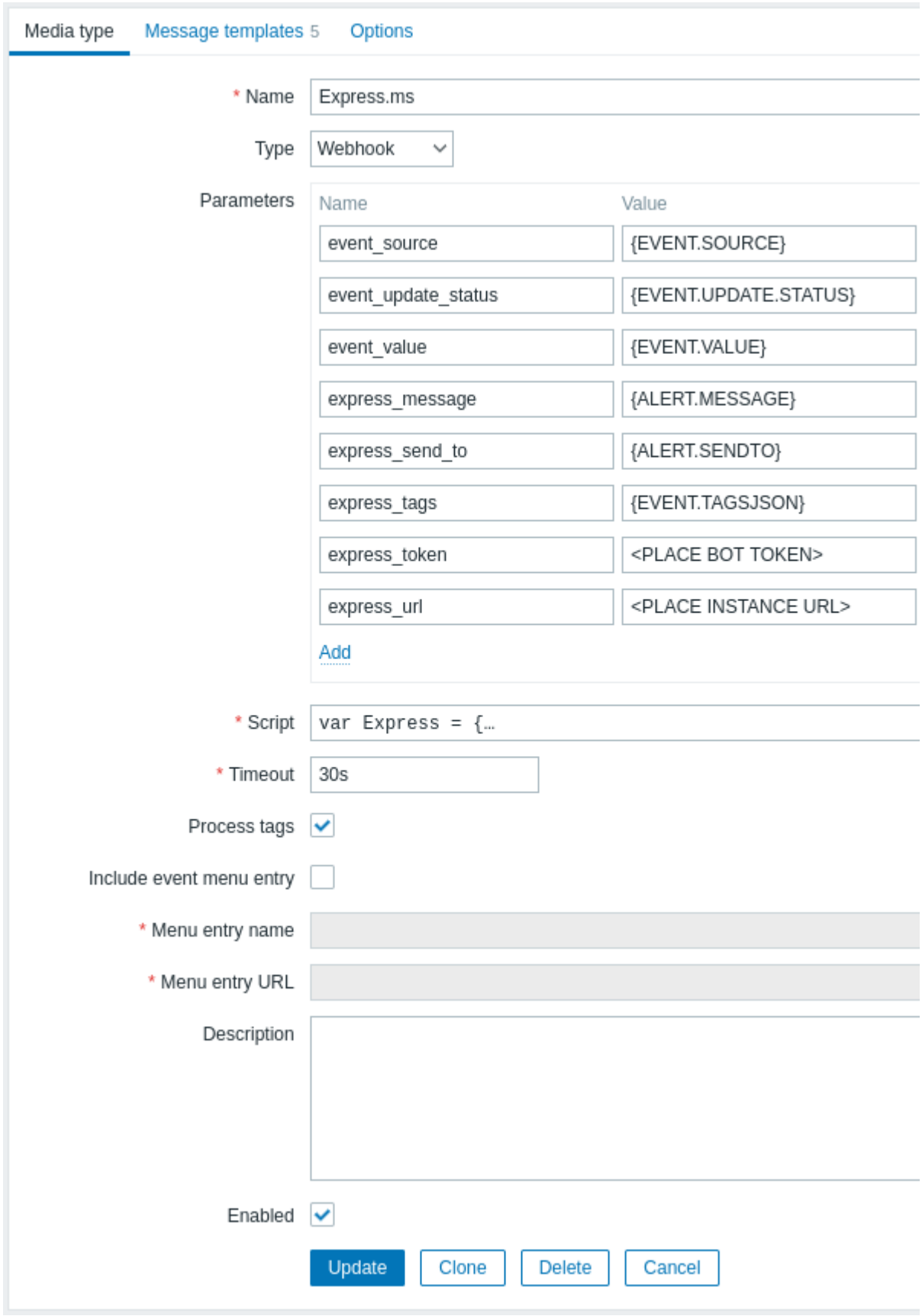
开始使用 webhook 集成:

1. 在已下载的 Zabbix 的 `templates/media` 目录中，找到所需的.xml 文件；或者从 Zabbix 的 git 仓库中下载
2. 将文件导入 到 Zabbix 安装，Webhook 将出现在媒体类型列表中。
3. 根据 *Readme.md* 文件中的说明来配置 webhook （也可以点击 webhook's 名称来快速访问 *Readme.md* ）。

从零开始创建一个自定义的 webhook:

- 进入管理 → 媒介类型
- 点击创建媒介类型

媒介类型选项卡包含了针对这种媒介类型的各种属性:



红色星号标记的为必填字段。

webhook 媒介类型的具体参数如下:

参数	说明
参数	webhook 变量作为属性和值对。 对于预先配置的 webhook，参数列表会随着服务的不同而变化。检查 webhook 的 *Readme.md* 参数说明文件。 对于新的 webhooks，默认情况下包含了几个常见的变量 (URL:<empty>, HTTPProxy:<empty>, To:{ALERT.SENDTO}, Subject:{ALERT.SUBJECT}, Message:{ALERT.MESSAGE}), 你可以保留或删除它们。 参数中支持问题通知中支持的所有宏 如果指定了 HTTP Proxy，该字段支持与监控项配置HTTP proxy字段相同的功能。Proxy 字符串可以加上前缀 `[scheme]://` 来指定使用哪种代理例如 https, socks4, socks5; 请参见documentation).
脚本	在点击参数字段 (或旁边的查看/编辑按钮) 时出现的块中输入 JavaScript 代码。这段代码将执行 webhook 操作。 脚本是接受参数-值对的一段功能代码，它的值应该使用 JSON.parse() 方法转换为 JSON 对象，例如：`var params = JSON.parse(value);`. 代码可以访问所有参数，它可以执行 HTTP GET、POST、PUT 和 DELETE 请求，并可控制 HTTP 头和请求正文。 脚本必须包含一个返回值，否则它将无效。它可以返回 OK 状态以及一个可选的标签列表，标签值 (参见 *Process tags* 选项)，或一个错误的字符串。 注意，脚本只有在创建警报之后才会执行。如果脚本被配置为返回和处理标签 (参见处理标签选项), 这些标签将不会在初始问题消息和恢复消息中的 {EVENT.TAGS} 和 {EVENT.RECOVERY.TAGS} 宏中解析，因为脚本还没有时间运行。 另请参阅：Webhook 开发指南, Webhook 脚本样例, 额外的 JavaScript 对象.
超时	JavaScript 执行超时 (1-60s, 默认为 30s)。 支持时间后缀, 例如 30s, 1m。
处理标签	选中复选框标以将返回的 JSON 属性值作为标签处理。这些标签将被添加到 Zabbix 中已经存在的 (如果有的话) 问题事件标签中。 如果 webhook 使用了标签 (Process tags 复选框被选中),webhook 应该返回一个至少带有一个空标签的 JSON 对象 `var result = {tags: {}};`. 返回的标签示例: *Jira ID: PROD-1234*, *Responsible: John Smith*, *Processed:<no value>*, 等.
包括事件菜单项	选中复选框以在事件菜单 中包含一个链接到创建的外部目标的条目。 若选中此项，webhook 就不应该被用来向不同的用户发送通知 (考虑创建一个专有用户) 或者在几个告警动作中关联单个问题事件.
菜单项名称	指定菜单入口名称。 支持 {EVENT.TAGS.<tag name>} 宏 只有当包括事件菜单项被选中时，该字段才为必填项。
菜单项 *URL*	指定菜单入口的 URL。 支持 {EVENT.TAGS.<tag name>} 宏 只有当包括事件菜单项被选中时，该字段才为必填项。

关于如何配置默认消息和告警处理选项的详细信息，请参见通用媒介类型参数

Warning:
即使 webhook 不使用默认消息，webhook 使用的操作类型的消息模板也必须定义。

用户媒介

媒介类型配置完成后，前往 管理 → 用户部分，为一个现有用户或创建一个新用户并指定其 webhook 媒介。关于如何为用户指定用户媒介的方式和指定其它媒介类型类似，具体请参见媒介类型。

如果 webhook 使用标签来存储 ticket\message ID, 避免将同一个 webhook 作为媒体分配给不同的用户，因为这样做可能会导致

webhook 错误 (适用于大多数使用了 *Include event menu entry* 选项)。在这种情况下，最好的做法是创建一个专用的用户来表示 webhook:

1. 配置好 webhook 媒体类型后，前往 管理 → 用户部分，并创建一个专用的 Zabbix 用户来表示 webhook - 例如, 为 Slack webhook 添加一个别名 *Slack*。所有的设置可以保持默认值（除了媒体），因为这个用户不会登录到 Zabbix。
2. 在用户配置中，进入 *Media* 选项卡并且填写add a webhook 所需的联系信息。如果 webhook 没有使用 *Send to* 字段, 可以输入任何支持的字符组合来绕过验证要求。
3. 至少授予该用户对要向其发送警报的所有主机有可读的权限。

配置告警动作时，请在 *Send to users* 字段中添加该用户 - 这将告诉 Zabbix 使用 webhook 来获取来自这个动作的通知。

配置告警动作

告警动作决定了哪些通知应该通过 webhook 发送。webhook 的configuring actions 步骤与所有其他媒体类型相同，但有以下例外:

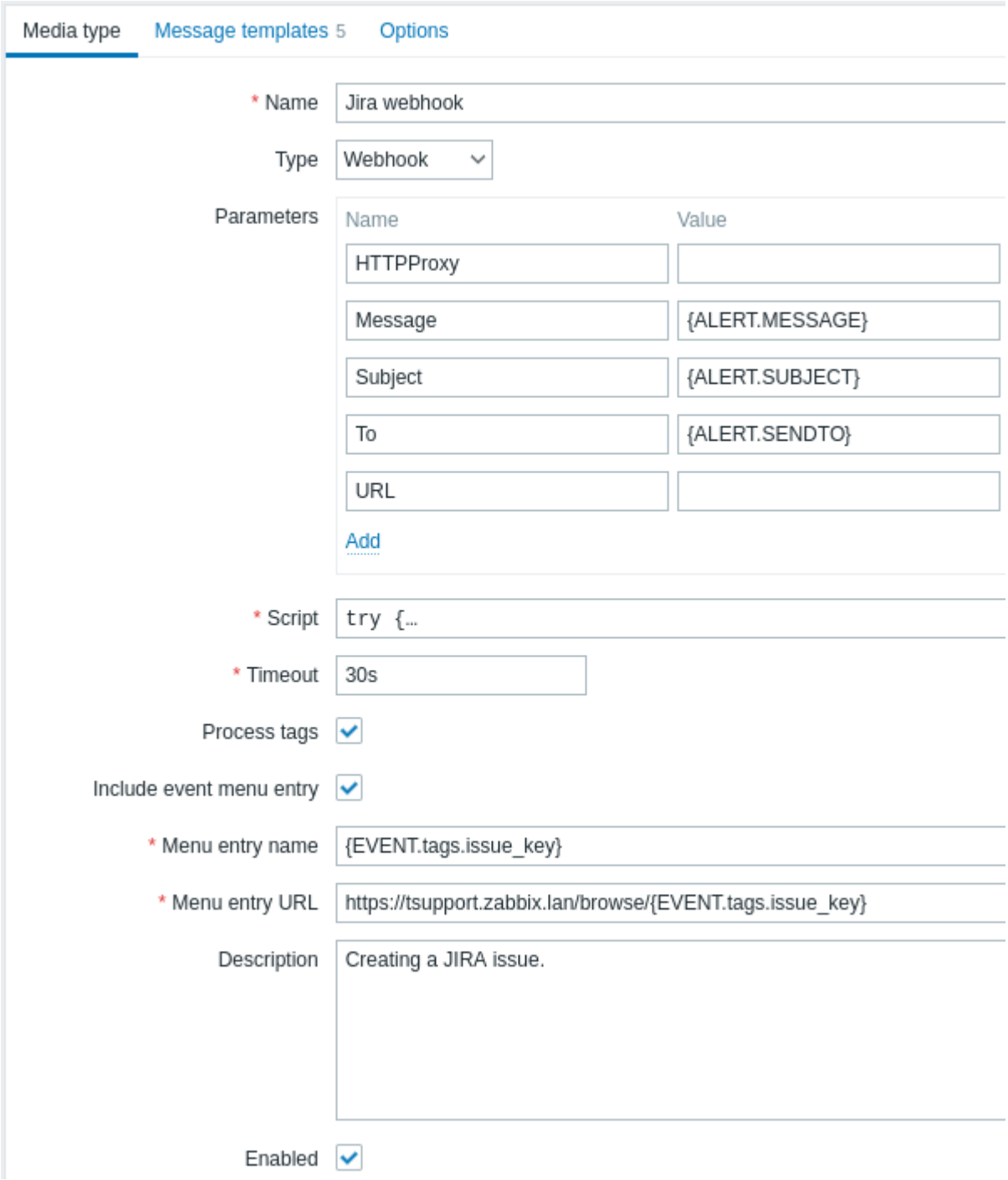
- 如果 webhook 使用标签来存储 ticket\message ID 以及后续的 update\resolve operations 操作, 这个 webhook 不应该用于单个问题事件的多个告警动作中。这适用于 Zabbix 提供的 Jira, Jira Service Desk, Mattermost, Opsgenie, OTRS, Redmine, ServiceNow, Slack, Zammad 和 Zendesk webhooks 以及大多数使用 *Include event menu entry* 选项的 webhook。如果操作或升级步骤属于同一个动作，允许在多个操作中使用 webhook。在不同的动作中使用 webhook 也是可以的，由于不同的筛选器条件，动作不会应用到相同的问题事件中。
- 当在动作中使用 webhook 用于internal events: 在动作操作配置中，选中 *Custom message* 复选框，输入自定义消息内容，否则通知不会被发送出去。

Webhook 脚本范例

概述

尽管 Zabbix 提供了大量现成的 webhook 集成，但您可能想要创建自己的 webhook。本节提供了自定义 webhook 脚本的示例. (在 *Script* 参数中使用). 有关其他 webhook 参数的说明，请参见webhook

Jira webhook (自定义)



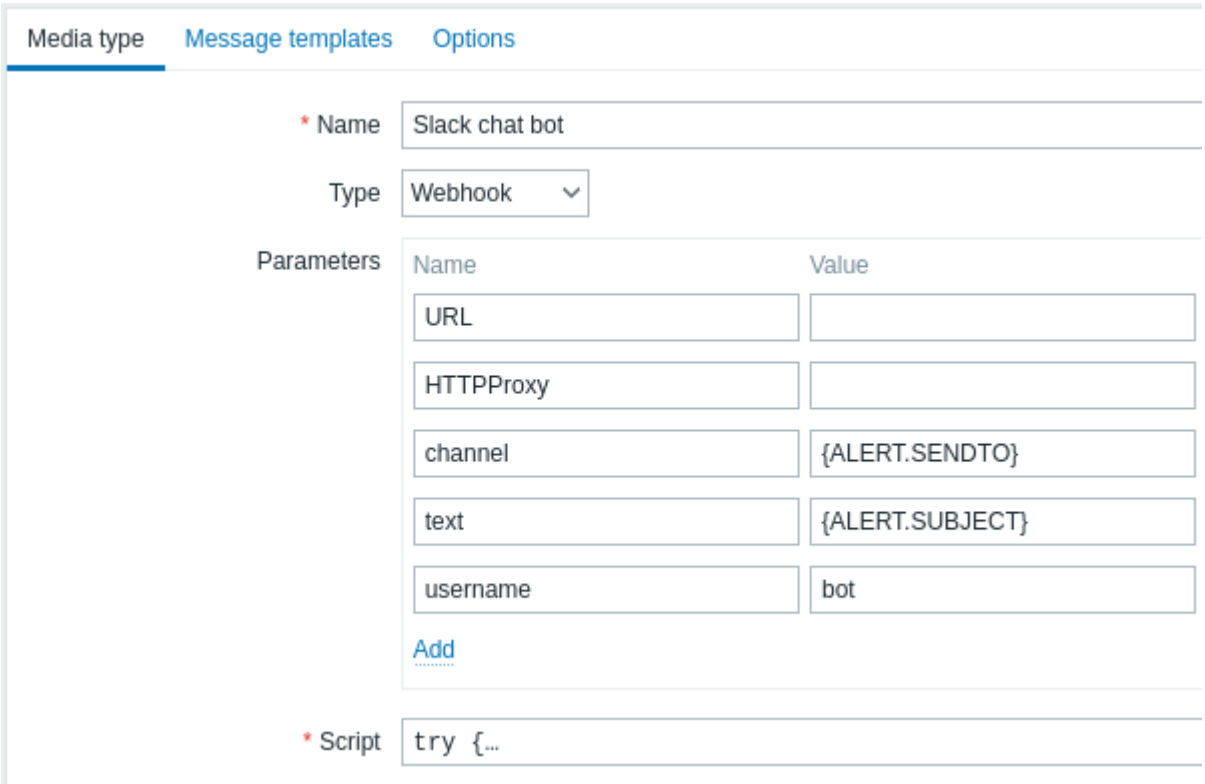
此脚本将创建一个 JIRA 问题，并返回关于所创建问题的一些信息。

```
try {
    Zabbix.log(4, '[ Jira webhook ] Started with params: ' + value);

    var result = {
            'tags': {
                'endpoint': 'jira'
            }
        },
        params = JSON.parse(value),
        req = new HttpRequest(),
        fields = {},
        resp;

    if (params.HTTPProxy) {
        req.setProxy(params.HTTPProxy);
    }
```

```
    req.addHeader('Content-Type: application/json');
    req.addHeader('Authorization: Basic ' + params.authentication);

    fields.summary = params.summary;
    fields.description = params.description;
    fields.project = {key: params.project_key};
    fields.issuetype = {id: params.issue_id};

    resp = req.post('https://tsupport.zabbix.lan/rest/api/2/issue/',
        JSON.stringify({"fields": fields})
    );

    if (req.getStatus() != 201) {
        throw 'Response code: ' + req.getStatus();
    }

    resp = JSON.parse(resp);
    result.tags.issue_id = resp.id;
    result.tags.issue_key = resp.key;

    return JSON.stringify(result);
}
catch (error) {
    Zabbix.log(4, '[ Jira webhook ] Issue creation failed json : ' + JSON.stringify({"fields": fields}));
    Zabbix.log(3, '[ Jira webhook ] issue creation failed : ' + error);

    throw 'Failed with error: ' + error;
}
```

Slack webhook (自定义)

此 webhook 将把 Zabbix 的通知转发到 Slack channel 中。




```
try {
    var params = JSON.parse(value),
        req = new HttpRequest(),
        response;

    if (params.HTTPProxy) {
```

```
        req.setProxy(params.HTTPProxy);
    }

    req.addHeader('Content-Type: application/x-www-form-urlencoded');

    Zabbix.log(4, '[ Slack webhook ] Webhook request with value=' + value);

    response = req.post(params.hook_url, 'payload=' + encodeURIComponent(value));
    Zabbix.log(4, '[ Slack webhook ] Responded with code: ' + req.Status() + '. Response: ' + response);

    try {
        response = JSON.parse(response);
    }
    catch (error) {
        if (req.getStatus() < 200 || req.getStatus() >= 300) {
            throw 'Request failed with status code ' + req.getStatus();
        }
        else {
            throw 'Request success, but response parsing failed.';
        }
    }

    if (req.getStatus() !== 200 || !response.ok || response.ok === 'false') {
        throw response.error;
    }

    return 'OK';
}
catch (error) {
    Zabbix.log(3, '[ Jira webhook ] Sending failed. Error: ' + error);

    throw 'Failed with error: ' + error;
}
```

2 动作

概述

如果您希望对产生的事件进行一些操作（例如发送通知），则需要配置动作。

可以根据所有支持类型的事件来定义动作：

- 触发器动作 - 当 trigger 的状态从 *OK* 变为 *PROBLEM* 或者从 *PROBLEM* 恢复到 *OK* 时
- 服务动作 - 当服务的状态从 *OK* 变为 *PROBLEM* 或者从 *PROBLEM* 恢复到 *OK* 时
- 自动发现动作 - 针对网络自动发现事件发生时
- 自动注册动作 - 当新的 agents 自动注册（或已注册主机元数据发生改变）时
- 内部动作 - 当监控项变成不支持状态或触发器进入未知状态时

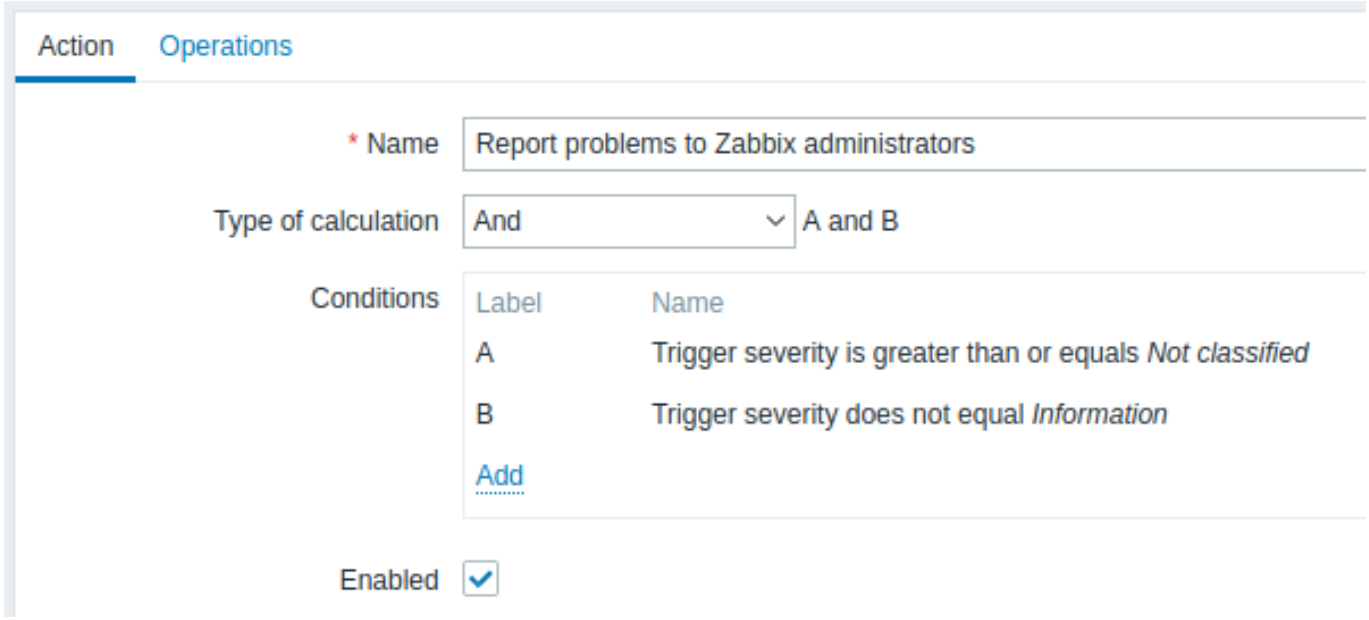
配置一个动作

要配置操作，请执行以下操作：

- ·转到配置 -> 动作并从子菜单中选择所需的操作类型（稍后您可以使用标题下拉菜单切换到另一种类型）
- ·点击创建动作
- ·命名动作
- ·选择执行操作的条件
- ·选择操作进行

注意可以在服务动作 部分配置服务操作。

一般动作属性：



所有必填输入字段都标有红色星号。

参数	说明
Name	唯一的动作名称。
Type of calculation	为操作条件（具有多个条件）选择评估选项： 和 - 所有条件必须满足 **Or** - 如果满足一个条件就足够了 **And/Or** - 两者的组合：AND 具有不同的条件类型，OR 具有相同的条件类型 ** 自定义表达式 ** - 用于评估操作条件的用户定义计算公式。
Conditions	操作条件列表。 单击添加添加新的条件。
Enabled	勾选复选框以启用该操作。否则，它将被禁用。

1 条件

概述

只有当事件满足了定义的条件集时，动作才会被执行。条件在配置action 时进行设置。

条件匹配区分大小写。

触发动作

以下条件可用于基于触发器的操作：

条件类型	支持的运算符	说明
Host group	等于 不等于	指定要排除的主机组或主机组。 等于 - 事件属于此主机组。 不等于 - 事件不属于此主机组。 指定父主机组会隐式选择所有嵌套的主机组。要仅指定父组，必须使用 不等于运算符另外设置所有嵌套组。
Template	等于 不等于	指定模板或要排除的模板。 等于 - 事件属于从该模板继承的触发器。 不等于 - 事件不属于从该模板继承的触发器。
Host	等于 不等于	指定主机或要排除的主机。 等于 - 事件属于此主机。 不等于 - 事件不属于此主机。
Tag name	等于 不等于 包含 不包含	指定事件标签或要排除的事件标签。 等于 - 事件有此标签 * * 不等于 **-** 事件没有这个标签 ** ** 包含 **-** 事件有一个包含这个字符串的标签 ** ** 不包含 ** - 事件没有包含这个字符串的标签

条件类型	支持的运算符	说明
Tag value	等于 不等于 包含 不包含	指定标签和值组合或要排除的事件标签和值组合。 等于 - 事件有这个标签和值 不等于 - 事件没有这个标签和值 包含 - 事件有一个包含这些字符串的标签和值 不包含 - 事件没有包含这些字符串的标签和值
Trigger	等于 不等于	指定触发器或要排除的触发器。 等于 - 事件由此触发器生成。 不等于 - 事件通过任何其他触发器生成，除了这个。
Trigger name	包含 不包含	指定触发器名称中的字符串或要排除的字符串。 包含 - 事件由触发器生成，名称中包含此字符串。 不包含 - 在触发器名称中找不到此字符串。 注意：输入的值将与展开所有宏的触发器名称进行比较。
Trigger severity	等于 不等于 大于或等于 小于或等于	指定触发器严重性。 等于 - 等于触发器严重性 不等于 - 不等于触发严重性 大于或等于 - 大于或等于触发严重性 小于或等于 - 小于或等于触发严重程度
Time period	在 不在	指定时间段或要排除的时间段。 在 - 事件时间在时间段内。 不在 - 事件时间不在时间段内。 请参阅时间段规范页面了解格式说明。 自 Zabbix 3.4.0 起开始支持用户宏 .
Problem is suppressed	否 是	如果由于主机维护问题被抑制（未显示），请指定。 否 - 问题未被抑制。 是 - 问题被抑制了。

发现操作

以下条件可用于基于发现的事件：

条件类型	支持的运算符	说明
Host IP	等于 不等于	指定 IP 地址范围或要为已发现主机排除的范围。 等于 - 主机 IP 在范围内。 ** 不等于 ** - 主机 IP 不在范围内。 可能有以下格式： 单个 IP：192.168.1.33 IP 地址范围：192.168.1-10.1-254 IP mask: 192.168.4.0/24 List: 192.168.1.1-254, 192.168.2.1-100, 192.168.2.200, 192.168.4.0/24 从 Zabbix 3.0.0 开始支持 list 格式的空格。
Service type	等于 不等于	指定已发现服务的服务类型或要排除的服务类型。 等于 - 匹配已发现服务。 不等于 - 与发现的服务不匹配。 可用的服务类型：SSH、LDAP、SMTP、FTP、HTTP、HTTPS （自 *Zabbix 2.2* 版本起可用）、POP、NNTP、IMAP、TCP、Zabbix agent，SNMPv1 agent、SNMPv2 agent、SNMPv3 agent、ICMP ping、telnet （自 *Zabbix 2.2* 版本起可用）。
Service port	等于 不等于	指定已发现服务的 TCP 端口范围或要排除的范围。 等于 - 服务端口在范围内。 ** 不等于 ** - 服务端口不在范围内。
Discovery rule	等于 不等于	指定发现规则或要排除的发现规则。 等于 - 使用此发现规则。 不等于 - 使用除此之外的任何其他发现规则。
Discovery check	等于 不等于	指定发现检查或要排除的发现检查。 等于 - 使用此发现检查。 不等于 - 使用除此之外的任何其他发现检查。
Discovery object	等于	指定发现的对象。 等于 - 等于发现的对象（设备或服务）。

条件类型	支持的运算符	说明
Discovery status	等于	**Up** - 匹配 “Host Up” 和 “Service Up” 事件 **Down** - 匹配 “Host Down” 和 “Service Down” 事件 ** 发现 ** - 匹配 “主机发现” 和 “服务发现” 事件 丢失 - 匹配 “主机丢失” 和 “服务丢失” 事件
Uptime/Downtime	大于或等于 小于或等于	“主机启动” 和 “服务启动” 事件的正常运行时间。“主机停机” 和 “服务停机” 事件的停机时间。 大于或等于 - 大于或等于。参数以秒为单位给出。 小于或等于 - 小于或等于。参数以秒为单位给出。
Received value	等于 不等于 大于等于 小于等于 包含 不包含	指定从 agent 接收的值（Zabbix，SNMP) 检查发现规则。字符串比较。如果为规则配置了多个 Zabbix agent 或 SNMP 检查，则会检查每个接收到的值（每个检查都会生成一个与所有条件匹配的新事件）。 等于 - 等于值。 不等于 - 不等于该值。 大于或等于 - 大于或等于该值。 小于或等于 - 小于或等于该值。 包含 - 包含子字符串。参数以字符串形式给出。 不包含 - 不包含子字符串。参数以字符串形式给出。
Proxy	等于 不等于	指定代理或要排除的代理。 等于 - 使用此代理。 不等于 - 使用除此之外的任何其他代理。

Note:
导致发现事件的发现规则中的服务检查不会同时发生。因此，如果在操作中为 “服务类型”、“服务端口” 或 “接收值” 条件配置了多个值，它们将一次与一个发现事件进行比较，但不会多个事件同时进行。因此，可能无法正确执行具有相同检查类型的多个值的操作。

自动注册操作

以下条件可用于基于活动 agent 自动注册的操作：

条件类型	支持的运算符	说明
Host metadata	包含 不包含 匹配 不匹配	指定主机元数据或要排除的主机元数据。 包含 - 主机元数据包含字符串。 不包含 - 主机元数据不包含字符串。 主机元数据可以在agent 配置文件中指定。 匹配 *** - ** 主机元数据与正则表达式匹配。** ** 不匹配 ** - 主机元数据与正则表达式不匹配。
Host name	包含 不包含 匹配 不匹配	指定主机名或要排除的主机名。 包含 - 主机名包含字符串。 不包含 - 主机名不包含字符串。 匹配 - 主机名匹配正则表达式。 不匹配 - 主机名与正则表达式不匹配。
Proxy	等于 不等于	指定代理或要排除的代理。 等于 - 使用此代理。 不等于 - 使用除此之外的任何其他代理。

内部事件动作

可以为基于内部事件的动作设置以下条件:

条件类型	支持的操作符	说明
事件类型	等于	处于 **“不支持”** 状态的监控项 - 匹配监控项从’ 正常’ 变为’ 不支持’ 状态的事件 处于 **“不支持”** 状态的底层自动发现规则 - 匹配底层自动发现规则从’ 正常’ 变为’ 不支持’ 状态的事件 处于 **“未知”** 状态的触发器 - 匹配触发器从’ 正常’ 变为’ 未知’ 状态的事件
主机组	等于 不等于	主机组或要排除的主机组 等于 - 属于该主机组的事件 不等于 - 不属于该主机组的事件
标签名称	等于 不等于 包含 不包含	标签名称或者要排除的标签名称 等于 - 具备该标签的事件 不等于 - 不具备该标签的事件 包含 - 标签名称中包含该字符串的事件 不包含 - 标签名称中不包含该字符串的事件
标签值	等于 不等于 包含 不包含	事件标签和值组合或要排除的标签和值组合 等于 - 具备该标签并且值等于给定值的事件 不等于 - 具备该标签但值不等于给定值的事件 包含 - 具备该标签并且值包含指定字符串的事件 不包含 - 具备该标签但值不包含指定字符串的事件
模板	等于 不等于	模板或要排除的模板 等于 - 属于从此模板继承的监控项/触发器/底层自动发现规则的事件 不等于 - 不属于从此模板继承的监控项/触发器/底层自动发现规则的事件
主机	等于 不等于	主机或要排除的主机 等于 - 属于此主机的事件 不等于 - 不属于此主机的事件

条件计算类型

计算条件的选项有以下几种：

- **And** - 必须满足所有条件

注意：当多个触发器被选择为 `Trigger =` 条件时，在它们之间是不允许使用”And” 来计算的。只能基于一个触发器的事件执行动作。

- **Or** - 只要满足一个条件就足够了
- **And/Or** - 两者的结合，AND 连接不同的条件类型，而 OR 连接相同的条件类型，例如：

Host group 等于 Oracle servers
Host group 等于 MySQL servers
Trigger name 包含’Database is down’
Trigger name 包含’Database is unavailable’

等价于

(Host group 等于 Oracle servers **or** Host group 等于 MySQL servers**) and (**Trigger name 包含’Database is down’ **or** Trigger name 包含’Database is unavailable’**)**

- **Custom expression** - 用户自定义动作条件的表达式。必须包含所有条件（以大写字母 A, B, C, ... 表示），可以包含空格、制表符、括号 ()、**and** (区分大小写), **or** (区分大小写), **not** (区分大小写)。

虽然前面关于 And/Or 的示例代表 (A or B) and (C or D)，但在自定义表达式中，您还有多种其他的计算方法：

(A and B) and (C or D)
(A and B) or (C and D)
((A or B) and C) or D
(not (A or B) and C) or not D
等等。

由于对象被删除导致动作被禁用的情况

如果在动作条件/操作中使用的某个对象（主机，模板，触发器等）被删除了，那么对应的条件/操作也会被删除，并且该动作将被禁用，以避免错误地执行该动作。用户可以重新启用动作。

当删除以下对象时会发生这种情况：

- 主机组（“主机组” 条件，特定主机组上的 “远程命令” 操作将被删除）；
- 主机（“主机” 条件，特定主机上的 “远程命令” 操作将被删除）；
- 模板（“模板” 条件，“链接到模板” 和 “从模板中取消链接” 操作将被删除）；
- 触发器（“触发器” 条件将被删除）；
- 自动发现规则（使用 “自动发现规则” 和 “自动发现检查” 条件时将被删除）。

注意: 如果远程命令有多个目标主机，我们删除了其中的一个，那么只有该主机将从目标列表中删除，操作本身将保留。但是，如果它是唯一的主机，那么操作也将被删除。” 链接到模板” 和” 从模板取消链接” 的操作也是一样。

当删除” 发送消息” 操作中使用的用户或用户组时，动作不会被禁用。

2 操作

概述

你可以为所有事件定义如下这些操作：

- 发送一条消息
- 执行一条远程命令

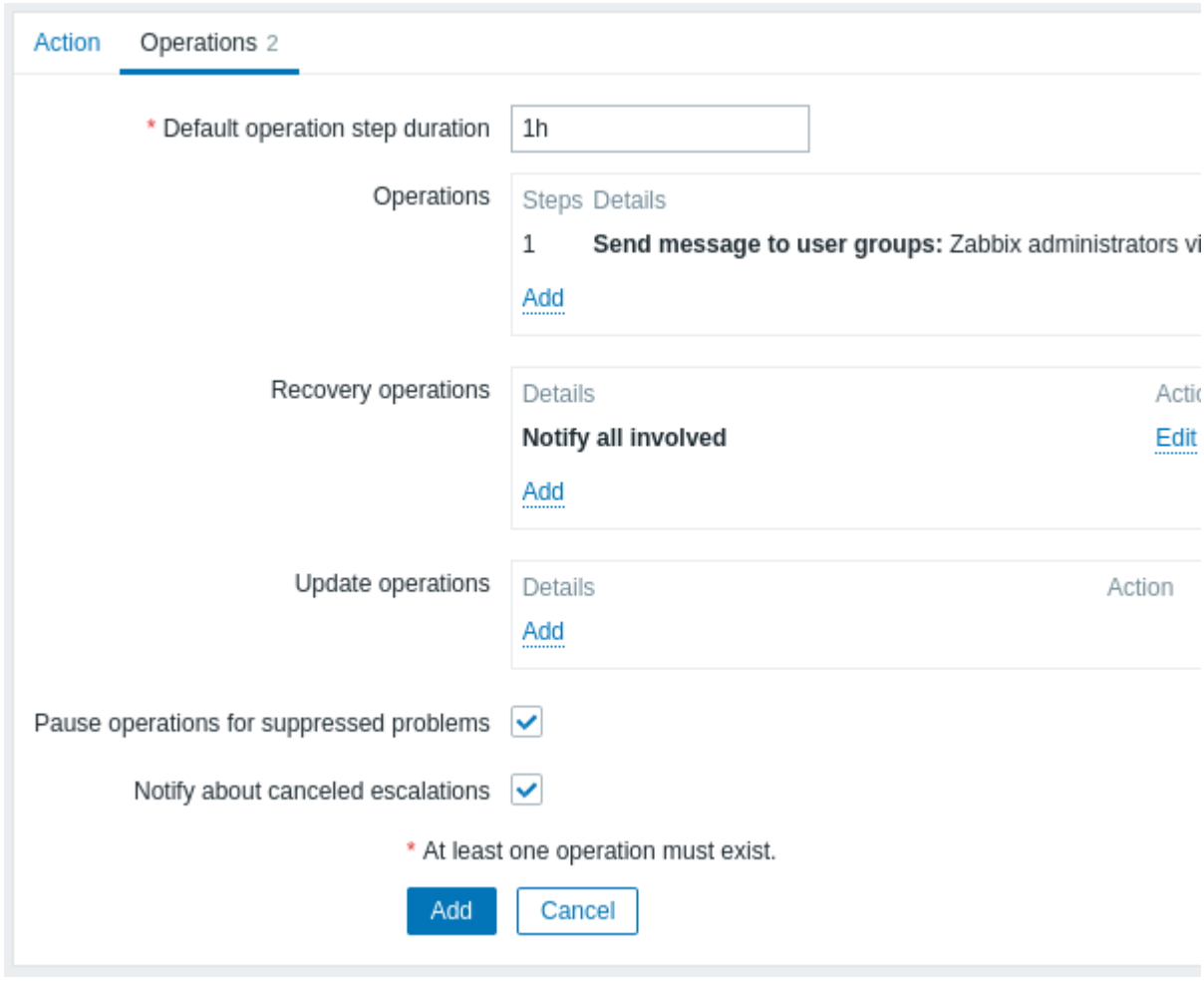
> **Attention:**
> 对用户定义的动作接受者，如果主机明确 “拒绝” 或者用户对主机完全没有定义的权限，Zabbix 服务器不会创建告警。

对于自动发现和自动注册事件，还有额外可用的操作：

- 添加主机
- 移除主机
- 启用主机
- 停用主机
- 添加到主机组
- 从主机组移除
- 链接到模板
- 取消到模板的链接
- 设置主机的资产模式

配置操作

要配置操作，请转到动作 配置中的 操作选项卡。



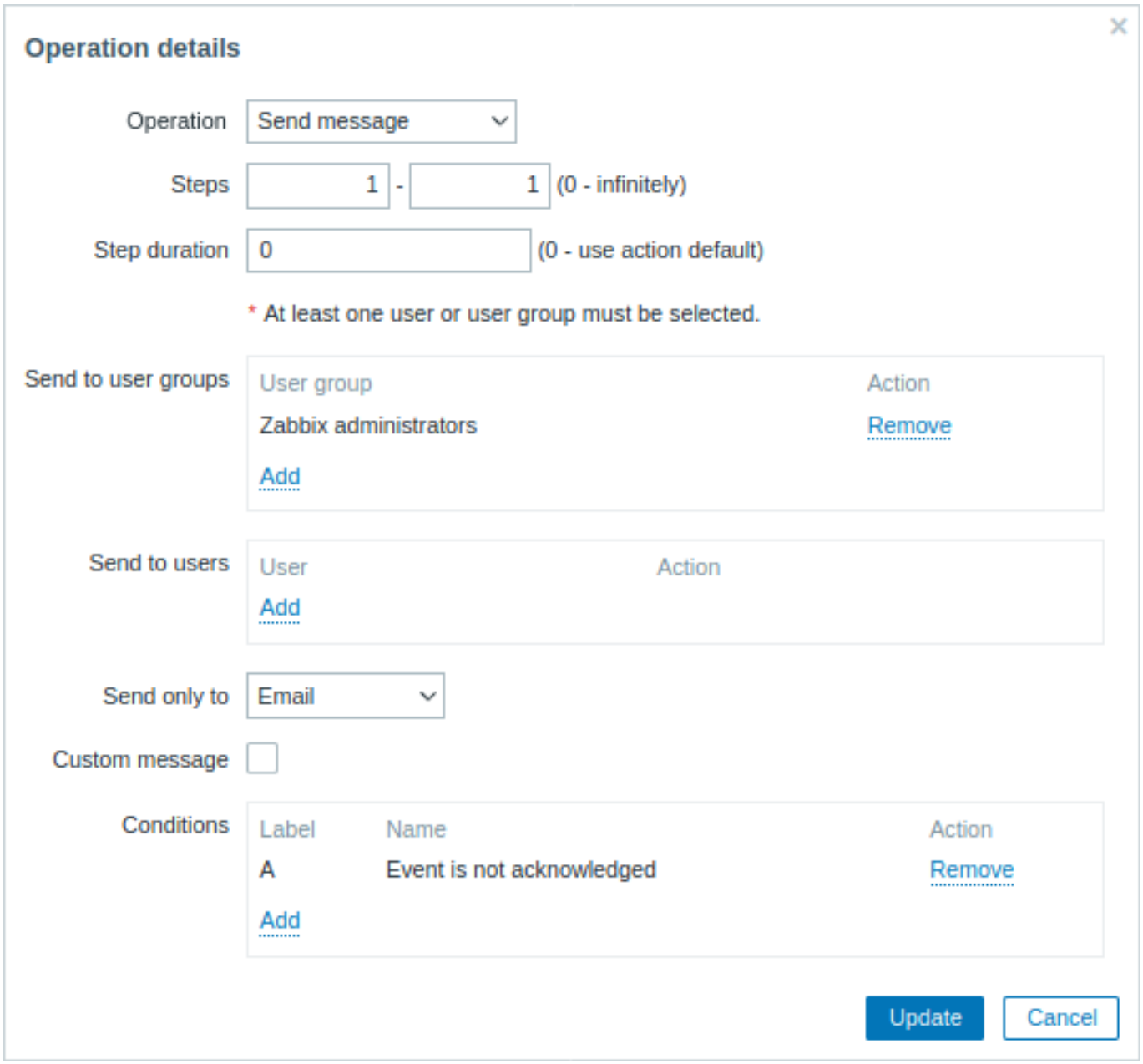
一般操作属性：

参数	说明
默认操作步骤持续时间	默认一个操作步骤的持续时间（60 秒到 1 周）。 例如，长达一个小时的步骤持续时间，意味着如果进行了该操作，一个小时后才会执行下个步骤。 支持时间后缀，例如 60s, 1m, 2h, 1d, since Zabbix 3.4.0. 从 Zabbix 3.4.0 开始支持用户宏
操作	显示动作的操作（如果有），以及以下详细信息： 步骤 - 操作分配到的升级步骤 详细信息 - 操作类型及其收件人/目标。 操作列表还显示使用的媒体类型（电子邮件、短信或脚本）以及通知收件人的姓名（在用户名后的括号中）。 * * 开始于 **-** 事件发生后多长时间执行操作 ** ** 持续时间（秒） **-** 显示步骤持续时间。如果步骤使用默认持续时间，则显示默认，如果使用自定义持续时间，则显示时间。** ** 动作 ** - 显示用于编辑和删除操作的链接。
恢复操作	显示动作的操作（如果有），以及以下详细信息： 详细信息 - 操作类型及其接收者/目标。 操作列表还显示媒体类型（电子邮件、短信或脚本）以及通知收件人的姓名（在用户名后的括号中）。 动作 - 显示用于编辑和删除操作的链接。
更新操作	显示动作的操作（如果有），以及以下详细信息： 详细信息 - 操作类型及其接收者/目标。 操作列表还显示媒体类型（电子邮件、短信或脚本）以及通知收件人的姓名（在用户名后的括号中）。 动作 - 显示用于编辑和删除操作的链接。
被抑制的问题暂停操作	标记此复选框以在维护期间延迟操作的开始。在维护之后，当操作开始时，包括维护期间事件内的所有操作都会执行。 请注意，此设置仅影响问题升级；恢复和更新操作不会受到影响。 如果取消选中此复选框，即使在维护期间也将立即执行操作。 此选项不适用于服务动作。
通知已取消的升级	取消标记此复选框以禁用有关已取消升级的通知（当主机、监控项、触发器或动作被禁用时）。

所有必填输入字段都标有红色星号。

要配置新操作的详细信息，请单击操作块中的 Add。要编辑现有操作，请单击操作旁边的 Edit。将打开一个弹出窗口，您可以在其中编辑操作步骤详细信息。

操作详情



参数		说明
操作		选择操作： 发送消息 - 向用户发送消息 < 远程命令名称 > - 执行远程命令。 如果先前在全局脚本 中定义了动作操作作为其范围，则命令可被执行。 更多操作可用于基于发现和自动注册的事件（见上文）。
步骤 步骤持续时间 操作类型：发送消息		
	发送给用户组	点击添加选择要将消息发送到的用户组。 用户组对主机必须至少具有 “读取”权限才能收到通知。
	发送给用户	点击添加选择用户发送消息。 用户对主机必须至少具有 “读取”权限才能收到到通知。

参数		说明
	仅发送至	将消息发送至所有定义的媒体类型或仅选定的媒体类型。
	自定义消息	如果选中，则可以配置自定义消息。对于通过webhooks的内部事件通知，自定义消息是必需的。
	主题	自定义消息的主题。主题可能包含宏。它被限制为 255 个字符。
	消息	自定义消息。该消息可能包含宏。它被限制为一定数量的字符，具体取决于数据库的类型（有关更多信息，请参阅发送消息）。
操作类型：远程命令		
	目标列表	选择要在其上执行命令的目标： 当前主机 - 命令在导致问题事件的触发器的主机上执行。如果触发器中有多个主机，此选项将不起作用。 主机 - 选择要在其上执行命令的主机。 主机组 - 选择执行命令的主机组。指定父主机组会隐式选择所有嵌套的主机组。因此，远程命令也将在来自嵌套组的主机上执行。 主机上的命令只执行一次，即使主机匹配不止一次（例如来自多个主机组；单独和来自主机组）。 如果在 Zabbix server 上执行自定义脚本，则目标列表没有意义。在这种情况下选择更多目标只会导致脚本在服务器上执行更多次。 请注意，对于全局脚本，目标选择还取决于全局脚本配置。 目标列表选项不适用于服务操作，因为在这种情况下，远程命令始终在 Zabbix 服务器上执行。
条件		执行操作的条件： 不确认 - 仅当事件未被确认时 确认 - 仅当事件被确认时。 条件 选项不适用于 服务动作。

完成后，单击添加将操作添加到 操作列表中。

1 发送消息

概述

Zabbix 提供发送消息作为主要操作之一的原因是，发送消息是通知人们有关某问题的最佳方式。

配置

为了能够从 Zabbix 发送和接收消息你必须：

- 定义发送消息的媒体

Warning:
如果要接收非触发器事件如发现、主动代理自动注册或内部事件，必须在用户媒体配置 中勾选默认触发器严重性（‘Not classified’）。

- 配置发送消息到已经定义的媒体的动作

Attention:
Zabbix 只向对产生事件的主机上至少一个触发器表达式有 ‘read’ 权限的用户发送消息。

你可以配置自定义方案使用升级发送消息。

Zabbix 发送的数据是 UTF-8 格式的（主题只包含 ASCII 字符不支持 UTF-8 编码），消息的主题和内容是遵循 'SMTP/MIME e-mail' 格式采用 base64 编码的。为了成功从 Zabbix 接收和读取邮件，邮件服务器和客户端必须支持标准 'SMTP/MIME e-mail' 格式。

宏扩展后的消息限制与远程命令消息限制相同。

跟踪消息

可以在 *Monitoring* → *Problems* 查看已发送消息的状态。

在 *Actions* 列可以查看已采取动作的汇总信息。绿色数字代表已发送消息，红色代表失败，*In progress* 表示动作已发起，*Failed* 表示动作没有执行成功。

如果点击时间时间查看事件详情，根据事件，*Message actions* 包含（或不包含）已发送消息的详细信息。

在 *Reports* → *Action log* 可以看到所有配置了动作的事件的已发生动作的详细信息。

2 远程命令

概述

使用远程命令，您可以预定义在某些情况下会在受监控主机上自动执行的命令。

因此远程命令是智能主动监控的强大机制。

在该功能最显而易见的用途中，你可以尝试：

- 自动重启某些没有响应的应用（web 服务器、中间件、CRM）
- 使用 IPMI 'reboot' 命令重启某些确实无法响应的服务器。
- 磁盘空间不足时自动释放（删除旧文件、清理）
- 根据 CPU 负载将 VM 从一台物理机迁移到另一台
- CPU（磁盘、内存或其他）资源不足时向云环境添加新节点

为远程命令配置操作与发送消息类似，唯一的区别是 Zabbix 将执行一个命令而不是发送消息。

远程命令可以由 Zabbix Server、Proxy 或 agent 执行。Zabbix agent 上的远程命令可以由 Zabbix Server 或通过 Proxy 执行。Zabbix agent 和 Zabbix Proxy 上的远程命令默认都是禁用的。它们可以通过以下方式启用：

- 在 agent 的配置中添加一个 `AllowKey=system.run[*]` 参数；
- 在代理的配置中将 `EnableRemoteCommands` 参数设为 '1'。

由 Zabbix 服务器执行的远程命令按照命令执行所述运行，包括退出代码检查。

即使目标主机在维护状态，远程命令也会执行。

远程命令限制

解析所有宏之后远程命令的限制取决于数据库和字符集（存储非 ASCII 字符需要一字节以上）：

数据库	字符限制	字节限制
MySQL	65535	65535
Oracle Database	2048	4000
PostgreSQL	65535	not limited
SQLite (only Zabbix proxy)	65535	not limited

以下教程提供了设置远程命令的分步指导。

配置

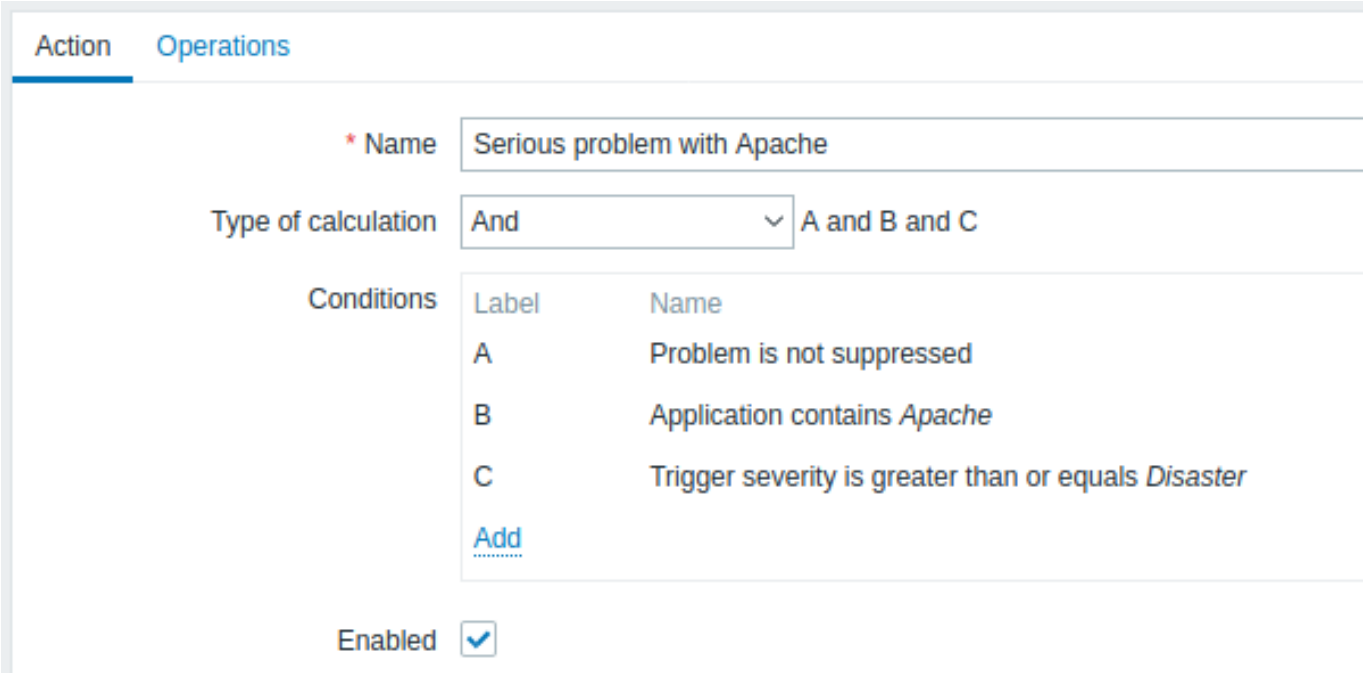
在 Zabbix agent 上执行远程命令（自定义脚本）必须先在 agent配置中启用。

必须确保添加了 `AllowKey=system.run[*]` 参数，并重启 agent 后台驻留程序。

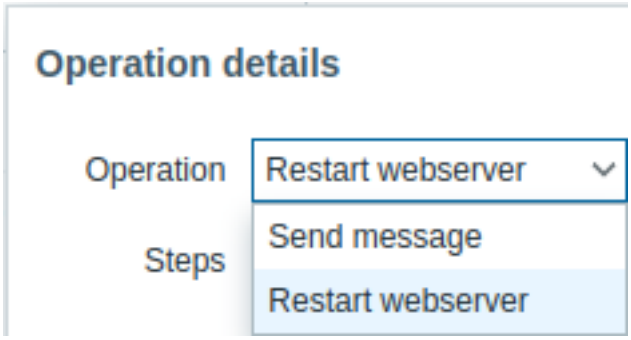
> **Attention:**
> Zabbix active agent 上远程命令不不生效。

然后，在 *Configuration* → *Actions* 配置新动作时：

- 定义适当的条件。在本例中，设置动作由 Apache 的任何灾难问题激活：



- 在 *Operations*选项卡，在 Operations/Recovery/Update 的操作块里点击
- 从 Operation 下拉菜单选择一个预定义的脚本



- 为脚本选择目标列表

预定义脚本

动作可用的所有脚本（webhook,、脚本、SSH、Telnet、IPMI）定义在全局脚本。

例如：

```
sudo /etc/init.d/apache restart
```

此例中，Zabbix 会尝试重启一个 Apache 进程。确保这个命令在 Zabbix agent 上执行（点击 *Zabbix agent* 按钮为 *Execute on*）。

Attention:
注意 **sudo** 的用法——Zabbix 用户默认没有权限重启系统服务。请查看以下关于配置 **sudo** 的提示。

Note:
Zabbix agent 应该运行在远程主机上并接受入方向的连接。Zabbix agent 在后台执行命令。

Zabbix agent 上的远程命令由 system.run[,nowait] 键执行时没有 timeout，并且不检查执行的结果。在 Zabbix Server 和 Zabbix Agent 上，远程命令执行时有 timeout，在 zabbix_server.conf 或 zabbix_proxy.conf 文件里配置 TrapperTimeout 参数，并且检查 执行的结果。

访问权限

确保 'zabbix' 用户拥有执行配置的命令的权限。用户可能有兴趣使用 **sudo** 给特权命令访问权限。要配置访问权限，用 root 执行以下命令：

```
# visudo
```

可以在 *sudoers* 文件使用示例的几行：

```
# allows 'zabbix' user to run all commands without password.
zabbix ALL=NOPASSWD: ALL

# allows 'zabbix' user to restart apache without password.
zabbix ALL=NOPASSWD: /etc/init.d/apache restart
```

Note:
在有些操作系统中，*sudoers* 文件禁止非本地用户执行命令，可以在*/etc/sudoers* 文件中取消注释 **requiretty** 选项。

有多个接口的远程命令

如果目标系统有多个所选类型（Zabbix agent 或 IPMI）的接口，远程命令会在默认的接口执行。

除 Zabbix agent 接口以外，可以在其它接口通过 SSH 或 Telnet 执行远程命令。可用的接口按如下顺序选择：

- Zabbix agent default interface
- SNMP default interface
- JMX default interface
- IPMI default interface

IPMI 远程命令

对于 IPMI 远程命令应使用如下语法：

```
<command> [<value>]
```

其中

- <command>——表示一个没有空格的 IPMI 命令。
- <value>——其值为 ‘on’、‘off’ 或任何无符号整数，<value> 是可选参数。

示例

可能在动作的操作中使用的全局脚本的示例。

示例 1

在特定条件下重启 Windows。

为了根据 Zabbix 检测到的问题自动重启 Windows，定义如下脚本：

脚本参数	值
Scope	’Action operation’
Type	’Script’
Command	c:\windows\system32\shutdown.exe -r -f

示例 2

用 IPMI 控制重启主机。

脚本参数	值
Scope	’Action operation’
Type	’IPMI’
Command	reset

示例 3

使用 IPMI 控制关闭主机电源。

脚本参数	值
Scope	’Action operation’
Type	’IPMI’
Command	power off

3 附加操作

概述

在本节你将看到对发现或自动注册事件的附加操作。

添加主机

主机在发现过程中中添加，一旦发现立即添加，而非在发现过程结束后添加。

> **Note:**
> 网络发现会因为太多不可达的主机或服务而用时很长，请耐心等待或使用合理 IP 地址范围。

添加主机时，其名称由标准的 ** gethostbyname ** 函数决定。如果可以解析主机，则使用解析名称。如果没有，则使用 IP 地址。此外，如果必须使用 IPv6 地址作为主机名，则所有“：”（冒号）将被替换为“_”（下划线），因为主机名中不允许冒号。

> **Attention:**
> 如果通过代理执行发现，当前主机名查找仍然发生在 Zabbix 服务器上。

> **Attention:**
> 如果一个新发现的主机与 Zabbix 的配置中已有的主机同名，1.8 之前的版本将添加一个同名的主机。Zabbix 1.8.1 及更高版本将 ** _ N 添加到主机名中， N** 是从 2 开始递增的正整数。

4 在消息中使用宏

概述

在消息主题和消息文本中，您可以使用宏来更有效地报告问题。

除了许多内置宏外，还支持用户宏 和表达式宏。Zabbix 支持可用的宏的完整列表 。

示例

此处的示例说明了如何在消息中使用宏。

示例 1

消息主题：

```
Problem: {TRIGGER.NAME}
```

当收到消息时，消息主题将被替换为类似如下内容：

```
Problem: Processor load is too high on Zabbix server
```

示例 2

消息：

```
Processor load is: {?last(/zabbix.zabbix.com/system.cpu.load[,avg1])}
```

当收到消息时，将被替换为类似如下内容：

```
Processor load is: 1.45
```

示例 3

消息：

```
Latest value: {?last(/{HOST.HOST}/{ITEM.KEY})}
MAX for 15 minutes: {?max(/{HOST.HOST}/{ITEM.KEY},15m)}
MIN for 15 minutes: {?min(/{HOST.HOST}/{ITEM.KEY},15m)}
```

当收到消息时，将被替换为类似如下内容：

```
Latest value: 1.45
MAX for 15 minutes: 2.33
MIN for 15 minutes: 1.01
```

示例 4

消息：

```
http://<server_ip_or_name>/zabbix/tr_events.php?triggerid={TRIGGER.ID}&eventid={EVENT.ID}
```

当收到消息时，将会包含 *Event details* 页面的链接，该页面提供该事件、它的触发器和近期由相同触发器产生的事件列表。

示例 5

在触发器表达式中通知来自多个主机的值。

消息：

```
Problem name: {TRIGGER.NAME}
Trigger expression: {TRIGGER.EXPRESSION}

1. Item value on {HOST.NAME1}: {ITEM.VALUE1} ({ITEM.NAME1})
2. Item value on {HOST.NAME2}: {ITEM.VALUE2} ({ITEM.NAME2})
```

当收到消息时，将被替换为类似如下内容：

```
Problem name: Processor load is too high on a local host
Trigger expression: last(/Myhost/system.cpu.load[percpu,avg1])>5 or last(/Myotherhost/system.cpu.load[perc

1. Item value on Myhost: 0.83 (Processor load (1 min average per core))
2. Item value on Myotherhost: 5.125 (Processor load (1 min average per core))
```

示例 6

在同一个恢复消息中接收故障事件和恢复事件的详情：

消息：

```
Problem:

Event ID: {EVENT.ID}
Event value: {EVENT.VALUE}
Event status: {EVENT.STATUS}
Event time: {EVENT.TIME}
Event date: {EVENT.DATE}
Event age: {EVENT.AGE}
Event acknowledgment: {EVENT.ACK.STATUS}
Event update history: {EVENT.UPDATE.HISTORY}

Recovery:

Event ID: {EVENT.RECOVERY.ID}
Event value: {EVENT.RECOVERY.VALUE}
Event status: {EVENT.RECOVERY.STATUS}
Event time: {EVENT.RECOVERY.TIME}
Event date: {EVENT.RECOVERY.DATE}
Operational data: {EVENT.OPDATA}
```

当收到消息时，宏将被替换为类似如下内容：

```
Problem:

Event ID: 21874
Event value: 1
Event status: PROBLEM
Event time: 13:04:30
Event date: 2018.01.02
Event age: 5m
Event acknowledgment: Yes
Event update history: 2018.01.02 13:05:51 "John Smith (Admin)"
Actions: acknowledged.

Recovery:
```

```
Event ID: 21896
Event value: 0
Event status: OK
Event time: 13:10:07
Event date: 2018.01.02
Operational data: Current value is 0.83
```

Attention:
自 Zabbix 2.2.0 开始支持原始故障事件和恢复事件单独的通知宏。

3 恢复操作

概述

恢复操作允许您在问题得到解决时收到通知。

恢复操作支持消息和远程命令。虽然可以添加多个操作，但不支持升级 - 所有操作都分配给一个步骤，因此将同时执行。

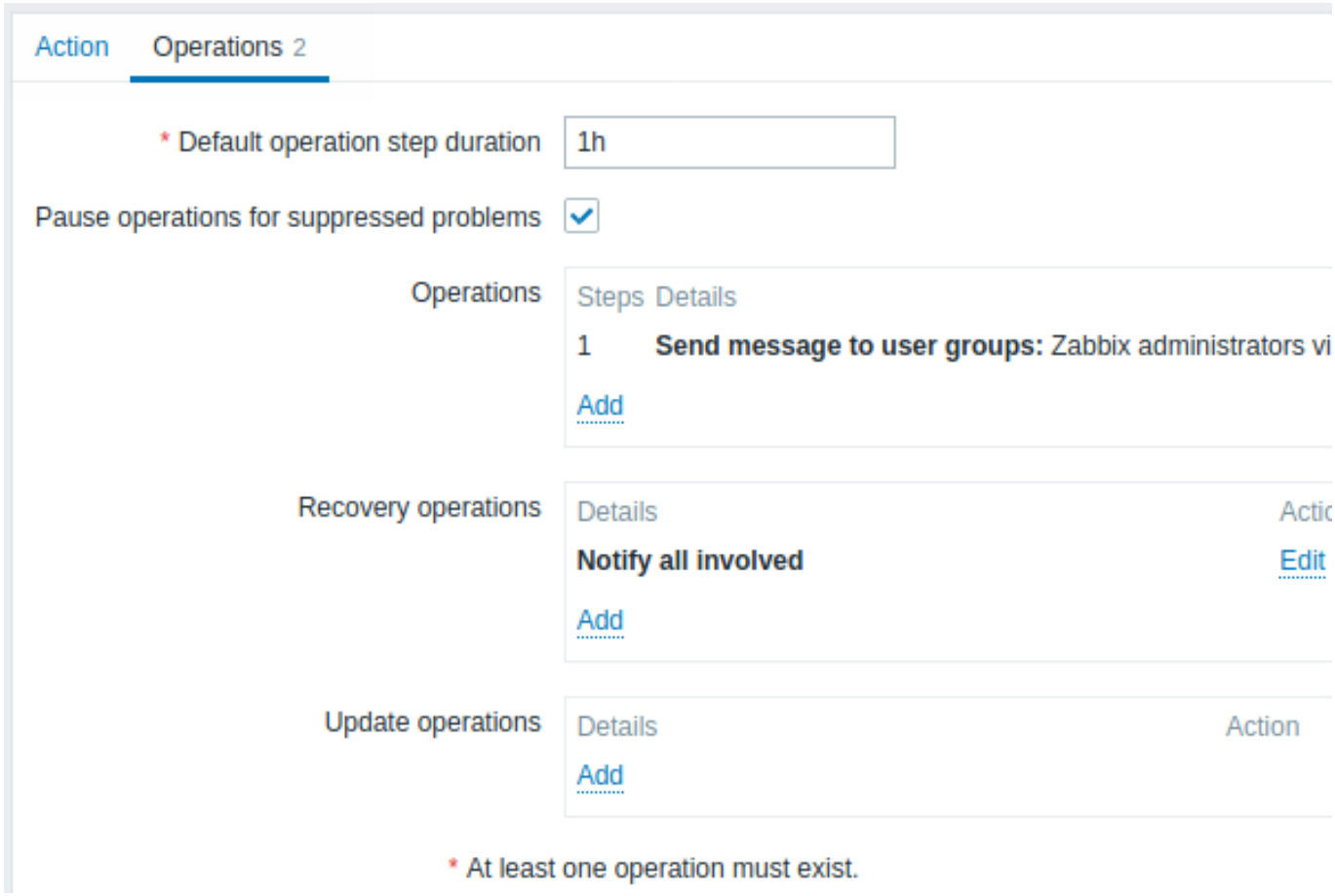
用例

恢复操作的一些用例如下：

1. 通知所有收到问题通知的用户恢复：
   - 选择通知所有相关人员作为操作类型。
2. 恢复后有多个操作：发送通知和执行远程命令：
   - 添加发送消息和执行命令的操作类型。
3. 在外部帮助台/工单系统中打开一个工单，并在问题解决后将其关闭：
   - 创建一个与服务台系统通信的外部脚本。
   - 创建一个具有执行此脚本并因此打开工单的操作的动作。
   - 进行恢复操作，使用其他参数执行此脚本并关闭票证。
   - 使用 {EVENT.ID} 宏来引用原始问题。

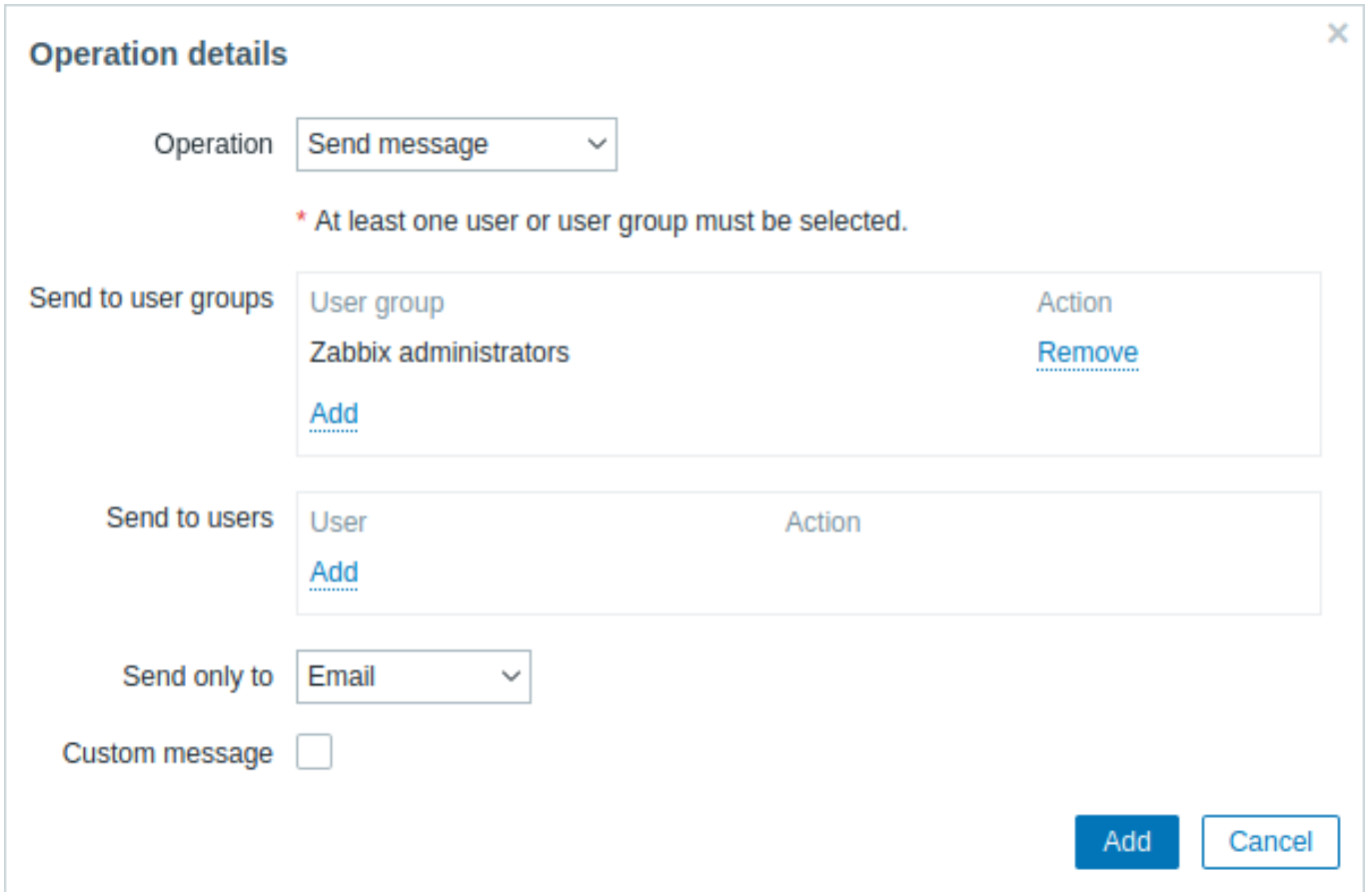
配置恢复操作

要配置恢复操作，请转到动作 配置中的 操作选项卡。



要配置新恢复操作的详细信息，请单击恢复操作块中的 Add。要编辑现有操作，请单击操作旁边的 Edit。将打开一个弹出窗口，您可以在其中编辑操作步骤详细信息。

恢复操作详情



三种操作类型可用于恢复事件：- 发送消息 - 向指定用户发送恢复消息 - 通知所有相关人员 - 向收到问题事件通知的所有用户发送恢复消息

- **<** 远程命令名称 **>** - 执行远程命令。如果先前在全局脚本 中定义并选择 动作操作作为其范围，则命令可用于执行。

每种操作类型的参数如下所述。所有必填输入字段都标有红色星号。完成后，单击 添加将操作添加到 恢复操作列表中。

Note:
请注意，如果在没有指定自定义消息的情况下在多个操作类型中定义相同的收件人，则不会发送重复的通知。

操作类型：发送消息

参数	说明
Send to user groups	点击添加选择要将恢复消息发送到的用户组。 用户组必须至少具有 “读取”权限 给主机以便得到通知。
Send to users	点击添加选择要将恢复消息发送到的用户。 用户必须至少具有对主机的 “读取”权限 为了得到通知。
Send only to	将默认恢复消息发送至所有定义的媒体类型或仅选定的媒体类型。
Custom message	如果选中，可以定义自定义消息。
Subject	自定义消息的主题。主题可能包含宏。
Message	自定义消息。该消息可能包含宏。

操作类型：远程命令

参数	说明
Target list	选择要在其上执行命令的目标： 当前主机 - 命令在导致问题事件的触发器的主机上执行。如果触发器中有多个主机，此选项将不起作用。 主机 - 选择要在其上执行命令的主机。 主机组 - 选择执行命令的主机组。指定父主机组会隐式选择所有嵌套的主机组。因此远程命令也将在来自嵌套组的主机上执行。 主机上的命令只执行一次，即使主机匹配不止一次（例如来自多个主机组；单独和来自主机组）。 如果命令是在 Zabbix server 上执行的，目标列表是没有意义的。在这种情况下选择更多目标只会导致命令在服务器上执行更多次。 请注意，对于全局脚本，目标选择还取决于全局脚本配置.

操作类型：通知所有相关人员

参数	描述
自定义消息	如果选中，则可以定义自定义消息。
主题	自定义消息的主题。主题可能包含宏。
消息	自定义消息，消息可能包含宏。

4 更新操作

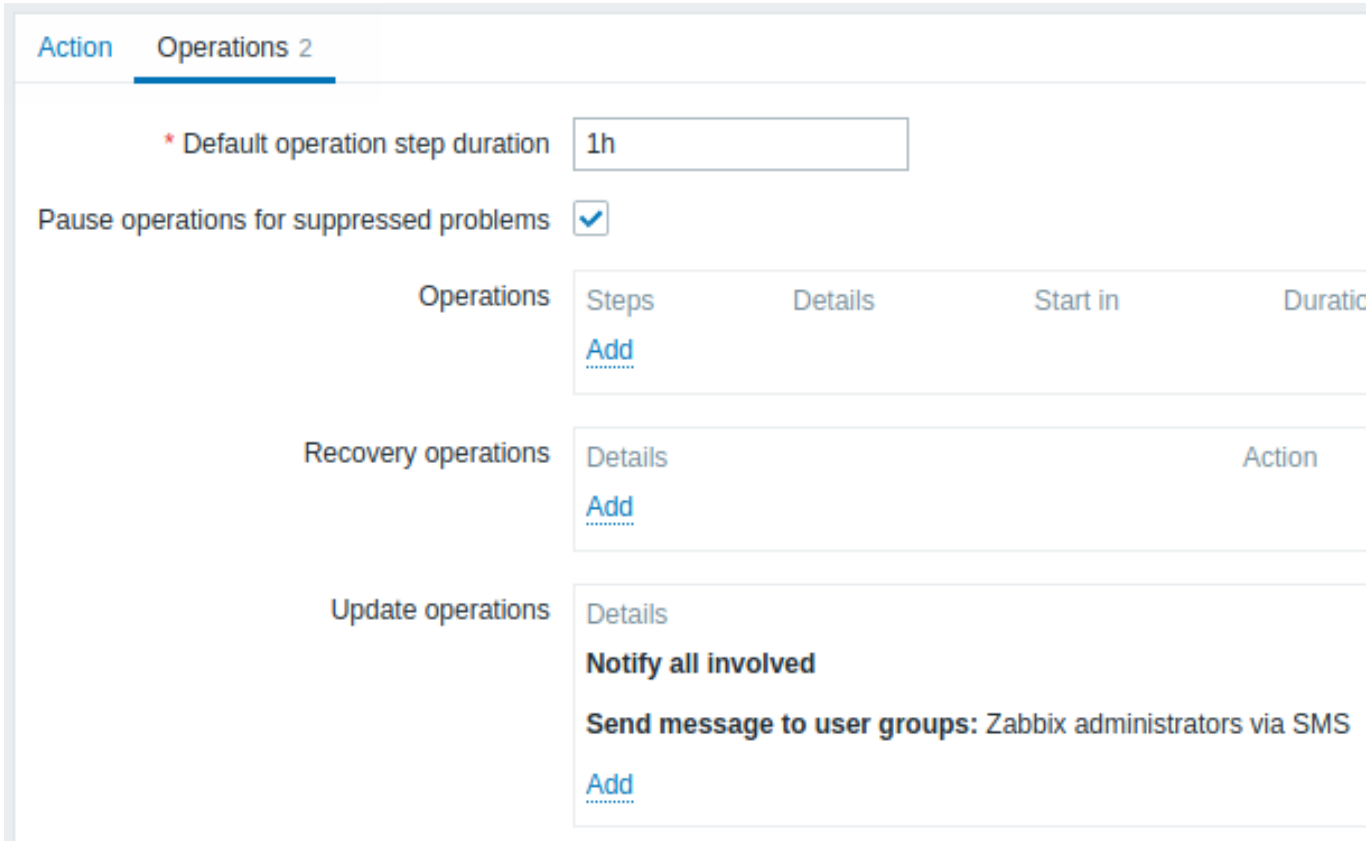
概述

更新操作在以下事件源中可用：- 触发器-当问题被其他用户更新时，即评论、确认、严重性已更改、关闭 (手动)；- 服务-当服务的严重性已更改，但服务仍未恢复时。

更新操作中同时支持消息和远程命令。虽然可以添加多个操作，但不支持升级-所有操作都分配给单个步骤，因此将同时执行。

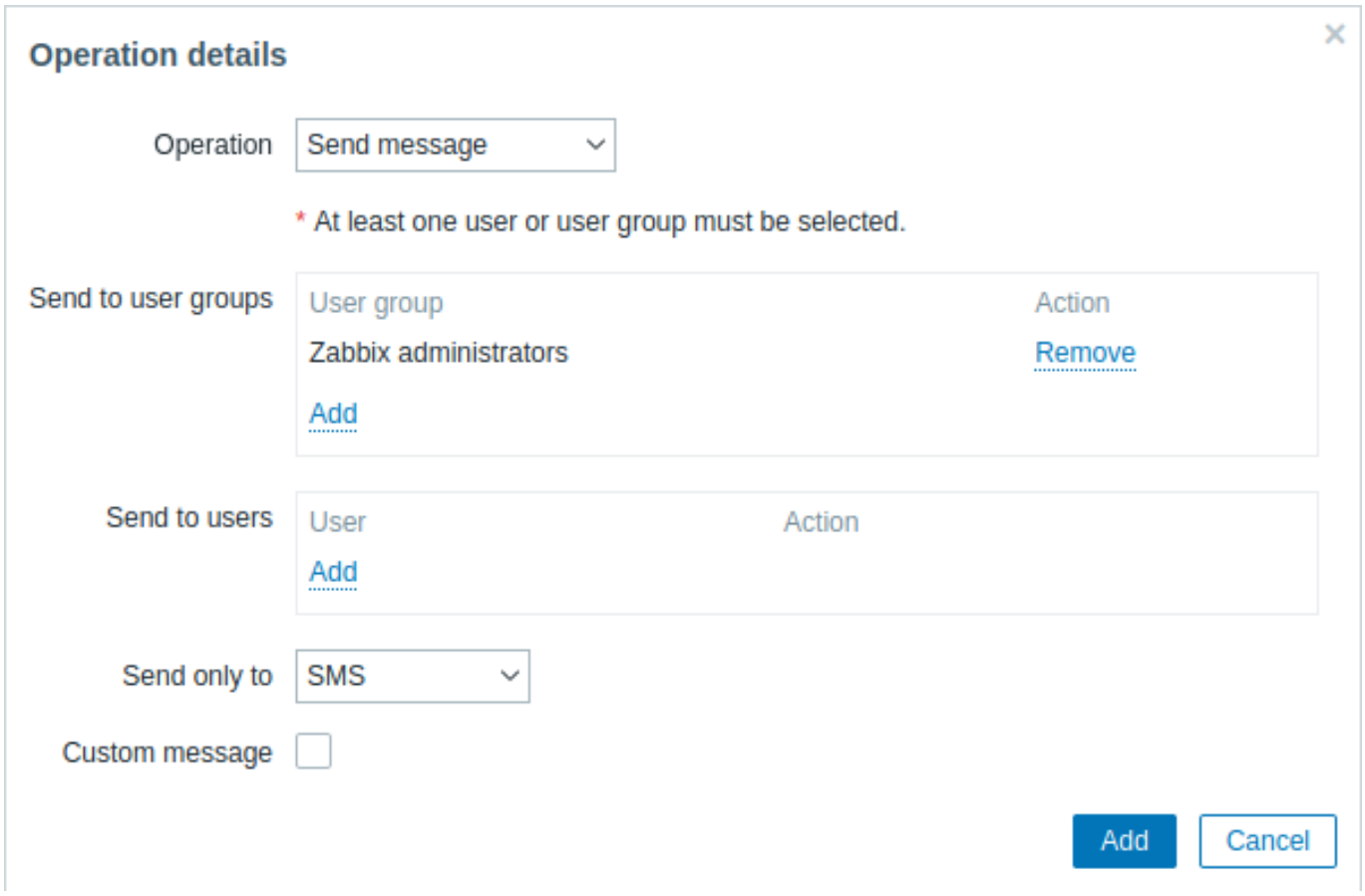
配置更新操作

要配置更新操作，请转到操作中的配置选项卡。



要配置新更新操作的详细信息，请在更新操作块中单击Add。要编辑现有操作，请在下个选项卡中单击Edit。将会打开一个弹出窗口，您可以在其中编辑操作步骤细节。

更新操作提供与恢复操作 相同的参数集。



5 通知升级

概述

通过 Escalations，您可以创建发送通知或执行远程命令的自定义场景。

实际应用中，这意味着：

- 用户可以立即收到新问题通知
- 通知可以重复，直到问题解决
- 发送通知可以延时
- 通知可以升级到另一个 “较高” 的用户组
- 可以立即执行远程命令，或者长时间不解决问题

操作会根据升级步骤进行通知升级。每一步都有一段时间。

您可以定义默认持续时间和单个步骤的自定义持续时间。一个升级步骤的最短持续时间为 60 秒。

您可以从任何步骤开始执行操作，例如发送通知或执行命令。第一步是立即采取行动。如果要延迟操作，可以将其分配给稍后的步骤。对于每个步骤，可以定义几个操作。

通知升级步骤的数量不受限制。

配置操作是即可定义 Escalations. Escalations 仅对问题操作支持，而不是恢复。

升级行为的其他方面

让我们考虑一下在不同情况下会发生什么。

包含几个升级步骤。

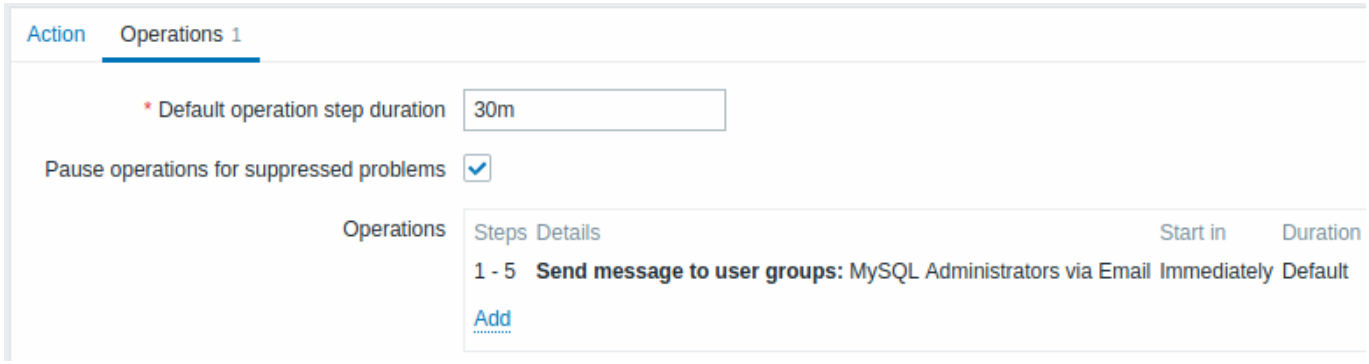
情境	行为
在发送初始问题通知后，相关主机进入维护状态	取决于操作配置中的暂停操作抑制问题设置，所有剩余的升级步骤都会在维护期间造成延迟的情况下或不延迟地执行。 维护期不会取消操作
在时间段操作条件中定义的时间段在发送初始通知后结束	执行所有剩余的升级步骤。时间段条件不能停止操作；它对操作何时启动/未启动而非操作具有影响
问题在维护期间开始，并在维护结束后继续 *(未解决)*	根据操作配置中的暂停被抑制问题的操作设置，所有升级步骤从维护结束时起或立即执行
* 问题在无数据维护期间开始，并在维护结束后继续 (未解决)	必须等待触发，然后执行所有上报步骤
不同的升级紧随其后并重叠	每个新升级的执行都会取代上一个升级，但至少有一个升级步骤总是在上一个升级上执行。这种行为与对触发器的每个问题评估产生的事件的操作相关
在升级过程中 *(如发送消息)*，基于任何类型的事件：* *-基于触发器事件禁用操作：* *-禁用触发器 * *-基于内部事件禁用主机或项目关于触发器：* *-基于内部事件关于项目/低级发现规则禁用触发器：* *-禁用项目 * *-禁用主机	发送正在进行的消息，然后在升级已发送。后续消息将在消息正文的开头显示取消文本 (注意：升级已取消) 以命名原因 (例如，注意：升级已取消：操作 *“<action name>”* 已禁用)。通过这种方式，收件人将被告知升级已取消，不再执行任何步骤。此消息将发送给之前收到通知的所有人。取消的原因也会记录到服务器日志文件中 (从 [Debug Level] 开始)(/manual/appendix/config/zabbix_server)3= 警告)
在升级过程中 *(如发送消息)*，操作被删除	请注意，如果操作已完成，但恢复操作已配置且尚未执行，则也会发送 *Escalation Cancelled* 消息 不再发送消息。信息被记录到服务器日志文件中 (从 [Debug Level] 开 始)(/manual/appendix/config/zabbix_server)3=Warning)，例如：`escalation Cancelled:action id:334 deleted`

升级示例

例 1

每 30 分钟向 “MySQL 管理员” 组发送一次重复通知（总共 5 次）。要配置：

- 在操作选项卡中，将默认操作步骤持续时间设置为’30m’（30 分钟）
- 将升级步骤设置为从 *1’* 到 *5’*
- 选择 “MySQL 管理员” 组作为邮件的收件人



通知将在问题开始后的 0:00、0:30、1:00、1:30、2:00 小时发送（当然，除非问题更快得到解决）。

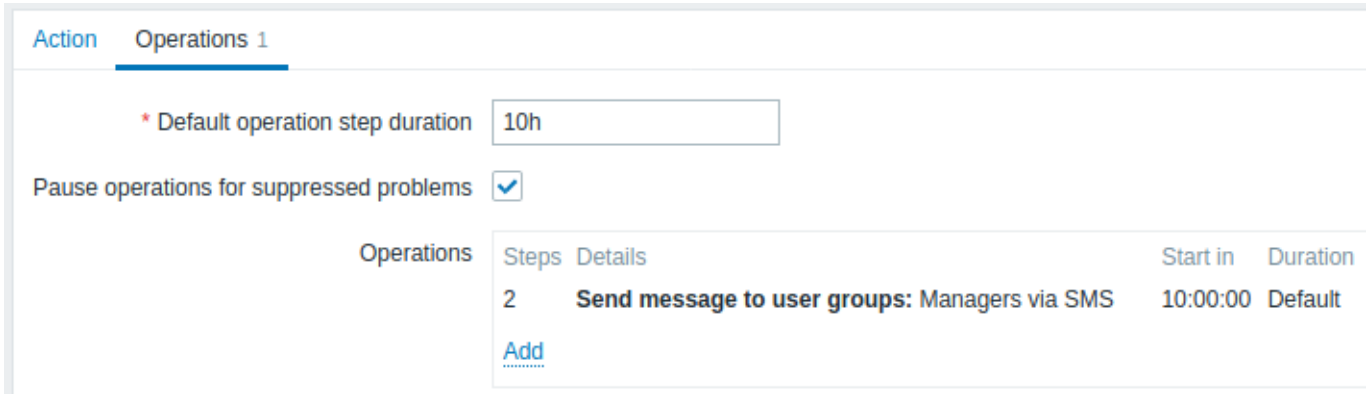
如果问题得到解决并配置了恢复消息，则会将其发送给在此升级方案中至少收到一条问题消息的人。

Note:
如果生成活动升级的触发器被禁用，Zabbix 会向所有已经收到通知的人发送一条关于该事件的信息性消息。

例 2

发送有关长期问题的延迟通知。要配置：

- 在 “操作” 选项卡中，将 “默认操作步骤持续时间” 设置为 “10 小时”（10 小时）
- 将升级步骤设置为从 *2’* 到 *2’*



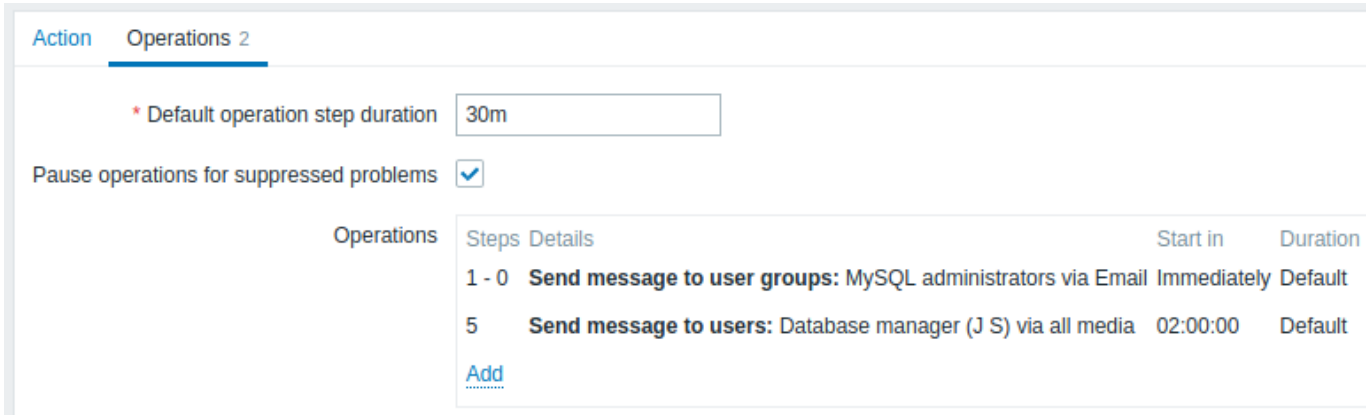
只有在升级场景的第 2 步，或问题开始 10 小时后，才会发送通知。

您可以将消息文本自定义为类似 “问题已超过 10 小时” 的内容。

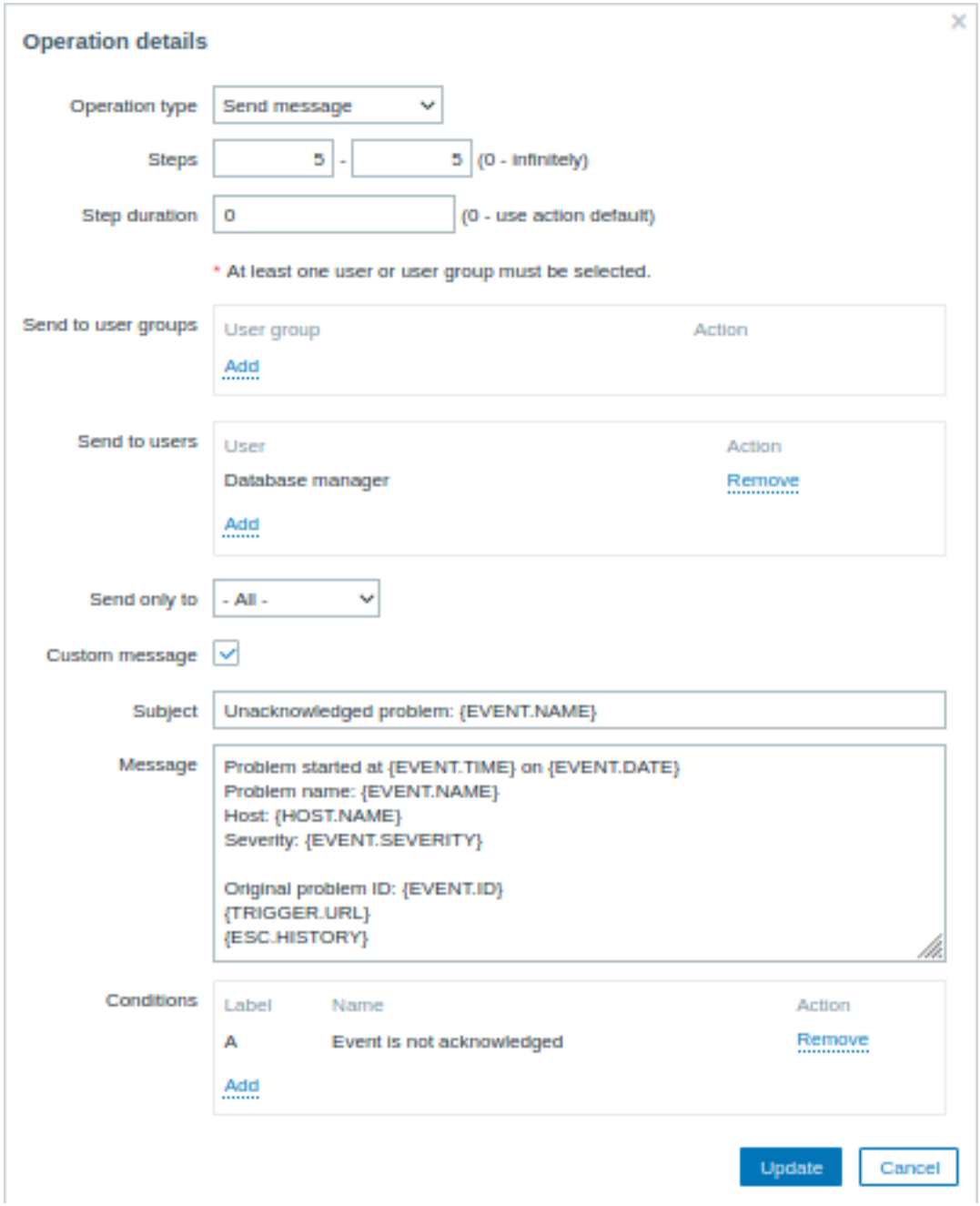
例 3

把问题上报给老板。

在上面的第一个示例中，我们配置了定期向 MySQL 管理员发送消息。在这种情况下，在问题升级到数据库管理器之前，管理员将收到四条消息。请注意，只有在问题尚未得到确认的情况下，经理才会收到一条消息，假设没有人在处理该问题。



操作 2 的详情：



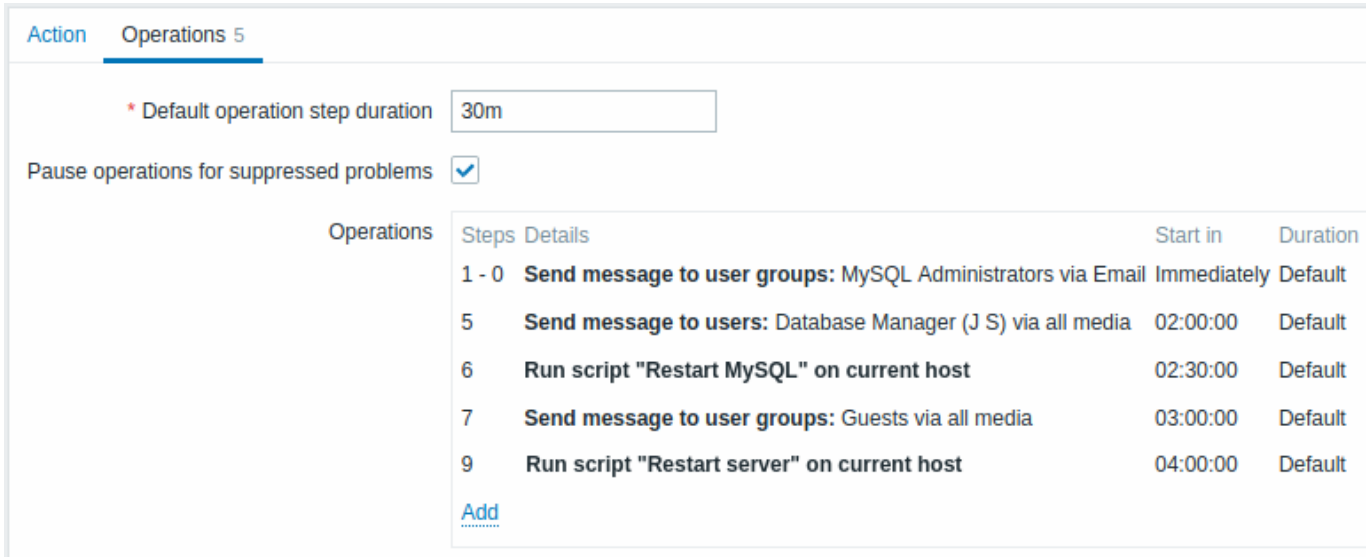
注意在定制消息中使用了 {ESC.HISTORY} 宏。宏将包含有关此升级之前执行的所有步骤的信息，例如发送的通知和执行的命令。

例 4

更复杂的情况。在向 MySQL 管理员发送多条消息并升级到 manager 后，Zabbix 将尝试重新启动 MySQL 数据库。如果问题存在 2:30 小时，但尚未确认，则会发生这种情况。

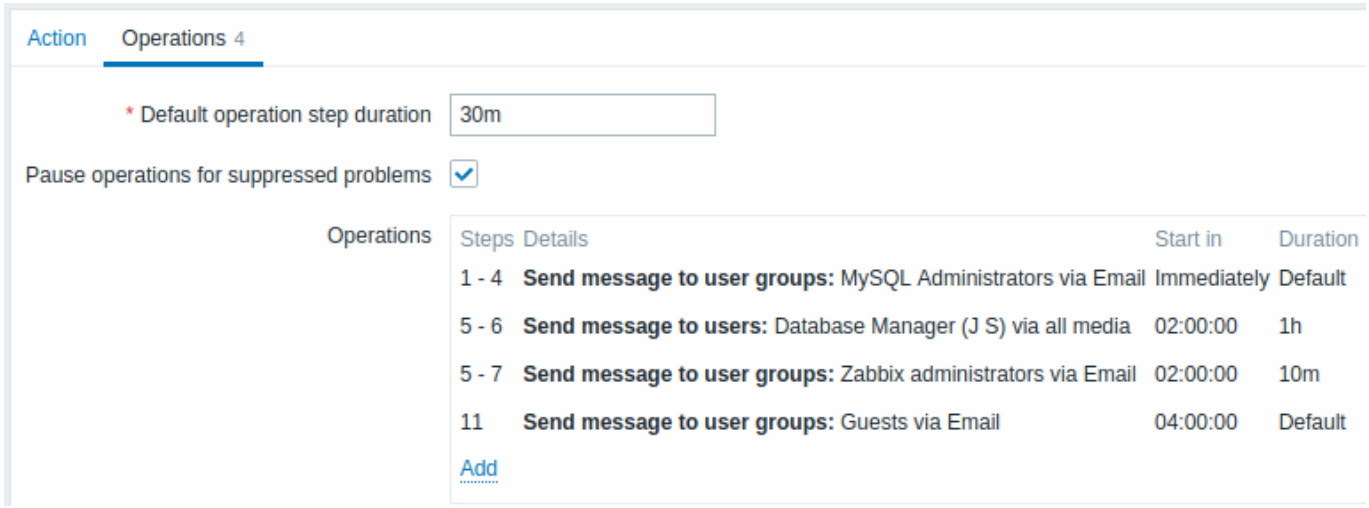
如果问题仍然存在，30 分钟后，Zabbix 将向所有来宾用户发送消息。

如果这没有帮助，再过一个小时，Zabbix 将使用 IPMI 命令使用 MySQL 数据库（第二个远程命令）重新启动服务器。



例 5

将多个操作分配给一个步骤并使用自定义间隔的升级。默认操作步骤持续时间为 30 分钟。



通知将按如下方式发送：

-在问题开始后的 0:00、0:30、1:00、1:30 向 MySQL 管理员发送 -在 2:00 和 2:10（而不是在 3:00；看到步骤 5 和 6 与下一个操作重叠，下一个操作中 10 分钟的较短自定义步骤持续时间将覆盖此处尝试设置的 1 小时的较长步骤持续时间）-在问题开始后的 2:00、2:10、2:20 向 Zabbix 管理员发送（自定义步骤持续时间为 10 分钟）-在问题开始后 4:00 向来宾用户发送（默认步骤持续时间为 30 分钟，在步骤 8 和 11 之间返回）

3 接收不受支持的项目的通知

概述

从 Zabbix 2.2 开始支持接收关于不受支持的监控项的通知。

它是 Zabbix 内部事件概念的一部分，允许在这些场合通知用户。内部事件 反映了状态的变化：

- 当监控项从 “正常” 变为 “不受支持”（并返回）
- 当触发器从 “正常” 变为 “未知”（并返回）
- 当底层自动发现规则从 “正常” 变为 “不受支持”（并返回）

本节介绍如何在监控项变得不受支持时接收通知。

配置

总的来说，以前在 Zabbix 中设置过警报的同学应该熟悉设置通知的过程。

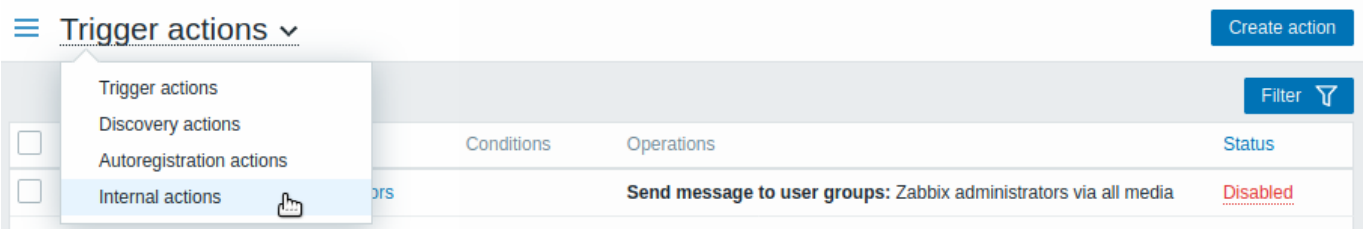
第一步

配置 [某些媒体]（媒体），例如电子邮件、短信或脚本以用于通知。请参阅手册的相应章节以执行此任务。

Attention:
对于内部事件通知，使用默认严重性（“未分类”），因此，如果要接收内部事件通知，请在配置用户媒体时保持选中状态。

第二步

转到配置 → 操作并从页面标题下拉列表中选择内部操作。

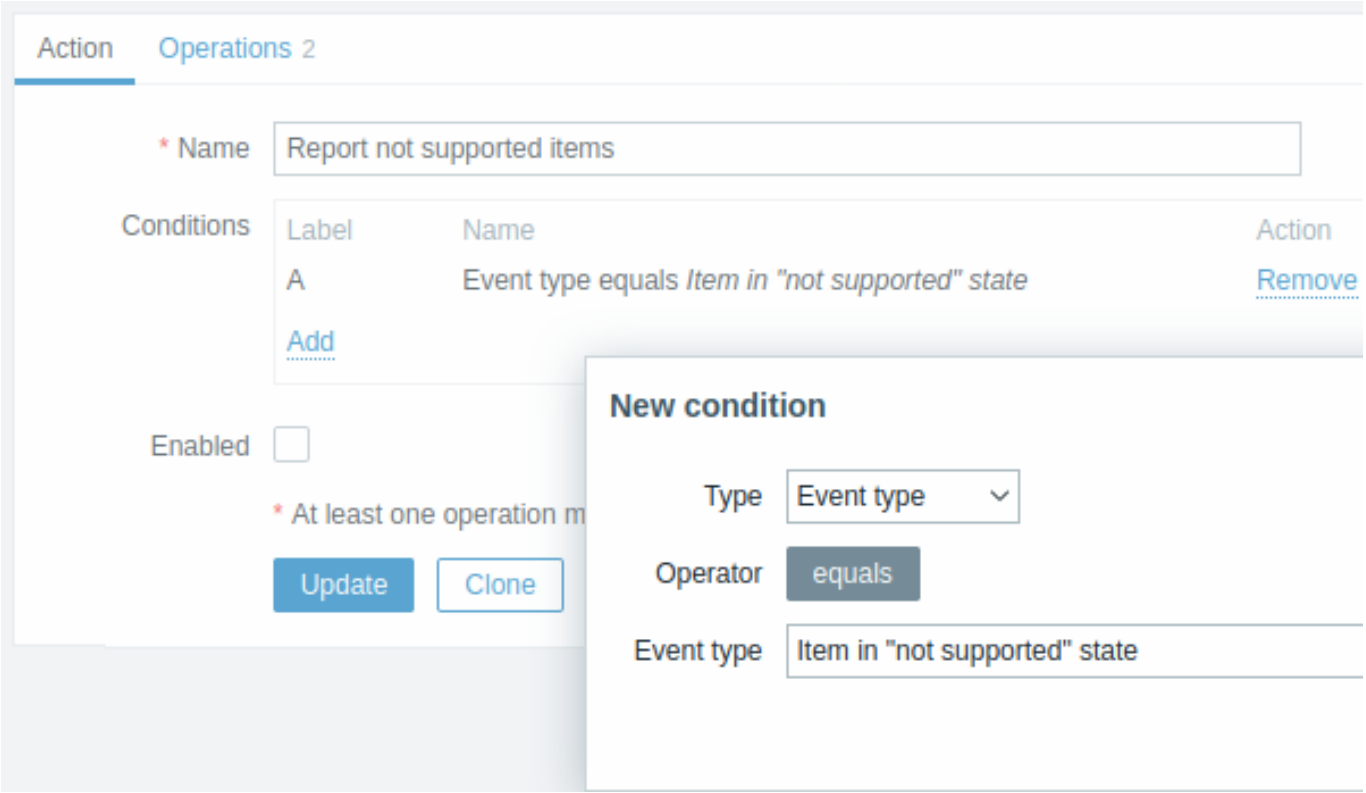


单击右侧的创建操作打开操作配置表单。

第三步

在操作选项卡中输入操作的名称。然后单击条件块中的 *Add*，添加一个新条件。

在新条件弹出窗口中，选择事件类型作为条件类型，然后选择处于 “不支持” 状态的项目作为事件类型值。

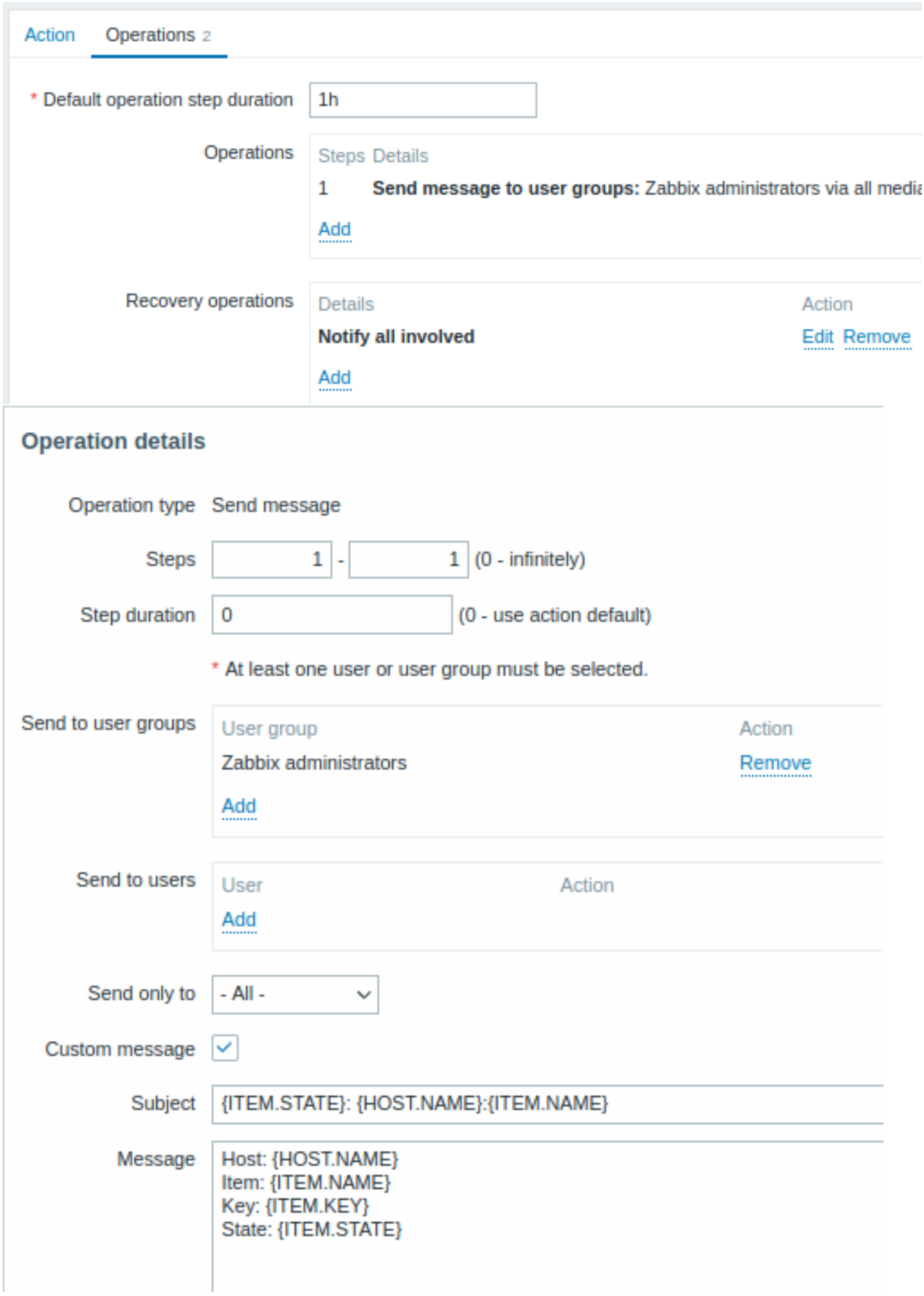


别忘了点击 *Add*，在 *Conditions* 块中实际列出条件。

第四步

在操作选项卡中，单击 *Operations* 块中的 *Add*，然后选择邮件的一些收件人（用户组/用户）和用于传递的媒体类型（或 “全部”）。

如果希望输入问题消息的自定义主题/内容，请选中 “自定义消息” 复选框。



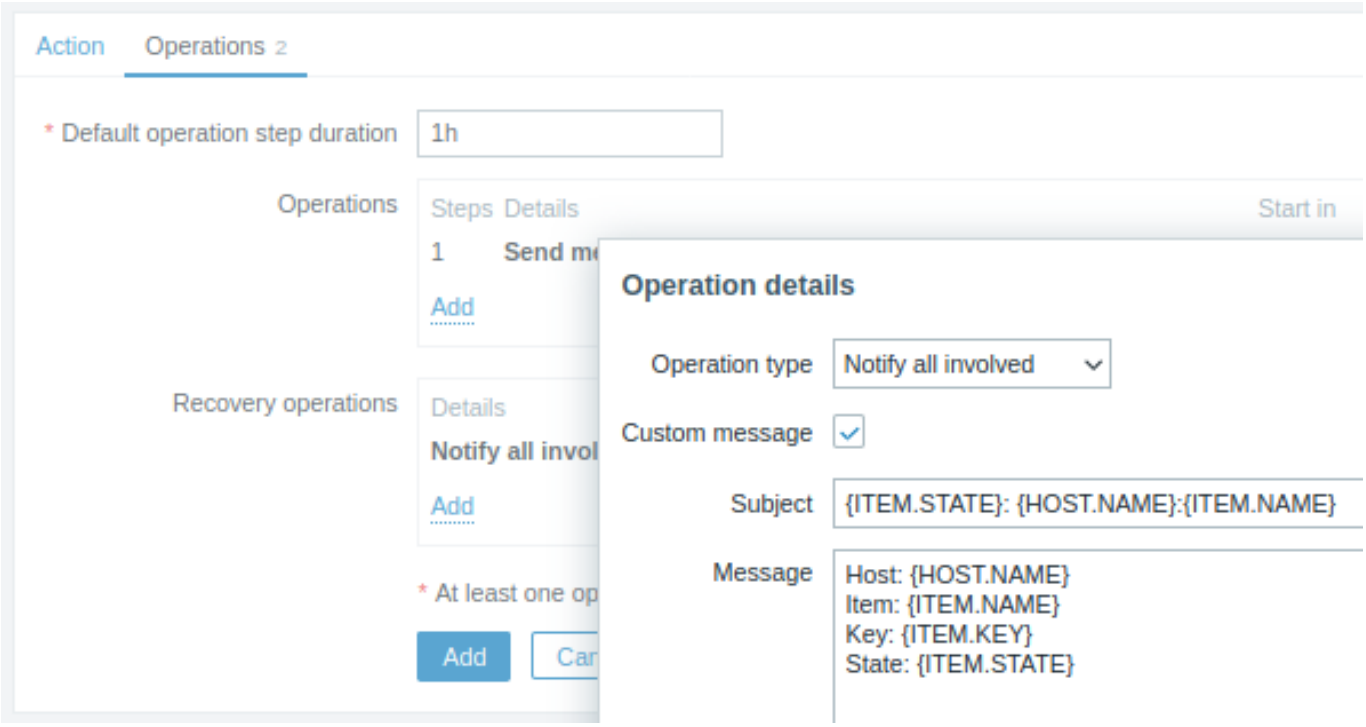
单击添加以实际列出操作块中的操作。

如果希望接收多个通知，请设置操作步骤持续时间（发送消息之间的间隔）并添加另一个步骤。

第五步

Recovery operations 块允许在项目返回正常状态时配置恢复通知。单击恢复操作块中的添加，选择操作类型、邮件收件人（用户组/用户）和用于传递的媒体类型（或“全部”）。

如果希望输入问题消息的自定义主题/内容，请选中“自定义消息”复选框。



在 *Operation details* 弹出窗口中单击 *Add*，以实际列出 *Recovery operations* 块中的操作。

第六步

完成后，单击表单底部的添加按钮。

就这样，你完了！现在，如果某些项目变得不受支持，您可以期待从 Zabbix 收到第一次通知。

11 宏变量

概述

Zabbix 支持许多内置宏，可用于各种情况。这些宏是由特定语法标识的变量：

`{MACRO}`

宏根据上下文解析为特定值。

有效使用宏可以节省时间，并使 Zabbix 配置更加透明。

在一种典型用途中，宏可以用于模板中。因此，模板上的触发器可能被命名为“{HOST.NAME} 上的处理器负载过高”。当模板应用于主机(如 Zabbix 服务器) 时，当触发器显示在监控部分时，名称将解析为“Zabbix 服务器上的处理器负载过高”。

宏可用于监控项键参数。宏只能用于参数的一部分，例如 `item.key[server_{HOST.HOST}_local]`。不需要双引号引用参数，因为如果解析宏中存在任何不明确的特殊符号，Zabbix 将处理这些符号。

除了内置宏之外，Zabbix 还支持用户定义宏、带有上下文的用户定义宏和用于低级发现的宏。

另见：- ·{MACRO} - 内置宏（参见完整列表）- ·{<macro>.<func>(<params>)} - 宏函数 - ·{$MACRO} - 用户定义的宏，可选的上下文 - ·{#MACRO} - 用于底层自动发现的宏（/manual/config/macros/lld_macros）- ·{?EXPRESSION} - 表达式宏

1 宏函数

概述

宏函数提供自定义 [Macro]（/manual/config/macros）值的功能。

有时宏可能解析为一个不一定容易使用的值。它可能很长，或者包含您想要提取的特定感兴趣的子字符串。这就是宏函数可以发挥作用的地方。

宏函数的语法是：

```
{<macro>.<func>(<params>)}
```

其中：

- <macro> - 要自定义的宏（例如 {ITEM.VALUE} 或 {#LLDMACRO}）
- <func> - 要应用的函数
- <params> - 以逗号分隔的函数参数列表。如果参数以 “ ”（空格）、“或包含” 开头，则必须将其引用。

例如：

```
{{TIME}.fmttime(format,time_shift)}
{{ITEM.VALUE}.regsub(pattern, output)}
{{#LLDMACRO}.regsub(pattern, output)}
```

支持的宏函数

函数	说明	参数	支持
fmtnum(<digits>)	数字格式，用于控制小数点后打印的位数	位数-小数点后的位数。不会产生尾随零	{ITEM.VALUE} {ITEM.LASTVALUE} 表达式宏
fmttime(<format>，<time\u shift>)			

函数				
		时间格式	格式-强制格式字符串，与 strftime 函数格式化兼容 **time\u shift**-应用于格式化前时间的时间移位；应以 `-<N><time\u unit>` 或 `+<N><time\u unit>` 开头，其中 *N*——要加或减的时间单位数 时间 _ 单位-小时、天、周、月或年 自 Zabbix 5.4 以来，time\u shift 参数支持多步时间操作，可能包括“/<time\u unit>”以切换到时间单位的开头 (“/d”-午夜，“/w”-一周的第一天 (周一)，“/M”-一个月的第一天，等等)。示例： “-1w”-正好 7 天前 “-1w/w”-前一周的星期一 “-1w/w+1d”-前一周的星期二 注意，时间运算是从左到右	{TIME}

函数			
iregsub(<pattern>，<output>)	通过正则表达式匹配提取子字符串(不区分大小写)	模式匹配的正则表达式 输出输出选项**\1-\9** 支持占位符来捕获组**\0** 返回匹配的文本	{ITEM.VALUE} {ITEM.LASTVALUE} 底层自动发现 宏(底层自动发现规则筛选器中除外)
regsub(<pattern>，<output>)	通过正则表达式匹配提取子字符串(区分大小写)	模式匹配的正则表达式 输出输出选项**\1-\9** 支持占位符来捕获组**\0** 返回匹配的文本	{ITEM.VALUE} {ITEM.LASTVALUE} 底层自动发现 宏(底层自动发现规则筛选器中除外)

如果函数在支持的位置中使用，但应用于不支持宏函数的宏，则宏的计算结果为 “未知”。

如果模式不是正确的正则表达式，则宏的计算结果为 “未知”(不包括底层自动发现宏，在这种情况下，函数将被忽略，宏将保持未展开)

如果在不支持宏函数的位置将宏函数应用于宏，则该函数将被忽略。

例子

以下示例说明了使用宏函数自定义宏值的方法：

收到的值	宏变量	输出
24.3413523	{{ITEM.VALUE}.fmtnum(2)}	24.34
24.3413523	{{ITEM.VALUE}.fmtnum(4)}	
12:36:01	{{TIME}.fmttime(%B)}	October
12:36:01	{{TIME}.fmttime(%d %B,-1M/M)}	1 September
123Log line	{{ITEM.VALUE}.regsub("[0-9]+, Problem)}	Problem
123 Log line	{{ITEM.VALUE}.regsub("^([0-9]+)", "Problem")	Problem
123 Log line	{{ITEM.VALUE}.regsub("^([0-9]+)", Problem ID: \1)}	Problem ID: 123
Log line	{{ITEM.VALUE}.regsub("Problem", "Problem ID: \1")}	Problem ID:
MySQL crashed errno 123	{{ITEM.VALUE}.regsub("(Problem)(.*)([0-9]+)", " Problem ID: \1_\2 ")}	Problem ID: MySQL crashed errno_123
123 Log line	{{ITEM.VALUE}.regsub("^[0-9]+", "Problem ID: \1")}	{ITEM.VALUE} (invalid regular expression)
customername_1	{{#IFALIAS}.regsub("(.*)_([0-9]+)", \1)}	customername
customername_1	{{#IFALIAS}.regsub("(.*)_([0-9]+)", \2)}	1

收到的值	宏变量	输出
customername_1	{{#IFALIAS}.regsub("(.*)_([0-9]+)", \1)}	{{#IFALIAS}.regsub("(.*)_([0-9]+)", \1)} (invalid regular expression)
customername_1	{$MACRO:"{{#IFALIAS}.regsub("(.*)_([0-9]+)", \1)}"}	{$MACRO:"{{#IFALIAS}.regsub("(.*)_([0-9]+)", \1)}"} (invalid regular expression)
customername_1	{$MACRO:"{{#IFALIAS}.regsub("(.*)_([0-9]+)", \2)}"}	{$MACRO:"{{#IFALIAS}.regsub("(.*)_([0-9]+)", \2)}"} (invalid regular expression)
customername_1	"{$MACRO:\"{{#IFALIAS}.regsub("(.*)_([0-9]+)", \1)}\"}"	"{$MACRO:\"{{#IFALIAS}.regsub("(.*)_([0-9]+)", \1)}\"}" (invalid regular expression)
customername_1	"{$MACRO:\"{{#IFALIAS}.regsub("(.*)_([0-9]+)", \2)}\"}"	"{$MACRO:\"{{#IFALIAS}.regsub("(.*)_([0-9]+)", \2)}\"}" (invalid regular expression)
customername_1	"{$MACRO:\"{{#IFALIAS}.regsub("(.*)_([0-9]+)", \1)}\"}"	"{$MACRO:\"{{#IFALIAS}.regsub("(.*)_([0-9]+)", \1)}\"}") (invalid regular expression)

查看完整的监控项值

已解析的文本/日志监控项的 {ITEM.VALUE} 和 {ITEM.LASTVALUE} 宏的长值在某些前端位置被截断为 20 个字符。要查看这些宏的完整值，您可以使用宏函数，例如：

```
{{ITEM.VALUE}.regsub("(.*)", \1)}<br> {{ITEM.LASTVALUE}.regsub("(.*)", \1)}
```

另请参阅：{ITEM.VALUE} 和 {ITEM.LASTVALUE} 宏详细信息。

2 用户宏

概述

除了开箱即用的宏supported之外，Zabbix 还支持用户宏，以提高灵活性。

用户宏可以在全局、模板和主机级别定义。这些宏有一种特殊的语法：

```
{$MACRO}
```

Zabbix 根据以下优先级解析宏：1. 主机级宏（先选中）2. 为主机的一级模板（即直接链接到主机的模板）定义的宏，按模板 ID 排序 3. 为主机二级模板定义的宏，按模板 ID 排序 4. 为主机的三级模板定义的宏，按模板 ID 等排序。5. 全局宏（最后选中）

换句话说，如果主机不存在宏，Zabbix 将尝试在深度不断增加的主机模板中找到它。如果仍然找不到，将使用全局宏（如果存在）。

Warning:
如果同一级别的多个链接模板上存在具有相同名称的宏，则将使用 ID 最低的模板中的宏。因此，在多个模板中使用相同名称的宏是一种配置风险。

如果 Zabbix 找不到宏，宏将无法解析。

Attention:
宏（包括用户宏）在配置部分（例如，在触发器列表中）被设计为不可解析，以使复杂的配置更加透明。

用户宏可用于：- 项目关键参数 - 项目更新间隔和灵活间隔 - 触发器名称和描述 - 触发器表达式参数和常量 (参见示例) - 许多其他位置-请参阅完整列表

全局和宿主宏的常见用例

- 在多个位置使用全局宏；然后更改宏值并一键将配置更改应用于所有位置
- 利用具有主机特定属性的模板：密码、端口号、文件名、正则表达式等。

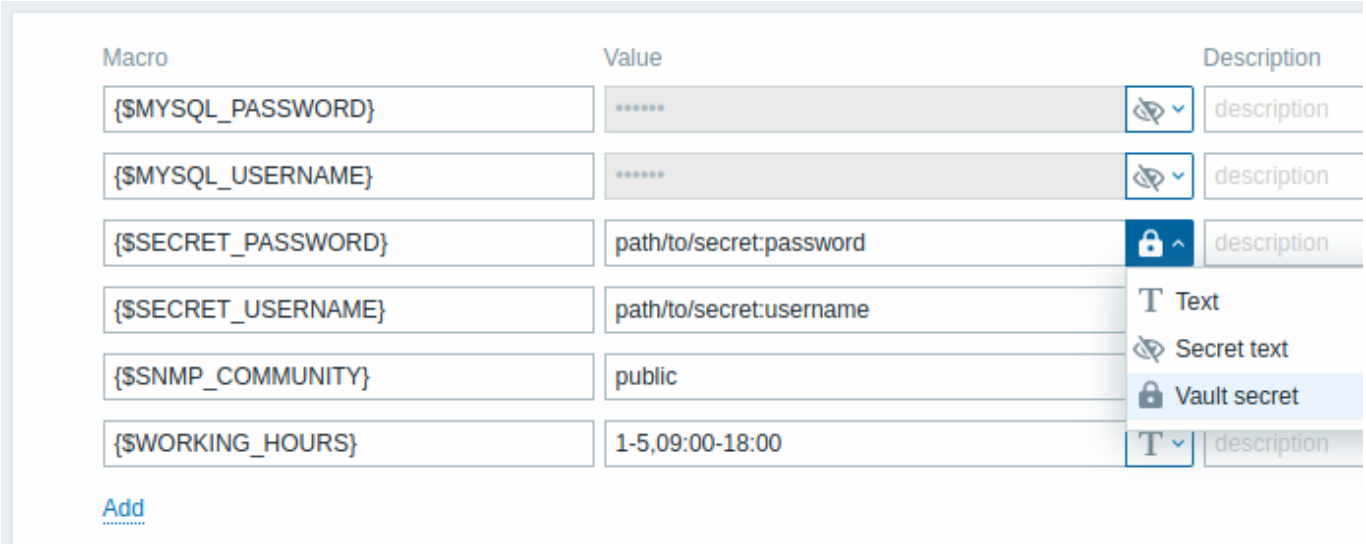
配置

要定义用户宏，请转到前端中的相应位置：

- 有关全局宏，请访问管理 → 一般 → 宏
- 对于主机和模板级宏，请打开主机或模板属性，然后查找 *macros* 选项卡

Note:
如果用户宏用于模板中的监控项或触发器，建议将该宏添加到模板中，即使该宏是在全局级别定义的。这样，如果宏类型为 *text*，则将模板导出为 XML 并将其导入另一个系统仍将允许其按预期工作。秘密宏的值不会导出。

用户宏具有以下属性：



参数	描述
宏	宏名称。名字必须用花括号括起来，并以美元符号开头 示例：{$FRONTEND_URL}。宏名称中允许使用以下字符：**A-Z**(仅大写)，**0-9**，_，.
值	宏值。支持三种值类型： **Text**(默认)-纯文本值 **Secret Text**-该值用星号屏蔽，可用于保护密码或共享密钥等敏感信息 **Vault secret**-该值包含指向Vault secret的参考路径 (如 “path:key”，例如 “secret/zabbix:password”) 注意当 secret 宏的值隐藏在视线之外时，可以通过在监控项中使用来显示该值。例如，在外部脚本中，可以使用引用秘密宏的 “echo” 语句向前端显示宏值，因为 Zabbix 服务器可以访问真正的宏值 要选择值类型，请单击值输入字段末尾的按钮：  图标表示文本宏  图标表示密文宏。悬停时，值字段将转换为  按钮，用于输入宏的新值 (要退出而不保存新值，请单击向后箭头 ()  图标表示秘密 vault 宏 用户宏值的最大长度为 2048 个字符 (在 5.2.0 之前的版本中为 255 个字符)
Description	用于提供有关此宏的更多信息的文本字段

Note:
包含秘密宏的 URLs 将无法工作，因为其中的宏将解析为 “*******”。

Attention:
在触发器表达式中，用户宏将解析是否引用参数或常量。如果引用主机、监控项键、函数、运算符或其他触发器表达式，则它们将不会解析。不能在触发器表达式中使用秘密宏。

示例

示例 1

在 “SSH daemon 状态” 项键中使用主机级宏：`net.tcp.service[ssh,,{$SSH_PORT}]`

此项可以分配给多个主机，前提是 **{$SSH_PORT}** 在这些主机上定义。

示例 2

在 “CPU 负载太高” 触发器中使用主机级宏：`last(/ca_001/system.cpu.load[,avg1])>{$MAX_CPULOAD}`

这样的触发器将在模板上创建，而不是在个别主机。

> **Note:**
> 如果要使用值的数量作为函数参数（例如，**max(/host/key,#3)**），在宏定义中包含如下哈希标记：SOME_PERIOD => #3

示例 3

在 “CPU 负载太高” 触发器中使用两个宏：`min(/ca_001/system.cpu.load[,avg1],{$CPULOAD_PERIOD})>{$MAX_CPULOAD}`

注意宏可以作为触发函数的参数，在这个示例函数 **min()**。

示例 4

将代理不可用条件与项目更新同步间隔：

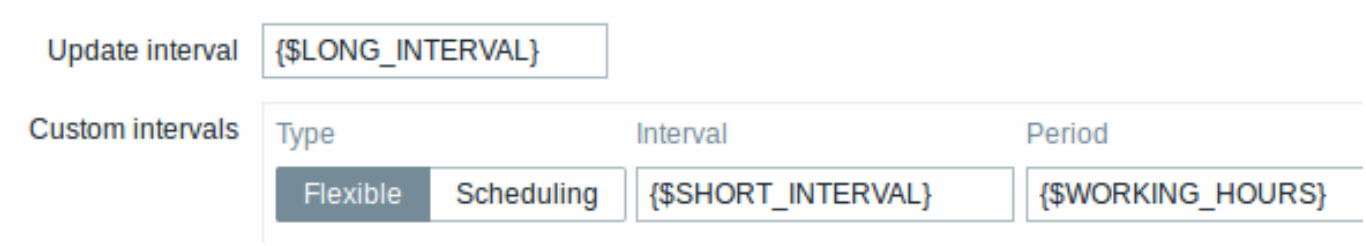
- 定义 {$INTERVAL} 宏并在项目更新间隔中使用它；
- 使用 {$INTERVAL} 作为代理不可用触发器的参数：

`nodata(/ca_001/agent.ping,{$INTERVAL})=1`

示例 5

集中配置工作时间：

- 创建一个全局 {$WORKING_HOURS} 宏，等于 1-5,09:00-18:00；
- 在 *Administration* → *General* → *Working time* 字段中使用它 图形界面；
- 在 *Administration* → *User* → *When active* 字段中使用它 媒体;
- 使用它在工作时间设置更频繁的项目轮询：

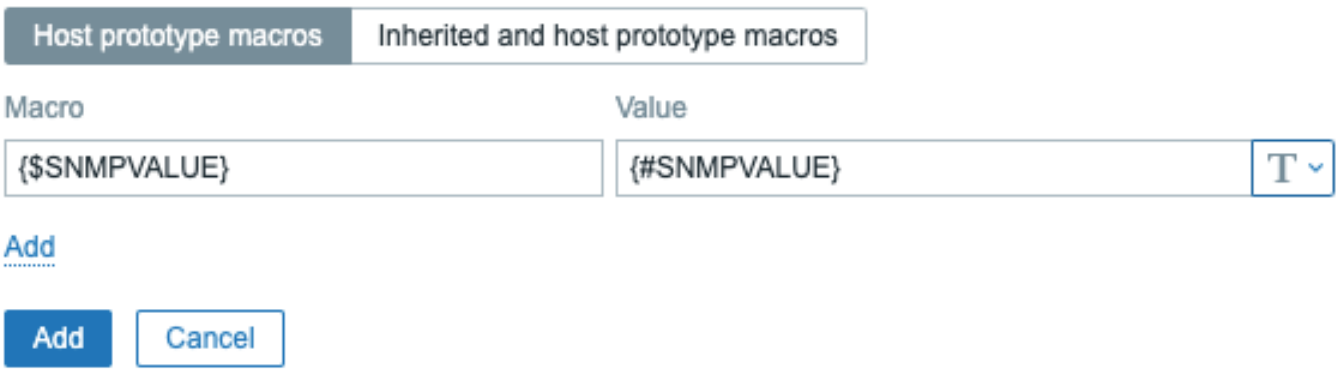


- 在时间段动作条件下使用；
- 在 *Administration* → *General* → *Macros* 中调整工作时间，如果需要的话。

示例 6

使用主机原型宏为发现的主机配置项目：

- 在主机原型上定义用户宏 {$SNMPVALUE} 和 {#SNMPVALUE} 低级发现 宏作为值：



- 将 *Generic SNMPv2* 模板分配给主机原型；
- 在 *Generic SNMPv2* 的 *SNMP OID* 字段中使用 {$SNMPVALUE} 模板项。

用户宏上下文

请参阅用户宏上下文。

3 带有上下文的用户宏

概述

可选上下文可用于 [user 宏]（/manual/config/macros/user_macros），允许覆盖具有特定于上下文的默认值。

上下文附加到宏名称；语法取决于是否上下文是一个静态文本值：

`{$MACRO:"静 态 文 本"}`

或正则表达式：

`{$MACRO:regex:"正 则 表 达 式"}`

请注意，具有正则表达式上下文的宏只能在用户宏配置。如果 `regex:` 前缀在其他地方被用作用户宏上下文，就像在触发器表达式中一样，它将被视为静态上下文。

上下文引用是可选的（另见 [重要注释]（#important_notes））。

宏上下文示例：

示例	说明
`{$LOW_SPACE_LIMIT}`	没有上下文的用户宏。
`{$LOW_SPACE_LIMIT:/tmp}`	带有上下文的用户宏（静态字符串）。
`{$LOW_SPACE_LIMIT:regex:"^/tmp$"}`	带有上下文的用户宏（正则表达式）。与 `{$LOW_SPACE_LIMIT:/tmp}` 相同。
`{$LOW_SPACE_LIMIT:regex:"^/var/log/.*$"}`	带有上下文的用户宏（正则表达式）。匹配所有以 /var/log/ 为前缀的字符串。

用例

可以定义具有上下文的用户宏以实现更灵活触发器表达式中的阈值（基于检索到的值低级发现）。例如，您可以定义以下宏：

- {$LOW_SPACE_LIMIT} = 10
- {$LOW_SPACE_LIMIT:/home} = 20
- {$LOW_SPACE_LIMIT: 正则表达式:”^\/[a-z]+$”} = 30

然后一个低级发现宏可以用作宏上下文已挂载文件系统发现的触发器原型：

·last(/host/vfs.fs.size[{#FSNAME},pfree])<{$LOW_SPACE_LIMIT:”{#FSNAME}”}

发现后将应用不同的低空间阈值根据发现的挂载点或文件系统类型触发。如果出现以下情况，将生成问题事件：

- /home 文件夹的可用磁盘空间不足 20%
- 匹配正则表达式模式的文件夹（如 /etc、/tmp 或 /var）具有不到 30% 的可用磁盘空间
- 与正则表达式模式不匹配且不是 /home 的文件夹有不到 10% 的可用磁盘空间

重要笔记

- 如果存在多个具有上下文的用户宏，Zabbix 将尝试首先匹配简单的上下文宏，然后匹配上下文宏未定义顺序的正则表达式。

> **Warning:**
> 不要创建不同的上下文宏匹配相同的字符串以避免未定义的行为。

- 如果在主机上找不到带有上下文的宏，则链接模板或全局，然后搜索没有上下文的宏。
- 上下文中仅支持低级发现宏。任何其他宏被忽略并被视为纯文本。

从技术上讲，宏上下文是使用类似于item key 参数，除了宏上下文是如果有 **,** 字符，则不被解析为多个参数：

- 如果上下文包含 }，则必须用 " 引用宏上下文字符或以 " 字符开头。引号内的引号 context 必须用 \ 字符转义。
- \ 字符本身没有被转义，这意味着它是不可能的有一个以 \ 字符结尾的引用上下文 - 宏 {$MACRO:”a:\b\c\”} 无效。
- 上下文中的前导空格被忽略，尾随空格是不是：
  - 例如 {$MACRO:A} 与 {$MACRO:A} 相同，但不同 {$ 宏：A}。
- 前导引号和尾随引号后的所有空格都是忽略，但引号内的所有空格都不是：
  - 宏 {$MACRO:”A”}、{$MACRO:”A”}、{$MACRO:”A”} 和 {$MACRO:”A” } 相同，但宏 {$MACRO:”A”} 和 {$MACRO:”A”} 不是。

以下宏都是等价的，因为它们具有相同的上下文：{$MACRO:A}、{$MACRO:A} 和 {$MACRO:"A"}。这是相反的带有项目键，其中'key[a]'、'key[a]' 和'key["a"]' 是语义上相同，但出于唯一性目的而不同。

4 低级发现宏

概述

[low-level] (/manual/discovery/low_level_discovery) 中使用了一种宏发现](LLD) 功能：

·{#MACRO}

它是一个在 LLD 规则中使用并返回实际值的宏文件系统名称、网络接口、SNMP OID 等。

这些宏可用于创建项目、触发器和图形原型。然后，当发现真正的文件系统时，网络接口等，这些宏被替换为实数值并且是创建真实项目、触发器和图表的基础。

这些宏也用于创建主机和主机组 *prototypes* 在虚拟机发现 中。

一些低级发现宏与 LLD "预打包"Zabbix 中的函数 - {#FSNAME}, {#FSTYPE}, {#IFNAME}, {#SNMPINDEX},{#SNMPVALUE}。然而，遵守这些名称不是强制性的，当创建一个自定义低级发现规则。然后您可以使用任何其他 LLD 宏名称和参考那个名字。

支持的位置

可以使用 LLD 宏：

- 在低级发现规则过滤器中
- 对于项目原型
  - 姓名
  - 关键参数
  - 单元
  - 更新间隔[1]
  - 历史存储期[1]
  - 趋势存储期[1]
  - 项目值预处理步骤
  - SNMP OID
  - IPMI 传感器领域
  - 计算项目公式
  - SSH 脚本和 Telnet 脚本
  - 数据库监控 SQL 查询
  - JMX 项目端点字段
  - 描述
  - HTTP 代理 URL 字段
  - HTTP 代理 HTTP 查询字段字段
  - HTTP 代理请求正文字段
  - HTTP 代理所需的状态代码字段
  - HTTP 代理标头字段键和值
  - HTTP 代理 HTTP 认证用户名字段
  - HTTP 代理 HTTP 认证密码字段
  - HTTP 代理 HTTP 代理字段
  - HTTP 代理 HTTP SSL 证书文件字段
  - HTTP 代理 HTTP SSL 密钥文件字段
  - HTTP 代理 HTTP SSL 密钥密码字段
  - HTTP 代理 HTTP 超时[1] 字段
  - 标签
- 用于触发器原型
  - 姓名
  - 运营数据
  - 表达式（仅在常量和函数参数中）
  - 网址
  - 描述
  - 标签
- 对于图形原型
  - 姓名
- 对于主机原型
  - 姓名
  - 可见名称

- 自定义接口字段：IP、DNS、端口、SNMP v1/v2 社区、SNMP v3 上下文名称、SNMP v3 安全名称、SNMP v3 身份验证密码，SNMP v3 隐私密码
- 主机组原型名称
- 主机标签值
- 主机宏值
- (参见完整列表)

在所有这些地方，LLD 宏都可以在静态用户macro 上下文。

使用宏函数

低级发现宏支持宏功能（在低级发现规则过滤器），允许提取某个部分使用正则表达式的宏值。

例如，您可能想要提取客户名称和接口用于事件标记的以下 LLD 宏中的编号：

·{#IFALIAS}=customername_1

为此，可以将 regsub 宏函数与触发器原型的事件标签值字段：



请注意，未引用的项目中不允许使用逗号key 参数，所以必须引用包含宏函数的参数。反斜杠 (\) 字符应用于转义范围。例子：

```
net.if.in["{{#IFALIAS}.regsub(\"(.*)_([0-9]+)\", \1)}",bytes]
```

有关宏函数语法的更多信息，请参阅：宏函数

自 Zabbix 以来，低级发现宏支持宏功能 4.0。

脚注

1 在标有 [1] 的字段中，单个宏必须填充整个字段。一个或多个字段中的多个宏不支持与文本混合。

5 表达式宏

概述

表达式宏对于公式计算很有用。它们是通过展开内部的所有宏并评估结果表达式来计算的。

表达式宏有一个特殊的语法：

·{?EXPRESSION}

{HOST.HOST<1-9>} 和 {ITEM.KEY<1-9>} 宏在表达式宏中受支持。从 Zabbix 6.0.9 开始，表达式宏中支持 {ITEM.KEY<1-9>} 宏。

用法

在以下位置：

·- 图表名称·- 地图元素标签·- 地图形状标签·- 地图链接标签

只有来自以下集合的单个函数：avg、last、max、min 允许作为表达式宏，例如：

·{?avg(/{HOST.HOST}/{ITEM.KEY},1h)} ·诸如 {?last(/host/item1)/last(/host/item2)}、{?count(/host/item1,5m)} 和 {?last(/host/item1)* 10} 在这些位置不正确。

然而，在：

·- 触发事件名称·- 基于触发器的通知和命令·- 问题更新通知和命令·复杂表达式是允许的，例如：

·{?trendavg(/host/item1,1M:now/M)/trendavg(/host/item1,1M:now/M-1y)*100}

也可参见：

- ·Supported macros 获取表达式宏的支持位置列表
- ·Example 在事件名称中使用表达式宏

12 用户和用户组

概述

Zabbix 中的所有用户都通过基于 web 的方式访问 Zabbix 应用程序前端。每个用户都被分配了一个唯一的登录名和密码。

所有用户密码都经过加密并存储在 Zabbix 数据库中。用户不能使用自己的用户名和密码直接登录。除非它们也已针对 UNIX 设置相应的 UNIX 服务器。Web 服务器和用户浏览器之间的通信可以使用 SSL 保护。

具有灵活的用户权限架构 你可以限制和区分以下权利：

- 访问管理 Zabbix 前端功能
- 在前端执行某些操作
- 访问主机组中的受监控主机
- 使用特定的 API 方法

1 配置用户

概述

初始 Zabbix 安装有两个预定义用户：

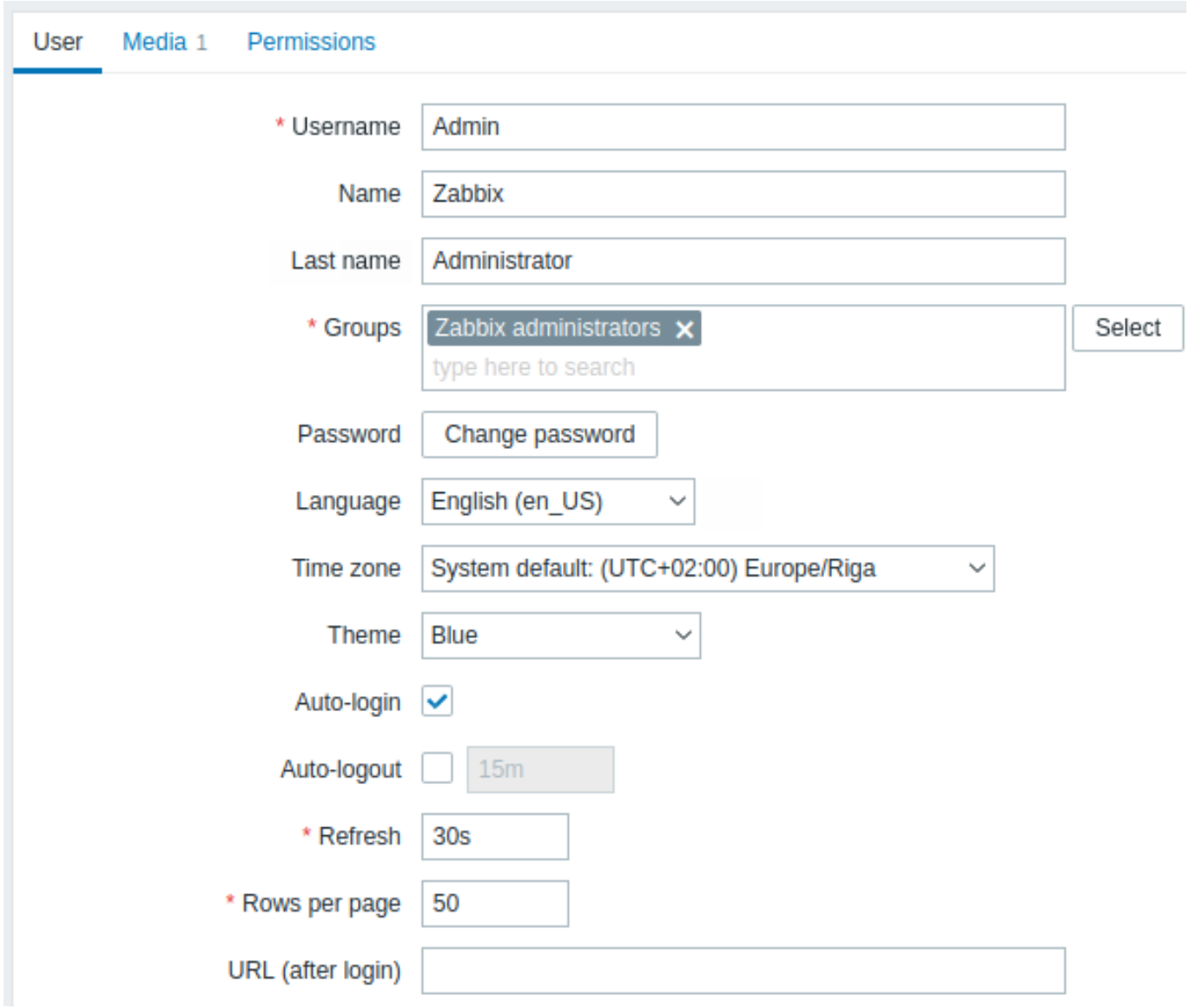
- 管理员 - Zabbix超级用户 具有完全权限；
- *guest* - 一个特殊的 Zabbix 用户。默认情况下禁用 “来宾” 用户。如果将其添加到客人用户组，您可以访问 Zabbix 中的监控页面而无需被请注意，默认情况下，‘guest’ 没有权限 Zabbix 对象。

要配置新用户：

- 转到管理 → 用户
- 点击创建用户（或用户名编辑现有的用户）
- 编辑表单中的用户属性

通用属性

User 选项卡包含一般用户属性：



所有必填字段都标有红色星号。

参数	说明
用户名	唯一的用户名，用作登录名。
姓名	用户名（可选）。 如果不为空，在确认信息和通知收件人信息中可见。
姓	用户姓（可选）。 如果不为空，在确认信息和通知收件人信息中可见。
Groups	选择用户所属的用户组。从 Zabbix 3.4.3 开始，此字段是自动完成的，因此开始输入用户组的名称将提供匹配组的下拉列表。向下滚动以选择。或者，单击 *Select* 以添加组。单击 “x” 以删除所选内容。 遵守用户组决定了用户将拥有 [访问]（权限）的主机组和主机。
Password	用于输入用户密码的两个字段。 使用现有密码，包含一个 *Password* 按钮，单击该按钮可打开密码字段。 请注意，超过 72 个字符的密码将被截断。
语言	Zabbix 前端的语言。 翻译工作需要 php gettext 扩展。
时区	选择时区以在用户级别覆盖全局时区，或选择系统默认以使用全局时区设置。
主题	定义前端的外观： 系统默认 - 使用默认系统设置 蓝色 - 标准蓝色主题 深色 - 替代深色主题 高对比度浅色 - 高对比度浅色主题 高对比度深色 - 高对比度深色主题

参数	说明
自动登录	勾选此复选框，让 Zabbix 记住用户并自动登录用户 30 天。浏览器 cookie 用于此目的。
自动注销	选中此复选框后，用户将在设置的秒数后自动注销（最少 90 秒，最多 1 天）。 时间后缀支持，例如 90s, 5m, 2h, 1d。 请注意，此选项将不起作用： * 如果启用了 “如果 Zabbix 服务器关闭则显示警告” 全局配置选项并且 Zabbix 前端保持打开状态； * 当监控菜单页面执行背景信息刷新时； * 如果登录时勾选了记住我 *30* 天选项。
Refresh	设置用于图形、纯文本数据等的刷新率。可设置为 0 禁用。
每页的行数	您可以确定每页将显示在列表中的行数。
URL（登录后）	您可以让 Zabbix 在用户登录成功后将用户转移到特定的 URL，例如问题页面。

用户媒体

Media 选项卡包含为用户定义的所有媒体的列表。媒体用于发送通知。点击 *Add* 分配媒体给用户。

请参阅媒体类型有关配置用户媒体的详细信息部分。

权限

Permissions 选项卡包含以下信息：

- 用户角色。用户不能更改自己的角色。
- 角色中定义的用户类型（用户、管理员、超级管理员）配置。
- 用户有权访问的主机组。“用户” 和 “管理员” 类型的用户默认情况下无权访问任何主机组和主机。要得到他们需要包含在具有访问权限的用户组中的访问权限到各自的主机组和主机。
- 访问 Zabbix 前端、模块的部分和元素的权限，和 API 方法。显示允许访问的元素在绿色。浅灰色表示对元素的访问是否认。
- 执行某些操作的权利。允许的操作是以绿色显示。浅灰色表示用户无权执行此操作。

有关详细信息，请参阅 [用户权限]（权限）页面。

2 权限

概述

您可以通过定义 Zabbix 中的用户权限来区分相应的用户角色。然后需要将非特权用户包括有权访问主机组数据的用户组。

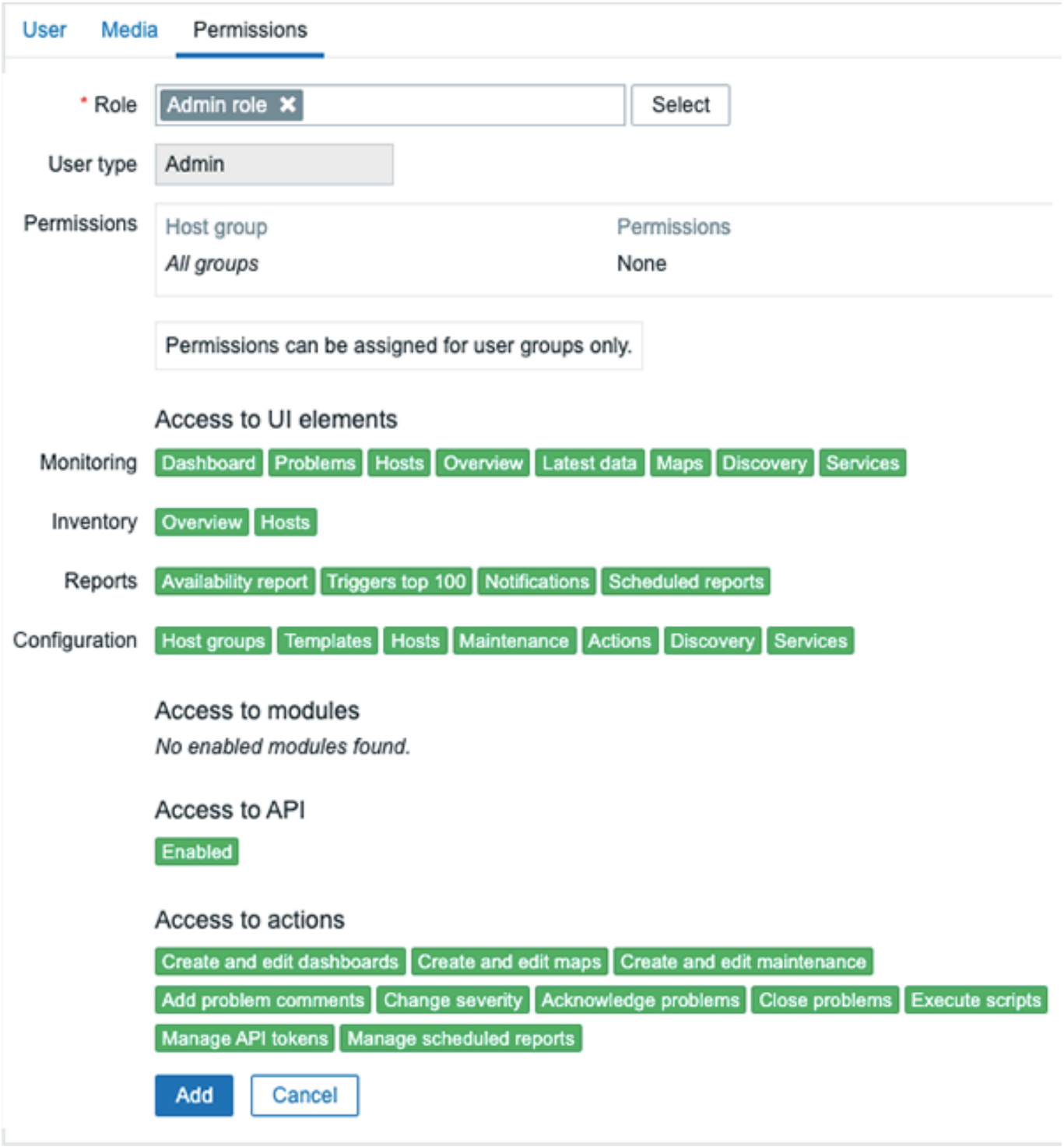
用户角色

用户角色定义了 UI 的哪些部分、哪些 API 方法以及哪些操作可供用户使用。以下角色是预定义的在 Zabbix：

用户类型	说明
访客角色	用户有权访问 “监控”、“清单” 和 “报告” 菜单部分，但无权执行任何操作。
用户角色	用户有权访问 “监控”、“清单” 和 “报告” 菜单部分。默认情况下，用户无权访问任何资源。必须明确分配对主机组的任何权限。
管理员角色	用户有权访问 “监控”、“库存”、“报告” 和 “配置” 菜单部分。默认情况下，用户无权访问任何主机组。必须明确授予对主机组的任
超级管理员角色	用户有权访问所有菜单部分。用户具有对所有主机组的读写访问权限。不能通过拒绝对特定主机组的访问来撤销权限。

用户角色在 *Administration→User roles* 部分进行配置。超级管理员可以修改或删除预定义的角色并使用自定义创建更多角色权限集。

要为用户分配角色，请转到用户中的权限选项卡配置表单，找到 *Role* 字段并选择一个角色。一旦角色被选中，将显示相关权限列表以下。



主机组权限

只有在主机组级别上，usergroups才能访问 Zabbix 中的任何主机数据。

这意味着个人用户不能直接被授予访问主机（或主机组）的权限。只有作为用户组的一部分，才能授予它对主机的访问权限，该用户组被授予对包含该主机的主机组的访问权限。

3 用户组

概述

用户组允许出于组织目的和为数据分配权限的目的对用户进行分组。主机组监控数据的权限分配给用户组，而不是单个用户。

将一组用户的可用信息与另一组用户的可用信息分开通常是有意义的。这可以通过将用户分组，然后为主机组分配不同的权限来实现。

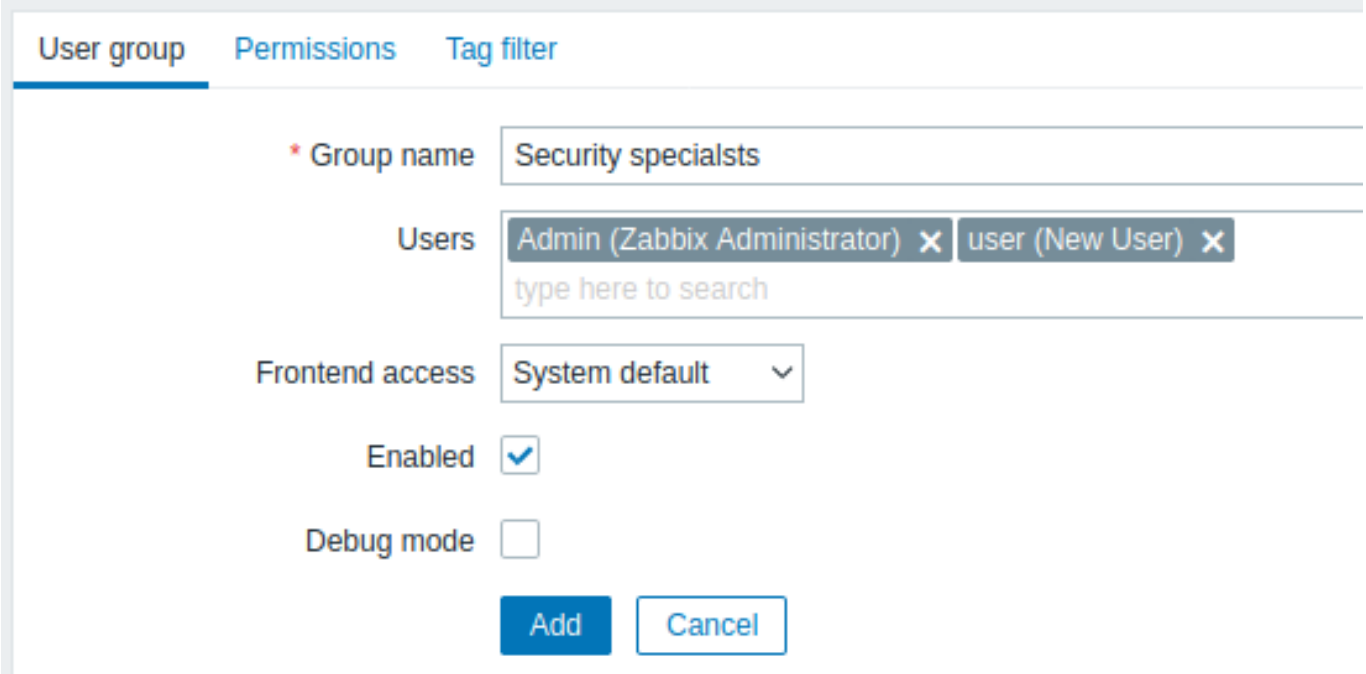
用户可以属于任意数量的组。

配置

配置用户组：

- 转到管理 → 用户组
- 点击创建用户组（或组名编辑现有组）
- 在表单中编辑组属性

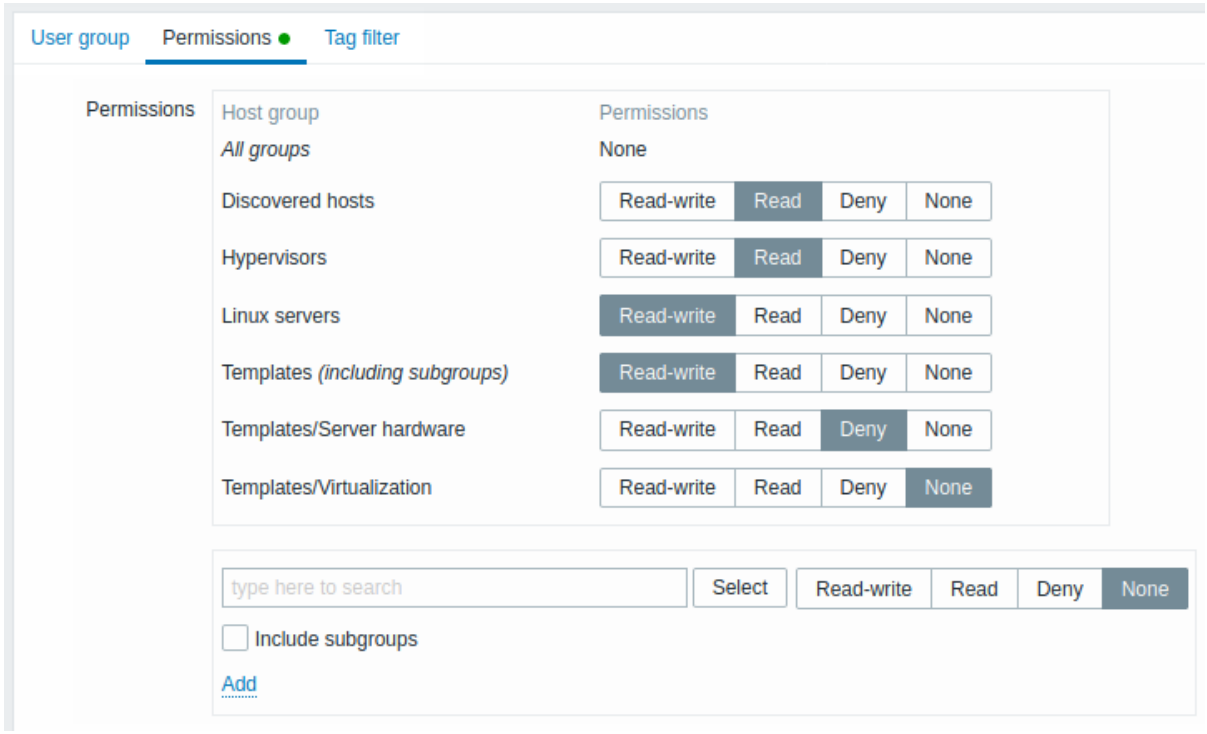
用户组选项卡包含常规组属性：



所有必填字段都标有红色星号。

参数	说明
组名	唯一组名。
Users	要将用户添加到组，请开始输入现有用户的名称。当出现匹配用户名的下拉菜单时，向下滚动以选择。 或者，您也可以单击 *Select* 按钮在弹出窗口中选择用户。
前端访问	如何验证组的用户。 系统默认 - 使用默认验证方法（设置全局） **Internal** - 使用 Zabbix 内部身份验证（即使全局使用 LDAP 身份验证）。如果 HTTP 身份验证是全局默认值，则忽略。 **LDAP** - 使用 LDAP 身份验证（即使内部身份验证是全局使用）。如果 HTTP 身份验证是全局默认值，则忽略。 禁用 - 该组禁止访问 Zabbix 前端
启用	用户组和组成员的状态。 选中 - 用户组和用户已启用 未选中 - 用户组和用户已禁用
调试模式	选中此复选框可为用户激活调试模式。

Permissions 选项卡允许您指定用户组对主机组（以及主机）数据的访问权限：



对主机组的当前权限显示在 *Permissions* 块中。

如果主机组的当前权限由所有嵌套主机组继承，则由主机组名称后括号中的 包括子组文本指示。

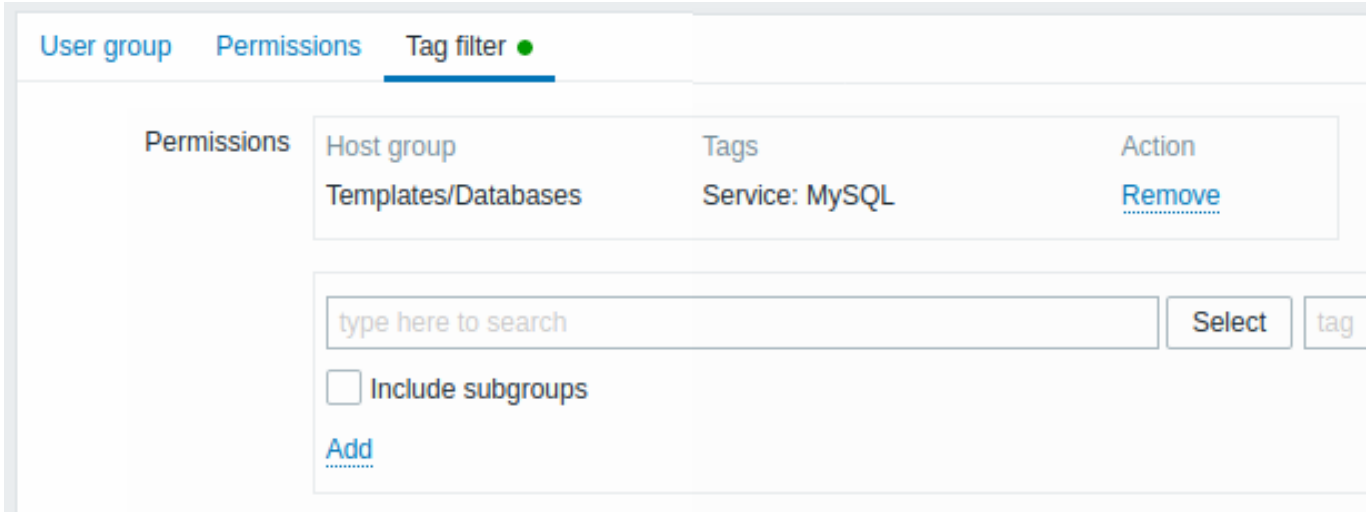
您可以更改对主机组的访问级别：

- 读写 - 对主机组的读写访问；
- **Read** - 对主机组的只读访问；
- 拒绝 - 拒绝访问主机组；
- 无 - 未设置权限。

使用下面的选择字段来选择主机组及其访问级别（请注意，如果主机组已在列表中，则选择 *None* 将从列表中删除主机组）。如果您希望包含嵌套主机组，请选中 *Include subgroups* 复选框。此字段是自动完成的，因此开始输入主机组的名称将提供匹配组的下拉列表。如果您希望查看所有主机组，请单击 *Select*。

请注意，主机组configuration 中的超级管理员用户可以对嵌套主机组强制执行与父主机组相同级别的权限。

标签过滤器选项卡允许您为用户组设置基于标签的权限，以查看按标签名称及其值过滤的问题：



要选择要为其应用标记过滤器的主机组，请单击 *Select* 以获取现有主机组的完整列表，或开始键入主机组的名称以获取匹配组的下拉列表。如果要将标签过滤器应用于嵌套主机组，请选中 *Include subgroups* 复选框。

标签过滤器允许将对主机组的访问与查看问题的可能性分开。

例如，如果数据库管理员只需要查看 “MySQL” 数据库问题，则需要先为数据库管理员创建一个用户组，然后指定 “Service” 标签名称和 “MySQL” 值。



如果指定了“服务”标签名称并且值字段留空，则相应的用户组将看到所选主机组标签名称为“服务”的所有问题。如果标签名称和值字段都留空但选择了主机组，则相应的用户组将看到所选主机组的所有问题。确保正确指定了标签名称和标签值，否则相应的用户组将看不到任何问题。

让我们回顾一个示例，当用户是多个选定用户组的成员时。在这种情况下，过滤将对标签使用 OR 条件。

用户组 **A**			用户组 **B**			两个组的用户（成员）的可见结果
标签过滤器						
主机组	标签名称	标签值	主机组	标签名称	标签值	
模板/数据库	服务	MySQL	模板/数据库	服务	Oracle	服务：MySQL 或 Oracle 问题可见
模板/数据库	空白	空白	模板/数据库	服务	Oracle	所有问题可见
未选择	空白	空白	模板/数据库	服务	Oracle	Service:Oracle 问题可见

Attention:
添加过滤器（例如，某个主机组“模板/数据库”中的所有标签）会导致无法看到其他主机组的问题。

多个用户组的主机访问

一个用户可以属于任意数量的用户组。这些组可能对主机具有不同的访问权限。

因此，了解非特权用户将能够访问哪些主机非常重要。例如，让我们考虑在用户组 A 和 B 中的用户在各种情况下对主机 **X**（在主机组 1 中）的访问将如何受到影响。

- 如果 A 组只有对主机组 1 的 *Read* 访问权限，但 B 组对主机组 1 的 *Read-write* 访问权限，则用户将获得对“X”的 **Read-write** 访问权限。

Attention:
从 Zabbix 2.2 开始，“读写”权限优先于“读取”权限。

- 在与上述相同的情况下，如果“X”同时也在被拒绝组 A 或 B 的主机组 2 中，则对“X”的访问将不可用，尽管 读写访问主机组 1。
- 如果 A 组没有定义权限，而 B 组对主机组 1 具有 读写访问权限，则用户将获得对“X”的读写访问权限。
- 如果 A 组对主机组 1 具有 *Deny* 访问权限，而 B 组对主机组 1 具有 *Read-write* 访问权限，则用户将获得对'X' **denied** 的访问权限。

其他详情

- 对主机具有 *Read-write* 访问权限的管理员级别用户将无法链接/取消链接模板，如果他无权访问 *Templates* 组。拥有对 *Templates* 组的 *Read* 访问权限，他将能够将模板链接/取消链接到主机，但是，在模板列表中将看不到任何模板，并且将无法在其他地方使用模板。
- 对主机具有 *Read* 访问权限的管理员级别用户将不会在配置部分主机列表中看到该主机；但是，可以在 IT 服务配置中访问主机触发器。
- 只要地图为空或只有图像，任何非超级管理员用户（包括“访客”）都可以查看网络地图。将主机、主机组或触发器添加到映射时，会尊重权限。
- 如果对相关主机的访问被明确“拒绝”，Zabbix 服务器将不会向定义为操作操作接收者的用户发送通知。

13 存储密钥

概述

可以在 HashiCorp 中秘密存储一些敏感信息 Vault KV Secrets Engine - 第 2 版。可以将密钥保存为：

- 用户宏值
- 数据库访问凭据

Zabbix 提供对 Vault 中机密的只读访问权限，假设密钥由其他人管理。

用户宏值

可以将用户宏值秘密存储在 Vault 中。

一个“Vault 机密”用户宏的值包含一个参考路径（如'path:key'，对于例如“秘密/zabbix：密码”）。

以下命令可用于设置路径的值示例中提到：

```
# 如果没有启用“secret/”挂载点，请注意必须使用“kv-v2”
$ vault secrets enable -path=secret/ kv-v2

# 在挂载点 "secret/" 和路径 "secret/zabbix" 下放置带有密钥密码的新密钥
$ vault kv put secret/zabbix 密码=<密码>

# 测试是否添加成功
$ vault kv 获取秘密/zabbix

# 最后用Curl进行测试，注意“data”需要手动添加在挂载点之后，“/v1”需要在挂载点之前添加，另见--capath参数
$ curl --header "X-Vault-Token: <VaultToken>" https://127.0.0.1:8200/v1/secret/data/zabbix
```

每次刷新配置数据时，Zabbix 服务器都会检索秘密值并存储在配置缓存中。必须在服务器配置中提供对引用路径进行只读访问的身份验证令牌（“VaultToken” 参数）。如果无法成功检索宏值，则使用该值的相应项目将变为不受支持。

也可以使用’secrets_reload’ 命令行option 从 Vault 触发秘密值的刷新。

Zabbix 代理从不与 Vault 通信以获取数据库凭据以外的任何机密。Zabbix 代理上的秘密值在每次配置同步时从 Zabbix 服务器检索，并以与 Zabbix 服务器相同的方式存储在配置缓存中。

这意味着 Zabbix 代理在重新启动后无法开始数据收集，直到它第一次从 Zabbix 服务器接收到配置数据更新。Zabbix server 和 proxy 之间必须开启加密；否则会记录服务器警告消息。

数据库凭据

支持将 Zabbix 服务器、代理和前端使用的数据库凭据秘密存储在 Vault 中：

- 可以选择在前端安装向导 中输入用于检索数据库凭据的 Vault 相关参数。

从 Vault 检索的数据库凭据将由前端缓存。请注意，文件系统临时文件目录用于前端的数据库凭据缓存。您可以使用 ZBX_DATA_CACHE_TTL 常量 来控制数据缓存刷新/失效的频率。

- 对于服务器/代理，VaultDBPath 配置参数可用于指定通过密钥 “密码” 和 “用户名” 检索数据库凭据的路径（例如：secret/zabbix/database）。

以下命令可用于设置示例中提到的路径的值：

```
# 如果没有启用“secret/”挂载点，请注意必须使用“kv-v2”
$ vault secrets enable -path=secret/ kv-v2

# 在挂载点 "secret/" 和路径 "secret/zabbix/database" 下放置带有密钥用户名和密码的新密钥
$ vault kv put secret/zabbix/database 用户名=zabbix 密码=<密码>

# 测试是否添加成功
$ vault kv 获取秘密/zabbix/数据库

# 最后用Curl进行测试，注意“data”需要手动添加在挂载点之后，“/v1”需要在挂载点之前添加，另见--capath参数
$ curl --header "X-Vault-Token: <VaultToken>" https://127.0.0.1:8200/v1/secret/data/zabbix/database
```

配置参数

对于 Zabbix 服务器/代理，为 Vault 身份验证和检索数据库凭据添加了新的配置参数：

- VaultToken - Vault 身份验证令牌（请参阅 Zabbix server/proxy配置文件详细）
- VaultURL - Vault 服务器 HTTP[S] URL
- VaultDBPath - 将通过密钥 “密码” 和 “用户名” 检索数据库凭据的保险库路径（例如：secret/zabbix/database）

Zabbix server 和 Zabbix proxy 在启动时会从 zabbix_server.conf 和 zabbix_proxy.conf 中读取 Vault 相关的配置参数。

Zabbix server 和 Zabbix proxy 会在启动时额外读取一次”VAULT_TOKEN” 环境变量并取消设置，使其无法通过 fork 脚本使用；如果 VaultToken 和 VAULT_TOKEN 都包含值，则这是一个错误。

> **Note:**
> 正斜杠和冒号是保留符号。正斜杠只能用于将挂载点与路径分开（例如，secret/zabbix，其中挂载点为 “secret”，“zabbix” 为路径），对于 Vault 宏，冒号只能用于将路径与密钥分开。如果需要创建名称由正斜杠分隔的挂载点（例如 foo/bar/zabbix，挂载点为 “foo/bar” 且路径为 “ zabbix” 为”foo%2Fbar/zabbix”) 并且挂载点名称或路径是否需要包含冒号。

配置 TLS

应将由证书颁发机构 (CA) 签名的证书添加到默认 CA 存储中。或者，可以使用 SSLCALocation 配置参数指定自定义 CA 存储位置；请注意，在这种情况下，必须使用 openssl c_rehash 实用程序准备证书目录，例如配置 SSLCALocation 并在该目录中复制 “ca.pem”，然后运行以下命令：

```
$ c_rehash .
```

14 定时报表

概述

本节提供有关配置计划报告的信息。

> **Attention:**
> 目前，对计划报告的支持是实验性的。

先决条件：

- 必须正确安装和配置 Zabbix Web 服务才能启用计划报告生成 - 有关说明，请参阅设置计划报告。
- 用户必须具有管理员或超级管理员类型的用户角色，并具有以下权限：
  - 访问用户界面元素块中的计划报告（查看报告）；
  - 在访问操作块（创建/编辑报告）中管理计划报告。

> **Note:**
> 对于多页仪表板，只有第一页包含在 PDF 报告中。

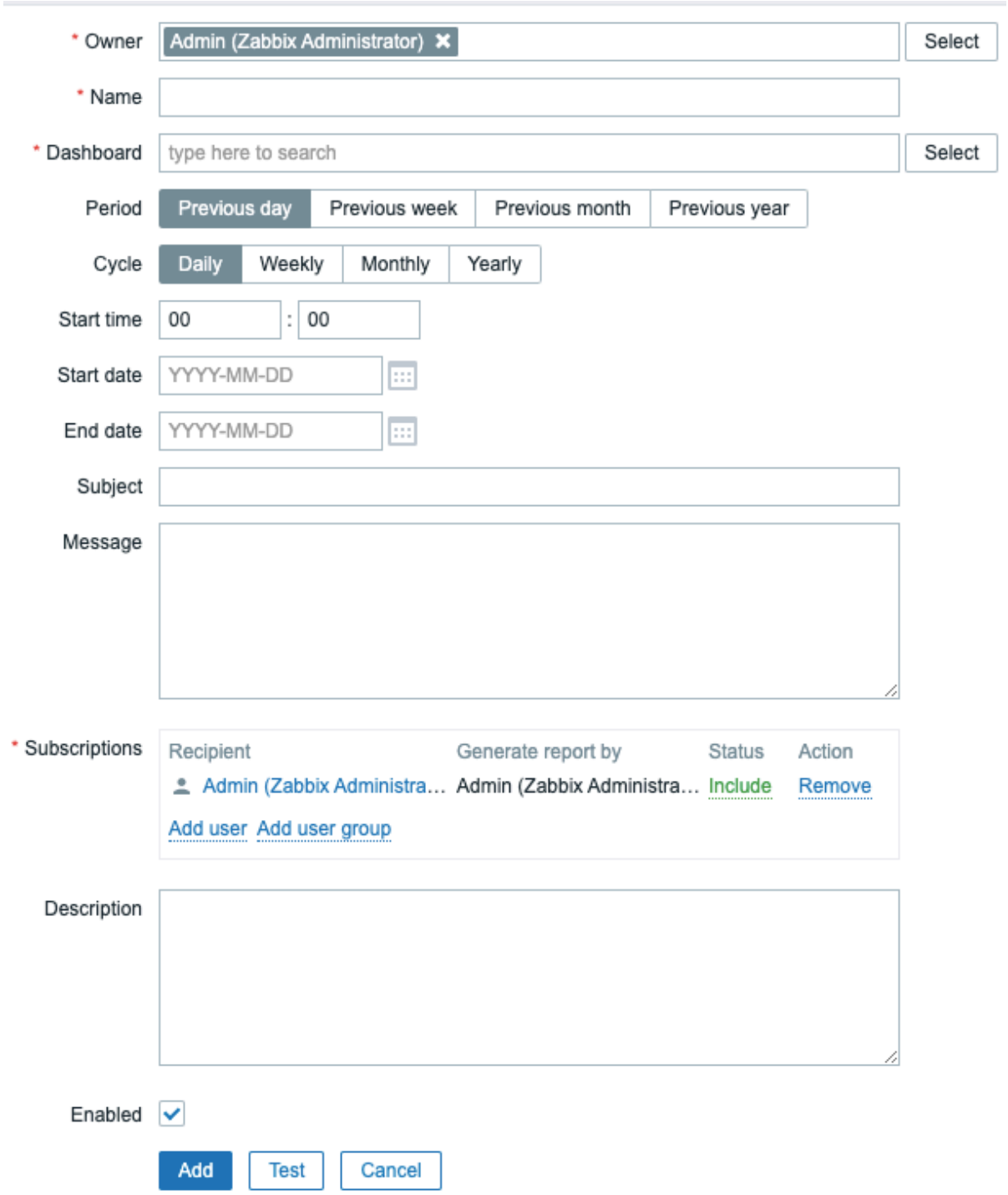
要在 Zabbix 前端创建计划报告，请执行以下操作：

- 前往：报告 → 计划报告
- 点击屏幕右上角的创建报告
- 在表格中输入报告的参数

您还可以通过打开现有报告创建报告，按克隆按钮，然后以不同的名称保存。

配置

计划报告选项卡包含常规报告属性。



所有必填输入字段都标有红色星号。

参数	说明
所有者	创建报告的用户。允许超级管理员级别的用户更改所有者。对于管理员级别的用户，此字段是只读的。
名称	报告的名称；必须是唯一的。
仪表板	报告所基于的仪表板；一次只能选择一个仪表板。要选择仪表板，请开始输入名称 - 将出现匹配仪表板的列表；向下滚动以选择。或者，您可以单击该字段旁边的选择，然后从弹出窗口的列表中选择一个仪表板。 如果仪表板包含多个页面，则只有第一页将作为报告发送。
期间	准备报告的时间段。选择可用选项之一：前一天、前一周、上个月、上一年。
周期	报告生成频率。可以每天、每周、每月或每年发送报告。每周模式允许选择发送报告的星期几。
开始时间	准备报告的时间，格式为 hh:mm。
重复	发送报告的星期几。此字段仅在 周期设置为每周时可用。
开始日期	应开始定期生成报告的日期
结束日期	应停止定期报告生成的日期。
主题	报告电子邮件的主题。支持 {TIME} 宏。
消息	报告电子邮件的正文。支持 {TIME} 宏。

参数	说明
订阅	报告收件人列表。默认情况下，仅包括报告所有者。任何配置了电子邮件媒介的 Zabbix 用户都可以指定为报告收件人。 按添加用户或添加用户组添加更多收件人。 按用户名编辑设置： 生成报告者 - 报告应代表报告所有者还是收件人生成。 状态 - 选择 包括将报告发送给用户或选择 排除以防止将报告发送给该用户。至少一名用户必须具有 包括状态。排除状态可用于从包括的用户组中排除特定用户。 请注意，权限不足的用户 ***** 将看到 Inaccessible user 或 Inaccessible user group *Recipient* 和 *Generate report by* 字段中的真实姓名；状态和 行动字段将显示为只读。
启用	报告状态。清除此复选框将禁用报告。
描述	报告的可选描述。此说明仅供内部使用，不会发送给报告收件人。

* 权限不足的用户是具有基于管理员用户类型的角色的用户，并且不是收件人或报告所有者所属的用户组的成员。

表单按钮

表单底部的按钮允许执行多项操作。

Add	添加报告。此按钮仅适用于新报告。
Update	更新报告的属性。
Clone	根据当前报告的属性创建另一个报告。
Test	通过向当前用户发送报告来测试报告配置是否正确。
Delete	删除报告。
Cancel	取消编辑报表属性。

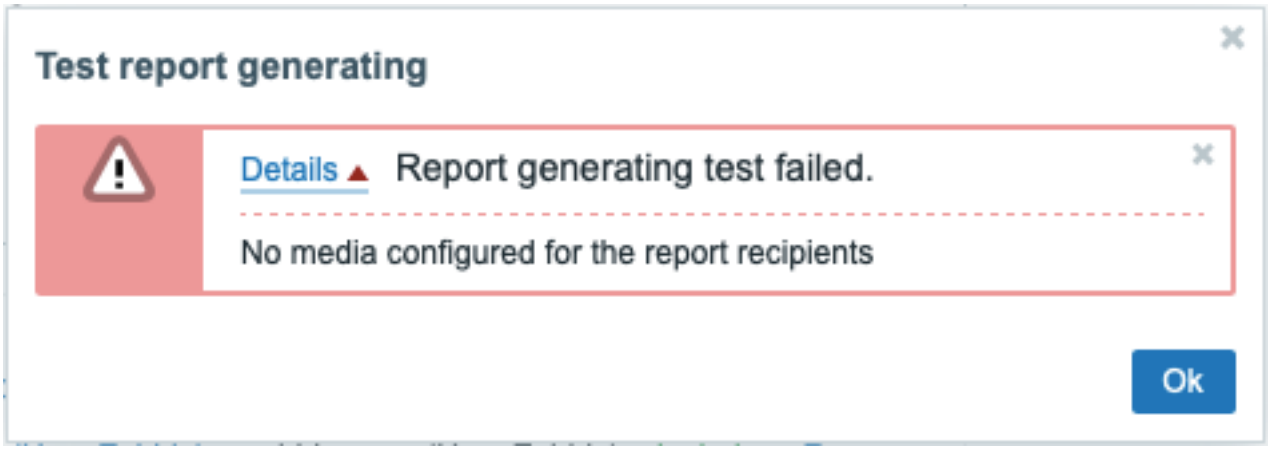
测试

要测试报告，请单击报告配置表单底部的测试按钮。

如果已从仪表板操作菜单 打开报告配置表单，则测试按钮不可用。

如果配置正确，则立即将测试报告发送给当前用户。对于测试报告，订阅者和 “生成者” 用户设置将被忽略。

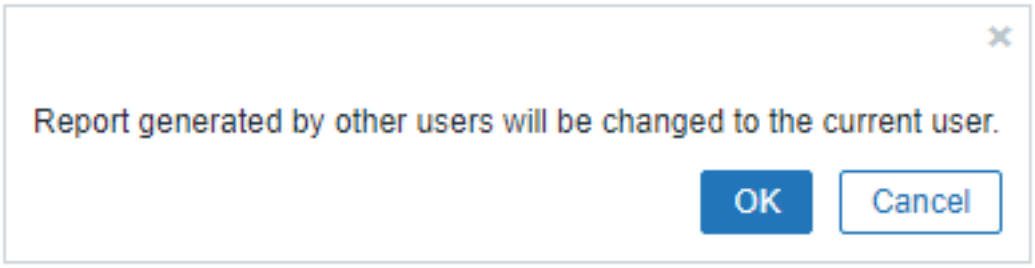
如果配置不正确，则会显示一条错误消息，描述可能的原因。



更新报告

要更新现有报告，请按报告名称，然后进行所需的配置更改并按更新按钮。

如果另一个用户更新了现有报告并且该用户更改了仪表板，则在按下更新按钮时，将显示警告消息 “其他用户生成的报告将更改为当前用户”。



在此步骤按 OK 将导致以下更改：

- 生成者设置将更新以显示上次编辑报告的用户（除非生成者设置为收件人）。
- 已显示为 *Inaccessible user* 或 *Inaccessible user group* 的用户将从报告订阅者列表中删除。

按取消将关闭弹出窗口并取消报告更新。

克隆报告

要快速克隆现有报告，请按现有报告配置表单底部的克隆按钮。克隆由其他用户创建的报表时，当前用户将成为新报表的所有者。

报告设置将根据用户权限复制到新的报告配置表单中：

- 如果克隆报告的用户没有仪表板的权限，则仪表板字段将被清除。
- 如果克隆报告的用户对订阅列表中的某些用户或用户组没有权限，则无法克隆无法访问的收件人。
- 生成者设置将更新以显示当前用户（除非生成者设置为收件人）。

更改所需设置和报告名称，然后按添加。

8. 服务监控

概览　服务监视功能适用于希望获得被监视基础设施的高级 (业务) 视图的人。一般情况下，我们对低级别细节不感兴趣，例如磁盘空间不足、处理器负载高等。我们感兴趣的是整个 IT 部门提供的服务的整体可用性，以及对识别 IT 基础设施的薄弱环节、各种 IT 服务的 SLA、现有 IT 基础设施的结构以及其他更高级别的信息。

Zabbix 服务监控为所有提到的问题提供了答案。

服务监控允许创建监控数据的层次结构表示。

一个非常简单的服务结构可能如下所示：

```
Service
|
|-Workstations
| |
| |-Workstation1
| |
| |-Workstation2
|
|-Servers
```

结构的每个节点都有属性状态。根据所选算法计算状态并将其传输到上层。各个节点的状态受映射问题状态的影响。问题映射是通过标签完成的。

如果检测到服务状态发生变化，Zabbix 可以在 Zabbix server 上发送通知或自动执行脚本。可以根据子服务的状态来定义父服务是否应该进入' 问题状态' 的灵活规则。然后可以使用服务问题数据来计算 SLA 并根据灵活的条件集发送 SLA 报告。

服务监控在 Services 菜单中配置，该菜单由以下部分组成：

- Services（服务）

服务部分允许通过添加父服务来构建受监控基础架构的层次结构，然后将子服务添加到父服务。

除了配置服务树之外，本节还提供了整个基础架构的概述，并允许快速识别导致服务状态更改的问题。

- Service actions（服务动作）

在本节中，您可以配置服务动作。服务动作是可选的，并允许：- 发送服务宕机的通知；- 在服务状态发生变化时在 Zabbix server 上执行远程命令；- 当服务再次启动时发送恢复通知。

- SLA

在本节中，您可以定义服务水平协议并为特定服务设置服务水平目标。

- SLA report（SLA 报告）

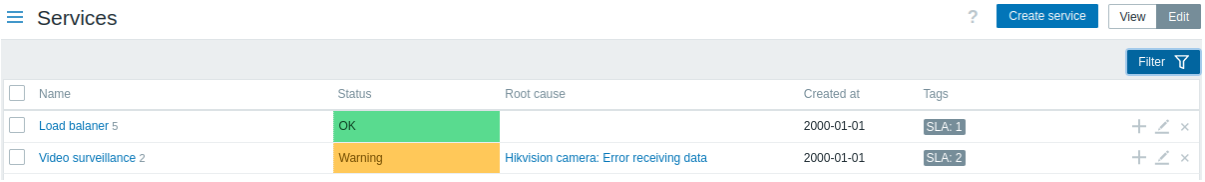
在此部分中，您可以查看 SLA 报告。

可以参照：

- SLA 监控配置示例
- Zabbix 6.0 以下版本升级服务 注意事项

1 服务树

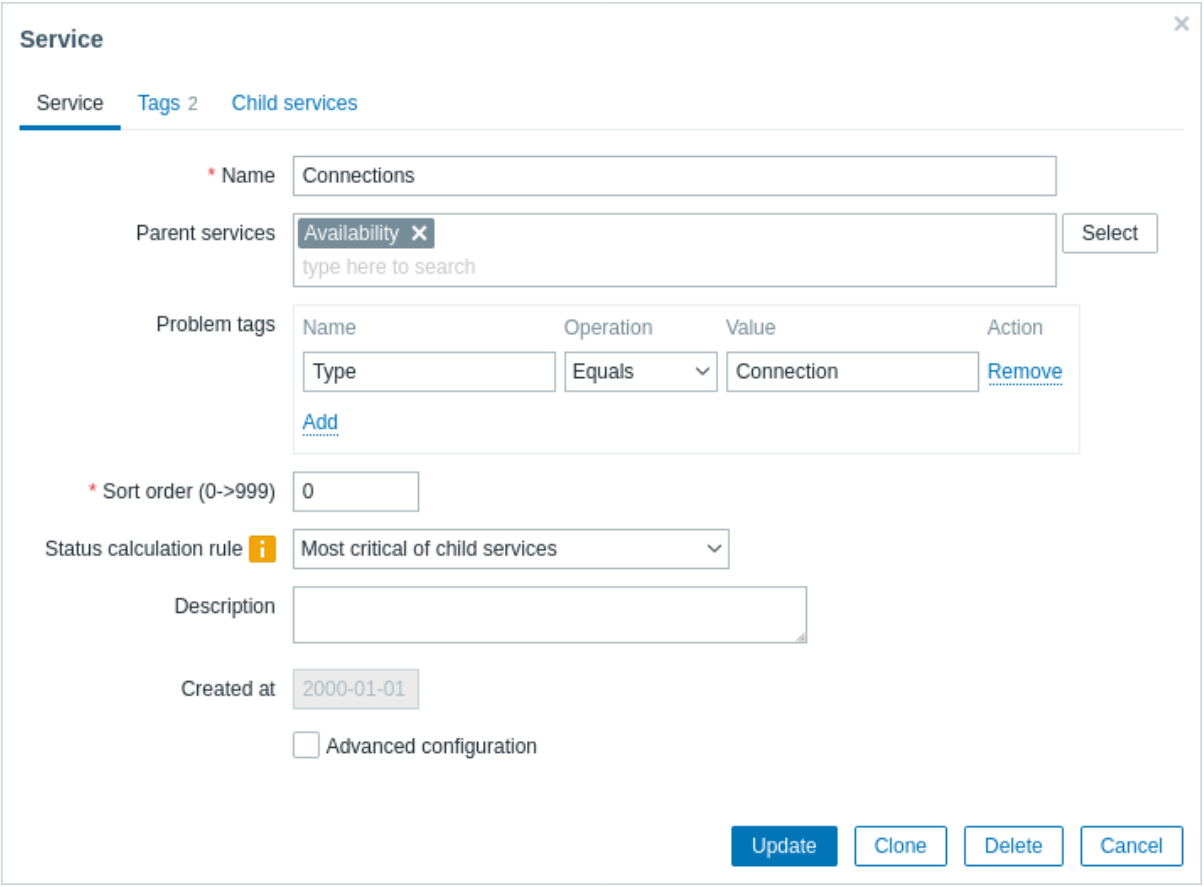
服务树在 *Services->Services* 菜单部分配置。在右上角，从View 切换到 Edit 模式。



要配置 新服务，请单击右上角的 *Create service* 按钮。

要快速添加子服务，您也可以按父服务旁边的加号图标。这将打开相同的服务配置表单，但将预先填写父服务参数。

服务配置　在 **Service** 选项卡中，指定所需的服务参数：

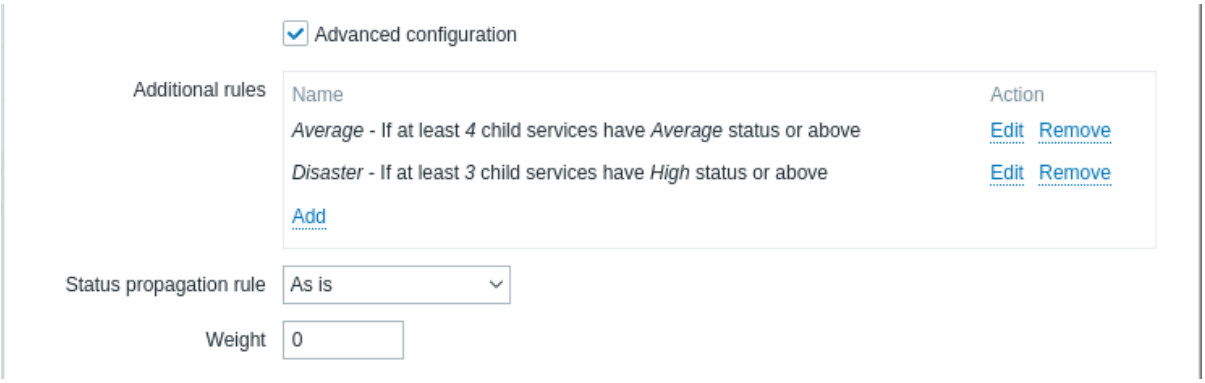


所有必填输入字段均标有红色星号。

参数	说明
Name	服务名称。
Parent services	服务所属的父服务。 如果您要添加最高级别的服务，请将此字段留空。 一个服务可以有多个父服务。在这种情况下，它将显示在每个父服务下的服务树中。

参数	说明
Problem tags	指定标签将问题数据映射到服务： **Equals** - 包括指定的标签名和值（区分大小写） **Contains** - 包括指定的标签名，其中标签值包含输入的字符串（子字符串的匹配不区分大小写） 标签名的匹配始终区分大小写。
Sort order	显式的排序顺序，最低的排第一位。
Status calculation rule	状态计算规则： **Most critical if all children have problems** - 如果所有子节点都有问题，则根据子节点中最严重的问题对服务状态进行着色。 **Most critical of child nodes** - 根据子节点中最严重的问题对服务状态进行着色 **Set status to OK** - 不计算服务状态 勾选下面的 高级配置复选框以配置其他状态计算规则。
Description	服务说明。
Advanced configuration	勾选复选框以访问高级配置 选项。

高级配置



参数	说明
Additional rules	单击 *Add* 以定义其他状态计算规则。
Set status to	如果 condition 匹配，则将服务状态设置为 *OK*（默认）、*Not classified*、*Information*、*Warning*、*Average*、*High* 或 *Disaster*。
Condition	选择子服务的条件： 如果至少 **(N)** 个子服务具有 **(Status)** 表示的状态或以上状态 如果至少 **(N%)** 的子服务具有 **(Status)** 表示的状态或以上状态 如果少于 **(N)** 个子服务具有 **(Status)** 表示的状态或以下状态 如果少于 **(N%)** 的子服务具有 **(Status)** 表示的状态或以下状态 如果具有 **(Status)** 状态的子服务的权重至少为 **(W)** 如果具有 **(Status)** 状态的子服务的权重至少为 **(N%)** 如果具有 **(Status)** 状态的子服务的权重小于 **(W)** 如果具有 **(Status)** 状态的子服务的权重少于 **(N%)** 如果指定了多个条件，并且情况与多个条件匹配，则将设置最高的严重性。
N (W)	在 condition 中设置 N 或 W (1-100000) 或 N% (1-100) 的值。
Status	选择 condition 中 *Status* 的值：*OK* (默认)、*Not classified*、*Information*、*Warning*、*Average*、*High* 或 *Disaster*。
Status propagation rule	将服务状态传播到父服务的规则： **As is** - 传播状态没有变化 **Increase by** - 您可以将传播的状态增加 1 到 5 个严重性 **Decrease by** - 您可以将传播的状态减少 1 到 5 个严重性 **Ignore this service** - 状态根本不会传播到父服务 **Fixed status** - 状态以静态方式传播，即始终保持不变
Weight	服务的权重（0（默认）到 1000000 之间的整数）。

Note:
附加状态计算规则只能用于将严重级别提高到根据 *Status calculation rule* 参数计算的级别之上。如果根据 additional rules，状态应为警告，但 *Status calculation rule* 状态为灾难 - 服务将具有灾难状态。

Tags 选项卡包含服务级别标签。服务级别标签用于标识服务。这种类型的标签不用于将问题映射到服务（为此，使用第一个选项卡中的问题标签）。

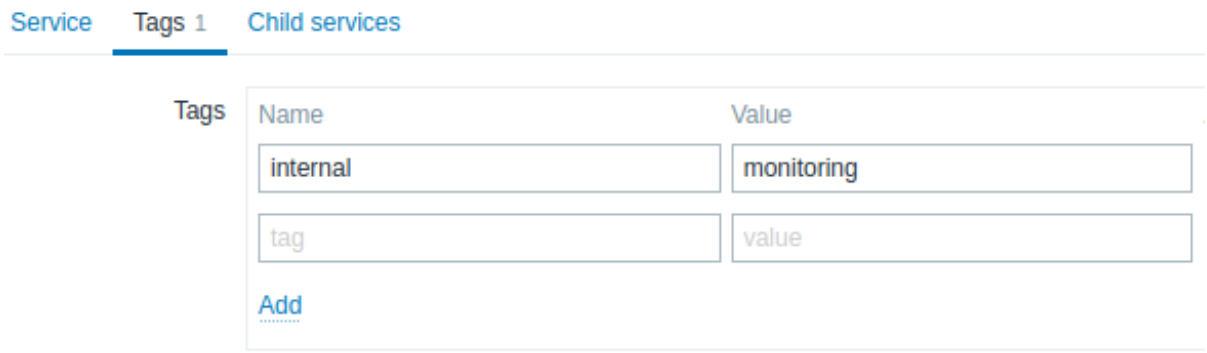
Child services 选项卡允许指定依赖服务。单击 *Add* 从现有服务列表中添加服务。如果要添加新的子服务，请先保存此服务，然后单击刚刚创建的服务旁边的加号图标。

标签　服务中有两种不同类型的标签：

- 服务标签
- 问题标签

服务标签

服务标签用于将服务与service actions 和SLA 相匹配。这些标签在 *Tabs* 服务配置选项卡中指定。对于映射 SLA，使用 *OR* 逻辑：如果服务至少有一个匹配的标签，则将其映射到 SLA。在服务操作中，映射规则是可配置的，可以使用 *AND*、*OR*、或 *AND/OR* 逻辑。

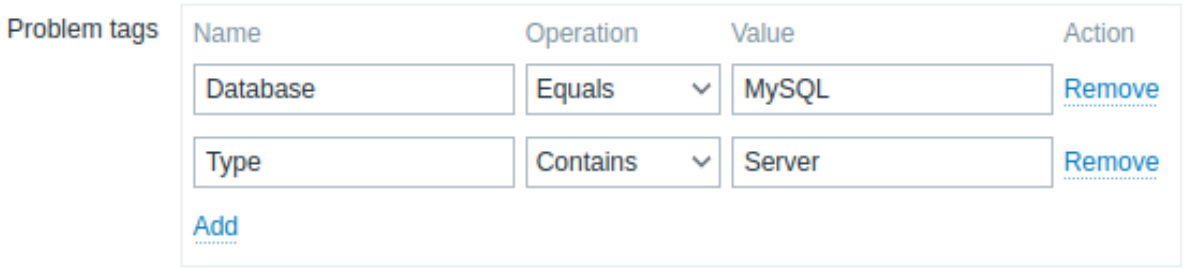


问题标签

问题标签用于匹配问题和服务。这些标签在主服务配置选项卡中指定。

只有最低层次级别的子服务可以定义问题标签并直接关联问题。如果问题标签匹配，服务状态将更改为与问题相同的状态。在多个问题的情况下，服务将具有最严重的状态。然后根据状态计算规则基于子服务状态计算父服务的状态。

如果指定了多个标签，则使用 *AND* 逻辑：问题必须将服务配置中指定的所有标签都映射到服务。

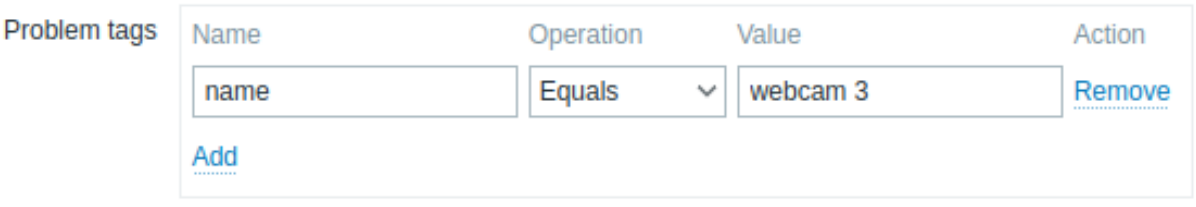


Note:
Zabbix 中的问题从模板、主机、项目、Web 场景和触发器的整个链中继承标签。这些标签中的任何一个都可用于匹配问题与服务。

示例：

问题 *Web camera 3 is down* 有标签 `type:video surveillance`，`floor:1st` 和 `name:webcam 3` ，状态为 *Warning*

服务 **Web camera 3** 具有指定的唯一问题标签：`name:webcam 3`



检测到此问题时，服务状态将从 *OK* 变为 *Warning*。

如果服务 **Web camera 3** 有问题标签 `name:webcam 3` 和 `floor:2nd`，则在检测到问题时不会更改其状态，因为仅部分条件满足。

修改现有服务

> **Note:**
> 以下描述的按钮仅在 *Services* 部分处于编辑模式时可见。

要编辑现有服务，请按服务旁边的铅笔图标。

要克隆现有服务，请按铅笔图标打开其配置，然后按克隆按钮。克隆服务时，会保留其父链接，而不会保留子链接。

要删除服务，请按服务旁边的 x 图标。删除父服务时，其子服务将不会被删除，并将在服务树中向上移动一级（第一级子服务将获得与删除的父服务相同的级别）。

服务列表下方的两个按钮提供了一些批量编辑选项：

- *Mass update* - 批量更新服务属性
- *Delete* - 删除服务

要使用这些选项，请在相应服务之前勾选复选框，然后单击所需按钮。

2 服务动作

概述　在本节中，您可以查看和配置服务动作

如果您希望某些动作因服务状态更改（OK ⇔ PROBLEM）而发生，则服务动作很有用，例如：

- 发送信息
- 重启 web 服务器

服务动作在功能上类似于 Zabbix 中的其他动作类型（例如，触发器动作）。

配置　要创建新的服务动作，请转到 *Services* 菜单的 *Service actions* 子部分，然后单击右上角的 *Create action*。

服务动作的配置方式与 Zabbix 中其他类型的动作相同。有关更多详细信息，请参阅配置动作。

主要区别在于：

- 用户对服务动作的访问取决于用户角色 授予的对服务的访问权限。
- 服务动作支持不同的条件 集。

条件　以下条件可用于服务动作：

条件类型	支持的运算符	说明
Service	等于 不等于	设置服务。 **equals** - 事件等于此服务。 **does not equal** - 事件不等于此服务。 指定父服务会隐式选择所有子服务。要仅指定父服务，必须使用 **does not equal** 运算符额外设置所有嵌套服务。
Service name	包含 不包含	设置服务名中的字符串。 **contains** - 事件由服务生成，服务名称中包含此字符串。 **does not contain** - 在服务名中找不到此字符串。

条件类型	支持的运算符	说明
Service tag name	等于 不等于 包含 不包含	设置事件标签。服务事件标签可以在服务配置部分 *Tag* 中定义。 **equals** - 事件标签名等于 tag 值 **does not equal** - 事件标签名不等于 tag 值 **contains** - 事件标签名包含指定字符串 **does not contain** - 事件标签名不包含指定字符串。
Service tag value	等于 不等于 包含 不包含	设置事件标签和值的组合。服务事件标签可以在服务配置部分 *Tag* 中定义。 **equals** - 事件标签值等于 value 值 **does not equal** - 事件标签值不等于 value 值 **contains** - 事件标签的值包含指定字符串 **does not contain** - 事件标签的值不包含指定字符串。

Attention:
确保在 *Administration->Media types* 菜单中为服务动作定义消息模板。否则，将不会发送通知。

3 服务级别协议 **SLA**

概述　创建服务 后, 就可以开始监控其性能是否满足服务级别协议 (SLA)。

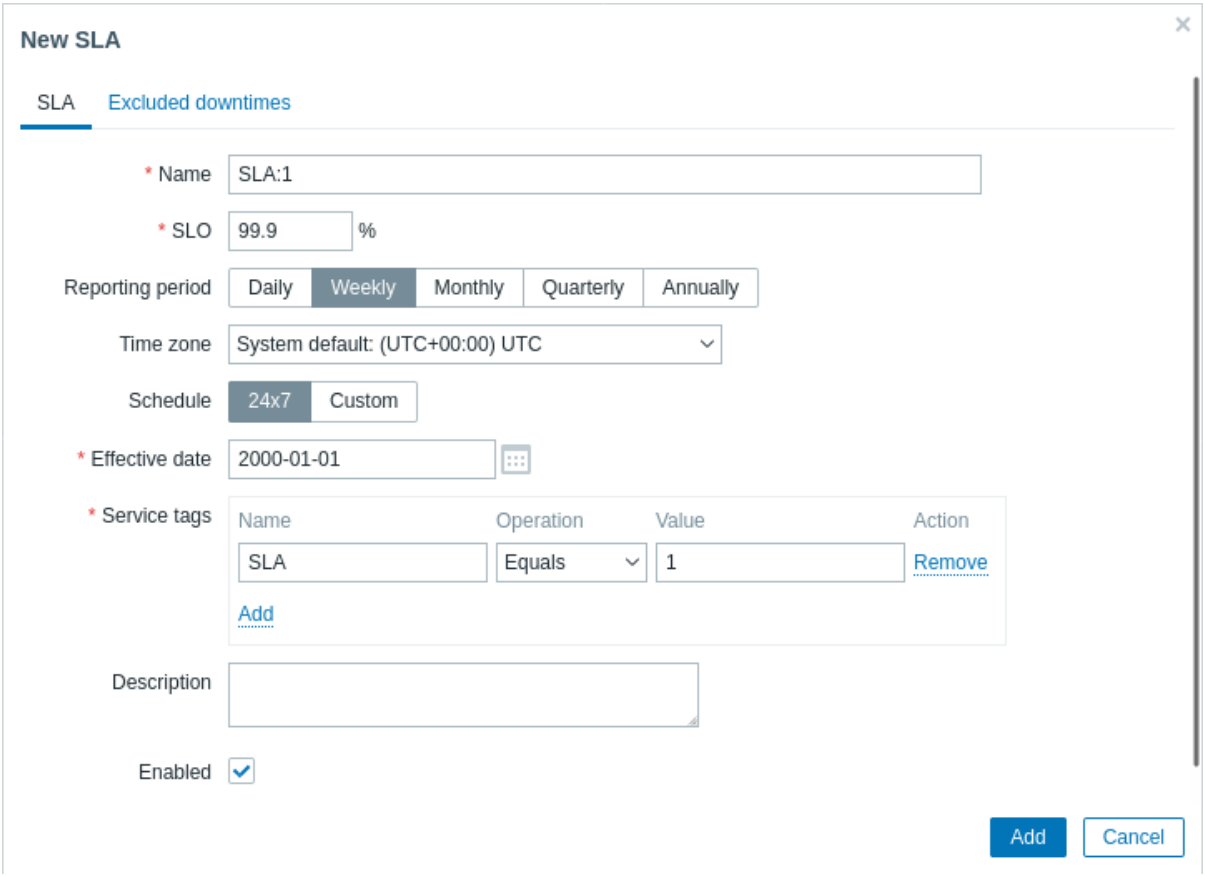
可以在服务*->SLA* 界面配置 SLA。SLA 定义了服务级别目标 (SLO)、预期正常运行时间和计划内的停机时间。

SLA 和服务通过服务标签匹配。同一个 SLA 可用于多个服务 - 其中每个服务的性能会进行分别测量。单一服务可以匹配多个 SLA - 其中每个 SLA 的数据会各自独立显示。

SLA 报告包含服务级别指标 (SLI) 数据，该数据表示实际的服务可用性。把 SLO(预期的可用性，以百分比% 显示) 和 SLI(真实的可用性，以百分比% 显示) 进行比较，可以确定服务是否满足 SLA 目标。

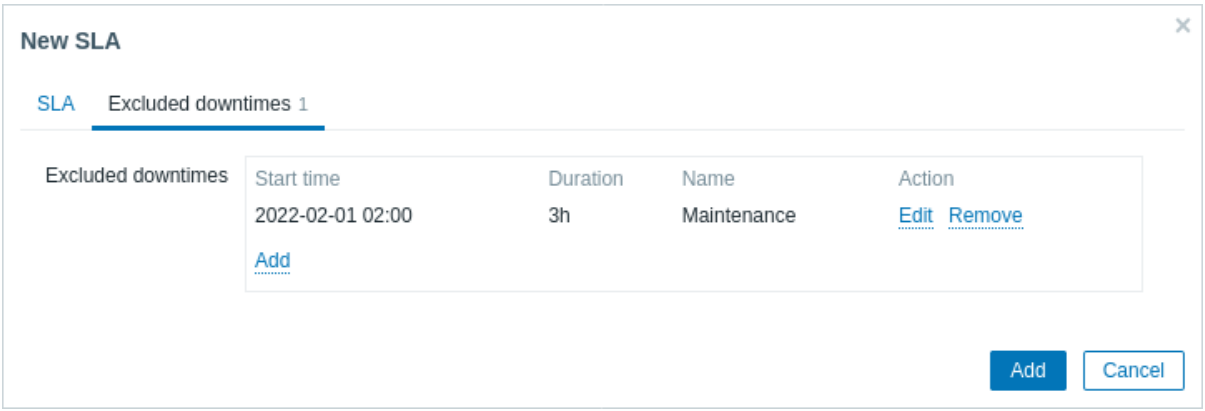
配置　点击 创建 *SLA* 按钮新建一个 SLA。

SLA 面板可以指定通用的 SLA 参数。



参数	描述
名称	输入 SLA 名称。
SLO	输入服务级别目标 (SLO)，格式为百分比。
报告周期	SLA 报告中使用的时间周期 - 每天, 每周, 每月, 每季度, 或 每年。
时区	SLA 的时区。
时间表	SLA 运行的时间表 - 24x7 或自定义。
生效日期	开始 SLA 计算的日期。
服务标签	添加服务标签，用于识别 SLA 对应的服务。 名称 - 服务标签名称，必须严格匹配，大小写敏感。 操作 - 如果标签值必须严格匹配 (大小写敏感)，就选择 等于。如果部分标签值匹配 (大小写不敏感)，则选择 包含。 值 - 需要进行搜索的服务标签值。 如果至少匹配了一个服务标签，则 SLA 应用于该服务。
描述	为 SLA 添加描述。
启用	勾选检查框来启用 SLA。

例外停机时间面板可以指定停机时间，SLA 不会将其计算在内。



点击 添加来配置例外停机时间，输入时间周期名称，开始日期和持续时长。

SLA 报告　SLA 报告中展示了服务的性能与 SLA 之间的对比数据。SLA 报告可以通过以下方式查看：

- 点击 *SLA* 界面的 SLA 报告超链接;
- 点击 服务界面下的信息 (info) 面板上的 SLA 名称;
- 在仪表盘 (Dashboard) 的窗体小部件 (widget) 中添加 *SLA* 报告.

SLA 配置好以后，服务界面下的信息面板同样会显示一些有关服务性能的信息。

4 配置示例

概述　此章节展示了监控 Zabbix 高可用集群作为服务的一个简单配置示例。

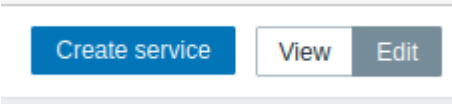
先决条件　在配置服务监控前，需要先配置对应的主机:

- *HA node 1* 至少一个触发器，至少一个标签 (最好是在触发器级别设置) `component:HA node 1`
- *HA node 2* 至少一个触发器，至少一个标签 (最好是在触发器级别设置) `component:HA node 2`

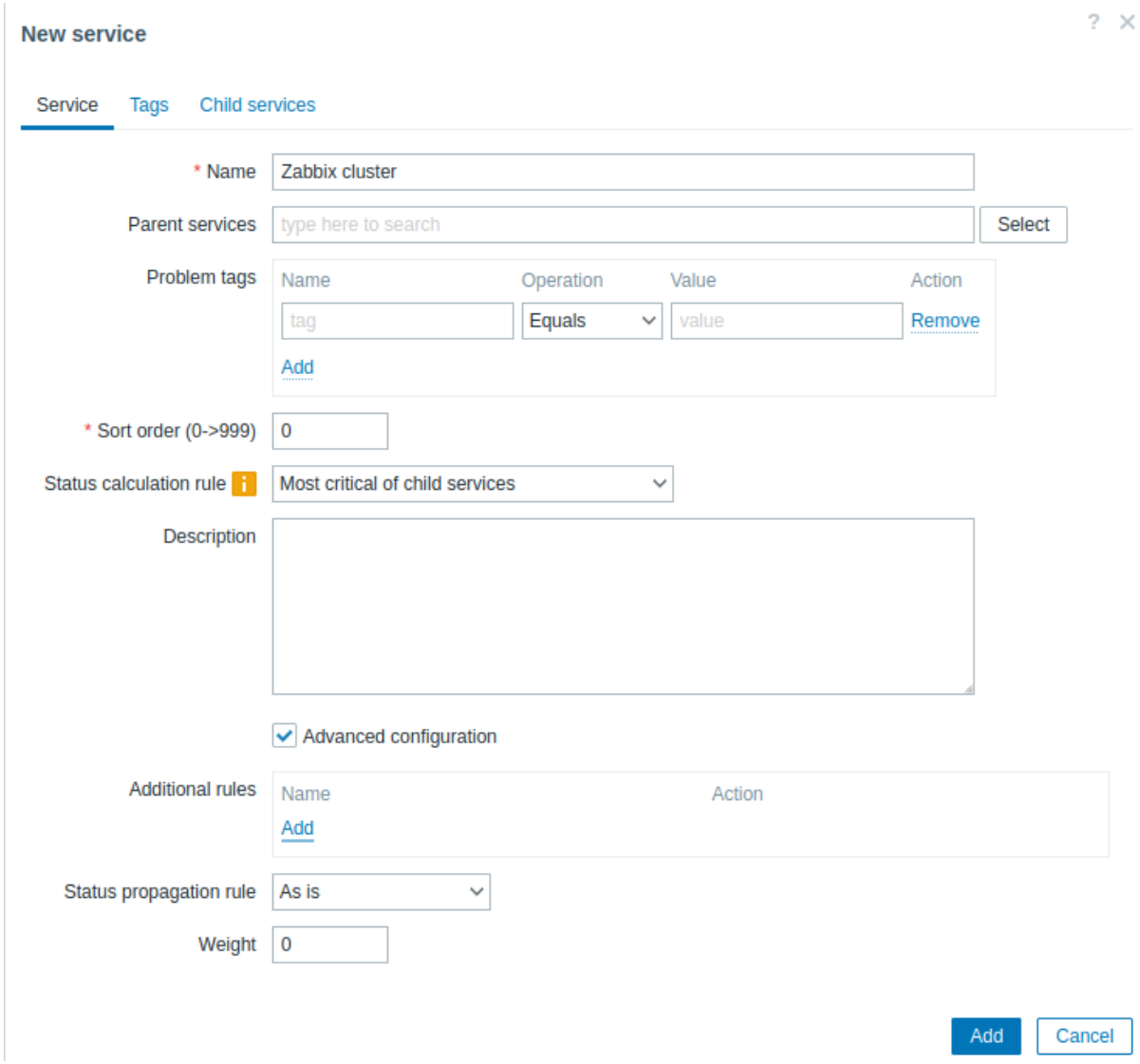
服务树　下一步是建立服务树。在此示例中，只包含基础的设置以及三个服务：*Zabbix cluster* (父) 和两个子服务 *Zabbix server node 1* 和 *Zabbix server node 2*.

```
Zabbix cluster
|
|- Zabbix server node 1
|- Zabbix server node 2
```

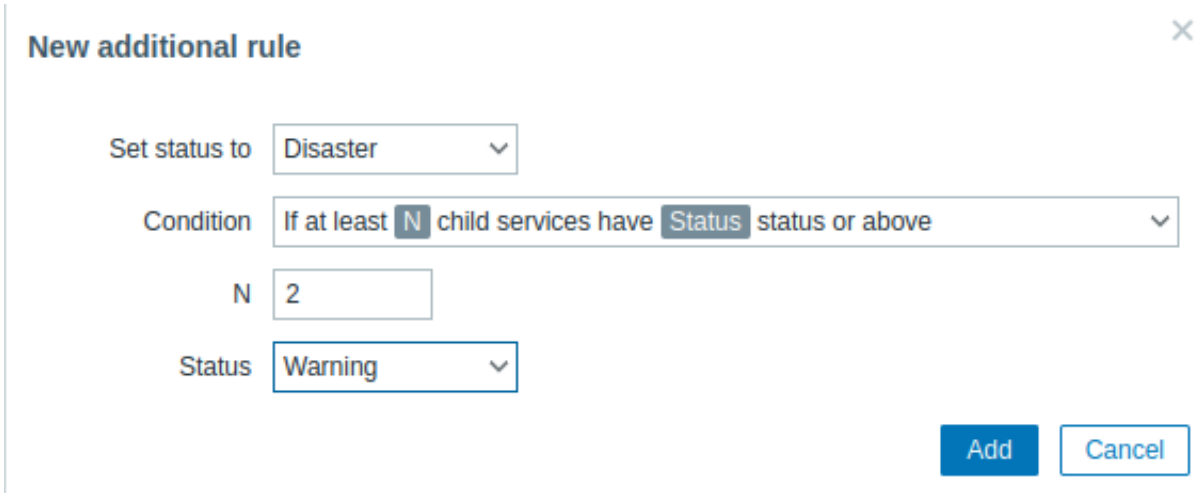
在服务页面，打开 编辑模式并点击创建服务：



在服务配置窗口中，输入名称 *Zabbix cluster* 并选取检查框高级配置。



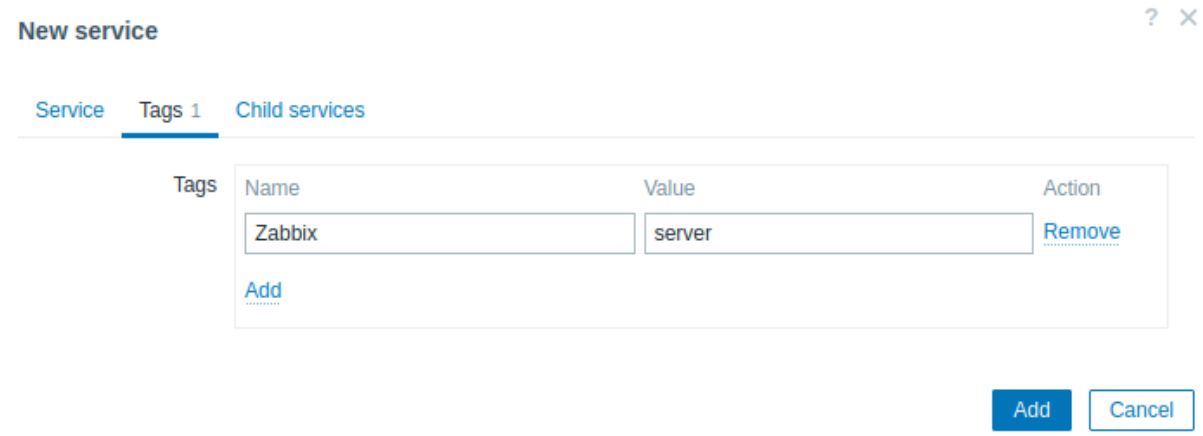
配置额外规则：



Zabbix 集群需要两个子服务 - 用于各自的 HA 节点。如果两个 HA 节点均产生至少为警告级别的告警信息，父服务的状态将设置为灾难。要实现上述需求，需要配置如下的额外规则：

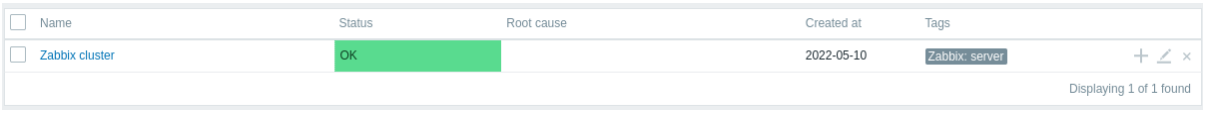
- 设置状态为：灾难
- 条件：如果至少 N 个子服务有状态状态或以上
- N: 2
- 状态: 警告

切换到标签界面并添加标签 `Zabbix:server`。此标签后续会用于服务动作和 SLA 报告。



保存新建的服务。

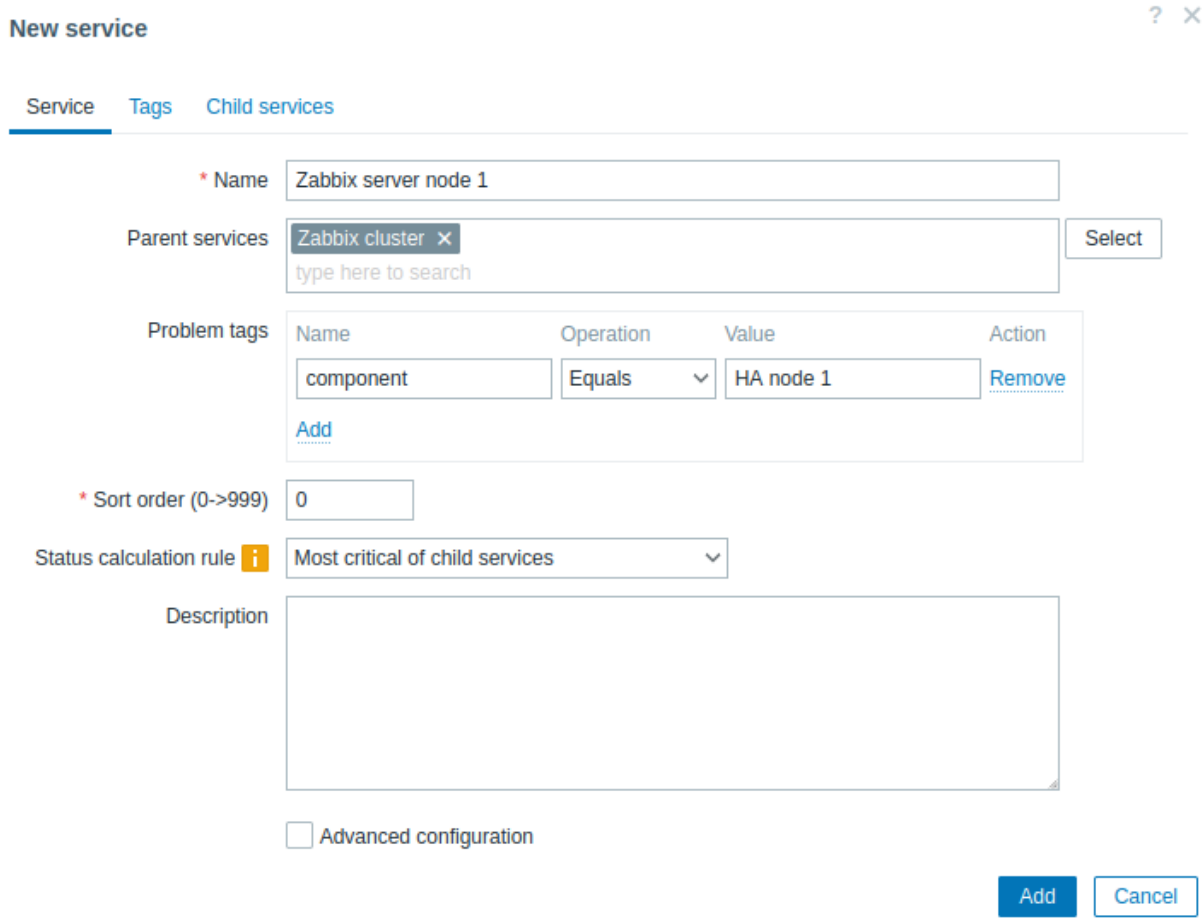
点击 Zabbix 集群服务旁边的加号图标 (只有编辑模式才能看到加号图标) 来创建子服务。



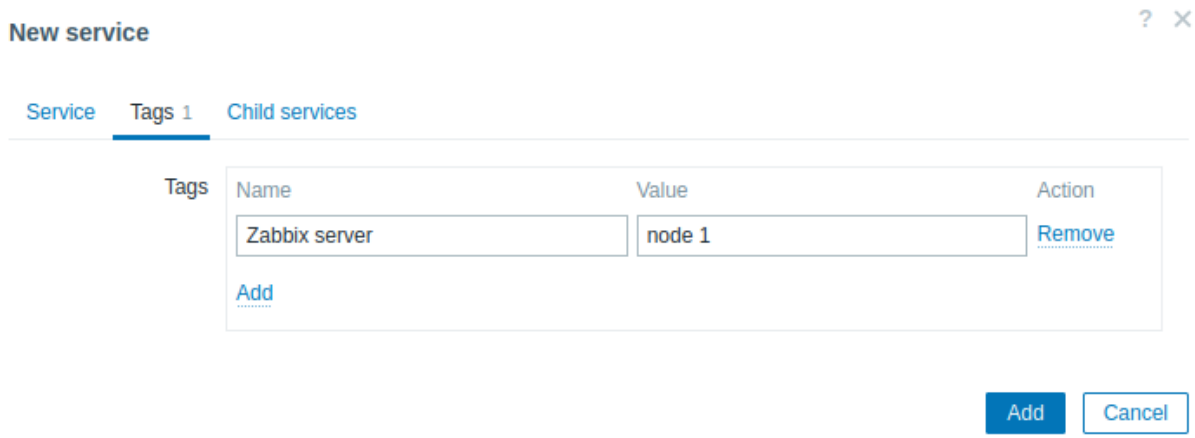
在服务配置窗口输入名称 *Zabbix server node 1*。注意，父服务的参数已经用 *Zabbix* 集群预先填充了。

该服务的可用性受主机 *HA node 1* 产生的问题所影响，通过 `component:HA node 1` 问题标签所标记。在问题标签参数中输入：

- 名称：组件
- 操作: 等于
- 值: HA node 1



切换到 标签面板并添加服务标签：`Zabbix server:node 1`。此标签会用于后续的服务动作和 SLA 报告。



保存新建的服务。

创建另一个子服务”Zabbix server node 2”.

设置问题标签:

- 名称：组件
- 操作：等于
- 值：HA node 2

切换到标签面板并添加服务标签：`Zabbix server:node 2`。

保存新建的服务。

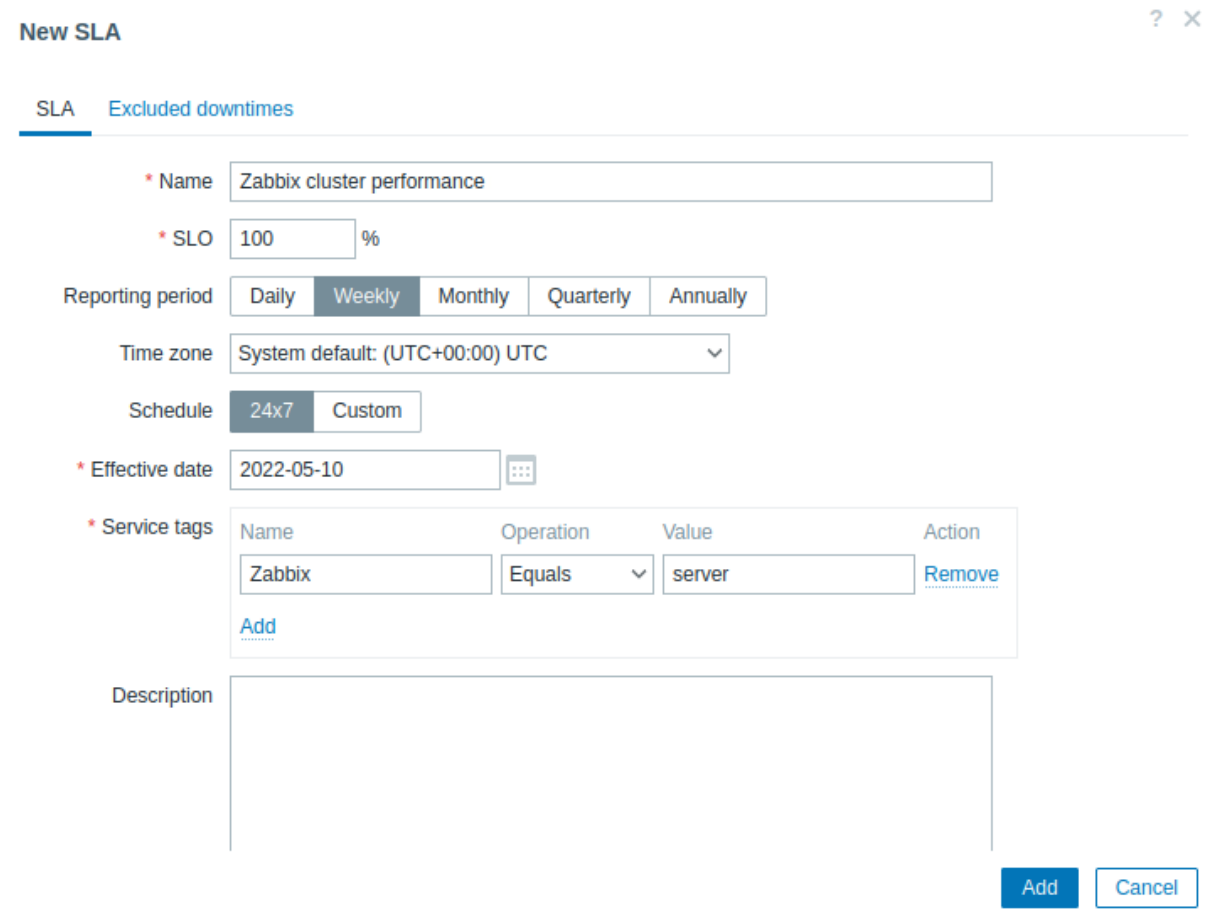
SLA 在此示例中，预期的 Zabbix 集群性能是 100%，排除了每半年一次的一小时维护时间。

首先需要添加一个服务级别协议。

转到 服务->*SLA* 菜单点击创建 SLA。输入名称 *Zabbix* 集群性能并设置 SLO 为 100%。

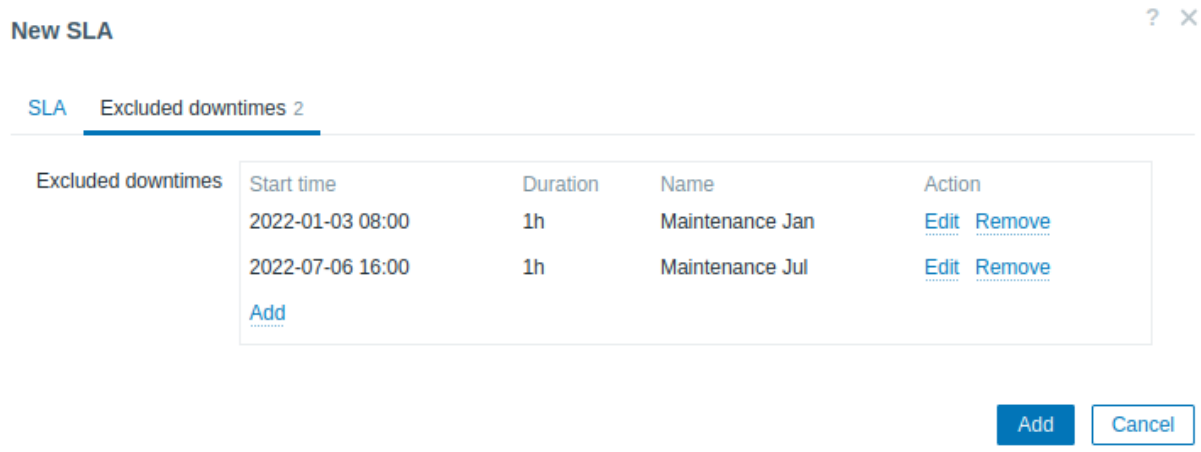
Zabbix 集群有 `Zabbix:server` 标签。要使用该 SLA 来测量 Zabbix 集群的性能，需要在服务标签的参数中指定：

- 名称：Zabbix
- 操作：Equals
- 值：server



在实际设置中，还可以更新所需的报告周期、时区和开始日期，或把时间表从 24/7 改为自定义。就这个例子来说，默认设置就够了。

切换到例外停机时间面板并添加用于例行维护的停机时间，这样 SLA 就不会将这些时间段计算在内了。在例外停机时间面板总点击添加 (Add) 链接，输入停机时间名称、计划开始时间和持续时长。

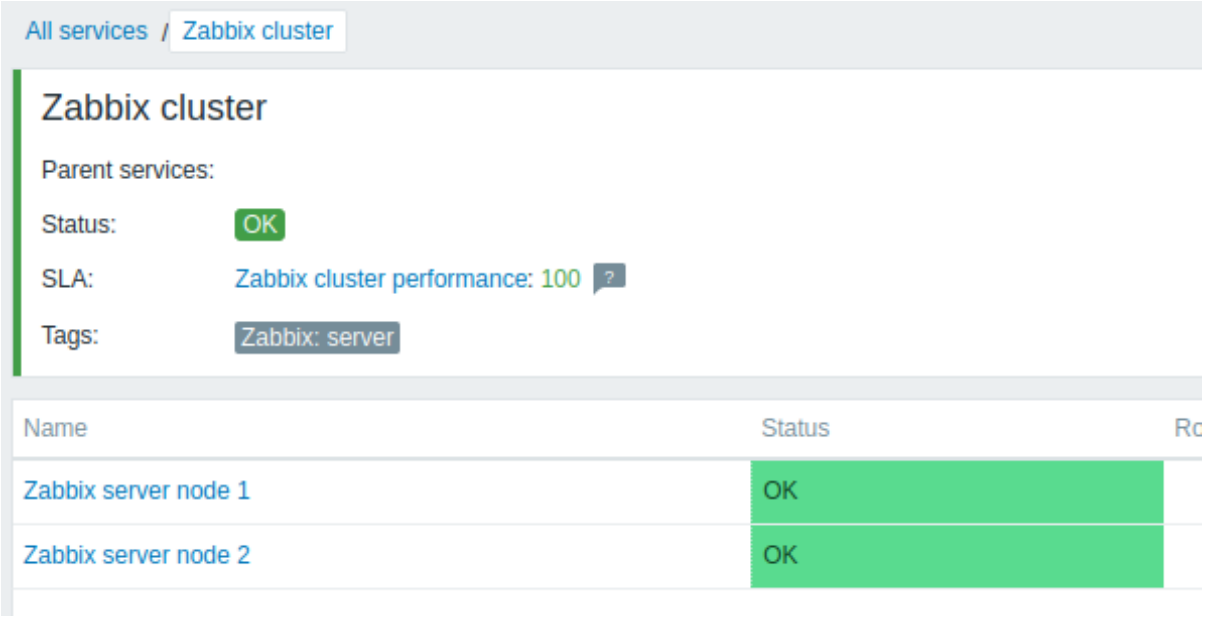


点击添加 (Add) 来保存新建的 SLA。

切换到 SLA 报告界面查看关于 Zabbix 集群的 SLA 报告。

Year	SLO	SLI	Uptime	Downtime	Error budget
2022	100%	100	36m 53s	0	0

还可以在服务界面下查看 SLA 信息。



9. Web 监控

概览　可以通过 Zabbix 检查网站的几个可用性。

Attention:
要执行 web 监控，Zabbix server 的初始配置 必须支持 cURL (libcurl)。

启用 Web 监控需要定义 Web 场景。Web 场景由一个或多个 HTTP 请求或 “步骤” 组成。这些步骤由 Zabbix server 以预设的顺序定期执行。如果主机由 proxy 监控，则这些步骤由 proxy 执行。

Web 场景以与监控项、触发器等相同的方式附加到主机/模板上。这意味着 Web 场景也可以在模板级别上创建，然后一次性应用于多个主机。

在任何 Web 场景中都会收集以下信息：

- 整个场景所有步骤的每秒平均下载速度
- 失败的步骤编号
- 最新的错误信息

在任何 Web 场景步骤中都会收集以下信息：

- 每秒下载速度
- 响应时间
- 响应码

更多详细信息，请参见web 监控项。

执行 Web 场景收集的数据保存在数据库中。数据自动用于图表、触发器和通知。

Zabbix 还可以检查检索到的 HTML 页面是否包含预定义的字符串。它可以执行模拟登录并遵循页面上模拟鼠标点击的路径。

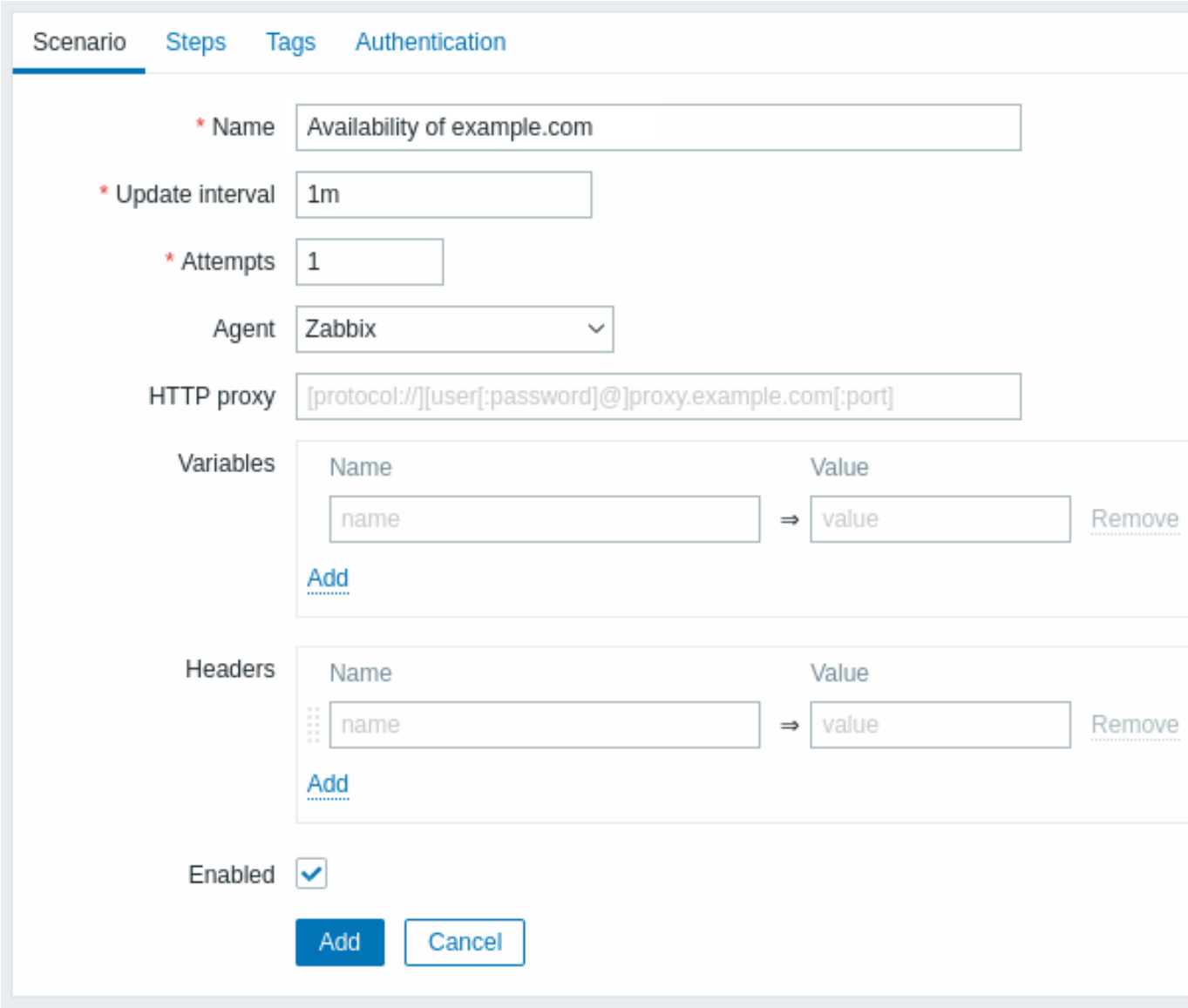
Zabbix web 监控同时支持 HTTP 和 HTTPS。在运行 Web 场景时，Zabbix 将选择性地跟随重定向（请参阅下面的 *Follow redirects* 选项）。最大重定向数硬编码为 10（使用 cURL 选项CURLOPT_MAXREDIRS）。在单个场景的执行过程中会保留所有 cookie。

查看使用 HTTPS 协议进行 Web 监控的已知问题 。

配置 **Web** 场景　配置 Web 场景流程：

- *Configuration*（配置）→ *Hosts*（主机）（或 模板）
- 单击主机/模板行中的 *Web*
- 单击右侧的 *Create scenario*（创建场景）（或在场景名称上编辑现有场景）
- 在表格中输入场景的参数

场景选项卡可以配置 Web 场景的常规参数。



所有必填字段都标有红色星号。

场景参数：

参数	描述
Host	场景所属的主机/模板的名称。
Name	场景的唯一名称。
Update interval	场景执行的频率。 支持时间后缀 ，例如 30s、1m、2h、1d。 支持用户宏 。注意：如果使用用户宏并更改其值 (例如 5m → 30s)，则将根据先前的值执行下一次检查（示例值在更远的将来）。
Attempts	尝试执行 Web 场景步骤的次数。如果出现网络问题（超时、无连接等），Zabbix 可以多次重复执行一个步骤。图集同样会影响场景的每个步骤。 最多可以指定 10 次尝试，默认值为 1。 注意：Zabbix 不会因为响应码错误或所需字符串不匹配而重复步骤。
Agent（客户端）	选择客户端。 Zabbix 会伪装成被选中的浏览器。当网站为不同的浏览器返回不同的内容时，这很有用。 用户宏可用于该字段。

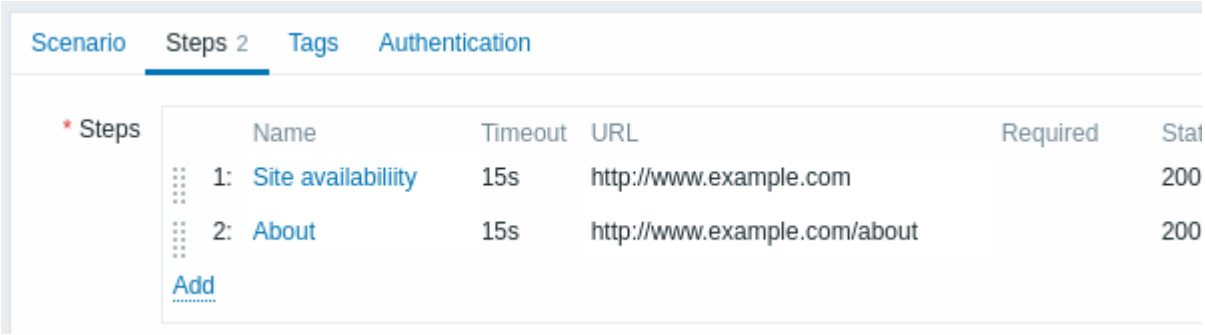
参数	描述
HTTP proxy（*HTTP* 代理）	您可以按照示例要求的格式使用 HTTP 代理 `[protocol://][username[:password]@]proxy.example.com[:port]`。 这将设置 CURLOPT_PROXY cURL 选项。 可选 `protocol://` 前缀可用于指定替代代理协议（cURL 7.21.7 中添加了协议前缀支持）。如果未指定协议，代理将被视为 HTTP 代理。 默认情况下，将使用 1080 端口。 如果指定，代理将覆盖代理相关的环境变量，如 http_proxy, HTTPS_PROXY。如果未指定，代理将不会覆盖与代理相关的环境变量。 输入的值 “按原样” 传递，不进行完整性检查。 您也可以输入 SOCKS 代理地址。如果您指定了错误的协议，则连接将失败并且该项目将变得不受支持。 注意：HTTP 代理仅支持简单身份验证。 用户宏可用于该字段。
Variables（变量）	可能在场景步骤中使用的变量（URL、post 变量）。 它们具有以下格式： **{macro1}**=value1 **{macro2}**=value2 **{macro3}**=regex:< 正则表达式 > 示例： {username}=Alexei {password}=kj3h5kJ34bd {hostid}=regex:hostid is ([0-9]+) 然后可以在步骤中以 {username}, {password} 和 {hostid} 的形式引用宏。Zabbix 会自动将它们替换为实际值。注意，变量 `regex:` 需要一步来获取正则表达式的值，因此提取的值只能应用于后面的步骤。 如果值部分以 `regex:` 然后将其后面的部分视为搜索网页的正则表达式，如果找到，则将匹配项存储在变量中。必须至少存在一个子组，以便可以提取匹配值。 支持用户宏和 {HOST.*} 宏 。 变量在用于查询字段或用于发布变量的表单数据时会自动进行 URL 编码，但在原始发布或直接在 URL 中使用时必须手动进行 URL 编码。
Headers（标头）	执行请求时将发送的自定义 HTTP 标头。可以使用默认和自定义标头。 标头将使用默认设置分配，具体取决于从场景级别的下拉列表中选择的代理类型，并将应用于所有步骤，除非它们是在步骤级别自定义定义的。 应该注意的是，在步骤级别定义标头会自动丢弃所有先前定义的标头，但通过从场景级别的下拉列表中选择 **“用户代理”** 来分配的默认标头除外。 但是，即使是 “User-Agent” 默认标头也可以通过在步骤级别指定它来覆盖。 要在场景级别取消设置标头，标头应该在步骤级别上命名并且属性不设置值。 标头应使用与它们在 HTTP 协议中出现的相同语法列出，可选择使用 CURLOPT_HTTPHEADER 选项支持的一些附加功能。 例如： Accept-Charset=utf-8 Accept-Language=en-US Content-Type=application/xml; charset=utf-8 用户宏和 {HOST.*} 宏 是支持的.
Enabled	如果选中此框，则该场景处于活动状态，否则 - 禁用。

请注意，在编辑现有场景时，表单中有两个额外的按钮可用：

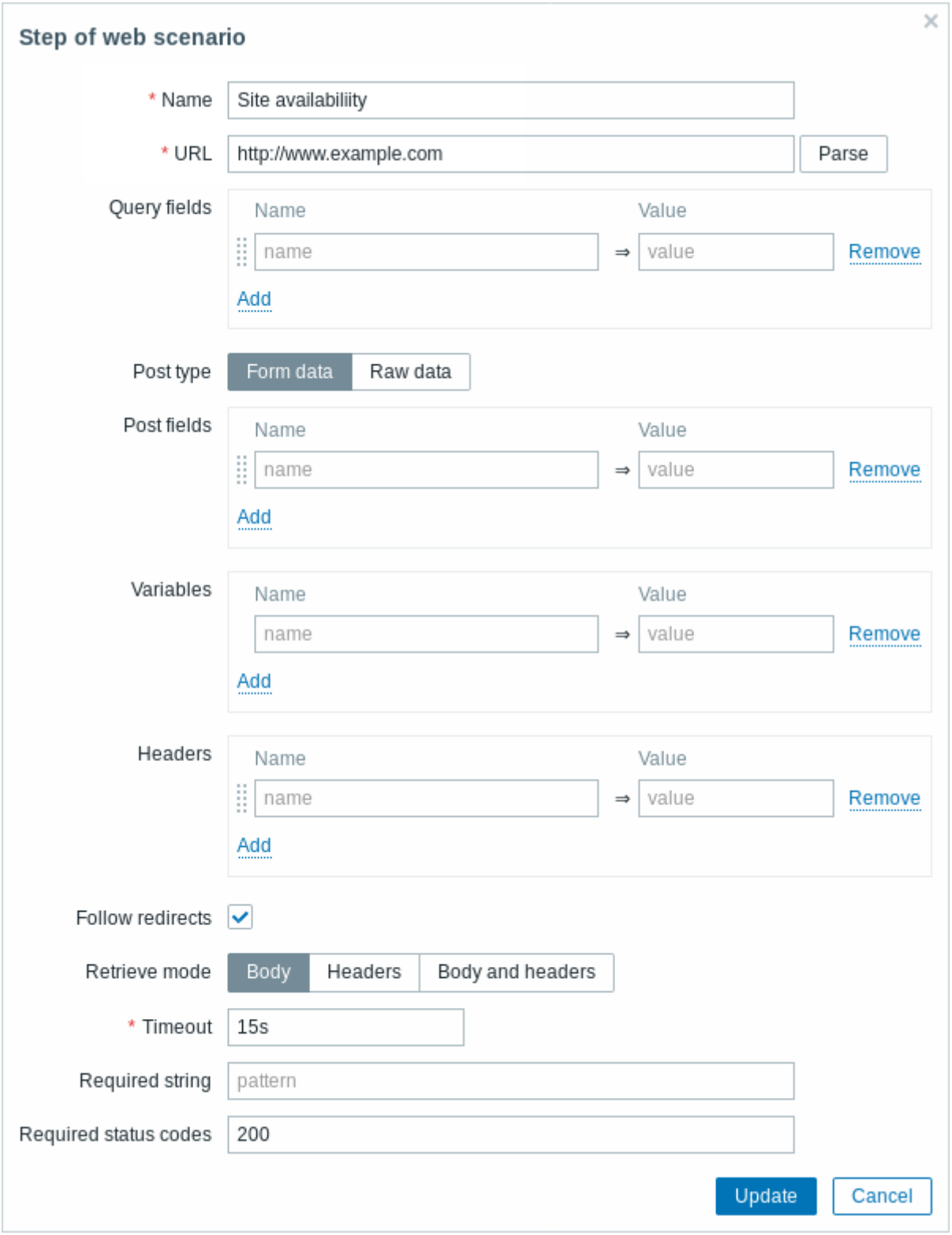
Clone	根据现有场景的属性创建另一个场景。
Clear history and trends	删除场景的历史和趋势数据。这将使服务器在删除数据后立即执行场景。

Note:
如果 *HTTP* 代理字段留空，使用 HTTP 代理的另一种方法是设置代理相关的环境变量。
对于 HTTP 检查 - 为 Zabbix server 用户设置 **http_proxy** 环境变量。例如，`http_proxy=http://proxy_ip:proxy_port`。
对于 HTTPS 检查 - 为 Zabbix server 用户设置 **HTTPS_proxy** 环境变量。例如，`HTTPS_PROXY=http://proxy_ip:proxy_port`。运行 shell 命令获得更多详细信息：*# man curl*。

Steps（步骤）选项卡可以配置 Web 场景步骤。要添加 Web 场景步骤，请单击 *Steps*（步骤）块中的 *Add*（添加）。



Note:
不得在 URL 中使用加密用户宏 因为它们会被解析成”******”。



配置步骤

步骤参数：

参数	描述
Name	唯一的步骤名称。

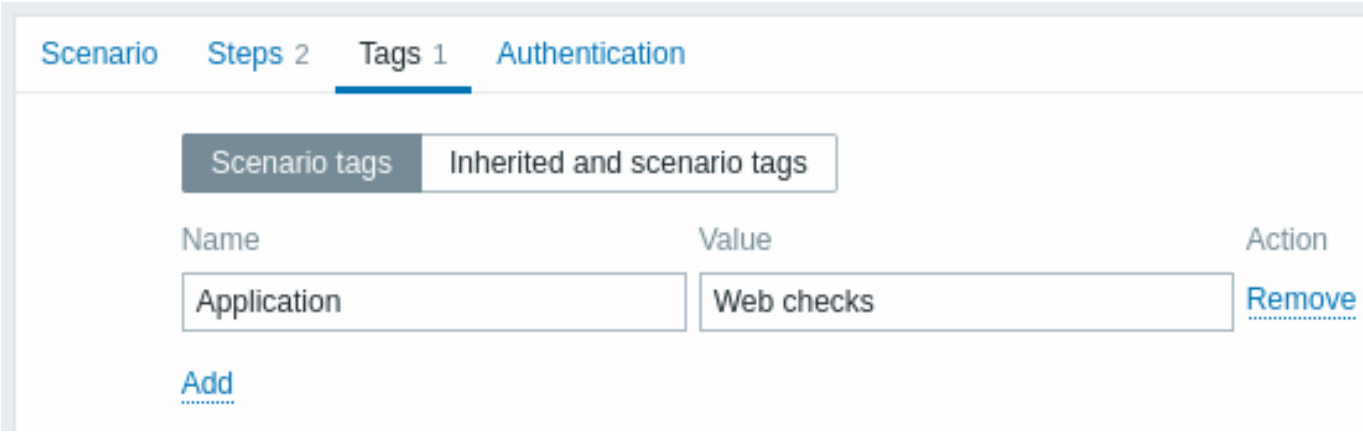
参数	描述
URL	用于连接和检索数据的 URL。例如： https://www.example.com http://www.example.com/download 域名可以用 Unicode 字符指定。在执行 Web 场景步骤时，它们会自动转换为 ASCII 码。 *Parse* 按钮可用于将可选查询字段（如?name=Admin&password=mypassword）与 URL 分开，将属性和值移动到 查询字段中以进行自动 URL 编码。 变量可以在 URL 中使用，使用 {macro} 语法。可以使用 {{macro}.urlencode()} 语法手动对变量进行 URL 编码。 支持用户宏和 {HOST.*} 宏 限制为 2048 个字符。
Query fields	URL 的 HTTP GET 变量。 指定为属性和值对。 值会自动进行 URL 编码。来自场景变量、用户宏或 {HOST.*} 宏的值被解析，然后自动进行 URL 编码。使用 {{macro}.urlencode()} 语法将对它们进行双重 URL 编码。 支持用户宏和 {HOST.*} {HOST.*} 宏 。
Post	HTTP POST 变量。 在 表单数据模式下，指定为属性和值对。 值会自动进行 URL 编码。来自场景变量、用户宏或 {HOST.*} 宏的值被解析，然后自动进行 URL 编码。 在 原始数据模式下，属性/值显示在单行上，并与 **&** 符号连接。 可以使用 {{macro}.urlencode()} 或 {{macro}.urldecode()} 语法手动对原始值进行 URL 编码/解码。 例如：id=2345&userid={user} 如果 {user} 被定义为 web 场景的变量，执行步骤时会被替换为它的值。如果您希望对变量进行 URL 编码，请将 {user} 替换为 {{user}.urlencode()}. 支持用户宏和 {HOST.*} {HOST.*} 宏 。
Variables	可用于 GET 和 POST 函数的步骤级变量。 指定为属性和值对。 步骤级变量会覆盖场景级变量或上一步中的变量。但是，步骤级变量的值仅影响之后的步骤（而不影响当前步骤）。 它们具有以下格式： **{macro}**=value **{macro}**=regex:<regular expression> 有关详细信息，请参阅场景 级别的变量描述。 变量在用于查询字段或用于发布变量的表单数据时会自动进行 URL 编码，但在原始发布或直接在 URL 中使用时必须手动进行 URL 编码。
Headers	执行请求时将发送的自定义 HTTP 标头。 指定为属性和值对。 步骤级别的标题将覆盖为场景指定的标题。 ** 应该注意的是，在步骤级别定义标头会自动丢弃所有先前定义的标头，但通过从场景级别的下拉列表中选择 “用户代理” 来分配的默认标头除外。* <*br*> 但是，即使是 *“User-Agent”* 默认标头也可以通过在步骤级别指定它来覆盖。<*br*> 例如，设置没有值的*’User-Agent’* 属性将删除在场景级别设置的 *User-Agent* 值。<*br*> 支持用户宏和 *{HOST.*}* 宏。<*br*> 设置 *CURLOPT_HTTPHEADER cURL* 选项。<*br*> 从 Zabbix 2.4 开始 * 支持指定自定义标头。
Follow redirects	标记复选框以遵循 HTTP 重定向。 设置 CURLOPT_FOLLOWLOCATION cURL 选项。
Retrieve mode	选择恢复模式： **Body** - 从 HTTP 响应中仅检索正文 **Headers** - 从 HTTP 响应中仅检索标头 **Body and headers** - 从 HTTP 响应中检索正文和标头
Timeout	Zabbix 处理 URL 的时间不会超过设定的时间（从 1 秒到最长 1 小时）。实际上，这个参数定义了连接到 URL 的最长时间和执行 HTTP 请求的最长时间。因此，Zabbix 在该步骤上花费的时间不会超过 **2 x Timeout** 秒。 支持时间后缀 ，例如 30s、1m、1h。支持用户宏 。

参数	描述
Required string	要求正则表达式模式。 除非检索到的内容 (HTML) 与所需的模式匹配，否则该步骤将失败。如果为空，则不检查所需的字符串。 例如： Zabbix Welcome.*admin 主页 注意：该字段不支持引用在 Zabbix 前端创建的正则表达式 。 支持用户宏和 {HOST.*} {HOST.*} 宏 。
Required status codes	预期的 HTTP 状态代码列表。如果 Zabbix 获得不在列表中的代码，则该步骤将失败。 如果为空，则不检查状态代码。 例如：支持 200,201,210-299 支持用户宏。

Note:
Web 场景步骤中的任何更改只有在保存整个场景时才会保存。

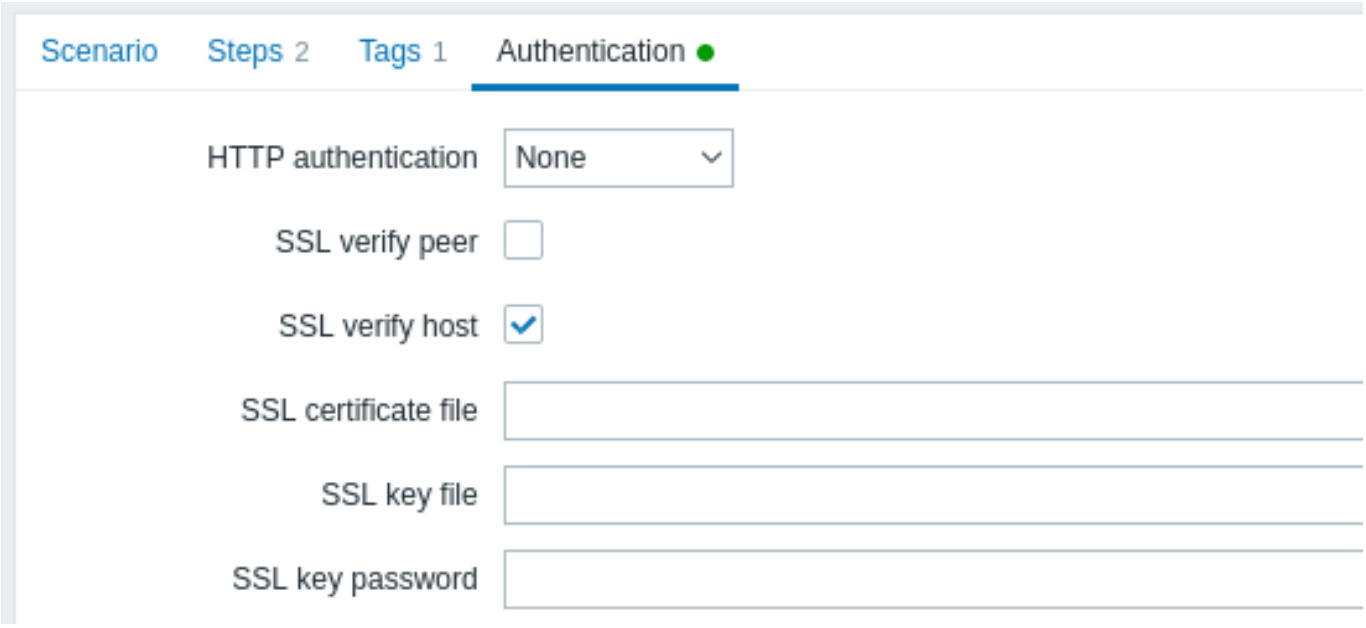
查看配置 Web 监控步骤的真实案例 。

配置标签　标签选项卡允许定义场景级别的标签。



标签允许过滤 Web 场景和 Web 监控项。

配置认证　认证选项卡允许配置场景身份认证选项。选项卡名称旁边的绿点表示启用了某种类型的 HTTP 身份验证。



认证参数：

参数	描述
Authentication（认证）	身份认证选项。 **None**（无） - 不使用身份认证。 **Basic**（基本） - 使用基本身份验证。 **NTLM** - 使用 NTLM（Windows NT LAN Manager）身份认证。 **Kerberos** - 使用 Kerberos 身份验证。另请参阅：Zabbix 配置 Kerberos 认证. **Digest** - 使用 Digest 身份认证。 选择身份认证将提供两个额外的字段用于输入用户名和密码。 用户宏可用于用户和密码字段。
SSL verify peer（*SSL* 验证对端）	勾选复选框以验证 Web 服务器的 SSL 证书。 服务器证书将自动从系统范围的证书颁发机构 (CA) 位置获取。您可以使用 Zabbix server 或 proxy 配置参数SSLCALocation 覆盖 CA 文件的位置。 这将设置 CURLOPT_SSL_VERIFYPEER cURL 选项。
SSL verify host（*SSL* 验证主机）	标记复选框以验证 Web 服务器证书的 *Common Name* 字段或 *Subject Alternate Name* 字段是否匹配。 这将设置 CURLOPT_SSL_VERIFYHOST cURL 选项。
SSL certificate file（*SSL* 证书文件）	用于客户端身份验证的 SSL 证书文件的名称。证书文件必须为 PEM[1] 格式。如果证书文件还包含私钥，请将 *SSL key file*（*SSL* 秘钥文件）字段留空。如果密钥已加密，请在 *SSL key password*（*SSL* 秘钥密码）字段中指定密码。包含此文件的目录由 Zabbix server 或 proxy 配置参数SSLCertLocation指定。 支持 `HOST.*` 宏和用户宏。 这将设置 CURLOPT_SSLCERT cURL 选项。
SSL key file（*SSL* 秘钥文件）	用于客户端身份验证的 SSL 私钥文件的名称。私钥文件必须是 PEM[1] 格式。包含此文件的目录由 Zabbix server 或 proxy 配置参数SSLKeyLocation 指定。 支持 `HOST.*` 宏和用户宏。 这将设置 CURLOPT_SSLKEY cURL 选项。
SSL key password（*SSL* 秘钥密码）	SSL 私钥文件密码。 支持用户宏。 这将设置 CURLOPT_KEYPASSWD cURL 选项。

Attention:
[1] Zabbix 仅支持 PEM 格式的证书和私钥文件。如果你有 PKCS #12 格式文件（通常带有扩展名 *.p12 或 *.pfx）的证书和私钥数据，可以使用以下命令从中生成 PEM 文件：

```
openssl pkcs12 -in ssl-cert.p12 -clcerts -nokeys -out ssl-cert.pem
openssl pkcs12 -in ssl-cert.p12 -nocerts -nodes · -out ssl-cert.key
```

Note:
Zabbix server 无需重启即可获取证书中的更改。

Note:
如果你在单个文件中有客户端证书和私钥，只需在”SSL certificate file（SSL 证书文件）” 字段中指定它，并将”SSL key file（SSL 秘钥文件）” 字段留空。证书和密钥必须仍为 PEM 格式。组合证书和密钥很容易：

```
cat client.crt client.key > client.pem
```

展示　要查看为主机配置的 Web 场景，请转到 *Monitoring*（监控）→ *Hosts*（主机），在列表中找到主机并单击最后一列中的 *Web* 超链接。单击方案名称以获取详细信息。



Web 场景的概述也可以通过 Web 监控小部件显示在 *Monitoring*（监控）→ *Dashboard*（仪表盘）中。

Web 场景执行的最新结果可在 *Monitoring*（监控）→ *Latest data*（最新数据）中找到。

扩展监控　记录收到的 HTML 页面内容是有必要的，尤其是当某些 Web 场景步骤失败。调试级别 5（跟踪）用于此目的。此级别可以在server 和proxy 配置文件中设置，也可以使用运行时控制选项（`-R log_level_increase="http poller,N"`，其中 N 是进程号）。以下示例演示了如何在已设置调试级别 4 的情况下启动扩展监控：

```
Increase log level of all http pollers:
shell> zabbix_server -R log_level_increase="http poller"

Increase log level of second http poller:
shell> zabbix_server -R log_level_increase="http poller,2"
```

如果不需要扩展 Web 监控，可以使用`-R log_level_decrease` 选项。

1 Web 监控项

概述

创建 Web 场景时会自动添加一些新的监控项以进行监控。

所有监控项都从 Web 场景继承标签。

场景监控项

一旦创建了一个场景，Zabbix 就会自动添加以下监控项。

监控项	描述
Download speed for scenario <Scenario>	该监控项将收集有关整个场景的下载速度（每秒字节数）的信息，即所有步骤的平均值。 监控项键值：web.test.in[Scenario,,bps] 类型：*Numeric(float)*
Failed step of scenario <Scenario>	该监控项将显示场景中失败的步骤数。如果所有步骤都成功执行，则返回 0。 监控项键值：web.test.fail[Scenario] 类型：*Numeric(unsigned)*
Last error message of scenario <Scenario>	该监控项返回场景的最后一条错误消息文本。仅当场景具有失败的步骤时才存储新值。如果所有步骤都正常，则不会收集新值。 监控项键值：web.test.error[Scenario] 类型：*Character*

将使用实际场景名称而不是”Scenario（场景）”。

Note:
Web 监控项添加了 30 天的历史记录和 90 天的趋势数据保留时长。

Note:
如果场景名称以双引号开头或包含逗号或方括号，它将在监控项键值中被正确引用。在其他情况下，将不执行额外的引用。

这些监控项可用于创建触发器和定义通知条件。

示例 1

要创建”Web scenario failed（Web 场景失败）” 触发器，您可以定义触发器表达式：

```
last(/host/web.test.fail[Scenario])<>0
```

确保将’Scenario’ 替换为场景的真实名称。

示例 2

要在触发器名称中创建一个带有有用问题描述的”Web scenario failed（Web 场景失败）” 触发器，您可以使用名称定义触发器：

```
Web scenario "Scenario" failed: {ITEM.VALUE}
```

和触发表达式：

```
length(last(/host/web.test.error[Scenario]))>0 and last(/host/web.test.fail[Scenario])>0
```

确保将’Scenario’ 替换为场景的真实名称。

示例 3

要创建”Web application is slow（Web 应用程序慢）” 的触发器，可以定义以下触发器表达式：

```
last(/host/web.test.in[Scenario,,bps])<10000
```

确保将’Scenario’ 替换为场景的真实名称。

场景步骤监控项

一旦创建了一个步骤，Zabbix 就会自动添加以下监控项。

监控项	描述
Download speed for step <Step> of scenario <Scenario>	该监控项收集有关该步骤的下载速度（每秒字节数）的信息。 监控项键值：web.test.in[Scenario,Step,bps] 类型：*Numeric(float)*
Response time for step <Step> of scenario <Scenario>	该监控项收集有关步骤响应时间的信息（以秒为单位）。响应时间是从请求开始到所有信息传输完毕的时间。 监控项键值：web.test.time[Scenario,Step,resp] 类型：*Numeric(float)*

监控项	描述
Response code for step <Step> of scenario <Scenario>	该监控项收集步骤的响应代码。 监控项键值：web.test.rspcode[Scenario,Step] 类型：*Numeric(unsigned)*

将分别使用实际场景和步骤名称代替”Scenario” 和”Step”。

Note:
Web 监控项添加了 30 天的历史记录和 90 天的趋势数据保留时长。

Note:
如果场景名称以双引号开头或包含逗号或方括号，它将在监控项键值中正确引用。在其他情况下，将不执行额外的引用。

这些监控项可用于创建触发器和定义通知条件。例如，要创建一个”Zabbix GUI login is too slow” 的触发器，可以定义一个触发器表达式：

```
last(/zabbix/web.test.time[ZABBIX GUI,Login,resp])>3
```

2 真实场景

概述

本节介绍使用 Web 监控的分步真实示例。

让我们使用 Zabbix web 监控来监控 Zabbix 的 web 界面。我们想知道它是否可用、是否提供正确的内容以及它的运行速度。为此，我们必须使用我们的用户名和密码登录。

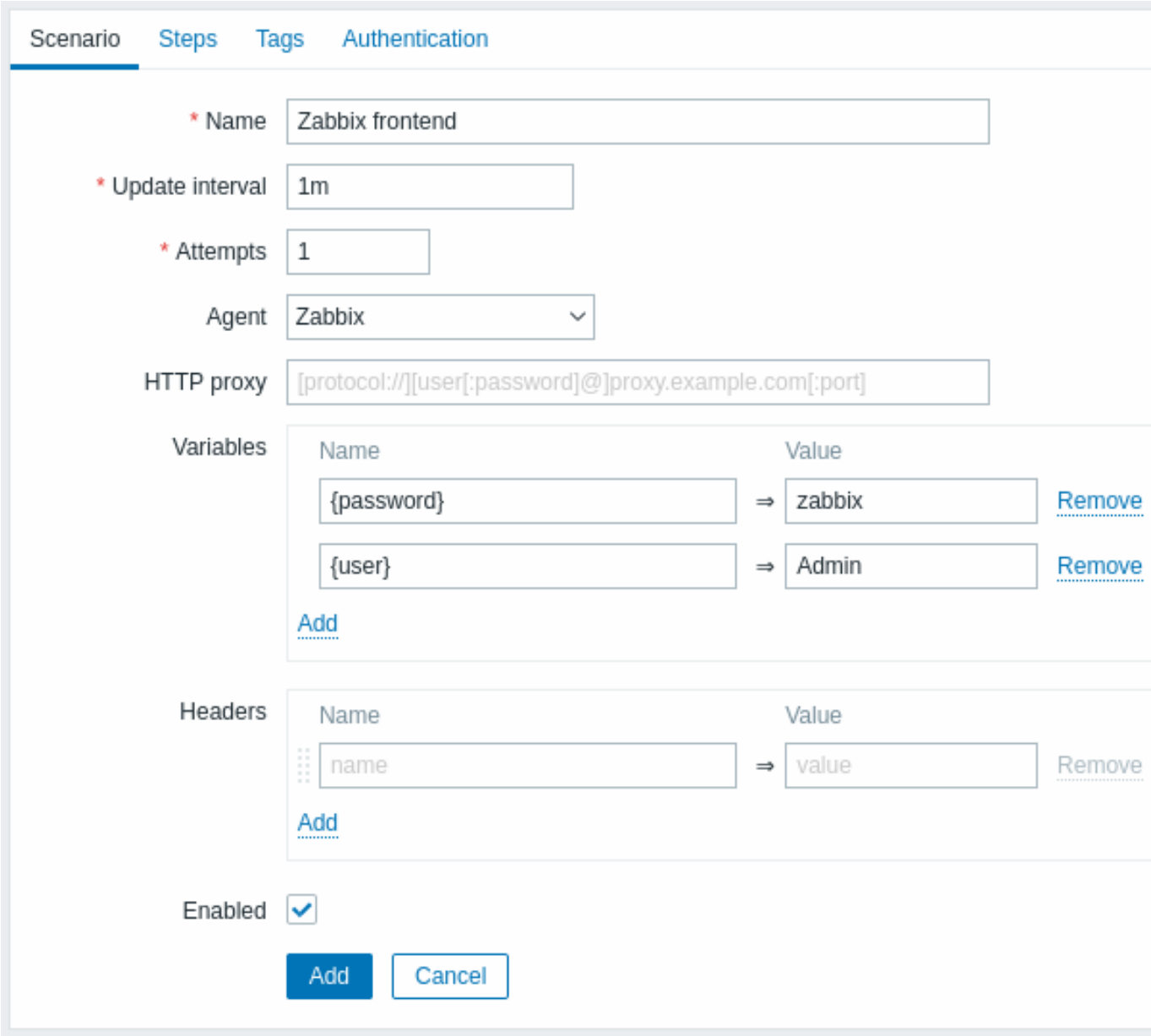
场景

第 1 步

添加一个新的 Web 场景。

我们将添加一个场景来监控 Zabbix 的 Web 界面。该场景将执行多个步骤。

点击 *Configuration*（配置）→ *Hosts*（主机），选择一个主机并单击该主机行中的 *Web* 。然后单击 *Create web scenario*（创建 *web* 场景）。



所有必填字段都标有红色星号。

在新的场景表单中，我们将场景命名为 *Zabbix frontend*。我们还将创建两个变量：{user} 和 {password}。

可能还想在标签选项卡中添加一个新的 *Application:Zabbix frontend* 标签。

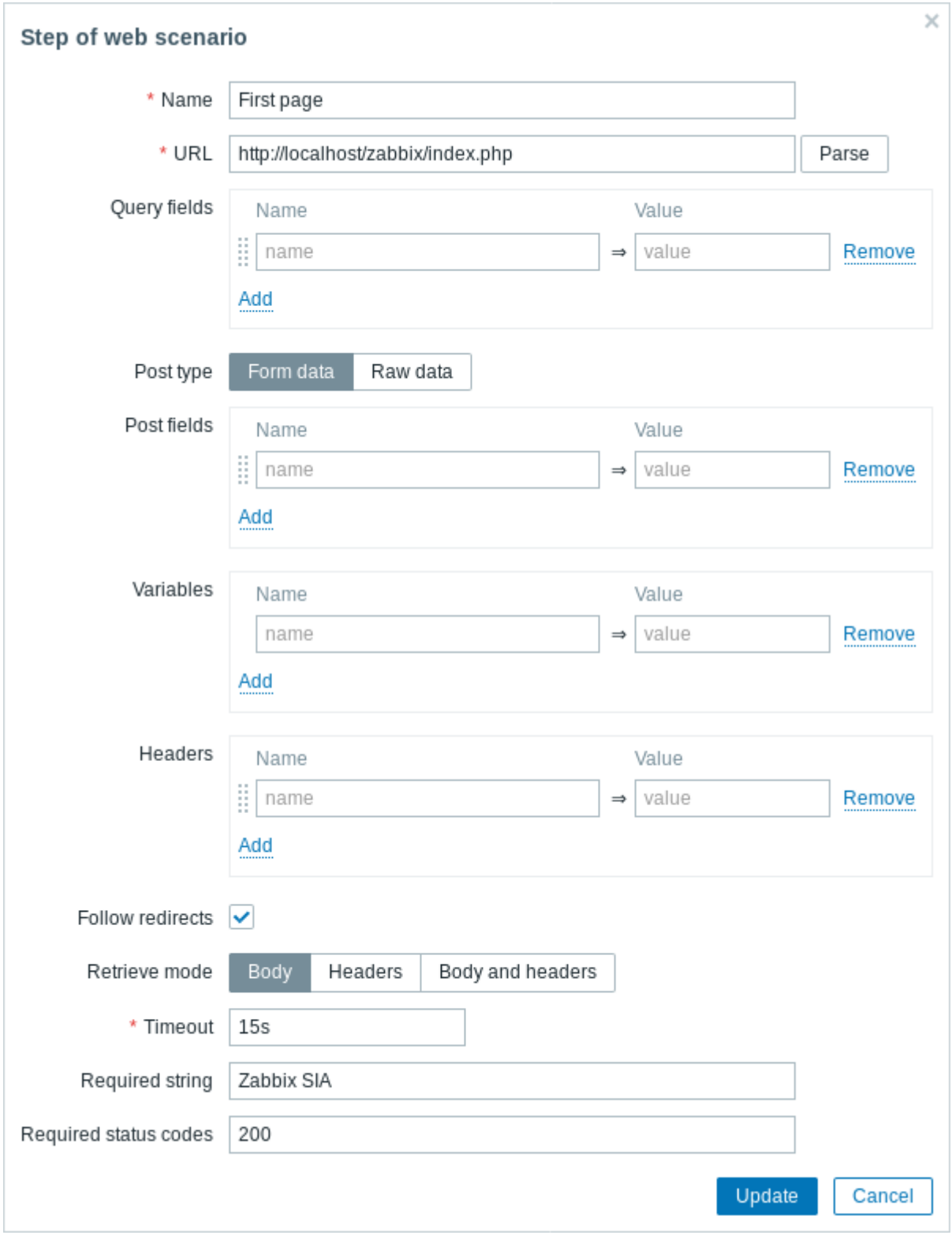
第 2 步

定义场景的步骤。

单击 *Steps*（步骤）中的 *Add*（添加）按钮以添加各个步骤。

Web scenario step 1（*Web* 场景步骤 *1*）

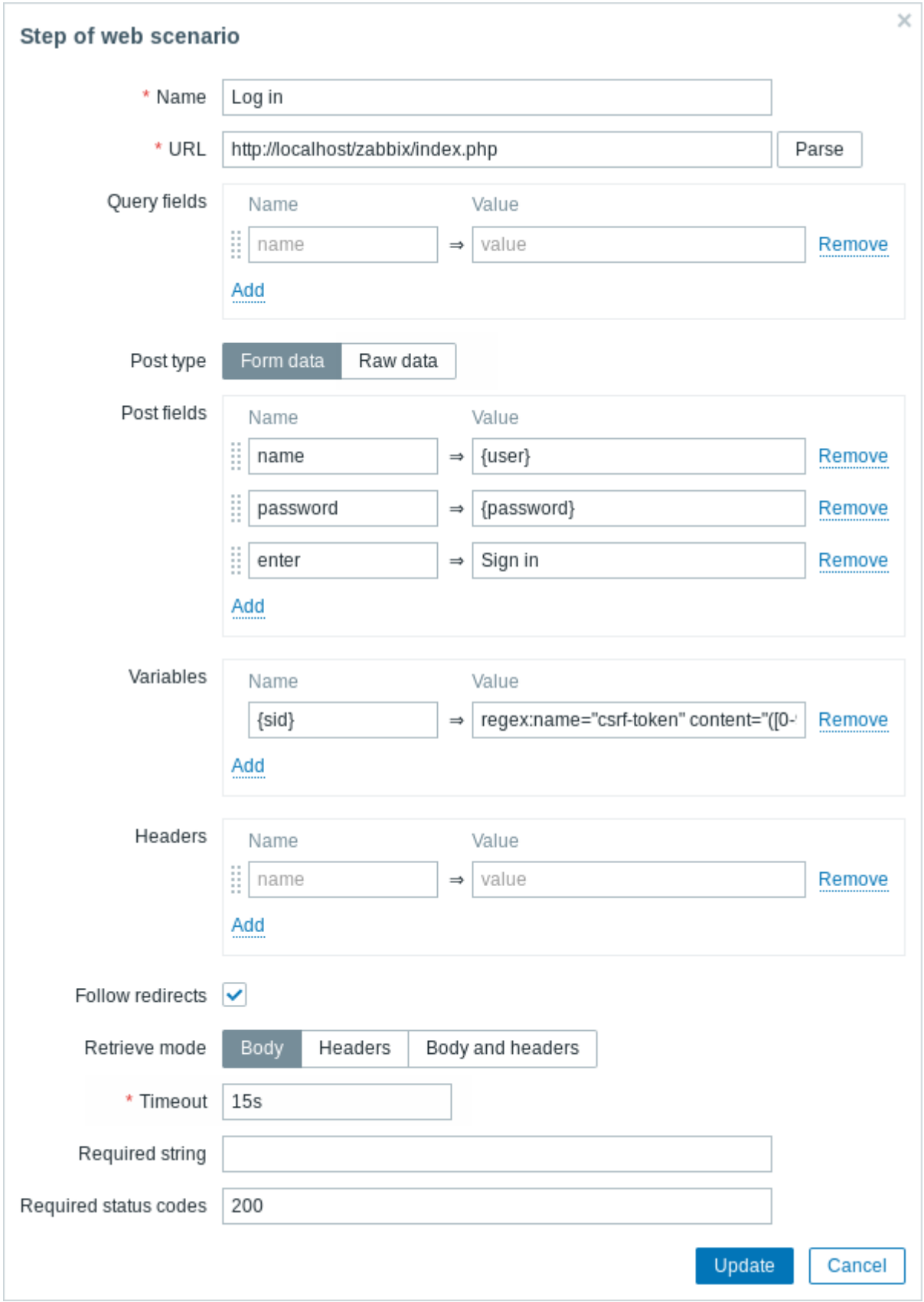
我们首先检查第一页是否正确响应，返回 HTTP 响应代码 200 并包含文本”Zabbix SIA”。



完成配置步骤后，单击 *Add*（添加）。

Web scenario step 2（*Web* 场景步骤 *2*）

我们继续登录 Zabbix 前端，并通过重用我们在场景级别定义的宏（变量）- {user} 和 {password}。

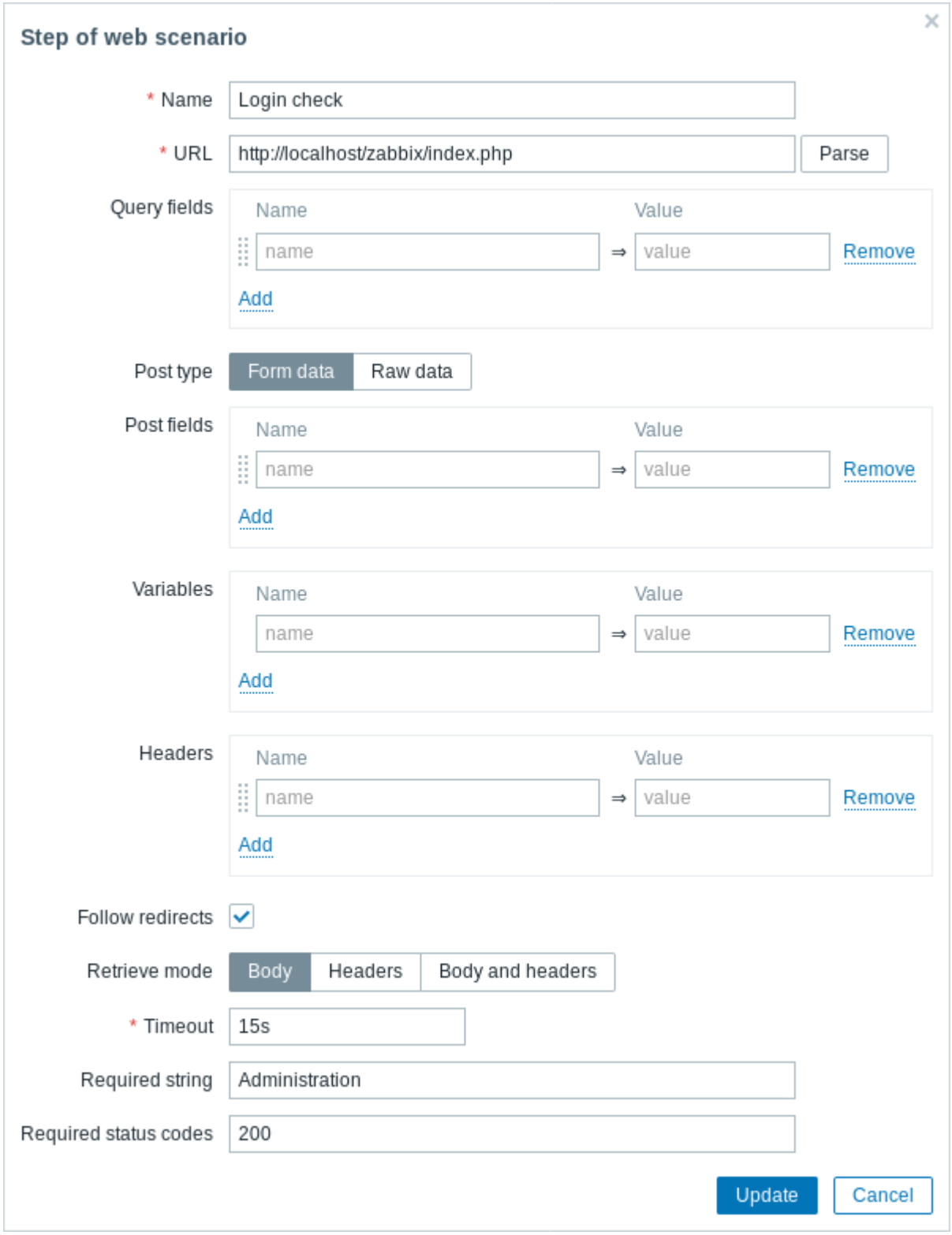


Attention:
注意 Zabbix 前端在登录时使用 JavaScript 重定向，因此首先我们必须登录，并且只有在进一步的步骤中，我们才能检查已登录的功能。此外，登录步骤必须使用 **index.php** 文件的完整 URL。

注意我们是如何使用正则表达式的变量语法'regex:name="csrf-token" content="([0-9a-z]{16})" ' 获取' {sid} ' 变量 (会话 ID) 的内容的。这个变量在步骤 4 中是必需的。

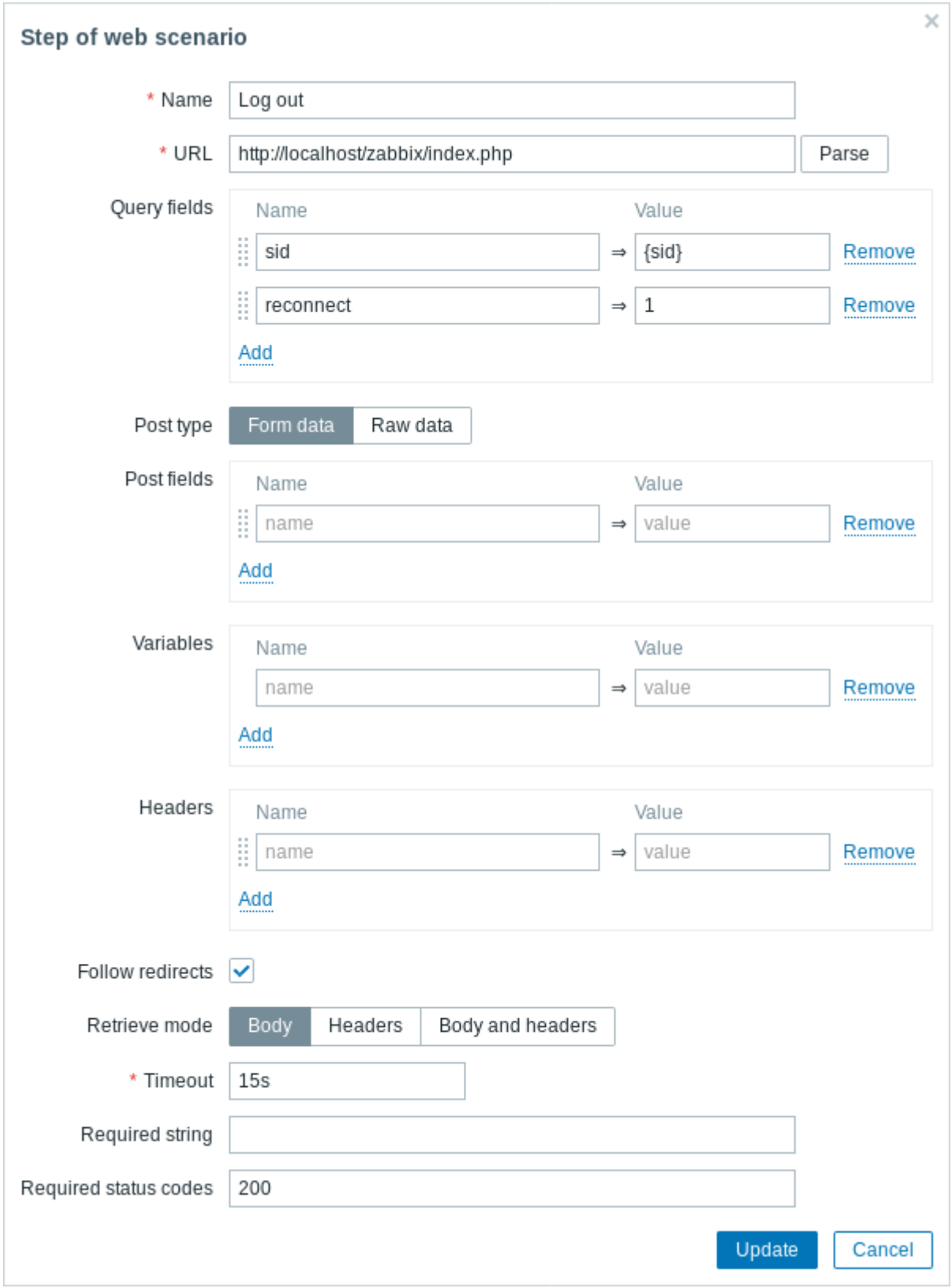
Web 场景步骤 *3*

登录后，我们应该验证。为此，我们检查仅在登录时可见的字符串，例如 **Administration**。



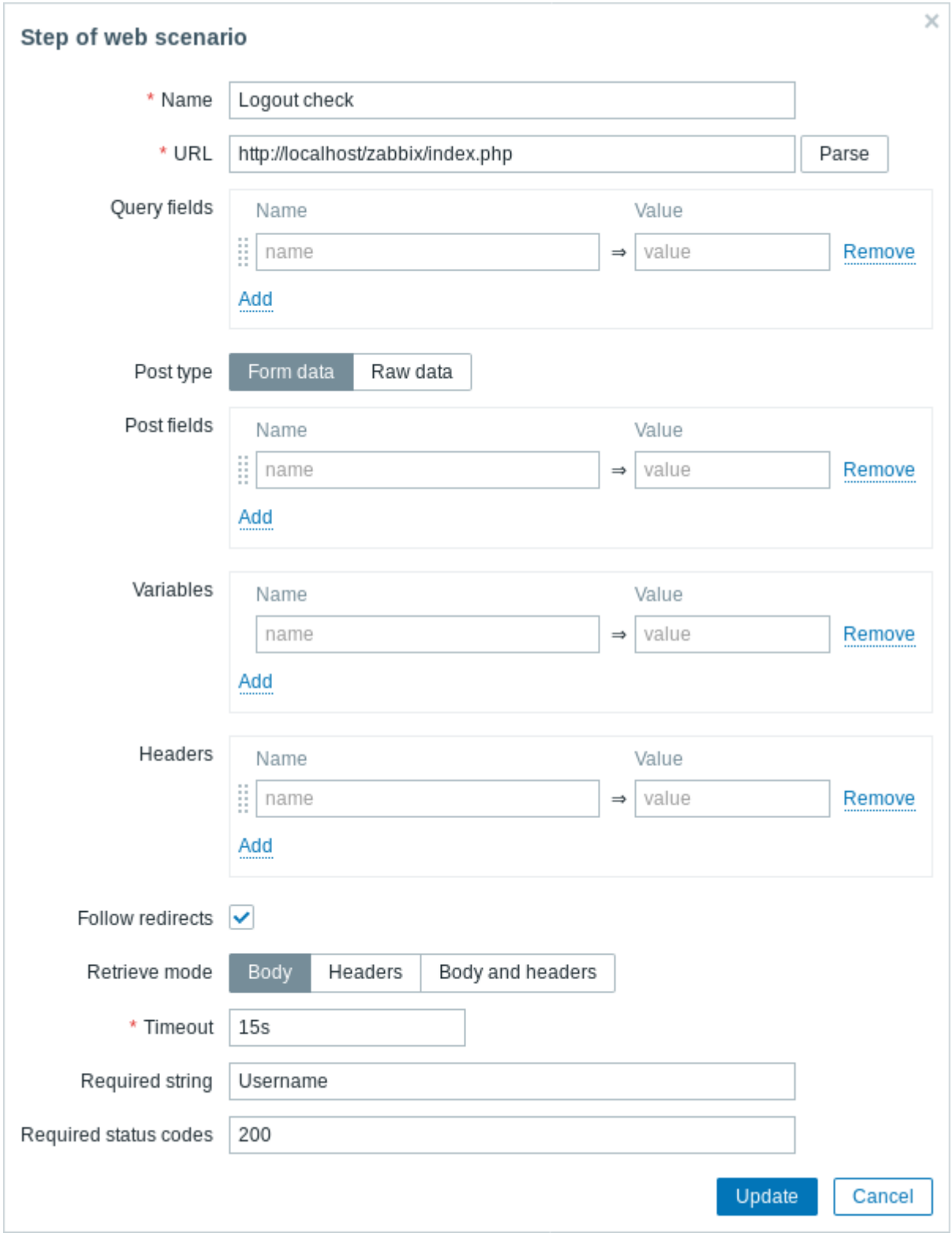
Web 场景步骤 *4*

已经验证了前端是可访问的并且可以登录并检索登录的内容，之后应该注销 - 否则 Zabbix 数据库将被大量打开的会话记录污染。



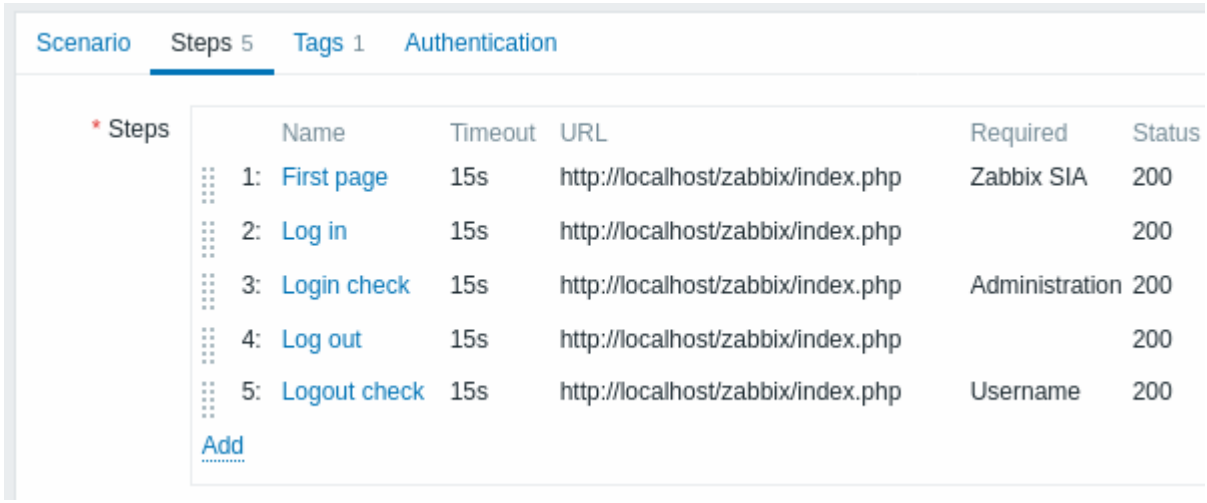
Web 场景步骤 *5*

可以通过查找 **Username**（用户名）字符串来确认是否已注销。



步骤的完整配置

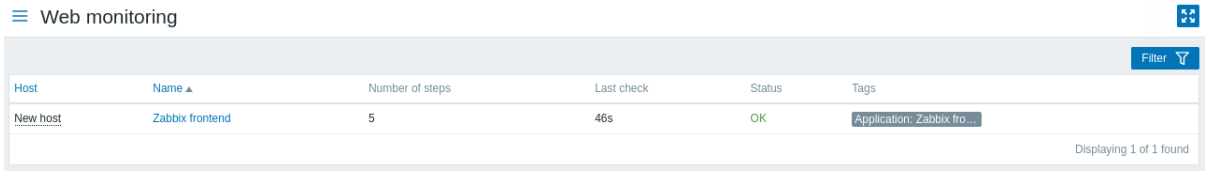
一个完整的 Web 场景步骤的配置如下所示：



步骤 3

保存完成的 Web 监控场景。

该场景将被添加到主机。要查看 Web 场景信息，请转到 *Monitoring*（监控中）→ *Hosts*（主机），在列表中找到主机，然后单击最后一列中的 Web 超链接。



点击场景名称可查看更详细的统计信息：



10. 虚拟机监控

概述　Zabbix 从 2.2.0 版开始支持对 VMware 环境的监控。

Zabbix 可以使用低级别发现规则自动发现 VMware 宿主机（即 VMware hypervisors）和虚拟机，并根据预定义的主机原型创建主机来监控它们。

Zabbix 中的默认数据集提供了几个现成的模板，用于监控 VMware vCenter 或 ESX 宿主机。

所需的最低 VMware vCenter 或 vSphere 版本为 5.1。

细节　虚拟机监控分两步完成。首先，虚拟机数据由 Zabbix 进程 *vmware collector* 收集。这些进程通过 SOAP 协议从 VMware Web 服务获取必要的信息，对其进行预处理并存储到 Zabbix 服务器共享内存中。然后，轮询器使用 Zabbix 简单检查VMware keys 检索此数据。

从 Zabbix 2.4.4 版本开始，收集的数据分为 2 种类型：VMware 配置数据和 VMware 性能计数器数据。这两种类型都由 *vmware collectors* 独立收集。因此，建议启用比受监视的 VMware 服务更多的收集器。否则，VMware 性能计数器统计信息的检索可能会因检索 VMware 配置数据而延迟（大型安装需要一段时间）。

目前只有数据存储、网络接口和磁盘设备统计信息以及自定义性能计数器项是基于 VMware 性能计数器信息的。

配置　为了使虚拟机监控可以正常工作，Zabbix 应该使用 --with-libxml2 和 --with-libcurl 编译选项进行编译 。

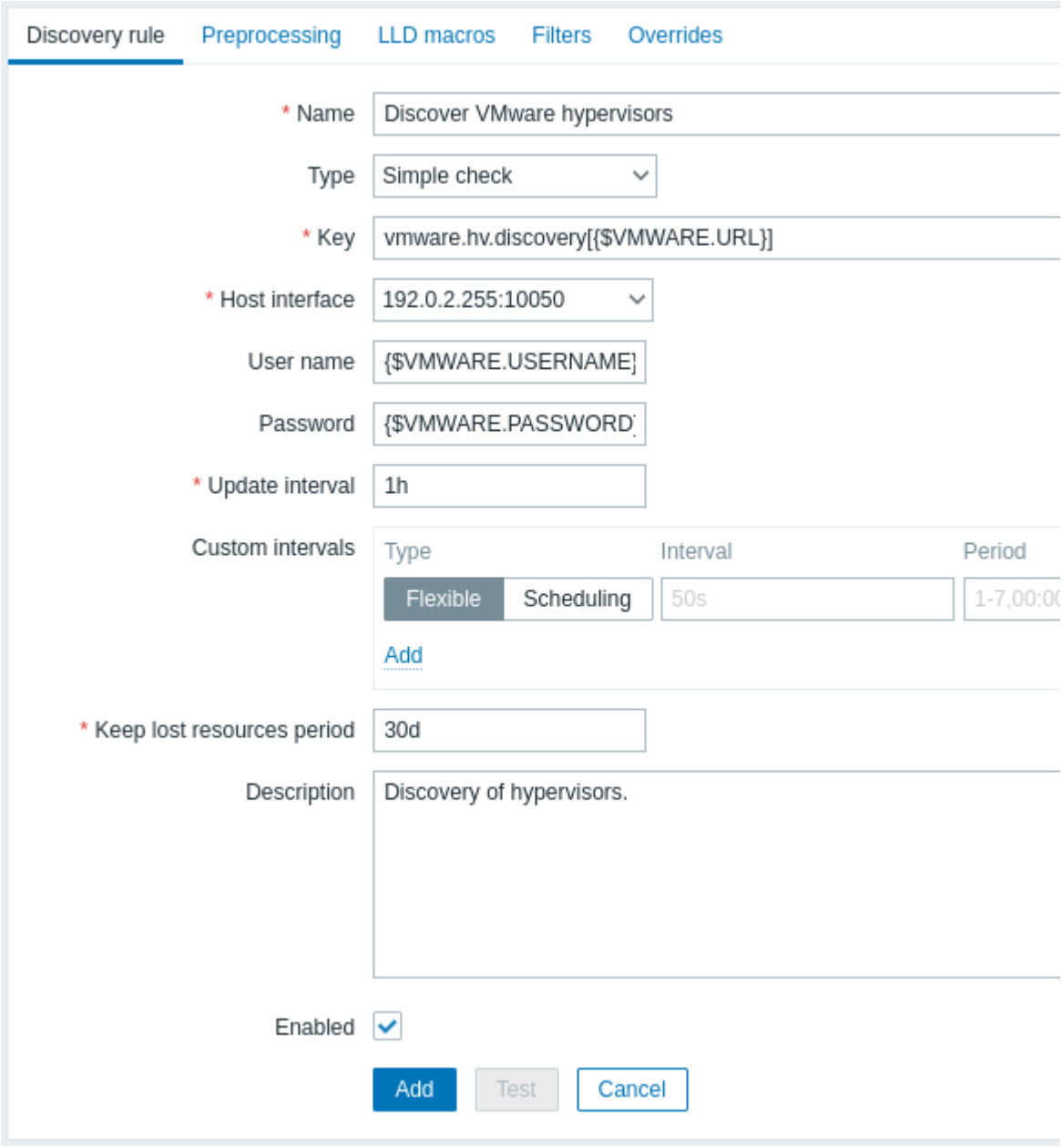
以下配置文件选项可用于调整虚拟机监控：

- **StartVMwareCollectors** - 预设的 vmware collector 实例的数量。
  此值取决于您要监控的 VMware 服务的数量。在大多数情况下，这应该是：
  *servicenum < StartVMwareCollectors < (servicenum * 2)*
  其中 *servicenum* 是 VMware 服务的数量。例如，如果有 1 个要监控的 VMware 服务，请将 StartVMwareCollectors 设置为 2，如果您有 3 个 VMware 服务，请将其设置为 5。请注意，在大多数情况下，此值不应小于 2，也不应大于监控的 VMware 服务数量的 2 倍。另外，此值还取决于您的 VMware 环境大小以及 *VMwareFrequency* 和 *VMwarePerfFrequency* 配置参数（见下文）。
- **VMwareCacheSize**
- **VMwareFrequency**
- **VMwarePerfFrequency**
- **VMwareTimeout**

有关更多详细信息，请参阅 Zabbix server 和proxy 的配置文件页面。

> **Attention:**
> 为了支持数据存储容量指标，Zabbix 要求 VMware 配置 vpxd.stats.maxQueryMetrics 参数的值至少为 64。另请参阅 VMware 知识库 文章。

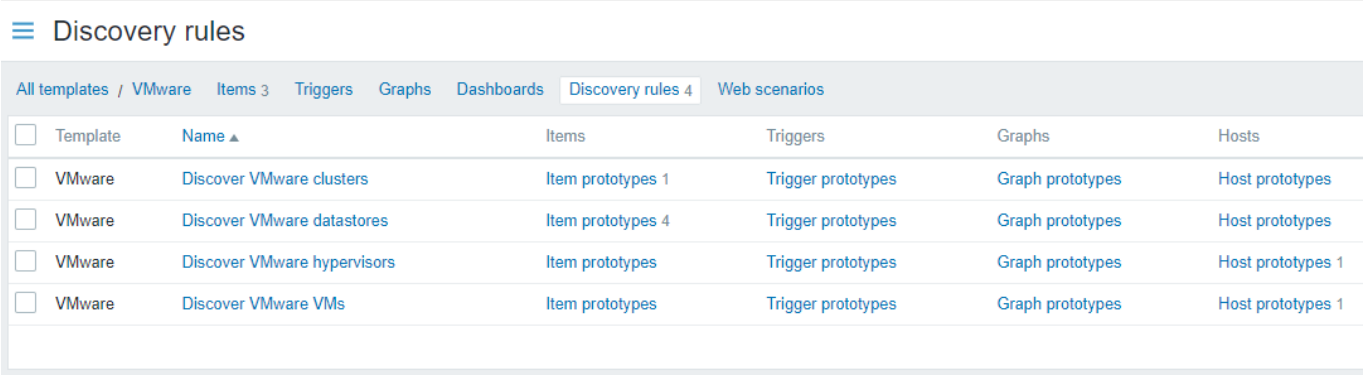
自动发现规则　Zabbix 可以使用低级别发现规则来自动发现 VMware 宿主机和虚拟机。



所有必填字段都标有红色星号。

上面屏幕截图中的发现规则键值是 *vmware.hv.discovery[{$URL}]*。

主机原型　可以使用低级别发现规则创建主机原型。当发现虚拟机时，这些原型就变成了真正的主机。原型在被发现之前不能有自己的监控项和触发器，除了来自所链接的模板的那些。发现的主机将属于 existing 主机。



为了使从主机原型创建的主机具有唯一的主机名，主机名（*Host name*）字段必须包含至少一个低级别发现宏。

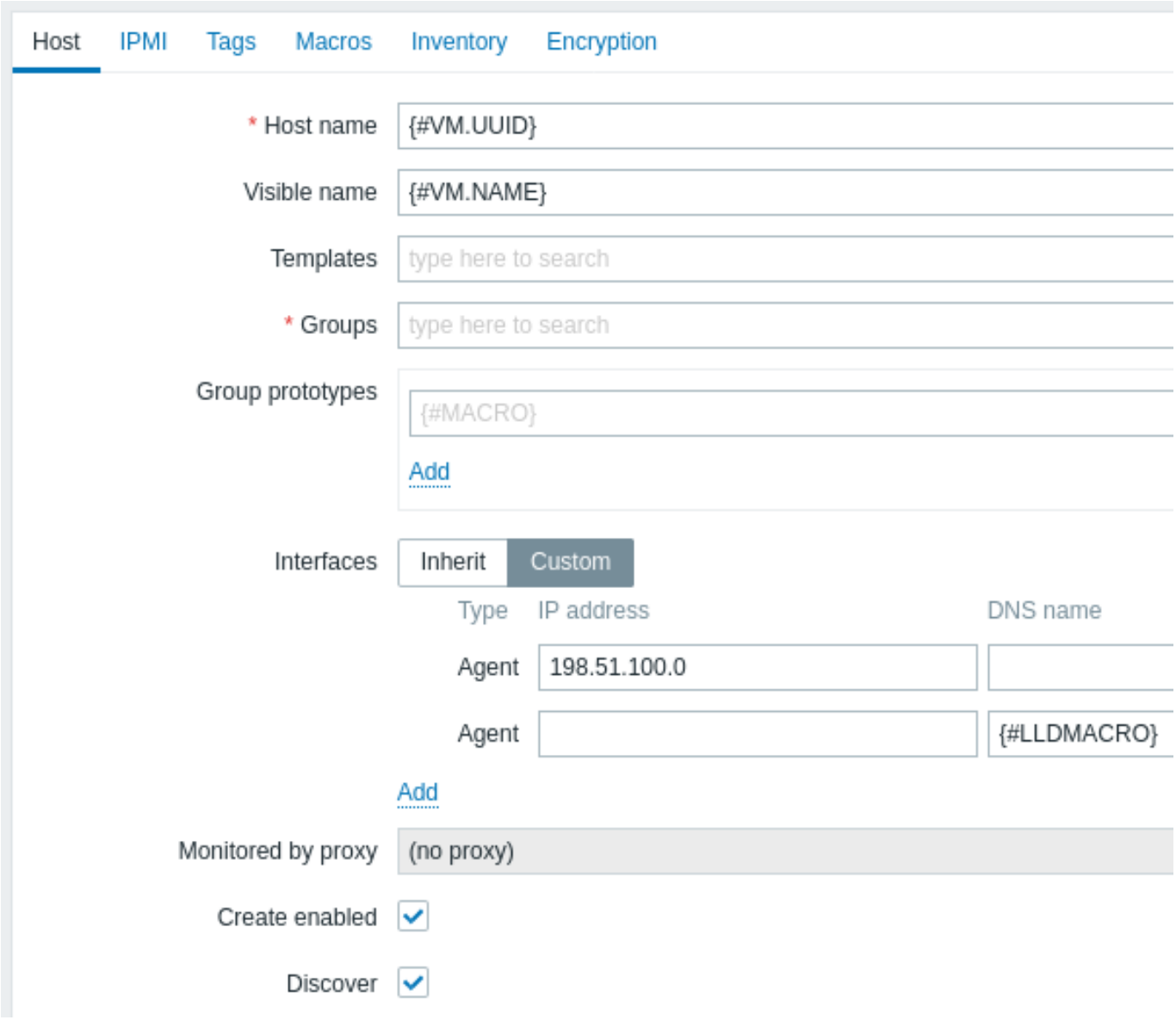
从 Zabbix 5.2 开始，发现的主机可以配置自定义接口或继承发现规则所属主机的 IP（默认）。要添加一个或多个自定义接口，请将 接口（*Interface*）选择器从 继承（*Inherit*）模式切换到 自定义（*Custom*）模式，然后按下Add 并从出现的下拉菜单中选择所需的接口类型。

可以为主机原型定义所有支持的接口类型：Zabbix agent、SNMP、JMX、IPMI。接口字段支持低级别发现宏和用户宏。如果指定了多个自定义接口 - 使用 默认（*Default*）列指定主接口。

注意：

- 如果选择 自定义但未指定接口，则将创建没有接口的主机。
- 如果为属于模板的主机原型选择了 继承，则发现的主机将继承模板链接到的主机的接口。

Warning:
如果主机接口包含不正确的数据，则不会创建主机。



LLD 宏还可用于可见名称、主机组原型字段、标签值或主机原型用户宏的值。

可以为主机原型指定的其他选项包括：

- 与现有主机组的链接
- 模板链接
- 加密

如果选中 *Create enabled* ，则添加的主机将处于启用状态。如果未选中，仍会添加主机，但处于禁用状态。

如果选中 *Discover* （默认），则将创建主机。如果未选中，则不会创建主机，除非在发现规则 中覆盖此设置。此功能在创建发现规则时提供了额外的灵活性。

发现的主机在主机列表中以创建它们的发现规则的名称为前缀。发现的主机可以手动删除，也可以根据 发现规则的保留丢失资源的天数（*Keep lost resources period (in days)*）被自动删除。大多数配置选项都是只读的，除了启用/禁用主机和主机清单。发现的主机不能拥有自己的主机原型。

开箱即用的模板　Zabbix 官方默认提供了几个开箱即用的模板，用于监控 VMware vCenter 或直接 ESX hypervisor。这些模板包含预配置的 LLD 规则以及一些用于监控虚拟化安装的内置检查。

VMware vCenter 和 ESX hypervisor 监控模板：

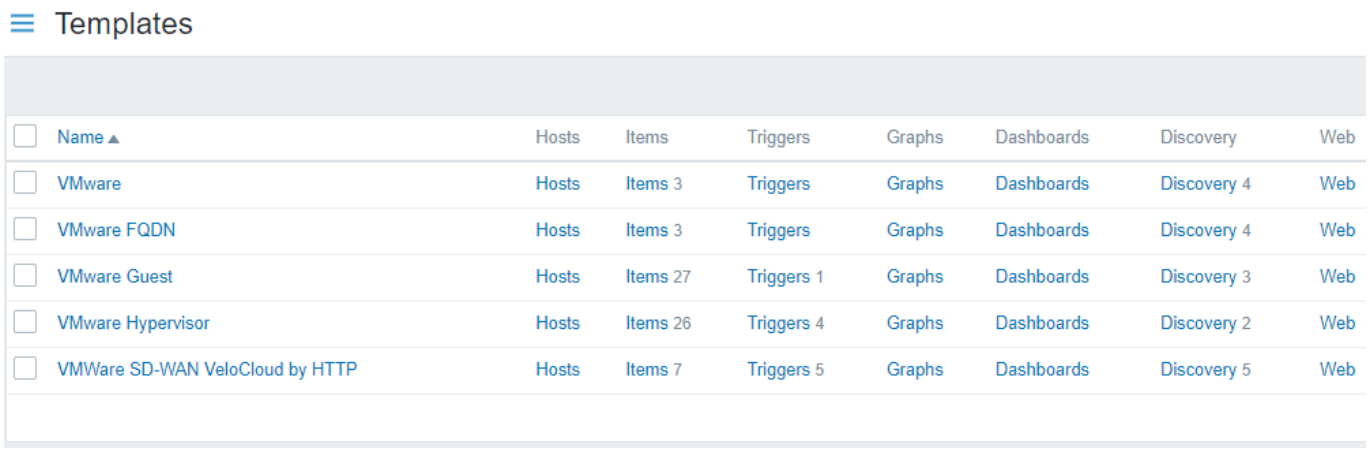
- *VMware* - 为相应的宏使用 UUID 数据；
- *VMware FQDN* - 将 FQDN 数据用于相应的宏。

Note:
为了使 *VMware FQDN* 模板正常工作，每个受监控的 VM 都应具有符合 FQDN 规则的唯一操作系统名称，并且必须在每台计算机上安装 VMware Tools。如果满足以上条件，建议使用 *VMware FQDN* 模板。在 Zabbix 5.2 中引入了使用自定义接口创建主机的功能后，*VMware FQDN* 模板的创建成为可能。

如果无法满足 FQDN 要求，*VMware* 模板仍然可以使用。*VMware* 模板有个已知问题：使用保存在 vCenter 中的名称（例如，VM1、VM2 等）创建已发现虚拟机的主机。如果稍后在这些主机上安装 Zabbix agent 并启用自动注册，由于不存在相应的主机，则自动注册进程将读取主机名作为他们启动时的注册名（例如，vm1.example.com、vm2.example.com 等）并创建一个新的主机。因此，每台机器都会有两个具有不同名称的重复主机。

发现虚拟主机使用的模板（通常，这些模板不应该被手动链接到其他主机）：

- *VMware Hypervisor*；
- *VMware Guest*。



主机配置　要使用 VMware 简单检查，主机必须定义以下用户宏：

- **{$VMWARE.URL}** - VMware 服务 (vCenter 或 ESX hypervisor) SDK
- **{$VMWARE.USERNAME}** - VMware 服务用户名
- **{$VMWARE.PASSWORD}** - VMware 服务 {$VMWARE.USERNAME} 用户密码

示例　以下示例演示了如何在 Zabbix 上快速设置 VMware 监控：

- 使用所需选项（--with-libxml2 和 --with-libcurl）编译 zabbix server
- 将 Zabbix server 配置文件中的 StartVMwareCollectors 选项设置为 1 或更多
- 创建一个新主机
- 设置 VMware 身份验证所需的主机宏：

```
    {{..:..:assets:en:manual:vm_monitoring:vm_host_macros.png|}}
* Link the host to the VMware service template:
    {{..:..:assets:en:manual:vm_monitoring:vm_host_templates.png|}}
* Click on the //Add// button to save the host
```

扩展日志记录　可以使用 debug 5 记录 VMware 收集器收集的数据以进行详细调试。可以在server 和proxy 配置文件中设置此级别，或者使用运行时控制选项（`-R log_level_increase="vmware collector,N"`，其中 N 是进程号）。以下示例演示了如何在已设置调试级别 4 的情况下启动扩展日志记录：

```
Increase log level of all vmware collectors:
shell> zabbix_server -R log_level_increase="vmware collector"

Increase log level of second vmware collector:
shell> zabbix_server -R log_level_increase="vmware collector,2"
```

如果不需要对 VMware 收集器数据进行扩展日志记录，则可以使用该`-R log_level_decrease` 选项将其停止。

故障排除

- 如果指标不可用，请确保在当前的 VMware vSphere 版本中它们是否不可用或默认关闭，或者是否未对性能指标数据库查询设置一些限制。更多详细信息，请参见 ZBX-12094 。
- 如果出现 *'config.vpxd.stats.maxQueryMetrics' is invalid or exceeds the maximum number of characters permitted*** 的报错，请向 vCenter Server 配置中添加参数 `config.vpxd.stats.maxQueryMetrics` 。此参数的值应与 VMware 的 *web.xml* 中 `maxQuerysize` 的值相同。有关详细信息，请参阅此 VMware 知识库 文章。

1 虚拟机自动发现相关键值字段

下表列出了虚拟机相关自动发现键值返回的字段。

监控项键值		
描述	字段	检索到的内容
vmware.cluster.discovery 发现集群。	{#CLUSTER.ID}	集群标识符。
	{#CLUSTER.NAME}	集群名称。
vmware.datastore.discovery 发现数据存储。	{#DATASTORE}	数据存储名称。
	{#DATASTORE.EXTENT}	具有 {#INSTANCE-NAME:PARTITIONID} 数组的 JSON 对象。
vmware.dc.discovery 发现数据中心。	{#DATACENTER}	数据中心名称。
	{#DATACENTER.ID}	数据中心 ID。
vmware.hv.discovery 发现宿主机。	{#HV.UUID}	唯一的宿主机标识符。
	{#HV.ID}	宿主机标识符 (HostSystem 受管对象名称)。
	{#HV.NAME}	宿主机名称。
	{#HV.NETNAME}	宿主机网络主机名。

监控项键值		
	{#HV.IP}	宿主机 IP 地址，可能为空。在具有多个网络接口的 HA 配置的情况下，观察到接口的以下选择优先级： - 首选与 vCenter IP 共享的子网 IP - 首选默认网关的子网 IP - 优先使用 ID 最低的接口的 IP 地址 自 Zabbix 5.2.2 起支持此字段。
	{#CLUSTER.NAME}	集群名称，可能为空。
	{#DATACENTER.NAME}	数据中心名称。
	{#PARENT.NAME}	存储 hypervisor 的容器的名称。 从 Zabbix 4.0.3 开始支持。
	{#PARENT.TYPE}	存储宿主机的容器类型。这些值可以是 Datacenter, Folder, ClusterComputeRes VMware, 其中'VMware'代表未知容器类型。 从 Zabbix 4.0.3 开始支持。
vmware.hv.datastore.discovery		

监控项键值		
发现 hypervisor 数据存储。多个管理程序可以使用同一个数据存储。	{#DATASTORE}	数据存储名称。
	{#MULTIPATH.COUNT}	已定义的数据存储路径数。
	{#MULTIPATH.PARTITION.COUNT}	可用磁盘分区数。
vmware.vm.discovery 发现虚拟机。	{#VM.UUID}	虚拟机唯一标识符。
	{#VM.ID}	虚拟机标识符（Virtual-Machine 托管对象名称）。
	{#VM.NAME}	虚拟机名称。
	{#HV.NAME}	宿主机名称。
	{#DATACENTER.NAME}	数据中心名称。
	{#CLUSTER.NAME}	集群名称，可能为空。
	{#VM.IP}	虚拟机 IP 地址，可能为空。自 Zabbix 5.2.2 起支持。
	{#VM.DNS}	虚拟机 DNS 名称，可能为空。自 Zabbix 5.2.2 起支持。
	{#VM.GUESTFAMILY}	客户虚拟机操作系统名称，可能为空。自 Zabbix 5.2.2 起支持。
	{#VM.GUESTNAME}	已安装的客户虚拟机操作系统名称，可能为空。自 Zabbix 5.2.2 起支持。

监控项键值		
	{#VM.FOLDER}	虚拟机父文件夹链，可作为嵌套组的值；文件夹名称与"/"组合，可能为空。自 Zabbix 5.4.2 起支持。
vmware.vm.net.if.discovery 发现虚拟机网络接口。	{#IFNAME}	网络接口名称。
vmware.vm.vfs.dev.discovery 发现虚拟机磁盘设备。	{#DISKNAME}	磁盘设备名称。
vmware.vm.vfs.fs.discovery 发现虚拟机文件系统。	{#FSNAME}	文件系统名称。

11. 维护期

概述　可以在 Zabbix 中定义主机群组、主机和特定触发器/服务的维护期。

有两种维护类型 - 有数据收集（with data collection）和没有数据收集（with no data collection）。

在"有数据收集（with data collection）"的维护期间，触发器照常处理，并在需要时创建事件。但是，如果在操作配置中选中了 暂停操作以解决被抑制的问题（*Pause operations for suppressed problems*）选项，则会暂停维护中的主机/触发器的问题升级。在这种情况下，只要维护期持续，可能包括发送通知或远程命令的升级步骤将被忽略。请注意，维护期间不会抑制问题恢复和更新操作，只会抑制升级。

例如，如果升级步骤安排在问题开始后的 0、30 和 60 分钟，并且在真正问题出现后的 10 分钟到 40 分钟之间进行半小时的维护，则将在半小时或 60 分钟和 90 分钟（假设问题仍然存在）后执行步骤 2 和 3。同样，如果在维护过程中出现问题，则在维护后开始升级。

要在维护期间正常（无延迟）接收问题通知，您必须取消选中操作配置中的 暂停操作以解决被抑制的问题（*Pause operations for suppressed problems*）选项。

Note:
如果至少有一台主机（在触发器表达式中使用）未处于维护模式，Zabbix 将发送问题通知。

Zabbix server 必须在维护期间运行。定时器进程负责在每分钟 0 秒将主机状态切换到维护状态或从维护状态切换。请注意，当主机进入维护时，Zabbix server 计时器进程将读取所有未解决的问题，以检查是否需要抑制这些问题。如果存在许多未解决的问题，这可能会对性能产生影响。Zabbix server 也会在启动时读取所有未解决的问题，即使当时没有配置维护。

无论维护类型如何（包括"无数据收集"维护），proxy 都将始终收集数据。如果设置了"无数据收集"，则 server 会忽略该数据。

当"无数据"维护结束时，使用 nodata() 函数的触发器在它们检查期间不会在下一次检查之前触发。

如果在主机维护和维护结束时添加日志类型监控项，则只会收集自维护结束以来的新日志文件条目。

如果为处于"无数据收集"维护类型的主机发送带时间戳的值（例如使用Zabbix sender），则该值将被丢弃，但是可以为过期的维护期发送带时间戳的值。

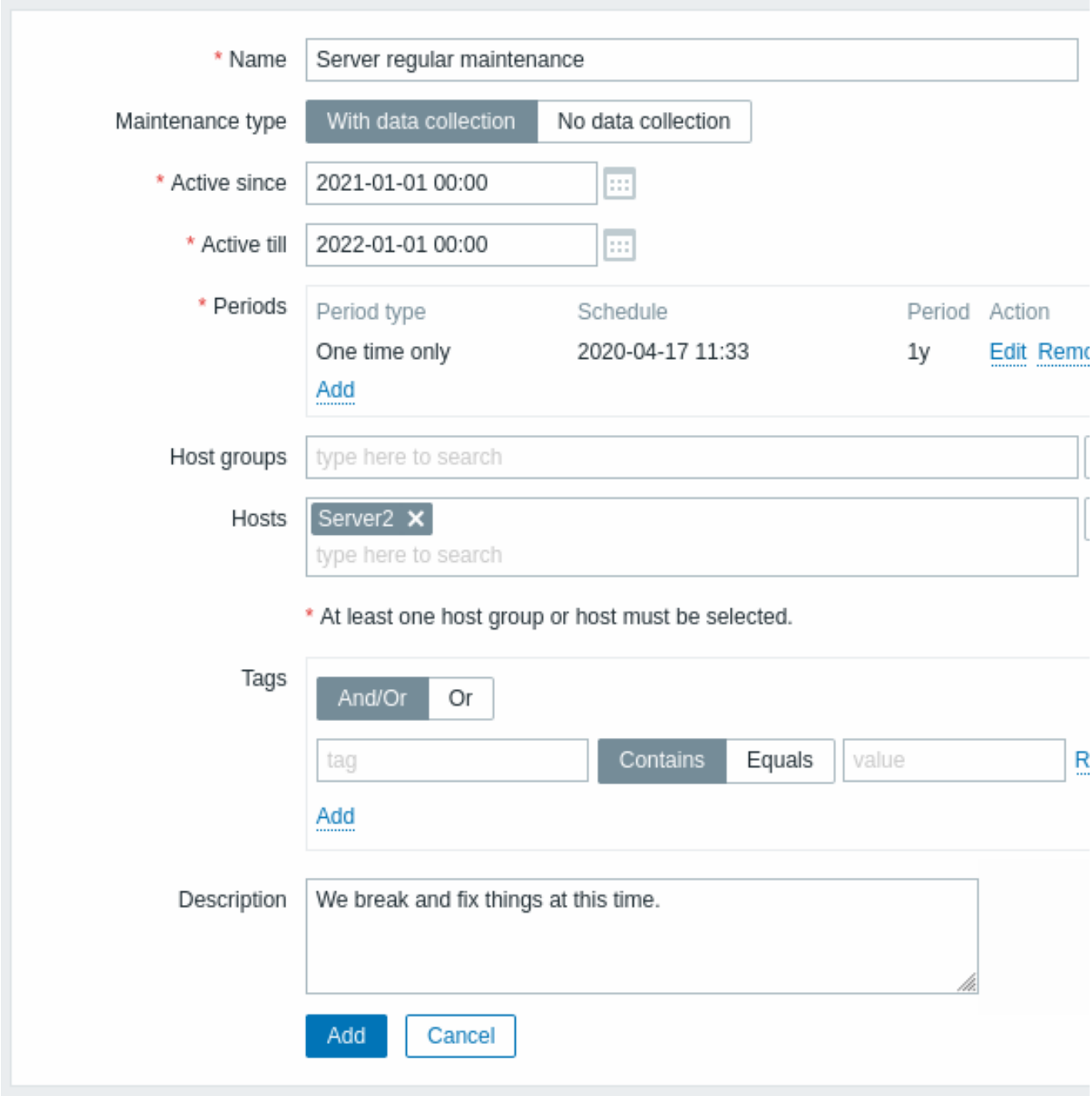
Attention:
为了确保重复维护周期（每天、每周、每月）的可预测性，Zabbix 的所有组件都需要使用同一个公共时区。

如果维护周期、主机、主机群组或标签由用户更改，更改将在配置缓存同步后生效。

Attention:
创建维护期时，使用创建它的用户的时区。但是，当定期维护期（*Daily*、*Weekly*、*Monthly*）被安排时，使用 Zabbix 服务器的时区。为了确保重复维护周期的可预测行为，需要为 Zabbix 的所有部分使用一个公共时区。

配置　配置维护期有以下操作步骤：

- 转到：配置（*Configuration*）→ 维护（*Maintenance*）
- 单击创建维护期（或现有维护期的名称）
- 在表格中输入维护参数



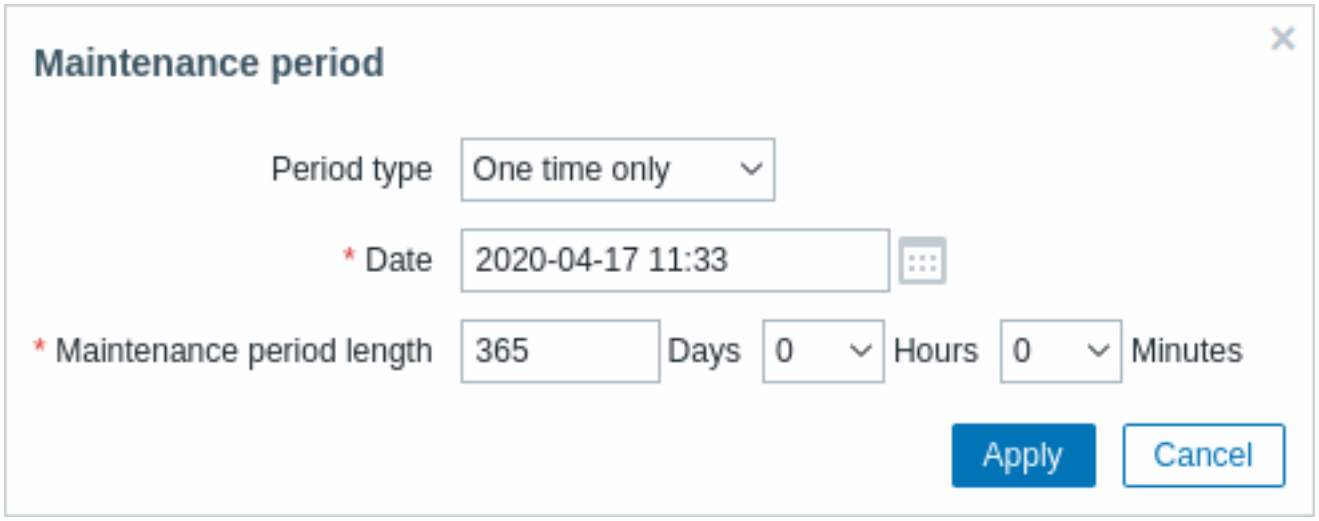
所有必填字段都标有红色星号。

参数	描述
名字（*Name*）	维护期的名称。
维护类型（*Maintenance type*）	可以设置两种维护方式： 有数据采集（**With data collection**） - 维护时服务器采集数据，处理触发器 无数据采集（**No data collection**） - 维护时服务器不采集数据

参数	描述
启用自从（*Active since*）	执行维护期的日期和时间变为活动状态。 注意：单独设置此时间不会激活维护期；为此，请转到期间（*Periods*）选项卡
启用直到（*Active till*）	执行维护期停止活动的日期和时间。
时期（*Periods*）	此块允许您定义进行维护的确切日期和时间。单击 Add 会打开一个弹窗，其中包含灵活的 * 维护期表单 *，您可以在其中定义维护计划。有关详细说明，请参阅维护期。
主机组（*Host groups*）	选择将为其激活维护的主机组。将为指定主机组中的所有主机激活维护。 此字段是自动完成的，因此开始输入它将显示所有可用主机组的下拉列表。 指定父主机组会隐式选择所有嵌套主机组。因此，维护也将在嵌套组的主机上激活。
主机（*Hosts*）	选择将为其激活维护的主机。此字段是自动完成的，因此输入它将显示所有可用主机的下拉列表。
标签（*Tags*）	如果指定了维护标签，对选定主机的维护仍将被激活，但只有当它们的标签匹配时才会抑制问题（即不采取任何措施）。 如果有多个标签，则计算如下： **And/Or** - 所有标签必须对应；但是标签名相同的标签由 Or 条件计算 **Or** - 一个标签对应即可 匹配标签值有两种方式： **Contains** - 区分大小写的子字符串匹配（标签值包含输入的字符串） **Equals** - 区分大小写字符串匹配（标签值等于输入的字符串）
描述（*Description*）	维护期说明。

维护期间

维护期间窗口用于安排定期或一次性维护的时间。该表单是动态的，可用字段根据所选的期间类型（*Period type*）而变化。



期间类型（Period type）	描述（Description）
一次性（*One time only*）	定义日期和时间，以及维护期的长度。
每日（*Daily*）	每天（*Every day(s)*） - 维护频率：1（默认）- 每天，2 - 每两天等。 于（时: 分）（*At (hour:minute)*） - 维护开始的一天中的时间 维护期长度（*Maintenance period length*） - 维护将持续多长时间。
每周（*Weekly*）	每周（*Every week(s)*） - 维护频率：1（默认）- 每天，2 - 每两天等。 星期几（*Day of week*） - 应该在哪一天进行维护。 在（时: 分）（*At (hour:minute)*） - 维护开始的一天中的时间 维护期长度（*Maintenance period length*） - 维护将持续多长时间。

期间类型（Period type）	描述（Description）
每月（*Monthly*）	月（*Month*） - 选择进行定期维护的所有月份。 日期（*Date*）：月的指定天（**Day of month**） - 如果维护发生在每个月的同一日期（例如，每月的第一天），请选择此选项。然后，在出现的新字段中选择所需的日期。 日期（*Date*）：星期几（**Day of week**） - 如果仅在特定日期（例如，每月的第一个星期一）进行维护，请选择此选项。然后，在下拉列表中选择每月所需的一周（第一、第二、第三、第四或最后一周）并标记维护日的复选框。 于（时: 分）（*At (hour:minute)*） - 一天中开始维护的时间 维护期长度（*Maintenance period length*） - 维护将持续多长时间。

完成后，点击 *Add* 将维护期添加到 期间（*Periods*）块。

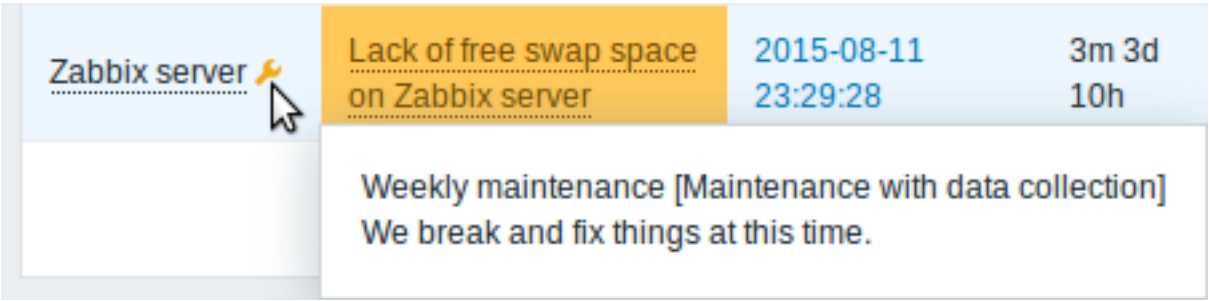
注意：

- 当 *Every day/Every week* 参数大于 1 时，起始日或周为 *Active since* 时间所在的日/周。例如：
  - 将 *Active since* 设置为 1 月 1 日 12:00 且每两天晚上 11 点设置一小时维护，将导致第一个维护期从 1 月 1 日晚上 11 点开始，而第二个维护期将从 1 月 3 日晚上 11 点开始;
  - 在相同的 *Active since* 时间和每两天凌晨 1 点设置一小时维护的情况下，第一个维护期将从 1 月 3 日凌晨 1 点开始，而第二个维护期将从 1 月 5 日凌晨 1 点开始。
- 夏令时 (**DST**) 更改不会影响维护时间。假设我们有两个小时的维护，通常从凌晨 1 点开始，到凌晨 3 点结束：
  - 如果在维护一小时后（凌晨 2 点）发生 DST 更改并且当前时间从 2:00 更改为 3:00，则维护将再持续一小时直到 4:00；
  - 如果在维护两个小时后（凌晨 3 点）发生 DST 更改并且当前时间从 3:00 更改为 2:00，则维护将停止，因为两个小时已过。

展示　显示维护中的主机

主机名旁边的橙色扳手图标 表示该主机正在维护中：

- 监控（*Monitoring*）→ 仪表板（*Dashboard*）
- 监控（*Monitoring*）→ 问题（*Problems*）
- 资产清单（*Inventory*）→ 主机（*Hosts*）→ 主机清单详细信息（*Host inventory details*）
- 配置（*Configuration*）→ 主机（*Hosts*）（参见 “状态（status）” 列）



将鼠标指针放在图标上时会显示维护详细信息。

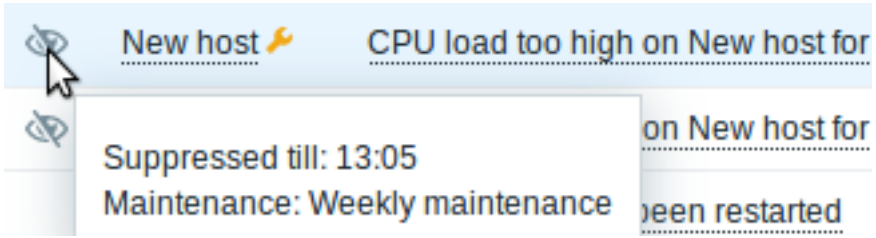
此外，维护中的主机在监控（*Monitoring*）→ 拓扑图（*Maps*）中显示为橙色背景。

显示抑制的问题

通常，维护中的主机问题会被抑制，即不会显示在前端。但是，也可以通过在以下位置选择 显示抑制的问题选项来配置显示抑制的问题：

- 监控（*Monitoring*） → 仪表板（*Dashboard*）（在问题主机（*Problem hosts*）、问题（*Problems*）、严重性问题（*Problems by severity*）、触发器概览（*Trigger overview*）小部件配置）
- 监控（*Monitoring*） → 问题（*Problems*）（在过滤器中）
- 监控（*Monitoring*） → 拓扑图（*Maps*）（在拓扑图配置中）
- 全局通知（在用户配置文件配置中）

显示抑制的问题时，会显示以下图标： 。将鼠标悬停在图标上会显示更多详细信息：



12. 正则表达式

概述　Zabbix 支持 Perl 兼容正则表达式（PCRE, PCRE2）。

Zabbix 中使用正则表达式有两种方式：

- 手动输入正则表达式
- 使用在 Zabbix 中创建的全局正则表达式

正则表达式　你可以在支持的位置手动输入正则表达式。注意，表达式可能不以 @ 开头，因为该符号在 Zabbix 中用于引用全局正则表达式。

Warning:
使用正则表达式时可能会耗尽堆栈。有关更多信息，请参见 pcrestack 帮助页 。

在多行匹配中，锚点 ^ 和 $ 分别匹配每行的开头/结尾，而不是整个字符串的开头/结尾。

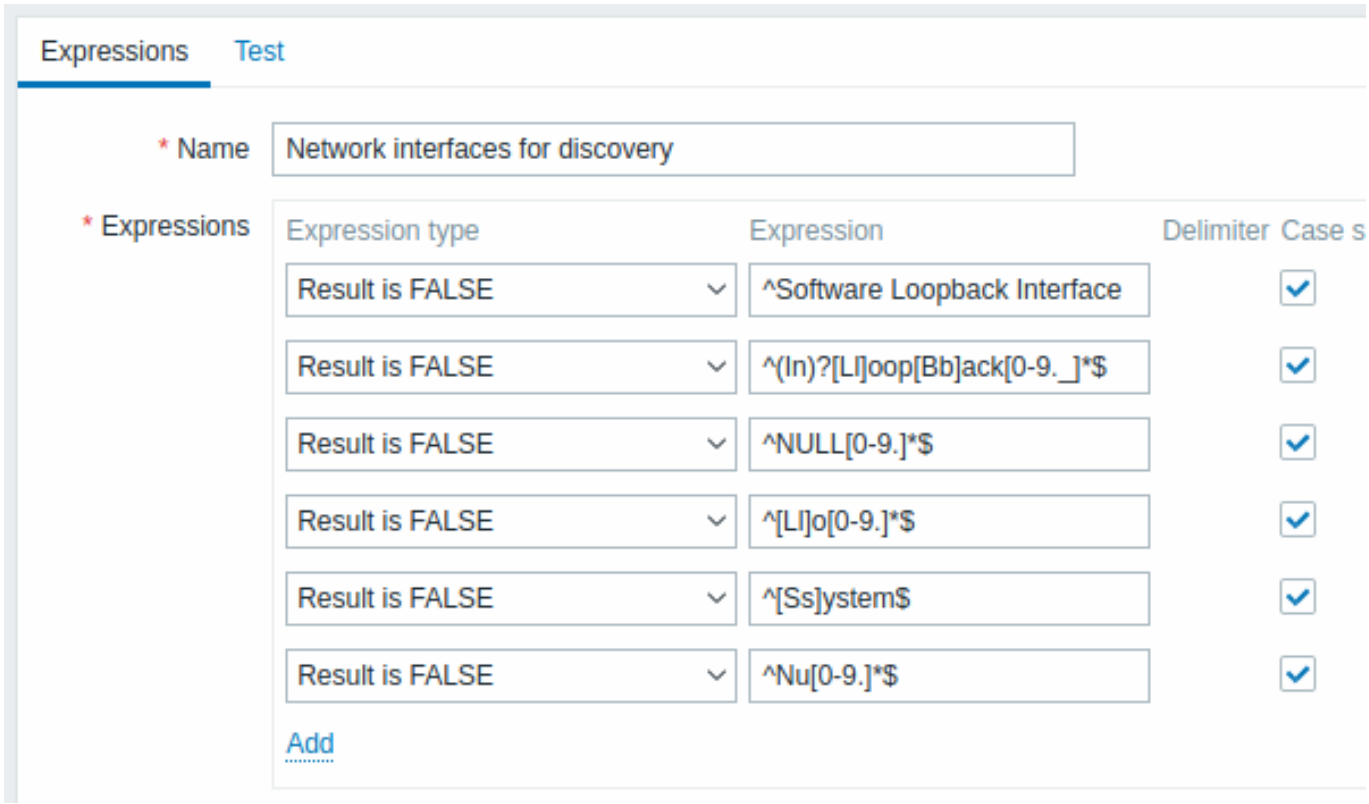
全局正则表达式　Zabbix 前端有一个用于创建和测试复杂正则表达式的高级编辑器。

一旦以这种方式创建了正则表达式，就可以在前端的多个位置通过引用其名称（以 @ 为前缀）使用它，例如 *@mycustomregexp*。

创建全局正则表达式：

- 点击 *Administration*（管理）→ *General*（一般）
- 从下拉列表中选择 *Regular expressions*（正则表达式）
- 点击 *New regular expression*（新建正则表达式）

表达式选项卡允许设置正则表达式名称并添加子表达式。



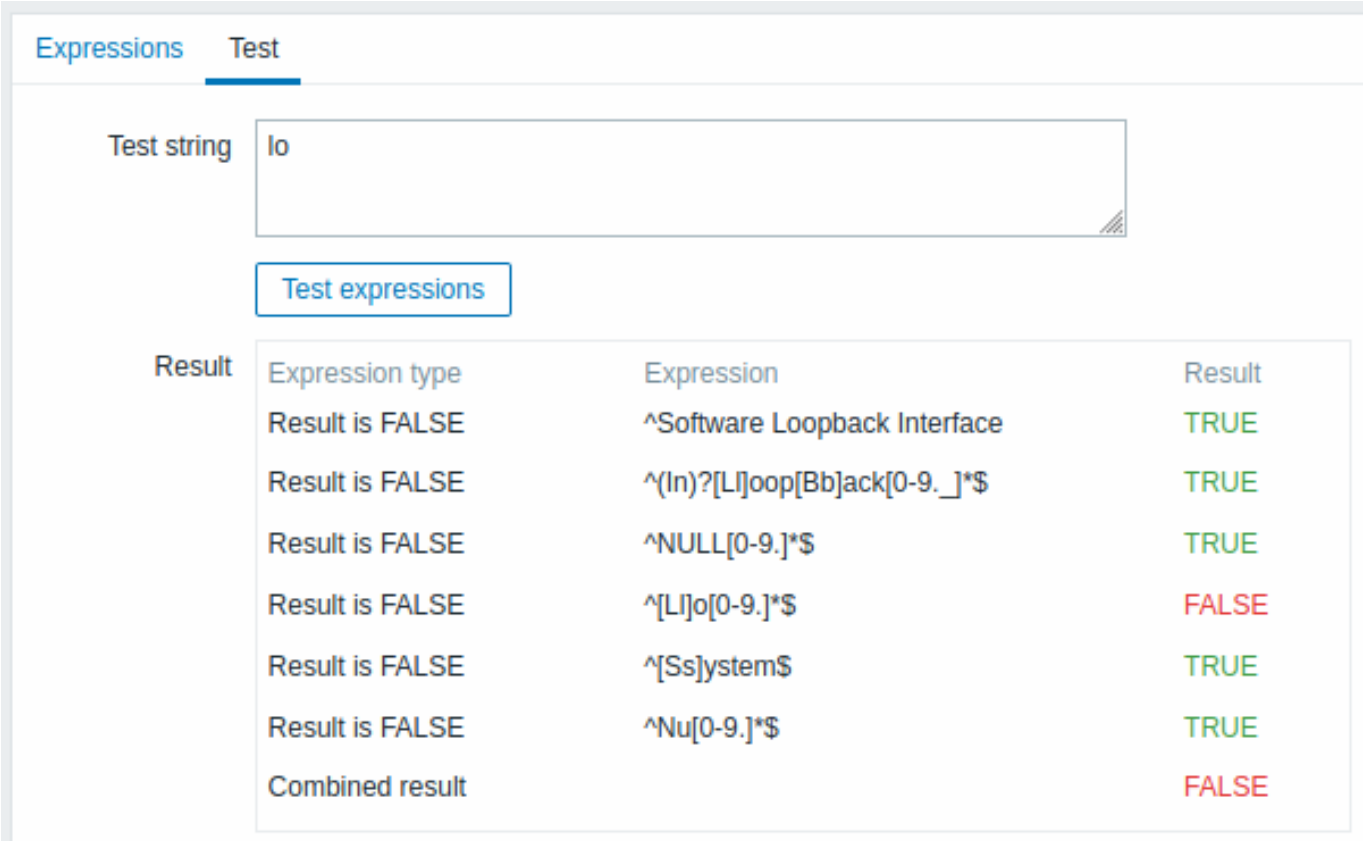
所有必填字段都标有红色星号。

参数		描述
Name（名字）		设置正则表达式名称。允许使用任何 Unicode 字符。
Expressions（表达式）		单击表达式块中的 *Add*（添加）以添加新的子表达式。
	Expression type（表达式类型）	选择表达式类型： **Character string included**（包含字符串） - 匹配子字符串 **Any character string included**（包含任何字符串） - 匹配分隔列表中的任何子字符串。分隔列表包括逗号（,）、点（.）或正斜杠（/）。 **Character string not included**（不包含字符串） - 匹配除子字符串以外的任何字符串 **Result is TRUE**（结果为 **TRUE**） - 匹配正则表达式 **Result is FALSE**（结果为 **FALSE**） - 不匹配正则表达式
	Expression（表达式）	输入子字符串/正则表达式。
Delimiter（分隔符）		逗号 (,)、点 (.) 或正斜杠 (/) 用于分隔正则表达式中的文本字符串。 当选择“包含任意字符串”的表达式类型时，此参数才有效。
Case sensitive（区分大小写）		一个复选框，用于指定正则表达式对大小写字母的敏感性。

表达式中的正斜杠 (/) 按字面意思处理，而不是分隔符。这样就可以保存包含斜杠的表达式而不会出错。

Attention:
Zabbix 中的自定义正则表达式名称可能包含逗号、空格等。在那些可能导致引用时误解的情况下（例如，监控项键值参数中的逗号）整个引用可能会像这样放在引号中：*”@My custom regexp for purpose1, purpose2”*。正则表达式名称不得在其他位置引用（例如，在 LLD 规则属性中）。

在 测试选项卡中，可以通过提供测试字符串来测试正则表达式及其子表达式。



结果显示每个子表达式的状态和总的自定义表达式状态。

总自定义表达式状态定义为组合结果。如果定义了多个子表达式，Zabbix 使用 AND 逻辑运算符计算 组合结果。这意味着如果至少一个 Result 为 False，组合结果也为 False 状态。

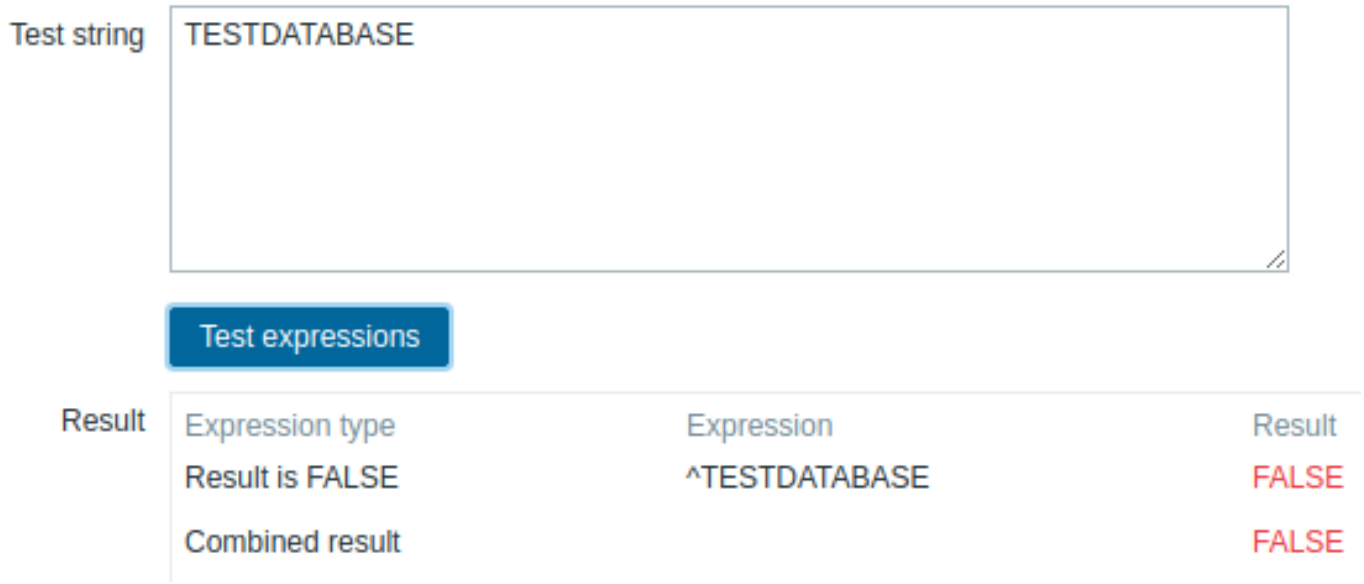
默认全局正则表达式　Zabbix 在其默认数据集中提供了几个全局正则表达式。

名字	表达式	匹配结果
用于发现的文件系统	^(btrfs|ext2|ext3|ext4|jfs|reiser|xfs|ffs|ufs|jfs|jfs2|vxfs|hfs|refs|apfs|ntfs|fat32|zfs)	”btrfs” 或”ext2” 或”ext3” 或”ext4” 或”jfs” 或”reiser” 或”xfs” 或”ffs” 或”ufs” 或”jfs” 或”jfs2” 或”vxfs” 或”hfs” 或”refs” 或”apfs” 或”ntfs” 或”fat32” 或”zfs”
用于发现的网络接口	^Software Loopback Interface	以”Software Loopback Interface（软件环回接口）” 开头的字符串。
	^lo$	”lo”
	^(In)?[Ll]oop[Bb]ack[0-9._]*$	以”In” 为开头，接着是”L” 或”l”，然后是”oop”，然后”B” 或”b”，再然后”ack” 组合的字符串。可以选择后跟任意数量的数字、点或下划线。
	^NULL[0-9.]*$	以”NULL” 开头的后可选地跟任意数量的数字或点的字符串。
	^[Ll]o[0-9.]*$	以”Lo” 或”lo” 开头的后跟任意数量的数字或点的字符串。
	^[Ss]ystem$	匹配”System” 或”system”。
	^Nu[0-9.]*$	以”Nu” 开头的后可选地跟任意数量的数字或点的字符串。
用于 *SNMP* 发现的存储设备	^(Physical memory|Virtual memory|Memory buffers|Cached memory|Swap space)$	”Physical memory” 或”Virtual memory” 或”Memory buffers” 或”Cached memory” 或”Swap space”。
用于发现的 *Windows* 服务名称	^(MMCSS|gupdate|SysmonLog|clr_optimization_v2.0.50727_32|clr_optimization_v4.0.30319_32)	”MMCSS” 或”gupdate” 或”SysmonLog” 或类似于”clr_optimization_v2.0.50727_32” 和”clr_optimization_v4.0.30319_32” 的除了换行符之外可以放置任何字符代替点的字符串。
用于发现的 *Windows* 服务启动状态	^(automatic|automatic delayed)$	”automatic（自动）” 或”automatic delayed（自动延迟）”。

示例　示例 1

在低级别自动发现中使用下列表达式发现具有特定名称的数据库以外的数据库：

^TESTDATABASE$

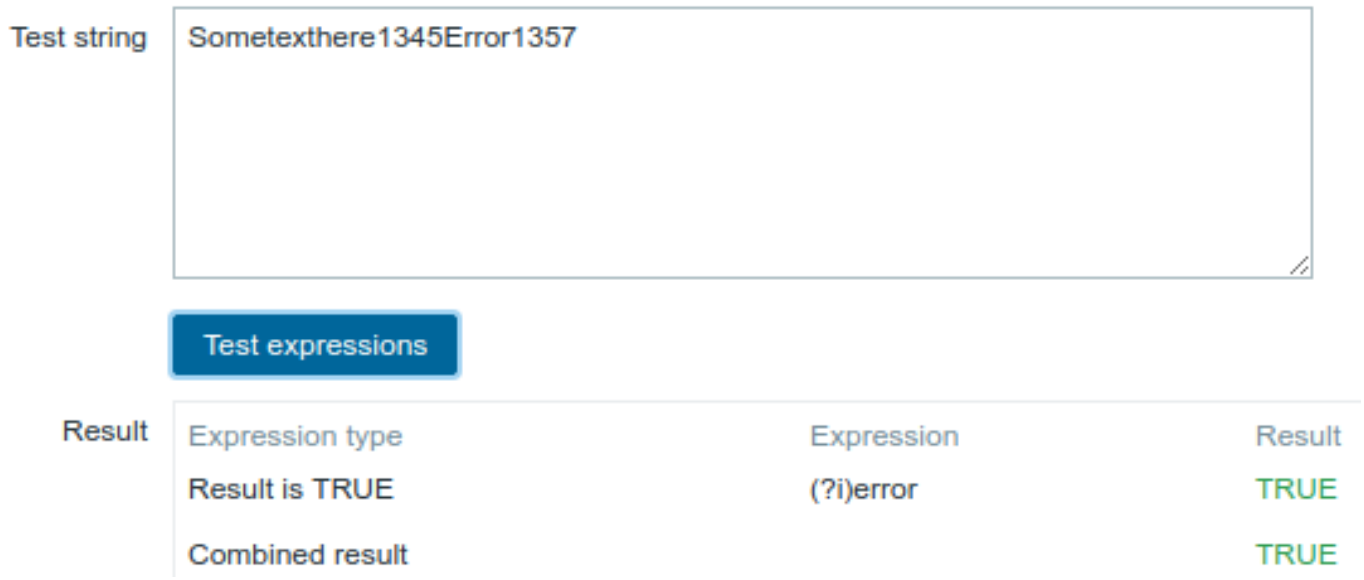


选择的 *Expression type*（表达式类型）：“Result is FALSE（结果为假）”。与名称不匹配，包含字符串”*TESTDATABASE*”。

使用内联正则表达式修饰符的示例

使用以下正则表达式，包括一个内联修饰符 (?i) 来匹配字符”error”：

```
(?i)error
```



选择的 *Expression type*（表达式类型）：”Result is TRUE（结果为真）”。字符”error” 被匹配。

另一个使用内联正则表达式修饰符的示例

使用以下包含多个内联修饰符的正则表达式来匹配特定行之后的字符：

```
(?<=match (?i)everything(?-i) after this line\n)(?sx).*# we add s modifier to allow . match newline charac
```



选择的表达式类型：”Result is TRUE（结果为真）”。匹配特定行之后的字符。

Attention:
g 修饰符不能在行中指定。可用修饰符列表可以参考 pcresyntax 手册页。有关 PCRE 语法的更多信息，请参阅 PCRE HTML 文档。

支持正则表达式的位置

位置	正则表达式	全局正则表达式	注释
Agent 监控项			
	eventlog[]	是	`regexp`, `severity`, `source`, `eventid` 参数
	log[]		`regexp` 参数
	log.count[]		
	logrt[]	是/否	`regexp` 参数 两者都支持，`file_regexp` 参数仅支持非全局表达式
	logrt.count[]		
	proc.cpu.util[]	否	`cmdline` 参数

位置	正则表达式	全局正则表达式	注释
	proc.mem[] proc.num[] sensor[]		Linux 2.4 的 `device` 和 `sensor` 参数
	system.hw.macaddr[]		`interface` 参数
	system.sw.packages[]		`package` 参数
	vfs.dir.count[]		`regex_incl`, `regex_excl`, `regex_excl_dir` 参数
	vfs.dir.size[]		`regex_incl`, `regex_excl`, `regex_excl_dir` 参数
	vfs.file.regexp[] vfs.file.regmatch[] web.page.regexp[]		`regexp` 参数
SNMP traps			
	snmptrap[]	是	`regexp` 参数
监控项值预处理	是	否	`pattern` 参数
触发器函数/可计算监控项			
	count()	是	`pattern` 参数 如果 `operator` 参数是 *regexp* 或 *iregexp*
	countunique()	是	
	find()	是	
	logeventid()	是	`pattern` 参数
	logsource()		
低级别自动发现			
	Filters (过滤)	是	是	正则表达式字段
	Overrides (覆盖)	是	否	在 *Operation* (操作）条件中 *matches*（匹配）、*does not match*（不匹配）选项
动作条件	是	否	在 *Host name* (主机名） and *Host metadata*（主机元数据）自动注册条件中 *matches*（匹配）、*does not match*（不匹配）选项
Web 监控	是	否	以 **regex:** 为前缀的变量 *Required string*（需要字符串）字段

位置	正则表达式	全局正则表达式	注释
用户宏上下文	是	否	在宏的上下文中带有 **regex:** 前缀
宏函数			
	regsub() 是 iregsub()	否	`pattern` 参数
图标映射	是	是	*Expression* field
值映射	是	否	*Value*（值）字段如果值映射是 `regexp`

13 问题确认

概览　Zabbix 中的问题事件可以由用户确认。

如果用户收到关于问题事件的通知，他可以转到 Zabbix 前端，使用下面列出的方式之一打开该问题的问题更新弹出窗口并确认问题。同时他们可以输入对这条事件的评论，说他们正在处理或者任何他们想说的内容。

通过这种方式，如果另一个系统用户发现了相同的问题，他们就会立即看到这个问题是否得到了确认以及到目前为止的评论。

这样，与多个系统用户协同解决问题的工作流程便得以进行。

确认状态也可以被定义动作操作。例如你可以定义，仅当事件在一段时间内未得到确认时才向更高级别的管理员发送通知。

要确认事件并对其进行评论，用户必须至少对相应的触发器具有读权限。若要更改问题的严重性或关闭问题，用户必须对相应的触发器具有读写权限。

有以下 几种方式来访问问题更新窗口，该窗口允许确认问题。

- 可以在 监控 *(Monitoring)*→ 问题 *(Problems)* 中选择问题，然后点击列表下方的 批量更新 *(Mass update)*
- 你可以点击 *Ack* 列显示问题的确认状态:
  - 监控 *(Monitoring)* → 仪表盘 *(Dashboard)* （问题 *(Problems)* 和 问题等级 *(Problems by severity)* 组件）
  - 监控 *(Monitoring)* → 问题 *(Problems)*
  - 监控 *(Monitoring)*→ 问题 *(Problems)*→ 事件详细信息 *(Event details)*

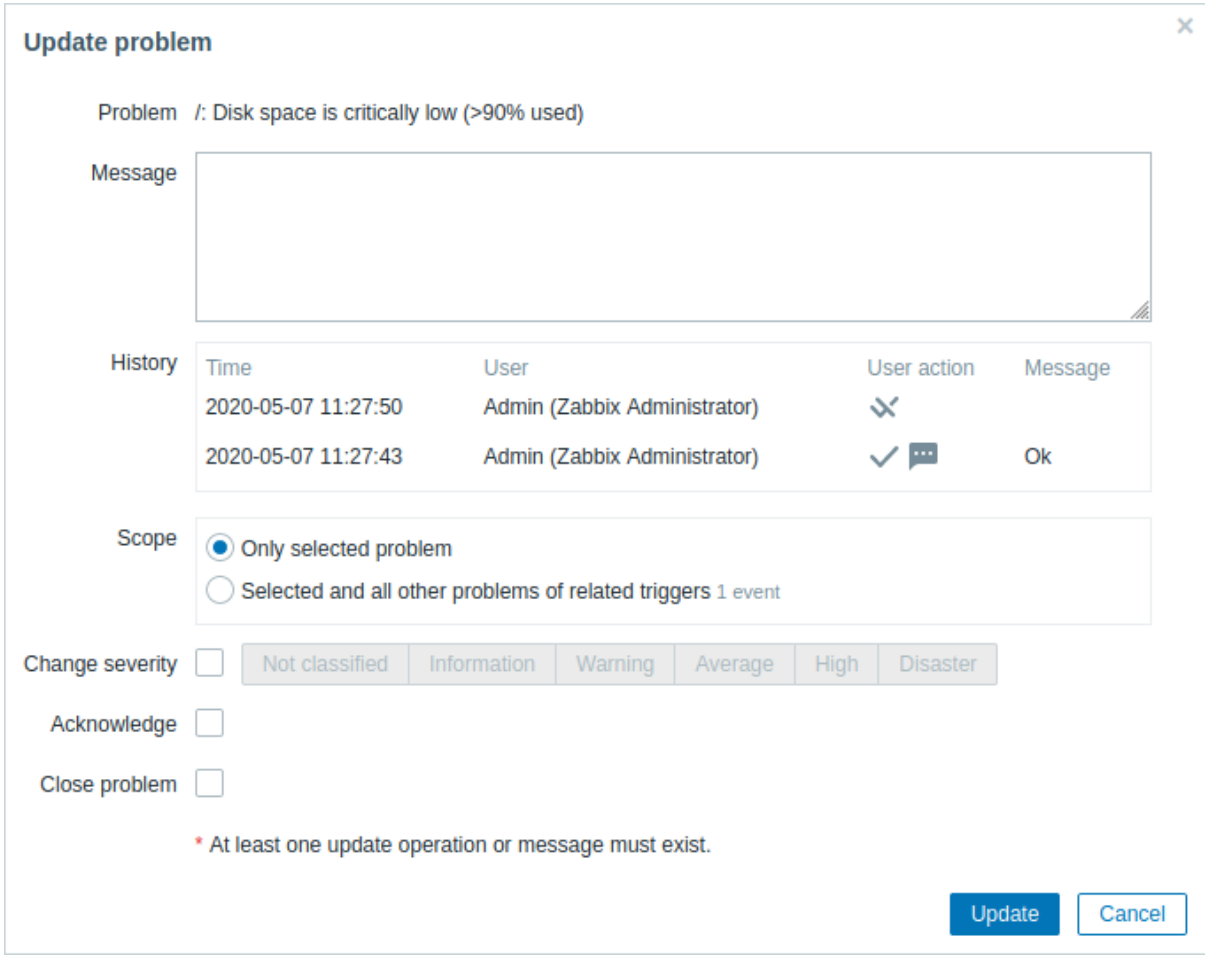
Ack 列包含 “Yes” 或 “No” 链接，分别表示已确认或未确认的问题。点击这些链接会跳转到问题更新弹出窗口。

- 你可以单机未解决的问题单元格:
  - 监控 *(Monitoring)*→ 仪表盘 *(Dashboard)* （触发器概览 *(Trigger overview)* 组件）

弹出菜单包含一个 确认 *(Acknowledge)* 选项，可跳转至问题更新窗口。

更新问题　问题更新弹出窗口允许：

- 评论问题
- 查看到目前为止的评论和操作
- 更改问题严重性
- 确认/取消确认 (acknowledge/unacknowledge) 问题
- 手动关闭问题



所有必填字段都标有红色星号。

参数	描述
问题 *(Problem)*	如果只选择了一个问题，则会显示问题名称。 如果选择了多个问题，则会显示已选择的 *N* 个问题。
信息 *(Message)*	输入文本以评论问题（最多 2048 个字符）。
历史 *(History)*	列出了之前的操作记录和对该问题的评论，以及时间和用户详细信息。 有关用于表示用户操作的图标的含义，请参阅事件详细信息 页面。 注意如果仅选择一个问题进行更新，则会显示历史记录。
范围 *(Scope)*	定义诸如更改严重性、确认或手动关闭问题等操作的范围： 仅选定的问题 **(Only selected problem)** - 仅影响此事件 选定的相关触发器的所有其他问题 **(Selected and all other problems of related triggers)** - 如果确认/关闭问题，将影响此事件和到目前为止尚未确认/关闭的所有其他问题。如果范围包含已经确认或关闭的问题，这些问题将不会重复确认/关闭。另一方面，消息的数量和严重性更改操作不受限制。
更改严重性 *(Change severity)*	标记复选框并单击严重性按钮以更新问题严重性。 如果至少有一个选定问题存在读写权限，则更改严重性的复选框可用。 当点击更新时，只有那些可读写的问题才会被更新。 如果所有选定的触发器都没有读写权限，则该复选框被禁用。
确认 *(Acknowledge)*	标记复选框以确认问题。 如果所选中至少有一个未确认的问题，则此复选框可用。 无法为已确认的问题添加另一个确认（但是可以添加另一个评论）。
取消确认 *(Unacknowledge)*	标记复选框以取消确认问题。 如果在选定项中至少存在一个已确认的问题，则此复选框可用。 不可能对已经确认的问题再增加另外一个确认（尽管可以增加另外的评论）。

参数	描述
关闭问题 *(Close problem)*	标记复选框以手动关闭选定的问题。 如果在触发器配置 中为至少一个选定的问题选中了 允许手动关闭选项，则关闭问题的复选框可用。只有那些允许关闭的问题在点击 更新 *(Update)* 时才会被关闭。 如果没有问题可以手动关闭，则该复选框被禁用。 已经关闭的问题不会重复关闭。

展示　根据确认信息，可以配置问题计数在仪表板或拓扑图中的显示方式。为此，你必须在 问题展示（*Problem display*）选项中进行选择，该选项在拓扑图配置（map configuration） 和仪表盘（dashboard widget）的问题（按严重性）中均可用。可以将所有的问题计数、未确认的问题计数与总数分开显示，或者仅显示未确认的问题计数。

根据问题更新信息（确认等），可以配置更新操作——发送消息或执行远程命令。

14. 配置导出/导入

概览　Zabbix 导出/导入功能，使得在一个 Zabbix 系统和另一个 Zabbix 系统之间交换各种配置实体成为可能。

此功能的典型用例:

- 共享模板或网络拓扑图：Zabbix 用户可以共享他们的配置参数
- 在 *share.zabbix.com* 上共享 web 场景：导出带有 web 场景的模板，并上传到 *share.zabbix.com*。其他人可以下载该模板并将模板文件导入用户自己的 Zabbix 平台中。
- 与第三方工具集成：通用的 YAML、XML 和 JSON 格式使得与第三方工具和应用程序的集成和数据导入/导出成为可能

可以导出/导入什么

可以导出/导入的对象有:

- 主机群组 (只能通过 *Zabbix API*)
- 模板
- 主机
- 网络设备拓扑图
- 媒介类型
- 图片

输出格式

数据可以使用 Zabbix web 前端或Zabbix API。支持的导出格式是 YAML, XML 和 JSON。

详细的导出信息

- 所有支持的元素都导出到一个文件中。
- 从链接模板继承的主机和模板实体 (项目、触发器、图形、发现规则) 不会导出。在宿主级别上对这些实体所做的任何更改 (如更改项目间隔、修改正则表达式或向低级别自动发现规则添加原型) 将在导出时丢失; 当导入时，所有来自链接模板的实体将被重新创建为原始链接模板上的实体。
- 由低级别自动发现创建的实体以及依赖于它们的任何实体都不会被导出。例如，为 lld 规则生成的项创建的触发器将不会被导出。

详细的导入信息

- 导入在第一个错误时停止。
- 导入图像时更新已有图像时，“imagetype” 字段被忽略，即无法通过导入更改图像类型。
- 当使用 “删除缺失” 选项导入主机/模板时，导入文件中不存在的主机/模板宏也将被删除。
- 监控项、触发器、图形、主机/模板应用程序、自动发现规则、监控项原型、触发器原型、图像原型的空标签是没有意义的，也就是说，它就像它丢失了一样。其他标签，例如，项目应用程序，是有意义的，即空的标签意味着没有项目的应用程序，缺失的标签意味着不更新应用程序。
- 导入支持 YAML, XML 和 JSON，导入文件必须有正确的文件扩展名: .yaml 和.yml for YAML, .xml for XML 和.json for JSON.
- 查看compatibility information 关于支持的 XML 版本。

```
zabbix_export:
  version: '6.0'
  date: '2020-04-22T06:20:11Z'
```

YAML 基本格式

zabbix_export:

Zabbix YAML 导出的根节点。

version: '6.0'

导出版本

date: '2020-04-22T06:20:11Z'

以 ISO 8601 长格式创建导出时的日期。

其他节点依赖于导出的对象

```
<?xml version="1.0" encoding="UTF-8"?>
<zabbix_export>
    <version>6.0</version>
    <date>2020-04-22T06:20:11Z</date>
</zabbix_export>
```

XML 格式

<?xml version="1.0" encoding="UTF-8"?>

XML 文档的默认头。

<zabbix_export>

Zabbix XML 导出的根元素。

<version>6.0</version>

导出版本。

<date>2020-04-22T06:20:11Z</date>

以 ISO 8601 长格式创建导出时的日期。

其他标记依赖于导出的对象。

```
{
    "zabbix_export": {
        "version": "6.0",
        "date": "2020-04-22T06:20:11Z"
    }
}
```

JSON 格式

"zabbix_export":

Zabbix JSON 导出的根节点。

"version": "6.0"

导出版本。

"date": "2020-04-22T06:20:11Z"

以 ISO 8601 长格式创建导出时的日期。

其他节点依赖于导出的对象。

1 主机群组

在前端主机组中，导出 仅支持主机导出或模板导出。当导出主机或模板时，该主机或模板所属的所有组都会被自动导出。

支持将主机组独立于主机或模板导出。

导出格式

```
groups:
  -
    name: 'Zabbix servers'
```

多个群组/单个群组

参数	类型	描述	详细
name	*string*	群组名称。	

2 模板

概览

模板导出 带有许多相关对象和对象关系。

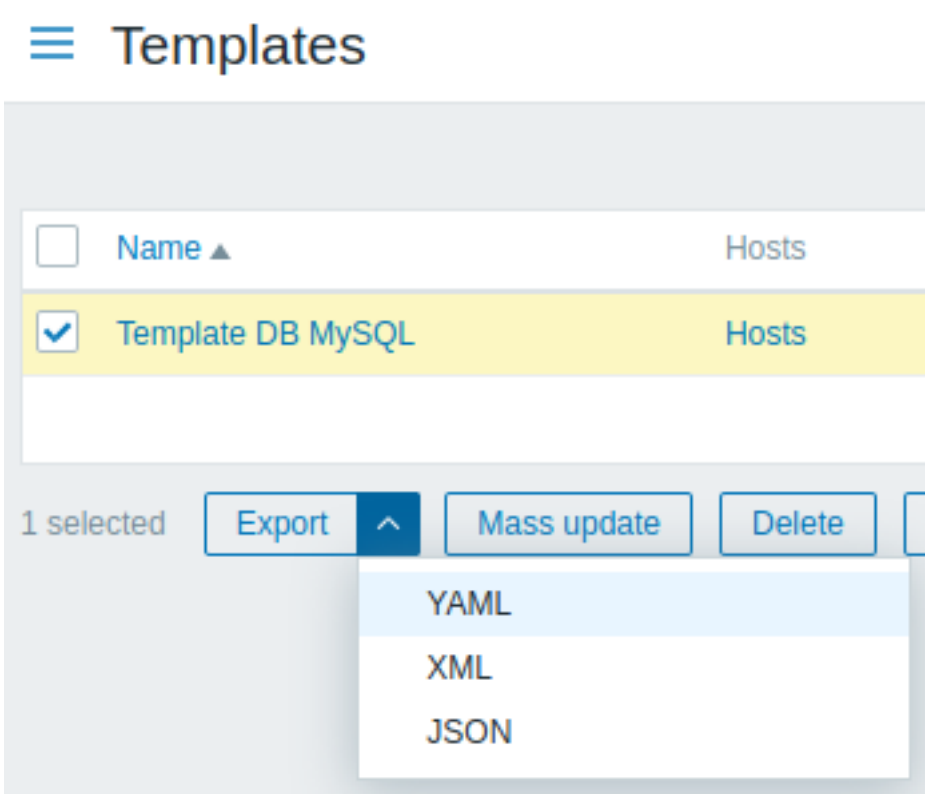
模板导出包含:

- 被链接的主机群组
- 模板数据
- 链接到其他模板
- 链接到其它主机群组
- 直接链接监控项
- 直接链接触发器
- 直接链接图形
- 直接链接仪表盘
- 直接链接自动发现规则及其所有原型（监控项原型、触发器原型、图形原型等等）
- 直接链接 web 监控场景
- 值映射

导出时

导出模板的步骤如下:

- 前往: 配置 → 模板
- 标记要导出的模板的复选框
- 点击列表下方的 *Export*



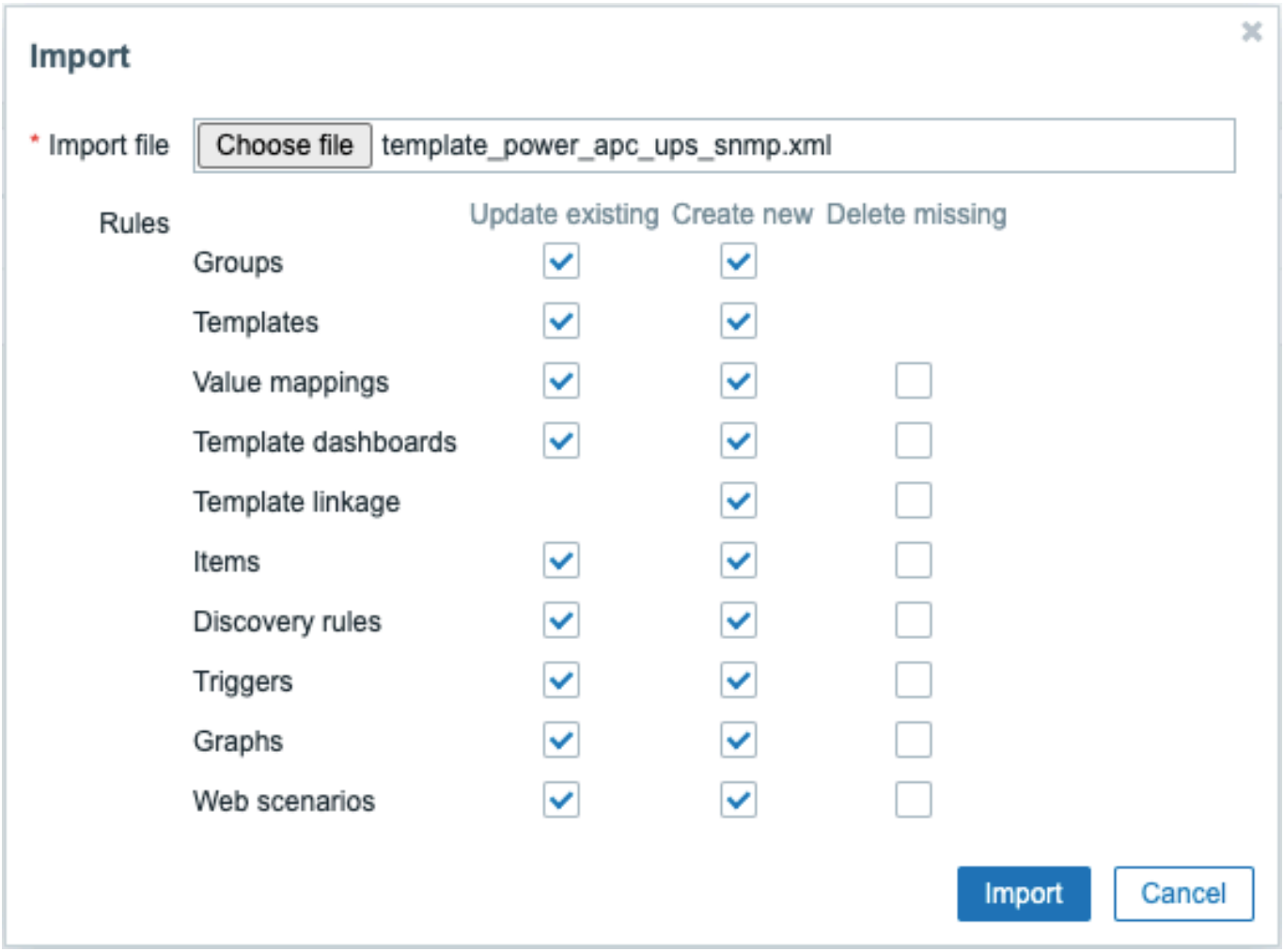
根据所选择的格式，模板被导出到一个默认名称的本地文件:

- *zabbix_export_templates.yaml* - 在 YAML 中导出 (导出的默认选项)
- *zabbix_export_templates.xml* - 在 XML 中导出
- *zabbix_export_templates.json* - 在 JSON 中导出

导入时

导入模板的步骤如下:

- 前往: 配置 → 模板
- 点击右边的 *Import*
- 选择导入文件
- 在导入规则中标记所需的选项
- 点击 导入

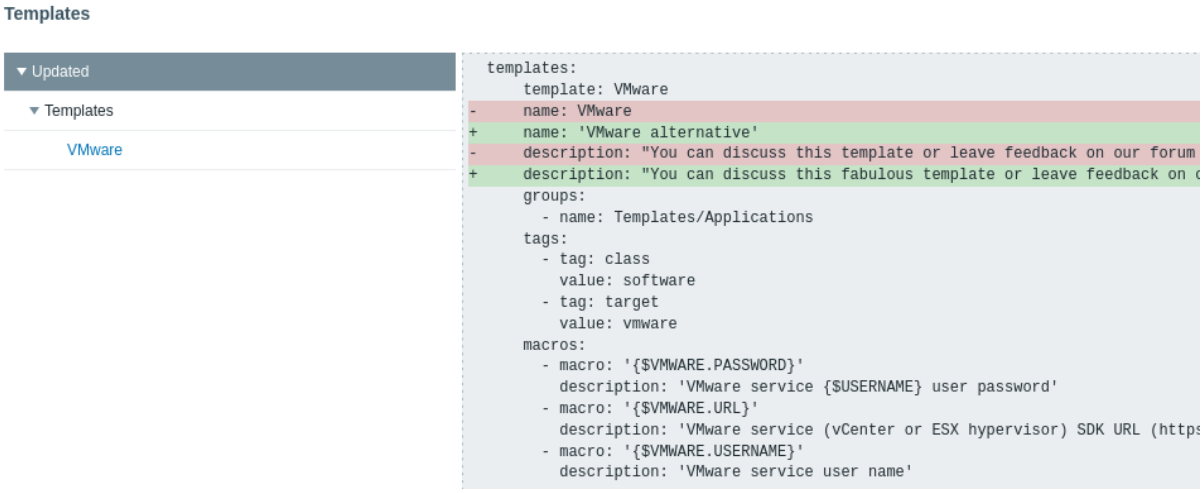


所有强制输入字段都用红色星号标记。

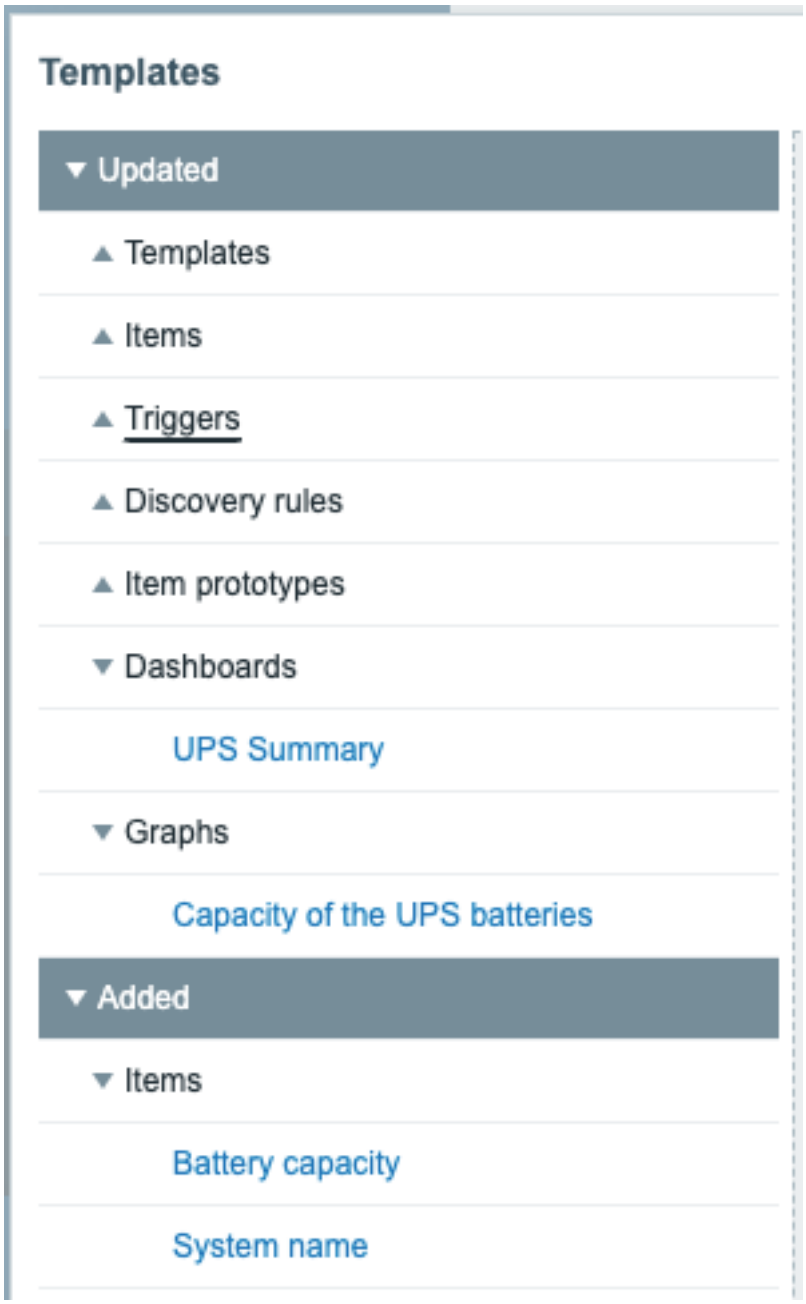
导入规则：

规则	描述
更新现有的	现有元素将使用从导入文件中获取的数据进行更新。否则，它们将不会被更新。
创建新的	导入将使用导入文件中的数据添加新元素。否则，它将不会添加它们。
删除错误	导入将删除导入文件中不存在的现有元素。否则，它将不会删除它们。 如果将 删除错误标记为模板链接，则导入文件中不存在的现有模板链接将从模板中移除，同时移除从可能未链接的模板中继承的所有实体 (监控项, 触发器, 等等)。

在下一个屏幕上，您将能够查看导入的模板的内容。如果这是一个新模板，所有的元素将以绿色列出。如果更新一个现有的模板，新的模板元素会以绿色高亮显示; 删除的模板元素用红色高亮显示; 未更改的元素以灰色背景列出。



可以使用左边的菜单浏览更改列表。部分被更新突出显示了对现有模板元素所做的所有更改。章节被添加列出了新的模板元素。每个部分中的元素按元素类型分组; 向下按灰色箭头以展开或折叠元素组。



查看模板更改，然后按导入执行模板导入。前端将显示导入成功或失败的消息。

导出格式

导出格式在 YAML 上:

```yaml
zabbix_export:
  version: '6.0'
  date: '2021-08-31T12:40:55Z'
  groups:
    -
      uuid: a571c0d144b14fd4a87a9d9b2aa9fcd6
      name: Templates/Applications
  templates:
    -
      uuid: 56079badd056419383cc26e6a4fcc7e0
      template: VMware
      name: VMware
      description: |
        You can discuss this template or leave feedback on our forum https://www.zabbix.com/forum/zabbix-s

        Template tooling version used: 0.38
      templates:
        -
          name: 'VMware macros'
      groups:
        -
          name: Templates/Applications
      items:
        -
          uuid: 5ce209f4d94f460488a74a92a52d92b1
          name: 'VMware: Event log'
          type: SIMPLE
          key: 'vmware.eventlog[{$VMWARE.URL},skip]'
          history: 7d
          trends: '0'
          value_type: LOG
          username: '{$VMWARE.USERNAME}'
          password: '{$VMWARE.PASSWORD}'
          description: 'Collect VMware event log. See also: https://www.zabbix.com/documentation/6.0/manua
          tags:
            -
              tag: Application
              value: VMware
        -
          uuid: ee2edadb8ce943ef81d25dbbba8667a4
          name: 'VMware: Full name'
          type: SIMPLE
          key: 'vmware.fullname[{$VMWARE.URL}]'
          delay: 1h
          history: 7d
          trends: '0'
          value_type: CHAR
          username: '{$VMWARE.USERNAME}'
          password: '{$VMWARE.PASSWORD}'
          description: 'VMware service full name.'
          preprocessing:
            -
              type: DISCARD_UNCHANGED_HEARTBEAT
              parameters:
                - 1d
          tags:
            -
              tag: Application
              value: VMware
        -
          uuid: a0ec9145f2234fbea79a28c57ebdb44d
```

```
      name: 'VMware: Version'
      type: SIMPLE
      key: 'vmware.version[{$VMWARE.URL}]'
      delay: 1h
      history: 7d
      trends: '0'
      value_type: CHAR
      username: '{$VMWARE.USERNAME}'
      password: '{$VMWARE.PASSWORD}'
      description: 'VMware service version.'
      preprocessing:
        -
          type: DISCARD_UNCHANGED_HEARTBEAT
          parameters:
            - 1d
      tags:
        -
          tag: Application
          value: VMware
  discovery_rules:
    -
      uuid: 16ffc933cce74cf28a6edf306aa99782
      name: 'Discover VMware clusters'
      type: SIMPLE
      key: 'vmware.cluster.discovery[{$VMWARE.URL}]'
      delay: 1h
      username: '{$VMWARE.USERNAME}'
      password: '{$VMWARE.PASSWORD}'
      description: 'Discovery of clusters'
      item_prototypes:
        -
          uuid: 46111f91dd564a459dbc1d396e2e6c76
          name: 'VMware: Status of "{#CLUSTER.NAME}" cluster'
          type: SIMPLE
          key: 'vmware.cluster.status[{$VMWARE.URL},{#CLUSTER.NAME}]'
          history: 7d
          username: '{$VMWARE.USERNAME}'
          password: '{$VMWARE.PASSWORD}'
          description: 'VMware cluster status.'
          valuemap:
            name: 'VMware status'
          tags:
            -
              tag: Application
              value: VMware
    -
      uuid: 8fb6a45cbe074b0cb6df53758e2c6623
      name: 'Discover VMware datastores'
      type: SIMPLE
      key: 'vmware.datastore.discovery[{$VMWARE.URL}]'
      delay: 1h
      username: '{$VMWARE.USERNAME}'
      password: '{$VMWARE.PASSWORD}'
      item_prototypes:
        -
          uuid: 4b61838ba4c34e709b25081ae5b059b5
          name: 'VMware: Average read latency of the datastore {#DATASTORE}'
          type: SIMPLE
          key: 'vmware.datastore.read[{$VMWARE.URL},{#DATASTORE},latency]'
          history: 7d
          username: '{$VMWARE.USERNAME}'
          password: '{$VMWARE.PASSWORD}'
```

```
      description: 'Amount of time for a read operation from the datastore (milliseconds).'
      tags:
        -
          tag: Application
          value: VMware
    -
      uuid: 5355c401dc244bc588ccd18767577c93
      name: 'VMware: Free space on datastore {#DATASTORE} (percentage)'
      type: SIMPLE
      key: 'vmware.datastore.size[{$VMWARE.URL},{#DATASTORE},pfree]'
      delay: 5m
      history: 7d
      value_type: FLOAT
      units: '%'
      username: '{$VMWARE.USERNAME}'
      password: '{$VMWARE.PASSWORD}'
      description: 'VMware datastore space in percentage from total.'
      tags:
        -
          tag: Application
          value: VMware
    -
      uuid: 84f13c4fde2d4a17baaf0c8c1eb4f2c0
      name: 'VMware: Total size of datastore {#DATASTORE}'
      type: SIMPLE
      key: 'vmware.datastore.size[{$VMWARE.URL},{#DATASTORE}]'
      delay: 5m
      history: 7d
      units: B
      username: '{$VMWARE.USERNAME}'
      password: '{$VMWARE.PASSWORD}'
      description: 'VMware datastore space in bytes.'
      tags:
        -
          tag: Application
          value: VMware
    -
      uuid: 540cd0fbc56c4b8ea19f2ff5839ce00d
      name: 'VMware: Average write latency of the datastore {#DATASTORE}'
      type: SIMPLE
      key: 'vmware.datastore.write[{$VMWARE.URL},{#DATASTORE},latency]'
      history: 7d
      username: '{$VMWARE.USERNAME}'
      password: '{$VMWARE.PASSWORD}'
      description: 'Amount of time for a write operation to the datastore (milliseconds).'
      tags:
        -
          tag: Application
          value: VMware
-
  uuid: a5bc075e89f248e7b411d8f960897a08
  name: 'Discover VMware hypervisors'
  type: SIMPLE
  key: 'vmware.hv.discovery[{$VMWARE.URL}]'
  delay: 1h
  username: '{$VMWARE.USERNAME}'
  password: '{$VMWARE.PASSWORD}'
  description: 'Discovery of hypervisors.'
  host_prototypes:
    -
      uuid: 051a1469d4d045cbbf818fcc843a352e
      host: '{#HV.UUID}'
```

```
            name: '{#HV.NAME}'
            group_links:
              -
                group:
                  name: Templates/Applications
            group_prototypes:
              -
                name: '{#CLUSTER.NAME}'
              -
                name: '{#DATACENTER.NAME}'
            templates:
              -
                name: 'VMware Hypervisor'
            macros:
              -
                macro: '{$VMWARE.HV.UUID}'
                value: '{#HV.UUID}'
                description: 'UUID of hypervisor.'
            custom_interfaces: 'YES'
            interfaces:
              -
                ip: '{#HV.IP}'
        -
          uuid: 9fd559f4e88c4677a1b874634dd686f5
          name: 'Discover VMware VMs'
          type: SIMPLE
          key: 'vmware.vm.discovery[{$VMWARE.URL}]'
          delay: 1h
          username: '{$VMWARE.USERNAME}'
          password: '{$VMWARE.PASSWORD}'
          description: 'Discovery of guest virtual machines.'
          host_prototypes:
            -
              uuid: 23b9ae9d6f33414880db1cb107115810
              host: '{#VM.UUID}'
              name: '{#VM.NAME}'
              group_links:
                -
                  group:
                    name: Templates/Applications
              group_prototypes:
                -
                  name: '{#CLUSTER.NAME} (vm)'
                -
                  name: '{#DATACENTER.NAME}/{#VM.FOLDER} (vm)'
                -
                  name: '{#HV.NAME}'
              templates:
                -
                  name: 'VMware Guest'
              macros:
                -
                  macro: '{$VMWARE.VM.UUID}'
                  value: '{#VM.UUID}'
                  description: 'UUID of guest virtual machine.'
              custom_interfaces: 'YES'
              interfaces:
                -
                  ip: '{#VM.IP}'
    valuemaps:
      -
        uuid: 3c59c22905054d42ac4ee8b72fe5f270
```

```
        name: 'VMware status'
        mappings:
          -
            value: '0'
            newvalue: gray
          -
            value: '1'
            newvalue: green
          -
            value: '2'
            newvalue: yellow
          -
            value: '3'
            newvalue: red
```

元素标签

下表解释了元素标记值。

模板标签

元素	元素属性	是否必须的	类型	范围	描述
templates		-			模板的根元素。
	uuid	x	string		此模板的唯一标识符。
	template	x	string		此模板的唯一模板名称。
	name	-	string		模板别名。
	description	-	text		模板描述。
groups		x			模板主机组的根元素。
	uuid	x	string		主机组的唯一标识。
	name	x	string		主机组名称
templates		-			链接模板的根元素。
	name	x	string		模板名称。
tags		-			模板标签的根元素。
	tag	x	string		标签名。
	value	-	string		标签值。
macros		-			模板用户宏的根元素。
	macro	x	string		用户宏名称。
	type	-	string	0 - TEXT (default) 1 - SECRET_TEXT 2 - VAULT	宏类型。
	value	-	string		用户宏值。
	description	-	string		用户宏描述。
valuemaps		-			模板值映射的根元素。
	uuid	x	string		-此值映射的唯一标识符。
	name	x	string		值映射名称。
	mapping	-			映射的根元素。
	value	x	string		值映射的值。
	newvalue	x	string		值映射的新值。

模板监控项标签

元素	元素属性	是否必须	类型	适用范围[1]	描述
items		-			监控项的根元素。
	uuid	x	string		监控项的唯一标识符。
	name	x	string		监控项的名称。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	type	-	string	0 - ZABBIX_PASSIVE (default) 2 - TRAP 3 - SIMPLE 5 - INTERNAL 7 - ZABBIX_ACTIVE 10 - EXTERNAL 11 - ODBC 12 - IPMI 13 - SSH 14 - TELNET 15 - CALCULATED 16 - JMX 17 - SNMP_TRAP 18 - DEPENDENT 19 - HTTP_AGENT 20 - SNMP_AGENT 21 - ITEM_TYPE_SCRIPT	监控项类型。
	snmp_oid	-	string		SNMP 对象 ID。 被 SNMP 监控项需要
	key	x	string		监控项键值。
	delay	-	string	Default: 1m	监控项更新时间间隔 接受秒或带后缀的时间单位 (30s, 1m, 2h, 1d). 可选一个或多个custom intervals 可以指定为灵活的间隔或调度。 多个间隔用分号分隔。 可能会使用用户宏。一个宏必须填满整个字段。不支持字段中的多个宏或宏与文本混合。 灵活间隔可以写成由正斜杠分隔的两个宏 (e.g. {$FLEX_INTERVAL}/{$FLEX
	history	-	string	Default: 90d	表示历史数据应存储多长时间的时间单位。带有后缀、用户宏或 LLD 宏的时间单位。
	trends	-	string	Default: 365d	表示趋势数据应存储多长时间的时间单位。带有后缀、用户宏或 LLD 宏的时间单位。
	status	-	string	0 - 启用 (默认) 1 - 禁用	监控项状态。
	value_type	-	string	0 - 浮点型 1 - 字符型 2 - LOG 3 - 无符号型 (默认) 4 - 文本	接收值类型。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	allowed_hosts	-	string		允许为该项发送数据的主机的 IP 地址列表 (逗号分隔)。 trap 和 HTTP 代理使用。
	units	-	string		返回值的单位 (bps, B, etc).
	params	-	text		其他参数取决于监控项的类型: —脚本、SSH 和 Telnet 监控项的执行脚本; —数据库监控项的 SQL 查询; —可计算监控项的公式。
	ipmi_sensor	-	string		IPMI 传感器. 仅 IPMI 监控项使用.
	authtype	-	string	SSH 代理项的认证类型: 0—PASSWORD(默认) 1—PUBLIC_KEY HTTP 代理项的认证类型: 0—NONE(默认) 1—BASIC 2—NTLM	认证类型。 仅供 SSH 和 HTTP 代理项使用。
	username	-	string		用户名进行身份验证。 用于简单检查、SSH、Telnet、数据库监视器、JMX 和 HTTP 代理项。 SSH 和 Telnet 监控项需要。 当由 JMX 代理使用时，密码也应该与用户名一起指定，或者这两个属性都应该留空。
	password	-	string		密码进行身份验证。 用于简单检查、SSH、Telnet、数据库监视器、JMX 和 HTTP 代理项。 当由 JMX 代理使用时，还应该指定用户名和密码，或者这两个属性都应该留空。
	publickey	-	string		公钥文件的名称。 SSH 代理配置项。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	privatekey	-	`string`		私钥文件名称。 SSH 代理配置项。
	port	-	`string`		监控项监控的自定义端口。 可以包含用户宏。 仅对 SNMP 监控项使用。
	description	-	`text`		监控项描述。
	inventory_link	-	`string`	0 - NONE 大写的主机资产清单字段名称。例如： 4 - ALIAS 6 - OS_FULL 14 - HARDWARE etc.	由监控项填充的主机资产字段。 参考主机资产页面获得支持的主机资产字段及其 id 的列表。
	logtimefmt	-	`string`		日志记录中时间的格式。 仅供日志项使用。
	jmx_endpoint	-	`string`		JMX 端点。 仅供 JMX 代理项使用。
	url	-	`string`		URL 字符串。 仅 HTTP agent 监控项使用.
	allow_traps	-	`string`	0 - NO (default) 1 - YES	允许填充值，就像在一个 trap 监控项。 仅供 HTTP 代理项使用。
	follow_redirects	-	`string`	0 - NO 1 - YES (default)	在共享数据时遵循 HTTP 响应重定向。 仅供 HTTP 代理项使用。
headers		-			HTTP(S) 请求报文头的根元素，其中报文头名称用作键，报文头值用作值。 仅供 HTTP 代理项使用。
	name	x	`string`		Header 名称.
	value	x	`string`		Header 值.
	http_proxy	-	`string`		HTTP 代理连接字符串。 仅供 HTTP 代理项使用。
	output_format	-	`string`	0 - RAW (default) 1 - JSON	如何处理反应。 仅供 HTTP 代理项使用。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	post_type	-	string	0 - RAW (default) 2 - JSON 3 - XML	发布数据体的类型。 仅供 HTTP 代理项使用。
	posts	-	string		HTTP(S) 请求体数据。 仅供 HTTP 代理项使用。
query_fields		-			查询参数的根元素。 仅供 HTTP 代理项使用。
	name	x	string		参数名称
	value	-	string		参数值
	request_method	-	string	0 - GET (default) 1 - POST 2 - PUT 3 - HEAD	请求方法。 仅供 HTTP 代理项使用。
	retrieve_mode	-	string	0 - BODY (default) 1 - HEADERS 2 - BOTH	应该存储响应的哪一部分。 仅供 HTTP 代理项使用。
	ssl_cert_file	-	string		Public SSL 密钥文件路径。 仅供 HTTP 代理项使用。
	ssl_key_file	-	string		Private SSL Key 文件路径。 仅供 HTTP 代理项使用。
	ssl_key_password	-	string		SSL 密钥文件的密码。 仅供 HTTP 代理项使用。
	status_codes	-	string		用逗号分隔的所需 HTTP 状态码范围。支持用户宏。 例如:200,200-{$M}，{$M}，200-400 仅供 HTTP 代理监控项使用。
	timeout	-	string		监控项数据轮询请求超时。支持用户宏。 HTTP 代理和 Script 监控项使用。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	verify_host	-	string	0 - NO (default) 1 - YES	验证 URL 中的主机名是否在通用名称字段或主机证书的主题备用名称字段中。 仅供 HTTP 代理项使用。
	verify_peer	-	string	0 - NO (default) 1 - YES	验证主机证书是否可信。 仅供 HTTP 代理项使用。
parameters		-			用户定义参数的根元素。 仅供脚本项使用。
	name	x	string		参数名称。 仅供脚本项使用。
	value	-	string		参数值。 仅供脚本项使用。
value map		-			值映射。
	name	x	string		要用于该项的值映射的名称。
preprocessing		-			监控项值预处理的根元素。
step		-			个别监控项值预处理步骤。
	type	x	string	1 - MULTIPLIER 2 - RTRIM 3 - LTRIM 4 - TRIM 5 - REGEX 6 - BOOL_TO_DECIMAL 7 - OCTAL_TO_DECIMAL 8 - HEX_TO_DECIMAL 9 - SIMPLE_CHANGE (calculated as (received value-previous value)) 10 - CHANGE_PER_SECOND (calculated as (received value-previous value)/(time now-time of last check)) 11 - XMLPATH 12 - JSONPATH 13 - IN_RANGE 14 - MATCHES_REGEX 15 - NOT_MATCHES_REGEX 16 - CHECK_JSON_ERROR 17 - CHECK_XML_ERROR 18 - CHECK_REGEX_ERROR 19 - DISCARD_UNCHANGED 20 - DISCARD_UNCHANGED_HEARTBEAT 21 - JAVASCRIPT 22 - PROMETHEUS_PATTERN 23 - PROMETHEUS_TO_JSON 24 - CSV_TO_JSON 25 - STR_REPLACE 26 - CHECK_NOT_SUPPORTED	监控项值预处理步骤的类型。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	parameters	-			监控项值预处理步骤的参数的根元素。
	parameter	x	string		监控项值预处理步骤的单个参数。
	error_handler	-	string	0 - ORIGINAL_ERROR (default) 1 - DISCARD_VALUE 2 - CUSTOM_VALUE 3 - CUSTOM_ERROR	在预处理步骤失败时使用的动作类型。
	error_handler_params	-	string		错误处理程序参数与’ error _handler’ 一起使用。
master_item		-			个别监控项的主监控项。
	key	x	string		相关监控项需要。 主监控项键值。
triggers		-			递归最多允许 3 个相关项，且相关项的最大计数为 29999。 简单触发器的根元素。
	有关触发器元素标签值，请参见模板触发器标签.				
tags		-			监控项标签的根元素。
	tag	x	string		标签名称。
	value	-	string		标签值。

模板 LLD 规则标签

元素	元素属性	是否必须	类型	适用范围	描述
discovery_rules	对于大多数元素标记值，请参阅常规项的元素标记值。下面只描述特定于底层发现规则的标记。	-			底层发现规则的根元素。

元素	元素属性	是否必须	类型	适用范围	描述
	type	-	`string`	0 - ZABBIX_PASSIVE (default) 2 - TRAP 3 - SIMPLE 5 - INTERNAL 7 - ZABBIX_ACTIVE 10 - EXTERNAL 11 - ODBC 12 - IPMI 13 - SSH 14 - TELNET 16 - JMX 18 - DEPENDENT 19 - HTTP_AGENT 20 - SNMP_AGENT	监控项类型。
	lifetime	-	`string`	Default: 30d	删除不再被发现的监控项的时间段。支持秒，带有后缀的时间单位或用户宏。
filter					单独的过滤。
	evaltype	-	`string`	0 - AND_OR (default) 1 - AND 2 - OR 3 - FORMULA	用于检查底层发现规则过滤条件的逻辑。
	formula	-	`string`		过滤器条件的自定义计算公式。
conditions		-			过滤器条件的根元素。
	macro	x	`string`		底层发现宏名。
	value	-	`string`		过滤值: 正则表达式或全局正则表达式。
	operator	-	`string`	8 - MATCHES_REGEX (default) 9 - NOT_MATCHES_REGEX	条件操作符。
	formulaid	x	`character`		用于从自定义表达式引用条件的任意惟一 ID。只能包含大写字母。该 ID 必须由用户在修改筛选条件时定义，但在之后请求它们时将重新生成。
lld_macro_paths		-			LLD 宏路径的根元素。
	lld_macro	x	`string`		底层发现宏名。
	path	x	`string`		将被赋值给相应宏的选择器。
preprocessing		-			LLD 规则值预处理。
step		-			单个 LLD 规则值预处理步骤。

元素	元素属性	是否必须	类型	适用范围	描述
	对于大多数元素标记值，请参阅模板项值预处理的元素标记值。下面只描述特定于模板底层发现值预处理的标记。				
	type	x	`string`	5 - REGEX 11 - XMLPATH 12 - JSONPATH 15 - NOT_MATCHES_REGEX 16 - CHECK_JSON_ERROR 17 - CHECK_XML_ERROR 20 - DISCARD_UNCHANGED_HEARTBEAT 21 - JAVASCRIPT 23 - PROMETHEUS_TO_JSON 24 - CSV_TO_JSON 25 - STR_REPLACE	监控项值预处理步骤的类型。
trigger_prototypes	有关触发器原型元素标记值，请参阅 *regular* 模板触发器 标签.	-			触发器原型的根元素。
graph_prototypes	有关图形原型元素标签值，请参阅 *regular* 模板图形 标签.	-			图原型的根元素。
host_prototypes	有关主机原型元素标签值，请参阅 *regular*主机 标签.	-			主机原型的根元素。
item_prototypes	有关监控项原型元素标签值，请参阅 *regular*模板监控项 *tags.*	-			监控项原型的根元素。
master_item		-			单个监控项原型的主监控项/监控项原型数据。
	key	x	`string`		主监控项原型键值/监控项原型键值。 相关项监控需要使用。

模板触发器标签

元素	元素属性	是否必须	类型	适用范围[1]	描述
triggers		-			触发器的根元素。
	uuid	x	`string`		此触发器的唯一标识符。
	expression	x	`string`		触发器表达式。
	recovery_mode	-	`string`	0 - EXPRESSION (default) 1 - RECOVERY_EXPRESSION 2 - NONE	生成 OK 事件的基础。
	recovery_expression	-	`string`		触发器恢复表达式。
	name	x	`string`		触发器名称。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	correlation_mode	-	string	0 - DISABLED (default) 1 - TAG_VALUE	关联模式 (没有事件关联或通过标记进行事件关联)。
	correlation_tag	-	string		用于事件关联的标记名称。
	url	-	string		与触发器关联的URL。
	status	-	string	0 - ENABLED(默认值) 1 - DISABLED	触发状态。
	priority	-	string	0 - NOT_CLASSIFIED (default) 1 - INFO 2 - WARNING 3 - AVERAGE 4 - HIGH 5 - DISASTER	触发器严重等级。
	description	-	text		触发器描述。
	type	-	string	0 - SINGLE (default) 1 - MULTIPLE	事件生成类型 (单个问题事件或多个问题事件)。
	manual_close	-	string	0 - NO (default) 1 - YES	手动关闭问题事件。
dependencies		-			依赖项的根元素。
	name	x	string		依赖触发器的名称。
	expression	x	string		依赖触发器表达式。
	recovery_expression	-	string		依赖项触发器恢复表达式。
tags		-			触发器标签的根元素。
	tag	x	string		标签名称。
	value	-	string		标签值。

模板图形标签

元素	元素属性	是否必须	类型	适用范围[1]	描述
graphs		-			图形的根元素。
	uuid	x	string		此图形的唯一标识符。
	name	x	string		图形的名称。
	width	-	integer	20-65535 (默认: 900)	图形宽度，以像素为单位。用于预览和饼图/分解图。
	height	-	integer	20-65535 (默认: 200)	图形高度，以像素为单位。用于预览和饼图/分解图。
	yaxismin	-	double	默认: 0	Y 轴最小值。 当'ymin_type_1'为固定值时使用。
	yaxismax	-	double	默认: 0	Y 轴最大值。 当'ymax_type_1'为固定值时使用。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	show_work_period	-	string	0 - NO 1 - YES(默认)	突出显示非工作时间。 用于普通图和堆叠图。
	show_triggers	-	string	0 - NO 1 - YES(默认)	将简单的触发值显示为一行。 用于普通图和堆叠图。
	type	-	string	0 - NORMAL(默认) 1 - STACKED 2 - PIE 3 - expanded	图类型。
	show_legend	-	string	0 - NO 1 - YES(默认)	显示图形图例。
	show_3d	-	string	0 - NO(默认) 1 - YES	启用 3D 样式。 用于饼图和分解饼图。
	percent_left	-	double	默认:0	显示左轴的百分位线。 仅用于普通图形。
	percent_right	-	double	默认:0	显示右轴的百分位线。 仅用于普通图形。
	ymin_type_1	-	string	0 - calculate(默认值) 1 - FIXED 2 - ITEM	Y 轴的最小值。 用于普通图和堆叠图。
	ymax_type_1	-	string	0 - calculate(默认值) 1 - FIXED 2 - ITEM	Y 轴的最大值。 用于普通图和堆叠图。
ymin_item_1		-			个别监控项的细节。 当'ymin_type_1'为 ITEM 时需要。
	host	x	string		监控项主机。
	key	x	string		监控项键值。
ymax_item_1		-			个别监控项的细节。 当'ymax_type_1'为 ITEM 时需要。
	host	x	string		监控项主机。
	key	x	string		监控项键值。
graph_items		x			图形项的根元素。
	sortorder	-	integer		画出顺序。先画出较小的值。可用于在另一个线或区域后面 (或前面) 画线或区域。
	drawtype	-	string	0 - SINGLE_LINE(默认) 1 - FILLED_REGION 2 - BOLD_LINE 3 - dot _LINE 4 -虚线 _LINE 5 -渐变 _LINE	图形项的绘制风格。 仅用于普通图形。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	color	-	string		元素颜色 (6 个符号，十六进制)。
	yaxisside	-	string	0 - LEFT(默认) 1 - RIGHT	图形项的 Y 比例将被绘制的部分。 用于普通图和堆叠图。
	calc_fnc	-	string	1 - MIN 2 - AVG(默认值) 4 - MAX 7 - ALL(最小值，平均值和最大值; 9 - LAST(仅用于饼图和分解饼图)	表示一个监控项存在多个值的数据。
	type	-	string	0 - SIMPLE(默认) 2 - GRAPH_SUM(监控项值代表整个饼图;仅用于饼图和分解饼图)	图形监控项类型。
item		x			单独监控项。
	host	x	string		监控项主机。
	key	x	string		监控项键值。

模板 web 场景标签

元素	元素属性	是否必须	类型	适用范围[1]	描述
httptests		-			根元素为 web 场景。
	uuid	x	string		此 web 场景的唯一标识符。
	name	x	string		web 场景名称。
	delay	-	string	默认: 1m	执行 web 场景的频率。支持秒，带有后缀的时间单位或用户宏。
	attempts	-	integer	1-10 (默认: 1)	尝试执行 web 场景步骤的次数。
	agent	-	string	默认: Zabbix	agent 客户端。Zabbix 将模拟为选中的浏览器。当一个网站为不同的浏览器返回不同的内容时，这是很有用的。
	http_proxy	-	string		指定要使用的 HTTP 代理，格式如下: `http://[username[:passw`
variables		-			步骤级变量 (宏) 的根元素，应该在此步骤之后应用。
	name	x	text		变量名。
	value	x	text		变量值。
headers		-			执行请求时将发送的 HTTP 头的根元素。报头应该使用与 HTTP 协议中相同的语法列出。
	name	x	text		报文头名称。
	value	x	text		报文头值。
	status	-	string	0 - ENABLED (默认) 1 - DISABLED	Web 场景状态。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	authentication	-	string	0 - NONE (默认) 1 - BASIC 2 - NTLM	认证方式。
	http_user	-	string		用于基本认证、HTTP 认证或 NTLM 认证的用户名。
	http_password	-	string		用于基本、HTTP 或 NTLM 身份验证的密码。
	verify_peer	-	string	0 - NO (默认) 1 - YES	验证 web 服务器的 SSL 证书。
	verify_host	-	string	0 - NO (默认) 1 - YES	验证 web 服务器证书的 Common Name 字段或 Subject Alternate Name 字段是否匹配。
	ssl_cert_file	-	string		用于客户端认证的 SSL 证书文件名 (必须为 PEM 格式)。
	ssl_key_file	-	string		用于客户端认证的 SSL 私钥文件名 (必须为 PEM 格式)。
	ssl_key_password	-	string		SSL 私钥文件密码。
steps		x			web 场景步骤的根元素。
	name	x	string		Web 场景步骤名。
	url	x	string		URL 的监测。
query_fields		-			查询字段的根元素——执行请求时将添加到 URL 的 HTTP 字段数组。
	name	x	string		查询字段名。
	value	-	string		查询字段值。
posts		-			HTTP POST 变量为字符串 (原始 POST 数据) 或 HTTP 字段数组 (表单字段数据)。
	name	x	string		字段名。
	value	x	string		字段值。
variables		-			步骤级变量 (宏) 的根元素，应该在此步骤之后应用。 如果变量值有'regex:' 前缀，那么它的值将根据'regex:' 前缀后面的正则表达式模式从这一步返回的数据中提取
	name	x	string		变量名。
	value	x	string		变量值。

元素	元素属性	是否必须	类型	适用范围[1]	描述
headers		-			执行请求时将发送的 HTTP 头的根元素。报头应该使用与 HTTP 协议中相同的语法列出。
	name	x	string		报文头名称。
	value	x	string		报文头值。
	follow_redirects	-	string	0 - NO 1 - YES (默认)	遵循 HTTP 重定向。
	retrieve_mode	-	string	0 - BODY (默认) 1 - HEADERS 2 - BOTH	HTTP 响应检索模式。
	timeout	-	string	默认: 15s	步骤执行的超时时间。秒，带有后缀或用户宏的时间单位。
	required	-	string		必须在响应中出现的文本。如果空的忽视。
	status_codes	-	string		被接受的 HTTP 状态码的逗号分隔列表。如果空的忽视。例如: 200-201,210-299
tags		-			web 场景标签的根元素。
	tag	x	string		标签名称。
	value	-	string		标签值。

模板仪表盘标签

元素	元素属性	是否必须	类型	使用范围[1]	描述
dashboards		-			模板仪表板的根元素。
	uuid	x	string		此仪表板的唯一标识符。
	name	x	string		模板仪表盘的名字。
	display period	-	integer		仪表盘页面的显示周期。
	auto_start	-	string	0 - no 1 - yes	幻灯片自动开始。
pages		-			模板仪表板页面的根元素。
	name	-	string		页面名称。
	display period	-	integer		页面显示。
	sortorder	-	integer		页面排序顺序。
widgets		-			模板仪表板小部件的根元素。
	type	x	string		小部件类型。
	name	-	string		小部件名称。
	x	-	integer	0-23	模板仪表板左侧的水平位置。
	y	-	integer	0-62	从模板仪表板顶部开始的垂直位置。
	width	-	integer	1-24	小部件宽度。
	height	-	integer	2-32	小部件的高度。

元素	元素属性	是否必须	类型	使用范围[1]	描述
	hide_header	-	string	0 - no 1 - yes	隐藏小部件标题。
fields		-			模板仪表板小部件字段的根元素。
	type	x	string	0 - INTEGER 1 - STRING 3 - HOST 4 - ITEM 5 - ITEM_PROTOTYPE 6 - GRAPH 7 - GRAPH_PROTOTYPE	小部件字段类型。
	name	x	string		小部件字段名。
	value	x	mixed		小部件字段值，取决于字段类型。

附注

[1] 对于字符串值，只有字符串将被导出 (例如"ZABBIX_ACTIVE")，而不使用该表中使用的编号。该表中的范围值 (对应于 API 值) 的编号仅用于排序。

3 主机

概述

导出（exported）的主机具有许多相关对象和对象关系。

主机导出的内容包含：

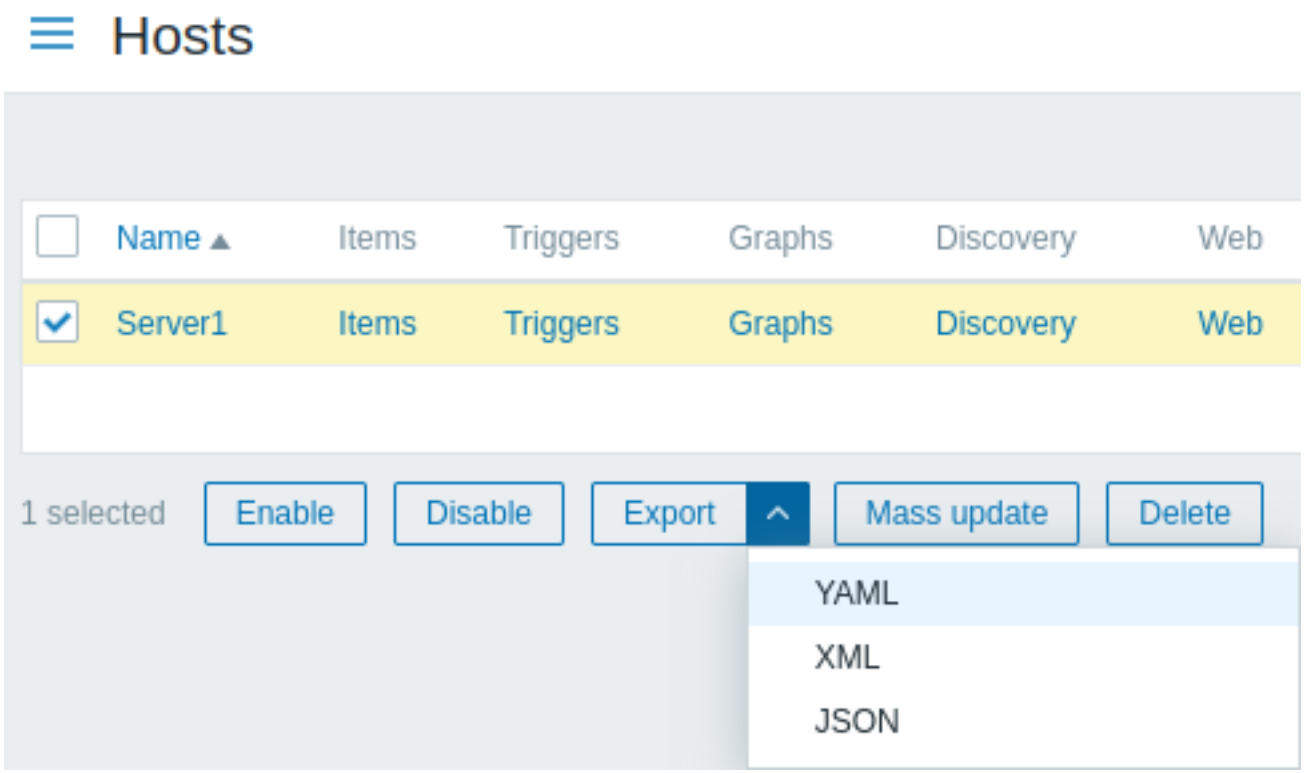
- 链接的主机组

- 主机数据
- 模板链接
- 主机组链接
- 主机接口
- 直接链接的应用集
- 直接链接的监控项
- 直接链接的触发器
- 直接链接的图形
- 直接链接的具有所有原型的发现规则
- 直接链接的 web 场景
- 主机宏
- 主机资产清单数据
- 值映射

导出

要导出主机，请执行以下操作:

- 前往: *Configuration* → *Hosts*
- 标记要导出的主机的复选框
- 单击列表下方的导出



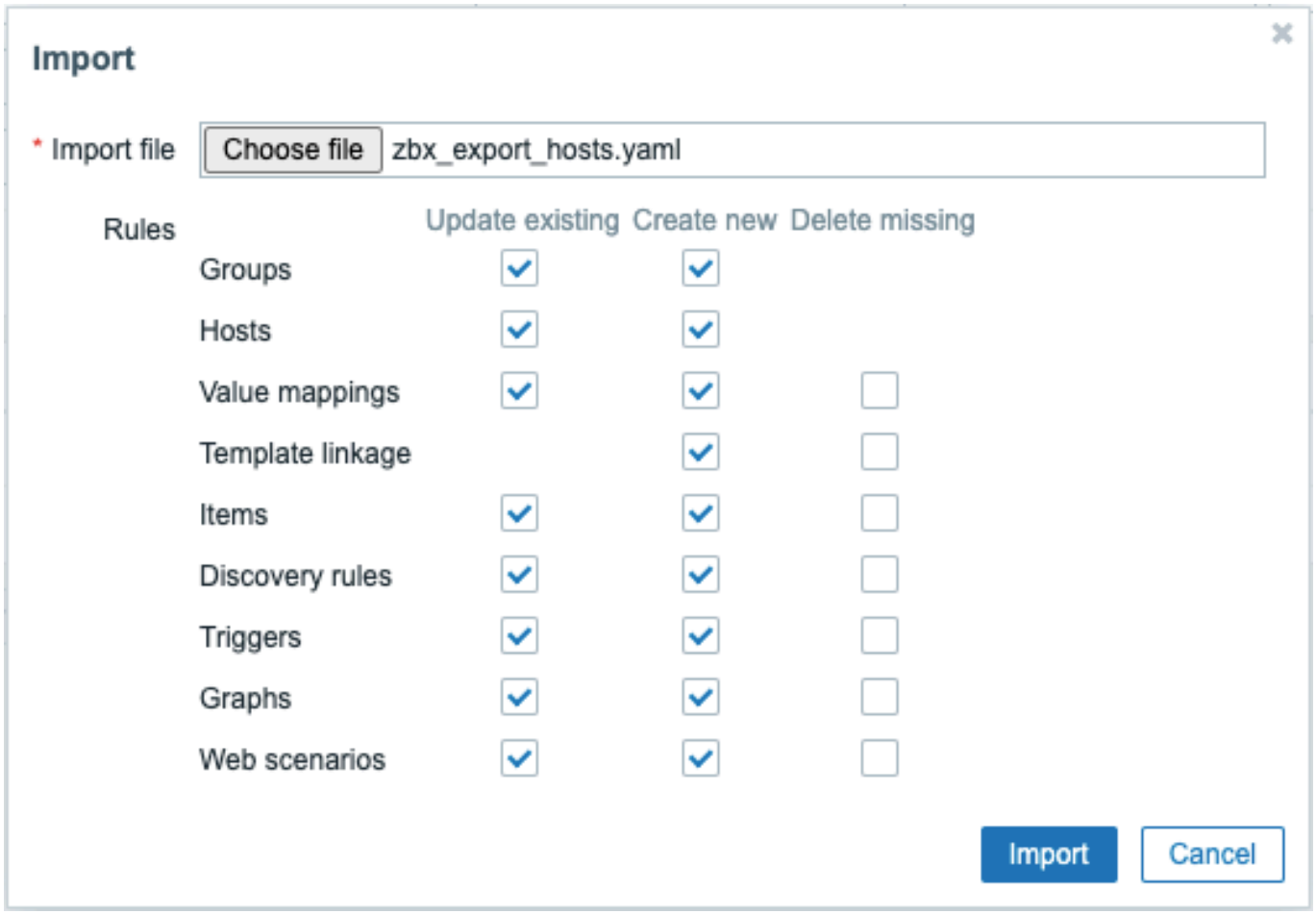
根据选定的格式，主机将导出到具有默认名称的本地文件：

- *zabbix_export_hosts.yaml* - 导出 YAML 格式 (默认导出)
- *zabbix_export_hosts.xml* - 导出 XML 格式
- *zabbix_export_hosts.json* - 导出 JSON 格式

导入

导入主机，按照如下操作：

- 切换到：配置（*Configuration*） → 主机（*Hosts*）
- 单击右侧的 _ 导入（Import）_ 按钮
- 选择导入文件
- 标记导入规则里的必选项
- 单击 _ 导入（Import）_ 按钮



导入的成功或失败消息将显示在前端。

导入规则:

规则	描述
更新已经存在	现有元素将使用从导入文件中获取的数据进行更新。否则它们将不会被更新。
创建新的	导入将使用导入文件中的数据添加新元素。否则它就不会添加它们。
删除缺失值	导入将删除导入文件中不存在的现有元素。否则它不会移除它们 如果为模板链接标记了 *Delete missing*，则导入文件中不存在的现有模板链接将与从可能未链接的模板继承的所有实体（监控项、触发器等）一起从主机中删除。

导出格式

导出格式使用 YAML:

```yaml
zabbix_export:
  version: '6.0'
  date: '2021-09-28T12:20:07Z'
  groups:
    -
      uuid: f2481361f99448eea617b7b1d4765566
      name: 'Discovered hosts'
    -
      uuid: 6f6799aa69e844b4b3918f779f2abf08
      name: 'Zabbix servers'
  hosts:
    -
      host: 'Zabbix server 1'
      name: 'Main Zabbix server'
      templates:
        -
          name: 'Linux by Zabbix agent'
        -
```

```
        name: 'Zabbix server health'
    groups:
      -
        name: 'Discovered hosts'
      -
        name: 'Zabbix servers'
    interfaces:
      -
        ip: 192.168.1.1
        interface_ref: if1
    items:
      -
        name: 'Zabbix trap'
        type: TRAP
        key: trap
        delay: '0'
        history: 1w
        preprocessing:
          -
            type: MULTIPLIER
            parameters:
              - '8'
        tags:
          -
            tag: Application
            value: 'Zabbix server'
        triggers:
          -
            expression: 'last(/Zabbix server 1/trap)=0'
            name: 'Last value is zero'
            priority: WARNING
            tags:
              -
                tag: Process
                value: 'Internal test'
    tags:
      -
        tag: Process
        value: Zabbix
    macros:
      -
        macro: '{$HOST.MACRO}'
        value: '123'
      -
        macro: '{$PASSWORD1}'
        type: SECRET_TEXT
    inventory:
      type: 'Zabbix server'
      name: yyyyyy-HP-Pro-3010-Small-Form-Factor-PC
      os: 'Linux yyyyyy-HP-Pro-3010-Small-Form-Factor-PC 4.4.0-165-generic #193-Ubuntu SMP Tue Sep 17 17
    inventory_mode: AUTOMATIC
graphs:
  -
    name: 'CPU utilization server'
    show_work_period: 'NO'
    show_triggers: 'NO'
    graph_items:
      -
        drawtype: FILLED_REGION
        color: FF5555
        item:
          host: 'Zabbix server 1'
```

```
        key: 'system.cpu.util[,steal]'
    -
      sortorder: '1'
      drawtype: FILLED_REGION
      color: 55FF55
      item:
        host: 'Zabbix server 1'
        key: 'system.cpu.util[,softirq]'
    -
      sortorder: '2'
      drawtype: FILLED_REGION
      color: '009999'
      item:
        host: 'Zabbix server 1'
        key: 'system.cpu.util[,interrupt]'
    -
      sortorder: '3'
      drawtype: FILLED_REGION
      color: '990099'
      item:
        host: 'Zabbix server 1'
        key: 'system.cpu.util[,nice]'
    -
      sortorder: '4'
      drawtype: FILLED_REGION
      color: '999900'
      item:
        host: 'Zabbix server 1'
        key: 'system.cpu.util[,iowait]'
    -
      sortorder: '5'
      drawtype: FILLED_REGION
      color: '990000'
      item:
        host: 'Zabbix server 1'
        key: 'system.cpu.util[,system]'
    -
      sortorder: '6'
      drawtype: FILLED_REGION
      color: '000099'
      calc_fnc: MIN
      item:
        host: 'Zabbix server 1'
        key: 'system.cpu.util[,user]'
    -
      sortorder: '7'
      drawtype: FILLED_REGION
      color: '009900'
      item:
        host: 'Zabbix server 1'
        key: 'system.cpu.util[,idle]'
```

元素标签

下表解释了元素标签值。

主机标签

元素	元素属性	是否必须	类型	适用范围[1]	描述
groups		x			主机群组根元素。
	name	x	string		主机群组名称。
hosts		-			主机的根元素。
	host	x	string		唯一的主机名。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	name	-	string		可见的主机名
	description	-	text		主机描述。
	status	-	string	0 - 启用 (默认) 1 - 禁用	主机状态。
	ipmi_authtype	-	string	-1 - DEFAULT (默认) 0 - NONE 1 - MD2 2 - MD5 4 - STRAIGHT 5 - OEM 6 - RMCP_PLUS	IPMI 会话身份验证类型。
	ipmi_privilege	-	string	1 - CALLBACK 2 - USER (默认) 3 - OPERATOR 4 - ADMIN 5 - OEM	IPMI 会话特权级别。
	ipmi_username	-	string		IPMI 检查的用户名。
	ipmi_password	-	string		IPMI 检查的密码。
proxy		-			Proxy.
	name	x	string		监视主机的 Proxy（如果有）的名称。
templates		-			链接模板的根元素。
	name	x	string		模板名称
interfaces		-			主机接口的根元素。
	default	-	string	0 - NO 1 - YES (默认)	这是否是主主机接口 主机上只能有一种类型的主接口。
	type	-	string	1 - ZABBIX (默认) 2 - SNMP 3 - IPMI 4 - JMX	接口类型
	useip	-	string	0 - NO 1 - YES (默认)	是否使用 IP 作为连接主机的接口（如果不使用，将使用 DNS）。
	ip	-	string		IP 地址，可以是 IPv4 或 IPv6 如果通过 IP 进行连接，则需要。
	dns	-	string		DNS 名称 如果通过 DNS 进行连接，则需要。
	port	-	string		端口号。支持用户宏。
	interface_ref	x	string	Format: `if<N>`	要在监控项中使用的接口引用名称。
details		-			接口详细信息的根元素。
	version	-	string	1 - SNMPV1 2 - SNMP_V2C (默认) 3 - SNMP_V3	使用此 SNMP 版本。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	community	-	string		SNMP 团体子.
	contextname	-	string		SNMPv1 和 SNMPv2 监控项需要。 SNMPv3 上下文名称.
	securityname	-	string		仅由 SNMPv3 监控项使用。 SNMPv3 安全名称.
	securitylevel	-	string	0 - NOAUTHNOPRIV (默认) 1 - AUTHNOPRIV 2 - AUTHPRIV	仅由 SNMPv3 监控项使用。 SNMPv3 安全等级。
	authprotocol	-	string	0 - MD5 (default) 1 - SHA1 2 - SHA224 3 - SHA256 4 - SHA384 5 - SHA512	仅由 SNMPv3 监控项使用。 SNMPv3 身份验证协议。 仅由 SNMPv3 监控项使用。
	authpassphrase	-	string		SNMPv3 身份验证密码短语。
	privprotocol	-	string	0 - DES (default) 1 - AES128 2 - AES192 3 - AES256 4 - AES192C 5 - AES256C	仅由 SNMPv3 监控项使用。 SNMPv3 私密协议. 仅由 SNMPv3 监控项使用。
	privpassphrase	-	string		SNMPv3 私密码短语.
	bulk	-	string	0 - NO 1 - YES (默认)	仅由 SNMPv3 监控项使用。 对 SNMP 使用批量请求。
items	对监控项元素标签值, 查看监控项 标签.	-			监控项根元素
tags		-			主机标签的根元素。
	tag	x	string		标签名称。
	value	-	string		标签值。
macros		-			宏的根元素。
	macro	x			用户宏名称。
	type	-	string	0 - TEXT (默认) 1 - SECRET_TEXT 2 - VAULT	宏类型。
	value	-	string		用户宏值。
	description	-	string		用户宏描述。
inventory		-			主机资产的根元素。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	<inventory_property>	-			单独的资产属性。 所有可用的资产的属性都列在各自的标签下, e.g. <type>, <name>, <os> (看到上面的例子)。
inventory_mode		-	`string`	-1 - DISABLED 0 - MANUAL (默认) 1 - AUTOMATIC	资产模式。
valuemaps		-			值映射的根元素。
	name	x	`string`		值映射的名称
	mapping	-			映射的根元素。
	value	x	`string`		映射的值。
	newvalue	x	`string`		映射的新值。

主机监控项标签

元素	元素属性	是否必须	类型	适用范围[1]	描述
items		-			监控项根元素。
	name	x	`string`		监控名称。
	type	-	`string`	0 - ZABBIX_PASSIVE (默认) 2 - TRAP 3 - SIMPLE 5 - INTERNAL 7 - ZABBIX_ACTIVE 10 - EXTERNAL 11 - ODBC 12 - IPMI 13 - SSH 14 - TELNET 15 - CALCULATED 16 - JMX 17 - SNMP_TRAP 18 - DEPENDENT 19 - HTTP_AGENT 20 - SNMP_AGENT 21 - ITEM_TYPE_SCRIPT	监控项类型。
	snmp_oid	-	`string`		SNMP OID. SNMP 监控项需要。
	key	x	`string`		监控项键值。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	delay	-	string	默认: 1m	监控项的更新间隔 请注意，对于 trapper 监控项，“延迟” 将始终为 “0” 接受秒或带有后缀的时间单位（30s、1m、2h、1d）。 可选一个或多个custom intervals 可以指定为灵活的时间间隔或计划 多个间隔用分号分隔。 可以适用用户宏。一个宏必须填充整个字段。不支持字段中的多个宏或与文本混合的宏。 灵活的间隔可以写成两个用正斜杠分隔的宏（例如，“{$FLEX_INTERVAL}/{$FLEX_P
	history	-	string	默认: 90d	历史数据应存储多长时间的时间单位。带有后缀、用户宏或 LLD 宏的时间单位。
	trends	-	string	默认: 365d	趋势数据应存储多长时间的时间单位。带有后缀、用户宏或 LLD 宏的时间单位。
	status	-	string	0 - ENABLED (默认) 1 - DISABLED	监控项状态。
	value_type	-	string	0 - FLOAT 1 - CHAR 2 - LOG 3 - UNSIGNED (默认) 4 - TEXT	接收值类型。
	allowed_hosts	-	string		允许发送监控项数据的主机的 IP 地址列表（逗号分隔） 由 trapper 和 HTTP 代理项使用。
	units	-	string		返回值的单位 (bps, B, etc)。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	params	-	`text`		其他参数取决于监控项的类型： 脚本、SSH 和 Telnet 监控项的已执行脚本 -SQL 查询数据库监控项 -可计算监控项的公式。
	ipmi_sensor	-	`string`		IPMI 传感器. 仅用于 IPMI 监控项。
	authtype	-	`string`	SSH agent 监控项的身份验证类型： 0 - PASSWORD (默认) 1 - PUBLIC_KEY HTTP agent 监控项的身份验证类型: 0 - NONE (默认) 1 - BASIC 2 - NTLM	身份验证类型 仅由 SSH 和 HTTP 代理项使用。
	username	-	`string`		用于身份验证的用户名。 用于简单检查, SSH, Telnet, database 监控, JMX and HTTP agent 监控项. SSH 和 Telnet 监控项所需。 当 JMX 代理使用时，密码也应该与用户名一起指定，或者两个属性都应该留空。
	password	-	`string`		验证密码。 用于简单检查, SSH, Telnet, database 监控, JMX and HTTP agent 监控项。 当 JMX 代理使用时，用户名也应该与密码一起指定，或者两个属性都应该留空。
	publickey	-	`string`		公钥文件的名称 SSH 代理监控项需要。
	privatekey	-	`string`		私钥文件的名称 SSH 代理监控项需要。
	description	-	`text`		监控项描述。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	inventory_link	-	`string`	0 - NONE 大写的主机资产字段名。例如: 4 - ALIAS 6 - OS_FULL 14 - HARDWARE 等等。	由监控项填充的主机资产字段。 参考主机资产页面 获取受支持的主机资产字段及其 ID 的列表。
	logtimefmt	-	`string`		日志监控项中的时间格式 仅由日志监控项使用。
	interface_ref	-	`string`	Format: `if<N>`	对主机接口的引用。
	jmx_endpoint	-	`string`		JMX 终端。 仅 JMX agent 监控项需要。
	url	-	`string`		URL 字符串。 仅 HTTP agent 监控项需要。
	allow_traps	-	`string`	0 - NO (默认) 1 - YES	允许像在 Traper 监控项中一样填充值。 仅 HTTP agent 监控项需要。
	follow_redirects	-	`string`	0 - NO 1 - YES (默认)	在共享数据时遵循 HTTP 响应重定向 仅 HTTP agent 监控项使用。
headers		-			HTTP（S）请求头的根元素，其中头名称用作键，头值用作值。 仅 HTTP agent 监控项使用。
	name	x	`string`		头名称
	value	x	`string`		头值
	http_proxy	-	`string`		HTTP(S) proxy 连接字符串。 仅 HTTP agent 监控项使用。
	output_format	-	`string`	0 - RAW (默认) 1 - JSON	如何处理回应。 仅 HTTP agent 监控项使用。
	post_type	-	`string`	0 - RAW (默认) 2 - JSON 3 - XML	post 数据体的类型。 仅 HTTP agent 监控项使用。
	posts	-	`string`		HTTP(S) 请求正文数据。 仅 HTTP agent 监控项使用。

元素	元素属性	是否必须	类型	适用范围[1]	描述
query_fields		-			查询参数的根元素。
					仅 HTTP agent 监控项使用。
	name	x	string		参数名称。
	value	-	string		参数值。
	request_method	-	string	0 - GET (默认) 1 - POST 2 - PUT 3 - HEAD	请求方法。
					仅 HTTP agent 监控项使用。
	retrieve_mode	-	string	0 - BODY (默认) 1 - HEADERS 2 - BOTH	应该存储响应的哪一部分
					仅 HTTP agent 监控项使用。
	ssl_cert_file	-	string		公共 SSL 密钥文件路径。
					仅 HTTP agent 监控项使用。
	ssl_key_file	-	string		私有 SSL 密钥文件路径。
					仅 HTTP agent 监控项使用。
	ssl_key_password	-	string		SSL 密钥文件的密码。
					仅 HTTP agent 监控项使用。
	status_codes	-	string		所需 HTTP 状态代码的范围，用逗号分隔。支持用户宏。 例如: 200,200-{$M},{$M},200-400
					仅 HTTP agent 监控项使用。
	timeout	-	string		监控项数据轮询请求超时。支持用户宏。
					仅 HTTP agent 监控项使用。
	verify_host	-	string	0 - NO (默认) 1 - YES	验证 URL 中的主机名是否位于主机证书的公用名字段或使用者备用名字段中。
					仅 HTTP agent 监控项使用。
	verify_peer	-	string	0 - NO (默认) 1 - YES	验证主机证书是否真实。
					仅 HTTP agent 监控项使用。

元素	元素属性	是否必须	类型	适用范围[1]	描述
parameters		-			用户定义参数的根元素。 仅 HTTP agent 监控项使用。
	name	x	string		参数名称。 仅 HTTP agent 监控项使用。
	value	-	string		参数值。 仅 HTTP agent 监控项使用。
value map		-			值映射。
	name	x	string		要用于监控项的值映射的名称。
preprocessing		-			监控项预处理的根元素。
step		-			单个监控项值预处理步骤。
	type	x	string	1 - MULTIPLIER 2 - RTRIM 3 - LTRIM 4 - TRIM 5 - REGEX 6 - BOOL_TO_DECIMAL 7 - OCTAL_TO_DECIMAL 8 - HEX_TO_DECIMAL 9 - SIMPLE_CHANGE (计算为 (接收值以前的值)) 10 - CHANGE_PER_SECOND (计算为 (接收值以前的值)/(现在时间-最近一次检查时间)) 11 - XMLPATH 12 - JSONPATH 13 - IN_RANGE 14 - MATCHES_REGEX 15 - NOT_MATCHES_REGEX 16 - CHECK_JSON_ERROR 17 - CHECK_XML_ERROR 18 - CHECK_REGEX_ERROR 19 - DISCARD_UNCHANGED 20 - DISCARD_UNCHANGED_HEARTBEAT 21 - JAVASCRIPT 22 - PROMETHEUS_PATTERN 23 - PROMETHEUS_TO_JSON 24 - CSV_TO_JSON 25 - STR_REPLACE 26 - CHECK_NOT_SUPPORTED 27 - XML_TO_JSON	监控项值预处理步骤的类型。
	parameters	-			监控项值预处理步骤参数的根元素。
	parameter	x	string		监控项值预处理步骤的单个参数。
	error_handler	-	string	0 - ORIGINAL_ERROR (默认) 1 - DISCARD_VALUE 2 - CUSTOM_VALUE 3 - CUSTOM_ERROR	预处理步骤失败时使用的操作类型。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	error_handler_params	-	string		与一起使用的错误处理程序参数'error_handler'.
master_item		-			单个监控项的被依赖主监控项。
	key	x	string		依赖监控项所需。 依赖监控项主监控项键值
triggers		-			最多允许递归 3 个依赖项，且依赖项的最大计数等于 29999。 简单触发器的根元素。
	* 有关触发器元素标记值，请参见主机触发器标签.*				
tags		-			监控项标签的根元素。
	tag	x	string		标签的名称
	value	-	string		标签值。

主机底层自动发现规则标签

元素	元素属性	是否必须	类型	适用范围[1]	描述
discovery_rules		-			低级别自动发现根元素。
	有关大多数图元标标签，请参见常规监控项的图元标签值。下面只描述特定于底层自动发现规则的标签。				
	type	-	string	0 - ZABBIX_PASSIVE (默认) 2 - TRAP 3 - SIMPLE 5 - INTERNAL 7 - ZABBIX_ACTIVE 10 - EXTERNAL 11 - ODBC 12 - IPMI 13 - SSH 14 - TELNET 16 - JMX 18 - DEPENDENT 19 - HTTP_AGENT 20 - SNMP_AGENT	监控项类型
	lifetime	-	string	默认: 30d	在该时间段之后，不再发现的监控项将被删除。支持秒，带后缀的时间单位或用户宏。
filter					单个过滤器。
	evaltype	-	string	0 - AND_OR (默认) 1 - AND 2 - OR 3 - FORMULA	用于检查底层自动发现规则筛选器条件的逻辑。
	formula	-	string		过滤条件的自定义计算公式。

元素	元素属性	是否必须	类型	适用范围[1]	描述
conditions		-			过滤条件的根元素。
	macro	x	string		底层自动发现宏名称
	value	-	string		筛选值：正则表达式或全局正则表达式。
	operator	-	string	8 - MATCHES_REGEX (默认) 9 - NOT_MATCHES_REGEX	条件操作者。
	formulaid	x	character		用于从自定义表达式引用条件的任意唯一 ID。只能包含大写字母。用户在修改过滤条件时必须定义 ID，但在之后请求时将重新生成 ID。
lld_macro_paths		-			底层自动发现宏路径根元素。
	lld_macro	x	string		底层自动发现宏名称
	path	x	string		将分配给相应宏的值的选择器。
preprocessing		-			底层自动发现值预处理。
step		-			单个底层自动发现规则值预处理步骤。
	有关大多数图元标签值，请参见主体监控值预处理的图元标签值。下面只描述特定于底层自动发现值预处理的标签。				
	type	x	string	5 - REGEX 11 - XMLPATH 12 - JSONPATH 15 - NOT_MATCHES_REGEX 16 - CHECK_JSON_ERROR 17 - CHECK_XML_ERROR 20 - DISCARD_UNCHANGED_HEARTBEAT 21 - JAVASCRIPT 23 - PROMETHEUS_TO_JSON 24 - CSV_TO_JSON 25 - STR_REPLACE 27 - XML_TO_JSON	监控项值预处理步骤的类型。
trigger_prototypes		-			触发器原型的根元素。
	有关触发器原型元素标签值，请参见常规主机触发器 标签。				
graph_prototypes		-			图形原型的根元素。
	有关图形原型元素标签值，请参见常规主机图形标签。				
host_prototypes		-			主机原型的根元素。
	有关主体原型图元标记值，请参见常规主机 标签。				

元素	元素属性	是否必须	类型	适用范围[1]	描述
item_prototypes		-			监控项原型的根元素。 有关监控项原型元素标记值，请参见常规主机监控项 标签。
master_item		-			单个监控项原型主监控项/监控项原型数据。
	key	x	string		依赖监控项原型主监控项/监控项原型键值的值. 依赖项所需。

主机触发器标签

元素	元素属性	是否必须	类型	适用范围[1]	描述
triggers		-			触发器根元素。
	expression	x	string		触发器表达式。
	recovery_mode	-	string	0 - EXPRESSION (默认) 1 - RECOVERY_EXPRESSION 2 - NONE	生成 OK 事件的基础。
	recovery_expression	-	string		触发器恢复表达式。
	name	x	string		触发器名称。
	correlation_mode	-	string	0 - DISABLED (默认) 1 - TAG_VALUE	关联模式（无事件关联或标记的事件关联）。
	correlation_tag	-	string		用于事件关联的标记名。
	url	-	string		与触发器关联的URL。
	status	-	string	0 - ENABLED (默认) 1 - DISABLED	触发器状态
	priority	-	string	0 - NOT_CLASSIFIED (默认) 1 - INFO 2 - WARNING 3 - AVERAGE 4 - HIGH 5 - DISASTER	触发器等级。
	description	-	text		触发器描述。
	type	-	string	0 - SINGLE (默认) 1 - MULTIPLE	事件生成类型(单个问题事件或多个问题事件)。
	manual_close	-	string	0 - NO (默认) 1 - YES	手动关闭问题事件。
dependencies		-			依赖项的根元素。
	name	x	string		依赖触发器名称。
	expression	x	string		依赖触发表达式。
	recovery_expression	-	string		依赖触发恢复表达式。
tags		-			事件标签的根元素。
	tag	x	string		标签名称。
	value	-	string		标签值。

主机图形标签

元素	元素属性	是否必须	类型	范围[1]	描述
graphs		-			图形的根元素。
	name	x	string		图形名称。
	width	-	integer	20-65535 (默认: 900)	图形宽度, 以像素为单位。用于预览和饼图/分解图。
	height	-	integer	20-65535 (默认: 200)	图形高度，以像素为单位。用于预览和饼图/分解图。
	yaxismin	-	double	默认: 0	Y 轴最小值。 当'ymin_type_1'为固定值时使用。
	yaxismax	-	double	默认: 0	Y 轴最大值。 当'ymax_type_1'为固定值时使用。
	show_work_period	-	string	0 - NO 1 - YES (默认)	突出非工作时间。 用于普通图和堆叠图。
	show_triggers	-	string	0 - NO 1 - YES (默认)	将简单的触发器值显示为一行。 用于普通图和堆叠图。
	type	-	string	0 - NORMAL (默认) 1 - STACKED 2 - PIE 3 - EXPLODED	图形类型。
	show_legend	-	string	0 - NO 1 - YES (默认)	显示图例。
	show_3d	-	string	0 - NO (默认) 1 - YES	开启 3De 风格。 用于饼图和分解饼图。
	percent_left	-	double	默认:0	显示左轴的百分位线。 仅用于普通图形。
	percent_right	-	double	默认:0	显示右轴的百分位线。 仅用于普通图形。
	ymin_type_1	-	string	0 - CALCULATED (默认) 1 - FIXED 2 - ITEM	Y 轴的最小值。 用于普通图和堆叠图。
	ymax_type_1	-	string	0 - CALCULATED (默认) 1 - FIXED 2 - ITEM	Y 轴最大值。 用于普通图和堆叠图。
ymin_item_1		-			个别监控项的细节。 当'ymin_type_1'为 ITEM 时需要。
	host	x	string		监控项主机。
	key	x	string		监控项键值。

元素	元素属性	是否必须	类型	范围[1]	描述
ymax_item_1		-			个别监控项的细节。 当'ymax_type_1'为 ITEM 时需要。
	host	x	string		监控项主机。
	key	x	string		监控项键值。
graph_items		x			图形监控项的根元素。
	sortorder	-	integer		画秩序。先画出较小的值。可用于在另一个线或区域后面 (或前面) 画线或区域。
	drawtype	-	string	0 - SINGLE_LINE (默认) 1 - FILLED_REGION 2 - BOLD_LINE 3 - DOTTED_LINE 4 - DASHED_LINE 5 - GRADIENT_LINE	图形监控项的绘制样式。 仅用于普通图形。
	color	-	string		元素颜色 (6 个符号，十六进制)。
	yaxisside	-	string	0 - LEFT (默认) 1 - RIGHT	将绘制图形监控项 Y 比例的图形的一边。 用于普通图和堆叠图。
	calc_fnc	-	string	1 - MIN 2 - AVG (默认) 4 - MAX 7 - ALL(最小、平均和最大; 9 - LAST(仅用于饼图和分解饼图)	如果某监控项存在多个值，则绘制的数据。
	type	-	string	0 - SIMPLE (默认) 2 - GRAPH_SUM (监控项的值代表整个饼图; 仅用于饼图和分解饼图)	监控项图形类型。
item		x			单一监控项。
	host	x	string		监控项主机。
	key	x	string		监控项键值。

主机 web 场景标签

元素	元素属性	是否必须	类型	适用范围[1]	描述
httptests		-			web 场景的根元素。
	name	x	string		Web 场景名称。
	delay	-	string	默认: 1m	执行 web 场景的频率。支持秒，带后缀的时间单位或用户宏。
	attempts	-	integer	1-10 (默认: 1)	尝试执行 web 场景步骤的次数。
	agent	-	string	默认: Zabbix	agent 客户端。Zabbix 将模拟为选中的浏览器。当一个网站为不同的浏览器返回不同的内容时，这是很有用的。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	http_proxy	-	string		指定要使用的 HTTP 代理，格式如下:http://[username[:pas
variables		-			场景步骤中可能使用的场景级变量 (宏) 的根元素。
	name	x	text		变量名称。
	value	x	text		变量值。
headers		-			执行请求时将发送的 HTTP 头的根元素。报文头应该使用与 HTTP 协议中相同的语法列出。
	name	x	text		头名称。
	value	x	text		头值。
	status	-	string	0 - ENABLED (默认) 1 - DISABLED	Web 场景状态。
	authentication	-	string	0 - NONE (默认) 1 - BASIC 2 - NTLM	认证方法。
	http_user	-	string		用于基本认证、HTTP 认证或 NTLM 认证的用户名。
	http_password	-	string		用于基本、HTTP 或 NTLM 身份验证的密码。
	verify_peer	-	string	0 - NO (默认) 1 - YES	验证 web 服务的 SSL 证书。
	verify_host	-	string	0 - NO (默认) 1 - YES	验证 web 服务证书的 Common Name 字段或 Subject Alternate Name 字段是否匹配。
	ssl_cert_file	-	string		用于客户端认证的 SSL 证书文件名 (必须为 PEM 格式)。
	ssl_key_file	-	string		用于客户端认证的 SSL 私钥文件名 (必须为 PEM 格式)。
	ssl_key_password	-	string		SSL 私钥文件密码。
steps		x			web 场景步骤的根元素。
	name	x	string		Web 场景步骤名。
	url	x	string		URL 监控。
query_fields		-			查询字段的根元素——执行请求时将添加到 URL 的 HTTP 字段数组。
	name	x	string		查询字段名。
	value	-	string		查询字段值。

元素	元素属性	是否必须	类型	适用范围[1]	描述
posts		-			HTTP POST 变量为字符串 (原始 POST 数据) 或 HTTP 字段数组 (表单字段数据)。
	name	x	string		Post 字段名。
	value	x	string		Post 字段值。
variables		-			步骤级变量 (宏) 的根元素，应该在此步骤之后应用。 如果变量值有'regex:' 前缀，那么它的值将根据'regex:' 前缀后面的正则表达式模式从这一步返回的数据中提取。
	name	x	string		变量名。
	value	x	string		变量值。
headers		-			执行请求时将发送的 HTTP 头的根元素。报头应该使用与 HTTP 协议中相同的语法列出。
	name	x	string		标题名称。
	value	x	string		标题值。
	follow_redirects	-	string	0 - NO 1 - YES (默认)	遵循 HTTP 重定向。
	retrieve_mode	-	string	0 - BODY (默认) 1 - HEADERS 2 - BOTH	HTTP 响应检索模式。
	timeout	-	string	默认: 15s	步骤执行的超时时间。秒，带有后缀或用户宏的时间单位。
	required	-	string		必须在响应中出现的文本。空值忽视。
	status_codes	-	string		被接受的 HTTP 状态码的逗号分隔列表。空值忽视。例如: 200-201,210-299
tags		-			web 场景标签的根元素。
	tag	x	string		标签名
	value	-	string		标签值。

附注

[1] 对于字符串值，只有字符串将被导出 (例如"ZABBIX_ACTIVE")，而不使用该表中使用的编号。该表中的范围值 (对应于 API 值) 的编号仅用于排序。

4 网络拓扑图

概览

网络拓扑图导出 包括:

- 所有相关的图片
- 拓扑图结构-所有拓扑图设置，所有包含的元素及其设置，拓扑图链接和拓扑图链接状态指示器

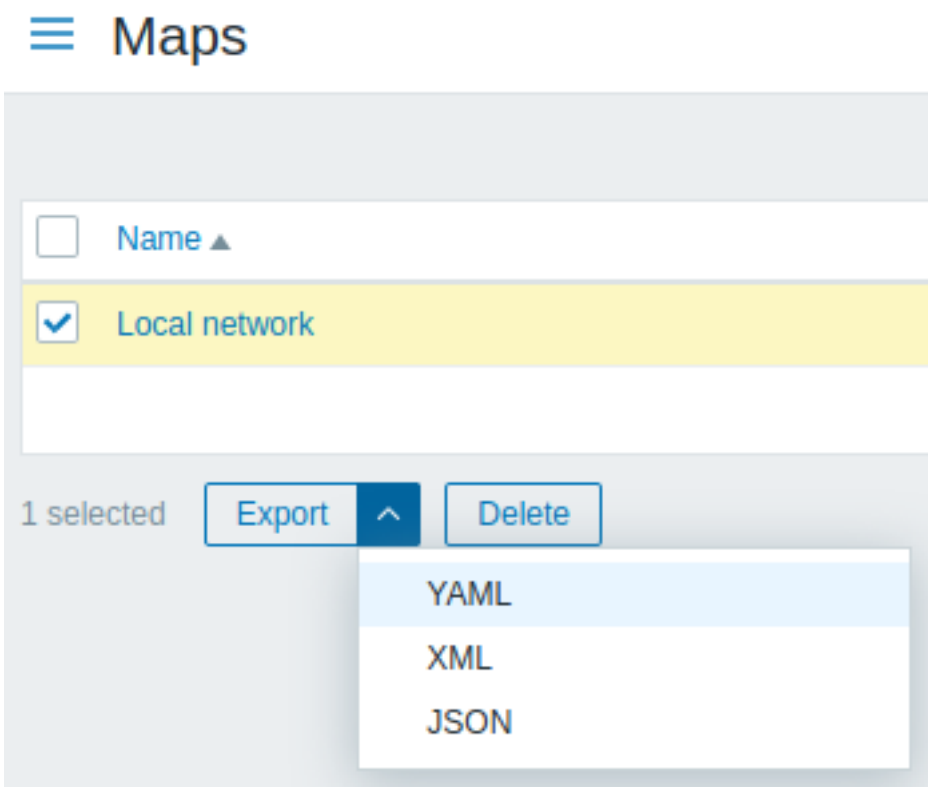
不导出与导出映射相关的主机组、主机、触发器、其他映射或其他元素。因此，如果映射引用的元素中至少有一个缺失，则导入它将失败。

网络拓扑图导出/导入支持从 Zabbix 1.8.2 开始。

导出时

要导出网络拓扑图时，请执行以下操作:

- 前往: *Monitoring* → *Maps*
- 标记要导出的网络拓扑图的复选框
- 点击列表下方的 *Export*



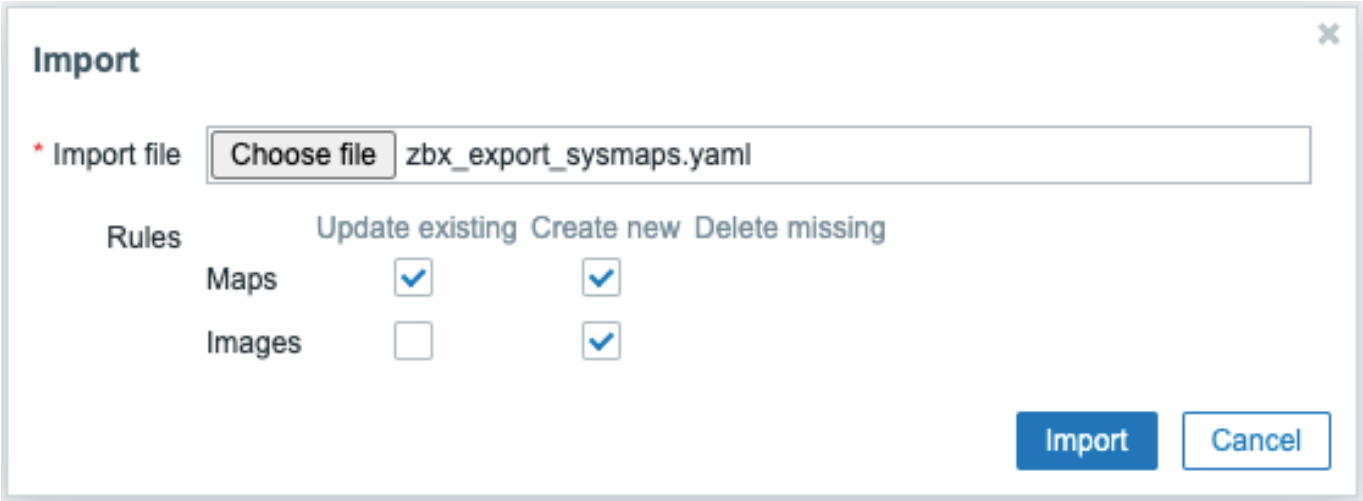
根据选择的格式，拓扑图被导出到一个默认名称的本地文件:

- *zabbix_export_maps.yaml* - 在 YAML 导出 (导出的默认选项)
- *zabbix_export_maps.xml* - 在 XML 导出
- *zabbix_export_maps.json* - 在 JSON 导出

导入时

导入网络拓扑图，请执行以下操作:

- 前往: *Monitoring* → *Maps*
- 点击右边的 *Import*
- 选择导入文件
- 在导入规则中标记所需的选项
- 点击 *Import*



所有强制输入字段都用红色星号标记。

前端将显示导入成功或失败的消息。

导入规则：

规则	描述
Update existing	现有的拓扑图将使用从导入文件中获取的数据进行更新。否则它们将不会被更新。
Create new	导入将使用导入文件中的数据添加新的拓扑图。否则它将不会添加它们。

如果您取消这两个拓扑图选项，并检查图像的相应选项，则只会导入图像。图像导入只对超级管理员用户有效。

如果替换现有的图像，将影响使用该图像的所有拓扑图

导出格式

以 YAML 格式导出:

```
zabbix_export:
  version: '6.0'
  date: '2021-08-31T12:55:10Z'
  images:
    -
      name: Zabbix_server_3D_(128)
      imagetype: '1'
      encodedImage: iVBOR...5CYII=
  maps:
    -
      name: 'Local network'
      width: '680'
      height: '200'
      label_type: '0'
      label_location: '0'
      highlight: '1'
      expandproblem: '1'
      markelements: '1'
      show_unack: '0'
      severity_min: '0'
      show_suppressed: '0'
      grid_size: '50'
      grid_show: '1'
      grid_align: '1'
      label_format: '0'
      label_type_host: '2'
      label_type_hostgroup: '2'
      label_type_trigger: '2'
      label_type_map: '2'
      label_type_image: '2'
```

```
    label_string_host: ''
    label_string_hostgroup: ''
    label_string_trigger: ''
    label_string_map: ''
    label_string_image: ''
    expand_macros: '1'
    background: {  }
    iconmap: {  }
    urls: {  }
    selements:
      -
        elementtype: '0'
        elements:
          -
            host: 'Zabbix server'
        label: |
          {HOST.NAME}
          {HOST.CONN}
        label_location: '0'
        x: '111'
        'y': '61'
        elementsubtype: '0'
        areatype: '0'
        width: '200'
        height: '200'
        viewtype: '0'
        use_iconmap: '0'
        selementid: '1'
        icon_off:
          name: Zabbix_server_3D_(128)
        icon_on: {  }
        icon_disabled: {  }
        icon_maintenance: {  }
        urls: {  }
        evaltype: '0'
    shapes:
      -
        type: '0'
        x: '0'
        'y': '0'
        width: '680'
        height: '15'
        text: '{MAP.NAME}'
        font: '9'
        font_size: '11'
        font_color: '000000'
        text_halign: '0'
        text_valign: '0'
        border_type: '0'
        border_width: '0'
        border_color: '000000'
        background_color: ''
        zindex: '0'
    lines: {  }
    links: {  }
```

元素标签

下表解释了元素标签值。

元素	元素属性	类型	适用范围	描述
images				图像的根元素。
	name	string		唯一的图像名称。

元素	元素属性	类型	适用范围	描述
	imagetype	integer	1 - image 2 - background	图像类型。
	encodedImage			Base64 编码的图像。
maps				拓扑图的根元素。
	name	string		唯一的拓扑图名称。
	width	integer		拓扑图宽度，以像素为单位。
	height	integer		拓扑图高度，以像素为单位。
	label_type	integer	0 - label 1 - host IP address 2 - element name 3 - status only 4 - nothing	拓扑图元素标签类型。
	label_location	integer	0 - bottom 1 - left 2 - right 3 - top	默认情况下拓扑图元素标签位置。
	highlight	integer	0 - no 1 - yes	为活动触发器和主机状态启用图标高亮显示。
	expandproblem	integer	0 - no 1 - yes	显示具有单个问题的元素的问题触发器。
	markelements	integer	0 - no 1 - yes	突出显示最近更改了状态的拓扑图元素。
	show_unack	integer	0 - count of all problems 1 - count of unacknowledged problems 2 - count of acknowledged and unacknowledged problems separately	问题显示。
	severity_min	integer	0 - not classified 1 - information 2 - warning 3 - average 4 - high 5 - disaster	拓扑图上默认显示的最小触发严重程度。
	show_suppressed	integer	0 - no 1 - yes	显示问题，否则将被抑制 (不显示)，因为主机维护。
	grid_size	integer	20, 40, 50, 75 or 100	拓扑图网格的单元格大小，以像素为单位，如果”grid_show=1”。
	grid_show	integer	0 - yes 1 - no	在拓扑图配置中显示网格。
	grid_align	integer	0 - yes 1 - no	自动对齐图标在拓扑图配置。
	label_format	integer	0 - no 1 - yes	使用高级标签配置。

元素	元素属性	类型	适用范围	描述
	label_type_host	integer	0 - label 1 - host IP address 2 - element name 3 - status only 4 - nothing 5 - custom label	显示主机标签，如果”label_format=1”。
	label_type_hostgroup	integer	0 - label 2 - element name 3 - status only 4 - nothing 5 - custom label	显示主机组标签，如果”label_format=1”。
	label_type_trigger	integer	0 - label 2 - element name 3 - status only 4 - nothing 5 - custom label	显示触发器标签，如果”label_format=1”。
	label_type_map	integer	0 - label 2 - element name 3 - status only 4 - nothing 5 - custom label	显示拓扑图标签，如果”label_format=1”。
	label_type_image	integer	0 - label 2 - element name 4 - nothing 5 - custom label	显示为图像标签，如果”label format=1”。
	label_string_host	string		自定义标签的主机元素，如果”label_type_host=5”。
	label_string_hostgroup	string		主机组元素的自定义标签，如果”label_type_hostgroup=5”。
	label_string_trigger	string		自定义触发器元素的标签，如果”label_type_trigger=5”。
	label_string_map	string		自定义标签的 map 元素，如果”label_type_map=5”。
	label_string_image	string		自定义标签的图像元素，如果”label_type_image=5”。
	expand_macros	integer	0 - no 1 - yes	在拓扑图配置中展开标签中的宏。
	background	id		背景图像的 ID(如果有)，如果”imagetype=2”。
	iconmap	id		图标映射的 ID(如果有)。
urls				由映射或每个映射元素使用。
	name	string		连接名。
	url	string		连接 URL。

元素	元素属性	类型	适用范围	描述
	elementtype	integer	0 - host 1 - map 2 - trigger 3 - host group 4 - image	连接所属的拓扑图项类型。
selements				
	elementtype	integer	0 - host 1 - map 2 - trigger 3 - host group 4 - image	拓扑图元素类型。
	label	string		Icon 标签。
	label_location	integer	-1 - use map default 0 - bottom 1 - left 2 - right 3 - top	
	x	integer		X 轴上的位置。
	y	integer		Y 轴上的位置。
	elementsubtype	integer	0 - single host group 1 - all host groups	元素子类型，如果”elementtype=3”。
	areatype	integer	0 - same as whole map 1 - custom size	区域大小，如果”elementsubtype=1”。
	width	integer		区域宽度，如果”areatype=1”。
	height	integer		区域高度，如果”areatype=1”。
	viewtype	integer	0 - place evenly in the area	区域布局算法，如果”elementsubtype=1”。
	use_iconmap	integer	0 - no 1 - yes	为这个元素使用图标映射。只有在拓扑图级别激活图标映射时才相关。
	selementid	id		唯一的元素记录 ID。
	evaltype	integer		标签的评估类型。
tags				问题标签 (用于主机和主机组元素)。如果给出了标签，只有这些标签的问题才会显示在拓扑图上。
	tag			标签名称。
	value			标签值。
	operator			操作者。
elements				在拓扑图 (主机、主机组、映射等) 上表示的 Zabbix 实体。
	host			
icon_off				当元素处于 “OK” 状态时要使用的图像。
icon_on				当元素处于 “问题” 状态时要使用的图像。
icon_disabled				元素被禁用时要使用的图像。
icon_maintenance				当元素处于维护状态时要使用的图像。
	name	string		唯一的图像名称。
shapes				

元素	元素属性	类型	适用范围	描述
	type	integer	0 - rectangle 1 - ellipse	形态类型。
	x	integer		形状的 X 坐标，以像素为单位。
	y	integer		以像素为单位的形状的 Y 坐标。
	width	integer		形状宽度。
	height	integer		形状高度。
	border_type	integer	0 - none 1 - bold line 2 - dotted line 3 - dashed line	形状的边框类型。
	border_width	integer		边框宽度 (以像素为单位)。
	border_color	string		用十六进制代码表示的边框颜色.
	text	string		文本内的形状。

元素	元素属性	类型	适用范围	描述
	font	integer	0 - Georgia, serif 1 - ”Palatino Linotype”, ”Book Antiqua”, Palatino, serif 2 - ”Times New Roman”, Times, serif 3 - Arial, Helvetica, sans-serif 4 - ”Arial Black”, Gadget, sans-serif 5 - ”Comic Sans MS”, cursive, sans-serif 6 - Impact, Charcoal, sans-serif 7 - ”Lucida Sans Unicode”, ”Lucida Grande”, sans-serif 8 - Tahoma, Geneva, sans-serif 9 - ”Trebuchet MS”, Helvetica, sans-serif 10 - Verdana, Geneva, sans-serif 11 - ”Courier New”, Courier, monospace 12 - ”Lucida Console”, Monaco, monospace	文本字体样式。
	font_size	integer		字体大小 (像素)。
	font_color	string		用十六进制代码表示的字体颜色。
	text_halign	integer	0 - center 1 - left 2 - right	文本的水平对齐。
	text_valign	integer	0 - middle 1 - top 2 - bottom	文本垂直对齐。

元素	元素属性	类型	适用范围	描述
	background_color	string		背景 (填充) 颜色以十六进制代码表示。
	zindex	integer		用于对所有形状和线条排序的值 (z-index)。
lines				
	x1	integer		直线点 1 的 X 坐标，单位是像素。
	y1	integer		直线点 1 的 Y 坐标，单位是像素。
	x2	integer		直线点 2 的 X 坐标，单位是像素。
	y2	integer		直线点 2 的 Y 坐标，单位是像素。
	line_type	integer	0 - none 1 - bold line 2 - dotted line 3 - dashed line	线条类型。
	line_width	integer		以像素为单位的线宽。
	line_color	string		用十六进制代码表示的线颜色。
	zindex	integer		用于对所有形状和线条排序的值 (z-index)。
links				拓扑图元素之间的链接。
	drawtype	integer	0 - line 2 - bold line 3 - dotted line 4 - dashed line	线条类型。
	color	string		连接颜色 (6 个符号，十六进制)。
	label	string		连接标记。
	selementid1	id		要连接的一个元素的 ID。
	selementid2	id		要连接的其他元素的 ID。
linktriggers				连接状态指标。
	drawtype	integer	0 - line 2 - bold line 3 - dotted line 4 - dashed line	当触发器处于 “问题” 状态时的连接样式。
	color	string		当触发器处于 “问题” 状态时，连接颜色 (6 个符号，十六进制)。
trigger				指示连路状态的触发器。
	description	string		触发器名称。
	expression	string		触发器表达式。
	recovery_expression	string		触发器恢复表达式。

5 媒介类型

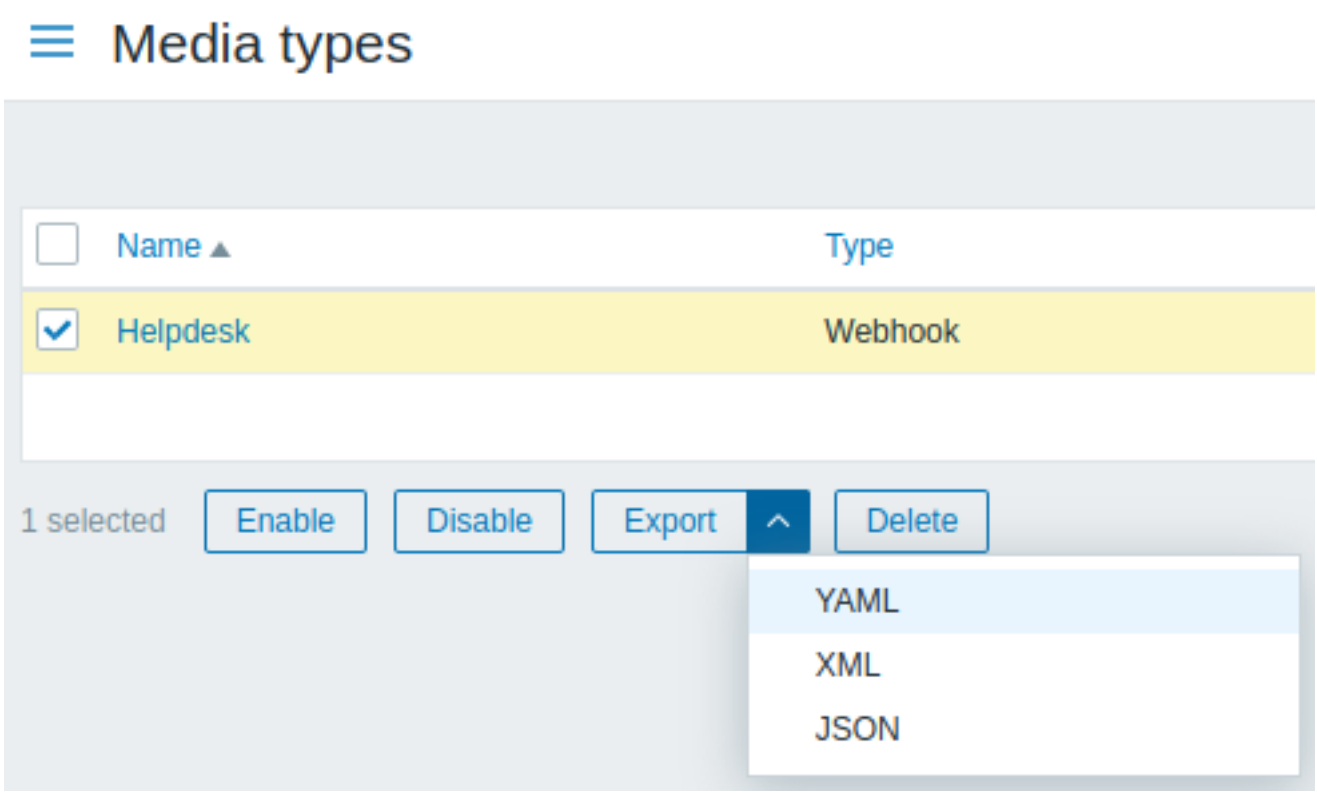
概述

媒介类型是与所有相关对象和对象关系一起导出的

导出时

导出媒体类型，请执行以下操作:

- 前往: *Administration* → *Media types*
- 标记要导出的媒介类型的复选框
- 点击列表下面的导出



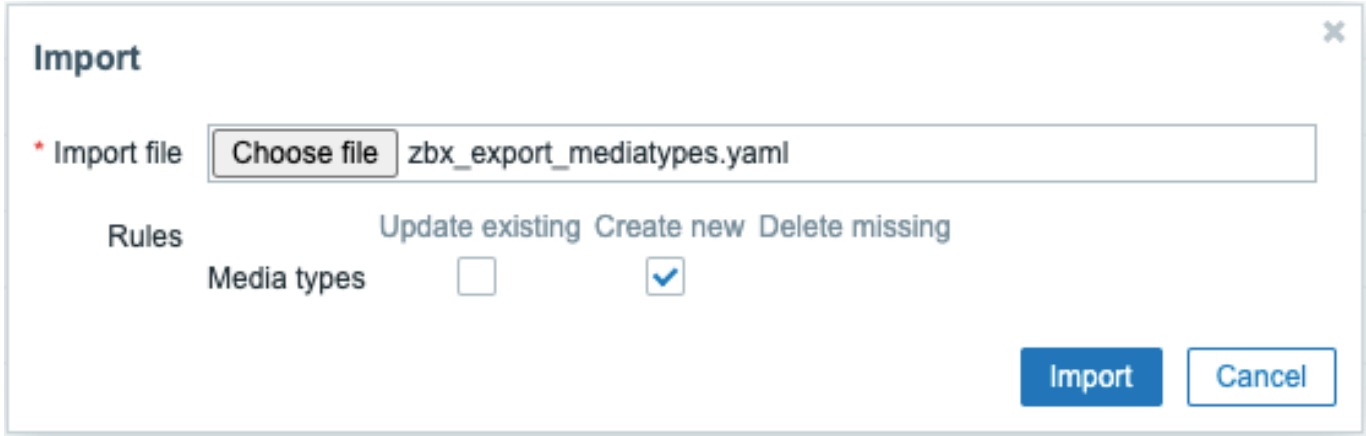
根据选择的格式，媒体类型被导出到一个默认名称的本地文件:

- *zabbix_export_mediatypes.yaml* - 在 YAML 导出 (导出的默认选项)
- *zabbix_export_mediatypes.xml* - 在 XML 导出
- *zabbix_export_mediatypes.json* - 在 JSON 导出

导入时

导入媒介类型步骤如下：

- 前往: *Administration* → *Media types*
- 点击右边的导入
- 选择要导入的文件
- 在导入规则中标记所需的选项
- 点击 导入



导入成功或失败的消息将在前端页面上显示。

导入规则:

规则	描述
Update existing	从导入文件中获取的数据更新。否则它们将不会被更新。

规则	描述
Create new	文件中的数据添加新元素。否则将不会添加它们。
Delete missing	在导入的文件中，该元素将会被删除。否则不会删除它们。

导出格式

导出到 YAML:

```yaml
zabbix_export:
  version: '6.0'
  date: '2021-08-31T13:34:17Z'
  media_types:
    -
      name: Pushover
      type: WEBHOOK
      parameters:
        -
          name: endpoint
          value: 'https://api.pushover.net/1/messages.json'
        -
          name: eventid
          value: '{EVENT.ID}'
        -
          name: event_nseverity
          value: '{EVENT.NSEVERITY}'
        -
          name: event_source
          value: '{EVENT.SOURCE}'
        -
          name: event_value
          value: '{EVENT.VALUE}'
        -
          name: expire
          value: '1200'
        -
          name: message
          value: '{ALERT.MESSAGE}'
        -
          name: priority_average
          value: '0'
        -
          name: priority_default
          value: '0'
        -
          name: priority_disaster
          value: '0'
        -
          name: priority_high
          value: '0'
        -
          name: priority_information
          value: '0'
        -
          name: priority_not_classified
          value: '0'
        -
          name: priority_warning
          value: '0'
        -
          name: retry
          value: '60'
        -
          name: title
```

```
      value: '{ALERT.SUBJECT}'
    -
      name: token
      value: '<PUSHOVER TOKEN HERE>'
    -
      name: triggerid
      value: '{TRIGGER.ID}'
    -
      name: url
      value: '{$ZABBIX.URL}'
    -
      name: url_title
      value: Zabbix
    -
      name: user
      value: '{ALERT.SENDTO}'
  max_sessions: '0'
  script: |
    try {
        var params = JSON.parse(value),
            request = new HttpRequest(),
            data,
            response,
            severities = [
                {name: 'not_classified', color: '#97AAB3'},
                {name: 'information', color: '#7499FF'},
                {name: 'warning', color: '#FFC859'},
                {name: 'average', color: '#FFA059'},
                {name: 'high', color: '#E97659'},
                {name: 'disaster', color: '#E45959'},
                {name: 'resolved', color: '#009900'},
                {name: 'default', color: '#000000'}
            ],
            priority;

        if (typeof params.HTTPProxy === 'string' && params.HTTPProxy.trim() !== '') {
            request.setProxy(params.HTTPProxy);
        }

        if ([0, 1, 2, 3].indexOf(parseInt(params.event_source)) === -1) {
            throw 'Incorrect "event_source" parameter given: "' + params.event_source + '".\nMust be 0

        if (params.event_value !== '0' && params.event_value !== '1'
            && (params.event_source === '0' || params.event_source === '3')) {
            throw 'Incorrect "event_value" parameter given: ' + params.event_value + '\nMust be 0 or 1

        if ([0, 1, 2, 3, 4, 5].indexOf(parseInt(params.event_nseverity)) === -1) {
            params.event_nseverity = '7';
        }

        if (params.event_value === '0') {
            params.event_nseverity = '6';
        }

        priority = params['priority_' + severities[params.event_nseverity].name] || params.priority_de

        if (isNaN(priority) || priority < -2 || priority > 2) {
            throw '"priority" should be -2..2';
        }
```

```
        if (params.event_source === '0' && isNaN(params.triggerid)) {
            throw 'field "triggerid" is not a number';
        }

        if (isNaN(params.eventid)) {
            throw 'field "eventid" is not a number';
        }

        if (typeof params.message !== 'string' || params.message.trim() === '') {
            throw 'field "message" cannot be empty';
        }

        data = {
            token: params.token,
            user: params.user,
            title: params.title,
            message: params.message,
            url: (params.event_source === '0')
                ? params.url + '/tr_events.php?triggerid=' + params.triggerid + '&eventid=' + params.e
                : params.url,
            url_title: params.url_title,
            priority: priority
        };

        if (priority == 2) {
            if (isNaN(params.retry) || params.retry < 30) {
                throw 'field "retry" should be a number with value of at least 30 if "priority" is set
            }

            if (isNaN(params.expire) || params.expire > 10800) {
                throw 'field "expire" should be a number with value of at most 10800 if "priority" is
            }

            data.retry = params.retry;
            data.expire = params.expire;
        }

        data = JSON.stringify(data);
        Zabbix.log(4, '[ Pushover Webhook ] Sending request: ' + params.endpoint + '\n' + data);

        request.addHeader('Content-Type: application/json');
        response = request.post(params.endpoint, data);

        Zabbix.log(4, '[ Pushover Webhook ] Received response with status code ' + request.getStatus()

        if (response !== null) {
            try {
                response = JSON.parse(response);
            }
            catch (error) {
                Zabbix.log(4, '[ Pushover Webhook ] Failed to parse response received from Pushover');
                response = null;
            }
        }

        if (request.getStatus() != 200 || response === null || typeof response !== 'object' || respons
            if (response !== null && typeof response === 'object' && typeof response.errors === 'objec
                    && typeof response.errors[0] === 'string') {
                throw response.errors[0];
            }
            else {
                throw 'Unknown error. Check debug log for more information.';
```

```
            }
        }

        return 'OK';
    }
    catch (error) {
        Zabbix.log(4, '[ Pushover Webhook ] Pushover notification failed: ' + error);
        throw 'Pushover notification failed: ' + error;
    }
description: |
  Please refer to setup guide here: https://git.zabbix.com/projects/ZBX/repos/zabbix/browse/template

  Set token parameter with to your Pushover application key.
  When assigning Pushover media to the Zabbix user - add user key into send to field.
message_templates:
  -
    event_source: TRIGGERS
    operation_mode: PROBLEM
    subject: 'Problem: {EVENT.NAME}'
    message: |
      Problem started at {EVENT.TIME} on {EVENT.DATE}
      Problem name: {EVENT.NAME}
      Host: {HOST.NAME}
      Severity: {EVENT.SEVERITY}
      Operational data: {EVENT.OPDATA}
      Original problem ID: {EVENT.ID}
      {TRIGGER.URL}
  -
    event_source: TRIGGERS
    operation_mode: RECOVERY
    subject: 'Resolved in {EVENT.DURATION}: {EVENT.NAME}'
    message: |
      Problem has been resolved at {EVENT.RECOVERY.TIME} on {EVENT.RECOVERY.DATE}
      Problem name: {EVENT.NAME}
      Problem duration: {EVENT.DURATION}
      Host: {HOST.NAME}
      Severity: {EVENT.SEVERITY}
      Original problem ID: {EVENT.ID}
      {TRIGGER.URL}
  -
    event_source: TRIGGERS
    operation_mode: UPDATE
    subject: 'Updated problem in {EVENT.AGE}: {EVENT.NAME}'
    message: |
      {USER.FULLNAME} {EVENT.UPDATE.ACTION} problem at {EVENT.UPDATE.DATE} {EVENT.UPDATE.TIME}.
      {EVENT.UPDATE.MESSAGE}

      Current problem status is {EVENT.STATUS}, age is {EVENT.AGE}, acknowledged: {EVENT.ACK.STATUS}
  -
    event_source: DISCOVERY
    operation_mode: PROBLEM
    subject: 'Discovery: {DISCOVERY.DEVICE.STATUS} {DISCOVERY.DEVICE.IPADDRESS}'
    message: |
      Discovery rule: {DISCOVERY.RULE.NAME}

      Device IP: {DISCOVERY.DEVICE.IPADDRESS}
      Device DNS: {DISCOVERY.DEVICE.DNS}
      Device status: {DISCOVERY.DEVICE.STATUS}
      Device uptime: {DISCOVERY.DEVICE.UPTIME}

      Device service name: {DISCOVERY.SERVICE.NAME}
      Device service port: {DISCOVERY.SERVICE.PORT}
```

```
      Device service status: {DISCOVERY.SERVICE.STATUS}
      Device service uptime: {DISCOVERY.SERVICE.UPTIME}
  -
    event_source: AUTOREGISTRATION
    operation_mode: PROBLEM
    subject: 'Autoregistration: {HOST.HOST}'
    message: |
      Host name: {HOST.HOST}
      Host IP: {HOST.IP}
      Agent port: {HOST.PORT}
```

元素标签

下表解释了元素标签值。

元素	元素属性	是否必须	类型	适用范围[1]	描述
media_types		-			媒体类型的根元素。
	name	x	string		媒介类型的名称。
	type	x	string	0 - EMAIL 1 - SMS 2 - SCRIPT 4 - WEBHOOK	媒体类型所使用的数据传输方式。
	status	-	string	0 - ENABLED (默认) 1 - DISABLED	是否启用媒体类型。
	max_sessions	-	integer	Possible values for SMS: 1 - (默认) 其他媒体类型的可能值:0- 100,0 - unlimited	可并行处理的警报的最大数量。
	attempts	-	integer	1-10 (默认: 3)	发送警报的最大尝试次数。
	attempt_interval	-	string	0-60s (默认: 10s) 接受带后缀的秒和时间单位。	重试间隔时间。
	description	-	string		媒体类型描述。
message_templates		-			媒体类型消息模板的根元素。
	event_source	x	string	0 - TRIGGERS 1 - DISCOVERY 2 - AUTOREGISTRATION 3 - INTERNAL	事件源。
	operation_mode	x	string	0 - PROBLEM 1 - RECOVERY 2 - UPDATE	操作模式。
	subject	-	string		信息主题。
	message	-	string		信息主体。
仅用于 e-mail 媒介类型					
	smtp_server	x	string		SMTP 服务。
	smtp_port	-	integer	默认: 25	SMTP 服务器连接的端口。
	smtp_helo	x	string		SMTP helo。
	smtp_email	x	string		将发送通知的电子邮件地址。
	smtp_security	-	string	0 - NONE (默认) 1 - STARTTLS 2 - SSL_OR_TLS	要使用的 SMTP 连接安全级别。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	smtp_verify_host	-	string	0 - NO (默认) 1 - YES	SSL 验证 SMTP 主机。如果 smtp_security 为 STARTTLS 或 SSL_OR_TLS，可选。
	smtp_verify_peer	-	string	0 - NO (默认) 1 - YES	SSL 验证 peer SMTP。如果 smtp_security 为 STARTTLS 或 SSL_OR_TLS，可选。
	smtp_authentication	-	string	0 - NONE (默认) 1 - PASSWORD	SMTP 身份验证方法使用。
	username	-	string		用户名。
	password	-	string		认证密码。
	content_type	-	string	0 - TEXT 1 - HTML (默认)	信息格式。
仅 SMS 媒介类型使用					
	gsm_modem	x	string		GSM modem 的序列号。
仅脚本媒介类型使用					
	script name	x	string		脚本名称。
parameters		-			脚本参数的根元素。
仅 webhook 媒介类型使用					
	script	x	string		脚本。
	timeout	-	string	1-60s (默认: 30s)	Javascript 脚本 HTTP 请求超时间隔。
	process_tags	-	string	0 - NO (默认) 1 - YES	是否处理返回的标签。
	show_event_menu	-	string	0 - NO (默认) 1 - YES	如果 {EVENT.TAGS.*} 在事件 _menu_url 和事件 _menu_name 字段中成功解析了，该字段表示在事件菜单中存在条目。
	event_menu_url	-	string		事件菜单入口的 URL。支持 {EVENT.TAGS.*} 宏。
	event_menu_name	-	string		事件菜单项的名称。支持 {EVENT.TAGS.*} 宏。
parameters		-			webhook 媒体类型参数的根元素。
	name	x	string		Webhook 参数名称。

元素	元素属性	是否必须	类型	适用范围[1]	描述
	value	-	`string`		Webhook 参数值。

附注

[1] 对于字符串值，只有字符串将被导出 (例如 “EMAIL”)，而不使用该表中使用的编号。该表中的范围值 (对应于 API 值) 的编号仅用于排序。

15. 发现

请使用侧边栏访问本章内容。

1 网络发现

概述

Zabbix 提供高效灵活的网络自动发现功能。

网络发现的优势：

- 加快部署速度
- 简化管理
- 在快速变化的环境中避免过度管理

Zabbix 网络发现依赖于以下信息：

- IP 地址范围
- 可用的外部服务 (FTP, SSH, WEB, POP3, IMAP, TCP 等)
- 来自 Zabbix agent 的信息 (仅支持未加密模式)
- 来自 SNMP agent 的信息

不支持：

- 网络拓扑发现

网络发现主要包含两个步骤：发现和动作。

发现

Zabbix 周期性地扫描网络发现规则中定义的 IP 地址范围。每条规则可单独配置检查频率。

注意，一条发现规则始终由一个独立的发现进程处理。一个 IP 地址范围不会拆分给多个发现进程处理。

每条规则可定义一组基于 IP 地址范围的服务检查。

Note:
发现规则中定义的每项检查各自独立执行。若其中任何一项检查未发现对应服务或检查失败，其它检查仍然正常执行。

对服务和主机 (IP) 的每次检查都会产生一个发现事件。

事件	服务检查结果
发现服务	服务从停止 (down) 状态到已启动 (up) 状态，或服务首次被发现。
服务已启动 *(up)*	服务保持已启动 (up) 状态。
服务丢失 *(lost)*	服务从已启动 (up) 状态到停止 (down) 状态。
服务停止 *(down)*	服务保持停止 (down) 状态。
发现主机	主机从全部服务停止 (down) 状态到至少一个服务是已启动 (up) 状态，或发现一个未注册的主机的服务。
主机已启动 *(up)*	主机至少有一个服务保持已启动 (up) 状态。
主机丢失 *(lost)*	主机从至少有一个服务已启动 (up) 状态到所有服务停止 (down) 状态。
主机停止 *(down)*	主机的所有服务保持停止 (down) 状态。

动作

发现事件可触发相关动作，例如：

- 发送通知
- 添加/删除主机
- 启用/禁用主机
- 添加主机到组
- 从组中移除主机
- 链接/取消链接主机到模板
- 执行远程脚本

这些动作可基于设备类型、IP、状态、运行时长 (uptime)/停机时长 (downtime) 等条件来配置。欲了解如何配置这些动作，参阅动作的操作 和条件 页面获取完整细节。

Note:
通过网络发现添加的主机，如果其链接的模板中有一个唯一属性 (如监控项的键) 与该主机上的唯一属性相同，或与该模板链接的其它主机上的唯一属性相同，则这台主机上的所有模板均会链接失败。

创建主机

选择 添加主机操作来创建主机。但无需 添加主机操作一样可以创建主机，只要该操作对主机造成了变更即可。此类操作有：

- 启用主机
- 禁用主机
- 添加主机到组
- 链接模板到主机

新建的主机会添加到 发现的主机组 (默认设置，也可到 管理 → 通用 → 其它界面自定义)。如果想更改主机的主机组，可以执行 从主机组中移除的操作 (选定" 发现的主机" 组) 并执行 添加到主机组的操作 (选定另一个主机组)，因为一个主机必须属于某个主机组。

命名主机

当添加主机时，主机名是反向 DNS 解析的结果，如果解析失败，则主机名设置为 IP 地址。如果 Zabbix server 执行网络发现，则解析就在 Zabbix server 上执行，如果 Zabbix proxy 执行网络发现，则解析在 Zabbix proxy 上执行。如果在 proxy 上解析失败，不会再到 Zabbix server 上做解析。如果同名主机已经存在，新发现的主机会在名字后面添加 **_2** 后缀，后续新发现的同名主机的后缀数字依次增加。

可以使用监控项的主机名覆盖 DNS/IP 解析的主机名，比如：

- 可以使用安装在服务器上的 Zabbix agent，通过 agent 的监控项来发现多台主机并自动给这些主机分配合适的主机名，主机名取决于监控项返回的字符串值
- 可以使用 SNMP agent 监控项发现多台网络设备并自动分配合适的主机名，主机名取决于监控项返回的字符串值

如果已经使用了监控项的返回值作为主机名，则接下来的网络发现不会更新主机名。如果不使用监控项的返回值作为主机名，则使用默认值 (DNS 名称)。

如果新发现的主机 IP 地址已经存在，那么不会创建该主机。然而如果发现动作中包含添加模板、添加到主机组等操作，则会在现有的主机上执行相应操作。

删除主机

如果发现的主机不再属于发现规则定义的 IP 地址范围内，则该主机自动从 监控 → 网络发现中删除。主机会立刻被删除。

创建主机接口

当通过网络发现规则添加主机时，主机接口遵循以下原则创建：

- 检测到的服务 - 比如，如果一个 SNMP 服务被检测到，则会创建一个 SNMP 接口
- 如果一个主机同时对 Zabbix agent 和 SNMP 请求做出响应，则会同时创建两种接口 (Zabbix agent 接口和 SNMP 接口)
- 如果设备唯一标识设置为 Zabbix agent 或 SNMP，则会创建所发现的第一个接口并将其设置为默认接口。发现的其它 IP 地址会设置为额外的接口。
- 如果主机只对 Zabbix agent 作出响应，则只有 agent 接口会被创建。如果主机以后对 SNMP 请求做出响应，则额外的 SNMP 接口会被创建。
- 如果通过网络发现规则创建三台独立的主机，并且规则中配置了设备唯一标识为"IP"，但该规则后续被修改了，并且主机 A、B 和 C 通过设备唯一标识计算出相同的结果 (相同 IP 地址)，则 B 和 C 会变成第一台主机 (主机 A) 的额外接口。主机 B 和主机 C 会保留。添加的接口会显示在监控 → 网络发现界面的" 发现的设备" 一栏，用黑体字和缩进标识，但" 监控的主机" 一栏只显示第一个创建的主机 A。不会对额外接口的 IP 地址检测" 运行时长/停机时长"(Uptime/Downtime)。

更改 proxy 设置

通过不同的 proxy 发现的主机始终被视为不同的主机。即便这种操作可以对不同的子网执行自动发现，但是为一个已纳入监控的子网替换 proxy 也很复杂，因为 proxy 的变更会同时应用到所有已发现的主机上。

比如下面在发现规则中替换 proxy 的步骤：

1. 禁用规则
2. 同步 proxy 配置
3. 替换规则中的 proxy
4. 替换此规则发现的所有主机的 proxy
5. 启用规则

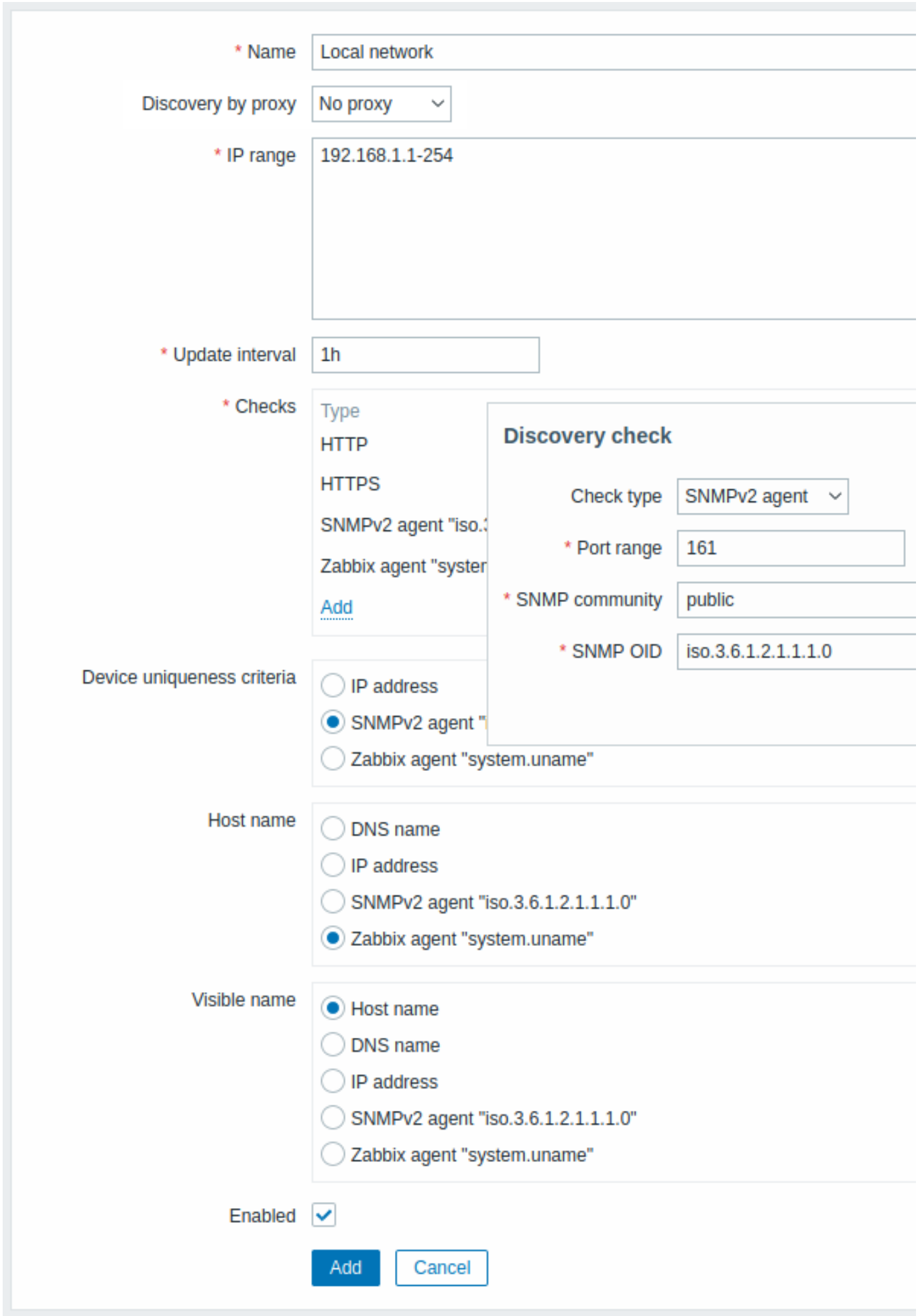
1 配置网络发现规则

概述

配置用于发现主机和服务的网络发现规则：

- 找到 配置 → 网络发现
- 点击 创建规则 (或点击规则名称，编辑现有规则)
- 编辑规则属性

规则属性



所有强制输入区域会标记为红色星号。

参数	描述
名称	唯一的规则名称。比如，”Local network”。
执行发现的 *proxy*	哪个组件执行网络发现: 没有 **proxy** - Zabbix server 执行网络发现 **<proxy** 名称 **>** - 此 proxy 执行网络发现
IP 地址范围	需要发现的 IP 地址范围。包括下列格式: 单 IP: 192.168.1.33 IP 地址范围: 192.168.1-10.1-255。IP 地址范围取决于包含的所有地址数量 (小于 64K)。 IP 子网掩码: 192.168.4.0/24 支持的 IP 掩码: /16 - /30 用于 IPv4 地址 /112 - /128 用于 IPv6 地址 列表: 192.168.1.1-255, 192.168.2.1-100, 192.168.2.200, 192.168.4.0/24 从 Zabbix 3.0.0 起支持空格、制表符和多行。
更新间隔	此参数定义了规则执行的时间间隔。 前一次发现任务执行完毕才会开始时间间隔计时，所以不会有重叠。 从 Zabbix 3.4.0 开始支持时间后缀，例如 30s, 1m, 2h, 1d。 从 Zabbix 3.4.0 开始支持用户宏。 注意如果使用了用户宏并且其值改变 (比如 1w → 1h)，下次检查会基于上一个值进行计算。(如果时间拉到很远的未来，会以示例值计算)。
检查	Zabbix 会使用此检查列表来执行网络发现。点击 Add 在弹窗中配置一个新检查。 支持的检查: SSH, LDAP, SMTP, FTP, HTTP, HTTPS, POP, NNTP, IMAP, TCP, Telnet, Zabbix agent, SNMPv1 agent, SNMPv2 agent, SNMPv3 agent, ICMP ping. 基于协议的自动发现使用 **net.tcp.service[]** 功能来检测每台主机，查询 SNMP OID 的 SNMP 服务不在此列。通过查询未加密模式的监控项来探测 Zabbix agent 的存在。请查阅agent 监控项获取更多信息。 ’ 端口’ 参数的格式如下： 单一端口: 22 端口范围: 22-45 列表: 22-45,55,60-70
设备唯一标识	唯一标识可以设置为： **IP** 地址 - 不对多台相同 IP 的设备做处理。如果一台相同 IP 的设备已经存在，则该设备视为已经发现并且不会添加新设备。 **<** 发现检查 **>** - 选择 Zabbix agent 或 SNMP agent 检查。
主机名称	设置主机的技术名称： **DNS** 名称 - DNS 名称 (默认) **IP** 地址 - IP 地址 **<** 发现检查 **>** - 通过发现检查收到的字符串值 (比如 Zabbix agent, SNMP agent 检查) 另请参阅: 主机命名. 从 4.2.0 版本起支持此选项。
可见名称	设置主机的可见名称: **Host name** - 技术主机名称 (默认) **DNS** 名称 - DNS 名称 **IP** 地址 - IP 地址 **<** 发现检查 **>** - 通过发现检查收到的字符串值 (比如 Zabbix agent, SNMP agent check) 另请参阅: 主机命名. 从 4.2.0 版本起支持此选项。
启用	勾选检查框启用规则。 如果取消勾选，则规则变成未启用状态且不会执行。

一个真实场景

在这个例子中，会演示如何给本地 IP 地址范围 192.168.1.1-192.168.1.254 设置网络发现规则。

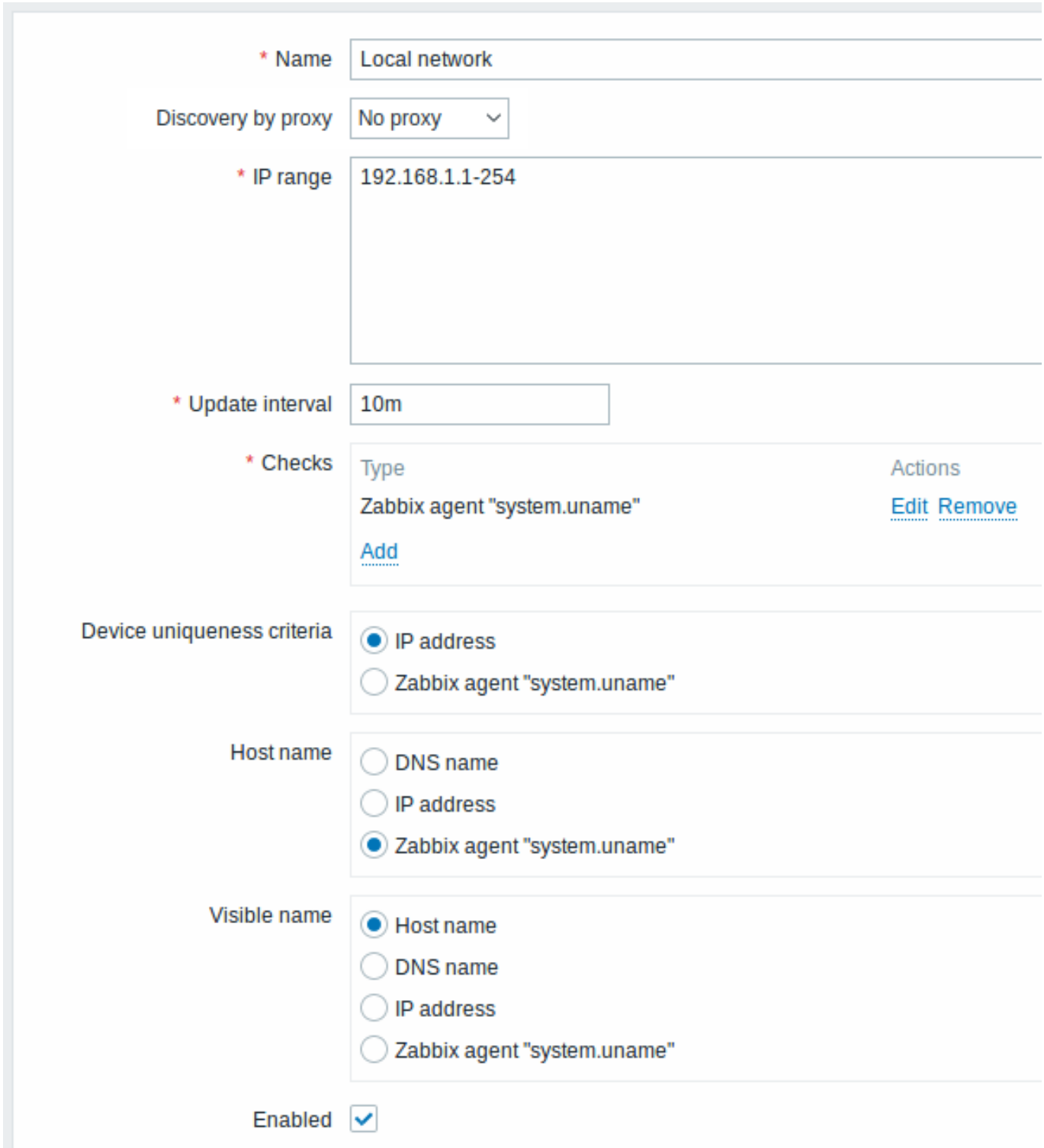
需要在这个场景中实现：

- 发现运行了 Zabbix agent 的主机

- 每十分钟执行一次发现
- 如果主机运行时长大于一小时，添加该主机到监控中
- 如果主机停机时长大于二十四小时，则删除主机
- 添加 Linux 主机到”Linux 服务器” 组中
- 添加 Windows 主机到”Windows 服务器” 组中
- 给 Linux 主机使用 *Linux* 模板
- 给 Windows 主机使用 *Windows* 模板

第一步

给 IP 地址范围定义网络发现规则。



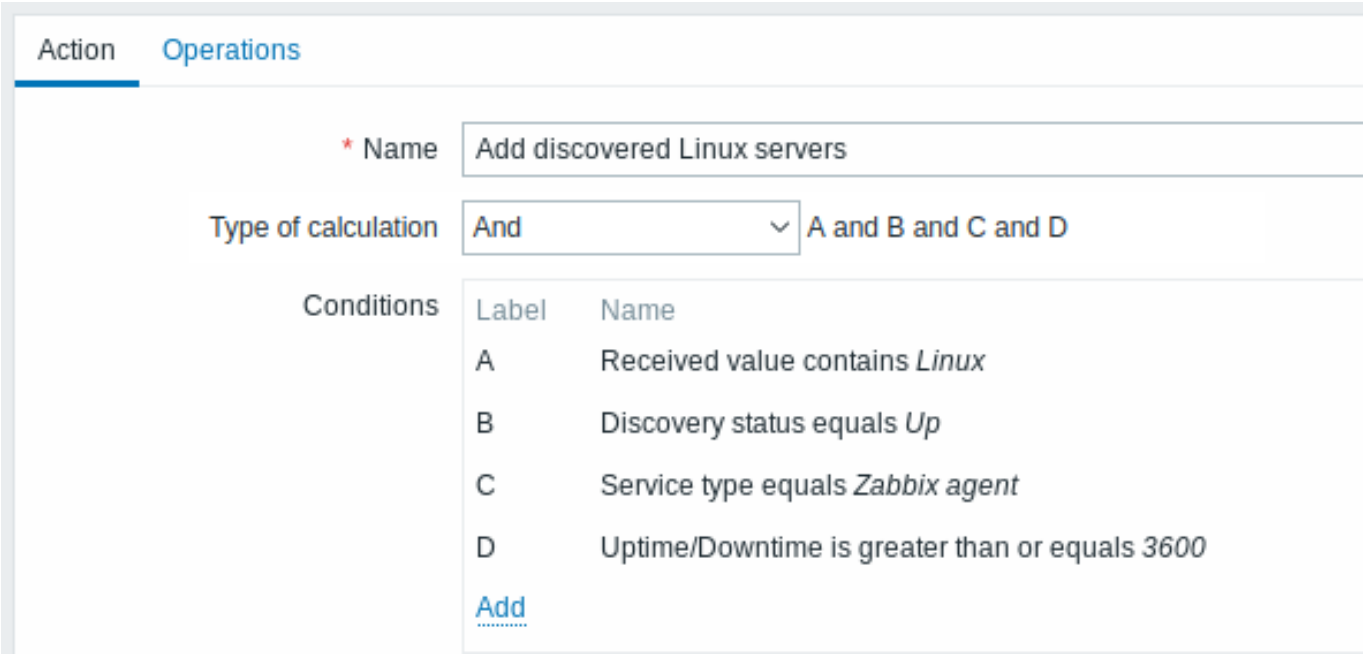
主机的发现机制是通过尝试连接主机上的 Zabbix agent 并获取 **system.uname** 这个键来实现的。从 Zabbix agent 获取到的值可以用于给主机命名以及为不同操作系统执行不同的动作。比如链接 Windows 服务器到 *Windows* 模板，链接 Linux 服务器到 *Linux* 模板。

规则每十分钟执行一次。

添加规则后，发现会自动执行，同时会产生相关事件。

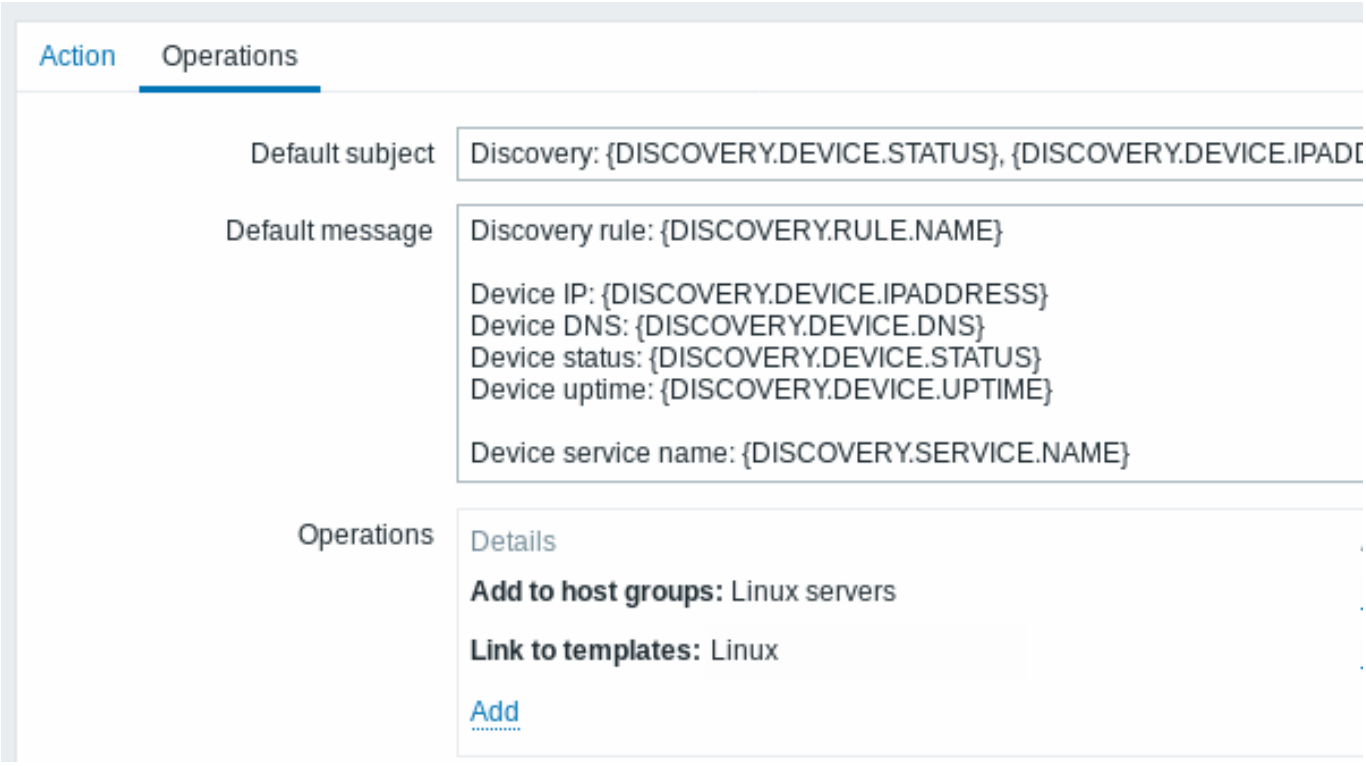
第二步

定义一个网络发现动作 ，用于添加发现的 Linux 服务器到对应的组并链接到对应模板。



同时满足下列条件才会使动作生效:

- ”Zabbix agent” 服务是已启动 (up) 状态的
- system.uname(规则里定义的 Zabbix agent 的键) 的值包含”Linux”
- 运行时长 (uptime) 大于 1 小时 (3600 秒)

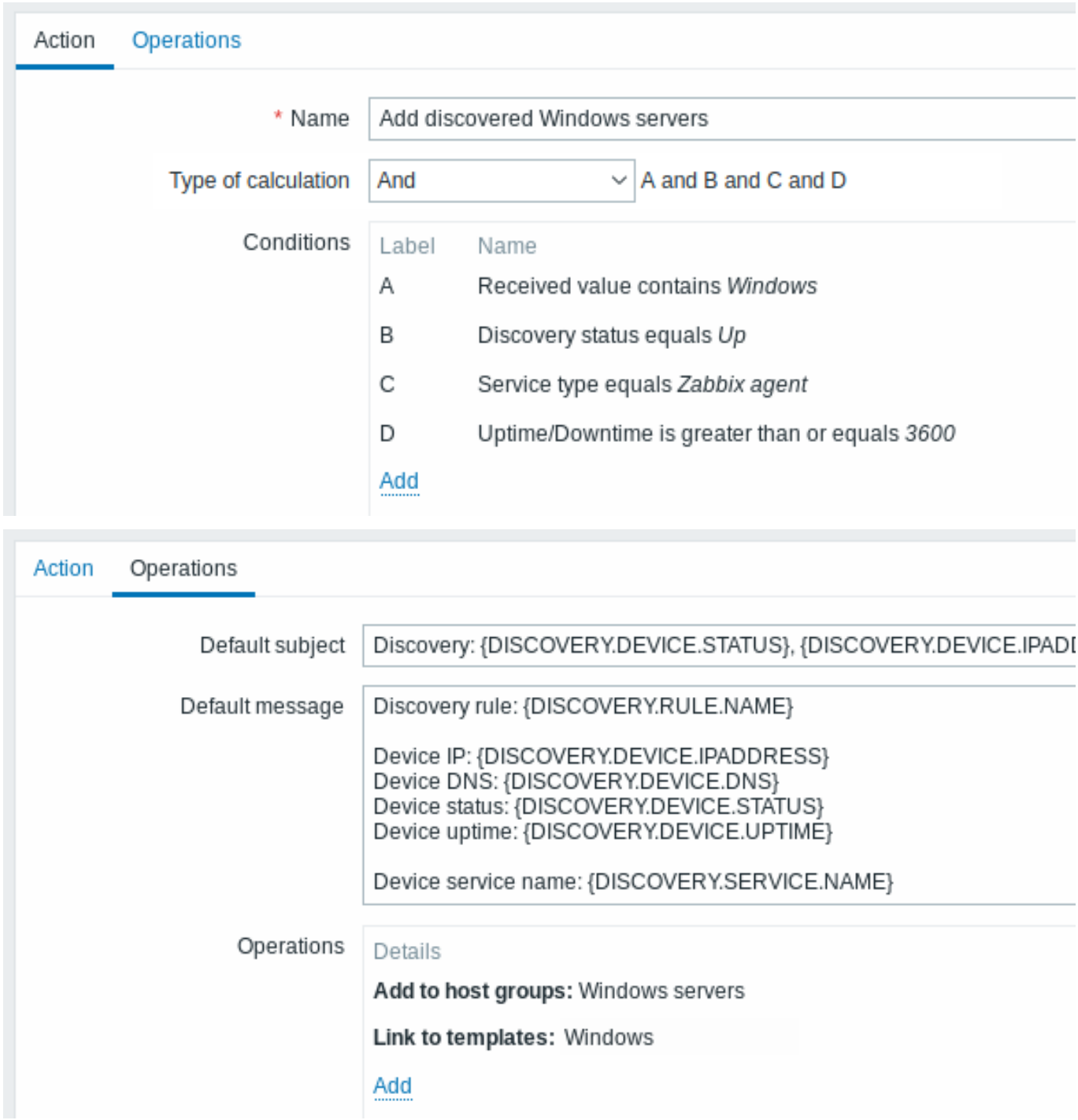


动作会执行下列操作:

- 添加发现的主机到”Linux 服务器” 组 (如果之前主机不存在，则同时创建主机)
- 链接主机到 *Linux* 模板。主机会自动纳入监控，监控使用”Linux” 模板中的监控项和触发器

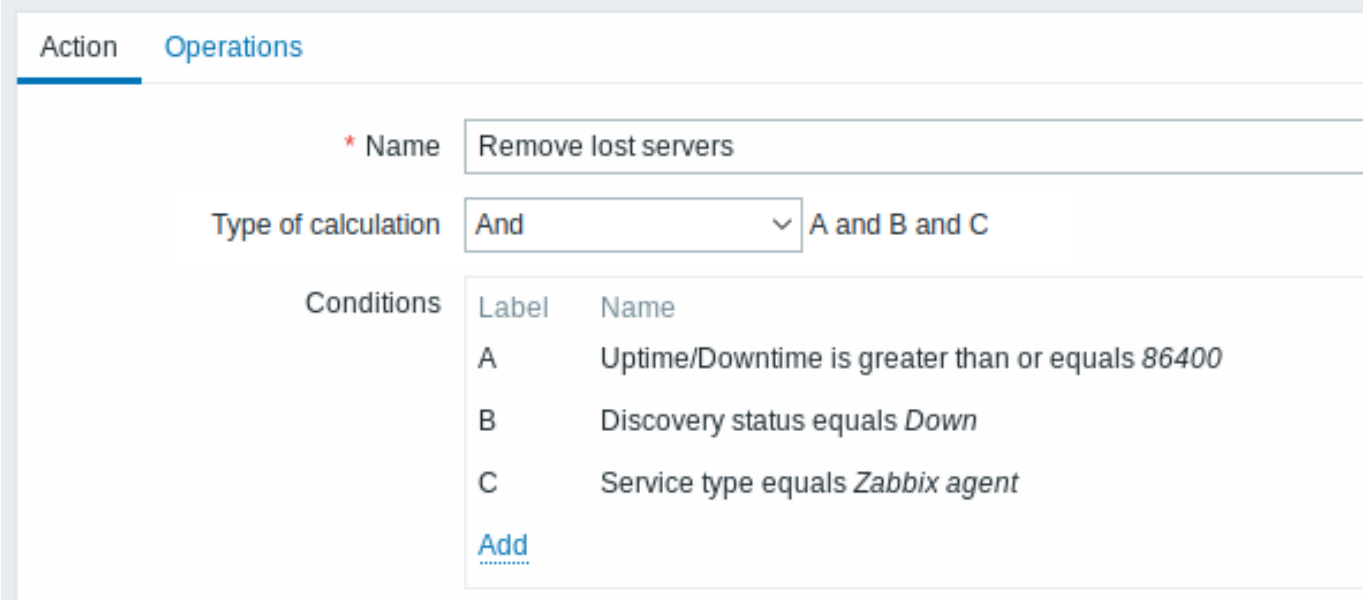
第三步

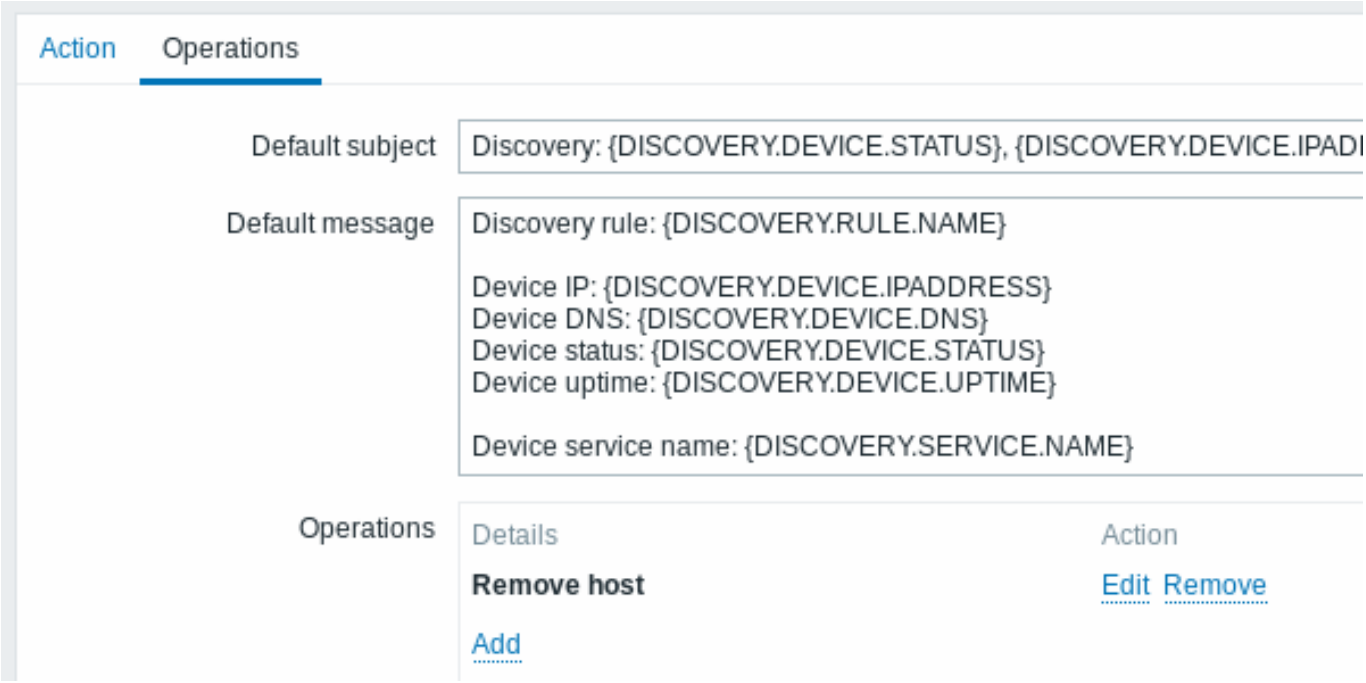
定义网络发现动作，添加发现的 Windows 服务器到对应组并链接到对应模板。



第四步

定义网络发现动作，删除丢失的 (lost) 主机。





如果”Zabbix agent” 服务停止 (down) 时间超过 24 小时 (86400 秒)，则会删除对应主机。

2 agent（主动模式）自动注册

概述

可以使用 Zabbix agent(主动模式) 自动注册功能来添加主机，自动注册后 Zabbix server 就可以开始进行监控了。通过这种方式添加的主机不必再手工配置。只要有未知的主动模式 agent 请求检查，就会触发自动注册。

该特性对于云上新增节点的自动监控很方便。一旦云上新增了一个节点，Zabbix 就开始自动收集该节点的性能和可用性数据。

对于通过 zabbix agent(被动模式) 添加的主机，agent（主动模式）自动注册同样可以监控它们。agent（主动模式）请求检查时，会将配置文件中定义的监听 IP(ListenIP) 或监听端口 (ListenPort) 参数一同发送给 Zabbix server。

注册新主机时，Zabbix server 使用收到的 IP 地址和端口信息来配置 agent。如果没收到 IP 地址信息，则使用入站连接中的对端 IP 地址。如果没收到端口信息，则使用 10050 作为 agent 的端口。

可以使用DNS 名称作为默认 agent 接口来进行自动注册。

以下情况会重新执行自动注册:

- 主机元数据 信息变更:
  - 由于 HostMetadata 信息变更并且 agent 重启

    - 由于 HostMetadataItem 信息变更
- 手动添加的主机丢失元数据信息
- 已注册的主机手动迁移到另一个 Zabbix proxy 上
- 从一个新的 Zabbix proxy 收到同一台主机的自动注册信息

配置

定义 Zabbix server

确保在配置文件 - zabbix_agentd.conf 中定义了 Zabbix server。

```
ServerActive=10.0.0.1
```

如果不在 zabbix_agentd.conf 中定义 主机名称 *(Hostname)*，则 Zabbix 会使用 agent 所在系统的主机名称给主机命名。Linux 系统的主机名称可以通过运行命令'hostname' 来获取。

如果定义的 主机名称 *(Hostname)* 是以逗号分隔的多台主机，那么其中定义的所有主机均会被创建。

agent 配置变更需要重启 agent 生效。

active agent 自动注册动作

一旦收到从 agent 发来的自动注册请求，Zabbix server 会调用动作。为了实现 agent 自动注册，必须配置" 自动注册" 这个事件源的动作。

> **Note:**
> 设置网络发现无需 active agent 自动注册参与。

在 Zabbix 界面上, 转到 配置 → 动作, 选择 自动注册作为事件源并点击 创建动作:

- 在动作 (Action) 选项卡中, 给动作设置一个名称
- 条件为可选设置。可以对主机名称/主机元数据设置字符串匹配或正则表达式匹配。如果使用" 主机元数据" 作为条件，参照下一小节。
- 在操作 (Operations) 选项卡中，添加相关操作，比如 - ' 添加主机'，' 添加主机组' (比如 发现的主机这个组)，' 链接到模板' 等。

> **Note:**
> 如果待自动注册的主机只支持主动模式 (active) 监控 (比如主机防火墙策略阻挡了 Zabbix server 向 agent 发起请求)，那么可以为主机创建这样的模板 *Template_Linux-active*。

新建的主机会添加到 发现的主机组 (默认可在这里配置：管理 → 通用 → 其它 如果想添加主机到其它组，可以先执行一个 从主机组中移除操作 (指定主机组" 发现的主机")，然后再执行 添加到主机组操作 (指定另一个主机组)，因为一个主机必须属于一个主机组。

自动注册的安全性考量

一种安全的自动注册方式是通过配置基于预共享密钥 (PSK) 的认证对通信加密。

可在 管理 → 通用 的右侧下拉菜单中的自动注册面板中配置加密级别，全局生效。可以选不加密，使用预共享密钥 (PSK) 的 TLS 加密或以上两者同时启用 (这样就会让一些主机注册时不经过加密，另一些主机经过加密方式注册)。

Zabbix server 新增主机前会对预共享密钥 (PSK) 进行核实。一旦核实成功，主机就会被添加，并且 从主机发起的/到主机的通信连接 设置为仅使用'PSK' 并且其身份/预共享密钥与全局的自动注册配置中设置的相同。

> **Attention:**
> 为确保使用 Zabbix proxy 时自动注册的安全性，Zabbix server 和 proxy 之间应该启用加密。

使用 DNS 作为默认接口

在自动注册过程中，主机接口 (HostInterface) 和主机接口监控项 (HostInterfaceItem) 配置参数的值允许自定义。

具体来说，如果主机使用 DNS 名称而不是 IP 地址作为默认 agent 接口进行自动注册时，对参数的值进行自定义就发挥作用了。这种情况下，DNS 名称应通过 HostInterface 或 HostInterfaceItem 参数进行设置。注意，如果上述参数的值发生改变，自动注册的主机接口也会更新。所以可以通过更换 DNS 名称来更新默认接口，或给 DNS 名称改成 IP 地址来更新接口。Zabbix agent 需要重启来使变更生效。::: noteclassic 如果不配置 HostInterface 和 HostInterfaceItem 这两个参数，那么 listen_dns 参数会从 IP 地址来解析。如果解析配置错误，可能会因为主机名称无效导致自动注册失败。

使用主机元数据

agent 会将主机名称和自动注册请求一同发送到 Zabbix server 上。某些情况下 (比如亚马逊云主机) 无法通过主机名称来区分发现的主机。agent 可以通过发送主机元数据这个可选配置来提供其它信息。

主机元数据在 agent 的配置文件 - zabbix_agentd.conf 中定义。有两种方式来设置主机元数据:

HostMetadata
HostMetadataItem

详情请参阅上述链接。

Attention:
每次 active agent 发送请求到 Zabbix server 刷新检查时，会尝试进行自动注册。请求之间的延迟在 agent 的RefreshActiveChecks 参数中定义。agent 重启后会立刻发送第一个请求。

例 1

使用主机元数据区分 Linux 和 Windows 主机。

假设你想让主机自动注册到 Zabbix server。网络中有配置了 active agent(参阅上述" 配置" 章节) 的 Windows 和 Linux 主机，并且 Zabbix 页面中有"Linux by Zabbix agent" 和"Windows by Zabbix agent" 这两个可用模板。于是你想在注册过程中将 Linux/Windows 模板自动应用到对应主机。在自动注册过程中，默认只有主机名称会发送给 Zabbix server，可是这些信息并不够。要想确保合适的模板应用到对应主机上，需要使用主机元数据。

前端页面配置

首先需要做的就是配置前端页面。新建两个动作。第一个动作:

- 名称: Linux 主机自动注册
- 条件: 主机元数据包含 *Linux*
- 操作: 链接到模板: Linux

Note:
这个例子中可以跳过" 添加主机" 操作这个步骤。虽然链接到模板需要先添加主机，但 Zabbix server 会自动完成添加主机操作。

第二个动作:

- 名称: Windows 主机自动注册
- 条件: 主机元数据包含 *Windows*
- 操作: 链接到模板: Windows

Agent 配置

现在需要配置 agent 了。添加下面一行到 agent 配置文件中:

```
HostMetadataItem=system.uname
```

通过这种方式能确保主机元数据包含"Linux" 或者"Windows" 其中一个值 (取决于 agent 所在的主机操作系统)。主机元数据的例子如下:

```
Linux: Linux server3 3.2.0-4-686-pae #1 SMP Debian 3.2.41-2 i686 GNU/Linux
Windows: Windows WIN-OPXGGSTYNHO 6.0.6001 Windows Server 2008 Service Pack 1 Intel IA-32
```

对配置文件做了变更后别忘记重启 agent。

例 2

第一步

使用主机元数据实施一些基本的保护措施，禁止非预期的主机进行注册。

前端页面配置

在前端页面上新建一个动作，使用难以猜测的密码来阻挡非预期的主机来注册:

- 名称: Linux 自动注册动作
- 条件:
  * 计算类型: AND
  * 条件 (A): 主机元数据包含 //Linux//
  * 条件 (B): 主机元数据包含 //21df83bf21bf0be663090bb8d4128558ab9b95fba66a6dbf834f8b91ae5e08ae//
* 操作:
  * 发送信息给用户: 通过所有媒介发送给Admin
  * 添加到主机组: Linux servers
  * 链接到模板: Linux

请注意单独使用此方法不会提供很强的保护，因为数据通过明文传输。需要重新加载配置缓存以使变更生效。

Agent 配置

添加下面一行到 agent 配置文件中:

```
HostMetadata=Linux    21df83bf21bf0be663090bb8d4128558ab9b95fba66a6dbf834f8b91ae5e08ae
```

”Linux” 指的是操作系统平台，剩下的字符串是难以猜测的密文.

对配置文件做了变更后别忘记重启 agent。

第二步

可以为已经注册的主机添加额外的监控内容。

前端页面配置

更新前端页面中的动作:

- 名称: Linux 自动注册动作
- 条件:
  * 计算类型: AND
  * 条件 (A): 主机元数据包含 Linux
  * 条件 (B): 主机元数据包含 21df83bf21bf0be663090bb8d4128558ab9b95fba66a6dbf834f8b91ae5e08ae
* 操作:
  * 发送信息给用户: 通过所有媒介发送给Admin
  * 添加到主机组: Linux servers
  * 链接到模板: Linux
  * 链接到模板: MySQL by Zabbix Agent

Agent 配置

更新下面一行到 agent 配置文件中:

```
HostMetadata=MySQL on Linux 21df83bf21bf0be663090bb8d4128558ab9b95fba66a6dbf834f8b91ae5e08ae
```

对配置文件做了变更后别忘记重启 agent。

3 底层自动发现

概述　底层自动发现可以自动为主机上的不同实体创建监控项、触发器和图表。比如，Zabbix 可以自动监控文件系统和网络接口，并且不需要为其手工创建监控项。另外，可以基于周期性自动发现的结果来删除无用的监控实体。

用户可以基于特定的 JSON 格式来自定义发现类型。

自动发现的大体流程如下：

首先，用户在” 配置” → ” 模板” → ” 自动发现” 一栏中创建发现规则。发现规则包含用来发现特定实体 (如文件系统和网络接口) 的 (1) 监控项以及监控项、触发器和图表的 (2) 原型。

用于自动发现的监控项与常规监控项没有大体区别：Zabbix server 向 agent(或者其它任何类型的监控代理) 请求监控项的值，agent 对其回复一个文本值。唯一区别是 agent 响应中包含所发现的一系列 JSON 数组。对自动发现检查规则有自定义的需求时才需要了解格式的细节方面，不过有必要知道返回值包含一系列宏 → 值的键值对。比如，监控项”net.if.discovery” 会返回两组键值对：”{#IFNAME}” → ”lo” 和”{#IFNAME}” → ”eth0”。

当创建实际的监控项、触发器、图表以及主机时，宏会替换成接收到的值。使用底层自动发现 (LLD) 的宏请参考选项 中的全部列表。

当通过自动发现收到监控项返回值的时候，Zabbix server 会查找宏 → 值键值对，每一对宏 → 值会基于原型生成监控项、触发器和图表。在上述”net.if.discovery” 的例子中，为环回接口”lo” 生成了一组监控项、触发器和图表，为”eth0” 接口生成了一组监控项、触发器和图表。

注意，自从 **Zabbix 4.2** 开始，底层自动发现返回的 JSON 数组的格式变化了。JSON 格式将不再包含”data” 对象。现在底层自动发现接收一个普通的 JSON 数组，用于实现一些新特性，如监控项值的预处理和对于 JSON 文件中用于底层自动发现的宏的路径自定义。

内置的自动发现的键已经可以实现在 JSON 文件的根路径返回一个底层自动发现数组。如果数组使用 {#MACRO} 作为键，则宏和值会自动提取。任何新版的内置的自动发现检查会使用不含”data” 元素的新语法格式。当处理一个底层自动发现的值时，第一个步骤就是定位根目录 (数组的根目录是 `$.` 或 `$.data`)。

当”data” 元素从所有跟自动发现有关的内置监控项中移除时，为了向后兼容，Zabbix 会接受带有”data” 元素的 JSON 格式，不过并不鼓励这么用。如果 JSON 数据包含一个对象，该对象只有一个”data” 数组，则”data” 数组的内容会通过 JSONPath `$.data` 自动提取出来。底层自动发现现在接受可选的自定义宏，可在 JSONPath 语法中自定义路径。

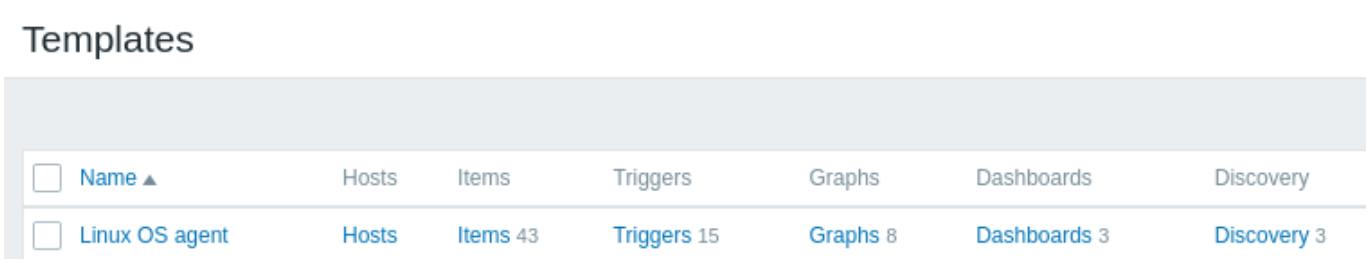
Warning:
上述变更会导致新的 agent 不再跟旧的 Zabbix server 有任何关联。

参阅: 发现的实体

配置底层自动发现　下面是一个文件系统自动发现的例子。

要配置自动发现, 需完成下列操作:

- 找到: 配置 → 模板或 主机
- 在对应模板/主机中点击 自动发现



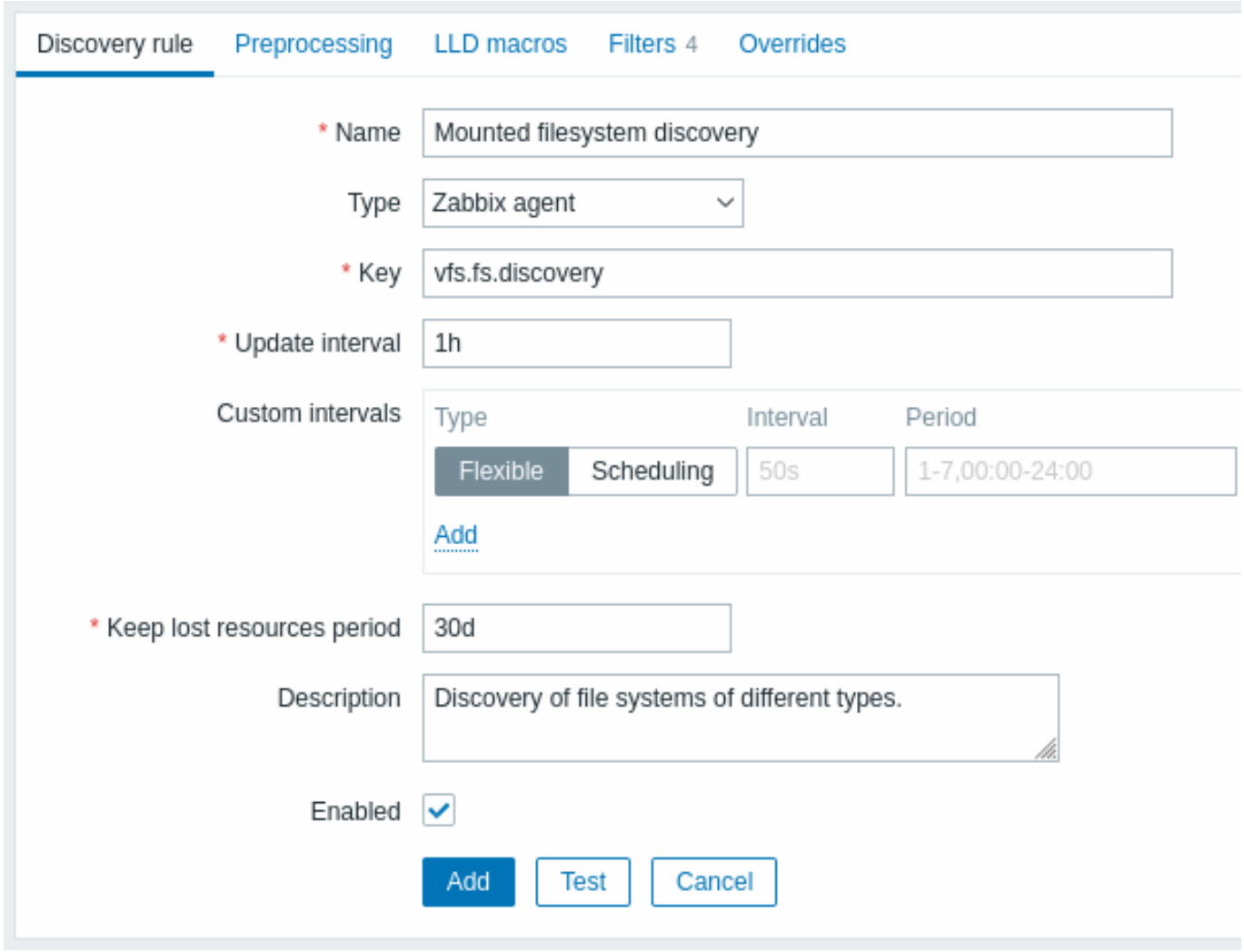
- 在屏幕右上角点击 创建自动发现规则
- 在发现规则表格中填入所需信息

自动发现规则

自动发现规则表格包含五个选项卡，从左到右表示自动发现的数据流：

- 自动发现规则 - 最重要的一项，指定了用于获取自动发现的数据的内置监控项或自定义脚本
- 预处理 - 对发现的数据进行预处理
- *LLD* 宏 - 提取已发现的监控项、触发器等监控指标上配置的宏的取值。
- 过滤 - 过滤发现的值
- 覆盖 - 允许修改发现对象的监控项、触发器、图表和主机的原型

自动发现规则选项卡包含用于自动发现的监控项的键 (以及一些通用的发现规则属性):



所有强制输入区域均会标记红色星号。

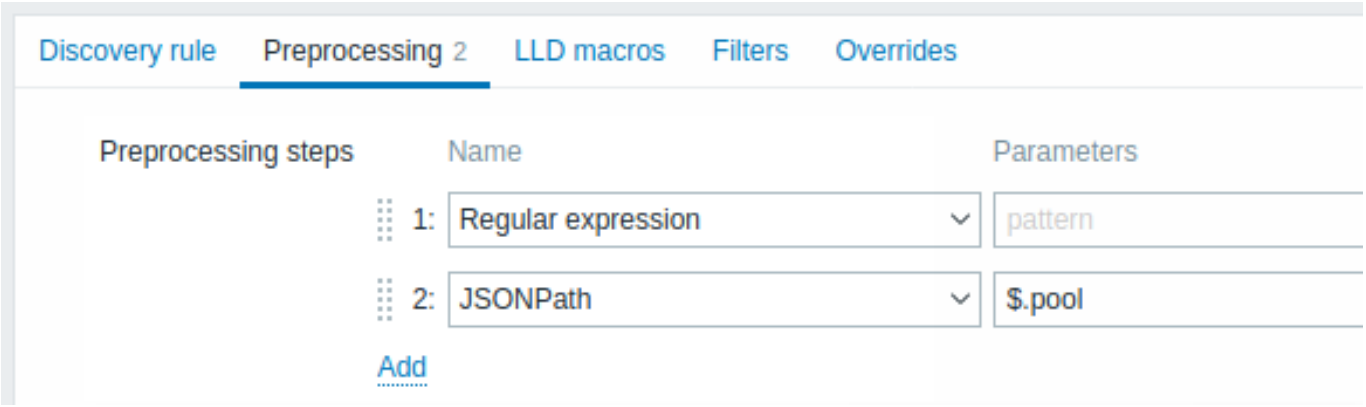
参数	描述
名称	自动发现规则名称。
类型	自动发现检查的类型。 此例中使用 *Zabbix agent* 这个监控项的键。 自动发现规则还可以定义为依赖型监控项，此监控项依赖标准监控项，但不可依赖于其它自动发现规则。对于一个依赖型监控项来说，选择对应类型 (依赖型监控项) 然后在’ 主监控项’ 区域指定主监控项。主监控项必须已存在。
键	输入监控项的键 (最大 2048 个字符)。 例如，可以使用内置的”vfs.fs.discovery” 监控项的键来返回一个 JSON 数组列表，包含计算机文件系统和计算机类型。 注意，文件系统自动发现还可以使用”vfs.fs.get” 键的发现结果，从 Zabbix 4.4.5 (参照示例) 开始支持此特性。
更新间隔	此区域指定了执行自动发现的频率。刚开始执行文件系统自动发现时，可以设置很小的间隔，但当发现之前设置的已经生效，就可以改为 30 分钟或更长。因为文件系统通常不太会变化。 从 Zabbix 3.4.0 开始支持时间后缀，比如 30s, 1m, 2h, 1d。 从 Zabbix 3.4.0 开始支持用户宏 。 注意: 如果设置了取值非零的自定义间隔，则更新间隔只能设置为’0’。 如果更新间隔设置为’0’, 并且自定义间隔 (类型：灵活或计划) 存在非零值, 则监控项的检查频率基于自定义间隔。 注意对于已存在的自动发现规则来说，通过点击 立刻检查按钮可以立即执行自动发现。
自定义间隔	可创建自定义规则来检查监控项: 灵活 - 创建一个 例外的更新间隔 (不同频率的更新间隔)。 计划 - 创建一个自定义的更新时间表。 详细信息参阅自定义间隔。从 Zabbix 3.0.0 开始支持该特性。
保存丢失资源倒计时	指定已发现的资源不被删除的倒计时时间间隔，当该资源的状态变成”不再被发现” 时开始倒计时 (从 1 小时到 25 年；或”0”)。 从 Zabbix 3.4.0 开始支持时间后缀，比如 2h, 1d。 从 Zabbix 3.4.0 开始支持用户宏。 注意*:* 如果值设置为”0”, 资源会马上被删除。不推荐使用”0”，因为错误的设置可能导致资源连同相关历史数据均被删除。
描述	输入描述信息。
启用	如果选定，则规则生效。

自动发现规则历史记录不会保存。

预处理

预处理选项卡定义发现结果的转换规则。可以定义一个或多个转换规则。转换规则执行的顺序以定义的顺序为准。所有预处理均由 Zabbix server 来执行。参阅:

- 预处理详细信息
- 预处理测试



类型	转换	描述
Text		

类型	转换	描述
	正则表达式	将接收的值匹配正则表达式的匹配模式 <pattern>，并替换为输出 <output>。正则表达式可以使用 \N 转义字符提取最多 10 个匹配的组。 参数: 匹配模式 - 正则表达式 输出 - 输出格式化的模板。一个 \N (N=1...9) 转义字符会替换为第 N 个匹配的组。一个 \0 转义字符替换为匹配的文本。 如果选取了 自定义失败检查框, 则可以自定义错误处理选项: 要么舍弃该值并设置一个特定值，要么设置一个错误信息。
	替换	搜索字符串并替换为另一个字符串或空字符串。所有搜索结果均会被替换。 参数: 搜索字符串 - 需要搜索的字符串, 大小写敏感 (强制要求) 替换 - 最终替换成为的字符串。替换成为的字符串可以是空值，通常用于删除搜索字符串。 可使用转义字符”\n \r \t \s” 查找替换换行符, 回车, 制表符和空格; 反斜线可以用”\\” 转义，转义序列可以用”\\n” 转义。底层自动发现过程中，换行符, 回车和制表符自动转义。 从 5.0.0 版本开始支持。
结构化数据		
	JSONPath	使用JSONPath 功能提取数据或将 JSON 数据分片。 如果选取了 自定义失败检查框, 万一预处理失败，监控项不会变为不支持。另外还可以自定义错误处理选项: 要么舍弃该值并设置一个特定值，要么设置一个错误信息。
	XML XPath	使用 XPath 功能提取数据或将 XML 数据分片。要使用此功能，Zabbix server 服务器需要安装 libxml 库。 例子: `number(/document/item/value)` 会从 `<document><item><value>10</value></item></do` 提取 10 `number(/document/item/@attribute)` 会从 `<document><item attribute="10"></item></document>` 提取 10 `/document/item` 会从 `<document><item><value>10</value></item></do` 提取 `<item><value>10</value></item>` 注意，不支持命名空间。 从 4.4.0 版本开始支持此特性。 如果选取了 自定义失败检查框, 则可以自定义错误处理选项: 要么舍弃该值并设置一个特定值，要么设置一个错误信息。
	CSV to JSON	转换 CSV 数据到 JSON 格式。 更多信息请参考: CSV 到 JSON 数据预处理. 从 4.4.0 版本开始支持此特性。
	XML to JSON	转换 XML 数据到 JSON 格式。 更多信息请参阅: 序列化规则. 如果选取了 自定义失败检查框, 则可以自定义错误处理选项: 要么舍弃该值并设置一个特定值，要么设置一个错误信息。
自定义脚本		

类型	转换	描述
	JavaScript	点击参数区域或点击 打开输入 Javascript 代码。 注意，可输入的 Javascript 代码长度取决于使用的数据库. 更多信息请参考: Javascript 代码预处理
验证		
	不匹配正则表达式	指定一个不匹配任何取值的正则表达式。 例如 `Error:(.*?)\.` 如果选取了 自定义失败检查框, 则可以自定义错误处理选项: 要么舍弃该值并设置一个特定值，要么设置一个错误信息。
	检查 *JSON* 数据错误	检查 JSONpath 路径下的应用层错误信息。如果执行成功并且信息不为空则不检查; 否则继续检查此预处理步骤之前的值。注意，这些外部服务错误原封不动报告给用户，不会添加预处理步骤信息。 例如 `$.errors`。如果收到一个这样的 JSON 数据 `{"errors":"e1"}` ，则下个预处理步骤不会执行。 如果选取了 自定义失败检查框, 则可以自定义错误处理选项: 要么舍弃该值并设置一个特定值，要么设置一个错误信息。
	检查 *XML* 数据错误	检查 xpath 路径下的应用层错误信息。如果执行成功并且信息不为空则不检查; 否则继续检查此预处理步骤之前的值。注意，这些外部服务错误原封不动报告给用户，不会添加预处理步骤信息。 无效的 XML 分析失败不会报告给用户。 从 4.4.0 版本开始支持此特性。 如果选取了 自定义失败检查框, 则可以自定义错误处理选项: 要么舍弃该值并设置一个特定值，要么设置一个错误信息。
限流		
	丢弃心跳时间内未改变的值	如果一个值在定义的时间周期内 (秒) 未改变，则丢弃该值。 秒数的范围为正整数 (最小 1 秒)。可使用时间后缀 (如 30s, 1m, 2h, 1d)。可使用用户宏和底层自动发现宏。 一个自动发现监控项只能指定一个限流选项。 比如 `1m`. 如果规则在 60 秒内收到两次相同的值，则相同的值会被丢弃。 注意: 改变监控项原型不会重置限流。仅当预处理步骤发生变化时，限流才会重置。
Prometheus		
	Prometheus 到 *JSON* 格式	转换 Prometheus 度量到 JSON 格式。 参阅Prometheus 检查 获取更多信息。

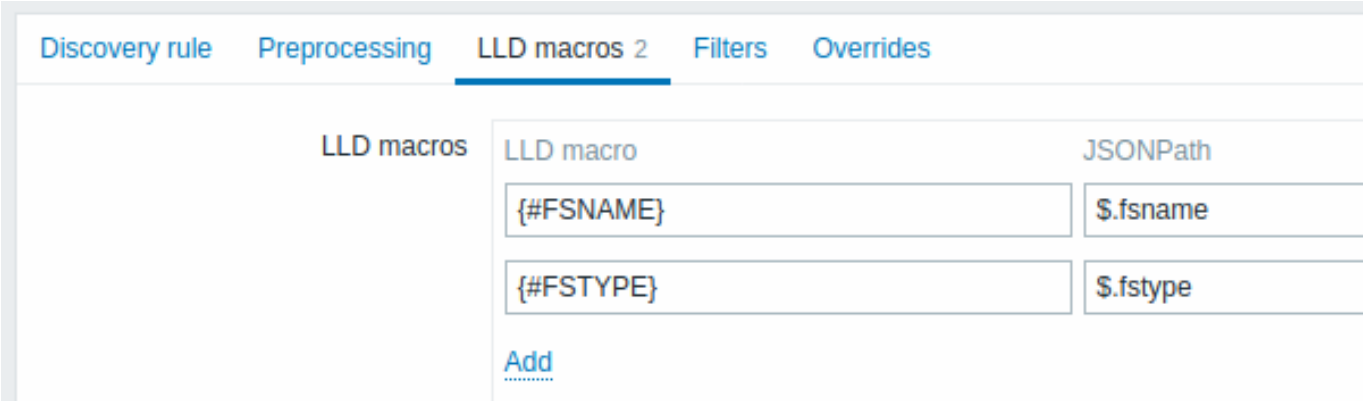
注意，如果自动发现规则已经通过模板应用到主机上，则此选项卡的内容是只读的。

自定义宏

LLD 宏选项卡可以自定义底层自动发现的宏。

如果返回的 JSON 数据不包含所需的宏时，自定义宏就派上用场了。例如:

- 用于文件系统自动发现的内置的 `vfs.fs.discovery` 键返回 JSON 数据，其中包含一些预定义的 LLD 宏，比如 {#FSNAME}, {#FSTYPE}。这些宏可直接用于监控项和触发器原型，(参考本页后续小节); 自定义宏不是强制的;
- `vfs.fs.get` 键同样返回 JSON 数据，包含文件系统数据, 但不包括任何预定义的 LLD 宏。此例中你可以自定义宏, 并把自定义的宏映射到 JSONPath 返回的 JSON 数据上:

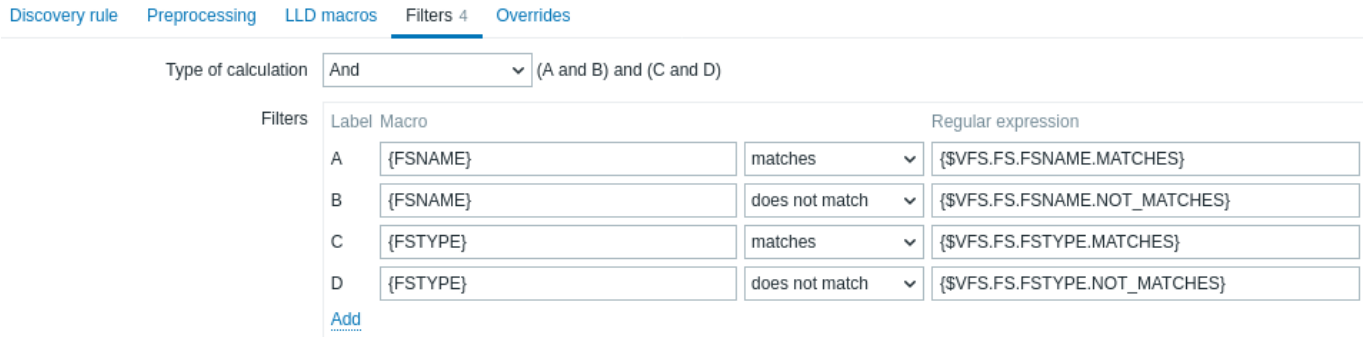


提取的值可用于已发现的监控项、触发器等实体上。注意，值会从自动发现的结果中提取出来，也会从任何到目前为止的预处理步骤的结果中提取出来。

参数	描述
LLD 宏 *JSONPath*	底层自动发现宏的名称，使用如下的语法格式：`{#MACRO}`. 用于提取 LLD 宏的取值的路径，使用 JSONPath 语法格式. 比如，`$.foo` 会从下面的 JSON 数据中提取"bar" 和"baz": `[{"foo":"bar"}, {"foo":"baz"}]` 从 JSON 数据中提取的值用于替换监控项、触发器和其它实体的原型中配置的 LLD 宏。 可以使用点标记法或括号标记法来指定 JSONPath。括号标记法应该用于任何使用特殊字符和编码的场景中，像 `$['unicode + special chars #1']['unicode + special chars #2']`。

过滤器

过滤器可用于生成只匹配过滤条件的监控项、触发器和图表。过滤器选项卡包含自动发现规则的过滤器配置，其可以过滤自动发现的值：



参数	描述
计算类别	可用的过滤器选项如下: 和 - 必须满足所有过滤条件; 或 - 只需满足其中一条过滤条件即可; 和/或 - 不同宏的名称使用 和，相同宏的名称使用 或; 自定义表达式 - 可自定义过滤器的计算公式。公式必须包含列表中的所有过滤器。最大限制 255 个符号。

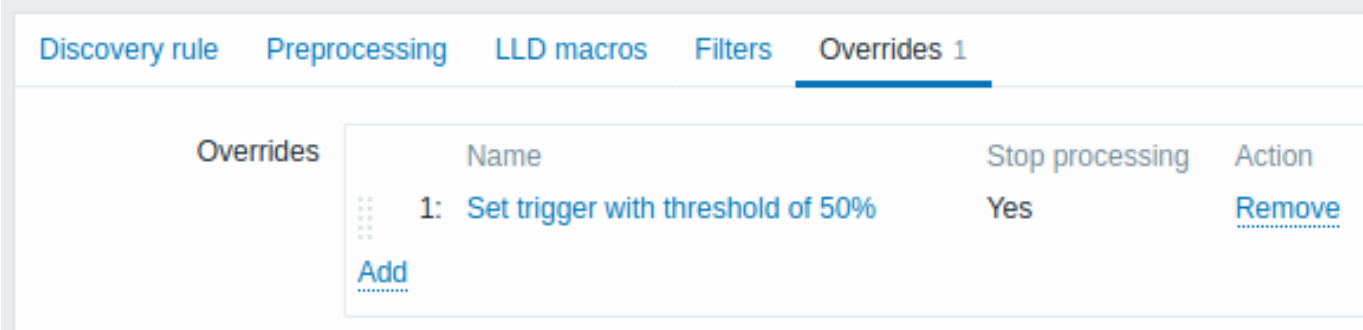
参数	描述
过滤器	可用下列过滤器条件操作符: 匹配, 不匹配, 存在, 不存在. 匹配和 不匹配操作符可识别与 Perl 兼容的正则表达式 (PCRE). 例如, 如果你只对文件系统 C:, D: 和 E: 感兴趣, 那么可将 {#FSNAME} 放入” 宏” 然后将正则表达式”^C|^D|^E” 放入” 正则表达式” 文本区域. 还可通过使用 {#FSTYPE} 宏 (比如. ”^ext|^reiserfs”) 过滤文件系统类型以及使用 {#FSDRIVETYPE} 宏 (比如, ”fixed”) 过滤驱动类型 (只支持 Windows agent)。 可以在” 正则表达式” 区域输入正则表达式或引用全局的正则表达式。 要测试正则表达式，可以使用”grep -E”, 比如: `for f in ext2 nfs reiserfs smbfs; do echo $f | grep -E '^ext` 宏可用于 Windows 系统上，此特性从 Zabbix **3.0.0** 开始支持。 存在和 不存在操作符可以基于响应数据中包含的特定 LLD 宏 (存在或缺失) 进行过滤 (从 5.4.0 版本开始支持)。 从 Zabbix **2.4.0** 版本开始支持定义多个过滤器。

用于 LLD 规则中的正则表达式中的拼写错误 (例如, 一个错误的” 文件系统自动发现” 的正则表达式) 可能造成很多个配置项、历史数据以及主机的事件信息被删除。

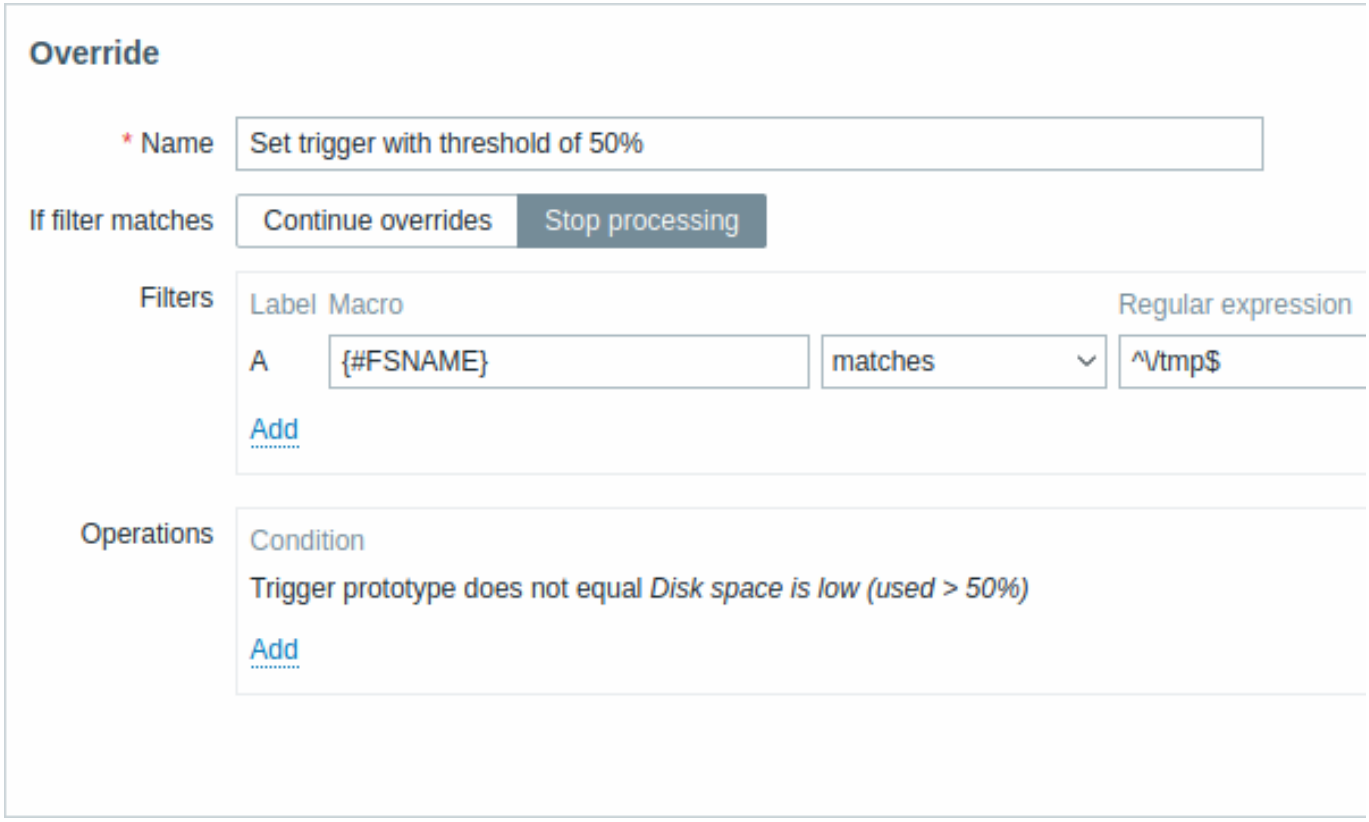
如果操作系统名称只能通过区分大小写来正确识别，那么 Zabbix 的 MYSQL 数据库必须配置成大小写敏感。

覆盖

覆盖选项卡允许设置规则来修改监控项，触发器、图形和主机原型或它们的属性以用于发现满足给定条件的对象。



覆盖（如果有）在可重新排序的拖放列表中显示，并且按照定义的顺序执行。配置一个新的覆盖，在 覆盖选项卡单击 Add 。要编辑现有覆盖，请单击覆盖名称。在弹出窗口中编辑覆盖规则详细信息。

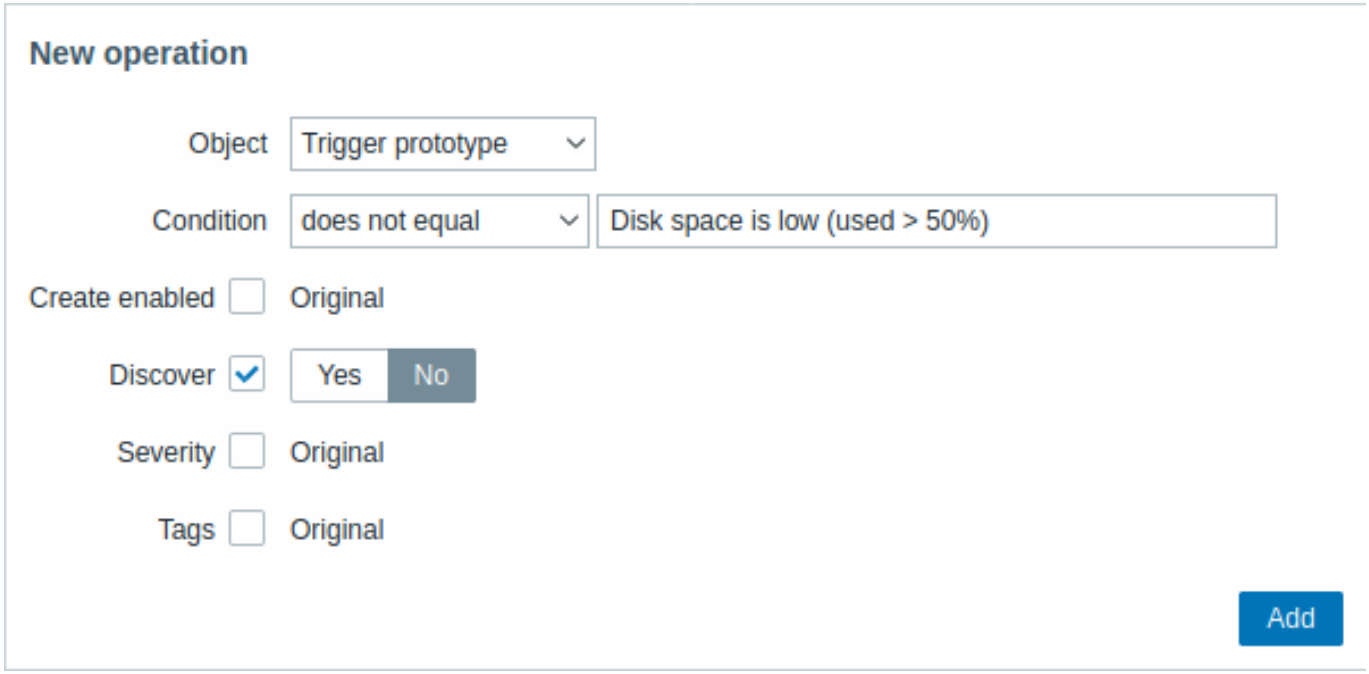


所有强制参数都标有红色星号。

参数	描述
名称	唯一的（根据 LLD 规则）覆盖名称。
如果过滤器匹配	当下面条件满足时，下一个覆盖是否需要执行: 继续覆盖 - 后续覆盖会执行。 停止处理 - 前面的操作 (如果有) 和当前操作会执行，后续覆盖会忽略。
过滤器	定义了覆盖会应用到哪些已发现实体上。覆盖过滤器在自动发现规则的过滤器 之后执行，其和自动发现规则的过滤器功能相同。
操作	覆盖操作包含下列内容: 条件 - 一个对象类型 (监控项原型/触发器原型/图表原型/主机原型) 和一个需要满足的条件 (等于/不等于/包含/不包含/匹配/不匹配) 动作 - 编辑和移除操作的链接。

配置一个操作

要配置一个新操作，点击操作面板上的 Add 。编辑现有操作，点击操作旁边的 Edit 。点击编辑按钮会出现弹窗。



参数
对象
条件
操作
模式
对象: 监控项原型
启用创建
发现
更新间隔
历史保存周期
趋势数据保存周期
标签
对象: 触发器原型
启用创建
发现
严重性
标签
对象: 图形原型
发现
对象: 主机原型
启用创建
发现
链接模板
标签
主机资产

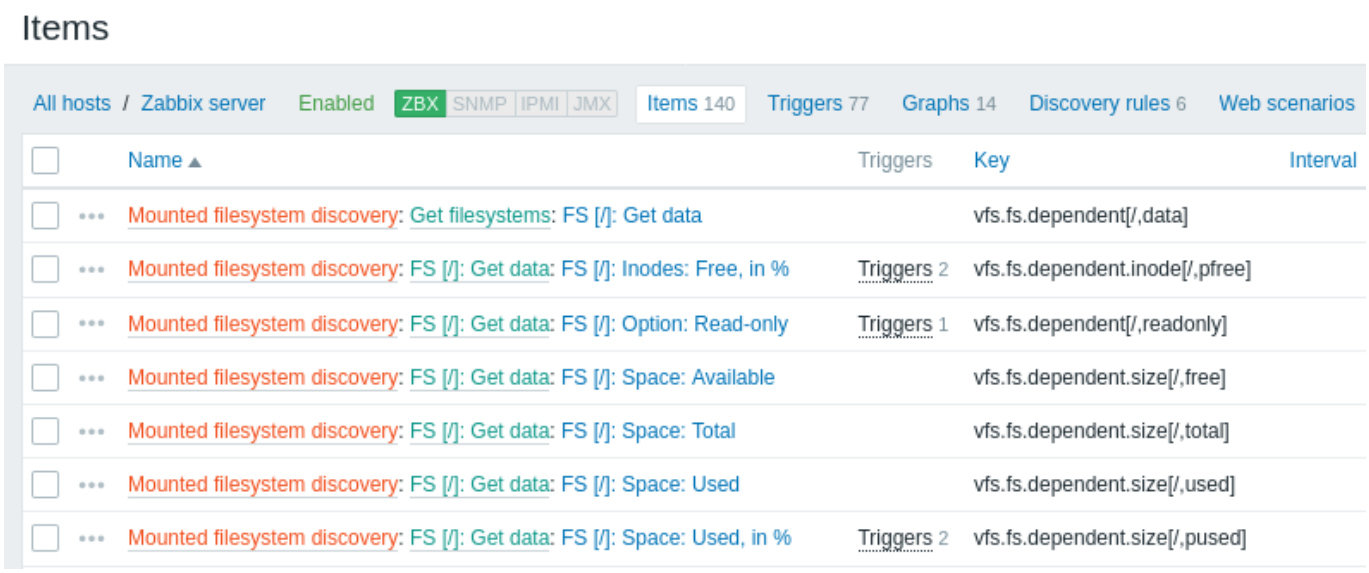
表格按钮

表格底部按钮允许执行一些操作。

.	.
Add	添加自动发现规则。此按钮只能用于新的自动发现规则。
Update	更新自动发现规则的属性。此按钮只能用于现有的自动发现规则。
Clone	基于当前自动发现规则的属性，创建另一个自动发现规则。

.	.
	立即运行自动发现规则。该自动发现规则必须已存在。参考更多信息. 注意当立即执行自动发现时, 配置缓存并没有更新, 所以自动发现的结果可能不会显示最近的变更。
Delete	删除自动发现规则。
Cancel	取消对自动发现规则的编辑。

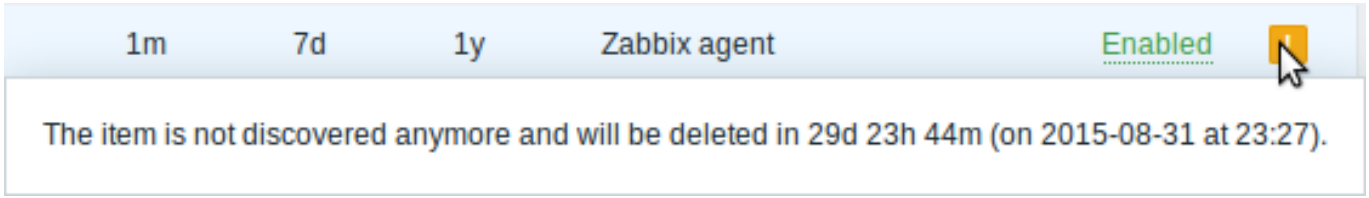
发现的实体　下面的截图展示了主机配置中已发现的监控项、触发器和图表。发现的实体用橙色的链接作为前缀标记，橙色链接指向关联的自动发现规则。



注意，如果跟现有实体的设备唯一标识相同，则不会创建新发现的实体。比如，一个监控项的键的名称重复或图表的名称重复。此类错误会显示在前端页面上，表明低级别自动发现无法创建对应实体。然而自动发现规则本身无法变为不支持的 (unsupported) 状态，因为一些实体无法创建，且必须跳过。自动发现规则会继续创建/更新其它实体。

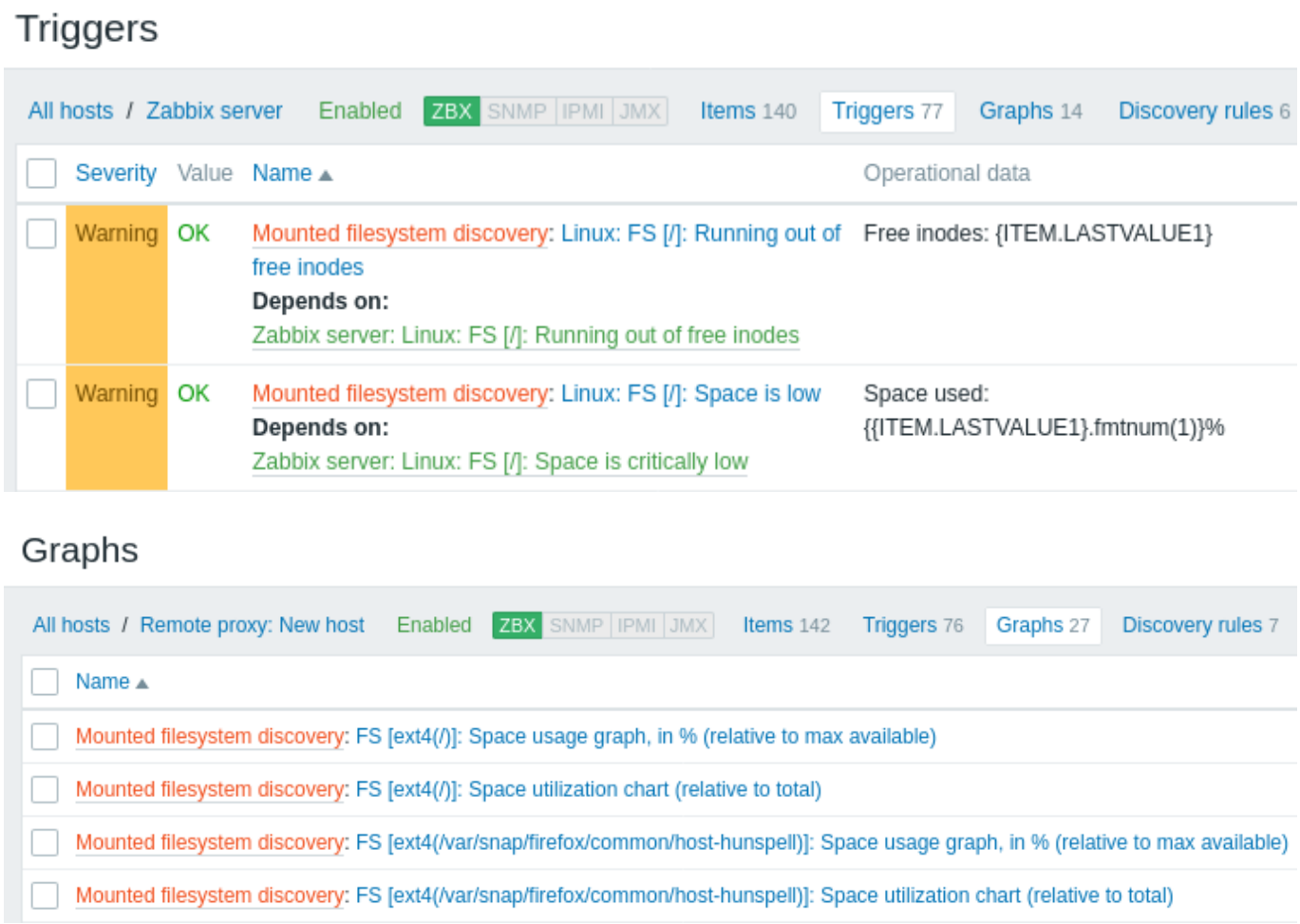
如果发现的实体 (文件系统、接口等) 不再被发现 (或过滤器过滤不出来)，则由低级别自动发现创建的监控项 (触发器、图表也类似) 会自动删除。此场景中，监控项、触发器和图表会在若干时间后被删除，数据保存的时间定义在 保存丢失资源周期输入框中。

当发现的实体变为' 不再被发现', 监控项列表中会显示一个涵盖整个生命周期的数据。移动鼠标到该数据上面，会显示离删除还剩多少天。



如果实体标记为已删除，但过期未删除 (由于禁用了发现规则或禁用了监控的主机)，则会在下次运行自动发现规则时删除这些实体。

包含其它实体的实体，且标记为已删除，如果在自动发现规则层面做了变更，则不会再更新。比如，基于低级别自动发现的触发器不会再更新，即使它们包含标记为已删除的实体。



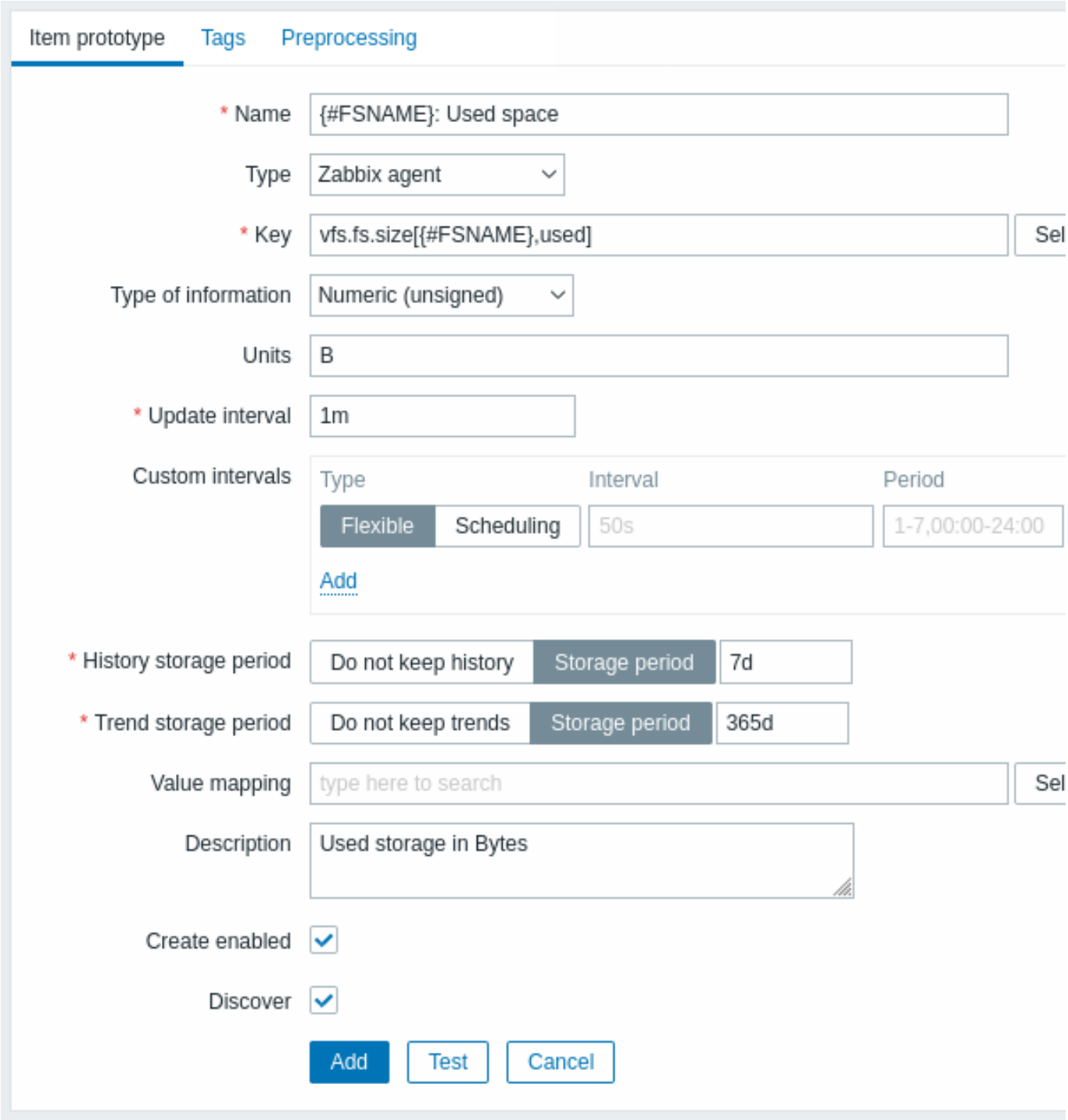
其它类型的自动发现　如果想立刻上手熟悉其它类型的自动发现，更多信息和如何实现 (how-to) 方面可参阅以下章节:

- 网络接口的自动发现;
- CPU 和 CPU 核心的自动发现;
- SNMP OID的自动发现;
- JMX 对象的自动发现;
- 使用ODBC SQL 查询的自动发现;
- Windows 服务的自动发现;
- Zabbix 主机接口的自动发现。

更多关于 JSON 格式的自动发现和关于如何使用 Perl 脚本对文件系统做自动发现，请参阅创建自定义 LLD 规则。

1 监控项原型

一旦创建了规则，下一步就是找到该规则下面的监控项，然后点击" 创建监控项原型" 来创建一个监控项原型。注意宏 {#FSNAME} 的使用方式，它被填充到需要输入文件系统名称的地方。当自动发现规则开始执行，这个宏会替换成发现的文件系统。



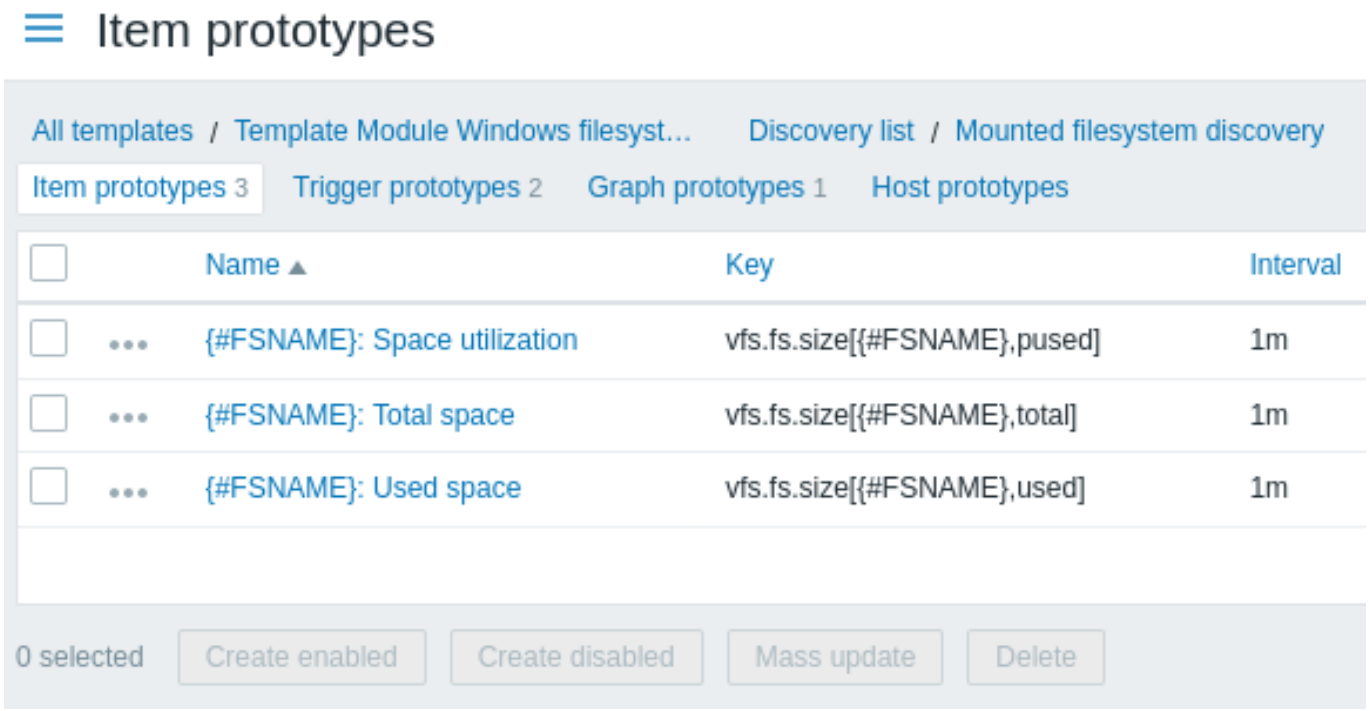
低级别自动发现宏 和用户宏 可以用在监控项原型配置中和监控项值的预处理参数中。注意，当宏用在更新间隔中时，单一的宏必须填满整个输入框。不支持一个输入框中使用多个宏，也不支持混合使用宏和文本。

Note:
LLD 宏的上下文特定转义字符用于安全使用正则表达式和 XPath 预处理参数中。

监控项原型的属性:

参数	描述
启用创建	如果选择，会添加并启用监控项。 如果不选择，会添加监控项到发现的实体中，但会处于禁用状态。
发现	如果选择 (默认)，会添加监控项到发现的实体中。 如果不选择，监控项不会添加到发现的实体中，除非设置被自动发现规则覆盖。

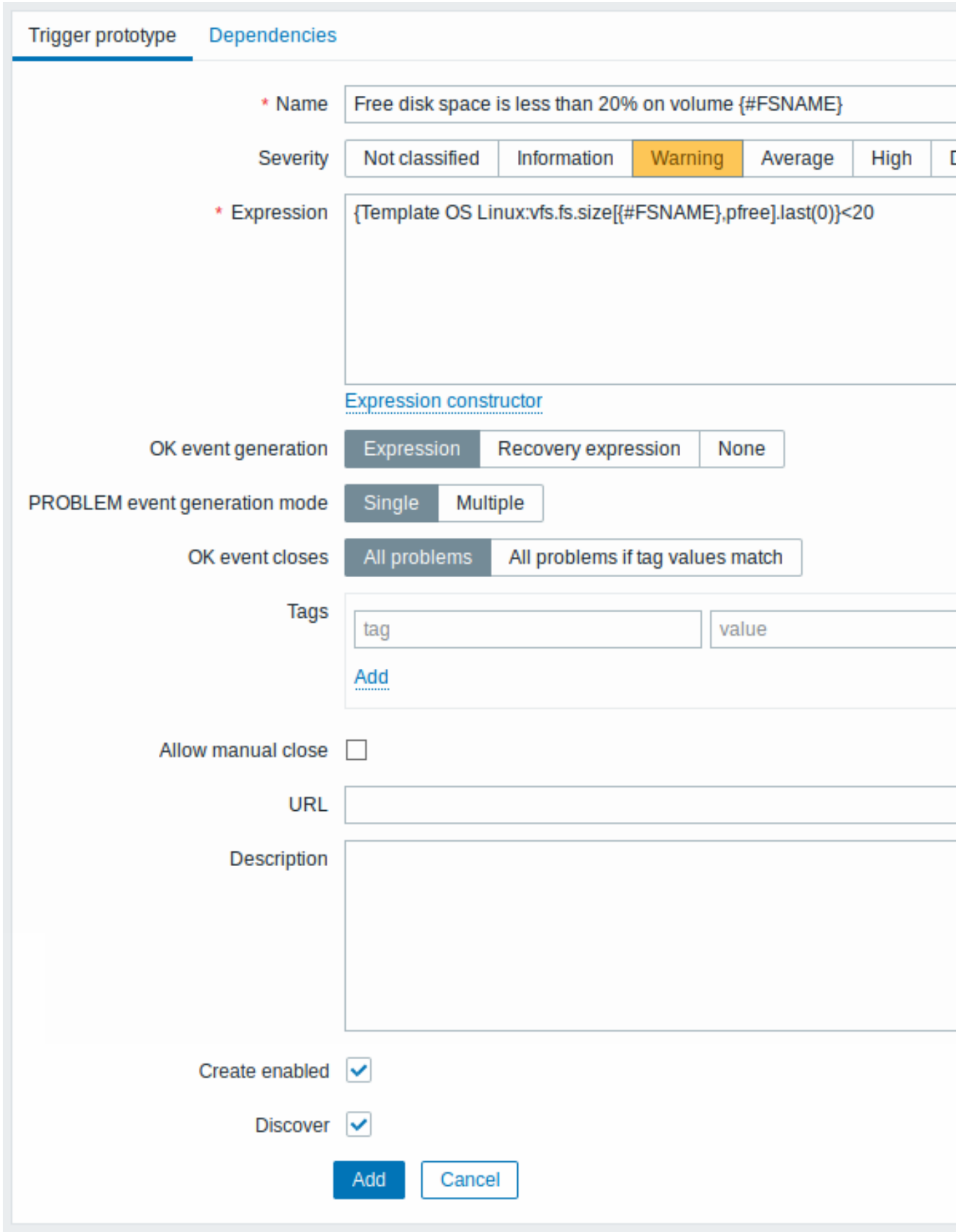
可以为每个感兴趣的文件系统度量创建多个监控项原型:



点击三个点的图标，打开菜单，找到具体的监控项原型，其中有这些选项:
 - 创建触发器原型 - 为此监控项原型创建一个触发器原型 - 触发器原型 - 点击查看此监控项中已有的触发器原型的列表 - 创建依赖监控项 - 为此监控项原型创建一个依赖监控项 大批量更新 可用于一次性更新多个监控项原型的属性。

2 触发器原型

可以用创建监控项原型相同的方式来创建触发器原型:



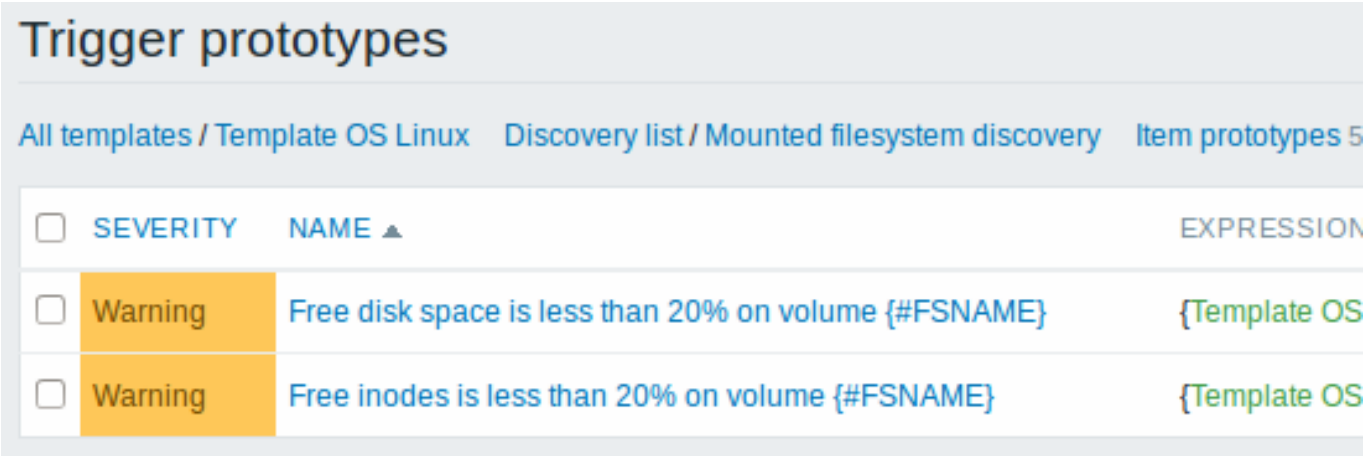
触发器原型的属性:

参数	描述
启用创建	如果选择，会添加并启用触发器原型。 如果未选择，会添加触发器到发现的实体中，但是处于禁用状态。

参数	描述
发现	如果选择 (默认)，触发器会添加到发现的实体中。 如果未选择，触发器不会添加到发现的实体中，除非设置被自动发现规则覆盖。

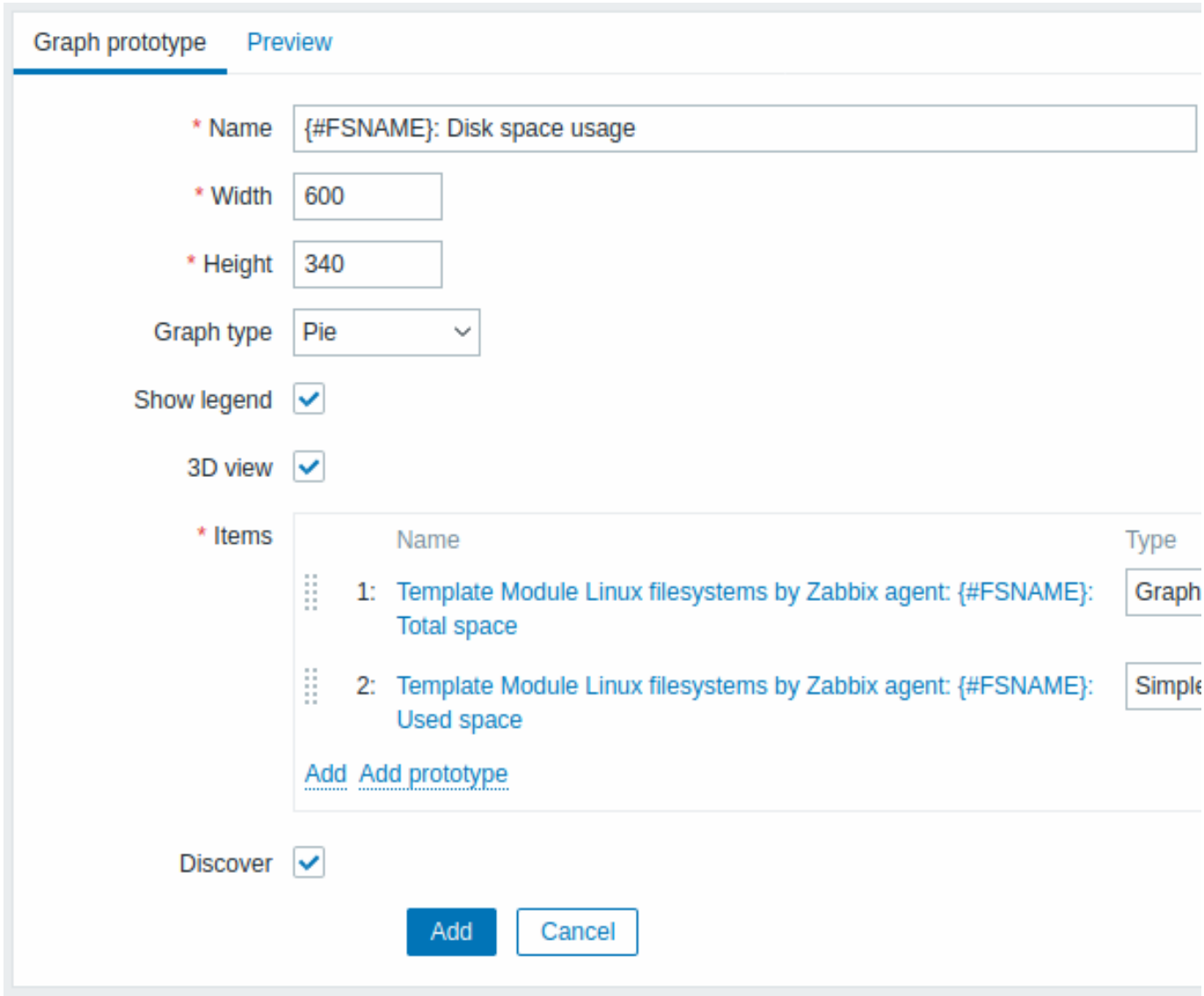
一旦实际的触发器从原型中创建出来，表达式中用于作对比的常量 (此示例中为'20') 可能需要实现一些灵活性。参考如何使用用户宏和上下文 来实现这种灵活性。

还可以定义在多个触发器原型之间定义依赖 关系 (从 Zabbix 3.0 起支持)。要实现此功能，找到 依赖选项卡。一个触发器原型可依赖于同一个 LLD 规则中的另一个触发器原型，或者依赖于一个常规的触发器。一个触发器原型不会依赖于另一个 LLD 规则中的触发器原型，也不会依赖于从一个触发器原型中创建的触发器。主机的触发器原型不能依赖于从模板中创建的触发器。



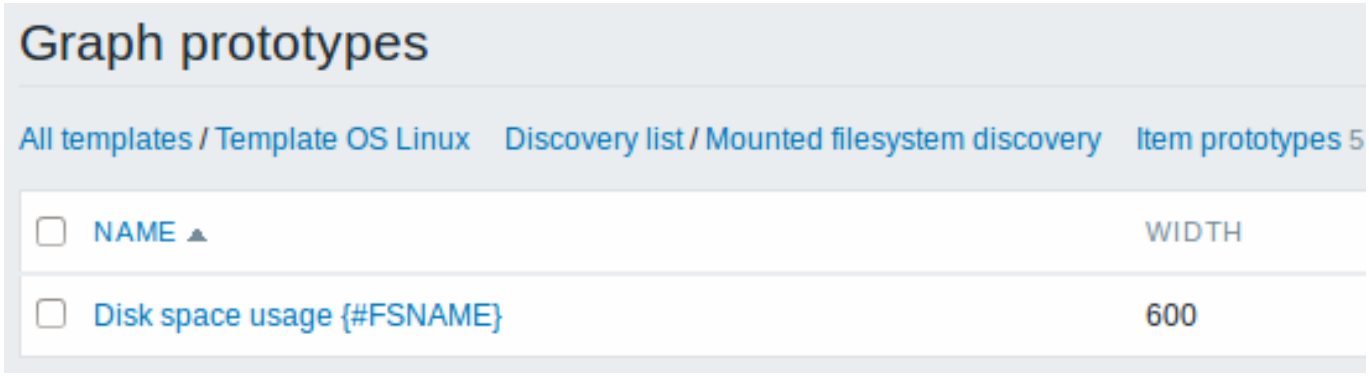
3 图形原型

还可以创建图形原型:

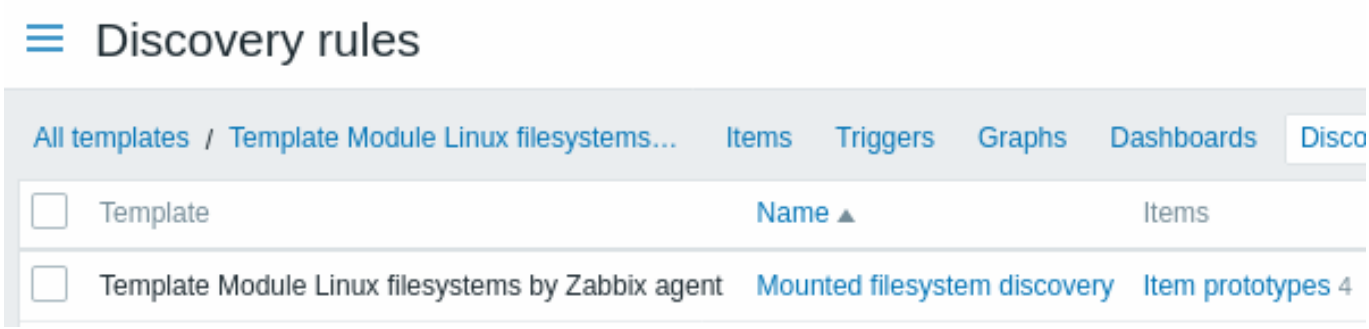


图形原型的属性:

参数	描述
发现	如果选择 (默认)，会添加图形到已发现的实体中。 如果未选择，图形不会添加到已发现的实体中，除非设置被自动发现规则覆盖。



最终创建了如下的自动发现规则。该规则有五个监控项原型、两个触发器原型和一个图形原型。



注意: 要配置主机原型，请参考虚拟机监控中关于主机原型 配置的章节。

4 低级别自动发现的注意事项

在用户宏的上下文中使用 LLD 宏

LLD 宏可用在用户宏的上下文中，比如在触发器原型中。

同一监控项的多条 LLD 规则

从 Zabbix agent 3.2 版本起，可以为相同的自动发现监控项定义多条低级别自动发现规则。

要实现此功能，需要定义 agent 参数 别名，允许在不同自动发现规则中使用同一监控项的不同键，比如 `vfs.fs.discovery[foo]`，`vfs.fs.discovery[bar]`，等。

返回值的数据大小限制

如果低级别自动发现规则返回的 JSON 数据由 Zabbix server 直接接收，则数据无大小限制，因为返回值不会保存到数据库中。同样，自定义的低级别自动发现规则返回的数据也无大小限制，但如果使用一个用户参数来获取自定义 LLD 数据，那么用户参数的返回值有限制(最大 512 KB)。如果数据由 Zabbix proxy 处理，数据必须保存到数据库中，所以数据大小受限于数据库限制。

5 自动发现规则

请使用侧边栏查看自动发现规则的配置示例。

1 自动发现已挂载的文件系统

概述

可对已挂载的文件系统及相关属性 (挂载点名称, 挂载点类型, 文件系统大小和 inode 数据) 进行自动发现。

要想实现此功能，需要结合使用:

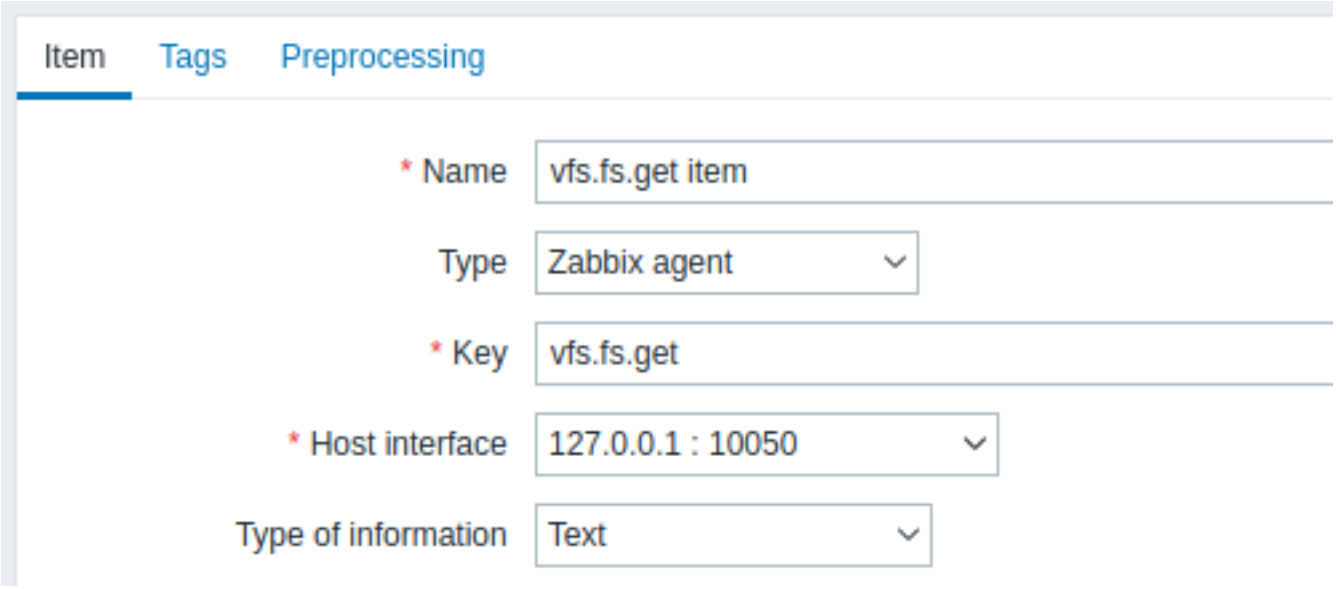
- 监控项 `vfs.fs.get` 作为主监控项
- 依赖型低级别自动发现规则和依赖型监控项原型

配置

主监控项

使用下面的键创建 Zabbix agent 监控项:

`vfs.fs.get`



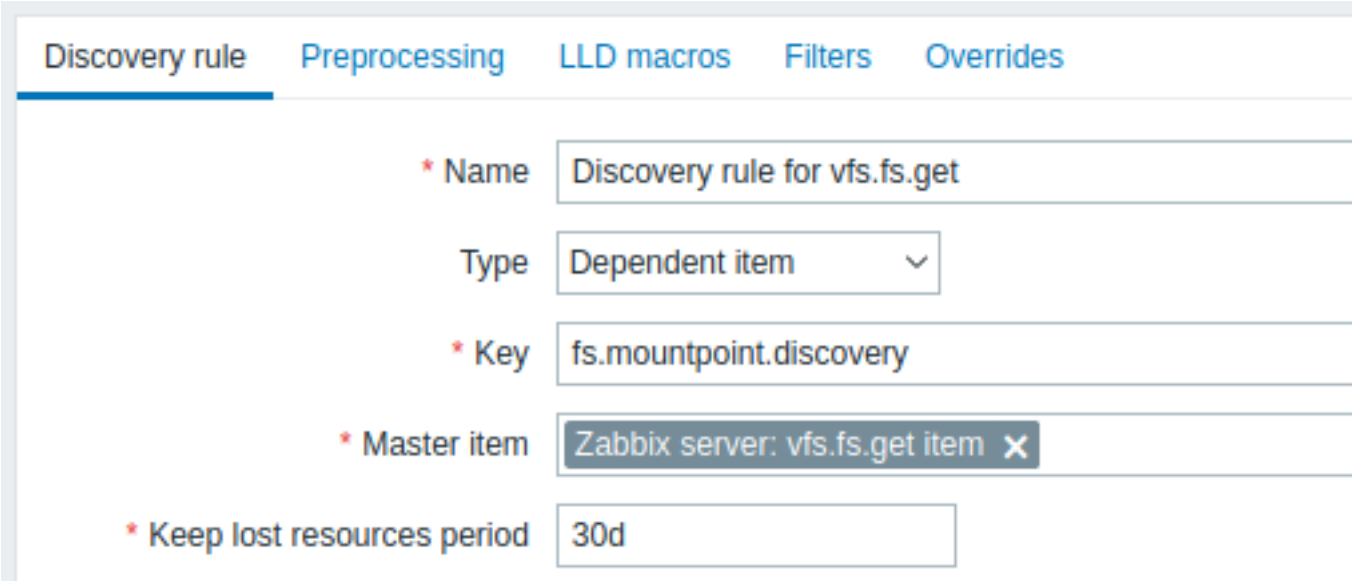
设置信息类型为”Text”，为了识别大块 JSON 数据。

用于已挂载文件系统的监控项返回的数据会包含类似下面的信息:

```
{
  "fsname": "/",
  "fstype": "rootfs",
  "bytes": {
    "total": 1000,
    "free": 500,
    "used": 500,
    "pfree": 50.00,
    "pused": 50.00
  },
  "inodes": {
    "total": 1000,
    "free": 500,
    "used": 500,
    "pfree": 50.00,
    "pused": 50.00
  }
}
```

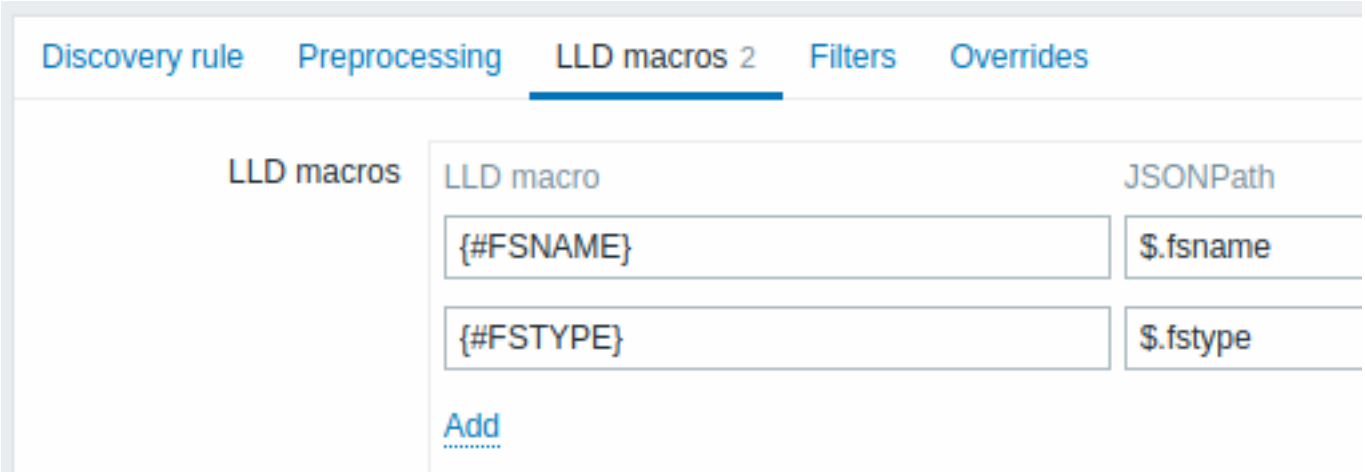
依赖型 LLD 规则

创建一个低级别自动发现规则作为” 依赖型监控项” 类型:



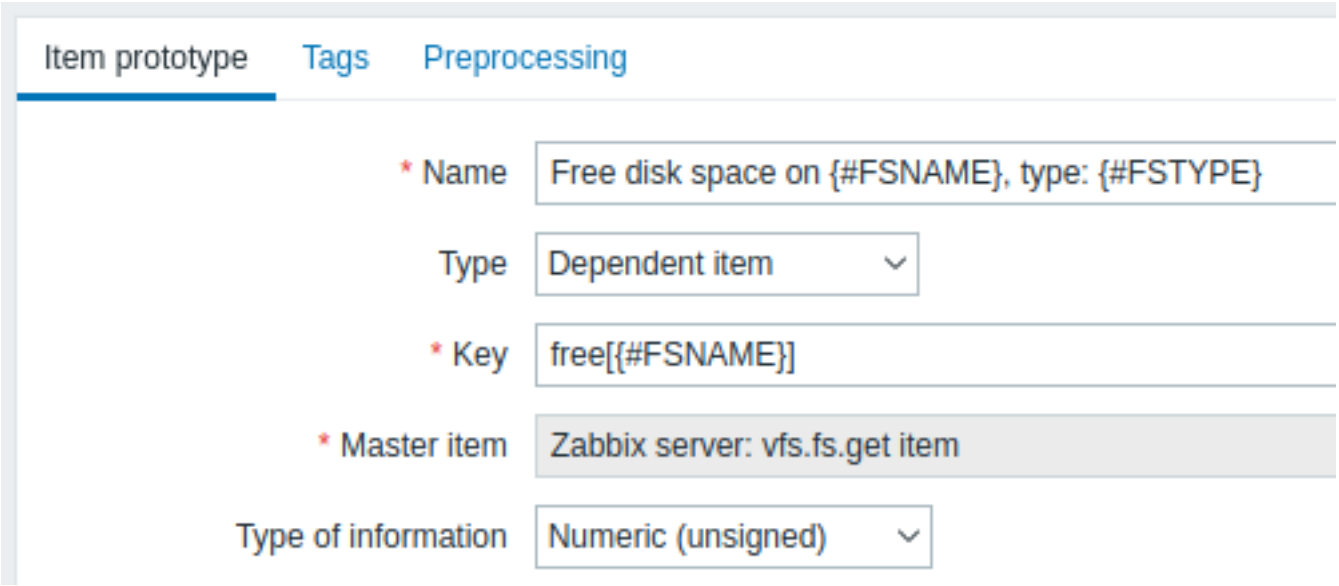
主监控项选择之前我们创建的 `vfs.fs.get` 监控项。

在”LLD 宏” 选项卡中自定义宏，使用对应的 JSONPath:



依赖型监控项原型

在此 LLD 规则中创建一个监控项原型，类型选择” 依赖型监控项”。此原型的主监控项选择我们之前创建的 `vfs.fs.get` 监控项。



注意，对于监控项原型的名称和键，其自定义宏的使用方式:

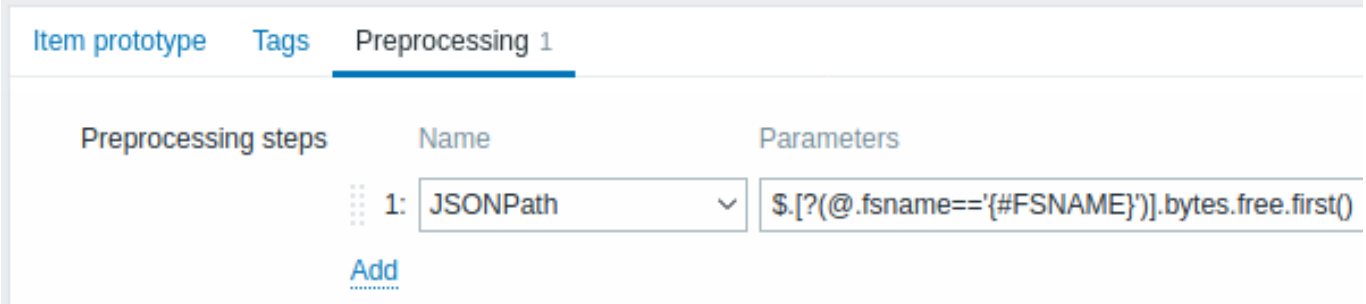
- 名称: `{#FSNAME}` 上的空闲磁盘空间, 类型: `{#FSTYPE}`
- 键: 空闲 `[{#FSNAME}]`

信息类型使用:

- 数字 *(无符号)* 用于像’free’, ’total’, ’used’ 这样的度量
- 数字 *(浮点数)* 用于像’pfree’, ’pused’ (百分比) 这样的度量

在监控项原型”Preprocessing” 选项卡中选择 JSONPath 并使用下面的 JSONPath 表达式作为参数:

```
$.[?(@.fsname=='{#FSNAME}')].bytes.free.first()
```



一旦自动发现开始工作，每个挂载点会创建一个监控项。监控项会返回对应挂载点的空闲空间大小 (字节)。

2 网络接口的自动发现

对网络接口进行自动发现与对文件系统 进行自动发现的工作方式类似。

监控项的键

在自动发现规则 中使用的键是

`net.if.discovery`

从 Zabbix agent 2.0 起支持使用此监控项。

支持的宏

可在自动发现规则的过滤器 和监控项、触发器、图表的原型中用使用 {#IFNAME} 宏。

下面是基于”net.if.discovery” 键创建的监控项原型:

- ”net.if.in[{#IFNAME},bytes]”,
- ”net.if.out[{#IFNAME},bytes]”.

注意，在 Windows 操作系统上也可使用 {#IFGUID} 宏获取返回值。

3 CPU 和 **CPU** 核心的自动发现

CPU 和 CPU 核心的自动发现与文件系统 的自动发现工作方式相似。

监控项的键

在自动发现规则 中使用的监控项的键是

`system.cpu.discovery`

从 Zabbix agent 2.4 起支持此监控项。

支持的宏

这个键使用两个宏 - {#CPU.NUMBER} 用于识别 CPU 序号，{#CPU.STATUS} 用于识别 CPU 状态。注意，没有办法清楚地区分出来实际的物理处理器、核心与超线程这三者。Linux、UNIX 和 BSD 系统上的 {#CPU.STATUS} 返回处理器的状态，状态可以是” 在线” 或者” 离线”。在 Windows 操作系统上，同样的宏可能会返回第三个取值 - ” 未知” - 表示已检测到处理器，但还未收集到相关信息。

为了保持与收集器提供的数据一致，CPU 自动发现功能依赖 agent 的收集器进程，通过此方式还能在获取数据方面节省资源，但是这会导致此监控项的键不支持二进制 agent 的测试 (-t) 命令行参数，并且会返回一个 NOT_SUPPORTED 状态，并附带信息，表明收集器进程未启动。

可基于 CPU 自动发现来创建监控项原型，比如:

- `system.cpu.util[{#CPU.NUMBER},<type>,<mode>]`
- `system.hw.cpu[{#CPU.NUMBER},<info>]`

关于监控项的键的详细信息, 参考Zabbix agent 监控项的键。

Notes on discovered performance counter items on NUMA systems

Some Windows performance counters might be not available for some of the logical processors at NUMA systems.

For example, items discovered with an item prototype with the key below might work only for the first NUMA node. Items for the other NUMA nodes might be in unsupported state.

`perf_counter[\Processor({#CPU.NUMBER})\% Processor Time,60]`

Also, a discovery rule with key ”system.cpu.discovery” does not provide any low-level discovery macro values to substitute into such performance counters as:

`perf_counter["\Processor Information(<NUMA node index>,<CPU index in NUMA node>)\% Processor Time",60]`

While system.cpu.util items rely on different performance counters to monitor CPU utilization percentage depending on the processor group count and processor count. Also, there is a rare bug related to processor groups on Windows.

4 SNMP OID 的自动发现

概述

本章中会演示如何对交换机执行一次 SNMP 的自动发现 。

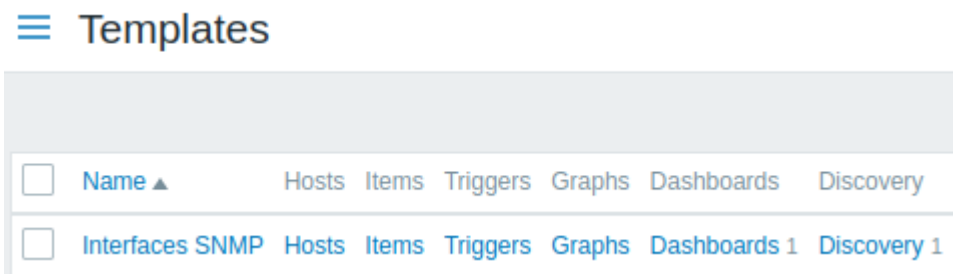
监控项的键

跟文件系统和网络接口自动发现不同，SNMP 自动发现的监控项无需配置”snmp.discovery” 键 - 只要监控项类型设置为 SNMP agent 就足够了。

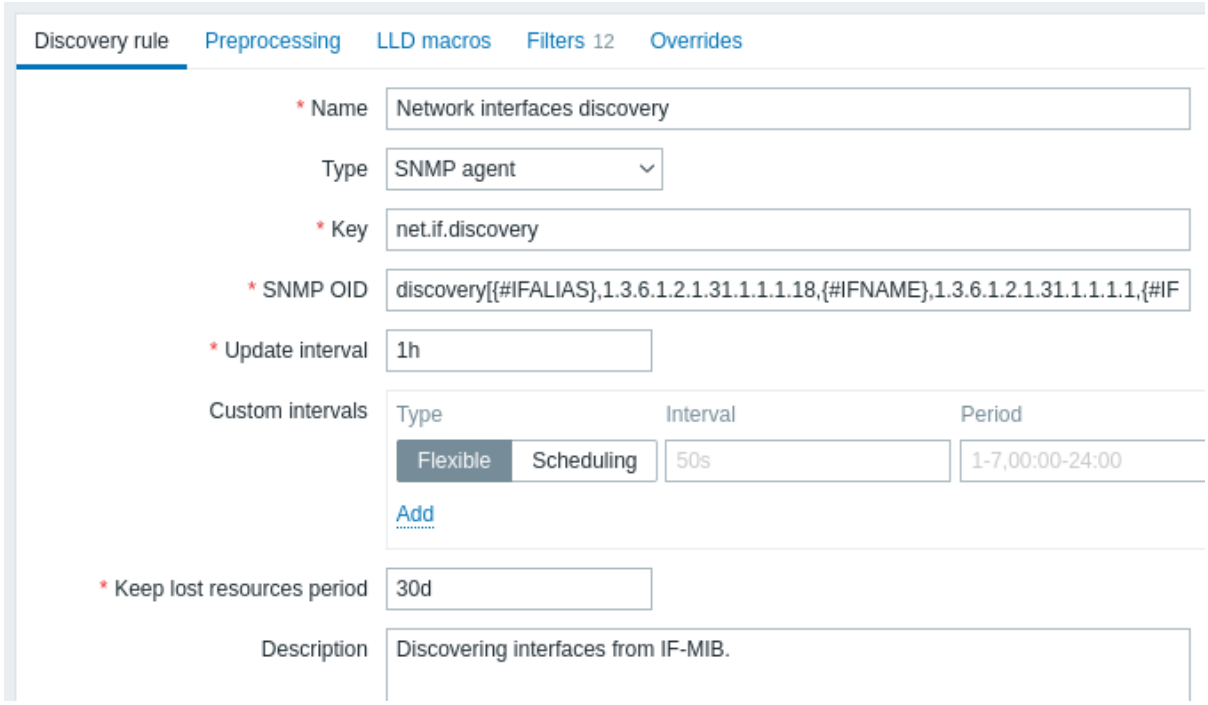
从 Zabbix server/proxy 2.0 起支持 SNMP OID 自动发现。

要配置自动发现规则，需要完成下面的步骤:

- 找到: 配置 → 模板
- 在对应模板所在的那一行点击 自动发现



- 点击屏幕右上角的 创建自动发现规则
- 填写下面截图中的自动发现规则表格



所有强制输入框标记为红色星号。

待发现的 OID 在 SNMP OID 输入框中定义，格式如下: `discovery[{#MACRO1}, oid1, {#MACRO2}, oid2, …,]`

其中 *{#MACRO1}*, *{#MACRO2}* ... 是有效的 LLD 宏名称，然后 *oid1*, *oid2*... 是可以为这些宏生成具体值的 OID。内置的宏 *{#SNMPINDEX}* 包含已发现 OID 的索引，这个宏可用于已发现的实体上。已发现的实体通过 *{#SNMPINDEX}* 宏的返回值来进行分组。

要理解上面说的，我们来演示一下对交换机执行 snmpwalk 操作:

```
$ snmpwalk -v 2c -c public 192.168.1.1 IF-MIB::ifDescr
IF-MIB::ifDescr.1 = STRING: WAN
IF-MIB::ifDescr.2 = STRING: LAN1
IF-MIB::ifDescr.3 = STRING: LAN2

$ snmpwalk -v 2c -c public 192.168.1.1 IF-MIB::ifPhysAddress
IF-MIB::ifPhysAddress.1 = STRING: 8:0:27:90:7a:75
IF-MIB::ifPhysAddress.2 = STRING: 8:0:27:90:7a:76
IF-MIB::ifPhysAddress.3 = STRING: 8:0:27:2b:af:9e
```

然后设置 SNMP OID 为: `discovery[{#IFDESCR}, ifDescr, {#IFPHYSADDRESS}, ifPhysAddress]`

现在规则会自动去发现对应实体，实体中的 {#IFDESCR} 宏需要设置为 **WAN**, **LAN1** 和 **LAN2**, {#IFPHYSADDRESS} 宏需要设置为 **8:0:27:90:7a:75**, **8:0:27:90:7a:76**, 和 **8:0:27:2b:af:9e**, {#SNMPINDEX} 宏需要设置为已发现的 OID 的索引值 **1**, **2** 和 **3**:

```
[
    {
        "{#SNMPINDEX}": "1",
        "{#IFDESCR}": "WAN",
        "{#IFPHYSADDRESS}": "8:0:27:90:7a:75"
    },
    {
        "{#SNMPINDEX}": "2",
        "{#IFDESCR}": "LAN1",
        "{#IFPHYSADDRESS}": "8:0:27:90:7a:76"
    },
    {
        "{#SNMPINDEX}": "3",
        "{#IFDESCR}": "LAN2",
        "{#IFPHYSADDRESS}": "8:0:27:2b:af:9e"
    }
]
```

如果实体没有指定的 OID，则对应的宏会被跳过。比如，假如存在下面的数据:

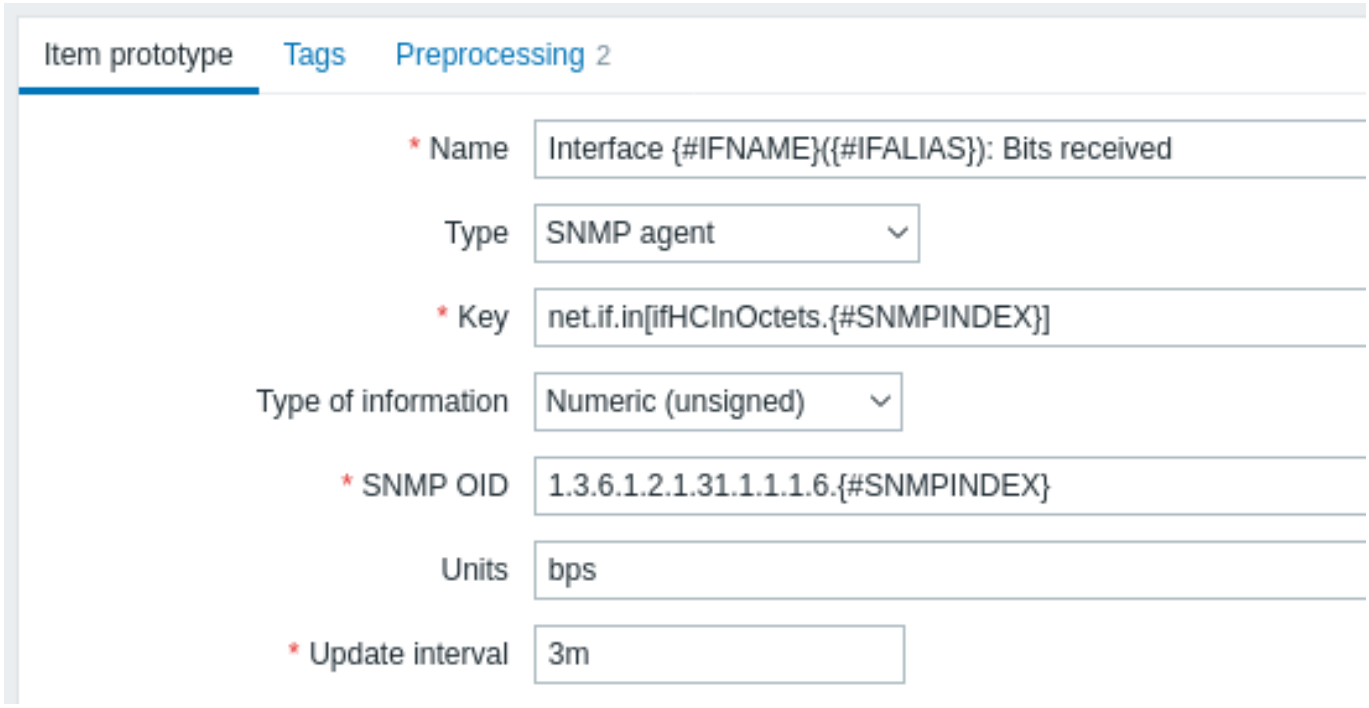
```
ifDescr.1 "Interface #1"
ifDescr.2 "Interface #2"
ifDescr.4 "Interface #4"

ifAlias.1 "eth0"
ifAlias.2 "eth1"
ifAlias.3 "eth2"
ifAlias.5 "eth4"
```

在此场景中，SNMP 自动发现 `discovery[{#IFDESCR}, ifDescr, {#IFALIAS}, ifAlias]` 会返回这种结构的数据:

```
[
    {
        "{#SNMPINDEX}": 1,
        "{#IFDESCR}": "Interface #1",
        "{#IFALIAS}": "eth0"
    },
    {
        "{#SNMPINDEX}": 2,
        "{#IFDESCR}": "Interface #2",
        "{#IFALIAS}": "eth1"
    },
    {
        "{#SNMPINDEX}": 3,
        "{#IFALIAS}": "eth2"
    },
    {
        "{#SNMPINDEX}": 4,
        "{#IFDESCR}": "Interface #4"
    },
    {
        "{#SNMPINDEX}": 5,
        "{#IFALIAS}": "eth4"
    }
]
```

监控项原型

下面的截图演示了如何在监控项原型中使用宏:



创建所需的监控项原型，数量不限:

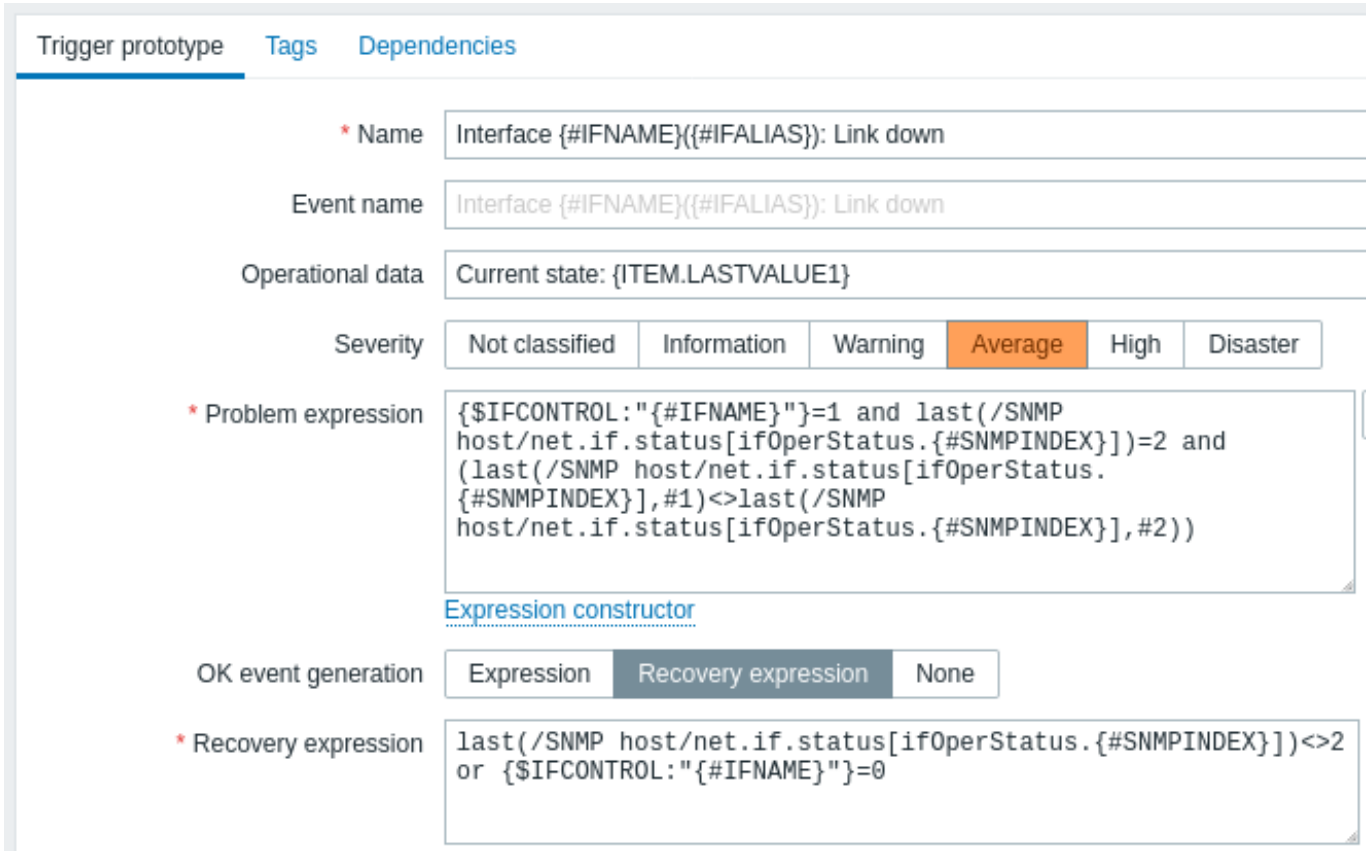
Item prototypes

All templates / Linux SNMP Discovery list / Network interfaces discovery Item prototypes 9 Trigger prototypes 4 Graph prototypes 1 Host prototypes

Name ▲	Key	Interval	History	Trends	Type	Create enabled
Interface {#IFNAME}({#IFALIAS}): Bits received	net.if.in[ifHCInOctets.{#SNMPINDEX}]	3m	7d	365d	SNMP agent	Yes
Interface {#IFNAME}({#IFALIAS}): Bits sent	net.if.out[ifHCOutOctets.{#SNMPINDEX}]	3m	7d	365d	SNMP agent	Yes
Interface {#IFNAME}({#IFALIAS}): Inbound packets discarded	net.if.in.discards[ifInDiscards.{#SNMPINDEX}]	3m	7d	365d	SNMP agent	Yes
Interface {#IFNAME}({#IFALIAS}): Inbound packets with errors	net.if.in.errors[ifInErrors.{#SNMPINDEX}]	3m	7d	365d	SNMP agent	Yes
Interface {#IFNAME}({#IFALIAS}): Interface type	net.if.type[ifType.{#SNMPINDEX}]	1h	7d	0d	SNMP agent	Yes
Interface {#IFNAME}({#IFALIAS}): Operational status	net.if.status[ifOperStatus.{#SNMPINDEX}]	1m	7d	0	SNMP agent	Yes
Interface {#IFNAME}({#IFALIAS}): Outbound packets discarded	net.if.out.discards[ifOutDiscards.{#SNMPINDEX}]	3m	7d	365d	SNMP agent	Yes
Interface {#IFNAME}({#IFALIAS}): Outbound packets with errors	net.if.out.errors[ifOutErrors.{#SNMPINDEX}]	3m	7d	365d	SNMP agent	Yes
Interface {#IFNAME}({#IFALIAS}): Speed	net.if.speed[ifHighSpeed.{#SNMPINDEX}]	5m	7d	0d	SNMP agent	Yes

触发器原型

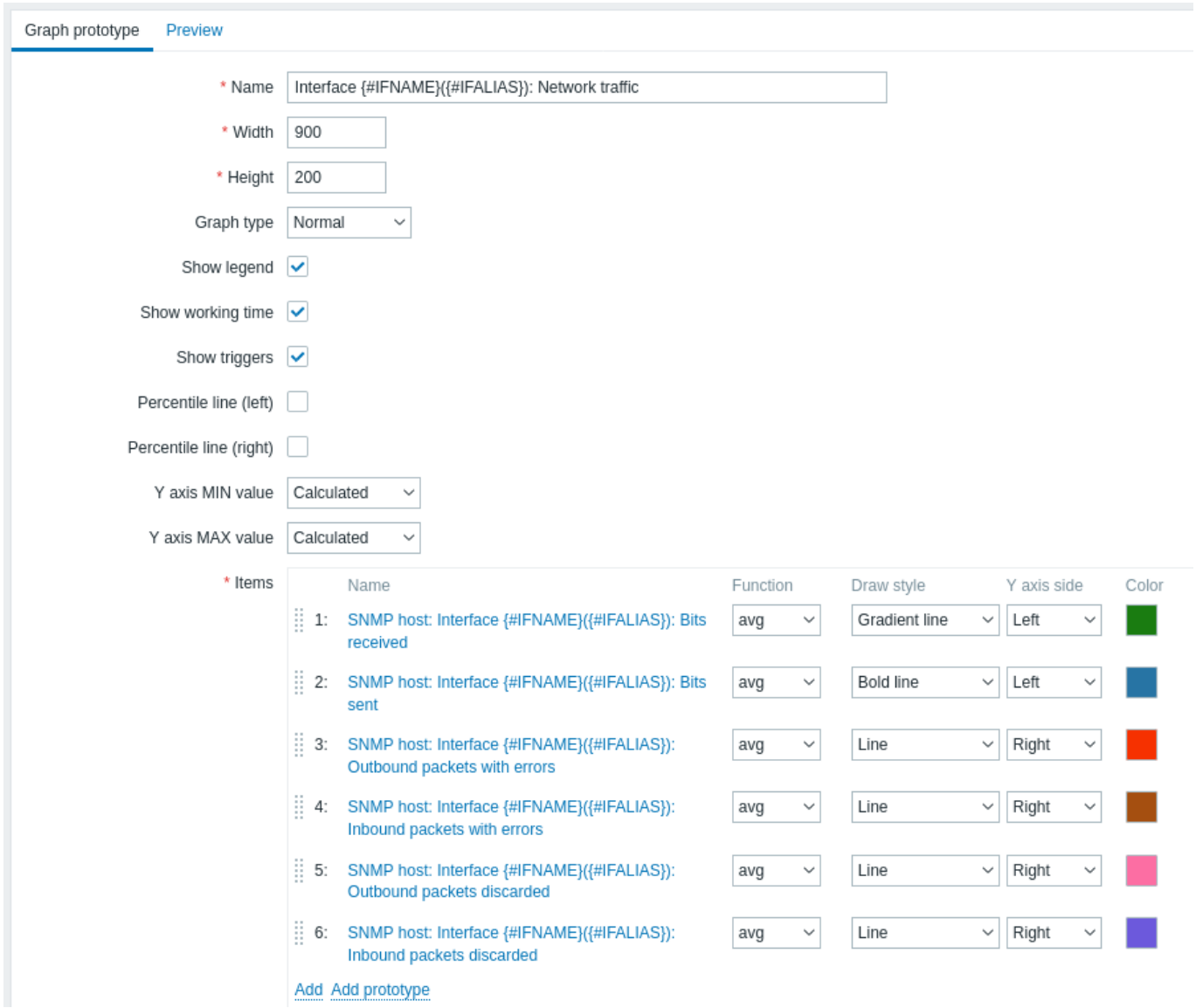
下面截图演示了如何在触发器原型中使用宏:

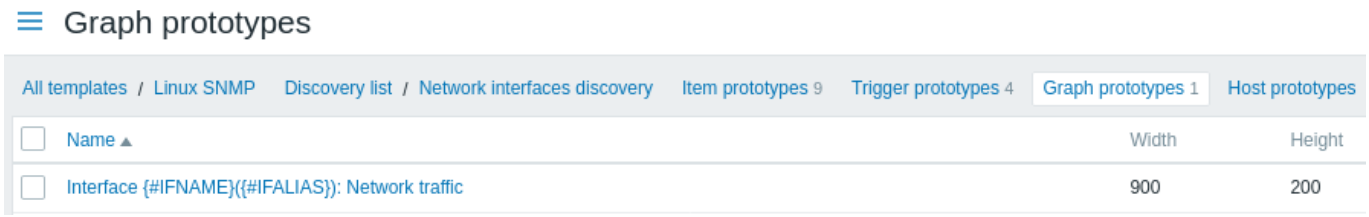




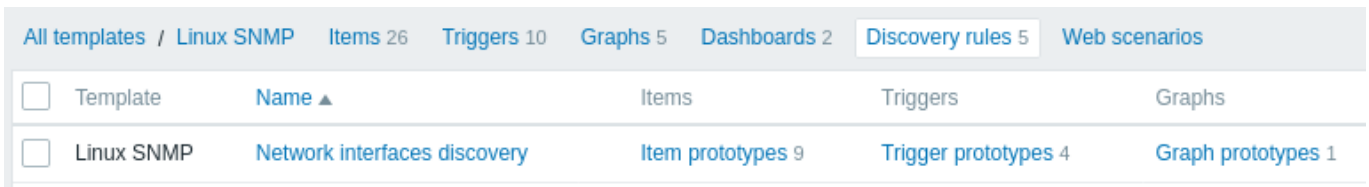
图形原型

下面的截图演示了如何在图形原型中使用宏:





刚才创建的自动发现规则整体展示:



发现的实体

Zabbix server 会基于 SNMP 自动发现规则的返回值来创建实际的监控项、触发器和图表。在主机配置中，这些实体会以一个橙色链接作为前缀来标识，链接指向生成该实体的自动发现规则。

Items

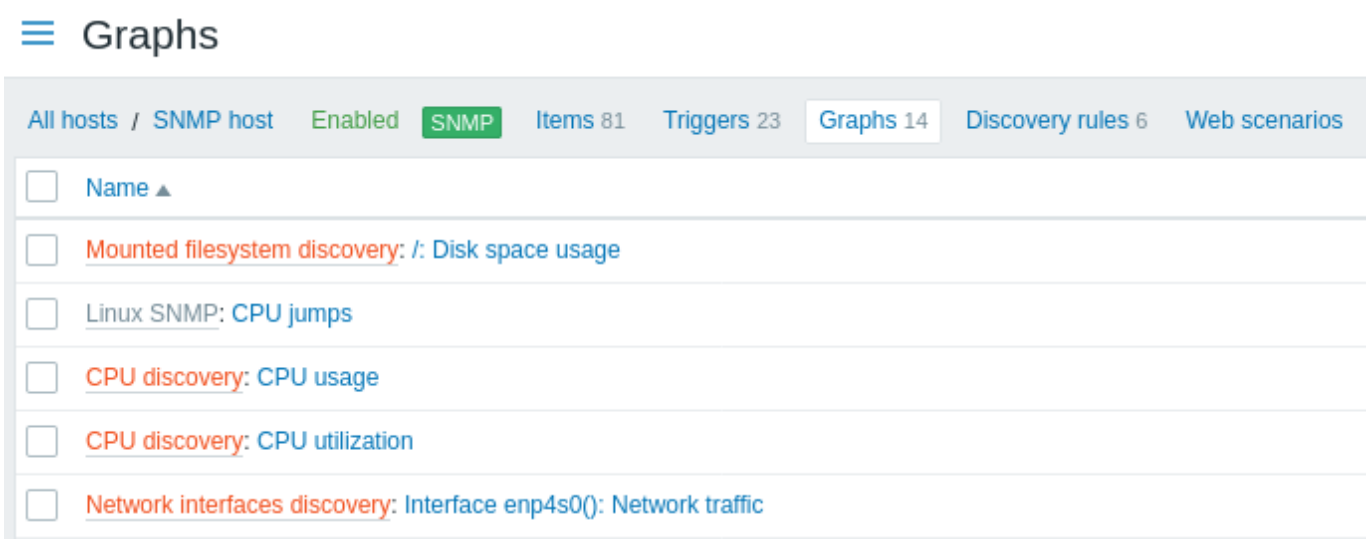
All hosts / SNMP host Enabled SNMP Items 81 Triggers 23 Graphs 14 Discovery rules 6 Web scenarios

Name ▲	Triggers	Key	Interval	History	Trends	Type	Status
Network interfaces discovery: Interface enp4s0(): Bits received	Triggers 1	net.if.in[ifHCInOctets.2]	3m	7d	365d	SNMP agent	Enabled
Network interfaces discovery: Interface enp4s0(): Bits sent	Triggers 1	net.if.out[ifHCOutOctets.2]	3m	7d	365d	SNMP agent	Enabled
Network interfaces discovery: Interface enp4s0(): Inbound packets discarded		net.if.in.discards[ifInDiscards.2]	3m	7d	365d	SNMP agent	Enabled
Network interfaces discovery: Interface enp4s0(): Inbound packets with errors	Triggers 1	net.if.in.errors[ifInErrors.2]	3m	7d	365d	SNMP agent	Enabled
Network interfaces discovery: Interface enp4s0(): Interface type	Triggers 1	net.if.type[ifType.2]	1h	7d	0d	SNMP agent	Enabled
Network interfaces discovery: Interface enp4s0(): Operational status	Triggers 2	net.if.status[ifOperStatus.2]	1m	7d	0	SNMP agent	Enabled
Network interfaces discovery: Interface enp4s0(): Outbound packets discarded		net.if.out.discards[ifOutDiscards.2]	3m	7d	365d	SNMP agent	Enabled
Network interfaces discovery: Interface enp4s0(): Outbound packets with errors	Triggers 1	net.if.out.errors[ifOutErrors.2]	3m	7d	365d	SNMP agent	Enabled
Network interfaces discovery: Interface enp4s0(): Speed	Triggers 2	net.if.speed[ifHighSpeed.2]	5m	7d	0d	SNMP agent	Enabled

Triggers

All hosts / SNMP host Enabled SNMP Items 81 Triggers 23 Graphs 14 Discovery rules 6 Web scenarios

Severity	Value	Name ▲	Operational data	Expression
Information	OK	Network interfaces discovery: Interface enp4s0(): Ethernet has changed to lower speed than it was before **Depends on:** SNMP host: Interface enp4s0(): Link down	Current reported speed: {ITEM.LASTVALUE1}	Problem: **change(**/SNMP host/net.if.speed[ifHighSpeed.2]**)**<0 and **last(**/SNMP host/net.if.speed[ifHighSpeed.2]**)**>0 and (**last(**/SNMP host/net.if.type[ifType.2]**)**=6 or **last(**/SNMP host/net.if.type[ifType.2]**)**=7 or **last(**/SNMP host/net.if.type[ifType.2]**)**=11 or **last(**/SNMP host/net.if.type[ifType.2]**)**=62 or **last(**/SNMP host/net.if.type[ifType.2]**)**=69 or **last(**/SNMP host/net.if.type[ifType.2]**)**=117) and (**last(**/SNMP host/net.if.status[ifOperStatus.2]**)**<>2) Recovery: (**change(**/SNMP host/net.if.speed[ifHighSpeed.2]**)**>0 and **last(**/SNMP host/net.if.speed[ifHighSpeed.2],#2**)**>0) or (**last(**/SNMP host/net.if.status[ifOperStatus.2]**)**=2)
Warning	OK	Network interfaces discovery: Interface enp4s0(): High bandwidth usage **Depends on:** SNMP host: Interface enp4s0(): Link down	In: {ITEM.LASTVALUE1}, out: {ITEM.LASTVALUE3}, speed: {ITEM.LASTVALUE2}	Problem: (**avg(**/SNMP host/net.if.in[ifHCInOctets.2],15m**)**>({$IF.UTIL.MAX:"enp4s0"}/100)***last(**/SNMP host/net.if.speed[ifHighSpeed.2]**)** or **avg(**/SNMP host/net.if.out[ifHCOutOctets.2],15m**)**>({$IF.UTIL.MAX:"enp4s0"}/100)***last(**/SNMP host/net.if.speed[ifHighSpeed.2]**)**) and **last(**/SNMP host/net.if.speed[ifHighSpeed.2]**)**>0 Recovery: **avg(**/SNMP host/net.if.in[ifHCInOctets.2],15m**)**<(({$IF.UTIL.MAX:"enp4s0"}-3)/100)***last(**/SNMP host/net.if.speed[ifHighSpeed.2]**)** and **avg(**/SNMP host/net.if.out[ifHCOutOctets.2],15m**)**<(({$IF.UTIL.MAX:"enp4s0"}-3)/100)***last(**/SNMP host/net.if.speed[ifHighSpeed.2]**)**
Warning	OK	Network interfaces discovery: Interface enp4s0(): High error rate **Depends on:** SNMP host: Interface enp4s0(): Link down	errors in: {ITEM.LASTVALUE1}, errors out: {ITEM.LASTVALUE2}	Problem: **min(**/SNMP host/net.if.in.errors[ifInErrors.2],5m**)**>{$IF.ERRORS.WARN:"enp4s0"} or **min(**/SNMP host/net.if.out.errors[ifOutErrors.2],5m**)**>{$IF.ERRORS.WARN:"enp4s0"} Recovery: **max(**/SNMP host/net.if.in.errors[ifInErrors.2],5m**)**<{$IF.ERRORS.WARN:"enp4s0"}*0.8 and **max(**/SNMP host/net.if.out.errors[ifOutErrors.2],5m**)**<{$IF.ERRORS.WARN:"enp4s0"}*0.8
Average	OK	Network interfaces discovery: Interface enp4s0(): Link down	Current state: {ITEM.LASTVALUE1}	Problem: {$IFCONTROL:"enp4s0"}=1 and **last(**/SNMP host/net.if.status[ifOperStatus.2]**)**=2 and (**last(**/SNMP host/net.if.status[ifOperStatus.2],#1**)**<>**last(**/SNMP host/net.if.status[ifOperStatus.2],#2**)**) Recovery: **last(**/SNMP host/net.if.status[ifOperStatus.2]**)**<>2 or {$IFCONTROL:"enp4s0"}=0



5 JMX 对象的自动发现

概述

可以自动发现 所有 JMX MBean 或 MBean 属性，也可以为这些对象的自动发现指定一个样式。

有必要理解自动发现规则配置中的 MBean 和 MBean 属性之间的区别。MBean 是一个对象，代表一个设备、一个应用程序或任何需要管理的资源。

例如，有一个 MBean，用来表示一个 web 服务器。其属性有连接数、线程数、请求超时时间、HTTP 文件缓存大小、内存使用率等。用通俗语言来类比一下，可以把一台咖啡机定义成一个 MBean，有这些属性会被监控：每杯的水量、某段时间平均消耗水量、每杯所需咖啡豆数量、咖啡豆和水的重新装填时间等。

监控项的键

在自动发现规则 配置中的 类型区域选择 **JMX agent** 。

JMX 对象自动发现有两个可用的键 - jmx.discovery[] 和 jmx.get[]:

监控项的键	返回值	参数	备注
jmx.discovery[< 自动发现模式 >,< 对象名称 >,< 唯一简短描述 >]			

监控项的键			
	此监控项返回一个 JSON 数组，其中包含 LLD 宏，宏描述了 MBean 对象或对象的属性。	自动发现模式 - 任选其一: 属性 (获取 JMX MBean 属性, 默认设置) 或者 *beans* (获取 JMX MBean) 对象名称 - 对象名称样式 (参考 文档) 用于识别获取到的 MBean 名称 (默认为空, 获取所有已注册的 bean) 唯一简短描述 - 一个唯一的描述字段，允许多个 JMX 监控项使用相同的自动发现模式和相同的主机对象名称 (可选)	例子: → jmx.discovery - 获取所有 JMX MBean 属性 → jmx.discovery[beans] - 获取所有 JMX MBean → jmx.discovery[attribu - 获取所有垃圾回收器属性 → jmx.discovery[beans, - 获取所有垃圾回收器 此监控项能返回的 MBean 属性有一些限制，取决于宏名称中的字符长度限制 (支持的字符可使用这个正则表达式来表示: `A-Z0-9_\.`). 例如, 要想发现带有连字符或非 ASCII 字符的 MBean 属性，需要使用 `jmx.get[]`。 从 Zabbix Java gate-way 3.4 开始支持此特性。

监控项的键		
jmx.get[< 自动发现模式 >,< 对象名称 >,< 唯一简短描述 >]		
此监控项返回一个 JSON 数组，包含 MBean 对象或对象的属性，与 jmx.discovery 相比，此监控项并不需要定义 LLD 宏。	自动发现模式 - 任选其一: 属性 (获取 JMX MBean 属性, 默认设置) 或 *beans* (获取 JMX MBean) 对象名称 - 对象名称样式 (参考文档) 用于识别获取到的 MBean 名称 (默认为空, 获取所有已注册的 bean) 唯一简短描述 - 一个唯一的描述字段，允许多个 JMX 监控项使用相同的自动发现模式和相同的主机对象名称 (可选)	一旦使用此监控项，需要自定义低级别自动发现宏，指向 JSONPath 返回的 JSON 数据。 从 Zabbix Java gateway 4.4 开始支持此特性。

Attention:
如果不传递参数，则会向 JMX 请求所有的 MBean 属性。如果不指定 JMX 自动发现的参数，或者试图接收一个很大范围内的所有属性，比如 `*:type=*,name=*`，此两者可能会导致潜在的性能问题。

使用 jmx.discovery

此监控项返回一个 JSON 对象，其中包含低级别自动发现的宏，用于描述 MBean 对象或对象的属性，比如 MBean 属性的自动发现 (为了清晰重新排版):

```
[
    {
        "{#JMXVALUE}":"0",
        "{#JMXTYPE}":"java.lang.Long",
```

```
        "{#JMXOBJ}":"java.lang:type=GarbageCollector,name=PS Scavenge",
        "{#JMXDESC}":"java.lang:type=GarbageCollector,name=PS Scavenge,CollectionCount",
        "{#JMXATTR}":"CollectionCount"
    },
    {
        "{#JMXVALUE}":"0",
        "{#JMXTYPE}":"java.lang.Long",
        "{#JMXOBJ}":"java.lang:type=GarbageCollector,name=PS Scavenge",
        "{#JMXDESC}":"java.lang:type=GarbageCollector,name=PS Scavenge,CollectionTime",
        "{#JMXATTR}":"CollectionTime"
    },
    {
        "{#JMXVALUE}":"true",
        "{#JMXTYPE}":"java.lang.Boolean",
        "{#JMXOBJ}":"java.lang:type=GarbageCollector,name=PS Scavenge",
        "{#JMXDESC}":"java.lang:type=GarbageCollector,name=PS Scavenge,Valid",
        "{#JMXATTR}":"Valid"
    },
    {
        "{#JMXVALUE}":"PS Scavenge",
        "{#JMXTYPE}":"java.lang.String",
        "{#JMXOBJ}":"java.lang:type=GarbageCollector,name=PS Scavenge",
        "{#JMXDESC}":"java.lang:type=GarbageCollector,name=PS Scavenge,Name",
        "{#JMXATTR}":"Name"
    },
    {
        "{#JMXVALUE}":"java.lang:type=GarbageCollector,name=PS Scavenge",
        "{#JMXTYPE}":"javax.management.ObjectName",
        "{#JMXOBJ}":"java.lang:type=GarbageCollector,name=PS Scavenge",
        "{#JMXDESC}":"java.lang:type=GarbageCollector,name=PS Scavenge,ObjectName",
        "{#JMXATTR}":"ObjectName"
    }
]
```

又比如 MBean 的自动发现 (为了清晰重新排版):

```
[
    {
        "{#JMXDOMAIN}":"java.lang",
        "{#JMXTYPE}":"GarbageCollector",
        "{#JMXOBJ}":"java.lang:type=GarbageCollector,name=PS Scavenge",
        "{#JMXNAME}":"PS Scavenge"
    }
]
```

支持的宏

支持在自动发现规则的过滤器和监控项、触发器、图形的原型中使用下面的宏：

宏	描述
MBean 属性的自动发现	
{#JMXVALUE}	属性的值。
{#JMXTYPE}	属性的类型。
{#JMXOBJ}	对象名称。
{#JMXDESC}	包含属性名称的对象名称。
{#JMXATTR}	属性名称。
MBean 的自动发现	
{#JMXDOMAIN}	MBean 的域。(*Zabbix* 的保留名称)
{#JMXOBJ}	对象名称。(*Zabbix* 的保留名称)
{#JMX<key property>}	MBean 属性 (类似 {#JMXTYPE}, {#JMXNAME}) (参考下面的限制)。

限制

从 MBean 属性的名称中创建 LLD 宏的名称时，有一些规则上的限制:

- 属性名称被改为大写
- 如果 LLD 宏名称中包含不支持的字符，则属性名称被忽略 (未生成 LLD 宏)。支持的字符可用下面的正则表达式来表示: `A-Z0-9_\.`
- 如果属性名称是”obj” 或”domain” 则会被忽略因为这与预留的 Zabbix 属性 {#JMXOBJ} 和 {#JMXDOMAIN} 的值重叠 (从 Zabbix 3.4.3 起支持此特性。)

请思考这个 jmx.discovery (使用”beans” 模式) 的例子。MBean 中定义了下面的属性:

```
name=test
  =Type
attributes []=1,2,3
Name=NameOfTheTest
domAin=some
```

作为 JMX 自动发现的结果，会产生下面的 LLD 宏:

- {#JMXDOMAIN} - Zabbix 内部创建, 描述了 MBean 的域
- {#JMXOBJ} - Zabbix 内部创建, 描述了 MBean 对象
- {#JMXNAME} - 从” 名称” 属性中创建

被忽略的属性有:

- тип : 该名称包含不支持的字符 (非 ASCII)
- attributes[] : 该名称包含不支持的字符 (不支持方括号)
- Name: 已经定义过了 (name=test)
- domAin: Zabbix 的保留名称

示例

关于使用 Mbean 创建 LLD 规则，下面来看两个更具体的例子。要理解收集 Mbean 数据的 LLD 规则和收集 Mbean 属性数据的 LLD 规则之间的区别，请看下面的表格:

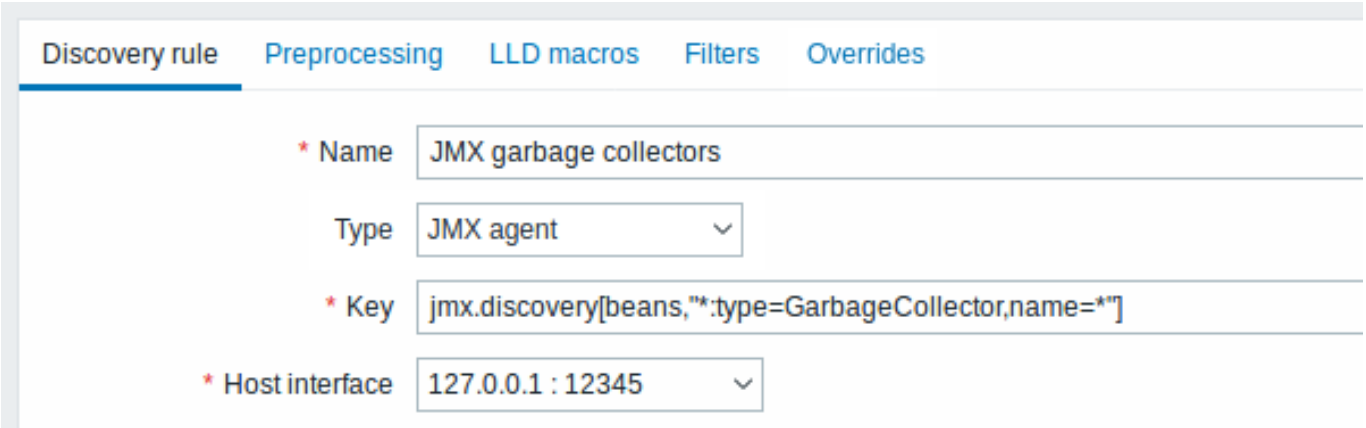
MBean1	MBean2	MBean3
MBean1Attribute1	MBean2Attribute1	MBean3Attribute1
MBean1Attribute2	MBean2Attribute2	MBean3Attribute2
MBean1Attribute3	MBean2Attribute3	MBean3Attribute3

例 1: Mbean 的自动发现

此规则会返回三个对象：该列的第一行: MBean1, MBean2, MBean3.

更多关于对象的信息请查阅 *MBean* 的自动发现小节中的支持的宏 表格。

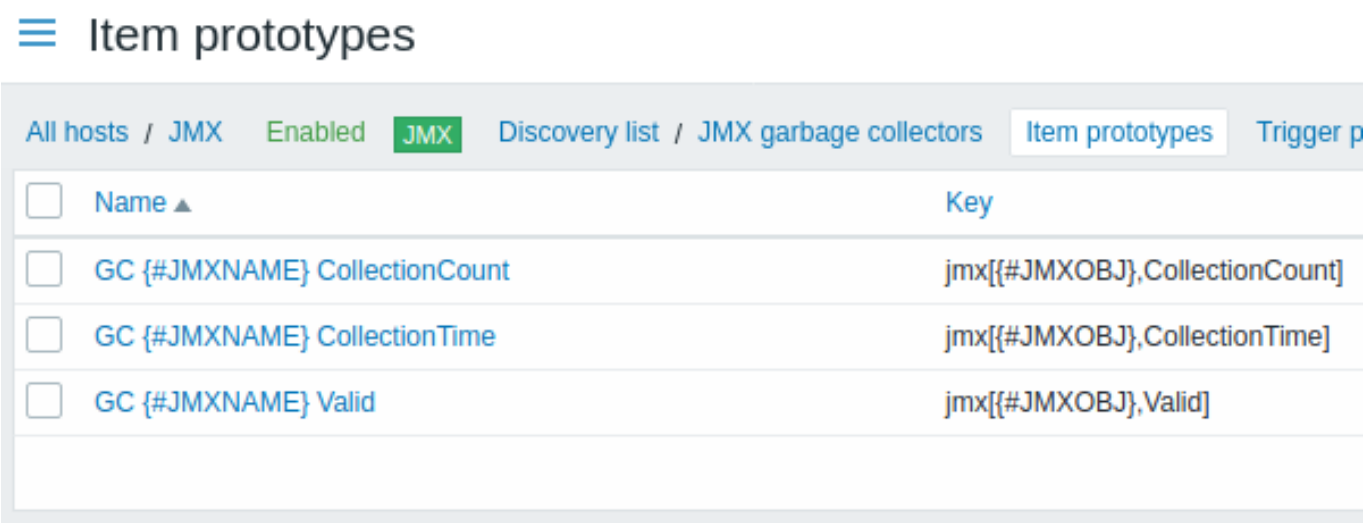
收集 Mbean 数据 (不包含属性) 的自动发现规则配置如下:



这里使用的键:

```
jmx.discovery[beans,"*:type=GarbageCollector,name=*"]
```

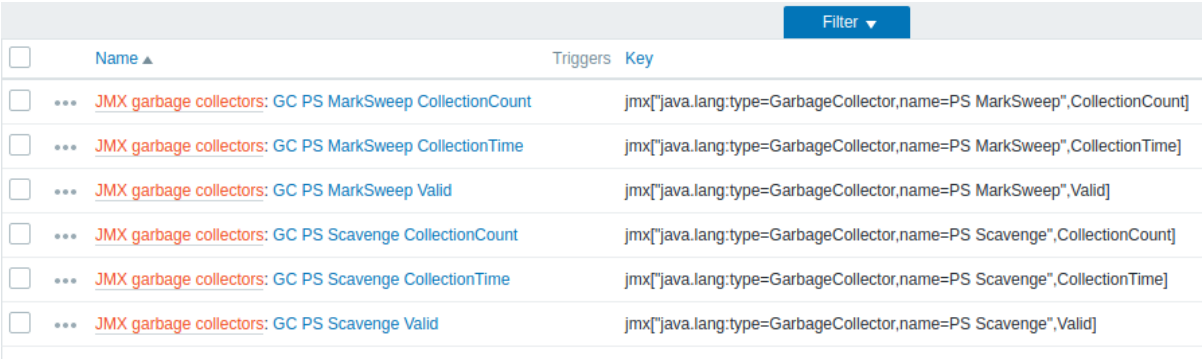
所有的垃圾回收器都会被发现，但不包含它们的属性数据。由于垃圾回收器的属性集都是相同的，所以可以在监控项原型中使用属性，像下面这样:



这里使用的键:

```
jmx[{#JMXOBJ},CollectionCount]
jmx[{#JMXOBJ},CollectionTime]
jmx[{#JMXOBJ},Valid]
```

LLD 自动发现规则会产生近似于下面的结果 (两个垃圾回收器的监控项被发现):

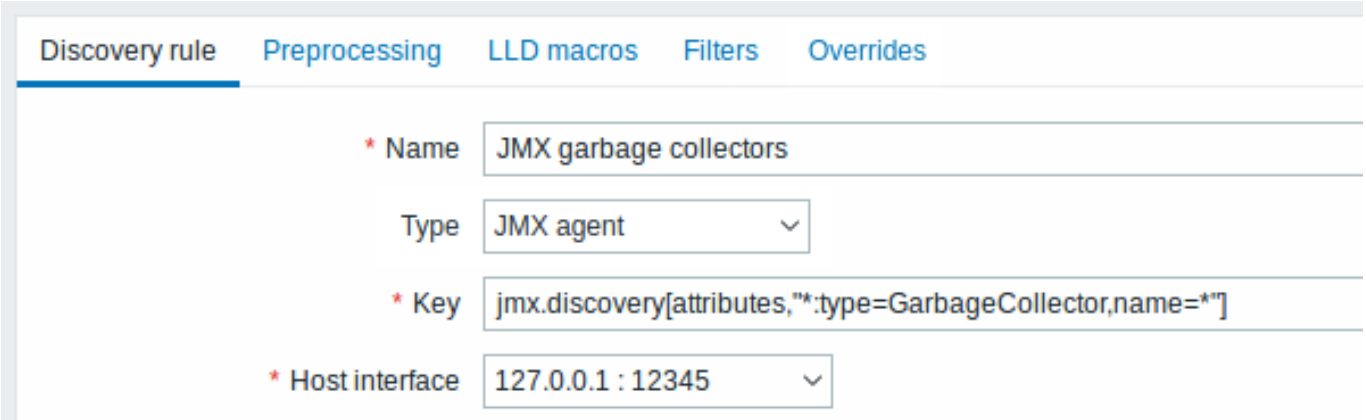


例 2: Mbean 属性的自动发现

此规则会返回下列九个对象：MBean1Attribute1，MBean2Attribute1，Mbean3Attribute1,MBean1Attribute2,MBean2Attribute2, Mbean3Attribute2, MBean1Attribute3, MBean2Attribute3, Mbean3Attribute3.

更多关于对象的信息请参考 *Mbean* 属性的自动发现小节中的支持的宏 表格。

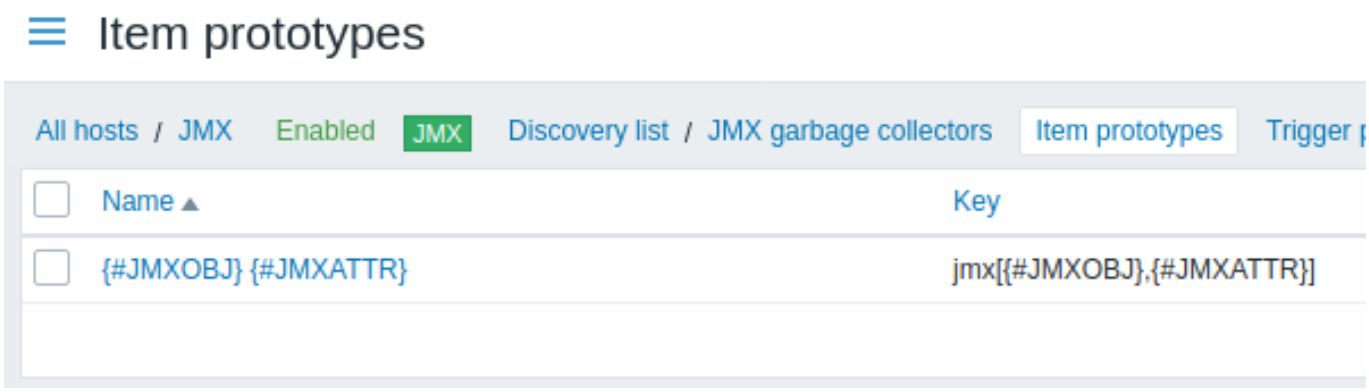
收集 Mbean 属性数据的自动发现规则配置如下：



这里使用的键:

```
jmx.discovery[attributes,"*:type=GarbageCollector,name=*"]
```

所有垃圾回收器连同其监控项的属性都会被发现。



在这个特定场景下，对于每个 MBean 属性，都会从原型中创建一个监控项。此配置的主要缺点是无法从触发器原型中创建触发器，因为只有一个监控项原型对应所有的属性。所以此配置可以用于数据采集，但不推荐用于自动监控。

使用 jmx.get

`jmx.get[]` 与 `jmx.discovery[]` 监控项很相似，但此监控项不会把 Java 对象的属性转换成低级别自动监控的宏的名称。所以此监控项的返回值没有一些限制 ，具体就是跟 LLD 宏的名称有关的限制，比如不能使用连字符或非 ASCII 字符的名称。

如果使用 `jmx.get[]` 做自动发现，则可在自动发现规则的自定义 LLD 宏选项卡中分别对宏进行配置，使用 JSONPath 映射到所需的值上。

MBean 的自动发现

自动发现的监控项: `jmx.get[beans,"com.example:type=*,*"]`

返回数据:

```
[
    {
        "object": "com.example:type=Hello,data-src=data-base,  =    ",
        "domain": "com.example",
        "properties": {
            "data-src": "data-base",
            "  ": "    ",
            "type": "Hello"
        }
    },
    {
        "object": "com.example:type=Atomic",
        "domain": "com.example",
        "properties": {
            "type": "Atomic"
        }
    }
]
```

MBean 属性的自动发现

自动发现监控项: `jmx.get[attributes,"com.example:type=*,*"]`

返回数据:

```
[
    {
        "object": "com.example:type=*",
        "domain": "com.example",
        "properties": {
            "type": "Simple"
        }
    },
    {
        "object": "com.zabbix:type=yes,domain=zabbix.com,data-source=/dev/rand,  =    ,obj=true",
        "domain": "com.zabbix",
        "properties": {
            "type": "Hello",
            "domain": "com.example",
```

```
            "data-source": "/dev/rand",
            "   ": "     ",
            "obj": true
        }
    }
]
```

6 IPMI 传感器的自动发现

概述

IPMI 传感器可被自动发现.

要实现此功能, 可结合使用:

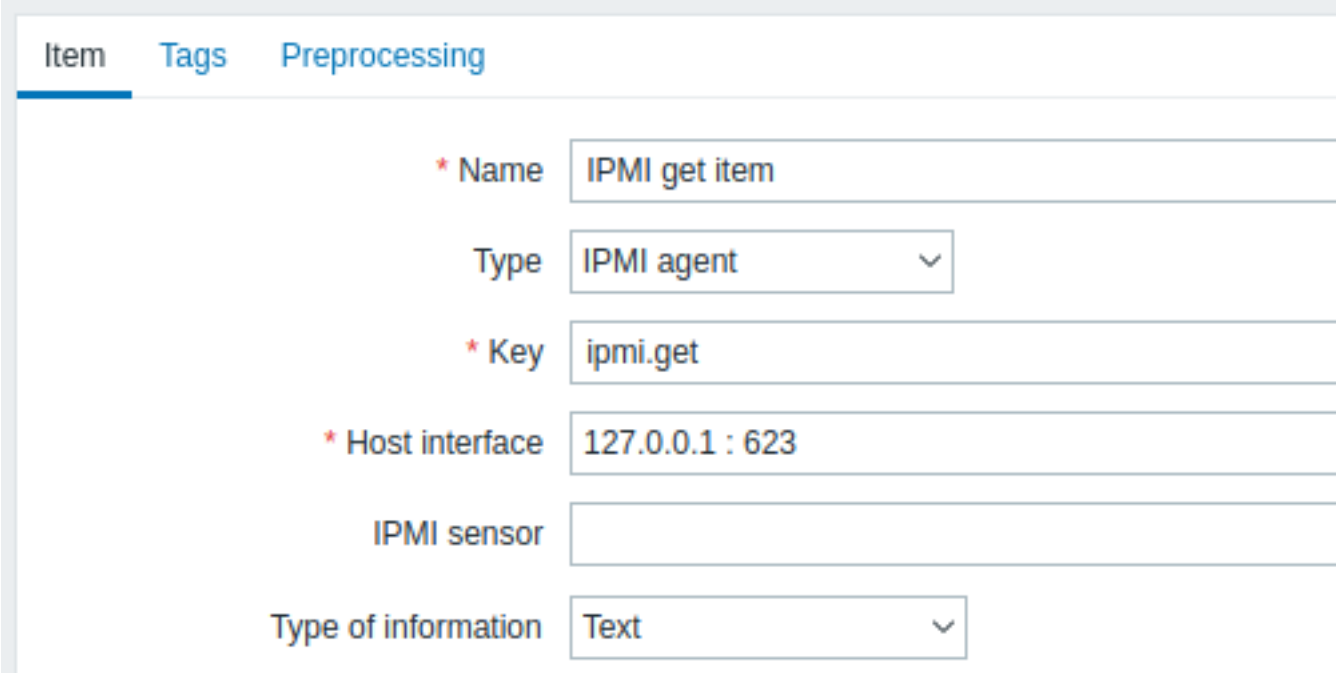
- IPMI 监控项 `ipmi.get` (从 Zabbix **5.0.0** 起支持) 作为主监控项
- 依赖型的低级别自动发现规则和低级别自动发现监控项原型

配置

主监控项

使用下面的键创建一个 IPMI 监控项:

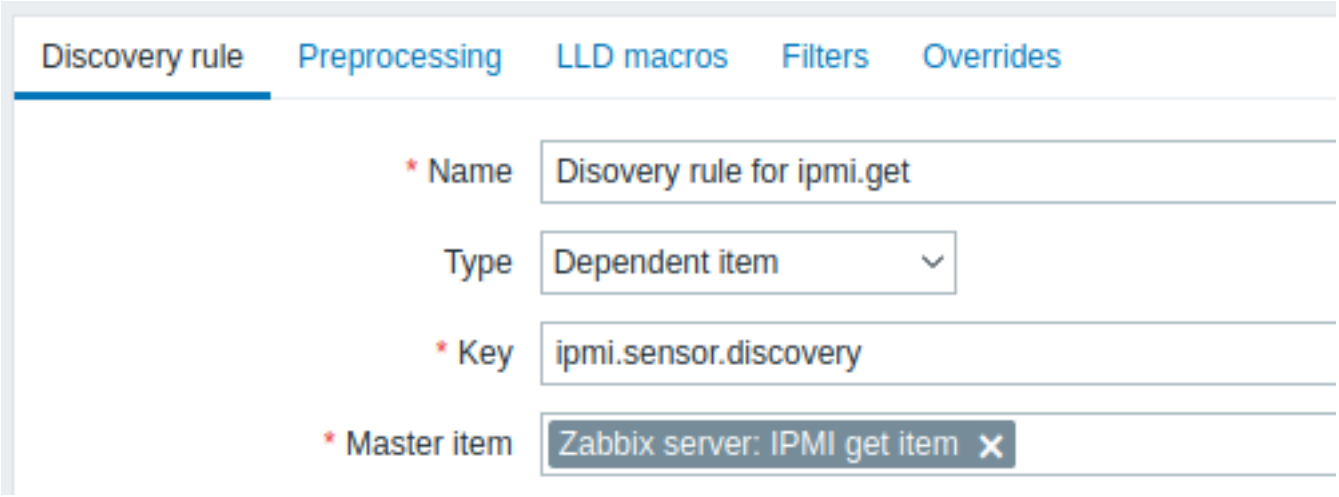
`ipmi.get`



设置信息类型为” 文本”，用于接收可能获取的大块 JSON 数据。

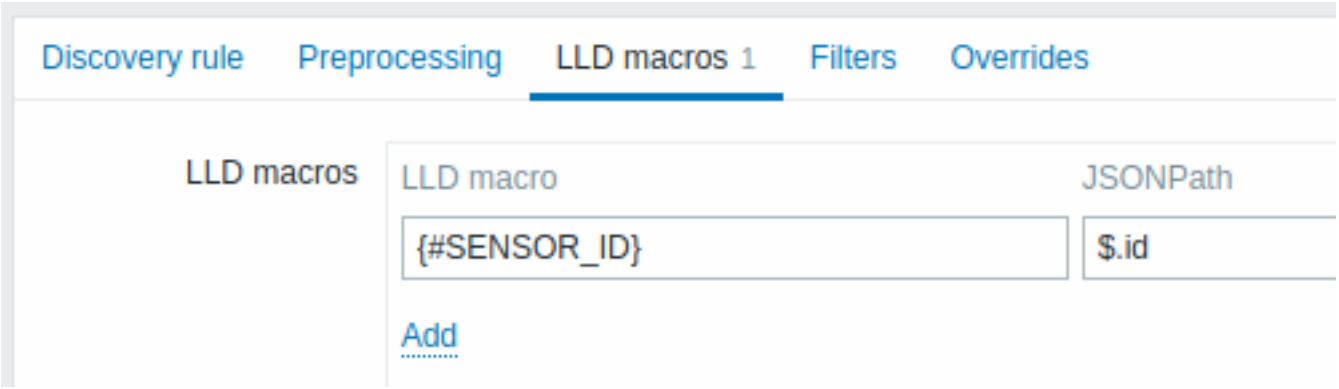
依赖型 LLD 规则

创建一个低级别自动发现规则，类型选择” 依赖型监控项”:



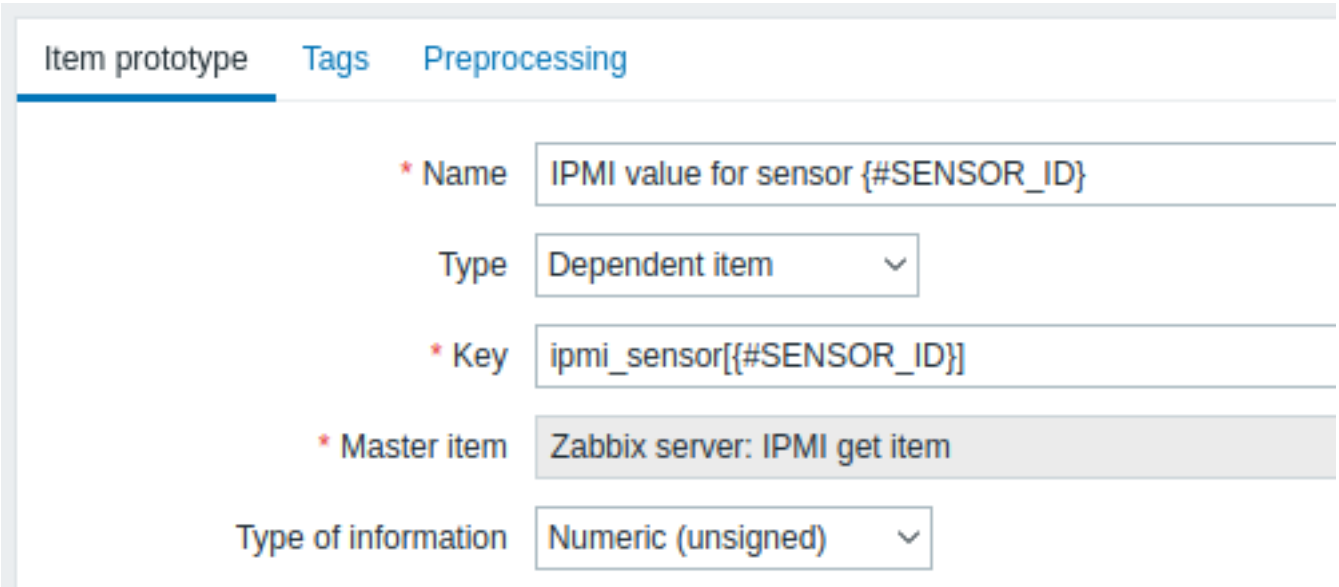
主监控项选择之前创建的 `ipmi.get` 监控项.

在”LLD 宏” 选项卡中用对应的 JSONPath 自定义一个宏:



依赖型监控项原型

在此 LLD 规则中创建一个监控项原型，类型选择” 依赖型监控项”. 此原型的主监控项选择之前创建的监控项 `ipmi.get`.



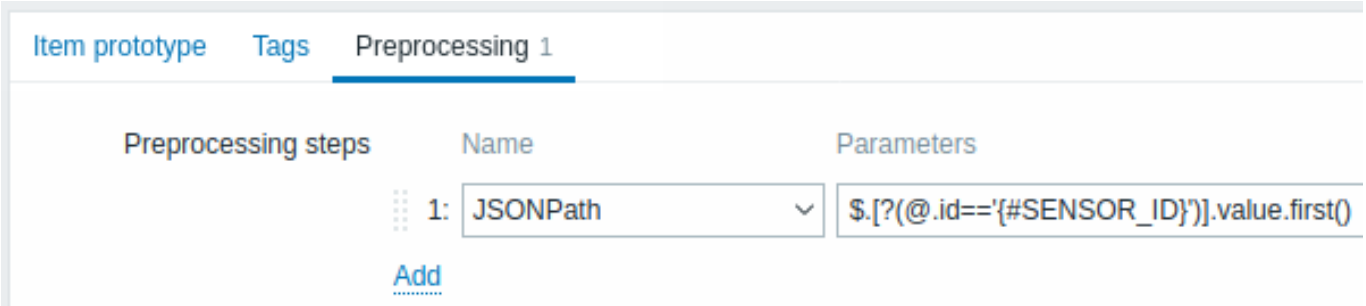
注意 {#SENSOR_ID} 宏在监控项原型的名称和键中的使用方式:

- 名称: 传感器 {#SENSOR_ID} 的 IPMI 值
- 键: ipmi_sensor[{#SENSOR_ID}]

信息类型选择 *Numeric (unsigned)*.

在监控项原型” 预处理” 选项卡中选择 JSONPath 并使用下面的 JSONPath 表达式作为参数:

```
$.[?(@.id=='{#SENSOR_ID}')].value.first()
```



一旦自动发现开始执行，每个 IPMI 传感器会创建一个对应的监控项。这个监控项会返回对应传感器的整数值.

7 系统服务的自动发现

概述

系统单元 (服务, 默认设置) 可被 Zabbix 自动发现.

监控项的键

在自动发现规则 中使用这个监控项

`systemd.unit.discovery`

Attention:
此监控项 的键只有 Zabbix agent 2 版本支持.

此监控项返回一个包含系统单元的 JSON 数据，如下所示:

```
[{
    "{#UNIT.NAME}": "mysqld.service",
    "{#UNIT.DESCRIPTION}": "MySQL Server",
    "{#UNIT.LOADSTATE}": "loaded",
    "{#UNIT.ACTIVESTATE}": "active",
    "{#UNIT.SUBSTATE}": "running",
    "{#UNIT.FOLLOWED}": "",
    "{#UNIT.PATH}": "/org/freedesktop/systemd1/unit/mysqld_2eservice",
    "{#UNIT.JOBID}": 0,
    "{#UNIT.JOBTYPE}": ""
    "{#UNIT.JOBPATH}": "/",
    "{#UNIT.UNITFILESTATE}": "enabled"
}, {
    "{#UNIT.NAME}": "systemd-journald.socket",
    "{#UNIT.DESCRIPTION}": "Journal Socket",
    "{#UNIT.LOADSTATE}": "loaded",
    "{#UNIT.ACTIVESTATE}": "active",
    "{#UNIT.SUBSTATE}": "running",
    "{#UNIT.FOLLOWED}": "",
    "{#UNIT.PATH}": "/org/freedesktop/systemd1/unit/systemd_2djournald_2esocket",
    "{#UNIT.JOBID}": 0,
    "{#UNIT.JOBTYPE}": "",
    "{#UNIT.JOBPATH}": "/"
    "{#UNIT.UNITFILESTATE}": "enabled"
}]
```

发现的禁用 systemd 单元

从 Zabbix 6.0.1 开始，还可以发现禁用的 systemd 单元。在这种情况下，生成的 JSON 中会返回三个宏：

- ·{#UNIT.PATH}
- ·{#UNIT.ACTIVESTATE}
- ·{#UNIT.UNITFILESTATE}.

Attention:
要从禁用的 systemd 单元的原型创建监控项和触发器，请确保调整（或删除）针对 {#UNIT.ACTIVESTATE} 和 {#UNIT.UNITFILESTATE} 的禁止 LLD 过滤器

支持的宏

在自动发现规则的过滤器 和监控项、触发器、图形的原型配置中支持的宏如下所示:

宏	描述
{#UNIT.NAME}	主单元名称。
{#UNIT.DESCRIPTION}	通俗易懂的描述。
{#UNIT.LOADSTATE}	加载状态 (比如单元文件是否成功加载)
{#UNIT.ACTIVESTATE}	激活状态 (比如单元是否已启动)
{#UNIT.SUBSTATE}	子状态 (激活状态的另一个更加细致的版本，根据单元类型不同而不同, 而激活状态更加通用)
{#UNIT.FOLLOWED}	此单元所跟踪的单元的状态; 如果没有则显示空字符串。
{#UNIT.PATH}	单元对象路径。
{#UNIT.JOBID}	如果在任务单元中有排队的任务，则显示任务 ID 的数字; 否则为 0。
{#UNIT.JOBTYPE}	任务类型。
{#UNIT.JOBPATH}	任务对象路径。
{#UNIT.UNITFILESTATE}	单元文件的安装状态。

监控项原型

监控项原型基于系统服务自动发现来创建, 比如:

- 监控项名称: {#UNIT.DESCRIPTION}; 监控项的键: `systemd.unit.info["{#UNIT.NAME}"]`
- 监控项名称: {#UNIT.DESCRIPTION}; 监控项的键: `systemd.unit.info["{#UNIT.NAME}",LoadState]`

从 Zabbix 4.4 开始支持 `systemd.unit.info` agent 监控项.

8 Windows 服务的自动发现

概述

与文件系统自动发现相似，Windows 服务同样可被 Zabbix 自动发现.

监控项的键

此监控项在自动发现规则 中使用的键是

`service.discovery`

从 Zabbix Windows agent 3.0 起支持此监控项。

支持的宏

在自动发现规则的过滤器 和监控项、触发器、图形的原型中支持使用下列宏:

宏	描述
{#SERVICE.NAME}	服务名称。
{#SERVICE.DISPLAYNAME}	显示服务名称。
{#SERVICE.DESCRIPTION}	服务描述。
{#SERVICE.STATE}	服务状态的数字值: 0 - 运行中 1 - 暂停 2 - 开始挂起 3 - 暂停挂起 4 - 继续挂起 5 - 停止挂起 6 - 已停止 7 - 未知
{#SERVICE.STATENAME}	服务状态名称 (运行中, 暂停, 开始挂起, 暂停挂起, 继续挂起, 停止挂起, 已停止或 未知)。
{#SERVICE.PATH}	服务的路径。
{#SERVICE.USER}	服务的用户。

宏	描述
{#SERVICE.STARTUP}	服务启用类型的数字值: 0 - 自动 1 - 延迟自动 2 - 手动 3 - 已禁用 4 - 未知
{#SERVICE.STARTUPNAME}	服务启动类型名称 (自动, 延迟自动, 手动, 已禁用, 未知)。
{#SERVICE.STARTUPTRIGGER}	如果有的话，显示下列服务启动类型的数字值: 0 - 没有启动触发器 1 - 有启动触发器 从 Zabbix 3.4.4 起支持此宏. 此宏用来发现这些服务启动类型很有用：自动 *(*触发器启动*)*, 延迟自动 *(*触发器启动*)* 和 手动 *(*触发器启动*)*。

可以基于 Windows 服务的自动发现来创建监控项 原型，比如

`service.info[{#SERVICE.NAME},<param>]`

其中 `param` 接受这些值: *state*, *displayname*, *path*, *user*, *startup* 或 *description*。

比如, 要获取服务的显示名称，可以使用”service.info[{#SERVICE.NAME},displayname]” 监控项. 如果 `param` 的值没有在 (”service.info[{#SERVICE.NAME}]”) 中指定，则使用默认的 *state* 参数。

9 Windows 性能计数器实例的自动发现

概述

支持对 Windows 性能计数器的实例进行自动发现。此特性对于多实例性能计数器很有用。

监控项的键

在自动发现规则 中使用的监控项是

`perf_instance.discovery[object]`

或者可以提供对象名称，仅支持英文名称，此名称不受操作系统本地化设置控制:

`perf_instance_en.discovery[object]`

例如:

```
perf_instance.discovery[Processador]
perf_instance_en.discovery[Processor]
```

从 Zabbix Windows agent 5.0.1 起支持这些监控项.

支持的宏

自动发现会返回 {#INSTANCE} 宏的所有对象实例, 可用于 perf_count 和 perf_count_en 监控项原型中。

```
[
    {"{#INSTANCE}":"0"},
    {"{#INSTANCE}":"1"},
    {"{#INSTANCE}":"_Total"}
]
```

例如, 假设自动发现规则中使用的监控项的键是：

`perf_instance.discovery[Processor]`

则可以创建这样的监控项原型：

`perf_counter["\Processor({#INSTANCE})\% Processor Time"]`

注意：

- 如果指定的对象找不到或者不支持可变实例，则自动发现监控项会变成 NOTSUPPORTED 状态。
- 如果指定的对象支持可变实例，但当前不存在任何实例，则会返回一个空的 JSON 数组。
- 重复的实例会被忽略。

10 利用 **WMI** 查询执行自动发现

概述

WMI 是一种 Windows 中的接口，功能强大，可用于获取各种信息，比如 Windows 组件、服务、状态和安装的软件。

WMI 可用于物理硬盘的自动发现和相关性能数据的采集、网络接口的自动发现、Hyper-V guest 的自动发现、监控 Windows 操作系统中的服务和其它实体。

此类低级别自动发现 通过使用 WQL 查询实现，其结果会自动转换成一个匹配低级别自动发现的 JSON 格式的 JSON 对象。

监控项的键

自动发现规则中使用的监控项是

`wmi.getall[<命 名 空 间>,<查 询>]`

此监控项 将查询结果转换成一个 JSON 数组. 比如:

```
select * from Win32_DiskDrive where Name like '%PHYSICALDRIVE%'
```

会转换成下面的数据:


```
[
    {
        "DeviceID" : "\\.\PHYSICALDRIVE0",
        "BytesPerSector" : 512,
        "Capabilities" : [
            3,
            4
        ],
        "CapabilityDescriptions" : [
            "Random Access",
            "Supports Writing"
        ],
        "Caption" : "VBOX HARDDISK ATA Device",
        "ConfigManagerErrorCode" : "0",
        "ConfigManagerUserConfig" : "false",
        "CreationClassName" : "Win32_DiskDrive",
        "Description" : "Disk drive",
        "FirmwareRevision" : "1.0",
        "Index" : 0,
        "InterfaceType" : "IDE"
    },
    {
        "DeviceID" : "\\.\PHYSICALDRIVE1",
        "BytesPerSector" : 512,
        "Capabilities" : [
            3,
            4
        ],
        "CapabilityDescriptions" : [
            "Random Access",
            "Supports Writing"
        ],
        "Caption" : "VBOX HARDDISK ATA Device",
        "ConfigManagerErrorCode" : "0",
        "ConfigManagerUserConfig" : "false",
        "CreationClassName" : "Win32_DiskDrive",
        "Description" : "Disk drive",
        "FirmwareRevision" : "1.0",
        "Index" : 1,
        "InterfaceType" : "IDE"
    }
]
```


从 Zabbix Windows agent 4.4 起支持此监控项。

低级别自动发现的宏

就算低级别自动发现返回的 JSON 数组中未创建任何宏，也可以使用一个额外步骤来定义宏，通过使用自定义 LLD 宏，将 JSONPath 指向返回的 JSON 值。

然后这些宏就可用于创建监控项、触发器等原型。

11 利用 ODBC SQL 查询执行自动发现

概述

此类低级别自动发现 通过使用 SQL 查询实现，其结果自动转换成一个匹配低级别自动发现的 JSON 格式的 JSON 对象。

监控项的键

设置监控项类型为” 数据库监控” 来执行 SQL 查询。所以为了让一个” 数据库监控” 类型的自动发现规则正常工作，可以参考ODBC 监控页面中的大部分指南。

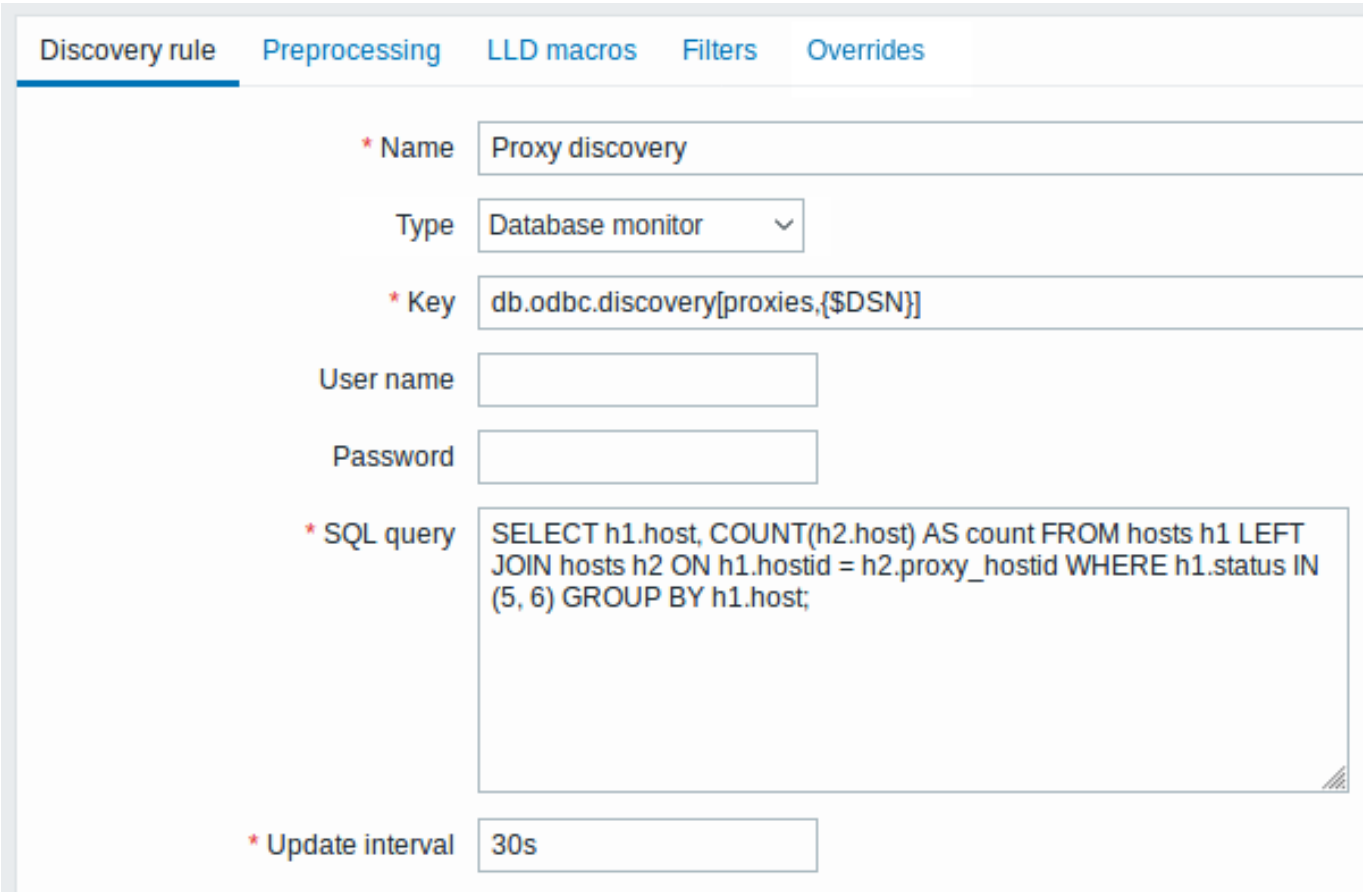
” 数据库监控” 类型的自动发现规则会用到两个键：

- **db.odbc.discovery**[< 唯一简短描述 >,<dsn(数据来源名称)>,< 连接字符串 >] - 此监控项将 SQL 查询结果转换为一个 JSON 数组，其中表的字段名称会转换为宏的名称，宏的名称与发现的对应值成对匹配。这些宏可用于创建监控项和触发器等原型。另请参阅: 使用 db.odbc.discovery。
- **db.odbc.get**[< 唯一简短描述 >,<dsn(数据来源名称)>,< 连接字符串 >] - 此监控项将 SQL 查询结果转换为一个 JSON 数组，保留原始表字段名称作为输入框的名称，以 JSON 格式表示，并与对应的已发现的值成对匹配。相比于 `db.odbc.discovery`[]，此监控项不在返回的 JSON 数组中创建低级别自动发现的宏，因此无需检查表字段名称是否是有效的宏名称。通过使用自定义 LLD 宏，将 JSONPath 指向对应 JSON 值，可作为额外步骤按需定义低级别自动发现的宏。另请参阅: 使用 db.odbc.get。

使用 db.odbc.discovery

让我们看一个 SQL 查询转换为 JSON 数组的真实案例。思考一下如何在 Zabbix 数据库上使用 ODBC 查询对 Zabbix proxy 执行低级别自动发现。此功能对于自动创建”zabbix[proxy,<name>,lastaccess]” 内部监控项 并监控存活的 proxy 很有用。

下面开始配置自动发现规则：



所有强制输入区域均标记为红色星号。

此处 Zabbix 数据库上的查询用于查询所有 Zabbix proxy，连同 proxy 所监控的主机数量. 比如，主机数量可用于过滤掉没有监控任何主机的 proxy：

```
mysql> SELECT h1.host, COUNT(h2.host) AS count FROM hosts h1 LEFT JOIN hosts h2 ON h1.hostid = h2.proxy_ho
+---------+-------+
| host    | count |
+---------+-------+
| Japan 1 |     5 |
| Japan 2 |    12 |
| Latvia  |     3 |
+---------+-------+
3 rows in set (0.01 sec)
```

通过"db.odbc.discovery[,{$DSN}]" 监控项的内部工作机制, 查询结果会自动转换为下面的 JSON 数组：

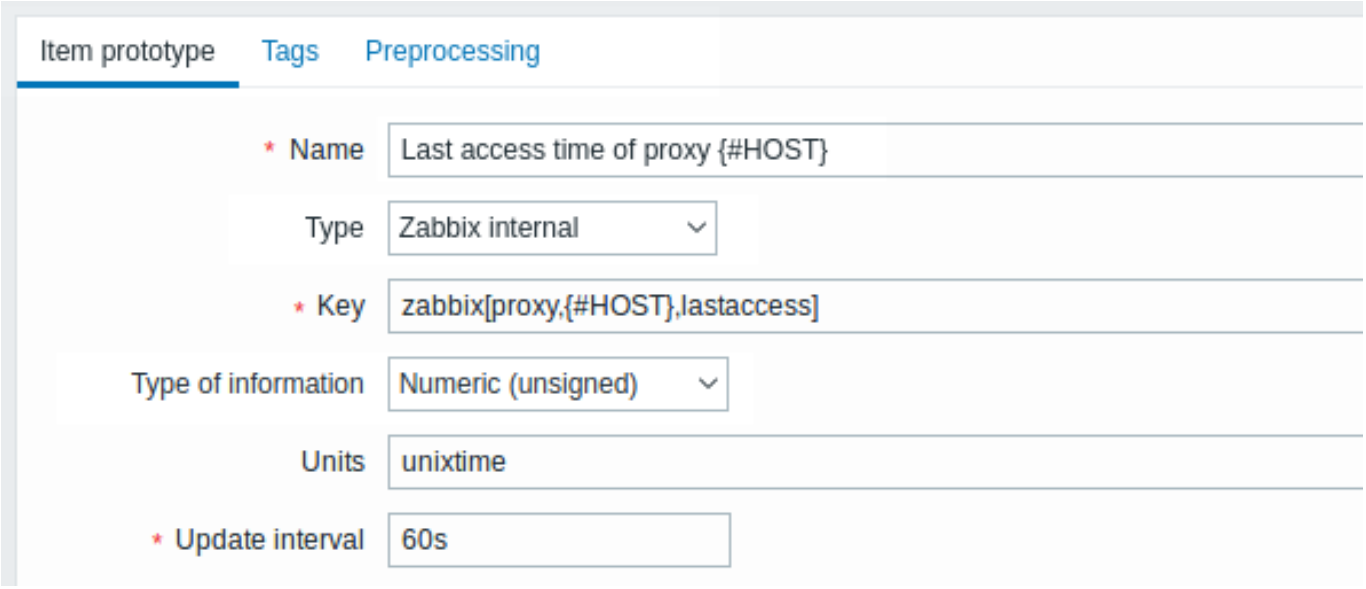
```
[
    {
        "{#HOST}": "Japan 1",
        "{#COUNT}": "5"
    },
    {
        "{#HOST}": "Japan 2",
        "{#COUNT}": "12"
    },
    {
        "{#HOST}": "Latvia",
        "{#COUNT}": "3"
    }
]
```

可以看到字段名称变为宏的名称, 所选的记录变成这些宏的值。

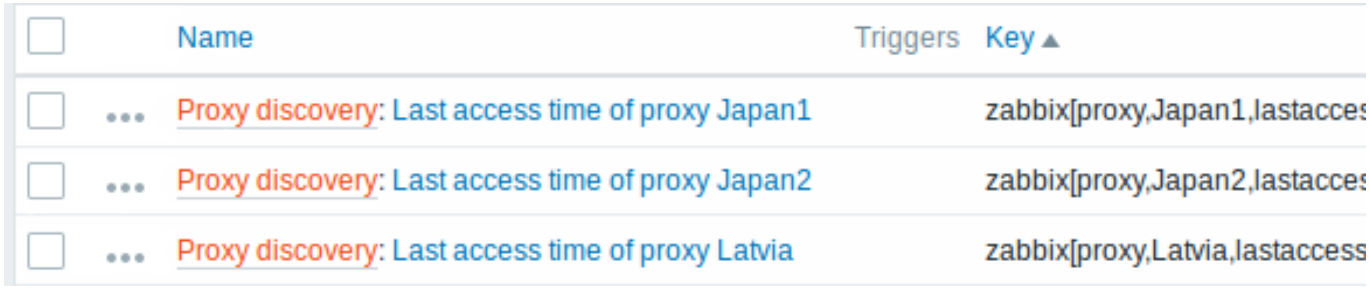
Note:
如果通过这种方式展示字段名称转换为宏名称不是很明显，那建议在上面的例子中使用字段别名，比如"COUNT(h2.host) AS count"。
如果字段名称无法转换为有效的宏名称，则自动发现规则变为 unsupport(不支持的) 状态，其错误消息显示不合规字段的编号.
如果需要额外帮助, debug 级别 DebugLevel=4 的 Zabbix server 日志文件中可找到获取的字段名称：

```
$ grep db.odbc.discovery /tmp/zabbix_server.log
 ...
 23876:20150114:153410.856 In db_odbc_discovery() query:'SELECT h1.host, COUNT(h2.host) FROM hosts h1 L
 23876:20150114:153410.860 db_odbc_discovery() column[1]:'host'
 23876:20150114:153410.860 db_odbc_discovery() column[2]:'COUNT(h2.host)'
 23876:20150114:153410.860 End of db_odbc_discovery():NOTSUPPORTED
 23876:20150114:153410.860 Item [Zabbix server:db.odbc.discovery[proxies,{$DSN}]] error: Cannot convert
```

现在理解了 SQL 查询如何转换成一个 JSON 对象后, 可以在监控项原型中使用 {#HOST} 宏了：



一旦自动发现开始执行, 会根据每个 proxy 创建一个对应的监控项：



使用 db.odbc.get

请看下面使用 db.odbc.get[,{$DSN}] 和相关 SQL 语句的例子:

```
mysql> SELECT h1.host, COUNT(h2.host) AS count FROM hosts h1 LEFT JOIN hosts h2 ON h1.hostid = h2.proxy_ho
+---------+-------+
| host    | count |
+---------+-------+
| Japan 1 |     5 |
| Japan 2 |    12 |
| Latvia  |     3 |
+---------+-------+
3 rows in set (0.01 sec)
```

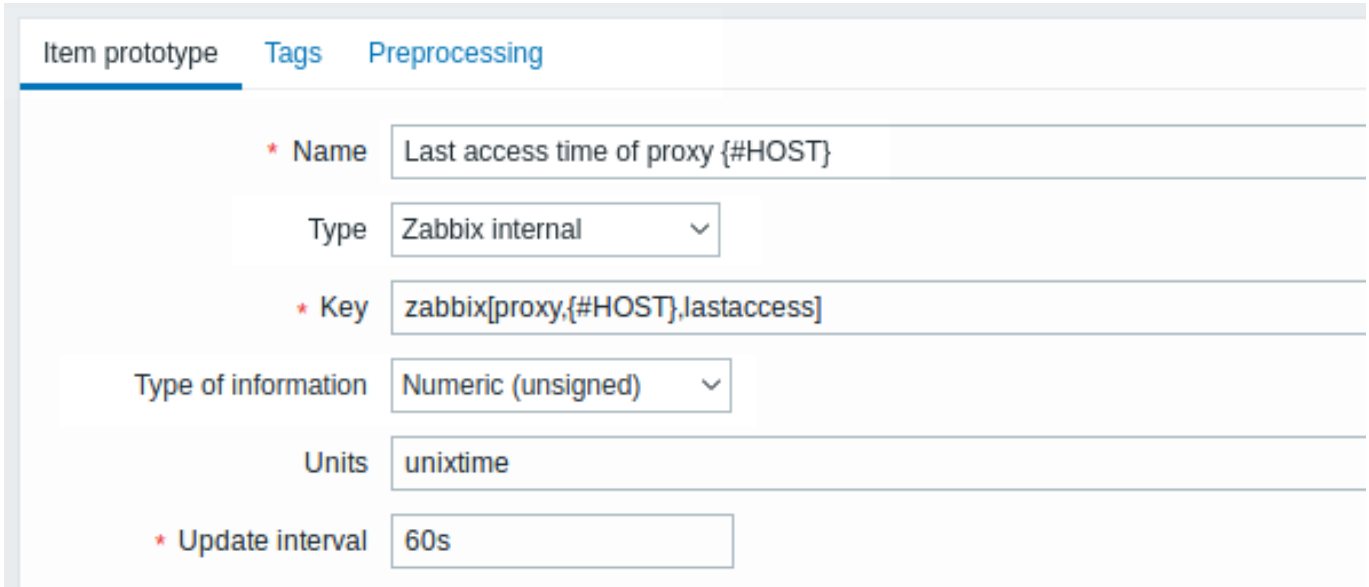
会返回这个 JSON 数组:

```
[
    {
        "host": "Japan 1",
        "count": "5"
    },
    {
        "host": "Japan 2",
        "count": "12"
    },
    {
        "host": "Latvia",
        "count": "3"
    }
]
```

可以看到, 返回的 JSON 数组中不包含低级别自动发现的宏. 然而, 可以在自动发现规则的LLD 宏 选项卡中利用 JSONPath 来自定义宏, 比如:

{#HOST} → $.host

现在这个 {#HOST} 宏可用在监控项原型中了:



12 利用 **Prometheus** 数据执行自动发现

概述

Prometheus 中的数据格式适用于低级别自动发现。

关于 Zabbix 如何执行 Prometheus 数据的查询，请参考Prometheus 检查。

配置

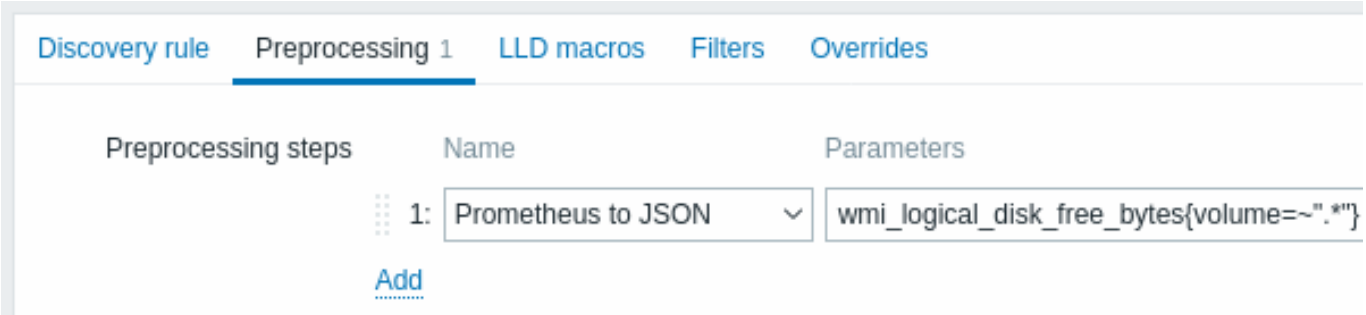
利用低级别自动发现规则创建一个依赖型监控项 ，附属于采集 Prometheus 数据的 HTTP 主监控项。

Prometheus to JSON

在自动发现规则中, 找到预处理选项卡, 然后选择 *Prometheus to JSON* 这个预处理选项. 自动发现需要使用 JSON 格式的数据, *Prometheus to JSON* 这个选项会返回所需的 JSON 格式, 并附带下列属性:

- 度量名称
- 度量值
- 帮助 (如果有)
- 类型 (如果有)
- 标签 (如果有)
- 原始行

例如, 需要查询 `wmi_logical_disk_free_bytes`:



从这些 Prometheus 的行中执行上述查询:

```
# HELP wmi_logical_disk_free_bytes Free space in bytes (LogicalDisk.PercentFreeSpace)
# TYPE wmi_logical_disk_free_bytes gauge
wmi_logical_disk_free_bytes{volume="C:"} 3.5180249088e+11
wmi_logical_disk_free_bytes{volume="D:"} 2.627731456e+09
wmi_logical_disk_free_bytes{volume="HarddiskVolume4"} 4.59276288e+08
```

会返回:

```
[
    {
        "name": "wmi_logical_disk_free_bytes",
        "help": "Free space in bytes (LogicalDisk.PercentFreeSpace)",
        "type": "gauge",
        "labels": {
            "volume": "C:"
         },
        "value": "3.5180249088e+11",
        "line_raw": "wmi_logical_disk_free_bytes{volume=\"C:\"} 3.5180249088e+11"
    },
    {
        "name": "wmi_logical_disk_free_bytes",
        "help": "Free space in bytes (LogicalDisk.PercentFreeSpace)",
        "type": "gauge",
        "labels": {
            "volume": "D:"
         },
        "value": "2.627731456e+09",
        "line_raw": "wmi_logical_disk_free_bytes{volume=\"D:\"} 2.627731456e+09"
    },
    {
```

```
        "name": "wmi_logical_disk_free_bytes",
        "help": "Free space in bytes (LogicalDisk.PercentFreeSpace)",
        "type": "gauge",
        "labels": {
            "volume": "HarddiskVolume4"
         },
        "value": "4.59276288e+08",
        "line_raw": "wmi_logical_disk_free_bytes{volume=\"HarddiskVolume4\"} 4.59276288e+08"
    }
]
```

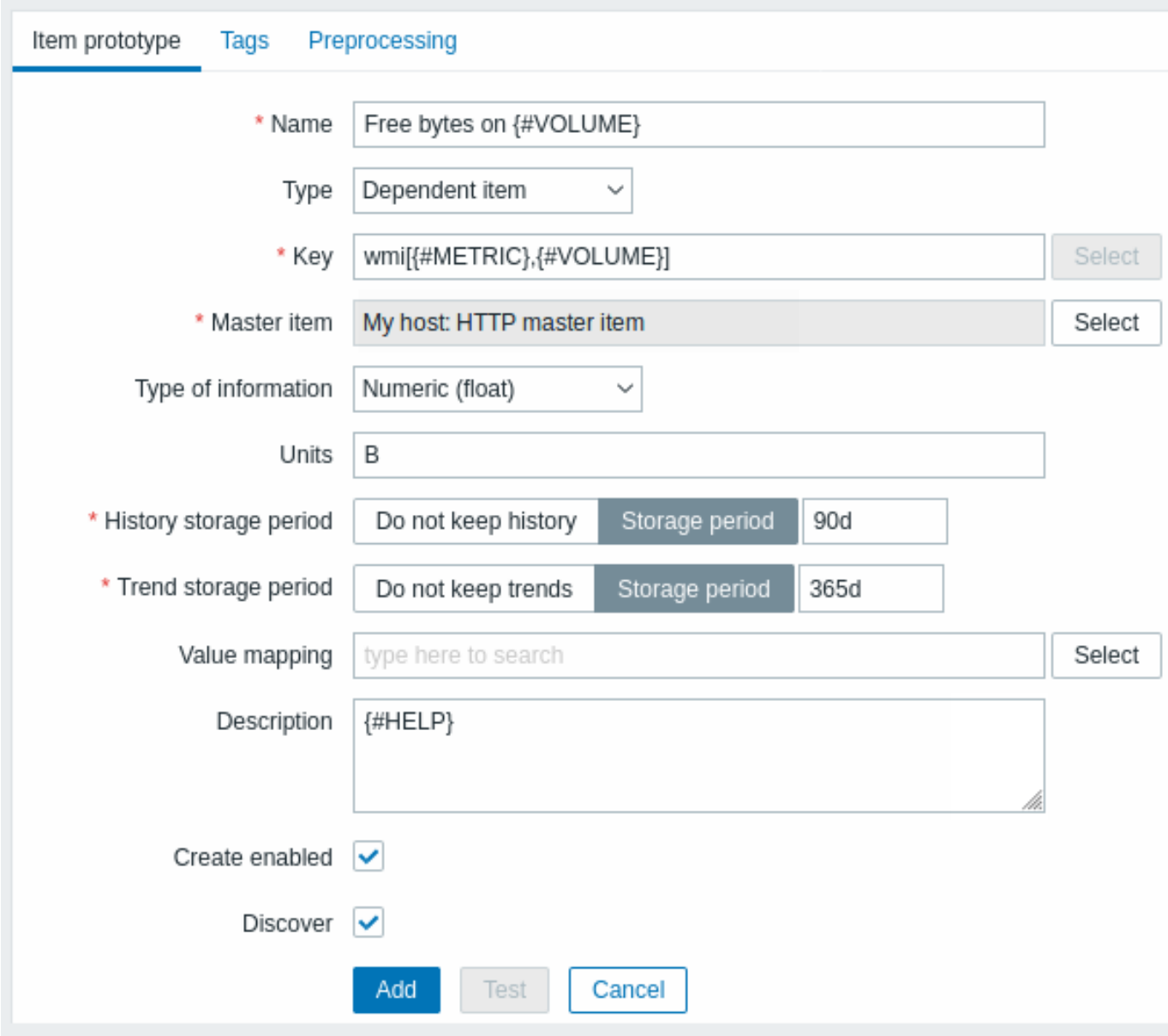
映射 LLD 宏

接下来需要找到 LLD 宏的配置面板并做下面的映射:

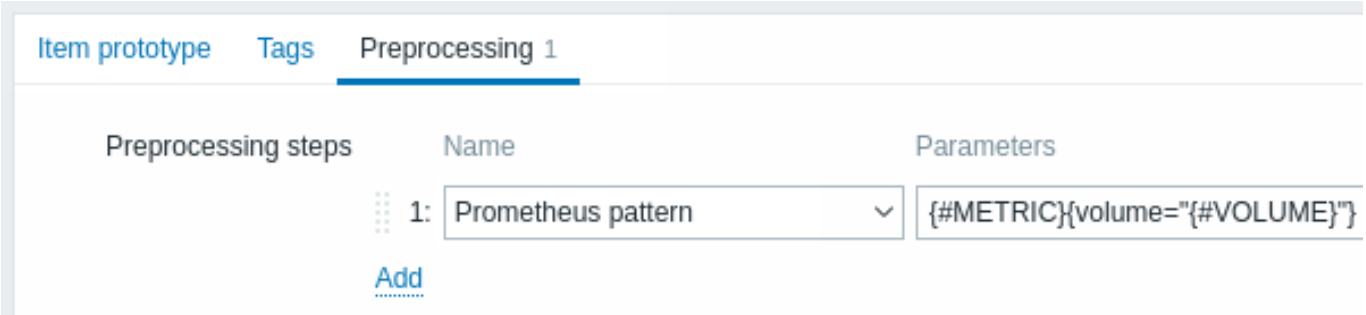
```
{#VOLUME}=$.labels['volume']
{#METRIC}=$['name']
{#HELP}=$['help']
```

监控项原型

可以创建这样的监控项原型:



并配置预处理选项:



13 块设备的自动发现

与文件系统 的自动发现类似, 块设备及设备的类型也可以被自动发现。

监控项的键

自动发现规则 中使用的键是

`vfs.dev.discovery`

从 Zabbix agent 4.4 起支持此监控项, 仅限 Linux 平台。

可以使用此监控项和下列配置来创建自动发现规则：

- 过滤器: **{#DEVNAME} matches** `sd[\D]$` - 发现名为”sd0”, ”sd1”, ”sd2”, ... 的设备
- 过滤器: **{#DEVTYPE} matches** `disk` **AND {#DEVNAME} does not match** `^loop.*` - 发现硬盘类型不以”loop” 开头的设备

支持的宏

此监控项返回两个宏 - {#DEVNAME} 和 {#DEVTYPE} 分别用于识别块设备名称和块设备类型, 例如:

```
[
   {
      "{#DEVNAME}":"loop1",
      "{#DEVTYPE}":"disk"
   },
   {
      "{#DEVNAME}":"dm-0",
      "{#DEVTYPE}":"disk"
   },
   {
      "{#DEVNAME}":"sda",
      "{#DEVTYPE}":"disk"
   },
   {
      "{#DEVNAME}":"sda1",
      "{#DEVTYPE}":"partition"
   }
]
```

块设备的自动发现允许使用 `vfs.dev.read[]` 和 `vfs.dev.write[]` 监控项和 {#DEVNAME} 宏来创建监控项原型, 例如:

- ”vfs.dev.read[{#DEVNAME},sps]”
- ”vfs.dev.write[{#DEVNAME},sps]”

{#DEVTYPE} 用于设备过滤。

14 主机接口的自动发现

概述

在 Zabbix 前端页面中配置的所有主机接口都可以自动发现。

监控项的键

自动发现规则中使用的键是

`zabbix[host,discovery,interfaces]`

内部监控项。从 Zabbix server 3.4 起支持此监控项。

此监控项返回一个 JSON 数组，包含以下关于接口的描述：

- IP 地址/DNS 主机名 (取决于 “连接到” 主机设置)
- 端口号
- 接口类型 (Zabbix agent, SNMP, JMX, IPMI)
- 是否是默认接口
- 批量请求 (bulk request) 特性是否启用 - 只适用于 SNMP 接口。

例如：

```
[{"{#IF.CONN}":"192.168.3.1","{#IF.IP}":"192.168.3.1","{#IF.DNS}":"","{#IF.PORT}":"10050","{#IF.TYPE}":"AG
```

多个接口的 JSON 数据按以下规则排序：

- 接口类型,
- 默认 - 默认接口在非默认接口前面,
- 接口 ID (升序排列)。

支持的宏

下列宏可以在监控项规则中的过滤器 和监控项、触发器、图形的原型中使用：

宏	描述
{#IF.CONN}	接口 IP 地址或 DNS 主机名。
{#IF.IP}	接口 IP 地址。
{#IF.DNS}	接口 DNS 主机名。
{#IF.PORT}	接口的端口号。
{#IF.TYPE}	接口类型 (”AGENT”, ”SNMP”, ”JMX”, or ”IPMI”)。
{#IF.DEFAULT}	接口默认状态： 0 - 非默认接口 1 - 默认接口
{#IF.SNMP.BULK}	接口的 SNMP 批量 (bulk) 处理状态： 0 - 禁用 1 - 启用 仅当接口类型为”SNMP” 时才返回该宏。

7 Custom LLD rules

Overview

It is also possible to create a completely custom LLD rule, discovering any type of entities - for example, databases on a database server.

To do so, a custom item should be created that returns JSON, specifying found objects and optionally - some properties of them. The amount of macros per entity is not limited - while the built-in discovery rules return either one or two macros (for example, two for filesystem discovery), it is possible to return more.

Example

The required JSON format is best illustrated with an example. Suppose we are running an old Zabbix 1.8 agent (one that does not support ”vfs.fs.discovery”), but we still need to discover file systems. Here is a simple Perl script for Linux that discovers mounted file systems and outputs JSON, which includes both file system name and type. One way to use it would be as a UserParameter with key ”vfs.fs.discovery_perl”:

```
####!/usr/bin/perl

$first = 1;

print "[\n";

for (`cat /proc/mounts`)
{
    ($fsname, $fstype) = m/\S+ (\S+) (\S+)/;

    print "\t,\n" if not $first;
    $first = 0;
```

```
    print "\t{\n";
    print "\t\t\"{#FSNAME}\":\"$fsname\",\n";
    print "\t\t\"{#FSTYPE}\":\"$fstype\"\n";
    print "\t}\n";
}

print "]\n";
```

Attention:
Allowed symbols for LLD macro names are **0-9** , **A-Z** , **_** , **.** Lowercase letters are not supported in the names.

An example of its output (reformatted for clarity) is shown below. JSON for custom discovery checks has to follow the same format.

```
[
    { "{#FSNAME}":"/",                          "{#FSTYPE}":"rootfs"   },
    { "{#FSNAME}":"/sys",                       "{#FSTYPE}":"sysfs"    },
    { "{#FSNAME}":"/proc",                      "{#FSTYPE}":"proc"     },
    { "{#FSNAME}":"/dev",                       "{#FSTYPE}":"devtmpfs" },
    { "{#FSNAME}":"/dev/pts",                   "{#FSTYPE}":"devpts"   },
    { "{#FSNAME}":"/lib/init/rw",               "{#FSTYPE}":"tmpfs"    },
    { "{#FSNAME}":"/dev/shm",                   "{#FSTYPE}":"tmpfs"    },
    { "{#FSNAME}":"/home",                      "{#FSTYPE}":"ext3"     },
    { "{#FSNAME}":"/tmp",                       "{#FSTYPE}":"ext3"     },
    { "{#FSNAME}":"/usr",                       "{#FSTYPE}":"ext3"     },
    { "{#FSNAME}":"/var",                       "{#FSTYPE}":"ext3"     },
    { "{#FSNAME}":"/sys/fs/fuse/connections",   "{#FSTYPE}":"fusectl"  }
]
```

In the previous example it is required that the keys match the LLD macro names used in prototypes, the alternative is to extract LLD macro values using JSONPath {#FSNAME} → $.fsname and {#FSTYPE} → $.fstype, thus making such script possible:

```
####!/usr/bin/perl

$first = 1;

print "[\n";

for (`cat /proc/mounts`)
{
    ($fsname, $fstype) = m/\S+ (\S+) (\S+)/;

    print "\t,\n" if not $first;
    $first = 0;

    print "\t{\n";
    print "\t\t\"fsname\":\"$fsname\",\n";
    print "\t\t\"fstype\":\"$fstype\"\n";
    print "\t}\n";
}

print "]\n";
```

An example of its output (reformatted for clarity) is shown below. JSON for custom discovery checks has to follow the same format.

```
[
    { "fsname":"/",                         "fstype":"rootfs"   },
    { "fsname":"/sys",                      "fstype":"sysfs"    },
    { "fsname":"/proc",                     "fstype":"proc"     },
    { "fsname":"/dev",                      "fstype":"devtmpfs" },
    { "fsname":"/dev/pts",                  "fstype":"devpts"   },
    { "fsname":"/lib/init/rw",              "fstype":"tmpfs"    },
    { "fsname":"/dev/shm",                  "fstype":"tmpfs"    },
    { "fsname":"/home",                     "fstype":"ext3"     },
    { "fsname":"/tmp",                      "fstype":"ext3"     },
```

```
    { "fsname":"/usr",                      "fstype":"ext3"     },
    { "fsname":"/var",                      "fstype":"ext3"     },
    { "fsname":"/sys/fs/fuse/connections",  "fstype":"fusectl"  }
]
```

Then, in the discovery rule's "Filter" field, we could specify "{#FSTYPE}" as a macro and "rootfs|ext3" as a regular expression.

Note:
You don't have to use macro names FSNAME/FSTYPE with custom LLD rules, you are free to use whatever names you like. In case JSONPath is used then LLD row will be an array element that can be an object, but it can be also another array or a value.

Note that, if using a user parameter, the return value is limited to 512 KB. For more details, see data limits for LLD return values.

16. 分布式监控

概览　Zabbix 提供了一种使用 Zabbix proxies. 监视分布式 IT 基础设施的有效和可靠的方法

Proxy 代理可以用来代表中央 Zabbix server 在本地收集数据，然后将数据报告给 Zabbix server。

Proxy 特性

在选择使用/不使用 proxy 时，必须考虑几个因素。

	Proxy
Lightweight	**Yes**
GUI	No
Works independently	**Yes**
Easy maintenance	**Yes**
Automatic DB creation[1]	**Yes**
Local administration	No
Ready for embedded hardware	**Yes**
One way TCP connections	**Yes**
Centralized configuration	**Yes**
Generates notifications	No

Note:
[1] 自动 DB 创建特性只适用于 SQLite。其他数据库需要手动设置。

1 proxy 代理

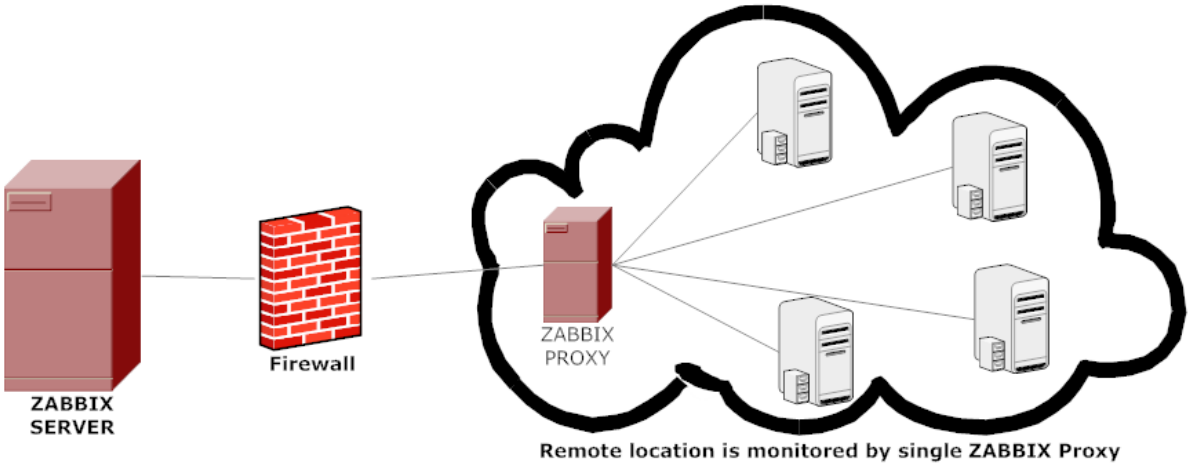
概览

Zabbix Proxy 可以代表 Zabbix server 收集性能和可用性数据。通过这种方式，proxy 可以自己承担一些收集数据的负载，并减轻 Zabbix Server 的负担。

此外，当所有 agents 和 proxy 都向一个 Zabbix server 报告并且所有数据都集中收集时，使用 Proxy 代理是实现集中式和分布式监控的最简单方法。

Zabbix proxy 可以被使用作为:

- 监控远程位置
- 监控通信不可靠的位置
- 在监视数千个设备时卸载 Zabbix 服务器
- 简化分布式监控的维护



proxy 只需要一个到 Zabbix server 的 TCP 连接。这样就可以更容易地绕过防火墙，因为您只需要配置一条防火墙规则。

Attention:
Zabbix proxy 代理必须使用单独的数据库。将其指向 Zabbix server 数据库将破坏配置。

proxy 收集的所有数据在传输到 server 之前都存储在本地。这种方式不会因为与 server 之间的任何临时通信问题而丢失数据。*ProxyLocalBuffer* 和 *ProxyOfflineBuffer* 参数在proxy 配置文件控制数据在本地保存多长时间。

Attention:
可能会出现这样的情况: 直接从 Zabbix server 数据库接收最新配置更改的 proxy 代理拥有比 Zabbix server 更最新的配置，而 Zabbix server 的配置可能因为CacheUpdateFrequency的值而不能快速更新。因此，proxy 代理可能会开始收集数据并将它们发送到忽略这些数据的 Zabbix server。

Zabbix proxy 代理是一个数据收集器。它不计算触发器、处理事件或发送警报。有关什么是 proxy 代理功能的概述，请查看下表：

功能		proxy 支持列表
Items		
	Zabbix agent checks	**Yes**
	Zabbix agent checks (active)	**Yes** [1]
	Simple checks	**Yes**
	Trapper items	**Yes**
	SNMP checks	**Yes**
	SNMP traps	**Yes**
	IPMI checks	**Yes**
	JMX checks	**Yes**
	Log file monitoring	**Yes**
	Internal checks	**Yes**
	SSH checks	**Yes**
	Telnet checks	**Yes**
	External checks	**Yes**
	Dependent items	**Yes**
	Script items	**Yes**
Built-in web monitoring		**Yes**
Item value preprocessing		**Yes**
Network discovery		**Yes**
Active agent autoregistration		**Yes**
Low-level discovery		**Yes**
Remote commands		**Yes**
Calculating triggers		*No*
Processing events		*No*
Event correlation		*No*
Sending alerts		*No*

[1] 为确保 agent 请求 proxy(而不是 server) 进行活动检查，proxy 代理必须被列于 **ServerActive** 配置文件中的参数。

过载保护

如果 Zabbix server 宕机一段时间，proxy 已经收集了大量数据，然后 server 启动，它可能会超载 (历史缓存使用率在一段时间内保持在 95-100%)。这种过载可能会导致性能下降，检查的处理速度比正常情况下要慢。对这种场景的保护是为了避免由于重载历史缓存而产生的问题。

当 Zabbix server 历史缓存满时，历史缓存写访问被限制，停止 server 数据收集进程。最常见的历史缓存过载情况是 server 停机后，proxy 上传收集的数据。为了避免这种情况，添加了 proxy 节流 (目前无法禁用)。

当历史缓存使用率达到 80% 时，Zabbix server 将停止接受来自 proxy 的数据。相反，这些 proxy 将被放在一个节流列表中。这将持续到缓存使用率下降到 60%。现在，server 将开始逐一接受来自节流列表定义的 proxy 的数据。这意味着在节流期间试图上载数据的第一个 proxy 将首先被接收数据，在此之前，server 将不会接受来自其他 proxy 的数据。

这种调节模式将继续，直到缓存使用率再次达到 80%，或者下降到 20%，或者调节列表为空。在第一种情况下，server 将再次停止接受 proxy 数据。在另外两种情况下，server 将开始正常工作，接受来自所有 proxy 的数据。

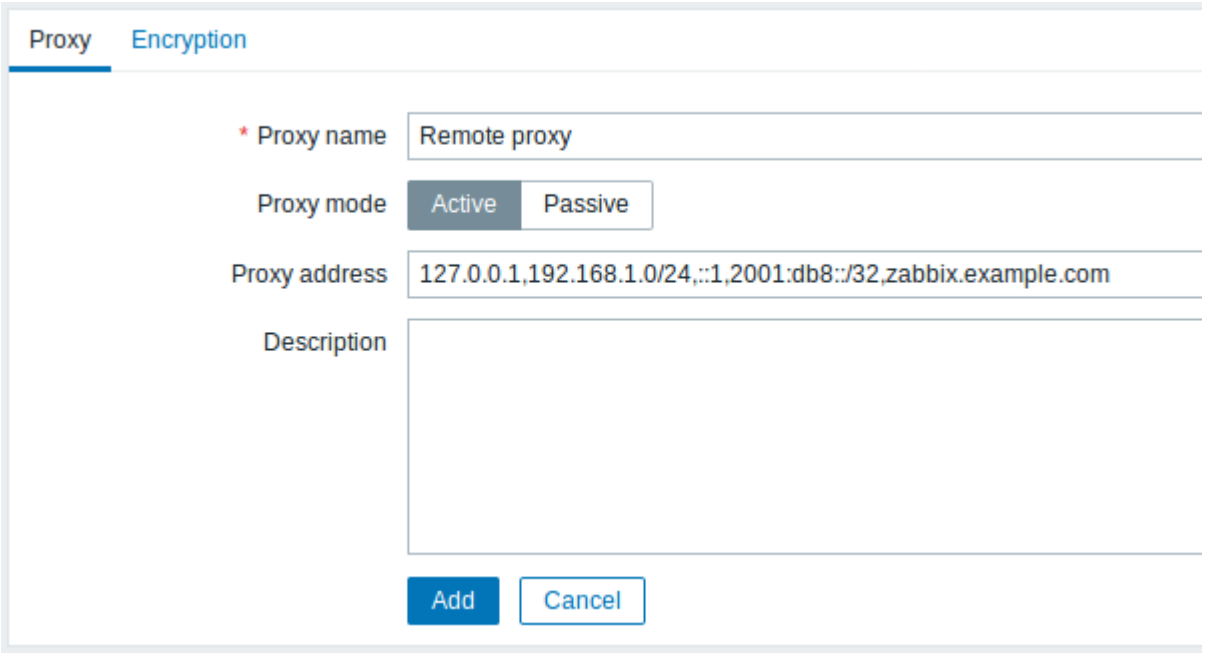
你可以使用 `zabbix[wcache,history,pused]` 内部项将 Zabbix server 的行为与一个度量关联起来。

配置　如果你安装 和配置 一个 proxy，下一步就是 Zabbix 前端配置。

添加一个 proxy 代理

在 Zabbix 前端中配置 proxy：

- 前往: *Administration* → *Proxies*
- 点击 创建 *proxy*



参数	描述
Proxy name	输入 proxy 名称。必须与 proxy 配置文件中的 “*Hostname*” 保持一致。
Proxy mode	选择 proxy 模式。 主动 - proxy 将连接到 Zabbix server 并请求配置数据 被动 - - Zabbix server 连接到 proxy *Note* 没有加密的通信 (敏感的)proxy 配置数据可能会成为可以访问 Zabbix 服务器的端口时，使用一个主动的 proxy。这是可能的，因为任何人都可以假装是一个活动的 proxy 并请求配置数据，如果身份验证没有发生或 proxy 地址不受限制在 *Proxy* 地址字段。
Proxy address	如果指定了，那么主动的 proxy 请求只接受这个以逗号分隔的 IP 地址列表 (可选的 CIDR 标记)，或者主动 Zabbix proxy 的 DNS 名称。 该字段仅在 “*Proxy* 模式” 字段中选择了 active proxy 时有效。宏是不支持的。 从 Zabbix 4.0.0 开始就支持这个选项。
Interface	输入被动 proxy 的接口详细信息。 该字段仅在 “*Proxy* 模式” 字段中选择被动 proxy 时有效。

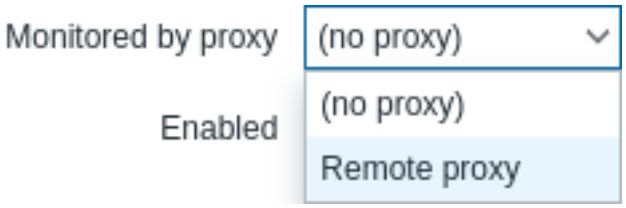
参数		描述
	IP address	被动 proxy 的 IP 地址 (可选)。
	DNS name	被动 proxy 的 DNS 名称 (可选)。
	Connect to	点击相应的按钮将告诉 Zabbix server 使用什么来从 proxy 检索数据: **IP** -连接 proxyIP 地址 (推荐) **DNS** -连接到 proxy DNS 名称
	Port	被动 proxy 的 TCP/UDP 端口号 (默认为 10051)。
Description		输入 proxy 描述信息。

加密选项卡允许您要求与 proxy 的加密连接。

参数	描述
Connections to proxy	服务器如何连接到被动 proxy: 不加密 (默认)，使用 PSK(预共享密钥)或证书。
Connections from proxy	选择从活动 proxy 中允许的连接类型。可以同时选择几种连接类型 (这对于测试和切换到其他连接类型很有用)。默认为 “不加密”。
Issuer	允许的证书颁发者。证书首先由 CA(证书颁发机构) 验证。如果它是有效的，由 CA 签名，那么 *Issuer* 字段可以用于进一步限制所允许的 CA。此字段是可选的，用于 Zabbix 安装使用来自多个 CA 的证书时。
Subject	证书允许的主题。证书首先由 CA 验证。如果它是有效的，由 CA 签名，那么 *Subject* 字段可以用来只允许一个值的 *Subject* 字符串。如果此字段为空，则接受由配置的 CA 签名的任何有效证书。
PSK identity	- 预共享密钥标识字符串。 不要把敏感信息放在 PSK 身份中，它在网络上未经加密传输，以通知接收方使用哪个 PSK。
PSK	Pre-shared 关键 (hex-string)。最大长度: 512 十六进制数字 (256 字节 PSK) 如果 Zabbix 使用 GnuTLS 或 OpenSSL 库，64 十六进制数字 (32 字节 PSK) 如果 Zabbix 使用 mbed TLS (PolarSSL) 库。例如： 1f87b595725ac58dd977beef14b97461a7c1045b9a1c963065002c5473194952

主机配置

你可以在主机配置表单中指定一个单独的主机应该被一个 proxy 监控，使用 *Monitored by proxy* 字段。



主机批量更新是指定主机应该由 proxy 监视的另一种方式。

17. 加密

概述　Zabbix 支持使用 TLS 协议 v.1.2 和 1.3(取决于加密库) 在 Zabbix 组件之间进行加密通信。支持证书加密和预共享密钥加密。

可以为连接配置加密:

- Zabbix server，Zabbix proxy，Zabbix agent，Zabbix _sender 和 Zabbix _get 工具
- 到 Zabbix 数据库从 zabbix 前端和 server/proxy

加密是可选和可配置的单个组件:

- 一些 proxy 代理和 agent 代理可以配置为使用服务器的基于证书的加密，而其他的可以使用基于预共享密钥的加密，而其他的则继续使用未加密的通信 (与前面一样)

- Server (proxy) 可以为不同的主机使用不同的加密配置。

Zabbix 守护程序为传入的加密和非加密连接使用一个侦听端口。添加加密并不需要在防火墙上打开新的端口。

限制因素

- 私钥以纯文本形式存储在 Zabbix 组件启动时可读的文件中
- 预共享密钥在 Zabbix 前端输入，并以明文形式存储在 Zabbix 数据库中
- 内置加密不能保护通信:
  - 运行 Zabbix 前端的 web 服务器和用户 web 浏览器之间
  - Zabbix 前端和 Zabbix server 之间
- 目前，每个加密连接都使用完整的 TLS 握手打开，没有实现会话缓存和票据
- 根据网络延迟，添加加密会增加监控项检查和操作的时间:
  - 例如，如果包延迟 100ms，那么打开 TCP 连接并发送未加密的请求大约需要 200ms。在建立 TLS 连接时，增加约 1000 毫秒的加密;
  - 超时可能需要增加，否则在代理上运行远程脚本的某些监控项和操作可能会在未加密的连接中工作，但在加密的连接中超时则会失败。
- 不支持加密网络设备自动发现。由网络发现执行的 Zabbix agent 检查将不加密，如果 Zabbix agent 被配置为拒绝未加密的连接，这样的检查将不会成功。

编译支持加密的 **Zabbix**　为了支持加密，Zabbix 必须编译并链接到受支持的加密库之一:

- GnuTLS - 从版本 3.1.18
- OpenSSL - 版本 1.0.1, 1.0.2, 1.1.0, 1.1.1
- LibreSSL - 测试版本 2.7.4, 2.8.2:
  - LibreSSL 2.6.x 是不支持的
  - LibreSSL 支持作为 OpenSSL 的兼容替代品；新的’ tls_*() ’ libressl 特定的 API 函数没有被使用。使用 LibreSSL 编译的 Zabbix 组件将不能使用 PSK，只能使用证书。通过指定相应的选项来 “configure” 脚本来选择库:
- `--with-gnutls[=DIR]`
- `--with-openssl[=DIR]` (也用于 LibreSSL)

例如，要用 *OpenSSL* 配置 server 和 agent 的源文件，你可以使用如下方法:

```
./configure --enable-server --enable-agent --with-mysql --enable-ipv6 --with-net-snmp --with-libcurl --wit
```

不同的 Zabbix 组件可以使用不同的加密库进行编译 (例如，服务器上有 *OpenSSL*，代理上有 *GnuTLS*)。

Attention:
如果您计划使用预共享密钥 (PSK)，请考虑在使用 PSK 的 Zabbix 组件中使用 *GnuTLS* 或 *OpenSSL 1.1.0*(或更新) 库。*GnuTLS* 和 *OpenSSL 1.1.0* 库支持 PSK 密码套件 向前兼容密码.*OpenSSL* 库的旧版本 (1.0.1,1.0.2c) 也支持 PSK，但可用的 PSK 密码套件不支持向前兼容密码。

连接加密管理　Zabbix 中的连接可以使用：

- 不加密 (默认)
- 基于 RSA 证书的加密
- 基于 PSK 基础加密

有两个重要的参数用于指定之间的加密 Zabbix 组件:

- TLSConnect -指定对传出连接使用何种加密 (未加密、PSK 或证书)
- TLSAccept - 指定允许传入连接的类型 (未加密的，PSK 或证书)。可以指定一个或多个值。

`TLSConnect` 在 Zabbix proxy(在主动模式下，仅指定到 server 的连接) 和 Zabbix agent(在主动检查中) 的配置文件中使用。在 Zabbix 前端 TLSConnect 等效的是配置 → 主机 →< 一些主机 >→ 加密标签中的连接到主机字段和管理 →*Proxy* 代理 →< 一些 *proxy* 代理 >→ 加密标签中的连接到 *proxy* 字段。如果为连接配置的加密类型失败，则不会尝试其他加密类型。

`TLSAccept` 在 Zabbix proxy(被动模式，仅指定来自 server 的连接) 和 Zabbix agent(被动检查) 的配置文件中使用。在 Zabbix 前端 TLSAccept 等价的是配置 → 主机 →< 一些主机 >→ 加密标签中的主机连接字段和管理 →*proxy* 代理 →< 一些 *proxy* 代理 >→ 加密标签中的 *proxy* 连接字段。

通常您只为传入加密配置一种类型的加密。但您可能希望切换加密类型，例如，从未加密切换到基于证书的加密，并尽可能减少停机时间和回滚可能性。为实现这一目标:

- 设置 `TLSAccept=unencrypted,cert` 在 agent 配置文件中设置，重启 Zabbix agent
- 使用证书测试 zabbix_get 到 agent 的连接。如果它能工作，你可以在 Zabbix 前端的 *Configuration→Hosts→<some host>→encryption* 选项卡中通过将 *Connections to host* 设置为 “Certificate” 来重新配置该 agent 的加密。
- 当 server 配置缓存被更新时 (如果主机被 proxy 监视，则 proxy 配置被更新)，那么到该 proxy 的连接将被加密

- 如果一切正常，您可以在 agent 配置文件中设置’ TLSAccept=cert ’，并重启 Zabbix agent。现在 agent 将只接受加密的基于证书的连接。未加密的和基于 psk 的连接将被拒绝。

以类似的方式，它在 server 和 proxy 上工作。如果在 Zabbix 前端主机配置来自主机的连接设置为 “证书”，那么只有基于证书的加密连接将被接受从 agent(主动检查) 和 Zabbix_sender(trapper 监控项)。

最可能的情况是将传入和传出连接配置为使用相同的加密类型或完全不加密。但在技术上，可以不对称地配置它，例如，对传入连接进行基于证书的加密，对传出连接进行基于 psk 的加密。

Zabbix 前端 “*Agent* 加密” 列的 “配置 → 主机” 中显示各主机的加密配置信息。例如:

例子	连接到主机	允许连接的主机	拒绝连接的主机
NONE	非加密的	非加密的	加密，证书和 psk 加密
CERT NONE PSK CERT	加密的, 基于证书	加密的, 基于证书	加密的基于 PSK
PSK NONE PSK CERT	加密的, 基于 PSK	加密的, 基于 PSK	非加密的基于证书加密的
PSK NONE PSK CERT	加密的, 基于 PSK	非加密和基于 PSK 加密	基于证书加密
CERT NONE PSK CERT	加密, 证书加密	未加密, PSK 或者 or 基于证书加密	-

:: noteimportant 默认情况下，连接是不加密的。如果使用加密那么必须分别为每个主机和 proxy 配置加密。

zabbix_get and zabbix_sender 与加密　请参阅zabbix_get和zabbix_sender 操作说明使用加密。

密码套件　默认情况下，密码套件是在 Zabbix 启动期间内部配置的，在 Zabbix 4.0.19 和 4.4.7 之前是不能由用户配置的。

从 Zabbix 4.0.19，4.4.7 开始也支持 GnuTLS 和 OpenSSL 用户配置的密码套件。用户可以根据他们的安全策略配置密码套件。使用这个特性是可选的 (内置的默认密码套件仍然可以使用)。

对于使用默认设置编译的加密库，Zabbix 内置规则通常会导致以下加密套件 (按优先级由高到低的顺序):

库	证书密码套件	PSK 密码套件
GnuTLS 3.1.18	TLS_ECDHE_RSA_AES_128_GCM_SHA256 TLS_ECDHE_RSA_AES_128_CBC_SHA256 TLS_ECDHE_RSA_AES_128_CBC_SHA1 TLS_RSA_AES_128_GCM_SHA256 TLS_RSA_AES_128_CBC_SHA256 TLS_RSA_AES_128_CBC_SHA1	TLS_ECDHE_PSK_AES_128_CBC_SHA256 TLS_ECDHE_PSK_AES_128_CBC_SHA1 TLS_PSK_AES_128_GCM_SHA256 TLS_PSK_AES_128_CBC_SHA256 TLS_PSK_AES_128_CBC_SHA1
OpenSSL 1.0.2c	ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA256 ECDHE-RSA-AES128-SHA AES128-GCM-SHA256 AES128-SHA256 AES128-SHA	PSK-AES128-CBC-SHA
OpenSSL 1.1.0	ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA256 ECDHE-RSA-AES128-SHA AES128-GCM-SHA256 AES128-CCM8 AES128-CCM AES128-SHA256 AES128-SHA	ECDHE-PSK-AES128-CBC-SHA256 ECDHE-PSK-AES128-CBC-SHA PSK-AES128-GCM-SHA256 PSK-AES128-CCM8 PSK-AES128-CCM PSK-AES128-CBC-SHA256 PSK-AES128-CBC-SHA

库	证书密码套件	PSK 密码套件
OpenSSL 1.1.1d	TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256 TLS_AES_128_GCM_SHA256 ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA256 ECDHE-RSA-AES128-SHA AES128-GCM-SHA256 AES128-CCM8 AES128-CCM AES128-SHA256 AES128-SHA	TLS_CHACHA20_POLY1305_SHA256 TLS_AES_128_GCM_SHA256 ECDHE-PSK-AES128-CBC-SHA256 ECDHE-PSK-AES128-CBC-SHA PSK-AES128-GCM-SHA256 PSK-AES128-CCM8 PSK-AES128-CCM PSK-AES128-CBC-SHA256 PSK-AES128-CBC-SHA

用户配置密码套件　内置的密码套件选择标准可以被用户配置的密码套件覆盖。

Attention:
用户配置的密码套件是针对了解 TLS 密码套件、其安全性和错误后果以及熟悉 TLS 故障排除的高级用户的特性。

内置的密码套件选择标准可以使用以下参数覆盖:

覆盖范围	参数	值	描述
证书的密码套件选择	TLSCipherCert13	有效 OpenSSL 1.1.1 cipher strings 对于 TLS 1.3 协议 (它们的值被传递到 OpenSSL 函数 SSL_CTX_set_ciphersuites())。	TLS 1.3 的基于证书的加密套件选择标准 仅支持 OpenSSL 1.1.1 及以上版本。
	TLSCipherCert	针对 TLS 1.2 的有效 OpenSSL密码字符串 或有效的 GnuTLS优先级字符串。它们的值分别传递给 SSL_CTX_set_cipher_list() 或 gnutls_priority_init() 函数。	针对 TLS 1.2/1.3 (GnuTLS)、TLS 1.2 (OpenSSL) 的基于证书的密码套件选择标准。
Ciphersuite selection for PSK	TLSCipherPSK13	有效的 OpenSSL 1.1.1密码字符串 对于 TLS 1.3 协议 (它们的值被传递到 OpenSSL 函数 SSL_CTX_set_ciphersuites())。	基于 psk 的 TLS 1.3 加密套件选择标准 仅支持 OpenSSL 1.1.1 及以上版本。

覆盖范围	参数	值	描述
	TLSCipherPSK	对于 TLS 1.2 有效的 OpenSSL密码字符串或有效的 GnuTLS优先级字符串。它们的值分别传递给 SSL_CTX_set_cipher_list() 或 gnutls_priority_init() 函数。	基于 psk 的 TLS 1.2/1.3 (GnuTLS)、TLS 1.2 (OpenSSL) 加密套件选择标准。
Combined ciphersuite list for certificate and PSK	TLSCipherAll13	TLS 1.3 协议的有效的 OpenSSL 1.1.1密码字符串(它们的值被传递到 OpenSSL 函数 SSL_CTX_set_ciphersuites())。	TLS 1.3 仅支持 OpenSSL 1.1.1 及以上版本。
	TLSCipherAll	对于 TLS 1.2 有效的 OpenSSL密码字符串或有效的 GnuTLS优先级字符串。它们的值分别传递给 SSL_CTX_set_cipher_list() 或 gnutls_priority_init() 函数。	针对 TLS 1.2/1.3 (GnuTLS)、TLS 1.2 (OpenSSL) 的加密套件选择标准.

在zabbix_get和zabbix_sender工具中覆盖密码套件选择 - 使用命令行参数:

- `--tls-cipher13`
- `--tls-cipher`

新参数为可选参数。如果没有指定参数，则使用内部的默认值。如果参数已定义，则它不能为空。

如果加密库中的 tlsciphher * 值设置失败，则 server、proxy 或 agent 将无法启动，并记录一个错误。

理解每个参数何时适用是很重要的。

外部连接

简单例子外部连接:

- 对于使用证书的外发连接-使用 TLSCipherCert13 或 TLSCipherCert
- 对于 PSK 外发连接，使用 TLSCipherPSK13 和 TLSCipherPSK
- 在 zabbix_get 和 zabbix_sender 实用程序中，可以使用命令行参数’——tls-cipher13 ’ 和’——tls-cipher ’(加密用’——tls-connect ’ 参数明确指定)

传入连接

对于传入连接来说，这有点复杂，因为规则是特定于组件和配置的。

对 Zabbix **agent**:

Agent 连接步骤	密码配置
TLSConnect=cert	TLSCipherCert, TLSCipherCert13
TLSConnect=psk	TLSCipherPSK, TLSCipherPSK13
TLSAccept=cert	TLSCipherCert, TLSCipherCert13
TLSAccept=psk	TLSCipherPSK, TLSCipherPSK13

Agent 连接步骤	密码配置
TLSAccept=cert,psk	TLSCipherAll, TLSCipherAll13

对 Zabbix **server** 和 **proxy** :

连接步骤	密码配置
Outgoing connections using PSK	TLSCipherPSK, TLSCipherPSK13
Incoming connections using certificates	TLSCipherAll, TLSCipherAll13
Incoming connections using PSK if server has no certificate	TLSCipherPSK, TLSCipherPSK13
Incoming connections using PSK if server has certificate	TLSCipherAll, TLSCipherAll13

在上面的两个表中可以看到一些模式:

- TLSCipherAll 和 TLSCipherAll13 只能在使用基于证书和 PSK 加密套件的组合列表时指定。发生这种情况有两种情况:server(proxy) 配置证书 (PSK 密码套件总是在 server、proxy 上配置如果加密库支持 PSK)，agent 被配置为接受基于证书和基于 PSK 的传入连接
- 在其他情况下，TLSCipherCert 和/或 TLSCipherPSK* 就足够了

下表显示了’ TLSCipher* ’ 的内置默认值。对于您自己的定制值来说，它们可能是一个很好的起点。

参数	GnuTLS 3.6.12
TLSCipherCert	NONE:+VERS-TLS1.2:+ECDHE-RSA:+RSA:+AES-128-GCM:+AES-128-CBC:+AEAD:+SHA256:+SHA1:+CURVE-ALL:+COMP-NULL:+SIGN-ALL:+CTYPE-X.509
TLSCipherPSK	NONE:+VERS-TLS1.2:+ECDHE-PSK:+PSK:+AES-128-GCM:+AES-128-CBC:+AEAD:+SHA256:+SHA1:+CURVE-ALL:+COMP-NULL:+SIGN-ALL
TLSCipherAll	NONE:+VERS-TLS1.2:+ECDHE-RSA:+RSA:+ECDHE-PSK:+PSK:+AES-128-GCM:+AES-128-CBC:+AEAD:+SHA256:+SHA1:+CURVE-ALL:+COMP-NULL:+SIGN-ALL:+CTYPE-X.509

参数	OpenSSL 1.1.1d [1]
TLSCipherCert13	
TLSCipherCert	EECDH+aRSA+AES128:RSA+aRSA+AES128
TLSCipherPSK13	TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256
TLSCipherPSK	kECDHEPSK+AES128:kPSK+AES128
TLSCipherAll13	
TLSCipherAll	EECDH+aRSA+AES128:RSA+aRSA+AES128:kECDHEPSK+AES128:kPSK+AES128

[1] 对于较早的 OpenSSL 版本 (1.0.1,1.0.2,1.1.0)，对于 LibreSSL 和如果在没有 PSK 支持的情况下编译 OpenSSL，默认值是不同的。

用户配置的密码套件示例

请参阅以下用户配置的密码套件示例:

- 测试密码字符串并只允许 PFS 密码套件
- 从 AES128 切换到 AES256

测试密码字符串并只允许 PFS 密码套件

要查看哪些加密套件已经被选中，你需要在配置文件中设置’DebugLevel=4’，或者对 zabbix_sender 使用’ -vv ’ 选项。

在获得所需的密码组之前，可能需要对 “TLSCipher” 参数进行一些实验。为了调整 *“TLSCipher”* 参数，多次重启 Zabbix server、proxy 或 agent 是不方便的。更方便的选项是使用 zabbix_sender 或’ openssl ’ 命令。让我们几种都看看如何实现。

1. 使用 zabbix_sender.

让我们做一个测试配置文件，例如/home/zabbix/test.conf，使用 zabbix_agentd.conf 文件的规则:

```
Hostname=nonexisting
ServerActive=nonexisting
```

```
TLSConnect=cert
TLSCAFile=/home/zabbix/ca.crt
TLSCertFile=/home/zabbix/agent.crt
TLSKeyFile=/home/zabbix/agent.key
TLSPSKIdentity=nonexisting
TLSPSKFile=/home/zabbix/agent.psk
```

在本例中，您需要有效的 CA 和代理证书以及 PSK。为您的环境调整证书和 PSK 文件路径和名称。

如果你不使用证书，而只使用 PSK，你可以制作一个更简单的测试文件:

```
Hostname=nonexisting
ServerActive=nonexisting

TLSConnect=psk
TLSPSKIdentity=nonexisting
TLSPSKFile=/home/zabbix/agentd.psk
```

运行 zabbix_sender(用 OpenSSL 1.1.d 编译的例子) 可以看到所选的密码套件:

```
$ zabbix_sender -vv -c /home/zabbix/test.conf -k nonexisting_item -o 1 2>&1 | grep ciphersuites
zabbix_sender [41271]: DEBUG: zbx_tls_init_child() certificate ciphersuites: TLS_AES_256_GCM_SHA384 TLS_
zabbix_sender [41271]: DEBUG: zbx_tls_init_child() PSK ciphersuites: TLS_CHACHA20_POLY1305_SHA256 TLS_AE
zabbix_sender [41271]: DEBUG: zbx_tls_init_child() certificate and PSK ciphersuites: TLS_AES_256_GCM_SHA
```

这里可以看到默认选择的密码套件。选择这些默认值是为了确保与运行在较早 OpenSSL 版本 (从 1.0.1 开始) 的系统上的 Zabbix 代理的互操作性。

在较新的系统中，你可以选择通过只允许一些密码套件来加强安全性，例如。只有 PFS(完全向前保密) 密码套件。让我们尝试只允许使用’ TLSCipher* ’ 参数的 PFS 加密套件。

Attention:
如果使用 PSK，结果将无法与使用 OpenSSL 1.0.1 和 1.0.2 的系统进行互操作。基于证书的加密应该可以工作。

添加两行到’ test.conf ’ 配置文件:

```
TLSCipherCert=EECDH+aRSA+AES128
TLSCipherPSK=kECDHEPSK+AES128
```

然后再测试一次:

```
$ zabbix_sender -vv -c /home/zabbix/test.conf -k nonexisting_item -o 1 2>&1 | grep ciphersuites
zabbix_sender [42892]: DEBUG: zbx_tls_init_child() certificate ciphersuites: TLS_AES_256_GCM_SHA384 TLS_
zabbix_sender [42892]: DEBUG: zbx_tls_init_child() PSK ciphersuites: TLS_CHACHA20_POLY1305_SHA256 TLS_AE
zabbix_sender [42892]: DEBUG: zbx_tls_init_child() certificate and PSK ciphersuites: TLS_AES_256_GCM_SHA
```

“证书密码套件” 和 “PSK 密码套件” 列表已更改 - 它们比以前短，只包含预期的 TLS 1.3 加密套件和 TLS 1.2 ECDHE-* 加密套件。

2. TLSCipherAll 和 TLSCipherAll13 不能被 zabbix_sender 测试; 它们不会影响上面示例中显示的 “证书和 PSK 密码套件” 的值。要调整 TLSCipherAll 和 TLSCipherAll13，你需要用 agent、proxy 或 server 进行实验。

因此，为了只允许 PFS 密码套件，您可能需要添加最多三个参数

```
TLSCipherCert=EECDH+aRSA+AES128
TLSCipherPSK=kECDHEPSK+AES128
TLSCipherAll=EECDH+aRSA+AES128:kECDHEPSK+AES128
```

如果 zabbix_agentd.conf、zabbix_proxy.conf 和 zabbix_server_conf 都配置了证书，并且 agent 也有 PSK。

如果 Zabbix 环境只使用基于 psk 的加密并且没有证书，那么只有一个:

```
TLSCipherPSK=kECDHEPSK+AES128
```

现在您已经了解了它的工作原理，您可以在 Zabbix 之外使用 “openssl” 命令测试密码套件的选择。让我们测试三个’ TLSCipher* ’ 参数值:

```
$ openssl ciphers EECDH+aRSA+AES128 | sed 's/:/ /g'
TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256 TLS_AES_128_GCM_SHA256 ECDHE-RSA-AES128-GCM-SHA256 E
$ openssl ciphers kECDHEPSK+AES128 | sed 's/:/ /g'
TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256 TLS_AES_128_GCM_SHA256 ECDHE-PSK-AES128-CBC-SHA256 E
$ openssl ciphers EECDH+aRSA+AES128:kECDHEPSK+AES128 | sed 's/:/ /g'
TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256 TLS_AES_128_GCM_SHA256 ECDHE-RSA-AES128-GCM-SHA256 E
```

你可能更喜欢 openssl ciphers 带参数 -V 执行得到更详细的信息输出:

```
$ openssl ciphers -V EECDH+aRSA+AES128:kECDHEPSK+AES128
          0x13,0x02 - TLS_AES_256_GCM_SHA384  TLSv1.3 Kx=any      Au=any  Enc=AESGCM(256) Mac=AEAD
          0x13,0x03 - TLS_CHACHA20_POLY1305_SHA256 TLSv1.3 Kx=any      Au=any  Enc=CHACHA20/POLY1305(256
          0x13,0x01 - TLS_AES_128_GCM_SHA256  TLSv1.3 Kx=any      Au=any  Enc=AESGCM(128) Mac=AEAD
          0xC0,0x2F - ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AESGCM(128) Mac=AEAD
          0xC0,0x27 - ECDHE-RSA-AES128-SHA256 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AES(128)  Mac=SHA256
          0xC0,0x13 - ECDHE-RSA-AES128-SHA    TLSv1 Kx=ECDH     Au=RSA  Enc=AES(128)  Mac=SHA1
          0xC0,0x37 - ECDHE-PSK-AES128-CBC-SHA256 TLSv1 Kx=ECDHEPSK Au=PSK  Enc=AES(128)  Mac=SHA256
          0xC0,0x35 - ECDHE-PSK-AES128-CBC-SHA TLSv1 Kx=ECDHEPSK Au=PSK  Enc=AES(128)  Mac=SHA1
```

类似地，你可以测试 GnuTLS 的优先级字符串:

```
$ gnutls-cli -l --priority=NONE:+VERS-TLS1.2:+ECDHE-RSA:+AES-128-GCM:+AES-128-CBC:+AEAD:+SHA256:+CURVE-A
Cipher suites for NONE:+VERS-TLS1.2:+ECDHE-RSA:+AES-128-GCM:+AES-128-CBC:+AEAD:+SHA256:+CURVE-ALL:+COMP-
TLS_ECDHE_RSA_AES_128_GCM_SHA256                        0xc0, 0x2f      TLS1.2
TLS_ECDHE_RSA_AES_128_CBC_SHA256                        0xc0, 0x27      TLS1.2

Protocols: VERS-TLS1.2
Ciphers: AES-128-GCM, AES-128-CBC
MACs: AEAD, SHA256
Key Exchange Algorithms: ECDHE-RSA
Groups: GROUP-SECP256R1, GROUP-SECP384R1, GROUP-SECP521R1, GROUP-X25519, GROUP-X448, GROUP-FFDHE2048, GR
PK-signatures: SIGN-RSA-SHA256, SIGN-RSA-PSS-SHA256, SIGN-RSA-PSS-RSAE-SHA256, SIGN-ECDSA-SHA256, SIGN-E
```

从 AES128 到 AES256 转换

Zabbix 使用 AES128 作为数据的内置默认值。让我们假设您正在使用证书，并希望切换到 OpenSSL 1.1.1 上的 AES256。这可以通过添加相应的参数来实现 zabbix_server.conf:

```
TLSCAFile=/home/zabbix/ca.crt
TLSCertFile=/home/zabbix/server.crt
TLSKeyFile=/home/zabbix/server.key
TLSCipherCert13=TLS_AES_256_GCM_SHA384
TLSCipherCert=EECDH+aRSA+AES256:-SHA1:-SHA384
TLSCipherPSK13=TLS_CHACHA20_POLY1305_SHA256
TLSCipherPSK=kECDHEPSK+AES256:-SHA1
TLSCipherAll13=TLS_AES_256_GCM_SHA384
TLSCipherAll=EECDH+aRSA+AES256:-SHA1:-SHA384
```

> **Attention:**
> 虽然只使用与证书相关的密码套件，但' TLSCipherPSK* ' 参数也被定义得很好，以避免它们的默认值包括较不安全的密码，以实现更广泛的互操作性。不能在 server/proxy 上完全禁用 PSK 密码套件。

并且在 zabbix_agentd.conf:

```
TLSConnect=cert
TLSAccept=cert
TLSCAFile=/home/zabbix/ca.crt
TLSCertFile=/home/zabbix/agent.crt
TLSKeyFile=/home/zabbix/agent.key
TLSCipherCert13=TLS_AES_256_GCM_SHA384
TLSCipherCert=EECDH+aRSA+AES256:-SHA1:-SHA384
```

1 使用证书

概览

Zabbix 可以使用 PEM 格式的 RSA 证书，由公共或内部认证机构（CA）签名。根据预先配置的 CA 证书进行证书验证。不支持自签名证书。可以选择使用证书撤销列表（CRL）。每个 Zabbix 组件只能配置一个证书。

有关如何设置和操作内部 CA 的更多信息，如何生成证书请求并签名，如何撤销证书，您可以找到许多在线操作，例如 OpenSSL PKI Tutorialv1.1 .

仔细考虑和测试证书扩展 - 请参阅使用 X.509 v3 证书扩展的限制。

证书配置参数

参数	是否必须	描述
TLSCAFile	*	包含用于对等证书验证的顶级 CA 证书的文件的完整路径名。在具有多个成员的证书链的情况下，它们必须被排序：较低级别的 CA 证书，然后是较高级别的 CA 证书。来自多个 CA 的证书可以包含在单个文件中。
TLSCRLFile		包含证书吊销列表的文件的完整路径名。看证书吊销清单 (CRL)。
TLSCertFile	*	包含证书（证书链）的文件的完整路径名。在有几个成员的证书链中，它们必须排序: 首先是 server、proxy 或 agent 证书，然后是低级 CA 证书，然后是高级 CA 证书。
TLSKeyFile	*	包含私钥的文件的完整路径名。设置此文件的访问权限—它必须只有 Zabbix 用户可读。
TLSServerCertIssuer		-允许的服务器证书颁发者。
TLSServerCertSubject		允许的服务器证书主题。

Configuration examples

After setting up the necessary certificates, configure Zabbix components to use certificate-based encryption.

Below are detailed steps for configuring:

- Zabbix server
- Zabbix proxy
- Zabbix agent

在 Zabbix server 上配置证书

1. 为了验证对等证书，Zabbix server 必须具有使用其顶级自签名根 CA 证书的文件访问权限。例如，如果我们期望来自两个独立根 CA 的证书，我们可以将其证书放入文件中 `/home/zabbix/zabbix_ca_file`：

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: DC=com, DC=zabbix, O=Zabbix SIA, OU=Development group, CN=Root1 CA
            ...
        Subject: DC=com, DC=zabbix, O=Zabbix SIA, OU=Development group, CN=Root1 CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
            ...
        X509v3 extensions:
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            X509v3 Basic Constraints: critical
                CA:TRUE
            ...
-----BEGIN CERTIFICATE-----
MIID2jCCAsKgAwIBAgIBATANBgkqhkiG9w0BAQUFADB+MRMwEQYKCZImiZPyLGQB
....
9wEzdN8uTrqoyU78gi12npLj08LegRKjb5hFTVmO
-----END CERTIFICATE-----
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: DC=com, DC=zabbix, O=Zabbix SIA, OU=Development group, CN=Root2 CA
```

```
            ...
        Subject: DC=com, DC=zabbix, O=Zabbix SIA, OU=Development group, CN=Root2 CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
            ....
        X509v3 extensions:
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            X509v3 Basic Constraints: critical
                CA:TRUE
            ....
-----BEGIN CERTIFICATE-----
MIID3DCCAsSgAwIBAgIBATANBgkqhkiG9w0BAQUFADB/MRMwEQYKCZImiZPyLGQB
...
vdGNYoSfvu41GQAR5Vj5FnRJRzv5XQOZ3B6894GY1zY=
-----END CERTIFICATE-----
```

2. 将 Zabbix 服务器证书链放入文件中，例如/home/zabbix/zabbix_server.crt:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: DC=com, DC=zabbix, O=Zabbix SIA, OU=Development group, CN=Signing CA
        ...
        Subject: DC=com, DC=zabbix, O=Zabbix SIA, OU=Development group, CN=Zabbix server
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                ...
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Basic Constraints:
                CA:FALSE
            ...
-----BEGIN CERTIFICATE-----
MIIECDCCAvCgAwIBAgIBATANBgkqhkiG9w0BAQUFADCBgTETMBEGCgmSJomT8ixk
...
h02u1GHiy46GI+xfR3LsPwFKlkTaaLaL/6aaoQ==
-----END CERTIFICATE-----
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 2 (0x2)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: DC=com, DC=zabbix, O=Zabbix SIA, OU=Development group, CN=Root1 CA
        ...
        Subject: DC=com, DC=zabbix, O=Zabbix SIA, OU=Development group, CN=Signing CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
            ...
        X509v3 extensions:
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
        ...
-----BEGIN CERTIFICATE-----
MIID4TCCAsmgAwIBAgIBAjANBgkqhkiG9w0BAQUFADB+MRMwEQYKCZImiZPyLGQB
...
```

```
dyCeWnvL7u5sd6ffo8iRny0QzbHKmQt/wUtcVIvWXdMIFJM0Hw==
-----END CERTIFICATE-----
```

先放入 Zabbix server 证书，随后是中间 CA 的证书。

3. 将 Zabbix server 私钥放入文件中，例如/home/zabbix/zabbix_server.key：

```
-----BEGIN PRIVATE KEY-----
MIIEwAIBADANBgkqhkiG9w0BAQEFAASCBKowggSmAgEAAoIBAQC9tIXIJoVnNXDl
...
IJLkhbybBYEf47MLhffWa7XvZTY=
-----END PRIVATE KEY-----
```

4. 在 Zabbix server 配置文件中编辑 TLS 参数，如下所示：

```
TLSCAFile=/home/zabbix/zabbix_ca_file
TLSCertFile=/home/zabbix/zabbix_server.crt
TLSKeyFile=/home/zabbix/zabbix_server.key
```

Zabbix proxy 配置基于证书的加密

1. 使用顶级 CA 证书，proxy 证书 (链) 和私钥准备文件，如在Zabbix server 上配置证书中所述。编辑参数 TLSCAFile，TLSCertFile，TLSKeyFile 在 proxy 配置相应。编辑参数 TLSCAFile，TLSCertFile，TLSKeyFile 在 proxy 配置相应。

2. 对于主动模式 proxy 编辑 TLSConnect 参数：

```
TLSConnect=cert
```

对于被动模式 proxy 编辑 TLSAccept 参数：

```
TLSAccept=cert
```

3. 现在你有一个基于证书的最小 proxy 配置。您可能希望通过设置 TLSServerCertIssuer 和 TLSServerCertSubject 参数来提高 proxy 安全性 (请参阅Restricting allowed certificate Issuer and Subject)。

4. 在最终的代理配置文件中，TLS 参数可能看起来像:

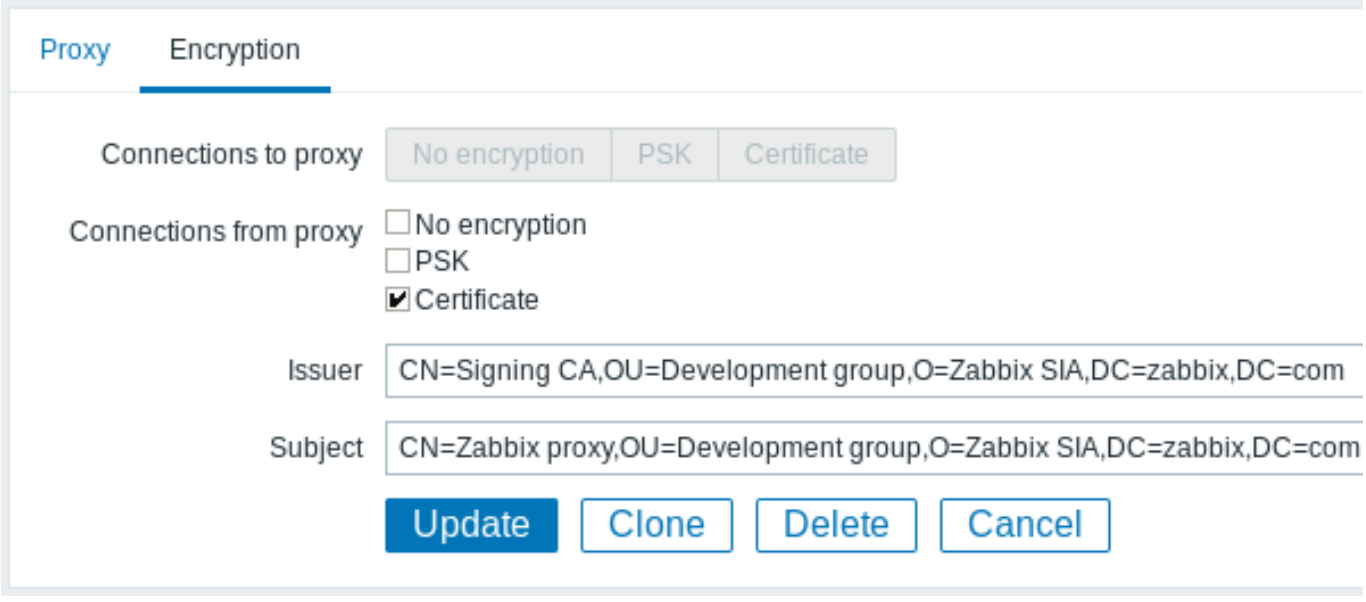
```
TLSConnect=cert
TLSAccept=cert
TLSCAFile=/home/zabbix/zabbix_ca_file
TLSServerCertIssuer=CN=Signing CA,OU=Development group,O=Zabbix SIA,DC=zabbix,DC=com
TLSServerCertSubject=CN=Zabbix server,OU=Development group,O=Zabbix SIA,DC=zabbix,DC=com
TLSCertFile=/home/zabbix/zabbix_proxy.crt
TLSKeyFile=/home/zabbix/zabbix_proxy.key
```

5. 在 Zabbix 前端配置此 proxy 的加密：

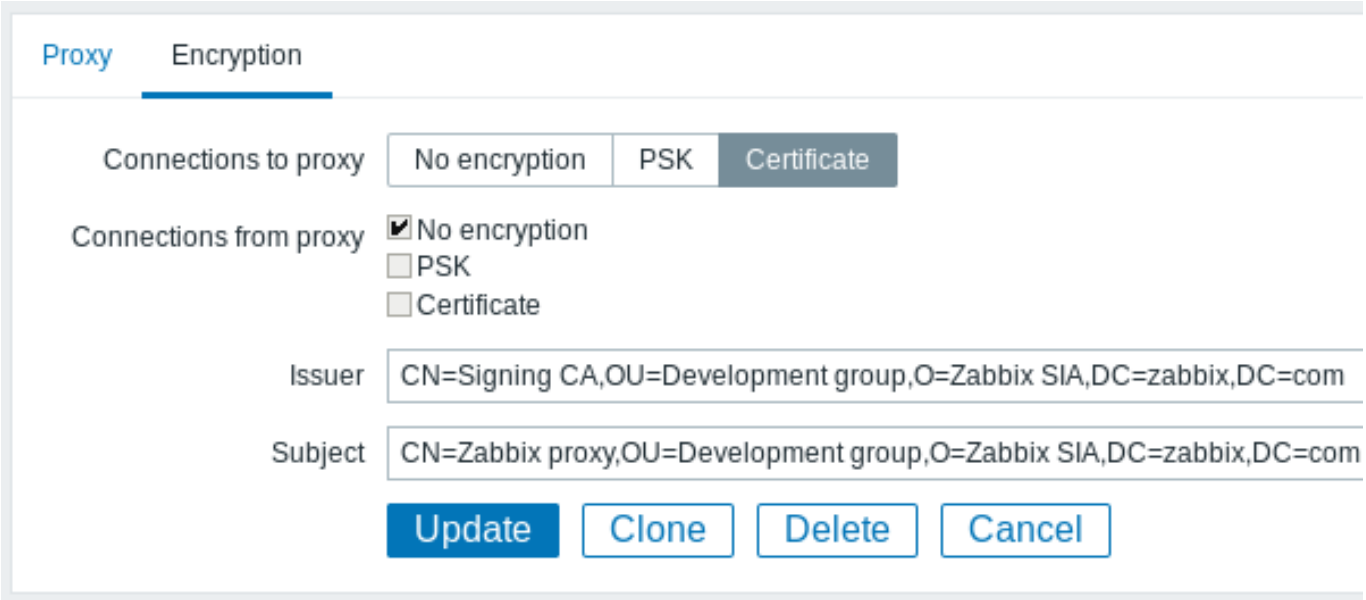
- 转到：管理 →*agent* 代理程序 *(proxies)*
- 选择代理，然后单击加密选项卡

在下面的例子中，发行人和主题字段被填充-请参阅限制允许的证书颁发者和主题 为什么以及如何使用这些字段。

主动模式 proxy



被动模式 proxy



Zabbix agent 配置基于证书的加密

1. 使用顶级 CA 证书，代理证书（链）和私钥准备文件，如在Zabbix server 配置证书中所述。编辑参数 `TLSCAFile`，`TLSCertFile`，`TLSKeyFile` 在 agent 配置相应。

2. 对于主动模式检查编辑 `TLSConnect` 参数：

```
TLSConnect=cert
```

对于被动模式检查编辑 `TLSAccept` 参数：

```
TLSAccept=cert
```

3. 现在，您有一个基于证书的最小 agent 配置。您可能希望通过设置 `TLSServerCertIssuer` 和 `TLSServerCertSubject` 参数提高 agent 安全性。(请参阅限制允许的证书发行者和主体)。

4. 在最终 agent 配置文件中，TLS 参数可能如下所示：

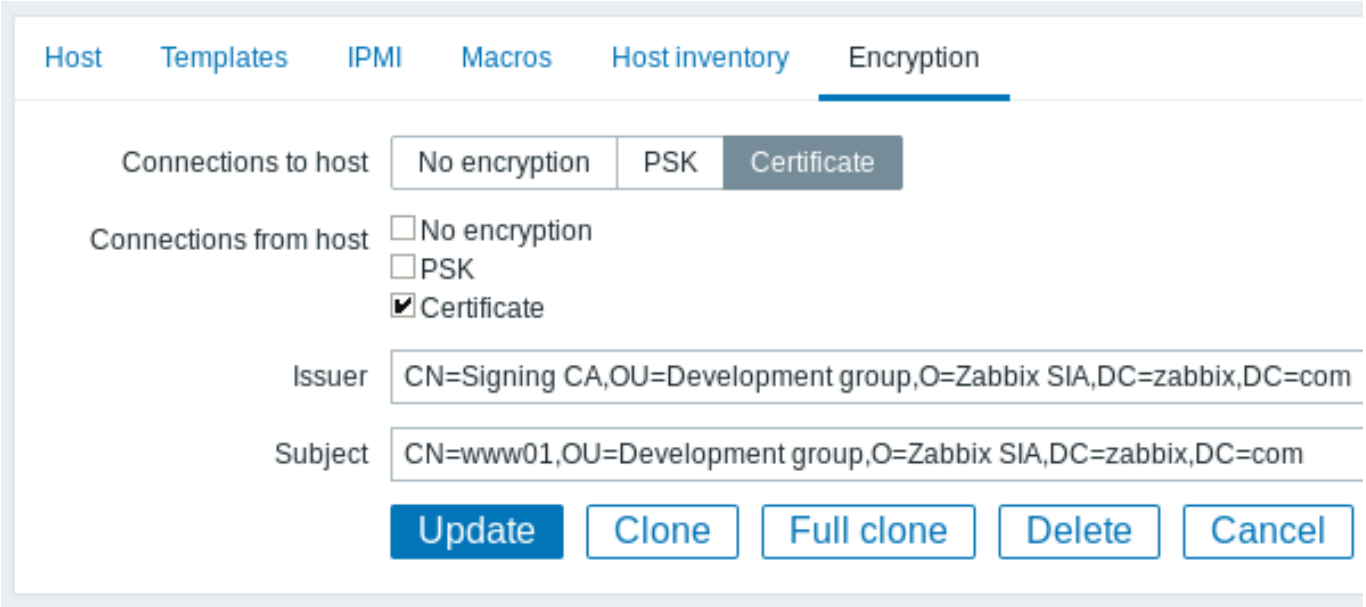
```
TLSConnect=cert
TLSAccept=cert
TLSCAFile=/home/zabbix/zabbix_ca_file
TLSServerCertIssuer=CN=Signing CA,OU=Development group,O=Zabbix SIA,DC=zabbix,DC=com
TLSServerCertSubject=CN=Zabbix proxy,OU=Development group,O=Zabbix SIA,DC=zabbix,DC=com
TLSCertFile=/home/zabbix/zabbix_agentd.crt
TLSKeyFile=/home/zabbix/zabbix_agentd.key
```

(例如，假设主机是通过 proxy 监视的，因此是 proxy 证书主体。)

5. 在 Zabbix 前端为此 agent 配置加密：

- 前往: *Configuration* → *Hosts*
- 选择主机然后点击加密选项卡

在下面的示例中，发行者和主体字段填写 - 请参阅限制允许的证书发行者和主体 原因以及如何使用这些字段。



Zabbix web service

1. Prepare files with the top-level CA certificates, the Zabbix web service certificate/certificate chain, and the private key as described in the Zabbix server section. Then, edit the `TLSCAFile`, `TLSCertFile`, and `TLSKeyFile` parameters in the Zabbix web service configuration file accordingly.

2. Edit an additional TLS parameter in the Zabbix web service configuration file: `TLSAccept=cert`

TLS parameters in the final web service configuration file may look as follows:

```
TLSAccept=cert
TLSCAFile=/home/zabbix/zabbix_ca_file
TLSCertFile=/home/zabbix/zabbix_web_service.crt
TLSKeyFile=/home/zabbix/zabbix_web_service.key
```

3. Configure Zabbix server to connect to the TLS-configured Zabbix web service by editing the `WebServiceURL` parameter in the Zabbix server configuration file:

```
WebServiceURL=https://example.com
```

限制允许的证书发行者和主体

当两个 Zabbix 组件（例如服务端和 agent）建立 TLS 连接时，他们会检查对方的证书。如果对等证书由受信任的 CA（具有预先配置的顶级证书 `TLSCAFile`）签名有效，尚未过期且通过其他检查项，则可以进行通信。在最简单的情况下，不会检查证书发行者和主体。

这存在一个风险 - 任何拥有有效证书的人都可以冒充任何人（例如，主机证书可以用来模拟服务器）。在内部 CA 签发证书的小型环境中，这种风险可能是可以接受的，冒充的风险较低。

如果您的顶级 CA 用于签发其他证书而不应被 Zabbix 接受，或者你想降低冒充风险，您可以通过指定其发行者 (Issuer) 和主体 (Subject) 字符串来限制允许的证书。

例如，您可以在 Zabbix proxy 配置文件中写：

```
TLSServerCertIssuer=CN=Signing CA,OU=Development group,O=Zabbix SIA,DC=zabbix,DC=comTLSServerCertSubject=C
```

通过这些设置，主动 proxy 将不会与证书中具有不同发行者或主体字符串的 Zabbix server 通信，被动 proxy 将不接受来自此类服务器的请求。

有关发行者或主体字符串匹配的说明：

1. 独立检查发行者和主体字符串。两者都是可选的。
2. 允许使用 UTF-8 字符。
3. 未指定的字符串等同于任何字符串都被接受。
4. 字符串按 “原样” 比较，它们必须完全一致才能匹配。
5. 不支持通配符和正则表达式。
6. 仅实现了RFC 4514 轻量级目录访问协议 (LDAP): 可区分名称的字符串表示中的一些要求:
    1. 转义字符’”’ (U+0022), ’+’ U+002B, ’,’ U+002C, ’;’ U+003B, ’<’ U+003C, ’>’ U+003E, ’\’ U+005C 在字符串中的任何地方。
    2. 字符串开头的转义字符空格 (’ ’ U+0020) 或数字符号 (’#’ U+0023)。
    3. 转义字符空间 (’ ’ U+0020) 在字符串的结尾。

7. 如果遇到空字符 (U+0000) (RFC4514允许)，则匹配失败。
8. 由于工作量太大，不支持RFC 4517 轻量级目录访问协议 (LDAP): 语法和匹配规则和RFC 4518 轻量级目录访问协议 (LDAP): 国际化字符串准备的要求。

颁发者和主题字符串中的字段顺序和格式都很重要! Zabbix 遵循RFC 4514的建议，并使用字段的 “倒序”。

相反的顺序可以通过下面的例子来说明:

```
TLSServerCertIssuer=CN=Signing CA,OU=Development group,O=Zabbix SIA,DC=zabbix,DC=com
TLSServerCertSubject=CN=Zabbix proxy,OU=Development group,O=Zabbix SIA,DC=zabbix,DC=com
```

注意，它以低级别 (CN) 字段开始，然后到中级 (OU, O) 字段，最后以顶级 (DC) 字段结束。

OpenSSL 默认显示证书颁发者和主题字段的 “正常” 顺序，取决于使用的其他选项:

```
$ openssl x509 -noout -in /home/zabbix/zabbix_proxy.crt -issuer -subject
issuer= /DC=com/DC=zabbix/O=Zabbix SIA/OU=Development group/CN=Signing CA
subject= /DC=com/DC=zabbix/O=Zabbix SIA/OU=Development group/CN=Zabbix proxy

$ openssl x509 -noout -text -in /home/zabbix/zabbix_proxy.crt
Certificate:
        ...
        Issuer: DC=com, DC=zabbix, O=Zabbix SIA, OU=Development group, CN=Signing CA
    ...
        Subject: DC=com, DC=zabbix, O=Zabbix SIA, OU=Development group, CN=Zabbix proxy
```

在这里，Issuer 和 Subject 字符串以顶级字段 (DC) 开始，以低级字段 (CN) 结束，空格和字段分隔符取决于所使用的选项。这些值在 Zabbix 发行人和主题字段中都不匹配!

Attention:
要获得在 Zabbix 中可用的正确的发行者和主题字符串，需要使用特殊选项调用 OpenSSL -nameopt esc_2253,esc_ctrl,utf8,dump_nostr,dump_unknown,dump_der,sep_comma_plus,dn_rev,sname:

```
$ openssl x509 -noout -issuer -subject \
        -nameopt esc_2253,esc_ctrl,utf8,dump_nostr,dump_unknown,dump_der,sep_comma_plus,dn_rev,sname \
        -in /home/zabbix/zabbix_proxy.crt
issuer= CN=Signing CA,OU=Development group,O=Zabbix SIA,DC=zabbix,DC=com
subject= CN=Zabbix proxy,OU=Development group,O=Zabbix SIA,DC=zabbix,DC=com
```

现在字符串字段是倒序的，字段是用逗号分隔的，可以在 Zabbix 配置文件和前端使用。

Rules for matching `Issuer` and `Subject` strings

The rules for matching `Issuer` and `Subject` strings are as follows:

- `Issuer` and `Subject` strings are checked independently. Both are optional.
- An unspecified string means that any string is accepted.
- Strings are compared *as is* and must match exactly.
- UTF-8 characters are supported. However, wildcards (*) or regular expressions are not supported.
- The following RFC 4514 requirements are implemented - characters that require escaping (with a '\' backslash, U+005C):
  - anywhere in the string: '"' (U+0022), '+' (U+002B), ',' (U+002C), ';' (U+003B), '<' (U+003C), '>' (U+003E), '\\' (U+005C);
  - at the beginning of the string: space (' ', U+0020) or number sign ('#', U+0023);
  - at the end of the string: space (' ', U+0020).
- Null characters (U+0000) are not supported. If a null character is encountered, the matching will fail.
- RFC 4517 and RFC 4518 standards are not supported.

For example, if `Issuer` and `Subject` organization (`O`) strings contain trailing spaces and the `Subject` organizational unit (`OU`) string contains double quotes, these characters must be escaped:

```
TLSServerCertIssuer=CN=Signing CA,OU=Development head,O=\ Example SIA\ ,DC=example,DC=com
TLSServerCertSubject=CN=Zabbix server,OU=Development group \"5\",O=\ Example SIA\ ,DC=example,DC=com
```

Field order and formatting

Zabbix follows the recommendations of RFC 4514, which specifies a "reverse" order for these fields, starting with the lowest-level fields (`CN`), proceeding to the mid-level fields (`OU`, `O`), and concluding with the highest-level fields (`DC`).

```
TLSServerCertIssuer=CN=Signing CA,OU=Development group,O=Zabbix SIA,DC=zabbix,DC=com
TLSServerCertSubject=CN=Zabbix proxy,OU=Development group,O=Zabbix SIA,DC=zabbix,DC=com
```

In contrast, OpenSSL by default displays the `Issuer` and `Subject` strings in top-level to low-level order. In the following example, `Issuer` and `Subject` fields start with the top-level (DC) and end with the low-level (CN) field. The formatting with spaces and field separators also varies based on the options used, and thus will not match the format required by Zabbix.

```
$ openssl x509 -noout -in /home/zabbix/zabbix_proxy.crt -issuer -subject
issuer= /DC=com/DC=zabbix/O=Zabbix SIA/OU=Development group/CN=Signing CA
subject= /DC=com/DC=zabbix/O=Zabbix SIA/OU=Development group/CN=Zabbix proxy

$ openssl x509 -noout -text -in /home/zabbix/zabbix_proxy.crt
Certificate:
    ...
        Issuer: DC=com, DC=zabbix, O=Zabbix SIA, OU=Development group, CN=Signing CA
        ...
        Subject: DC=com, DC=zabbix, O=Zabbix SIA, OU=Development group, CN=Zabbix proxy
```

To format *Issuer* and *Subject* strings correctly for Zabbix, invoke OpenSSL with the following options:

```
$ openssl x509 -noout -issuer -subject \
    -nameopt esc_2253,esc_ctrl,utf8,dump_nostr,dump_unknown,dump_der,sep_comma_plus,dn_rev,sname\
    -in /home/zabbix/zabbix_proxy.crt
```

The output will then be in reverse order, comma-separated, and usable in Zabbix configuration files and frontend:

```
issuer= CN=Signing CA,OU=Development group,O=Zabbix SIA,DC=zabbix,DC=com
subject= CN=Zabbix proxy,OU=Development group,O=Zabbix SIA,DC=zabbix,DC=com
```

使用 X.509 v3 证书扩展的限制

- 主题备用名称（***subjectAltName***）扩展名。
  Zabbix 不支持来自 _subjectAltName_ 扩展名的替代主体名称（如 IP 地址，电子邮件地址）。只能在 Zabbix 中检查 “主体” 字段的值（请参阅限制允许的证书发行者和主体）。如果证书使用 _subjectAltName_ 扩展名，那么结果取决于加密工具包的特定组合。Zabbix 组件被编译（可能工作或不工作，Zabbix 可能拒绝接受来自对等体的证书）
- 扩展密钥使用扩展。
  如果使用，则通常需要 *clientAuth*（TLS WWW 客户端身份验证）和 *serverAuth*（TLS WWW 服务器身份验证）。
  例如，被动检查的 zabbix agent 是作为 TLS 服务器，所以”serverAuth 必须在 *agent* 证书设置。对于主动检查 *agent* 证书需要 clientAuth 进行设置。GnuTLS* 在违规使用情况下发出警告，但允许通信进行。
- 名称限制扩展。
  并不是所有的加密工具包都支持它。此扩展可能会阻止 Zabbix 加载 CA 证书，此部分被标记为 _ 关键 *(critical)*（取决于特定的加密工具包）。

证书撤销列表 (CRL)

如果证书被破坏，CA 可以通过将其包含在 CRL 中来撤销它。CRLs 可以通过 “TLSCRLFile” 参数在 server、proxy 和 agent 配置文件中配置。例如:

```
TLSCRLFile=/home/zabbix/zabbix_crl_file
```

其中’ zabbix_crl_file ’ 可能包含来自多个 CAs 的 CRLs，如下所示:

```
-----BEGIN X509 CRL-----
MIIB/DCB5QIBATANBgkqhkiG9w0BAQUFADCBgTETMBEGCgmSJomT8ixkARkWA2Nv
...
treZeUPjb7LSmZ3K2hpbZN7SoOZcAoHQ3GWd9npuctg=
-----END X509 CRL-----
-----BEGIN X509 CRL-----
MIIB+TCB4gIBATANBgkqhkiG9w0BAQUFADB/MRMwEQYKCZImiZPyLGQBGRYDY29t
...
CAEebS2CND3ShBedZ8YSil59O6JvaDP61lR5lNs=
-----END X509 CRL-----
```

CRL 文件只在 Zabbix 启动时加载。更新 CRL 需要重新启动。

Attention:
如果 Zabbix 组件是用 *OpenSSL* 编译的，并且使用了 CRLs，那么证书链中的每个顶级和中级 CA 必须在’ TLSCRLFile’ 中有一个对应的 CRL(可以为空)。

2 使用预共享秘钥

概述

Zabbix 中的每个预共享密钥（PSK）实际上是一对：

- 非秘密 PSK 标识字符串，
- 秘密 PSK 字符串值。

PSK 标识字符串是非空 UTF-8 字符串。例如，”PSK ID 001 Zabbix agentd”。它是一个唯一的名称，Zabbix 组件通过该名称引用此特定 PSK。不要将敏感信息放在 PSK 标识字符串中 - 它是通过未加密的网络传输的。

PSK 值是一个难以猜测的十六进制数字字符串，例如，”e560cb0d918d26d31b4f642181f5f570ad89a390931102e5391d08327ba434e9”。

尺寸限制

Zabbix 中的 PSK 标识和值有大小限制，在某些情况下，加密库可以具有下限:

组件	PSK 标识最大尺寸	PSK 值最小尺寸	PSK 值最大尺寸
Zabbix	128 个 UTF-8 字符	128 位（16 字节 PSK，输入为 32 个十六进制数字）	2048 位（256 字节 PSK，输入为 512 个十六进制数字）
GnuTLS	128 字节（可能包括 UTF-8 字符）	-	2048 位（256 字节 PSK，输入为 512 个十六进制数字）
OpenSSL 1.0.x, 1.1.0	127 字节（可能包括 UTF-8 字符）	-	2048 位（256 字节 PSK，输入为 512 个十六进制数字）
OpenSSL 1.1.1	127 字节（可能包括 UTF-8 字符）	-	512 位（64 字节 PSK，输入为 128 个十六进制数字）
OpenSSL 1.1.1a 及更高版本	127 字节（可能包括 UTF-8 字符）	-	2048 位（256 字节 PSK，输入为 512 个十六进制数字）

Zabbix 前端允许配置最多 128 个字符长的 PSK 标识字符串和 2048 位长的 PSK，而不考虑使用哪种加密库。
如果某些 Zabbix 组件支持下限，则用户有责任为这些组件配置具有允许长度的 PSK 标识和值。
超过长度限制会导致 Zabbix 组件之间的通信故障。

在 Zabbix 服务器使用 PSK 连接到代理之前，服务器会在数据库（实际上是在配置缓存中）中查找为该代理配置的 PSK 标识和 PSK 值。收到连接后，代理使用其配置文件中的 PSK 标识和 PSK 值。如果双方具有相同的 PSK 标识字符串和 PSK 值，则连接可能会成功。

用户有责任确保没有两个具有相同标识字符串但值不同的 PSK。否则，使用带有此 PSK 标识字符串的 PSK 的 Zabbix 组件之间的通信可能会出现不可预测的中断。

生成 PSK

例如，可以使用以下命令生成 256 位（32 字节）PSK:

- 对于 *OpenSSL*:

```
$ openssl rand -hex 32
af8ced32dfe8714e548694e2d29e1a14ba6fa13f216cb35c19d0feb1084b0429
```

- 对于 *GnuTLS*:

请注意，上面的”psktool” 会生成一个具有 PSK 标识及其关联 PSK 的数据库文件。Zabbix 期望 PSK 文件中只有一个 PSK，因此应从文件中删除标识字符串和冒号（’:’）。

为服务器-代理通信配置 PSK（示例）

在代理主机上，将 PSK 值写入文件，例如，`/home/zabbix/zabbix_agentd.psk`。该文件必须在第一个文本字符串中包含 PSK，例如：

```
1f87b595725ac58dd977beef14b97461a7c1045b9a1c963065002c5473194952
```

设置 PSK 文件的访问权限 - 它必须只能由 Zabbix 用户读取。

在代理配置文件 `zabbix_agentd.conf` 中编辑 TLS 参数，例如，设置：

```
TLSConnect=psk
TLSAccept=psk
TLSPSKFile=/home/zabbix/zabbix_agentd.psk
TLSPSKIdentity=PSK 001
```

代理将连接到服务器（活动检查）并只接受来自 zabbix_get 使用 PSK 的服务器连接。PSK 标识将是"PSK 001"。

重新启动代理。现在，您可以使用 zabbix_get 测试连接，例如:

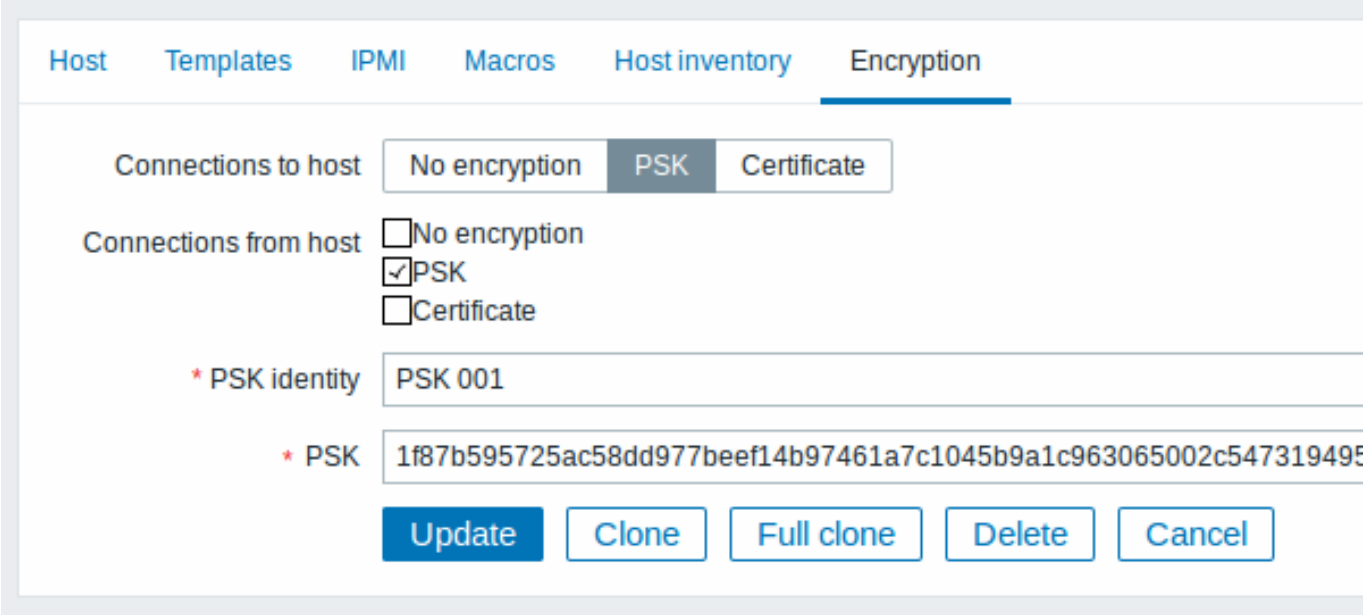
```
$ zabbix_get -s 127.0.0.1 -k "system.cpu.load[all,avg1]" --tls-connect=psk \
--tls-psk-identity="PSK 001" --tls-psk-file=/home/zabbix/zabbix_agentd.psk
```

(为了最大限度地减少停机时间，请参阅如何在连接加密管理) 中更改连接类型

在 Zabbix 前端中为此代理配置 PSK 加密：

- 转到: 配置 → 主机
- 选择主机，然后单击 加密选项卡

例如:



所有必填字段都标有红色星号。

当配置缓存与数据库同步时，新连接将使用 PSK。检查服务器和代理日志文件中是否有错误消息。

为服务器配置 PSK - 活动代理通信（示例）

在代理上，将 PSK 值写入文件，例如, /home/zabbix/zabbix_proxy.psk。该文件必须在第一个文本字符串中包含 PSK，例如：

```
e560cb0d918d26d31b4f642181f5f570ad89a390931102e5391d08327ba434e9
```

设置 PSK 文件的访问权限 - 它必须只能由 Zabbix 用户读取。

在代理配置文件 zabbix_proxy.conf 中编辑 TLS 参数, 例如，设置:

```
TLSConnect=psk
TLSPSKFile=/home/zabbix/zabbix_proxy.psk
TLSPSKIdentity=PSK 002
```

代理将使用 PSK 连接到服务器。PSK 标识将是"PSK 002"。

(为最大限度地减少停机时间，请参阅连接加密管理)。

在 Zabbix 前端中为此代理配置 PSK。转到管理 → 代理, 选择代理, 转到" 加密" 选项卡。在" 来自代理的连接" 中，标记 PSK。将"PSK 002" 粘贴到"PSK 标识" 字段中，将"e560cb0d918d26d31b4f642181f5f570ad89a390931102e5391d08327ba434e9" 粘贴到"PSK" 字段。点击" 更新"。

重新启动代理。它将开始使用基于 PSK 的加密连接到服务器。检查服务器和代理日志文件中是否有错误消息。

对于被动代理，该过程非常相似。唯一的区别是 - 在代理配置文件中设置 TLSAccept=psk，并将 Zabbix 前端中的" 连接到代理" 设置为 PSK。

3 故障排除

一般建议

- 首先了解哪些组件充当 TLS 客户端，哪个组件在出现问题时充当 TLS 服务器。Zabbix server、proxies 和 agents，取决于它们之间的交互，都可以用作 TLS 服务器和客户端。例如，Zabbix server 充当 TLS 客户端连接到 agent 进行被动检查。Agent 充当 TLS 服务器的角色。
  Zabbix agent 充当 TLS 客户端从 proxy 请求活动检查列表。Proxy 充当 TLS 服务器的角色。
  "zabbix_get" 和"zabbix_sender" 应用程序始终充当 TLS 客户端。
- Zabbix 使用相互身份验证。
  每一方都会验证其对等方，并可能拒绝连接。
  例如，如果 agent 的证书无效，连接到 agent 的 Zabbix server 可以立即关闭连接。反之亦然 - 如果 server 不受 agent 信任，则接受来自 server 的连接的 Zabbix agent 也可以关闭连接。
- 检查两端的日志文件 - 在 TLS 客户端和 TLS 服务器中。拒绝连接的一方可以记录被拒绝的确切原因。另一方经常报告相当一般的错误（例如，" 对等方关闭的连接"，" 连接未正确终止"）。
- 有时，错误配置的加密会导致令人困惑的错误消息，而无法指向真正的原因。在下面的小节中，我们尝试提供（仍有未尽之处）消息和可能原因的集合，以帮助进行故障排除。请注意，不同的加密工具包（OpenSSL，GnuTLS）在相同的问题情况下通常会产生不同的错误消息。有时，错误消息甚至取决于双方加密工具包的特定组合。

1 连接类型或权限问题

Server 配置为使用 PSK 连接到 agent，但 agent 仅接受未加密的连接

在 server 或 proxy 日志中有如下显示 (对于 *GnuTLS* 3.3.16)

```
Get value from agent failed: zbx_tls_connect(): gnutls_handshake() failed: \
    -110 The TLS connection was non-properly terminated.
```

在 server 或 proxy 日志中有如下显示 (对于 *OpenSSL* 1.0.2c)

```
Get value from agent failed: TCP connection successful, cannot establish TLS to [[127.0.0.1]:10050]: \
    Connection closed by peer. Check allowed connection types and access rights
```

一端使用证书连接，但另一端仅接受 PSK，反之亦然

在任意日志当中有如下显示 (对于 *GnuTLS*)：

```
failed to accept an incoming connection: from 127.0.0.1: zbx_tls_accept(): gnutls_handshake() failed:\
    -21 Could not negotiate a supported cipher suite.
```

在任意日志当中有如下显示 (对于 *OpenSSL* 1.0.2c)：

```
failed to accept an incoming connection: from 127.0.0.1: TLS handshake returned error code 1:\
    file .\ssl\s3_srvr.c line 1411: error:1408A0C1:SSL routines:ssl3_get_client_hello:no shared cipher:\
    TLS write fatal alert "handshake failure"
```

尝试使用使用 TLS 支持编译的 Zabbix sender 将数据发送到在没有 TLS 的情况下编译的 Zabbix Server/proxy

在发起连接方的日志中有如下显示：

Linux:

```
...In zbx_tls_init_child()
...OpenSSL library (version OpenSSL 1.1.1  11 Sep 2018) initialized
...
...In zbx_tls_connect(): psk_identity:"PSK test sender"
...End of zbx_tls_connect():FAIL error:'connection closed by peer'
...send value error: TCP successful, cannot establish TLS to [[localhost]:10051]: connection closed by pee
```

Windows:

```
...OpenSSL library (version OpenSSL 1.1.1a  20 Nov 2018) initialized
...
...In zbx_tls_connect(): psk_identity:"PSK test sender"
...zbx_psk_client_cb() requested PSK identity "PSK test sender"
...End of zbx_tls_connect():FAIL error:'SSL_connect() I/O error: [0x00000000] The operation completed succ
...send value error: TCP successful, cannot establish TLS to [[192.168.1.2]:10051]: SSL_connect() I/O erro
```

在接受连接端的日志中有如下显示：

```
...failed to accept an incoming connection: from 127.0.0.1: support for TLS was not compiled in
```

一端使用 PSK 连接，但另一端使用 LibreSSL 或已编译为无加密支持

LibreSSL 不支持 PSK。

在发起连接端的日志中有如下显示：

```
...TCP successful, cannot establish TLS to [[192.168.1.2]:10050]: SSL_connect() I/O error: [0] Success
```

在接受连接端的日志中：

```
...failed to accept an incoming connection: from 192.168.1.2: support for PSK was not compiled in
```

在 Zabbix 前端有如下显示：

```
Get value from agent failed: TCP successful, cannot establish TLS to [[192.168.1.2]:10050]: SSL_connect()
```

一端使用 PSK 连接，但另一端使用禁用 PSK 支持的 OpenSSL

在发起连接端的日志中有如下显示：

```
...TCP successful, cannot establish TLS to [[192.168.1.2]:10050]: SSL_connect() set result code to SSL_ERR
```

在接受连接端的日志中有如下显示：

```
...failed to accept an incoming connection: from 192.168.1.2: TLS handshake set result code to 1: file ssl
```

2 证书问题

OpenSSL 与 CRL 一起使用，对于证书链中的某些 CA，其 CRL 不包括在"TLSCRLFile" 中

在 *OpenSSL* 对端的情况下，在 TLS 服务器日志中，会有如下显示：

```
failed to accept an incoming connection: from 127.0.0.1: TLS handshake with 127.0.0.1 returned error code
    file s3_srvr.c line 3251: error:14089086: SSL routines:ssl3_get_client_certificate:certificate verify
    TLS write fatal alert "unknown CA"
```

在 *GnuTLS* 对端的情况下，在 TLS 服务器日志中，会有如下显示：

```
failed to accept an incoming connection: from 127.0.0.1: TLS handshake with 127.0.0.1 returned error code
    file rsa_pk1.c line 103: error:0407006A: rsa routines:RSA_padding_check_PKCS1_type_1:\
    block type is not 01 file rsa_eay.c line 705: error:04067072: rsa routines:RSA_EAY_PUBLIC_DECRYPT:padd
```

CRL 已过期或者在服务器操作期间过期

OpenSSL, 在服务器日志中会议如下显示：

- 到期前：

```
cannot connect to proxy "proxy-openssl-1.0.1e": TCP successful, cannot establish TLS to [[127.0.0.1]:20004
    SSL_connect() returned SSL_ERROR_SSL: file s3_clnt.c line 1253: error:14090086:\
    SSL routines:ssl3_get_server_certificate:certificate verify failed:\
    TLS write fatal alert "certificate revoked"
```

- 到期后：

```
cannot connect to proxy "proxy-openssl-1.0.1e": TCP successful, cannot establish TLS to [[127.0.0.1]:20004
    SSL_connect() returned SSL_ERROR_SSL: file s3_clnt.c line 1253: error:14090086:\
    SSL routines:ssl3_get_server_certificate:certificate verify failed:\
    TLS write fatal alert "certificate expired"
```

这里需要注意的是，使用有效的 CRL，吊销的证书将报告为" 证书已吊销"。当 CRL 过期时，错误消息将更改为" 证书已过期"，这非常具有误导性。

GnuTLS, 在服务器日志中会议如下显示：

- 到期前后显示相同：

```
cannot connect to proxy "proxy-openssl-1.0.1e": TCP successful, cannot establish TLS to [[127.0.0.1]:20004
      invalid peer certificate: The certificate is NOT trusted. The certificate chain is revoked.
```

自签证书，未知 CA

OpenSSL, 在日志中显示：

```
error:'self signed certificate: SSL_connect() set result code to SSL_ERROR_SSL: file ../ssl/statem/statem_
      line 1924: error:1416F086:SSL routines:tls_process_server_certificate:certificate verify failed:\
      TLS write fatal alert "unknown CA"'
```

当服务器证书错误地具有相同的颁发者和主题字符串时，尽管该字符串是由 CA 签名的，但颁发者和主题在顶级 CA 证书中相同，但在服务器证书中不能相同时，就会看到这种情况。(这同样适用于 proxy 和 agent 证书。)

3 PSK 问题

PSK 包含奇数个十六进制数字

Proxy 或者 agent 不能启动，proxy 或者 agent 日志中会有如下消息：

```
invalid PSK in file "/home/zabbix/zabbix_proxy.psk"
```

长度超过 128 个字节的 PSK 标识字符串将传递到 GnuTLS

在 TLS 客户端日志中会显示如下信息：

```
gnutls_handshake() failed: -110 The TLS connection was non-properly terminated.
```

在 TLS 服务器端日志中会显示如下信息：

```
gnutls_handshake() failed: -90 The SRP username supplied is illegal.
```

OpenSSL 1.1.1 使用了过长的 PSK 值

在连接发起端的日志中，会有如下信息显示：

```
...OpenSSL library (version OpenSSL 1.1.1  11 Sep 2018) initialized
...
...In zbx_tls_connect(): psk_identity:"PSK 1"
...zbx_psk_client_cb() requested PSK identity "PSK 1"
...End of zbx_tls_connect():FAIL error:'SSL_connect() set result code to SSL_ERROR_SSL: file ssl\statem\ex
```

在连接接收端的日志中，会有如下信息显示：

```
...Message from 123.123.123.123 is missing header. Message ignored.
```

将 OpenSSL 从 1.0.x 或 1.1.0 升级到 1.1.1 以及 PSK 值的长度超过 512 位（64 字节 PSK，以 128 个十六进制数字输入）时，通常会出现此问题。

参考: 值大小限制

18. Web 界面

概述　为了可以从任何平台在任何位置轻松的访问 Zabbix，我们提供了基于 Web 的界面。

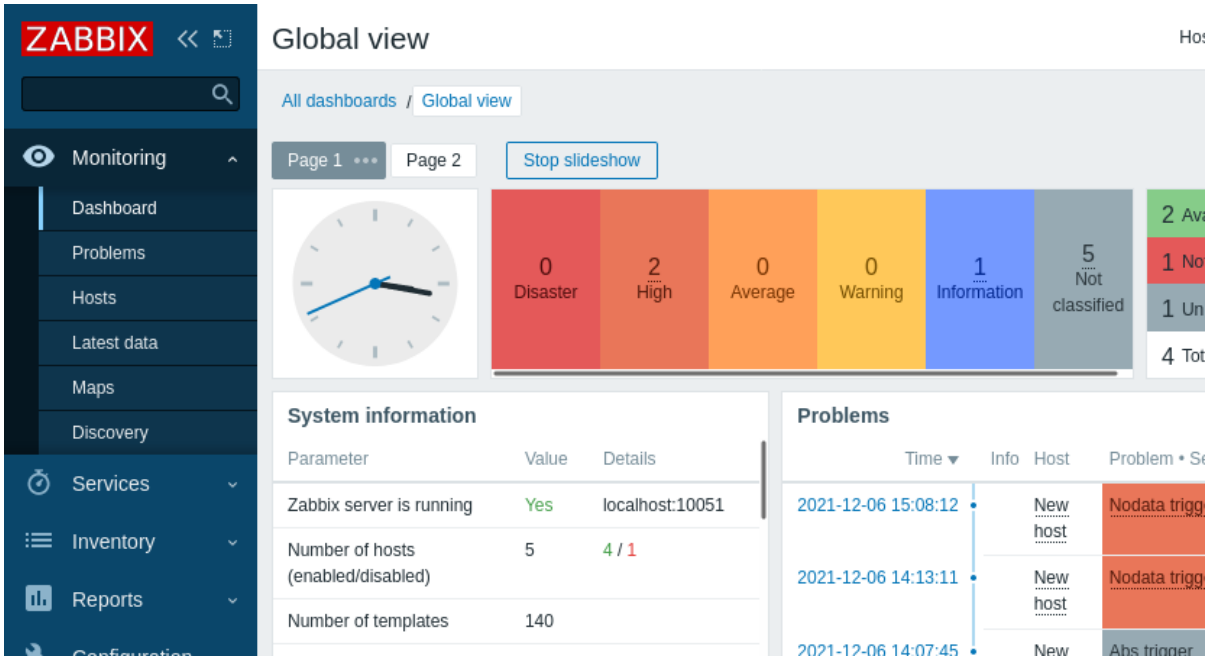
> **Note:**
> 如果使用超过一个前端实例，需要确保语言环境和库（LDAP, SAML 等等）已经安装并在所有前端有相同的配置

1 菜单

概述

侧边栏的垂直菜单提供了到多个 Zabbix 前端组件的访问。

在默认主题，此菜单是深蓝色的。

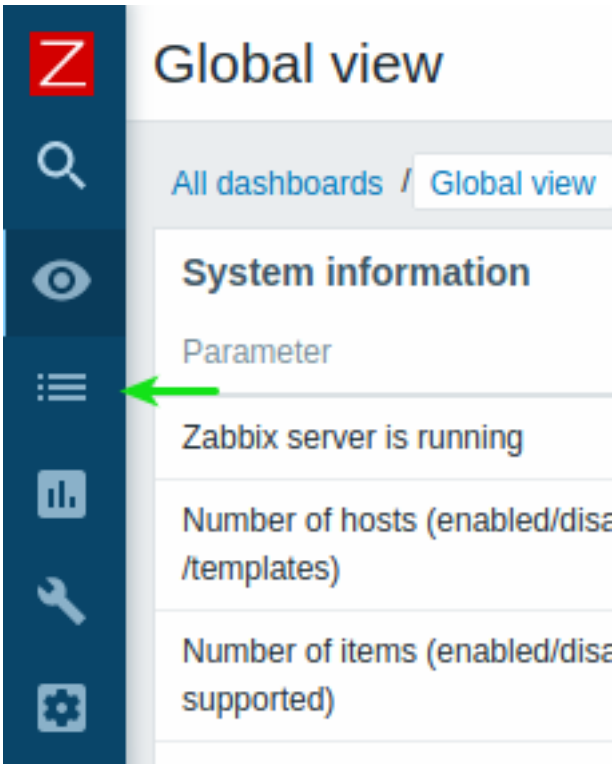


使用菜单

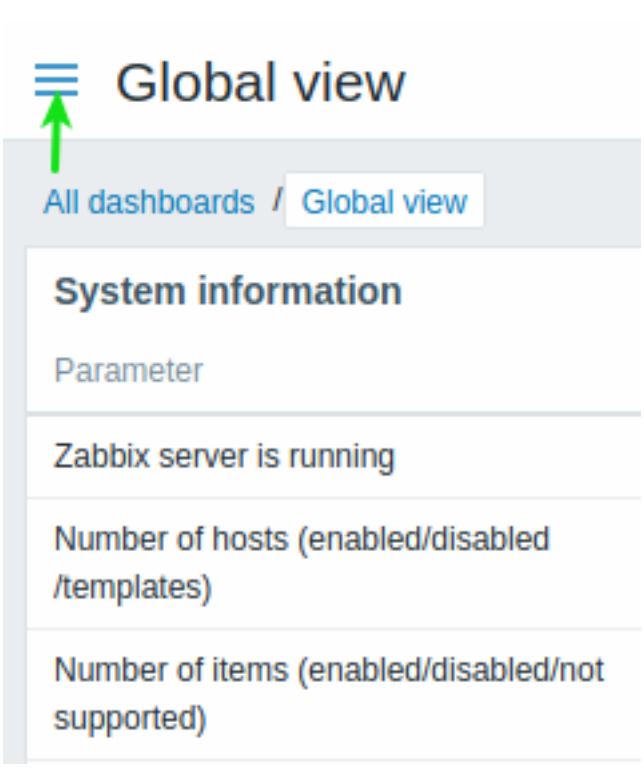
全局搜索 框位于 Zabbix logo 的下方。

此菜单可以折叠或完全隐藏：

- 要折叠，请点击 Zabbix logo 旁边的 
- 要隐藏，请点击 Zabbix logo 旁边的 



只有图标可见的折叠菜单。



隐藏菜单。

折叠菜单

当菜单折叠成图标时，完整的菜单会在光标移动到菜单上方时重新显示。请注意菜单会在页面内容的上方显示；要把页面内容移到右边，你需要点击扩展按钮。如果光标又移动到完整菜单的外部，菜单在两秒后又会折叠。

你也可以按 Tab 键使折叠的菜单重新显示。重复按 Tab 键会允许聚焦到下一个菜单元素。

隐藏菜单

即使菜单完全隐藏，显示完整菜单也只需在汉堡图标点击一次鼠标。请注意菜单会在页面内容的上方重新出现；要把页面内容移到右边，你需要点击显示侧边栏按钮取消隐藏。

2 前端部件

1 监测

概述

监测菜单的作用是显示数据。任何 Zabbix 采集、可视化和操作的信息都会在监测菜单的各个部件中显示。

查看模式按钮

位于右上角的下列按钮在每个组件中都是常见的：

	在 kiosk 模式显示页面。在此模式仅显示页面内容。 要退出 kiosk 模式，移动鼠标直到  退出按钮出现并点击它。你会返回普通模式。

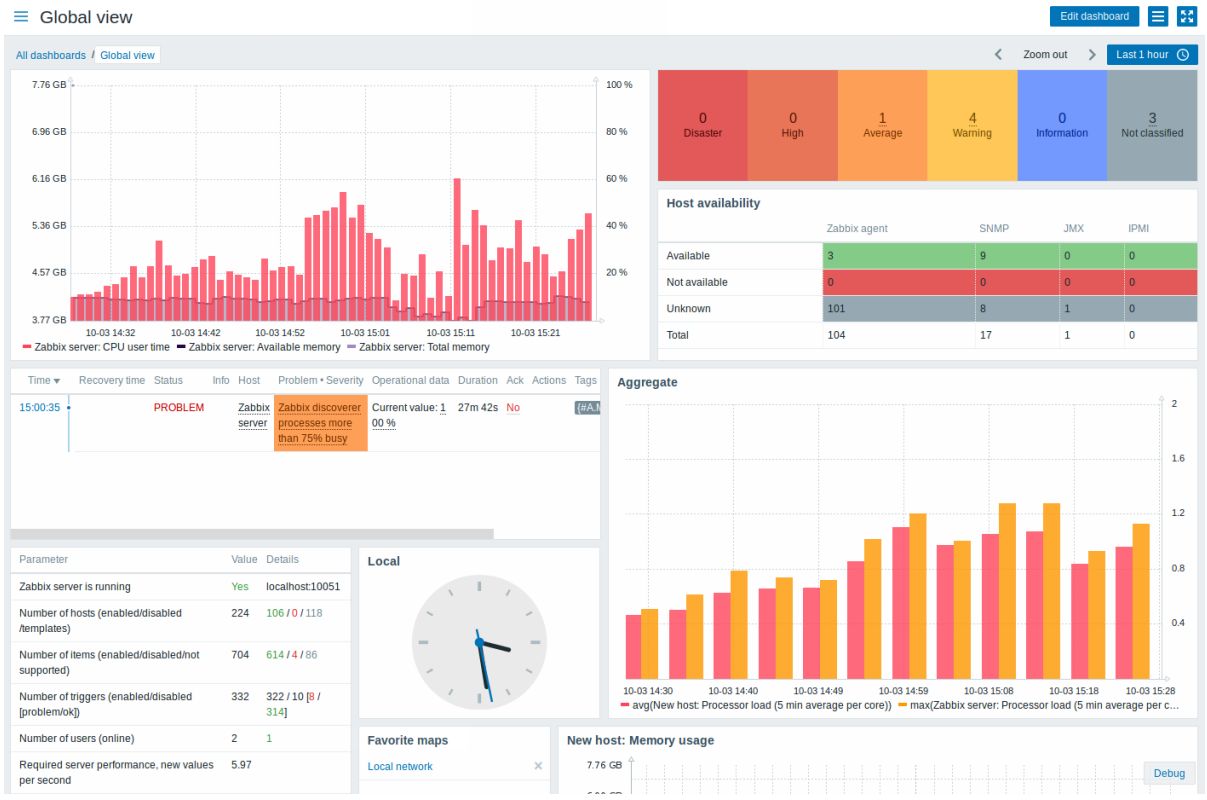
1 仪表盘

概述

监测 → 仪表盘组件是用于显示在仪表盘中所有重要信息的概览。

尽管一次只能显示一个仪表盘，配置多个仪表盘是可行的。每个仪表盘可以包含一个或可滑动显示的多个页面。

一个仪表盘页面可包含多个组件，每个组件是用于显示一种特定类型和来源的信息，可以是摘要，地图，图表或时钟等。



页面和组件在仪表盘编辑模式可以被添加到仪表盘或修改。页面可以在查看模式查看和轮换。

在图表组件中显示的时间区间是被组件上方的时间区间选择器控制的。右侧的时间区间选择器标签显示了当前选择的时间区间。点击 tab 标签允许扩展和折叠时间区间选择器。

请注意当仪表盘在 kiosk 模式显示并且仅显示组件时，可在图表中双击来缩小图表时间区间。

仪表板大小

仪表板的最小宽度为 1200 像素。仪表盘不会收缩到这个宽度以下；如果浏览器窗口小于水平滚动条，则会显示水平滚动条。

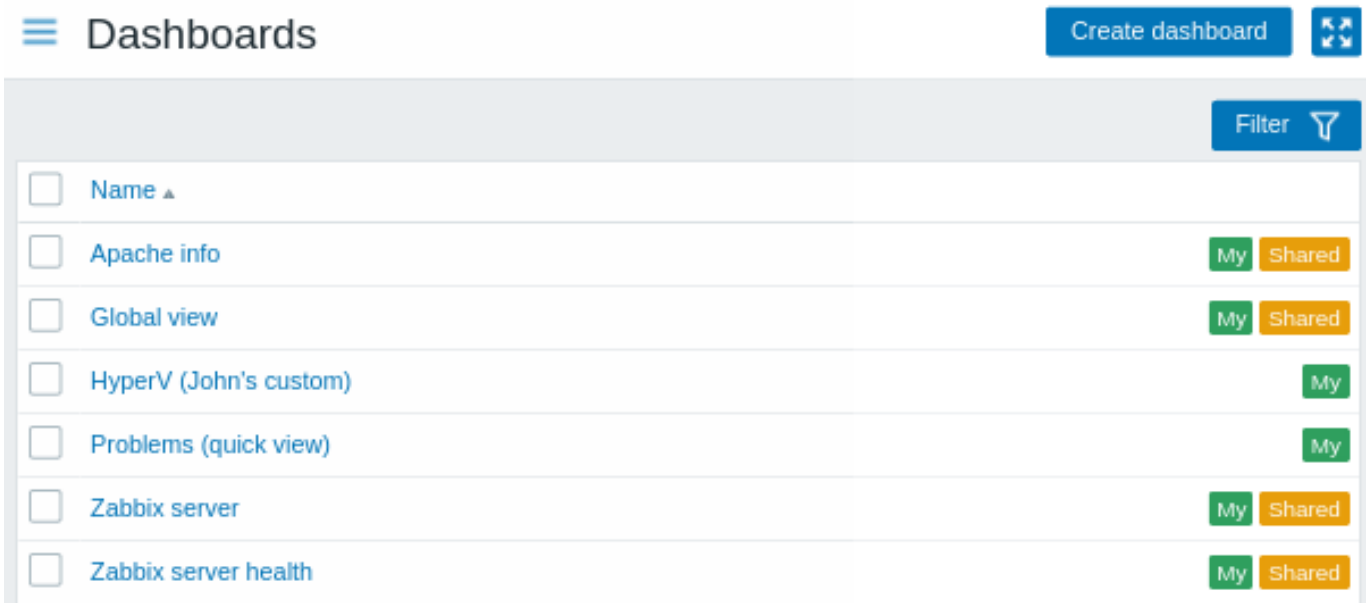
仪表板的最大宽度是浏览器窗口宽度。仪表板小部件水平拉伸以适合窗口。同时，仪表板小部件不能水平拉伸超出窗口限制。

从技术上讲，仪表板由 12 个宽度始终相等的水平列组成，可以动态拉伸/收缩（但总计不小于 1200 像素）。

在垂直方向上，仪表板最多可包含 64 行。每行的高度固定为 70 像素。一个小部件最多可以有 32 行高。

查看仪表盘

可点击组件标题下方的所有仪表盘来查看所有配置的仪表盘。



仪表盘和一个分享标签一起显示：

- 我的 - 表示一个私有仪表盘
- 分享 - 表示一个公有仪表盘或分享给某用户或用户组的私有仪表盘

列表右上方的过滤器允许通过名称和当前用户创建来过滤仪表盘。

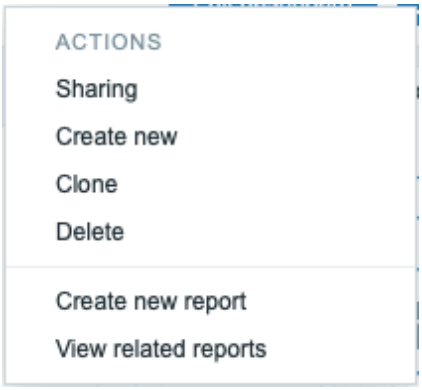
要删除一个或多个仪表盘，勾选对应仪表盘的多选框，并点击列表下方的删除。

查看和编辑仪表盘

要查看一个仪表盘，需在仪表盘列表中点击它的名称。

当查看一个仪表盘时，下列选项是可选的：

Edit dashboard	切换到仪表盘编辑模式。 新建仪表盘和点击组件的 编辑按钮时也会进入编辑模式
	打开动作菜单（查看下方的动作描述）。



分享 - 编辑仪表盘的分享偏好。仪表盘可以被设为公有或私有。公有仪表盘对所有用户可见。私有仪表盘仅对所有者可见。私有仪表盘可被所有者向其他用户和用户组分享。查看分享配置的详细情况，可查看拓扑图configuration 组件。
创建新的 - 创建一个新的仪表盘。
克隆 - 通过复制已有仪表盘的属性来创建一个新的仪表盘。首先你会被提示输入仪表盘参数。然后，新仪表盘将在编辑模式打开，包含原仪表盘的所有组件。
删除 - 删除仪表盘。
创建新的报告 - 打开报告configuration form弹窗。如果该用户无管理定期报告的权限，此选项会被禁用。
查看相关报告 - 打开基于该仪表盘的已有报告的弹窗。如果没有相关报告或该用户无管理定期报告的权限，此选项会被禁用。



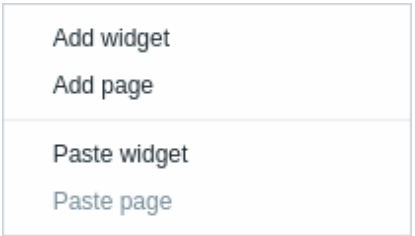
显示仅页面内容 (kiosk 模式).
Kiosk 模式也可通过下列 URL 参数访问：
`/zabbix.php?action=dashboard.view&kiosk=`
要退回到普通模式：
`/zabbix.php?action=dashboard.view&kiosk=`

当 编辑一个仪表盘时，下列选项是可选的：



编辑通用仪表盘参数.

添加一个新组件。
点击箭头按钮会打开动作菜单（查看下方的动作描述）。



添加组件 - 添加一个新组件
添加页面 - 添加一个新页面
粘贴组件 - 粘贴一个被复制的组件。如果没有被复制的组件，此选项是禁用的。一次只有一个实体（组件或页面）可以被复制。
粘贴页面 - 粘贴一个被复制的页面。如果没有被复制的页面，此选项是禁用的。

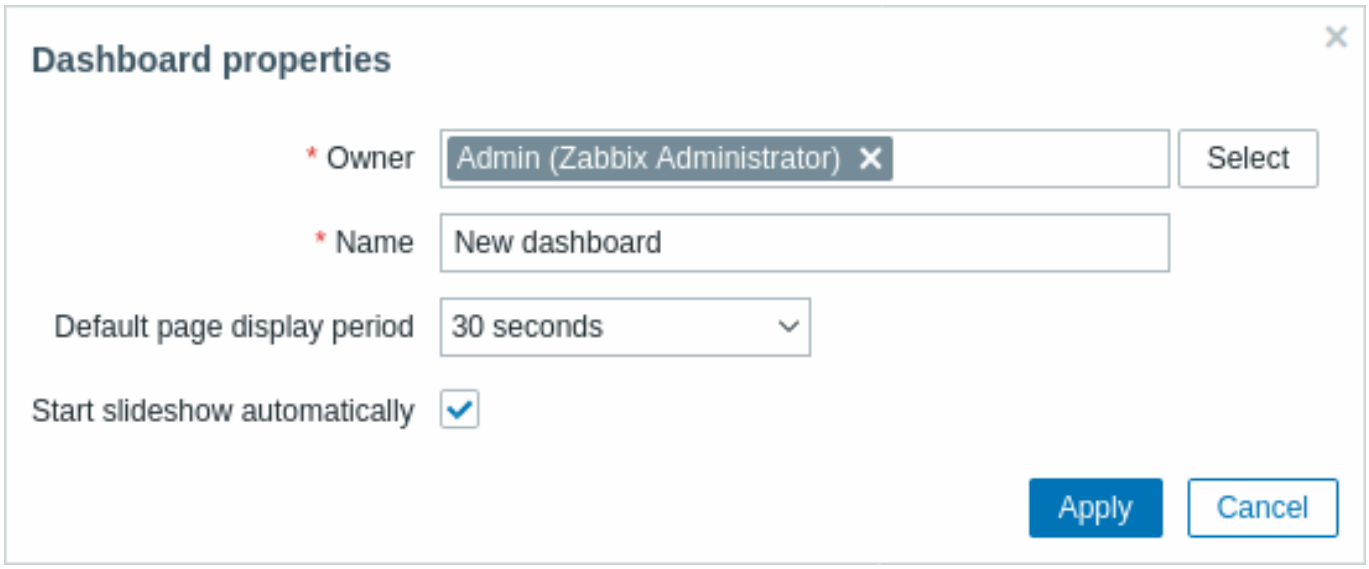
Save changes	保存仪表盘的更改。
Cancel	取消仪表盘的更改。

创建仪表盘

有以下两种方式创建一个新的仪表盘：

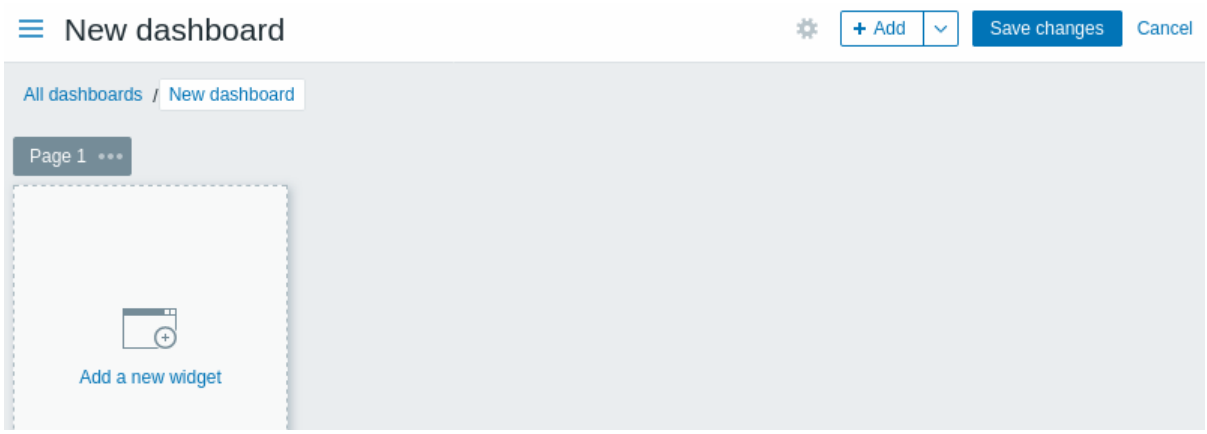
- 在查看所有仪表盘时，点击创建仪表盘
- 在查看单个仪表盘时，点击动作菜单的创建新的

您首先会被要求输入通用的仪表盘参数：



参数	描述
所有者	选择成为仪表盘所有者的系统用户。
名称	输入仪表盘名称。
默认页面显示时长	选择仪表盘页面被轮换到滑动显示下一页前的显示时长。
自动开始滑动显示	当有多个仪表盘页面时，勾选此多选框来自动运行滑动显示。

当你点击应用时，一个空仪表盘会被打开：



您可以给仪表盘添加组件和页面。

点击保存更改按钮来保存仪表盘。如果你点击取消，仪表盘不会被创建。

添加组件

添加组件到仪表盘：

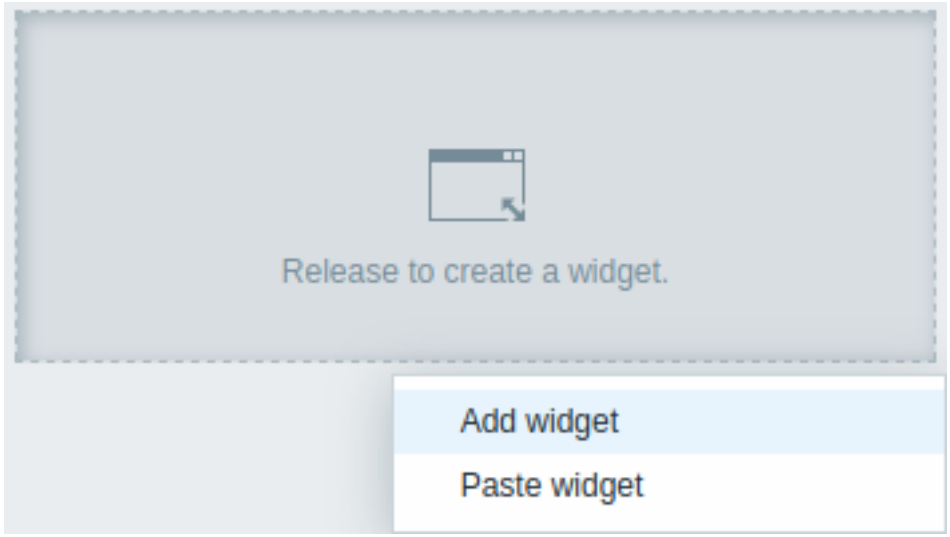
- 点击 

按钮或点击箭头打开动作菜单的添加组件选项。填写组件配置表单。组件会被以默认尺寸创建并被放置在已有组件（如果有）的后方；

或

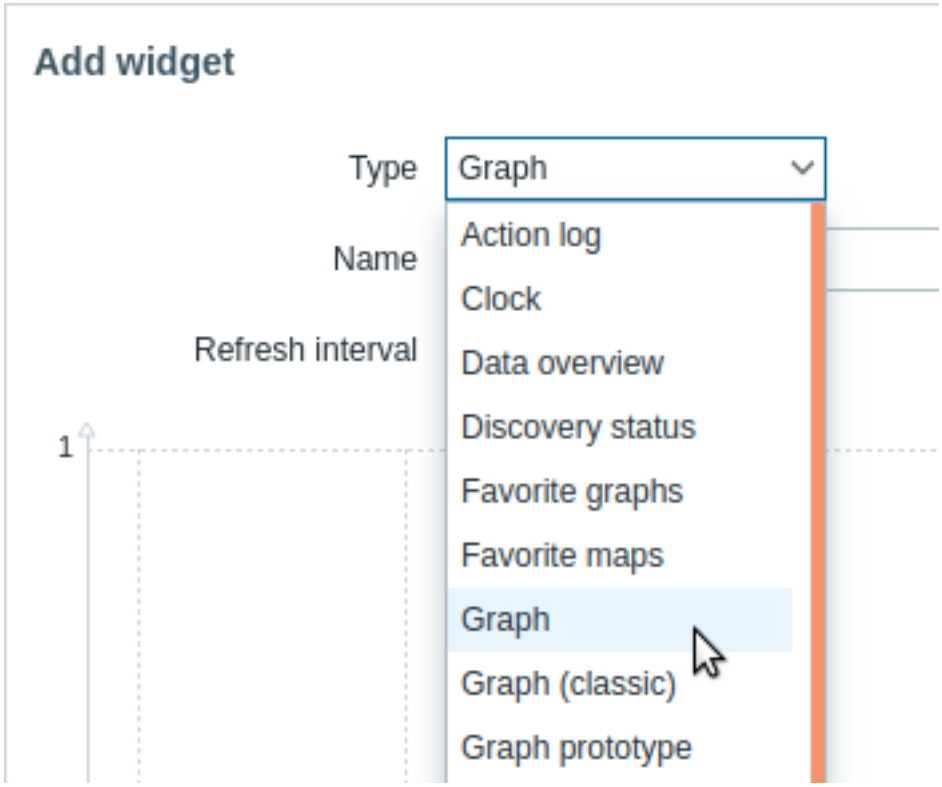
- 移动您的鼠标到需要放置新组件的位置。在仪表盘的任意空插槽上悬停鼠标，占位符会出现。再点击以打开组件配置表单。在填写表单后组件会以默认尺寸被创建，或如果默认尺寸大于可用空间，将使用所有可用空间。可选的，您可以点击并拖动占位符到需要的组件尺寸，并释放，再填写组件配置表单。(请注意当有组件被复制进剪切板时，您会首先被提示选择添加组件和 粘贴组件选项来创建组件。)





在组件配置表单：

- 选择组件类型
- 输入组件参数
- 点击添加



组件

以下组件可被添加到仪表盘：

- 动作记录
- 时钟
- 数据概览
- 自动发现状态
- 收藏的图表
- 收藏的拓扑图
- 地理图
- 图表
- 图表（传统）
- 图表原型
- 主机可用性
- 监控项值
- 拓扑图
- 拓扑图导航树
- 纯文本
- 问题主机
- 问题
- 系统信息
- 按严重性的问题
- 前排主机
- 触发器概览
- URL
- Web 监控

在仪表盘编辑模式，可修改组件的大小，以及通过点击组件标题并拖动到新位置来移动组件。并且，您可以点击组件右上角的以下按钮来：

- - 编辑组件;
- - 访问组件菜单

点击仪表盘的保存修改来永久保存对组件的修改。

复制/粘贴组件

仪表盘组件可被复制和粘贴，允许通过现有组件的属性创建一个新的组件。可以在同一个仪表盘进行复制，或者在不同标签页打开的多个仪表盘之间进行复制。

可使用组件菜单来复制。要粘贴组件：

- 在编辑仪表盘时，点击添加按钮旁的箭头并选择粘贴组件选项
- 在选择仪表盘的某区域来添加组件时，使用粘贴组件选项（需要先复制一个组件，粘贴选项才会可用）

使用组件菜单中的粘贴选项，一个复制的组件可被用于粘贴覆盖一个现有的组件。

创建一个滑动显示

一个滑动显示会在仪表盘包含两个及以上页面（查看添加页面）且符合以下条件时自动运行：

- 仪表盘属性中的自动开始滑动显示选项被选中
- 仪表盘 URL 包含 `slideshow=1` 参数

页面根据仪表盘和单个页面属性中的间隔轮换。点击：

- 停止滑动显示 - 停止滑动显示
- 开始滑动显示 - 开始滑动显示



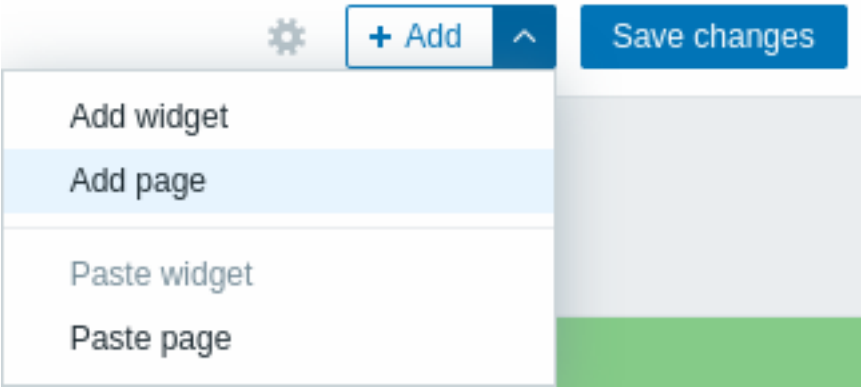
滑动显示相关的控制同样在kiosk 模式 中可用（当只显示页面内容是）：

-  - 停止滑动显示
-  - 开始滑动显示
-  - 返回上一页
-  - 进入下一页

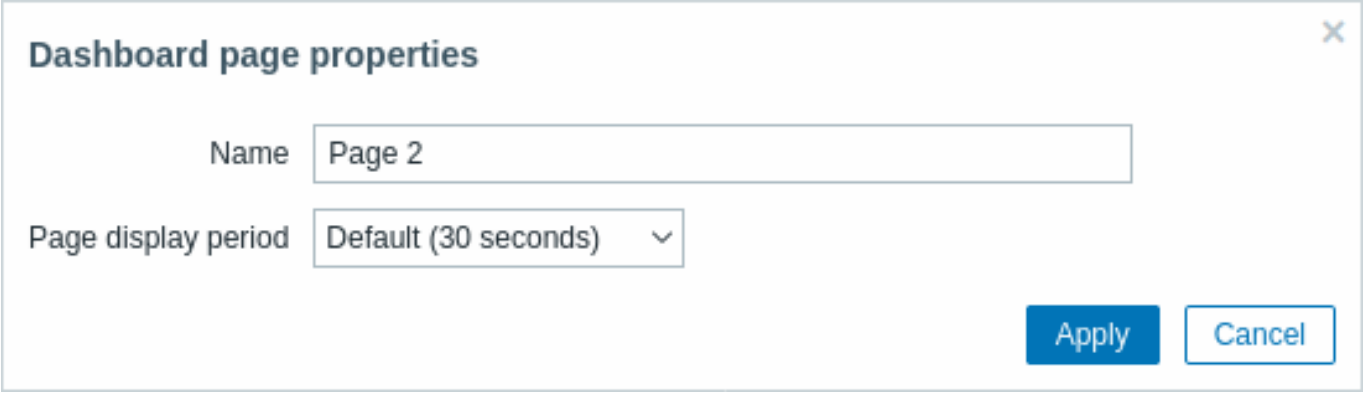
添加页面

要添加新页面到仪表盘：

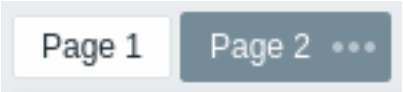
- 确保仪表盘在编辑模式
- 点击添加按钮旁边的箭头，再选择添加页面选项



- 填充通用页面参数并点击应用。如果您将名称留空，页面会以页面 `N` 为名称添加，'N' 是递增的页数。页面显示时间范围允许自定义页面在滑动显示中显示的时间。



新页面会被添加，并显示在一个新标签页（页面 *2*）中。



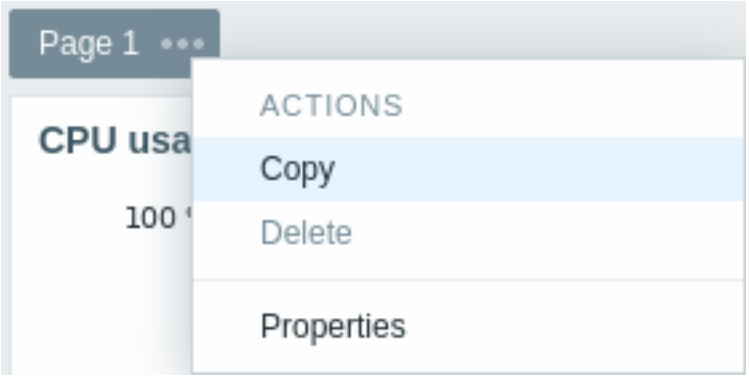
这些页面可在页面标签上拖动进行重新排序。重新排序维持原页面名称。一直可通过点击标签页进入页面。

当新页面被添加时，它是空的，您可以按之前的描述添加组件。

复制/粘贴页面

仪表盘页面可被复制和粘贴，可通过现有页面的属性创建一个新页面。他们可粘贴到同一个仪表盘或不同仪表盘中。

要粘贴一个现有页面到仪表盘，先复制，使用页面菜单：

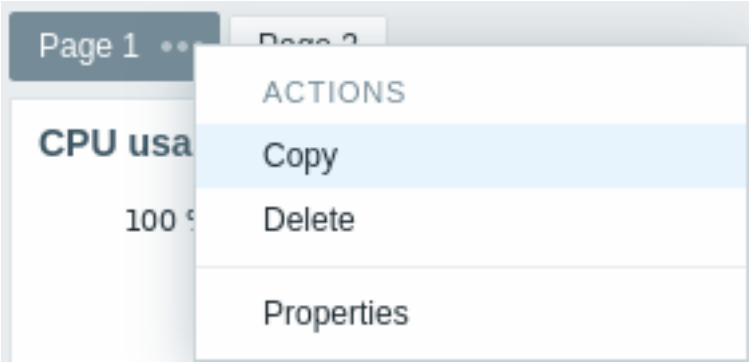


要粘贴已复制的页面：

- 确保仪表盘在编辑模式
- 点击添加按钮旁的箭头，并选择粘贴页面选项。

页面菜单

页面菜单可通过点击页面名称旁的三个点  来打开：

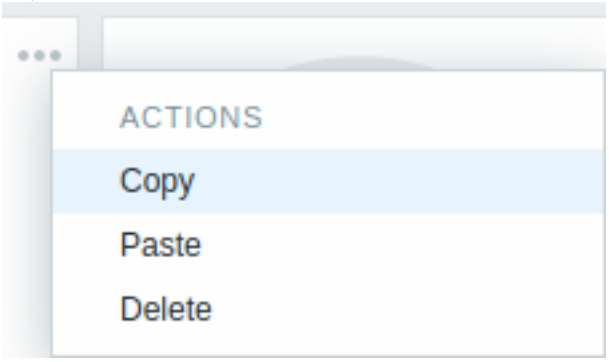
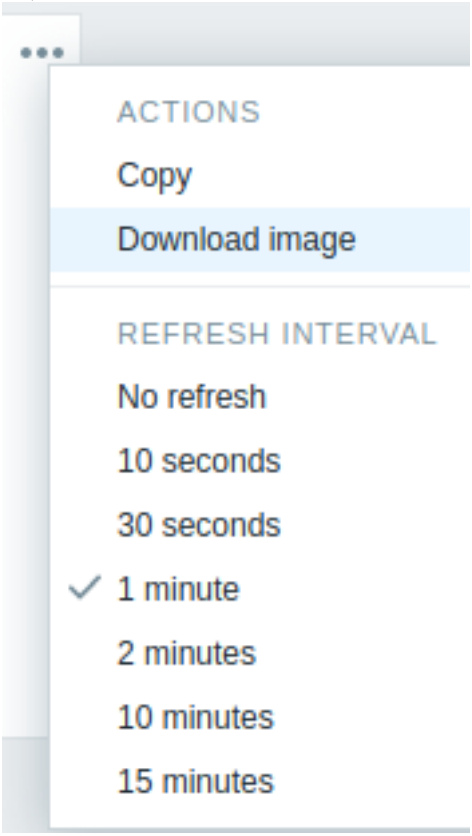


它包含以下选项：

- 复制 - 复制页面
- 删除 - 删除页面（页面只能在仪表盘编辑模式删除）
- 属性 - 自定义页面参数（名称和页面在滑动显示中的显示时间范围）

组件菜单

组件菜单在仪表盘编辑或查看模式包括不同的选项：

组件菜单	选项
在仪表盘编辑模式： 	复制 - 复制组件 粘贴 - 粘贴一个复制的组件来覆盖当前组件 当没有复制的组件时，此选项被禁用。 删除 - 删除组件
在仪表盘查看模式： 	复制 - 复制组件 下载图片 - 将组件下载为 PNG 图片 (只对图表/传统图表组件可用) 刷新间隔 - 选择刷新间隔 组件内容

动态组件

当配置 以下组件时：

- 图表（简单或自定义）
- 监控项值
- 纯文本
- URL

有一个名为动态监控项的额外的选项。您可以勾选此框将组件设为动态。例如，能够根据选择的主机显示不同的内容。

现在，当保存仪表盘时，您会发现一个新的主机选择字段出现在仪表盘上方（选择按钮允许在弹窗选择主机群组）：



此时您得到了可以根据选择的主机显示内容的一个组件。这样的好处是您不需要仅仅因为想要查看多个主机的相同图表，而去创建额外的组件。

仪表盘的权限

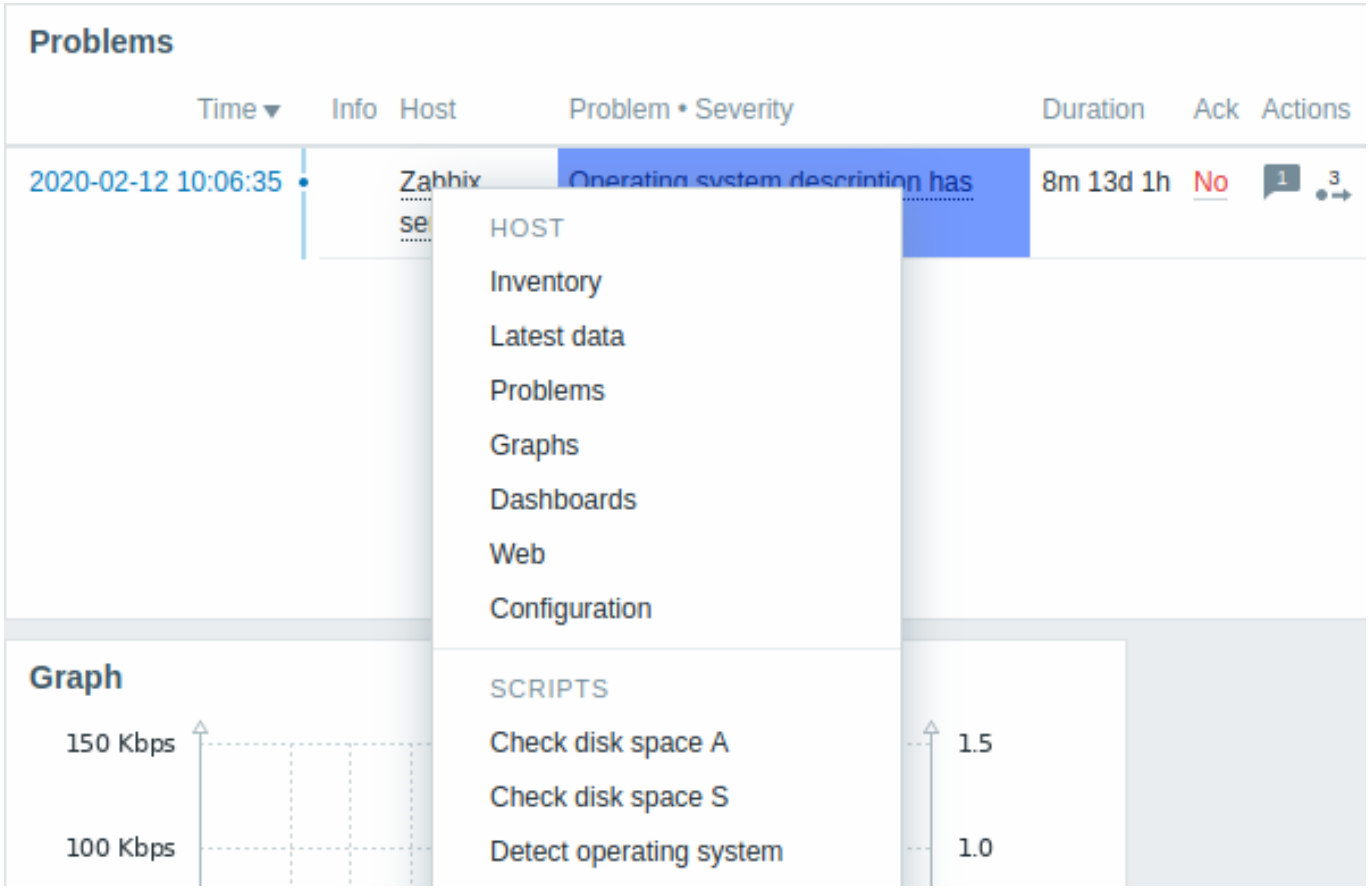
一般用户和管理员类型的用户对仪表盘的权限有以下限制：

- 他们可以查看和克隆仪表盘如果他们有只读以上的权限；

- 他们可以查看和删除仪表盘如果他们有读/写权限；
- 他们不可以修改仪表盘所有者。

主机菜单

在问题组件中点击主机会显示主机菜单。它包括到主机资产记录、最新数据、问题、图表、仪表盘、web 场景和配置的链接。请注意主机配置只对管理员和超级管理员用户可用。



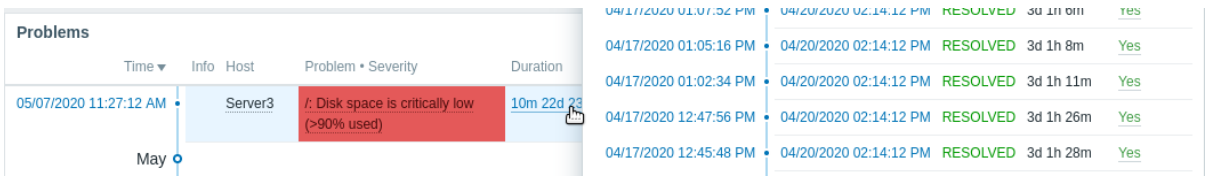
全局脚本 也可在主机菜单中运行。这些脚本的作用域需要被定义为' 手动主机动作' 才能在主机菜单中可用。

主机菜单可在以下前端部件中点击主机来访问：

- 监测 → 问题
- 监测 → 问题 → Event details
- 监测 → 主机
- 监测 → 主机 → Web 监测
- 监测 → 最新数据
- 监测 → 地理图
- 报表 → 触发器 top 100

问题事件弹窗

问题事件弹窗包括此触发器的问题事件列表和问题描述和可点击的 URL（如果定义了）。



要显示问题事件弹窗：

- 滚动鼠标到问题时间的上方（问题组件的时间栏）。当您把鼠标从时间移开时，弹窗会消失。
- 点击问题时间（问题组件的时间栏）。弹窗只会在您再次点击问题时间时消失。

Attention:
触发器描述中的 {ITEM.VALUE} 和 {ITEM.LASTVALUE} 的解析值被截取为 20 个字符。查看完整数据，您可以使用通过宏使用宏函数，例如 `{{ITEM.VALUE}.regsub("(.*)", \1)}`, `{{ITEM.LASTVALUE}.regsub("(.*)", \1)}`。

仪表盘组件

概述

此章节提供了仪表盘组件的通用参数细节。

通用参数

以下参数对每个组件都是通用的：

名称	输入组件名称。
刷新间隔	配置默认刷新间隔。组件的默认刷新间隔从不刷新到 *15* 分钟取决于组件的类型。示例：URL 组件是不刷新，动作记录组件是 *1* 分钟，时钟组件是 *15* 分钟。
显示组件头部	勾选多选框来永久显示组件头部。 当取消勾选时，无论在查看还是编辑模式，为了节省空间，头部会被隐藏，只有在鼠标位于组件上方时向上滑动能重新显示头部。 当拖动组件到新位置时，头部也会部分可见。

组件的刷新间隔可为所有相关用户设为一个默认值，也可为每个用户单独设置其刷新间隔：

- 要为所有相关用户设为一个默认值，切换到编辑模式（点击编辑仪表盘按钮，找到正确的组件，点击编辑按钮打开组件的编辑表单），之后在下拉菜单中选择需要的刷新间隔。
- 要为每个用户单独设置其刷新间隔，在查看模式中，点击一个组件的 按钮。

一个用户设置的单独的刷新间隔优先级高于组件设置，并且一旦设置了单独的刷新间隔，即使组件设置被修改，它也会一直保持不变。

查看每个组件的特殊参数，可进入单独的组件页面：

- 动作记录
- 时钟
- 自动发现状态
- 收藏的图表
- 收藏的拓扑图
- 地理图
- 图表
- 图表（传统）
- 图表原型
- 主机可用性
- 监控项值
- 拓扑图
- 拓扑图导航树
- 纯文本
- 问题主机
- 问题
- SLA 报表
- 系统信息
- 按严重性的问题
- 前排主机
- 触发器概览
- URL
- Web 监控

弃用的组件：

- 数据概览

Attention:
弃用的组件将在后续的大版本被移除。

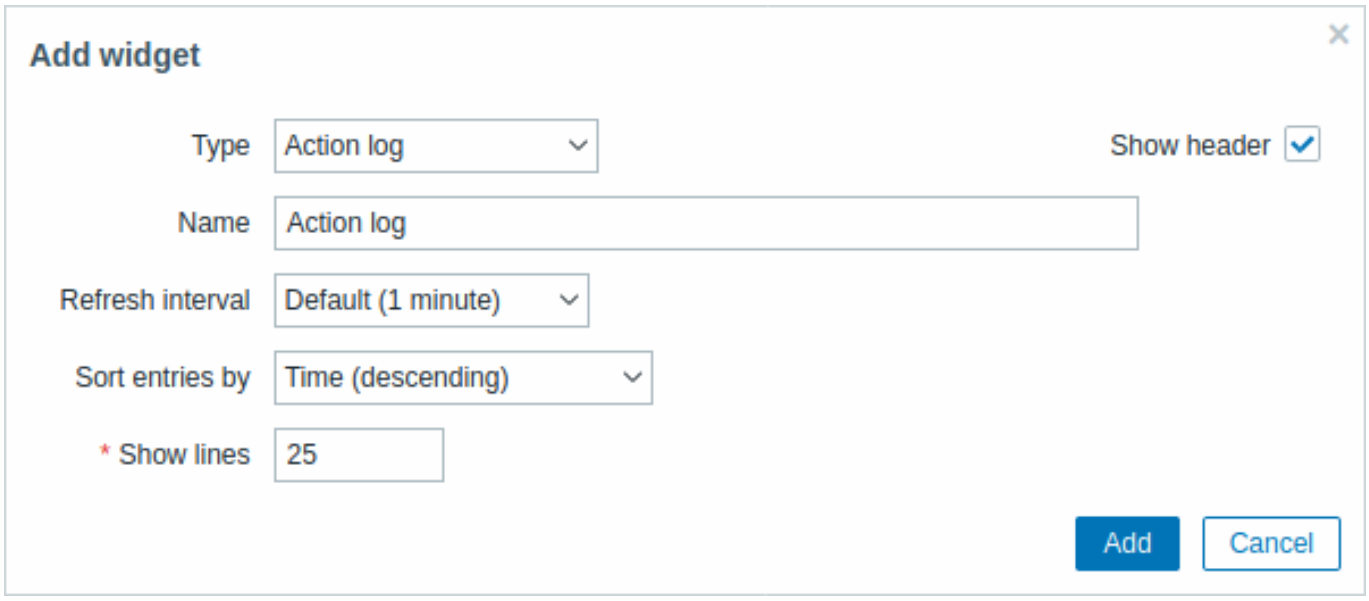
1 动作记录

概述

在动作记录组件，你可以显示动作操作（提示，远程命令）的细节。它复制了管理 → 审计的信息。

配置

选择动作记录为类型来配置：



除了所有组件的通用 参数外，您也可以设置以下选项：

排序方式	排序方式: 时间（降序或升序） 类型（降序或升序） 状态（降序或升序） 接收人（降序或升序）.
显示行	设置组件中可以显示多少行动作记录。

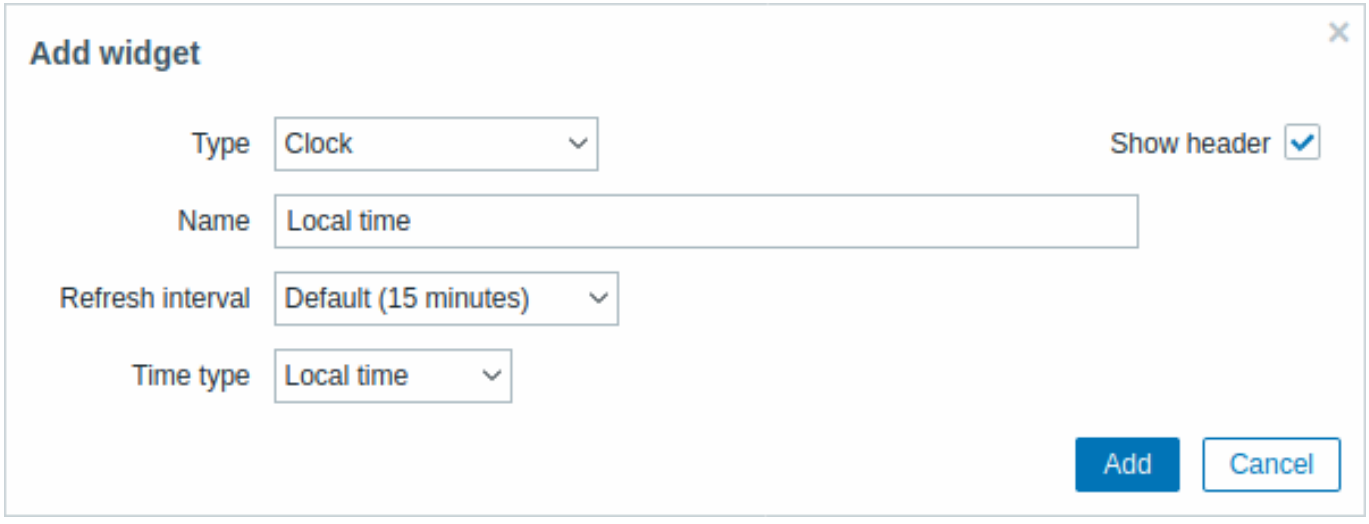
2 时钟

概述

在时钟组件，你可以显示本地、服务器、或特定的主机时钟。

配置

选择时钟为类型来配置：



除了所有组件的通用 参数外，您也可以设置以下选项：

时间类型	选择本地、服务器、或特定主机时间。 服务器时间会与全局设置的时区或 Zabbix 用户的时区相同。
监控项	选择显示的时间的监控项。要显示主机时间，使用 `system.localtime[local]` 监控项。此监控项必须在主机上存在。 此字段仅在主机时间被选择时可用。

3 数据概览

Attention:
此组件已被弃用，将在后续的大版本被移除。

概述

在数据概览组件，您可以显示一组主机的最新数据。

问题监控项的颜色是基于问题严重程度的，可在问题更新页面调整。

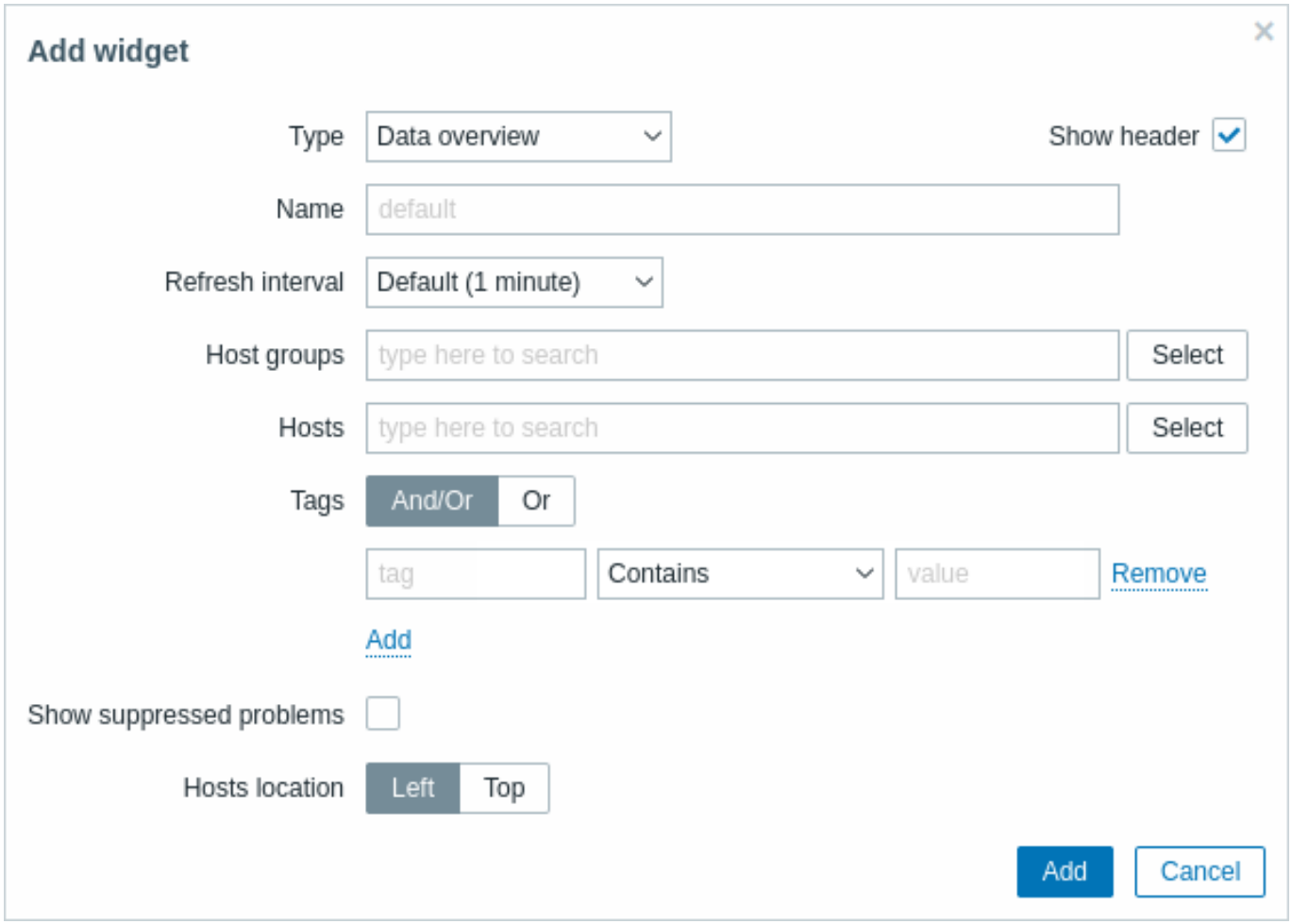
默认只会显示 24 小时内的值。此限制是为了减少大量载入最新数据的初始化时间而推行的。此限制通过管理 → 常用 → 界面设置 中的历史最长显示时间选项进行配置。

点击一条数据可链接到预定义的图表或最新值。

请注意默认显示 50 条记录（在管理 → 常用 → 界面设置 中配置），使用表格位中最多可展示的的组件数量选项。如果存在比配置的条数更多的记录，表格下放会显示一条消息，请求提供更具体的过滤条件。不会进行分页。

配置

选择数据概览为类型来配置：



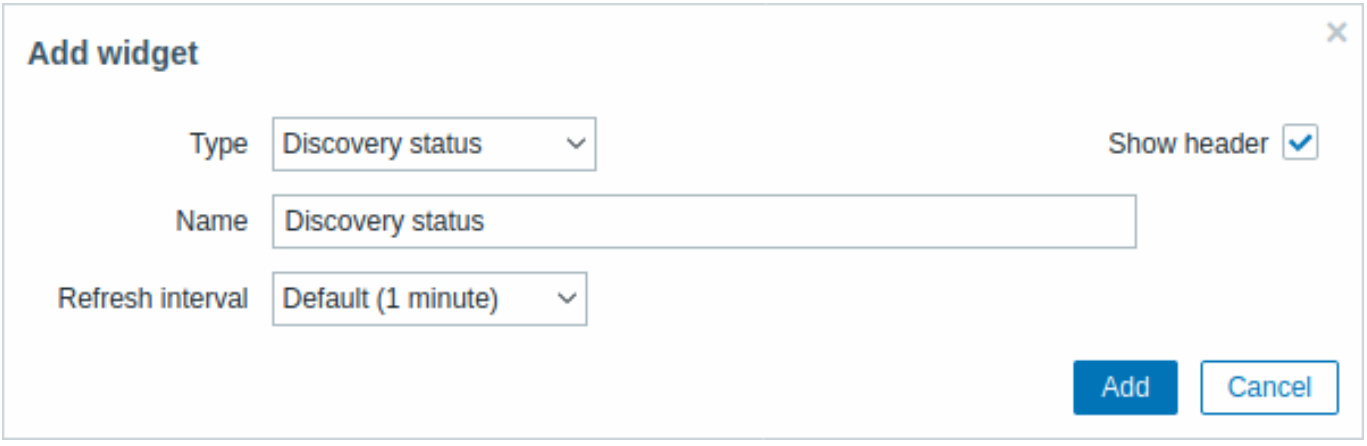
除了所有组件的通用 参数外，您也可以设置以下选项：

主机群组	选择主机群组。此字段是自动完成的，所以输入群组的名称后，会出现一个包含匹配群组的下拉菜单。向下滚动来选择。点击'x'来移除已选择的群组。
主机	选择主机。此字段是自动完成的，所以输入主机的名称后，会出现一个包含匹配主机的下拉菜单。向下滚动来选择。点击'x' 来移除已选择的主机。
标签	指定标签来限制组件中显示的监控项数据量。可选择包含或不包含特定标签和标签值。可设置多个条件。标签名称匹配一直是大小写敏感的。 每种条件下有多种运算符： 存在 - 包含特定标签名称 等于 - 包含特定标签名称和值（大小写敏感） 包含 - 特定标签名称的值包含输入的字符串（部分匹配，大小写敏感）， 不存在 - 不包含特定标签名称 不等于 - 不包含特定标签名称和值（大小写敏感） 不包含 - 特定标签名称的值不包含输入的字符串（部分匹配，大小写敏感） 这些条件有两种计算类型： 与/或 - 必须满足所有条件，相同标签名称的条件将会按或条件进行分组 或 - 只需满足一个条件
显示抑制的问题	勾选多选框来显示因主机维护不显示的问题。
主机位置	选择主机位置 - 左侧或顶部。

4 自动发现状态

概述

此组件显示活动的网络发现规则的状态汇总。



所有配置参数都是组件的通用 参数

5 常用图表

概述

该组件包含了最常用的图表的快捷方式，按字母排序。

当你查看一个图表时，点击它的  添加到收藏夹按钮。

所有的配置参数对所有组件都是通用 的。

6 常用拓扑图

概述

该组件包含了最常用的拓扑图的快捷方式，按字母排序。

当你查看一个拓扑图时，点击它的  添加到收藏夹按钮。

所有的配置参数对所有组件都是通用 的。

7 Geomap

概述

Geomap 使用基于 JavaScript 的开源交互式地图库 Leaflet 将主机位置标记在地图上。

Note:
Zabbix 内置了多种地图供应商并支持自定义地图供应商，以及自建地图（可在 *Administration → General → Geographical maps* 菜单部分配置）。

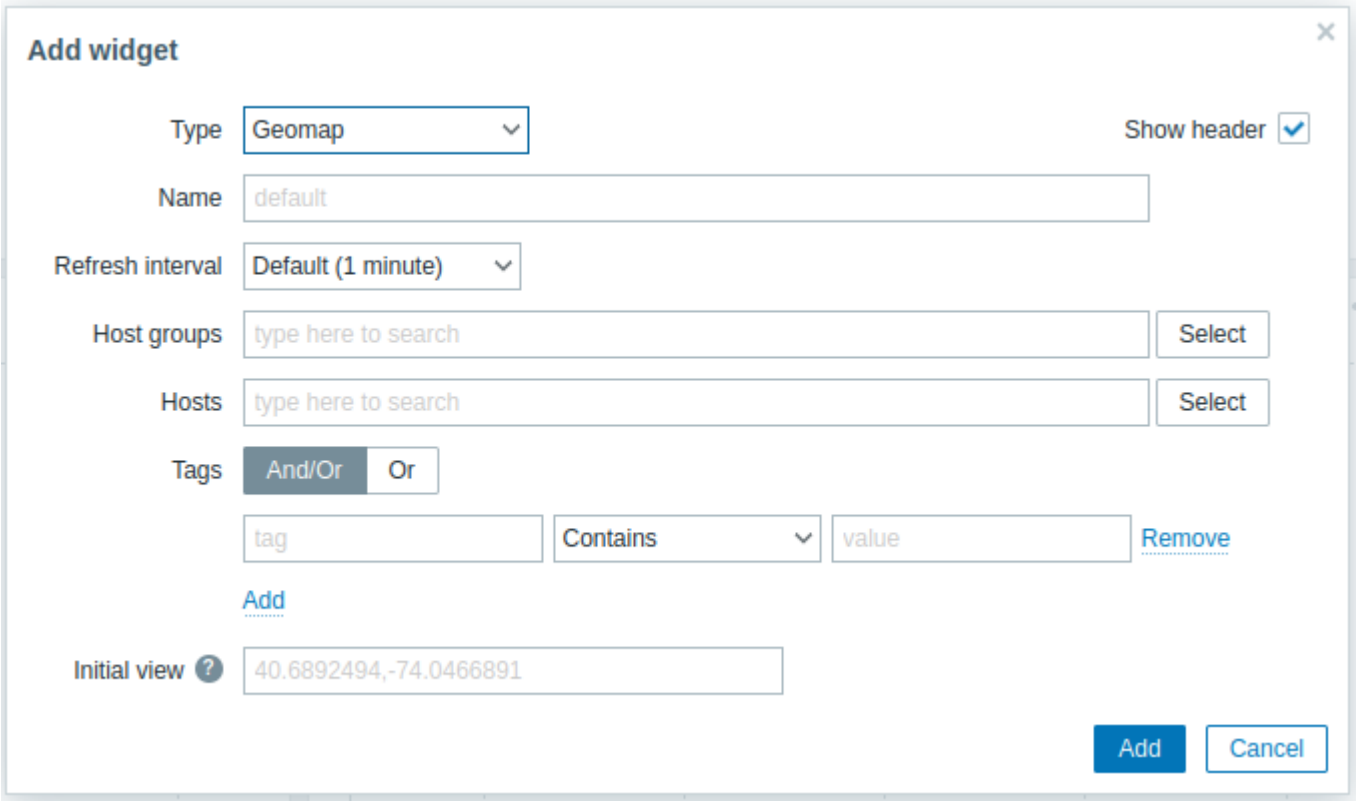
默认情况下，组件会显示所有的已启用且定义了地理坐标的主机。可以在组件参数中配置主机过滤参数。

主机坐标的有效值：

- 纬度：从-90 到 90（整数或浮点数皆可）
- 经度：从-180 到 180（整数或浮点数皆可）

配置

要添加组件，请选择 *Geomap* 作为类型。



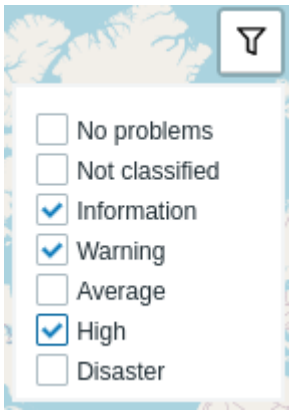
通用 的配置参数对所有组件都是可用的，您还可以设置下列专属配置项：

主机组	选择要在地图上显示的主机组。这个字段支持自动补全，输入一个组的名称在下拉框选择你需要的主机组名称。点击'x' 来移除已选择的主机组。 如果在 主机组和 主机字段中均未选择任何内容，所有具有有效坐标的主机都会被显示出来。

主机	选择要在地图上显示的主机。这个字段支持自动补全，输入一个组的名称在下拉框选择你需要的主机名称。点击'x' 来移除已选择的主机。 如果在 主机组和 主机两个字段都没有选择内容，所有具有有效坐标的主机都会被显示出来。
标签	指定标签以控制组件中显示的主机数量。支持包含以及排除特定的标签和标签值。支持多条件限定。标签名称匹配区分大小写。 可用的判断运算符： **Exists** - 包含指定的标签名称 **Equals** - 包含指定的标记名称和值（区分大小写） **Contains** - 指定的标签值包含输入的字符串（支持通配，不区分大小写） **Does not exist** - 排除指定的标签名称 **Does not equal** - 排除指定的标签名称和值（区分大小写） **Does not contain** - 不包括标签值包含输入字符串的指定标签名称（支持通配，不区分大小写） 有两种条件的计算类型： **And/Or** - 必须满足所有条件，具有相同标签名称的条件将被 Or 条件分组 **Or** - 任一条件满足即可
初始视图	逗号分隔的中心坐标和一个可选的缩放级别，以便在小组件最初加载时显示，格式为 `<latitude>,<longitude>,<zoom>` 如果指定了初始缩放，Geomap 小组件将以给定的缩放级别加载。否则，初始缩放为最大缩放 的一半。 如果设置了默认视图，初始视图将被忽略（见下文）。 示例： => 40.6892494,-74.0466891,14 => 40.6892494,-122.0466891

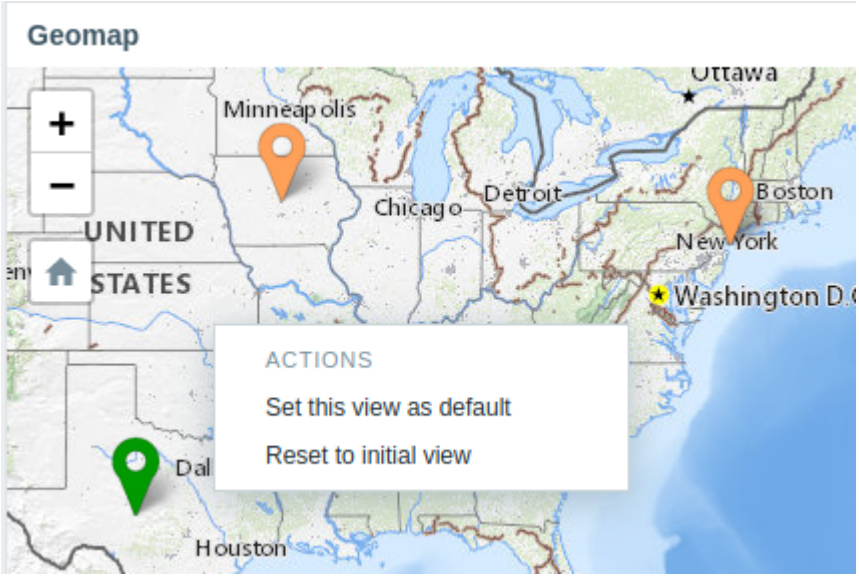
地图上的主机标记会显示最严重问题的颜色，如果主机没有问题则为绿色。点击一个主机标记可以查看该主机的可见名称以及按严重程度汇总未解决的问题数量。点击可见名称将打开主机菜单。

地图上显示的主机可以按问题的严重程度进行过滤。按组件右上角的过滤器图标，标记所需的严重程度。



通过组件左上角的加号和减号按钮或使用鼠标滚轮或触摸板来放大和缩小地图。如果需要把当前视图作为默认状态，可以在地图内任意位置点击右键，选择 *Set this view as default* 把当前视图设置为默认状态。此设置会覆盖当前用户的默认视图设定。通过在在地图内任意位置点击右键，选择 *Reset the initial view*。功能恢复默认视图。

当设置了初始视图或默认视图时，您可通过点击左边的主页图标返回到这个视图。



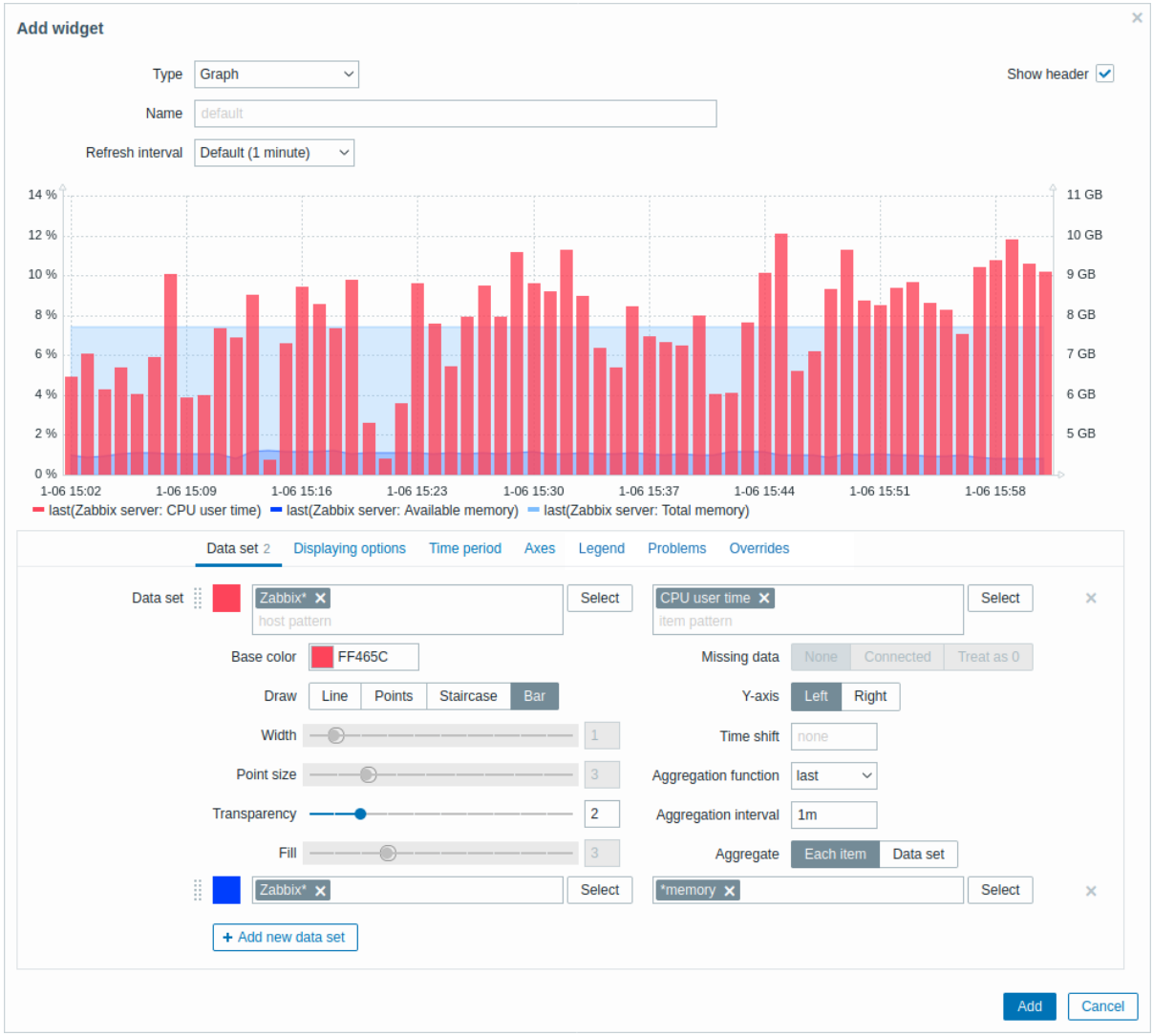
8 图表

概述

图表组件提供了一种现代的、多功能的方式，借助矢量图像绘制技术将 Zabbix 收集的数据可视化。这个图表组件从 Zabbix 4.0 开始被支持。请注意，Zabbix 4.0 之前支持的图表部件仍然可以作为图表（经典）使用。

配置

选择 图表作为类型：



通过数据集选项卡添加数据以及可视化效果：

数据集		选择要在图表显示的主机和监控项，你也可以用主机和项目标签，支持通配符（例如，* 将返回匹配零个或更多字符的结果）。要使用通配只需要输入特定文本再按下回车。当你输入的时候，所有匹配的主机会在下拉框显示。 图表最多支持显示 50 个监控项。 主机参数和监控项参数必填。 通配符不可以被添加，因为它会被解析，因此如果有其他匹配项（例如 item2、item3）也会显示出来，无法单独添加名为 “item*” 的项目。
	基色	调整基色，可以通过颜色选择器或者手动输入。基色用于为监控项显示指定的颜色。基色参数必填。
	绘制	选择绘制的方式。可以选择线（默认设置）、点、梯图和条状。 请注意，如果线/梯图中只有一个数据点，那么不管绘制类型如何，它都会被绘制成一个点。点大小是根据线宽计算的，点大小不能小于 3 像素，即使线宽更小也是如此。
	线宽	设置线宽。这个选项在绘制线和梯图类型时可用。
	点大小	设置点的大小。这个选项在选择绘制点类型时可用。

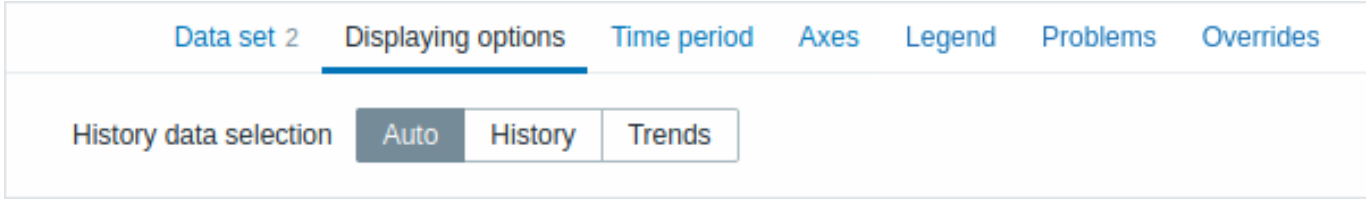
	透明度	设置透明度。
	填充率	设置填充率。这个选项在选择绘制线或梯图类型时可用。
	缺失数据	选择显示缺失数据的方式： **None** - 留空 **Connected** - 将缺失数据两侧连接 **Treat as 0** - 缺失数据按照数值 0 绘制图形 不适用于 点和 条状绘制类型。
	Y 轴	选择图表上 Y 轴显示内容。
	间隔	如果需要，可以设置时间间隔。支持输入时间单位。
	聚合函数	指定使用的聚合函数： **min** - 显示最小值 **max** - 显示最大值 **avg** -显示平均值 **sum** - 显示值的总和 **count** - 显示数值的个数 **first** - 显示第一个值 ** last** - 显示最后一个值 **none** - 显示所有值（无聚合） 聚合允许显示所选区间（5 分钟、1 小时、1 天）的聚合值，而不是所有值。另请参阅：聚合图形。 从 Zabbix 4.4 开始支持这个选项。
	聚合间隔	指定聚合的时间间隔。您可以在此字段中使用时间单位。输入纯数字视作秒。 从 Zabbix 4.4 开始支持这个选项。
	聚合	指定是否聚合： **Each item** - 数据集中的每个监控项将被汇总并单独显示。 **Data set** - 所有的数据集监控项将被汇总并显示为一个值。 从 Zabbix 4.4 开始支持这个选项。

现有的数据集会显示在一个列表中。你可以：

- 

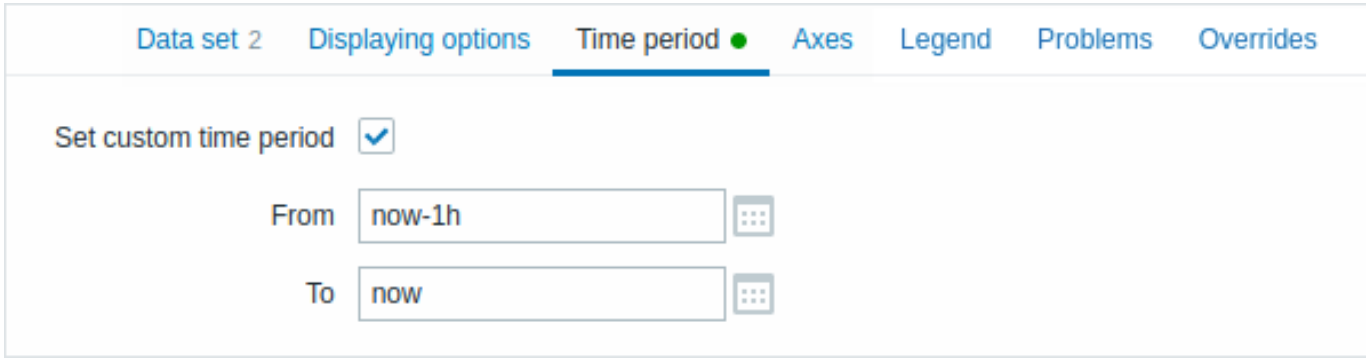
- 点击这个按钮来添加一个新的数据集
- - 点击颜色图标，展开/折叠数据集细节设置
-  - 点击移动图标，将数据集拖到列表中的新位置。

显示选项选项卡允许定义历史数据选择：



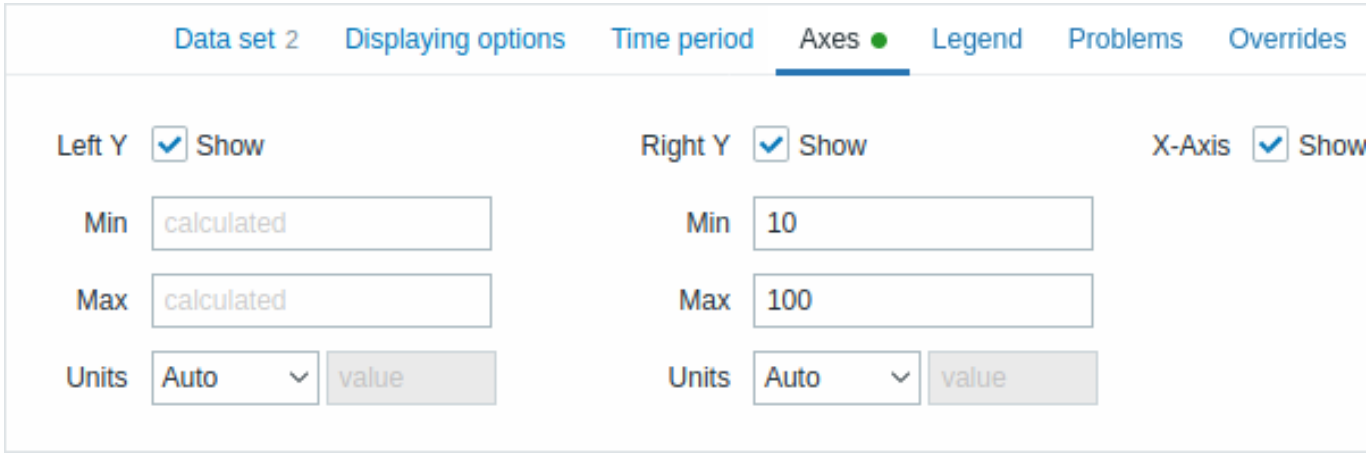
历史数据选择	设置图表数据来源： **Auto** - 数据的来源是根据经典的图表算法 (默认) **History** - 来自历史的数据 **Trends** - 来自趋势的数据

时间段选项卡允许设置一个自定义的时间段：



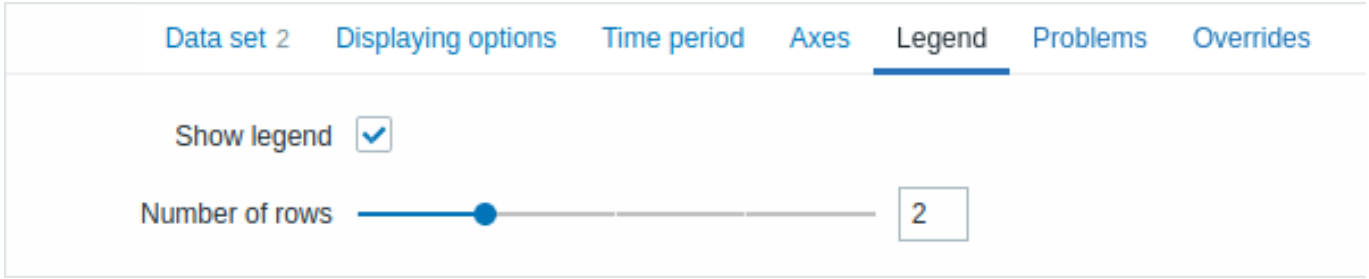
设置自定义时间段	勾选此复选框以设置图表的自定义时间段（默认未勾选）。
From	设置图表的自定义时间段的开始时间。
To	设置图表的自定义时间段的结束时间。

轴选项卡允许自定义轴的显示方式：



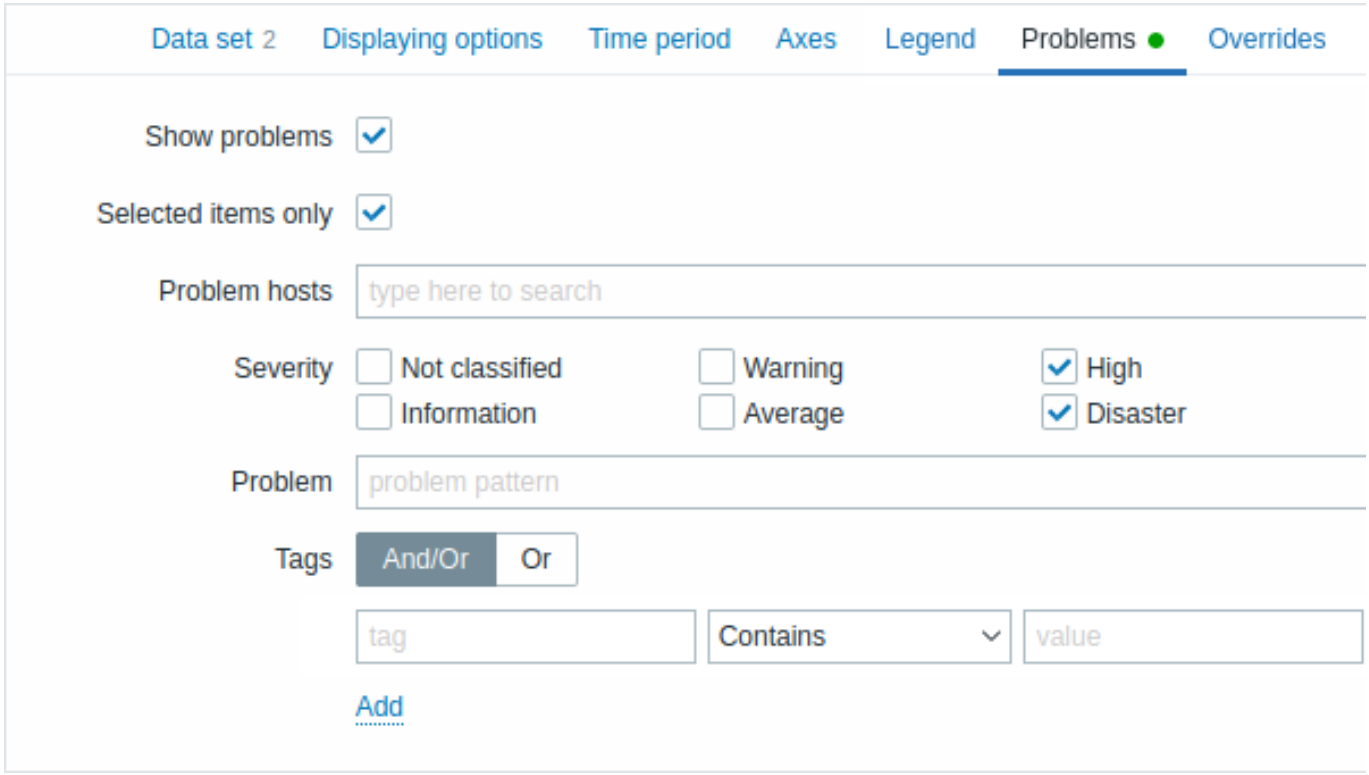
左 Y	勾选复选框使左 Y 轴可见。如果在数据集或覆盖选项卡中没有选择，该复选框可能会被禁用。
右 Y	勾选复选框，使右 Y 轴可见。如果在数据集或覆盖选项卡中没有选择，该复选框可能被禁用。
X-Axis	取消该复选框可以隐藏 X 轴（默认为勾选）。
Min	设置相应轴的最小值。指定 Y 轴的可视范围最小值。
Max	设置对应轴的最大值。
Units	从下拉菜单中选择图轴数值的单位。如果选择了自动选项，轴值将使用相应轴的第一个监控项的单位来显示。静态选项允许你指定相应轴的自定义名称。如果选择了静态选项，并且字段留空，那么相应轴的名称仅显示数值。

图例选项卡允许自定义图形图例：



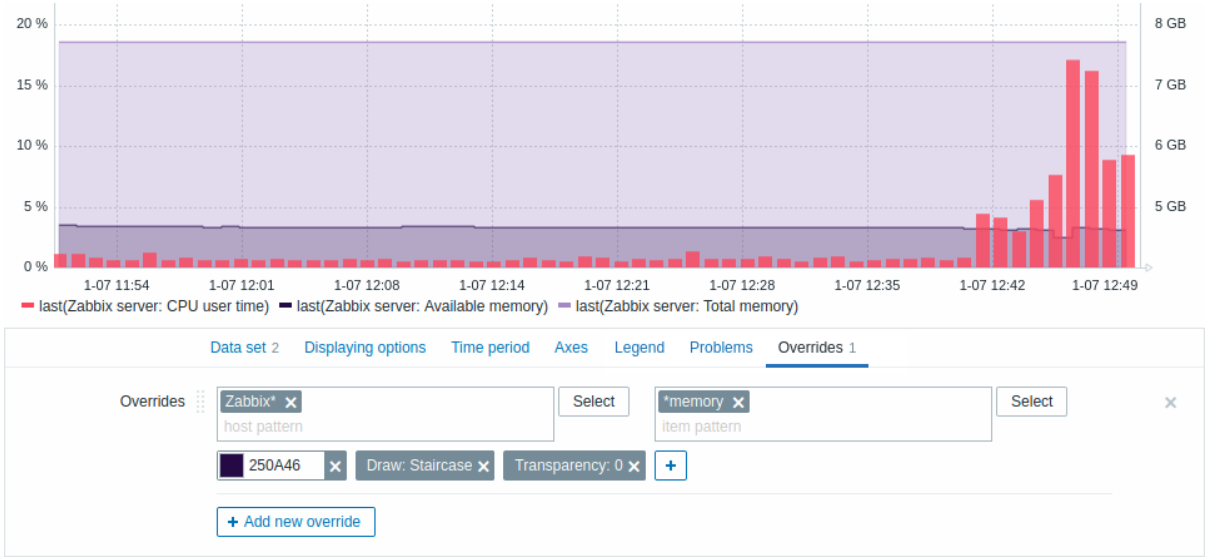
显示图例	选中启用查看图例（默认）。
行数	设置要在图表上显示的行数。

问题选项卡允许自定义问题显示：



显示问题	选中启用复选框，使问题显示在图表上（默认为禁用）。
仅选择的监控项	选中启用复选框只显示选定监控项的问题在图表上。
问题主机	选择要在图表上显示的问题主机。可以使用通配符模式（例如，* 将返回匹配零个或更多字符的结果）要使用通配只需要输入特定文本再按下回车。当你输入的时候，所有匹配的主机会在下拉框显示。
严重性	标记要在图表上显示的问题严重程度。
问题	指定要在图表上显示的问题名称。
标签	可以通过包括以及排除特定的标签和标签值的方法，指定标签以限制组件中同时显示的 Web 监控项的数量。标签名称匹配区分大小写。 可以通过以下运算符对每个条件进行筛选： **Exists** - 存在特定的标签名称 **Equals** - 存在特定的标签名称和值（区分大小写） **Contains** - 输入的字符串存在特定的标签名称（通配，不区分大小写） **Does not exist** - 排除存在特定的标签名称 **Does not equal** - 排除存在特定的标签名称和值（区分大小写） **Does not contain** - 排除输入的字符串存在特定的标签名称（通配，不区分大小写） 支持两种运算方式： **And/Or** - 所有条件同时满足，标签名相同的条件将按 Or 条件分组 **Or** - 满足任意条件即可

覆盖选项卡允许为数据集添加自定义覆盖：

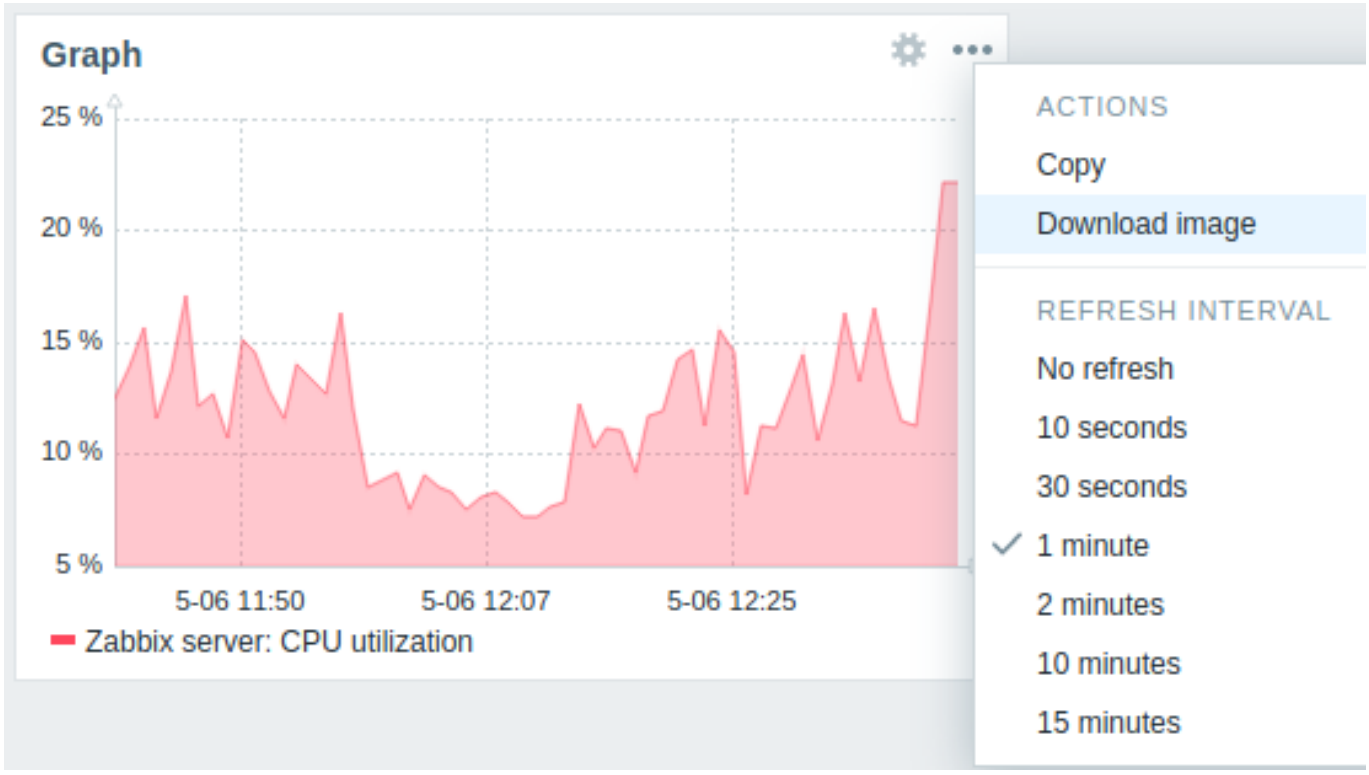


当使用 * 通配符为一个数据集选择了几个监控项，而你想改变这些监控项的默认显示方式（如默认的底色或任何其他属性）时，可以通过覆盖实现。

现有的覆盖（如果有的话）会显示在一个列表中。要添加一个新的覆盖：

- 点击 + Add new override 按钮
- 选择要在图表显示的主机和监控项，你也可以用主机和项目标签，支持通配符（例如，将返回匹配零个或更多字符的结果）。要使用通配只需要输入特定文本再按下回车。当你输入的时候，所有匹配的主机会在下拉框显示。通配符不可以被添加，因为它会被解析。例如一个项目为*"item"*，如果存在匹配项也会显示（例如 item2, item3）。主机参数和监控项参数必填。
- 点击 + 以选择覆盖参数。至少应选择一个覆盖参数。关于参数描述，见上面的数据集标签。

图表组件显示的信息可以通过组件菜单: 保存.png 图片：



组件的截图将被保存到浏览器的 "下载" 文件夹。

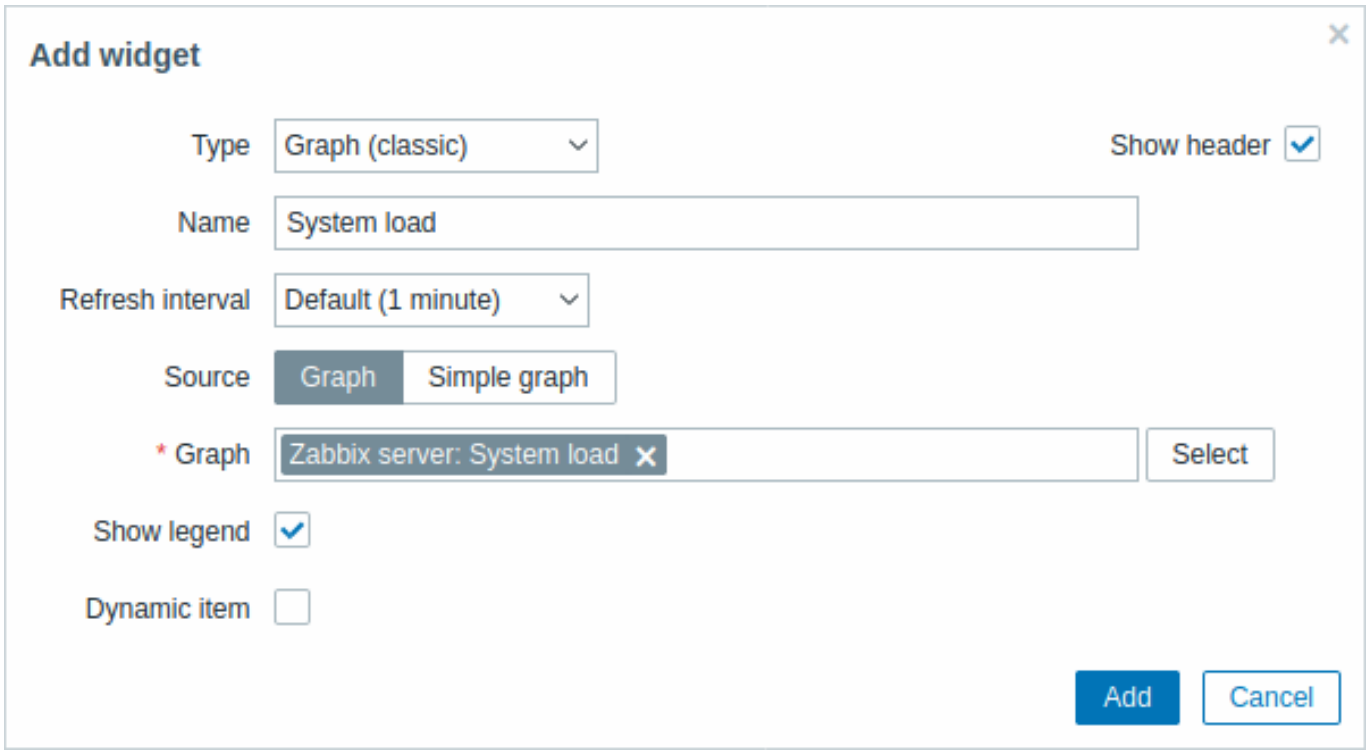
9 图表（经典）

概述

在经典图表组件中，你可以显示一个单一的自定义图表或简单图表。

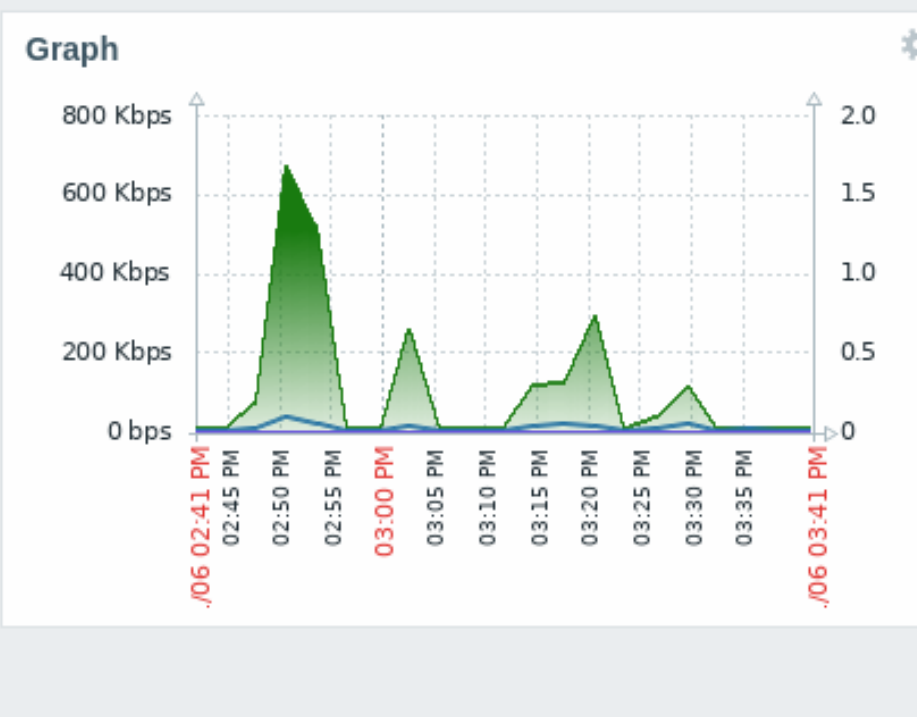
配置

选择图表（经典）作为类型：



通用 的配置参数对所有组件都是可用的，您还可以设置下列专属配置项：

来源	选择图表类型： 图表 - 自定义图表 简单图表 - 简单图表
图表	选择要显示的自定义图表。 如果选择“图表”作为来源，则此选项可用。
监控项	选择要在简单图表中显示的项目。 如果选择“简单图表”作为来源，则此选项可用。
显示图例	取消标记此复选框以隐藏图表上的图例（默认标记）。
动态项目	设置图表以根据所选主机显示不同的数据。



图表组件显示的信息可以通过组件菜单保存.png 图片：

组件的截图将被保存到浏览器的 “下载” 文件夹。

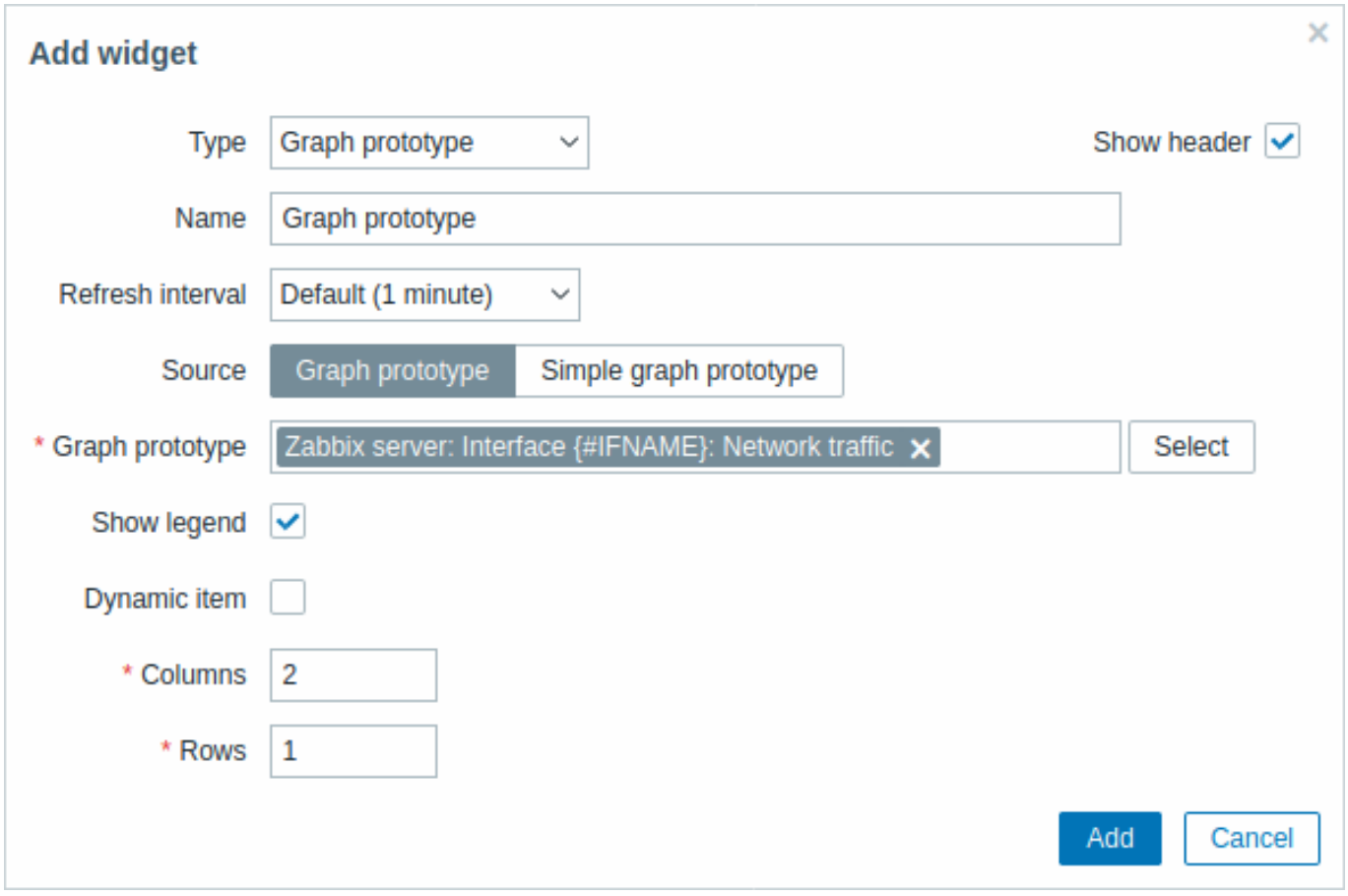
10 图形原型

概述

在图形原型组件中，你可以显示一个由图形原型或监控项原型通过自动发现创建的图形网格。

配置

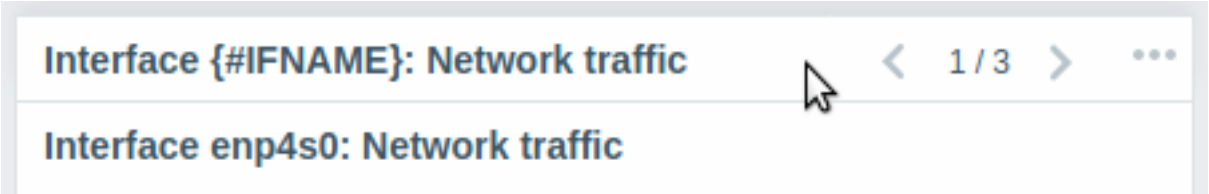
选择 图形原型作为组件类型：



通用 的配置参数对所有组件都是可用的，您还可以设置下列专属配置项：

来源	选择来源：图形原型或简单图形原型。
图形原型	选择图形原型以显示图形原型的已发现图形。 如果选择“图形原型”作为源，则此选项可用。
监控项原型	选择监控项原型以显示基于监控项原型的已发现监控项的简单图形。 如果选择“简单图形原型”作为源，则此选项可用。
显示图例	选择复选框以在图表上显示图例（默认选中）。
动态项目	根据选定的主机，设置图形显示不同的数据。
列	输入要在图形原型组件中显示的图形的列数。
行	输入要在图形原型组件中显示的图形行数。

虽然通过列和行的设置允许在小部件中合并显示多个图表，但展示的图表可能比小部件中的列/行多。在这种情况下，组件中可以使用分页，并且顶部标题允许使用左右箭头在页面之间切换。



######11 主机可用性 {#manual-web_interface-frontend_sections-monitoring-dashboard-widgets-host_availability}

概述

在主机可用性组件中，关于主机可用性的高级统计数据显示在四个彩色的列/线中。



水平显示（列）



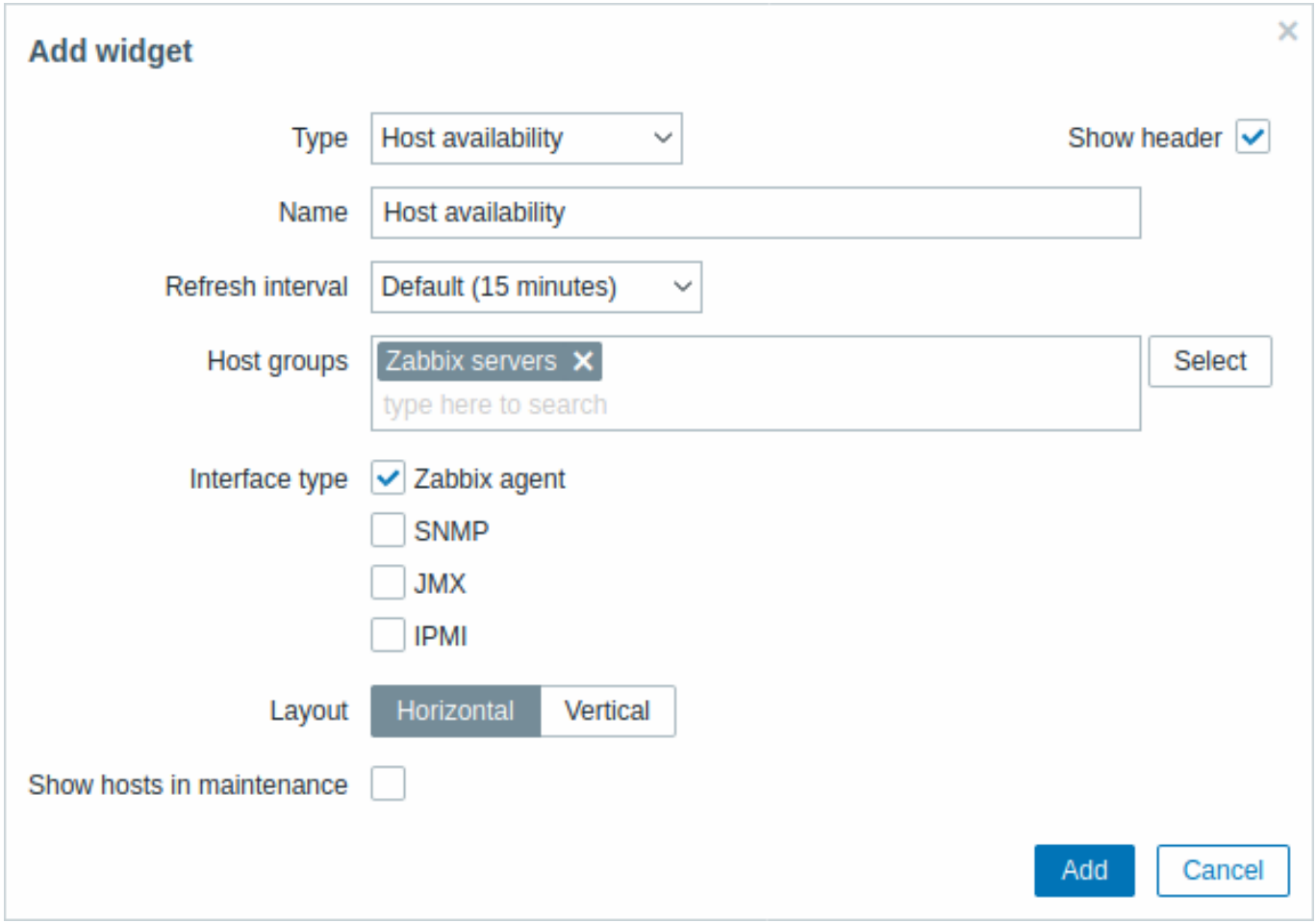
垂直显示（行）

每一列/行中的主机可用性的计算方法如下：

- 可用 - 所有接口都可用的主机
- 不可用 - 至少有一个接口不可用的主机
- 未知 - 至少有一个接口未知的主机（没有任何接口属于不可用状态）
- 总数 - 所有主机的总数

配置

选择 主机可用性作为类型：



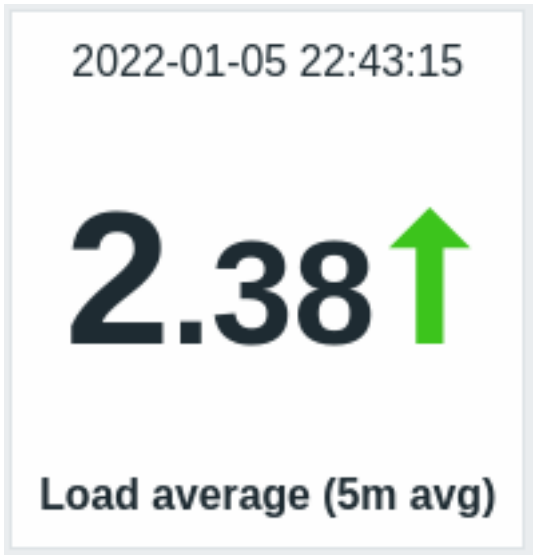
通用 的配置参数对所有组件都是可用的，您还可以设置下列专属配置项：

主机组	选择主机组。这个字段支持自动补全，输入一个组的名称在下拉框选择你需要的主机组名称。点击'x' 来移除已选择的主机组。
接口类型	选择你需要查看的主机接口的可用性数据。 如果没有选择选项，默认状态下所有接口的可用性将显示出来。
布局	选择水平显示（列）或垂直显示（行）。
显示维护中的主机	统计数量时包含维护中的主机。

12 监控项数据

概述

此组件对于突出显示单个监控项的数值很帮助。



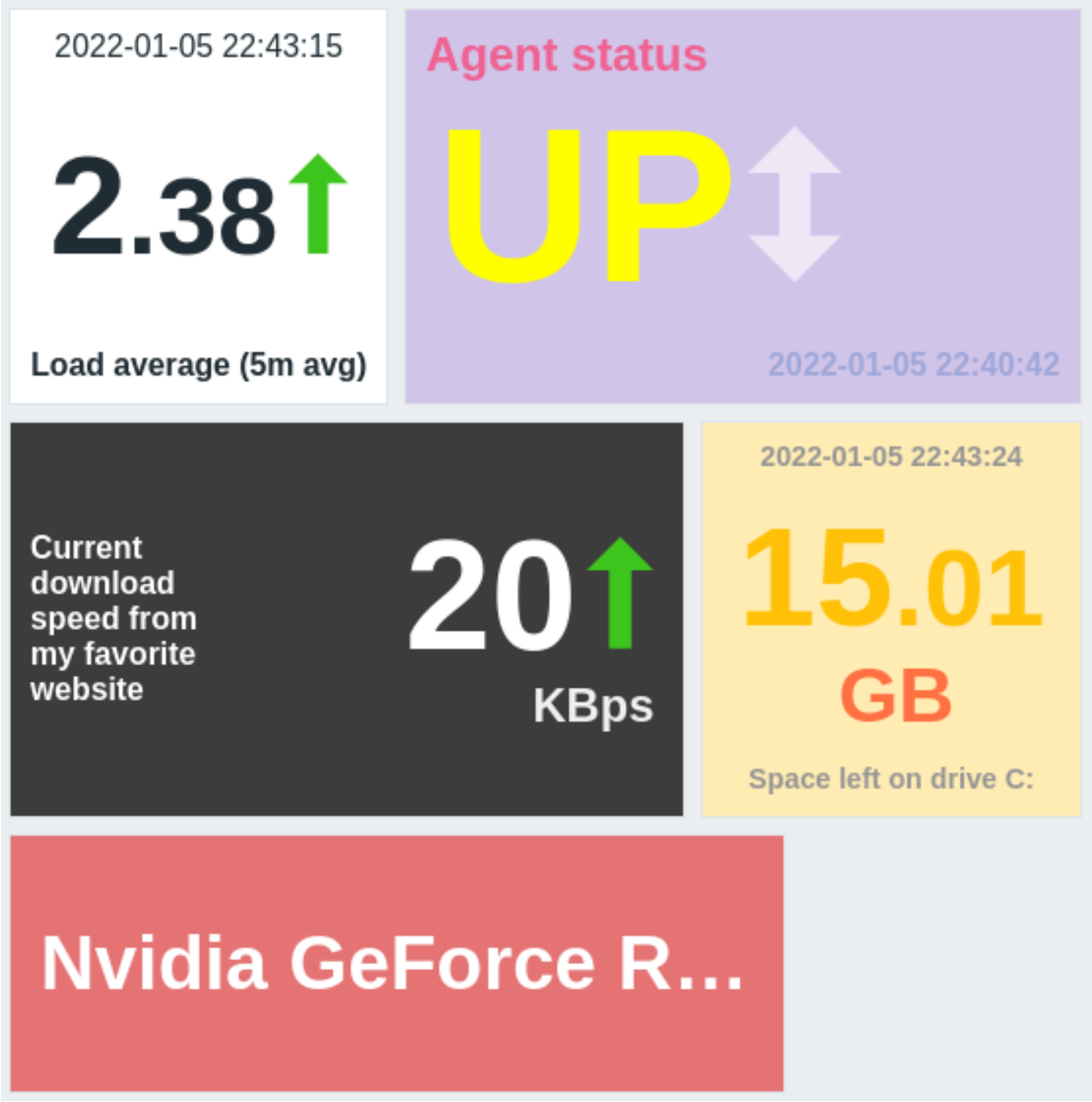
除了数值本身，如果需要，还可以显示其他元素：

- 数值统计的时间
- 监控项描述
- 数值变化的指示器
- 监控项单位

该组件可以显示数字和字符串文本。字符串值单行显示，如果需要的话会被截断。如果没有项目的值，则显示” 无数据”。

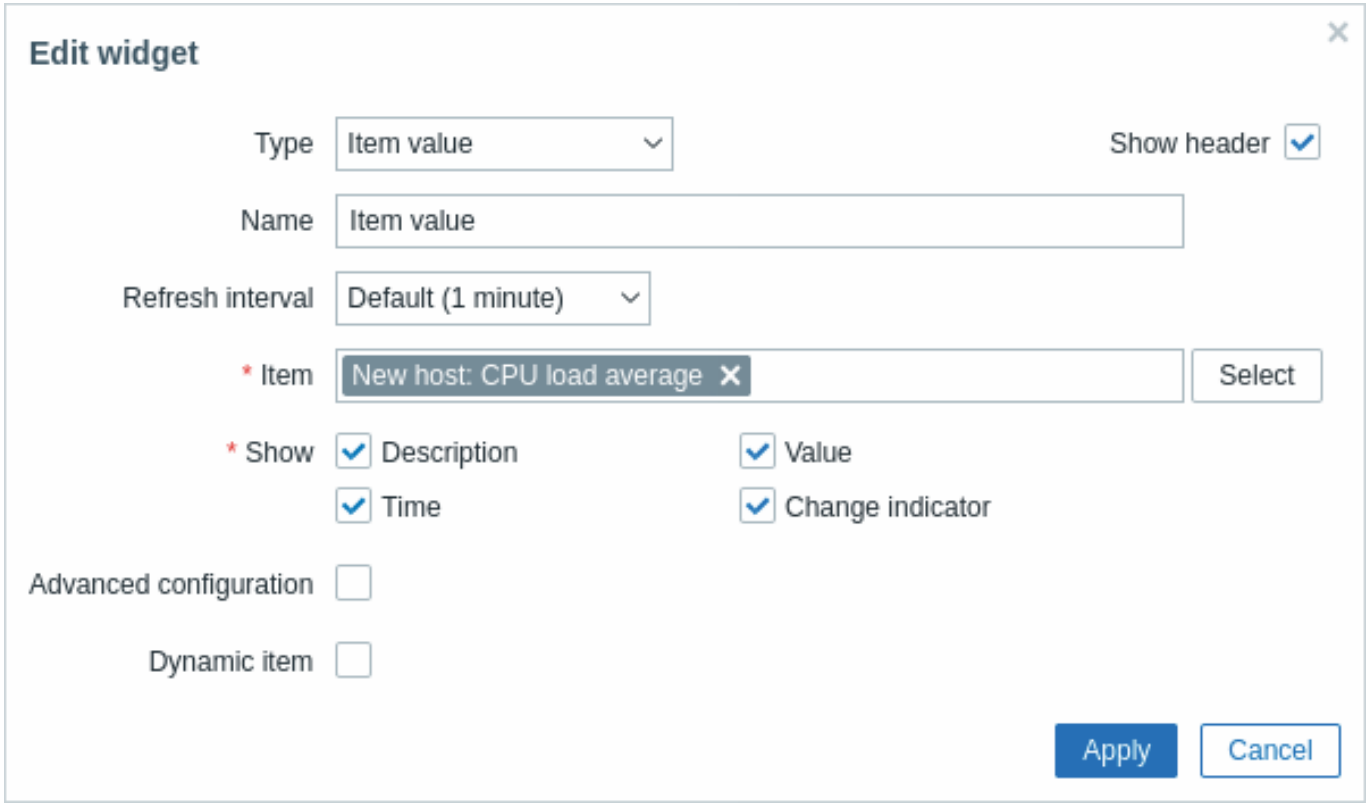
点击该值会出现数据监控项的 ad-hoc 图表或监控项的最新数据。

组件和其中的所有元素可以使用高级配置 选项进行视觉上的微调，允许创建各种各样的视觉风格：



配置

选择项目值作为组件类型：



通用 的配置参数对所有组件都是可用的，您还可以设置下列专属配置项：

监控项	选择监控项。
显示	选中复选框以显示相应的元素（描述、值、时间、变化指示器）。取消标记来隐藏。 至少选择一个元素。
高级配置	选中复选框以显示高级配置选项。
动态项目	选中复选框，取决于主机的不同会显示不同的数据。

高级配置

如果”高级配置”复选框被选中（如下图），高级配置选项就可用，而且只适用于那些在”显示”区域被选中的元素（见上文）。

此外，高级配置允许改变整个组件的背景颜色。



描述	输入监控项描述。该描述可以覆盖默认的监控项名称。支持多行描述。可以使用文本和支持的变量的组合。 支持 {HOST.*}、{ITEM.*}、{INVENTORY.*} 和用户变量。
水平位置	选择监控项显示的水平位置--左边、右边或居中。
垂直位置	选择监控项显示的垂直位置--顶部、底部或居中。
大小	输入监控项数据的字体大小高度（相对于组件总高度的百分比）。
粗体	选中复选框，粗体显示监控项数据。。
颜色	从颜色选择器中选择监控项数据字体的颜色。 D 代表默认颜色（取决于前端主题）。如需恢复默认值，请单击颜色选择器中的 *Use default* 按钮。
数据	
小数位数	此选项设置数据显示的小数点位数。此选项只影响浮点值。
尺寸	此选项为小数点后数字字体设置尺寸高度（相对于小组件总高度的百分比）
水平位置	选择监控项显示的水平位置--左边、右边或居中。
垂直位置	选择监控项显示的垂直位置--顶部、底部或居中。

大小	输入项目值的字体大小高度（相对于小部件总高度的百分比）。 请注意，监控项数据的位置是优先考虑的；其他元素必须为该值让出空间。不过对于变化指示器，如果数值太大，它将被截断以显示变化指示器。
粗体	选中复选框，粗体显示监控项数据。
颜色	从颜色选择器中选择监控项数据字体的颜色。 >D 代表默认颜色（取决于前端主题）。如需恢复默认值，请单击颜色选择器中的 *Use default* 按钮。
单位	选中复选框设置监控项的单位。如果你输入一个单位名称，它将覆盖监控项的单位。
位置	选择监控项单位的位置--在数值的上方、下方、前面或后面。
大小	输入监控项数据的字体大小高度（相对于组件总高度的百分比）。
粗体	选中复选框，粗体显示监控项。
颜色	从颜色选择器中选择监控项数据单位字体的颜色。 >D 代表默认颜色（取决于前端主题）。如需恢复默认值，请单击颜色选择器中的 *Use default* 按钮。
时间	监控项历史数据的时间。
水平位置	选择时间显示的水平位置--左边、右边或居中。
垂直位置	选择时间显示的垂直位置--顶部、底部或居中。
大小	输入时间的字体大小高度（相对于小部件总高度的百分比）。
粗体	选中复选框以粗体显示时间。
颜色	从颜色选择器中选择时间颜色。 D 代表默认颜色（取决于前端主题）。如需恢复默认值，请单击颜色选择器中的 *Use default* 按钮。
变化指标	从颜色选择器中选择变化指标的颜色。变化指标如下： ↑ - 监控项数据值增大（对于监控项数值） ↓ - 监控项数据值降低（对于监控项数值） **↕* * - 监控项数据值内容产生变化（对于监控项数值以及文本） D 代表默认颜色（取决于前端主题）。如需恢复默认值，请单击颜色选择器中的 *Use default* 按钮。 变化指示器的垂直大小等于值的大小（对于监控项数值的整数部分）。 注意只有一个值时，不会显示向上和向下的指示器。
背景颜色	从颜色选择器中选择整个组件的背景颜色。 D 代表默认颜色（取决于前端主题）。如需恢复默认值，请单击颜色选择器中的 *Use default* 按钮。

注意多个元素不能部署在同一空间；如果它们被放在同一空间，将显示一个错误信息。

13 拓扑图

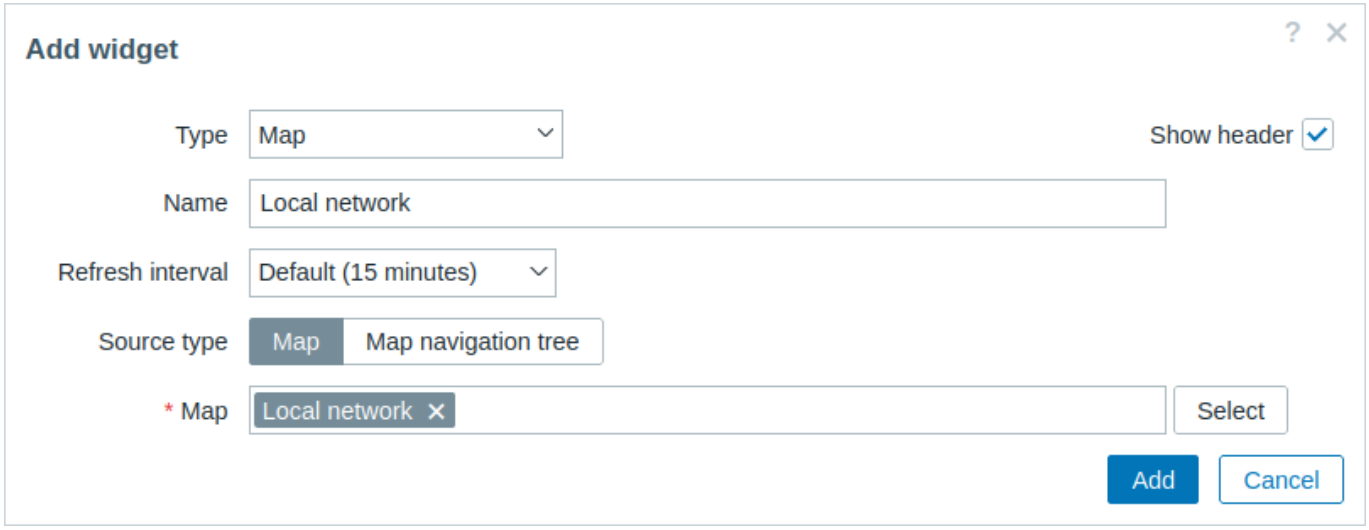
概述

在拓扑图组件中，你可以通过以下两种方式展现：

- 一张配置好的网络拓扑图
- 拓扑图导航树中选择一个配置好的网络地图（当点击导航树中的地图名称时）。

配置

选择 拓扑图作为类型：



通用 的配置参数对所有组件都是可用的，您还可以设置下列专属配置项：

来源类型	选择显示的内容： 拓扑图 - 网络拓扑图 拓扑图导航树 - 选定的拓扑图导航树中的一个拓扑图
拓扑图	选择要显示的拓扑图。 这个选项在" 拓扑图" 被选为源类型时可用。
筛选	选择要显示的拓扑图导航树的拓扑图。 这个选项在" 拓扑图导航树" 被选为源类型时可用。

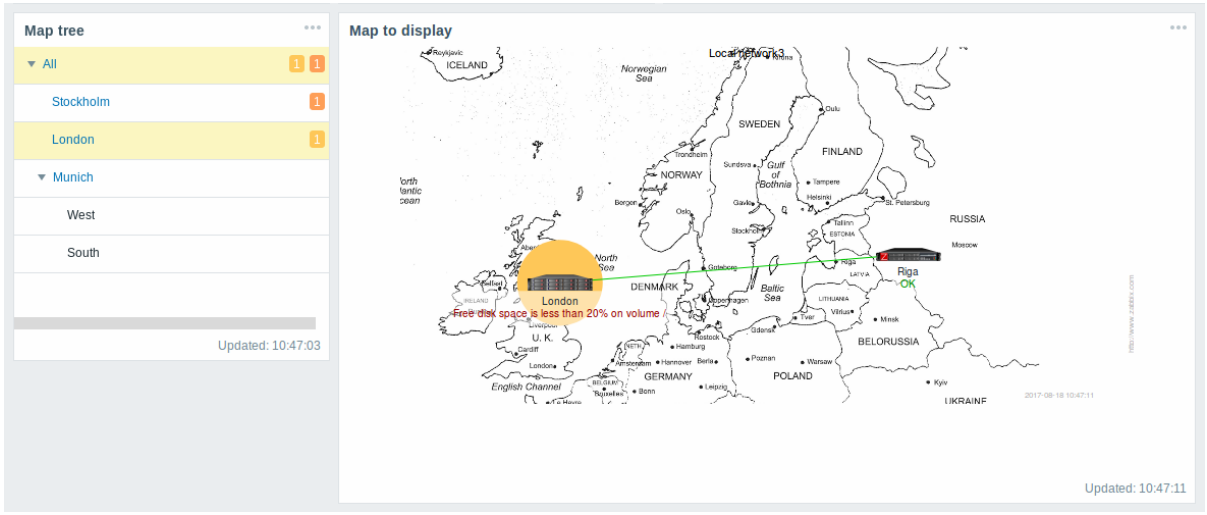
另请参阅：IE11 浏览器中的已知问题

14 拓扑图导航树

概述

这个组件允许建立一个现有拓扑图的层次结构，同时也显示每个包含的拓扑图和拓扑图组的问题统计。

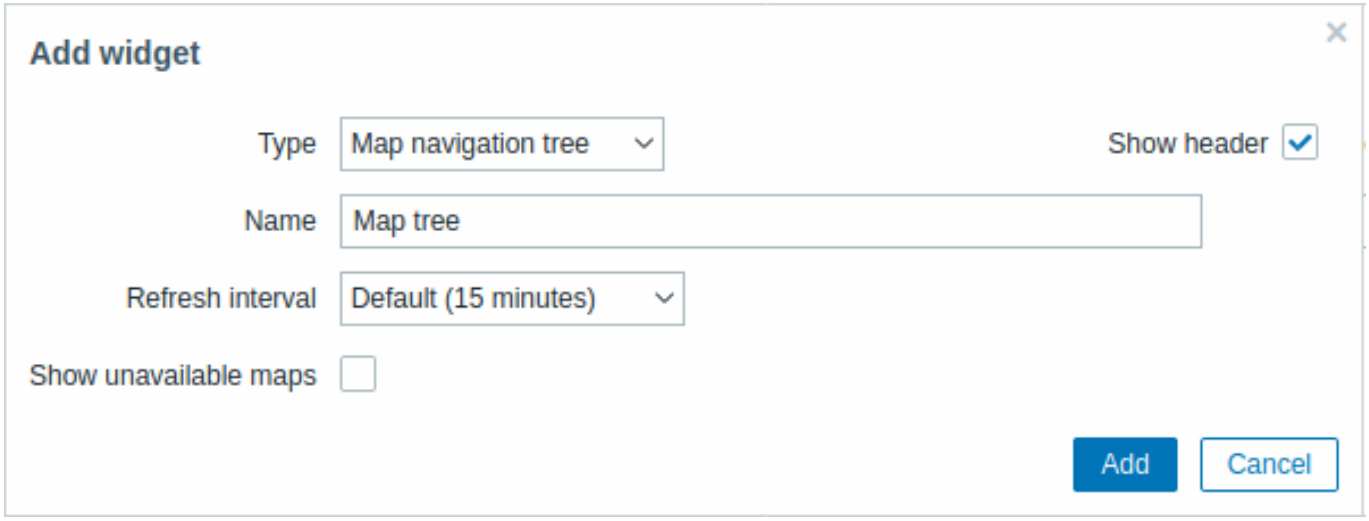
如果您将拓扑图组件链接到导航树上，他将变得更佳实用。在这种情况下，点击导航树中的拓扑图名称，就会在拓扑图组件中显示拓扑图的全貌。



在层次结构中，顶级拓扑图会统计显示所有子拓扑图的问题和顶级拓扑图自身问题的总和。

配置

选择 拓扑图导航树作为类型：

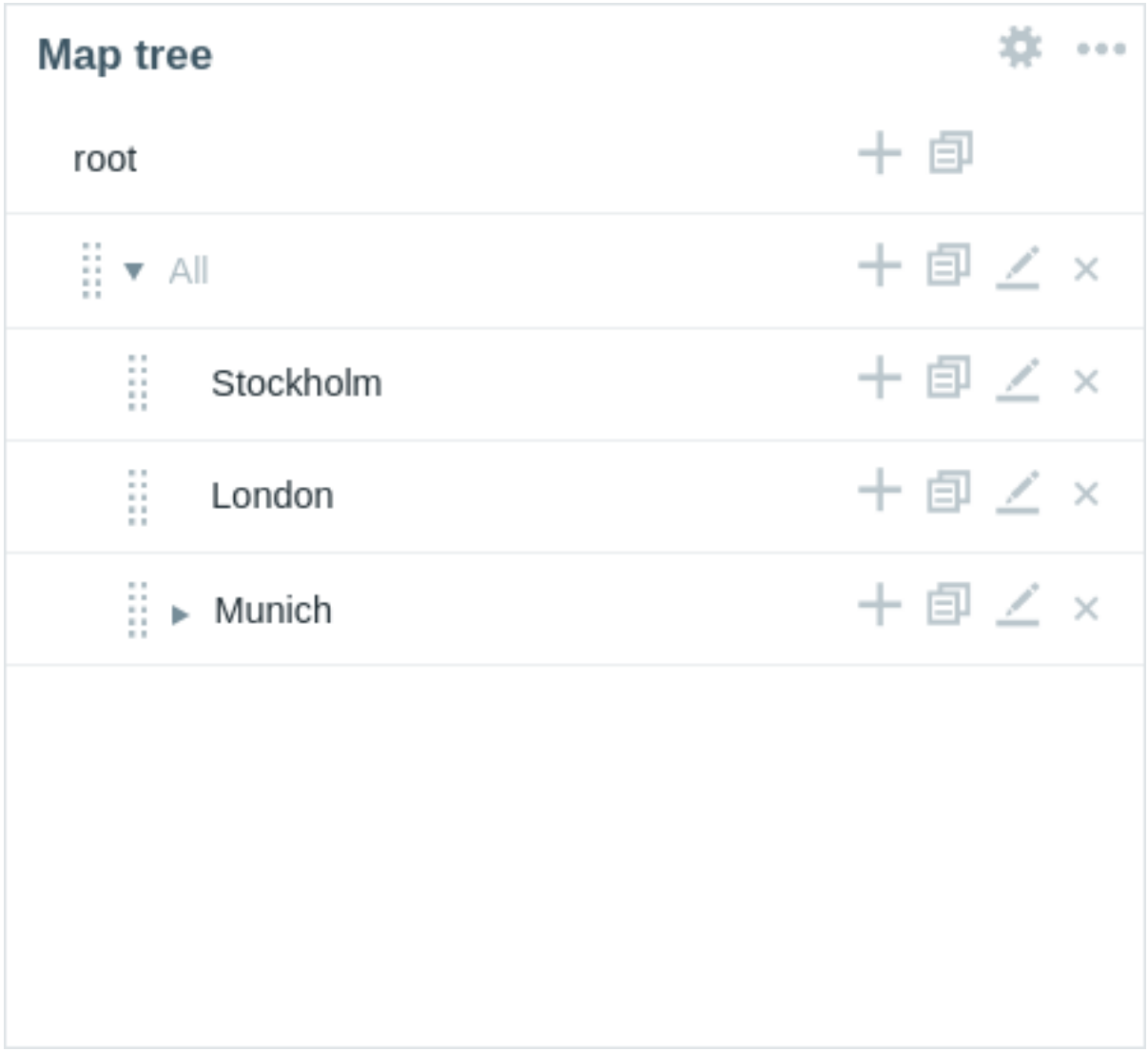


通用 的配置参数对所有组件都是可用的，您还可以设置下列专属配置项：

显示不可用的拓扑图	选中复选框以显示用户没有访问权限的拓扑图。 导航树中不可用的拓扑图会以灰色的图标显示。 注意，如果该复选框被选中，即使父级拓扑图不可用，也会显示可用的子拓扑图。如果没有选中，父拓扑图不可用时，即使有权限的子拓扑图也无法显示。 问题计数是根据可用拓扑图和可用拓扑图元素计算的。

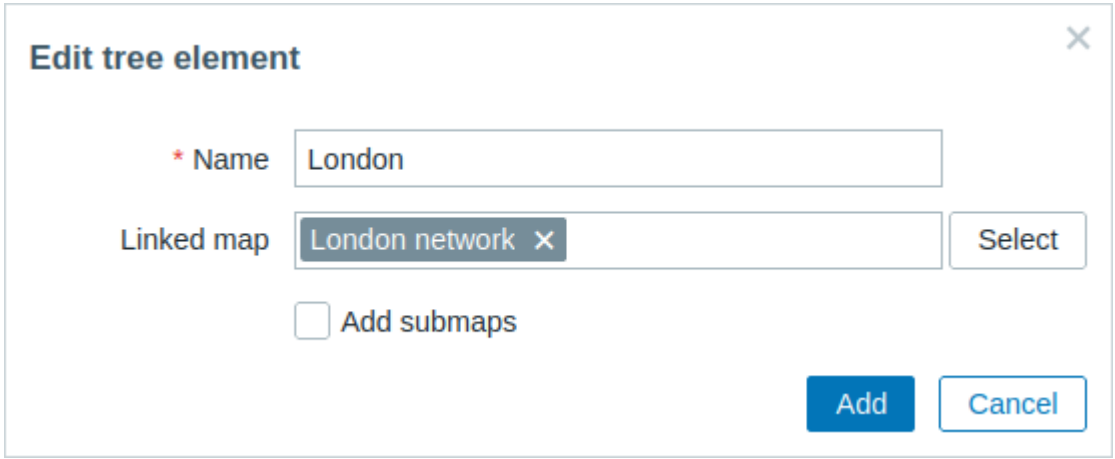
Navigation tree elements are displayed in a list. You can:

- drag an element (including its child elements) to a new place in the list;
- expand or collapse an element to display or hide its child elements;
- add a child element (with or without a linked map) to an element;
- add multiple child elements (with linked maps) to an element;
- edit an element;
- remove an element (including its child elements).



Element configuration

To configure a navigation tree element, either add a new element or edit an existing element.



The following navigation tree element configuration parameters are available:

Name	Enter the navigation tree element name.
Linked map	Select the map to link to the navigation tree element. This field is auto-complete so starting to type the name of a map will offer a dropdown of matching maps.
Add submaps	Mark this checkbox to add the submaps of the linked map as child elements to the navigation tree element.

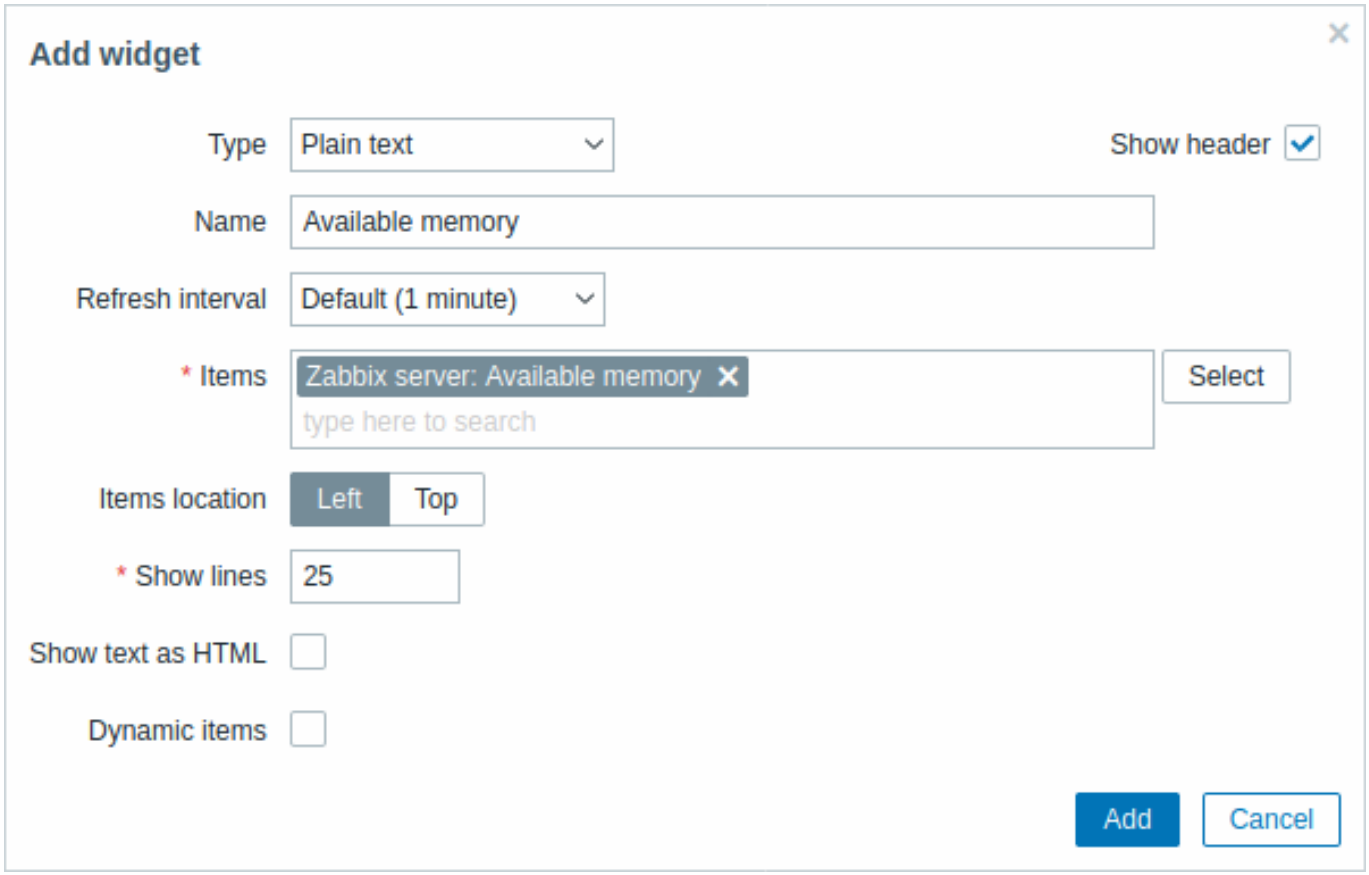
15 纯文本

概述

在纯文本组件中，显示最新的纯文本监控项数据。

配置

选择 纯文本作为类型：



通用 的配置参数对所有组件都是可用的，您还可以设置下列专属配置项：

监控项	选择监控项。
监控项位置	选择要在组件中显示的选定监控项的位置。
显示行数	设置将在组件中显示多少最新的数据行。
显示文本为 *HTML*	设置以 HTML 方式显示文本。
动态监控项	设置根据选定的主机显示不同的数据。

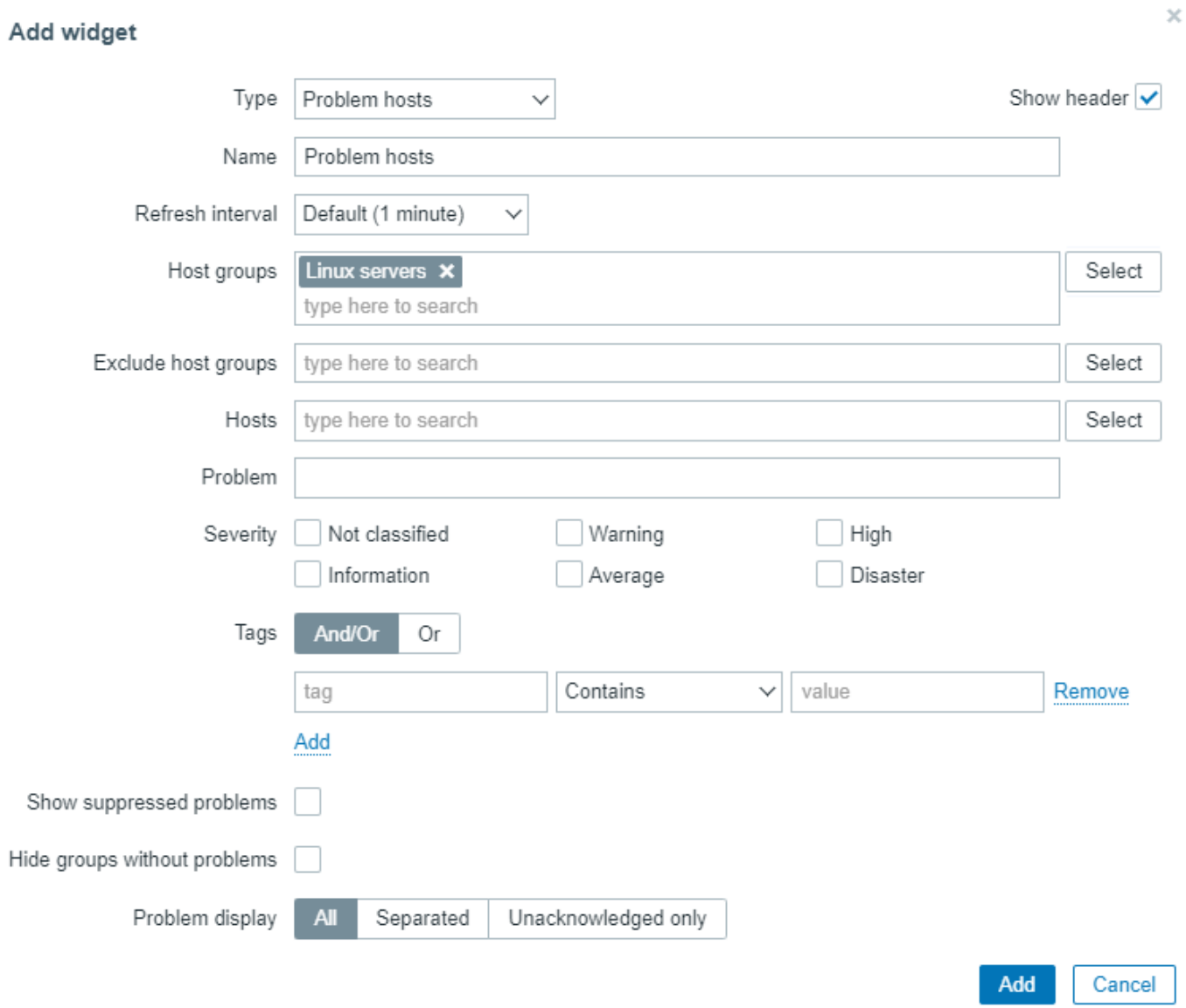
16 问题主机

概述

在问题主机组件中，可以显示关于主机可用性的高级信息。

配置

选择 问题主机作为类型：



通用 的配置参数对所有组件都是可用的，您还可以设置下列专属配置项：

参数	说明
主机组	输入要在组件中显示的主机组。这个字段支持自动补全，输入一个组的名称在下拉框选择你需要的主机组名称。 指定一个父级主机组意味着同时选择所有嵌套的主机组。 来自这些主机组的主机数据将被显示在组件中。如果没有输入主机组，所有的主机组将被显示。
排除主机组	输入要从组件中隐藏的主机组。这个字段支持自动补全，输入一个组的名称在下拉框选择你需要的主机组名称。 指定一个父级主机组意味着同时选择所有嵌套的主机组。 来自这些主机组的主机数据将不会显示在组件中。例如主机 001, 002, 003 属于 A 组，主机 002, 003 也属于 B 组。如果我们选择同时显示 A 组和排除 B 组，只有主机 001 的数据将显示在仪表板中。
主机	输入要在组件中显示的主机。这个字段支持自动补全，输入一个组的名称在下拉框选择你需要的主机名称。 如果未输入主机，将显示所有主机。
问题	可以输入问题名称来控制显示的主机数量，仅显示问题名称包含输入的文本的主机。不支持通配。
严重性	标记要在组件中显示的问题严重性。

参数	说明
标签	可以通过包括以及排除特定的标签和标签值的方法，指定标签以限制组件中同时显示的 Web 监控项的数量。标签名称匹配区分大小写。 可以通过以下运算符对每个条件进行筛选： **Exists** - 存在特定的标签名称 **Equals** - 存在特定的标签名称和值（区分大小写） **Contains** - 输入的字符串存在特定的标签名称（通配，不区分大小写） **Does not exist** - 排除存在特定的标签名称 **Does not equal** - 排除存在特定的标签名称和值（区分大小写） **Does not contain** - 排除输入的字符串存在特定的标签名称（通配，不区分大小写） 支持两种运算方式： **And/Or** - 所有条件同时满足，标签名相同的条件将按 Or 条件分组 **Or** - 满足任意条件即可
显示抑制的问题	选中复选框以显示因主机维护而被抑制（不显示）的问题。
隐藏没有问题的组	选中复选框以隐藏没有问题的主机组的数据。
问题显示	按照以下方法计算问题数量： **All** - 所有的问题都计入总数 **Separated** - 独立计算未确认问题和问题总数 **Unacknowledged only** - 仅计算未确认问题的数量。

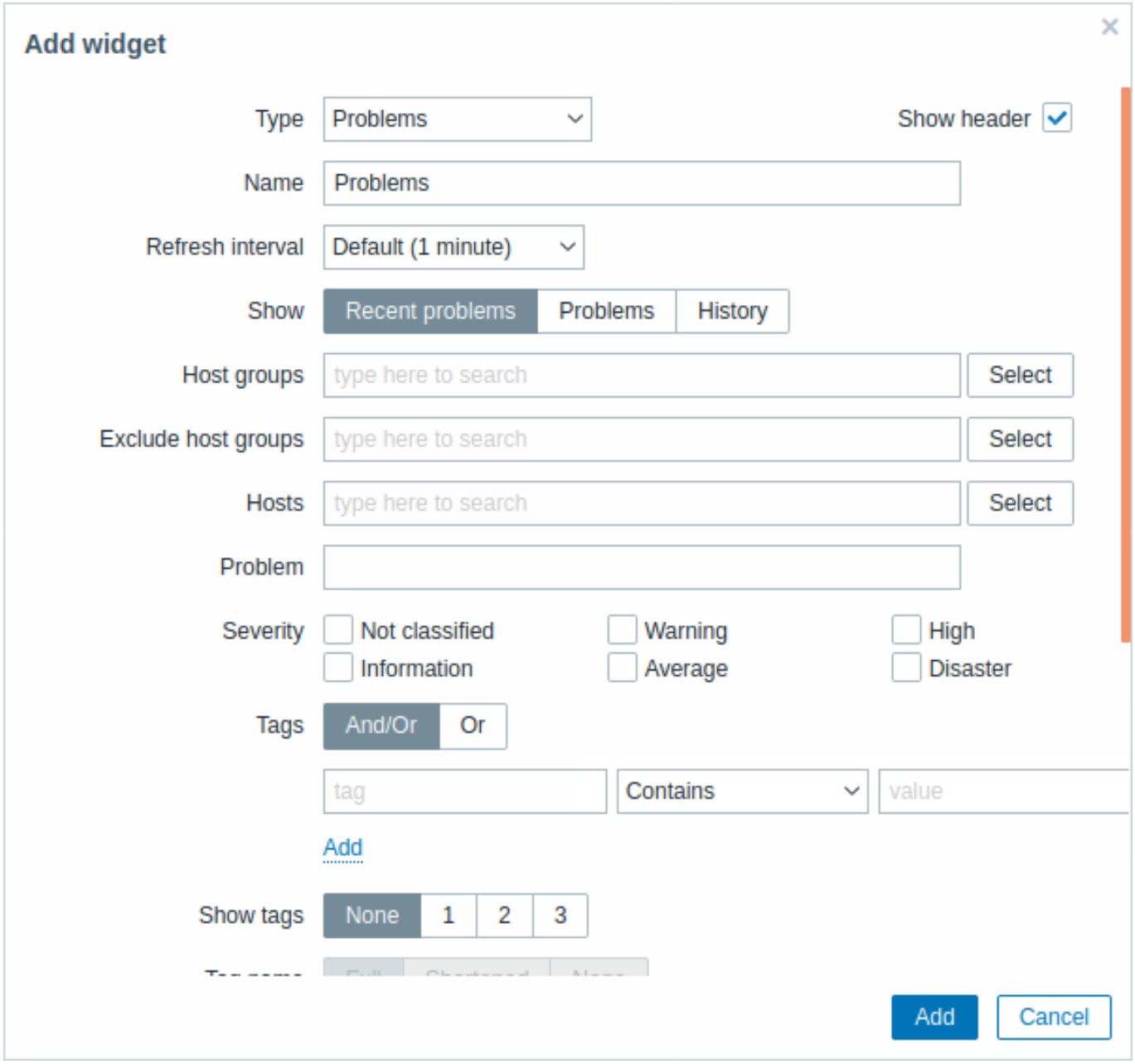
17 问题

概述

在这个组件中，你可以显示当前存在的问题。这个组件中的信息类似于 *Monitoring → Problems*。

配置

选择 问题作为类型：



您可以通过各种方式限制组件中显示的问题数量--按问题状态、问题名称、严重程度、主机组、主机、事件标签、确认状态等。

参数	说明
显示	按问题状态过滤： 最近的问题 - 显示未解决和最近解决的问题（默认） 问题 - 显示未解决的问题 历史记录 - 显示所有事件的历史
主机组	输入要在组件中显示的主机组。这个字段支持自动补全，输入一个组的名称在下拉框选择你需要的主机组名称。 指定一个父级主机组意味着同时选择所有嵌套的主机组。 来自这些主机组的主机数据将被显示在组件中。如果没有输入主机组，所有的主机组将被显示。
排除主机组	输入要从组件中隐藏的主机组。这个字段支持自动补全，输入一个组的名称在下拉框选择你需要的主机组名称。 指定一个父级主机组意味着同时选择所有嵌套的主机组。 来自这些主机组的主机数据将不会显示在组件中。例如主机 001, 002, 003 属于 A 组，主机 002, 003 也属于 B 组。如果我们选择同时显示 A 组和排除 B 组，只有主机 001 的数据将显示在仪表板中。
主机	输入要在组件中显示的主机。这个字段支持自动补全，输入一个组的名称在下拉框选择你需要的主机名称。 如果未输入主机，将显示所有主机。

参数	说明
问题	可以输入问题名称来控制显示的主机数量，仅显示问题名称包含输入的文本的主机。不支持通配。
严重性	标记要在组件中显示的问题严重性。
标签	可以通过包括以及排除特定的标签和标签值的方法，指定标签以限制组件中同时显示的 Web 监控项的数量。标签名称匹配区分大小写。 可以通过以下运算符对每个条件进行筛选： **Exists** - 存在特定的标签名称 **Equals** - 存在特定的标签名称和值（区分大小写） **Contains** - 输入的字符串存在特定的标签名称（通配，不区分大小写） **Does not exist** - 排除存在特定的标签名称 **Does not equal** - 排除存在特定的标签名称和值（区分大小写） **Does not contain** - 排除输入的字符串存在特定的标签名称（通配，不区分大小写） 支持两种运算方式： **And/Or** - 所有条件同时满足，标签名相同的条件将按 Or 条件分组 **Or** - 满足任意条件即可 过滤模式限定的标签会被优先显示，可以被标签显示优先级覆盖，见下文。
显示标签	选择显示标签的数量： **None** - 监测 → 问题中没有标签列 **1** - 标签列包含一个标签 < br>**2** - 标签列包含两个标签 **3** - 标签列包含三个标签 要查看该问题的所有标签，请将鼠标移到三个点的图标上。
标签名称	选择标签名称显示模式： **Full** - 标记的名称和值被完整地显示出来 **Shortened** - 标签名称被缩短为 3 个符号；标签值被完整显示 无 - 只显示标签值，不显示名称。
标签显示优先级	输入问题的标签显示优先级，以逗号分隔的标签列表（例如：服务、应用程序、应用）。优先级仅匹配标签名称不匹配具体数据。此列表中的标签将始终被优先显示，而不是按字母顺序自然排序。
显示操作数据	选择显示操作数据的模式: **None** - 不显示操作数据 **Separately** - 操作数据显示在单独的列中 **With problem name** - 将操作数据附加到问题名称，使用括号表示操作数据
显示抑制的问题	选中复选框以显示因主机维护而被抑制（不显示）的问题。
仅显示未确认的问题	选中复选框只显示未确认的问题。
排序依据	按以下顺序排序条目： **Time**（降序或升序） **Severity** （降序或升序） **Problem name**（降序或升序） **Host**（降序或升序）
显示时间线	选中复选框显示时间轴
显示行	选中显示的问题最大行数。

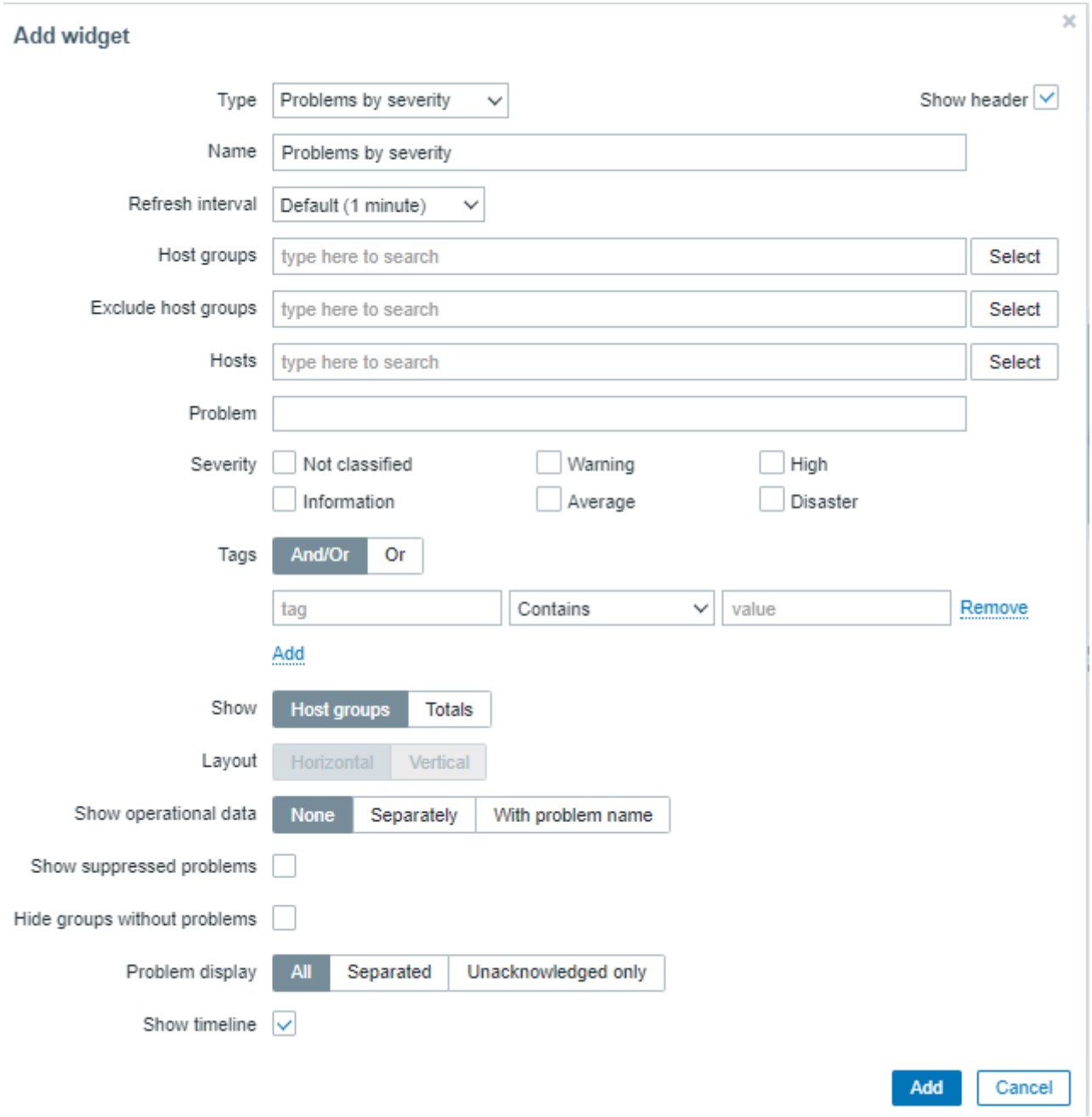
18 问题严重性

概述

在这个组件中，您可以按严重程度显示问题。您可以限制在组件中显示哪些主机和触发器，并定义问题计数的显示方式。

配置

点击配置，选择 *Problems by severity* 类型：



通用 的配置参数对所有组件都是可用的，您还可以设置下列专属配置项：

参数	说明
主机组	输入要在组件中显示的主机组。这个字段支持自动补全，输入一个组的名称在下拉框选择你需要的主机组名称。 指定一个父级主机组意味着同时选择所有嵌套的主机组。 来自这些主机组的主机数据将被显示在组件中。如果没有输入主机组，所有的主机组将被显示。
排除主机组	输入要从组件中隐藏的主机组。这个字段支持自动补全，输入一个组的名称在下拉框选择你需要的主机组名称。 指定一个父级主机组意味着同时选择所有嵌套的主机组。 来自这些主机组的主机数据将不会显示在组件中。例如主机 001, 002, 003 属于 A 组，主机 002, 003 也属于 B 组。如果我们选择同时显示 A 组和排除 B 组，只有主机 001 的数据将显示在仪表板中。
主机	输入要在组件中显示的主机。这个字段支持自动补全，输入一个主机的名称在下拉框选择你需要的主机名称。 如果未输入主机，将显示所有主机。
问题	可以输入问题名称来控制显示的主机数量，仅显示问题名称包含输入的文本的主机。宏不会展开。
严重性	标记要在组件中显示的问题严重性。

参数	说明
标签	可以通过包括以及排除特定的标签和标签值的方法，指定标签以限制组件中同时显示的 Web 监控项的数量。可以设置多个条件，标签名称匹配始终区分大小写。 可以通过以下运算符对每个条件进行筛选： **Exists** - 存在特定的标签名称 **Equals** - 存在特定的标签名称和值（区分大小写） **Contains** - 输入的字符串存在特定的标签名称（通配，不区分大小写） **Does not exist** - 排除存在特定的标签名称 **Does not equal** - 排除存在特定的标签名称和值（区分大小写） **Does not contain** - 排除输入的字符串存在特定的标签名称（字符串匹配，不区分大小写） 支持两种运算方式： **And/Or** - 所有条件同时满足，标签名相同的条件将按 Or 条件分组 **Or** - 满足任意条件即可
显示	选择显示选项： **Host groups** - 按照主机组显示问题 **Totals** - 在与问题严重性相对应的彩色块中显示所有选定主机组的问题总数。
布局	选择布局选项： **Horizontal** - 所有颜色方块水平显示 **Vertical** - 所有颜色方块垂直显示。 如果”总数”被选为显示选项，这个字段可修改。
显示抑制的问题	选中复选框以显示因主机维护而被抑制（不显示）的问题。
隐藏没有问题的组	选中复选框以隐藏没有问题的主机组以及数据。
显示操作数据	选中复选框以显示操作数据（参见 监控 → 问题中操作数据 的描述）。
问题显示	将问题计数显示为： **All** - 将显示全部问题总数 **Separated** - 未确认的问题总数和全部问题总数单独显示 **Unacknowledged only** - 只显示未确认的问题数。
显示时间线	选中复选框以显示时间线。

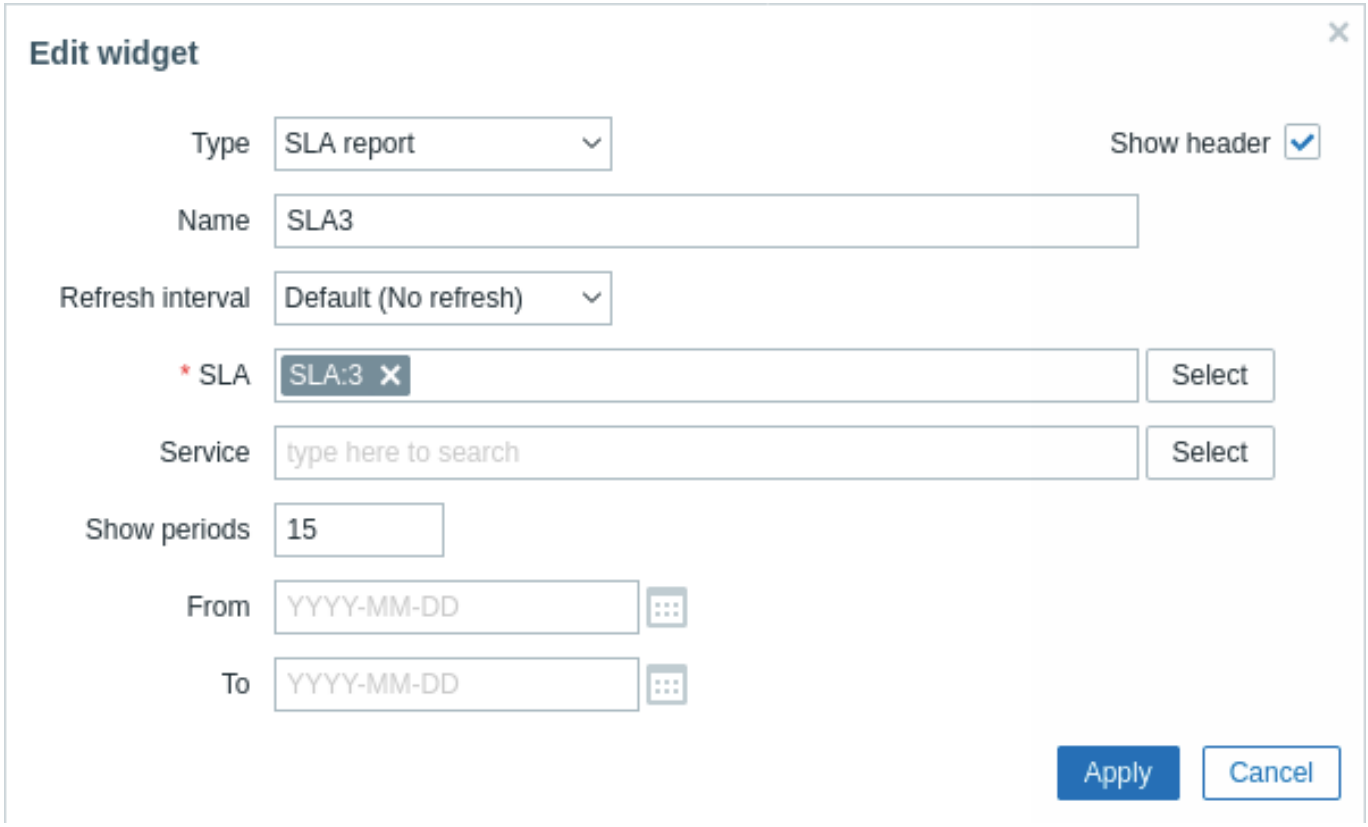
19 SLA 报表

概览

该组件用于显示SLA 报表. 功能上类似于 服务 -> *SLA* 报表选项。

配置

点击配置，类型选择 *SLA* 报表：



除了所有小部件通用参数 外，您还可以设置以下特定选项：

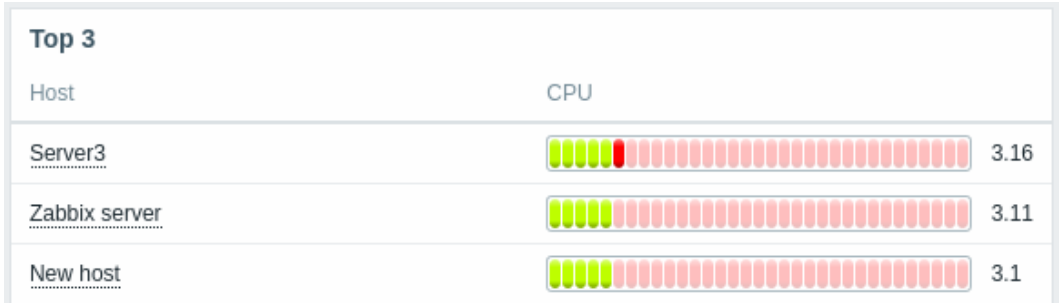
SLA	选择 SLA 报表。
Service	选择服务报表。
Show periods	设置小部件中显示的时间段（默认为 20，最大为 100）。
From	选择报告的开始日期 支持相对日期：`now`、`now/d`、`now/w-1w` 等；支持的日期修饰符：`d`、`w`、`M`、`y`。
To	选择报告的结束日期 支持相对日期：`now`、`now/d`、`now/w-1w` 等；支持的日期修饰符：`d`、`w`、`M`、`y`。

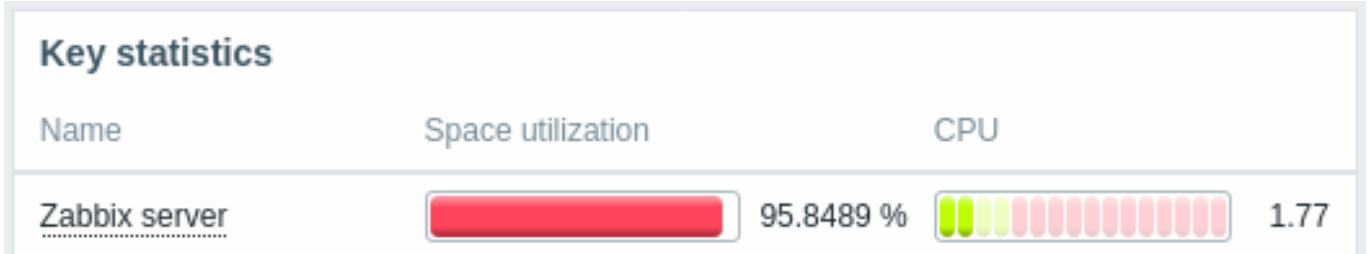
20 Top 主机

概览

此小部件提供了一种创建自定义表格以显示数据情况的方法，允许显示对容量规划有用的类似 *Top N* 的报告和进度条报告。

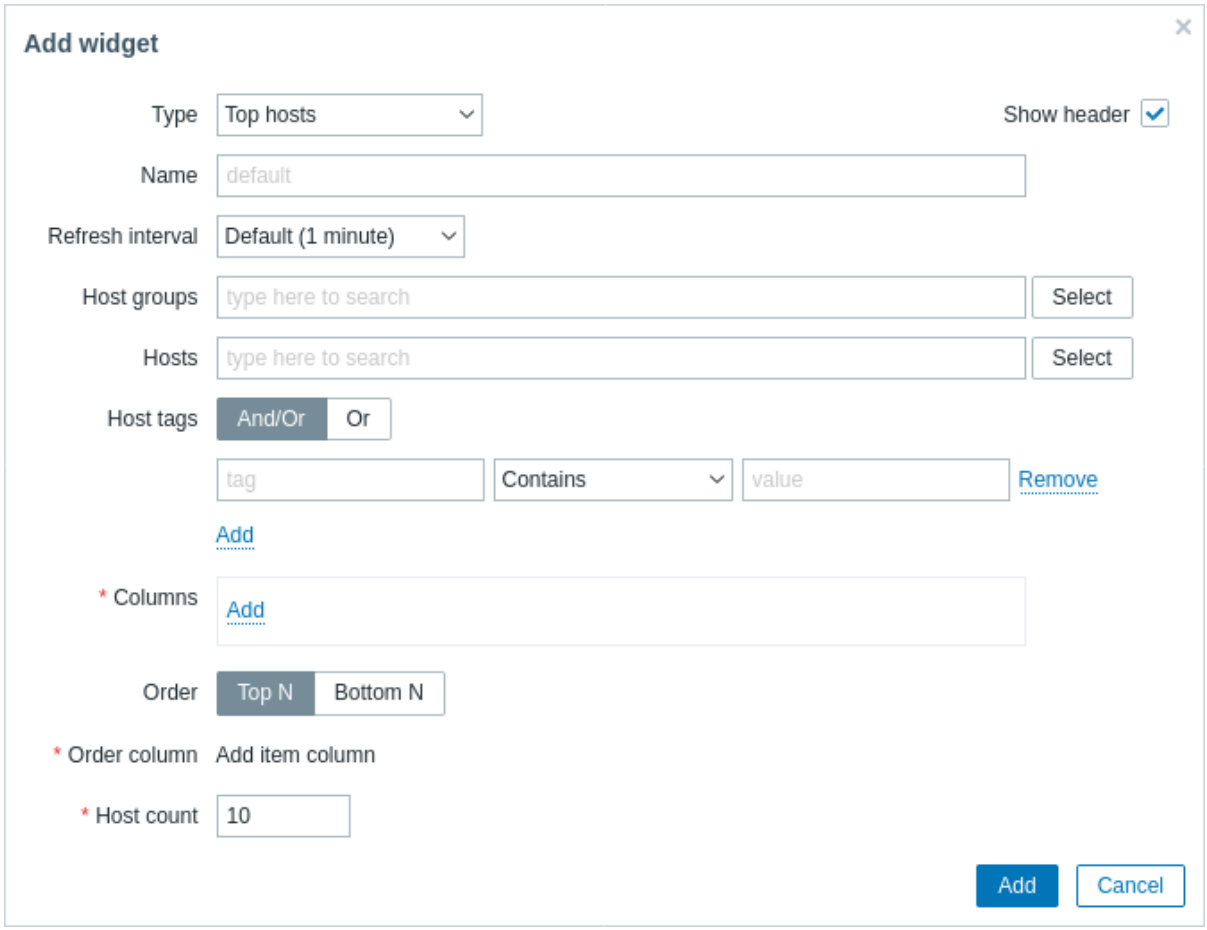
最大可显示 100 个主机。





配置

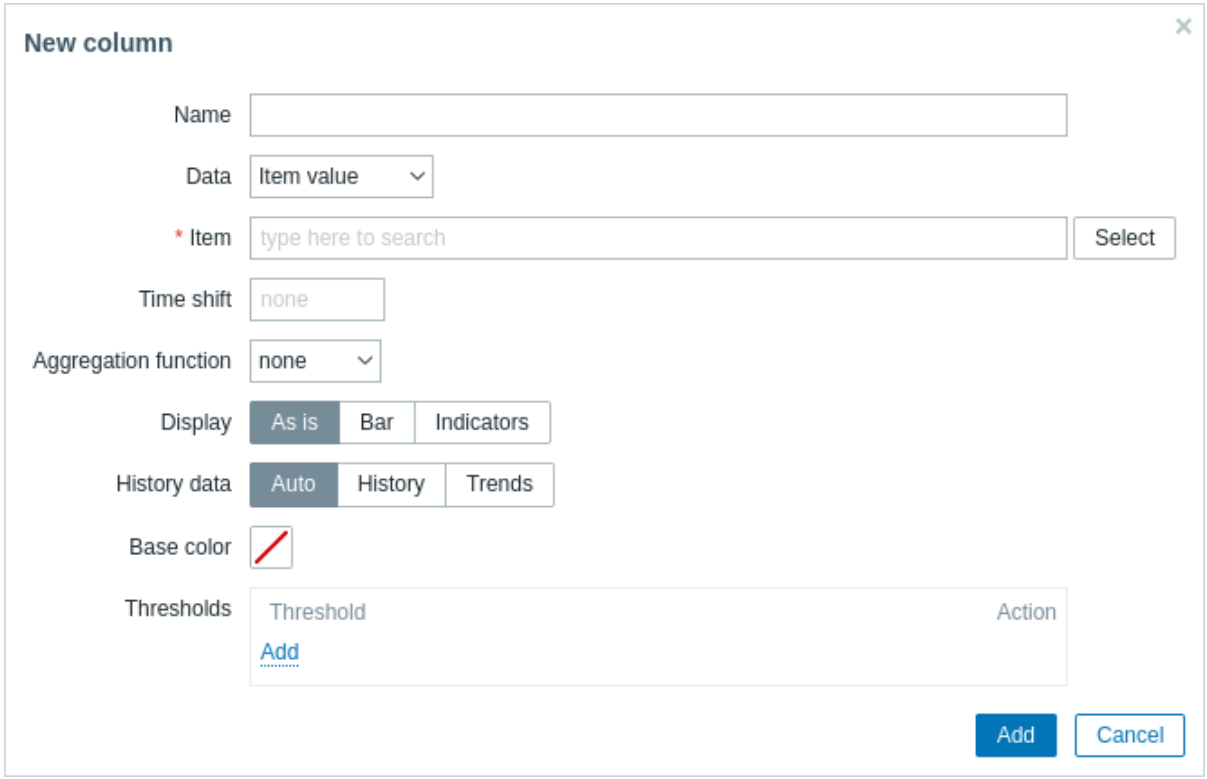
点击配置，选择类型为 *Top hosts*：



除了所有小部件通用参数 外，您还可以设置以下特定选项：

Host groups	要显示数据的主机组。
Hosts	要显示数据的主机。
Host tags	要显示数据的主机。用于限制小部件中显示的主机数量的专用标签。可以包括或排除特定标签和标签值。可以设置多个条件。标签名匹配始终区分大小写 每个条件都有几个可用的运算符： 存在-包括指定的标签名 等于-包括指定的标签名和值（区分大小写） 包含-包括指定的标签名，其中标签值包含输入的字符串（子字符串匹配，不区分大小写） 不存在-排除指定的标签名相等**-**排除指定的标签名和值（区分大小写）** ** 不包含**-**排除指定的标签名，其中标签值包含输入的字符串（子字符串匹配，不区分大小写）** ** 条件有两种计算类型：** ** 和/或 **-必须满足所有条件，具有相同标签名的条件将按或条件分组，如果满足一个条件，则足够了
Columns	添加要显示的数据列（# 列） 列顺序决定了它们从左到右的显示 通过在列名之前的句柄上下拖动，可以对列重新排序。
Order	指定行的顺序： **TopN**- 按排序列聚合值降序排列 **Bottom N**- 按排序列聚合值升序排列
Order column	从定义的列列表中指定要用于 *Top N* 或 *Bottom N* 排序的列。
Host count	要显示的主机行数（1 - 100）。

列配置



通用列参数:

Name	列的名称。
Data	要在列中显示的数据类型： 监控项的值-指定监控项的值 主机名-在监控项的值列中指定监控项的主机名 文本-静态文本字符串
Base color	列的背景色；如果监控项的值数据显示为条形图/指示器，则填充颜色 对于监控项的值数据，如果监控项的值超过指定的“阈值”之一，则默认颜色可以由自定义颜色覆盖。

监控项的值列的特定参数：

Item	选择监控项 从 Zabbix 6.0.4 开始，支持选择非数字监控项。
Time shift	如果需要，请指定时间偏移 您可以在这个字段使用时间后缀 ，允许负值。
Aggregation function	指定要使用的聚合函数： **min** - 显示最小值 **max** - 显示最大值 **avg** - 显示平均值 **sum** - 显示值的总和 **count** - 显示值的数量 **first** - 显示第一个值 **last** - 显示最后一个值 **none** - 显示所有值 (不推荐) 聚合允许显示所选间隔（5 分钟、一小时、一天）的聚合值，而不是所有值 请注意，如果此设置不是“none”，则此列中只能显示数字监控项。
Aggregation interval	指定聚合值的间隔。您可以在此字段中使用时间后缀 。没有后缀的数值将被视为秒 如果聚合函数为“none”，则不显示此字段。

Display	定义应如何显示值： **As is** - 按常规文本显示 **Bar** - 按实心彩色填充条形图显示 **Indicators** - 按分段彩色填充条形图显示 请注意，如果此设置不是“as is”，则此列中只能显示数字监控项。
History	从历史或趋势中获取数据： **Auto** - 自动选择 **History** - 获取历史数据 **Trends** - 获取趋势数据 此设置仅适用于数字数据。非数字数据将始终取自历史记录。
Min	条形图/指示器的最小值。
Max	条形图/指示器的最大值。
Thresholds	指定背景/填充颜色应更改的阈值。保存时，列表将按升序排序 请注意，如果使用阈值，则此列中只能显示数字项。

文本列的特定参数：

Text	输入要显示的字符串。可能包含主机和资产宏(/manual/appendix/macros/supported_by_location).

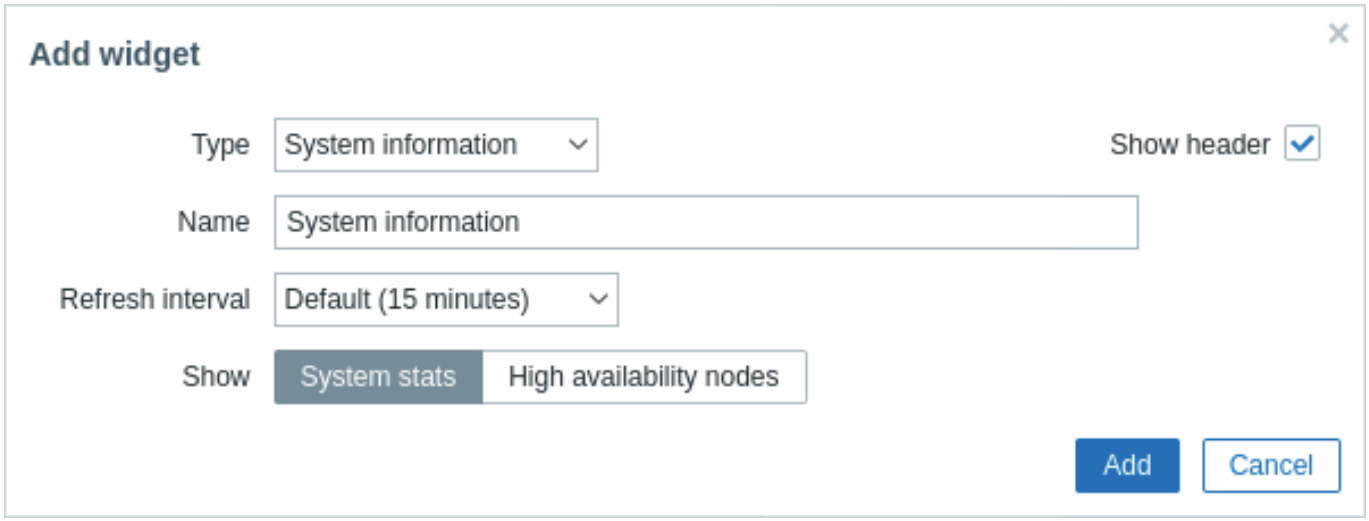
20 系统信息

概述

这个组件显示的信息与报告 →系统信息中的信息类似，但是单个仪表盘组件只能显示系统统计信息或高可用性节点（不能同时显示）。

配置

选择 *System information* 作为类型：



通用 的配置参数对所有组件都是可用的。

21 触发器概览

概述

在触发器概览组件中，你可以显示一组主机的触发状态。

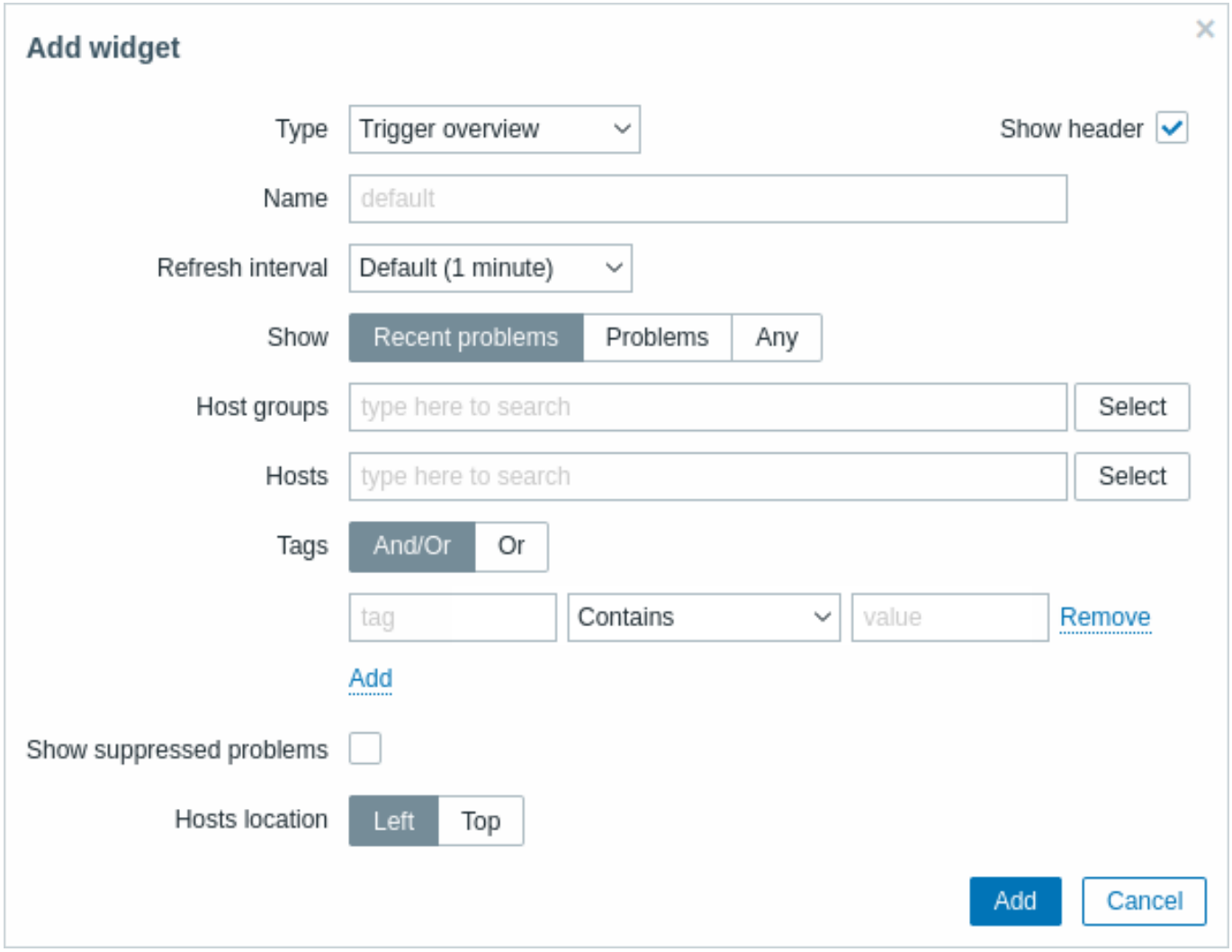
- 触发器状态显示为彩色块（问题触发器的颜色取决于问题严重程度的颜色，可以通过问题更新 screen) 调整)。注意，最近的触发器变化（2 分钟内）的彩色块会呈现为闪烁状态。
- 蓝色的向上和向下箭头表示有依赖关系的触发器。鼠标移动时，会显示依赖关系的细节。
- 一个复选框图标表示已确认的问题。触发器的所有问题或已解决的问题必须被确认才能显示该图标。

点击触发器块，可以访问与该触发器的问题事件、问题确认屏幕、触发器配置、触发器 URL 或简单的图表/最新值列表有关的相关链接。

注意，默认情况下显示 50 条记录（可在管理 → 一般 →*GUI*中配置，使用表格的最大列数和行数选项）。如果存在的记录多于配置显示的记录，则在表的底部显示一条信息，要求提供更具体的过滤标准。不会分页显示。请注意，首先应用此限制，然后再对数据进行任何进一步过滤，例如通过标签。

配置

选择触发器概览作为配置类型：



通用 的配置参数对所有组件都是可用的，您还可以设置下列专属配置项：

显示	按问题状态过滤： 最近的问题 - 显示未解决和最近解决的问题（默认） 问题 - 显示未解决的问题 任何 - 显示所有事件的历史
主机组	选择主机组。这个字段支持自动补全，输入一个组的名称在下拉框选择你需要的主机组名称。
主机	选择主机。这个字段支持自动补全，输入一个主机的名称在下拉框选择你需要的主机名称。单击 “x” 删除对应主机。

标签	指定标签以限制小部件中显示的监控项和触发器数据的数量。可以包括也可以排除特定的标签和标签值。可以设置多个条件，标签名称匹配始终区分大小写。 可以通过以下运算符对每个条件进行筛选： **Exists** - 存在特定的标签名称 **Equals** - 存在特定的标签名称和值（区分大小写） **Contains** - 输入的字符串存在特定的标签名称（通配，不区分大小写） **Does not exist** - 排除存在特定的标签名称 **Does not equal** - 排除存在特定的标签名称和值（区分大小写） **Does not contain** - 排除输入的字符串存在特定的标签名称（通配，不区分大小写） 支持两种运算方式： **And/Or** - 所有条件同时满足，标签名相同的条件将按 Or 条件分组 **Or** - 满足任意条件即可
显示抑制的问题	选中复选框显示因为主机维护状态而被抑制（未显示）的问题。
主机位置	选择主机位置 - 左侧或顶部。

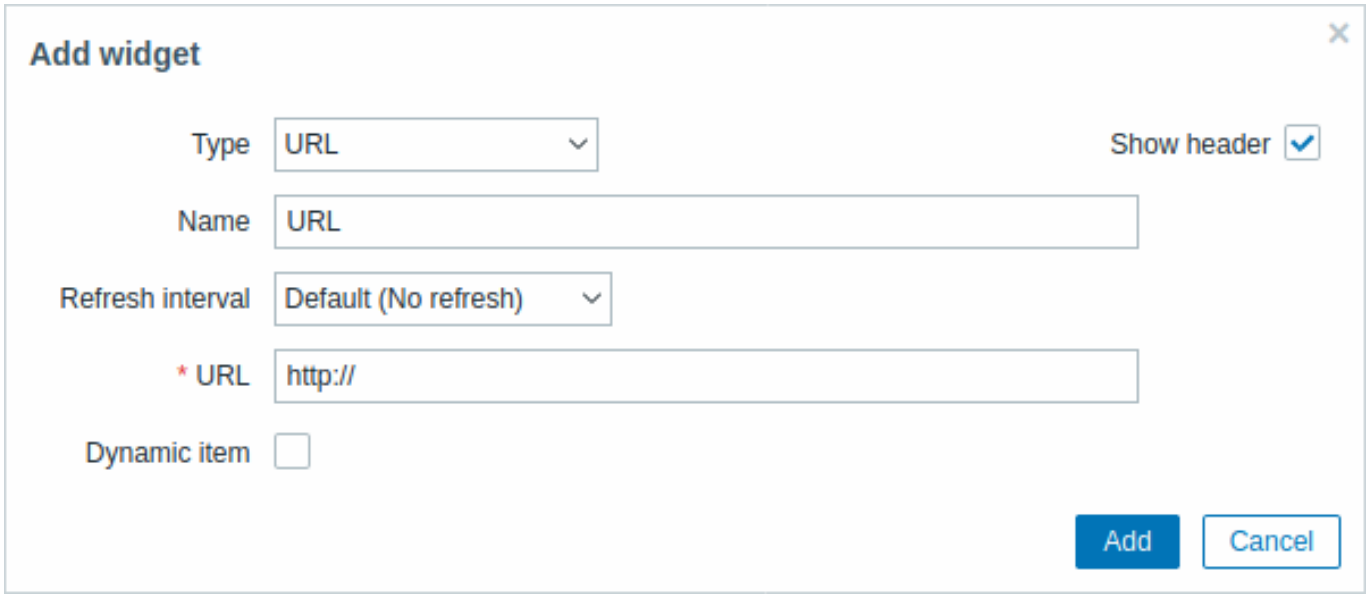
22 网址

概述

这个组件显示从指定的网址获取的内容。

配置

选择 *URL* 作为配置类型：



通用 的配置参数对所有组件都是可用的，您还可以设置下列专属配置项：

URL	输入要显示的 URL。 从 Zabbix 4.4.8 版本开始可以使用相对路径。 支持 {HOST.*} 宏。
动态监控项	设置为根据所选主机显示不同的 URL 内容。 在 URL 中使用 {HOST.*} 宏时可生效。

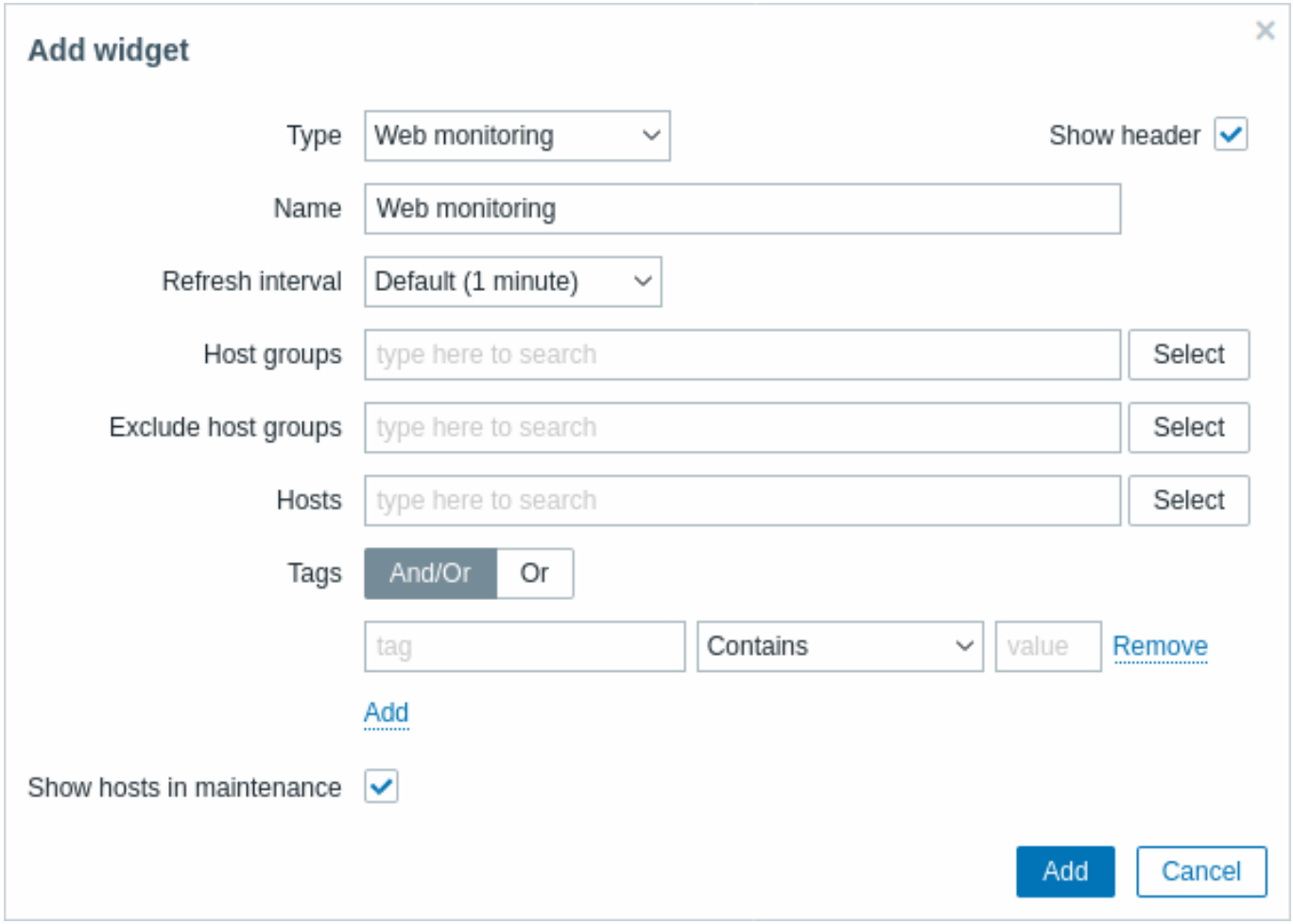
Attention:
如果是通过 HTTPS 访问 Zabbix 前端的，浏览器可能无法访问包含在部件中的 HTTP 页面。

23 Web 监控

概述

这个小部件显示 Web 拨测监视场景的状态摘要。

配置



Note:
在用户没有权限访问某些组件元素的情况下，该元素的名称在组件的配置中会显示为不可访问。这会导致无法访问的监控项、无法访问的主机、无法访问的组、无法访问的拓扑图和无法访问的图表出现，而不是显示该元素的”真实”名称。

除了所有小部件通用 的配置参数之外，您还可以设置下列专属配置项：

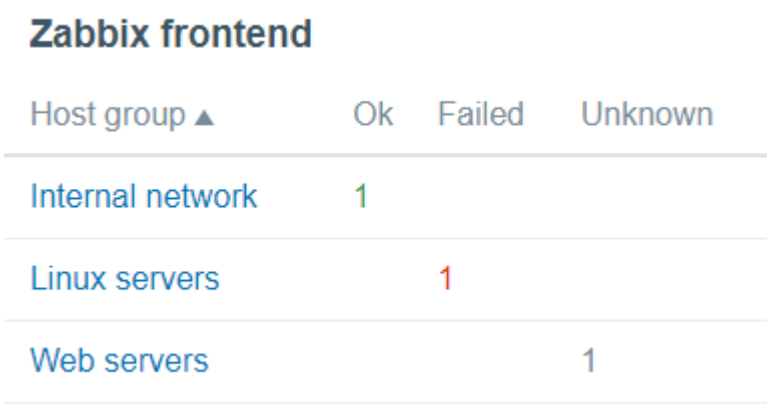
参数	说明
主机组	输入要在组件中显示的主机组。这个字段支持自动补全，输入一个组的名称在下拉框选择你需要的主机组名称。 指定一个父级主机组意味着同时选择所有嵌套的主机组。 来自这些主机组的主机数据将被显示在组件中。如果没有输入主机组，所有的主机组将被显示。
排除主机组	输入要从组件中隐藏的主机组。这个字段支持自动补全，输入一个组的名称在下拉框选择你需要的主机组名称。 指定一个父级主机组意味着同时选择所有嵌套的主机组。 来自这些主机组的主机数据将不会显示在组件中。例如主机 001, 002, 003 属于 A 组，主机 002, 003 也属于 B 组。如果我们选择同时显示 A 组和排除 B 组，只有主机 001 的数据将显示在仪表板中。
主机	输入要在组件中显示的主机。这个字段支持自动补全，输入一个主机的名称在下拉框选择你需要的主机名称。 如果未输入主机，将显示所有主机。

参数	说明
标签	可以通过包括以及排除特定的标签和标签值的方法，指定标签以限制组件中同时显示的 Web 监控项的数量。标签名称匹配区分大小写。 可以通过以下运算符对每个条件进行筛选： **Exists** - 存在特定的标签名称 **Equals** - 存在特定的标签名称和值（区分大小写） **Contains** - 输入的字符串存在特定的标签名称（通配，不区分大小写） **Does not exist** - 排除存在特定的标签名称 **Does not equal** - 排除存在特定的标签名称和值（区分大小写） **Does not contain** - 排除输入的字符串存在特定的标签名称（通配，不区分大小写） 支持两种运算方式： **And/Or** - 所有条件同时满足，标签名相同的条件将按 Or 条件分组 **>Or** - 满足任意条件即可
显示维护中的主机	加入正在维护状态的主机。

Web monitoring widget

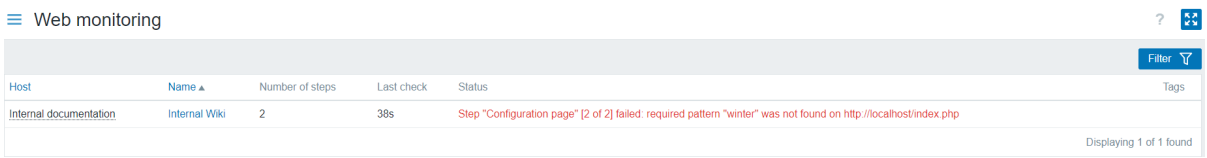
Once you have completed the configuration, you might like to see the widget with the data it displays. To do it, go to *Monitoring* → *Dashboards*, click on the name of a dashboard where you created the widget.

In this example, you can see the widget named ”Zabbix frontend” displaying the status of the web monitoring for three host groups: ”Internal network,” ”Linux servers,” and ”Web servers.”

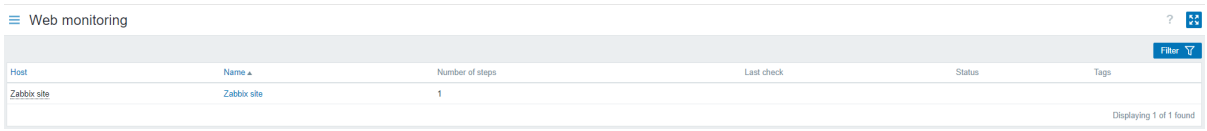


A web monitoring widget displays the following information:

- a name of a widget; below it, there are four columns:
  - *Host group* - displays a list of host groups that contain hosts having web scenarios configured;
  - *Ok* - displays a number of web scenarios (in green color) when two conditions are observed:
    - Zabbix has collected the latest data for a web scenario(s);
    - all steps that were configured in a web scenario are in ”Ok” *Status*.
  - *Failed* - displays a number of web scenarios (in red color), which have some failed steps:
    - click on the host name, and it will open a new window; the *Status* column provides detailed information (in red color) on the step where Zabbix failed to collect the data; and also,
    - gives a hint for the parameter that has to be corrected in the configuration form.



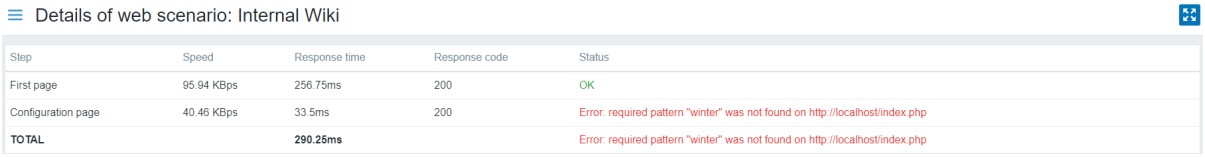
- *Unknown* - displays a number of web scenarios (in grey color) for which Zabbix has neither collected data, nor has an information about the failed steps.



Viewing the status and data

Clickable links in the widget allow to easily navigate and quickly acquire a full information on each web scenario. Thus, to view:

- the Status of a web scenario, click on the name of a host group.
- more detailed statistics, click on the scenario name. In this example it is ”Zabbix frontend”.
- the details in the case of *Failed* status, click on a host group name; in the window that opens, click on a web scenario name in the *Name* column; it will open more detailed information on the configured steps for which Zabbix failed to collect the data.



Now, you can return to the web scenario configuration form and correct your settings.

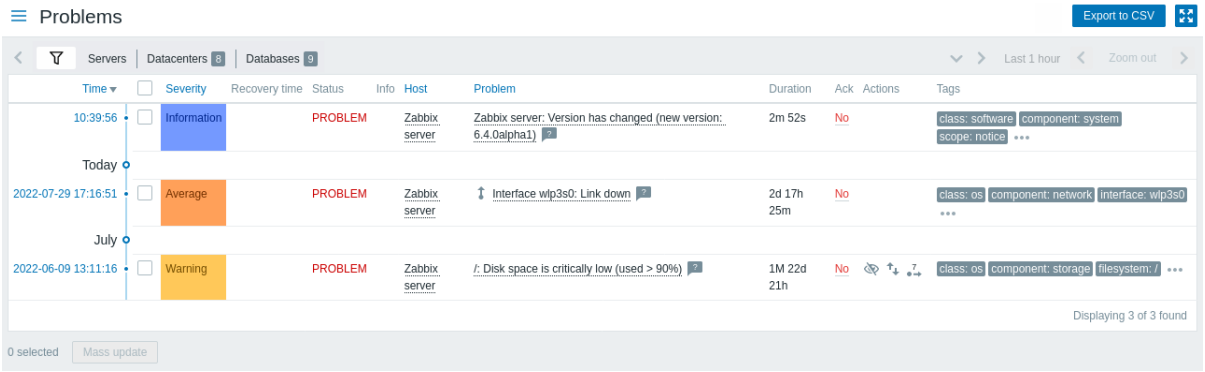
To view the details in the case of *Unknown* status, you can repeat the same steps as explained for *Failed*.

Attention:
At the first monitoring instance, a web scenario is always displayed in *Unknown* state, which is switched to *Failed* or *Ok* state right after the first check. In the case when a host is monitored by the proxy, the status change occurs in accordance with the data collection frequency configured on the proxy.

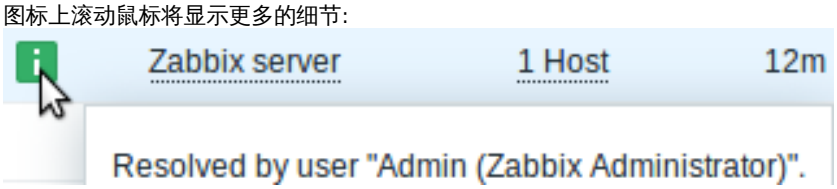
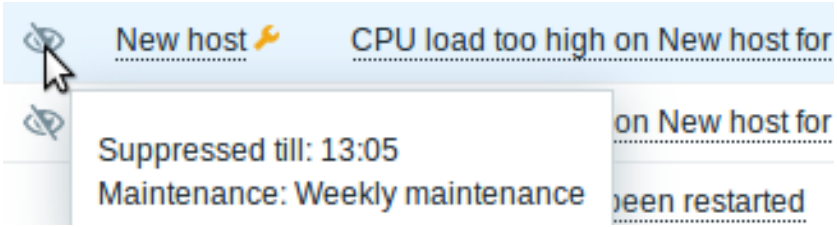
2 问题

概述

在 监测 → 问题中，你可以看到当前存在哪些问题。问题指处在” 问题” 状态下的触发器。



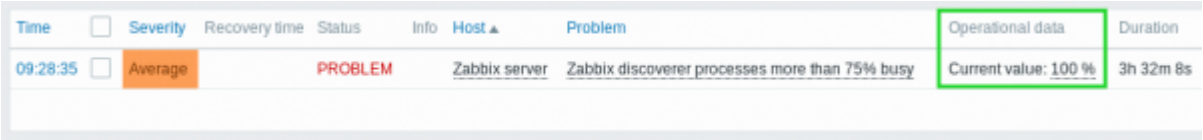
列	描述
时间	显示问题开始时间。
严重性	显示问题严重等级。 问题严重性最初基于基础问题触发器的严重性，但是，在事件发生后，可以使用 问题更新进行更新screen。在问题期间，问题严重性的颜色用作单元格背景。
恢复时间	显示问题解决时间。
状态	显示问题状态: 问题 - 未解决的问题 解决 - 最近解决的问题。您可以使用筛选器隐藏最近解决的问题。 新的和最近解决的问题闪烁 2 分钟。已解决的问题显示时间为 5 分钟。这两个值可以在 管理 → 常规 → *Trigger displaying options*中进行配置.

列	描述
信息	如果问题通过全局关联关闭或在更新问题时手动关闭，则会显示绿色信息图标。在图标上滚动鼠标将显示更多的细节:  如果显示被抑制的问题，则会显示以下图标 (请参阅筛选器中的 显示被抑制的问题选项)。将鼠标滚动到图标上将显示更多详细信息： 
主机	显示问题主机。
问题	显示问题名称。 问题名称基于基础问题触发器的名称。 触发器名称中的宏在问题发生时被解析，解析后的值不再更新。 注意可以在问题名称后面加上运营数据 显示一些最新的监控项值。 单击问题名称将显示事件菜单. 将鼠标悬停在问题名称后面的 图标上，将会显示触发器描述 (对于那些有触发器描述的问题)。
运行数据	Operational data 展示了监控项最新的值。 如果在触发器级别上配置，运行数据可以是文本宏和监控项值宏的组合。如果没有在触发器级别配置运行数据，则显示表达式中所有监控项的最新值。 仅当在过滤器中为显示运行数据选择 单独时才会显示此列。
持续时间	显示问题持续时间。 参阅: Negative problem duration
确认	显示问题的确认状态： 是 - 绿色文本表示问题已得到确认。如果一个问题的所有事件都被确认，则认为该问题已被确认。 不 - 红色链接表示未确认事件的。 如果您单击该链接，将跳转到problem update 页面，其中可以对问题执行各种操作，包括注释和确认问题。
动作	有关该问题的活动历史记录使用符号图标显示： - 已执行的动作。还会显示动作数。 - 问题严重性已提高 (例如：信息 → 警告) - 问题严重性已降低 (例如：警告 → 信息) - 问题严重程度已更改，但恢复到原来的级别 (例如：警告 → 信息 → 警告) - 已经采取了动作。动作的数量也会显示出来。 - 已经采取了动作，至少有一次正在进行中。动作的数量也会显示出来。 - 已经采取了动作，至少有一次失败了。动作的数量也会显示出来。 将鼠标滚动到图标上时，将显示包含动作详细信息的弹出窗口。参阅viewing details 来了解弹出窗口中所采取的动作图标的更多信息。
标签	标签 显示 (如果有)。 另外，可以显示来自外部标签系统的标签 (参阅 过程标签选项配置webhooks).

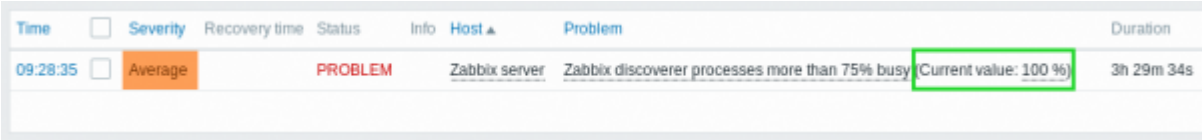
问题的运行数据

可以显示当前问题的运行数据。即最新的监控项值，而不是发生问题时的监控项值。

在 监控 → 问题过滤器中或者在相应的仪表板小部件的配置中，通过选择以下三个选项之一来配置运行数据显示：- *None* - 不显示运行数据 - *Separately* - 在单独的列中显示运行数据



- ·*With problem name* - 运行数据将附加到问题名称和括号中。仅当触发器配置中的 “运行数据” 字段为非空时，才会将运行数据附加到问题名称中。



可以在 “运行数据” 字段中为每个触发器配置运行数据的内容。该字段接受带有宏的任意字符串，最重要的是宏 `{ITEM.LASTVALUE<1-9>}`。此字段中的 `{ITEM.LASTVALUE<1-9>}` 将始终解析为触发器表达式中各监控项的最新值，此字段中的 `{ITEM.VALUE<1-9>}` 将在触发状态更改时解析为监控项值 (即: 变成 Problem，变成 OK，被用户手动关闭或被关联关闭)。

问题持续时间为负

在某些常见情况下，可能会出现持续时间为负数，即问题解决时间早于问题创建时间时，例如：

- ·如果某个主机被代理监视并且发生了网络错误，导致在一段时间内没有接收到来自代理的数据，则服务器将触发 no-data(/host/key) 触发器。当连接恢复时，服务器将从具有过去时间的代理接收项目数据。然后，nodata (/host/key) 问题将被解决，并且会出现问题持续时间为负数。
- ·当恢复问题事件的监控项数据由 Zabbix Sender 发送并且数据时间戳早于问题创建时间，也将显示问题持续时间为负数。

问题持续时间为负数，不会影响SLA 计算 或特定触发器的可用性报告 ；它既不会减少也不会延长问题时间。:::

批量编辑选项

下面的按钮提供了批量编辑选项：

- ·批量更新 - 通过导航到·problem update 页面来更新选择的问题。

要使用此选项，请选中相应问题之前的复选框，然后单击 批量更新按钮。

按钮

右侧的按钮提供以下选项：

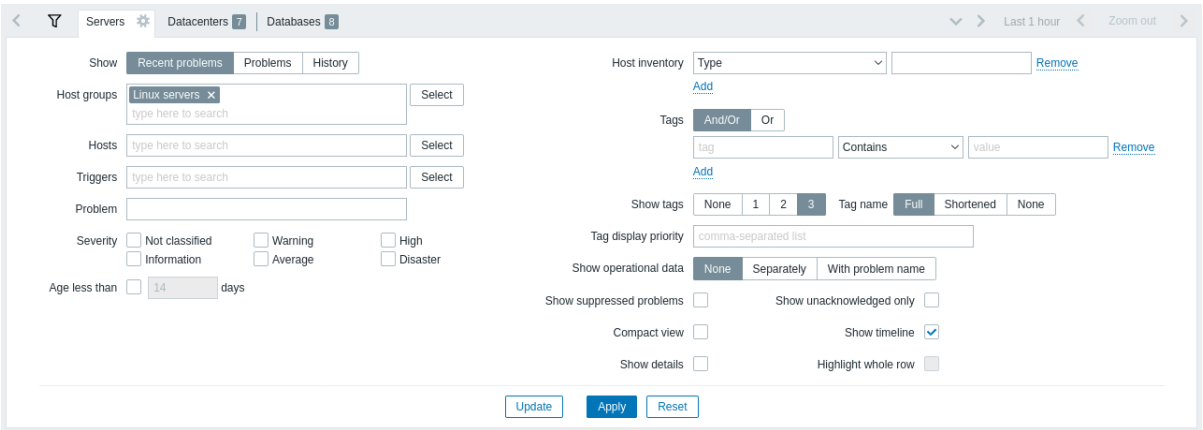
.	.
Export to CSV	将页面所有内容导出到 CSV 文件。

在监视页面介绍了所有通用的查看模式按钮。

使用过滤器

您可以使用过滤器来显示您感兴趣的问题。为了获得更好的搜索性能，使用未解析的宏搜索数据。

过滤器位于表格的上方。过滤器设置可以另存为选项卡，然后通过单击过滤器上方的选项卡快速访问。

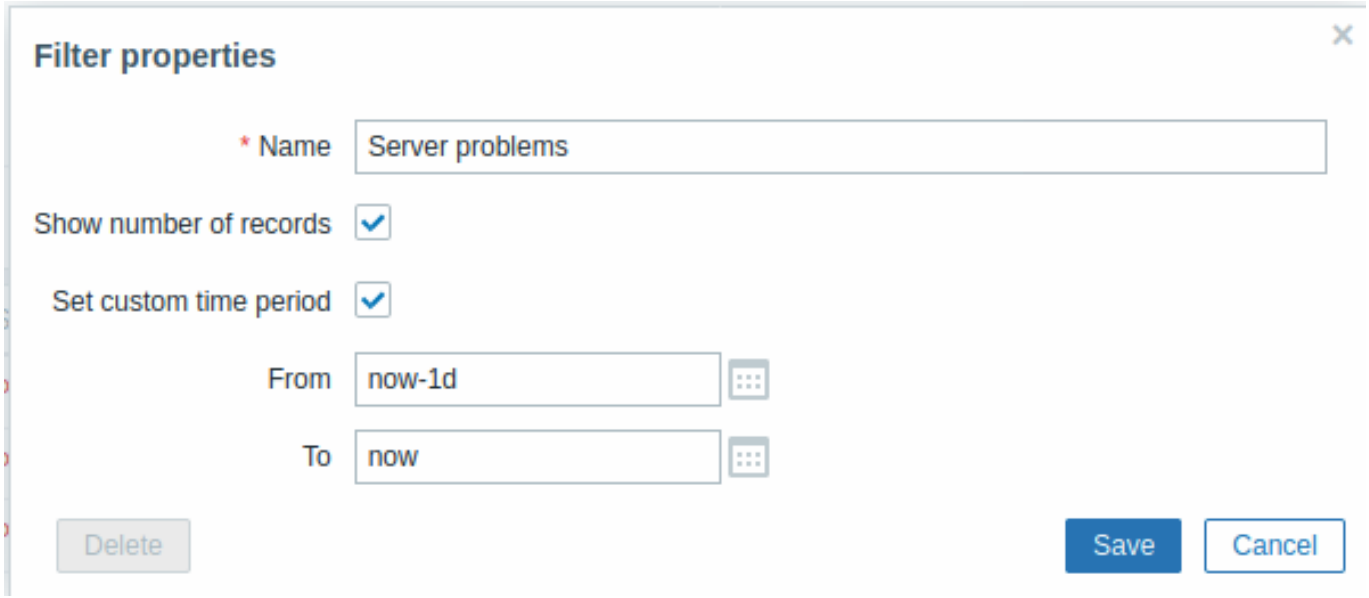


参数	描述
显示	按问题状态筛选： 显示最近问题 - 未解决和最近解决的问题 (默认) 问题 - 显示未解决的问题 历史 - 显示所有事件的历史记录
主机组	筛选一个或多个主机组。 指定父主机组将选择所有包含的主机组。
主机	筛选一个或多个主机。
触发器	筛选一个或多个触发器。
问题	按问题名称筛选。
严重性	按触发器（问题）严重性进行筛选。
持续时间小于	按问题的持续时间进行筛选。
主机资产记录	按类型和值进行筛选。
标签	按事件标签 名称和值筛选。可以包含和排除特定标签和标签值。可以设置几个条件。标签名称匹配始终区分大小写。 每个条件都有多个运算符可用： 存在 - 包括指定的标签名称 等于 - 包括指定的标签名称和值（区分大小写） 包含 -包含指定的标签名称，其中标签值包含输入的字符串 (子字符串匹配，不区分大小写) 不存在 - 排除指定的标签名称 不等于 - 排除指定的标签名称和值（区分大小写） 不包含 - 不包含指定的标签名称，其中标签值包含输入的字符串 (子字符串匹配，不区分大小写) 条件有两种计算类型： 与/或 - 必须满足所有条件，具有相同标签名称的条件将根据或条件分组 或 - 只需要满足一个条件 过滤后，此处指定的标签将首先显示问题，除非被 * 标签显示优先级 (见下文) 列表覆盖。
显示标签	选择显示的标签数： 没有 - 无标签列在 监控 → 问题 **1** - 标签列包含 1 个标签 **2** - 标签列包含 2 个标签 **3** - 标签列包含 3 个标签 要查看问题的所有标签将鼠标悬停在 3 个点图标上。
标签名	选择标签名称显示模式： 完整 - 标签名称和值将完整显示 缩短显示 - 标签名称被缩短为 3 个符号; 标签值将全部显示 没有 - 只显示标签值; 没有名字
标签显示优先级	输入问题的标签显示优先级，作为逗号分隔的标签列表 (例如: 服务、许可、申请)。只应使用标签名称，不显示标签值。此列表的标签将始终首先显示，并覆盖按字母顺序的自然排序。
显示运行数据	选择显示 [运行数据] 模式 (#operational_data_of_problems): 无 - 不显示运行数据 单独的 - 在单独的列里显示运行数据 使用问题名称 - 将运行数据附加到问题名称，使用括号表示运行数据。
显示隐藏的问题	标记复选框以显示由于主机维护而被隐藏（未显示）的问题。
紧凑视图	标记复选框以启用精简视图。
显示详情	标记该复选框以显示问题的基础触发器表达式。如果标记了 紧凑视图复选框，则禁用。
仅显示未确认	标记该复选框以仅显示未确认的问题。
显示时间线	标记该复选框以显示可视时间线和分组。如果标记了紧凑视图复选框，则禁用。
整行高亮	标记该复选框以突出显示未解决的问题的整行。问题严重性颜色用于突出显示。 仅当 紧凑视图复选框在标准蓝色和深色主题中标记时才启用。整行高亮在高对比度主题中不可用。

“收藏夹” 筛选器的选项卡

常用的过滤器参数集可以保存在选项卡中。

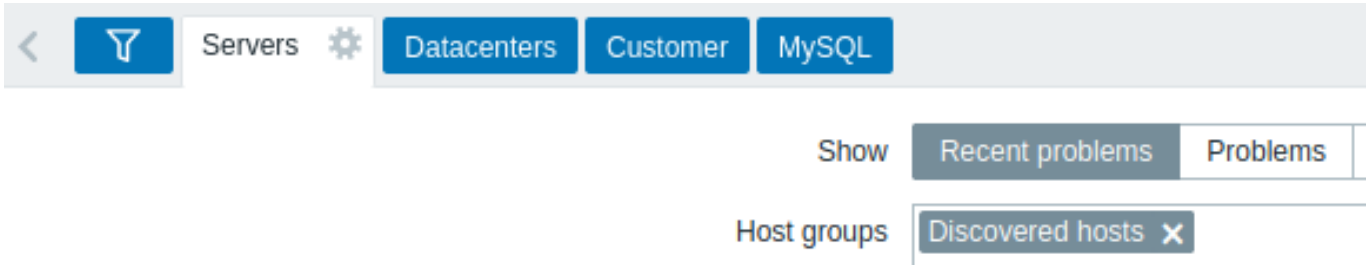
若要存储一组新的过滤器参数，请打开主选项卡并配置过滤器设置，然后按 另存为按钮。在新的弹出窗口中，定义 过滤器属性。



参数	描述
名称	要在选项卡列表中显示的过滤器的名称。
显示记录数	如果您希望问题的数量显示在选项卡名称旁边，请勾选。
设置自定义时间段	检查以设置此过滤器集的特定默认时间段。如果设置了，则只能通过更新过滤器设置来更改此选项卡的时间段。对于没有自定义时间段的选项卡，时间范围可以通过按右上角的时间选择按钮来更改 (按钮名称取决于所选的时间间隔: 本周、过去 30 分钟、昨天等). 此选项仅适用于 监视 → 问题中的过滤器。
从/到	时间段 以绝对 (`Y-m-d H:i:s`) 或相对时间语法 (`now-1d`) 定义开始和结束, 设置自定义时间段选中则可用。

保存后，过滤器将创建为命名过滤器选项卡并立即激活。

要编辑现有过滤器的筛选属性，请按活动选项卡名称旁边的齿轮符号。



注意：- ·要隐藏过滤器，请按当前选项卡的名称。再次按下活动选项卡名称打开过滤器。- ·可以通过拖放来重新排列过滤器选项卡。- ·支持键盘导航：使用箭头在选项卡之间切换，按下 *Enter* 打开。- ·按右上角的向下箭头图标将打开已保存过滤器选项卡的完整列表作为下拉菜单。- ·可以通过拖放重新排列过滤器选项卡。- ·如果已保存过滤器的设置已更改（但未保存），过滤器名称后会显示一个绿点。要根据新设置更新过滤器，请单击显示的 更新按钮，而不是 另存为按钮。- ·当前过滤器设置被记住在用户配置文件中。当用户再次打开页面时，过滤器设置将保持不变。

若要共享过滤器，请复制活动过滤器的 URL 并将其发送给其他人。打开此 URL 后，其他用户将能够将这组参数作为永久过滤器保存在其 Zabbix 帐户中。
另请参见：页面参数。:::

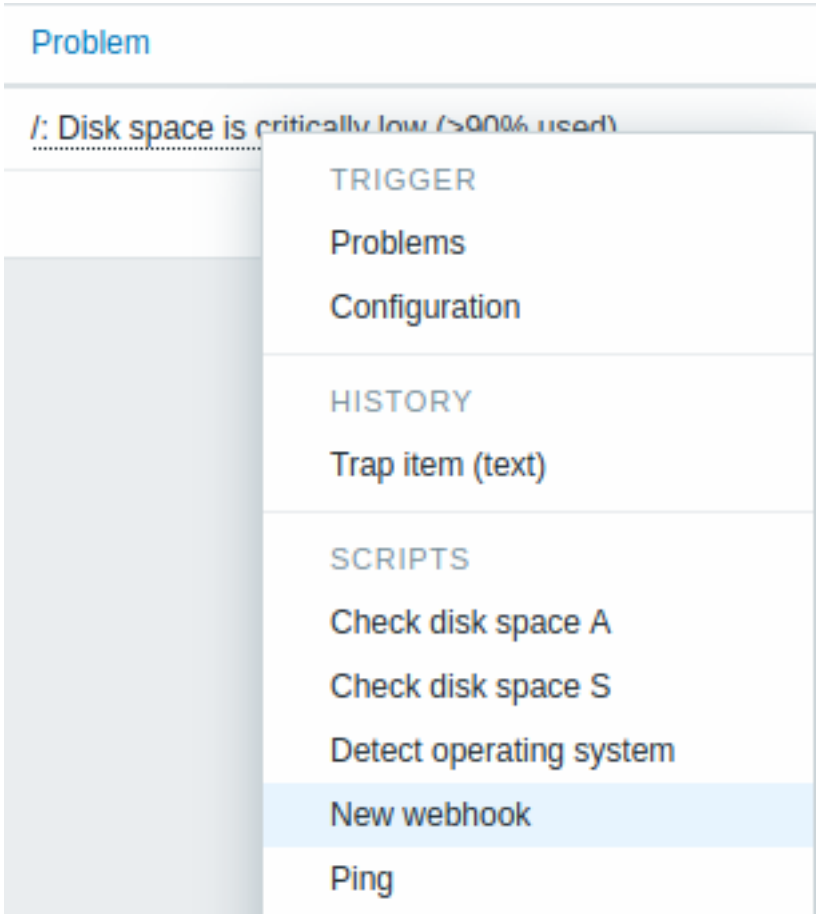
过滤器按钮

.	.
Apply	应用指定的过滤器条件（不保存）。
Reset	重置当前过滤器并返回当前选项卡的已保存参数。在主选项卡上，这将清除过滤器。
Save as	在新选项卡中保存当前的过滤器参数。只在主选项卡上可用。

.	.
Update	将选项卡参数替换为当前指定的参数。在主选项卡上不可用。

活动菜单

单击问题名称将显示事件菜单：

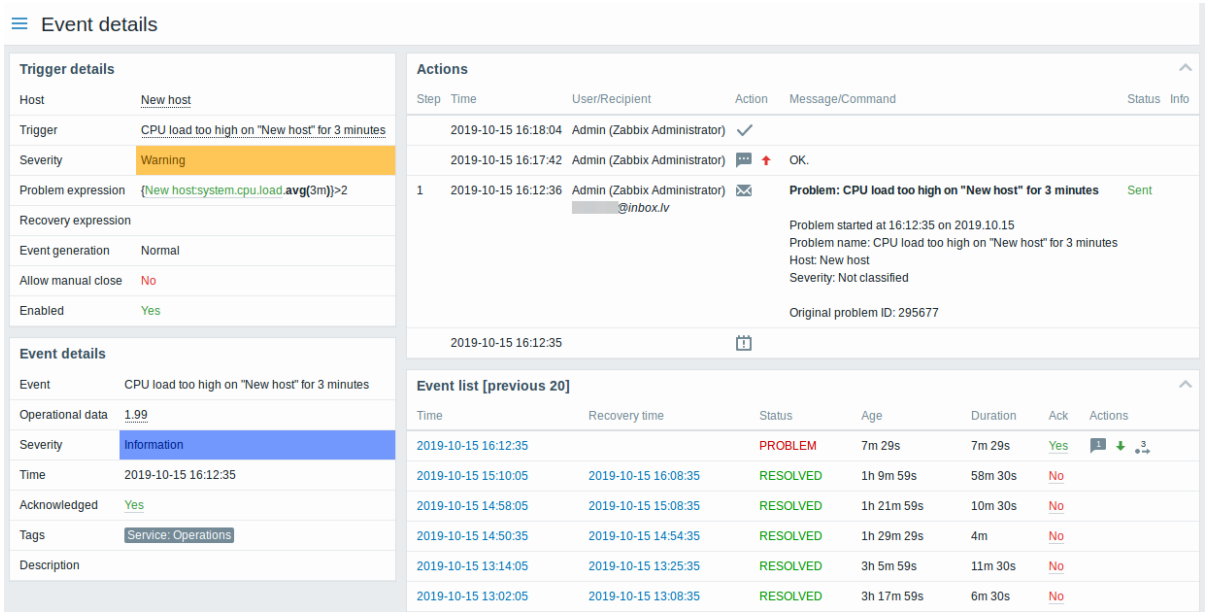


事件菜单允许：

- 过滤问题触发器
- 访问触发器配置
- 访问基础项目的简单图表/项目历史记录
- 访问问题的外部工单（如果已配置，请参阅配置webhooks) 时的选项
- 执行全局脚本(这些脚本需要将其作用域定义为' 手动事件操作')。此功能对于运行用于在外部系统中管理问题工单的脚本可能很方便。

查看详细信息

在 监视 → 问题异常开始和恢复的时间都有链接，单击链接可以打开更多事件细节。



请注意触发器和问题时间的严重性是有区别的。对于问题事件，它已使用 _“更新问题”_屏幕进行了更新。

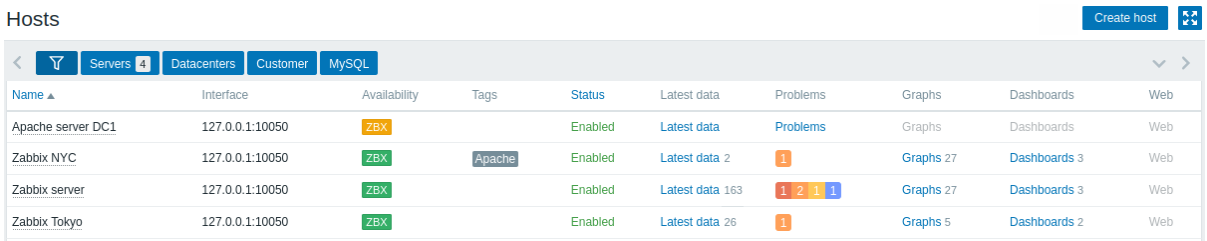
在操作列表中，以下图标用于表示活动类型：

- · -生成的问题事件
- · -消息已发送
- · - 确认问题事件
- · -未确认的问题事件
- · -添加了注释
- · -问题严重性已增加（例如：信息 → 警告）
- · -问题严重性已降低（例如：警告 → 信息）
- · - 问题严重性已更改，但已恢复到原始级别（例如：警告 → 信息 → 警告）
- · - 已执行远程命令
- · -问题事件已恢复
- · - 问题已手动关闭

3 主机

概述

监控 → 主机部分显示受监控主机的完整列表，其中包含有关主机接口、可用性、标签、当前问题、状态（启用/禁用）的详细信息，以及轻松导航到主机的最新数据、问题历史记录、图表、仪表板的链接和 WEB 场景。



列	描述
名称	可见的主机名。单击名称将显示主机菜单. 名称后面的橙色扳手图标 表示此主机正在维护中。 单击列标题可按名称升序或降序对主机进行排序。
接口	显示主机的主界面。

列	描述
可用性	每个已配置接口的主机可用性 图标仅表示已配置的接口类型 (Zabbix 代理、SNMP、IPMI、JMX) 如果将鼠标放在图标上，则会显示此类型所有接口的弹出列表，其中包含每个接口的详细信息、状态和错误。 对于没有接口的主机，该列为空。 一种类型的所有接口的当前状态由相应的图标颜色显示： 绿色 - 所有接口可用 黄色 - 至少一个接口可用同时至少一个不可用; 其他可以具有任何值，包括' 未知' 红色 - 没有可用的接口 灰色 - 至少一个接口未知 (无可用) 请注意，活动的 Zabbix 代理项目不会影响主机可用性。
标签	主机和所有链接模板的标记 ，宏未解析。
状态	主机状态 - 启用或 禁用。 单击列标题可按状态以升序或降序对主机进行排序。
最新数据	单击该链接将打开 监测- 最新数据页面，其中包含从主机收集的所有最新数据。 具有最新数据的监控项数以灰色显示（自 Zabbix 6.0.5 起显示）。
问题	按严重性排序的打开的主机问题数。正方形的颜色表示问题的严重性。方块上的数字表示给定严重性下的问题数。 单击该图标将打开当前主机的 监测 - 问题页面。 如果主机没有问题，则指向此主机的问题部分的链接将显示为文本。 使用过滤器选择是否应包括隐含的问题（默认情况下不包括）。
图形	单击该链接将显示主机配置的图形。图形的数量以灰色显示。 如果主机没有图形，则禁用链接（灰色文本）并且不显示任何数字。
仪表板	单击该链接将显示主机配置的仪表板。仪表板的数量以灰色显示。 如果主机没有仪表板，则链接将被禁用（灰色文本），并且不显示任何数字。
WEB 监测	单击该链接将显示为主机配置的 Web 监测方案。Web 监测方案的数量以灰色显示。 如果主机没有 Web 监测方案，则链接将被禁用（灰色文本），并且不显示任何数字。

按钮

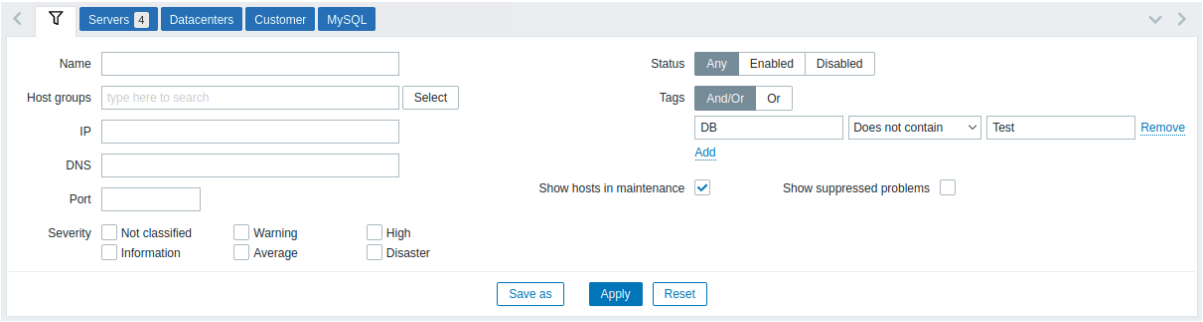
创建主机允许创建新主机。此按钮仅对管理员和超级管理员用户可用。

监视页介绍了所有部分通用的视图模式按钮。

使用过滤器

您可以使用过滤器仅显示您感兴趣的主机。为了获得更好的搜索性能，使用未解析的宏搜索数据。

过滤器位于表的上方。可以按名称，主机组，IP 或 DNS，接口端口，标签，问题严重性，状态（启用/禁用/任何）过滤主机；您还可以选择是否显示被抑制的问题和当前正在维护的主机。

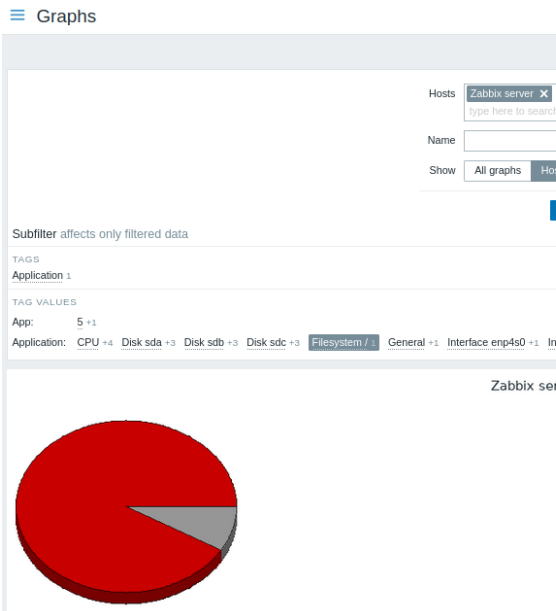


参数	描述
名称	按可见主机名过滤。
主机组	按一个或多个主机组进行过滤。 指定父主机组将隐式选择所有嵌套的主机组。
IP 地址	按照 IP 地址过滤。
DNS	按 DNS 名称过滤。
端口	按端口号过滤。
严重性	按问题严重性过滤。默认情况下，将显示所有严重性的问题。如果未禁止显示，则会显示问题。
状态	按主机状态过滤。

参数	描述
标签	按主机标记名称和值过滤。主机可以按主机级标签以及所有链接模板（包括父模板）中的标签进行过滤。 可以包含和排除特定标签和标签值。可以设置几个条件。标签名称匹配始终区分大小写。 每个条件都有多个运算符可用： 存在 - 包括指定的标签名称 等于 - 包括指定的标签名称和值（区分大小写） 包含 - 包括指定的标签名称，其中标签值包含输入的字符串（子字符串匹配，不区分大小写） 不存在 - 排除指定的标签名称 不等于 - 排除指定的标签名称和值（区分大小写） 不包含 - 排除标签值包含输入字符串的指定标签名称（子字符串匹配，不区分大小写） 有两种计算类型条件： 与/或 -必须满足所有条件，具有相同标签名称的条件将根据或条件分组 或 - 只需要满足一个条件
显示维护中主机	标记该复选框以显示处于维护状态的主机（默认显示）。
显示处理的问题	标记复选框以显示因主机维护而被抑制（未显示）的问题。

1 图表

概述



可以通过单击相应主机的图形。从 监控→主机访问主机图。任意自定义图形 以及任意简单图形。

图表的排序方式为：

- ·图形名称（自定义图形）
- ·项目名称（简单图形）

禁用主机的图表也可访问。

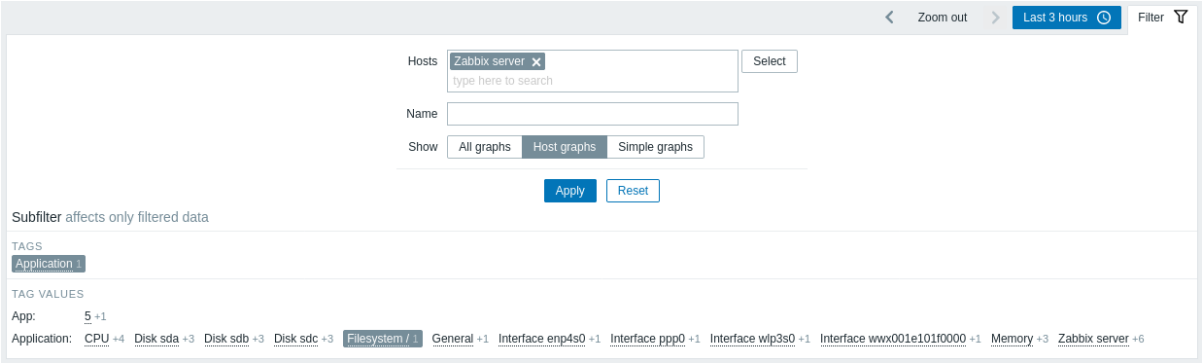
时间段选择器

记下图表上方的时间段选择器。它允许通过一次鼠标单击选择经常需要的时间段。

另请参见：时间段选择器

使用过滤器

为了查询特定的图像，通过使用过滤起来选中它。该过滤器允许一次指定一个主机（主机是必需的），然后通过从列表中选择或通过按图表名称模式的方法来快速搜索指定主机图表。



如果在过滤器里没有选择主机，则不会显示图形。

使用子筛选器

子筛选器可用于快速一键访问相关图形。子过滤器从主过滤器自主运行- 结果立即过滤，无需单击主过滤器中的 应用。

请注意，子筛选器仅允许从主筛选器进一步修改筛选。

与主筛选器不同，子筛选器与每个表刷新请求一起更新，以始终获取可用筛选选项及其计数器编号的最新信息。

子筛选器显示 可单击的链接允许根据公共实体（标记名称或标记值）筛选图形。一旦单击实体，图形就会立即被过滤；所选图元将以灰色背景突出显示。若要删除筛选，请再次单击该实体。若要将其他实体添加到筛选结果中，请单击另一个实体。

水平方向显示的实体数限制为 100 个。如果还有更多，则在末尾显示一个三点图标；它是可点击的。垂直列表（比如带值的标签）限制为 20 个。如果有更多，将显示一个三点图标：它是可点击的。

每个可单击实体旁边的数字表示它在主筛选器结果中的图形数。

选择一个图元后，将显示具有其他可用图元的数字，并带有加号，指示可以添加到当前选定内容中的图形数。

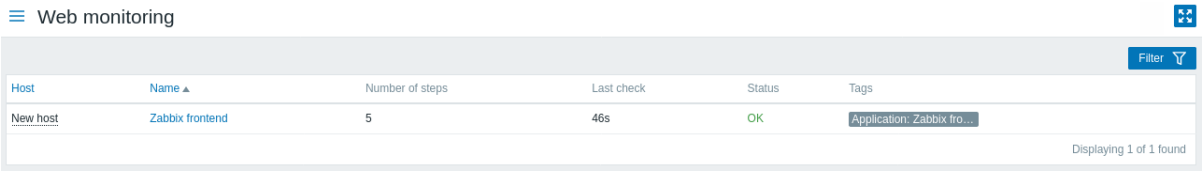
按钮

监视页上介绍了所有部分通用的视图模式按钮。

2 Web 场景

概述

主机Web 场景 (/manual/web_monitoring) 信息可以通过单击相应主机的监测 → 主机访问。



对于处于禁用状态的主机，其 WEB 场景的监控数据依旧可以访问。但请注意，这类主机的名称将会是红色字体。

每页所能展示的最多场景的数量，取决于用户设置中的 每页行数。

默认情况下，仅显示过去 24 小时内的值。引入此限制的目的是缩短大页面最新数据的初始加载时间。你可以通过在 管理 → 常规菜单中更改 最长可显示历史期限参数的值来延长此时间。

方案名称链接到有关它的更详细的统计信息：



使用过滤器

该页显示所选主机的所有 Web 监测的列表。若要查看其他主机或主机组的 Web 监测而不返回到 监测 → 主机页面，请在筛选器中选择该主机或组。你还可以根据标签筛选方案。

按钮

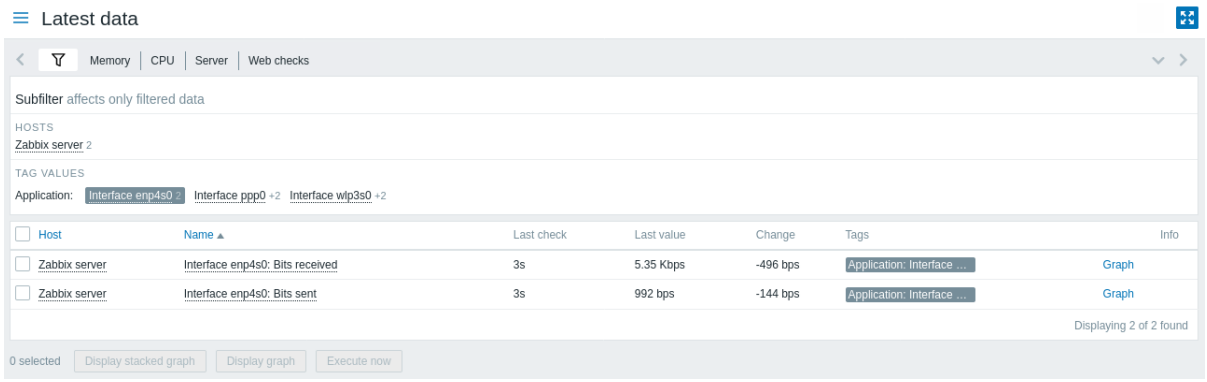
监视页上介绍了所有部分通用的显示模式按钮。

4 最新数据

概述

在本节中，您可以查看按项目收集的最新值。

图表也可用于展示项目值。



本节包含：

- 过滤器 (默认情况下处于折叠状态)
- 子过滤器 (从不折叠)
- 项目列表

显示项目及其名称、自上次检查以来的时间、最后一个值、更改金额、标记以及指向项目值的简单图形/历史记录的链接。

单击项目名称将打开项目菜单，其中包含指向可用图形和项目配置的链接。

项目列表中的标签是可点击的。如果单击某个标记，则此标记将在子筛选器中启用。项目列表现在显示与此标记和子筛选器中以前选择的任何其他标记对应的项目。请注意，以这种方式筛选项目后，列表中的标记将不再可点击。基于标签的进一步修改（例如，删除，添加另一个过滤器）必须在子过滤器中完成。

将显示最后一个值列中显示的值应用了单位换算和值映射。要查看原始数据，请将鼠标悬停在显示值上。

如果某个项目有错误，例如：已变得不受支持，则 信息 列中将显示一个信息图标。将鼠标悬停在图标上可查看详细信息。

带有问号的图标将显示在具有说明的所有项目的项目名称旁边。将鼠标悬停在此图标上，可查看包含项目描述的工具提示。

将鼠标悬停在此图标上，可查看包含项目描述的工具提示。如果项目所属的主机处于维护状态，则在主机名称后面会显示一个橙色扳手图标

注意：已禁用主机的名称显示为红色。禁用主机的数据（包括图形和项目值列表）也可以在 最新数据中访问。

默认情况下，只显示最近 24 小时内的值。引入此限制的目的是缩短大页面最新数据的初始加载时间。在管理 → *[常规](/manual/web_interface/frontend_sections/administration/general#gui)* 中更改最大历史记录显示周期参数的值来延长此时间段。

:: noteimportant 对于更新频率为 1 天或以上的项目，将永远不会显示更改量（使用默认设置）。此外，在这种情况下，如果超过 24 小时前收到最后一个值，则根本不会显示该值。:::

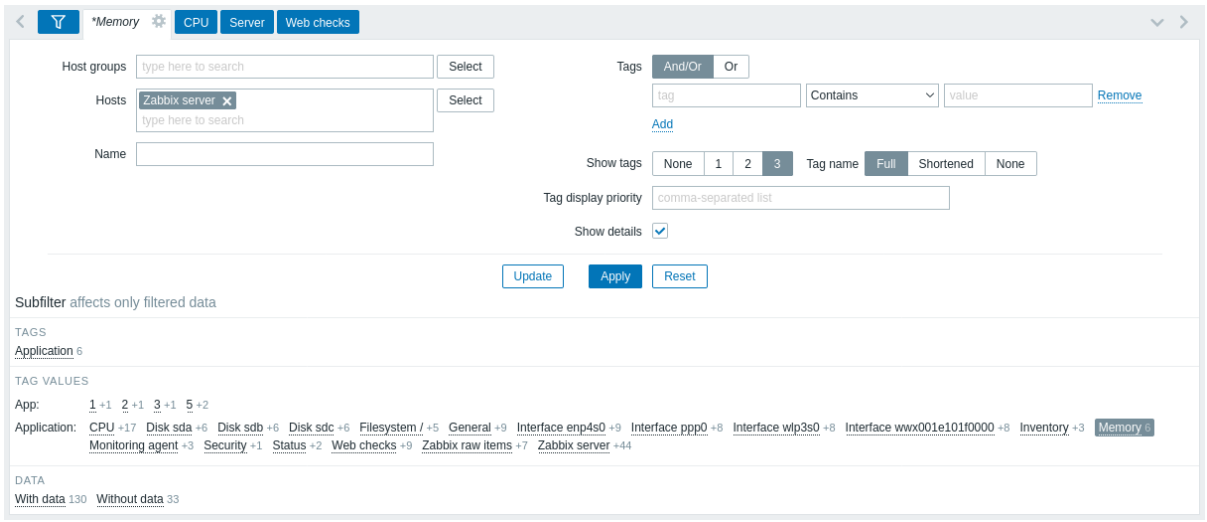
按钮

监控页介绍了所有部分通用的监控模式按钮。

使用过滤器

您可以使用筛选器仅显示您感兴趣的项目。为了获得更好的搜索性能，搜索带有未解析的宏的数据。

过滤器图标 位于表和子过滤器的上方。单击它可以展开过滤器。



过滤器允许按主机组、主机、项目名称、标记和其他设置缩小列表范围。在过滤器中指定父主机组将隐式选择所有嵌套主机组。有关按标记筛选的详细信息，请参阅监测 -> 问题 。

显示详细信息允许扩展为监控项显示的信息。将显示刷新间隔、历史记录和趋势设置、监控项类型和监控项错误（支持/不支持）等详细信息。

保存过滤器

最喜欢的过滤器设置可以保存为选项卡，然后通过单击过滤器上方的相应选项卡快速访问。

更多详情请参考保存过滤器。

使用子过滤器

子筛选器可用于快速一键式访问相关监控项组。子过滤器从主过滤器自主运行 - 结果立即过滤，无需单击主过滤器中的 “应用”。

请注意，子过滤器仅允许从主过滤器进一步修改筛选。

与主过滤器不同，子过滤器与每个表刷新请求一起更新，以始终获取可用过滤选项及其计数器编号的最新信息。

子过滤器显示 可单击的链接，允许根据公共实体（主机、标记名称或标记值）过滤监控项。单击实体后，将立即过滤监控项; 所选图元将以灰色背景突出显示。若要删除过滤，请再次单击该实体。若要将其他实体添加到过滤结果中，请单击另一个实体。

对于每个实体组（例如标签、主机），最多显示 10 行实体。如果有更多实体，可以通过单击末尾显示的三点图标将此列表扩展到最多 1000 个条目（前端定义 中 SUBFILTER_VALUES_PER_GROUP 的值）。一旦展开到最大值，列表就无法折叠。（注意不可扩展的最大值 100 是 Zabbix 6.0.5 之前的限制。）

在标签值列表中最多显示 10 行标签名称。如果有更多带值的标签名称，可以通过单击底部显示的三点图标将此列表扩展到最多 200 个标签名称。一旦展开到最大值，列表就无法折叠。（注意 Zabbix 6.0.5 之前的限制是不可扩展的最大值 20 行。）

对于每个标记名称，最多显示 10 行值（可扩展到 1000 个条目（前端定义 中 SUBFILTER_VALUES_PER_GROUP 的值））。

只有在主过滤器中未选择任何主机或选择多个主机时，子过滤器中的主机选项才可用。

默认情况下，监控项列表中会显示有数据和无数据的监控项。如果在主过滤器中只选择了一个主机，则子过滤器会提供选项来过滤此主机的仅具有数据的监控项、仅不具有数据的监控项或两者。

每个可点击实体旁边的数字表示它在主过滤器结果中的监控项数。没有监控项的实体不会显示，除非它们之前在子过滤器中被选中。

一旦选择了一个实体，就会显示其他可用实体的数量，并带有一个加号，表示可以将多少监控项添加到当前选择中。

图表

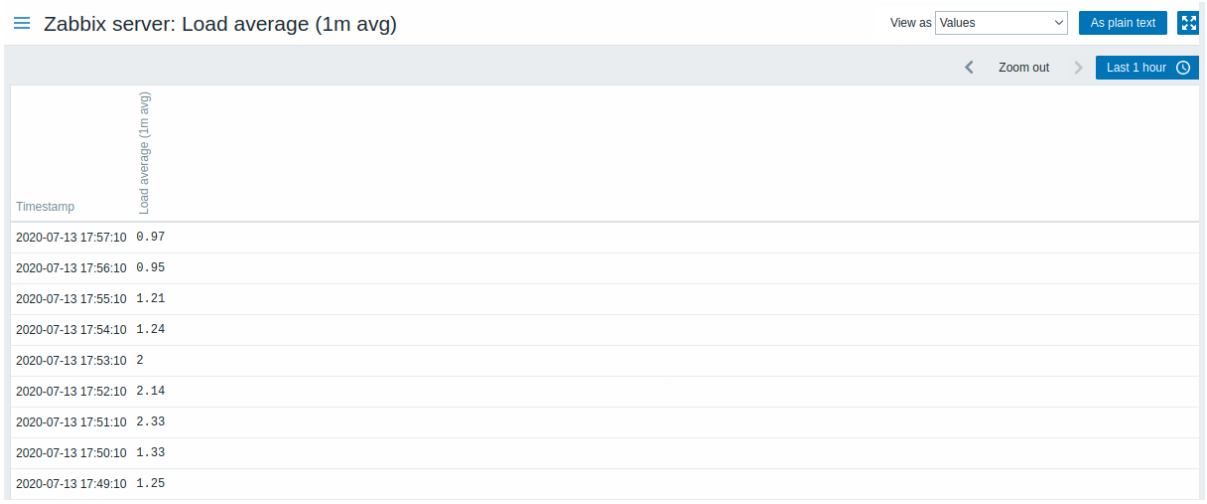
用于比较监控项的临时图形

您可以使用第一列中的复选框选择多个监控项，然后在简单或堆叠的临时图形中比较它们的数据。为此，请选择感兴趣的监控项，然后单击表格下方的所需图表按钮。

指向历史数据/简单图表

最新值列表中的最后一列提供：

- ·历史记录链接（适用于所有文本项）- 指向显示以前监控项值历史记录的列表（值/500 个最新值）。
- ·图形链接 (适用于所有数字监控项) -指向一个简单图形。然后，一旦显示图形，右下角的的下拉列表也可以切换到 （值/*500* 个最新值）。



此列表中显示的值为” 原始”，即不做任何处理。

显示的值的总量由 搜索和过滤结果的限制参数的值定义，该参数在“管理 → 一般”设置。:::

5 拓扑图

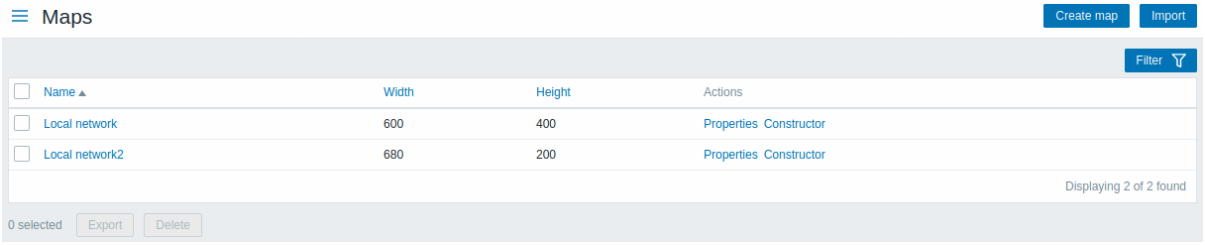
概览

在监测 → 拓扑图部分，您可以配置，管理和查看网络拓扑图。

当您打开此部分时，您将看到您访问的最后一张拓扑图或您可以访问的所有拓扑图的列表。拓扑图列表可以按名称过滤。

所有拓扑图都可以是公共的或私有的。所有用户都可以使用公共拓扑图，而私有拓扑图只能由其所有者和对其共享的用户访问。

拓扑图列表



显示的数据：

列	#### 描述
名称	拓扑图名称。单击名称可以查看拓扑图。
宽度	显示拓扑图宽度。
高度	显示拓扑图高度。
动作	有两个动作可以用： 属性 - 编辑拓扑图通用属性 构造函数 - 通过网络化来添加拓扑图元素

要配置 新地图，请单击右上角的的 创建地图按钮。要从 YAML,、XML、或 JSON 文件导入拓扑图，请单击右上角的 导入按钮。导入拓扑图的用户将被设置为其所有者。

列表下方的两个按钮有一些批量编辑选项：

- ·导出 - 将拓扑图导出为 YAML,、XML、或 JSON 文件
- ·删除 - 删除拓扑图

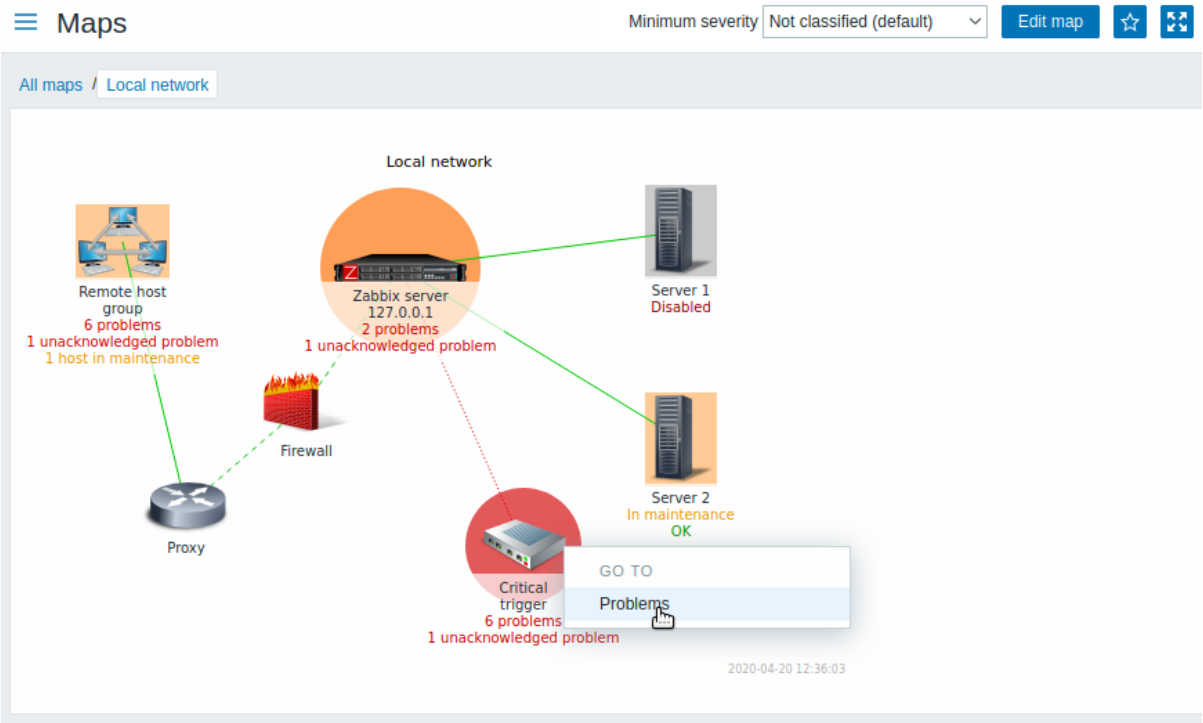
要使用这些选项，请选中各个拓扑图之前的复选框，然后单击所需的按钮。

使用过滤器

您可以使用过滤器仅显示您感兴趣的地图。为了获得更好的搜索性能，使用未解析的宏搜索数据。

查看拓扑图

要查看某个拓扑图，单击所有拓扑图列表中对应的名称。



您可以使用拓扑图标题栏中的下拉列表来选择要显示问题触发器的最低严重性级别。默认严重性是在拓扑图配置中设置的级别。如果拓扑图包含子拓扑图，则导航到子拓扑图将保留上层级别拓扑图严重性。

图标高亮显示

如果一个拓扑图元素处于问题状态，则以圆圈突出显示。圆的填充颜色对应于问题触发器的严重性颜色。该元素仅展示选定严重性级别或更高级别的问题。如果所有问题都得到确认，圆形周围会显示一个加粗的绿色边框。

另外：

- ·如果一个主机在维护 状态状态，则以橙色背景块高亮显示。请注意，维护期高亮显示的优先级高于问题严重性高亮显示。* 禁用（未监视）主机以灰色背景块高亮显示。

如果在拓扑图配置中选中了图标高亮复选框，则图标会高亮显示。

最近更改的标记

元素周围向内指向的红色三角形表示最近的触发状态变化 - 最近 30 分钟内触发状态发生更改。如果在拓扑图配置中选择了 “触发器状态上的标记组件改变” 复选框，则会显示这些三角形。

链接

点击拓扑图元素会打开一个包含一些可用链接的菜单。

按钮

右侧的按钮提供以下选项：

.	.
Edit map	跳转到聚合图形构造器来编辑聚合图形。
	把聚合图形添加到仪表盘中的 “收藏夹” 小构件中。
	在仪表盘“收藏夹” 小构件中的聚合图形。单击会从 “收藏夹” 中移除聚合图形。

在监控页介绍了所有部分通用的显示模式按钮。

在拓扑图中读取摘要

一个隐藏的可用属性 “气泡 (aria-label)”，允许使用屏幕阅读器阅读拓扑图信息。通用拓扑图描述和单个元素描述都可以用，格式如下：

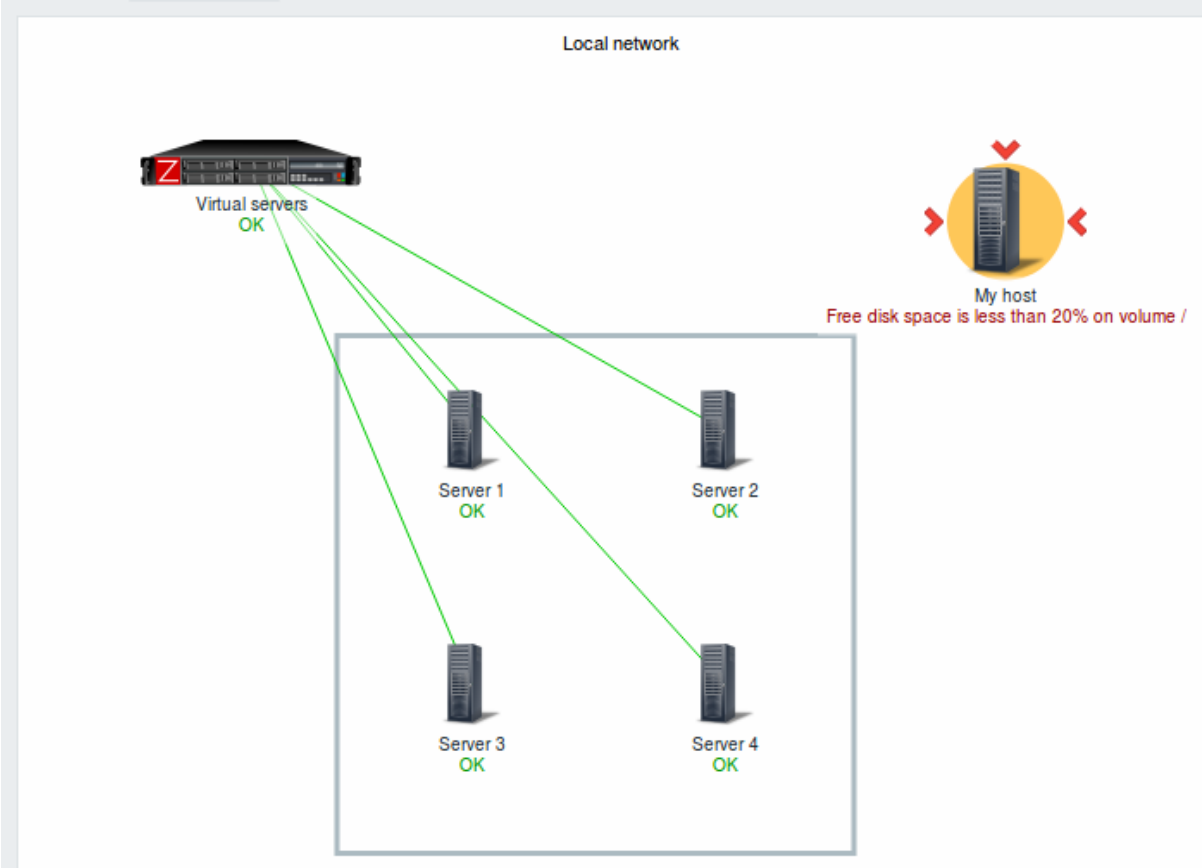
- ·拓扑图描述 :·`<Map name>, <* of * items in problem state>, <* problems in total>.`

- ·描述某个元素的某个问题：·<Element type>, Status <Element status>, <Element name>, <Problem description>.
- ·描述某个元素的多个问题：·<Element type>, Status <Element status>, <Element name>, <* problems>.
- ·描述某个没有问题的元素：·<Element type>, Status <Element status>, <Element name>.

例如，可使用这个描述：

·'Local network, 1 of 6 elements in problem state, 1 problem in total. Host, Status problem, My host, Free disk space is less than 20% on volume /. Host group, Status ok, Virtual servers. Host, Status ok, Server 1. Host, Status ok, Server 2. Host, Status ok, Server 3. Host, Status ok, Server 4. '

以下拓扑图：



引用网络拓扑图

可以被 sysmapid 和 mapname GET 参数引用。例如，

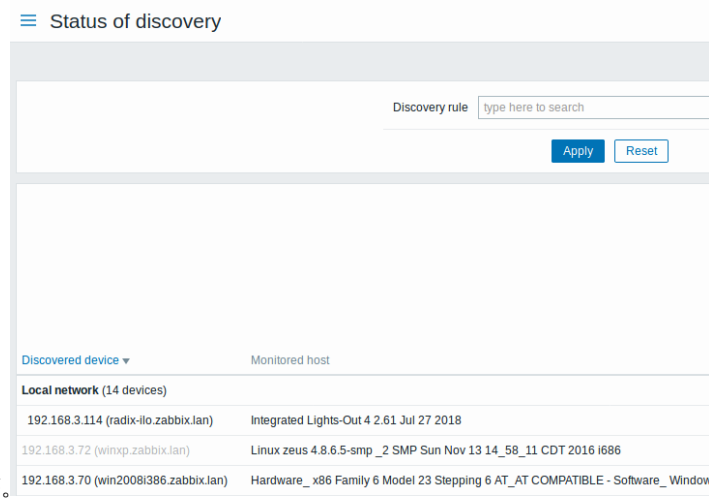
·http://zabbix/zabbix/zabbix.php?action=map.view&mapname=Local%20network

将打开名称是本地网络的拓扑图。

如果同时指定了 sysmapid （地图 ID）和 mapname (地图名称) 时，mapname 具有更高有优先级。

6 自动发现

概览



在 监控→自动发现这部分，显示网络发现 的结果。发现的设备按发现规则排序。

如果设备已被监控，主机名将列在 监控的主机列中，以及在 _ 正常运行/停机时间 _ 列中显示发现设备的 持续时间/发现后的丢失时间。

之后，请按照列显示每个发现的设备的各个服务的状态 (红色单元格显示已关闭的服务)。服务正常运行时间或停机时间都包含在单元中。

Attention:
这些服务中至少有一个在设备中被发现，才会有显示其状态的列。

按钮

监控中介绍了通用的显示模式按钮。

使用过滤器

可以使用筛选器显示感兴趣的发现规则。为了获得更好的搜索性能，使用未解析的宏搜索数据。如果在筛选器中未选择任何内容，则会显示所有已启用的发现规则。

若要选择要显示的特定发现规则，请开始在筛选器中键入其名称。将列出所有匹配的已启用发现规则以供选择。可以选择多个发现规则。

2 Services

概述

本菜单是展示 Zabbix Services service monitoring 功能。

1 Services

Overview

In this section you can see a high-level status of whole services that have been configured in Zabbix, based on your infrastructure.

A service may be a hierarchy consisting of several levels of other services, called ”child” services, which are attributes to the overall status of the service (see also an overview of the service monitoring functionality.)

The main categories of service status are *OK* or *Problem*, where the *Problem* status is expressed by the corresponding problem severity name and color.

While the view mode allows to monitor services with their status and other details, you can also configure the service hierarchy in this section (add/edit services, child services) by switching to the edit mode.

To switch from the view to the edit mode (and back) click on the respective button in the upper right corner:

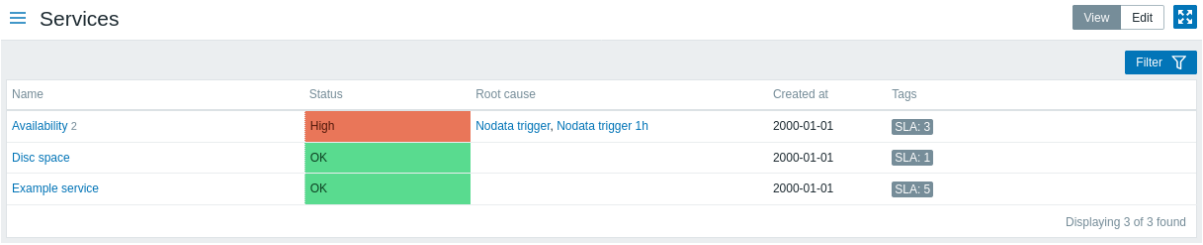
- 

- view services
- 

- add/edit services, and child services

Note that access to editing depends on user role settings.

Viewing services



A list of the existing services is displayed.

Displayed data:

Parameter	Description
Name	Service name. The service name is a link to service details. The number after the name indicates how many child services the service has.
Status	Service status: **OK** - no problems **(trigger color and severity)** - indicates a problem and its severity. If there are multiple problems, the color and severity of the problem with highest severity is displayed.
Root cause	Underlying problems that directly or indirectly affect the service status are listed. The same problems are listed as returned by the {SERVICE.ROOTCAUSE} macro. Click on the problem name to see more details about it in *Monitoring → Problems*. Problems that do not affect the service status are not in the list.
Created at	The time when the service was created is displayed.
Tags	Tagsof the service are displayed. Tags are used to identify a service in service actions and SLAs.

Buttons

View mode buttons being common for all sections are described on the Monitoring page.

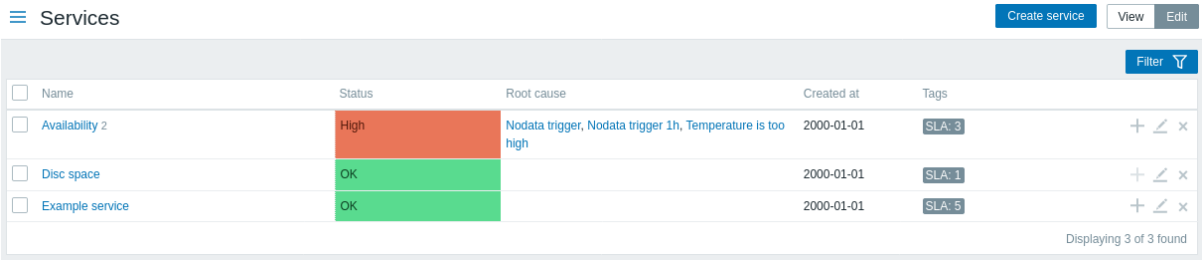
Using filter

You can use the filter to display only the services you are interested in.

Editing services

Click on the *Edit* button to access the edit mode. When in edit mode, the listing is complemented with checkboxes before the entries and also these additional options:

-  - add a child service to this service
-  - edit this service
-  - delete this service

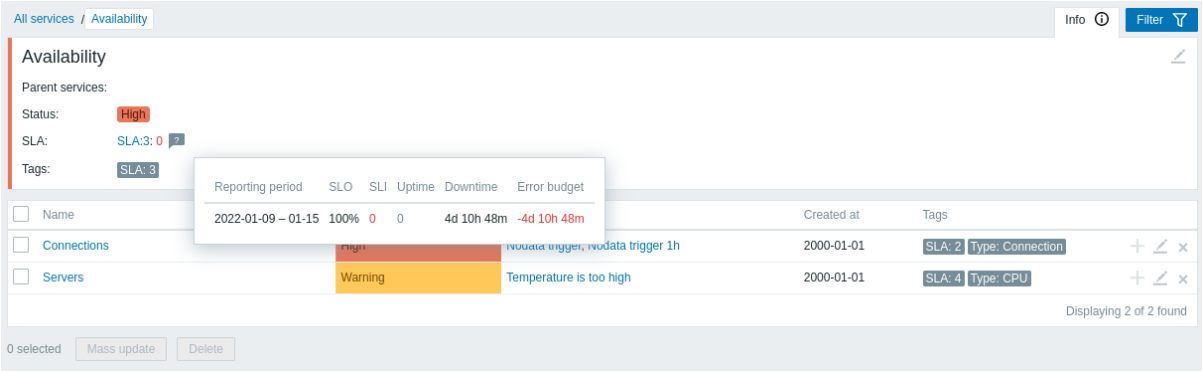


To configure a new service, click on the *Create service* button in the top right-hand corner.

Service details

To access service details, click on the service name. To return to the list of all services, click on *All services*.

Service details include the info box and the list of child services.



To access the info box, click on the *Info* tab. The info box contains the following entries:

- Names of parent services (if any)
- Current status of this service
- Current SLA(s) of this service, in the format `SLA name:service level indicator`. 'SLA name' is also a link to the SLA report for this service. If you position the mouse on the info box next to the service-level indicator (SLI), a pop-up info list is displayed with SLI details. The service-level indicator displays the current service level, in percentage.
- Service tags

The info box also contains a link to the service configuration.

To use the filter for child services, click on the *Filter* tab.

When in edit mode, the child service listing is complemented with additional editing options:

-  - add a child service to this service
-  - edit this service
-  - delete this service

2 服务操作

概述

在 *Services* → *Service actions* 中，用户可以配置 和维护服务。

配置的操作将显示在用户与权限 相关的列表中。用户只能看到授予访问权限的服务的操作.

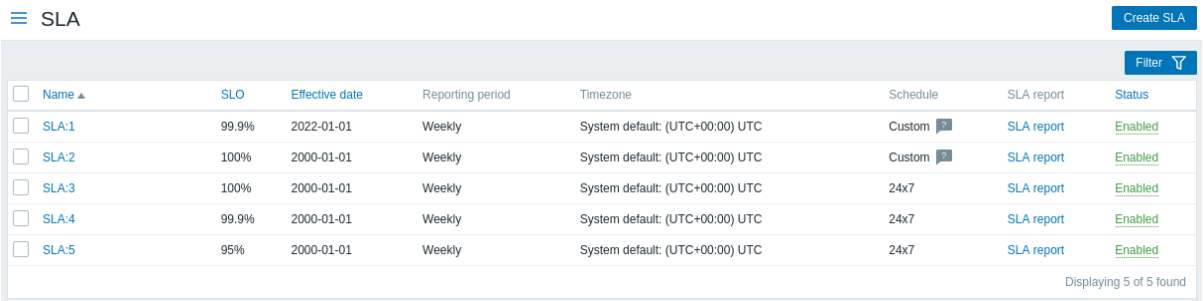
显示的数据，过滤器和可用的批量编辑选项与其他类型的动作.

3 SLA

概述

本节允许查看和配置 SLA。

SLAs



显示已配置 SLA 的列表。注意只有与用户可访问的服务相关的 SLA 才会显示（只读，除非为用户角色启用了管理 *SLA*）。

显示的数据：

参数	描述
Name	显示 SLA 名称。 该名称链接到SLA 配置。
SLO	显示服务级别目标（SLO）。
Effective date	显示开始 SLA 计算的日期。
Reporting period	显示 SLA 报告中使用的期间 - 每天，每周，每月，每季，或者每年。
Time zone	显示 SLA 时区。
Schedule	显示 SLA 计划 - 全天候或定制。
SLA report	点击链接查看此 SLA 的 SLA 报告。
Status	显示 SLA 状态 - 可用或者禁用。

4 SLA 报告

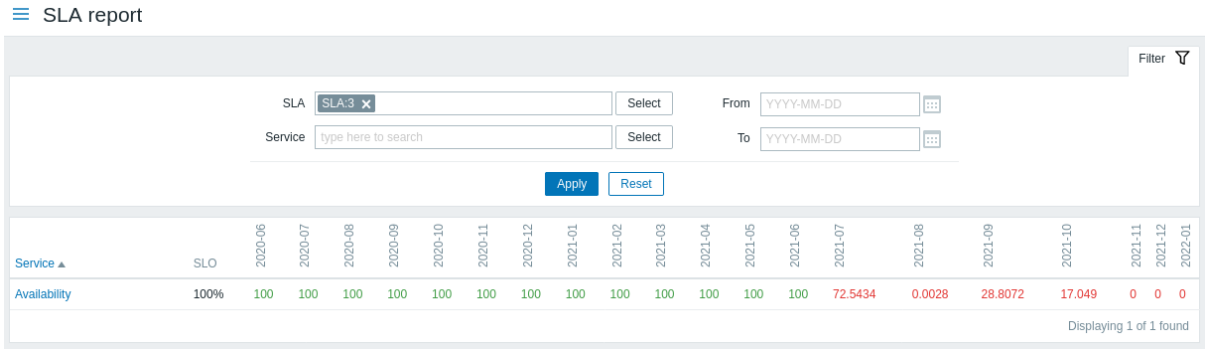
概述

您可以根据筛选条件查看 SLA 报表。

SLA 报表也可以显示为dashboard widget.

报告

可以根据 SLA 名称和服务名称筛选报告。也可以根据时间段来显示。

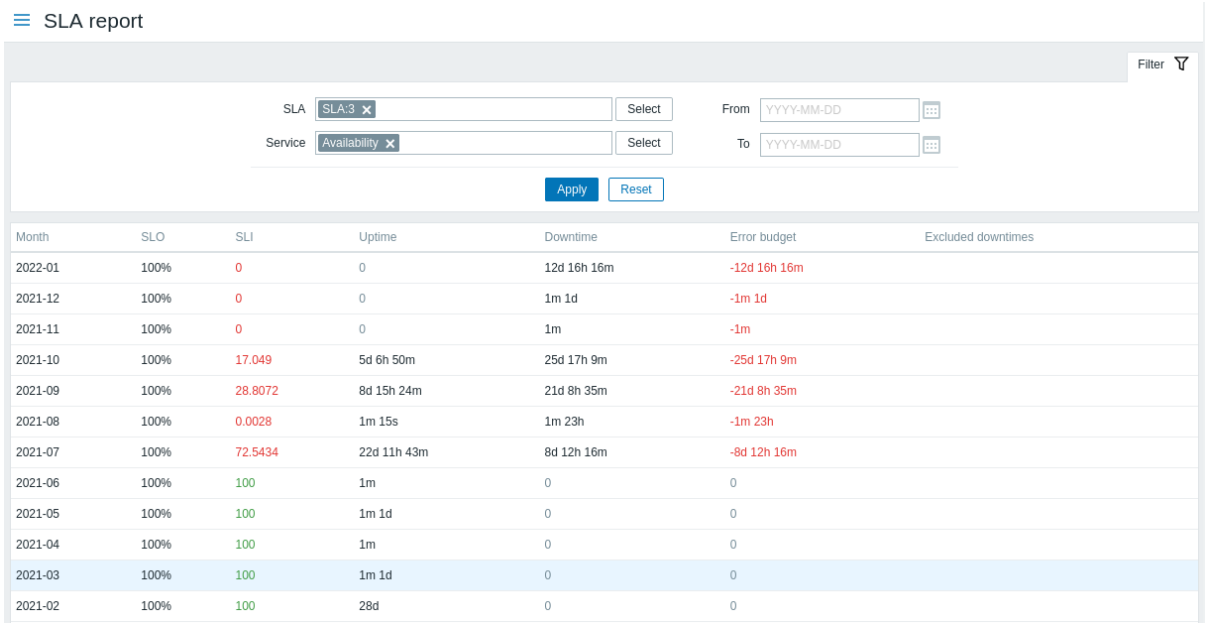


每列（周期）显示该期间的 SLI。违反 SLO 的 SLIs 将以红色突出显示。

报表中显示 20 个周期。输入 *From* 日期和 *To* 日期，最多显示 100 个周期。

报告细节

如果单击报告中的服务名称，则可以显示更详细的报告视图。



3 资产记录

概览

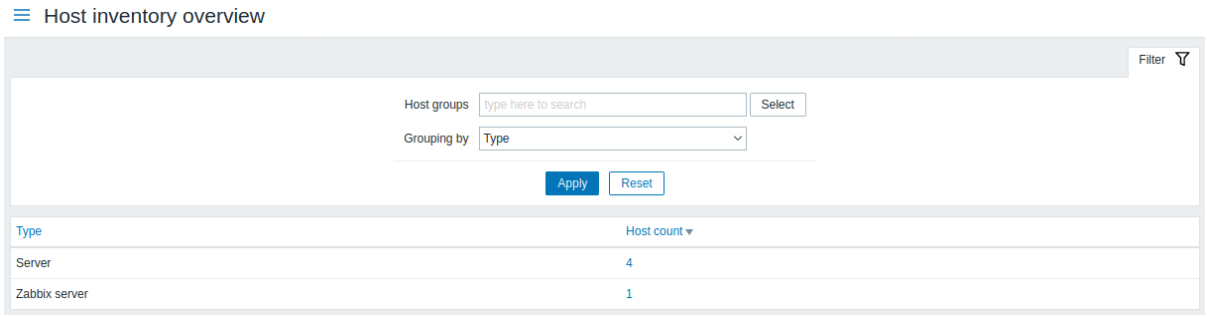
”资产清单”菜单的特点是，通过选择的参数提供主机资产数据的概览，以及查看主机资产详细信息的能力。

1 概览

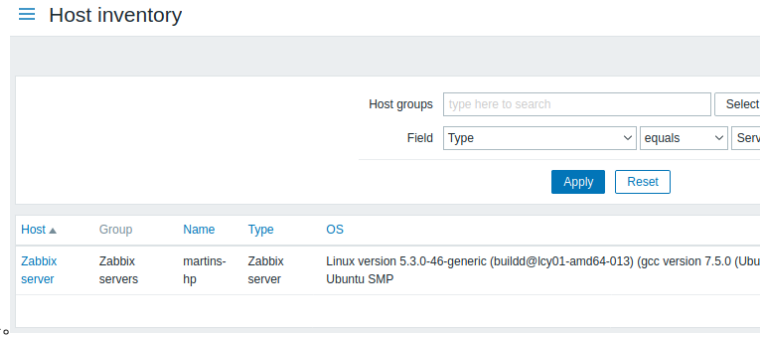
概览

资产记录 → 概览部分提供了有关主机资产数据概览的方法。

要显示的概览，请选择一个主机组（或所有组）和显示数据的资产字段。将显示与所选字段的每个条目相对应的主机数量。



概览的完整性取决于维护了多少主机资产信息。



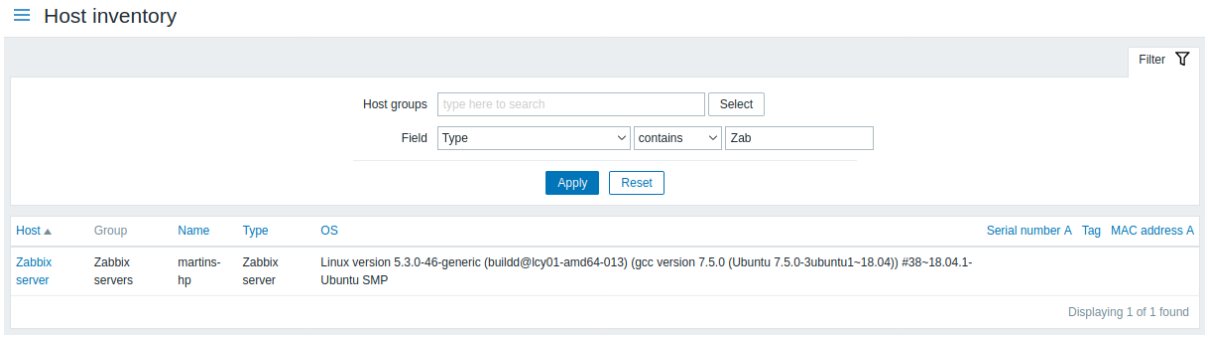
主机计数列中的数字是链接; 它们导致这些主机在 主机资产表中被过滤掉。

2 主机

概览

在 资产记录 → 主机将显示主机的资产记录信息 。

您可以按主机组和任何资产字段过滤主机，来显示您感兴趣的主机。

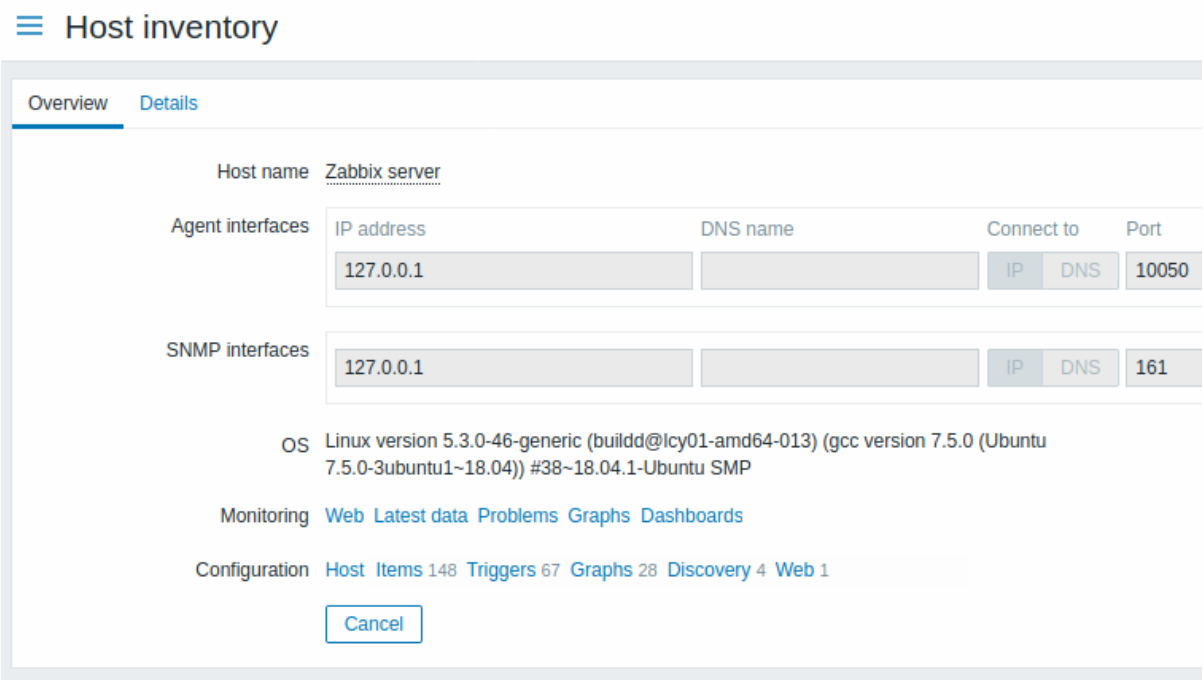


要显示所有主机清单，在组下拉列表中不选择任何主机组，清除过滤器中的比较字段，然后按 “过滤器”。

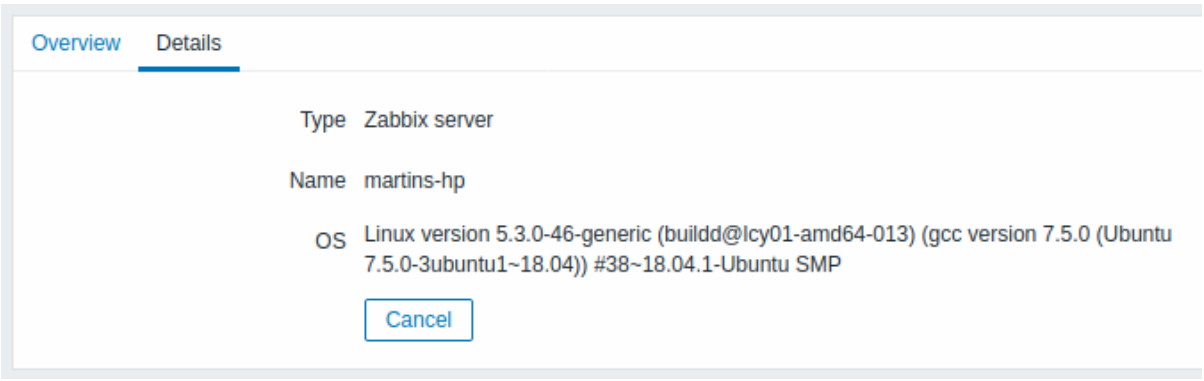
虽然表中只显示了一些关键的资产记录字段，但您也可以查看该主机的所有可用资产信息。如果想这么查看请单击列表中第一列主机名。

资产详情

在 概览选项卡中，包含有关主机的一些通用信息以及预定义脚本的链接，最新的监视数据和主机配置选项：



在 细节选项卡包含主机的所有可用资产记录明细：



资产数据的完整性取决于与主机保持多少库存信息。如果没有维护信息，则细节选项卡禁用。

4 报表

概览

“报表” 菜单包含多个部分，其中包含各种预定义和用户可自定义的报告，这些报告侧重于显示系统信息，触发器和收集数据等参数的概括。

1 系统信息

概览

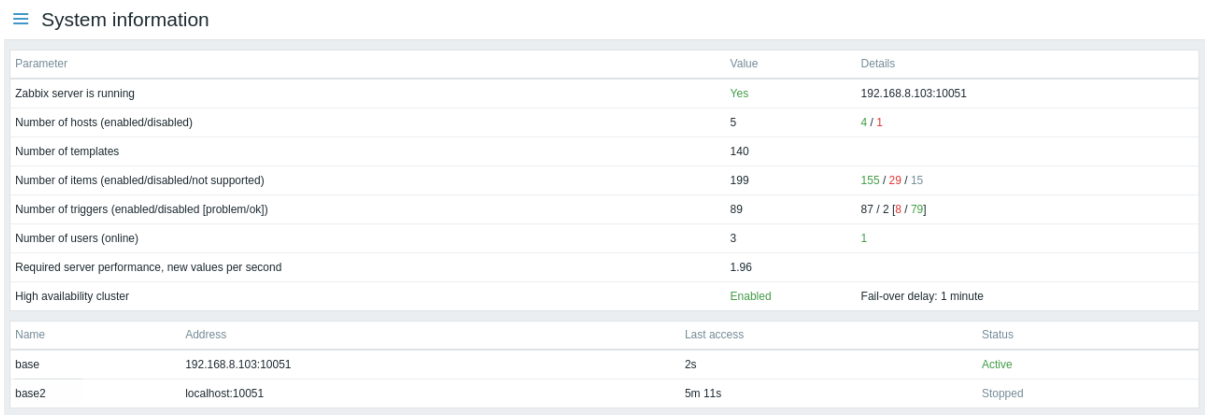
报表 → 系统信息中，显示关键系统数据的摘要。

请注意，在高可用性设置中，可以通过编辑 ui/conf/zabbix.conf.php 文件 - 取消注释并将 $ZBX_SERVER、$ZBX_SERVER_PORT 重定向到显示的活动服务器以外的服务器，从而重定向系统信息源（服务器实例）。

启用高可用性设置后，系统统计信息下方将显示一个单独的块，其中包含高可用性节点的详细信息。此块仅对 Zabbix 超级管理员用户可见。

系统信息也可做为仪表盘部件。

系统统计信息



显示的数据：

参数	值	详情
Zabbix server 运行中	Zabbix 服务器的状态: 是 - 服务器正在运行 否 - 服务器不在运行 提示: 为了确保 Web 前端知道服务器正在运行，服务器上必须至少有一个触发器进程 (zabbix_server.conf 文件中的 StartTrappers 参数 > 0)。	Zabbix 服务器的位置和端口。
主机数	将显示配置的主机总数。	已启用主机/已禁用监测主机的数量。
模板数	显示模板总数。	
监控项数量	将显示监控项目总数。	受监视/禁用/不支持的监控项目数。 被禁用主机上的监控项也会被统计。
触发器数量	将显示触发器总数。	启用/禁用的触发器数。[触发器处于问题/正常状态。] 分配给已禁用主机或取决于已禁用项目的触发器将计为已禁用。
用户数	将显示配置的用户总数。	在线用户数量。
请求服务器的性能, 每秒新值	将显示 Zabbix 服务器每秒能处理的新值的预期数量。	所需的服务器性能是一个估计值，可以用于参考。对于处理的精确数值，使用 `zabbix[wcache,values,all]` 内部监控项。 来自受监视主机的已启用监控项将包括在计算中。日志监控项计为每个监控项更新间隔的一个值。计算常规间隔值；不计算灵活和计划间隔值。在” 无数据” 维护期间，不会调整计算。采集器监控项不计算在内。
数据库历史记录表已升级	数据库升级状态: 否 - 数据库历史记录表尚未升级	如果数据库升级到数值（浮点）值的扩展范围尚未完成，则会显示此字段。参阅启用数值（浮点）值扩展范围的 说明 。
高可用性 *HA* 集群	Zabbix 服务器的高可用群集状态： 已禁用 - 独立服务器 已启用 - 至少存在一个高可用性节点	如果启用，将显示故障转移延迟。

在下列条件下，系统信息将显示错误消息：

- ·使用的数据库没有所需的字符集或排序规则 (UTF-8)；
- ·数据库的版本低于或高于支持范围 ·(仅适用于具有超级管理员角色类型的用户)。
- ·TimescaleDB 的Housekeeping 配置不正确（历史或趋势表包含压缩块，·但覆盖监控项历史周期或 覆盖监控项趋势周期选项被禁用）

高可用性节点

如果启用了高可用性节点 ，则会显示另一个数据块，其中包含每个高可用性节点的状态。



Name	Address	Last access	Status
node-active	192.168.1.13:10051	12s	Active
node6	192.168.1.10:10053	1h 2m 40s	Unavailable
node7	192.168.1.11:10053	3m 40s	Unavailable
node4	192.168.1.8:10052	1h 34m 29s	Stopped
node5	192.168.1.9:10053	1h 9m 51s	Stopped
node8	192.168.1.12:10051	21m 16s	Stopped
node1	192.168.1.5:10051	17s	Standby
node2	192.168.1.6:10051	16s	Standby
node3	192.168.1.7:10052	16 2021-10-20 17:58:47	Standby

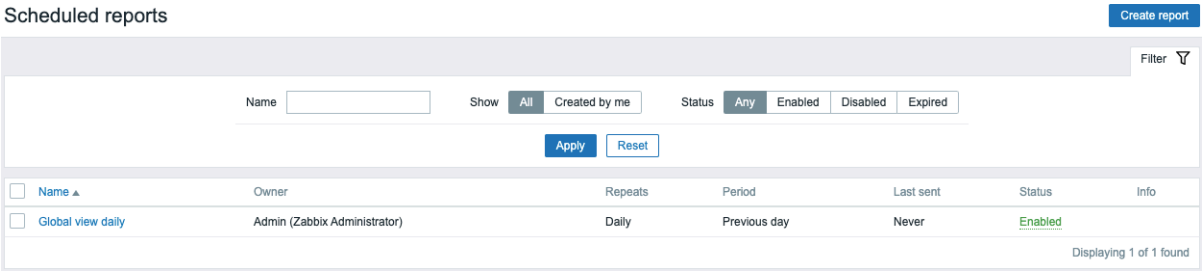
显示的数据：

列	描述
名字	服务器配置中定义的节点名称。
地址	节点 IP 地址和端口。
上次访问	节点上次访问的时间。 将鼠标悬停在单元格上以长格式显示上次访问的时间戳。
状态	节点状态： 活动 - 节点已启动且在工作中 不可用 - 节点未被看到超过故障转移延迟（你可能想知道原因） 已停止 - 节点已停止或无法启动（你可能希望启动或删除它） 备用 - 节点已启动并等待

2 计划报表

概览

在 报表 → 计划报表中，具有足够权限的用户可以配置计划生成 PDF 版的仪表盘，这些报告将通过电子邮件发送给指定的收件人。



打开的屏幕显示有关计划报告的信息，可以将其过滤掉以便于导航 - 请参考下面的使用过滤器 部分。

显示的数据：

列	描述
名字	报告的名字
所有者	创建报告的用户
重复	报告生成频率（每天/每周/每月/每年）
期间	编写报表的期间
最后发送	发送最新报告日期和时间
状态	报告的当前状态（启用/禁用/过期）。具有足够权限的用户可以通过单击它来更改状态- 从”已启用”更改为”已禁用”（然后返回）；从”已过期”到”已禁用”（再返回）。对权限不足的用户显示为文本。
信息	显示信息图标： 红色图标表示报告生成失败; 将鼠标悬停在它上面将显示一个包含错误信息的工具提示。 黄色图标表示已生成报告，但向部分（或全部）收件人发送报告失败或报告已过期; 将鼠标悬停在其上将显示包含其他信息的工具提示

使用过滤器

你可以使用过滤器来缩小报告列表的范围。为了获得更好的搜索性能，使用未解析的宏搜索数据。

以下筛选选项可用：

- ·名称- 允许部分名称匹配;
- ·报表所有者 - 由当前用户或所有报表创建;
- ·状态 - 选择任意 (显示所有报告) 之间，启用、禁用、或过期

筛选器位于计划报告栏上方。可以通过单击右上角的检索条件选项卡来打开和折叠它。

批量更新

有时，您可能希望一次更改状态或删除多个报告。您可以为此使用批量更新功能，而不是打开每个单独的报告进行编辑。

要批量更新某些报告，请执行以下操作：

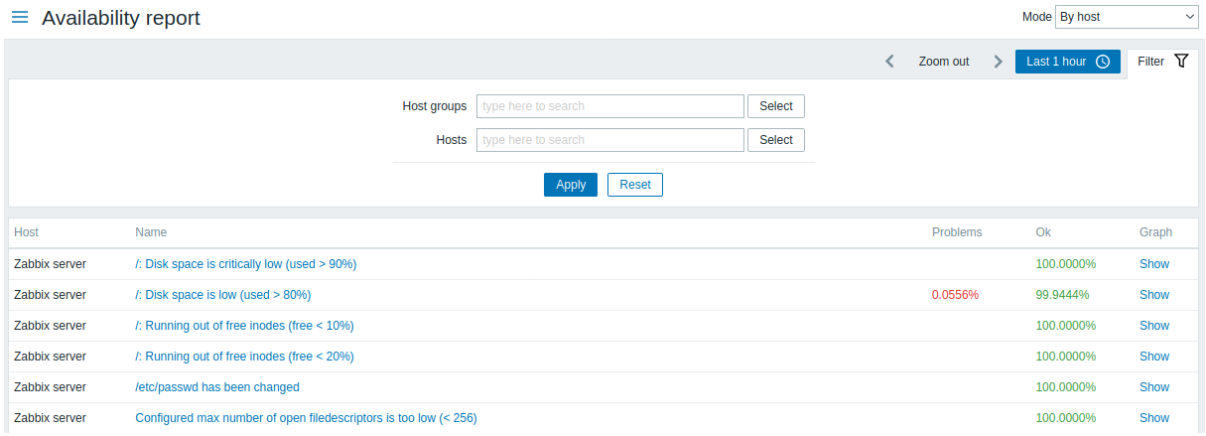
- ·在列表中标记要更新的报告的复选框
- ·单击列表下方的必填按钮进行更改（启用、禁用或删除）。

3 可用性报表

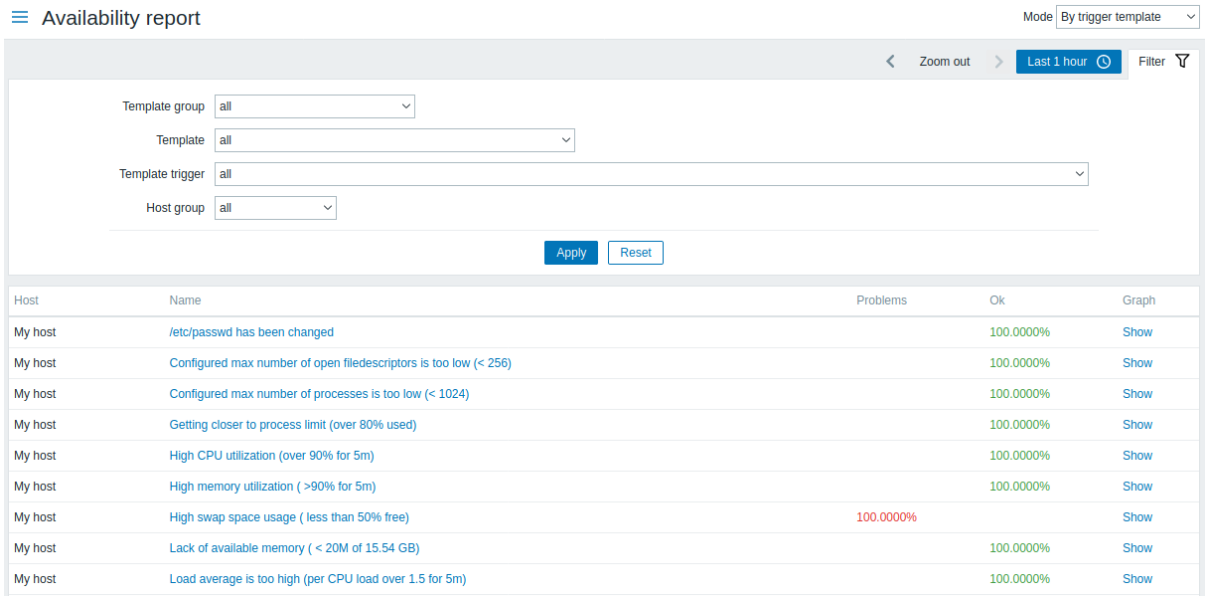
概览

在报表 → 可用性报表中，您可以看到每个触发器处于问题/正常状态的时间比例。显示每个状态的时间百分比。

因此，很容易确定系统中各种元素的可用性情况。



从右上角的下拉列表中，可以选择选择模式-是按主机显示触发器，还是按属于模板的触发器显示触发器。



触发器的名称是指向该触发器最新事件的链接。

使用过滤器

过滤器可以帮助缩小显示的主机和/或触发器的数量。为了获得更好的搜索性能，使用未解析的宏搜索数据。

过滤器位于可用性报表栏下方。点击左侧的过滤器选项卡，可以将其打开和折叠。

按触发器模板过滤

在使用触发器模板模式中，可以通过下方列出的一个或者多个参数对结果进行过滤。

参数	描述
模板组	从属于该组的模板中选择具有触发器的所有主机。可以选择至少包含一个模板的任何主机组。
模板	从所选模板和所有嵌套模板中选择具有触发器的主机。仅显示从所选模板继承的触发器。如果嵌套模板具有其他的触发器，则不会显
//模板触发器//	选择具有所选触发器的主机。选定主机的其他触发器将不被显示。
主机组	选择属于该组的主机。

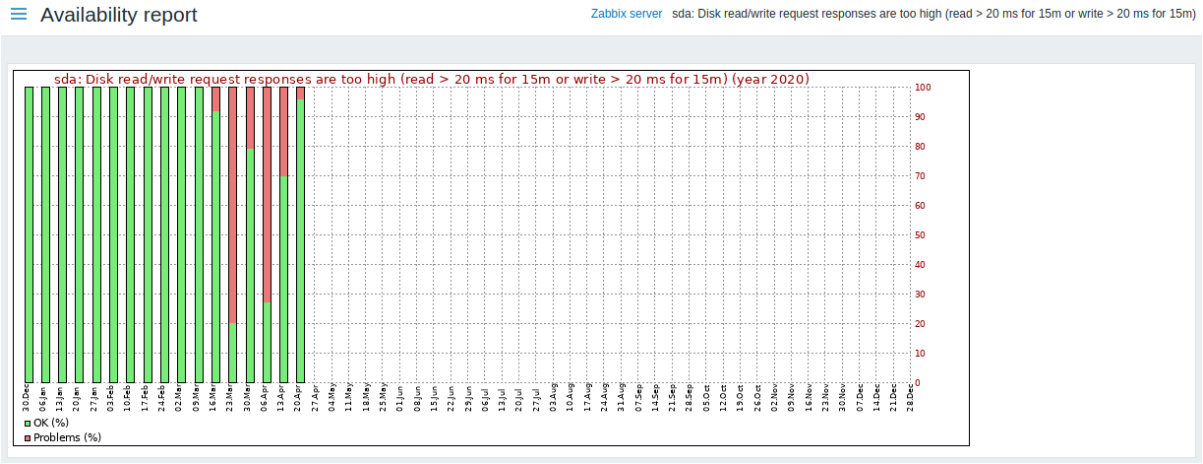
按主机过滤

在按主机模式中，可以通过主机或者主机组对结果进行过滤。指定父主机组会隐式选择所有嵌套的主机组。

时间段选择器

时间段选择器允许通过单击鼠标选择经常需要的周期。通过单击过滤器旁边的时间段选项卡，可以打开时间段选择器。

单击图形列中的查看，将显示一个条形图，其中可用性信息以条形图的形式显示，每个条形图代表当前一年中过去的一周。

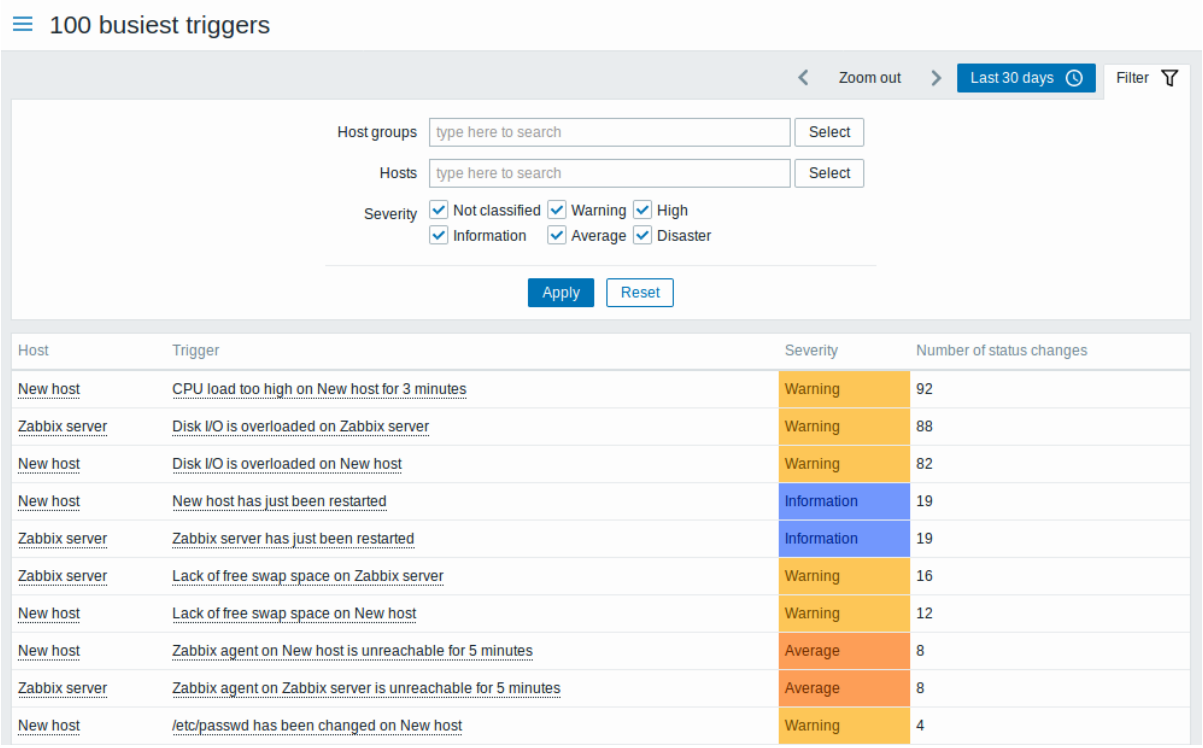


条形图的绿色部分代表正常时间，红色部分代表异常时间。

4 触发器前 **100**

概览

在报表 → 触发器前 *100* 中，您可以看到在评估期间最常更改其状态的触发器，并按状态更改的数量进行排序。



Host	Trigger	Severity	Number of status changes
New host	CPU load too high on New host for 3 minutes	Warning	92
Zabbix server	Disk I/O is overloaded on Zabbix server	Warning	88
New host	Disk I/O is overloaded on New host	Warning	82
New host	New host has just been restarted	Information	19
Zabbix server	Zabbix server has just been restarted	Information	19
Zabbix server	Lack of free swap space on Zabbix server	Warning	16
New host	Lack of free swap space on New host	Warning	12
New host	Zabbix agent on New host is unreachable for 5 minutes	Average	8
Zabbix server	Zabbix agent on Zabbix server is unreachable for 5 minutes	Average	8
New host	/etc/passwd has been changed on New host	Warning	4

主机和触发器列条目都提供一些有用选项的链接：

- 主机 - 链接到主机的用户定义脚本、最新数据、清单、图表和仪表盘
- 触发器 - 链接到最新事件、触发器配置表单和简单图表

使用过滤器

您可以使用过滤器按主机组、主机或触发器严重性显示触发器。指定父主机组会隐式选择所有嵌套的主机组。为了获得更好的搜索性能，使用未解析的宏搜索数据。

过滤器位于 *100* 个最忙触发器栏下方。点击左侧的过滤器选项卡，可以将其打开和折叠。

时间段选择器

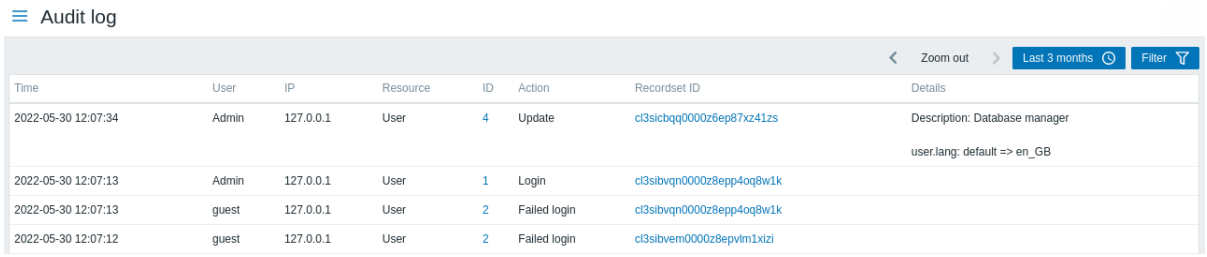
时间段选择器允许通过单击鼠标选择经常需要的周期。通过单击过滤器旁边的时间段选项卡，可以打开时间段选择器。

5 审计

概览

在报表 → 审计部分，用户可以查看前端所做更改的记录。

Note:
应在管理设置中启用审计日志，以显示审计记录。如果禁用日志，前端更改的历史不会被记录到数据库中，并且无法查看审计记录。

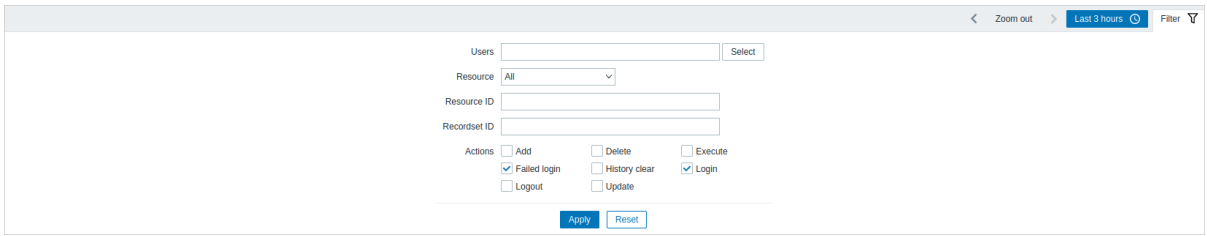


审计日志显示以下数据：

列	描述
时间	审计记录的时间戳。
用户	执行活动的用户。
IP	初始化活动的 IP 地址。
资源	受影响资源的类型（主机、主机组等）。
动作	活动类型：登录、注销、添加、更新、删除、启用，或 禁用。
ID	受影响资源的 ID。点击超链接，将会按此资源 ID 进行过滤审计日志记录。
记录集 *ID*	由同一个前端操作而创建的所有审计日志记录的共享 ID。例如：当链接一个模板到一台主机，会为每个继承的模板实体（监控项、触发器等）创建一条单独的审计日志记录。- 所有的这些记录将会有一个共同的记录集 ID。 点击超链接，将会按此记录集 ID 进行过滤审计日志记录。
详情	资源描述和所执行活动的详细信息。如果一条记录包含两行以上，将显示一条额外的链接详细信息。单击此链接可查看变更的完整列表。

使用过滤器

过滤器位于审计日志栏下方。通过点击右上角的过滤器选项卡，可以将其打开和折叠。



您可以使用筛选器按照用户、受影响的资源、资源 ID 和前端操作 (记录集 ID) 来缩小记录的范围。可以在过滤器中选择资源的一个或多个操作 (例如，添加、更新、删除等)。从 Zabbix 6.0.5 开始，可以选择一个或多个操作。

为了获得更好的搜索性能，将使用未解析的宏搜索所有数据。

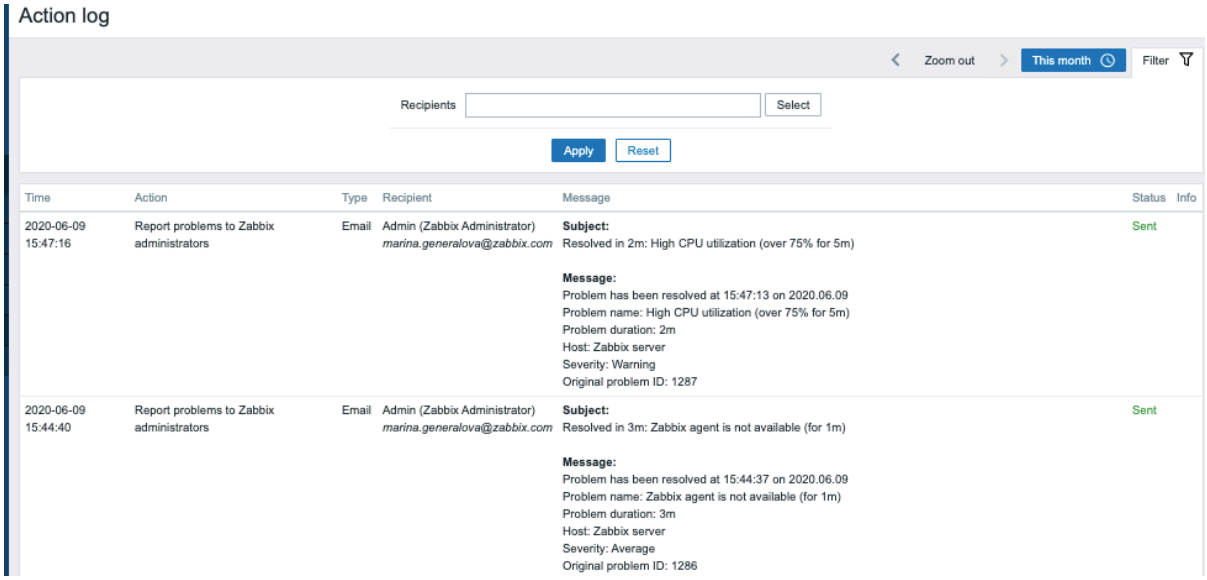
时间段选择器

时间段选择器允许通过单击鼠标选择经常需要的周期。通过单击过滤器旁边的时间段选项卡，可以打开时间段选择器。

6 动作日志

概览

在报表 → 动作日志部分，用户可以查看动作中执行的操作（通知、远程命令）的详细信息。



显示数据：

列	描述
时间	操作的时间戳。
动作	显示导致操作的动作名称。
类型	显示的操作类型 - 邮件或 命令。
收件人	显示通知收件人的用户名、姓名、姓氏（括号中）和电子邮件地址。
消息	显示消息/远程命令的内容。 远程命令与目标主机之间用冒号分隔：< 主机 >:< 命令 >。如果在 Zabbix 服务器上执行远程命令，则信息具有如下格式：Zabbix 服务器:< 命令 >。
状态	显示操作状态： 进行中 - 动作在进行中 对于正在进行的动作，显示剩余的重试次数 - 服务器将尝试发送通知的剩余次数。 已发送 - 通知已经发送 已执行 - 命令已经执行 未发送 - 动作尚未完成。
信息	显示关于动作执行的错误信息（如果有）。

使用过虑器

您可以使用过滤器按消息收件人缩小记录范围。为了获得更好的搜索性能，使用未解析的宏搜索数据。

过滤器位于动作日志栏下方。通过点击左侧的过滤器选项卡，可以将其打开和折叠。

时间段选择器

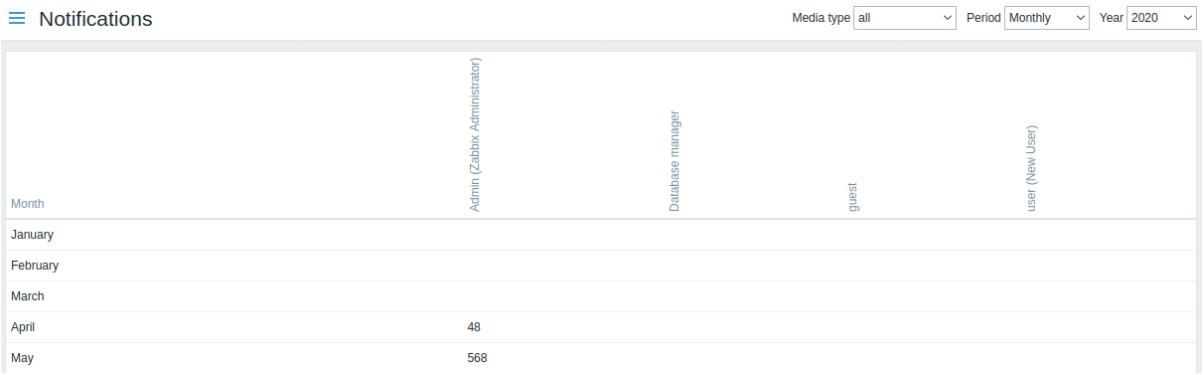
时间段选择器允许通过单击鼠标选择经常需要的周期。通过单击过滤器旁边的时间段选项卡，可以打开时间段选择器。

7 通知

概览

在报表 → 通知部分，将显示发送给每个用户的通知数量的报告。

从右上角的下拉列表中，您可以选择发送通知的媒体类型（或全部）、周期（每天/周/月/年的数据）和年份。



每列显示每个系统用户的总数。

5 配置

概述

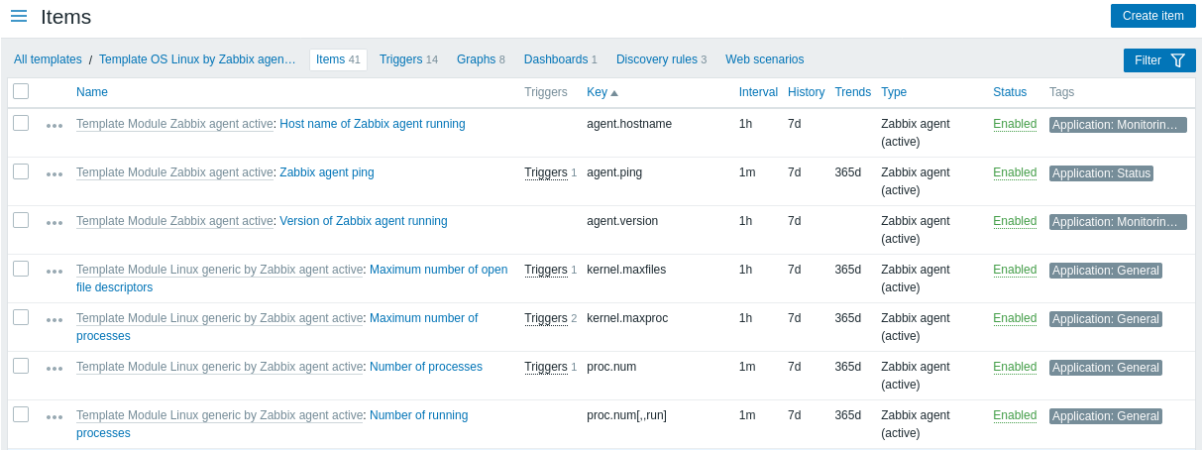
“配置” 菜单包含设置 Zabbix 的主要功能，例如主机和主机组，数据收集，数据阈值，发送问题通知，创建数据可视化等。

1 监控项

概览

要查看模板的监控项列表，可以点击配置 → 模板，再点击相应模板的监控项

将会显示存在的监控项列表。



显示数据:

列	描述
监控项菜单	点击三个点图标，打开这个特定项目的菜单，有以下选项: 新建触发器 - 基于本监控项新建触发器 触发器 - 点击可以看见该监控项的已配置的触发器列表 新建依赖监控项 - 新建一个依赖此监控项的监控项 新建依赖自动发现规则 - 新建一个依赖此监控项的自动发现规则
模板	该监控项属于的模板。 仅当筛选器中选择了多个模板时，此列才会显示。
名称	监控项的名称显示为蓝色链接，包含监控项详细信息。 单击监控项名称链接将打开该监控项配置表单. 如果监控项是从另一个模板继承的，则模板名会以灰色链接的形式显示在项名之前。 单击模板链接将打开该模板的监控项列表。
触发器	将鼠标移到触发器上将显示一个信息框，显示与该监控项相关的触发器。 触发器个数以灰色显示。
键值	显示监控项键值。
时间间隔	显示监测频率。
历史	显示监控项数据历史记录保存的天数。

列	描述
趋势	显示监控项趋势历史数据将保存多少天的。
类型	显示监控项类型 (Zabbix agent, SNMP agent, 简单监测，等等)。
状态	显示监控项状态 - 启用 or 禁用. 通过点击状态，您可以改变它-从启用到禁用（或者从禁用到启用）。
标签	显示监控项标签 最多可以显示三个标记 (名称: 值对)。如果有更多的标签，会显示成一个 “...” 链接，鼠标悬停上面就能看到所有的标签。

要配置新监控项，请单击右上角的新建监控项按钮。

批量编辑选项

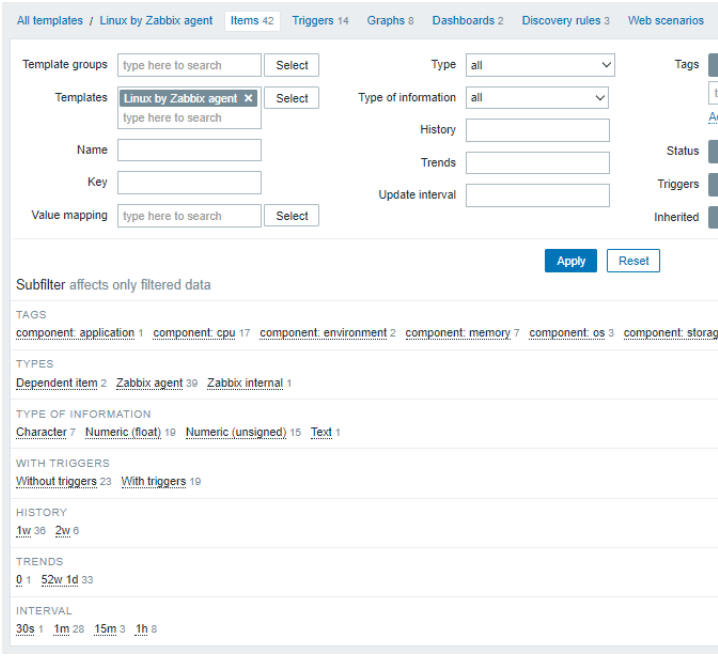
下面的按钮提供了一些批量编辑选项:

- 启用 - 把监控项状态设为启用。
- 禁用 -把监控项状态设为禁用。
- 拷贝 -把监控项拷贝到其他主机或模板中。
- 批量更新 -一次为多个监控项 [更新多个属性](/manual/config/items/itemupdate)。
- 删除 -删除监控项。

要使用这些选项，请在相应监控项前勾选复选框，然后点击所需的按钮。

使用过滤器

监控项列表可能包含很多监控项。通过使用过滤器，您可以过滤掉其中的一些内容，从而快速找到您要寻找的项目。为了获得更好的搜索性能，在搜索数据时不会解析宏。



右上角有 *Filter* 图标。点击它将打开一个过滤器，您可以指定所需的过滤条件。

参数	描述
主机组	根据一个或多个主机组进行过滤。 主机组至少包含一个模板才能被选中。 选中父主机组，默认将选择所有嵌套的主机组。
模板	通过一个或多个模板进行筛选。
名称	通过监控项名称进行筛选。
键值	按监控项的键值进行过滤
值映射	根据使用的值映射过滤。 当 “模板” 选项为空时，此参数不显示
类型	按监控项类型 (Zabbix agent、SNMP agent 等) 进行过滤
信息类型	根据信息类型 (数字无符号，浮点数等) 进行过滤
历史	根据监控项历史记录保存的时间进行过滤
趋势	根据监控项趋势保持的时间进行过滤。
更新间隔	按监控项更新时间间隔进行过滤

参数	描述
标签	通过指定标签来限制显示监控项的数量。可以包含或排除特定的标签和标签值。可以设置多个条件。标记名称匹配必须区分大小写。 对于每个条件都有几种操作符可用: 存在 -包括指定的标签名称 等于 -包括指定的标签名称和值 (区分大小写) 包含 -包括指定的标签名称，其中标签值包含输入的字符串 (子字符串匹配，不区分大小写) 不存在 -排除指定的标签名称 不等于 -排除指定的标签名称和值 (区分大小写) 不包含 -排除指定的标签名称，其中标签值包含输入的字符串 (子字符串匹配不区分大小写) 条件的计算类型有两种: 与/或 -如果有多个条件满足，则这些标签名相同的条件会用或拼接 或 -如果其中一个条件满足
状态	按监控项状态过滤- 启用或禁用
触发器	过滤有 (或没有) 触发器的监控项
继承	过滤从链接模板中继承 (或不继承) 的监控项

过滤器下面的子过滤器提供了进一步的过滤选项 (对于已经过滤的数据)。可以选择几组具有公共参数值的监控项。在单击一个组时，它将突出显示，只有具有此参数值的监控项留在列表中。

Using subfilter

The subfilter allows to further modify the filtering from the main filter.

It contains clickable links for a quick access to related items. Items are related by common entity - tag name or value, item type, item state, etc. When an entity is clicked, the entity is highlighted with a gray background, and items are immediately filtered (no need to click *Apply* in the main filter). Clicking another entity adds it to the filtered results. Clicking the entity again removes the filtering.

Subfilters are generated based on the filtered data, which is limited to 1000 records. If you want to see more records in the subfilter, you need to increase the value of *Limit for search and filter results* parameter (in *Administration -> General -> GUI*).

Unlike the main filter, the subfilter is updated with each table refresh request to always have up-to-date information of available filtering options and their counter numbers.

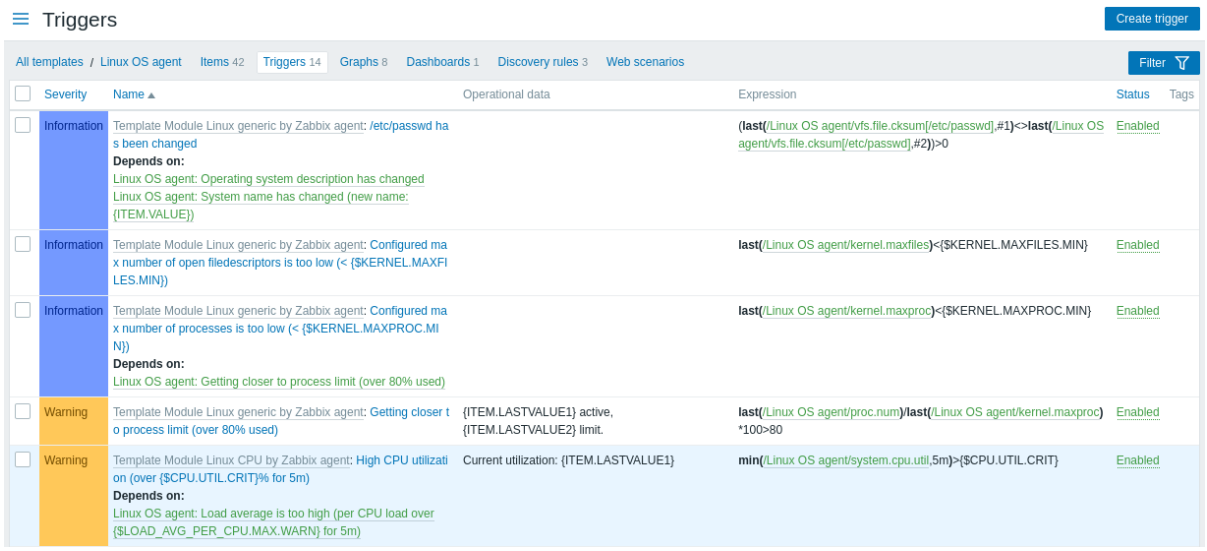
For each entity group (tags, tag values, etc.), up to 10 rows of entities are displayed. If there are more entities, this list can be expanded to display a maximum of 1000 entries (the value of SUBFILTER_VALUES_PER_GROUP in frontend definitions) by clicking the three-dot icon at the end of the list. For *Tag values*, the list can be expanded to display a maximum of 200 tag names with their corresponding values. Note that once fully expanded, the list cannot be collapsed.

A number next to each clickable entity indicates the number of items grouped in it (based on the results of the main filter). When an entity is clicked, the numbers with other available entities are displayed with a plus sign indicating how many items may be added to the current selection. Entities without items are not displayed unless selected in the subfilter before.

2 触发器

概览

在配置 → 模板中，点击对应模板的触发器，可查看对应模板的触发器列表。



显示数据:

列	描述
严重等级	该触发器的级别，通过名称和单元格背景颜色显示
模板	触发器所属的模板。 当筛选框中选择了多个模板时，此字段才会显示。
名称	触发器的名称，以蓝色链接的形式显示，可了解触发器的详细信息。 单击触发器名称链接将打开触发器配置表格. 如果触发器是从其他模板继承的，则模板名称会以灰色链接的形式显示在触发器名称前面。单击模板链接将打开该模板级别的触发器列表。
运行时数据	触发器的运行时数据定义，包含在监控 → 问题中动态解析的任意字符串和宏变量。
表达式	显示触发器表达式。表达式的模板-监控项部分显示为一个链接，连接到监控项配置形式。
状态	显示触发状态-启用” 或 “禁用”。通过单击状态，您可以更改它-从启用到禁用 (并返回)。
标签	如果触发器包含标签，此列显示标签名称和值。

单击右上角的 “创建触发器” 按钮，可新建触发器。

批量编辑选项

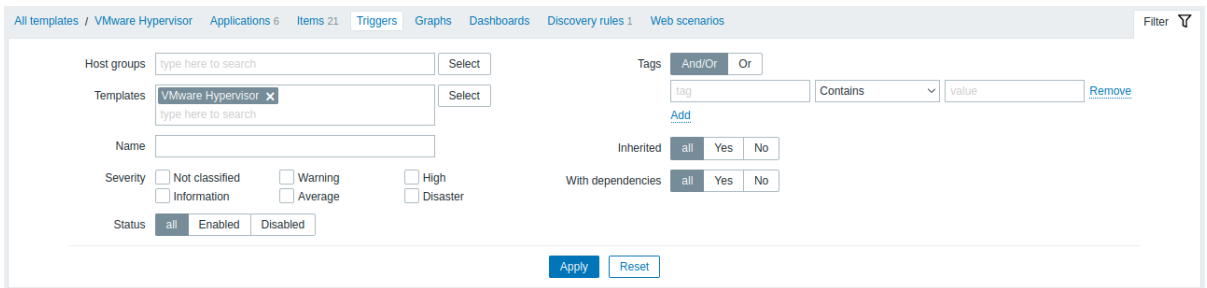
下面的按钮提供了一些批量编辑选项:

- 启用 -将触发器状态更改为启用
- 禁用 -将触发器状态更改为禁用
- 复制 -将触发器复制到其他主机或模板
- 批量更新 -一次更新多个触发器的几个属性
- 删除 -删除触发器

要使用这些选项，请在各个触发器前标记复选框，然后单击所需的按钮。

使用筛选器可以只显示您感兴趣的触发器。为了获得更好的搜索性能，在搜索数据时不解析宏。

右上角有过滤器图标。点击它将打开一个过滤器，您可以指定所需的过滤条件。



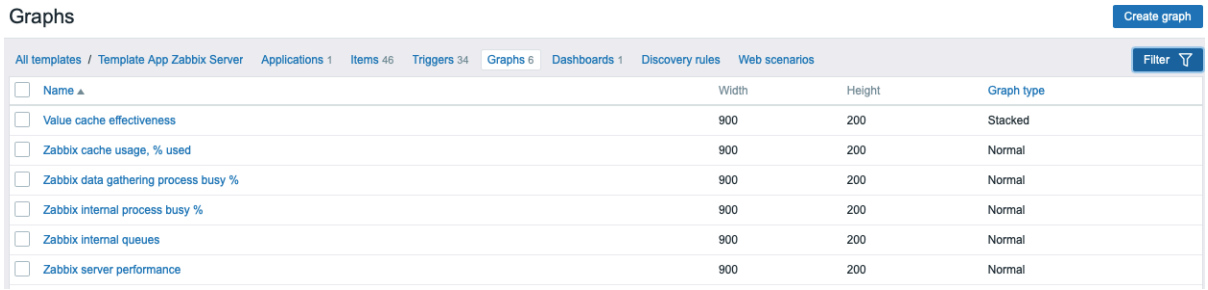
参数	描述
主机组	根据一个或多个主机组进行过滤。 主机组至少包含一个模板才能被选中。 选中父主机组，默认将选择所有嵌套的主机组。
模板	通过一个或多个模板进行筛选。 如果已经选择了以上的主机组，则只能选择这些主机组的模板。
名称	通过触发器名称进行筛选。
严重级别	按一个或多个触发级别进行过滤
状态	按触发器状态进行过滤
标签	通过指定标签来限制显示监控项的数量。可以包含或排除特定的标签和标签值。可以设置多个条件。标记名称匹配必须区分大小写。 对于每个条件都有几种操作符可用: 存在 -包括指定的标签名称 等于 -包括指定的标签名称和值 (区分大小写) 包含 -包括指定的标签名称，其中标签值包含输入的字符串 (子字符串匹配，不区分大小写) 不存在 -排除指定的标签名称 不等于 -排除指定的标签名称和值 (区分大小写) 不包含 -排除指定的标签名称，其中标签值包含输入的字符串 (子字符串匹配不区分大小写) 条件的计算类型有两种: 与/或 -如果有多个条件满足，则这些标签名相同的条件会用或拼接 或 -如果其中一个条件满足
继承	过滤从链接模板中继承 (或不继承) 的监控项
依赖项	过滤有 (或没有) 依赖项的触发器

3 图形

概述

模板的自定义图形列表可以通过配置 → 模板点击相应模板的图形来访问。

显示现有图形的列表。



显示数据:

列	描述
模板	图所属模板。 当过滤框中选择了多个模板时，此字段才会显示
Name	Name of the custom graph, displayed as a blue link to graph details. Clicking on the graph name link opens the graph configuration form. If the graph is inherited from another template, the template name is displayed before the graph name, as a gray link. Clicking on the template link will open the graph list on that template level.
名称	自定义图形名称，显示为蓝色链接，指向图形的详细信息。 点击图形名称链接打开图形配置表单。 如果该图是从其他模板继承的，则模板名以灰色链接显示在图名前面。单击模板链接将打开该模板级别的图形列表
Width	Graph width is displayed.
Height	Graph height is displayed.

列	描述
Graph type	Graph type is displayed - *Normal*, *Stacked*, *Pie* or *Exploded*.

To configure a new graph, click on the *Create graph* button at the top right corner.

| 宽度 | 显示图形的宽度 | | 高度 | 显示图形高度 | | 图类型 | 显示图类型- 正常、堆叠、饼状或分解 |

若要配置新图形，请单击右上角的 “* 创建图形” 按钮。

批量编辑选项

下面的按钮提供了一些批量编辑选项:

- 复制 -将图形复制到其他主机或模板中
- 删除 -删除图形

要使用这些选项，请在需操作的图形前勾选复选框，然后单击所需的按钮。

使用过滤器

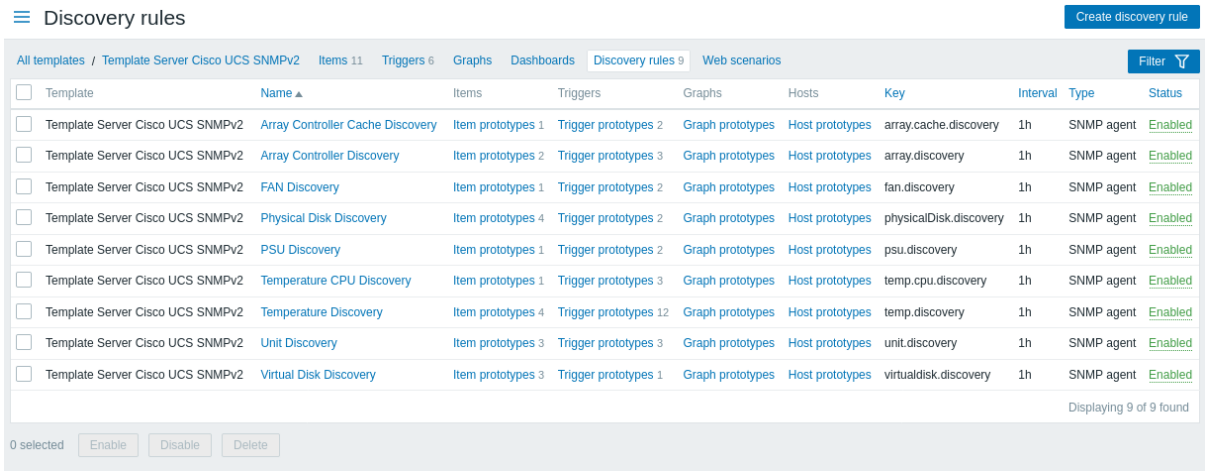
可以根据主机组和模板对图进行过滤。为了获得更好的搜索性能，在搜索数据时不解析宏。

4 自动发现规则

概述

模板的低级发现规则列表可以在配置成 → 模板中通过点击 * 自动发现 * 访问。

显示已存在的低级别自动发现规则列表。也可以独立于模板查看所有自动发现规则，或者通过更改过滤器设置查看特定主机组的所有自动发现规则。



显示数据:

列	描述
模板	自动发现规则属于哪个模板。
名称	规则名称，显示为蓝色链路。 单击规则名打开低级自动发现规则配置表单。 如果自动发现规则是从其他模板继承的，则模板名称以灰色链接显示在规则名称前面。单击模板链接将打开该模板级别的自动发现规则列表
监控项	显示到监控项原型列表的链接。 已存在的监控项原型数量以灰色显示
触发器	显示触发器原型列表的链接。 已存在的触发器原型数量以灰色显示
图形	显示图形原型列表的链接。 当前已有的图形原型数量，以灰色显示
主机	显示的主机原型列表链接。 当前主机原型数量以灰色显示
键	显示自动发现的监控项的键
时间间隔	显示自动发现频率
类型	显示自动发现的项目类型 (Zabbix agent、SNMP agent 等)

列	描述
状态	自动发现规则状态- 已启用或禁用。通过单击状态，您可以更改它-从启用到禁用（或者反过来）

单击右上角的 “创建自动发现规则” 按钮，可以配置新的低级别发现规则。

批量编辑选项

下面的按钮提供了一些批量编辑选项:

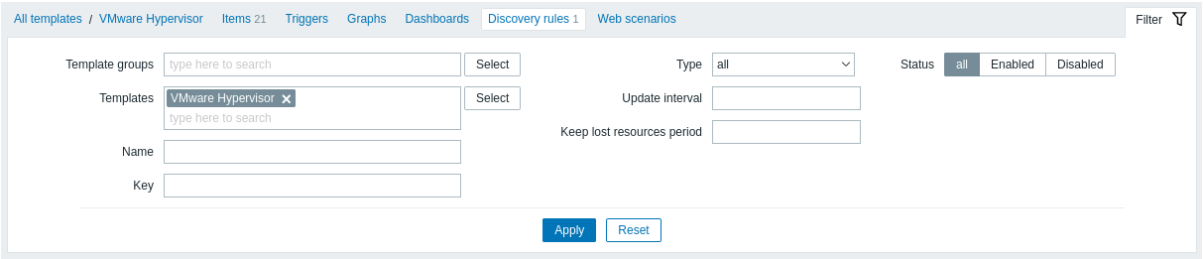
- 启用—修改低级别发现规则的状态为 “* 已启用”
- 禁用 -将低级别发现规则的状态修改为 “禁用”
- 删除 -删除低级别发现规则

要使用这些选项，请在各自的发现规则之前标记复选框，然后单击所需的按钮。

使用过滤器

您可以使用筛选器只显示感兴趣的自动发现规则。为了获得更好的搜索性能，在搜索数据时不解析宏。

右上角有过滤器图标。单击它将打开一个过滤器，您可以在其中指定所需的过滤条件，如模板、发现规则名称、项目键、项目类型等。



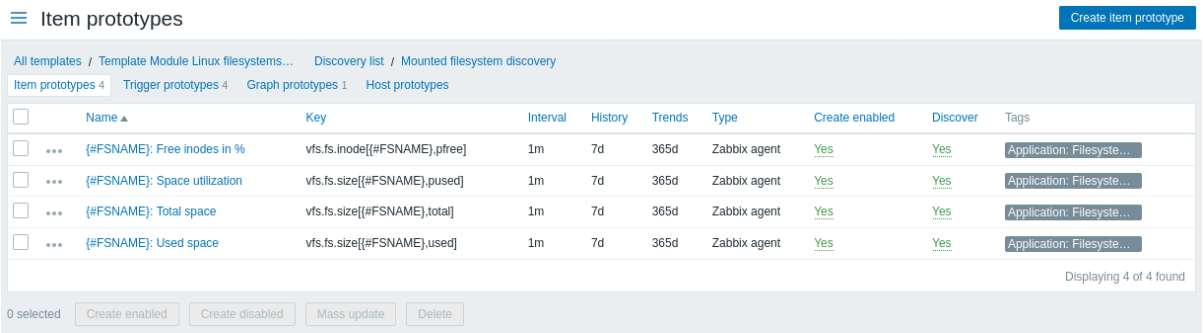
参数	描述
主机组	过滤一个或多个主机组。 只能选择包含至少一个模板的主机组。 指定父主机组时，将默认选择所有嵌套的主机组
模板	过滤一个或多个模板
名称	根据自动发现规则名称进行过滤
键	根据自动发现项键进行过滤
类型	根据自动发现项类型进行过滤
更新间隔	按更新间隔过滤。 对 Zabbix 捕获器和依赖项不可用
丢失资源保留周期	按保留丢失资源周期过滤
状态	根据发现规则状态进行过滤 (全部/已启用/已禁用)

1 监控项原型

概述

这部分展示的是模板上的低等级发现规则的配置项原型。

如果模板链接到主机，则监控项原型将成为在低级别发现期间创建真实主机监控项的基础。



展示的数据：

列	描述
Name	项原型的名称，展示为蓝色链接. 点击名称打开项原型配置格式. 如果项原型属于一个链接模板，模板名称显示在项名称前面，是灰色的链接。点击模板链接将打开链接模板级别的项原型列表。
Key	展示项原型的键值.
Interval	显示检查的频率.
History	展示保存项数据历史的天数.
Trends	展示保存项趋势历史的天数.
Type	展示项原型的类型 (Zabbix agent, SNMP agent, simple check, 等等)。
Create enabled	基于这个原型创建项: **Yes** - 启用 **No** - 禁用。你可以通过点击来切换’Yes’ 和’No’。
Discover	发现项基于这个原型: **Yes** - 发现 **No** - 不发现。你可以通过点击来切换’Yes’ 和’No’。
Tags	展示标签的项原型.

点击右上方的 *Create item prototype* 按钮来创建新的监控项原型。

批量编辑选项

列表下的按钮提供了一批量编辑选择：

- *Create enabled* - 将监控项创建为 *Enabled*
- *Create disabled* - 将监控项创建为 *Disabled*
- *Mass update* - 批量更新监控项原型
- *Delete* - 删除监控项原型

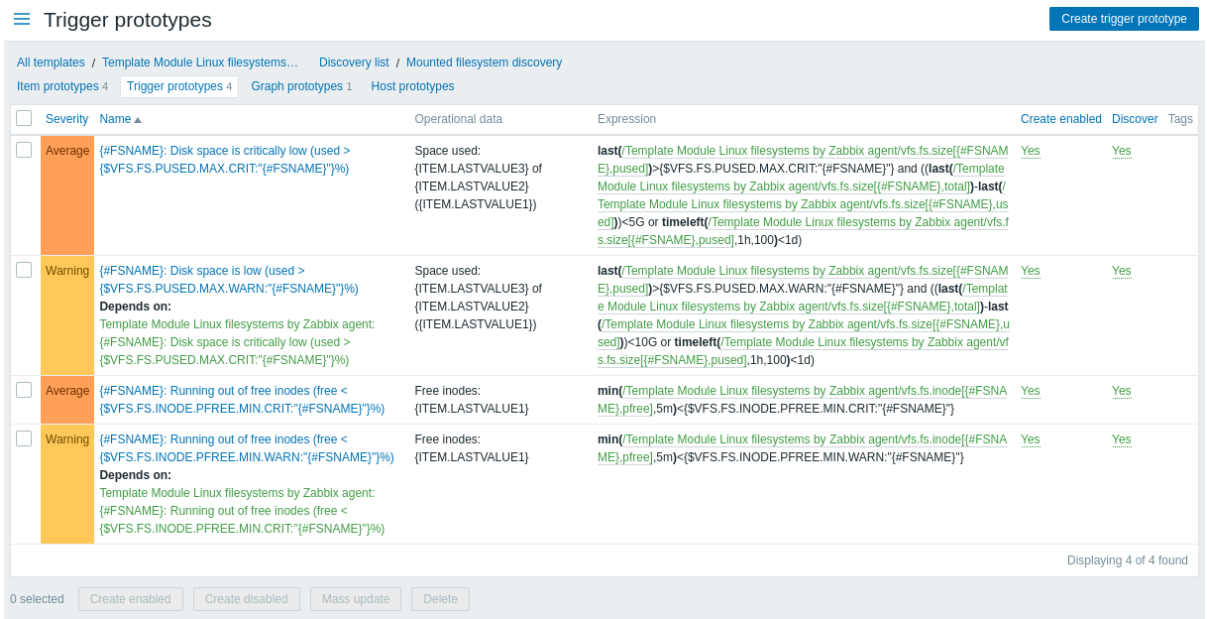
要使用这些选项，在相应的监控项原型之前勾选复选框，然后点击所需的按钮。

2 触发器原型

概述

在本节中，模板上配置的低级发现规则的触发器原型如下。

如果模板链接到主机，触发器原型将成为在低级别发现期间创建真实主机触发器的基础。



数据如下：

列	描述
Name	触发器原型的名称，展示为蓝色的链接. 点击名称打开触发器原型配置格式. 如果触发器属于一个链接模板，模板名称显示在触发器名称前面，为灰色链接. 点击模板链接会在模板层面打开触发器原型列表.
Operational data	将显示触发器的操作数据格式，其中包含将在 *Monitoring* → *Problems* 中动态解析的任意字符串和宏。
Create enabled	创建触发器基于这个原型: **Yes** - 启用 **No** - 禁用. 你可以通过点击来切换’Yes’ 和’No’。
Discover	发现触发器基于这个原型: **Yes** - 发现 **No** - 没发现. 你可以通过点击来切换’Yes’ 和’No’。
Tags	展示了触发器原型的标签.

创建新的触发器原型，点击右上方的 *Create trigger prototype* 按钮。

批量编辑选项

列表下方的按钮提供了一些批量编辑的选项：

- *Create enabled* - 将触发器创建为 *Enabled*
- *Create disabled* - 将触发器创建为 *Disabled*
- *Mass update* - 批量更新这些触发器原型
- *Delete* - 删除这些触发器原型

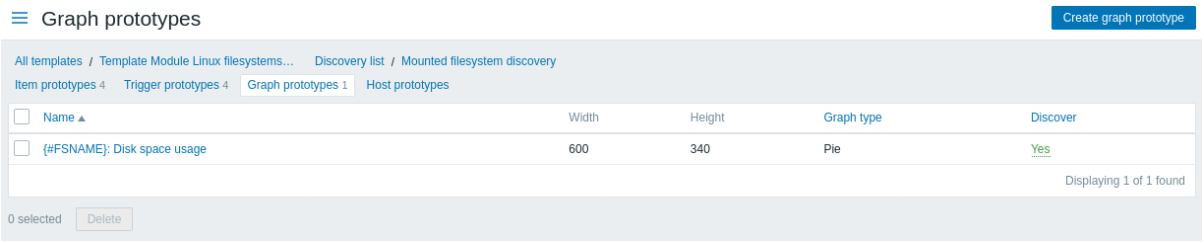
要使用这些选项，勾选对应的触发器原型复选框，然后单击所需的按钮。

3 图形原型

概述

在本段中，模板中低级发现规则的配置图形原型展示如下。

如果模板链接到主机，图形原型将成为在低级发现期间创建真实主机图形的基础。



展示的数据:

列	描述
Name	图形原型的名称，展示位一个蓝色链接。 点击名称打开图形原型配置格式。 如果图形原型属于一个关联的模板，模板名称显示在图形名称前面，为灰色链接。在关联的模板层面点击模板链接将打开图形原型列表.
Width	图形原型的宽度如所示。
Height	图形原型的高度如所示。
Type	图形原型的类型如所示- *Normal*, *Stacked*, *Pie* 或者 *Exploded*.
Discover	基于这个原型发现图形: **Yes** - 发现 **No** - 没发现。你可以通过点击’Yes’ 和’No’ 来切换。

要配置新的图形原型，点击右上角的 *Create graph prototype* 。

批量编辑选项

列表下的按钮提供了一些批量编辑的选项：

- *Delete* - 删除这些图形原型

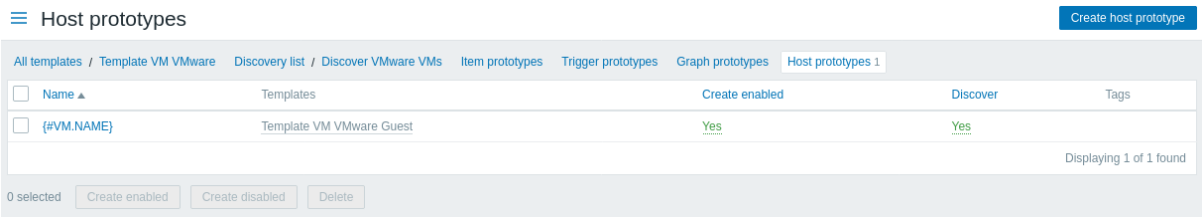
要使用这些选项，请在相应的图形原型之前标记复选框，然后单击所需的按钮。

4 主机原型

概述

在本节中，将显示在模板上配置的低级发现规则的主机原型。

如果模板链接到主机，主机原型将成为低级发现时创建真实主机的基础。



显示数据：

列	描述
Name	主机原型的名称，显示为蓝色链接。 单击名称会打开主机原型配置表单。 如果主机原型属于一个关联的模板，模板名称显示在主机名称前面，作为灰色链接。 单击模板链接将打开链接模板级别的主机原型列表。
Templates	显示主机原型的模板。
Create enabled	基于此原型创建主机: **Yes** - 启用 **No** - 禁用。你可以通过点击来切换'Yes' 和'No'
Discover	基于此原型发现主机: **Yes** - 发现 **No** - 没发现. 你可以通过点击来切换'Yes' 和'No'
Tags	显示主机原型的标签。

要创建新的主机原型，点击右上方的 *Create host prototype* 按钮。

批量编辑选项

列表下放的按钮提供了一些批量编辑的选项：

- *Create enabled* - 将这些主机创建为 *Enabled*
- *Create disabled* - 将这些主机创建为 *Disabled*
- *Delete* - 删除这些主机原型

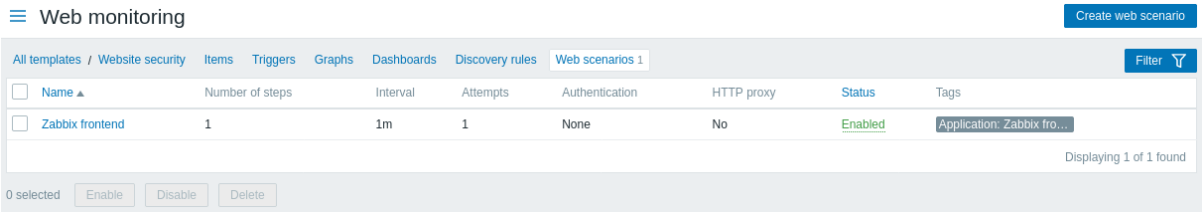
要使用这些选项，请在相应主机原型之前勾选复选框，然后单击所需按钮。

5 Web 场景

概述

模板的 web 场景列表可以从配置 → 模板点击相应的模板的 *web* 访问。

显示已存在的 web 场景列表。



显示数据:

列	描述
名称	web 场景名称。点击 web 场景名称打开 web 场景配置表单。 如果 web 场景是从其他模板继承的，则 web 场景名称前以灰色链接显示模板名称。点击模板链接将打开该模板级别的 web 场景列表
步骤数	场景包含的步骤数
更新周期	场景多长时间更新一次
重试次数	执行 web 场景步骤的重试次数
认证	显示认证方式-基本，NTLM on None
HTTP 代理	如果不使用 HTTP 代理，则显示 “No”
状态	Web 场景状态- 启用或禁用。 通过单击状态可以更改
标签	Web 场景标签。 最多可以显示三个标签 (名称: 键值对)。如果有更多的标签，会显示一个 “...” 链接，当标签鼠标悬停，会看到所有的标签。

要配置新的 web 场景，请单击右上角的创建 *web* 场景按钮。

批量编辑选项

下面的按钮提供了一些批量编辑选项:

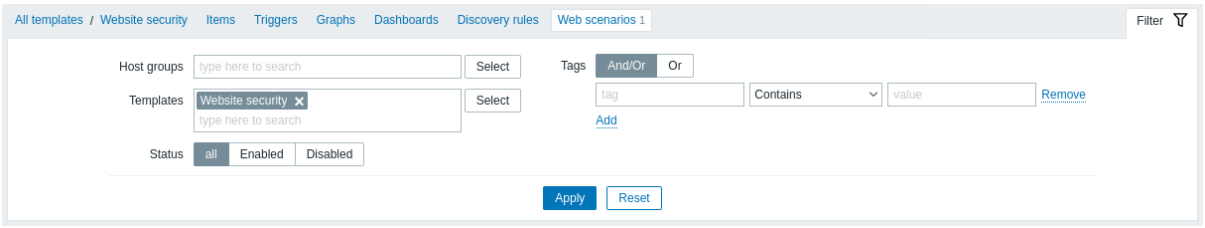
- 启用—将场景状态修改为 “启用”
- 禁用—将场景状态修改为禁用
- 删除 -删除 web 场景

要使用这些选项，请在各自的 web 场景前标记复选框，然后单击所需的按钮。

使用过滤器

您可以使用筛选器只显示您感兴趣的场景。为了获得更好的搜索性能，在搜索数据时不解析宏。

过滤器链接在 web 场景列表的上方。单击后，会出现一个过滤器，可以根据主机组、模板、状态和标签对场景进行过滤。

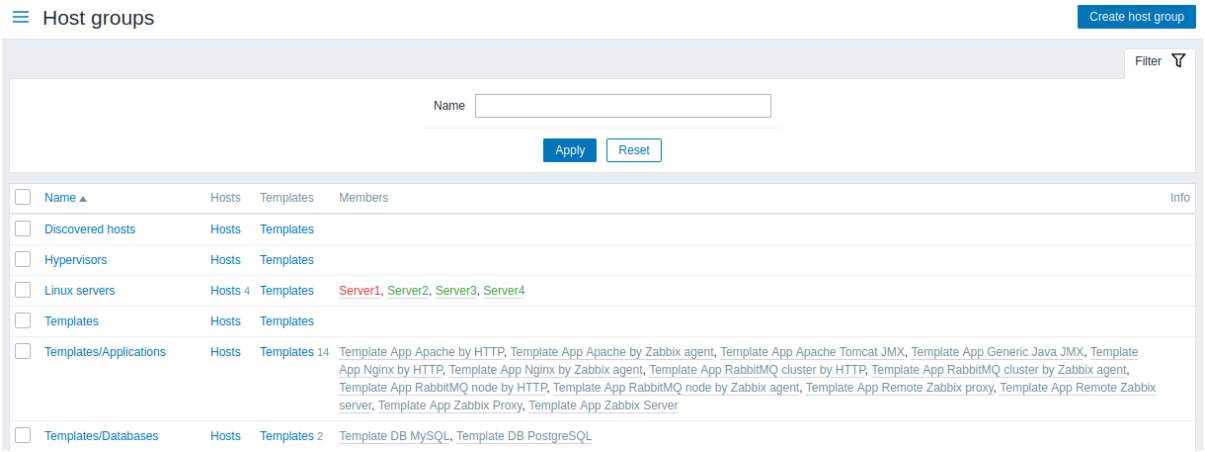


1 主机组

概述

在配置 → 主机组部分中，用户可以配置和维护主机组。主机组可以既包含模板，也包含主机。

列表将显示现有主机组及其详细信息。您可以按名称搜索和过滤主机组。



数据展示：

列名	描述
名称	主机组的名字。点击主机组名称打开主机组配置表.
主机	主机组中的主机个数 (为灰色)。单击 “主机” 将呈现属于该组的主机列表。
模板	主机组中的模板个数 (为灰色)。单击 “模板” 将呈现属于该组的模板列表。
成员	组成员的名字。模板名称为灰色，已监控的主机名显示为蓝色，未监控状态的主机名显示为红色。单击可显示 template/host 配置表。
信息	有关主机组的错误信息（如果有错误信息的话）

批量编辑选项

列表下方的按钮提供了一些批量编辑选项：

- 启用主机 - 将组中所有主机的状态更改为 “已监控”
- 禁用主机 - 将组中所有主机的状态更改为 “未监控”
- 删除 - 删除主机组

要使用这些选项，请在相应主机组之前选中复选框，然后单击所需按钮。

使用过滤器

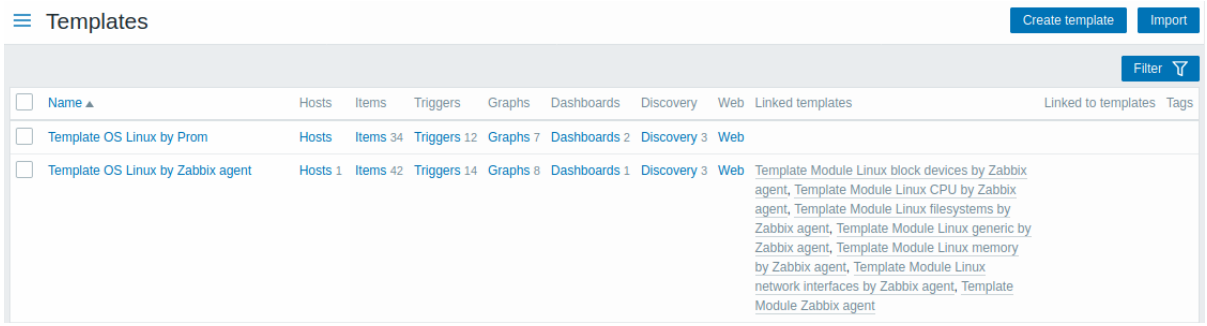
你可以通过使用过滤器仅仅展示你感兴趣的主机组。为了更好的搜索性能，搜索数据的时候不会解析宏

2 模板

概述

在配置 → 模板部分，用户可以配置和维护模板。

显示现有模板及其详细信息的列表如下：



从标题栏中的右侧的下拉列表中，您可以选择是显示所有模板还是仅显示属于组的模板。您也可以按名称搜索和过滤模板。

显示数据：

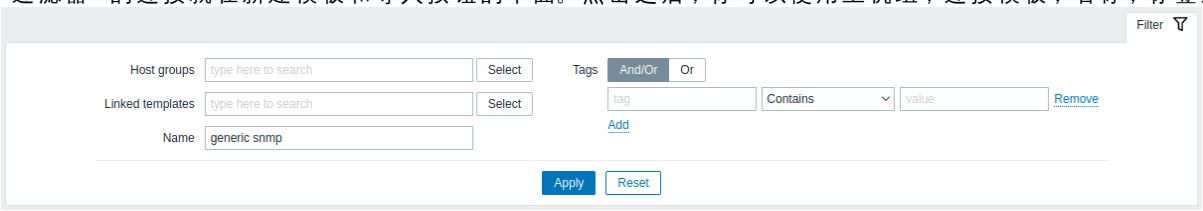
列	描述
名称	模板名称，单击模板名称打开模板配置表.
主机	模板连接的可编辑的主机；不包括只读的主机。 点击 “主机” 会打开和当前模板相关联的主机列表。
实体（监控项，触发器，图标，仪表盘，*Web*）	模板中实体的数量（以灰色显示）。在该实体的整个列表中，单击实体名将过滤掉属于该模板的实体
链接的模板	链接到当前模板的模板，在嵌套设置中，模板将继承所链接的模板的所有实体。
链接到模板	链接到的模板。继承了该模板所有试题的子模板 从 Zabbix 5.0.3 版本开始，这一列不在包括主机
标签	模板的标签 ，不支持宏解析

点击右上角创建模板按钮，即可配置一个新的模板。点击右上角导入按钮，即可从 YAML，XML，或者 JSON 文件导入模板。

使用过滤器

你可以通过过滤器，只看你感兴趣的模板。为了更好的搜索性能，在搜索数据的时候不支持宏。

“过滤器” 的连接就在新建模板和导入按钮的下面。点击之后，你可以使用主机组，连接模板，名称，标签来过滤主机组。



参数	描述
主机组	通过一个或多个主机组过滤 指定一个父主机组，即可选中所有嵌套的主机组
关联模板	过滤直接关联的模板
名称	过滤模板名称
标签	过滤模板标签名和标签值 只能通过模板标签进行过滤（集成的标签无效）。可以包含或者排除特定的标签名和标签值。可以设置几个条件。标记名称匹配必须区分大小写。 存在-包括指定的标签名 等于-包括指定的标签名称和标签值 (区分大小写) 包含-包括指定的标签名称，其中标签值包含输入的字符串 (子字符串匹配，区分大小写) 不存在-排除指定的标签名称 不等于-排除指定的标签名称和值 (区分大小写) 不包含-排除指定的标签名称，其中标签值包含输入的字符串 (子字符串匹配，区分大小写) 与/或-条件的计算类型有两种: 和/或 -如果有多个条件满足，则这些标签名相同的条件会用或拼接 或-如果其中一个条件满足就可以了

批量编辑选项

列表下方的按钮提供了一些批量编辑选项：

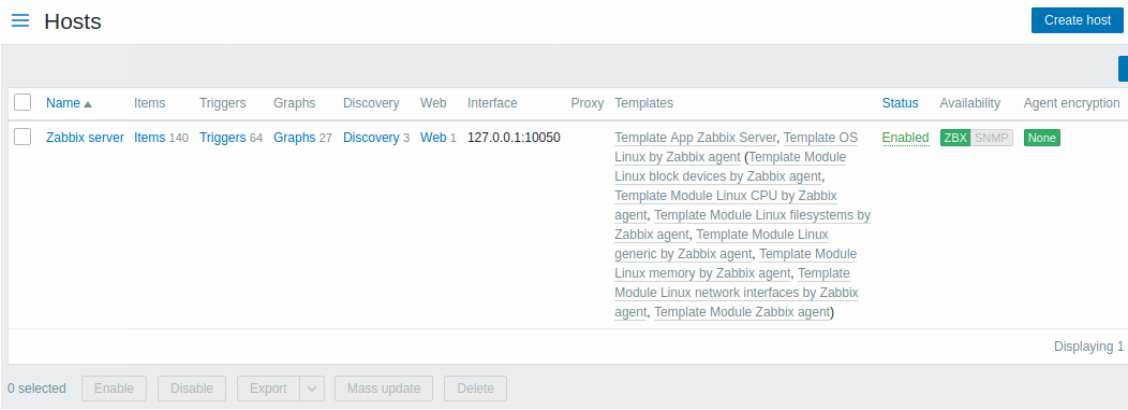
- 导出 - 将模板导出成 YAML，XML，或者 JSON 文件
- 批量更新 -一次为多个模板更新多个属性
- 删除 - 删除模板，模板与主机链接的实体（监控项，触发器等）不变
- 删除时并清理 - 删除模板及其与主机链接的实体

要使用这些选项，请在勾上相应模板前的复选框，然后单击所需的按钮。

3 主机

概览

在 “配置 → 主机” 区域，用户可以对主机进行配置和维护。



显示现有主机的列表及其详细信息。

显示数据:

列	描述
Name	主机名。单击主机名打开主机配置表单.

列	描述
实体 (监控项, 触发器, 图形, 自动发现, *Web*)	点击实体名称会显示主机的项目、触发器等。对应的实体数以灰色显示
接口	显示主机主接口
代理	如果主机通过代理监控，则显示 “代理名称”。 仅当监控途径过滤器 “任意” 或 “代理” 时，显示此字段
模板	显示主机链接的模板。如果链接模板中包含其他模板，则将它们显示在圆括号中，中间用逗号分隔。单击模板名称将打开它的配置表单。
状态	主机状态显示为- 启用或禁用。通过单击状态，您可以更改它。 橙色扳手图标 在主机状态表明该主机处于维护状态之前。当鼠标停留在图标上时，显示维护详情
可用性	主机可用性 每个配置的接口。 图标仅代表已配置的接口类型 (Zabbix agent、SNMP、IPMI、JMX)。如果你把鼠标放在图标上，就会弹出一个包含所有此类界面的列表，其中包含每个界面的详细信息、状态和错误。 对于没有接口的主机，该列为空。 以图标颜色表示当前所有接口的状态: 绿色-所有接口可用 黄色 -至少一个接口可用，至少一个接口不可用; 其他值可以是任意值，包括’unknown’ 红色 -没有可用的接口 灰色-至少有一个未知的接口 (无不可用) 注意活动的 Zabbix 代理项不影响主机可用性
Agent 加密	主机连接的加密状态: 无 -不加密 **PSK** -使用预共享密钥 **Cert** -使用证书
信息	显示主机的错误信息
标签	标签主机，未解析宏

要配置新主机，请单击右上角的新建主机按钮。要从 YAML、XML 或 JSON 文件导入主机，请单击右上角的 * import * 按钮。

批量编辑选项

下面的按钮提供了一些批量编辑选项:

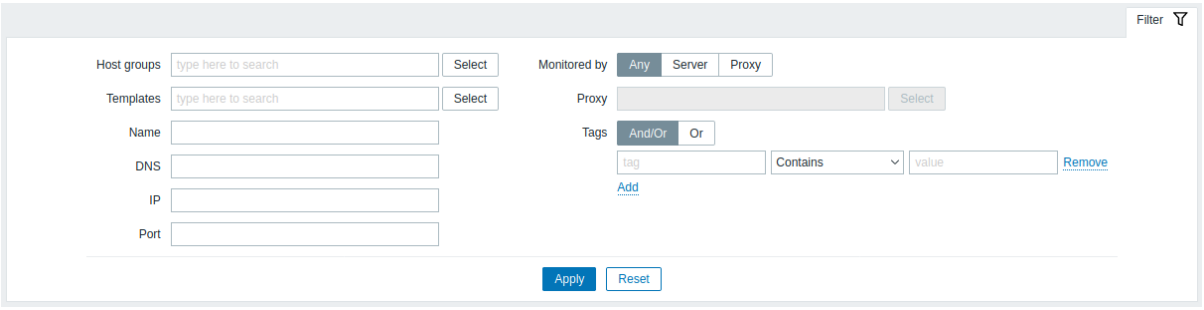
- 启用 -将主机状态修改为 “已监控”
- 禁用 -将主机状态修改为 “未监控”
- 导出 -导出主机到 YAML, XML 或 JSON 文件
- 批量更新 - 一次为多个主机更新多个属性
- 删除—删除主机

要使用这些选项，请在相应主机前勾选复选框，然后点击所需的按钮。

使用过滤器

您可以使用筛选器只显示您感兴趣的主机。为了获得更好的搜索性能，在搜索数据时不解析宏。

在主机列表的上方可以找到过滤器链接。如果单击它，可使用过滤器，根据主机组、链接模板、名称、DNS、IP、端口号 (如果它们是由服务器或代理监控的)、代理名称和标签来过滤主机。



参数	描述
主机组	过滤一个或多个主机组。 指定父主机组时，将默认选择所有嵌套的主机组

参数	描述
模板	通过链接模板过滤
名称	按可见主机名过滤
DNS	按 DNS 名称过滤
IP	按 IP 地址过滤
端口	根据端口号进行过滤
被监控	过滤只监控服务器、只监控代理或同时监控的主机
代理	过滤此处指定的代理所监控的主机
标签	根据主机标签名称和值过滤。 可以包含或排除特定的标签和标签值。可以设置多个条件。标记名称匹配必须区分大小写。 对于每个条件都有几种操作符可用: 存在 -包括指定的标签名称 等于 -包括指定的标签名称和值 (区分大小写) 包含 -包括指定的标签名称，其中标签值包含输入的字符串 (子字符串匹配，不区分大小写) 不存在 -排除指定的标签名称 不等于 -排除指定的标签名称和值 (区分大小写) 不包含 -排除指定的标签名称，其中标签值包含输入的字符串 (子字符串匹配不区分大小写) 条件的计算类型有两种: 与/或 -如果有多个条件满足，则这些标签名相同的条件会用或拼接 或 -如果其中一个条件满足

读取主机可用性

主机可用性图标反映 Zabbix 服务器上的当前主机接口状态。因此，在前端:

- 如果你禁用一个主机，可用性图标不会立即变成灰色 (未知状态)，因为服务器必须先同步配置更改;
- 如果启用主机，可用性图标不会立即变为绿色 (可用)，因为服务器必须先同步配置更改并开始轮询主机

未知接口状态

如果符合以下情况，Zabbix 服务器相应代理接口 (Zabbix、SNMP、IPMI、JMX) 的状态置为未知:

- 界面上没有启用的监控项 (它们被移除或禁用了)
- 只有 Zabbix agent 主动模式的监控项;
- 没有这种类型的接口的轮询器 (例如 StartPollers=0);
- host 被禁用;
- 如果是由代理监控时，主机被设置为由代理，另一个不同的代理或服务器监控的;
- 主机被一个可能离线的代理监控 (在最大心跳间隔-1 小时内没有收到来自代理的更新)。

在服务器配置缓存同步之后，将接口可用性设置为未知。在由代理监控的主机上恢复接口可用性 (可用/不可用) 需要在同步代理配置缓存后完成。

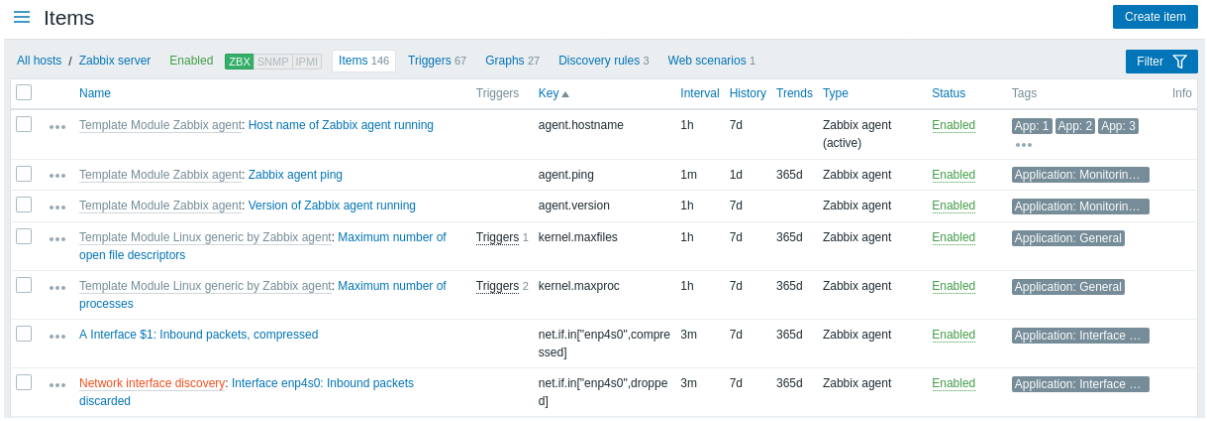
请参阅更多关于主机接口的详细信息不可触达。

1 监控项

概述

在 “配置 → 主机” 中，点击相应主机的 “项目”，可访问相应主机的项目列表。

将显示现有项目的列表。



显示数据:

列	描述
监控项菜单	单击三个点图标，打开带有这些选项的特定项目的菜单： 最新数据, 看到最新的数据项 创建触发器 —基于这个监控项创建一个触发器 触发器 - 单击此处可查看包含指向此监控项的已配置的触发器的链接的列表 新建依赖监控项 - 在该监控项基础上新建依赖监控项 新建依赖自动发现规则 - 在该监控项基础上新建依赖自动发现规则
主机	监控项所在主机。 当过滤框中选择了多个主机时，此字段才会显示
名称	监控项名称，显示为项目详细信息的蓝色链接。 点击监控项名称链接打开监控项配置表单。 如果主机条目属于某个模板，则在条目名称前以灰色链接的形式显示模板名称。单击模板链接将打开模板级的项目列表。 如果项目是从项目原型创建的，其名称前面是低级发现规则名称，用橙色表示。单击发现规则名称将打开项目原型列表
触发器	将鼠标移到触发器上，会显示一个信息框，显示与该监控项相关的触发器。 触发器个数以灰色显示
键	显示监控项的键
间隔	检查频率。 注意被动监控项也可以通过按下立即检查按钮来立即检查
历史	项目数据历史将保留多少天
趋势	项目趋势历史将保留多少天显示
类型	显示项目类型 (Zabbix agent、SNMP agent、simple check 等)
状态	显示项目状态- 启用，禁用或不支持。你可以通过点击它来改变状态-从启用到禁用 (或反过来); 从不支持到禁用 (或反过来)
标签	显示项目标签。 最多可以显示三个标签 (名称: 值对)。如果有更多的标签，一个 “…” 链接显示，允许看到所有的标签鼠标悬停。
信息	正常显示，此栏无图标。如果出现错误，则会显示一个带有字母 “i” 的方形图标。将鼠标悬停在图标上可以看到一个带有错误描述的工具提示。

要配置新项目，请单击右上角的创建项目按钮。

批量编辑选项

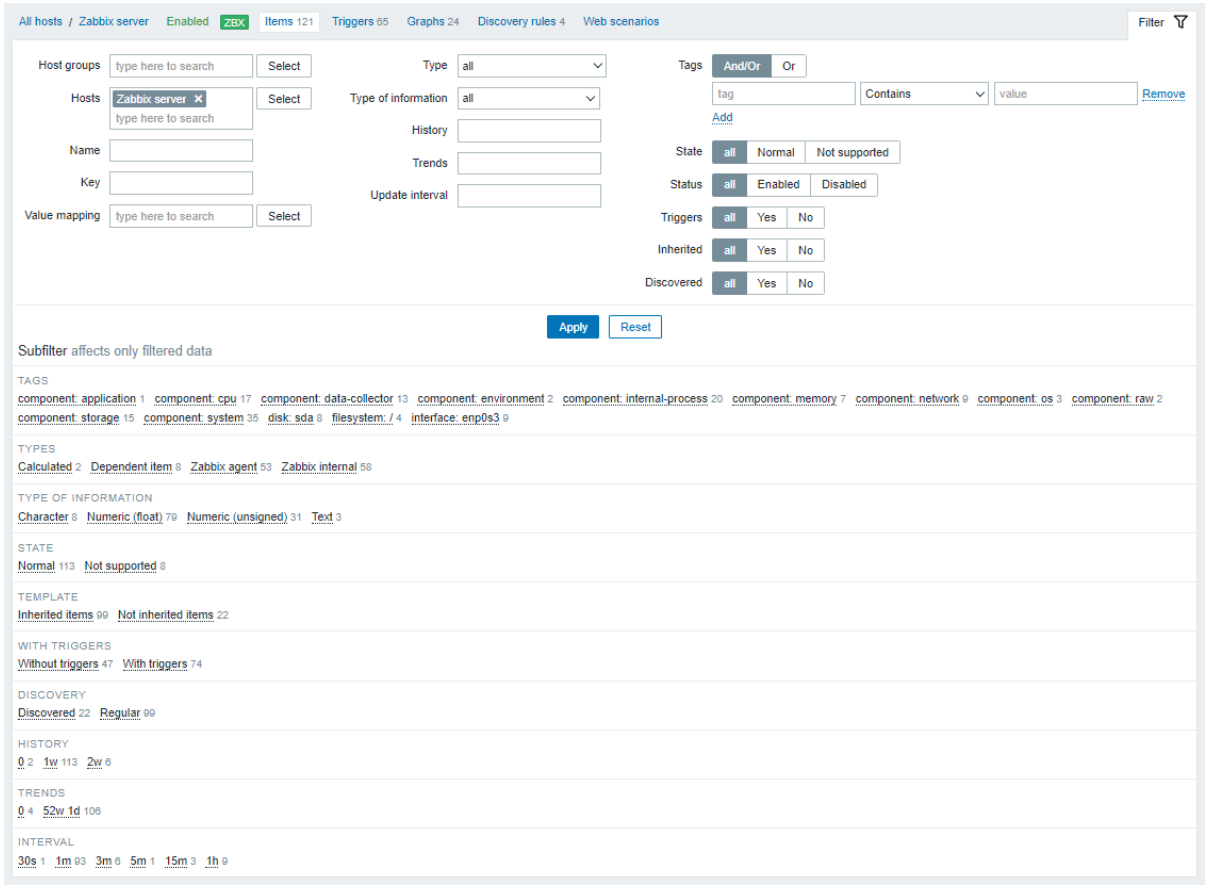
下面的按钮提供了一些批量编辑选项:

- 启用 -将主机状态修改为 “已监控”
- 禁用 -将主机状态修改为 “未监控”
- 现在检查 -立即执行检查新项目的值。仅支持被动检查 (请参阅更多细节)。请注意，当立即检查值时，配置缓存不会更新，因此这些值不会反映监控项配置的最新的更改。
- 清除历史数据—删除项目的历史和趋势数据。
- 拷贝 -拷贝监控项到其他主机或模板中。
- 批量更新 - 一次为多个监控项更新多个属性
- 删除—删除监控项

要使用这些选项，请在相应主机前勾选复选框，然后点击所需的按钮。

使用过滤器

您可以使用筛选器只显示您感兴趣的监控项。为了获得更好的搜索性能，在搜索数据时不解析宏。右上角有 *Filter* 图标。点击它将打开一个过滤器，您可以指定所需的过滤条件。



参数	描述
主机组	过滤一个或多个主机组。 指定父主机组，将默认选择所有嵌套的主机组。 不能选择只包含模板的主机组
主机	过滤一个或多个主机
名称	根据项目名称进行过滤
键	按项目键过滤
值映射	根据使用的值映射过滤。 当 “*Hosts*” 选项为空时，此参数不显示
类型	按项目类型 (Zabbix agent、SNMP agent 等) 进行过滤
信息类型	根据信息类型 (数字无符号，浮点数等) 进行过滤
历史	根据条目历史记录保存的时间进行筛选
趋势	根据项目趋势保持的时间。
更新时间间隔	按项目更新时间间隔
标签	指定标签限制显示条目的数量。可以包含或排除特定的标签和标签值。 可以设置多个条件。标记名称匹配必须区分大小写。 对于每个条件都有几种操作符可用: 存在 -包括指定的标签名称 等于 -包括指定的标签名称和值 (区分大小写) 包含 -包括指定的标签名称，其中标签值包含输入的字符串 (子字符串匹配，不区分大小写) 不存在 -排除指定的标签名称 不等于 -排除指定的标签名称和值 (区分大小写) 不包含 -排除指定的标签名称，其中标签值包含输入的字符串 (子字符串匹配不区分大小写) 条件的计算类型有两种: 与/或 -如果有多个条件满足，则这些标签名相同的条件会用或拼接 或 -如果其中一个条件满足
支持状态	按项目状态过滤- 正常或不支持
状态	按项目状态过滤- 启用或禁用
触发器	过滤有 (或没有) 触发器的项目

参数	描述
继承	从模板中继承 (或不继承) 的过滤项
发现	过滤底层发现 (或未发现) 的项目

过滤器下面的子过滤器提供了进一步的过滤选项 (对于已经过滤的数据)。可以选择具有公共参数值的项目组。在单击一个组时，它将突出显示，只有具有此参数值的项目保留在列表中。

Using subfilter

The subfilter allows to further modify the filtering from the main filter.

It contains clickable links for a quick access to related items. Items are related by common entity - tag name or value, item type, item state, etc. When an entity is clicked, the entity is highlighted with a gray background, and items are immediately filtered (no need to click *Apply* in the main filter). Clicking another entity adds it to the filtered results. Clicking the entity again removes the filtering.

Subfilters are generated based on the filtered data, which is limited to 1000 records. If you want to see more records in the subfilter, you need to increase the value of *Limit for search and filter results* parameter (in *Administration -> General -> GUI*).

Unlike the main filter, the subfilter is updated with each table refresh request to always have up-to-date information of available filtering options and their counter numbers.

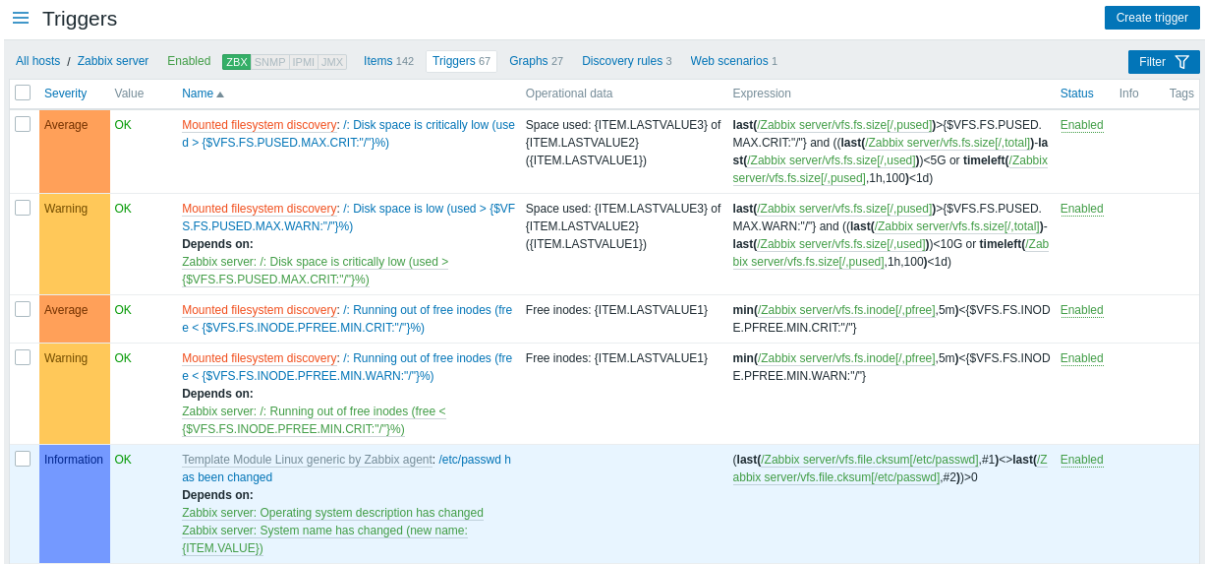
For each entity group (tags, tag values, etc.), up to 10 rows of entities are displayed. If there are more entities, this list can be expanded to display a maximum of 1000 entries (the value of SUBFILTER_VALUES_PER_GROUP in frontend definitions) by clicking the three-dot icon at the end of the list. For *Tag values*, the list can be expanded to display a maximum of 200 tag names with their corresponding values. Note that once fully expanded, the list cannot be collapsed.

A number next to each clickable entity indicates the number of items grouped in it (based on the results of the main filter). When an entity is clicked, the numbers with other available entities are displayed with a plus sign indicating how many items may be added to the current selection. Entities without items are not displayed unless selected in the subfilter before.

2 触发器

概述

在“配置 → 主机”中，点击对应主机的触发器 *，可查看对应主机的触发列表。



显示数据:

列	描述
严重级别	该触发器的级别，通过名称和单元格背景颜色显示
值	显示触发值: **OK**—触发状态为 OK。 问题—触发状态为问题
主机	触发器所在主机。 当过滤框中选择了多个主机时，此字段才会显示

列	描述
名称	触发器名称，显示为蓝色链接，表示触发器详细信息。 点击触发器名称链接打开触发器配置表单。 如果主机触发器属于模板，则模板名称显示在触发器名称前面，为灰色链接。单击模板链接将打开模板级别的触发器列表。 如果触发器是从触发器原型创建的，其名称前面是低级自动发现规则名称，用橙色表示。单击发现规则名称将打开触发器原型列表
运行数据	触发器的运行数据定义，包含将在监控 → 问题中动态解析的任意字符串和宏
表达式	显示触发器表达式，表达式的主机-监控项部分以链接的形式显示，导向监控项的配置表单。
状态	触发状态- 启用、禁用或位置。通过点击状态，您可以改变它-从启用到禁用 (或反过来); 从未知到禁用 (或反过来)。 触发器被禁用后，不再在前端显示问题，但不删除
信息	正常情况下，此列无图标。如果出现错误，则会显示一个带有字母 “i” 的方形图标。将鼠标悬停在图标上可以看到一个带有错误描述的工具提示。
标签	当触发器包含标签时，此列显示标签名称和值

如需配置新触发器，请单击右上角的 “创建触发器” 按钮。

批量编辑选项

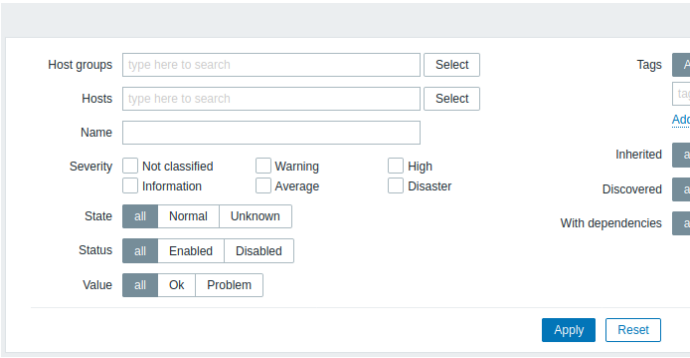
下面的按钮提供了一些批量编辑选项:

- 启用 -将主机状态修改为 “已监控”
- 禁用 -将主机状态修改为 “未监控”
- 复制 -将触发器复制到其他主机或模板。
- 批量更新 - 一次为多个主机更新多个属性
- 删除—删除主机

要使用这些选项，请在相应主机前勾选复选框，然后点击所需的按钮。

使用过滤器

您可以使用筛选器只显示您感兴趣的触发器。为了获得更好的搜索性能，在搜索数据时不解析宏。



右上角有国旅汽车图标。点击它将打开一个过滤器，您可以指定所需的过滤条件。

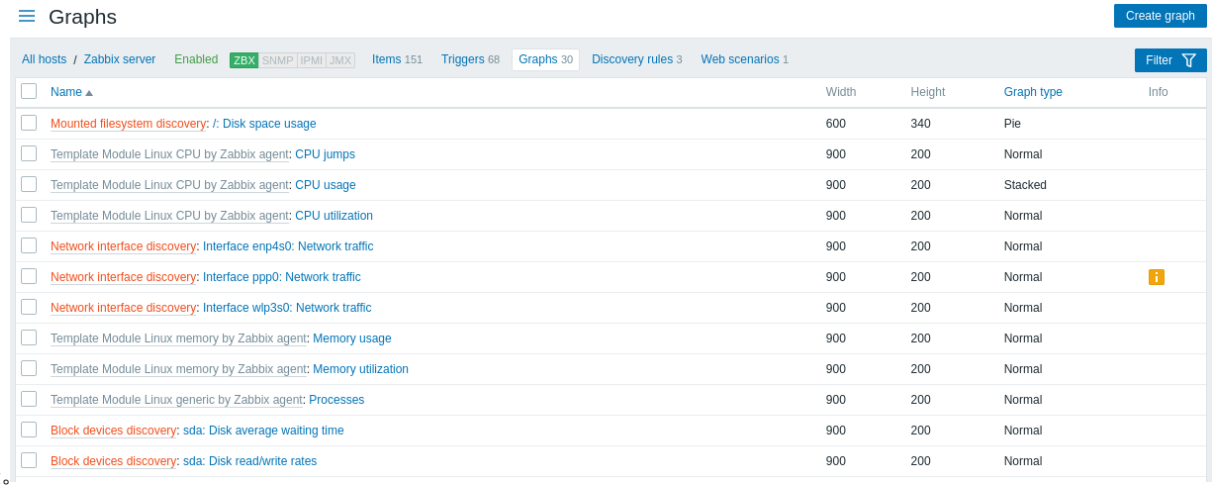
参数	描述
主机组	过滤一个或多个主机组。 指定父主机组时，将默认选择所有嵌套的主机组。 不能选择只包含模板的主机组
主机	过滤一个或多个主机。 如果上面已经选择了主机组，则只能选择这些主机组
名称	按触发器名称筛选
严重级别	按一个或多个触发级别进行过滤
支持状态	按触发支持状态进行过滤
状态	按触发状态进行过滤
值	按触发值过滤

参数	描述
标签	按触发标记名称和值进行筛选。可以包含或排除特定的标记和标记值。 可以设置几个条件。标记名称匹配总是区分大小写。 对于每个条件都有几种操作符可用: 存在 -包括指定的标签名称 等于 -包括指定的标签名称和值 (区分大小写) 包含 -包括指定的标签名称，其中标签值包含输入的字符串 (子字符串匹配，不区分大小写) 不存在 -排除指定的标签名称 不等于 -排除指定的标签名称和值 (区分大小写) 不包含 -排除指定的标签名称，其中标签值包含输入的字符串 (子字符串匹配不区分大小写) 条件的计算类型有两种: 与/或 -如果有多个条件满足，则这些标签名相同的条件会用或拼接 或 -如果其中一个条件满足
继承	从模板中继承 (或不继承) 的过滤器
已发现	过滤底层发现 (或未发现) 的触发器
依赖项	过滤有 (或没有) 依赖项的触发器

3 图形

概述

主机的自定义图形列表可在配置 → 主机中点击相应主机的 *Graphs* 查看。



显示现有图形的列表。

显示数据:

列	描述是
名称	自定义图形名称，以蓝色链接显示，链接到图形的详细信息。 点击图形名称链接打开图形配置窗体。 如果主机图属于某个模板，则模板名以灰色链接显示在图名前面。单击模板链接将打开模板级的图形列表。 如果图是从图原型创建的，它的名称前面是低级发现规则名称，用橙色表示。单击发现规则名称将打开图形原型列表
宽度	显示图形的宽度
高度	显示图形高度
图类型	显示图类型- 正常、堆叠、饼状或分解
信息	图形正常时，此列无图标。如果出现错误，则会显示一个带有字母 “i” 的方形图标。将鼠标悬停在图标上可以看到一个带有错误描述的工具提示。

若要配置新图形，请单击右上角的 “* 创建图形” 按钮。

批量编辑选项

下面的按钮提供了一些批量编辑选项:

- 复制 -将图形复制到其他主机或模板中
- 删除 -删除图形

要使用这些选项，请在各自的图形前标记复选框，然后单击所需的按钮。

使用过滤器

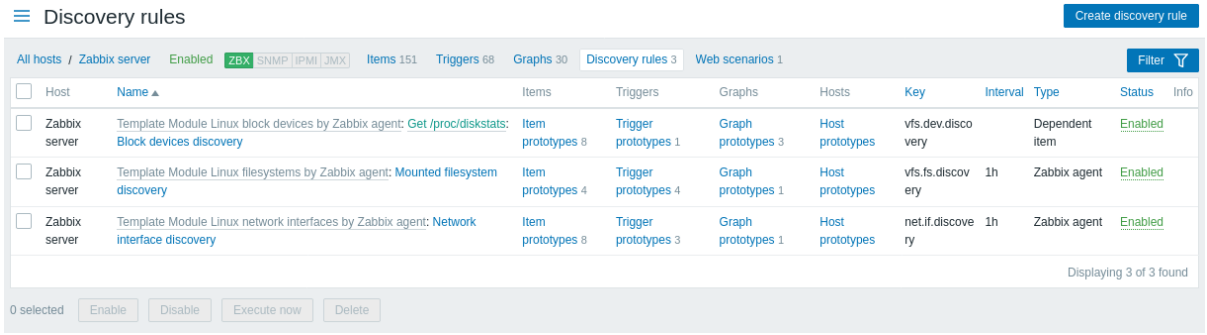
可以根据主机组和主机对图形进行过滤。为了获得更好的搜索性能，在搜索数据时不解析宏。

4 自动发现规则

概述

单击主机的“自动发现”，可在“配置 → 主机”中查看主机的低级别自动发现规则列表。

显示已存在的低级别自动发现规则列表。也可以独立于主机查看所有自动发现规则，或者通过更改过滤器设置，查看特定主机组的所有自动发现规则。



显示数据:

列	描述
Host	显示可见的主机名。 如果没有可见的主机名，则显示技术主机名
名称	规则名称，显示为蓝色链接。 单击规则名打开低级发现规则配置表单。 如果发现规则属于某个模板，则模板名称以灰色链接显示在规则名称前面。单击模板链接将打开模板级别的规则列表。
监控项	显示到监控项原型列表的链接。 已存在的监控项原型数量以灰色显示
触发器	显示触发器原型列表的链接。 已存在的触发器原型数量以灰色显示
图形	显示图形原型列表的链接。 当前已有的图形原型数量，以灰色显示
主机	显示主机原型列表链接。 当前主机原型数量以灰色显示
键	显示发现的监控项的键
时间间隔	显示发现频率。 注意发现也可以通过按列表下方的现在检查按钮立即执行
类型	显示发现的项目类型 (Zabbix agent、SNMP agent 等)
状态	发现规则状态- 已启用、未启用或不支持。通过点击状态，您可以改变它-从启用到禁用 (或反过来); 从不支持到禁用 (或反过来)
信息	正常情况下，此列无图标。如果出现错误，则会显示一个带有字母“i”的方形图标。将鼠标悬停在图标上可以看到一个带有错误描述的工具提示。

单击右上角的“创建发现规则”按钮，可以配置新的低级别发现规则。

批量编辑选项

下面的按钮提供了一些批量编辑选项:

- 启用 -将主机状态修改为“已监控”
- 禁用 -将主机状态修改为“未监控”
- 现在检查 -立即执行检查新项目的值。仅支持被动检查 (请参阅更多细节)。请注意，当立即检查值时，配置缓存不会更新，因此这些值不会反映监控项配置的最新的更改。

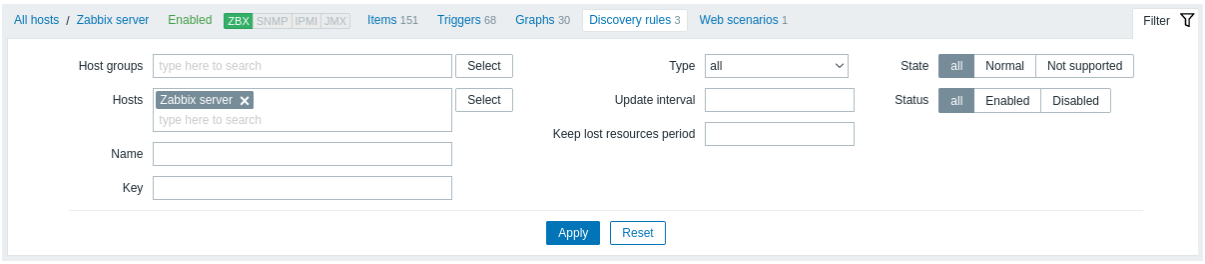
- 删除—删除监控项

要使用这些选项，请在各自的发现规则之前标记复选框，然后单击所需的按钮。

使用过滤器

您可以使用筛选器只显示您感兴趣的发现规则。为了获得更好的搜索性能，在搜索数据时不解析宏。

在发现规则列表的上方可以找到过滤器链接。如果单击它，将出现一个过滤器，可以根据主机组、主机、名称、项目键、项目类型和其他参数过滤发现规则。

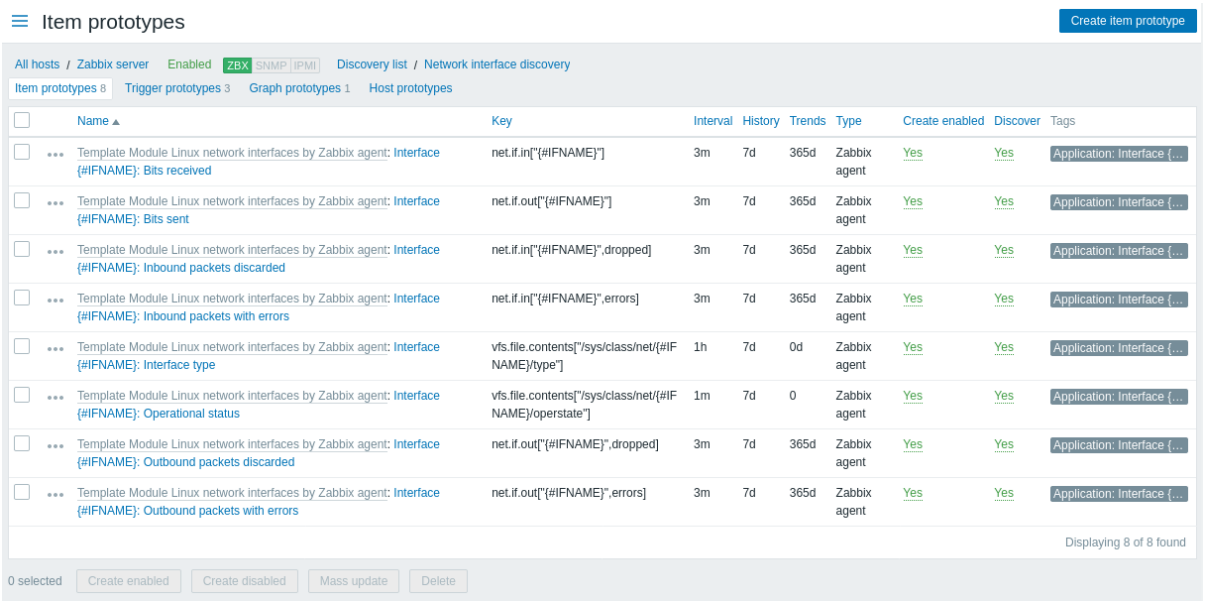


参数	描述
主机组	过滤一个或多个主机组。 指定父主机组时，将默认选择所有嵌套的主机组
主机	过滤一个或多个主机
名称	根据发现规则名称进行过滤
键	根据发现项键进行过滤
类型	根据发现项类型进行过滤
更新间隔	按更新间隔过滤。 对 Zabbix 捕获器和依赖项不可用
丢失资源保留周期	按丢失资源保留周期过滤
SNMP OID	SNMP OID 过滤。 仅当类型为 “*SNMP agent*” 时有效
支持状态	根据发现规则状态进行过滤 (支持全部/正常/不支持)
状态	根据发现规则状态进行过滤 (全部/已启用/已禁用)

1 监控项原型

概览

本节将展示主机低级别自动发现规则的监控项原型。监控项原型是主机通过低级别自动发现创建实际监控项的基础。



展示数据：

栏	描述
Name	监控项原型名称，显示为蓝色链接。 点击名称打开监控项原型配置。 如果监控项原型属于一个模板，模板名称会以灰色链接显示在规则名称之前。点击模板链接会在模板级别打开监控项原型列表。
Key	监控项原型的键值。
Interval	检查频率。
History	监控项数据历史显示的天数。
Trends	监控项趋势历史显示的天数。
Type	监控项原型显示的类型（Zabbix agent，SNMP agent，简单检查等）。
Create enabled	基于原型创建监控项为： **Yes** - 启用 **No** - 禁用。可以通过点击在 'Yes' 和 'No' 之间切换。
Discover	基于此原型发现监控项： **Yes** - 自动发现 **No** - 不自动发现。可以通过点击在 'Yes' 和 'No' 之间切换。
Tags	监控项原型显示的标签。

点击右上角的创建监控项原型按钮配置新的监控项原型。

批量编辑选项

列表之下的按钮提供了一些批量编辑选项：

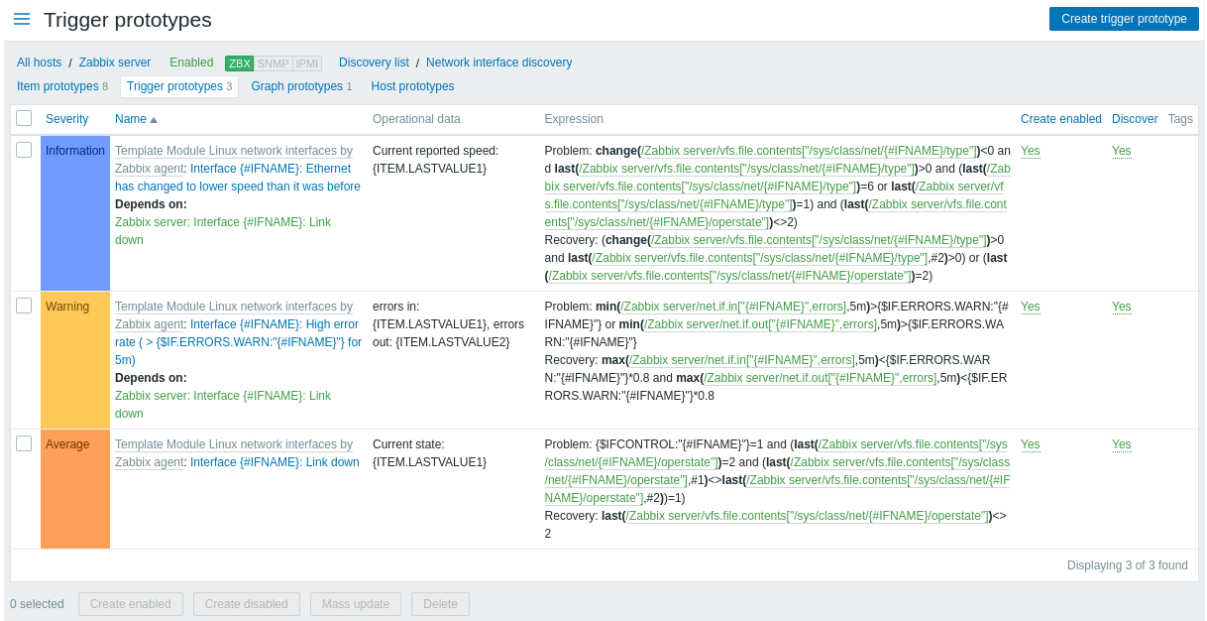
- 启用创建 - 创建启用状态监控项
- 禁用创建 - 创建禁用状态监控项
- 批量更新 - 批量更新监控项原型
- 删除 - 删除监控项原型

要使用这些选项，勾选对应的监控项原型之前的复选框，然后在所需按钮上点击。

2 触发器原型

概述

本节将展示主机低级别自动发现规则的触发器原型。触发器原型是主机通过低级别自动发现创建实际触发器 的基础。



展示数据：

栏	描述
Name	触发器原型名称，显示为蓝色链接。 点击名称打开触发器原型配置表格。 如果触发器原型属于一个模板，模板名称会以灰色链接显示在规则名称之前。点击模板链接会在模板级别打开触发器原型列表。
Operational data	所展示触发器的数据格式, 包括任意字符串和宏，会在监控 → 问题中动态解析。
Create enabled	基于此原型发现触发器： **Yes** - 启用 **No** - 禁用。可以通过点击在 'Yes' 和 'No' 之间切换。
Discover	基于此原型发现触发器： **Yes** - 自动发现 **No** - 不自动发现。可以通过点击在 'Yes' 和 'No' 之间切换。
Tags	触发器原型显示的标签。

点击右上角的创建触发器原型按钮配置新的触发器原型。

批量编辑选项

列表之下的按钮提供了一些批量编辑选项：

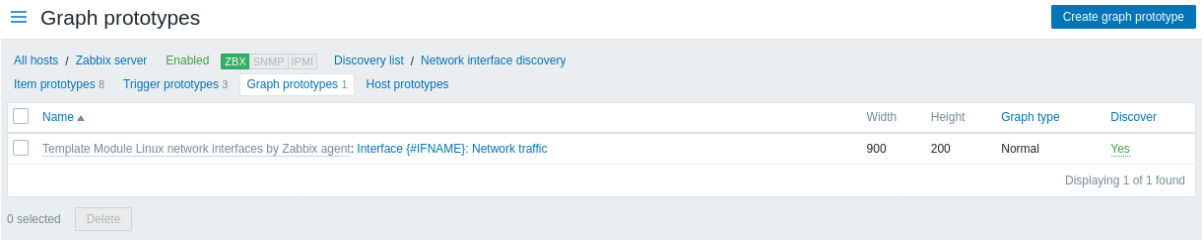
- 启用创建 - 创建启用状态触发器
- 禁用创建 - 创建禁用状态触发器
- 批量更新 - 批量更新触发器原型
- 删除 - 删除触发器原型

要使用这些选项，勾选对应的触发器原型之前的复选框，然后在所需按钮上点击。

3 图形原型

概述

本节将展示主机的低级别自动发现的图形原型。图形原型是在低级别自动发现期间创建的实际主机图形的基础。



显示数据:

列	描述
名字	图形原型的名字, 用蓝色链接显示 点击名字打开图形原型配置表。 I 如果图形原型属于链接模板，则模板名称以灰色链接的形式显示在图形名称之前。 点击模板链接的名字将打开链接模板级别的图形原型列表。
宽度	显示图形原型的宽度。
长度	显示图形原型的长度。
类型	显示图形原型的类型 - 折线图、堆叠面积图、饼图或分离型饼图。
Discover	发现基于此原型的图： 是 - 启用 discover。 否 - 不启用 discover。你可以通过点击在 "是" 和 "否" 之间切换。

要配置一个新的图形原型，点击右上角的 创建图形原型按钮。

批量编辑选项

批量编辑选项列表下面的按钮提供了一些批量编辑的选项：

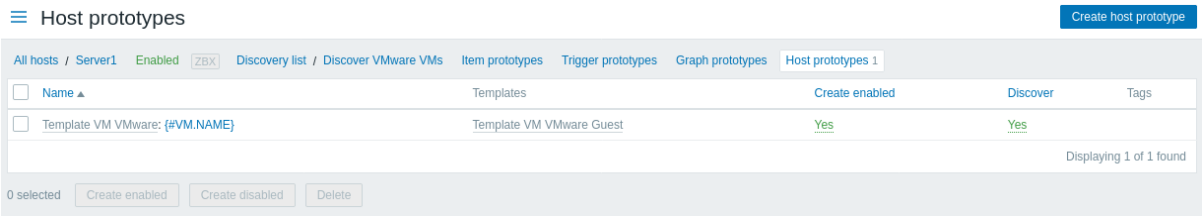
- 删除 - 删除这些图形原型。

要使用这些选项，请在相应的图形原型之前标记复选框，然后单击所需按钮。

4 主机原型

概述

此区域将显示该主机上低级别自动发现规则的主机原型。主机原型是在低级别自动发现期间创建的真正主机 的基础。



显示的数据:

列	描述
名称	主机原型的名称，显示为蓝色链接。 单击名称将打开主机原型的配置表单。 如果主机原型属于链接模板，则在主机名前显示模板名，链接显示为灰色。点击模板链接将打开链接模板级别的主机原型列表。
模板	显示主机原型的模板。
创建时启用	基于此原型创建主机 Create the host based on this prototype as: 是 - 启用 否 - 禁用。您也可以通过单击该值，在’ 是’ 和’ 否’ 之间进行切换。
发现	根据此原型发现主机: 是 - 发现 否 - 不发现。您也可以通过单击该值，在’ 是’ 和’ 否’ 之间进行切换。
标签	显示主机原型的标签信息。

单击右上角的 创建主机原型按钮，可以配置新的主机原型。

批量编辑选项

列表下方的按钮提供了批量编辑的选项：

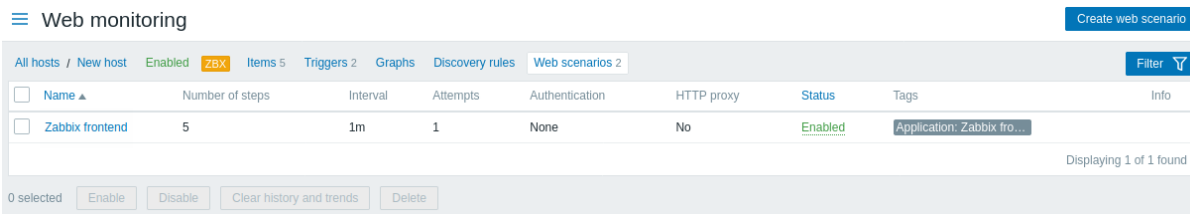
- 创建时启用 - 创建这些主机为 启用
- 创建时禁用 - 创建这些主机为 禁用
- 删除 - 删除主机原型

要使用这些选项，请在相应的主机原型之前勾选复选框，然后单击所需按钮。

5 Web 场景

概述

主机的 web 场景列表可在 “*Configuration→Hosts*” 中点击对应主机的 “*web*”。



显示已存在的 web 场景列表。

显示数据:

列	描述
名称	web 场景名称。点击 web 场景名称打开 web 场景配置表单。 如果 web 场景是从其他模板继承的，则 web 场景名称前以灰色链接显示模板名称。点击模板链接将打开该模板级别的 web 场景列表
步骤数	场景包含的步骤数
更新周期	场景多长时间更新一次
重试次数	执行 web 场景步骤的重试次数
认证	显示认证方式-基本，NTLM on None
HTTP 代理	如果不使用 HTTP 代理，则显示 “No”
状态	Web 场景状态- 启用或禁用。 通过单击状态可以更改
标签	Web 场景标签。 最多可以显示三个标签 (名称: 键值对)。如果有更多的标签，会显示一个 “...” 链接，当标签鼠标悬停，会看到所有的标签。
信息	如果一切正常，此列中没有显示图标。如果出现错误，则会显示一个带有字母 “i” 的方形图标。将鼠标悬停在图标上可以看到一个带有错误描述的工具提示。

要配置新的 web 场景，请单击右上角的创建 *web* 场景按钮。

批量编辑选项

下面的按钮提供了一些批量编辑选项:

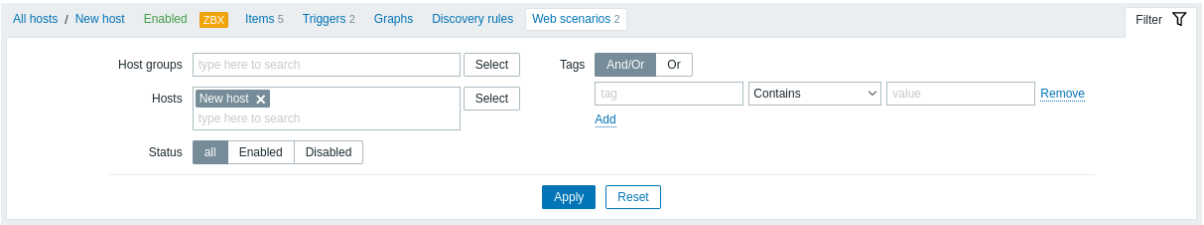
- 启用—将场景状态修改为启用
- 禁用—将场景状态修改为禁用
- 清除历史数据 — 清除场景的历史数据和趋势数据
- 删除 -删除 web 场景

要使用这些选项，请在各自的 web 场景前标记复选框，然后单击所需的按钮。

使用过滤器

您可以使用筛选器只显示您感兴趣的场景。为了获得更好的搜索性能，在搜索数据时不解析宏。

过滤器链接在 web 场景列表的上方。单击后，会出现一个过滤器，可以根据主机组、模板、状态和标签对场景进行过滤。

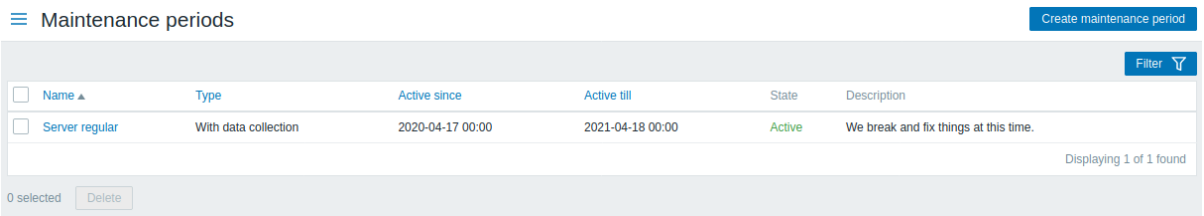


4 维护

概述

用户可以在 “配置 → 维护” 中配置和维护主机的维护期。

将显示现有维护周期的列表及其详细信息。



显示数据:

列	描述
名称	维护期名称。点击维护期名称，打开维护期配置表.
类型	维护类型: 有数据收集或无数据收集

列	描述
开始时间	执行维护周期的日期和时间变为活跃。 注: 这次不激活维护周期; 维护周期需要单独设置
活动到	维护周期停止活动的日期和时间
状态	维护期的状态: 接近 -将很快变为活跃状态 活跃 -已激活 过期 -不再活跃
描述	显示维护期的描述信息

单击右上角的 “创建维护周期” 按钮，可以配置新的维护期。

批量编辑选项

列表下方的按钮提供了一个批量编辑选项:

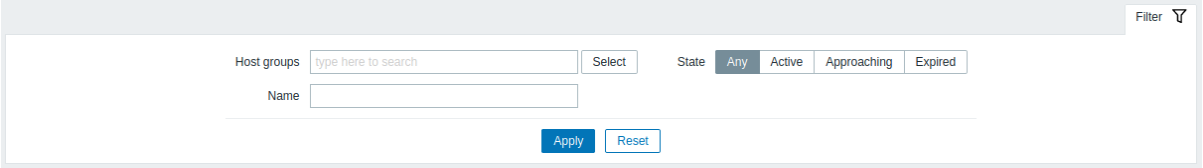
- ·删除 - 删除维护期

若要使用此选项，请在相应的维护期之前标记复选框，并单击删除。

使用过滤器

您可以使用筛选器只显示您感兴趣的维护期。为了获得更好的搜索性能，在搜索数据时不解析宏。

在维护周期列表的上方可以找到过滤器链接。如果单击它，就会出现一个筛选器，可以根据主机组、名称和状态筛选维护周期。



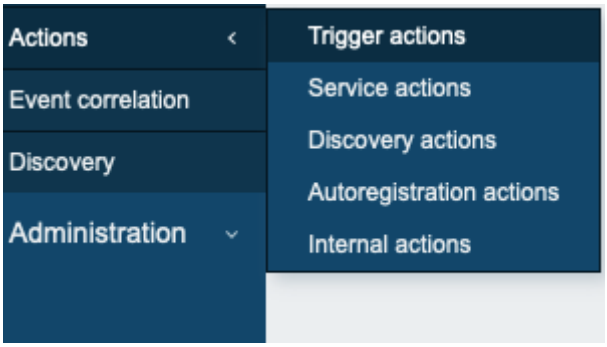
5 动作

概述

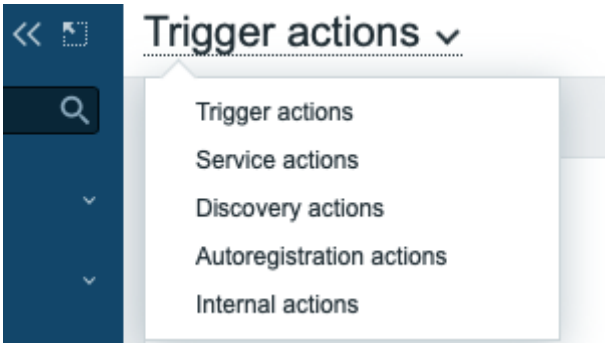
在配置 → 动作部分，用户可以配置和维护动作。

显示的动作是分配给所选事件源的动作 (触发器、服务、发现、自动注册、内部动作)。

动作按事件源 (触发器、服务、发现、自动注册、内部动作) 分组。可用子节的列表出现在按下动作在配置菜单部分。它也可以通过使用在左上角的标题下拉菜单在子部分之间切换。



Third-level menu.



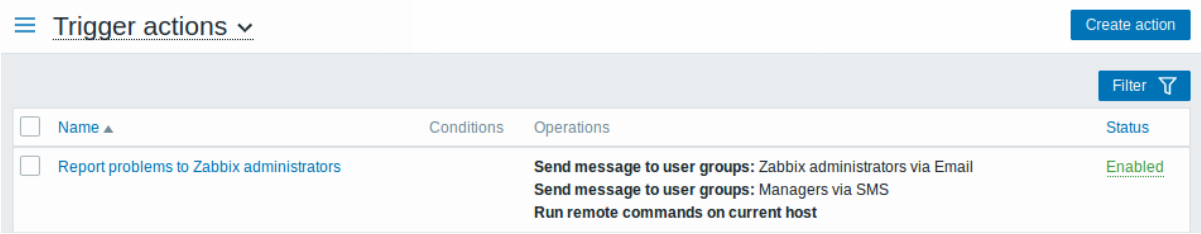
Title dropdown.

选择一个小节后，将显示一个包含现有动作及其详细信息列表的页面。

对于没有 “超级管理员” 权限的用户，根据权限设置显示动作。这意味着在某些情况下，由于某些权限限制，没有超级管理员权限的用户无法查看完整的动作列表。对于没有 “超级管理员” 权限的用户，当满足以下条件时，将显示动作:

—对主机组、主机、模板和运行条件中的触发器具有读写权限 —对动作、恢复动作、更新动作中的主机组、主机、模板具有读写权限 —用户对动作、恢复动作、更新动作中的用户组和用户具有读权限

Note:
在 *Services->Service actions* 菜单部分中，以类似的方式维护服务的操作。用户对特定服务操作的访问取决于 “访问服务” 菜单部分中设置的用户角色权限。



显示数据：

列	描述
名称	动作名称。单击动作名称打开动作配置表单.
条件	显示动作条件
动作	显示操作。 从 Zabbix 2.2 开始，操作列表还显示了用于通知的媒体类型 (电子邮件、SMS 或脚本)，以及通知接收者的姓名和姓氏 (在用户名后面的括号中)。 根据动作类型不同，动作操作可以是通知或远程指令
状态	操作状态- 启用或禁用。 通过单击状态，您可以更改它。 请参阅升级小节了解更多关于在升级过程中禁用动作会发生什么情况

要配置一个新动作，请单击右上角的新建动作按钮。

批量编辑选项

下面的按钮提供了一些批量编辑选项:

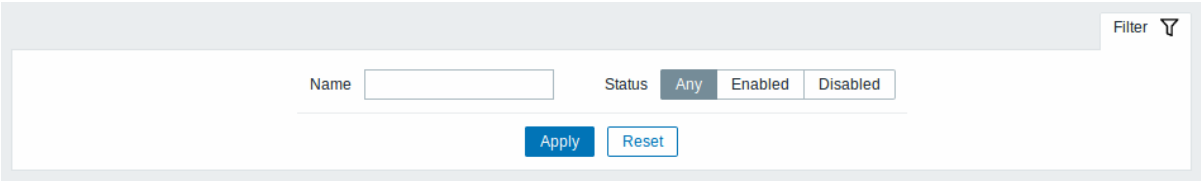
- 启用 -将动作状态修改为 “已监控”
- 禁用 -将动作状态修改为 “未监控”
- 删除 - 删除动作

要使用这些选项，请在相应主机前勾选复选框，然后点击所需的按钮。

使用过滤器

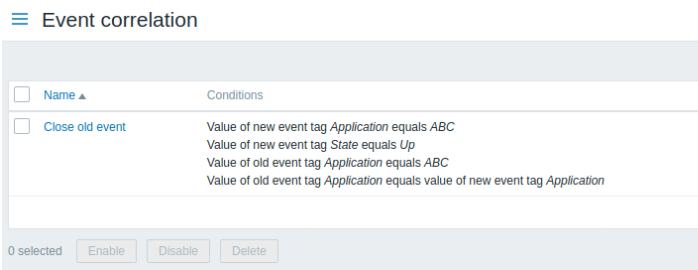
您可以使用筛选器只显示您感兴趣的动作。为了获得更好的搜索性能，在搜索数据时不解析宏。

过滤器链接可以在动作列表的上方找到。如果单击它，就会出现一个过滤器，您可以根据名称和状态过滤动作。



6 事件关联

概述



在 “配置→事件关联” 区域，用户可以配置和维护 Zabbix 事件的全局关联规则。

显示数据:

列	描述
名称	关联规则名称。单击相关规则名称将打开规则配置表单
条件	显示关联规则条件
操作	显示关联规则操作

列	描述
状态	显示相关规则状态- 启用或禁用。 通过单击状态可以更改

要配置新的关联规则，请单击右上角的 “创建关联” 按钮。

批量编辑选项

下面的按钮提供了一些批量编辑选项:

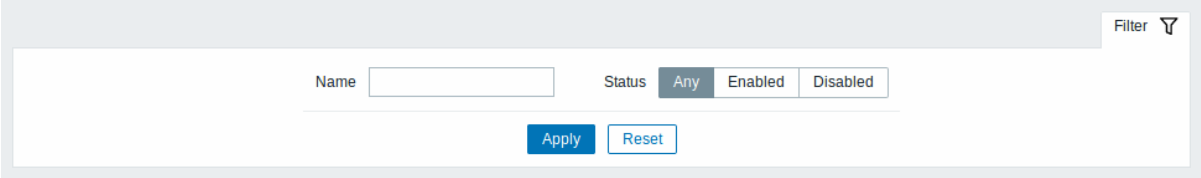
- 使能 -将关联规则的状态修改为 “启用”
- 禁用 -将关联规则的状态修改为 “禁用”
- 删除 -删除关联规则

要使用这些选项，请在相关规则之前标记复选框，然后单击所需的按钮。

使用过滤器

您可以使用筛选器只显示您感兴趣的相关规则。为了获得更好的搜索性能，在搜索数据时不解析宏。

在相关规则列表的上方可以找到过滤器链接。如果单击它，就会出现一个过滤器，您可以根据名称和状态过滤相关规则。

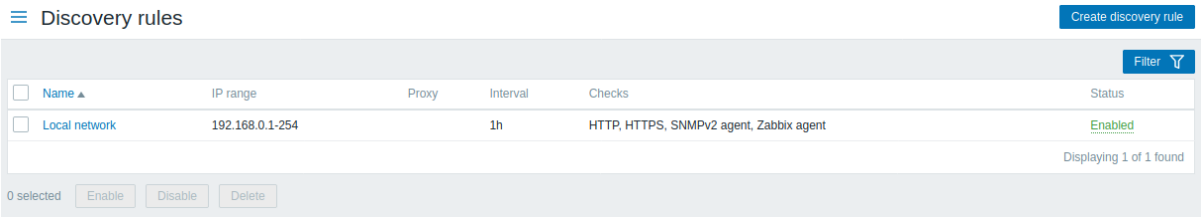


7 自动发现

概述

在 “配置 → 发现” 区域，用户可以对自动发现规则进行配置和维护。

将显示现有自动发现规则的列表及其详细信息。



显示数据:

列	描述
名称	自私发现规则的名称。单击自动发现规则名称将打开自动发现规则配置表单.
IP 地址范围	显示网络扫描使用的 IP 地址范围
代理	如果是代理执行自动发现，则显示代理名称
时间间隔	自动发现频率
检查	显示发现时使用的检查类型
状态	操作状态- 启用或禁用。 通过单击状态可以更改

单击右上角的 “创建自动发现规则” 按钮，可以配置新的自动发现规则。

批量编辑选项

下面的按钮提供了一些批量编辑选项:

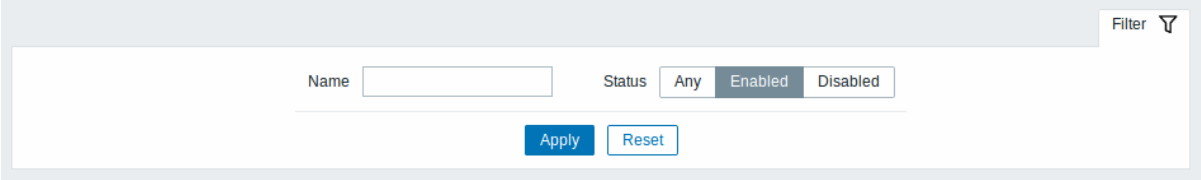
- ·启用 - 修改自动发现规则的状态为 “启用”
- ·禁用 - 将自动发现规则的状态修改为 “禁用”
- ·删除 - 删除自动发现规则

要使用这些选项，请在各自的自动发现规则之前标记复选框，然后单击所需的按钮。

使用过滤器

您可以使用筛选器只显示您感兴趣的自动发现规则。为了获得更好的搜索性能，在搜索数据时不解析宏。

在自动发现规则列表的上方可以找到过滤器链接。如果单击它，就会出现一个过滤器，您可以根据名称和状态过滤自动发现规则。



6 管理

概述

管理主菜单用于 Zabbix 的管理功能。此菜单仅适用于超级管理员组 类型的用户。

1 常规

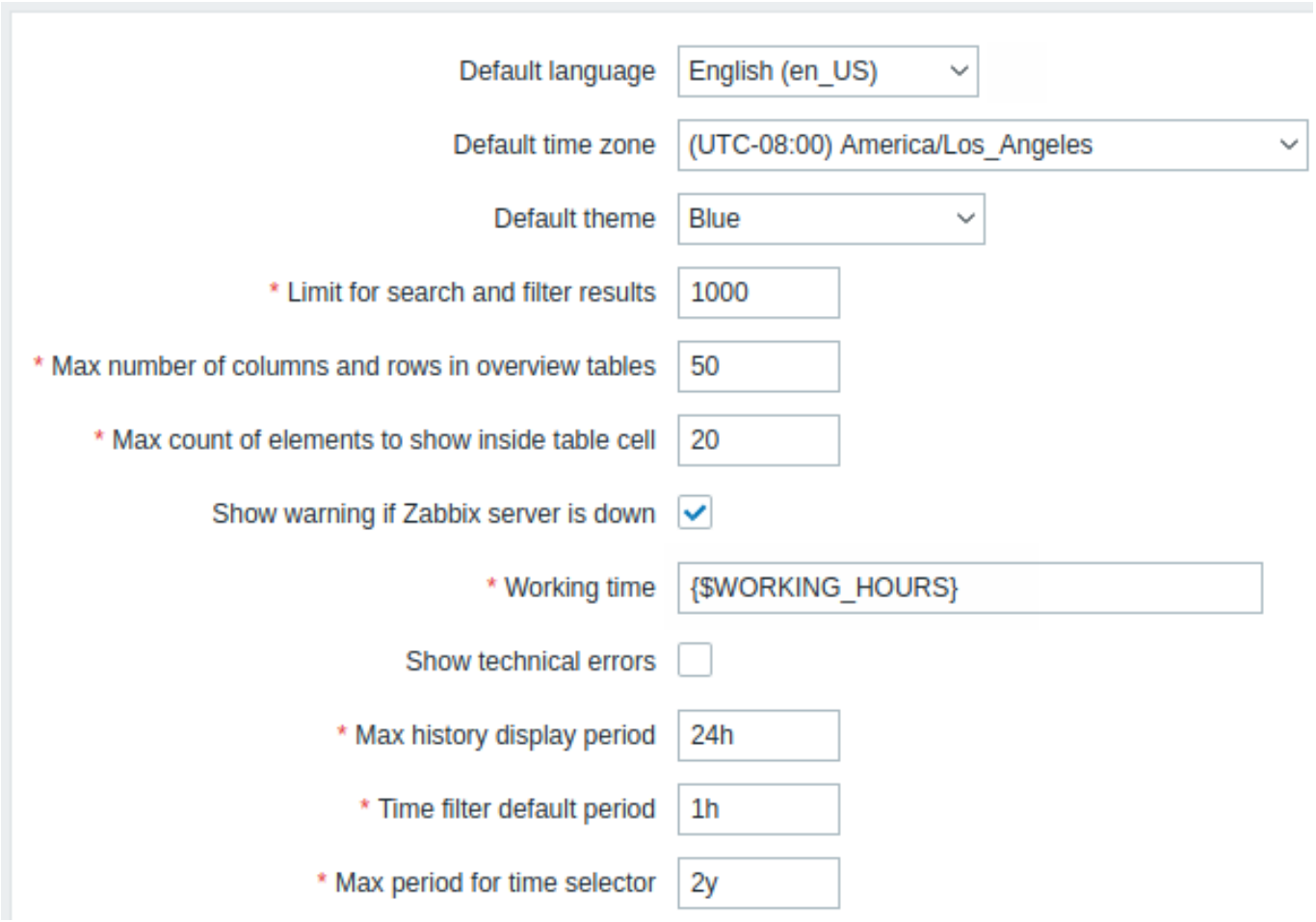
概述

Administration → *General* 部分包含许多子部分，用于设置与前端相关的默认值和自定义 Zabbix。

按下 *Administration* 菜单部分中的 *General* 后，可用子部分的列表出现。也可以使用左上角的标题下拉菜单在小节之间切换。

1 图形用户界面

本节提供了几个前端相关的定制默认值。

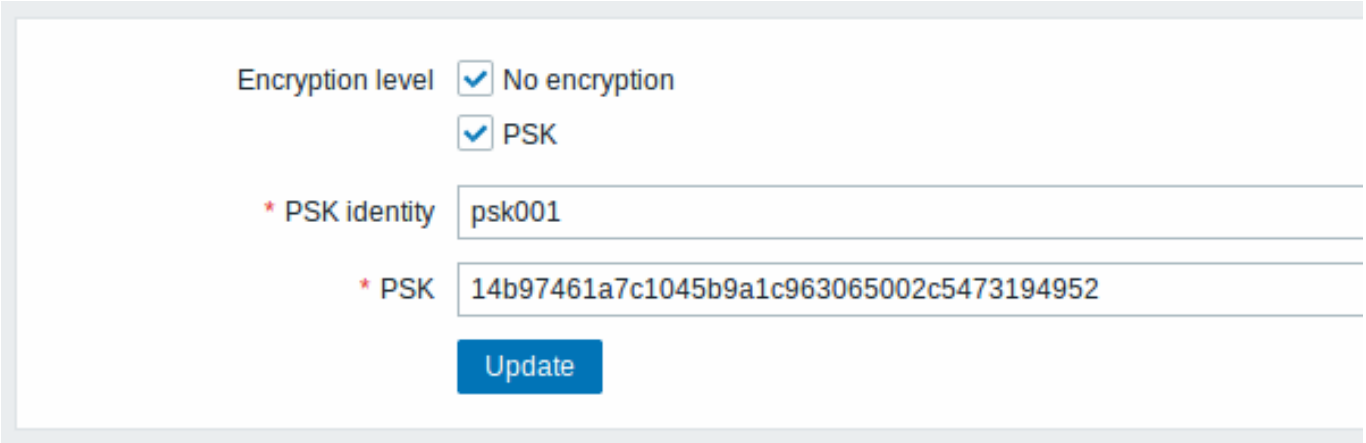


配置参数：

参数	说明
默认语言	未在其个人资料中指定语言的用户和来宾用户的默认语言。 有关详细信息，请参阅安装其他前端语言。
默认时区	未在其个人资料中指定时区的用户和来宾用户的默认时区。
默认主题	未在个人资料中指定主题的用户和来宾用户的默认主题。
搜索和过滤结果的限制	将在 Web 界面列表中显示的元素（行）的最大数量，例如，在配置 → 主机中。 注意：如果设置为，则为例如: '50'，只有前 50 个元素将显示在所有受影响的前端列表中。如果某个列表包含超过 50 个元素，则显示为 *"Displaying 1 to 50 of **50+** found"* 中的'+' 号。此外，如果使用过滤但仍然有超过 50 个匹配项，则只会显示前 50 个。
概览表中的最大列数和行数	要在数据概览和触发器概览仪表板小部件中显示的最大列数和行数。相同的限制适用于列和行。如果存在多于显示的行和/或列，系统将在表格底部显示警告："未显示所有结果。请提供更具体的搜索条件。"
最大元素计数 * * 显示在表格单元格内	对于在单个表格单元格中显示的条目，不会显示超出此处配置的数量。
如果 *Zabbix* 服务器关闭时显示警告	如果无法访问 Zabbix 服务器（可能已关闭），此参数可以在浏览器窗口中显示警告消息。即使用户向下滚动页面，该消息仍然可见。将鼠标悬停在上方时，消息会暂时隐藏以显示其下方的内容。 自 Zabbix 2.0.1 起支持此参数。
工作时间	这个系统范围的参数定义工作时间。在图表中，工作时间显示为白色背景，非工作时间显示为灰色。 有关时间格式的说明，请参见时间段规范页面。 支持用户宏（自 Zabbix 3.4.0 起）。
显示技术错误	如果选中，所有注册用户都可以看到技术错误 (PHP/SQL)。如果未选中，则该信息仅对Zabbix Super Admins 和属于启用了debug mode 的用户组的用户可用。
最大历史显示周期	在 *Monitoring* 小节中显示历史数据的最大时间周期：最新数据、*Web* 和数据概览仪表板小部件。 允许范围：24 小时（默认）- 1 周。时间后缀，例如支持 1w（一周）、36h（36 小时）。
时间过滤器默认时间段	默认情况下用于图表和仪表板的时间段。允许的范围：1 分钟 - 10 年（默认值：1 小时）。 时间后缀，例如支持 10m（十分钟）、5w（五周）。 注意：当用户在查看图表时更改时间段时，此时间段将存储为用户偏好，替换全局默认值或之前的用户选择.
时间选择器的最大时间段	图表和仪表板的最大可用时间段。用户将无法可视化旧数据。允许范围：1 年 - 10 年（默认：2 年）。 时间后缀，例如支持 1y（一年）、365w（365 周）。

2 自动注册

在本节中，您可以配置活动代理自动注册的加密级别。



标有星号的参数是强制性的。

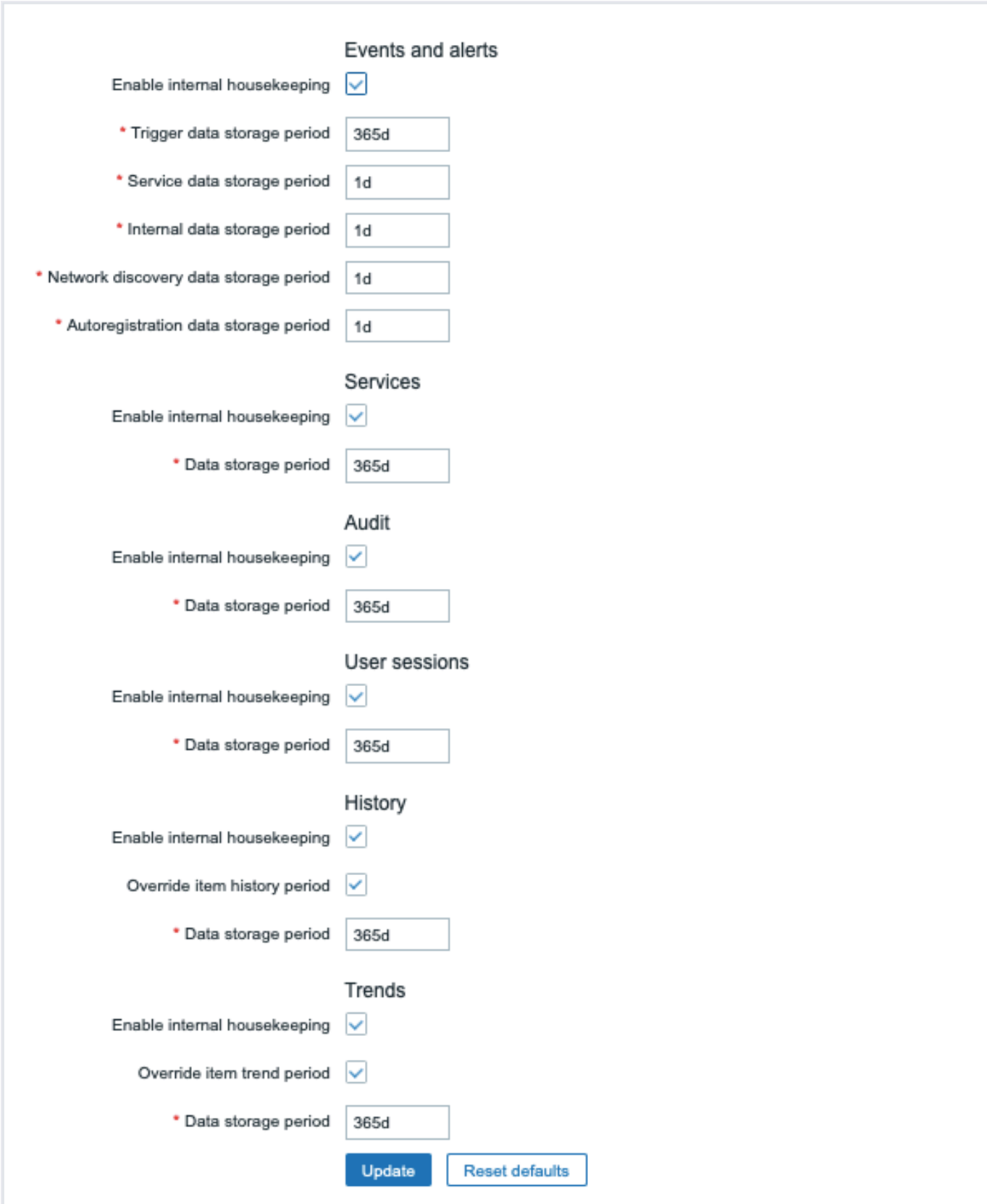
配置参数：

参数	说明
加密级别	为加密级别选择一个或两个选项： 不加密 - 允许未加密的连接 **PSK** - 允许使用预共享密钥的 TLS 加密连接
PSK 身份	输入预共享密钥身份字符串。 此字段仅在选择 “PSK” 作为加密级别时可用。 不要将敏感信息放入 PSK 身份中，它是通过网络以未加密方式传输以通知接收器使用哪个 PSK。
PSK	输入预共享密钥（偶数个十六进制字符）。 最大长度：如果 Zabbix 使用 GnuTLS 或 OpenSSL 库，则为 512 个十六进制数字（256 字节 PSK），64 个十六进制数字（32 -byte PSK) 如果 Zabbix 使用 mbed TLS (PolarSSL) 库。 示例： 1f87b595725ac58dd977beef14b97461a7c1045b9a1c963065002c5473194952 此字段仅在选择 “PSK” 作为加密级别时可用。

参阅：Secure autoregistration

3 管家服务

管家是一个周期性的进程，由 Zabbix 服务器执行。该过程删除过时的信息和用户删除的信息。



在本节中，可以针对每个任务分别启用或禁用内务管理任务：事件和警报/IT 服务/用户会话/历史记录/趋势。审核内务管理设置在单独的菜单部分 中可用。

如果启用了管家，则可以设置数据记录在被管家删除之前将保留多少天。

删除项目/触发器也会删除该项目/触发器产生的问题。

此外，只有在与问题没有任何关联的情况下，管家才会删除事件。这意味着如果一个事件是问题或恢复事件，在相关问题记录被删除之前不会被删除。管家会先删除问题，后删除事件，以避免过时的事件或潜在的问题问题记录。

对于历史和趋势，有一个附加选项可用：覆盖项目历史周期和覆盖项目趋势周期。此选项允许全局设置项目历史/趋势将保留多少天（1 小时至 25 年；或 “0”），在这种情况下覆盖为历史存储期/趋势存储期中的单个项目设置的值项目配置中的字段。请注意，对于具有配置选项

不保留历史记录和/或不保留趋势启用的项目，存储期不会被覆盖。

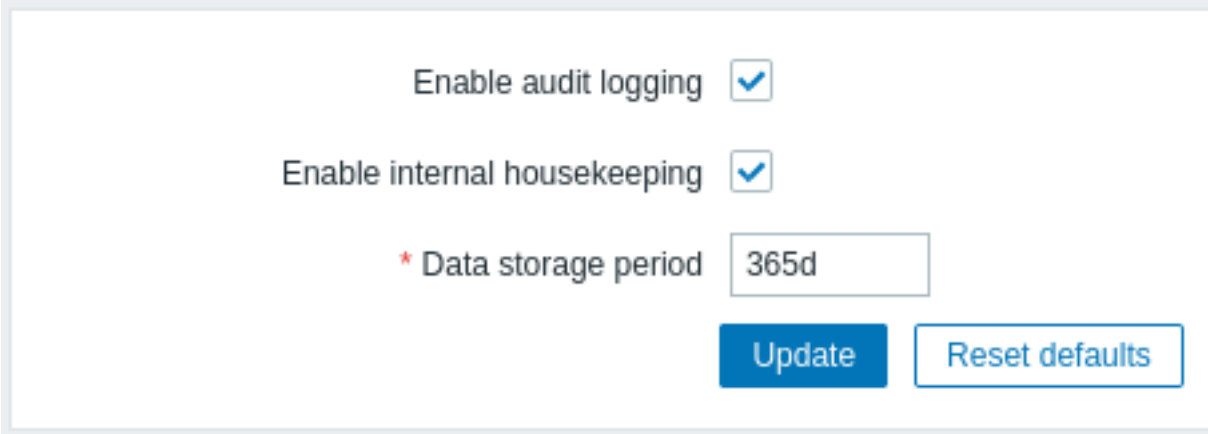
即使禁用内部管理，也可以覆盖历史/趋势存储期。因此，当使用外部管家时，可以使用历史数据存储周期字段设置历史存储周期。

Attention:
如果使用 TimescaleDB，为了充分利用 TimescaleDB 对历史和趋势表的自动分区，覆盖项目历史周期和覆盖项目趋势周期选项必须启用以及启用内部管理选项历史和趋势。否则，保存在这些表中的数据仍将存储在分区中，但是，管家将通过删除单个记录而不是删除过时的分区来清理历史记录和趋势。当启用删除过期分区时，Zabbix 服务器和前端将不再跟踪已删除的项目，并且在删除过期分区时将清除已删除项目的历史记录。

句号字段支持时间后缀，例如 1d（一天），1w（一周）。最短为 1 天（历史为 1 小时），最长为 25 年。重置默认值按钮允许恢复所做的任何更改。

4 审计日志

此部分允许配置审核日志设置。

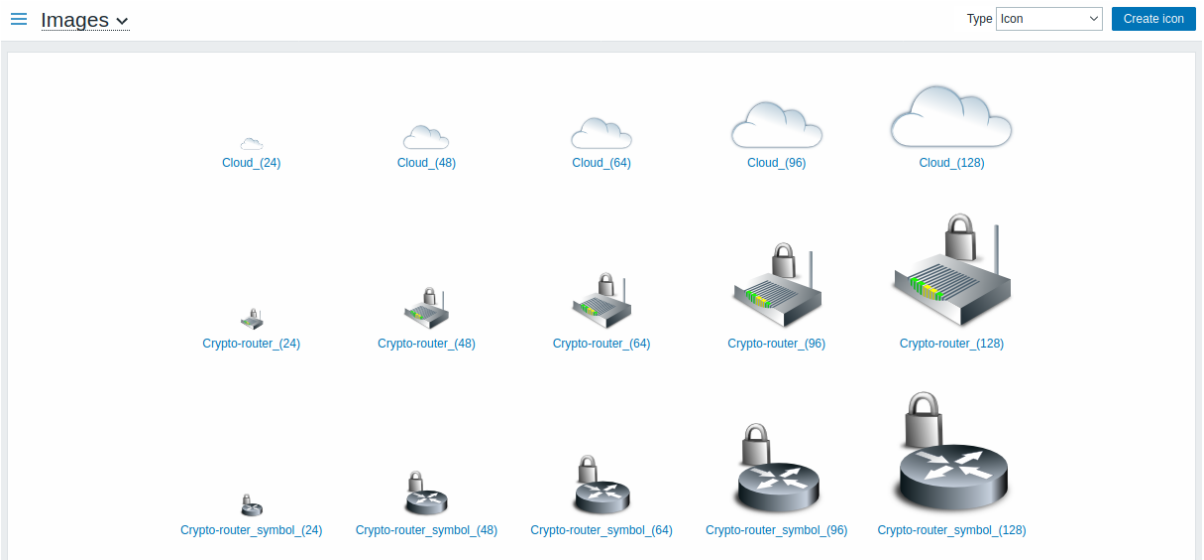


以下参数可用：

参数	说明
启用审计日志	启用/禁用审计日志。默认选中。
启用内部管理	启用/禁用内部管理以进行审计。默认选中。
数据存储期限	审核记录在被管家删除之前应保留的天数。如果启用了管家服务，则为必需项。默认值：365 天。

5 图片

图片部分显示了 Zabbix 中所有可用的图片。图像存储在数据库中。

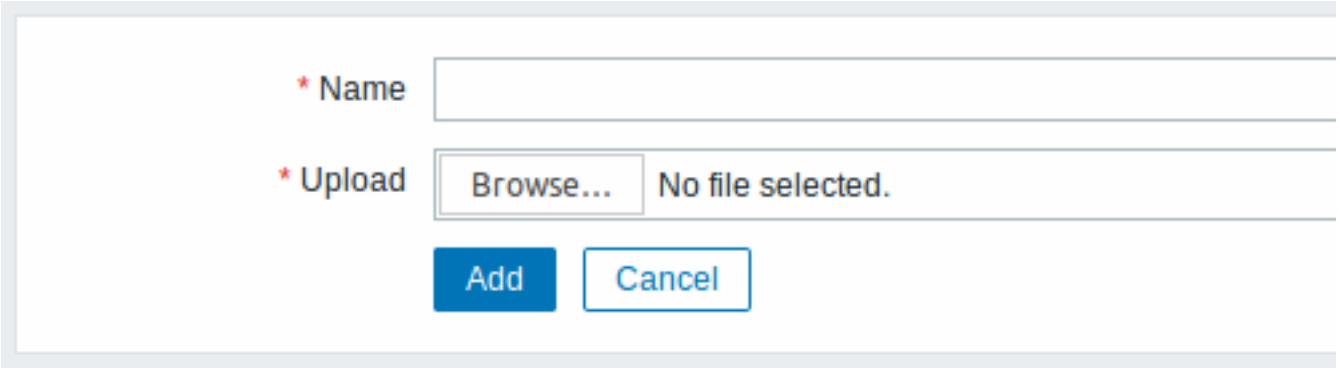


Type 下拉菜单允许您在图标和背景图像之间切换：

- ·图标用于显示network map 元素
- ·背景用作网络地图的背景图片

添加图片

您可以通过单击右上角的创建图标或创建背景按钮来添加自己的图像。



图片属性：

参数	说明
名称	图像的唯一名称。
上传	从本地系统中选择要上传到 Zabbix 的文件（PNG、JPEG、GIF）。 注意可能会上传其他在上传过程中会转换为 PNG 的格式。GD 库用于图像处理，因此支持的格式取决于使用的库版本（Zabbix 需要 2.0.28 或更高版本）。

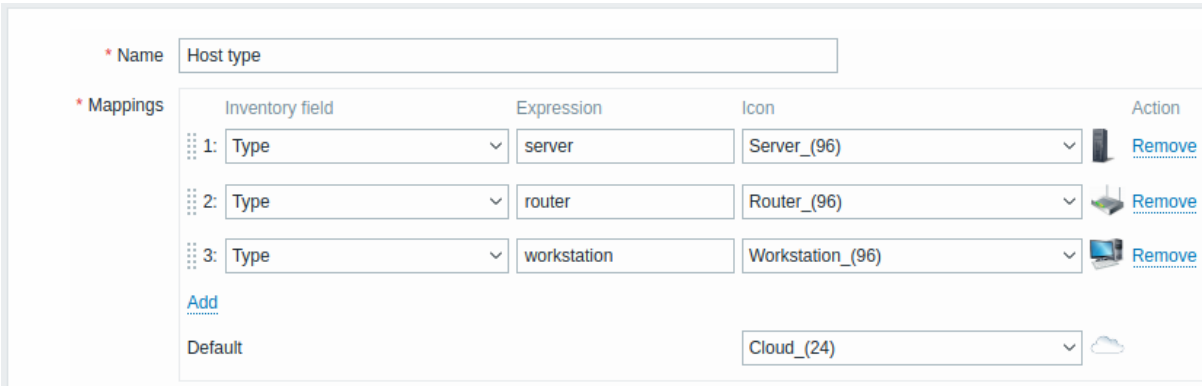
上传文件的最大大小受 ZBX_MAX_IMAGE_SIZE 值的限制，即 1024x1024 字节或 1 MB。如果图像大小接近 1 MB，并且 MySQL 配置参数的 “max_allowed_packet” 默认为 1MB，则图像上传可能会失败。在这种情况下，增加 max_allowed_packet 范围。

6 图标映射

本节允许创建特定主机与特定主机的映射图标。主机清单字段信息用于创建映射。

然后可以在网络拓扑图配置 分配自动匹配主机的适当图标。

要创建新的图标地图，请单击右上角的创建图标地图角落。



配置参数：

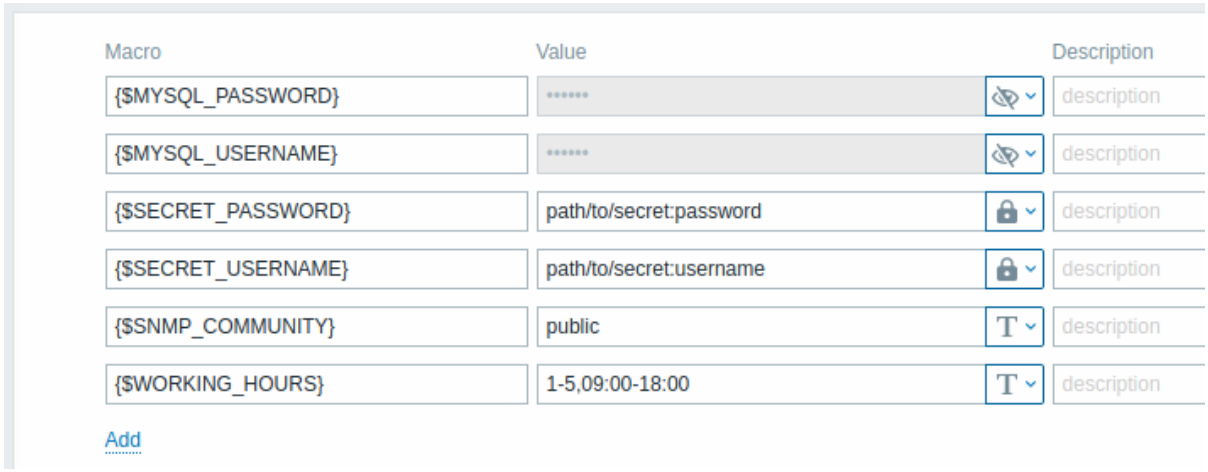
参数	说明
名称	图标图片的唯一名称。
映射	映射列表。映射的顺序决定了哪个优先。您可以通过拖放在列表中上下移动映射。
库存字段	将被调查以寻找匹配的主机库存字段。
表达式	描述匹配的正则表达式。
图标	找到表达式匹配时使用的图标。
默认	要使用的默认图标。

7 正则表达式

本节允许创建可在前端多个位置使用的自定义正则表达式。有关详细信息，请参阅Regular expressions 部分。

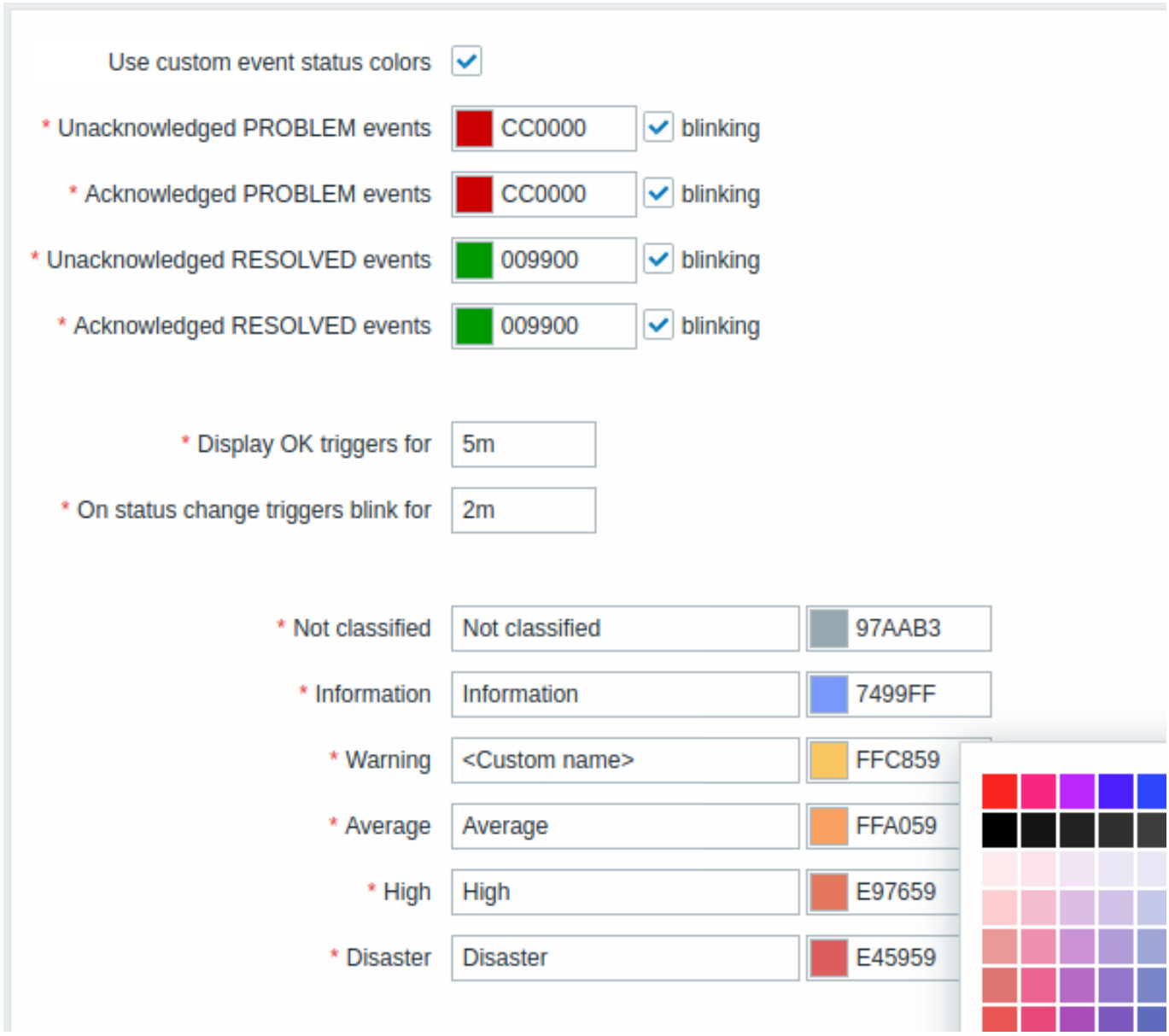
8 宏

本节允许定义系统范围的user macros 作为名称-值对。请注意，宏值可以保存为纯文本、机密文本或 Vault 机密。还支持添加描述。



9 触发器显示选项

此部分允许自定义触发状态在前端的显示方式以及trigger severity 名称和颜色。

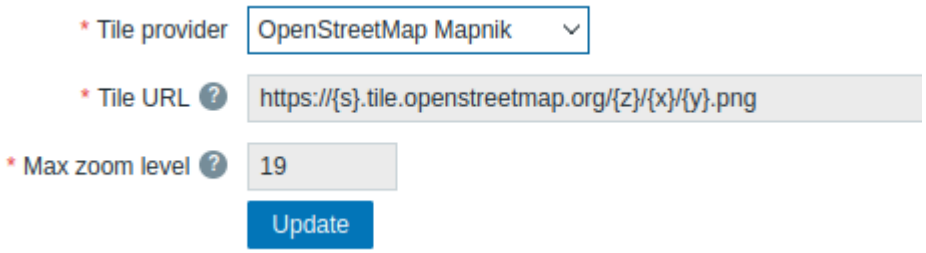


参数	说明
使用自定义事件状态颜色	选中此参数将打开已确认/未确认问题的颜色自定义。

参数	说明
未确认的问题事件, 已确认的问题事件, 未确认的已解决事件, 已确认已解决事件	输入新的颜色代码或单击颜色以从提供的调色板。 如果选中 *blinking* 复选框，触发器将在状态更改时闪烁一段时间以变得更加明显。
显示 *OK* 触发器	显示 OK 触发器的时间段。允许范围：0 - 24 小时。时间后缀，例如支持 5m、2h、1d。
在状态变化时触发器闪烁	触发器闪烁的长度。允许范围：0 - 24 小时。时间后缀，例如支持 5m、2h、1d。
未分类, 信息, 警告, 平均, 高, 灾难	自定义严重性名称和/或颜色显示而不是系统默认值。 输入新颜色代码或单击颜色以从提供的调色板中选择新颜色。 请注意，此处输入的自定义严重性名称将用于所有区域设置。如果您需要为某些用户将它们翻译成其他语言，请参阅自定义触发器严重性 页面。

10 地理地图

此部分允许为 Geomap 仪表板部件 选择地理地图切片服务提供者和配置服务提供者设置。为了使用地理地图提供可视化，Zabbix 使用开源 JavaScript 交互式地图库 Leaflet。请注意，Zabbix 无法控制第三方提供商提供的瓦片图像质量，包括预定义的瓦片提供商。



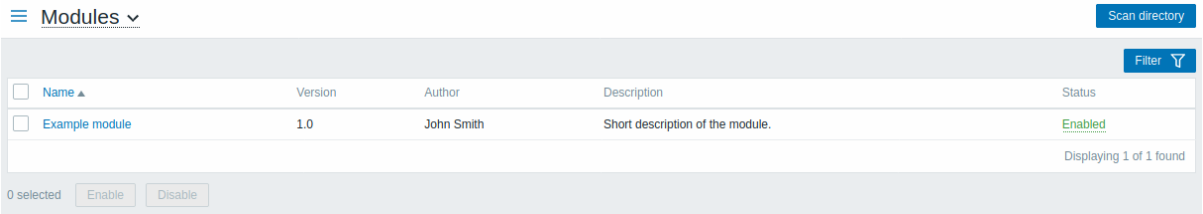
参数	说明
磁贴提供者	选择可用磁贴服务提供者之一或选择其他以添加另一个磁贴提供者或自托管磁贴（请参阅使用自定义磁贴服务提供者）。
Tile URL	用于在地图上加载和显示切片图层的 URL 模板。仅当 *Tile provider* 设置为 *Other* 时，此字段才可编辑。 支持以下占位符： *{s}* 表示可用子域之一； *{z }* 表示 URL 中的缩放级别参数； *{x}* 和 *{y}* 表示图块坐标； *{r}* 可用于在 URL 中添加 “@2x” 加载视网膜图块。 示例：`https://{s}.example.com/{z}/{x}/{y}{r}.png`
属性	要在地图上的小文本框中显示的图块提供商属性数据。仅当 *Tile provider* 设置为 *Other* 时，此字段才可编辑。
最大缩放级别	地图的最大缩放级别。仅当 *Tile provider* 设置为 *Other* 时，此字段才可编辑。

使用自定义瓦片服务提供者

Geomap 小部件能够从自定义的自托管或第三方切片提供商服务加载光栅切片图像。要使用自定义第三方切片提供商服务或自托管切片文件夹或服务器，请在 *Tile provider* 字段中选择 *Other*，并在 *Tile URL* 字段中使用适当的占位符指定自定义 URL。

11 模块

本节允许管理自定义前端模块。

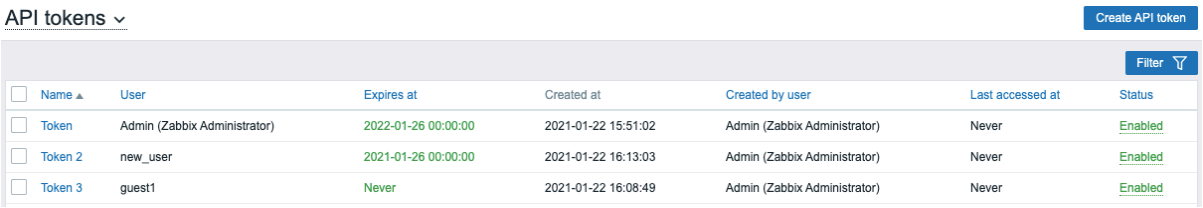


单击 *Scan directory* 以注册/取消注册任何自定义模块。已注册的模块及其详细信息将显示在列表中。未注册的模块将从列表中删除。

您可以按名称或状态（启用/禁用）过滤模块。单击列表中的模块状态以启用/禁用模块。您还可以通过在列表中选择模块然后单击列表下方的启用/禁用按钮来批量启用/禁用模块。

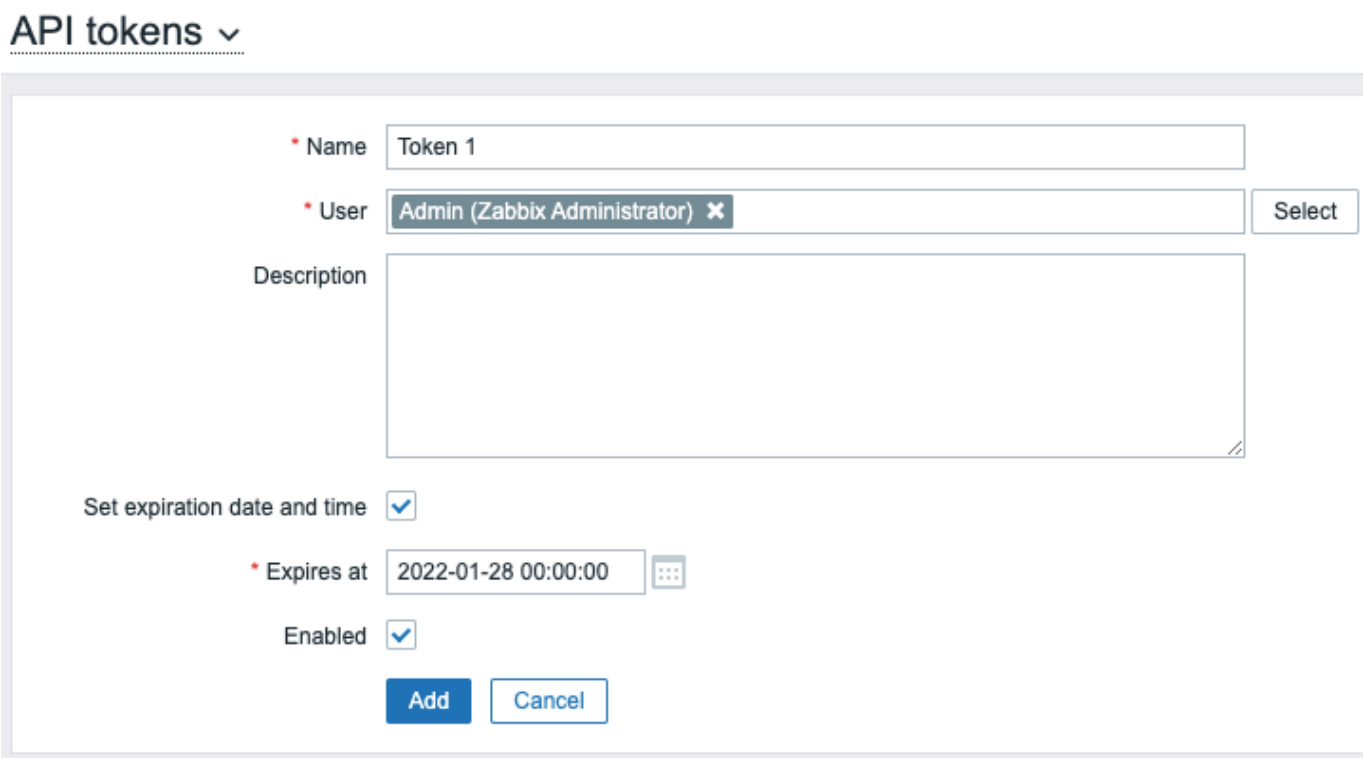
12 API 令牌

此部分允许创建和管理 API 令牌。



您可以按名称、分配令牌的用户、到期日期、创建令牌的用户或状态（启用/禁用）过滤 API 令牌。单击列表中的令牌状态可快速启用/禁用令牌。您还可以通过在列表中选择令牌然后单击列表下方的启用/禁用按钮来批量启用/禁用令牌。

要创建新令牌，请按右上角的 *Create API token* 按钮，然后在令牌配置屏幕中填写必填字段：



参数	说明
名称	令牌的可见名称。
User	应将令牌分配给的用户。要快速选择用户，请开始输入用户名、名字或姓氏，然后从自动完成列表中选择所需的用户。或者，您可以按选择按钮并从完整用户列表中选择一个用户。一个令牌只能分配给一个用户。
描述	可选的令牌描述。
设置到期日期和时间	如果令牌不应有到期日期，请取消标记此复选框。
到期日期	单击日历图标选择令牌到期日期或手动输入日期，格式为 YYYY-MM-DD hh:mm:ss
启用	如果您需要在禁用状态下创建令牌，请取消选中此复选框。

按添加以创建令牌。在下一个屏幕上，复制并保存在安全的地方 *Auth token* 值在关闭页面之前，然后按关闭。令牌将出现在列表中。

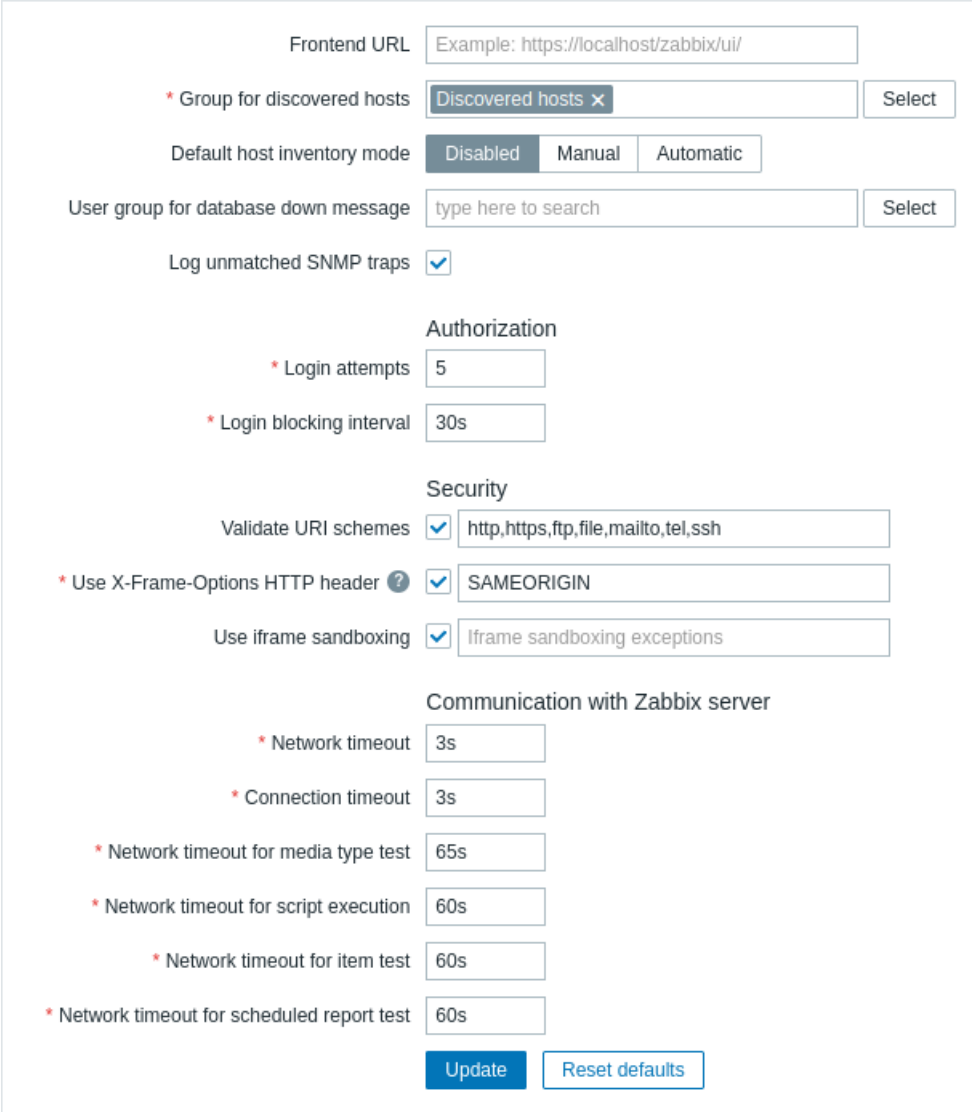
Warning:
Auth token 值以后无法再次查看。它仅在创建令牌后立即可用。如果您丢失了保存的令牌，您将不得不重新生成它，这样做会创建一个新的授权字符串。

单击令牌名称以编辑名称、描述、到期日期设置或令牌状态。请注意，无法更改令牌分配给哪个用户。按更新按钮保存更改。如果令牌丢失或暴露，您可以按 Regenerate 按钮生成新的令牌值。将出现一个确认对话框，询问您确认此操作，因为在此操作之后生成的令牌将变得无效。

仅当在其用户角色权限中允许管理 *API* 令牌时，无法访问 管理菜单部分的用户才能在 用户配置文件 → *API* 令牌部分中查看和修改分配给他们的令牌的详细信息。

13 其他参数

本节允许配置其他各种前端参数。



参数	说明
前端 *URL*	Zabbix Web 界面的 URL。此参数由 Zabbix Web 服务用于与前端通信，应指定以启用计划报告。
已发现主机组	由network discovery 和agent autoregistration 发现的主机将自动放置在此处选择的主机组中。
默认主机清单模式	主机清单的默认模式。每当服务器或前端创建新主机或主机原型时都会遵循它，除非在主机发现/自动注册期间被 *Set host inventory mode* 操作覆盖。

参数	说明
数据库关闭消息的用户组	发送警报消息的用户组或 “无”。 Zabbix 服务器取决于后端数据库的可用性。没有数据库它就无法工作。如果数据库关闭，Zabbix 可以通知选定的用户。通知将使用所有配置的用户媒体条目发送到此处设置的用户组。Zabbix 服务器不会停止；它将等到数据库再次返回以继续处理。 通知包含以下内容： `[MySQL|PostgreSQL|Oracle] 数据库 <DB Name> [on <DB Host>:<DB Port>] 不可用：< 错误消息取决于 DBMS（数据库）的类型 >` <DB Host> 如果定义为空值且未添加 <DB Port>，则不会将其添加到消息中如果它是默认值（“0”）。警报管理器（一个特殊的 Zabbix 服务器进程）每 10 秒尝试建立一个到数据库的新连接。如果数据库仍处于关闭状态，警报管理器会重复发送警报，但频率不会超过每 15 分钟一次。
记录不匹配的 *SNMP* 陷阱	如果没有找到对应的 SNMP 接口，则记录SNMP 陷阱。

授权

参数	说明
登录尝试	在登录被阻止之前的不成功登录尝试次数。
登录阻止间隔	超过登录尝试限制时将禁止登录的时间段。

安全

参数	说明
验证 *URI* 方案	取消选中该框以禁用针对有效 *URI* 方案中定义的白名单的 URI 方案验证。(默认启用)。
有效的 *URI* 方案	允许的 URI 方案的逗号分隔列表。适用于前端中的所有字段，其中使用 URI（例如，映射元素 URL）。 此字段仅在选择验证 *URI* 方案时才可编辑。
X-Frame-Options HTTP 标头	HTTP X-Frame-options 标头的值。支持的值： **SAMEORIGIN**（默认）- 页面只能显示在与页面本身同源的框架中。 **DENY** - 页面不能显示在框架中，无论站点是否尝试这样做。 **null** - 禁用 X-Frame-options 标头（不推荐）。 或逗号分隔的主机名列表（字符串）。如果不允许列出的主机名，则使用 SAMEORIGIN 选项。
使用 *iframe* 沙箱	此参数决定是否将检索到的 URL 内容放入沙箱。请注意，不建议关闭沙盒。
Iframe 沙盒例外	如果启用沙盒并且此字段为空，则适用所有沙盒属性限制。要禁用某些限制，请在此字段中指定它们。这只会禁用此处列出的限制，其他限制仍将适用。有关更多信息，请参阅 沙盒属性 描述。

与 Zabbix 服务器通信

参数	说明
网络超时	关闭空闲套接字前等待多少秒（如果之前已经建立了与 Zabbix 服务器的连接，但在此期间前端无法完成读取/发送数据操作，则连接将被丢弃）。 允许范围：1 - 300s（默认值：3s）。
连接超时	在停止尝试连接到 Zabbix 服务器之前等待多少秒。允许范围：1 - 30s (默认：3s)。
媒体类型测试的网络超时	测试媒体类型时等待响应的秒数。允许范围：1 - 300s（默认值：65s）。
脚本执行的网络超时	执行脚本时等待响应的秒数。允许范围：1 - 300s（默认值：60s）。
项目测试的网络超时	测试项目时等待返回数据的秒数。允许范围：1 - 300s（默认值：60s）。
计划报告测试的网络超时	测试计划报告时等待返回数据的秒数。允许范围：1 - 300s（默认值：60s）。

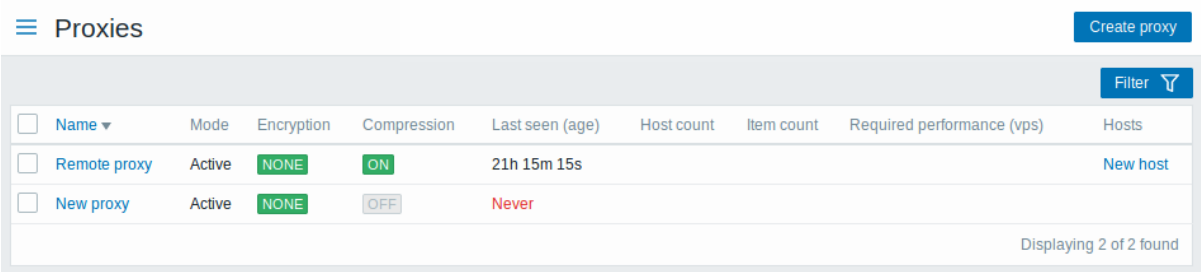
2 Proxies

概述

在 管理 → *Proxies* 部分代理分布式监控 可以在 Zabbix 前端配置。

代理

将显示现有代理及其详细信息的列表。



显示数据：

栏目	说明
名称	代理的名称。点击代理名称打开代理配置表单。
Mode	显示代理模式 - *Active* 或 *Passive*。
加密	显示来自代理的连接的加密状态： 无 - 未加密 **PSK** - 使用预共享密钥 **Cert** - 使用证书
上次看到（年龄）	显示代理上次被服务器看到的时间。
主机计数	显示分配给代理的启用主机的数量。
Item count	显示分配给代理的启用主机上启用的项目数。
所需性能 *(vps)*	显示所需的代理性能（每秒需要收集的值的数量）。
Hosts	列出代理监视的所有主机。单击主机名打开主机配置表单。

要配置新 proxy，请单击右上角顶部的创建 *proxy* 按钮。

批量编辑选项

列表下方的按钮提供了一些批量编辑选项：

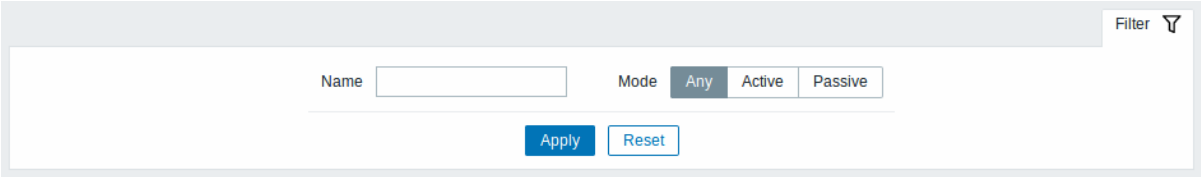
- 启用主机 - 将 proxy 监控的主机状态更改为已监控
- 禁用主机 - 将 proxy 监控的主机状态更改为未监控
- *Delete* - 删除 proxy

要使用这些选项，请在相应 proxy 之前标记复选框，然后单击所需按钮。

使用过滤器

您可以使用过滤器仅显示您感兴趣的 proxy。为了获得更好的搜索性能，使用未解析的宏搜索数据。

Filter 链接位于 proxy 列表上方。如果单击它，将出现一个过滤器，您可以在其中按名称和模式过滤 proxy。



3 认证

概述

Administration → *Authentication* 部分允许指定 Zabbix 的全局用户认证方法和内部密码要求。可用 internal、HTTP、LDAP 和 SAML 方式进行验证。

默认身份验证

默认情况下，Zabbix 对所有用户使用内部 Zabbix 身份验证。可以将默认方法更改为LDAP 系统范围或仅对特定用户组启用 LDAP 身份验证。

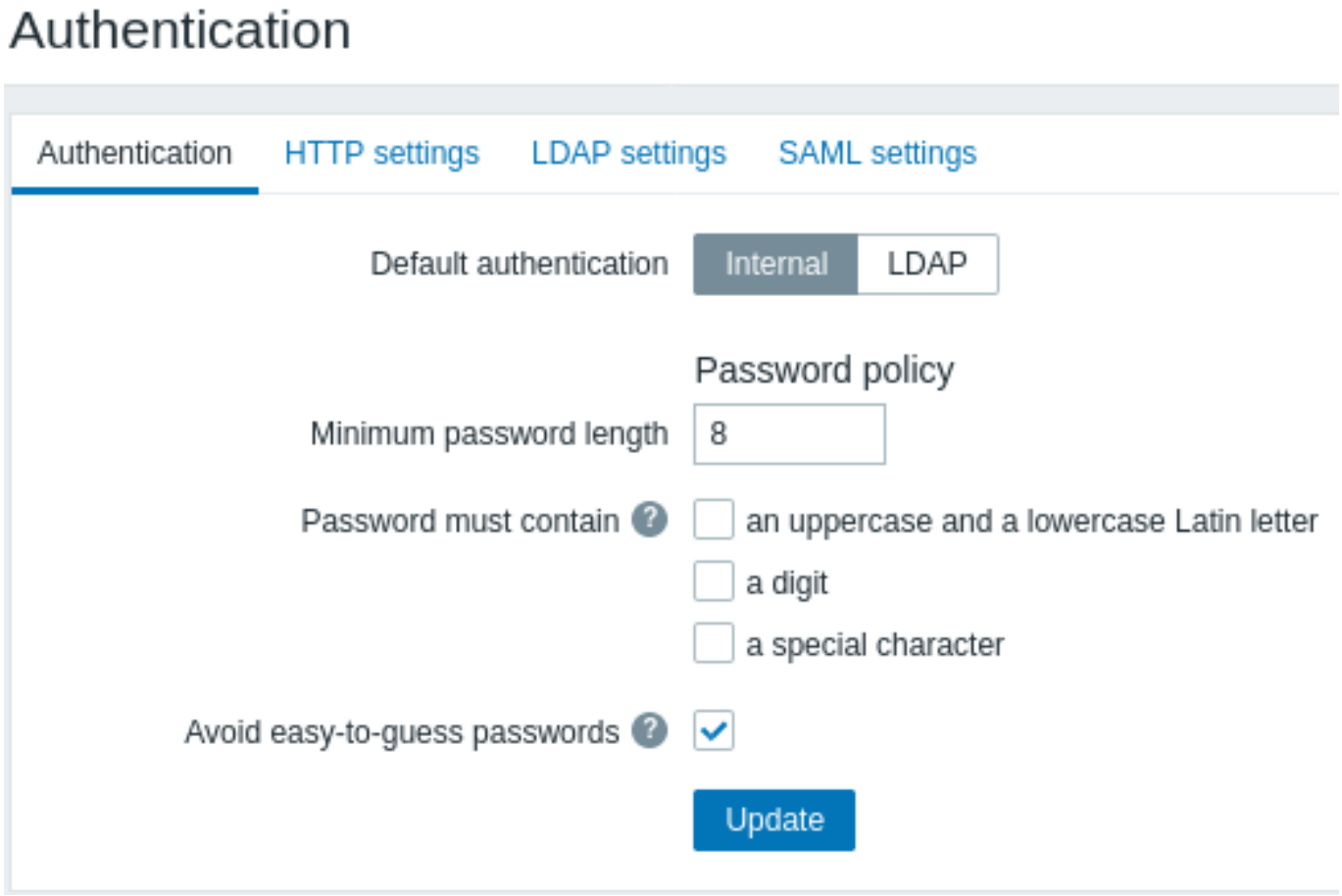
要将 LDAP 设置为所有用户的默认身份验证方法，请导航至 *LDAP* 选项卡并配置身份验证参数，然后返回 *Authentication* 选项卡并将 *Default authentication* 选择器切换到 LDAP

请注意，身份验证方法可以在用户组 级别。即使 LDAP 认证全局设置，部分用户组仍然可以由 Zabbix 认证。这些组必须有前端访问 设置到内部。反之亦然，如果全局使用内部身份验证，可以为特定用户指定和使用 LDAP 身份验证详细信息前端访问 是设置为 LDAP。如果一个用户包含在至少一个用户组中 LDAP 认证，此用户将无法使用内部身份验证方法。

HTTP 和SAML 2.0除了默认的身份验证方法之外，还可以使用身份验证方法身份验证方法。

内部认证

Authentication 选项卡允许定义自定义密码复杂性内部 Zabbix 用户的要求。



可以配置以下密码策略选项：

参数	说明
最小密码长度	默认情况下，最小密码长度设置为 8。支持范围：1-70。请注意，超过 72 个字符的密码将被截断。
密码必须包含	标记一个或多个复选框以要求在密码中使用指定字符： -一个大写和一个小写拉丁字母 -一个数字 -一个特殊字符
避免使用容易猜到的密码	将鼠标悬停在问号上可查看每个选项的字符列表提示。 如果标记，将根据以下要求检查密码： - 不得包含用户名、姓氏或用户名 - 不得为常见或上下文之一-特定密码。 常用密码和上下文特定密码列表是从 NCSC“前 100k 密码” 列表、SecLists“前 1M 密码” 列表和 Zabbix 上下文特定密码列表自动生成的. 不允许内部用户设置此列表中包含的密码，因为此类密码因其常见用途而被视为弱密码。

密码复杂性要求的变化不会影响现有的用户密码，但如果现有用户选择更改密码，新密码必须满足当前要求。一个提示要求列表将显示在 *Password* 字段旁边用户配置 和用户配置表单 可访问从管理 → 用户菜单。

HTTP 认证

基于 HTTP 或 Web 服务器的身份验证（例如：Basic Authentication、NTLM/Kerberos）可用于检查用户名和密码。请注意，用户也必须存在于 Zabbix 中，但是它的不会使用 Zabbix 密码。

Attention:
当心！确保网络服务器身份验证在切换之前已配置且正常工作。

配置参数：

参数	说明
启用 *HTTP* 身份验证	选中复选框以启用 HTTP 身份验证。
默认登录表单	指定是否将未经身份验证的用户定向到： **Zabbix** 登录表单 - 标准 Zabbix 登录页面。 **HTTP** 登录表单 - HTTP 登录页面。 建议仅对 `index_http.php` 页面启用基于 Web 服务器的身份验证。如果 *Default login form* 设置为 “HTTP login page”，如果 Web 服务器身份验证模块将在 `$_SERVER` 变量中设置有效的用户登录名，则用户将自动登录。 支持的 `$_SERVER` 键是 `PHP_AUTH_USER`，`REMOTE_USER`，`AUTH_USER`。
删除域名	应从用户名中删除的以逗号分隔的域名列表。 例如 `comp,any` - 如果用户名是’Admin@any’，’comp\Admin’，用户将作为’Admin’ 登录；如果用户名是’notacompany\Admin’，登录将被拒绝。
区分大小写的登录	取消选中该复选框以禁用用户名的区分大小写登录（默认启用）。 例如禁用区分大小写的登录并使用例如 “ADMIN” 用户登录，即使 Zabbix 用户是 “Admin”。 注意禁用区分大小写登录的情况下，如果在 Zabbix 数据库存在多个用户，登录将被拒绝（例如 Admin、admin）。

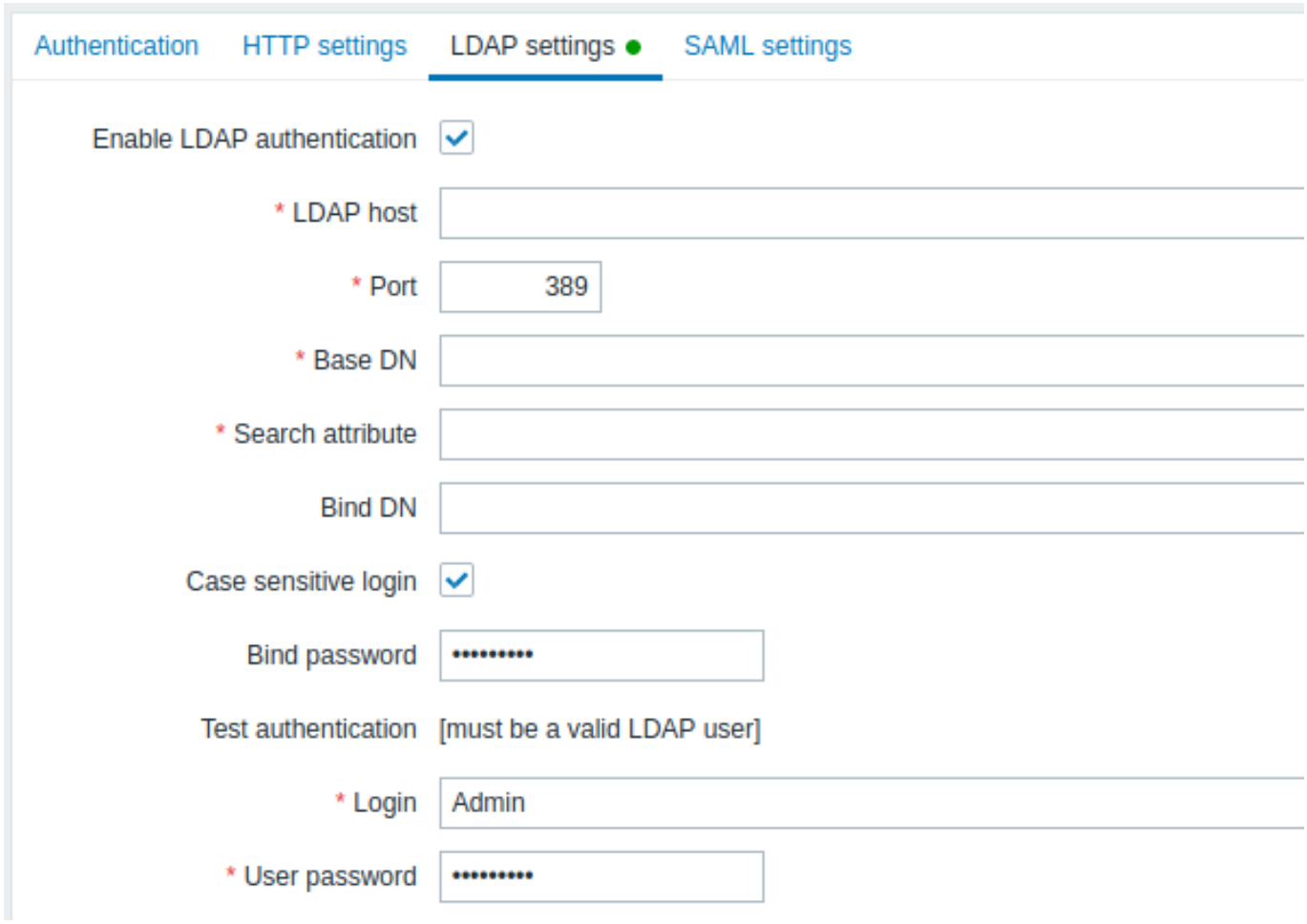
Note:
在 Web 服务器身份验证的情况下，所有用户（即使前端访问 设置到 LDAP/内部）将由 Web 服务器进行身份验证，而不是由 Zabbix！

Note:
对于无法使用 HTTP 登录的内部用户导致 401 的凭据 (默认设置为 HTTP 登录表单) 错误，您可能需要添加一个 `ErrorDocument 401 /index.php?form=default` 行到基本身份验证指令，将重定向到常规的 Zabbix 登录表单。

LDAP 身份验证

外部 LDAP 身份验证可用于检查用户名和密码。请注意，用户也必须存在于 Zabbix 中，但不会使用其 Zabbix 密码。

Zabbix LDAP 身份验证至少适用于 Microsoft Active Directory 和 OpenLDAP。



配置参数：

参数	说明
启用 *LDAP* 身份验证	选中复选框以启用 LDAP 身份验证。
LDAP 主机	LDAP 服务器的名称。例如：ldap://ldap.zabbix.com 对于安全的 LDAP 服务器，请使用 *ldaps* 协议。 ldaps://ldap.zabbix.com 对于 OpenLDAP 2.xx 及更高版本，完整的可以使用 ldap://hostname:port 或 ldaps://hostname:port 形式的 LDAP URI。
端口	LDAP 服务器的端口。默认值为 389。 对于安全 LDAP 连接，端口号通常为 636。 在使用完整 LDAP URI 时不使用。
Base DN	搜索帐户的基本路径： ou=Users,ou=system（对于 OpenLDAP）， DC=company,DC=com（对于 Microsoft Active Directory）
搜索属性	用于搜索的 LDAP 帐户属性： uid（用于 OpenLDAP）、 sAMAccountName（用于 Microsoft Active Directory）
绑定 *DN*	LDAP 帐户，用于在 LDAP 服务器上进行绑定和搜索，示例： uid=ldap_search,ou=system (for OpenLDAP), CN=ldap_search,OU=user_group ,DC=company,DC=com（用于 Microsoft Active Directory） 还支持匿名绑定。
区分大小写的登录	取消选中该复选框以禁用用户名的区分大小写登录（默认启用）。 例如禁用区分大小写的登录并使用例如 “ADMIN” 用户登录，即使 Zabbix 用户是 “Admin”。 注意禁用区分大小写登录的情况下，如果存在多个用户，登录将被拒绝具有相似用户名的 Zabbix 数据库（例如 Admin、admin）。
绑定密码	LDAP 服务器上绑定和搜索帐号的 LDAP 密码。
测试认证	测试部分的标题

参数	说明
Login	测试用户的名称（当前登录在 Zabbix 前端）。此用户名必须存在于 LDAP 服务器中。 如果 Zabbix 无法对测试用户进行身份验证，它将不会激活 LDAP 身份验证。
用户密码	测试用户的 LDAP 密码。

Warning:
如果证书出现问题，要使安全的 LDAP 连接 (ldaps) 正常工作，您可能需要在 /etc/openldap/ldap.conf 配置文件中添加 `TLS_REQCERT allow` 行。它可能会降低连接到 LDAP 目录的安全性。

Note:
建议创建一个单独的 LDAP 帐户 (*Bind DN*) 以在 LDAP 中以最低权限在 LDAP 服务器上执行绑定和搜索，而不是使用真实用户帐户（用于登录 Zabbix 前端）。这种方法提供了更高的安全性，并且当用户在 LDAP 服务器中更改自己的密码时，不需要更改 *Bind password*。在上表中，它是 *ldap_search* 帐户名。

SAML 身份验证

SAML 2.0 身份验证可用于登录 Zabbix。请注意，用户必须存在于 Zabbix 中，但不会使用其 Zabbix 密码。如果身份验证成功，则 Zabbix 会将本地用户名与 SAML 返回的用户名属性进行匹配。

如果启用 SAML 身份验证，用户将能够在本地登录或通过 SAML 单点登录之间进行选择。

设置身份提供者

为了与 Zabbix 合作，SAML 身份提供商 (onelogin.com、auth0.com、okta.com 等) 需要按如下方式配置：

- *Assertion Consumer URL* 应该设置为·`<path_to_zabbix_ui>/index_sso.php?acs`
- *Single Logout URL* 应设置为·`<path_to_zabbix_ui>/index_sso.php?sls`

`<path_to_zabbix_ui>` 示例：%% https://example.com/zabbix/ui, http://another.example.com/zabbix, http://<any_public_ip_address>/zabb
%%

设置 Zabbix

Attention:
如果要在前端使用 SAML 身份验证，则需要安装 php-openssl。

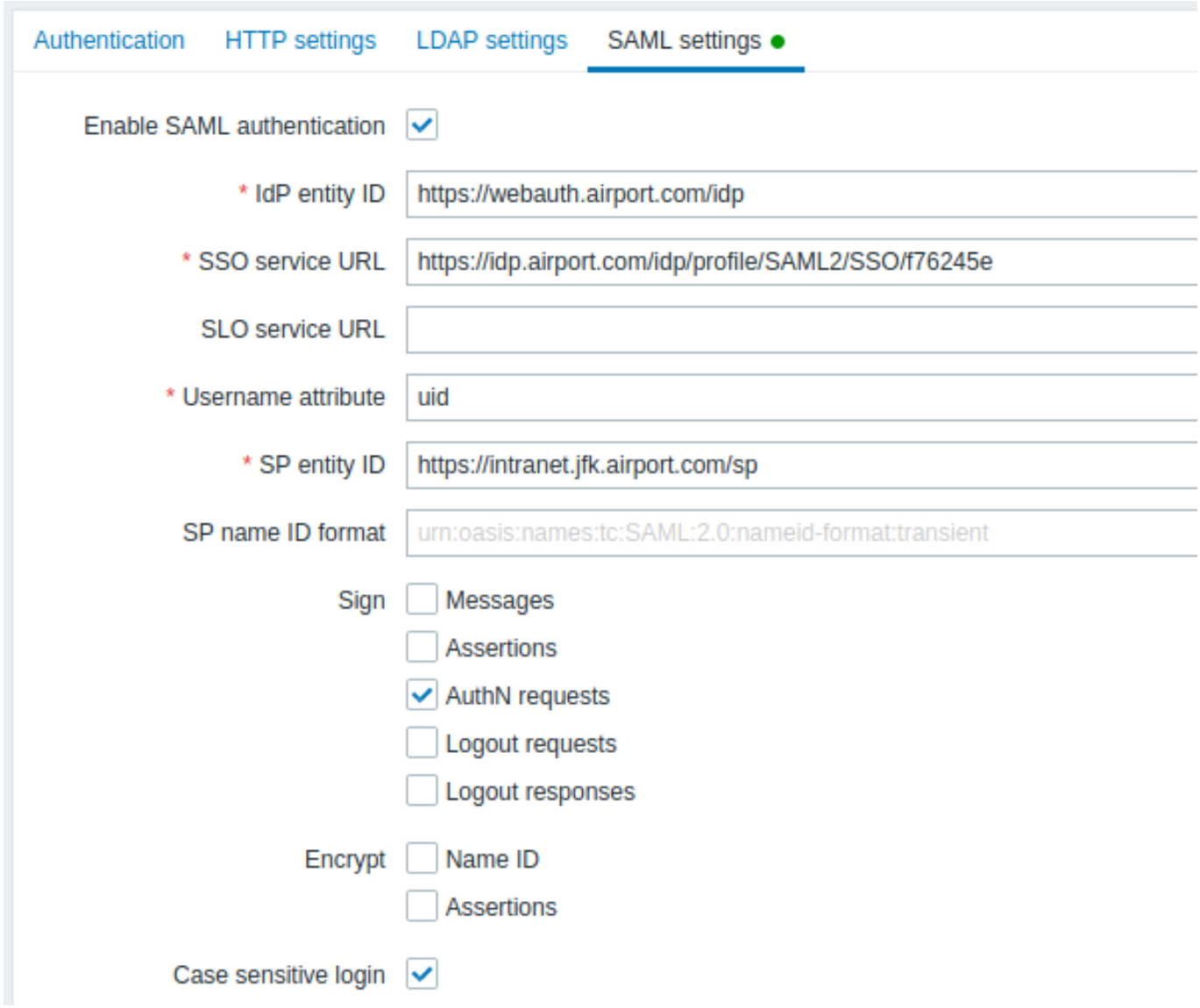
要使用 SAML 身份验证，Zabbix 应按以下方式配置：

1. 私钥和证书应存储在 *ui/conf/certs/* 中，除非zabbix.conf.php 中提供了自定义路径。

默认情况下，Zabbix 将在以下位置查找：

- ·ui/conf/certs/sp.key - SP 私钥文件
- ·ui/conf/certs/sp.crt - SP 证书文件
- ·ui/conf/certs/idp.crt - IDP 证书文件

2. 所有最重要的设置都可以在 Zabbix 前端进行配置。但是，可以在配置文件 中指定其他设置。



配置参数，在 Zabbix 前端可用：

参数	说明
启用 *SAML* 身份验证	选中复选框以启用 SAML 身份验证。
IDP 实体 *ID*	SAML 身份提供者的唯一标识符。
SSO 服务 *URL*	用户登录时将被重定向到的 URL。
SLO 服务 *URL*	用户注销时将被重定向到的 URL。如果留空，则不会使用 SLO 服务。
// 用户名属性//	登录 Zabbix 时用作用户名的 SAML 属性。 支持的值列表由身份提供者确定。 示例： uid 用户主体名 samaccountname 用户名 用户用户名 urn:oid:0.9.2342.19200300.100.1.1 urn:oid:1.3.6.1.4.1.5923.1.1.1.13 br>urn:oid:0.9.2342.19200300.100.1.44
SP 实体 *ID*	SAML 服务提供者的唯一标识符。

参数	说明
SP 名称 *ID* 格式	定义应使用的名称标识符格式。 示例： urn:oasis:names:tc:SAML:2.0:nameid-format:persistent urn:oasis:names:tc:SAML:2.0:nameid-format:transient urn:oasis:names:tc:SAML:2.0:nameid-format:kerberos <urn:oasis:names :tc:SAML:2.0:nameid-format:entity>
Sign	标记复选框以选择应为其启用 SAML 签名的实体： 消息 断言 *AuthN* 请求 注销请求 注销回复
加密	标记复选框以选择应启用 SAML 加密的实体： 断言 名称 *ID*
区分大小写的登录	标记复选框以启用用户名区分大小写的登录（默认禁用）。 例如禁用区分大小写的登录并使用例如 “ADMIN” 用户登录，即使 Zabbix 用户是 “Admin”。 注意禁用区分大小写登录的情况下，如果存在多个用户，登录将被拒绝具有相似用户名的 Zabbix 数据库（例如 Admin、admin）。

高级设置

可以在 Zabbix 前端配置文件 (*zabbix.conf.php*) 中配置额外的 SAML 参数：

- $SSO[’SP_KEY’] = ’<SP 私钥文件的路径 >’;
- $SSO[’SP_CERT’] = ’<SP 证书文件的路径 >’;
- $SSO[’IDP_CERT’] = ’<IDP 证书文件的路径 >’;
- $SSO[’ 设置’]

Zabbix 使用 OneLogin 的 SAML PHP Toolkit 库（版本 3.4.1）。$SSO[’SETTINGS’] 部分的结构应该与库使用的结构相似。配置选项的说明见官方库文档。

只有以下选项可以设置为 $SSO[’SETTINGS’] 的一部分：

- *strict*
- *baseurl*
- *compress*
- *contactPerson*
- *organization*
- *sp*（仅在此列表中指定的选项）
- *attributeConsumingService*
- *x509certNew*
- *idp*（仅在此列表中指定的选项）
  - *singleLogoutService*（只有一个选项）
  - *responseUrl*
  - *certFingerprint*
  - *certFingerprintAlgorithm*
  - *x509certMulti*
- *security*（仅在此列表中指定的选项）
- *signMetadata* -- *wantNameId*
- *requestedAuthnContext*
- *requestedAuthnContextComparison*
- *wantXMLValidation*
- *relaxDestinationValidation*
- *destinationStrictlyMatches*
- *rejectUnsolicitedResponsesWithInResponseTo*
- 签名算法
- *digestAlgorithm*
- 小写 *Urlencoding*

所有其他选项将从数据库中获取并且不能被覆盖。*debug* 选项将被忽略。

此外，如果 Zabbix UI 位于代理或负载均衡器之后，则可以使用自定义 *use_proxy_headers* 选项：

- *false* (默认) - 忽略该选项；
- *true* - 使用 X-Forwarded-* HTTP 标头来构建基本 URL。

如果使用负载均衡器连接到 Zabbix 实例，其中负载均衡器使用 TLS/SSL 而 Zabbix 不使用，您必须指明'baseurl'、'strict'和'use_proxy_headers' 参数，如下所示：

$SSO_SETTINGS=['strict' => false, 'baseurl' => "https://zabbix.example.com/zabbix/", 'use_proxy_headers' => true] 配置示例：

$SSO['SETTINGS'] = ['security' => ['signatureAlgorithm' => 'http://www.w3.org/2001/04/xmldsig-more#rsa-sha384' 'digestAlgorithm' => 'http://www.w3.org/2001/04/xmldsig-more#sha384', // ...], // ...];

Frontend configuration with Kerberos/ADFS

The Zabbix frontend configuration file (*zabbix.conf.php*) can be used to configure SSO with Kerberos authentication and ADFS:

```
$SSO['SETTINGS'] = [
    'security' => [
        'requestedAuthnContext' => [
            'urn:oasis:names:tc:SAML:2.0:ac:classes:Kerberos',
        ],
        'requestedAuthnContextComparison' => 'exact'
    ]
];
```

In this case, in the SAML configuration *SP name ID* field set:

`urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified`

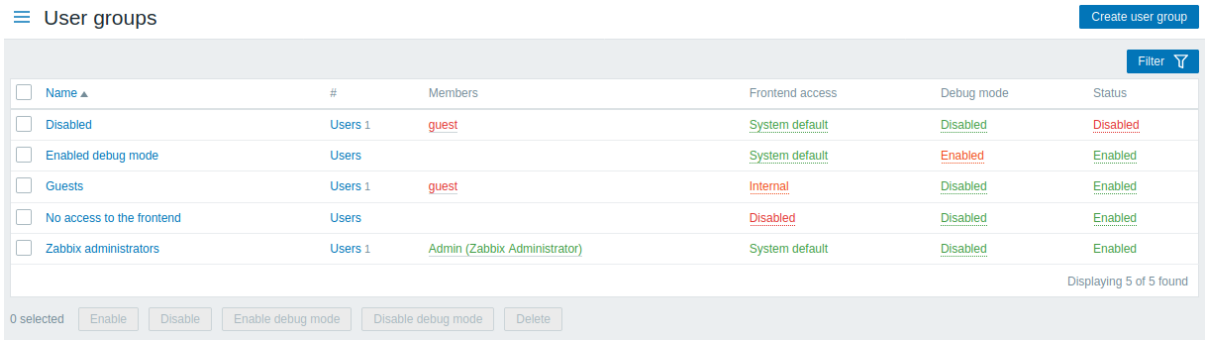
4 用户组

概述

在 管理 → 用户组菜单下维护系统的用户组。

用户组

将显示现有用户组及其详细信息的列表。



显示数据：

栏目	说明
名称	用户组的名称。点击用户组名打开用户组配置表单。
#	组中的用户数。单击 用户将显示在用户列表中过滤掉的各个用户。
成员	用户组中各个用户的用户名（括号中包含姓名和姓氏）。单击用户名将打开用户配置表单。来自禁用组的用户显示为红色。
前端访问	显示前端访问级别： 系统默认 - Zabbix, LDAP or HTTP 认证; 取决于选择的身份验证 [方法]（身份验证） 内部 - 无论系统设置如何，用户都由 Zabbix 进行身份验证 停用 - 此用户的前端访问被禁用。 通过单击当前级别，您可以更改它。
调试模式	调试模式 显示状态 - *Enabled* 或 *Disabled*。通过单击状态，您可以更改它。
状态	显示用户组状态 - 启用或 停用。通过单击状态可以更改。

要配置新用户组，请单击右上角的创建用户组按钮。

批量编辑选项

列表下方的按钮提供了一些批量编辑选项：

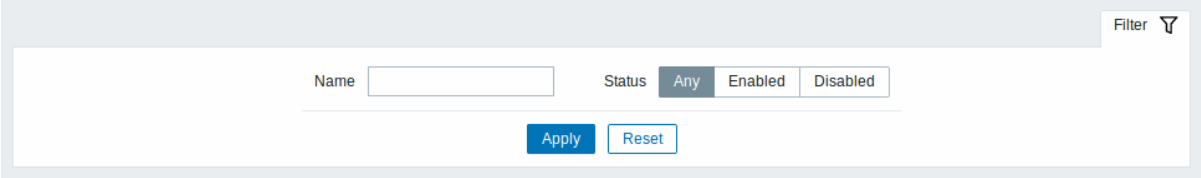
- ·启用 - 将用户组状态更改为启用
- ·停用 - 将用户组状态更改为 停用
- ·启用调试模式 - 为用户组启用调试模式
- ·禁用调试模式 - 禁用用户组的调试模式
- ·停用 - 删除用户组

要使用这些选项，请在相应用户组之前标记复选框，然后单击所需按钮。

使用过滤器

您可以使用过滤器仅显示您感兴趣的用户组。为了获得更好的搜索性能，使用未解析的宏搜索数据。

过滤器链接位于用户组列表上方。如果单击它，将出现一个过滤器，您可以在其中按名称和状态过滤用户组。



5 用户角色

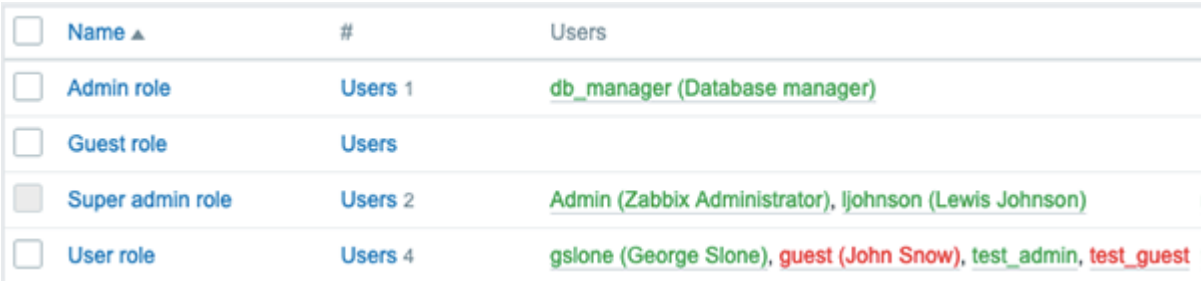
概述

在 管理 → 用户角色部分中，可以分配给系统用户的角色和每个角色的特定权限得到维护。

默认用户角色

默认情况下，Zabbix 配置了四个用户角色，它们具有一组预定义的权限：

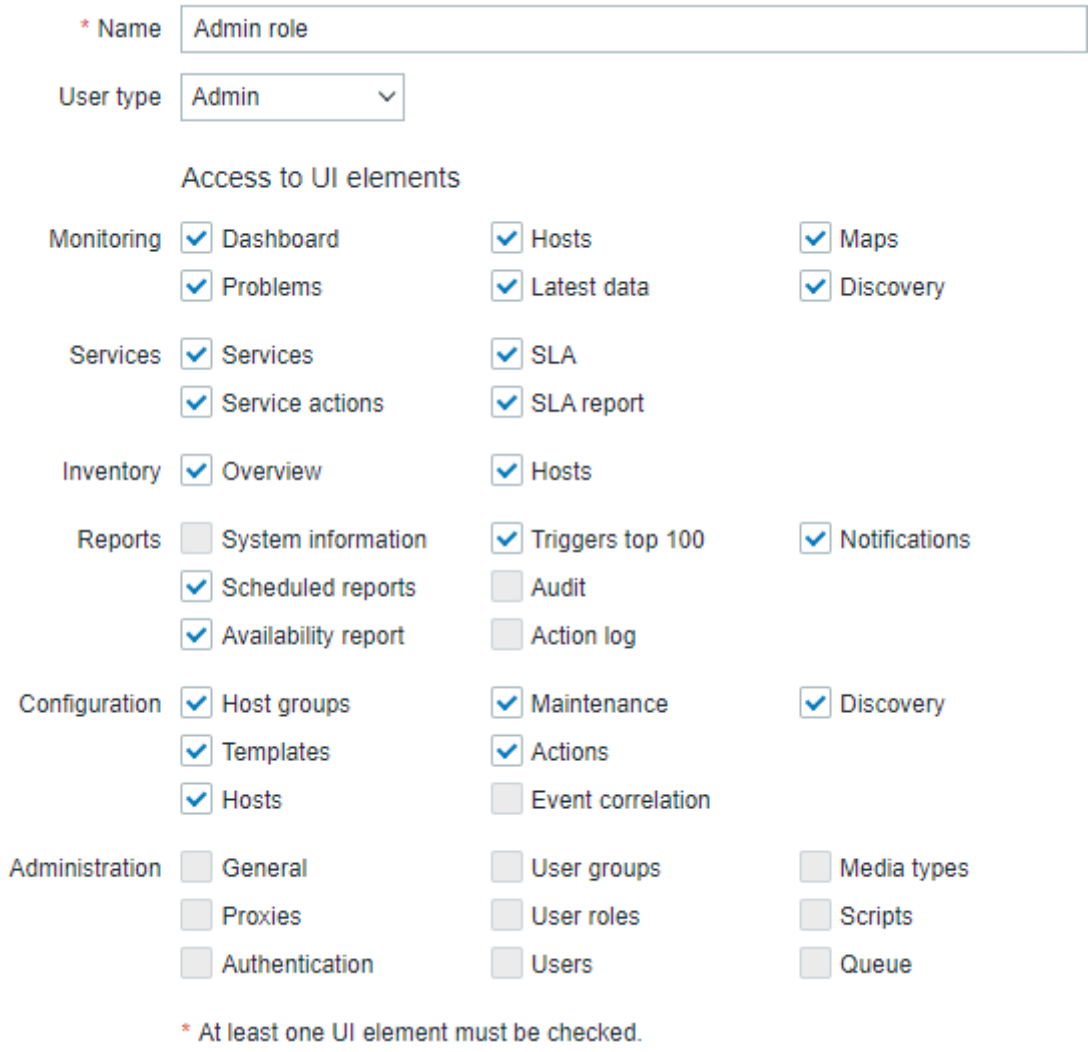
- ·管理员角色
- ·嘉宾角色
- ·超级管理员角色
- ·用户角色



Note:
默认 超级管理员角色不能修改或删除，因为 Zabbix 中必须至少存在一个具有无限权限的超级管理员。

具有超级管理员类型和适当权限的 Zabbix 用户可以修改或删除现有角色或创建新的自定义角色。

要创建新角色，请单击右上角的创建用户角色按钮。要更新现有角色，请按角色名称以打开配置表单。



Zabbix 中预先存在的用户角色的可用权限选项以及默认权限集如下所述。

参数	说明	默认用户角色			
		超级管理员角色	管理员角色	用户角色	访客角色
名称	角色可见名称。	超级管理员角色	管理员角色	用户角色	访客角色
用户类型	选定的用户类型决定了可用权限的列表。 选择用户类型后，默认情况下会授予此用户类型的所有可用权限。 取消选中该复选框可撤消该用户的某些权限用户角色。 此用户类型不可用的权限复选框呈灰色显示。	超级管理员	管理员	用户	用户
访问 UI 元素					
监控					
仪表板	启用/禁用对特定监控菜单部分和基础页面的访问。	是	是	是	是
问题					
主机					
最新数据					
地图					

参数	说明	默认用户角色			
发现				否	否
服务					
服务	启用/禁用对特定服务菜单部分和基础页面的访问。	是	是	是	是
服务行动					
SLA					
SLA 报告					
资产管理					
概览	启用/禁用对特定库存菜单部分和基础页面的访问。	是	是	是	是
主机					
报告					
系统信息	启用/禁用对特定报告菜单部分和基础页面的访问。	是	否	否	否
可用性报告			是	是	是
触发前 100 名					
审核			否	否	否
动作日志					
通知			是		
预定报告					
配置					
主机组	启用/禁用对特定配置菜单部分和基础页面的访问。	是	是	否	否
模板					
主机					
维护					
动作					
事件关联			否		
发现			是		
管理					
常规	启用/禁用对特定管理菜单部分和基础页面的访问。	是	否	否	否
proxies					
认证					
用户组					
用户角色					
用户					
媒介类型					
脚本					
队列					
对新 *UI* 元素的默认访问	启用/禁用对自定义 UI 元素的访问。模块，如果存在，将在下面列出。	是	是	是	是
访问服务					
对服务的读写访问权限	选择对服务的读写访问权限： 无 - 完全没有访问权限 全部 - 对所有服务的访问权限为读写权限 * * 服务列表 ** - 为读写访问选择服务 读写访问优先于只读访问，并由子服务动态继承。	是	是	否	否
使用标签对服务进行读写访问	指定标签名称和可选的值，以额外授予对匹配标签的服务的读写访问权限。 如果在 *Read-* 中选择了 “服务列表”，则此选项可用对 *services* 参数的写访问。 读写访问优先于只读访问，并由子服务动态继承。				

参数	说明	默认用户角色			
对服务的只读访问权限	选择对服务的只读访问权限： 无 - 根本没有访问权限 全部 - 对所有服务的访问权限都是只读的 * * 服务列表 ** - 选择服务进行只读访问 只读访问不优先于读写访问，由子服务动态继承。				
对带有标签的服务的只读访问权限	指定标签名称和（可选）值以额外授予对匹配标签的服务的只读访问权限。 如果在 *Read-* 中选择了 “服务列表”，则此选项可用仅访问 *services* 参数。 只读访问不优先于读写访问，由子服务动态继承。				
访问模块					
< 模块名称 >	允许/拒绝对特定模块的访问。本节仅显示启用的模块。无法授予或限制对当前禁用的模块的访问权限。	是	是	是	是
对新模块的默认访问	启用/禁用对将来可能添加的模块的访问。				
访问 API					
启用	启用/禁用对 API 的访问。	是	是	是	否
API 方法	选择允许列表仅允许指定的 API 方法或选择拒绝列表仅限制指定的 API 方法。 在搜索字段中，开始输入方法名称，然后选择方法从自动完成列表中。 您还可以按 “选择” 按钮并从可用于此用户类型的完整列表中选择方法。请注意，如果未选中 “访问操作” 块中的某些操作，用户将无法使用与该操作相关的 API 方法。 支持通配符。示例：`dashboard.*`（’dashboard.’ API 服务的所有方法）`*`（任何方法）、`*.export`（来自所有 API 服务的具有’.export’ 名称的方法）。 如果没有指定方法，允许/拒绝列表规则将被忽略。				
访问操作					
创建和编辑仪表板	清除此复选框还将撤销对相应元素使用 `.create`、`.update` 和 `.delete` API 方法的权限。	是	是	是	否
创建和编辑地图					
创建和编辑维护				否	
添加问题评论	清除此复选框还将撤销通过 `event.acknowledge` API 方法执行相应操作的权限。			是	
更改严重性					
确认问题					
关闭问题					
执行脚本	清除此复选框也将撤销使用 `script.execute` API 方法的权限。				
管理 API 令牌	清除此复选框还将撤销使用所有 `token.` API 方法的权利。				
管理预定报告	清除此复选框还将撤销使用所有 “报告”API 方法的权利。			否	
管理 SLA	启用/禁用管理SLA 的权限。				
默认访问新操作	启用/禁用访问新操作。			是	

备注：

- 每个用户只能分配一个角色。
- 如果某个元素受到限制，即使在浏览器中输入指向该元素的直接 URL，用户也无法访问它。

- ·*User* 或 *Admin* 类型的用户不能更改他们自己的角色设置。
- ·超级管理员类型的用户可以修改他们自己角色的设置（不适用于默认的超级管理员角色），但不能修改用户类型。
- ·各级用户不能更改自己的用户类型。

参阅：

- ·配置用户

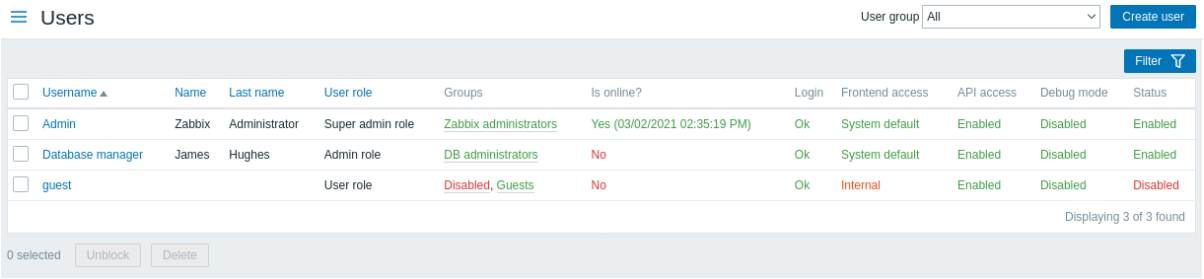
6 用户

概述

在 管理 → 用户菜单下维护系统的用户。

用户

将显示现有用户的列表及其详细信息。



从 用户栏右侧的下拉列表中，您可以选择是显示所有用户还是显示属于某个特定组的用户。

显示数据：

栏目	说明
用户名	登录 Zabbix 的用户名。点击用户名打开用户配置表单。
名	用户的名字。
姓	用户姓。
用户角色	显示用户角色。
用户组	列出了用户所属的组。单击用户组名称会打开用户组配置表单。禁用的组以红色显示。
在线吗？	显示用户的在线状态 - 是或 否。最后用户活动的时间显示在括号中。
登录	显示用户的登录状态 - 正常或 锁定。如果超过 *Administration→General* 部分中设置的不成功登录尝试次数（默认为 5 次），用户可能会被暂时阻止。通过点击 锁定您可以解除对用户的阻止。
前端访问	显示前端访问级别 - 系统默认、内部或禁用，取决于为整个用户组设置的一个。
API 访问	显示 API 访问状态 - 启用或 停用，取决于用户角色的设置。
调试模式	显示调试模式状态 - 启用或 停用，取决于为整个用户组设置的一组。
状态	显示用户状态 - 启用或 停用，取决于为整个用户组设置的一组。

要配置新用户，请单击右上角的 创建用户按钮。

批量编辑选项

列表下方的按钮提供了一些批量编辑选项：

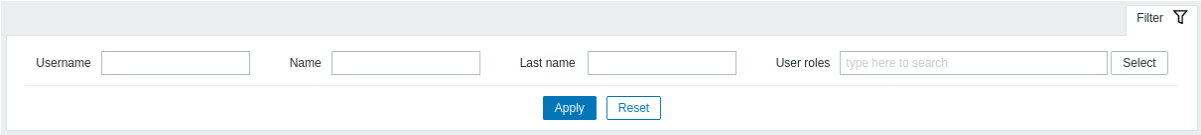
- ·解锁 - 重新启用对被阻止用户的系统访问
- ·删除 - 删除用户

要使用这些选项，请在相应用户之前标记复选框，然后单击所需按钮。

使用过滤器

您可以使用过滤器仅显示您感兴趣的用户。为了获得更好的搜索性能，使用未解析的宏搜索数据。

Filter 链接位于用户列表上方。如果单击它，则会出现一个过滤器，您可以在其中按用户名、姓名、姓氏和用户角色过滤用户。



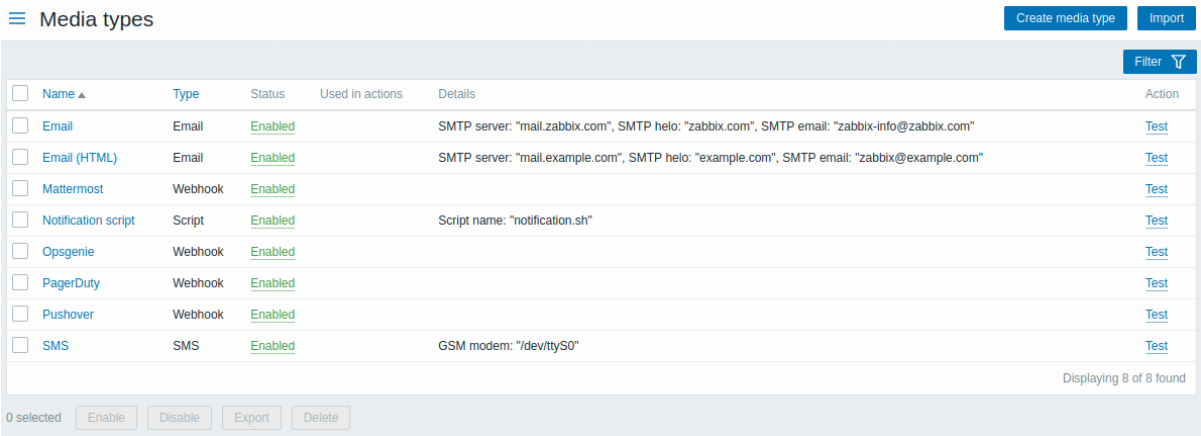
7 媒介类型

概述

在 管理 → 媒介类型部分，用户可以配置和维护媒介类型信息。

媒介类型信息包含使用媒介作为通知传递渠道的一般说明。具体细节，例如发送通知的个人电子邮件地址，由个人用户保存。

显示现有媒介类型及其详细信息的列表。



显示数据：

栏目	说明
名称	媒介类型的名称。点击名称打开媒介类型配置表单。
Type	显示媒介的类型（电子邮件、SMS 等）。
Status	显示媒介类型状态 - 启用或 停用。 单击状态可以更改它。
在操作中使用	直接使用媒介类型的所有操作（在 *Send only to* 下拉列表中选择）都会显示。 单击动作名称打开动作配置表单。
详细信息	显示媒介类型的详细信息。
Actions	以下操作可用： **Test** - 点击打开一个测试表格，您可以在其中输入媒介类型参数（例如，带有测试主题和正文的收件人地址）并将测试消息发送到验证配置的媒介类型是否有效。另请参阅：媒介类型测试。

要配置新的媒介类型，请单击右上角的创建媒介类型按钮。

要从 XML 导入媒体类型，请单击右上角的 导入按钮。

批量编辑选项

列表下方的按钮提供了一些批量编辑选项：

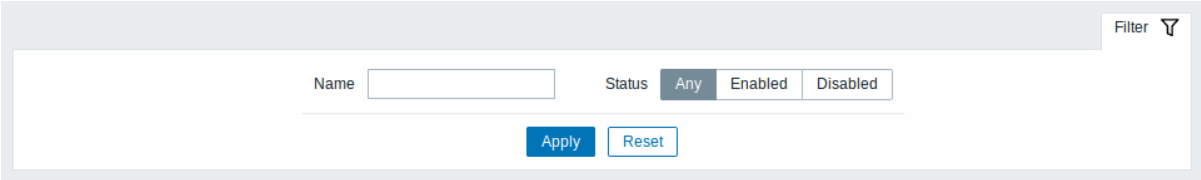
- ·启用 - 将媒介类型状态更改为 启用
- ·停用 - 将媒介类型状态更改为 *Disabled*
- ·导出 - 将媒介类型导出为 YAML、XML 或 JSON 文件
- ·删除 - 删除媒介类型

要使用这些选项，请在相应媒介类型之前标记复选框，然后单击所需按钮。

使用过滤器

可以使用过滤器仅显示您感兴趣的媒介类型。为了获得更好的搜索性能，搜索数据时未解析宏。

过滤器链接位于媒介类型列表上方。如果单击它，则会出现一个过滤器，可以按名称和状态过滤媒介类型。



8 脚本

概述

在 管理 → 脚本部分，可以配置和维护用户定义的全局脚本。

全局脚本，取决于配置的范围和用户权限，可用于执行：

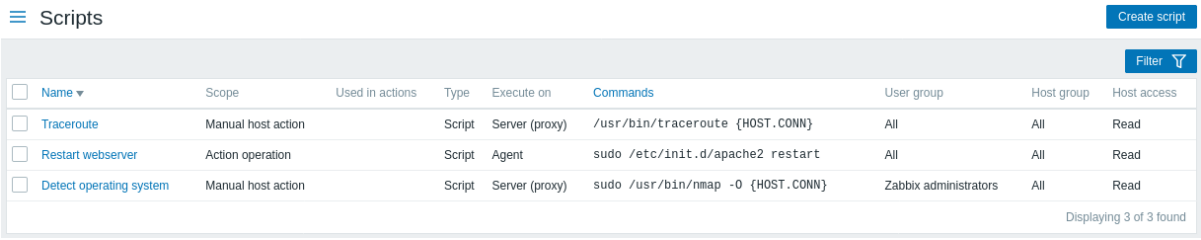
- 从主机菜单 中的各个前端位置（仪表板、问题、最新数据、拓扑图等）
- 从事件菜单
- 可以作为动作操作运行

脚本仅在 Zabbix agent、Zabbix server（proxy）或 Zabbix server 上执行。另请参见命令执行。

默认情况下，Zabbix agent 和 Zabbix proxy 远程脚本都被禁用。它们可以通过以下方式启用：

- 在 agent 配置中添加 `AllowKey=system.run[*]` 参数；
- 在 proxy 配置中将 EnableRemoteCommands 参数设置为 “1”

将显示现有脚本及其详细信息的列表。



显示数据：

栏目	说明
名称	脚本的名称。单击脚本名称打开脚本配置表单。
范围	脚本的范围 - 动作操作、手动主机动作或手动事件动作。此设置确定脚本可用的位置。
用于动作	显示使用脚本的动作。
类型	显示脚本类型 - *Webhook*、*Script*、*SSH*、*Telnet* 或 *IPMI* 命令。
执行在	显示脚本是在 Zabbix agent、Zabbix server（proxy）还是仅在 Zabbix server 上执行。
命令	显示脚本中要执行的所有命令。
用户组	显示脚本可用的用户组（或 全部所有用户组）。
主机组	显示脚本可用的主机组（或 全部用于所有主机组）。
主机访问	显示主机组的权限级别 - 读或 写。只有具有所需权限级别的用户才能执行脚本。

要配置新脚本，请单击右上角的 创建脚本按钮。

批量编辑选项

列表下方的按钮提供了一个批量编辑选项：

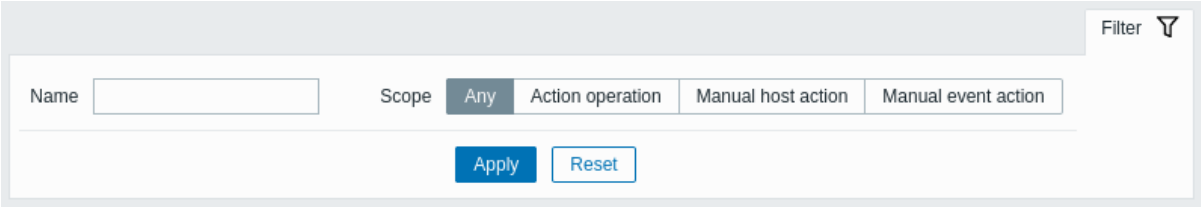
- 删除 - 删除脚本

要使用此选项，请标记相应脚本前的复选框并单击 删除。

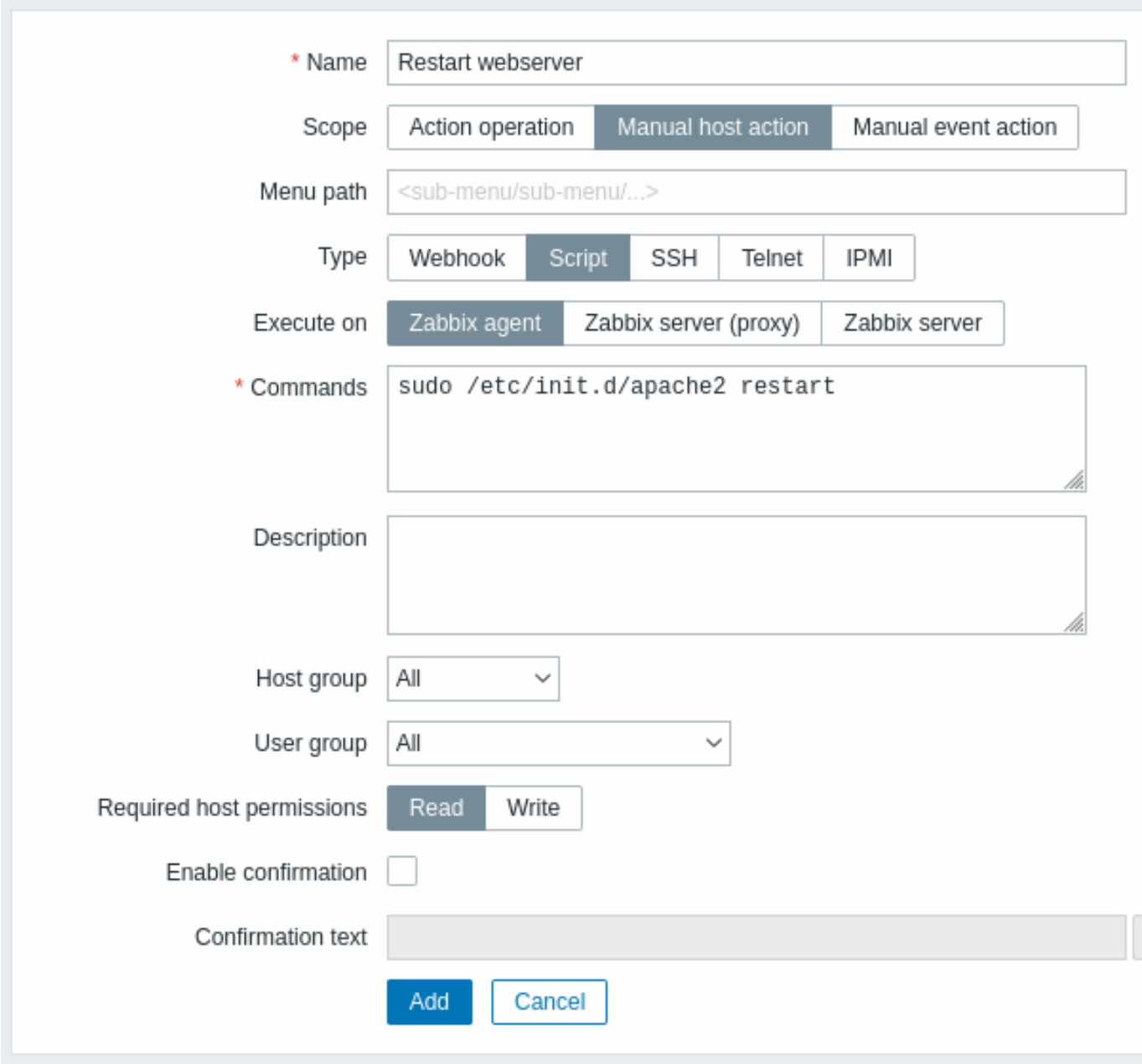
使用过滤器

您可以使用过滤器仅显示您感兴趣的脚本。为了获得更好的搜索性能，搜索数据时未解析宏。

过滤器链接位于脚本列表上方。如果您单击它，则会出现一个过滤器，您可以在其中按名称和范围过滤脚本。



配置全局脚本



脚本属性：

参数	说明
名称	脚本的唯一名称。 例如清除 `/tmp` 文件系统

参数		说明
范围		脚本的范围 - 动作操作、手动主机动作或手动事件动作。此设置确定脚本可以在哪里使用 - 在操作操作的远程命令中，来自主机菜单 或来自事件菜单。 将范围设置为 “操作操作” 使脚本可供所有有权访问 配置 → 动作的用户使用。 如果脚本实际用于动作，它的范围不能从 “动作操作” 改变。 宏支持 范围影响可用宏的范围。例如，脚本支持与用户相关的宏 ({USER.*})，以允许传递有关启动脚本的用户的信息。但是，如果脚本范围是操作操作，则不支持它们，因为操作操作会自动执行。 要了解支持哪些宏，请搜索 “基于触发器的通知和命令/基于触发器的命令”，支持的宏 表中的 “手动主机操作脚本” 和 “手动事件操作脚本”。请注意，如果宏可以解析为带有空格的值（例如，主机名），请不要忘记根据需要引用。
菜单路径		脚本所需的菜单路径。例如，`Default` 或 `Default/`，将在各自的目录中显示脚本。菜单可以嵌套，例如主菜单/子菜单 1/子菜单 2。通过监控部分中的主机/事件菜单访问脚本时，它们将根据给定目录进行组织。 仅当 “手动主机操作” 或 “手动事件操作” 选择为 范围时，才会显示此字段。
类型		点击相应的按钮选择脚本类型： **Webhook**, **Script**, **SSH**, **Telnet** or **IPMI** 命令。
	脚本类型：Webhook	
	参数	将 webhook 变量指定为属性值对。 另请参阅：Webhook 媒体配置。 宏和参数值支持自定义用户宏。宏支持取决于脚本的范围 (参见上面的范围)。
	脚本	在单击参数字段（或旁边的查看/编辑按钮）时出现的块中输入 JavaScript 代码。 宏支持取决于脚本（请参阅上面的 *Scope*）。 另请参阅：Webhook 媒体配置，其他 Javascript 对象。
	超时	JavaScript 执行超时（1-60 秒，默认 30 秒）。 支持时间后缀，例如 30s，1m。
	脚本类型：脚本	
	执行于	点击相应的按钮执行 shell 脚本： **Zabbix agent** - 脚本将由 Zabbix agent 执行（如果 system.run 项是allowed) 在主机上 **Zabbix server (proxy)** - 脚本将由 Zabbix server 或 proxy 执行（如果由EnableRemoteCommands) - 取决于主机是由服务器监控还是代理监控 **Zabbix server** - 脚本将仅由 Zabbix server 执行
	命令	输入要在脚本中执行的命令的完整路径。 宏支持取决于脚本的范围（参见上面的 *Scope*）。支持自定义用户宏。
	脚本类型：SSH	
	认证方式	选择认证方式-密码或公钥。
	用户名	输入用户名。
	密码	输入密码。 如果选择 “密码” 作为验证方法，则此字段可用。

参数		说明
	公钥文件	输入公钥文件的路径。 如果选择 “公钥” 作为身份验证方法，则此字段可用。
	私钥文件	输入私钥文件的路径。 如果选择 “公钥” 作为验证方法，则此字段可用。
	密码	输入密码。 如果选择 “公钥” 作为身份验证方法，则此字段可用。
	端口	输入端口。
	命令	输入命令。 宏支持取决于脚本的范围（参见上面的 *Scope*）。支持自定义用户宏。
	脚本类型：Telnet	
	用户名	输入用户名。
	密码	输入密码。
	端口	输入端口。
	命令	输入命令。 宏支持取决于脚本的范围（参见上面的 *Scope*）。支持自定义用户宏。
	脚本类型：IPMI	
	命令	输入 IPMI 命令。 宏支持取决于脚本的范围（参见上面的 *Scope*）。支持自定义用户宏。
描述		输入脚本的描述。
主机组		选择脚本可用于的主机组（或 *All* 用于所有主机组）。
用户组		选择脚本可用的用户组（或所有用户组的全部）。 此字段仅在 “手动主机操作” 或 “手动事件操作’ 被选为 *Scope*。
必需的主机权限		选择主机组的权限级别 - *Read* 或 *Write*。Only users with the required permission level will have access to executing the script. This field is displayed only if ’Manual host action’ or ’Manual event action’ is selected as *Scope*.
启用确认		标记复选框以在执行脚本之前显示确认消息。此功能对于具有潜在危险的操作（如重新启动脚本）或可能需要很长时间的操作可能特别有用。 仅当 “手动主机操作” 或 “手动事件操作” 选为 *Scope* 时才会显示此选项.
确认文本		为使用上面的复选框启用的确认弹出窗口输入自定义确认文本（例如，远程系统将重新启动。您确定吗？)。要查看文本的外观，请单击字段旁边的测试确认。 支持 {HOST.*} 和 {USER.*} 宏。支持自定义用户宏。 注意：测试确认消息时不会展开宏。 仅当 “手动主机操作” 时才会显示此字段’ 或’ 手动事件操作’ 被选为 *Scope*。

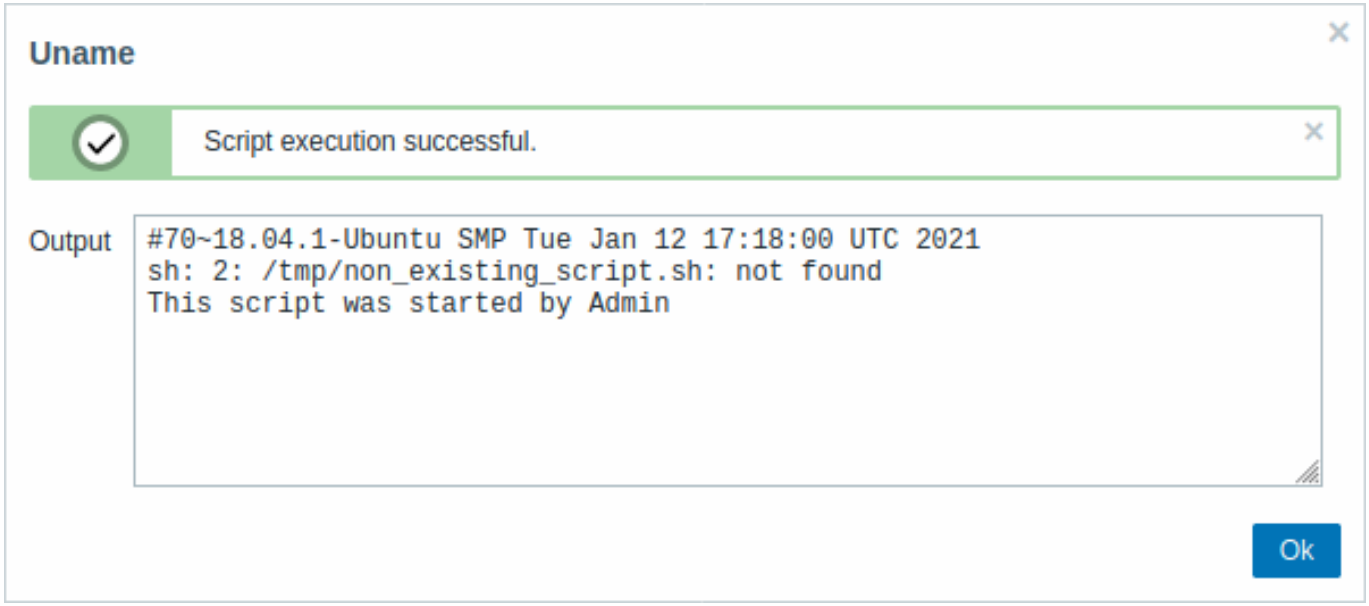
脚本执行和结果

Zabbix 服务器运行的脚本按照命令执行 部分中描述的顺序执行，包括退出代码检查。脚本结果将显示在脚本运行后将出现的弹出窗口。

注：脚本的返回值是标准输出和标准错误。

请参阅下面的脚本示例和结果窗口：

```
uname -v
/tmp/non_existing_script.sh
echo "This script was started by {USER.USERNAME}"
```



脚本结果不显示脚本本身。

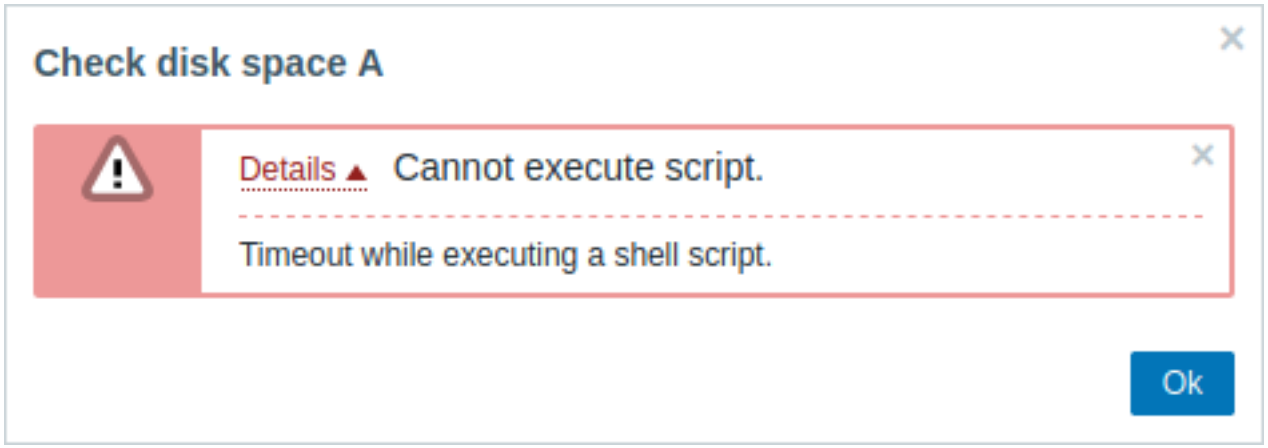
脚本超时

Zabbix agent

您可能会遇到执行脚本时发生超时的情况。

请参阅在 Zabbix agent 上运行的脚本示例和下面的结果窗口：

```
sleep 5
df -h
```



在这种情况下，错误消息如下：

```
Timeout while executing a shell script.
```

为了避免这种情况，建议通过修改【Zabbix agent configuration】(/manual/appendix/ config/zabbix_agentd) 和Zabbix server 配置)。

如果Zabbix agent configuration 中的 Timeout 参数仍然更改，则会出现以下错误消息：

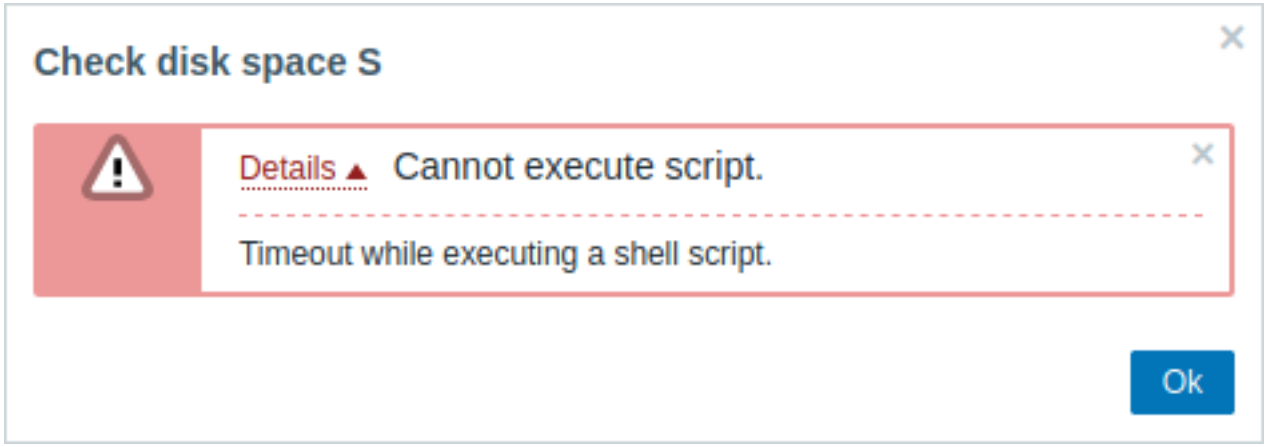
```
Get value from agent failed: ZBX_TCP_READ() timed out.
```

这意味着在Zabbix agent configuration中进行了修改，并且需要在Zabbix server configuration中修改 Timeout 设置。

Zabbix server/proxy

查看在 Zabbix server 上运行的脚本示例和下面的结果窗口：

```
sleep 11
df -h
```



还建议优化脚本本身（而不是通过修改 [Zabbix server configuration]（/manual/appendix/config/zabbix_server）将 TrapperTimeout 参数调整为相应的值（在我们的例子中，>'11'））。

9 队列

概述

在 管理 → 队列菜单下显示等待更新的项目。

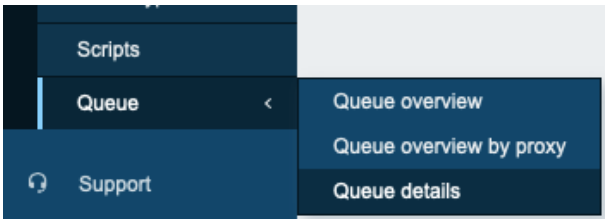
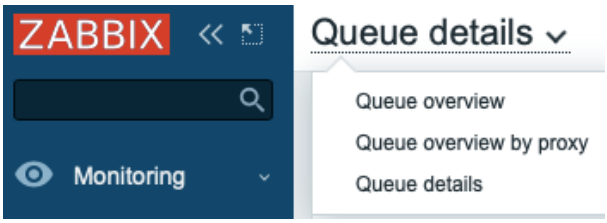
理想情况下，当您打开此部分时，它应该全部为 "绿色"，表示队列中没有项目。如果所有项目都立即更新，则没有等待。但是，由于服务器性能不足、连接问题或 agent 问题，某些项目可能会延迟，信息会显示在此部分中。有关详细信息，请参阅队列 部分。

Note:
队列仅在 Zabbix server 运行时可用

管理 → 队列部分包含以下页面：

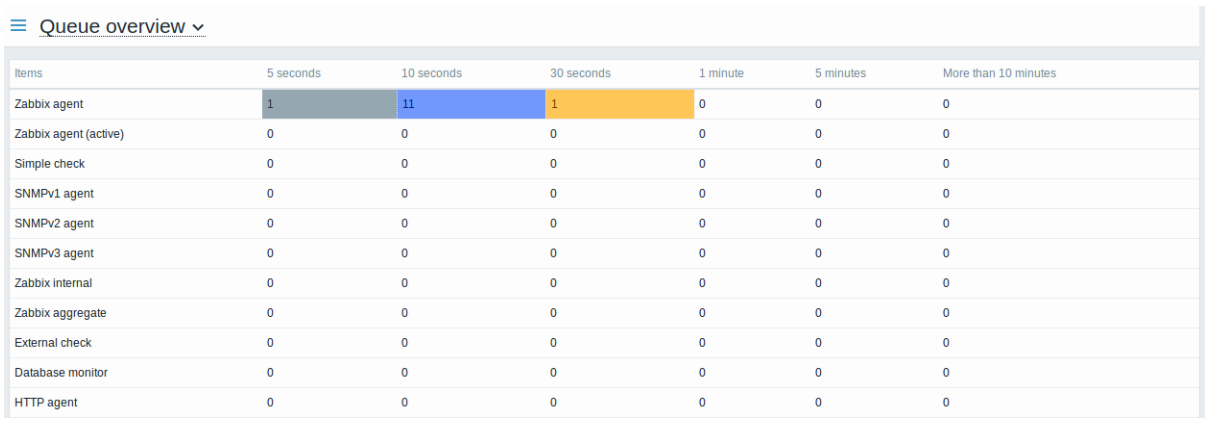
- 队列概览——按项目类型显示队列；
- proxy 队列概览——显示 proxy 队列；
- 队列详细信息——显示延迟项目列表。

按下 管理菜单部分中的 队列后，可用页面列表出现。也可以使用左上角的标题下拉菜单在页面之间切换。

.	.
	
三级菜单。	标题下拉菜单。

监控项类型概览

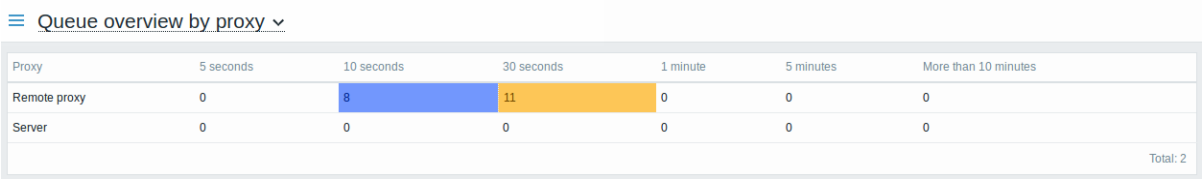
在此屏幕中，很容易找到问题是否与一种或多种监控项类型有关。



每行包含一个监控项类型。每列显示等待监控项的数量 - 分别等待 5-10 秒/10-30 秒/30-60 秒/1-5 分钟/5-10 分钟或超过 10 分钟。

proxy 概览

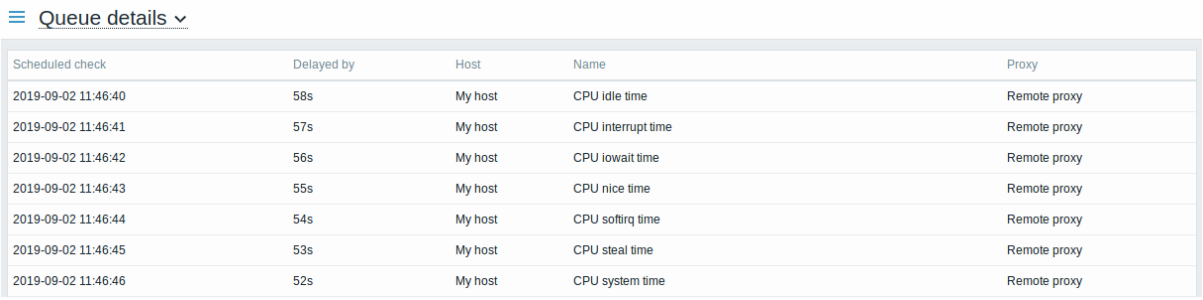
在此屏幕中，很容易找到问题是否与 poxy、server 之一有关。



每行包含一个 proxy，server 位于列表的最后。每列显示等待监控项的数量 - 分别等待 5-10 秒/10-30 秒/30-60 秒/1-5 分钟/5-10 分钟或超过 10 分钟。

等待的监控项清单

在此屏幕中，列出了每个等待的监控项。



显示数据：

栏目	说明
定期检查	显示检查到期的时间。
延迟	显示延迟的长度。
主机	显示监控项的主机名。
名称	显示等待监控项的名称。
Proxy	如果主机通过 proxy 监控，则显示 proxy 名称。

可能的错误信息

可能会遇到没有显示数据并出现以下错误消息的情况：



这种情况下的错误信息如下：

无法显示项目队列。没有权限

当 zabbix.conf.php 中的 PHP 配置参数 $ZBX_SERVER_PORT 或 $ZBX_SERVER 指向使用不同数据库的现有 Zabbix 服务器时，就会发生这种情况。

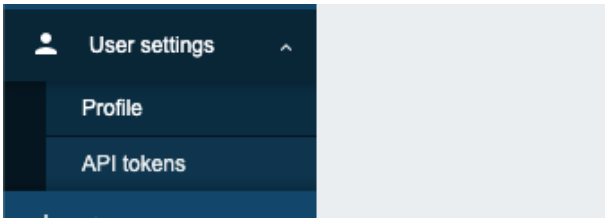
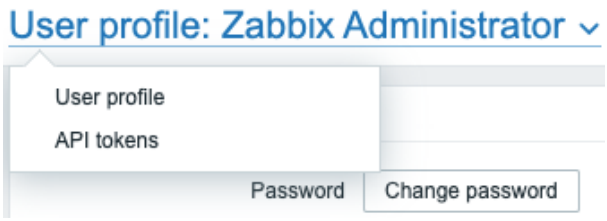
3 用户设置

概述

根据用户角色权限，用户设置部分可能包含以下页面：

- 用户配置文件 - 用于自定义某些 Zabbix 前端功能；
- *API* 令牌 - 用于管理分配给当前用户的 API 令牌。

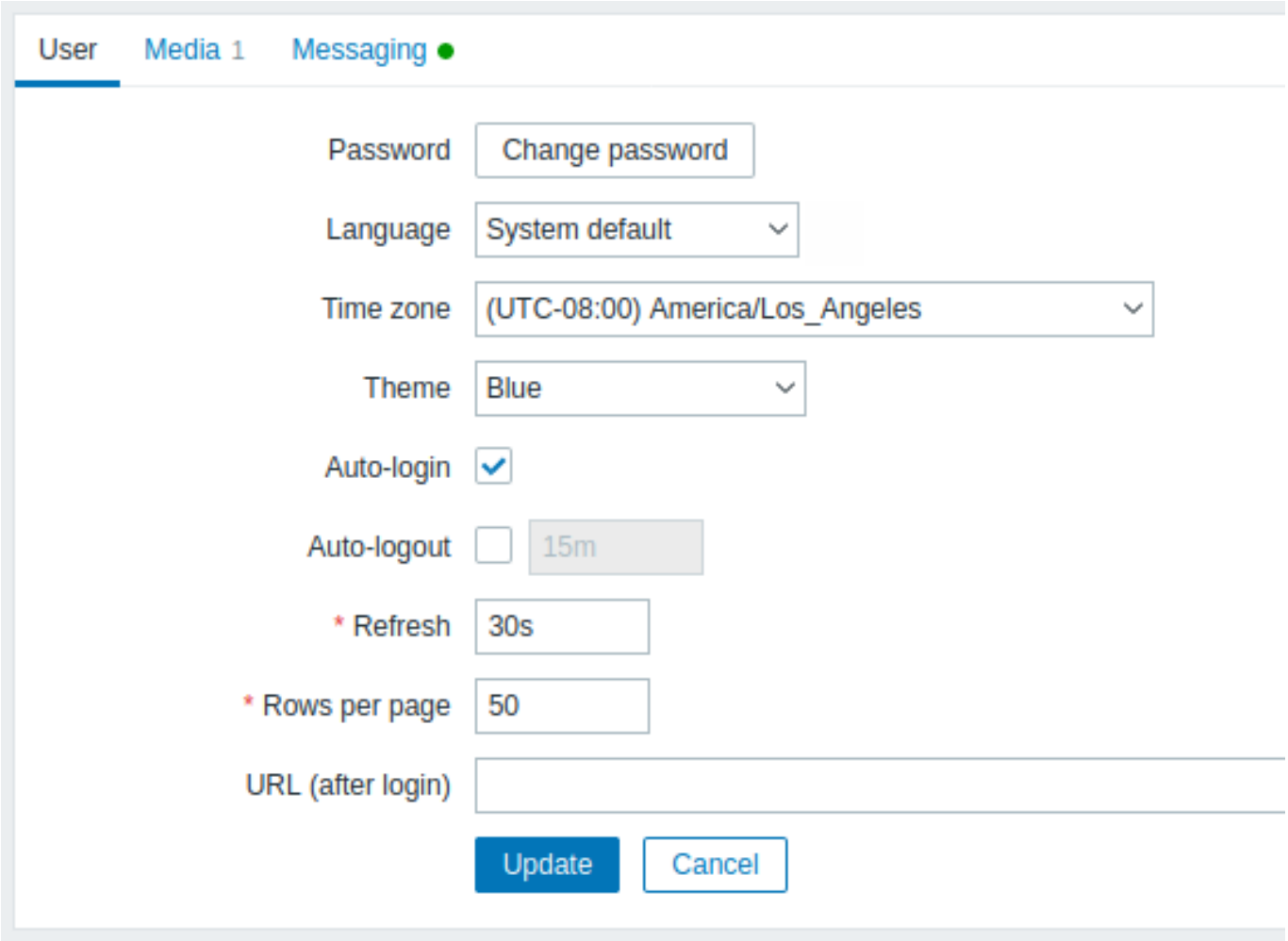
按下 Zabbix 菜单底部附近的  用户图标后，可用页面列表出现（对访客不可用）。也可以使用左上角的标题下拉菜单在页面之间切换。

.	.
	
三级菜单。	标题下拉菜单。

用户配置

用户配置部分提供了设置自定义界面语言、颜色主题、列表中显示的行数等选项。此处所做的更改将仅应用于当前用户。

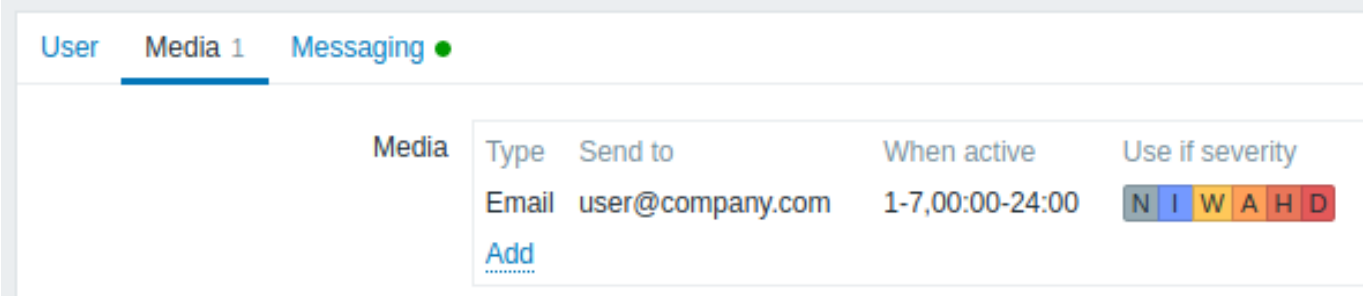
用户选项卡允许您设置各种用户偏好。



参数	说明
密码	点击链接显示两个用于输入新密码的字段。
语言	选择界面语言或选择系统默认以使用默认系统设置。 有关详细信息，请参阅安装其他前端语言。
时区	选择时区以在用户级别覆盖全局时区，或选择系统默认以使用全局时区设置。
主题	为您的个人资料选择一个颜色主题： 系统默认 - 使用默认系统设置 蓝色 - 标准蓝色主题 深色 - 替代深色主题 高对比度浅色 - 高对比度浅色主题 高对比度深色 - 高对比度深色主题
自动登录	标记此复选框以使 Zabbix 记住您并自动登录 30 天。浏览器 cookie 用于此目的。

参数	说明
自动注销	选中此复选框后，您将在设置的秒数（最少 90 秒，最多 1 天）后自动注销。 时间后缀 是支持，例如 90s、5m、2h、1d。 请注意，此选项将不起作用： * 当监控菜单页面执行背景信息刷新时。如果页面在特定时间间隔内刷新数据（仪表板、图表、最新数据等）保持打开状态，则会话生命周期会延长，分别禁用自动注销功能； * 如果使用 *Remember me for* 登录 *30* 天选项已选中。 自动注销可以接受 0，这意味着在配置文件设置更新后自动注销被禁用。
刷新	您可以在监控菜单上设置页面中信息的刷新频率，但仪表板除外，它对每个小部件使用自己的刷新参数。 [时间后缀](/manual/appendix/后缀）被支持，例如 30s，5m，2h，1d。
每页行数	您可以设置列表中每页显示的行数。更少的行（和更少的显示记录）意味着更快的加载时间。
URL（登录后）	您可以设置登录后显示的特定 URL。例如，它可以是 监控 → 触发器的 URL，而不是默认的 监控 → 仪表板。

媒介选项卡允许您为用户指定媒介详细信息，例如类型、要使用的地址以及何时使用它们来传递通知。

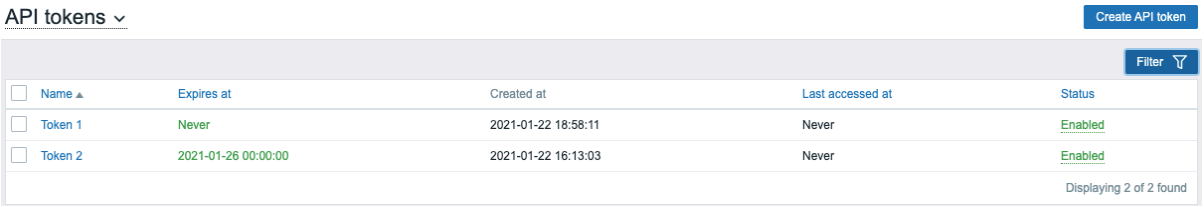


Note:
只有管理员级别 用户（管理员和超级管理员）可以更改自己的媒介详细信息。

消息选项卡允许您设置全局通知。

API 令牌

API 令牌部分允许查看分配给用户的令牌、编辑令牌详细信息和创建新令牌。只有在用户角色 设置中允许 管理 *API* 令牌操作时，此部分才对用户可用。



您可以按名称、到期日期或状态（启用/禁用）过滤 API 令牌。单击列表中的令牌状态可快速启用/禁用令牌。您还可以通过在列表中选择令牌然后单击列表下方的启用/禁用按钮来批量启用/禁用令牌。

Attention:
用户无法在 Zabbix 中查看分配给他们的令牌的 *Auth token* 值。*Auth token* 值仅显示一次 - 在创建令牌后立即显示。如果它已经丢失，则必须重新生成令牌。

1 全局通知

概述

全局通知是一种在 Zabbix 前端屏幕上显示当前正在发生的问题的方式。

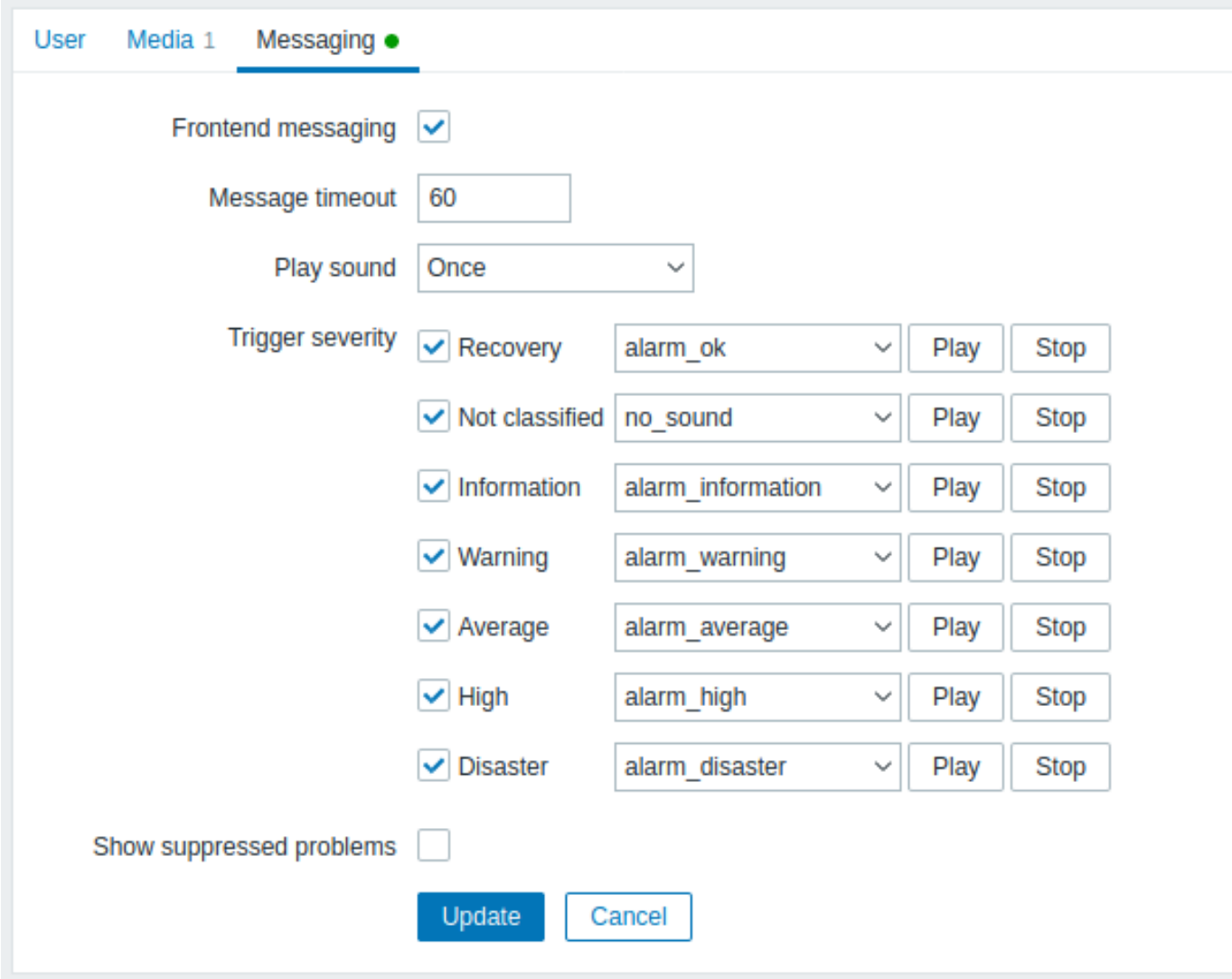
如果没有全局通知，在 问题或 仪表板以外的其他位置工作将不会显示有关当前正在发生的问题的任何信息。无论您身在何处，全局通知都会显示此信息。

全局通知涉及显示消息和 [播放声音]（声音）。

Attention:
默认情况下，最近的浏览器版本可能会禁用声音的自动播放。在这种情况下，您需要手动更改此设置。

配置

可以在用户配置设置 的 消息选项卡中为每个用户启用全局通知。



参数	说明
前端消息传递	标记复选框以启用全局通知。
消息超时	您可以设置消息显示的时长。默认情况下，消息将在屏幕上停留 60 秒。 支持时间后缀，例如 30s，5m，2h，1d。
播放声音	您可以设置声音的播放时间。 一次 - 声音播放一次并完全播放。 **10** 秒 - 声音重复 10 秒。 消息超时 - 消息可见时重复声音。
触发严重性	您可以设置将激活全局通知和声音的触发严重性。您还可以选择适合各种严重性的声音。 如果未标记严重性，则根本不会显示任何消息。 此外，仅针对已标记的严重性显示恢复消息。因此，如果您标记 *Recovery* 和 *Disaster*，将显示全局通知以显示问题和灾难严重性触发器的恢复。
显示抑制的问题	标记复选框以显示由于主机维护而被抑制（未显示）的问题的通知。

显示全局消息

当消息到达时，它们会显示在右侧的浮动部分中。该部分可以通过拖动部分标题自由重新定位。



对于本节，有几个控件可用：

- · 延后警报按钮使当前活动的警报声音静音；
- · 静音/取消静音按钮在播放和不播放警报声之间切换。

2 浏览器提示音

概述

声音用于全局通知。

对于要在 Zabbix 前端播放的声音，必须在用户配置文件 消息选项卡中启用前端消息，并检查所有触发严重性，并且还应在全局通知弹出窗口中启用声音。

如果由于某些原因无法在设备上播放音频，全局通知弹出窗口中的 按钮将永久保持 “静音” 状态，并显示 “无法支持此设备的通知音频” 消息。将在悬停在 按钮上时显示。

仅支持 MP3 格式声音，包括默认音频剪辑。

Zabbix 前端的声音已在 Linux 上的最新 Firefox/Opera 浏览器和 Windows 上的 Chrome、Firefox、Microsoft Edge、Opera 和 Safari 浏览器中测试成功。

Attention:
默认情况下，最近的浏览器版本可能会禁用声音的自动播放。在这种情况下，您需要手动更改此设置。

4 全局搜索

可以在 Zabbix 前端搜索主机、主机组和模板。

搜索输入框位于菜单中 Zabbix logo 的下方。可以通过按 回车键或单击 搜索图标来开始搜索。



如果主机名的任何部分包含输入的字符串，则会出现一个下拉列表，列出所有此类主机（匹配部分以橙色突出显示）。如果该主机的可见名称与作为搜索字符串输入的技术名称匹配，则下拉列表还将列出该主机；匹配的主机将被列出，但没有任何突出显示。

可搜索的属性

可以通过以下属性搜索主机：

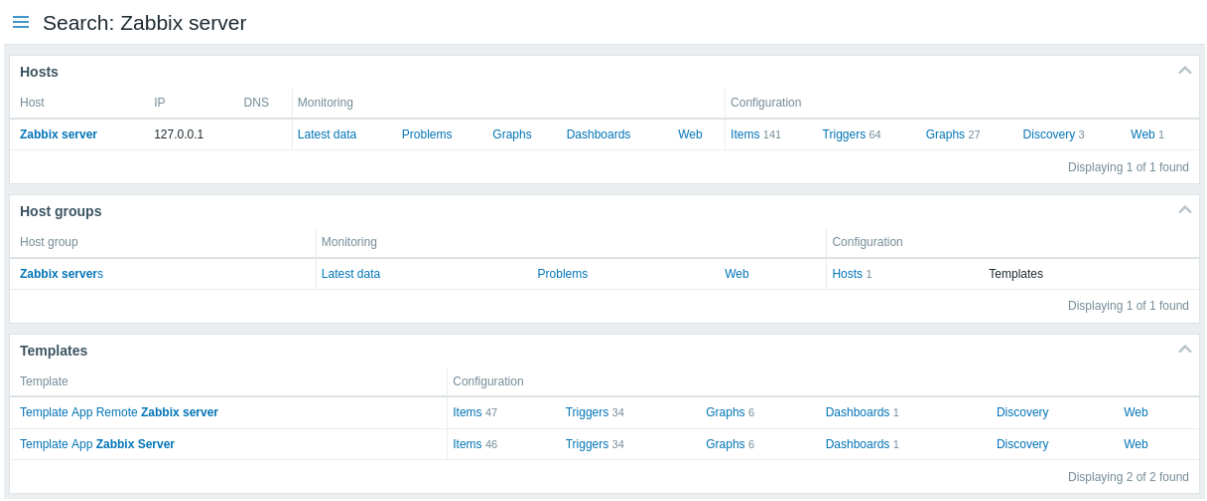
- 主机名
- 可见的名字
- IP 地址
- DNS 名称

可以按名称搜索主机组。指定父主机组会隐式选择所有嵌套主机组。

可以按名称或可见名称搜索模板。如果您按与（模板/主机的）可见名称不同的名称进行搜索，则在搜索结果中，它会显示在括号中可见名称的下方。

搜索结果

搜索结果由主机、主机组和模板的三个独立块组成。



可以折叠/展开每个单独的区域。条目计数显示在底部，例如，*Displaying 13 of 13 found*。一个区域内显示的总条目数限制为 100 个。

每个条目都提供了指向监控和配置数据的链接。请参阅链接的完整列表。

对于所有配置数据（例如监控项、触发器、图表），找到的实体数量由实体名称旁边的灰色数字显示。注意，如果有零个实体，则不显示数字。

启用的主机显示为蓝色，禁用的主机显示为红色。

可用链接

对于每个条目，以下链接可用：

- 主机
  - 监控
    - 最新数据
    - 问题
    - 图表

        * 主机仪表板
        * 网络场景
    - 配置
        * 监控项
        * 触发器
        * 图表
        * 自动发现规则
        * 网络场景

- 主机组
    - 监控
        * 最新数据
        * 问题
        * 网络场景
    - 配置
        * 主机
        * 模板

- 模板
    - 配置
        * 监控项
        * 触发器
        * 图表
        * 模板仪表板
        * 自动发现规则
        * 网络场景

5 前端维护模式

概述

可以暂时禁用 Zabbix Web 前端以禁止对其进行访问。这对于保护 Zabbix 数据库免受用户发起的任何更改非常有用，从而保护数据库的完整性。

当 Zabbix 前端处于维护模式时，可以停止 Zabbix 数据库并执行维护任务。

用户使用已定义 IP 地址将能够在维护模式下正常访问前端。

配置

为了启用维护模式，必须修改 `maintenance.inc.php` 文件（位于网络服务器上 Zabbix HTML 文档目录的 /conf 中）以取消注释以下行：

```
// 维护模式。
define('ZBX_DENY_GUI_ACCESS', 1);

// IP 地址数组，允许连接到前端（可选）。
$ZBX_GUI_ACCESS_IP_RANGE = array('127.0.0.1');

// 警告屏幕上显示的消息（可选）。
$ZBX_GUI_ACCESS_MESSAGE = 'We are upgrading MySQL database till 15:00. Stay tuned...';
```

大多数情况下，“maintenance.inc.php” 文件位于 Web 服务器上 Zabbix HTML 文档目录的 “/conf” 中。但是，目录的位置可能因操作系统和它使用的 Web 服务器而异。

例如，位置：

- ·SUSE and RedHat is `/etc/zabbix/web/maintenance.inc.php`.
- ·Debian-based systems is `/usr/share/zabbix/conf/`.

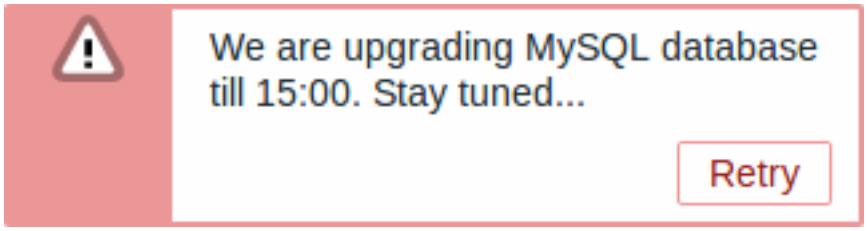
可以参见Copying PHP files.

参数	详情
ZBX_DENY_GUI_ACCESS	启用维护模式： 1 - 启用维护模式，否则禁用

参数	详情
ZBX_GUI_ACCESS_IP_RANGE	IP 地址数组，允许连接到前端（可选）。 例如： `array('192.168.1.1', ' 192.168.1.2')`
ZBX_GUI_ACCESS_MESSAGE	您可以输入一条消息来通知用户有关维护的信息（可选）。

显示

在维护模式下尝试访问 Zabbix 前端时，将显示以下屏幕。屏幕每 30 秒刷新一次，以便在维护结束时无需用户干预即可返回正常状态。



ZBX_GUI_ACCESS_IP_RANGE 中定义的 IP 地址将能够像往常一样访问前端。

6 页面参数

概述

大多数 Zabbix Web 界面页面都支持各种 HTTP GET 参数来控制显示的内容。它们可以通过在 URL 之后指定 “参数 = 值 “对来传递，通过问号 (?) 与 URL 隔开，并通过与号 (&) 彼此隔开。

监控 → 问题

支持以下参数：

- `show` - 过滤选项 “显示”：1 - 最近的问题，2 - 全部，3 - 处于问题状态
- `name` - 过滤选项 “问题”：自由格式字符串
- `severities` - 过滤选项”Severity”：所选严重性的数组，格式为’severities[*]=*’（用严重性级别替换 *）：0 - 未分类，1 - 信息，2 - 警告，3 - 一般严重，4 - 严重，5 - 灾难
- `inventory` - 过滤选项 “主机库存”：资产字段数组：[field]、[value]
- `evaltype` - 过滤选项 “Tags”，标签过滤策略：0 - And/Or, 2 - Or
- `tags` - 过滤选项 “标签”：定义标签的数组：[tag]、[operator]、[value]
- `show_tags` - 过滤选项 “显示标签”：0 - 无，1 - 一，2 - 二，3 - 三
- `tag_name_format` - 过滤选项 “标签名称”：0 - 全名，1 - 缩写，2 - 无
- `tag_priority` - 过滤选项 “标签显示优先级”：标签显示优先级的逗号分隔字符串
- `show_suppressed` - 过滤选项 “显示抑制的问题”：应该是’show_suppressed=1’ 来显示
- `unacknowledged` - 过滤选项 “仅显示未确认”：应该是’unacknowledged=1’ 才能显示
- `compact_view` - 过滤选项 “Compact view”：应该是’compact_view=1’ 来显示
- `highlight_row` - 过滤选项 “突出显示整行”（使用问题颜色作为每个问题行的背景颜色）：应该为 “1” 以突出显示；只有在设置了’compact_view’ 时才能设置
- `filter_name` - 过滤器属性选项 “名称”：自由格式字符串
- `filter_show_counter` - 过滤属性选项 “显示记录数”：1 - 显示，0 - 不显示
- `filter_custom_time` - 过滤器属性选项 “设置自定义时间段”：1 - 设置，0 - 不设置
- `sort` - 排序列：时钟、主机、严重性、名称
- `sortorder` - 排序顺序或结果：DESC - 降序，ASC - 升序
- `age_state` - 过滤选项 “持续时间小于”：应该是 “age_state = 1” 以启用 “持续时间”。仅在’show’ 等于 3 时使用。
- `age` - 过滤选项 “持续时间小于”：天
- `groupids` - 过滤选项 “主机组”：主机组 ID 数组
- `hostids` - 过滤选项 “Hosts”：主机 ID 数组
- `triggerids` - 过滤选项 “触发器”：触发器 ID 数组
- `show_timeline` - 过滤选项 “显示时间线”：应该是’show_timeline=1’ 来显示
- `details` - 过滤选项 “显示详细信息”：应该是’details=1’ 来显示
- `from` - 日期范围开始，可以是 “相对”（例如：now-1m）。仅在’filter_custom_time’ 等于 1 时使用。
- `to` - 日期范围结束，可以是 “相对”（例如：now-1m）。仅在’filter_custom_time’ 等于 1 时使用。

信息亭（Kiosk）模式

可以使用 URL 参数激活支持的前端页面中的信息亭模式。例如，在仪表板中：

- /zabbix.php?action=dashboard.view&kiosk=1 - 激活信息亭模式
- /zabbix.php?action=dashboard.view&kiosk=0 - 激活普通模式

幻灯片模式

可以在仪表板中激活幻灯片模式：

- /zabbix.php?action=dashboard.view&slideshow=1 - 激活幻灯片模式

7 定义

概述

虽然可以使用前端本身配置前端中的许多东西，但目前只能通过编辑定义文件来进行一些自定义。

该文件是 defines.inc.php，位于 Zabbix HTML 文档目录的 /include 中。

参数

此文件中用户可能感兴趣的参数：

- ZBX_MIN_PERIOD

最小图表周期，以秒为单位。默认为一分钟。

- GRAPH_YAXIS_SIDE_DEFAULT

向自定义图表添加项目时，简单图表中 Y 轴的默认位置和下拉框的默认值。可能的值：0 - 左，1 - 右。

默认值：0

- ZBX_SESSION_NAME（自 4.0.0 起可用）

用作 Zabbix 前端会话 cookie 名称的字符串。

默认值：zbx_sessionid

- ZBX_DATA_CACHE_TTL（从 5.2.0 开始可用）

用于使Vault 响应 的数据缓存无效的 TTL 超时秒数。设置 0 以禁用 Vault 响应缓存。

默认值：60

- SUBFILTER_VALUES_PER_GROUP (自 6.0.5 起可用)

每组子过滤值的数量（例如，最新数据的子过滤）。

默认值：1000

8 定制主题风格

概述

默认情况下，Zabbix 提供了许多预定义的主题。您可以按照此处提供的分步程序来创建您自己的。如果您创建了一些不错的东西，请随时与 Zabbix 社区分享您的工作成果。

步骤 1

要定义您自己的主题，您需要创建一个 CSS 文件并将其保存在 assets/styles/ 文件夹中（例如，*custom-theme.css*）。您可以从不同的主题复制文件并基于它创建主题，也可以从头开始。

步骤 2

将您的主题添加到 APP::getThemes() 方法返回的主题列表中。您可以通过覆盖 APP 类中的 ZBase::getThemes() 方法来做到这一点。这可以通过在 *include/classes/core/APP.php* 的右大括号之前添加以下代码来完成：

```
public static function getThemes() {
    return array_merge(parent::getThemes(), [
        'custom-theme' => _('Custom theme')
    ]);
}
```

Attention:
请注意，您在第一对引号中指定的名称必须与不带扩展名的主题文件的名称匹配。

要添加多个主题，只需将它们列在第一个主题下即可，例如：

```
public static function getThemes() {
    return array_merge(parent::getThemes(), [
        'custom-theme' => _('Custom theme'),
        'anothertheme' => _('Another theme'),
        'onemoretheme' => _('One more theme')
    ]);
}
```

请注意，除了最后一个主题之外的每个主题都必须有一个尾随逗号。

Note:
要更改图形颜色，必须在 *graph_theme* 数据库表中添加条目。

步骤 3

激活新主题。

在 Zabbix 前端，您可以将此主题设置为默认主题，也可以在用户配置文件中更改您的主题。

享受新的外观和感觉！

9 调试模式

概述

调试模式可用于诊断前端页面的性能问题。

配置

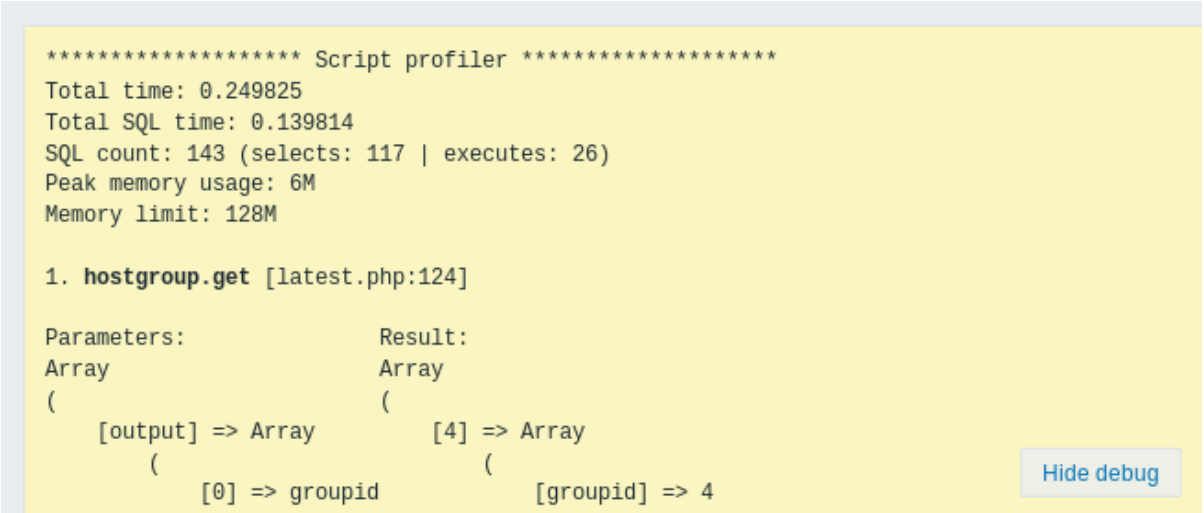
可以为属于用户组的用户激活调试模式：

- 当配置一个用户组;
- 当查看配置的用户组。

当为用户组启用 调试模式时，其用户将在浏览器窗口的右下角看到一个 调试按钮：



单击 调试按钮会在页面内容下方打开一个新窗口，其中包含页面的 SQL 统计信息，以及 API 调用和单个 SQL 语句的列表：



如果页面出现性能问题，此窗口可用于搜索问题的根本原因。

Warning:
启用 调试模式会对前端性能产生负面影响。

10 Zabbix 使用的 cookie

概述

此页面提供了 Zabbix 使用的 cookie 列表。

名称	描述	值	过期/最大年龄	HttpOnly[a]	安全[a]
ZBX_SESSION_NAME	存储会话数据，存储为 base64 编码的 JSON		会话（浏览会话结束时过期）	+	+（仅当在 Web 服务器上启用 HTTPS 时）
tab	活动标签号；此 cookie 仅用于具有多个选项卡的页面（例如 *Host*、*Trigger* 或 *Action* 配置页面），并在用户从主选项卡导航到另一个选项卡时创建（例如 *Tags* 或 *Dependencies* 选项卡）。 0 用于主选项卡。	示例：1	会话（浏览会话结束时到期）	-	-
browserwarning_ignore	是否应该忽略有关使用过时浏览器的警告。	yes	会话（浏览会话结束时过期）	-	-
system-message-ok	页面重新加载后立即显示的消息。	纯文本消息	会话（浏览会话结束时到期）或页面重新加载后立即显示	+	-
system-message-error	重新加载页面后立即显示的错误消息。	纯文本消息	会话（在浏览会话结束时到期）或页面重新加载后立即显示	+	-

[a] 根据 声明 这些是芯片引脚上的电压，一般来说可能需要缩放。

[a] `Secure` 表示 cookie 只能通过来自客户端的安全 HTTPS 连接传输。当设置为 “true” 时，只有存在安全连接时才会设置 cookie。

Note:
不支持通过 webserver 指令在 Zabbix cookie 上强制使用’HttpOnly’ 标志。

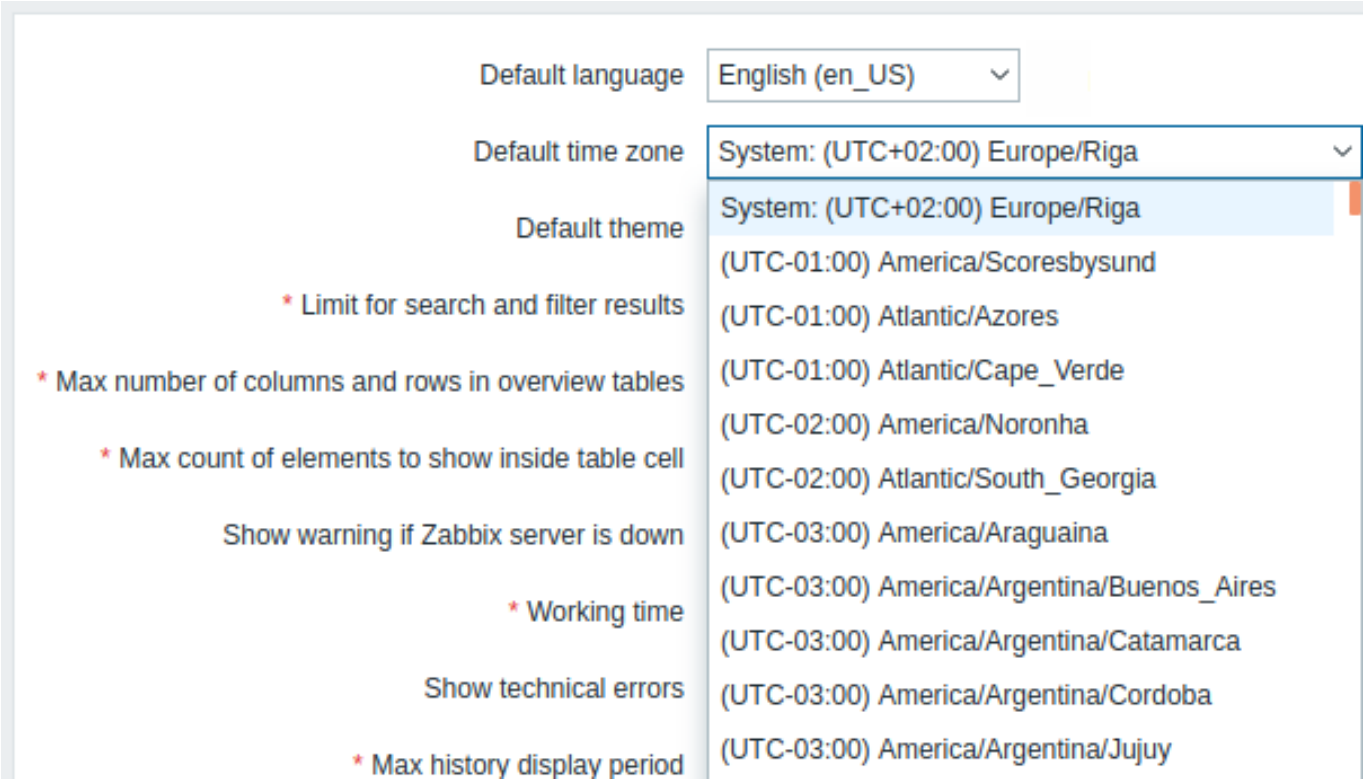
[5] ：当 `HttpOnly` 为’true’ 时，cookie 将只能通过 HTTP 协议访问。这意味着 cookie 将无法通过 JavaScript 等脚本语言访问。此设置可以有效帮助减少通过 XSS 攻击进行的身份盗用（尽管并非所有浏览器都支持）。

11 时区

概述

前端时区可以在前端全局设置，并针对个人用户进行调整。

[5] 根据 声明 这些是芯片引脚上的电压，一般来说可能需要缩放。



如果选择 *System*，前端将使用 Web 服务器时区（包括 php.ini 的'date.timezone' 的值，如果设置），而 Zabbix 服务器将使用机器所在的时区继续运行。

Note:
Zabbix 服务器将仅在通知中扩展宏时使用指定的全局/用户时区（例如 {EVENT.TIME} 可以为每个用户扩展到不同的时区）以及发送通知的时间限制（请参阅 "当活动时" 设置用户媒介配置）。

配置

全局时区：

- 安装前端时可以手动设置
- 可以在 *Administration* → *General* → *GUI*中修改

用户级时区：

- 可以在配置/更新 用户时设置
- 可以由每个用户在他们的用户配置文件 中设置

13 Resetting password

Overview This section describes the steps for resetting user passwords in Zabbix.

Steps Turn to your Zabbix administrator if you have forgotten your Zabbix password and cannot log in.

A Super administrator user can change passwords for all users in the user configuration form.

If a Super administrator has forgotten their password and cannot log in, the following SQL query must be run to apply the default password to the Super admin user (replace 'Admin' with the appropriate Super admin username):

```
UPDATE users SET passwd = '$2a$10$ZXIvHAEP2ZM.dLXTm6uPHOMVlARXX7cqjbhM6Fn0cANzkCQBWpMrS' WHERE username =
```

After running this query, the user password will be set to *zabbix*. Make sure to change the default password on the first login.

19. API

概述　Zabbix API 允许您以编程方式检索和修改 Zabbix 的配置，并提供对历史数据的访问。它被广泛用于：

- 创建新的应用程序与 Zabbix 一起工作；
- 将 Zabbix 集成到第三方软件中；
- 自动化日常任务。

Zabbix API 是基于 HTTP 的 API，并且作为 web 前端的一部分提供。它使用 JSON-RPC 2.0 协议，这意味着两件事：

- 'API' 由一系列独立的方法组成；
- 客户端和 API 之间的请求和响应使用 JSON 格式进行编码。

有关协议和 JSON 的更多信息，请参见 JSON-RPC 2.0 规范 和 JSON 格式主页。

有关将 Zabbix 功能集成到您的 Python 应用程序中的更多信息，请参见 zabbix_utils Python 库用于 Zabbix API。

结构　API 由一系列方法组成，这些方法名义上被分组到不同的 API 中。每种方法执行一个特定任务。例如，`host.create` 方法属于 *host* API，用于创建新的主机。从历史上看，API 有时被称为" 类"。

> **Note:**
> 大多数 API 至少包含四种方法：`get`, `create`, `update` 和 `delete`，分别用于获取、创建、更新和删除数据，但有些 API 可能提供完全不同的一组方法。

执行请求　一旦您设置好了前端，就可以使用远程 HTTP 请求来调用 API。为此，您需要发送 HTTP POST 请求到前端目录中的 `api_jsonrpc.php` 文件。例如，如果您的 Zabbix 前端安装在 `https://example.com/zabbix`，调用 `apiinfo.version` 方法的 HTTP 请求可能看起来像这样：

```
curl --request POST \
  --url 'https://example.com/zabbix/api_jsonrpc.php' \
  --header 'Content-Type: application/json-rpc' \
  --data '{"jsonrpc":"2.0","method":"apiinfo.version","params":{},"id":1}'
```

请求必须将 `Content-Type` 头部设置为以下值之一：`application/json-rpc`、`application/json` 或 `application/jsonrequest`。

请求对象包含以下属性：

- `jsonrpc` - API 使用的 JSON-RPC 协议版本（Zabbix API 实现了 JSON-RPC 版本 2.0）；
- `method` - 正在调用的 API 方法；
- `params` - 将传递给 API 方法的参数；
- `id` - 请求的任意标识符。

如果请求正确，API 返回的响应应该像这样：

```
{
    "jsonrpc": "2.0",
    "result": "7.0.0",
    "id": 1
}
```

响应对象依次包含以下属性：

- `jsonrpc` - JSON-RPC 协议的版本；
- `result` - 方法返回的数据；
- `id` - 对应请求的标识符。

示例工作流程　下面的部分将以更详细的示例向您展示一些使用方法。

认证　要在 Zabbix 中访问任何数据，您需要：

- 使用现有的API 令牌（在 Zabbix 前端创建或使用Token API）；
- 使用通过user.login 方法获得的认证令牌。

例如，如果您想以标准 *Admin* 用户身份登录以获取一个新的认证令牌，那么 JSON 请求看起来可能像这样：

```
curl --request POST \
  --url 'https://example.com/zabbix/api_jsonrpc.php' \
  --header 'Content-Type: application/json-rpc' \
  --data '{"jsonrpc":"2.0","method":"user.login","params":{"username":"Admin","password":"zabbix"},"id":1}
```

如果您正确提供了凭据，API 返回的响应应该包含用户认证令牌：

```
{
    "jsonrpc": "2.0",
    "result": "0424bd59b807674191e7d77572075f33",
    "id": 1
}
```

获取主机　现在您有一个有效的用户认证令牌，可以用来访问 Zabbix 中的数据。例如，您可以使用host.get 方法来检索所有已配置的主机 的 ID、主机名和接口：

请求：

```
curl --request POST \
  --url 'https://example.com/zabbix/api_jsonrpc.php' \
  --header 'Authorization: Bearer ${AUTHORIZATION_TOKEN}' \
  --header 'Content-Type: application/json-rpc' \
  --data @data.json
```

Note:
data.json 是一个包含 JSON 查询的文件。您可以在 --data 参数中传递查询，而不是使用文件。

data.json

```
{
    "jsonrpc": "2.0",
    "method": "host.get",
    "params": {
        "output": [
            "hostid",
            "host"
        ],
        "selectInterfaces": [
            "interfaceid",
            "ip"
        ]
    },
    "id": 2
}
```

响应对象将包含有关主机的请求数据：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "hostid": "10084",
            "host": "Zabbix server",
            "interfaces": [
                {
                    "interfaceid": "1",
                    "ip": "127.0.0.1"
                }
            ]
        }
    ],
    "id": 2
}
```

Note:
出于性能考虑，始终建议列出您想要检索的对象属性。这样，您将避免检索所有内容。

创建新监控项　现在，使用您从上一个 host.get 请求中获取的数据，在主机"Zabbix server" 上创建一个新的监控项。这可以通过使用item.create 方法来完成：

```
curl --request POST \
  --url 'https://example.com/zabbix/api_jsonrpc.php' \
  --header 'Authorization: Bearer ${AUTHORIZATION_TOKEN}' \
  --header 'Content-Type: application/json-rpc' \
  --data '{"jsonrpc":"2.0","method":"item.create","params":{"name":"Free disk space on /home/joe/","key_":
```

一个成功的响应将包含新创建监控项的 ID，该 ID 可用于在随后的请求中引用该监控项。

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "24759"
        ]
    },
    "id": 3
}
```

Note:
item.create 方法以及其他 创建方法也可以接受对象数组，并用一个 API 调用创建多个监控项。

创建多个触发器　因此，如果 创建方法接受数组，您可以添加多个触发器，例如：

```
curl --request POST \
  --url 'https://example.com/zabbix/api_jsonrpc.php' \
  --header 'Authorization: Bearer ${AUTHORIZATION_TOKEN}' \
  --header 'Content-Type: application/json-rpc' \
  --data '{"jsonrpc":"2.0","method":"trigger.create","params":[{"description":"Processor load is too high
```

成功的响应将包含新创建的触发器的 ID：

```
{
    "jsonrpc": "2.0",
    "result": {
        "triggerids": [
            "17369",
            "17370"
        ]
    },
    "id": 4
}
```

更新监控项　将监控项的状态设置为 “0” 以启用它：

```
curl --request POST \
  --url 'https://example.com/zabbix/api_jsonrpc.php' \
  --header 'Authorization: Bearer ${AUTHORIZATION_TOKEN}' \
  --header 'Content-Type: application/json-rpc' \
  --data '{"jsonrpc":"2.0","method":"item.update","params":{"itemid":"10092","status":0},"id":5}'
```

成功的响应将包含更新后的监控项 ID：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "10092"
        ]
    },
    "id": 5
}
```

Note:
item.update 方法以及其他 更新方法也可以接受对象数组，并用一个 API 调用更新多个监控项。

更新多个触发器　将多个触发器的状态设置为 “0” 以启用它们：

```
curl --request POST \
  --url 'https://example.com/zabbix/api_jsonrpc.php' \
  --header 'Authorization: Bearer ${AUTHORIZATION_TOKEN}' \
  --header 'Content-Type: application/json-rpc' \
  --data '{"jsonrpc":"2.0","method":"trigger.update","params":[{"triggerid":"13938","status":0},{"triggeri
```

成功的响应将包含已更新触发器的 ID：

```
{
    "jsonrpc": "2.0",
    "result": {
        "triggerids": [
            "13938",
            "13939"
        ]
    },
    "id": 6
}
```

Note:
这是更新的首选方法。一些 API 方法，如 host.massupdate，允许编写更简洁的代码。然而，不建议使用这些方法，因为它们将在未来版本中被移除。

错误处理　到目前为止，您尝试的所有操作都运行得很好。但是，如果您尝试对 API 进行错误的调用会发生什么呢？尝试通过调用host.create 但不包含必需的 groups 参数来创建另一个主机：

```
curl --request POST \
  --url 'https://example.com/zabbix/api_jsonrpc.php' \
  --header 'Authorization: Bearer ${AUTHORIZATION_TOKEN}' \
  --header 'Content-Type: application/json-rpc' \
  --data '{"jsonrpc":"2.0","method":"host.create","params":{"host":"Linux server","interfaces":[{"type":1,
```

然后，响应将包含一个错误消息：

```
{
    "jsonrpc": "2.0",
    "error": {
        "code": -32602,
        "message": "Invalid params.",
        "data": "No groups for host \"Linux server\"."
    },
    "id": 7
}
```

如果发生错误，响应对象将不包含 result 属性，而是包含 error 属性，其中包含以下数据：

- code - 错误代码；
- message - 简短的错误摘要；
- data - 更详细的错误信息。

错误可能在各种情况下发生，例如，使用错误的输入值、会话超时或尝试访问不存在的对象。您的应用程序应该能够优雅地处理这些类型的错误。

API 版本　为了简化 API 版本管理，自 Zabbix 2.0.4 起，API 的版本与 Zabbix 本身的版本相匹配。您可以使用apiinfo.version 方法来查看您正在使用的 API 版本。这对于调整您的应用程序以使用特定版本的功能非常有用。

Zabbix 保证在同一主要版本内的向后兼容性。当在主要版本之间进行不兼容的更改时，Zabbix 通常会在下一个版本中将旧功能标记为已弃用，并在随后的版本中删除它们。偶尔，Zabbix 可能会在主要版本之间移除功能，而不提供任何向后兼容性。重要的是，您永远不要依赖任何已弃用的功能，并尽快迁移到较新的替代方案。

Note:
您可以在API 更新日志 中查看对 API 所做的所有更改。

延伸阅读　现在，您已经具备了开始使用 Zabbix API 的足够知识，但请不要止步于此。为了进一步阅读，建议您查看可用 API 列表。

方法参考

本节概述了 Zabbix API 提供的函数，并将帮助您找到可用的类和方法。

监控　Zabbix API 允许您访问在监控期间收集的历史记录和其他数据。

高可用集群

检索服务器节点及其状态的列表。

High availability cluster API

历史记录

检索 Zabbix 监控进程收集的历史值，以便进行演示或进一步处理。

History API

趋势

检索由 Zabbix Server 计算的趋势值以进行展示或进一步处理。

Trend API

事件

检索由触发器、网络发现和其他 Zabbix 系统生成的事件，以实现更灵活的情况管理或第三方工具集成。

Event API

问题

根据给定的参数检索问题。

Problem API

服务监控

Create a hierarchy representation of monitored IT infrastructure/business services data.

Service API

服务水平协议

定义服务级别目标（SLO），检索有关服务性能的详细服务级别指示器（SLI）信息。

SLA API

任务

与 Zabbix Server task manager 交互，创建任务并检索响应。

Task API

配置　Zabbix API 允许您管理监控系统的配置。

主机和主机组

管理主机组，主机及其相关的一切，包括主机接口，主机宏和维护期。

Host API | Host group API | Host interface API | User macro API | Value map API | Maintenance API

监控项

定义要监控的监控项。

Item API

触发器

配置触发器以通知您系统中的问题。管理触发器依赖关系。

Trigger API

图形

编辑图形或单独的图形项，以便更好地呈现收集的数据。

Graph API | Graph item API

模板

管理模板并将其链接到主机或其他模板。

Template API | Value map API

导入和导出

导出和导入 Zabbix 配置数据，用于配置备份，迁移或大规模配置更新。

Configuration API

低级别自动发现

配置低级发现规则以及项目，触发器和图形原型来监视动态实体。

LLD rule API | Item prototype API | Trigger prototype API | Graph prototype API | Host prototype API

事件关联性

创建自定义事件相关规则。

Correlation API

动作和警报

定义动作和报警，以通知用户某些事件或自动执行远程命令。获取有关生成的警报及其接收者的信息。

Action API | Alert API

服务

管理服务以进行服务级别监视，并检索有关任何服务的详细 SLA 信息。

Service API

仪表盘

管理仪表板并基于它们生成定时报表。

Dashboard API | Template dashboard API | Report API

拓扑图

配置拓扑图用于创建 IT 基础架构的详细动态展现。

Map API

Web 监控

配置 Web 场景以监控 Web 应用程序和服务。

Web scenario API

网络发现

管理网络级发现规则以自动查找和监控新主机。获得对所发现的服务和主机的信息的完全访问。

Discovery rule API | Discovery check API | Discovered host API | Discovered service API

管理　使用 Zabbix API，您可以更改监控系统的管理设置。

用户

添加将有权访问 Zabbix 的用户，将其分配到用户组并授予权限。创建角色以精细管理用户权限。跟踪每个用户已完成的配置更改。配置媒介类型和用户接收警报的多种方式。

User API | User group API | User role API | Media type API | Audit log API

通用

用于更改某些全局配置选项。

Autoregistration API | Icon map API | Image API | User macro API | Settings API | Housekeeping API

正则表达式

管理全局正则表达式。

Regular expression API

Proxies

用于管理分布式监控设置中使用的 proxies。

Proxy API

认证

更改身份认证配置选项。

Authentication API

API Tokens

管理认证 tokens.

Token API

脚本

配置和执行脚本以帮助您完成日常任务。

Script API

API 信息　检索 Zabbix API 的版本，以便应用程序可以使用特定于版本的功能。

API info API

API 信息

此类用于检索有关 API 的元信息。

可用方法：

- apiinfo.version - 检索 Zabbix API 版本

版本

描述

`string apiinfo.version(array)`

此方法允许检索 Zabbix API 的版本。

> **Attention:**
> 此方法仅适用于未经身份验证的用户且必须在发送 JSON-RPC 请求中不加 `auth` 参数的情况下调用。

参数

`(array)` 该方法接受一个空的数组。

返回值

`(string)` 返回 Zabbix API 的版本。

> **Note:**
> 从 Zabbix 2.0.4 版本开始，API 的版本与 Zabbix 的版本相匹配。

示例

检索 API 版本

检索 Zabbix API 版本。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "apiinfo.version",
    "params": [],
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": "4.0.0",
    "id": 1
}
```

来源

ui/include/classes/api/services/CAPIInfo.php 中的 CAPIInfo::version()。

LLD 规则

此类用于底层发现规则。

对象引用：

- LLD rule

可用方法：

- discoveryrule.copy - 复制 LLD 规则
- discoveryrule.create - 创建 LLD 规则
- discoveryrule.delete - 删除 LLD 规则
- discoveryrule.get - 获取 LLD 规则
- discoveryrule.update - 更新 LLD 规则

> LLD 规则对象

以下对象与 自动发现规则 API 直接相关。

LLD 规则

低级发现规则对象具有以下属性。

属性	类型	描述
itemid	string	*(只读)* LLD 规则的 ID。
delay (必需)	string	LLD 规则的更新间隔。接受秒或带有后缀的时间单位，以及一个或多个由灵活间隔和调度间隔组成的自定义间隔，这些间隔作为序列化的字符串。也接受用户宏。灵活间隔可以写成两个由正斜杠分隔的宏。间隔由分号分隔。 对于 Zabbix trapper、依赖项和具有 `mqtt.get` 键的 Zabbix agent（主动）是可选的。
hostid (必需)	string	LLD 规则所属的主机的 ID。
interfaceid (必需)	string	LLD 规则的主机接口的 ID。仅用于主机 LLD 规则。 对于 Zabbix agent（主动）、Zabbix 内部、Zabbix trapper、依赖项、数据 monitor 和脚本 LLD 规则不是必需的。对于 HTTP agent LLD 规则是可选的。
key_ (必需)	string	LLD 规则键。
name (必需)	string	LLD 规则的名称。

属性	类型	描述
type (必需)	integer	LLD 规则的类型。 可能的值： 0 - Zabbix agent； 2 - Zabbix trapper； 3 - 简单检查； 5 - Zabbix 内部； 7 - Zabbix agent（主动）； 10 - 外部检查； 11 - 数据库 monitor； 12 - IPMI agent ； 13 - SSH agent； 14 - TELNET agent； 16 - JMX agent ； 18 - 依赖项； 19 - HTTP agent； 20 - SNMP agent； 21 - 脚本。
url (必需)	string	URL 字符串，对于 HTTP agent LLD 规则是必需的。支持用户宏，{HOST.IP}，{HOST.CONN}，{HOST.DNS}，{HOST.HOST}，{HOST.NAME}，{ITEM.ID}，{ITEM.KEY}。
allow_traps	integer	HTTP agent LLD 规则字段。允许像在 trapper 监控项类型中一样填充值。 0 - *(默认)* 不允许接受传入数据。 1 - 允许接受传入数据。
authtype	integer	仅由 SSH agent 或 HTTP agent LLD 规则使用。 SSH agent 认证方法可能的值： 0 - *(默认)* 密码； 1 - 公钥。 HTTP agent 认证方法可能的值： 0 - *(默认)* 无 1 - 基本 2 - NTLM
description	string	LLD 规则的描述。
error	string	*(只读)* 如果更新 LLD 规则值存在问题，则错误文本。
follow_redirects	integer	HTTP agent LLD 规则字段。在轮询数据时跟随响应重定向。 0 - 不跟随重定向。 1 - *(默认)* 跟随重定向。
headers	object	HTTP agent LLD 规则字段。HTTP(S) 请求头的对象，其中头部名称用作键，头部值作为值。 示例： { "User-Agent": "Zabbix" }
http_proxy	string	HTTP agent LLD 规则字段。HTTP(S) agent 连接字符串。
ipmi_sensor	string	IPMI 传感器。仅由 IPMI LLD 规则使用。
jmx_endpoint	string	JMX agent 自定义连接字符串。 默认值： service:jmx:rmi:///jndi/rmi://{HOST.CONN}:{HOST.PORT}/jmxrmi
lifetime	string	不再发现的监控项将被删除的时间周期。接受秒、带有后缀的时间单位和用户宏。 默认值：30d。
master_itemid	integer	主项 ID。 允许递归最多 3 个依赖项，依赖项的最大数量等于 999。 发现规则不能是另一个发现规则的主项。 对于依赖项是必需的。

属性	类型	描述
output_format	integer	HTTP agent LLD 规则字段。是否应将响应转换为 JSON。 0 - *(默认)* 存储原始数据。 1 - 转换为 JSON。
params	string	根据 LLD 规则的类型而变化的附加参数： - 对于 SSH 和 Telnet LLD 规则执行的脚本； - 对于数据库 monitorLLD 规则的 SQL 查询； - 对于计算 LLD 规则的公式。
parameters	array	脚本类型 LLD 规则的附加参数。具有'name' 和'value' 属性的对象数组，其中名称必须是唯一的。
password	string	认证密码。由简单检查、SSH、Telnet、数据库 monitor、JMX 和 HTTP agent LLD 规则使用。
post_type	integer	HTTP agent LLD 规则字段。存储在 posts 属性中的 post 数据正文的类型。 0 - *(默认)* 原始数据。 2 - JSON 数据。 3 - XML 数据。
posts	string	HTTP agent LLD 规则字段。HTTP(S) 请求正文数据。与 post_type 一起使用。
privatekey	string	私钥文件的名称。
publickey	string	公钥文件的名称。
query_fields	array	HTTP agent LLD 规则字段。查询参数。具有'key':'value' 对的对象数组，其中值可以是空字符串。
request_method	integer	HTTP agent LLD 规则字段。请求方法的类型。 0 - *(默认)* GET 1 - POST 2 - PUT 3 - HEAD
retrieve_mode	integer	HTTP agent LLD 规则字段。应该存储响应的哪一部分。 0 - *(默认)* 正文。 1 - 头。 2 - 正文和头都将被存储。 对于 request_method HEAD，只允许值 1。
snmp_oid	string	SNMP OID。
ssl_cert_file	string	HTTP agent LLD 规则字段。公共 SSL 密钥文件路径。
ssl_key_file	string	HTTP agent LLD 规则字段。私有 SSL 密钥文件路径。
ssl_key_password	string	HTTP agent LLD 规则字段。SSL 密钥文件的密码。
state	integer	*(只读)* LLD 规则的状态。 可能的值： 0 - *(默认)* 正常； 1 - 不支持。
status	integer	LLD 规则的状态。 可能的值： 0 - *(默认)* 启用的 LLD 规则； 1 - 禁用的 LLD 规则。
status_codes	string	HTTP agent LLD 规则字段。所需的 HTTP 状态代码范围，用逗号分隔。还支持作为逗号分隔列表的一部分的用户宏。 示例：200,200-{$M},{$M},200-400
templateid	string	(只读) 父模板 LLD 规则的 ID。
timeout	string	监控项数据轮询请求的超时时间。用于 HTTP agent 和脚本 LLD 规则。支持用户宏。 默认值：3s 最大值：60s
trapper_hosts	string	允许的主机。由 trapper LLD 规则或 HTTP agent LLD 规则使用。

属性	类型	描述
username	string	认证用户名。由简单检查、SSH、Telnet、数据库 monitor、JMX 和 HTTP agent LLD 规则使用。 由 SSH 和 Telnet LLD 规则要求。
uuid	string	通用唯一标识符，用于将导入的 LLD 规则链接到已存在的规则。仅用于模板上的 LLD 规则。如果没有给出，则自动生成。
verify_host	integer	HTTP agent LLD 规则字段。是否验证连接的主机名是否与主机证书中的名称匹配。 0 - *(默认)* 不验证。 1 - 验证。
verify_peer	integer	HTTP agent LLD 规则字段。是否验证主机的证书是否真实。 0 - *(默认)* 不验证。 1 - 验证。

请注意，对于一些方法（更新、删除），必需/可选参数组合是不同的。

LLD 规则过滤器

LLD 规则过滤器对象定义了一组可以使用的条件过滤发现的对象。它具有以下属性：

属性	类型	描述
conditions (必需)	array	用于过滤结果的过滤条件集。
evaltype (必填)	integer	过滤条件评估方法。 可用值： 0 - 和/或； 1 - 和； 2 -或; 3 - 自定义表达式。
eval_formula	string	(只读）生成的表达式，将用于评估过滤条件。该表达式包含通过其 “公式” 引用特定过滤条件的 ID。`eval_formula` 的值等于具有自定义表达式的过滤器的 `formula` 的值。
formula	string	用户定义的表达式，用于使用自定义表达式评估过滤器的条件。表达式必须包含通过其 “公式” 引用特定过滤条件的 ID。表达式中使用的 ID 必须与过滤条件中定义的 ID 完全匹配：任何条件都不能保持未使用或省略。 自定义表达式过滤器必需。

LLD 规则过滤条件

LLD 规则过滤条件对象定义了对 LLD 宏的值执行的单独检查。它具有以下属性：

属性	类型	说明
macro (必需)	string	用于执行检查的 LLD 宏。
value (必需)	string	要比较的值。
formulaid	string	用于从自定义表达式引用条件的任意唯一 ID。只能包含大写字母。ID 必须由用户在修改过滤条件时定义，但在以后请求时会重新生成。
operator	integer	条件运算符。 可用值： 8 - （默认值）匹配正则表达式； 9 - 不匹配正则表达式； 12 - 存在； 13 - 不存在。

Note:
为了更好地了解如何使用各种过滤器表达式的类型，参见示例discoveryrule.get 和discoveryrule.create 方法页。

LLD 宏路径

LLD 宏路径具有以下属性:

属性	类型	描述
lld_macro （必需）	string	LLD 宏。
path （必需）	string	选择器将值分配给相应宏。

LLD 规则预处理

LLD 规则预处理对象具有以下属性。

属性	类型	描述
type (必需)	integer	预处理选项类型。 可能的值： 5 - 正则表达式匹配； 11 - XML XPath； 12 - JSONPath； 15 - 不匹配正则表达式； 16 - 检查 JSON 中的错误； 17 - 检查 XML 中的错误； 20 - 通过心跳丢弃不变数据； 21 - JavaScript； 23 - Prometheus 转 JSON； 24 - CSV 转 JSON； 25 - 替换； 27 - XML 转 JSON。
params (必需)	string	预处理选项使用的附加参数。多个参数用 LF (\n) 字符分隔。

属性	类型	描述
error_handler (必需)	integer	预处理步骤失败时使用的动作类型。 可能的值： 0 - 错误消息由 Zabbix server 设置； 1 - 丢弃值； 2 - 设置自定义值； 3 - 设置自定义错误消息。
error_handler_params (必需)	string	错误处理程序参数。与错误处理程序一起使用。 如果错误处理程序是 0 或 1，则必须为空。 如果错误处理程序是 2，可以为空。 如果错误处理程序是 3，则不能为空。

每种预处理类型支持以下参数和错误处理程序。

预处理类型	名称	参数 1	参数 2	参数 3	支持的错误处理程序
5	正则表达式	pattern[1]	output[2]		0, 1, 2, 3
11	XML XPath	path[3]			0, 1, 2, 3
12	JSONPath	path[3]			0, 1, 2, 3
15	不匹配正则表达式	pattern[1]			0, 1, 2, 3
16	检查 JSON 中的错误	path[3]			0, 1, 2, 3
17	检查 XML 中的错误	path[3]			0, 1, 2, 3
20	通过心跳丢弃不变数据	seconds[4, 5]			
21	JavaScript	script[2]			
23	Prometheus 转 JSON	pattern[5, 6]			0, 1, 2, 3
24	CSV 转 JSON	character[2]	character[2]	0,1	0, 1, 2, 3
25	替换	search string[2]	replacement[2]		
27	XML 转 JSON				0, 1, 2, 3

[1] 正则表达式
[2] 字符串
[3] JSONPath 或 XML XPath
[4] 正整数（支持时间后缀，例如 30s, 1m, 2h, 1d)
[5] 用户宏
[6] 遵循语法的 Prometheus 模式：`<metric name>{<label name>="<label value>", ...} == <value>`。每个 Prometheus 模式组件（指标、标签名称、标签值和指标值）都可以是用户宏。

LLD 规则覆盖

LLD 规则覆盖对象定义了一组规则（过滤器、条件和操作），用于覆盖不同原型对象的属性。它具有以下属性：

属性	类型	描述
name (必需)	string	唯一的覆盖名称。
step (必需)	integer	覆盖的唯一订单号。
stop	integer	如果匹配，则停止处理下一个覆盖。 可用值： 0 - （默认值）不停止处理覆盖； 1 - 如果过滤器匹配，则停止处理覆盖。
filter	object	覆盖过滤器。
operations	array	覆盖操作。

LLD 规则覆盖过滤器

LLD 规则覆盖过滤器对象定义了一组条件，如果它们与发现的对象匹配，则应用覆盖。它具有以下属性：

属性	类型	描述
evaltype (必需)	integer	覆盖过滤条件评估方法。 可能的值： 0 - 和/或； 1 - 和； 2 - 或; 3 - 自定义表达式。
conditions (必需)	array	用于匹配已发现对象的覆盖过滤条件集。
eval_formula	string	(只读）生成的表达式，将用于评估覆盖过滤条件。该表达式包含通过其 “公式” 引用特定覆盖过滤条件的 ID。`eval_formula` 的值等于具有自定义表达式的过滤器的 `formula` 的值。
formula	string	用户定义的表达式，用于评估具有自定义表达式的覆盖过滤器的条件。表达式必须包含通过其 “公式” 引用特定覆盖过滤条件的 ID。表达式中使用的 ID 必须与覆盖过滤条件中定义的 ID 完全匹配：任何条件都不能保持未使用或省略。 自定义表达式覆盖过滤器是必需的。

LLD 规则覆盖过滤条件

LLD 规则覆盖过滤条件对象定义了对 LLD 宏的值执行的单独检查。它具有以下属性：

属性	类型	描述
macro (必需)	string	用于执行检查的 LLD 宏。
value (必需)	string	要比较的值。
formulaid	string	用于从自定义表达式引用条件的任意唯一 ID。只能包含大写字母。ID 必须由用户在修改过滤条件时定义，但在以后请求时会重新生成。

属性	类型	描述
operator	integer	条件运算符。 可用值： 8 - （默认值）匹配正则表达式； 9 - 不匹配正则表达式； 12 - 存在；< br>13 - 不存在。

LLD 规则覆盖操作

LLD 规则覆盖操作是对原型对象执行的条件和操作的组合。它具有以下属性：

属性	类型	描述
operationobject (必需)	integer	发现的执行操作的对象的类型。 可用值： 0 - 项目原型； 1 - 触发器原型； 2 - 图形原型； 3 - 主机原型。
operator	integer	覆盖条件运算符。 可能的值： 0 - （默认值）等于； 1 - 不等于； 2 - 包含； 3 - 不包含； 8 - 匹配； 9 - 不匹配。
value	string	根据所选对象匹配项、触发器、图形或主机原型名称的模式。
opstatus	object	覆盖项目、触发器和宿主原型对象的操作状态对象。
opdiscover	object	覆盖操作发现状态对象（所有对象类型）。
opperiod	object	覆盖项目原型对象的操作周期（更新间隔）对象。
ophistory	object	覆盖项目原型对象的操作历史对象。
optrends	object	覆盖项目原型对象的操作趋势对象。
opseverity	object	覆盖触发器原型对象的操作严重性对象。
optag	array	覆盖触发器和宿主原型对象的操作标记对象。
optemplate	array	覆盖宿主原型对象的操作模板对象。
opinventory	object	覆盖主机原型对象的操作清单对象。

LLD 规则覆盖操作状态

LLD 规则覆盖设置为已发现对象的操作状态。它具有以下属性：

属性	类型	描述
status （必需）	integer	覆盖选定对象的状态。 可用值： 0 - 启用创建； 1 - 禁止创建。

LLD 规则覆盖操作发现

设置为已发现对象的 LLD 规则覆盖操作发现状态。它具有以下属性：

属性	类型	描述
discover （必需）	integer	覆盖选定对象的发现状态。 可用值： 0 - 是，持续发现对象； 1 - 否，不会发现新对象，现有对象将被标记为丢失。

LLD 规则覆盖操作周期

LLD 规则覆盖操作周期是设置为已发现项目的更新间隔值（支持自定义间隔）。它具有以下属性：

属性	类型	描述
delay （必需）	string	覆盖项目原型的更新间隔。接受秒或带后缀 (30s，1m，2h，1d) 的时间单位以及灵活的调度间隔和用户宏或 LLD 宏。多个区间用分号隔开。

LLD 规则覆盖操作历史记录

LLD 规则覆盖设置为已发现监控项的操作历史记录值。它具有以下属性：

属性	类型	描述
history （必需）	string	覆盖监控项原型的历史，这是历史数据应存储多长时间的时间单位。也接受用户宏和 LLD 宏。

LLD 规则覆盖操作趋势

LLD 规则覆盖设置为已发现监控项的操作趋势值。它具有以下属性：

属性	类型	描述
trends （必需）	string	覆盖监控项原型的趋势，这是趋势数据应存储多长时间的时间单位。也接受用户宏和 LLD 宏。

LLD 规则覆盖操作严重性

LLD 规则覆盖设置为发现触发器的操作严重性值。它具有以下属性：

属性	类型	描述
severity （必需）	integer	覆盖触发器原型的严重性。 可用值：0 - （默认值）未分类； 1 - 信息； 2 - 警告； 3 - 一般严重； 4 - 比较严重； 5 - 灾难。

LLD 规则覆盖操作标记

LLD 规则覆盖操作标记对象包含设置为发现对象的标记名称和值。它具有以下属性：

属性	类型	描述
tag （必需）	string	标签名字。
value	string	标签值。

LLD 规则覆盖操作模板

LLD 规则覆盖链接到已发现主机的操作模板对象。它具有以下属性：

属性	类型	描述
templateid （必需）	string	覆盖主机原型链接模板的模板。

LLD 规则覆盖操作清单

LLD 规则覆盖设置为已发现主机的操作库存模式值。它具有以下属性：

属性	类型	描述
inventory_mode （必需）	integer	覆盖主机原型库存模式。 可用值： -1 - 禁用； 0 - （默认值）手动； 1 - 自动。

创建

描述

`object discoveryrule.create(object/array lldRules)`

此方法允许创建新的 LLD 规则。

> **Note:**
> 此方法只有 *Admin(管理员)* 和 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

`(object/array)` 创建的 LLD 规则。

除了标准 LLD 规则属性，该方法还接受以下参数。

参数	类型	描述
filter	object	LLD 规则过滤。
preprocessing	array	LLD 规则预处理选项。
lld_macro_paths	array	LLD 规则lld_macro_path选项。
overrides	array	LLD 规则覆盖选项。

返回值

（object）返回一个对象，其中包含在 itemids 属性下创建的 LLD 规则的 ID。返回 ID 的顺序与传递的 LLD 规则的顺序相匹配。

示例

创建 LLD 规则

创建 Zabbix agent LLD 规则以发现挂载的文件系统。发现的监控项将每 30 秒更新一次。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "discoveryrule.create",
    "params": {
        "name": "Mounted filesystem discovery",
        "key_": "vfs.fs.discovery",
        "hostid": "10197",
        "type": "0",
        "interfaceid": "112",
        "delay": "30s"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "27665"
        ]
    },
    "id": 1
}
```

使用过滤器

使用一组条件创建 LLD 规则以过滤结果。条件将使用逻辑 “和” 运算符组合在一起。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "discoveryrule.create",
    "params": {
        "name": "Filtered LLD rule",
        "key_": "lld",
        "hostid": "10116",
        "type": "0",
        "interfaceid": "13",
        "delay": "30s",
        "filter": {
            "evaltype": 1,
            "conditions": [
                {
                    "macro": "{#MACRO1}",
                    "value": "@regex1"
```

```
                },
                {
                    "macro": "{#MACRO2}",
                    "value": "@regex2",
                    "operator": "9"
                },
                {
                    "macro": "{#MACRO3}",
                    "value": "",
                    "operator": "12"
                },
                {
                    "macro": "{#MACRO4}",
                    "value": "",
                    "operator": "13"
                }
            ]
        }
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "27665"
        ]
    },
    "id": 1
}
```

使用宏路径创建 LLD 规则

请求：

```
{
    "jsonrpc": "2.0",
    "method": "discoveryrule.create",
    "params": {
        "name": "LLD rule with LLD macro paths",
        "key_": "lld",
        "hostid": "10116",
        "type": "0",
        "interfaceid": "13",
        "delay": "30s",
        "lld_macro_paths": [
            {
                "lld_macro": "{#MACRO1}",
                "path": "$.path.1"
            },
            {
                "lld_macro": "{#MACRO2}",
                "path": "$.path.2"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "27665"
        ]
    },
    "id": 1
}
```

使用自定义表达式过滤器

使用将使用自定义表达式评估条件的过滤器创建 LLD 规则。LLD 规则必须只发现 “{#MACRO1}” 宏值同时匹配正则表达式 “regex1” 和 “regex2” 且 “{#MACRO2}” 值匹配 “regex3” 或 “regex4” 的对象”。公式 ID“A”、“B”、“C” 和 “D” 是任意选择的。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "discoveryrule.create",
    "params": {
        "name": "Filtered LLD rule",
        "key_": "lld",
        "hostid": "10116",
        "type": "0",
        "interfaceid": "13",
        "delay": "30s",
        "filter": {
            "evaltype": 3,
            "formula": "(A and B) and (C or D)",
            "conditions": [
                {
                    "macro": "{#MACRO1}",
                    "value": "@regex1",
                    "formulaid": "A"
                },
                {
                    "macro": "{#MACRO1}",
                    "value": "@regex2",
                    "formulaid": "B"
                },
                {
                    "macro": "{#MACRO2}",
                    "value": "@regex3",
                    "formulaid": "C"
                },
                {
                    "macro": "{#MACRO2}",
                    "value": "@regex4",
                    "formulaid": "D"
                }
            ]
        }
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "27665"
```

```
        ]
    },
    "id": 1
}
```

使用自定义查询字段和标题

使用自定义查询字段和标题创建 LLD 规则。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "discoveryrule.create",
    "params": {
        "hostid": "10257",
        "interfaceid": "5",
        "type": "19",
        "name": "API HTTP agent",
        "key_": "api_discovery_rule",
        "value_type": "3",
        "delay": "5s",
        "url": "http://127.0.0.1?discoverer.php",
        "query_fields": [
            {
                "mode": "json"
            },
            {
                "elements":"2"
            }
        ],
        "headers": {
            "X-Type": "api",
            "Authorization": "Bearer mF_A.B5f-2.1JcM"
        },
        "allow_traps": "1",
        "trapper_hosts": "127.0.0.1",
        "id": 35,
        "auth": "d678e0b85688ce578ff061bd29a20d3b",
    }
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "28336"
        ]
    },
    "id": 35
}
```

使用 LLD 规则创建预处理

请求：

```
{
    "jsonrpc": "2.0",
    "method": "discoveryrule.create",
    "params": {
        "name": "Discovery rule with preprocessing",
        "key_": "lld.with.preprocessing",
        "hostid": "10001",
        "ruleid": "27665",
```

```
        "type": 0,
        "value_type": 3,
        "delay": "60s",
        "interfaceid": "1155",
        "preprocessing": [
            {
                "type": "20",
                "params": "20",
                "error_handler": "0",
                "error_handler_params": ""
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "44211"
        ]
    },
    "id": 1
}
```

创建具有覆盖的 LLD 规则

请求：

```
{
    "jsonrpc": "2.0",
    "method": "discoveryrule.create",
    "params": {
        "name": "Discover database host",
        "key_": "lld.with.overrides",
        "hostid": "10001",
        "type": 0,
        "value_type": 3,
        "delay": "60s",
        "interfaceid": "1155",
        "overrides": [
            {
                "name": "Discover MySQL host",
                "step": "1",
                "stop": "1",
                "filter": {
                    "evaltype": "2",
                    "conditions": [
                        {
                            "macro": "{#UNIT.NAME}",
                            "operator": "8",
                            "value": "^mysqld\\.service$"
                        },
                        {
                            "macro": "{#UNIT.NAME}",
                            "operator": "8",
                            "value": "^mariadb\\.service$"
                        }
                    ]
                },
                "operations": [
```

```
                    {
                        "operationobject": "3",
                        "operator": "2",
                        "value": "Database host",
                        "opstatus": {
                            "status": "0"
                        },
                        "optemplate": [
                            {
                                "templateid": "10170"
                            }
                        ],
                        "optag": [
                            {
                                "tag": "Database",
                                "value": "MySQL"
                            }
                        ]
                    }
                ]
            },
            {
                "name": "Discover PostgreSQL host",
                "step": "2",
                "stop": "1",
                "filter": {
                    "evaltype": "0",
                    "conditions": [
                        {
                            "macro": "{#UNIT.NAME}",
                            "operator": "8",
                            "value": "^postgresql\\.service$"
                        }
                    ]
                },
                "operations": [
                    {
                        "operationobject": "3",
                        "operator": "2",
                        "value": "Database host",
                        "opstatus": {
                            "status": "0"
                        },
                        "optemplate": [
                            {
                                "templateid": "10263"
                            }
                        ],
                        "optag": [
                            {
                                "tag": "Database",
                                "value": "PostgreSQL"
                            }
                        ]
                    }
                ]
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "30980"
        ]
    },
    "id": 1
}
```

创建脚本 LLD 规则

使用脚本 LLD 规则创建一个简单的数据集合。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "discoveryrule.create",
    "params": {
        "name": "Script example",
        "key_": "custom.script.lldrule",
        "hostid": "12345",
        "type": 21,
        "value_type": 4,
        "params": "var request = new CurlHttpRequest();\nreturn request.Post(\"https://postman-echo.com/po
        "parameters": [{
            "name": "host",
            "value": "{HOST.CONN}"
        }],
        "timeout": "6s",
        "delay": "30s"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 2
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "23865"
        ]
    },
    "id": 3
}
```

参见

- LLD 规则过滤
- LLD 宏路径
- LLD 规则预处理

来源

ui/include/classes/api/services/CDiscoveryRule.php 中的 CDiscoveryRule::create()。

删除

描述

`object discoveryrule.delete(array lldRuleIds)`

此方法允许删除 LLD 规则。

Note:
此方法只有 *Admin(管理员)* 和 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(array) 要删除的 LLD 规则的 ID。

返回值

(object) 返回一个对象，其中包含在 ruleids 属性下删除的 LLD 规则的 IDs。

示例

删除多条 LLD 规则

删除两条 LLD 规则。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "discoveryrule.delete",
    "params": [
        "27665",
        "27668"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "ruleids": [
            "27665",
            "27668"
        ]
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CDiscoveryRule.php 中的 CDiscoveryRule::delete()。

复制

描述

`object discoveryrule.copy(object parameters)`

此方法允许将带有所有原型的 LLD 规则复制到给定主机。

Note:
此方法只有 *Admin(管理员)* 和 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(object) 定义要复制的 LLD 规则和目标主机的参数。

参数	类型	描述
discoveryids	array	要复制的 LLD 规则的 ID。
hostids	array	将 LLD 规则复制到的主机的 ID。

返回值

(boolean) 如果复制成功，则返回 true。

示例

LLD 规则复制到多个主机

将 LLD 规则复制到两台主机。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "discoveryrule.copy",
    "params": {
        "discoveryids": [
            "27426"
        ],
        "hostids": [
            "10196",
            "10197"
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": true,
    "id": 1
}
```

源码

CDiscoveryRule::copy() 在 *ui/include/classes/api/services/CDiscoveryRule.php* 文件中。

更新

描述

`object discoveryrule.update(object/array lldRules)`

此方法更新已存在的 LLD 规则。

Note:
此方法只有 *Admin(管理员)* 和 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(object/array) 要更新的 LLD 规则属性。

必须为每个 LLD 规则定义 itemid 属性，所有其他属性都是可选的。只有传递的属性会被更新，其他属性将保持不变。

除了标准 LLD 规则属性，该方法还接受以下参数。

参数	类型	描述
filter	object	替换当前过滤的 LLD 规则过滤。
preprocessing	array	替换现有预处理选项的 LLD 规则预处理选项。
lld_macro_paths	array	替换现有 lld_macro_path 选项的 LLD 规则lld_macro_path选项。
overrides	array	替换现有覆盖选项的 LLD 规则覆盖选项。

返回值

(object) 返回一个对象，其中包含 itemids 属性下更新的 LLD 规则的 ID。

示例

LLD 规则添加过滤器

添加一个过滤器，以便 *{#FSTYPE}* 宏的内容与 *@File systems for discovery* 正则表达式匹配。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "discoveryrule.update",
    "params": {
        "itemid": "22450",
        "filter": {
            "evaltype": 1,
            "conditions": [
                {
                    "macro": "{#FSTYPE}",
                    "value": "@File systems for discovery"
                }
            ]
        }
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "22450"
        ]
    },
    "id": 1
}
```

添加 LLD 宏路径

请求：

```
{
    "jsonrpc": "2.0",
    "method": "discoveryrule.update",
    "params": {
        "itemid": "22450",
        "lld_macro_paths": [
            {
                "lld_macro": "{#MACRO1}",
                "path": "$.json.path"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "22450"
        ]
    },
```

```
    "id": 1
}
```

禁止 trapping

禁止自动发现规则 trapping。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "discoveryrule.update",
    "params": {
        "itemid": "28336",
        "allow_traps": "0"
    },
    "id": 36,
    "auth": "d678e0b85688ce578ff061bd29a20d3b"
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "28336"
        ]
    },
    "id": 36
}
```

更新 LLD 规则预处理选项

使用预处理规则 “JSONPath” 更新 LLD 规则。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "discoveryrule.update",
    "params": {
        "itemid": "44211",
        "preprocessing": [
            {
                "type": "12",
                "params": "$.path.to.json",
                "error_handler": "2",
                "error_handler_params": "5"
            }
        ]
    },
    "auth": "700ca65537074ec963db7efabda78259",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "44211"
        ]
    },
    "id": 1
}
```

更新 LLD 规则脚本

使用不同的脚本更新 LLD 规则脚本，并删除之前脚本使用的不必要参数。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "discoveryrule.update",
    "params": {
        "itemid": "23865",
        "parameters": [],
        "script": "Zabbix.Log(3, 'Log test');\nreturn 1;"
    },
    "auth": "700ca65537074ec963db7efabda78259",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "23865"
        ]
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CDiscoveryRule.php 中的 CDiscoveryRule::update()。

获取

描述

`integer/array discoveryrule.get(object parameters)`

该方法允许根据给定的参数检索 LLD 规则。

> **Note:**
> 此方法对于任何用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

`(object)` 定义所需输出的参数。

该方法支持以下参数。

参数	类型	描述
itemids	string/array	只返回给定 ID 的 LLD 规则。
groupids	string/array	只返回属于给定组的主机的 LLD 规则。
hostids	string/array	只返回属于给定主机的 LLD 规则。
inherited	boolean	如果设置为 `true`，则只返回从模板继承的 LLD 规则。
interfaceids	string/array	只返回使用给定主机接口的 LLD 规则。
monitored	boolean	如果设置为 `true`，则只返回属于被监控主机的启用的 LLD 规则。
templated	boolean	如果设置为 `true`，则只返回属于模板的 LLD 规则。
templateids	string/array	只返回属于给定模板的 LLD 规则。
selectFilter	query	返回使用 LLD 规则的过滤的数据的filter属性。
selectGraphs	query	返回属于 LLD 规则的图表原型的graphs属性。 支持 `count`。
selectHostPrototypes	query	返回属于 LLD 规则的主机原型的hostPrototypes属性。 支持 `count`。

参数	类型	描述
selectHosts	query	返回一个hosts属性，包含 LLD 规则所属的主机数组。
selectItems	query	返回属于 LLD 规则的监控项原型的item属性。 支持 `count`。
selectTriggers	query	返回属于 LLD 规则的触发器原型的triggers属性。 支持 `count`。
selectLLDMacroPaths	query	返回一个lld_macro_paths属性，包含分配给每个对应宏的 LLD 宏和值路径的列表。
selectPreprocessing	query	返回一个preprocessing属性，包含 LLD 规则预处理选项。
selectOverrides	query	返回一个lld_rule_overrides属性，包含在原型对象上执行的覆盖过滤、条件和操作的列表。
filter	object	只返回完全匹配给定过滤的结果。 接受一个数组，其中键是属性名称，值可以是单个值或值数组。 支持额外的过滤： `host` - LLD 规则所属主机的技术名称。
limitSelects	integer	限制子选择返回的记录数。 适用于以下子选择： `selectItems`； `selectGraphs`； `selectTriggers`。
sortfield	string/array	根据给定属性对结果进行排序。 可能的值有：`itemid`, `name`, `key_`, `delay`, `type` 和 `status`。
countOutput	boolean	这些参数对所有 `get` 方法都是通用的，详细描述在参考注释中。
editable	boolean	
excludeSearch	boolean	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

`(integer/array)` 返回任一值:

- 对象数组；
- 检索到的对象的计数，如果使用了 `countOutput` 参数。

示例

检索主机自动发现规则

返回主机"10202" 所有自动发现规则。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "discoveryrule.get",
    "params": {
        "output": "extend",
        "hostids": "10202"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：


```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "itemid": "27425",
            "type": "0",
            "snmp_oid": "",
            "hostid": "10202",
            "name": "Network interface discovery",
            "key_": "net.if.discovery",
            "delay": "1h",
            "state": "0",
            "status": "0",
            "trapper_hosts": "",
            "error": "",
            "templateid": "22444",
            "params": "",
            "ipmi_sensor": "",
            "authtype": "0",
            "username": "",
            "password": "",
            "publickey": "",
            "privatekey": "",
            "interfaceid": "119",
            "description": "Discovery of network interfaces as defined in global regular expression \"Netw
            "lifetime": "30d",
            "jmx_endpoint": "",
            "master_itemid": "0",
            "timeout": "3s",
            "url": "",
            "query_fields": [],
            "posts": "",
            "status_codes": "200",
            "follow_redirects": "1",
            "post_type": "0",
            "http_proxy": "",
            "headers": [],
            "retrieve_mode": "0",
            "request_method": "0",
            "ssl_cert_file": "",
            "ssl_key_file": "",
            "ssl_key_password": "",
            "verify_peer": "0",
            "verify_host": "0",
            "allow_traps": "0",
            "parameters": []
        },
        {
            "itemid": "27426",
            "type": "0",
            "snmp_oid": "",
            "hostid": "10202",
            "name": "Mounted filesystem discovery",
            "key_": "vfs.fs.discovery",
            "delay": "1h",
            "state": "0",
            "status": "0",
            "trapper_hosts": "",
            "error": "",
            "templateid": "22450",
            "params": "",
```

```
            "ipmi_sensor": "",
            "authtype": "0",
            "username": "",
            "password": "",
            "publickey": "",
            "privatekey": "",
            "interfaceid": "119",
            "description": "Discovery of file systems of different types as defined in global regular expr
            "lifetime": "30d",
            "jmx_endpoint": "",
            "master_itemid": "0",
            "timeout": "3s",
            "url": "",
            "query_fields": [],
            "posts": "",
            "status_codes": "200",
            "follow_redirects": "1",
            "post_type": "0",
            "http_proxy": "",
            "headers": [],
            "retrieve_mode": "0",
            "request_method": "0",
            "ssl_cert_file": "",
            "ssl_key_file": "",
            "ssl_key_password": "",
            "verify_peer": "0",
            "verify_host": "0",
            "allow_traps": "0",
            "parameters": []
        }
    ],
    "id": 1
}
```


检索过滤条件

检索 LLD 规则的名称 “24681” 及其过滤条件。过滤器使用 “and” 求值类型，因此 `formula` 属性为空，并自动生成 `eval_formula`。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "discoveryrule.get",
    "params": {
        "output": [
            "name"
        ],
        "selectFilter": "extend",
        "itemids": ["24681"]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：


```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "itemid": "24681",
            "name": "Filtered LLD rule",
            "filter": {
                "evaltype": "1",
                "formula": "",
```

```
                "conditions": [
                    {
                        "macro": "{#MACRO1}",
                        "value": "@regex1",
                        "operator": "8",
                        "formulaid": "A"
                    },
                    {
                        "macro": "{#MACRO2}",
                        "value": "@regex2",
                        "operator": "9",
                        "formulaid": "B"
                    },
                    {
                        "macro": "{#MACRO3}",
                        "value": "",
                        "operator": "12",
                        "formulaid": "C"
                    },
                    {
                        "macro": "{#MACRO4}",
                        "value": "",
                        "operator": "13",
                        "formulaid": "D"
                    }
                ],
                "eval_formula": "A and B and C and D"
            }
        }
    ],
    "id": 1
}
```

通过 URL 检索 LLD 规则

通过规则 URL 字段值检索主机的 LLD 规则。仅支持为 LLD 规则定义的 URL 字符串的完全匹配。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "discoveryrule.get",
    "params": {
        "hostids": "10257",
        "filter": {
            "type": "19",
            "url": "http://127.0.0.1/discoverer.php"
        }
    },
    "id": 39,
    "auth": "d678e0b85688ce578ff061bd29a20d3b"
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "itemid": "28336",
            "type": "19",
            "snmp_oid": "",
            "hostid": "10257",
            "name": "API HTTP agent",
            "key_": "api_discovery_rule",
```

```
            "delay": "5s",
            "history": "90d",
            "trends": "0",
            "status": "0",
            "value_type": "4",
            "trapper_hosts": "",
            "units": "",
            "error": "",
            "logtimefmt": "",
            "templateid": "0",
            "valuemapid": "0",
            "params": "",
            "ipmi_sensor": "",
            "authtype": "0",
            "username": "",
            "password": "",
            "publickey": "",
            "privatekey": "",
            "flags": "1",
            "interfaceid": "5",
            "description": "",
            "inventory_link": "0",
            "lifetime": "30d",
            "state": "0",
            "jmx_endpoint": "",
            "master_itemid": "0",
            "timeout": "3s",
            "url": "http://127.0.0.1/discoverer.php",
            "query_fields": [
                {
                    "mode": "json"
                },
                {
                    "elements": "2"
                }
            ],
            "posts": "",
            "status_codes": "200",
            "follow_redirects": "1",
            "post_type": "0",
            "http_proxy": "",
            "headers": {
                "X-Type": "api",
                "Authorization": "Bearer mF_A.B5f-2.1JcM"
            },
            "retrieve_mode": "0",
            "request_method": "1",
            "ssl_cert_file": "",
            "ssl_key_file": "",
            "ssl_key_password": "",
            "verify_peer": "0",
            "verify_host": "0",
            "allow_traps": "0",
            "parameters": []
        }
    ],
    "id": 39
}
```


使用覆盖检索 LLD 规则

检索具有各种覆盖设置的 LLD 规则。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "discoveryrule.get",
    "params": {
        "output": ["name"],
        "itemids": "30980",
        "selectOverrides": ["name", "step", "stop", "filter", "operations"]
    },
    "id": 39,
    "auth": "d678e0b85688ce578ff061bd29a20d3b"
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "name": "Discover database host"
            "overrides": [
                {
                    "name": "Discover MySQL host",
                    "step": "1",
                    "stop": "1",
                    "filter": {
                        "evaltype": "2",
                        "formula": "",
                        "conditions": [
                            {
                                "macro": "{#UNIT.NAME}",
                                "operator": "8",
                                "value": "^mysqld\\.service$"
                                "formulaid": "A"
                            },
                            {
                                "macro": "{#UNIT.NAME}",
                                "operator": "8",
                                "value": "^mariadb\\.service$"
                                "formulaid": "B"
                            }
                        ],
                        "eval_formula": "A or B"
                    },
                    "operations": [
                        {
                            "operationobject": "3",
                            "operator": "2",
                            "value": "Database host",
                            "opstatus": {
                                "status": "0"
                            },
                            "optag": [
                                {
                                    "tag": "Database",
                                    "value": "MySQL"
                                }
                            ],
                            "optemplate": [
                                {
                                    "templateid": "10170"
                                }
                            ]
                        }
```

```
                    ]
                },
                {
                    "name": "Discover PostgreSQL host",
                    "step": "2",
                    "stop": "1",
                    "filter": {
                        "evaltype": "0",
                        "formula": "",
                        "conditions": [
                            {
                                "macro": "{#UNIT.NAME}",
                                "operator": "8",
                                "value": "^postgresql\\.service$"
                                "formulaid": "A"
                            }
                        ],
                        "eval_formula": "A"
                    },
                    "operations": [
                        {
                            "operationobject": "3",
                            "operator": "2",
                            "value": "Database host",
                            "opstatus": {
                                "status": "0"
                            },
                            "optag": [
                                {
                                    "tag": "Database",
                                    "value": "PostgreSQL"
                                }
                            ],
                            "optemplate": [
                                {
                                    "templateid": "10263"
                                }
                            ]
                        }
                    ]
                }
            ]
        }
    ],
    "id": 39
}
```


参见

- 图形原型
- 主机
- 监控项原型
- LLD 规则过滤
- 触发器原型

来源

ui/include/classes/api/services/CDiscoveryRule.php 中的 CDiscoveryRule::get()。

Proxy

此类被设计用来处理 Proxy。

对象引用：

- Proxy
- Proxy interface

可用方法：

- proxy.create - 创建新 proxy
- proxy.delete - 删除 proxy
- proxy.get - 获取 proxy
- proxy.update - 更新 proxy

> Proxy 对象

以下对象与 `proxy` API 直接相关。

Proxy

proxy 对象具有以下属性。

属性	类型	描述
proxyid	string	(只读) proxy ID。
主机 (必需)	string	proxy 名称。
状态 (必需)	integer	proxy 类型. 可能的值： 5 - 主动 proxy； 6 - 被动 proxy。
description	text	proxy 的描述。
lastaccess	timestamp	(只读) proxy 上次连接到服务器的时间。
tls_connect	integer	链接到的主机。 可用值为： 1 - （默认）不加密； 2 - PSK； 4 - 证书。
tls_accept	integer	来自主机的连接。 可用位图值： 1 - （默认）无加密； 2 - PSK； 4 - 证书。
tls_issuer	string	证书颁发者。
tls_subject	string	证书主题。
tls_psk_identity	string	(只写) PSK 身份。如果 `tls_connect` 或 `tls_accept` 启用了 PSK，则为必需。不要将敏感信息放入 PSK 标识中，它会通过网络以未加密方式传输，以通知接收者使用哪个 PSK。

属性	类型	描述
tls_psk	string	(只写) 预共享密钥，至少 32 个十六进制数字。如果 `tls_connect` 或 `tls_accept` 启用了 PSK，则为必需。
proxy_address	string	逗号分隔的活动 Zabbix proxy 的 IP 地址或 DNS 名称。
auto_compress	integer	(只读) 表示 Zabbix server 和 proxy 之间的通信是否被压缩。 可能的值为： 0 - 不压缩； 1 - 启用压缩。

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

proxy 接口

proxy 接口对象定义用于连接被动 proxy 的接口。它具有以下属性。

属性	类型	描述
dns (必需)	string	要连接到的 DNS 名称。 如果通过 IP 地址建立连接，则可以为空。
ip (必需)	string	要连接的 IP 地址。 如果通过 DNS 名称进行连接，则可以为空。
端口 (必需)	string	要连接的端口号。
useip (必需)	integer	是否应通过 IP 地址进行连接。 可用值： 0 - 使用 DNS 名称连接； 1 -使用 IP 地址连接。

创建

描述

`object proxy.create(object/array proxies)`

此方法允许创建新的 proxy。

> **Note:**
> 此方法对于任何用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(object/array) 创建 proxy。

除了标准 proxy 属性，该方法还接受以下参数。

参数	类型	描述
hosts	array	由 proxy 监视的主机。如果一个主机已经被另一个代理监视，那么它将被重新分配给当前 proxy。 此主机必须拥有 hostid 属性。
interface	object	创建主机接口用于被动 proxy。 被动 proxy 是必需的。

返回值

(object) 返回一个对象，该对象包含在 proxyids 属性下创建的 proxy 的 ids，返回的 Ids 的顺序与所传递的代理的顺序相匹配。

示例

创建一个主动代理

创建主动代理"Active proxy"，并分配要监控的主机。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "proxy.create",
    "params": {
        "host": "Active proxy",
        "status": "5",
        "hosts": [
            {
                "hostid": "10279"
            }
        ]
    },
    "auth": "ab9638041ec6922cb14b07982b268f47",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "proxyids": [
            "10280"
        ]
    },
    "id": 1
}
```

创建一个被动代理

创建一个被动代理 "Passive proxy"，并分配两台要监控的主机。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "proxy.create",
    "params": {
        "host": "Passive proxy",
        "status": "6",
        "interface": {
            "ip": "127.0.0.1",
            "dns": "",
            "useip": "1",
            "port": "10051"
        },
        "hosts": [
            {
                "hostid": "10192"
            },
            {
                "hostid": "10139"
            }
        ]
    },
    "auth": "ab9638041ec6922cb14b07982b268f47",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "proxyids": [
            "10284"
        ]
    },
    "id": 1
}
```

参见

- 主机
- 代理接口

来源

ui/include/classes/api/services/CProxy.php 中的 CProxy::create()。

删除

描述

`object proxy.delete(array proxies)`

此方法允许删除 proxy。

> **Note:**
> 此方法对于任何用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(array) 要删除的 proxy IDs。

返回值

(object) 返回一个对象，其中包含 proxyids 属性下已删除 proxy 的 ID。

示例

删除多个 proxy

删除两个 proxy。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "proxy.delete",
    "params": [
        "10286",
        "10285"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "proxyids": [
            "10286",
            "10285"
        ]
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CProxy.php 中的 CProxy::delete()。

更新

描述

·object proxy.delete(array proxies)·

此方法允许更新现有的 proxy。

> **Note:**
> 此方法对于任何用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(object/array) 待更新的 proxy 属性。

每个 proxy 必须定义 proxyid 属性，其他的所有属性都是可选的。只有传递的属性将被更新，其他的所有属性将保持不变。

除了标准 proxy 属性，该方法还接受以下参数。

参数	类型	描述
hosts	array	proxy 监视的主机。 如果主机已被其他 proxy 监控，则会将其重新分配给当前 proxy。 主机必须定义 hostid 属性。
interface	object	主机接口替换被动 proxy 的现有接口。

返回值

(object) 返回一个对象，该对象包含 proxyids 属性下更新的 proxy 的 ID。

示例

更改 proxy 监控的主机

更新 proxy 以监视两个给定主机。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "proxy.update",
    "params": {
        "proxyid": "10293",
        "hosts": [
            "10294",
            "10295"
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "proxyids": [
            "10293"
        ]
    },
    "id": 1
}
```

更改 proxy 状态

将代理更改为主动 proxy，并重命名为 “Active proxy”。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "proxy.update",
    "params": {
        "proxyid": "10293",
        "host": "Active proxy",
        "status": "5"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "proxyids": [
            "10293"
        ]
    },
    "id": 1
}
```

参见

- 主机
- proxy 接口

来源

CProxy::update() in *ui/include/classes/api/services/CProxy.php*.

获取

描述

`integer/array proxy.get(object parameters)`

该方法允许根据给定的参数检索 proxy。

Note:
此方法对于任何用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

`(object)` 定义所需输出的参数。

此方法支持以下参数。

参数	类型	描述
proxyids	string/array	仅返回具有给定 ID 的 proxy。
selectHosts	query	返回一个主机属性，其中包含 proxy 监控的主机。
selectInterface	query	返回一个接口属性，其中包含被被动 proxy 使用的 proxy 接口。
sortfield	string/array	按照给定的属性对结果进行排序。 可用值：`hostid`，`host` 和 `status`。
countOutput	boolean	这些参数对所有 `get` 方法是通用的，详细描述请查看参考说明。
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

`(integer/array)` 返回其中一个：

- 一个对象数组；
- 如果使用了 `countOutput` 参数，被检索对象的数量。

示例

检索所有 proxy

检索所有已配置的 proxy 及其接口。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "proxy.get",
    "params": {
        "output": "extend",
        "selectInterface": "extend"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "host": "Active proxy",
            "status": "5",
            "lastaccess": "0",
            "description": "",
            "tls_connect": "1",
            "tls_accept": "1",
            "tls_issuer": "",
            "tls_subject": "",
            "proxy_address": "",
            "auto_compress": "0",
            "proxyid": "30091",
            "interface": []
        },
        {
            "host": "Passive proxy",
            "status": "6",
            "lastaccess": "0",
            "description": "",
            "tls_connect": "1",
            "tls_accept": "1",
            "tls_issuer": "",
            "tls_subject": "",
            "proxy_address": "",
            "auto_compress": "0",
            "proxyid": "30092",
            "interface": {
                "interfaceid": "30109",
                "hostid": "30092",
                "useip": "1",
                "ip": "127.0.0.1",
                "dns": "",
                "port": "10051"
            }
        }
    ],
    "id": 1
}
```


参见

- 主机
- proxy 接口

来源

CProxy::get() in *ui/include/classes/api/services/CProxy.php*.

SLA

这个类旨在使用 SLA（服务等级协议）对象来评估 IT 基础设施和商业服务的性能。

对象引用：

- SLA
- SLA 时间表
- SLA 排除的停机时间
- SLA 服务标签

可用方法：

- sla.create - 创建新的 SLA
- sla.delete - 删除 SLA

- sla.get - 检索 SLA
- sla.getsli - 检索 SLA 的服务等级指标（SLI）数据
- sla.update - 更新 SLA

> SLA 对象

以下对象与 `sla`（服务级别协议）API 直接相关。

SLA

SLA 对象具有以下属性。

属性	类型	Description
slaid	字符串	*(只读)*SLA 的 ID
name (必填)	字符串	SLA 的名称
period (必填)	整型	SLA 的报告期。 可能的值： 0 - 每天； 1 - 每周； 2 - 每月； 3 - 每季度； 4 - 每年。
slo (必填)	浮点数	以百分比表示的最低可接受服务水平目标。如果服务水平指标 (SLI) 下降，则 SLA 被认为处于问题/未完成状态。 可能的值：0-100（最多 4 个小数位）。
effective_date	整型	SLA 的生效日期。 可能的值：UTC 日期时间戳。
timezone (必填)	字符串	报告时区，例如：`Europe/London`、`UTC`。 有关支持的时区的完整列表，请参阅 PHP 文档。
status	整型	SLA 的状态。 可能的值： 0 - （默认）禁用 SLA； 1 - 启用 SLA。
description	字符串	SLA 的描述

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

SLA 时间表

SLA 计划对象定义了连接的服务计划处于工作状态的时间段。它具有以下属性。

属性	类型	描述
period_from (必填)	整型	每周重复周期的开始时间（含）。 可能的值：秒数（从星期日开始计算）。
period_to (必填)	整型	每周重复周期的结束时间（不包括）。 可能的值：秒数（从星期日开始计算）。

SLA 排除停机时间

排除的停机时间对象定义了计划连接的服务无法正常工作的时间段，不影响 SLI，例如正在进行有计划的维护。

属性	类型	Description
name (必填)	字符串	排除的停机时间的名称。
period_from (必填)	整型	排除的停机时间（含）的开始时间。 可能的值：时间戳。
period_to (必填)	整型	排除停机时间的结束时间（不包括）。 可能的值：时间戳。

SLA 服务标签

SLA 服务标签对象链接服务以包括在 SLA 的计算中。它具有以下属性。

属性	类型	描述
tag (必选)	string	SLA 服务标签名称。
operator	integer	SLA 服务标签操作符。 可能的值： 0 - *(默认)* 等于； 2 - 包含。
value	string	SLA 服务标签值。

sla.get

描述

`integer/array sla.get(object parameters)`

该方法允许根据给定的参数检索 SLA 对象。

Note:
此方法适用于任何类型的用户。权限调用方法可以在用户角色设置中撤销。请参阅用户角色 了解更多信息。

参数

`(object)` 定义所需输出的参数。

该方法支持以下参数。

参数	类型	描述
slaids	字符串/数组	仅返回具有给定 ID 的 SLA。
serviceids	字符串/数组	仅返回与特定服务匹配的 SLA。
selectSchedule	查询	返回带有 SLA 计划的 “计划” 属性。 支持 “计数”。
selectExcludedDowntimes	查询	返回包含 SLA 排除的停机时间的 `excluded_downtimes` 属性。 支持 计数。
selectServiceTags	查询	返回带有 SLA 服务标签的 `service_tags` 属性。 支持 计数。
sortfield	字符串/数组	按给定属性对结果进行排序。 可能的值为：`slaid`、`name`、`period`、`slo`、`effective_date`、`timezone`、`status` 和 `description`。
countOutput	布尔	参考注释 中详细描述了所有 “get” 方法通用的这些参数。
editable	布尔	

参数	类型	描述
excludeSearch	布尔	
filter	对象	
limit	整数	
output	查询	
preservekeys	布尔	
search	对象	
searchByAny	布尔	
searchWildcardsEnabled	布尔	
sortorder	字符串/数组	
startSearch	布尔	

返回值

(integer/array) 返回以下任一选项

- 对象数组;
- 如果使用 `countOutput` 参数，被检索对象的数量。

示例

检索所有 SLAs

检索有关所有 SLA 及其属性的所有数据。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "sla.get",
    "params": {
        "output": "extend",
        "selectSchedule": ["period_from", "period_to"],
        "selectExcludedDowntimes": ["name", "period_from", "period_to"],
        "selectServiceTags": ["tag", "operator", "value"],
        "preservekeys": true
    },
    "auth": "85dd04b94cbfad794616eb923be13c71",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "1": {
            "slaid": "1",
            "name": "Database Uptime",
            "period": "1",
            "slo": "99.9995",
            "effective_date": "1672444800",
            "timezone": "America/Toronto",
            "status": "1",
            "description": "Provide excellent uptime for main SQL database engines.",
            "service_tags": [
                {
                    "tag": "Database",
                    "operator": "0",
                    "value": "MySQL"
                },
                {
                    "tag": "Database",
                    "operator": "0",
                    "value": "PostgreSQL"
                }
```

```
            ],
            "schedule": [
                {
                    "period_from": "0",
                    "period_to": "601200"
                }
            ],
            "excluded_downtimes": [
                {
                    "name": "Software version upgrade rollout",
                    "period_from": "1648760400",
                    "period_to": "1648764900"
                }
            ]
        }
    },
    "id": 1
}
```


来源

CSla:get() in *ui/include/classes/api/services/CSla.php*.

sla.getsli

描述

`object sla.getsli(object parameters)`

此方法允许计算服务等级协议（SLA）的服务等级指标（SLI）数据。

> **Note:**
> 此方法适用于任何类型的用户。可以在用户角色设置中撤销调用方法的权限。更多信息请参阅用户角色。

参数

(object) 包含 SLA ID、报告期以及可选的服务 ID 的参数 - 用于计算 SLI。

参数	类型	描述
slaid (必选)	string	返回可用性信息的 SLA 的 ID。
period_from	integer	报告 SLI 的起始日期（包括）。 可能的值：时间戳。
period_to	integer	报告 SLI 的结束日期（不包括）。 可能的值：时间戳。
periods	array	希望报告的周期数量。 可能的值：1-100
serviceids	string/array	返回 SLI 的服务的 ID。

时段划分

下表展示了基于参数组合返回的时段切片的安排。

返回的时段不会早于基于 SLA 生效日期的第一个可用时段，也不会超出当前时段。

参数			描述
period_from	**period_to**	**periods**	
-	-	-	返回最后 20 个时段。
-	-	指定	根据 **periods** 参数指定返回最后几个时段。
-	指定	-	返回指定 **period_to** 日期之前的最后 20 个时段。
-	指定	指定	返回指定 **period_to** 日期之前的、由 **periods** 参数指定的最后几个时段。
指定	-	-	从指定的 **period_from** 日期开始返回前 20 个时段。
指定	-	指定	从指定的 **period_from** 日期开始返回由 **periods** 参数指定的前几个时段。

参数			描述
指定	指定	-	返回在指定日期范围内的最多 100 个时段。
指定	指定	指定	返回在指定日期范围内、由 **periods** 参数指定的时段。

返回值

(object) 返回计算的结果

属性	类型	描述
periods	数组	报告期间的列表。 每个报告期间都表示为一个对象，包括： - *period_from* - 报告期间的开始日期（时间戳）。 - *period_to* - 结束报告期间的日期（时间戳）。 期间按 period_from 字段升序排序。
serviceids	数组	报告期间的服务 ID 列表。 未定义列表的排序顺序。即使 serviceids 参数被传递给 sla.getsli 方法。
sli	数组	每个报告的期间和服务的 SLI 数据（作为二维数组）。 periods 属性的索引用作 sli 的第一个维度 serviceids 属性的索引用作 sli 属性的 **second** 维度。

SLI 数据

每个报告期间和服务返回的 SLI 数据包括：

属性	类型	描述
uptime	整型	在计划的正常运行时间内服务处于 *OK* 状态的时间量，减去排除的停机时间。
downtime	整型	在计划的正常运行时间内服务处于 _not OK_ 状态的时间量，减去排除的停机时间。
sli	浮点数	SLI（占总正常运行时间的百分比），基于正常运行时间和停机时间。
error_budget	整型	基于 SLI 和 SLO 的错误预算（以秒为单位）。
excluded_downtimes	数组	此报告期间排除的停机时间数组。 每个对象将包含以下参数： - *name* - 排除的停机时间的名称。 - *period_from* - 开始日期和时间（含）排除的停机时间。 - *period_to* - 排除的停机时间的结束日期和时间（不包括）。 排除的停机时间按 period_from 字段升序排序。

示例

计算 SLI

检索与 SLA ID 为”5” 相关联的服务 ID 为”50”、”60” 和”70” 的 SLI 数据。从 2021 年 11 月 01 日开始检索 3 个周期的数据。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "sla.getsli",
    "params": {
        "slaid": "5",
        "serviceids": [
            50,
            60,
            70
        ],
```

```
        "periods": 3,
        "period_from": "1635724800"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "periods": [
            {
                "period_from": 1635724800,
                "period_to": 1638316800
            },
            {
                "period_from": 1638316800,
                "period_to": 1640995200
            },
            {
                "period_from": 1640995200,
                "period_to": 1643673600
            }
        ],
        "serviceids": [
            50,
            60,
            70
        ],
        "sli": [
            [
                {
                    "uptime": 1186212,
                    "downtime": 0,
                    "sli": 100,
                    "error_budget": 0,
                    "excluded_downtimes": [
                        {
                            "name": "Excluded Downtime - 1",
                            "period_from": 1637836212,
                            "period_to": 1638316800
                        }
                    ]
                },
                {
                    "uptime": 1186212,
                    "downtime": 0,
                    "sli": 100,
                    "error_budget": 0,
                    "excluded_downtimes": [
                        {
                            "name": "Excluded Downtime - 1",
                            "period_from": 1637836212,
                            "period_to": 1638316800
                        }
                    ]
                },
                {
                    "uptime": 1186212,
                    "downtime": 0,
                    "sli": 100,
```

```
                "error_budget": 0,
                "excluded_downtimes": [
                    {
                        "name": "Excluded Downtime - 1",
                        "period_from": 1637836212,
                        "period_to": 1638316800
                    }
                ]
            }
        ],
        [
            {
                "uptime": 1147548,
                "downtime": 0,
                "sli": 100,
                "error_budget": 0,
                "excluded_downtimes": [
                    {
                        "name": "Excluded Downtime - 1",
                        "period_from": 1638439200,
                        "period_to": 1639109652
                    }
                ]
            },
            {
                "uptime": 1147548,
                "downtime": 0,
                "sli": 100,
                "error_budget": 0,
                "excluded_downtimes": [
                    {
                        "name": "Excluded Downtime - 1",
                        "period_from": 1638439200,
                        "period_to": 1639109652
                    }
                ]
            },
            {
                "uptime": 1147548,
                "downtime": 0,
                "sli": 100,
                "error_budget": 0,
                "excluded_downtimes": [
                    {
                        "name": "Excluded Downtime - 1",
                        "period_from": 1638439200,
                        "period_to": 1639109652
                    }
                ]
            }
        ],
        [
            {
                "uptime": 1674000,
                "downtime": 0,
                "sli": 100,
                "error_budget": 0,
                "excluded_downtimes": []
            },
            {
                "uptime": 1674000,
                "downtime": 0,
```

```
                    "sli": 100,
                    "error_budget": 0,
                    "excluded_downtimes": []
                },
                {
                    "uptime": 1674000,
                    "downtime": 0,
                    "sli": 100,
                    "error_budget": 0,
                    "excluded_downtimes": []
                }
            ]
        ]
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CSla.php 中的 CSla::getSli()

sla 创建

描述

`object sla.create(object/array SLAs)`

这个方法可以用来创建主机

> **Note:**
> 这个方法仅允许 管理员和 超级管理员用户类型。可以在用户角色中撤销调用方法的权限设置。参考User roles获取详情

参数

(object/array) 要创建的 SLA 对象

除了标准 SLA 属性, 该方法接受以下参数

参数	类型	Description
service_tags (必选)	数组	为 SLA 创建的 SLA 服务标签. 必须指定至少一个服务标签
schedule	数组	为 SLA 创建 SLA 计划. 指定空参数将被解释为 24x7 计划 默认: 24x7 计划。
excluded_downtimes	数组	SLA 排除了为 SLA 创建的停机时间

返回值

(object) 返回一个包含已创建 SLA 的 ID 的对象，在 `slaids` 属性下。返回的 ID 的顺序匹配通过的 SLA 的顺序。

示例

创建 SLA

指示为以下对象创建 SLA 条目：* 跟踪 SQL 引擎相关服务的正常运行时间；* 除周六最后一小时外的所有工作日的自定义时间表；* 生效日期为 2022 年最后一天；¶ * 从 7 月 4 日午夜开始，计划停机时间为 1 小时 15 分钟；* SLA 周报计算将开启；* 可接受的最低 SLO 为 99.9995%。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "sla.create",
    "params": [
        {
            "name": "Database Uptime",
```

```
            "slo": "99.9995",
            "period": "1",
            "timezone": "America/Toronto",
            "description": "Provide excellent uptime for main database engines.",
            "effective_date": 1672444800,
            "status": 1,
            "schedule": [
                {
                    "period_from": 0,
                    "period_to": 601200
                }
            ],
            "service_tags": [
                {
                    "tag": "Database",
                    "operator": "0",
                    "value": "MySQL"
                },
                {
                    "tag": "Database",
                    "operator": "0",
                    "value": "PostgreSQL"
                }
            ],
            "excluded_downtimes": [
                {
                    "name": "Software version upgrade rollout",
                    "period_from": "1648760400",
                    "period_to": "1648764900"
                }
            ]
        }
    ],
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```


响应:


```
{
    "jsonrpc": "2.0",
    "result": {
        "slaids": [
            "5"
        ]
    },
    "id": 1
}
```


来源

CSla::create() in *ui/include/classes/api/services/CSla.php*.

sla 删除

描述

`object sla.delete(array slaids)`

该方法运行删除 SLA 实体::: noteclassic 此方法只有 *Admin*（管理员）和 *Super admin*（超级管理员）用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(`array`) 要删除的 SLA 的 id 列表

返回值（object）返回一个包含已删除 SLA 的 ID 的对象 ¶ 在 slaids 属性下。

示例

删除多个 SLA

删除两个 SLA 实体

请求:

```
{
    "jsonrpc": "2.0",
    "method": "sla.delete",
    "params": [
        "4",
        "5"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "slaids": [
            "4",
            "5"
        ]
    },
    "id": 1
}
```

来源

CSla::delete() in *ui/include/classes/api/services/CSla.php*.

sla 更新

描述

此方法允许更新现有的 SLA 条目。

> **Note:**
> 此方法仅适用于 *Admin* 和 *Super admin* 用户类型。可以在用户角色中撤销调用该方法的权限设置。请参阅用户角色 了解更多信息。

参数

（object/array）SLA 属性将被更新

必须为每个 SLA 定义 slaid 属性，所有其他属性是可选的。只有传递的属性会被更新，所有其他的将保持不变。

除了标准 SLA 属性，方法接受以下参数。

参数	类型	描述
service_tags	数组	SLA 服务标签 替换当前的 SLA 服务标签。 必须至少指定一个服务标签。
schedule	数组	SLA schedule 替换当前的。 将参数指定为空将被解释为 24x7 计划。
excluded_downtimes	数组	SLA 排除停机时间 替换当前的。

返回值

（object）返回一个对象，该对象包含 slaids 属性下更新的 SLA 的 ID。

示例

更新服务标签

为 NoSQL 相关服务每月计算一次 ID 为 “5” 的 SLA，不改变其时间表或排除停机时间；将 SLO 设置为 95%。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "sla.update",
    "params": [
        {
            "slaid": "5",
            "name": "NoSQL Database engines",
            "slo": "95",
            "period": 2,
            "service_tags": [
                {
                    "tag": "Database",
                    "operator": "0",
                    "value": "Redis"
                },
                {
                    "tag": "Database",
                    "operator": "0",
                    "value": "MongoDB"
                }
            ]
        }
    ],
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "slaids": [
            "5"
        ]
    },
    "id": 1
}
```

更改 SLA 计划

将 ID 为 “5” 的 SLA 切换为 24x7 计划。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "service.update",
    "params": {
        "slaid": "5",
        "schedule": []
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "slaids": [
```

```
            "5"
        ]
    },
    "id": 1
}
```

更改 SLA 的排除停机时间

对于 ID 为 “5” 的 SLA，添加在 2022 年 4 月 6 日计划的 4 小时 RAM 升级停机时间，同时保留（需要重新定义）先前存在的 7 月 4 日软件升级计划。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "service.update",
    "params": {
        "slaid": "5",
        "excluded_downtimes": [
            {
                "name": "Software version upgrade rollout",
                "period_from": "1648760400",
                "period_to": "1648764900"
            },
            {
                "name": "RAM upgrade",
                "period_from": "1649192400",
                "period_to": "1649206800"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "slaids": [
            "5"
        ]
    },
    "id": 1
}
```

来源

CSla::update() in *ui/include/classes/api/services/CSla.php*.

Web 场景

此类用于 Web 场景。

对象引用：

- Web scenario
- Scenario step

可用方法：

- httptest.create - 创建新的 Web 场景
- httptest.delete - 删除 Web 场景
- httptest.get - 获取 Web 场景

- httptest.update - 更新 Web 场景

> Web 场景对象

以下对象都是与 `webcheck` 直接相关的 API。

Web 场景

Web 场景对象具有以下属性。

属性	类型	描述
httptestid	string	*(readonly)* Web 场景的 ID。
hostid (required)	string	Web 场景所属主机的 ID。
name (required)	string	Web 场景的名称。
agent	string	Web 场景将使用的 User agent 字符串。 默认值：Zabbix
authentication	integer	Web 场景将使用的认证方法。 可能的值： 0 - *(default)* 无； 1 - 基本 HTTP 认证； 2 - NTLM 认证。
delay	string	Web 场景的执行间隔。接受秒数，时间单位后缀和用户宏。 默认值：1m。
headers	array 参考HTTP 字段	执行请求时将发送的 HTTP 头部。
http_password	string	用于基本 HTTP 或 NTLM 认证的密码。
http_proxy	string	Web 场景使用的代理，格式为 *http://[username[:password]@]proxy.example.com[:p*
http_user	string	用于基本 HTTP 或 NTLM 认证的用户名。
nextcheck	timestamp	*(readonly)* 下次 Web 场景执行的时间。
retries	integer	Web 场景在失败前尝试执行每个步骤的次数。 默认值：1。
ssl_cert_file	string	用于客户端认证的 SSL 证书文件名称（必须为 PEM 格式）。
ssl_key_file	string	用于客户端认证的 SSL 私钥文件名称（必须为 PEM 格式）。
ssl_key_password	string	SSL 私钥密码。
status	integer	Web 场景是否启用。 可能的值是： 0 - *(default)* 启用； 1 - 禁用。
templateid	string	*(readonly)* 父模板 Web 场景的 ID。
variables	array 参考HTTP 字段	Web 场景变量。
verify_host	integer	是否验证连接的主机名是否与主机证书中的名称匹配。 可能的值是： 0 - *(default)* 跳过主机验证； 1 - 验证主机。
verify_peer	integer	是否验证主机的证书是否真实。 可能的值是： 0 - *(default)* 跳过对等方验证； 1 - 验证对等方。

属性	类型	描述
uuid	string	全局唯一标识符，用于将导入的 Web 场景链接到已存在的 Web 场景。仅用于模板上的 Web 场景。

请注意，对于一些方法（更新，删除），所需/可选参数的组合是不同的。

Web 场景标签

Web 场景标签对象具有如下属性。

属性	类型	描述
tag (必填)	string	Web 场景标签的名称。
value	string	Web 场景标签的值。

场景步骤

场景步骤对象定义了一个特定的 Web 场景检查。具有如下属性。

属性	类型	描述
httpstepid	string	*(只读)* Web 场景步骤 ID。
name (必填)	string	Web 场景的名称。
no (必填)	integer	一个 Web 场景步骤的序列号。
url (必填)	string	待检查的 URL。
follow_redirects	integer	是否遵循 HTTP 重定向。 可用值： 0 - 不遵循重定向； 1 - *(默认)* 遵循重定向。
headers	array of HTTP fields	执行请求时将发送的 HTTP 请求头。 场景步骤请求头将被重写为 Web 场景指定的请求头。
httptestid	string	*(只读)* 步骤所属 Web 场景的 ID。
posts	string array of HTTP fields	HTTP POST 变量作为一个字符串（原始的 post 数据） 或一个HTTP fields (表单字段数据) 的数组。
required	string	响应中必须出现的文本。
retrieve_mode	integer	场景步骤必须获取的一部分 HTTP 响应。 可用值： 0 - *(默认)* 仅返回内容； 1 - 仅请求头； 2 - 请求头和内容。

属性	类型	描述
status_codes	string	用逗号隔开的所需 HTTP 状态码的范围。
timeout	string	请求超时时间(秒)。接受秒，带后缀的时间单位和用户宏。 默认：15s。最大值：1h。最小值: 1s。
variables	array of HTTP fields	场景步骤的变量。
query_fields	array of HTTP fields	查询字段 - 执行请求时将被添加到 URL 中的HTTP fields数组。

HTTP 字段

HTTP 字段定义了名称和值，用于指定查询字段数据中的变量，HTTP 请求头，POST 表单字段数据。具有如下属性。

属性	类型	描述
name (必填)	string	请求头/变量/POST 或 GET 字段的名称。
value (必填)	string	请求头/变量/POST 或 GET 字段的值。

创建

描述

`object httptest.create(object/array webScenarios)`

此方法允许创建新的 Web 场景。

Note:
创建一个 Web 场景将自动创建一组web monitoring items。

Note:
此方法只有 *Admin(管理员)* 和 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看User roles。

参数

`(object/array)` 需要创建的 Web 场景。

除了standard web scenario properties，此方法接受如下参数。

参数	类型	描述
steps (必填)	array	Web 场景steps。
tags	array	Web 场景tags。

返回值

`(object)` 返回一个包含已创建 Web 场景 ID 的对象，ID 在 `httptestids` 属性下。返回的 IDs 的顺序和传递的 Web 场景的顺序相匹配。

示例

创建一个 Web 场景

创建一个 Web 场景用于监控公司主页。此场景具有两个步骤，检查主页和 “关于” 页，并保证它们返回 HTTP 状态码 200。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "httptest.create",
    "params": {
        "name": "Homepage check",
        "hostid": "10085",
        "steps": [
            {
                "name": "Homepage",
                "url": "http://example.com",
                "status_codes": "200",
                "no": 1
            },
            {
                "name": "Homepage / About",
                "url": "http://example.com/about",
                "status_codes": "200",
                "no": 2
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "httptestids": [
            "5"
        ]
    },
    "id": 1
}
```

参见

- Scenario step

来源

CHttpTest::create() in *ui/include/classes/api/services/CHttpTest.php*.

删除

描述

`object httptest.delete(array webScenarioIds)`

此方法允许删除 Web 场景。

> **Note:**
> 此方法只有 *Admin(管理员)* 和 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看User roles。

参数

(array) 需要被删除的 Web 场景的 ID。

返回值

(object) 返回一个包含被删除的 Web 场景的 ID 的对象，ID 在 httptestids 属性下。

示例

删除多个 Web 场景

删除两个 Web 场景。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "httptest.delete",
    "params": [
        "2",
        "3"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "httptestids": [
            "2",
            "3"
        ]
    },
    "id": 1
}
```

来源

CHttpTest::delete() in *ui/include/classes/api/services/CHttpTest.php*.

更新

描述

`object httptest.update(object/array webScenarios)`

此方法允许更新存在的 Web 场景。

> **Note:**
> 此方法只有 *Admin(管理员)* 和 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看User roles。

参数

(object/array) 需要被更新的 Web 场景属性。

每个 Web 场景必须定义 httptestid 属性，其他的所有属性是可选的。只有通过的属性将被更新，其他的属性保持不变。

除了standard web scenario properties以外，此方法接受以下参数。

参数	类型	描述
steps	array	场景步骤用于替换现有步骤。
tags	array	Web 场景标签。

返回值

(object) 返回一个包含被更新 Web 场景的 ID 的对象，ID 在 httptestid 属性下。

示例

启用一个 Web 场景

启动一个 Web 场景，即将其状态设置为 “0”。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "httptest.update",
    "params": {
        "httptestid": "5",
        "status": 0
    },
    "auth": "700ca65537074ec963db7efabda78259",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "httptestids": [
            "5"
        ]
    },
    "id": 1
}
```

参见

- Scenario step

来源

CHttpTest::update() in *ui/include/classes/api/services/CHttpTest.php*.

获取

描述

`integer/array httptest.get(object parameters)`

此方法允许根据给出的参数检索 Web 场景。

> **Note:**
> 此方法适用于任何类型的用户。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看User roles。

参数

`(object)` 定义期望输出的参数。

此方法支持以下参数。

参数	类型	描述
groupids	string/array	仅返回属于给定主机组的 Web 场景。
hostids	string/array	仅返回属于给定主机的 Web 场景。
httptestids	string/array	仅返回给定 ID 的 Web 场景。
inherited	boolean	如果设置为 `true`，仅返回从模板继承的 Web 场景。
monitored	boolean	如果设置为 `true`，仅返回属于被监控主机的可用 Web 场景。

参数	类型	描述
templated	boolean	如果设置为 `true`，仅返回属于模板的 Web 场景。
templateids	string/array	仅返回属于给定模板的 Web 场景。
expandName	flag	以 Web 场景的名称展开宏。
expandStepName	flag	以场景步骤的名称展开宏。
evaltype	integer	用于标签搜索的规则。 可用值： 0 - (默认) And/Or； 2 - Or。
tags	array of objects	仅返回给定标签的 Web 场景。根据标签进行精确匹配，并根据运算符值按标签值进行区分大小写或不区分大小写的搜索。 格式：`[{"tag": "<tag>", "value": "<value>", "operator": "<operator>"}, ...]`。 一个空数组返回所有的 Web 场景。 可能的运算符类型： 0 - (默认) Like； 1 - Equal； 2 - Not like； 3 - Not equal； 4 - Exists； 5 - Not exists。
selectHosts	query	在hosts属性中，以一个数组的方式返回 Web 场景所属的主机。
selectSteps	query	在steps属性中返回 Web 场景步骤。 支持 `count`。
selectTags	query	在tags属性中返回 Web 场景标签。
sortfield	string/array	按照给定属性对结果进行排序。 可能的值：`httptestid` 和 `name`。
countOutput	boolean	这些参数对所有 `get` 方法是公共的，详细描述请参见参考说明 。

参数	类型	描述
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

（integer/array）返回以下其中一种结果：

- 一个对象数组；
- 如果使用了参数 countOutput，则返回检索到的对象的数量。

示例

检索一个 Web 场景

检索关于 Web 场景 “4” 的所有数据。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "httptest.get",
    "params": {
        "output": "extend",
        "selectSteps": "extend",
        "httptestids": "9"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "httptestid": "9",
            "name": "Homepage check",
            "nextcheck": "0",
            "delay": "1m",
            "status": "0",
            "variables": [],
            "agent": "Zabbix",
            "authentication": "0",
            "http_user": "",
            "http_password": "",
            "hostid": "10084",
            "templateid": "0",
            "http_proxy": "",
            "retries": "1",
            "ssl_cert_file": "",
            "ssl_key_file": "",
            "ssl_key_password": "",
            "verify_peer": "0",
            "verify_host": "0",
            "headers": [],
```

```
            "steps": [
                {
                    "httpstepid": "36",
                    "httptestid": "9",
                    "name": "Homepage",
                    "no": "1",
                    "url": "http://example.com",
                    "timeout": "15s",
                    "posts": "",
                    "required": "",
                    "status_codes": "200",
                    "variables": [
                        {
                            "name":"{var}",
                            "value":"12"
                        }
                    ],
                    "follow_redirects": "1",
                    "retrieve_mode": "0",
                    "headers": [],
                    "query_fields": []
                },
                {
                    "httpstepid": "37",
                    "httptestid": "9",
                    "name": "Homepage / About",
                    "no": "2",
                    "url": "http://example.com/about",
                    "timeout": "15s",
                    "posts": "",
                    "required": "",
                    "status_codes": "200",
                    "variables": [],
                    "follow_redirects": "1",
                    "retrieve_mode": "0",
                    "headers": [],
                    "query_fields": []
                }
            ]
        }
    ],
    "id": 1
}
```


参见

- Host
- Scenario step

来源

CHttpTest::get() in *ui/include/classes/api/services/CHttpTest.php*.

主机

这个类是设计用于处理主机。

对象引用:

- Host
- Host inventory

相关方法:

- host.create - 创建新的主机
- host.delete - 删除主机

- host.get - 获取主机信息
- host.massadd - 主机配置批量添加
- host.massremove - 主机配置批量删除
- host.massupdate - 主机配置批量更新
- host.update - 更新主机

> 主机对象

以下对象直接与主机 API 相关。

主机

主机对象具有以下属性。

Property	Type	Description
hostid	字符	*(只读)* 主机的 ID。
host (必须)	字符	主机的正式名称。
description	文本	主机的描述信息。
flags	整数	*(只读)* 主机的来源。 取值范围: 0—普通主机; 4—自动发现的主机。
inventory_mode	整数	主机资产模式。 取值范围为: -1 - *(默认)* 禁用的; 0 - 手动的; 1 - 自动的。
ipmi_authtype	整数	IPMI 验证算法。 可能的值: -1 - *(默认)* default; 0 - none; 1 - MD2; 2 - MD5 4 - straight; 5 - OEM; 6 - RMCP+.
ipmi_password	字符	IPMI 密码。
ipmi_privilege	整数	IPMI 权限等级。 可能的值: 1 - callback; 2 - *(默认)* user; 3 - operator; 4 - admin; 5 - OEM.
ipmi_username	字符	IPMI 用户名。
maintenance_from	时间戳	*(只读)* 有效维护启动时间。
maintenance_status	整数	*(只读)* 有效的维护状态。 取值包括: 0 - *(默认)* 不维护; 1 -有效维护。

Property	Type	Description
maintenance_type	整数	*(只读)* 有效的维护类型。 可能的值是: 0 - *(默认)* 数据收集维护; 1 - 数据不收集维护。
maintenanceid	字符	*(只读)* 主机上当前生效的维护 ID。
name	字符	可见的主机名。 默认: `host` 属性值。
proxy_hostid	字符	用于监控主机的 Proxy ID。
status	整数	主机的状态和功能。 可能的值是: 0 - *(默认)* 开启监控的主机; 1 - 没有开启监控的主机。
tls_connect	整数	链接到主机。 可能的值是: 1 - *(默认)* 没有加密; 2 - PSK; 4 - 已加密。
tls_accept	整数	连接主机。 可能的位图值如下:: 1 - *(默认)* 未加密; 2 - PSK; 4 - 已认证.
tls_issuer	字符	证书问题。
tls_subject	字符	证书发行者。
tls_psk_identity	字符	*(只写)* PSK 认证。 如果启用了 `tls_connect` 或 `tls_accept` ，则需要。 不要将敏感信息放入 PSK 标识中，该信息通过网络明文传输，以告知接收端使用哪个 PSK。
tls_psk	字符	*(只写)* 预共享密钥，至少 32 位十六进制数字。如果启用了 `tls_connect` 或 `tls_accept` ，则需要。

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

主机资产

主机资产对象具有以下属性。

Note:
每个属性都有自己的唯一 ID 号，该 ID 号用于将主机资产字段与监控项关联起来。

ID	属性	类型	描述	最大长度
4	alias	string	别名。	128 个字符
11	asset_tag	string	资产标签。	64 个字符
28	chassis	string	机箱。	64 个字符
23	contact	string	联系人。	取决于所使用的数据库： - SQL 数据库为 65535 个字符 - Oracle 数据库为 2048 个字符
32	contract_number	string	合同号码。	64 个字符
47	date_hw_decomm	string	硬件退役日期。	64 个字符
46	date_hw_expiry	string	硬件维护到期日期。	64 个字符
45	date_hw_install	string	硬件安装日期。	64 个字符
44	date_hw_purchase	string	硬件购买日期。	64 个字符
34	deployment_status	string	部署状态。	64 个字符
14	hardware	string	硬件。	255 个字符
15	hardware_full	string	硬件详细信息。	取决于所使用的数据库： - SQL 数据库为 65535 个字符 - Oracle 数据库为 2048 个字符
39	host_netmask	string	主机子网掩码。	39 个字符
38	host_networks	string	主机网络。	取决于所使用的数据库： - SQL 数据库为 65535 个字符 - Oracle 数据库为 2048 个字符
40	host_router	string	主机路由器。	39 个字符
30	hw_arch	string	硬件架构。	32 个字符
33	installer_name	string	安装程序名称。	64 个字符
24	location	string	位置。	取决于所使用的数据库： - SQL 数据库为 65535 个字符 - Oracle 数据库为 2048 个字符
25	location_lat	string	位置纬度。	16 个字符
26	location_lon	string	位置经度。	16 个字符
12	macaddress_a	string	MAC 地址 A.	64 个字符
13	macaddress_b	string	MAC 地址 B.	64 个字符
29	model	string	模式。	64 个字符
3	name	string	名称。	128 个字符
27	notes	string	备注。	取决于所使用的数据库： - SQL 数据库为 65535 个字符 - Oracle 数据库为 2048 个字符
41	oob_ip	string	OOB IP 地址。	39 个字符
42	oob_netmask	string	OOB 主机子网掩码。	39 个字符
43	oob_router	string	OOB 路由器。	39 个字符
5	os	string	操作系统名称。	128 个字符
6	os_full	string	详细的操作系统名称。	255 个字符
7	os_short	string	操作系统简称。	128 个字符
61	poc_1_cell	string	主要 POC 手机号码。	64 个字符
58	poc_1_email	string	主要电子邮件。	128 个字符
57	poc_1_name	string	主要 POC 名称。	128 个字符
63	poc_1_notes	string	主要 POC 注释。	取决于所使用的数据库： - SQL 数据库为 65535 个字符 - Oracle 数据库为 2048 个字符
59	poc_1_phone_a	string	主要 POC 电话 A.	64 个字符
60	poc_1_phone_b	string	主要 POC 电话 B.	64 个字符
62	poc_1_screen	string	主要 POC 屏幕名称。	64 个字符
68	poc_2_cell	string	辅助 POC 手机号码。	64 个字符
65	poc_2_email	string	辅助 POC 电子邮件。	128 个字符
64	poc_2_name	string	二级 POC 名称。	128 个字符
70	poc_2_notes	string	次要 POC 注释。	取决于所使用的数据库： - SQL 数据库为 65535 个字符 - Oracle 数据库为 2048 个字符

ID	属性	类型	描述	最大长度
66	poc_2_phone_a	string	辅助 POC 电话 A.	64 个字符
67	poc_2_phone_b	string	辅助 POC 电话 B.	64 个字符
69	poc_2_screen	string	次要 POC 屏幕名称。	64 个字符
8	serialno_a	string	序列号 A.	64 个字符
9	serialno_b	string	序列号 B.	64 个字符
48	site_address_a	string	站点地址 A.	128 个字符
49	site_address_b	string	站点地址 B.	128 个字符
50	site_address_c	string	站点地址 C.	128 个字符
51	site_city	string	站点城市。	128 个字符
53	site_country	string	网站国家。	64 个字符
56	site_notes	string	站点注释。	取决于所使用的数据库： - SQL 数据库为 65535 个字符 - Oracle 数据库为 2048 个字符
55	site_rack	string	站点机架位置。	128 个字符
52	site_state	string	站点状态。	64 个字符
54	site_zip	string	站点邮政编码。	64 个字符
16	software	string	软件。	255 个字符
18	software_app_a	string	软件应用程序 A.	64 个字符
19	software_app_b	string	软件应用程序 B.	64 个字符
20	software_app_c	string	软件应用程序 C.	64 个字符
21	software_app_d	string	软件应用程序 D.	64 个字符
22	software_app_e	string	软件应用程序 E.	64 个字符
17	software_full	string	软件详细信息。	取决于所使用的数据库： - SQL 数据库为 65535 个字符 - Oracle 数据库为 2048 个字符
10	tag	string	Tag.	64 个字符
1	type	string	类型。	64 个字符
2	type_full	string	类型详细信息。	64 个字符
35	url_a	string	URL A.	255 个字符
36	url_b	string	URL B.	255 个字符
37	url_c	string	URL C.	255 个字符
31	vendor	string	厂家。	64 个字符

Host 标记

主机标记对象具有以下属性。

属性	类	描述
tag (required)	string	主机标记名称。
value	string	主机标记的值。

创建

描述

```
object host.create(object/array hosts)
```

这个方法可以用来创建主机。

Note:
这个方法仅允许 管理员和 超级管理员用户类型。可以在用户角色中撤销调用方法的权限设置。参考User roles获取详情

参数

（对象/数组）要创建的主机。

另外，对于标准的主机属性, 该方法接受下列参数。

参数	类型	描述
groups (必选)	object/array	把主机添加到目标组。 主机组必须已定义 groupid 属性。
interfaces	object/array	为主机创建的接口。
tags	object/array	主机标签。
templates	object/array	主机连接的模板。 模板必须已定义 templateid 属性。
macros	object/array	为主机创建的用户宏。
inventory	object	主机资产清单属性。

返回值

（对象）返回包含已创建主机 ID 的属性 hostids ，返回 ID 的顺序与传入主机的顺序一致。

示例

创建一个主机

创建一个具有 IP 接口和标签的 “Linux Server” 主机，将其添加到主机组中，链接一个模板并且把 MAC 地址设置到主机资产清单里。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.create",
    "params": {
        "host": "Linux server",
        "interfaces": [
            {
                "type": 1,
                "main": 1,
                "useip": 1,
                "ip": "192.168.3.1",
                "dns": "",
                "port": "10050"
            }
        ],
        "groups": [
            {
                "groupid": "50"
            }
        ],
        "tags": [
            {
                "tag": "Host name",
                "value": "Linux server"
            }
        ],
        "templates": [
            {
                "templateid": "20045"
            }
        ],
        "macros": [
            {
                "macro": "{$USER_ID}",
                "value": "123321"
            },
            {
                "macro": "{$USER_LOCATION}",
                "value": "0:0:0",
                "description": "latitude, longitude and altitude coordinates"
            }
        ],
        "inventory_mode": 0,
```

```
        "inventory": {
            "macaddress_a": "01234",
            "macaddress_b": "56768"
        }
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostids": [
            "107819"
        ]
    },
    "id": 1
}
```

创建一个具有 SNMP 接口的主机

创建一个名为 “SNMP host” 的主机，并创建 SNMPv3 接口。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.create",
    "params": {
        "host": "SNMP host",
        "interfaces": [
            {
                "type": 2,
                "main": 1,
                "useip": 1,
                "ip": "127.0.0.1",
                "dns": "",
                "port": "161",
                "details": {
                    "version": 3,
                    "bulk": 0,
                    "securityname": "mysecurityname",
                    "contextname": "",
                    "securitylevel": 1
                }
            }
        ],
        "groups": [
            {
                "groupid": "4"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostids": [
            "10658"
```

```
            ]
        },
        "id": 1
}
```

创建使用 PSK 加密的主机

创建一个名为 “PSK 主机” 的主机，并配置使用 PSK 加密。请注意，该主机必须预先配置为使用 PSK。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.create",
    "params": {
        "host": "PSK host",
        "interfaces": [
            {
                "type": 1,
                "ip": "192.168.3.1",
                "dns": "",
                "port": "10050",
                "useip": 1,
                "main": 1
            }
        ],
        "groups": [
            {
                "groupid": "2"
            }
        ],
        "tls_accept": 2,
        "tls_connect": 2,
        "tls_psk_identity": "PSK 001",
        "tls_psk": "1f87b595725ac58dd977beef14b97461a7c1045b9a1c963065002c5473194952"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostids": [
            "10590"
        ]
    },
    "id": 1
}
```

另请参阅

- 主机组
- 模板
- 用户宏
- 主机接口
- 主机资产
- 主机标签

源代码

CHost::create() in *ui/include/classes/api/services/CHost.php*.

删除

描述

`object host.delete(array hosts)`

这个方法允许删除主机。

> **Note:**
> 这个方法仅适用 管理员和 超级管理员用户类型。可以在用户角色中撤销调用方法的权限设置。参考User roles获取详情

参数

(array) 要删除主机的 ID。

返回值

(object) 返回一个具有 hostids 属性（其中包含被删除主机的 ID）的对象。

示例

删除多个主机

删除两个主机。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.delete",
    "params": [
        "13",
        "32"
    ],
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostids": [
            "13",
            "32"
        ]
    },
    "id": 1
}
```

源代码

CHost::delete() in *ui/include/classes/api/services/CHost.php*.

批量更新

描述

`object host.massupdate(对象参数)`

该方法可以在多台主机上同时替换或移除相关对象和更新属性。

> **Note:**
> 这个方法仅允许 管理员和 超级管理员用户类型。可以在用户角色中撤销调用方法的权限设置。详情参考用户角色

参数

(对象) 参数包含要更新的主机 id 和应该更新的属性。

除了标准主机属性以外，此方法可以接受如下参数:

参数	类	描述
hosts (必填)	对象/数组	待更新的主机 。 主机必须定义了 `hostid` 属性。
groups	对象/数组	更换主机所属的主机组。 主机组必须定义了 `groupid` 属性。
interfaces	对象/数组	主机接口 用于替换给定主机上的当前主机接口。
inventory	对象	主机清单属性。 主机清单模式不能使用 `inventory` 参数更新, 事情 `inventory_mode` 代替。
macros	对象/数组	用户宏 替换给定主机上的当前用户宏。
templates	对象/数组	模板替换给定主机上当前链接的模板。 模板必须定义了 `templateid` 属性。
templates_clear	对象/数组	模板 断开和清除给定主机的链接。 模板必须定义了 `templateid` 属性。

返回值

（对象）返回一个对象，该对象包含 `hostids` 属性下更新的主机的 id。

示例

启用多个主机

启用监控两个主机，将 status 设置为 0

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.massupdate",
    "params": {
        "hosts": [
            {
                "hostid": "69665"
            },
            {
                "hostid": "69666"
            }
        ],
        "status": 0
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostids": [
            "69665",
            "69666"
        ]
    },
    "id": 1
}
```

另请参阅

- 主机更新
- 主机批量创建
- 主机批量删除
- 主机组
- 模板
- 用户宏
- 主机接口

源代码

CHost::massUpdate() in *ui/include/classes/api/services/CHost.php*.

批量添加

描述

`host.massadd(对象参数)` 对象

此方法允许同时添加多个相关对象到所有给定的主机。

> **Note:**
> 这个方法仅允许 管理员和 超级管理员用户类型。可以在用户角色中撤销调用方法的权限设置。详情参考用户角色

参数

(对象) 参数包含要更新的主机 id 和要添加到所有主机的对象。

该方法接受以下参数。

参数	类	描述
hosts (必填)	对象/数组	待更新的主机。 主机必须定义了 `hostid` 属性。
groups	对象/数组	需要添加到指定主机的主机组。 主机组必须定义了 `groupid` 属性。
interfaces	对象/数组	为给定的主机创建主机接口。
macros	对象/数组	为给定主机创建的用户宏。
templates	对象/数组	链接到给定主机的模板。 模板必须定义了 `templateid` 属性。

返回值

(对象) 返回一个对象，该对象包含 `hostids` 属性下更新的主机的 id。

示例

添加宏

向两个主机添加两个新的宏。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.massadd",
    "params": {
        "hosts": [
            {
                "hostid": "10160"
            },
            {
                "hostid": "10167"
            }
        ],
        "macros": [
            {
                "macro": "{$TEST1}",
                "value": "MACROTEST1"
            },
            {
                "macro": "{$TEST2}",
                "value": "MACROTEST2",
                "description": "Test description"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostids": [
            "10160",
            "10167"
        ]
    },
    "id": 1
}
```

另请参阅

- 主机更新
- 主机组
- 模板
- 用户宏
- 主机接口

源代码

CHost::massAdd() in *ui/include/classes/api/services/CHost.php*.

更新

描述

`host.update(主机对象/数组) 对象`

该方法允许更新现有主机。

> **Note:**
> 这个方法仅允许 管理员和 超级管理员用户类型。可以在用户角色中撤销调用方法的权限设置。详情参考用户角色

参数

(对象/数组) 待更新的主机属性。

`hostid` 属性必须为每台主机定义，其他属性都是可选的。只有给定的属性将被更新，其他所有属性将保持不变。

但是，请注意，更新主机技术名称也将根据主机的技术名称值更新主机的可见名称 (如果没有给出或为空)。

除了标准主机属性以外，此方法接受如下参数:

参数	类	描述
groups	对象/数组	更换主机所属的主机组。 主机组必须已定义 `groupid` 属性。 所有未在请求中列出的主机组将被解除链接。
interfaces	对象/数组	主机接口用于替换当前主机接口。 所有未在请求中列出的接口将被删除。
tags	对象/数组	主机标签替换当前主机标签。 所有未在请求中列出的标签将被删除。
inventory	object	主机清单属性。
macros	对象/数组	用户宏替换当前用户宏。 所有未在请求中列出的宏将被删除。
templates	对象/数组	模板替换当前链接的模板。所有未在请求中列出的模板只会被解除链接。 模板必须定义了 `templateid` 属性。
templates_clear	对象/数组	模板取消与主机的链接并清除。 模板必须定义了 `templateid` 属性。

Note:
与 Zabbix 前端不同，当 `name` (可见主机名) 与 `host` (技术主机名) 相同时，通过 API 更新 `host` 不会自动更新 `name` 。这两个属性都需要显式更新。

返回值

(`object`) 返回一个对象，该对象包含 `hostids` 属性下更新的主机的 id。

示例

开启主机监控

启用主机监控，即设置主机监控状态为 0。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.update",
    "params": {
        "hostid": "10126",
        "status": 0
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostids": [
            "10126"
        ]
    },
    "id": 1
}
```

取消链接模板

从主机上取消链接并清除两个模板。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.update",
    "params": {
        "hostid": "10126",
        "templates_clear": [
            {
                "templateid": "10124"
            },
            {
                "templateid": "10125"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostids": [
            "10126"
        ]
    },
    "id": 1
}
```

更新主机宏

用两个新的宏替换所有主机宏。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.update",
```

```
    "params": {
        "hostid": "10126",
        "macros": [
            {
                "macro": "{$PASS}",
                "value": "password"
            },
            {
                "macro": "{$DISC}",
                "value": "sda",
                "description": "Updated description"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostids": [
            "10126"
        ]
    },
    "id": 1
}
```

更新主机资产

改变主机资产模式和添加位置。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.update",
    "params": {
        "hostid": "10387",
        "inventory_mode": 0,
        "inventory": {
            "location": "Latvia, Riga"
        }
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostids": [
            "10387"
        ]
    },
    "id": 1
}
```

更新主机标签

用一个新的主机标签替换所有的主机标签。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.update",
    "params": {
        "hostid": "10387",
        "tags": {
            "tag": "OS",
            "value": "CentOS 7"
        }
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostids": [
            "10387"
        ]
    },
    "id": 1
}
```

更新主机加密

更新主机"10590" 仅使用 PSK 加密来连接到 Zabbix 服务器，并更改 PSK 身份和 PSK 密钥。请注意，主机必须预先配置使用 PSK。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "host.update",
    "params": {
        "hostid": "10590",
        "tls_connect": 1,
        "tls_accept": 2,
        "tls_psk_identity": "PSK 002",
        "tls_psk": "e560cb0d918d26d31b4f642181f5f570ad89a390931102e5391d08327ba434e9"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostids": [
            "10590"
        ]
    },
    "id": 1
}
```

另请参阅

- 主机批量添加
- 主机批量更新
- 主机批量移除
- 主机组
- 模板
- 主机宏
- 主机接口

- 主机资产
- 主机标签

源代码

CHost::update() in *ui/include/classes/api/services/CHost.php*.

获取

描述

`host.get(对象参数) 整数/数组`

此方法允许根据指定的参数获取主机。

> **Note:**
> 任何用户类型均可使用此方法。可以在用户角色中撤销调用方法的权限设置。详情参考用户角色

参数

`(object)` 定义所需输出的参数。

该方法支持以下参数。

参数	类型	描述
groupids	string/array	只返回属于给定组的主机。
dserviceids	string/array	只返回与给定发现服务相关的主机。
graphids	string/array	只返回具有给定图表的主机。
hostids	string/array	只返回具有给定主机 ID 的主机。
httptestids	string/array	只返回具有给定网页测试的主机。
interfaceids	string/array	只返回使用给定接口的主机。
itemids	string/array	只返回具有给定监控项的主机。
maintenanceids	string/array	只返回受给定维护影响的主机。
monitored_hosts	flag	只返回被监控的主机。
proxy_hosts	flag	只返回 proxy。
proxyids	string/array	只返回由给定 proxy 监控的主机。
templated_hosts	flag	同时返回主机和模板。
templateids	string/array	只返回链接到给定模板的主机。
triggerids	string/array	只返回具有给定触发器的主机。
with_items	flag	只返回具有监控项的主机。 覆盖 `with_monitored_items` 和 `with_simple_graph_items` 参数。
with_item_prototypes	flag	只返回具有监控项原型的主机。 覆盖 `with_simple_graph_item_prototypes` 参数。
with_simple_graph_item_prototypes	flag	只返回具有启用信息类型为数值监控项原型的主机。
with_graphs	flag	只返回具有图表的主机。
with_graph_prototypes	flag	只返回具有图表原型的主机。
with_httptests	flag	只返回具有网页检查的主机。 覆盖 `with_monitored_httptests` 参数。
with_monitored_httptests	flag	只返回具有启用网页测试的主机。
with_monitored_items	flag	只返回具有启用监控项的主机。 覆盖 `with_simple_graph_items` 参数。
with_monitored_triggers	flag	只返回具有启用触发器的主机。所有在触发器中使用的监控项也必须被启用。
with_simple_graph_items	flag	只返回具有信息类型为数值监控项的主机。

参数	类型	描述
with_triggers	flag	只返回具有触发器的主机。 覆盖 `with_monitored_triggers` 参数。
withProblemsSuppressed	boolean	返回存在被抑制问题的主机。 可能的值： `null` - (默认) 所有主机； `true` - 只有存在被抑制问题的主机； `false` - 只有存在未被抑制问题的主机。
evaltype	integer	标签搜索的规则。 可能的值： 0 - (默认) And/Or； 2 - Or。
severities	integer/array	只返回具有给定严重性问题的主机。仅当问题对象是触发器时适用。
tags	object/array	只返回具有给定标签的主机。根据运算符的值，对标签进行精确匹配或对标签值进行大小写敏感或不敏感的搜索。 格式：`[{"tag": "<tag>", "value": "<value>", "operator": "<operator>"}, ...]`。 空数组返回所有主机。 可能的运算符值： 0 - (默认) 包含； 1 - 等于； 2 - 不像； 3 - 不等于 4 - 存在； 5 - 不存在。
inheritedTags	boolean	返回在其所有链接模板中也具有给定 `tags` 的主机。默认值： 可能的值： `true` - 链接模板也必须具有给定的 `tags`； `false` - (默认) 忽略链接模板标签。
selectDiscoveries	query	返回具有主机低级发现规则的discoveries属性。 支持 `count`。
selectDiscoveryRule	query	返回具有创建主机的低级发现规则的discoveryRule属性（来自 VMware 监控中的主机原型）。
selectGraphs	query	返回具有主机图表的graphs属性。 支持 `count`。
selectGroups	query	返回主机所属的groups属性的主机组数据。

参数	类型	描述
selectHostDiscovery	query	返回一个 `hostDiscovery` 属性，其中包含主机发现对象数据。 主机发现对象将发现的主机链接到主机原型或将主机原型链接到 LLD 规则，并具有以下属性： `host` - *(字符串)* 主机原型的主机； `hostid` - *(字符串)* 发现的主机或主机原型的 ID； `parent_hostid` - *(字符串)* 从创建主机的主机原型的 ID； `parent_itemid` - *(字符串)* 创建发现主机的 LLD 规则的 ID； `lastcheck` - *(时间戳)* 最后发现主机的时间； `ts_delete` - *(时间戳)* 将删除不再发现的主机的时间。
selectHttpTests	query	返回具有主机 Web 场景的httpTests属性。 支持 `count`。
selectInterfaces	query	返回具有主机接口的interfaces属性。 支持 `count`。
selectInventory	query	返回具有主机清单数据的inventory属性。
selectItems	query	返回具有主机监控项的items属性。 支持 `count`。
selectMacros	query	返回具有主机宏的macros属性。
selectParentTemplates	query	返回具有主机链接的模板的parentTemplates属性。 支持 `count`。
selectDashboards	query	返回具有dashboards属性。 支持 `count`。
selectTags	query	返回具有主机标签的tags属性。
selectInheritedTags	query	返回具有在所有链接到主机的模板上的inheritedTags属性的标签。
selectTriggers	query	返回具有主机触发器的triggers属性。 支持 `count`。
selectValueMaps	query	返回具有主机值映射的valuemaps属性。
filter	object	只返回完全匹配给定过滤的结果。 接受一个数组，其中键是属性名称，值可以是单个值或要匹配的值数组。 允许按接口属性过滤。不适用于 `text` 字段。

参数	类型	描述
limitSelects	integer	限制子选择返回的记录数。 适用于以下子选择： `selectParentTemplates` - 结果将按 `host` 排序； `selectInterfaces`； `selectItems` - 按 `name` 排序； `selectDiscoveries` - 按 `name` 排序； `selectTriggers` - 按 `description` 排序； `selectGraphs` - 按 `name` 排序； `selectDashboards` - 按 `name` 排序。
search	object	返回与给定模式匹配的结果（不区分大小写）。 接受一个数组，其中键是属性名称，值是要搜索的字符串。如果没有给定其他选项，这将执行 `LIKE "%...%"` 搜索。
searchInventory	object	允许按接口属性搜索。仅适用于 `string` 和 `text` 字段。 只返回清单数据与给定通配符搜索匹配的主机。 此参数受 `search` 的相同附加参数影响。
sortfield	string/array	根据给定属性对结果进行排序。 可能的值有：`hostid`, `host`, `name`, `status`。
countOutput	boolean	这些参数对所有 `get` 方法都是通用的，详细描述在参考注释中。
editable	boolean	
excludeSearch	boolean	
limit	integer	
output	query	
preservekeys	boolean	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

（整数/数组）返回其中之一:

- 一组对象;
- 如果使用了 `countOutput` 参数，则返回获取的对象数量。

示例

按名称检索数据

获取所有关于"Zabbix server" 和"Linux server" 两个主机的数据。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.get",
    "params": {
        "filter": {
            "host": [
```

```
                "Zabbix server",
                "Linux server"
            ]
        }
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "maintenances": [],
            "hostid": "10160",
            "proxy_hostid": "0",
            "host": "Zabbix server",
            "status": "0",
            "lastaccess": "0",
            "ipmi_authtype": "-1",
            "ipmi_privilege": "2",
            "ipmi_username": "",
            "ipmi_password": "",
            "ipmi_disable_until": "0",
            "snmp_disable_until": "0",
            "maintenanceid": "0",
            "maintenance_status": "0",
            "maintenance_type": "0",
            "maintenance_from": "0",
            "name": "Zabbix server",
            "description": "The Zabbix monitoring server.",
            "tls_connect": "1",
            "tls_accept": "1",
            "tls_issuer": "",
            "tls_subject": ""
        },
        {
            "maintenances": [],
            "hostid": "10167",
            "proxy_hostid": "0",
            "host": "Linux server",
            "status": "0",
            "lastaccess": "0",
            "ipmi_authtype": "-1",
            "ipmi_privilege": "2",
            "ipmi_username": "",
            "ipmi_password": "",
            "ipmi_disable_until": "0",
            "snmp_disable_until": "0",
            "maintenanceid": "0",
            "maintenance_status": "0",
            "maintenance_type": "0",
            "maintenance_from": "0",
            "name": "Linux server",
            "description": "",
            "tls_connect": "1",
            "tls_accept": "1",
            "tls_issuer": "",
            "tls_subject": ""
        }
    ],
```

```
    "id": 1
}
```

获取主机组

获取主机"Zabbix server" 所属的主机组，并不检索主机本身的详细信息。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.get",
    "params": {
        "output": ["hostid"],
        "selectGroups": "extend",
        "filter": {
            "host": [
                "Zabbix server"
            ]
        }
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 2
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "hostid": "10085",
            "groups": [
                {
                    "groupid": "2",
                    "name": "Linux servers",
                    "internal": "0",
                    "flags": "0"
                },
                {
                    "groupid": "4",
                    "name": "Zabbix servers",
                    "internal": "0",
                    "flags": "0"
                }
            ]
        }
    ],
    "id": 2
}
```

获取关联的模板

获取主机"10084" 关联的模板的 ID 和名称。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.get",
    "params": {
        "output": ["hostid"],
        "selectParentTemplates": [
            "templateid",
            "name"
        ],
        "hostids": "10084"
```

```
    },
    "id": 1,
    "auth": "70785d2b494a7302309b48afcdb3a401"
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "hostid": "10084",
            "parentTemplates": [
                {
                    "name": "Linux",
                    "templateid": "10001"
                },
                {
                    "name": "Zabbix Server",
                    "templateid": "10047"
                }
            ]
        }
    ],
    "id": 1
}
```

根据模板检索主机

检索与"10001" (*Zabbix agent* 的 *Linux*) 模板相关联的主机。

请求 :

```
{
    "jsonrpc": "2.0",
    "method": "host.get",
    "params": {
        "output": ["hostid", "name"],
        "templateids": "10001"
    },
    "auth": "70785d2b494a7302309b48afcdb3a401",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "templateid": "10001",
            "hosts": [
                {
                    "hostid": "10084",
                    "name": "Zabbix server"
                },
                {
                    "hostid": "10603",
                    "name": "Host 1"
                },
                {
                    "hostid": "10604",
                    "name": "Host 2"
                }
            ]
        }
```

```
    ],
    "id": 1
}
```

按主机资产数据进行搜索

检索主机资产 “OS” 字段中包含 “Linux” 的主机。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.get",
    "params": {
        "output": [
            "host"
        ],
        "selectInventory": [
            "os"
        ],
        "searchInventory": {
            "os": "Linux"
        }
    },
    "id": 2,
    "auth": "7f9e00124c75e8f25facd5c093f3e9a0"
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "hostid": "10084",
            "host": "Zabbix server",
            "inventory": {
                "os": "Linux Ubuntu"
            }
        },
        {
            "hostid": "10107",
            "host": "Linux server",
            "inventory": {
                "os": "Linux Mint"
            }
        }
    ],
    "id": 1
}
```

按主机标记搜索

检索标记为 “主机名” 等于 “Linux 服务器” 的主机。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.get",
    "params": {
        "output": ["hostid"],
        "selectTags": "extend",
        "evaltype": 0,
        "tags": [
            {
                "tag": "Host name",
```

```
                "value": "Linux server",
                "operator": 1
            }
        ]
    },
    "auth": "7f9e00124c75e8f25facd5c093f3e9a0",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "hostid": "10085",
            "tags": [
                {
                    "tag": "Host name",
                    "value": "Linux server"
                },
                {
                    "tag": "OS",
                    "value": "RHEL 7"
                }
            ]
        }
    ],
    "id": 1
}
```

检索不仅在主机级而且在其链接的父模板中具有这些标记的主机。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.get",
    "params": {
        "output": ["name"],
        "tags": [{"tag": "A", "value": "1", "operator": "0"}],
        "inheritedTags": true
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "hostid": "10623",
            "name": "PC room 1"
        },
        {
            "hostid": "10601",
            "name": "Office"
        }
    ],
    "id": 1
}
```

搜索主机标签和模板标签

检索带有标记的主机以及链接到父模板的所有标记。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.get",
    "params": {
        "output": ["name"],
        "hostids": 10502,
        "selectTags": ["tag", "value"],
        "selectInheritedTags": ["tag", "value"]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "hostid": "10502",
            "name": "Desktop",
            "tags": [
                {
                    "tag": "A",
                    "value": "1"
                }
            ],
            "inheritedTags": [
                {
                    "tag": "B",
                    "value": "2"
                }
            ]
        }
    ],
    "id": 1
}
```

根据问题严重程度搜索主机

检索有 “灾难” 问题的主机。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.get",
    "params": {
        "output": ["name"],
        "severities": 5
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "hostid": "10160",
            "name": "Zabbix server"
```

```
        }
    ],
    "id": 1
}
```

检索具有 “一般严重” 和 “严重” 问题的主机。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.get",
    "params": {
        "output": ["name"],
        "severities": [3, 4]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "hostid": "20170",
            "name": "Database"
        },
        {
            "hostid": "20183",
            "name": "workstation"
        }
    ],
    "id": 1
}
```

另请参阅

- 主机组
- 模板
- 用户宏
- 主机接口

源代码

CHost::get() in *ui/include/classes/api/services/CHost.php*.

配置批量删除

描述

`host.massremove(对象参数) 对象`

该方法允许从多个主机中移除相关对象。

> **Note:**
> 这个方法仅允许 管理员和 超级管理员用户类型。可以在用户角色中撤销调用方法的权限设置。详情参考用户角色

参数

（对象）参数包含要更新的主机 id 和应该删除的对象。

参数	类型	描述
hostids (必填)	字符串/数组	待更新的主机 id。
groupids	字符串/数组	从主机组中移除指定主机。

参数	类型	描述
interfaces	对象/数组	要从给定主机中移除的主机接口。 主机接口对象必须定义了 ip 、dns 和 port 属性。
macros	字符串/数组	要从给定主机中删除的用户宏。
templateids	字符串/数组	要从给定主机断开链接的模板。
templateids_clear	字符串/数组	要从给定主机解除链接和清除的模板。

返回值

（对象）返回一个对象，该对象包含 hostids 属性下更新的主机的 id。

示例

删除模板的链接

将模板与两台主机的链接解除，并删除所有模板实体。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "host.massremove",
    "params": {
        "hostids": ["69665", "69666"],
        "templateids_clear": "325"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostids": [
            "69665",
            "69666"
        ]
    },
    "id": 1
}
```

另请参阅

- 主机更新
- 用户宏
- 主机接口

源代码

CHost::massRemove() in *ui/include/classes/api/services/CHost.php*.

主机原型

该类被设计用来处理主机原型。

对象引用:

- Host prototype
- Host prototype inventory
- Group link
- Group prototype

可用方法:

- hostprototype.create - 创建新的主机原型

- hostprototype.delete - 删除主机原型
- hostprototype.get - 获取主机原型
- hostprototype.update - 更新主机原型

> 主机原型对象

以下对象与主机原型 API 直接相关。

主机原型

主机原型对象具有以下属性。

属性	类型	描述
hostid	string	*(readonly)* 主机原型的 ID。
host (required)	string	主机原型的技术名称。
name	string	可见名称的主机原型。 默认值：`host` 属性值。
status	integer	主机原型的状态。 可能的值是： 0 - *(default)* 监控中的主机； 1 - 未监控的主机。
inventory_mode	integer	主机库存填充模式。 可能的值是： -1 - *(default)* 禁用； 0 - 手动； 1 - 自动。
templateid	string	*(readonly)* 父模板主机原型的 ID。
discover	integer	主机原型的发现状态。 可能的值是： 0 - *(default)* 将发现新主机； 1 - 不会发现新主机，并将现有主机标记为丢失。
custom_interfaces	integer	由主机原型创建的主机的接口来源。 可能的值是： 0 - *(default)* 从父主机继承接口； 1 - 使用主机原型自定义接口。
uuid	string	通用唯一标识符，用于将导入的主机原型链接到已存在的主机原型。仅用于模板上的主机原型。如果没有提供，则自动生成。

注意，对于一些方法（更新，删除），所需/可选参数的组合是不同的。

组链接

组链接对象将主机原型与主机组关联，并具有以下属性。

属性	类型	描述
groupid (required)	string	主机组的 ID。

组原型

组原型对象定义了一个将为发现的主机创建的组，并具有以下属性。

属性	类型	描述
name (required)	string	组原型的名称。

主机原型标签

主机原型标签对象有以下属性：

属性	类型	描述
tag (必选)	字符串	主机原型标签名称。
value	字符串	主机原型标签值。

自定义接口

自定义接口对象与以下属性：

属性	类型	描述
dns	字符串	接口使用的 DNS 名称。 如果直接通过 DNS 连接，则必选，可以包含宏
ip	字符串	接口使用的 IP 如果直接通过 IP 连接，则必选，可以包含宏
main (必选)	整型	该接口是否在主机上用作默认接口，在主机上只能将某种类型的一个接口设置为默认值。 可能的值: 0 - 不是默认; 1 - 默认。
port (必选)	字符串	接口使用的端口号。可以包含用户宏和 LLD 宏。
type (必选)	整型	接口类型。 可能的值: 1 - agent; 2 - SNMP; 3 - IPMI; 4 - JMX。
useip (必选)	整型	是否通过 IP 进行连接。 可能的值有: 0 - 使用主机 DNS 名称进行连接; 1 - 使用此主机接口的 IP 地址进行连接
details	数组	接口的附加对象。 如果接口’类型’是 SNMP 则 必须。

自定义接口详细信息

详细信息对象有以下属性：

属性	类型	描述
version (必选)	整型	SNMP 接口版本。 可能的值有: 1 - SNMPv1; 2 - SNMPv2c; 3 - SNMPv3
bulk	整型	是否使用批量的 SNMP 请求 可能的值有: 0 - 不使用批量请求; 1 - *(默认)* - 使用批量请求。
community	字符串	SNMP 团体字。仅用于 SNMPv1 和 SNMPv2 接口。
securityname	字符串	SNMPv3 安全名称。仅用于 SNMPv3 的接口
securitylevel	整型	SNMPv3 安全级别。仅用于 SNMPv3 的接口 可能的值有: 0 - *(默认)* - noAuthNoPriv; 1 - authNoPriv; 2 - authPriv。
authpassphrase	字符串	SNMPv3 身份认证密码。仅用于 SNMPv3 的接口。
privpassphrase	字符串	SNMPv3 私有密码，仅用于 SNMPv3 的接口。
authprotocol	整型	SNMPv3 认证协议。仅用于 SNMPv3 的接口。 可能的值有: 0 - *(默认)* - MD5; 1 - SHA1; 2 - SHA224; 3 - SHA256; 4 - SHA384; 5 - SHA512。
privprotocol	整型	SNMPv3 隐私协议。仅用于 SNMPv3 的接口。 可能的值有: 0 - *(默认)* - DES; 1 - AES128; 2 - AES192; 3 - AES256; 4 - AES192C; 5 - AES256C。

属性	类型	描述
contextname	字符串	SNMPv3 上下文名称。仅用于 SNMPv3 的接口。

创建

描述

`object hostprototype.create(object/array hostPrototypes)`

此方法允许创建新的主机原型。

Note:
该方法适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。有关详细信息，参阅用户角色。

参数

`(object/array)` 创建的主机原型

除标准的主机原型属性外，该方法接受以下参数:

参数	类型	描述
groupLinks (必选)	数组	主机原型创建的组链接。
ruleid (必选)	字符串	主机原型所属的 LLD 规则的 ID。
groupPrototypes	数组	为主机原型创建的组原型。
macros	对象/数组	为主机原型创建的用户宏。
tags	对象/数组	主机原型标签。
interfaces	对象/数组	主机原型自定义接口。
templates	对象/数组	连接到主机原型的模板。 模板必须定义 `templateid` 属性。

返回值

`(object)` 在 `hostids` 属性中返回包含创建主机原型的 ID 对象，返回的 ID 顺序和传入的主机原型的顺序一致。

示例

`######` 创建主机原型

在 LLD 规则 “23542” 上使用组原型”{#HV.NAME}”、标签对”Datacenter”: ”{#DATACENTER.NAME}” 和自定义团体字为 {$SNMP_COMMUNITY} 的 SNMPv2 接口 127.0.0.1:161, 创建主机原型”{#VM.NAME}” 并将其链接到主机组”2”。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostprototype.create",
    "params": {
        "host": "{#VM.NAME}",
        "ruleid": "23542",
        "custom_interfaces": "1",
        "groupLinks": [
            {
                "groupid": "2"
            }
        ],
        "groupPrototypes": [
            {
                "name": "{#HV.NAME}"
            }
        ],
```

```
        "tags": [
            {
                "tag": "Datacenter",
                "value": "{#DATACENTER.NAME}"
            }
        ],
        "interfaces": [
            {
                "main": "1",
                "type": "2",
                "useip": "1",
                "ip": "127.0.0.1",
                "dns": "",
                "port": "161",
                "details": {
                    "version": "2",
                    "bulk": "1",
                    "community": "{$SNMP_COMMUNITY}"
                }
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostids": [
            "10103"
        ]
    },
    "id": 1
}
```

参考

- Group link
- Group prototype
- Host prototype tag
- Custom interface
- User macro

来源

CHostPrototype::create() in *ui/include/classes/api/services/CHostPrototype.php*。

删除

描述

`object hostprototype.delete(array hostPrototypeIds)`

该方法允许删除主机原型

> **Note:**
> 该方法适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。有关详细信息，参阅用户角色。

参数

(array) 要删除主机原型的 ID。

返回值

(object) 在 hostids 属性中返回已删除主机原型的 ID 对象。

示例

删除多个主机原型

删除两个主机原型:

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostprototype.delete",
    "params": [
        "10103",
        "10105"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostids": [
            "10103",
            "10105"
        ]
    },
    "id": 1
}
```

来源

CHostPrototype::delete() in *ui/include/classes/api/services/CHostPrototype.php*。

更新

描述

`object hostprototype.update(object/array hostPrototypes)`

该方法运行更新存在的主机原型

> **Note:**
> 该方法适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。有关详细信息，参阅用户角色。

参数

(object/array) 要更新的主机原型属性。

必须为每个主机原型定义 hostid 属性，所有其他属性都是可选的。只有选中的属性将被更新，所有其他的属性将保持不变。除标准主机原型属性外，该方法还接受以下参数。

参数	类型	描述
groupLinks	队列	替换当前主机原型上的组链接的组链接。
groupPrototypes	数组	替换主机原型上存在的组原型的组原型。
macros	对象/数组	替换当前用户宏的用户宏。 所有未在请求列表中宏将被删除。

参数	类型	描述
tags	对象/数组	替换当前标签的主机原型标签。 所有未在请求列表中的标签将被删除。
interfaces	对象/数组	替换当前接口的主机原型自定义接口。 自定义接口对象包含所有参数。 所有未在请求列表中的接口将被删除
templates	对象/数组	替换当前连接模板中的模板。 模板中的 `templateid` 属性必须被定义。

返回值

(object) 在 `hostids` 属性中返回包含已更新主机原型 ID 的对象。

示例

禁用主机原型

禁用主机原型可以设置其状态为 1。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostprototype.update",
    "params": {
        "hostid": "10092",
        "status": 1
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostids": [
            "10092"
        ]
    },
    "id": 1
}
```

更新主机原型标签

替换新的主机原型标签

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostprototype.update",
    "params": {
        "hostid": "10092",
```

```
        "tags": [
            {
                "tag": "Datacenter",
                "value": "{#DATACENTER.NAME}"
            },
            {
                "tag": "Instance type",
                "value": "{#INSTANCE_TYPE}"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostids": [
            "10092"
        ]
    },
    "id": 1
}
```

更新主机原型自定义接口

用主机原型自定义接口替换继承的接口。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostprototype.update",
    "params": {
        "hostid": "10092",
        "custom_interfaces": "1",
        "interfaces": [
            {
                "main": "1",
                "type": "2",
                "useip": "1",
                "ip": "127.0.0.1",
                "dns": "",
                "port": "161",
                "details": {
                    "version": "2",
                    "bulk": "1",
                    "community": "{$SNMP_COMMUNITY}"
                }
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostids": [
            "10092"
```

```
        ]
    },
    "id": 1
}
```

参考

- Group link
- Group prototype
- Host prototype tag
- Custom interface
- User macro

来源

CHostPrototype::update() in *ui/include/classes/api/services/CHostPrototype.php*.

查询

描述

`integer/array hostprototype.get(object parameters)`

该方法允许根据给定的参数获取主机原型记录。

> **Note:**
> 该方法适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。有关详细信息，参阅用户角色。

参数

`(object)` 定义所需输出的参数。

该方法支持以下参数。

参数	类型	描述
hostids	string/array	仅返回给定 ID 的主机原型。
discoveryids	string/array	仅返回属于给定 LLD 规则的主机原型。
inherited	boolean	如果设置为 `true`，则仅返回从模板继承的监控项。
selectDiscoveryRule	query	返回一个带有discoveryRule 属性的 LLD 规则，该主机原型属于该规则。
selectInterfaces	query	返回一个带有interfaces 属性的主机原型自定义接口。
selectGroupLinks	query	返回一个带有groupLinks 属性的主机原型的组链接。
selectGroupPrototypes	query	返回一个带有groupPrototypes 属性的主机原型的组原型。
selectMacros	query	返回一个带有macros 属性的主机原型宏。
selectParentHost	query	返回一个带有parentHost 属性的父主机，该主机原型属于该主机。
selectTags	query	返回一个带有tags 属性的主机原型标签。
selectTemplates	query	返回一个带有与主机原型链接的templates 属性。 支持 `count`。
sortfield	string/array	根据给定的属性对结果进行排序。 可能的值有：`hostid`，`host`，`name` 和 `status`。
countOutput	boolean	这些参数对所有 `get` 方法都是通用的，详细描述在参考说明 中。
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

(integer/array) 返回以下:

- 一组对象;
- 如果 countOutput 参数被使用，返回对象的数量。

示例

从 LLD 规则中获取主机原型

从 LLD 规则中获取所有主机原型，组链接，组原型和标签

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostprototype.get",
    "params": {
        "output": "extend",
        "selectInterfaces": "extend",
        "selectGroupLinks": "extend",
        "selectGroupPrototypes": "extend",
        "selectTags": "extend",
        "discoveryids": "23554"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "hostid": "10092",
            "host": "{#HV.UUID}",
            "name": "{#HV.UUID}",
            "status": "0",
            "templateid": "0",
            "discover": "0",
            "custom_interfaces": "1",
            "inventory_mode": "-1",
            "groupLinks": [
                {
                    "group_prototypeid": "4",
                    "hostid": "10092",
                    "groupid": "7",
                    "templateid": "0"
                }
            ],
            "groupPrototypes": [
                {
                    "group_prototypeid": "7",
                    "hostid": "10092",
                    "name": "{#CLUSTER.NAME}",
                    "templateid": "0"
                }
            ],
            "tags": [
                {
                    "tag": "Datacenter",
                    "value": "{#DATACENTER.NAME}"
                },
                {
                    "tag": "Instance type",
```

```
                    "value": "{#INSTANCE_TYPE}"
                }
            ],
            "interfaces": [
                {
                    "main": "1",
                    "type": "2",
                    "useip": "1",
                    "ip": "127.0.0.1",
                    "dns": "",
                    "port": "161",
                    "details": {
                        "version": "2",
                        "bulk": "1",
                        "community": "{$SNMP_COMMUNITY}"
                    }
                }
            ]
        }
    ],
    "id": 1
}
```

参考

- Group link
- Group prototype
- User macro

来源

CHostPrototype::get() in *ui/include/classes/api/services/CHostPrototype.php*。

主机接口

这个类是设计用于处理主机接口.

对象引用:

- Host interface

可用方法:

- hostinterface.create - 创建新的主机接口
- hostinterface.delete - 删除主机接口
- hostinterface.get - 获取主机接口
- hostinterface.massadd - 批量添加主机接口
- hostinterface.massremove - 批量删除主机接口
- hostinterface.replacehostinterfaces - 替换主机接口
- hostinterface.update - 更新主机接口

> 主机接口对象

以下对象与 主机接口 API 直接相关。

主机接口

主机接口对象具有以下属性。

注意，IP 和 DNS 都是必需的。如果您不想使用 DNS，请将其设置为空字符串.

属性	类型	描述
available	整型	*(只读)* 主机接口的可用性。 可能的值有: 0 - *(默认)* 未知; 1 - 可用; 2 - 不可用。
details	数组	接口的附加对象。 如果接口'type' 是SNMP，则必选
disable_until	时间戳	*(只读)* 不可用主机接口的下一次轮询时间。
dns (必选)	字符串	接口使用的 DNS 名称 如果通过 IP 直接连接，则可以为空。
error	字符串	*(只读)* 主机接口不可用的错误文本。
errors_from	时间戳	*(只读)* 主机接口不可用开始时间。
hostid (必选)	字符串	接口所属的主机 ID。
interfaceid	字符串	*(只读)* 接口的 ID。
ip (必选)	字符串	接口使用的 IP 地址。 如果使用 DNS 域名连接，可以设置为空。
main (必选)	整型	该接口是否在主机上用作默认的接口，在主机上只能将某种类型的一个接口设置为默认值。 可能的值有: 0 - 不是默认; 1 - 默认。
port (必选)	字符串	接口使用的端口号，可以包含用户宏。
type (必选)	整型	接口类型。 可能的值有: 1 - agent; 2 - SNMP; 3 - IPMI; 4 - JMX。
useip (必选)	整型	是否通过 IP 进行连接。 可能的值有: 0 - 使用主机 DNS 进行连接; 1 - 使用主机接口的主机 IP 进行连接。

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

详细信息标签

详细信息对象具有以下属性。

属性	类型	描述
version (必选)	整型	SNMP 接口版本。 可能的值有: 1 - SNMPv1; 2 - SNMPv2c; 3 - SNMPv3
bulk	整型	是否使用批量的 SNMP 请求. 可能的值有: 0 - 不使用批量请求; 1 - (默认) - 使用批量请求。
community	字符串	SNMP 团体字 (必选)。仅在 SNMPv1 和 SNMPv2 接口中使用。
securityname	字符串	SNMPv3 安全名称。仅在 SNMPv3 接口中使用。
securitylevel	字符串	SNMPv3 安全级别。仅在 SNMPv3 接口中使用。 可能的值有: 0 - (默认) - noAuthNoPriv; 1 - authNoPriv; 2 - authPriv.
authpassphrase	字符串	SNMPv3 身份验证密码。仅在 SNMPv3 接口中使用。
privpassphrase	string	SNMPv3 隐私密码。仅在 SNMPv3 接口中使用。
authprotocol	integer	SNMPv3 身份认证协议。仅在 SNMPv3 接口中使用。 可能的值有: 0 - (默认) - MD5; 1 - SHA1; 2 - SHA224; 3 - SHA256; 4 - SHA384; 5 - SHA512.

属性	类型	描述
privprotocol	整型	SNMPv3 隐私协议。仅在 SNMPv3 接口中使用。 可能的值有: 0 - (默认) - DES; 1 - AES128; 2 - AES192; 3 - AES256; 4 - AES192C; 5 - AES256C.
contextname	string	SNMPv3 上下文名称。仅在 SNMPv3 接口中使用。

创建

描述

`object hostinterface.create(object/array hostInterfaces)`

该方法允许创建新的主机接口。

> **Note:**
> 该方法适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。有关详细信息，参阅用户角色。

参数

(object/array) 创建主机接口，该方法接受主机接口的标准主机接口属性。

返回值

(object) 在 interfaceids 属性中返回已创建主机接口 ID 对象。返回的 ID 顺序与传入的主机接口顺序保持一致。

示例

创建主机接口

给 ID 为 30052 主机创建辅助 IP 代理接口

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostinterface.create",
    "params": {
        "hostid": "30052",
        "main": "0",
        "type": "1",
        "useip": "1"
        "ip": "127.0.0.1",
        "dns": "",
        "port": "10050",
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "interfaceids": [
            "30062"
        ]
```

```
    },
    "id": 1
}
```

创建 SNMP 详细信息的主机接口

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostinterface.create",
    "params": {
        "hostid": "10456",
        "main": "0",
        "type": "2",
        "useip": "1",
        "ip": "127.0.0.1",
        "dns": "",
        "port": "1601",
        "details": {
            "version": "2",
            "bulk": "1",
            "community": "{$SNMP_COMMUNITY}"
        }
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "interfaceids": [
            "30063"
        ]
    },
    "id": 1
}
```

参考

- hostinterface.massadd
- host.massadd

来源

CHostInterface::create() in *ui/include/classes/api/services/CHostInterface.php*。

删除

描述

`object hostinterface.delete(hostInterfaceIds 数组)`

该方法允许删除主机接口

> **Note:**
> 该方法适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。有关详细信息，参阅用户角色。

参数

要删除主机接口的 （数组）ID。

返回值

(object) 在 `interfaceids` 属性中返回已删除主机接口 ID 对象。

示例

删除一个主机接口

删除 ID 为 30062 的主机接口.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostinterface.delete",
    "params": [
        "30062"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "interfaceids": [
            "30062"
        ]
    },
    "id": 1
}
```

参考

- hostinterface.massremove
- host.massremove

来源

CHostInterface::delete() in *ui/include/classes/api/services/CHostInterface.php*.

批量删除

描述

`object hostinterface.massremove(object parameters)`

该方法允许从给定的主机列表中批量删除主机接口

> **Note:**
> 该方法适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。有关详细信息，参阅用户角色。

参数

`(object)` 参数包含更新的主机的 ID 和被删除的主机接口。

参数	类型	描述
hostids (必选)	字符串/数组	被更新的主机 ID
interfaces (必选)	对象/数组	要删除的给定主机接口 主机接口对象必须定义 ip, dns 和 port 属性。

返回值

`(object)` 在 `interfaceids` 属性中返回已删除主机接口 ID 的对象。

示例

删除接口

从两台主机中删除”127.0.0.1” SNMP 接口。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostinterface.massremove",
    "params": {
        "hostids": [
            "30050",
            "30052"
        ],
        "interfaces": {
            "dns": "",
            "ip": "127.0.0.1",
            "port": "161"
        }
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "interfaceids": [
            "30069",
            "30070"
        ]
    },
    "id": 1
}
```

参考

- hostinterface.delete
- host.massremove

来源

CHostInterface::massRemove() in *ui/include/classes/api/services/CHostInterface.php*。

批量添加

描述

`object hostinterface.massadd(object parameters)`

该方法允许同时向多个主机添加主机接口。

> **Note:**
> 该方法适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。有关详细信息，参阅用户角色。

参数

`(object)` 包含要在给定主机上创建的主机接口的参数

该方法接受以下参数：

参数	类型	描述
hosts (必选)	对象/数组	更新的主机。 主机必须已定义 `hostid` 属性。
interfaces (必选)	对象/数组	主机上创建主机接口。

返回值

(object) 在 iterfaceids 属性中返回包含已创建主机接口 ID 的对象。

示例

创建接口

在两个主机上创建接口

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostinterface.massadd",
    "params": {
        "hosts": [
            {
                "hostid": "30050"
            },
            {
                "hostid": "30052"
            }
        ],
        "interfaces": {
            "dns": "",
            "ip": "127.0.0.1",
            "main": 0,
            "port": "10050",
            "type": 1,
            "useip": 1
        }
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "interfaceids": [
            "30069",
            "30070"
        ]
    },
    "id": 1
}
```

参考

- 主机接口创建
- 主机接口批量添加
- 主机

来源

CHostInterface::massAdd() in *ui/include/classes/api/services/CHostInterface.php*.

更新

描述

`object hostinterface.update(object/array hostInterfaces)`

该方法允许更新已存在的主机接口

> **Note:**
> 该方法适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。有关详细信息，参阅用户角色。

参数

(object/array) 要更新的主机接口属性

必须为每个主机接口定义 interfaceid 属性，所有其他属性都是可选的。只有给定的属性将被更新，所有其他属性将保持不变。

返回值

(object) 在 interfaceids 属性中返回已更新主机接口 ID 的对象。

示例

更新主机接口端口

更新主机接口的端口

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostinterface.update",
    "params": {
        "interfaceid": "30048",
        "port": "30050"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "interfaceids": [
            "30048"
        ]
    },
    "id": 1
}
```

来源

CHostInterface::update() in *ui/include/classes/api/services/CHostInterface.php*。

替换

描述

`object hostinterface.replacehostinterfaces(object parameters)`

此方法允许给指定主机替换所有主机接口。

> **Note:**
> 该方法适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。有关详细信息，参阅用户角色。

参数

(object) 包含要更新的主机 ID 和新主机接口的参数。

参数	类型	描述
hostid (必选)	字符串	要更新的主机 ID
interfaces (必选)	对象/数组	替换当前主机接口的主机接口。

返回值

(object) 在 interfaceids 属性中返回已创建主机接口 ID 的对象。

示例

替换主机接口

用单个代理接口替换所有主机接口。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostinterface.replacehostinterfaces",
    "params": {
        "hostid": "30052",
        "interfaces": {
            "dns": "",
            "ip": "127.0.0.1",
            "main": 1,
            "port": "10050",
            "type": 1,
            "useip": 1
        }
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "interfaceids": [
            "30081"
        ]
    },
    "id": 1
}
```

参考

- host.update
- host.massupdate

来源

CHostInterface::replaceHostInterfaces() in *ui/include/classes/api/services/CHostInterface.php*。

获取

描述

`integer/array hostinterface.get(object parameters)`

此方法允许获取给定参数的主机接口记录。

该方法适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。有关详细信息，参阅用户角色。:::

参数

`(object)` 定义所需输出的参数。

该方法支持以下参数。

参数	类型	描述
hostids	string/array	只返回由指定主机使用的主机接口。
interfaceids	string/array	只返回具有给定 ID 的主机接口。
itemids	string/array	只返回由指定监控项使用的主机接口。
triggerids	string/array	只返回由给定触发器中的监控项使用的主机接口。
selectItems	query	返回一个监控项属性，包含使用该接口的监控项。 支持 `count`。

参数	类型	描述
selectHosts	query	返回一个主机属性，包含使用该接口的主机数组。
limitSelects	integer	限制子查询返回的记录数。 适用于以下子查询： `selectItems`。
sortfield	string/array	根据给定属性对结果进行排序。 可能的值包括：`interfaceid`, `dns`, `ip`。
countOutput	boolean	这些参数对于所有获取方法都是通用的，在参考说明页面中有详细描述。
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

`(integer/array)` 返回:

- 一组对象;
- 如果设置了 `countOutput` 参数, 则返回获取到的对象数量。

示例

获取主机接口

获取 ID 为 30057 的主机使用的接口所有数据。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostinterface.get",
    "params": {
        "output": "extend",
        "hostids": "30057"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "interfaceid": "50039",
            "hostid": "30057",
            "main": "1",
            "type": "1",
            "useip": "1",
            "ip": "::1",
            "dns": "",
            "port": "10050",
            "available": "0",
            "error": "",
            "errors_from": "0",
            "disable_until": "0",
            "details": []
        },
```

```
        {
            "interfaceid": "55082",
            "hostid": "30057",
            "main": "0",
            "type": "1",
            "useip": "1",
            "ip": "127.0.0.1",
            "dns": "",
            "port": "10051",
            "available": "0",
            "error": "",
            "errors_from": "0",
            "disable_until": "0",
            "details": {
                "version": "2",
                "bulk": "0",
                "community": "{$SNMP_COMMUNITY}"
            }
        }
    ],
    "id": 1
}
```


参考

- 主机
- 监控项

来源

CHostInterface::get() in *ui/include/classes/api/services/CHostInterface.php*.

主机组

该类用于管理主机组。

对象引用:

- 主机组

可用的方法:

- hostgroup.create - 新建主机组
- hostgroup.delete - 删除主机组
- hostgroup.get - 获取主机组
- hostgroup.massadd - 添加相关对象到主机组
- hostgroup.massremove - 从主机组删除相关对象
- hostgroup.massupdate - 从主机组替换或删除相关对象
- hostgroup.update - 更新主机组

> 主机组对象

以下对象是和主机组直接相关的 API。

主机组

主机组对象具有以下属性。

属性	类型	描述
groupid	string	*(readonly)* 主机组的 ID。
name (required)	string	主机组的名称。

属性	类型	描述
flags	integer	*(readonly)* 主机组的来源。 可能的值： 0 - 普通主机组； 4 - 发现的主机组。
internal	integer	*(readonly)* 该组是否被系统内部使用。系统内部使用的组不能被删除。 可能的值： 0 - *(default)* 非内部； 1 - 内部。
uuid	string	通用唯一标识符，用于将导入的主机组链接到已存在的主机组。如果没有提供，则自动生成。

注意，对于一些方法（更新，删除），所需/可选参数的组合是不同的。

创建

描述

`hostgroup.create(hostGroup 对象/数组) 对象`

通过该方式可以创建新的主机组。

> **Note:**
> 这个方法仅允许 超级管理员用户类型。可以在用户角色中撤销调用方法的权限设置。详情参考用户角色

参数

创建 （对象/数组）主机组. 该方法接受具有标准主机组属性 的主机组。

返回值

（对象）在 `groupids` 属性下返回包含已创建主机组 ID 的对象. 返回主机组 ID 的顺序与传入的主机组顺序一致.

示例

创建一个主机组

创建名为”Linux servers” 的主机组。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostgroup.create",
    "params": {
        "name": "Linux servers"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "groupids": [
            "107819"
        ]
    },
    "id": 1
}
```

源代码

CHostGroup::create() in *ui/include/classes/api/services/CHostGroup.php*.

删除

描述

`hostgroup.delete(hostGroupId 数组) 对象`

此方法允许删除主机组。

如果主机组有以下情况，则不能被删除:

- 包含仅属于该主机组的主机;
- 被标记为内部;
- 被主机原型引用;
- 在全局脚本中使用;
- 在相关条件下使用。

> **Note:**
> 这个方法仅允许 管理员和 超级管理员用户类型。可以在用户角色中撤销调用方法的权限设置。详情参考用户角色

参数

要删除主机组的 （数组）ID。

返回值

（对象）在 `groupids` 属性中返回包含已删主机组 ID 的对象。

示例

删除多个主机组

删除两个主机组。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostgroup.delete",
    "params": [
        "107824",
        "107825"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "groupids": [
            "107824",
            "107825"
        ]
    },
    "id": 1
}
```

源代码

CHostGroup::delete() in *ui/include/classes/api/services/CHostGroup.php*.

批量删除

描述

`hostgroup.massremove(对象参数) 对象`

通过该方法可以将相关对象从多个主机组中移除。

Note:
这个方法仅允许 管理员和 超级管理员用户类型。可以在用户角色中撤销调用方法的权限设置。详情参考用户角色

参数

（对象）参数包含要更新的主机组 id 和需要删除的对象。

参数	类	描述
groupids (必填)	字符串/数组	需要更新的主机组 id。
hostids	字符串/数组	待从所有主机组中移除的主机。
templateids	字符串/数组	待从所有主机组移除模板。

返回值

（对象）返回一个对象，该对象包含 `groupids` 属性下的已更新主机组的 id。

示例

从主机组中删除主机

从给定的主机组中删除两个主机。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostgroup.massremove",
    "params": {
        "groupids": [
            "5",
            "6"
        ],
        "hostids": [
            "30050",
            "30001"
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "groupids": [
            "5",
            "6"
        ]
    },
    "id": 1
}
```

源代码

CHostGroup::massRemove() in *ui/include/classes/api/services/CHostGroup.php*.

批量更新

描述

`hostgroup.massupdate(对象参数) 对象`

通过该方式，可以将多个主机组中的主机和模板替换为指定的主机和模板。

Note:
这个方法仅允许 管理员和 超级管理员用户类型。可以在用户角色中撤销调用方法的权限设置。详情参考用户角色

参数

（对象）参数包含要更新的主机组 id 和应该更新的对象。

参数	类	描述
groups (必填)	对象/数组	待更新的主机组。 主机组必须定义了 `groupid` 属性。
hosts (必填)	对象/数组	主机替换给定主机组上的当前主机。 除上述主机外，其他所有主机将被排除在主机组之外。 发现的主机不受影响。 主机必须定义了 `hostid` 属性。
templates (必填)	对象/数组	模板用于替换给定主机组上的当前模板。 除以上提到的模板外，其他所有模板将被排除在主机组之外。 模板必须定义了 `templateid` 属性。

返回值

（对象）返回一个对象，该对象包含 `groupids` 属性下的已更新主机组的 id。

示例

替换主机组中的主机

将主机组中的所有主机替换为 hostid 为 30050 的主机，并解除主机组中所有模板的链接。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostgroup.massupdate",
    "params": {
        "groups": [
            {
                "groupid": "6"
            }
        ],
        "hosts": [
            {
                "hostid": "30050"
            }
        ],
        "templates": []
    },
    "auth": "f223adf833b2bf2ff38574a67bba6372",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "groupids": [
            "6",
        ]
    },
    "id": 1
}
```

另请参阅

- 主机组更新
- 主机组批量添加
- 主机
- 模板

源代码

CHostGroup::massUpdate() in *ui/include/classes/api/services/CHostGroup.php*.

批量添加

描述

`hostgroup.massadd(对象参数)` 对象

该方法允许在给定的主机组中同时添加多个相关对象。

> **Note:**
> 这个方法仅允许 管理员和 超级管理员用户类型。可以在用户角色中撤销调用方法的权限设置。详情参考用户角色

参数

(对象)参数包含要更新的主机组 id 和要添加到所有主机组的对象。

该方法接受以下参数。

参数	类	描述
groups (必填)	对象/数组	待更新的主机组。 主机组必须定义了 `groupid` 属性。
hosts	对象/数组	待添加给所有主机组的主机。 主机必须定义了 `hostid` 属性。
templates	对象/数组	需要添加到所有主机组的模板。 模板必须定义了 `templateid` 属性。

返回值

(对象)返回一个对象,该对象包含 `groupids` 属性下的已更新主机组的 id。

示例

给主机组添加主机

给 ID 为 5 和 6 的主机组添加两个主机。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostgroup.massadd",
    "params": {
        "groups": [
            {
                "groupid": "5"
            },
```

```
            {
                "groupid": "6"
            }
        ],
        "hosts": [
            {
                "hostid": "30050"
            },
            {
                "hostid": "30001"
            }
        ]
    },
    "auth": "f223adf833b2bf2ff38574a67bba6372",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "groupids": [
            "5",
            "6"
        ]
    },
    "id": 1
}
```

另请参阅

- 主机
- 模板

源代码

CHostGroup::massAdd() in *ui/include/classes/api/services/CHostGroup.php*.

更新

描述

`hostgroup.update(hostGroups 对象/数组) 对象`

通过该方式，可以对已有的主机组进行更新。

> **Note:**
> 这个方法仅允许 管理员和 超级管理员用户类型。可以在用户角色中撤销调用方法的权限设置。详情参考用户角色

参数

(对象/数组) 主机组属性 将被更新。

`groupid` 属性必须为每个主机组定义，所有其他属性都是可选的。只有给定的属性将被更新，其他所有属性将保持不变。

返回值

(对象) 返回一个对象，该对象包含 `groupids` 属性下的已更新主机组的 id。

示例

重命名主机组

将主机组重命名为 “Linux hosts”。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostgroup.update",
    "params": {
        "groupid": "7",
        "name": "Linux hosts"
    },
    "auth": "700ca65537074ec963db7efabda78259",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "groupids": [
            "7"
        ]
    },
    "id": 1
}
```

源代码

CHostGroup::update() in *ui/include/classes/api/services/CHostGroup.php*.

获取

描述

`hostgroup.get(对象参数) 整数/数组`

该方法允许根据给定的参数检索主机组。

Note:
这个方法仅允许任何用户类型。可以在用户角色中撤销调用方法的权限设置。详情参考用户角色

参数

(对象)参数定义所需的输出。

该方法支持以下参数。

参数	类	描述
graphids	字符串/数组	只返回包含给定图形的主机或模板的主机组。
groupids	字符串/数组	只返回具有给定主机组 id 的主机组。
hostids	字符串/数组	只返回包含给定主机的主机组。
maintenanceids	字符串/数组	只返回受给定维护影响的主机组。
monitored_hosts	标记	只返回包含监控主机的主机组。
real_hosts	标记	只返回包含主机的主机组。
templated_hosts	标记	只返回包含模板的主机组。
templateids	字符串/数组	只返回包含给定模板的主机组。
triggerids	字符串/数组	只返回包含具有给定触发器的主机或模板的主机组。

参数	类	描述
with_graphs	标记	只返回包含有图形的主机的主机组。
with_graph_prototypes	标记	只返回包含带有图形原型的主机的主机组。
with_hosts_and_templates	标记	只返回包含主机 或模板的主机组。
with_httptests	标记	只返回包含 web 检查主机的主机组。 覆盖 `with_monitored_httptests` 参数。
with_items	标记	只返回包含有监控项的主机或模板的主机组。 覆盖 `with_monitored_items` 和 `with_simple_graph_items` 参数。
with_item_prototypes	标记	只返回包含带有监控项原型的主机的主机组。 覆盖 `with_simple_graph_item_p` 参数。
with_simple_graph_item_prototypes	标记	只返回包含具有监控项原型的主机的主机组，监控项原型支持创建并具有数字类型的信息。
with_monitored_httptests	标记	只返回包含启用 web 检查的主机的主机组。
with_monitored_items	标记	只返回包含启用监控项的主机或模板的主机组。 覆盖 `with_simple_graph_items` 参数。
with_monitored_triggers	标记	只返回包含启用触发器的主机的主机组。触发器中使用的所有监控项也必须启用。
with_simple_graph_items	标记	只返回包含带有数字监控项的主机的主机组。
with_triggers	标记	只返回包含有触发器的主机的主机组。 覆盖 `with_monitored_triggers` 参数。
selectDiscoveryRule	查询	返回发现规则 属性和创建主机组的 LLD 规则。

参数	类	描述
selectGroupDiscovery	查询	返回 groupDiscovery 属性和主机组发现对象. 发现主机组对象将发现的主机组链接到主机组原型，具有以下属性: groupid - （字符串）发现的主机组 ID; lastcheck - （时间戳）最后发现主机组的时间; name - （字符串）主机组原型的名称; parent_group_prototypeid - （字符串）创建主机组的主机组原型 ID; ts_delete - （时间戳）当主机组未被发现时，删除该主机组的时间。
selectHosts	查询	返回主机属性和属于主机组的主机。 支持 count.
selectTemplates	查询	返回模板 属性和属于主机组的模板。 支持 count.
limitSelects	整型	限制子查询返回的记录数量。 适用于下列子查询项: selectHosts - 结果将按 host 排序; selectTemplates - r 结果将按 host 排序。
sortfield	字符串/数组	根据给定的属性对结果进行排序。 可能的值是: groupid, name.
countOutput	布尔值	这些对所有 get 方法通用的参数在参考注释 中有详细描述。
editable	布尔值	
excludeSearch	布尔值	
filter	对象	
limit	整型	
output	查询	
preservekeys	布尔值	
search	对象	
searchByAny	布尔值	

参数	类	描述
searchWildcardsEnabled	布尔值	
sortorder	字符串/数组	
startSearch	布尔值	

返回值

（整数/数组）返回其中之一:

- 一组对象;
- 如果使用了 `countOutput` 参数, 返回对象的数量。

示例

按名称检索数据

检索关于 “Zabbix servers” 和 “Linux servers” 两个主机组的所有数据。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "hostgroup.get",
    "params": {
        "output": "extend",
        "filter": {
            "name": [
                "Zabbix servers",
                "Linux servers"
            ]
        }
    },
    "auth": "6f38cddc44cfbb6c1bd186f9a220b5a0",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "groupid": "2",
            "name": "Linux servers",
            "internal": "0"
        },
        {
            "groupid": "4",
            "name": "Zabbix servers",
            "internal": "0"
        }
    ],
    "id": 1
}
```

另请参考

- 主机
- 模板

源代码

CHostGroup::get() in *ui/include/classes/api/services/CHostGroup.php*.

事件

此类用于处理事件。

对象引用：

- 事件

可用方法：

- event.get - 获取事件
- event.acknowledge - 确认事件

> 事件对象

以下对象与 `event` API 直接相关。

事件

Note:
事件由 Zabbix server 创建，不能通过 API 修改。

事件对象具有以下属性。

属性	类型	描述
eventid	string	事件的 ID。
source	integer	事件的类型。 可能的值： 0 - 由触发器创建的事件； 1 - 由发现规则创建的事件； 2 - 由主动 agent 自动注册创建的事件； 3 - 内部事件； 4 - 在服务状态更新时创建的事件。
object	integer	与事件相关联的对象的类型。 触发器事件的可能值： 0 - 触发器。 发现事件的可能值： 1 - 发现的主机； 2 - 发现的服务。 自动注册事件的可能值： 3 - 自动注册的主机。 内部事件的可能值： 0 - 触发器； 4 - 监控项； 5 - LLD 规则。 服务事件的可能值： 6 - 服务。
objectid	string	相关对象的 ID。
acknowledged	integer	事件是否已被确认。
clock	timestamp	创建事件的时间。
ns	integer	创建事件时的纳秒数。
name	string	解析后的事件名称。

属性	类型	描述
value	integer	相关对象的状态。 触发器和服务事件的可能值： 0 - 正常； 1 - 问题。 发现事件的可能值： 0 - 主机或服务正常； 1 - 主机或服务异常； 2 - 发现主机或服务； 3 - 主机或服务丢失。 内部事件的可能值： 0 - ”正常”状态； 1 - ”未知”或”不支持”状态。 主动 agent 自动注册事件不使用此参数。
severity	integer	事件的当前严重性。 可能的值： 0 - 未分类； 1 - 信息； 2 - 警告； 3 - 中等； 4 - 高； 5 - 灾难。
r_eventid	string	恢复事件 ID
c_eventid	string	用于重写（关闭）当前事件的事件 ID，根据全局关联规则。查看 `correlationid` 以识别确切的关联规则。 此参数仅在事件由全局关联规则关闭时定义。
correlationid	string	生成问题关闭的关联规则的 ID。 此参数仅在事件由全局关联规则关闭时定义。
userid	string	如果是手动关闭的事件，用户 ID。
suppressed	integer	事件是否被抑制。 可能的值： 0 - 事件处于正常状态； 1 - 事件被抑制。
opdata	string	带有扩展宏的运营数据。
urls	array	活跃的媒体类型 URL。

事件标签

事件标签对象具有以下属性。

属性	类型	描述
tag	string	事件标签名称。
value	string	事件标签值。

媒体类型 URL

媒体类型 URL 对象具有以下属性。

属性	类型	描述
name	string	定义的媒体类型 URL 名称。
url	string	定义的媒体类型 URL 值。

结果将仅包含具有启用的事件菜单条目的活动媒体类型。属性中使用的宏将被扩展，但如果其中一个属性包含未扩展的宏，则两个属性都将从结果中排除。支持的宏，请参见支持的宏。

确认

描述

`object event.acknowledge(object/array parameters)`

此方法允许更新事件。可以执行以下更新操作：

- 关闭事件。如果事件已解决，则将跳过此操作。
- 确认事件。如果事件已被确认，则将跳过此操作。
- ·取消确认事件。如果未确认事件，则将跳过此操作。
- 添加消息。
- 更改事件严重性。如果事件已经具有相同的严重性，则将跳过此操作。

只能更新触发器事件。
只能更新问题事件。
需要触发器的读/写权限才能关闭事件或更改事件的严重性。
要关闭事件，触发器中应允许手动关闭。

此方法适用于任何类型的用户。可以在用户角色设置中撤销调用该方法的权限。参阅用户角色。

参数

(`object/array`) 参数包含事件的 ID 和和应该执行的更新操作。

参数	类型	描述
eventids (必填)	string/object	要确认的事件的 ID。
action (必填)	integer	事件更新操作。这是位掩码字段，可以接受任何值组合。 可用值： 1 - 关闭问题； 2 - 确认事件； 4 - 添加消息； 8 - 更改严重性； 16 - 取消确认事件。
message	string	消息的文本。 如果操作包含 “添加消息” 标志，则该字段必填。
severity	integer	事件的新严重性。 如果操作包含 “更改严重性” 标志，则该字段 必填。 可用值： 0 - 未分类； 1 - 信息； 2 - 警告； 3 - 一般严重； 4 - 严重； 5 - 灾难。

返回值

(`object`) 返回一个对象，该对象包含 `eventids` 属性下更新的事件 ID。

示例

确认事件

确认单个事件并留言。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "event.acknowledge",
    "params": {
    "eventids": "20427",
    "action": 6,
    "message": "Problem resolved."
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
    "eventids": [
    "20427"
    ]
    },
    "id": 1
}
```

更改事件的严重性

更改多个事件的严重性并留言。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "event.acknowledge",
    "params": {
    "eventids": ["20427", "20428"],
    "action": 12,
    "message": "Maintenance required to fix it.",
    "severity": 4
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
    "eventids": [
    "20427",
    "20428"
    ]
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CEvent.php 中的 CEvent::acknowledge() 。

获取

描述

`integer/array event.get(object parameters)`

该方法允许根据给定的参数检索事件。

Attention:
如果管家尚未删除这些事件，则此方法可能会返回已删除实体的事件。

Note:
此方法适用于任何类型的用户。可以在用户角色设置中撤销调用该方法的权限。查看用户角色。

参数

`(object)` 定义所需要输出的参数。

此方法支持以下参数。

参数	类型	描述
eventids	string/array	仅返回具有给定 ID 的事件。
groupids	string/array	仅返回由属于给定主机组的对象创建的事件。
hostids	string/array	仅返回由属于给定主机的对象创建的事件。
objectids	string/array	仅返回由给定对象创建的事件。
source	integer	仅返回给定类型的事件。 参阅事件对象页面获取支持的事件类型列表。 默认值：0 - 触发器事件。
object	integer	仅返回由给定类型的对象创建的事件。 参阅事件对象页面获取支持的事件类型列表。 默认值：0 - 触发器。
acknowledged	boolean	如果设置为 `true` 仅返回已确认的事件。
suppressed	boolean	`true` - 只返回被抑制的事件； `false` - 返回正常状态的事件。
severities	integer/array	仅返回具有给定严重性的事件。仅当对象是触发器时才适用。
evaltype	integer	标签搜索规则。 可用值： 0 - （默认）和/或； 2 - 或。

参数	类型	描述
tags	对象数组	仅返回具有给定标签的事件。通过标签进行精确匹配，通过值和操作符进行不区分大小写的匹配。 格式：`[{"tag": "<tag>", "value": "<value>", "operator": "<operator>"}, ...]`。 空数组返回所有事件。 可能的运算符类型： 0 - （默认）Like（类似于……）； 1 - Equal（等于）； 2 - Not like（不类似于……）； 3 - Not equal（不等于）； 4 - Exists（存在）； 5 - Not exists（不存在）。
eventid_from	string	仅返回 ID 大于或等于给定 ID 的事件。
eventid_till	string	仅返回 ID 小于或等于给定 ID 的事件。
time_from	timestamp	仅返回在给定时间或在给定时间之后创建的事件。
time_till	timestamp	仅返回在给定时间或在给定时间之前创建的事件。
problem_time_from	timestamp	仅返回从 `problem_time_from` 设定的时间开始处于问题状态的事件。 仅当事件源是触发器事件且对象为触发 时适用。如果指定了 `problem_time_till`，则为强制性。
problem_time_till	timestamp	仅返回在 `problem_time_still` 设定的时间之前处于问题状态的事件。 仅当事件源是触发器事件且对象为触发 时适用。如果指定了 `problem_time_from`，则为强制性。
value	integer/array	仅返回具有给定值的事件。

参数	类型	描述
selectHosts	query	返回主机 属性，其中 hosts 包含创建事件的对象。仅支持由触发器、监控项或 LLD 规则生成的事件。
selectRelatedObject	query	返回带有创建事件的对象的’relatedObject ’属性。返回的对象类型取决于事件类型。
select_alerts	query	返回由事件生成的告警的告警 属性。警报按时间倒序排列。

参数	类型	描述
select_acknowledges	query	返回带有事件更新的 `acknowledges` 属性。事件更新按时间倒序排列。 事件更新对象具有以下属性： `acknowledgeid` - (string) 确认事件的 ID； `userid` - (string) 更新事件的用户 ID； `eventid` - (string) 更新事件的 ID； `clock` - (timestamp) 事件被更新的时间； `message` - (string) 消息文本； `action` - (integer) 更新执行的操作，参见event.acknowledge； `old_severity` - (integer) 此更新操作之前的事件严重性； `new_severity` - (integer) 此更新操作后的事件严重性； `username` - (string) 更新该事件的用户的 username（用户名）； `name` - (string) 更新该事件的用户的 name（可见名）； `surname` - (string) 更新事件的用户的 surname（姓）。 支持 `count`。
selectTags	query	返回事件标签的标签 属性。
selectSuppressionData	query	返回维护列表的 `suppression_data` 属性： `maintenanceid` - (string) 维护期 ID； `suppress_until` - (integer) 事件被抑制的时间。

参数	类型	描述
sortfield	string/array	按给定属性对结果进行排序。 可用值：`eventid`，`objectid` 和 `clock`。
countOutput	boolean	这些参数对于所有的 `get` 方法都是通用的，详细描述请参见参考说明 页面。
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

`(integer/array)` 返回其中之一：

- 一个对象数组；
- 如果使用了 `countOutput` 参数，则返回检索到的对象的计数。

示例

检索触发器事件

检索 ID 为”13926” 的触发器的最新事件。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "event.get",
    "params": {
    "output": "extend",
    "select_acknowledges": "extend",
    "selectTags": "extend",
    "selectSuppressionData": "extend",
    "objectids": "13926",
    "sortfield": ["clock", "eventid"],
    "sortorder": "DESC"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
    {
    "eventid": "9695",
    "source": "0",
    "object": "0",
    "objectid": "13926",
```

```
"clock": "1347970410",
"value": "1",
"acknowledged": "1",
"ns": "413316245",
"name": "MySQL is down",
"severity": "5",
"r_eventid": "0",
"c_eventid": "0",
"correlationid": "0",
"userid": "0",
"opdata": "",
"acknowledges": [
{
"acknowledgeid": "1",
"userid": "1",
"eventid": "9695",
"clock": "1350640590",
"message": "Problem resolved.\n\r----[BULK ACKNOWLEDGE]----",
"action": "6",
"old_severity": "0",
"new_severity": "0",
"username": "Admin",
"name": "Zabbix",
"surname": "Administrator"
}
],
"suppression_data": [
{
"maintenanceid": "15",
"suppress_until": "1472511600"
}
],
"suppressed": "1",
"tags": [
{
"tag": "service",
"value": "mysqld"
},
{
"tag": "error",
"value": ""
}
]
},
{
"eventid": "9671",
"source": "0",
"object": "0",
"objectid": "13926",
"clock": "1347970347",
"value": "0",
"acknowledged": "0",
"ns": "0",
"name": "Unavailable by ICMP ping",
"severity": "4",
"r_eventid": "0",
"c_eventid": "0",
"correlationid": "0",
"userid": "0",
"opdata": "",
"acknowledges": [],
"suppression_data": [],
```

```
    "suppressed": "0",
    "tags": []
    }
    ],
    "id": 1
}
```

按时间段检索事件

按时间倒序检索在 2012 年 10 月 9 日至 10 日之间创建的所有事件。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "event.get",
    "params": {
    "output": "extend",
    "time_from": "1349797228",
    "time_till": "1350661228",
    "sortfield": ["clock", "eventid"],
    "sortorder": "desc"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
    {
    "eventid": "20616",
    "source": "0",
    "object": "0",
    "objectid": "14282",
    "clock": "1350477814",
    "value": "1",
    "acknowledged": "0",
    "ns": "0",
    "name": "Less than 25% free in the history cache",
    "severity": "3",
    "r_eventid": "0",
    "c_eventid": "0",
    "correlationid": "0",
    "userid": "0",
    "opdata": "",
    "suppressed": "0"
    },
    {
    "eventid": "20617",
    "source": "0",
    "object": "0",
    "objectid": "14283",
    "clock": "1350477814",
    "value": "0",
    "acknowledged": "0",
    "ns": "0",
    "name": "Zabbix trapper processes more than 75% busy",
    "severity": "3",
    "r_eventid": "0",
    "c_eventid": "0",
    "correlationid": "0",
    "userid": "0",
```

```
    "opdata": "",
    "suppressed": "0"
    },
    {
    "eventid": "20618",
    "source": "0",
    "object": "0",
    "objectid": "14284",
    "clock": "1350477815",
    "value": "1",
    "acknowledged": "0",
    "ns": "0",
    "name": "High ICMP ping loss",
    "severity": "3",
    "r_eventid": "0",
    "c_eventid": "0",
    "correlationid": "0",
    "userid": "0",
    "opdata": "",
    "suppressed": "0"
    }
    ],
    "id": 1
}
```


参见

- 告警
- 监控项
- 主机
- LLD 规则
- 触发器

来源

ui/include/classes/api/services/CEvent.php 中的 CEvent::get() 。

令牌

此类用于管理令牌。

对象引用：

- 令牌

可用方法：

- token.create - 创建新的令牌
- token.delete - 删除令牌
- token.get - 获取令牌
- token.update - 更新令牌
- token.generate - 生成令牌

> 令牌对象

以下对象与 令牌 API 直接相关。

令牌

令牌对象具有以下属性。

属性	类型	描述
tokenid	string	（只读）令牌 ID。

属性	类型	描述
name （必需）	string	令牌名称。
description	text	令牌描述。
userid	string	（只读更新）一个被分配令牌的用户。 默认：当前用户。
lastaccess	timestamp	（只读）验证令牌的最近日期和时间。 如果该令牌从未经过身份验证，则为零。
status	integer	令牌状态。 可用值： 0 - （默认）启用令牌； 1 - 禁用令牌。
expires_at	timestamp	令牌的过期日期和时间。 永不过期的令牌为 0。
created_at	timestamp	（只读）令牌的创建日期和时间。
creator_userid	string	（只读）令牌的创建者。

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

创建

描述

`object token.create(object/array tokens)`

此方法允许创建新的令牌。

> **Note:**
> 用户角色需要具有管理 *API* 令牌权限才能管理其他用户的令牌。

> **Attention:**
> 通过此方法创建的令牌在使用前也需要生成。

参数

`(object/array)` 要创建的令牌。

此方法接受令牌带有规范的令牌属性standard token properties。

返回值

`(object)` 返回一个对象其中包含在 `tokenids` 属性下创建的令牌的 ID。返回的 ID 的顺序与传递的令牌的顺序相匹配。

示例

创建令牌

创建一个永不过期的已启用令牌，并对 ID 为 2 的用户进行身份验证。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "token.create",
    "params": {
        "name": "Your token",
        "userid": "2"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "tokenids": [
            "188"
        ]
    },
    "id": 1
}
```

创建 2021 年 1 月 21 日到期的禁用令牌。此令牌将对当前用户进行身份验证。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "token.create",
    "params": {
        "name": "Your token",
        "status": "1",
        "expires_at": "1611238072"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "tokenids": [
            "189"
        ]
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CToken.php 中的 CToken::create()。

删除

描述

`object token.delete(array tokenids)`

此方法允许删除令牌。

> **Note:**
> 用户角色需要具有管理 *API* 令牌权限才能管理其他用户的令牌。

参数

(array) 要删除的令牌的 ID。

返回值

(object) 返回一个对象，其中包含 tokenids 属性下已删除令牌的 ID。

示例

删除多个令牌

删除两个令牌。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "token.delete",
    "params": [
        "188",
        "192"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "tokenids": [
            "188",
            "192"
        ]
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CToken.php 中的 CToken::delete()。

更新

描述

`object token.update(object/array tokens)`

此方法允许更新现有的令牌。

> **Note:**
> 用户角色需要具有管理 *API* 令牌权限才能管理其他用户的令牌。

参数

(object/array) 要更新的令牌属性。

必须为每个令牌定义 tokenid 属性，所有其他属性都是可选的。只有被传递的属性会被更新，所有其他的将保持不变。

此方法接受具有标准的令牌属性的的令牌。

返回值

(object) 返回一个对象，其中包含 tokenids 属性下已被更新的令牌的 ID。

示例

删除过期令牌

从令牌中删除到期日期。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "token.update",
    "params": {
        "tokenid": "2",
        "expires_at": "0"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "tokenids": [
            "2"
        ]
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CToken.php 中的 CToken::update()。

生成

描述

`object token.generate(array tokenids)`

此方法允许生成令牌。

> **Note:**
> 用户角色需要具有管理 *API* 令牌权限才能管理其他用户的令牌。

> **Attention:**
> 此方法仅能在令牌已经创建的情况下生成令牌。

参数

(array) 要生成的令牌的 ID。

返回值

(array) 返回一个对象数组，其中包含 `tokenid` 属性下生成的令牌的 ID 和 `token` 属性下生成的授权字符串。

属性	类型	描述
tokenid	string	令牌 ID。
token	string	为此令牌生成的授权字符串。

示例

生成多个令牌

生成两个令牌。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "token.generate",
    "params": [
        "1",
        "2"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
```

```
            "tokenid": "1",
            "token": "bbcfce79a2d95037502f7e9a534906d3466c9a1484beb6ea0f4e7be28e8b8ce2"
        },
        {
            "tokenid": "2",
            "token": "fa1258a83d518eabd87698a96bd7f07e5a6ae8aeb8463cae33d50b91dd21bd6d"
        }
    ],
    "id": 0
}
```


来源

ui/include/classes/api/services/CToken.php 中的 CToken::generate()。

获取

描述

`integer/array token.get(object parameters)`

此方法允许根据给定的参数获取令牌。

Note:
只允许 *Super admin(超级管理员)* 用户可以查看其他用户的令牌。

参数

`(object)` 定义期望输出的参数。

此方法支持以下参数。

参数	类型	描述
tokenids	string/array	仅返回给定 ID 的令牌。
userids	string/array	仅返回为给定用户创建的令牌。
token	string	仅返回为给定身份验证令牌创建的令牌。
valid_at	timestamp	仅返回在给定的日期和时间内是有效(未过期) 的令牌。
expired_at	timestamp	仅返回在给定的日期和时间内是过期(无效) 的令牌。
sortfield	string/array	按给定的属性对结果进行排序。 可用值:`tokenid`，`name`，`lastaccess`，`status`，`expires_at` 和 `created_at`。
countOutput	boolean	这些参数对所有的 `get` 方法是通用的，详情请参阅参考说明 。
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	

参数	类型	描述
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

(integer/array) 返回其中之一：

- 对象数组；
- 如果使用了 countOutput 参数，则检索到对象的数量。

示例

检索令牌

检索 ID 为 “2” 的令牌的所有数据。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "token.get",
    "params": {
        "output": "extend",
        "tokenids": "2"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "tokenid": "1",
            "name": "The Token",
            "description": "",
            "userid": "1",
            "lastaccess": "0",
            "status": "0",
            "expires_at": "1609406220",
            "created_at": "1611239454",
            "creator_userid": "1"
        }
    ],
    "id": 1
}
```

来源

ui/include/classes/api/services/CToken.php 中的 CToken::get()。

仪表盘

此类被设计用于仪表盘。

对象引用：

- 仪表盘
- 仪表盘页面
- 仪表盘部件
- 仪表盘部件字段

- 仪表盘用户
- 仪表盘用户组

可用方法：

- dashboard.create - 创建新的仪表盘
- dashboard.delete - 删除仪表盘
- dashboard.get - 获取仪表盘
- dashboard.update - 更新仪表盘

> 仪表盘对象

下列对象与 `dashboard`（仪表盘） API 直接相关。

仪表盘

仪表盘对象具有以下属性：

属性	类型	说明
dashboardid	字符	(只读）仪表盘 ID。
name (必需)	字符	仪表盘名称。
userid	字符	仪表盘所属用户的用户 ID。
private	整数	仪表盘的共享类型。 可用值： 0 - 公共仪表盘； 1 - （默认）私有仪表盘。
display_period	整数	默认页面显示周期(秒)。 可用值：10，30，60，120，600，1800，3600。 默认：30。
auto_start	整数	自动幻灯片放映。 可用值： 0 - 不自动开始幻灯片放映； 1 - （默认）自动开始幻灯片放映。

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

仪表盘页面

仪表盘页面对象具有如下属性：

属性	类型	说明
dashboard_pageid	string	(只读）仪表盘页面 ID。
name	string	仪表盘页面名称。 默认：空字符串。

属性	类型	说明
display_period	integer	仪表盘页面显示周期（秒）。 可用值：0，10，30，60，120，600，1800，3600。 默认：0（使用默认页显示周期）。
widgets	array	仪表盘部件对象数组。

仪表盘控件

仪表盘控件对象具有以下属性。

属性	类型	描述
widgetid	string	*(只读)* 仪表盘控件的 ID。
类型 (必需)	string	仪表盘控件的类型。 可能的值有： actionlog - 动作日志； clock - 时钟； *(已弃用)* dataover - 数据概览； discovery - 发现状态； favgraphs - 常用图表； favmaps - 常用地图； geomap - 地理图； graph - 图表 (经典)； graphprototype - 图表原型； hostavail - 主机可用性； item - 监控项值； map - 地图； navtree - 地图导航树； plaintext - 纯文本； problemhosts - 问题主机； problems - 问题； problemsbysv - 按严重性的问题； slareport - SLA 报告； svggraph - 图表； systeminfo - 系统信息； tophosts - Top 主机； trigover - 触发器概览； url - URL； web - Web 监控。
name	string	自定义控件名称。
x	integer	从仪表盘左侧的水平位置。 有效值范围从 0 到 23。
y	integer	从仪表盘顶部的垂直位置。 有效值范围从 0 到 62。
width	integer	控件宽度。 有效值范围从 1 到 24。
height	integer	控件高度。 有效值范围从 2 到 32。

属性	类型	描述
view_mode	integer	控件视图模式。 可能的值： 0 - (默认) 默认控件视图； 1 - 带隐藏标题；
fields	array	仪表盘控件字段对象的数组。 特定控件必需；请参阅仪表盘控件字段中的各个控件。

仪表板小部件字段

仪表板小部件字段对象具有以下属性。

属性	类型	描述
type (必填)	整数	小部件字段的类型。 可能的值： 0 - 整数； 1 - 字符串； 2 - 主机组; 3 - 主机; 4 - 监控项; 5 - 监控项原型; 6 - 图形; 7 - 图形原型; 8 - 映射; 9 - 服务； 10 - SLA。
name (必填)	string	小部件字段名称。 可能的值：参见仪表板小部件字段。
value (必填)	mixed	小部件字段值取决于类型。 可能的值：请参阅仪表板小部件字段 .

仪表盘用户组

基于用户组的仪表盘权限列表。具有以下属性：

属性	类型	说明
usrgrpid （必需）	string	用户组 ID。
permission （必需）	integer	权限级别类型。 可用值： 2 - 只读； 3 - 读写。

仪表盘用户

基于用户的仪表盘权限列表。其具有以下属性：

属性	类型	说明
userid （必需）	string	用户 ID。
permission （必需）	integer	权限级别类型。 可用值： 2 - 只读； 3 - 读写。

仪表盘组件字段

仪表盘小部件参数导航链接和相应仪表盘小部件字段对象的可能属性值。

要查看每个小部件的参数和属性值，请访问各个小部件的独立页面：

- 动作日志
- 时钟
- 发现状态
- 常用图表
- 常用地图
- 地理地图
- 图表
- 图表（经典）
- 图表原型
- 主机可用性
- 监控项值
- 地图
- 地图导航树
- 纯文本
- 问题主机
- 问题
- SLA 报表
- 系统信息
- 按严重性的问题
- TOP 主机
- 触发器概览
- URL
- Web 监控

已弃用的小部件：

- 数据概览

Attention:
已弃用的小部件将在即将发布的主要版本中被移除。

1 Action log

Description

These parameters and the possible property values for the respective dashboard widget field objects allow to configure the *Action log* widget in `dashboard.create` and `dashboard.update` methods.

Parameters

The following parameters are supported for the *Action log* widget.

Parameter	type	name	value
Refresh interval	0	rf_rate	0 - No refresh; 10 - 10 seconds; 30 - 30 seconds; 60 - *(default)* 1 minute; 120 - 2 minutes; 600 - 10 minutes; 900 - 15 minutes.

Parameter	type	name	value
Sort entries by	0	sort_triggers	3 - Time (ascending); 4 - *(default)* Time (descending); 5 - Type (ascending); 6 - Type (descending); 7 - Status (ascending); 8 - Status (descending); 11 - Recipient (ascending); 12 - Recipient (descending).
Show lines	0	show_lines	Valid values range from 1-100. Default: 25.

Examples

The following examples aim to only describe the configuration of the dashboard widget field objects for the *Action log* widget. For more information on configuring a dashboard, see `dashboard.create`.

Configuring an *Action log* widget

Configure an *Action log* widget that displays 10 entries of action operation details, sorted by time (in ascending order).

Request:

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "actionlog",
                        "name": "Action log",
                        "x": 0,
                        "y": 0,
                        "width": 12,
                        "height": 5,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 0,
                                "name": "show_lines",
                                "value": 10
                            },
                            {
                                "type": 0,
                                "name": "sort_triggers",
                                "value": 3
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
```

```
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

See also

- Dashboard widget field
- dashboard.create
- dashboard.update

2 时钟

描述

这些参数和各自仪表盘小部件字段对象的可能属性值允许在 dashboard.create 和 dashboard.update 方法中配置时钟仪表盘小部件。

Attention:
在创建或更新仪表盘时，不会验证小部件的 fields 属性。这允许用户修改内置小部件并创建自定义小部件，但同时也引入了错误创建或更新小部件的风险。为确保时钟小部件的成功创建或更新，请参考下面表格中概述的参数要求。

参数

以下参数适用于时钟小部件。

参数	类型	名称	值
刷新间隔	0	rf_rate	0 - 不刷新； 10 - 10 秒； 30 - 30 秒； 60 - 1 分钟； 120 - 2 分钟； 600 - 10 分钟； 900 - (默认) 15 分钟。
时间类型	0	time_type	0 - (默认) 本地时间； 1 - 服务器时间； 2 - 主机时间。

如果时间类型设置为 “主机时间”，则支持以下参数。

参数	类型	名称	值
监控项 (必需)	4	itemid	监控项 ID。

示例

以下示例仅旨在描述时钟仪表盘小部件字段对象的配置。有关配置仪表盘的更多信息，请参阅dashboard.create。

配置时钟小部件

配置一个显示 Zabbix 服务器时间的时钟小部件。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "clock",
                        "name": "Clock",
                        "x": 0,
                        "y": 0,
                        "width": 4,
                        "height": 3,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 0,
                                "name": "time_type",
                                "value": 1
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
```

```
}
```

另请参阅

- 仪表盘小部件字段
- `dashboard.create`
- `dashboard.update`

3 数据概览

Attention:
此小部件已被弃用，并将在下一个主要版本中被移除。请考虑改用*Top* 主机小部件。

描述

这些参数以及各自仪表盘小部件字段对象的可能属性值允许在 `dashboard.create` 和 `dashboard.update` 方法中配置数据概览小部件。

Attention:
在创建或更新仪表盘时，不会验证小部件 `fields` 属性。这允许用户修改内置小部件并创建自定义小部件，但也存在创建或更新小部件不正确的风险。为确保成功创建或更新数据概览小部件，请参阅以下表格中概述的参数要求。

参数

以下参数受 “数据概览” 仪表盘支持。

参数		类型	名称	值
刷新间隔		0	rf_rate	0 - 不刷新； 10 - 10 秒； 30 - 30 秒； 60 - *(默认)* 1 分钟； 120 - 2 分钟； 600 - 10 分钟； 900 - 15 分钟。
主机组		2	groupids	主机组 ID。 注意：若要配置多个主机组，请为每个主机组创建一个仪表盘小部件字段对象。
主机		3	hostids	主机 ID。 注意：若要配置多个主机，请为每个主机创建一个仪表盘小部件字段对象。对于多个主机，参数主机组必须完全不配置或配置至少一个属于已配置主机的主机组。
标签（属性名中的数字（例如 tags.tag.0）引用标签评估列表中的标签顺序）				
	评估类型	0	evaltype	0 - *(默认)* And/Or； 2 - Or。
	标签名	1	tags.tag.0	任何字符串值。 如果配置标签，则必须配置参数标签名。

参数		类型	名称	值
	操作符	0	tags.operator.0	0 - 包含； 1 - 等于； 2 - 不包含； 3 - 不等于； 4 - 存在； 5 - 不存在。 如果配置标签，则必须配置参数操作符。
	标签值	1	tags.value.0	任何字符串值。 如果配置标签，则必须配置参数标签值。
显示已忽略的问题		0	show_suppressed	0 - *(默认)* 已禁用； 1 - 已启用。
主机位置		0	style	0 - *(默认)* 左侧； 1 - 顶部。

示例

以下示例仅用于描述 “数据概览” 仪表盘小部件字段对象的配置。有关配置仪表盘的更多信息，请参见dashboard.create。

配置 “数据概览” 仪表盘小部件

配置一个 “数据概览” 仪表盘小部件，以显示主机 “10084” 的数据，并且仅显示标签名称为 “component” 且其值为 “cpu” 的监控项的数据。另外，将主机位置设置为顶部显示。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "dataover",
                        "name": "Data overview",
                        "x": 0,
                        "y": 0,
                        "width": 12,
                        "height": 5,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 3,
                                "name": "hostids",
                                "value": 10084
                            },
                            {
                                "type": 1,
                                "name": "tags.tag.0",
                                "value": "component"
                            },
                            {
                                "type": 0,
                                "name": "tags.operator.0",
                                "value": 0
                            },
```

```
                            {
                                "type": 1,
                                "name": "tags.value.0",
                                "value": "cpu"
                            },
                            {
                                "type": 0,
                                "name": "style",
                                "value": 1
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

另请参阅

- 仪表盘小部件字段
- `dashboard.create`
- `dashboard.update`

4 Discovery status

Description

These parameters and the possible property values for the respective dashboard widget field objects allow to configure the *Discovery status* widget in `dashboard.create` and `dashboard.update` methods.

Parameters

The following parameters are supported for the *Discovery status* widget.

Parameter	type	name	value
Refresh interval	0	rf_rate	0 - No refresh; 10 - 10 seconds; 30 - 30 seconds; 60 - *(default)* 1 minute; 120 - 2 minutes; 600 - 10 minutes; 900 - 15 minutes.

Examples

The following examples aim to only describe the configuration of the dashboard widget field objects for the *Discovery status* widget. For more information on configuring a dashboard, see `dashboard.create`.

Configuring *Discovery status* widget

Configure a *Discovery status* widget with the refresh interval set to 15 minutes.

Request:

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "discovery",
                        "name": "Discovery status",
                        "x": 0,
                        "y": 0,
                        "width": 6,
                        "height": 3,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 0,
                                "name": "rf_rate",
                                "value": 900
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
```

```
    "id": 1
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

See also

- Dashboard widget field
- `dashboard.create`
- `dashboard.update`

5 Favorite graphs

Description

These parameters and the possible property values for the respective dashboard widget field objects allow to configure the *Favorite graphs* widget in `dashboard.create` and `dashboard.update` methods.

Parameters

The following parameters are supported for the *Favorite graphs* widget.

Parameter	type	name	value
Refresh interval	0	rf_rate	0 - No refresh; 10 - 10 seconds; 30 - 30 seconds; 60 - 1 minute; 120 - 2 minutes; 600 - 10 minutes; 900 - *(default)* 15 minutes.

Examples

The following examples aim to only describe the configuration of the dashboard widget field objects for the *Favorite graphs* widget. For more information on configuring a dashboard, see `dashboard.create`.

Configuring a *Favorite graphs* widget

Configure a *Favorite graphs* widget with the refresh interval set to 10 minutes.

Request:

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "favgraphs",
                        "name": "Favorite graphs",
```

```
                        "x": 0,
                        "y": 0,
                        "width": 4,
                        "height": 3,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 0,
                                "name": "rf_rate",
                                "value": 600
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

See also

- Dashboard widget field
- `dashboard.create`
- `dashboard.update`

6 Favorite maps

Description

These parameters and the possible property values for the respective dashboard widget field objects allow to configure the *Favorite maps* widget in `dashboard.create` and `dashboard.update` methods.

Parameters

The following parameters are supported for the *Favorite maps* widget.

Parameter	type	name	value
Refresh interval	0	rf_rate	0 - No refresh; 10 - 10 seconds; 30 - 30 seconds; 60 - 1 minute; 120 - 2 minutes; 600 - 10 minutes; 900 - *(default)* 15 minutes.

Examples

The following examples aim to only describe the configuration of the dashboard widget field objects for the *Favorite maps* widget. For more information on configuring a dashboard, see `dashboard.create`.

Configuring a *Favorite maps* widget

Configure a *Favorite maps* widget with the refresh interval set to 10 minutes.

Request:

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "favmaps",
                        "name": "Favorite maps",
                        "x": 0,
                        "y": 0,
                        "width": 4,
                        "height": 3,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 0,
                                "name": "rf_rate",
                                "value": 600
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
```

```
    "id": 1
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

See also

- Dashboard widget field
- dashboard.create
- dashboard.update

7 地理地图

描述

这些参数以及各自仪表盘小部件字段对象的可能属性值允许在 dashboard.create 和 dashboard.update 方法中配置*Geomap*小部件。

Attention:
在创建或更新仪表盘时，不会验证小部件 fields 属性。这允许用户修改内置小部件并创建自定义小部件，但也可能导致不正确创建或更新小部件的风险。为确保成功创建或更新地理地图小部件，请参考下表概述的参数要求。

参数

地理地图小部件支持以下参数。

参数	类型	名称	值
刷新间隔	0	rf_rate	0 - 不刷新； 10 - 10 秒； 30 - 30 秒； 60 - (默认) 1 分钟； 120 - 2 分钟； 600 - 10 分钟； 900 - 15 分钟。
主机组	2	groupids	主机组 ID。 注意：要配置多个主机组，请为每个主机组创建一个仪表盘小部件字段对象。
主机	3	hostids	主机 ID。 注意：要配置多个主机，请为每个主机创建一个仪表盘小部件字段对象。对于多个主机，参数 主机组必须完全不配置或配置至少一个属于已配置主机的主机组。

参数		类型	名称	值	
Tags (属性名称中的数字（例如 tags.tag.0）引用标签评估列表中的标签顺序)					
	评估类型	0	evaltype	0 - *(默认)* And/Or； 2 - Or。	
		标签名称	1	tags.tag.0	任何字符串值。 如果配置 *Tags*，则必须设置参数 标签名称。
		运算符	0	tags.operator.0	0 - 包含； 1 - 等于； 2 - 不包含； 3 - 不等于； 4 - 存在； 5 - 不存在。 如果配置 *Tags*，则必须设置参数 运算符。
		标签值	1	tags.value.0	任何字符串值。 如果配置 *Tags*，则必须设置参数 标签值。
初始视图		1	default_view	以逗号分隔的纬度、经度和缩放级别(可选，有效值范围为 0-30)。 示例： 40.6892494,-74.0466891,10。	

示例

以下示例仅旨在描述地理地图仪表盘小部件字段对象的配置。有关配置仪表盘的更多信息，请参阅dashboard.create。

配置地理地图小部件

配置一个地理地图小部件，该小部件基于以下标签配置显示来自主机组”2” 和”22” 的主机：标签名为”component” 且其值为”node”，或者标签名为”location” 且其值等于”New York”。此外，将地图的初始视图设置为坐标”40.6892494”（纬度）、”-74.0466891”（经度），并设置缩放级别为”10”。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "geomap",
                        "name": "Geomap",
                        "x": 0,
                        "y": 0,
                        "width": 12,
```

```
                    "height": 5,
                    "view_mode": 0,
                    "fields": [
                        {
                            "type": 2,
                            "name": "groupids",
                            "value": 22
                        },
                        {
                            "type": 2,
                            "name": "groupids",
                            "value": 2
                        },
                        {
                            "type": 1,
                            "name": "default_view",
                            "value": "40.6892494,-74.0466891,10"
                        },
                        {
                            "type": 0,
                            "name": "evaltype",
                            "value": 2
                        },
                        {
                            "type": 1,
                            "name": "tags.tag.0",
                            "value": "component"
                        },
                        {
                            "type": 0,
                            "name": "tags.operator.0",
                            "value": 0
                        },
                        {
                            "type": 1,
                            "name": "tags.value.0",
                            "value": "node"
                        },
                        {
                            "type": 1,
                            "name": "tags.tag.1",
                            "value": "location"
                        },
                        {
                            "type": 0,
                            "name": "tags.operator.1",
                            "value": 1
                        },
                        {
                            "type": 1,
                            "name": "tags.value.1",
                            "value": "New York"
                        }
                    ]
                }
            ]
        }
    ],
    "userGroups": [
        {
            "usrgrpid": 7,
            "permission": 2
```

```
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

另请参阅

- 仪表盘小部件字段
- dashboard.create
- dashboard.update

8 图表

描述

这些参数以及各自仪表盘小部件字段对象的可能属性值允许在 dashboard.create 和 dashboard.update 方法中配置图表小部件。

Attention:
在创建或更新仪表盘时，不会验证小部件 fields 属性。这允许用户修改内置小部件并创建自定义小部件，但也存在创建或更新小部件不正确的风险。为确保成功创建或更新图表小部件，请参考下面表格中概述的参数要求。

参数

图表小部件支持以下参数。

参数	类型	名称	值
刷新间隔	0	rf_rate	0 - 不刷新； 10 - 10 秒； 30 - 30 秒； 60 - *(默认)* 1 分钟； 120 - 2 分钟； 600 - 10 分钟； 900 - 15 分钟。

数据集

以下参数支持配置数据集。

Note:
属性名称中的第一个数字（例如 ds.hosts.0.0，ds.items.0.0）代表特定的数据集，如果存在第二个数字，则代表已配置的主机或监控项。

参数	类型	名称	值
主机模式 (必填)	1	ds.hosts.0.0	主机名称或模式（例如 Zabbix*）。
监控项模式 (必填)	1	ds.items.0.0	监控项名称或模式（例如 *: Number of processed *values per second）。
颜色	1	ds.color.0	十六进制颜色代码（例如 FF0000）。 默认值：FF465C。
绘制方式	0	ds.type.0	0 - (默认) 线条； 1 - 点； 2 - 阶梯； 3 - 条形。
宽度	0	ds.width.0	有效值范围从 1-10。 默认值：1。 如果绘制方式设置为” 点” 或” 条形”，则宽度参数不可用。
点大小	0	ds.pointsize.0	有效值范围从 1-10。 默认值：3。 如果绘制方式设置为” 线条”、” 阶梯” 或” 条形”，则点大小参数不可用。
透明度	0	ds.transparency.0	有效值范围从 1-10。 默认值：5。
填充	0	ds.fill.0	有效值范围从 1-10。 默认值：3。 如果绘制方式设置为” 点” 或” 条形”，则填充参数不可用。
缺失数据	0	ds.missingdatafunc.0	0 - (默认) 无； 1 - 连接； 2 - 视为 0。 如果绘制方式设置为” 点” 或” 条形”，则缺失数据参数不可用。
Y 轴	0	ds.axisy.0	0 - (默认) 左侧； 1 - 右侧。
时间偏移	1	ds.timeshift.0	有效的时间字符串（例如 3600，1h 等）。 您可以使用时间后缀。也允许使用负值。 默认值：""（空）。
聚合函数	0	ds.aggregate_function.0	0 - (默认) 无； 1 - 最小值； 2 - 最大值； 3 - 平均值； 4 - 计数； 5 - 总和； 6 - 第一个； 7 - 最后一个。

参数	类型	名称	值
聚合间隔	1	ds.aggregate_interval.0	有效的时间字符串（例如 3600，1h 等）。 您可以使用时间后缀。 默认值：1h。
聚合	0	ds.aggregate_grouping.0	0 - *(默认)* 每个监控项； 1 - 数据集。 如果聚合函数设置为”无”，则聚合参数不可用。

显示选项

以下参数支持配置显示选项。

参数	类型	名称	值
历史数据选择	0	source	0 - *(默认)* 自动； 1 - 历史； 2 - 趋势。

时间段

以下参数支持配置时间段。

参数	类型	名称	值
设置自定义时间段	0	graph_time	0 - *(默认)* 禁用； 1 - 启用。
从	1	time_from	有效的时间字符串，格式为 `YYYY-MM-DD hh:mm:ss`。 也支持相对时间段值（如 `now`，`now/d`，`now/w-1w` 等）。 默认值：`now-1h`。
到	1	time_to	有效的时间字符串值，格式为 `YYYY-MM-DD hh:mm:ss`。 也支持相对时间段值（如 `now`，`now/d`，`now/w-1w` 等）。 默认值：`now`。

坐标轴

以下参数支持配置坐标轴。

参数	类型	名称	值
左 *Y* 轴	0	lefty	0 - 禁用； 1 - *(默认)* 启用。 如果 *Y* 轴（在数据集配置中）设置为”左”，则此参数可用。
右 *Y* 轴	0	righty	0 - *(默认)* 禁用； 1 - 启用。 如果 *Y* 轴（在数据集配置中）设置为”右”，则此参数可用。
最小值	1	lefty_min righty_min	任何数值。 默认值：""（空）。
最大值	1	lefty_max righty_max	任何数值。 默认值：""（空）。
单位（类型）	0	lefty_units righty_units	0 - *(默认)* 自动； 1 - 静态。
单位（值）	1	lefty_static_units	任何字符串值。 默认值：""（空）。

参数	类型	名称	值
X 轴	0	righty_static_units xaxis	0 - 禁用； 1 - *(*默认*)* 启用。

图例

以下参数支持配置图例。

参数	类型	名称	值
显示图例	0	legend	0 - 禁用； 1 - *(*默认*)* 启用。
行数	0	legend_lines	有效值范围从 1-5。 默认值：1。

问题

以下参数支持配置问题。

参数		类型	名称	值
显示问题		0	show_problems	0 - *(*默认*)* 禁用； 1 - 启用。
仅选定项		0	graph_item_problems	0 - 禁用； 1 - *(*默认*)* 启用。
问题主机		1	problemhosts.0	主机 名称。 注意：属性名称中的数字引用了配置的主机。要配置多个主机，请为每个主机创建一个仪表板小部件字段对象。
严重性		0	severities	0 - 未分类； 1 - 信息； 2 - 警告； 3 - 平均； 4 - 高； 5 - 灾难。 默认值：空（全部启用）。
问题		1	problem_name	注意：要配置多个值，请为每个值创建一个仪表板小部件字段对象。 问题事件名称（不区分大小写，全名或部分名称）。
标签（属性名称中的数字（例如 tags.tag.0）引用标签评估列表中的标签顺序）				
	评估类型	0	evaltype	0 - *(*默认*)* 与/或； 2 - 或。
	标签名称	1	tags.tag.0	任何字符串值。 如果配置标签，则需要参数标签名称。

参数		类型	名称	值
	操作符	0	tags.operator.0	0 - 包含； 1 - 等于； 2 - 不包含； 3 - 不等于； 4 - 存在； 5 - 不存在。 如果配置标签，则需要参数操作符。
	标签值	1	tags.value.0	任何字符串值。 如果配置标签，则需要参数标签值。

覆盖设置

以下参数支持配置覆盖设置。

Note:
属性名中的第一个数字（例如，or.hosts.0.0，or.items.0.0）表示特定的数据集，如果存在第二个数字，则表示已配置的主机或监控项。

参数	类型	名称	值
主机模式 (必填项)	1	or.hosts.0.0	主机名称或模式（例如，Zabbix*）。
监控项模式 (必填项)	1	or.items.0.0	监控项名称或模式（例如，*: Number of processed *values per second）。
基础颜色	1	or.color.0	十六进制颜色代码（例如，FF0000）。
宽度	0	or.width.0	有效值范围从 1-10。
绘制方式	0	or.type.0	0 - 线条； 1 - 点； 2 - 阶梯； 3 - 条形。
透明度	0	or.transparency.0	有效值范围从 1-10。
填充	0	or.fill.0	有效值范围从 1-10。
点大小	0	or.pointsize.0	有效值范围从 1-10。
缺失数据	0	or.missingdatafunc.0	0 - 无； 1 - 连接； 2 - 当作 0 处理。
Y 轴	0	or.axisy.0	0 - 左侧； 1 - 右侧。
时间偏移	1	or.timeshift.0	有效的时间字符串（例如，3600，1h 等）。 您可以使用时间后缀。允许使用负值。

示例

以下示例旨在仅描述图表仪表盘小部件的字段对象配置。有关配置仪表盘的更多信息，请参阅dashboard.create。

配置图表小部件

按以下方式配置图表小部件：

- 一个主机上的总共 9 个监控项分为 4 个数据集。
- 每个数据集由一条具有自定义颜色、宽度、透明度和填充的线条组成。
- 数据集 4 配置了聚合。
- 图表中的数据显示过去 3 小时的时间段。
- 图表中的问题仅针对配置的监控项显示。
- 图表有两个 Y 轴，其中右侧 Y 轴仅显示数据集 4 的值。
- 图表图例以 2 行显示配置的监控项。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "svggraph",
                        "name": "Graph",
                        "x": 0,
                        "y": 0,
                        "width": 12,
                        "height": 5,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 1,
                                "name": "ds.hosts.0.0",
                                "value": "Zabbix server"
                            },
                            {
                                "type": 1,
                                "name": "ds.items.0.0",
                                "value": "Zabbix server: Utilization of poller data collector processes, i
                            },
                            {
                                "type": 1,
                                "name": "ds.color.0",
                                "value": "FF0000"
                            },
                            {
                                "type": 0,
                                "name": "ds.width.0",
                                "value": 3
                            },
                            {
                                "type": 0,
                                "name": "ds.transparency.0",
                                "value": 3
                            },
                            {
                                "type": 0,
                                "name": "ds.fill.0",
                                "value": 1
                            },
                            {
                                "type": 1,
                                "name": "ds.hosts.1.0",
                                "value": "Zabbix server"
                            },
                            {
                                "type": 1,
                                "name": "ds.items.1.0",
                                "value": "Zabbix server: Utilization of trapper data collector processes,
                            },
                            {
                                "type": 1,
                                "name": "ds.color.1",
```

```
                "value": "BF00FF"
            },
            {
                "type": 0,
                "name": "ds.width.1",
                "value": 3
            },
            {
                "type": 0,
                "name": "ds.transparency.1",
                "value": 3
            },
            {
                "type": 0,
                "name": "ds.fill.1",
                "value": 1
            },
            {
                "type": 1,
                "name": "ds.hosts.2.0",
                "value": "Zabbix server"
            },
            {
                "type": 1,
                "name": "ds.items.2.0",
                "value": "Zabbix server: Utilization of history syncer internal processes,
            },
            {
                "type": 1,
                "name": "ds.color.2",
                "value": "0040FF"
            },
            {
                "type": 0,
                "name": "ds.width.2",
                "value": 3
            },
            {
                "type": 0,
                "name": "ds.transparency.2",
                "value": 3
            },
            {
                "type": 0,
                "name": "ds.fill.2",
                "value": 1
            },
            {
                "type": 1,
                "name": "ds.hosts.3.0",
                "value": "Zabbix server"
            },
            {
                "type": 1,
                "name": "ds.items.3.0",
                "value": "*: Number of processed *values per second"
            },
            {
                "type": 1,
                "name": "ds.color.3",
                "value": "000000"
            },
```

```
                        {
                            "type": 0,
                            "name": "ds.transparency.3",
                            "value": 0
                        },
                        {
                            "type": 0,
                            "name": "ds.fill.3",
                            "value": 0
                        },
                        {
                            "type": 0,
                            "name": "ds.axisy.3",
                            "value": 1
                        },
                        {
                            "type": 0,
                            "name": "ds.aggregate_function.3",
                            "value": 3
                        },
                        {
                            "type": 1,
                            "name": "ds.aggregate_interval.3",
                            "value": "1m"
                        },
                        {
                            "type": 0,
                            "name": "ds.aggregate_grouping.3",
                            "value": 1
                        },
                        {
                            "type": 0,
                            "name": "graph_time",
                            "value": 1
                        },
                        {
                            "type": 1,
                            "name": "time_from",
                            "value": "now-3h"
                        },
                        {
                            "type": 0,
                            "name": "legend_lines",
                            "value": 2
                        },
                        {
                            "type": 0,
                            "name": "show_problems",
                            "value": 1
                        }
                    ]
                }
            ]
        }
    ],
    "userGroups": [
        {
            "usrgrpid": 7,
            "permission": 2
        }
    ],
    "users": [
```

```
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

另请参阅

- 仪表盘小部件字段
- dashboard.create
- dashboard.update

9 图表 (经典)

描述

这些参数和相应仪表盘小部件字段对象的可能属性值允许在 dashboard.create 和 dashboard.update 方法中配置图表 *(经典)*小部件。

Attention:
在创建或更新仪表盘时，不会验证小部件 fields 属性。这允许用户修改内置小部件并创建自定义小部件，但同时也引入了创建或更新小部件时出错的风险。为确保成功创建或更新图表 *(经典)* 小部件，请参考下表概述的参数要求。

参数

以下参数支持 图表 *(经典)* 仪表盘小部件。

参数	类型	名称	值
刷新间隔	0	rf_rate	0 - 不刷新； 10 - 10 秒； 30 - 30 秒； 60 - *(默认)* 1 分钟； 120 - 2 分钟； 600 - 10 分钟； 900 - 15 分钟。
来源	0	source_type	0 - *(默认)* 图形； 1 - 简单图形。
图形	6	graphid	图形 ID。 如果 来源设置为" 图形"，则需要参数 图形。
监控项	4	itemid	监控项 ID。 如果 来源设置为" 简单图形"，则需要参数 监控项。
显示图例	0	show_legend	0 - 禁用； 1 - *(默认)* 启用。

参数	类型	名称	值
动态监控项	0	dynamic	0 - *(默认)* 禁用； 1 - 启用。

示例

以下示例旨在仅描述 图表 *(经典)* 仪表盘小部件字段对象的配置。有关配置仪表盘的更多信息，请参见dashboard.create。

配置 图表 *(经典)* 小部件

配置一个 图表 *(经典)* 小部件，用于显示监控项"42269" 的简单图形。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "graph",
                        "name": "Graph (classic)",
                        "x": 0,
                        "y": 0,
                        "width": 12,
                        "height": 5,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 0,
                                "name": "source_type",
                                "value": 1
                            },
                            {
                                "type": 4,
                                "name": "itemid",
                                "value": 42269
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
```

```
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

另请参阅

- 仪表盘小部件字段
- dashboard.create
- dashboard.update

10 图表原型

描述

这些参数以及相应仪表盘小部件字段对象可能的属性值允许在 dashboard.create 和 dashboard.update 方法中配置图表原型 小部件。

Attention:
在创建或更新仪表盘时，小部件 fields 属性不会被验证。这允许用户修改内置小部件 并创建自定义小部件，但同时也引入了错误创建或更新小部件的风险。为确保成功创建或更新 图表原型小部件，请参考下表中概述的参数要求。

参数

以下参数支持图表原型小部件。

参数	类型	名称	值
刷新间隔	0	rf_rate	0 - 不刷新； 10 - 10 秒； 30 - 30 秒； 60 - *(默认)* 1 分钟； 120 - 2 分钟； 600 - 10 分钟； 900 - 15 分钟。
来源	0	source_type	2 - *(默认)* 图表原型； 3 - 简单图表原型。
图表原型	7	graphid	图表原型 ID。 如果来源设置为” 图表原型”，需要参数图表原型。
监控项原型	5	itemid	监控项原型 ID。 如果来源设置为” 简单图表原型”，需要参数监控项原型。
显示图例	0	show_legend	0 - 禁用； 1 - *(默认)* 启用。
动态监控项	0	dynamic	0 - *(默认)* 禁用； 1 - 启用。
列	0	columns	有效值范围从 1-24。 默认：2。
行	0	rows	有效值范围从 1-16。 默认：1。

示例

以下示例旨在仅描述图表原型小部件的仪表盘小部件字段对象的配置。有关配置仪表盘的更多信息，请参见dashboard.create。

配置图表 *(经典)* 小部件

配置一个图表 *(经典)* 小部件，它显示由低级发现创建的 3 个图表（3 列，1 行）的网格，这些图表基于监控项原型（ID: ”42316”）。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "graphprototype",
                        "name": "Graph prototype",
                        "x": 0,
                        "y": 0,
                        "width": 16,
                        "height": 5,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 0,
                                "name": "source_type",
                                "value": 3
                            },
                            {
                                "type": 5,
                                "name": "itemid",
                                "value": 42316
                            },
                            {
                                "type": 0,
                                "name": "columns",
                                "value": 3
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

另请参阅

- 仪表盘小部件字段
- dashboard.create
- dashboard.update

11 Host availability

Description

These parameters and the possible property values for the respective dashboard widget field objects allow to configure the *Host availability* widget in dashboard.create and dashboard.update methods.

Parameters

The following parameters are supported for the *Host availability* widget.

Parameter	type	name	value
Refresh interval	0	rf_rate	0 - No refresh; 10 - 10 seconds; 30 - 30 seconds; 60 - 1 minute; 120 - 2 minutes; 600 - 10 minutes; 900 - *(default)* 15 minutes.
Host groups	2	groupids	Host group ID. Note: To configure multiple host groups, create a dashboard widget field object for each host group.
Interface type	0	interface_type	0 - None; 1 - Zabbix agent; 2 - SNMP; 3 - IPMI; 4 - JMX. Default: 1, 2, 3, 4 (all enabled). Note: To configure multiple values, create a dashboard widget field object for each value.
Layout	0	layout	0 - *(default)* Horizontal; 1 - Vertical.
Show hosts in maintenance	0	maintenance	0 - *(default)* Disabled; 1 - Enabled.

Examples

The following examples aim to only describe the configuration of the dashboard widget field objects for the *Host availability* widget. For more information on configuring a dashboard, see dashboard.create.

Configuring a *Host availability* widget

Configure a *Host availability* widget that displays availability information (in a vertical layout) for hosts in host group ”4” with ”Zabbix agent” and ”SNMP” interfaces configured.

Request:

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "hostavail",
                        "name": "Host availability",
                        "x": 0,
                        "y": 0,
                        "width": 6,
                        "height": 3,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 2,
                                "name": "groupids",
                                "value": 4
                            },
                            {
                                "type": 0,
                                "name": "interface_type",
                                "value": 1
                            },
                            {
                                "type": 0,
                                "name": "interface_type",
                                "value": 2
                            },
                            {
                                "type": 0,
                                "name": "layout",
                                "value": 1
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

See also

- Dashboard widget field
- dashboard.create
- dashboard.update

12 监控项值

描述

这些参数及其可能的属性值允许配置 dashboard.create 和 dashboard.update 方法中的监控项值小部件。

Attention:
在创建或更新仪表盘时，不会验证小部件 fields 属性。这允许用户修改内置小部件并创建自定义小部件，但同时也引入了错误创建或更新小部件的风险。为确保监控项值小部件的成功创建或更新，请参考下表中概述的参数要求。

参数

以下参数支持监控项值小部件。

参数	类型	名称	值
刷新间隔	0	rf_rate	0 - 不刷新； 10 - 10 秒； 30 - 30 秒； 60 - *(默认)* 1 分钟； 120 - 2 分钟； 600 - 10 分钟； 900 - 15 分钟。
监控项 (必需)	4	itemid	监控项 ID。
显示	0	show	1 - 描述； 2 - 值； 3 - 时间； 4 - 变化指示器。 默认：1, 2, 3, 4（全部启用）。 注意：要配置多个值，请为每个值创建一个仪表盘小部件字段对象。
动态监控项	0	dynamic	0 - *(默认)* 禁用； 1 - 启用。
高级配置	0	adv_conf	0 - *(默认)* 禁用； 1 - 启用。

高级配置

如果高级配置设置为”启用”，则支持以下参数。

参数	类型	名称	值
背景颜色	1	bg_color	十六进制颜色代码（例如 FF0000）。 默认："""（空）。

描述

如果高级配置设置为" 启用"，并且显示设置为" 描述"，则支持以下参数。

参数	类型	名称	值
描述	1	description	任何字符串值，包括宏。 支持的宏：{HOST.*}, {ITEM.*}, {INVENTORY.*}, 用户宏。 默认：{ITEM.NAME}。
水平位置	0	desc_h_pos	0 - 左； 1 - *(默认)* 中心； 2 - 右。 两个或更多的元素（描述、值、时间）不能共享相同的水平位置和垂直位置。
垂直位置	0	desc_v_pos	0 - 顶部； 1 - 中间； 2 - *(默认)* 底部。 两个或更多的元素（描述、值、时间）不能共享相同的水平位置和垂直位置。
大小	0	desc_size	有效值范围从 1-100。 默认：15。
加粗	0	desc_bold	0 - *(默认)* 禁用； 1 - 启用。
颜色	1	desc_color	十六进制颜色代码（例如 FF0000）。 默认："""（空）。

值

如果高级配置设置为" 启用"，并且显示设置为" 值"，则支持以下参数。

参数	类型	名称	值
小数位	0	decimal_places	有效值范围从 1-10。 默认：2。
大小	0	decimal_size	有效值范围从 1-100。 默认：35。
水平位置	0	value_h_pos	0 - 左； 1 - *(默认)* 中心； 2 - 右。 两个或更多的元素（描述、值、时间）不能共享相同的水平位置和垂直位置。
垂直位置	0	value_v_pos	0 - 顶部； 1 - *(默认)* 中间； 2 - 底部。 两个或更多的元素（描述、值、时间）不能共享相同的水平位置和垂直位置。
大小	0	value_size	有效值范围从 1-100。 默认：45。
加粗	0	value_bold	0 - 禁用； 1 - *(默认)* 启用。

参数	类型	名称	值
颜色	1	value_color	十六进制颜色代码（例如 FF0000）。 默认：""（空）。
单位(复选框)	0	units_show	0 - 禁用； 1 - *(默认)* 启用。
单位(值)	1	units	任何字符串值。
位置	0	units_pos	0 - 值之前； 1 - 值上方； 2 - *(默认)* 值之后； 3 - 值下方。
大小	0	units_size	有效值范围从 1-100。 默认：35。
加粗	0	units_bold	0 - 禁用； 1 - *(默认)* 启用。
颜色	1	units_color	十六进制颜色代码（例如 FF0000）。 默认：""（空）。

时间

如果高级配置设置为” 启用”，并且显示设置为” 时间”，则支持以下参数。

参数	类型	名称	值
水平位置	0	time_h_pos	0 - 左； 1 - *(默认)* 中心； 2 - 右。
垂直位置	0	time_v_pos	两个或更多的元素（描述、值、时间）不能共享相同的水平位置和垂直位置。 0 - *(默认)* 顶部； 1 - 中间； 2 - 底部。
大小	0	time_size	两个或更多的元素（描述、值、时间）不能共享相同的水平位置和垂直位置。 有效值范围从 1-100。 默认：15。
加粗	0	time_bold	0 - *(默认)* 禁用； 1 - 启用。
颜色	1	time_color	十六进制颜色代码（例如 FF0000）。 默认：""（空）。

变化指示器

如果高级配置设置为” 启用”，并且显示设置为” 变化指示器”，则支持以下参数。

参数	类型	名称	值
上升变化指示器颜色	1	up_color	十六进制颜色代码（例如 FF0000）。 默认：""（空）。
下降变化指示器颜色	1	down_color	十六进制颜色代码（例如 FF0000）。 默认：""（空）。

参数	类型	名称	值
无变化变化指示器颜色	1	updown_color	十六进制颜色代码（例如 FF0000）。 默认："" （空）。

示例

以下示例旨在仅描述仪表盘小部件字段对象的配置，用于监控项值小部件。有关配置仪表盘的更多信息，请参见dashboard.create。

配置一个监控项值小部件

配置一个监控项值小部件，用于展示项目 “42266” 的 Zabbix agent 可用性。此外，还可以使用多个高级选项对仪表盘小部件进行视觉微调。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "item",
                        "name": "Item value",
                        "x": 0,
                        "y": 0,
                        "width": 4,
                        "height": 3,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 4,
                                "name": "itemid",
                                "value": 42266
                            },
                            {
                                "type": 0,
                                "name": "show",
                                "value": 1
                            },
                            {
                                "type": 0,
                                "name": "show",
                                "value": 2
                            },
                            {
                                "type": 0,
                                "name": "show",
                                "value": 3
                            },
                            {
                                "type": 0,
                                "name": "adv_conf",
                                "value": 1
                            },
                            {
                                "type": 1,
                                "name": "bg_color",
                                "value": "D1C4E9"
                            },
```

```
                {
                    "type": 1,
                    "name": "description",
                    "value": "Agent status"
                },
                {
                    "type": 0,
                    "name": "desc_h_pos",
                    "value": 0
                },
                {
                    "type": 0,
                    "name": "desc_v_pos",
                    "value": 0
                },
                {
                    "type": 0,
                    "name": "desc_bold",
                    "value": 1
                },
                {
                    "type": 1,
                    "name": "desc_color",
                    "value": "F06291"
                },
                {
                    "type": 0,
                    "name": "value_h_pos",
                    "value": 0
                },
                {
                    "type": 0,
                    "name": "value_size",
                    "value": 25
                },
                {
                    "type": 1,
                    "name": "value_color",
                    "value": "FFFF00"
                },
                {
                    "type": 0,
                    "name": "units_show",
                    "value": 0
                },
                {
                    "type": 0,
                    "name": "time_h_pos",
                    "value": 2
                },
                {
                    "type": 0,
                    "name": "time_v_pos",
                    "value": 2
                },
                {
                    "type": 0,
                    "name": "time_size",
                    "value": 10
                },
                {
                    "type": 0,
```

```
                                "name": "time_bold",
                                "value": 1
                            },
                            {
                                "type": 1,
                                "name": "time_color",
                                "value": "9FA8DA"
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

另请参阅

- 仪表盘小部件字段
- dashboard.create
- dashboard.update

13 拓扑图

描述

这些参数及其可能的属性值允许配置 dashboard.create 和 dashboard.update 方法中的拓扑图小部件。

Attention:
在创建或更新仪表盘时，不会验证小部件 fields 属性。这允许用户修改内置小部件并创建自定义小部件，但同时也引入了错误创建或更新小部件的风险。为确保拓扑图小部件的成功创建或更新，请参考下表中概述的参数要求。

参数

以下参数支持拓扑图小部件。

参数	类型	名称	值
刷新间隔	0	rf_rate	0 - 不刷新； 10 - 10 秒； 30 - 30 秒； 60 - 1 分钟； 120 - 2 分钟； 600 - 10 分钟； 900 - *(默认)* 15 分钟。
源类型	0	source_type	1 - *(默认)* 拓扑图； 2 - 拓扑图导航树。
拓扑图	8	sysmapid	拓扑图 ID。 如果源类型设置为” 拓扑图”，则需要参数拓扑图。
链接小部件引用	1	filter_widget_reference	有效的拓扑图导航树小部件参数引用值。 如果源类型设置为” 拓扑图导航树”，则需要参数链接小部件引用。

示例

以下示例旨在仅描述拓扑图小部件的仪表盘小部件字段对象的配置。有关配置仪表盘的更多信息，请参见dashboard.create。

配置一个拓扑图小部件

配置一个拓扑图小部件，用于展示地图”1”。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "map",
                        "name": "Map",
                        "x": 0,
                        "y": 0,
                        "width": 18,
                        "height": 5,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 8,
                                "name": "sysmapid",
                                "value": 1
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
```

```
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

配置一个链接的拓扑图小部件

配置一个拓扑图小部件，该小部件链接到一个拓扑图导航树小部件。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "map",
                        "name": "Map",
                        "x": 0,
                        "y": 5,
                        "width": 18,
                        "height": 5,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 0,
                                "name": "source_type",
                                "value": 2
                            },
                            {
                                "type": 1,
                                "name": "filter_widget_reference",
                                "value": "ABCDE"
                            }
                        ]
                    },
                    {
                        "type": "navtree",
                        "name": "Map navigation tree",
                        "x": 0,
                        "y": 0,
                        "width": 6,
```

```
                "height": 5,
                "view_mode": 0,
                "fields": [
                    {
                        "type": 1,
                        "name": "navtree.name.1",
                        "value": "Element A"
                    },
                    {
                        "type": 1,
                        "name": "navtree.name.2",
                        "value": "Element B"
                    },
                    {
                        "type": 1,
                        "name": "navtree.name.3",
                        "value": "Element C"
                    },
                    {
                        "type": 1,
                        "name": "navtree.name.4",
                        "value": "Element A1"
                    },
                    {
                        "type": 1,
                        "name": "navtree.name.5",
                        "value": "Element A2"
                    },
                    {
                        "type": 1,
                        "name": "navtree.name.6",
                        "value": "Element B1"
                    },
                    {
                        "type": 1,
                        "name": "navtree.name.7",
                        "value": "Element B2"
                    },
                    {
                        "type": 0,
                        "name": "navtree.parent.4",
                        "value": 1
                    },
                    {
                        "type": 0,
                        "name": "navtree.parent.5",
                        "value": 1
                    },
                    {
                        "type": 0,
                        "name": "navtree.parent.6",
                        "value": 2
                    },
                    {
                        "type": 0,
                        "name": "navtree.parent.7",
                        "value": 2
                    },
                    {
                        "type": 0,
                        "name": "navtree.order.1",
                        "value": 1
```

```
                            },
                            {
                                "type": 0,
                                "name": "navtree.order.2",
                                "value": 2
                            },
                            {
                                "type": 0,
                                "name": "navtree.order.3",
                                "value": 3
                            },
                            {
                                "type": 0,
                                "name": "navtree.order.4",
                                "value": 1
                            },
                            {
                                "type": 0,
                                "name": "navtree.order.5",
                                "value": 2
                            },
                            {
                                "type": 0,
                                "name": "navtree.order.6",
                                "value": 1
                            },
                            {
                                "type": 0,
                                "name": "navtree.order.7",
                                "value": 2
                            },
                            {
                                "type": 8,
                                "name": "navtree.sysmapid.6",
                                "value": 1
                            },
                            {
                                "type": 1,
                                "name": "reference",
                                "value": "ABCDE"
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

参见

- 仪表盘小部件字段
- dashboard.create
- dashboard.update
- 拓扑图导航树

14 拓扑图导航树

描述

这些参数及其可能的属性值允许配置 dashboard.create 和 dashboard.update 方法中的拓扑图导航树小部件。

Attention:
在创建或更新仪表盘时，不会验证小部件 fields 属性。这允许用户修改内置小部件并创建自定义小部件，但同时也引入了错误创建或更新小部件的风险。为确保拓扑图导航树小部件的成功创建或更新，请参考下表中概述的参数要求。

参数

以下参数支持拓扑图导航树小部件。

参数	类型	名称	值
刷新间隔	0	rf_rate	0 - 不刷新； 10 - 10 秒； 30 - 30 秒； 60 - 1 分钟； 120 - 2 分钟； 600 - 10 分钟； 900 - *(默认)* 15 分钟。
显示不可用地图	1	show_unavailable	0 - *(默认)* 禁用； 1 - 启用。
引用	1	reference	由 5 个字符组成的任何字符串值（例如 ABCDE, JBPNL 等）。 参数 引用值在拓扑图 小部件（链接小部件引用）中使用，用于与 拓扑图导航树小部件链接。

以下参数支持配置拓扑图导航树元素。

参数	类型	名称	值
名称	1	navtree.name.1	任何字符串值。
链接的地图	8	navtree.sysmapid.1	注意：属性名称中的数字设置元素编号。 地图 ID。
创建元素层级的参数	0	navtree.parent.1	注意：属性名称中的数字引用链接到地图的元素。 父元素编号。 注意：属性名称中的数字引用子元素。属性值引用父元素。

参数	类型	名称	值
	0	navtree.order.1	拓扑图导航树中的元素位置。 注意：属性名称中的数字引用元素编号。属性值引用拓扑图导航树中的元素位置。父元素位置在整个拓扑图导航树中确定。子元素位置在父元素内确定。

示例

以下示例旨在描述拓扑图导航树小部件的仪表盘配置。更多仪表盘配置信息，请参考dashboard.create。

配置一个拓扑图导航树小部件

配置一个拓扑图导航树小部件，展示以下拓扑图导航树结构：

- 元素 A
  - 子元素 A1
  - 子元素 A2
- 元素 B
  - 子元素 B1（包含可在一个链接的拓扑图小部件中展示的链接地图"1"）
  - 子元素 B2
- 元素 C

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "navtree",
                        "name": "Map navigation tree",
                        "x": 0,
                        "y": 0,
                        "width": 6,
                        "height": 5,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 1,
                                "name": "navtree.name.1",
                                "value": "Element A"
                            },
                            {
                                "type": 1,
                                "name": "navtree.name.2",
                                "value": "Element B"
                            },
                            {
                                "type": 1,
                                "name": "navtree.name.3",
                                "value": "Element C"
                            },
                            {
                                "type": 1,
                                "name": "navtree.name.4",
                                "value": "Element A1"
                            },
                            {
```

```
            "type": 1,
            "name": "navtree.name.5",
            "value": "Element A2"
        },
        {
            "type": 1,
            "name": "navtree.name.6",
            "value": "Element B1"
        },
        {
            "type": 1,
            "name": "navtree.name.7",
            "value": "Element B2"
        },
        {
            "type": 0,
            "name": "navtree.parent.4",
            "value": 1
        },
        {
            "type": 0,
            "name": "navtree.parent.5",
            "value": 1
        },
        {
            "type": 0,
            "name": "navtree.parent.6",
            "value": 2
        },
        {
            "type": 0,
            "name": "navtree.parent.7",
            "value": 2
        },
        {
            "type": 0,
            "name": "navtree.order.1",
            "value": 1
        },
        {
            "type": 0,
            "name": "navtree.order.2",
            "value": 2
        },
        {
            "type": 0,
            "name": "navtree.order.3",
            "value": 3
        },
        {
            "type": 0,
            "name": "navtree.order.4",
            "value": 1
        },
        {
            "type": 0,
            "name": "navtree.order.5",
            "value": 2
        },
        {
            "type": 0,
            "name": "navtree.order.6",
```

```
                                "value": 1
                            },
                            {
                                "type": 0,
                                "name": "navtree.order.7",
                                "value": 2
                            },
                            {
                                "type": 8,
                                "name": "navtree.sysmapid.6",
                                "value": 1
                            },
                            {
                                "type": 1,
                                "name": "reference",
                                "value": "ABCDE"
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

参见

- 仪表盘小部件字段
- dashboard.create
- dashboard.update
- 拓扑图

15 纯文本

描述

这些参数及其可能的属性值允许配置 dashboard.create 和 dashboard.update 方法中的纯文本小部件。

Attention:
在创建或更新仪表盘时，不会验证小部件 `fields` 属性。这允许用户修改内置小部件并创建自定义小部件，但同时也引入了错误创建或更新小部件的风险。为确保纯文本小部件的成功创建或更新，请参考下表中概述的参数要求。

参数

以下参数支持纯文本小部件。

参数	类型	名称	值
刷新间隔	0	rf_rate	0 - 不刷新； 10 - 10 秒； 30 - 30 秒； 60 - *(默认)* 1 分钟； 120 - 2 分钟； 600 - 10 分钟； 900 - 15 分钟。
监控项 (必需)	4	itemids	监控项 ID。 注意：要配置多个监控项，为每个监控项创建一个仪表盘小部件字段对象。
监控项位置	0	style	0 - *(默认)* 左； 1 - 上。
显示行数	0	show_lines	有效值范围从 1-100。 默认：25。
将文本显示为 *HTML*	0	show_as_html	0 - *(默认)* 禁用； 1 - 启用。
动态监控项	0	dynamic	0 - *(默认)* 禁用； 1 - 启用。

示例

以下示例旨在仅描述纯文本仪表盘小部件的配置。有关配置仪表盘的更多信息，请参见`dashboard.create`。

配置一个纯文本小部件

配置一个纯文本小部件，用于展示监控项”42269” 和”42253” 的最新数据。此外，将监控项名称配置在数据列的顶部，并且只展示 15 行数据。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "plaintext",
                        "name": "Plain text",
                        "x": 0,
                        "y": 0,
                        "width": 6,
                        "height": 3,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 4,
                                "name": "itemids",
                                "value": 42269
```

```
                            },
                            {
                                "type": 4,
                                "name": "itemids",
                                "value": 42253
                            },
                            {
                                "type": 0,
                                "name": "style",
                                "value": 1
                            },
                            {
                                "type": 0,
                                "name": "show_lines",
                                "value": 15
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

另请参阅

- 仪表盘小部件字段
- dashboard.create
- dashboard.update

16 问题主机

描述

这些参数及其可能的属性值允许配置 dashboard.create 和 dashboard.update 方法中的问题主机小部件。

Attention:
在创建或更新仪表盘时，不会验证小部件 `fields` 属性。这允许用户修改内置小部件并创建自定义小部件，但同时也引入了错误创建或更新小部件的风险。为确保问题主机小部件的成功创建或更新，请参考下表中概述的参数要求。

参数

以下参数支持问题主机小部件。

参数	类型	名称	值
刷新间隔	0	rf_rate	0 - 不刷新； 10 - 10 秒； 30 - 30 秒； 60 - (默认) 1 分钟； 120 - 2 分钟； 600 - 10 分钟； 900 - 15 分钟。
主机组	2	groupids	主机组 ID。 注意：要配置多个主机组，为每个主机组创建一个仪表盘小部件字段对象。
排除主机组	2	exclude_groupids	主机组 ID。 注意：要排除多个主机组，为每个主机组创建一个仪表盘小部件字段对象。
主机	3	hostids	主机 ID。 注意：要配置多个主机，为每个主机创建一个仪表盘小部件字段对象。对于多个主机，参数 主机组必须要么完全未配置，要么配置了至少一个主机组，且配置的主机属于这些主机组。
问题	1	problem	问题事件名称（不区分大小写，可以是完整名称或部分名称）。
严重性	0	severities	0 - 未分类； 1 - 信息； 2 - 警告； 3 - 一般； 4 - 高； 5 - 灾难。 默认：空（全部启用）。 注意：要配置多个值，为每个值创建一个仪表盘小部件字段对象。

参数		类型	名称	值
标签(属性名称中的数字(例如tags.tag.0)引用标签评估列表中的标签顺序)				
	评估类型	0	evaltype	0 - (默认) 与/或； 2 - 或。
	标签名称	1	tags.tag.0	任何字符串值。 如果配置 标签，需要参数 标签名称。
	运算符	0	tags.operator.0	0 - 包含； 1 - 等于； 2 - 不包含； 3 - 不等于； 4 - 存在； 5 - 不存在。
	标签值	1	tags.value.0	如果配置 标签，需要参数 运算符。 任何字符串值。 如果配置 标签，需要参数 标签值。
显示被抑制的问题		0	show_suppressed	0 - (默认) 禁用； 1 - 启用。
隐藏无问题的组		0	hide_empty_groups	0 - (默认) 禁用； 1 - 启用。

参数	类型	名称	值
问题显示	0	ext_ack	0 - (默认) 全部； 1 - 仅未确认的； 2 - 分开的。

示例

以下示例旨在仅描述问题主机仪表盘小部件的配置。有关配置仪表盘的更多信息，请参见dashboard.create。

配置一个问题主机小部件

配置一个问题主机小部件，用于展示主机组”2” 和”4” 中的主机，这些主机存在包含字符串”CPU” 的名称的问题，并且具有以下严重性：”Warning”（警告），”Average”（一般），”High”（高），”Disaster”（灾难）。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "problemhosts",
                        "name": "Problem hosts",
                        "x": 0,
                        "y": 0,
                        "width": 12,
                        "height": 5,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 2,
                                "name": "groupids",
                                "value": 2
                            },
                            {
                                "type": 2,
                                "name": "groupids",
                                "value": 4
                            },
                            {
                                "type": 1,
                                "name": "problem",
                                "value": "cpu"
                            },
                            {
                                "type": 0,
                                "name": "severities",
                                "value": 2
                            },
                            {
                                "type": 0,
                                "name": "severities",
                                "value": 3
                            },
                            {
                                "type": 0,
```

```
                                "name": "severities",
                                "value": 4
                            },
                            {
                                "type": 0,
                                "name": "severities",
                                "value": 5
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

另请参阅

- 仪表盘小部件字段
- dashboard.create
- dashboard.update

17 问题

描述

这些参数及其可能的属性值允许配置 dashboard.create 和 dashboard.update 方法中的问题小部件。

> **Attention:**
> 在创建或更新仪表盘时，不会验证小部件 fields 属性。这允许用户修改内置小部件并创建自定义小部件，但同时也引入了错误创建或更新小部件的风险。为确保问题小部件的成功创建或更新，请参考下表中概述的参数要求。

参数

以下参数支持问题小部件。

参数	类型	名称	值
刷新间隔	0	rf_rate	0 - 不刷新； 10 - 10 秒； 30 - 30 秒； 60 - (默认) 1 分钟； 120 - 2 分钟； 600 - 10 分钟； 900 - 15 分钟。
显示	0	show	1 - (默认) 最近的问题； 2 - 历史； 3 - 问题。
主机组	2	groupids	主机组 ID。 注意：要配置多个主机组，为每个主机组创建一个仪表盘小部件字段对象。
排除主机组	2	exclude_groupids	主机组 ID。 注意：要排除多个主机组，为每个主机组创建一个仪表盘小部件字段对象。
主机	3	hostids	主机 ID。 注意：要配置多个主机，为每个主机创建一个仪表盘小部件字段对象。对于多个主机，参数 主机组要么完全未配置，要么配置了至少一个主机组，且配置的主机属于这些主机组。
问题	1	problem	问题事件名称（不区分大小写，完整名称或部分名称）。
严重性	0	severities	0 - 未分类； 1 - 信息； 2 - 警告； 3 - 一般； 4 - 高； 5 - 灾难。 默认：空（全部启用）。 注意：要配置多个值，为每个值创建一个仪表盘小部件字段对象。

参数		类型	名称	值
标签(属性名称中的数字(例如tags.tag.0)引用标签评估列表中的标签顺序)				
	评估类型	0	evaltype	0 - (默认) 与/或； 2 - 或。
	标签名称	1	tags.tag.0	任何字符串值。 如果配置 标签，需要参数 标签名称。
	运算符	0	tags.operator.0	0 - 包含； 1 - 等于； 2 - 不包含； 3 - 不等于； 4 - 存在； 5 - 不存在。 如果配置 标签，需要参数 运算符。
	标签值	1	tags.value.0	任何字符串值。 如果配置 标签，需要参数 标签值。
显示标签		0	show_tags	0 - (默认) 无； 1 - 1； 2 - 2； 3 - 3。
标签名称(格式)		0	tag_name_format	0 - (默认) 完整； 1 - 缩短； 2 - 无。 如果 显示标签设置为” 无”，则参数 标签名称（格式）不可用。

参数	类型	名称	值
标签显示优先级	1	tag_priority	标签的逗号分隔列表。 如果 显示标签设置为” 无”，则参数 标签显示优先级不可用。
显示运营数据	0	show_opdata	0 - (默认) 无； 1 - 分开； 2 - 与问题名称一起。
显示被抑制的问题	0	show_suppressed	0 - (默认) 禁用； 1 - 启用。
仅显示未确认的	0	unacknowledged	0 - (默认) 禁用； 1 - 启用。
按以下排序条目	0	sort_triggers	1 - 严重性（降序）； 2 - 主机（升序）； 3 - 时间（升序）； 4 - (默认) 时间（降序）； 13 - 严重性（升序）； 14 - 主机（降序）； 15 - 问题（升序）； 16 - 问题（降序）。 除了” 时间（降序）” 和” 时间（升序）”，参数 显示时间线必须设置为” 禁用”。
显示时间线	0	show_timeline	0 - 禁用； 1 - (默认) 启用。 如果 按以下排序条目设置为” 时间（降序）” 或” 时间（升序）”，则参数 显示时间线可用。
显示行数	0	show_lines	有效值范围从 1-100。 默认：25。

示例

以下示例旨在仅描述问题仪表盘小部件的配置。有关配置仪表盘的更多信息，请参见dashboard.create。

配置一个问题小部件

配置一个问题小部件，用于展示主机组”4” 中满足以下条件的问题：

- 具有名称为”scope” 的标签，包含值”performance” 或”availability”，或”capacity” 的问题。
- 具有以下严重性的问题：”Warning”（警告），”Average”（一般），”High”（过高），”Disaster”（灾难）。

此外，配置小部件以显示标签和操作数据。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "problems",
                        "name": "Problems",
                        "x": 0,
                        "y": 0,
                        "width": 12,
                        "height": 5,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 2,
                                "name": "groupids",
                                "value": 4
                            },
                            {
                                "type": 1,
                                "name": "tags.tag.0",
                                "value": "scope"
                            },
                            {
                                "type": 0,
                                "name": "tags.operator.0",
                                "value": 0
                            },
                            {
                                "type": 1,
                                "name": "tags.value.0",
                                "value": "performance"
                            },
                            {
                                "type": 1,
                                "name": "tags.tag.1",
                                "value": "scope"
                            },
                            {
                                "type": 0,
                                "name": "tags.operator.1",
                                "value": 0
                            },
                            {
                                "type": 1,
                                "name": "tags.value.1",
                                "value": "availability"
                            },
                            {
                                "type": 1,
                                "name": "tags.tag.2",
                                "value": "scope"
                            },
                            {
                                "type": 0,
                                "name": "tags.operator.2",
```

```
                                "value": 0
                            },
                            {
                                "type": 1,
                                "name": "tags.value.2",
                                "value": "capacity"
                            },
                            {
                                "type": 0,
                                "name": "severities",
                                "value": 2
                            },
                            {
                                "type": 0,
                                "name": "severities",
                                "value": 3
                            },
                            {
                                "type": 0,
                                "name": "severities",
                                "value": 4
                            },
                            {
                                "type": 0,
                                "name": "severities",
                                "value": 5
                            },
                            {
                                "type": 0,
                                "name": "show_tags",
                                "value": 1
                            },
                            {
                                "type": 0,
                                "name": "show_opdata",
                                "value": 1
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```


响应:


```
{
    "jsonrpc": "2.0",
```

```
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

另请参阅

- 仪表盘小部件字段
- dashboard.create
- dashboard.update

18 问题严重性

描述

这些参数及其可能的属性值允许配置 dashboard.create 和 dashboard.update 方法中的按严重性分类的问题小部件。

Attention:
在创建或更新仪表盘时，不会验证小部件 fields 属性。这允许用户修改内置小部件并创建自定义小部件，但同时也引入了错误创建或更新小部件的风险。为确保按严重性分类的问题小部件的成功创建或更新，请参考下表中概述的参数要求。

参数

以下参数支持按严重性分类的问题小部件。

参数	类型	名称	值
刷新间隔	0	rf_rate	0 - 不刷新； 10 - 10 秒； 30 - 30 秒； 60 - (默认) 1 分钟； 120 - 2 分钟； 600 - 10 分钟； 900 - 15 分钟。
主机组	2	groupids	主机组 ID。 注意：要配置多个主机组，为每个主机组创建一个仪表盘小部件字段对象。
排除主机组	2	exclude_groupids	主机组 ID。 注意：要排除多个主机组，为每个主机组创建一个仪表盘小部件字段对象。
主机	3	hostids	主机 ID。 注意：要配置多个主机，为每个主机创建一个仪表盘小部件字段对象。对于多个主机，参数 主机组必须要么完全未配置，要么配置了至少一个主机组，且配置的主机属于这些主机组。
问题	1	problem	问题事件名称（不区分大小写，可以是完整名称或部分名称）。

参数	类型	名称	值
严重性	0	severities	0 - 未分类； 1 - 信息； 2 - 警告； 3 - 一般； 4 - 高； 5 - 灾难。 默认：空（全部启用）。 注意：要配置多个值，为每个值创建一个仪表盘小部件字段对象。
标签(属性名称中的数字(例如tags.tag.0)引用标签评估列表中的标签顺序)			
评估类型	0	evaltype	0 - (默认) 与/或； 2 - 或。
标签名称	1	tags.tag.0	任何字符串值。 如果配置 标签，需要参数 标签名称。
运算符	0	tags.operator.0	0 - 包含； 1 - 等于； 2 - 不包含； 3 - 不等于； 4 - 存在； 5 - 不存在。 如果配置 标签，需要参数 运算符。
标签值	1	tags.value.0	任何字符串值。 如果配置 标签，需要参数 标签值。
显示	0	show_type	0 - (默认) 主机组； 1 - 总计。
布局	0	layout	0 - (默认) 水平； 1 - 垂直。 如果 显示设置为” 主机组”，则参数 布局不可用。

参数	类型	名称	值
显示运营数据	0	show_opdata	0 - (默认) 无； 1 - 分开； 2 - 与问题名称一起。
显示被抑制的问题	0	show_suppressed	0 - (默认) 禁用； 1 - 启用。
隐藏无问题的组	0	hide_empty_groups	0 - (默认) 禁用； 1 - 启用。 如果 显示设置为” 总计”，则参数 隐藏无问题的组不可用。
问题显示	0	ext_ack	0 - (默认) 全部； 1 - 仅未确认的； 2 - 分开的。
显示时间线	0	show_timeline	0 - 禁用； 1 - (默认) 启用。

示例

以下示例旨在仅描述按严重性分类的问题仪表盘小部件的配置。有关配置仪表盘的更多信息，请参见dashboard.create。

配置 按严重性分类的问题控件

配置一个 按严重性分类的问题控件，用于显示所有主机组的问题总数。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "problemsbysv",
                        "name": "Problems by severity",
                        "x": 0,
                        "y": 0,
                        "width": 12,
                        "height": 5,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 0,
```

```
                                "name": "show_type",
                                "value": 1
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```


响应:


```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```


另请参阅

- 仪表盘小部件字段
- dashboard.create
- dashboard.update

19 SLA 报表

描述

这些参数及其可能的属性值允许配置 dashboard.create 和 dashboard.update 方法中的*SLA 报表*小部件。

Attention:
在创建或更新仪表盘时，不会验证小部件 fields 属性。这允许用户修改内置小部件并创建自定义小部件，但同时也引入了错误创建或更新小部件的风险。为确保 *SLA* 报告小部件的成功创建或更新，请参考下表中概述的参数要求。

参数

以下参数支持 *SLA* 报表小部件。

参数	类型	名称	值
刷新间隔	0	rf_rate	0 - *(默认)* 不刷新； 10 - 10 秒； 30 - 30 秒； 60 - 1 分钟； 120 - 2 分钟； 600 - 10 分钟； 900 - 15 分钟。
SLA (必需)	10	slaid	SLA ID。
服务	9	serviceid	服务 ID。
显示周期	0	show_periods	有效值范围从 1-100。 默认：20。
从	1	date_from	格式为 YYYY-MM-DD 的有效日期字符串。 支持使用修饰符 d, w, M, y 的相对日期（例如 now, now/d, now/w-1w 等）。
至	1	date_to	格式为 YYYY-MM-DD 的有效日期字符串。 支持使用修饰符 d, w, M, y 的相对日期（例如 now, now/d, now/w-1w 等）。

示例

以下示例旨在仅描述 *SLA* 报表仪表盘小部件的配置。有关配置仪表盘的更多信息，请参见dashboard.create。

配置一个 *SLA* 报表小部件

配置一个 *SLA* 报表小部件，用于展示 SLA "4" 服务"2" 在过去 30 天期间的 SLA 报表。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "slareport",
                        "name": "SLA report",
                        "x": 0,
                        "y": 0,
                        "width": 12,
                        "height": 5,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 10,
                                "name": "slaid",
                                "value": 4
                            },
                            {
                                "type": 9,
                                "name": "serviceid",
                                "value": 2
                            },
                            {
                                "type": 1,
                                "name": "date_from",
                                "value": "now-30d"
                            },
```

```
                            {
                                "type": 1,
                                "name": "date_to",
                                "value": "now"
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

另请参阅

- 仪表盘小部件字段
- `dashboard.create`
- `dashboard.update`

20 System information

Description

These parameters and the possible property values for the respective dashboard widget field objects allow to configure the *System Information* widget in `dashboard.create` and `dashboard.update` methods.

Parameters

The following parameters are supported for the *System Information* widget.

Parameter	type	name	value
Refresh interval	0	rf_rate	0 - No refresh; 10 - 10 seconds; 30 - 30 seconds; 60 - 1 minute; 120 - 2 minutes; 600 - 10 minutes; 900 - *(default)* 15 minutes.
Show	0	info_type	0 - *(default)* System stats; 1 - High availability nodes.

Examples

The following examples aim to only describe the configuration of the dashboard widget field objects for the *System information* widget. For more information on configuring a dashboard, see `dashboard.create`.

Configuring a *System information* widget

Configure a *System information* widget that displays system stats with a refresh interval of 10 minutes.

Request:

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "systeminfo",
                        "name": "System information",
                        "x": 0,
                        "y": 0,
                        "width": 12,
                        "height": 5,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 0,
                                "name": "rf_rate",
                                "value": 600
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
```

```
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

See also

- Dashboard widget field
- dashboard.create
- dashboard.update

21 TOP 主机

描述

这些参数及其可能的属性值允许配置 dashboard.create 和 dashboard.update 方法中的*TOP* 主机仪表盘小部件。

Attention:
在创建或更新仪表盘时，不会验证小部件 fields 属性。这允许用户修改内置小部件并创建自定义小部件，但同时也引入了错误创建或更新小部件的风险。为确保 *TOP* 主机小部件的成功创建或更新，请参考下表中概述的参数要求。

参数

以下参数支持 *TOP* 主机小部件。

参数	类型	名称	值
刷新间隔	0	rf_rate	0 - 不刷新； 10 - 10 秒； 30 - 30 秒； 60 - (默认) 1 分钟； 120 - 2 分钟； 600 - 10 分钟； 900 - 15 分钟。
主机组	2	groupids	主机组 ID。 注意：要配置多个主机组，为每个主机组创建一个仪表盘小部件字段对象。
主机	3	hostids	主机 ID。 注意：要配置多个主机，为每个主机创建一个仪表盘小部件字段对象。对于多个主机，参数 主机组必须要么完全未配置，要么配置了至少一个主机组，且配置的主机属于这些主机组。

参数		类型	名称	值
主机标签(属性名称中的数字(例如tags.tag.0)引用标签评估列表中的标签顺序)				
	评估类型	0	evaltype	0 - (默认) 与/或； 2 - 或。
	标签名称	1	tags.tag.0	任何字符串值。 如果配置 标签，需要参数 标签名称。
	运算符	0	tags.operator.0	0 - 包含； 1 - 等于； 2 - 不包含； 3 - 不等于； 4 - 存在； 5 - 不存在。 如果配置 标签，需要参数 运算符。
	标签值	1	tags.value.0	任何字符串值。 如果配置 标签，需要参数 标签值。
列(见下文)				
顺序		0	order	2 - (默认) 顶部 N； 3 - 底部 N。
排序列		0	column	从配置列中的列数值。
主机数量		0	count	有效值范围从 1-100。 默认：10。

列

列具有通用参数以及根据参数数据配置而变化的附加参数。

Note:
对于所有与列相关的参数，属性名称中的数字（例如 columns.name.0）引用了为该参数配置的列。

对于所有列，支持以下参数。

参数	类型	名称	值
名称	1	columns.name.0	任何字符串值。
数据 (必需)	0	columns.data.0	1 - 监控项值； 2 - 主机名； 3 - 文本。
基础颜色 (必需)	1	columns.base_color.0	十六进制颜色代码（例如 FF0000）。

监控项值

如果数据设置为" 监控项值"，则支持以下参数。

Note:
阈值属性名称中的第一个数字（例如 columnsthresholds.color.0.0）引用了配置阈值的列，而第二个数字引用了列表中阈值的位置，按升序排序。然而，如果阈值配置的顺序不同，更新 Zabbix 前端小部件配置后的值将按升序排序（例如 `"threshold.threshold.0":"5"` → `"threshold.threshold.0":"1"`; `"threshold.threshold.1":"1"` → `"threshold.threshold.1": "5"`）。

参数	类型	名称	值
监控项	1	columns.item.0	有效的监控项名称。
时间偏移 (必需)	1	columns.timeshift.0	有效的数字或时间字符串值（例如 3600 或 1h）。 您可以使用时间后缀。允许负值。
聚合函数	0	columns.aggregate_function.0	0 - (默认) 无； 1 - 最小值； 2 - 最大值； 3 - 平均值； 4 - 计数； 5 - 求和； 6 - 第一个； 7 - 最后一个。
聚合间隔	1	columns.aggregate_interval.0	有效的时间字符串（例如 3600, 1h 等）。 您可以使用时间后缀。 如果 聚合函数设置为 无，则参数 聚合间隔不可用。 默认：1h。
显示	0	columns.display.0	1 - (默认) 原样； 2 - 条形图； 3 - 指示器。
最小值	1	columns.min.0	任何数值。 如果 显示设置为" 原样"，则参数 最小值不可用。
最大值	1	columns.max.0	任何数值。 如果 显示设置为" 原样"，则参数 最大值不可用。

参数	类型	名称	值
历史数据阈值	0	columns.history.0	1 - (默认) 自动； 2 - 历史记录； 3 - 趋势。
颜色	1	columnsthresholds.color.0	十六进制颜色代码（例如 FF0000）。 默认："（空）。
阈值	1	columnsthresholds.threshold.0	任何字符串值。

文本

如果数据设置为" 文本"，则支持以下参数。

参数	类型	名称	值
文本	1	columns.text.0	任何字符串值，包括宏。 支持的宏：{HOST.*}, {INVENTORY.*}*。 如果数据设置为" 文本"，则需要参数 文本。

示例

以下示例旨在仅描述 *TOP* 主机仪表盘小部件的配置。有关配置仪表盘的更多信息，请参见dashboard.create。

配置一个 *TOP* 主机小部件

配置一个 *TOP* 主机小部件，用于展示主机组"4" 中按 CPU 利用率排名的主机。此外，配置以下自定义列：" 主机名"，" 利用率"，"1 分钟平均"，"5 分钟平均"，"15 分钟平均"，" 进程数"。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "tophosts",
                        "name": "Top hosts",
                        "x": 0,
                        "y": 0,
                        "width": 12,
                        "height": 5,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 2,
                                "name": "groupids",
                                "value": 4
                            },
                            {
                                "type": 1,
                                "name": "columns.name.0",
                                "value": ""
```

```
                    },
                    {
                        "type": 0,
                        "name": "columns.data.0",
                        "value": 2
                    },
                    {
                        "type": 1,
                        "name": "columns.base_color.0",
                        "value": "FFFFFF"
                    },
                    {
                        "type": 1,
                        "name": "columns.timeshift.0",
                        "value": ""
                    },
                    {
                        "type": 1,
                        "name": "columns.name.1",
                        "value": "Utilization"
                    },
                    {
                        "type": 0,
                        "name": "columns.data.1",
                        "value": 1
                    },
                    {
                        "type": 1,
                        "name": "columns.base_color.1",
                        "value": "4CAF50"
                    },
                    {
                        "type": 1,
                        "name": "columns.timeshift.1",
                        "value": ""
                    },
                    {
                        "type": 1,
                        "name": "columns.item.1",
                        "value": "CPU utilization"
                    },
                    {
                        "type": 0,
                        "name": "columns.display.1",
                        "value": 3
                    },
                    {
                        "type": 1,
                        "name": "columns.min.1",
                        "value": "0"
                    },
                    {
                        "type": 1,
                        "name": "columns.max.1",
                        "value": "100"
                    },
                    {
                        "type": 1,
                        "name": "columnsthresholds.color.1.0",
                        "value": "FFFF00"
                    },
                    {
```

```
                "type": 1,
                "name": "columnsthresholds.threshold.1.0",
                "value": "50"
            },
            {
                "type": 1,
                "name": "columnsthresholds.color.1.1",
                "value": "FF8000"
            },
            {
                "type": 1,
                "name": "columnsthresholds.threshold.1.1",
                "value": "80"
            },
            {
                "type": 1,
                "name": "columnsthresholds.color.1.2",
                "value": "FF4000"
            },
            {
                "type": 1,
                "name": "columnsthresholds.threshold.1.2",
                "value": "90"
            },
            {
                "type": 1,
                "name": "columns.name.2",
                "value": "1m avg"
            },
            {
                "type": 0,
                "name": "columns.data.2",
                "value": 1
            },
            {
                "type": 1,
                "name": "columns.base_color.2",
                "value": "FFFFFF"
            },
            {
                "type": 1,
                "name": "columns.timeshift.2",
                "value": ""
            },
            {
                "type": 1,
                "name": "columns.item.2",
                "value": "Load average (1m avg)"
            },
            {
                "type": 1,
                "name": "columns.name.3",
                "value": "5m avg"
            },
            {
                "type": 0,
                "name": "columns.data.3",
                "value": 1
            },
            {
                "type": 1,
                "name": "columns.base_color.3",
```

```
            "value": "FFFFFF"
        },
        {
            "type": 1,
            "name": "columns.timeshift.3",
            "value": ""
        },
        {
            "type": 1,
            "name": "columns.item.3",
            "value": "Load average (5m avg)"
        },
        {
            "type": 1,
            "name": "columns.name.4",
            "value": "15m avg"
        },
        {
            "type": 0,
            "name": "columns.data.4",
            "value": 1
        },
        {
            "type": 1,
            "name": "columns.base_color.4",
            "value": "FFFFFF"
        },
        {
            "type": 1,
            "name": "columns.timeshift.4",
            "value": ""
        },
        {
            "type": 1,
            "name": "columns.item.4",
            "value": "Load average (15m avg)"
        },
        {
            "type": 1,
            "name": "columns.name.5",
            "value": "Processes"
        },
        {
            "type": 0,
            "name": "columns.data.5",
            "value": 1
        },
        {
            "type": 1,
            "name": "columns.base_color.5",
            "value": "FFFFFF"
        },
        {
            "type": 1,
            "name": "columns.timeshift.5",
            "value": ""
        },
        {
            "type": 1,
            "name": "columns.item.5",
            "value": "Number of processes"
        },
```

```
                            {
                                "type": 0,
                                "name": "column",
                                "value": 1
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

另请参阅

- 仪表盘小部件字段
- dashboard.create
- dashboard.update

22 触发器概览

描述

这些参数及其可能的属性值允许配置 dashboard.create 和 dashboard.update 方法中的触发器概览仪表盘小部件。

Attention:
在创建或更新仪表盘时，不会验证小部件 fields 属性。这允许用户修改内置小部件并创建自定义小部件，但同时也引入了错误创建或更新小部件的风险。为确保触发器概览小部件的成功创建或更新，请参考下表中概述的参数要求。

参数

以下参数支持触发器概览小部件。

参数	类型	名称	值
刷新间隔	0	rf_rate	0 - 不刷新； 10 - 10 秒； 30 - 30 秒； 60 - (默认) 1 分钟； 120 - 2 分钟； 600 - 10 分钟； 900 - 15 分钟。
显示	0	show	1 - (默认) 最近的问题； 2 - 任意； 3 - 问题。
主机组	2	groupids	主机组 ID。 注意：要配置多个主机组，为每个主机组创建一个仪表盘小部件字段对象。
主机	3	hostids	主机 ID。 注意：要配置多个主机，为每个主机创建一个仪表盘小部件字段对象。对于多个主机，参数 主机组必须要么完全未配置，要么配置了至少一个主机组，且配置的主机属于这些主机组。
标签(属性名称中的数字(例如 tags.tag.0)引用标签评估列表中的标签顺序)			
评估类型	0	evaltype	0 - (默认) 与/或； 2 - 或。
标签名称	1	tags.tag.0	任何字符串值。 如果配置 标签，需要参数 标签名称。
运算符	0	tags.operator.0	0 - 包含； 1 - 等于； 2 - 不包含； 3 - 不等于； 4 - 存在； 5 - 不存在。 如果配置 标签，需要参数 运算符。

参数	类型	名称	值
标签值	1	tags.value.0	任何字符串值。 如果配置 标签，需要参数 标签值。
显示被抑制的问题	0	show_suppressed	0 - (默认) 禁用； 1 - 启用。
主机位置	0	style	0 - (默认) 左侧； 1 - 顶部。

示例

以下示例旨在仅描述触发器概览仪表盘小部件的配置。有关配置仪表盘的更多信息，请参见dashboard.create。

配置一个触发器概览小部件

配置一个触发器概览小部件，用于展示所有具有名为”scope” 标签且包含值”availability” 的触发器的主机组的触发器状态。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "trigover",
                        "name": "Trigger overview",
                        "x": 0,
                        "y": 0,
                        "width": 12,
                        "height": 5,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 1,
                                "name": "tags.tag.0",
                                "value": "scope"
                            },
                            {
                                "type": 0,
                                "name": "tags.operator.0",
                                "value": 0
                            },
                            {
                                "type": 1,
                                "name": "tags.value.0",
                                "value": "availability"
                            }
                        ]
                    }
```

```
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

另请参阅

- 仪表盘小部件字段
- dashboard.create
- dashboard.update

23 URL

描述

这些参数及其可能的属性值允许配置 dashboard.create 和 dashboard.update 方法中的*URL*仪表盘小部件。

Attention:
在创建或更新仪表盘时，不会验证小部件 fields 属性。这允许用户修改内置小部件并创建自定义小部件，但同时也引入了错误创建或更新小部件的风险。为确保 *URL* 小部件的成功创建或更新，请参考下表中概述的参数要求。

参数

以下参数支持 *URL* 小部件。

参数	类型	名称	值
刷新间隔	0	rf_rate	0 - *(默认)* 不刷新； 10 - 10 秒； 30 - 30 秒； 60 - 1 分钟； 120 - 2 分钟； 600 - 10 分钟； 900 - 15 分钟。
URL (必需)	1	url	有效的 URL 字符串。

参数	类型	名称	值
动态项	0	dynamic	0 - *(默认)* 禁用； 1 - 启用。

示例

以下示例旨在仅描述 *URL* 仪表盘小部件的配置。有关配置仪表盘的更多信息，请参见dashboard.create。

配置一个 *URL* 小部件

配置一个 *URL* 小部件，用于展示 Zabbix 手册的首页。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "url",
                        "name": "URL",
                        "x": 0,
                        "y": 0,
                        "width": 12,
                        "height": 5,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 1,
                                "name": "url",
                                "value": "https://www.zabbix.com/documentation/6.0/en"
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
```

```
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

另请参阅

- 仪表盘小部件字段
- dashboard.create
- dashboard.update

24 Web 监测

描述

这些参数及其可能的属性值允许配置 dashboard.create 和 dashboard.update 方法中的*Web* 监控仪表盘小部件。

Attention:
在创建或更新仪表盘时，不会验证小部件 fields 属性。这允许用户修改内置小部件并创建自定义小部件，但同时也引入了错误创建或更新小部件的风险。为确保 *Web* 监控小部件的成功创建或更新，请参考下表中概述的参数要求。

参数

以下参数支持 *Web* 监控小部件。

参数	类型	名称	值
刷新间隔	0	rf_rate	0 - 不刷新； 10 - 10 秒； 30 - 30 秒； 60 - (默认) 1 分钟； 120 - 2 分钟； 600 - 10 分钟； 900 - 15 分钟。
主机组	2	groupids	主机组 ID。 注意：要配置多个主机组，为每个主机组创建一个仪表盘小部件字段对象。
排除主机组	2	exclude_groupids	主机组 ID。 注意：要排除多个主机组，为每个主机组创建一个仪表盘小部件字段对象。
主机	3	hostids	主机 ID。 注意：要配置多个主机，为每个主机创建一个仪表盘小部件字段对象。对于多个主机，参数 主机组必须要么完全未配置，要么配置了至少一个主机组，且配置的主机属于这些主机组。

参数		类型	名称	值
标签(属性名称中的数字(例如tags.tag.0)引用标签评估列表中的标签顺序)				
	评估类型	0	evaltype	0 - (默认) 与/或； 2 - 或。
	标签名称	1	tags.tag.0	任何字符串值。 如果配置 标签，需要参数 标签名称。
	运算符	0	tags.operator.0	0 - 包含； 1 - 等于； 2 - 不包含； 3 - 不等于； 4 - 存在； 5 - 不存在。
	标签值	1	tags.value.0	如果配置 标签，需要参数 运算符。 任何字符串值。 如果配置 标签，需要参数 标签值。
显示维护中的主机		0	maintenance	0 - 禁用； 1 - (默认) 启用。

示例

以下示例旨在仅描述 *Web* 监控仪表盘小部件字段对象的配置。有关配置仪表盘的更多信息，请参见dashboard.create。

配置 *Web* 监控小部件

配置一个 *Web* 监控小部件，用于显示主机组"4" 的活跃 Web 监控场景的状态摘要。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "web",
                        "name": "Web monitoring",
                        "x": 0,
                        "y": 0,
                        "width": 6,
                        "height": 3,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 2,
                                "name": "groupids",
                                "value": 4
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": 7,
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": 1,
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "3"
        ]
    },
    "id": 1
}
```

另请参阅

- 仪表盘小部件字段
- dashboard.create
- dashboard.update

创建

描述

`object dashboard.create(object/array dashboards)`

此方法允许创建新的仪表盘。

> **Note:**
> 此方法对于任何用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(object/array) 要创建的仪表盘。

除了标准仪表盘属性，此方法还接受以下参数：

参数	类型	说明
pages（必需）	array	仪表盘上创建的仪表盘页面 。仪表盘页面的顺序将与指定的顺序相同。pages 属性至少需要一个仪表盘页面对象。
users	array	仪表盘上创建的仪表盘用户 。
userGroups	array	仪表盘上创建的仪表盘用户组 。

返回值

(object) 返回一个对象，该对象包含 dashboardids 属性下创建的仪表盘的 ID。返回的 ID 的顺序与所传递的仪表盘的顺序相匹配。

示例

创建仪表盘

创建一个名为 “My dashboard” 的仪表盘，其中有一个带有标签的问题部件，并使用了两种类型的共享（用户组和用户）。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.create",
    "params": {
        "name": "My dashboard",
        "display_period": 30,
        "auto_start": 1,
        "pages": [
            {
                "widgets": [
                    {
                        "type": "problems",
                        "x": 0,
                        "y": 0,
                        "width": 12,
                        "height": 5,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 1,
                                "name": "tags.tag.0",
                                "value": "service"
                            },
                            {
                                "type": 0,
                                "name": "tags.operator.0",
                                "value": 1
                            },
                            {
                                "type": 1,
                                "name": "tags.value.0",
```

```
                                "value": "zabbix_server"
                            }
                        ]
                    }
                ]
            }
        ],
        "userGroups": [
            {
                "usrgrpid": "7",
                "permission": 2
            }
        ],
        "users": [
            {
                "userid": "4",
                "permission": 3
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "2"
        ]
    },
    "id": 1
}
```

参阅

- 仪表盘页面
- 仪表盘部件
- 仪表盘部件字段
- 仪表盘用户
- 仪表盘用户组

来源

ui/include/classes/api/services/CDashboard.php 中的 CDashboard::create()。

删除

描述

`object dashboard.delete(array dashboardids)`

此方法用于删除仪表盘。

> **Note:**
> 此方法对于任何用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(array) 要删除的仪表盘 ID。

返回值

(object) 返回一个对象，该对象包含 dashboardids 属性下删除的仪表盘 ID。

示例

删除多个仪表盘

删除两个仪表盘

请求:

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.delete",
    "params": [
        "2",
        "3"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "2",
            "3"
        ]
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CDashboard.php 中的 CDashboard::delete() 。

更新

描述

`object dashboard.update(object/array dashboards)`

此方法允许更新已有的仪表盘。

> **Note:**
> 此方法对于任何用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

`(object/array)` 要更新的仪表盘属性。

必须为每个仪表盘定义 `dashboardid` 属性，其它的属性都是可选的。只有指定的属性会被更新，其它属性都将保持不变。

除了标准仪表盘属性以外，此方法还接受如下参数：

属性	类型	说明
pages	array	更新的仪表盘页面。 仪表盘页面由 dashboard_pageid 属性更新。将为没有 dashboard_pageid 属性的对象创建新的仪表盘页面，如果不重新使用，现有仪表盘页面将被删除。仪表盘页面的顺序将与指定的顺序相同。只会更新仪表盘页面的指定属性。“pages”属性至少需要一个仪表盘页面对象。
users	array	更新的仪表盘用户。
userGroups	array	更新的仪表盘用户组。

返回值

（object）返回一个对象，该对象包含 dashboardids 属性下更新的仪表盘 ID。

示例

重命名一个仪表盘

重命名一个仪表盘为”SQL server status”。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.update",
    "params": {
        "dashboardid": "2",
        "name": "SQL server status"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "2"
        ]
    },
    "id": 1
}
```

更新仪表盘页面

重命名第一个仪表盘页面，替换第二个仪表盘页面上的部件，并添加一个新页面作为第三个页面。删除所有其他仪表盘页面。

请求：

```
{
    "jsonrpc": "2.0",
```

```
    "method": "dashboard.update",
    "params": {
        "dashboardid": "2",
        "pages": [
            {
                "dashboard_pageid": 1,
                "name": 'Renamed Page'
            },
            {
                "dashboard_pageid": 2,
                "widgets": [
                    {
                        "type": "clock",
                        "x": 0,
                        "y": 0,
                        "width": 4,
                        "height": 3
                    }
                ]
            },
            {
                "display_period": 60
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "2"
        ]
    },
    "id": 2
}
```

变更仪表盘所有者

仅适用于管理员以及超级管理员用户。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.update",
    "params": {
        "dashboardid": "2",
        "userid": "1"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 2
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "2"
        ]
```

```
    },
    "id": 2
}
```

参阅

- 仪表盘页面
- 仪表盘部件
- 仪表盘部件字段
- 仪表盘用户
- 仪表盘用户组

来源

ui/include/classes/api/services/CDashboard.php 中的 CDashboard::update() 。

获取

描述

`integer/array dashboard.get(object parameters)`

此方法允许根据给定的参数获取仪表盘。

Note:
此方法对于任何用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

`(object)` 定义需要输出的参数。

此方法支持以下参数。

属性	类型	描述
dashboardids	string/array	要获取的仪表盘 ID。
selectPages	query	返回按照正确排序的仪表盘页面的页面 属性。
selectUsers	query	返回共享仪表盘的用户的用户 属性。
selectUserGroups	query	返回共享仪表盘的用户组的用户组属性。
sortfield	string/array	根据给定的属性对结果进行排序。 可用值：`dashboardid`。
countOutput	boolean	这些参数对于所有 get 方法都是通用的，详情请参考参考评论
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

`(integer/array)` 返回其中一个：

- 一个对象数组；
- 如果使用了 countOutput 参数，将返回获取对象的数量。

示例

通过 ID 检索仪表盘

检索仪表盘”1” 和”2” 的所有数据。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dashboard.get",
    "params": {
        "output": "extend",
        "selectPages": "extend",
        "selectUsers": "extend",
        "selectUserGroups": "extend",
        "dashboardids": [
            "1",
            "2"
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "dashboardid": "1",
            "name": "Dashboard",
            "userid": "1",
            "private": "0",
            "display_period": "30",
            "auto_start": "1",
            "users": [],
            "userGroups": [],
            "pages": [
                {
                    "dashboard_pageid": "1",
                    "name": "",
                    "display_period": "0",
                    "widgets": [
                        {
                            "widgetid": "9",
                            "type": "systeminfo",
                            "name": "",
                            "x": "12",
                            "y": "8",
                            "width": "12",
                            "height": "5",
                            "view_mode": "0",
                            "fields": []
                        },
                        {
                            "widgetid": "8",
                            "type": "problemsbysv",
                            "name": "",
                            "x": "12",
                            "y": "4",
                            "width": "12",
                            "height": "4",
                            "view_mode": "0",
                            "fields": []
                        },
                        {
                            "widgetid": "7",
                            "type": "problemhosts",
                            "name": "",
```

```
                "x": "12",
                "y": "0",
                "width": "12",
                "height": "4",
                "view_mode": "0",
                "fields": []
            },
            {
                "widgetid": "6",
                "type": "discovery",
                "name": "",
                "x": "6",
                "y": "9",
                "width": "6",
                "height": "4",
                "view_mode": "0",
                "fields": []
            },
            {
                "widgetid": "5",
                "type": "web",
                "name": "",
                "x": "0",
                "y": "9",
                "width": "6",
                "height": "4",
                "view_mode": "0",
                "fields": []
            },
            {
                "widgetid": "4",
                "type": "problems",
                "name": "",
                "x": "0",
                "y": "3",
                "width": "12",
                "height": "6",
                "view_mode": "0",
                "fields": []
            },
            {
                "widgetid": "3",
                "type": "favmaps",
                "name": "",
                "x": "8",
                "y": "0",
                "width": "4",
                "height": "3",
                "view_mode": "0",
                "fields": []
            },
            {
                "widgetid": "1",
                "type": "favgraphs",
                "name": "",
                "x": "0",
                "y": "0",
                "width": "4",
                "height": "3",
                "view_mode": "0",
                "fields": []
            }
```

```
                ]
            },
            {
                "dashboard_pageid": "2",
                "name": "",
                "display_period": "0",
                "widgets": []
            },
            {
                "dashboard_pageid": "3",
                "name": "Custom page name",
                "display_period": "60",
                "widgets": []
            }
        ]
    },
    {
        "dashboardid": "2",
        "name": "My dashboard",
        "userid": "1",
        "private": "1",
        "display_period": "60",
        "auto_start": "1",
        "users": [
            {
                "userid": "4",
                "permission": "3"
            }
        ],
        "userGroups": [
            {
                "usrgrpid": "7",
                "permission": "2"
            }
        ],
        "pages": [
            {
                "dashboard_pageid": "4",
                "name": "",
                "display_period": "0",
                "widgets": [
                    {
                        "widgetid": "10",
                        "type": "problems",
                        "name": "",
                        "x": "0",
                        "y": "0",
                        "width": "12",
                        "height": "5",
                        "view_mode": "0",
                        "fields": [
                            {
                                "type": "2",
                                "name": "groupids",
                                "value": "4"
                            }
                        ]
                    }
                ]
            }
        ]
    }
```

```
    ],
    "id": 1
}
```

参阅

- 仪表盘页面
- 仪表盘部件
- 仪表盘部件字段
- 仪表盘用户
- 仪表盘用户组

来源

ui/include/classes/api/services/CDashboard.php 中的 CDashboard::get() 。

任务

此类用于管理任务 (例如检查监控项或不需要重新加载的低级别自动发现规则).

参考对象：

- 任务
- “立即检查” 请求对象
- ‘诊断信息’ 请求对象
- 统计请求对象
- 统计结果对象

可用方法：

- task.create - 创建新的任务
- task.get - 检索任务

> 任务对象

以下对象都是与 `task` 直接相关的 API。

任务对象具有以下属性:

属性	类型	描述
taskid	string	*(只读)* 任务的 ID。
type (必须)	integer	任务的类型. 可能的值： 1 - 诊断信息; 6 - 立即检查。
status	integer	*(只读)* 任务状态。 可能的值： 1 - 新任务； 2 - 进行中的任务； 3 - 已完成的任务； 4 - 过期的任务。
clock	timestamp	*(只读)* 任务创建时间。
ttl	integer	*(只读)* 任务过期时间（秒）。
proxy_hostid	string	被统计诊断信息的代理 ID。 不包含 “立即检查” 任务。

属性	类型	描述
request (必须)	object	根据任务类型的任务请求对象： “立即检查” 任务的对象为详细描述如下； “诊断信息” 任务的对象为详细描述如下。
result	object	*(只读)* 诊断信息任务的结果对象。如果结果还未准备好，可能会包含 NULL。结果对象为详细描述如下。

“立即检查” 请求对象

“立即检查” 任务请求对象具有以下属性。

属性	类型	描述
itemid	string	监控项和低级别自动发现规则的 ID。

“诊断信息” 请求对象

诊断信息任务请求对象有以下这些属性。所有类型属性的统计请求对象详细描述如下。

属性	类型	描述
historycache	object	历史缓存统计请求。在服务器和代理上都可用。
valuecache	object	监控项缓存统计请求。在服务器上可用。
preprocessing	object	预处理管理器统计请求。在服务器和代理上都可用。
alerting	object	告警管理器统计请求。在服务器上可用。
lld	object	LLD 管理器统计请求。在服务器上可用。

统计请求对象

Statistic request 对象用于定义应收集的关于服务器/代理内部进程的信息类型。它具有以下属性。

属性	类型	描述
stats	query	要返回的统计对象属性。每种类型的诊断信息统计的可用字段列表如下详细描述如下。 默认值：`extend` 将返回所有可用的统计字段。
top	object	对象对返回的统计值进行排序和限制。每种类型的诊断信息统计的可用字段列表如下详细描述如下。 例如： { “source.alerts”: 10 }

每种类型的诊断信息请求可用的统计字段列表

可以为每种类型的诊断信息请求属性请求以下统计字段。

诊断类型	可用字段	描述
historycache	items	监控项缓存的数量。
	values	值缓存的数量。
	memory	共享内存统计信息 (空闲空间、使用的块数、空闲块数、空闲块的最大大小)。
	memory.data	历史数据缓存共享内存统计信息。
	memory.index	历史索引缓存共享内存统计信息。
valuecache	items	缓存监控项的数量。
	values	缓存值的数量。
	memory	共享内存统计信息 (空闲空间、使用的块数、空闲块数、空闲块的最大大小)。
	mode	值缓存模式。
preprocessing	values	队列值的数量
	preproc.values	包含预处理步骤的队列值的数量。
alerting	alerts	告警队列的数量。
lld	rules	规则的队列数量。
	values	队列值的数量。

可用于每种诊断信息请求的排序字段列表

以下统计字段可用于排序和限制请求的信息。

诊断类型	可用字段	类型
historycache	values	integer
valuecache	values	integer
	request.values	integer
preprocessing	values	integer
alerting	media.alerts	integer
	source.alerts	integer
lld	values	integer

统计结果对象

在任务对象的 ‘result’ 字段中检索统计结果对象。

属性	类型	描述
status	integer	*(只读)* 任务结果的状态. 可能的值： -1 - 执行任务时发生错误； 0 - 任务结果已创建。
data	string/object	结果根据特定诊断信息任务的统计请求对象。如果在执行任务期间发生错误，则包含错误消息字符串。

task.create

描述

`object task.create(object/array tasks)`

此方法允许创建新任务（例如，在不重新加载配置的情况下收集诊断数据或检查监控项或低级别自动发现规则）。

Note:
此方法只有 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(object/array) 要创建的任务。

该方法接受以下参数。

参数	类型	描述
type (必须)	integer	任务类型. 可能的值： 1 - 诊断信息； 6 - 立即检查.
request (必须)	object	任务请求对象取决于任务类型。任务对象 中描述了请求对象的正确格式。
proxy_hostid	integer	关于哪个诊断信息任务将收集数据的代理。 忽略 “立即检查” 任务。

注意 “立即检查” 任务只能为以下类型的监控项/自动发现规则创建：

- Zabbix agent
- SNMPv1/v2/v3 agent
- Simple check
- Internal check
- External check
- Database monitor
- HTTP agent
- IPMI agent
- SSH agent
- TELNET agent
- Calculated check
- JMX agent

返回值

(object) 返回一个对象，该对象包含 taskids 属性下创建的任务的 ID。为每个监控项和低级别自动发现规则创建一个任务。返回的 ID 的顺序与传递的 itemids 的顺序匹配。

示例

创建一个任务

为两个监控项 check now 创建一个任务。其一是一个监控项，另一个是一个低级别自动发现规则。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "task.create",
    "params": [
        {
            "type": "6",
            "request": {
                "itemid": "10092"
            }
        },
        {
            "type": "6",
            "request": {
                "itemid": "10093"
            }
        }
    ],
    "auth": "700ca65537074ec963db7efabda78259",
```

```
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "taskids": [
            "1",
            "2"
        ]
    },
    "id": 1
}
```

创建一个 diagnostic information 的任务。请求：

```
{
    "jsonrpc": "2.0",
    "method": "task.create",
    "params": [
        {
            "type": "1",
            "request": {
                "alerting": {
                    "stats": [
                        "alerts"
                    ],
                    "top": {
                        "media.alerts": 10
                    }
                },
                "lld": {
                    "stats": "extend",
                    "top": {
                        "values": 5
                    }
                }
            },
            "proxy_hostid": 0
        }
    ],
    "auth": "700ca65537074ec963db7efabda78259",
    "id": 2
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "taskids": [
            "3"
        ]
    },
    "id": 2
}
```

参阅

- 任务
- 'Check now' 请求对象
- 'Diagnostic information' 请求对象
- 统计请求对象

来源

CTask::create() in *ui/include/classes/api/services/CTask.php*.

task.get

描述

`integer/array task.get(object parameters)`

该方法允许根据给定的参数检索任务。该方法仅返回有关 “诊断信息” 任务的详细信息。

> **Note:**
> 此方法只有 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色

参数

`(object)` 定义所需输出的参数。

该方法支持以下参数。

参数	类型	描述
taskids	string/array	仅返回具有给定 ID 的任务。
output	query	这些参数对于所有 `get` 方法都是通用的，在reference commentary中有详细描述。
preservekeys	boolean	

返回值

`(integer/array)` 返回一个对象的数组。

示例

通过 ID 获取 task

检索任务 “1” 的所有数据。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "task.get",
    "params": {
        "output": "extend",
        "taskids": "1"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "taskid": "1",
            "type": "7",
            "status": "3",
            "clock": "1601039076",
            "ttl": "3600",
            "proxy_hostid": null,
            "request": {
                "alerting": {
                    "stats": [
                        "alerts"
                    ],
                    "top": {
```

```
                        "media.alerts": 10
                    }
                },
                "lld": {
                    "stats": "extend",
                    "top": {
                        "values": 5
                    }
                }
            },
            "result": {
                "data": {
                    "alerting": {
                        "alerts": 0,
                        "top": {
                            "media.alerts": []
                        },
                        "time": 0.000663
                    },
                    "lld": {
                        "rules": 0,
                        "values": 0,
                        "top": {
                            "values": []
                        },
                        "time": 0.000442
                    }
                },
                "status": "0"
            }
        }
    ],
    "id": 1
}
```


参阅

- 任务
- 统计结果对象

来源

CTask::get() in *ui/include/classes/api/services/CTask.php*.

值映射

此类用于值映射。

对象引用：

- Value map

可用方法：

- valuemap.create - 创建新的值映射
- valuemap.delete - 删除值映射
- valuemap.get - 获取值映射
- valuemap.update - 更新值映射

> 值映射对象

以下对象都是与 valuemap 直接相关的 API。

值映射

值映射对象具有以下属性。

属性	类型	描述
valuemapid	string	*(只读)* 值映射的 ID。
hostid (必选)	id	值映射主机或模板 ID。
name (必选)	string	值映射的名称。
mappings (必选)	array	当前值映射的值映射。映射对象在下面详细描述。
uuid	string	通用唯一标识符，用于将导入的值映射链接到已存在的映射。仅用于模板上的值映射。如果未给出，则自动生成。

请注意，对于一些方法（更新，删除），必选/可选参数组合是不同的。

值映射关系

值映射关系对象定义了值映射的值映射关系。它具有如下属性。

属性	类型	描述
type	integer	映射匹配类型。等于 0,1,2,3,4 类型的 `value` 字段不能为空，类型 5 的 `value` 字段必须为空。 可用值： 0 - (默认) 精确匹配; 1 - 如果值大于或者等于，映射将会被应用[1]； 2 - 如果值小于或者等于，映射将会被应用[1]； 3 - 如果值在一个范围（包含范围边界），允许定义用逗号符号分隔的多个范围，映射将会被应用[1]； 4 - 如果值和正真表达式匹配，映射将会被应用[2]； 5 - 默认值，如果没有找到其他的匹配，映射将会被应用。
value (必填)	string	原始值。 “默认” 类型的映射不是必填的。
newvalue (必填)	string	原始值的映射值。

[1] 仅支持具有 “无符号数”，“浮点数” 值类型的监控项。
[2] 仅支持具有 “字符” 值类型的监控项。

创建

描述

`object valuemap.create(object/array valuemaps)`

此方法允许创建新的值映射。

> **Note:**
> 此方法只有 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(object/array) 需要创建的值映射。

此方法接受具有standard value map properties的值映射。

返回值

(object) 返回一个包含已创建值映射的 ID 的对象，ID 在 `valuemapids` 属性下。返回的 IDs 的顺序和传递的值映射的顺序相匹配。

示例

创建一个值映射

创建一个包含两个映射的值映射。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "valuemap.create",
    "params": {
        "hostid": "50009",
        "name": "Service state",
        "mappings": [
            {
                "type": "1",
                "value": "1",
                "newvalue": "Up"
            },
            {
                "type": "5",
                "newvalue": "Down"
            }
        ]
    },
    "auth": "57562fd409b3b3b9a4d916d45207bbcb",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "valuemapids": [
            "1"
        ]
    },
    "id": 1
}
```

来源

CValueMap::create() in *ui/include/classes/api/services/CValueMap.php*.

删除

描述

`object valuemap.delete(array valuemapids)`

此方法允许删除值映射。

Note:
此方法只有 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(array) 需要删除的值映射的 ID。

返回值

(object) 返回一个包含被删除的值映射 ID 的对象，ID 在 valuemapids 属性下。

示例

删除多个值映射

删除两个值映射。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "valuemap.delete",
    "params": [
        "1",
        "2"
    ],
    "auth": "57562fd409b3b3b9a4d916d45207bbcb",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "valuemapids": [
            "1",
            "2"
        ]
    },
    "id": 1
}
```

来源

CValueMap::delete() in *ui/include/classes/api/services/CValueMap.php*.

更新

描述

`object valuemap.update(object/array valuemaps)`

此方法允许更新已存在的值映射。

Note:
此方法只有 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(object/array) Value map properties 更新。

每个值映射必须要定义 valuemapid 属性，其他属性都是可选的。只会更新传递的属性，其他的属性保持不变。

返回值

(object) 返回一个对象，包含被更新的值映射的 ID，ID 在 valuemapids 属性下。

示例

更新值映射名称

将值映射名称更新为 “Device status”。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "valuemap.update",
    "params": {
        "valuemapid": "2",
        "name": "Device status"
    },
    "auth": "57562fd409b3b3b9a4d916d45207bbcb",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "valuemapids": [
            "2"
        ]
    },
    "id": 1
}
```

更新一个值映射的映射关系。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "valuemap.update",
    "params": {
        "valuemapid": "2",
        "mappings": [
            {
                "type": "0",
                "value": "0",
                "newvalue": "Online"
            },
            {
                "type": "0",
                "value": "1",
                "newvalue": "Offline"
            }
        ]
    },
    "auth": "57562fd409b3b3b9a4d916d45207bbcb",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "valuemapids": [
            "2"
        ]
    },
    "id": 1
}
```

来源

CValueMap::update() in *ui/include/classes/api/services/CValueMap.php*.

获取

描述

`integer/array valuemap.get(object parameters)`

此方法允许根据给出的参数检索值映射。

> **Note:**
> 此方法对于任何用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(object) 定义期望输出的参数。

此方法支持以下参数。

参数	类型	描述
valuemapids	string/array	仅返回给定 ID 的值映射。
selectMappings	query	返回mappings 属性中当前值映射的值映射关系。 支持 `count`。
sortfield	string/array	按照给定的属性对结果进行排序。 可用值：`valuemapid`，`name`。
countOutput	boolean	这些参数对所有 `get` 方法是公共的，详细描述请参见参考说明 。
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

(integer/array) 返回以下其中一种结果：

- 一个数组对象；
- 如果使用了参数 `countOutput`，则返回检索到的对象的数量。

示例

检索值映射

检索所有配置的值映射。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "valuemap.get",
    "params": {
        "output": "extend"
    },
    "auth": "57562fd409b3b3b9a4d916d45207bbcb",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
```

```
            "valuemapid": "4",
            "name": "APC Battery Replacement Status"
        },
        {
            "valuemapid": "5",
            "name": "APC Battery Status"
        },
        {
            "valuemapid": "7",
            "name": "Dell Open Manage System Status"
        }
    ],
    "id": 1
}
```

根据映射关系，检索一个值映射。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "valuemap.get",
    "params": {
        "output": "extend",
        "selectMappings": "extend",
        "valuemapids": ["4"]
    },
    "auth": "57562fd409b3b3b9a4d916d45207bbcb",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "valuemapid": "4",
            "name": "APC Battery Replacement Status",
            "mappings": [
                {
                    "type": "0",
                    "value": "1",
                    "newvalue": "unknown"
                },
                {
                    "type": "0",
                    "value": "2",
                    "newvalue": "notInstalled"
                },
                {
                    "type": "0",
                    "value": "3",
                    "newvalue": "ok"
                },
                {
                    "type": "0",
                    "value": "4",
                    "newvalue": "failed"
                },
                {
                    "type": "0",
                    "value": "5",
                    "newvalue": "highTemperature"
                },
```

```
                {
                    "type": "0",
                    "value": "6",
                    "newvalue": "replaceImmediately"
                },
                {
                    "type": "0",
                    "value": "7",
                    "newvalue": "lowCapacity"
                }
            ]
        }
    ],
    "id": 1
}
```

来源

CValueMap::get() in *ui/include/classes/api/services/CValueMap.php*.

关联

此类用于关联。

对象引用：

- 关联

可用方法：

- correlation.create - 创建新关联
- correlation.delete - 删除关联
- correlation.get - 获取关联
- correlation.update - 更新关联

> 关联对象

以下对象与关联 API 直接相关。

关联

关联对象具有以下属性。

属性	类型	说明
correlationid	字符	（只读）关联 ID。
name （必需）	字符	关联名称。
description	字符	关联描述。
status	整数	关联是否开启。 可用值： 0 - （默认）启用； 1 - 禁用。

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

关联操作

关联操作对象定义执行关联时将执行的操作。它具有如下属性。

属性	类型	说明
type （必需）	integer	操作类型。 可用值： 0 - 关闭旧事件； 1 - 关闭新事件。

关联过滤

关联过滤对象定义了执行配置的关联操作必须满足的一组条件。它具有如下属性。

属性	类型	说明
evaltype (必需)	integer	过滤条件的评估方法。 可用值： 0 - 与/或； 1 - 与； 2 - 或； 3 - 自定义表达式。
conditions (必需)	array	用来过滤结果的一组过滤条件集。
eval_formula	string	(只读）生成的表达式将用于评估过滤条件。该表达式包含通过“formulaid”引用特定筛选条件的ID。对于具有自定义表达式的筛选，`eval_formula`的值等于`formula`的值。
formula	string	用于具有自定义表达式的过滤评估条件。该表达式必须包含通过“formulaid”引用特定筛选条件的ID。表达式中使用的ID必须与过滤条件中定义的ID完全匹配：没有条件时可以不使用或省略。 自定义表达式过滤必需。

关联过滤条件

关联过滤条件对象定义了在运行关联操作之前必须检查的特定条件。

属性	类型	说明
type (required)	integer	条件类型。 可用值： 0 - 旧事件标签； 1 - 新事件标签； 2 - 新事件主机组； 3 - 事件标签对； 4 - 旧事件标签值； 5 - 新事件标签值。
tag	string	标签（旧或新）。条件类型是：0, 1, 4, 5 时需要。
groupid	string	主机组 ID。条件类型是：2 时需要。
oldtag	string	旧事件标签。条件类型是：3 时需要。
newtag	string	新事件标签。条件类型是：3 时需要。
value	string	事件标签 (旧或新) 值。条件类型是：4, 5 时需要。
formulaid	string	任意唯一 ID，用于引用一个自定义表达式中的条件。只能包含大写字母。当修改过滤条件时，该 ID 必须由用户定义，但以后请求它们时会重新生成。
operator	integer	条件运算符。 条件类型是：2, 4, 5 时需要。

Note:
为了更好地了解如何使用具有各种类型的过滤表达式，请参阅correlation.get 和correlation.create 方法页面上的示例。

以下运算符和值都支持每种条件类型。

条件	条件名称	支持运算符	期望的值
2	主机组	=, <>	主机组 ID。
4	旧事件标签值	=, <>, like, not like	string
5	新事件标签值	=, <>, like, not like	string

创建

描述

`object correlation.create(object/array correlations)`

该方法允许创建新的关联。

Note:
此方法只有 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(object/array) 要创建的关联。

除了标准关联属性以外，此方法还接受如下参数。

参数	类型	说明
operations（必需）	array	创建关联的关联操作。
filter（必需）	object	关联的关联过滤 对象。

返回值

(object) 返回一个对象，该对象包含 correlationids 属性下创建的关联的 ID。返回的 ID 的顺序与所传递的关联的顺序相匹配。

示例

创建一个新的事件标签关联

使用具有一个条件和一个操作的评估方法 AND/OR 创建一个关联。默认情况下，这个关联将被启用。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "correlation.create",
    "params": {
        "name": "new event tag correlation",
        "filter": {
            "evaltype": 0,
            "conditions": [
                {
                    "type": 1,
                    "tag": "ok"
                }
            ]
        },
        "operations": [
            {
                "type": 0
            }
        ]
    },
    "auth": "343baad4f88b4106b9b5961e77437688",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "correlationids": [
            "1"
        ]
    },
    "id": 1
}
```

使用一个自定义表达式过滤

使用自定义过滤条件创建一个关联。公式 ID A 或 B 是任意选择的。条件类型为 “主机组”，操作符为 “<>”。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "correlation.create",
    "params": {
```

```
        "name": "new host group correlation",
        "description": "a custom description",
        "status": 0,
        "filter": {
            "evaltype": 3,
            "formula": "A or B",
            "conditions": [
                {
                    "type": 2,
                    "operator": 1,
                    "formulaid": "A"
                },
                {
                    "type": 2,
                    "operator": 1,
                    "formulaid": "B"
                }
            ]
        },
        "operations": [
            {
                "type": 1
            }
        ]
    },
    "auth": "343baad4f88b4106b9b5961e77437688",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "correlationids": [
            "2"
        ]
    },
    "id": 1
}
```

参见

- 关联过滤
- 关联操作

来源

ui/include/classes/api/services/CCorrelation.php 中的 CCorrelation::create()。

删除

描述

`object correlation.delete(array correlationids)`

此方法允许删除关联。

> **Note:**
> 此方法只有 *Super admin(*超级管理员*)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(array) 要删除的关联 ID。

返回值

(object) 返回一个对象，该对象包含 correlationids 属性下删除的关联 ID。

示例

删除多个关联

删除两个关联。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "correlation.delete",
    "params": [
        "1",
        "2"
    ],
    "auth": "343baad4f88b4106b9b5961e77437688",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "correlaionids": [
            "1",
            "2"
        ]
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CCorrelation.php 中的 CCorrelation::delete()。

更新

描述

`object correlation.update(object/array correlations)`

此方法允许更新已有的关联。

> **Note:**
> 此方法只有 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(object/array) 需要更新的关联属性。

必须为每个关联定义 correlationid 属性，其它的属性都是可选的。只有传递的属性会被更新，其它属性都将保持不变。

除了标准关联属性以外，此方法还接受以下参数。

参数	类型	说明
filter	object	替换当前过滤器的关联过滤 对象。
operations	array	替换现有操作的关联操作。

返回值

(object) 返回一个对象，该对象包含 “correlationids” 属性下更新的关联的 ID。

示例

禁用关联

请求:

```
{
    "jsonrpc": "2.0",
    "method": "correlation.update",
    "params": {
        "correlationid": "1",
        "status": "1"
    },
    "auth": "343baad4f88b4106b9b5961e77437688",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "correlationids": [
            "1"
        ]
    },
    "id": 1
}
```

替换条件，但是评估方法不变

请求:

```
{
    "jsonrpc": "2.0",
    "method": "correlation.update",
    "params": {
        "correlationid": "1",
        "filter": {
            "conditions": [
                {
                    "type": 3,
                    "oldtag": "error",
                    "newtag": "ok"
                }
            ]
        }
    },
    "auth": "343baad4f88b4106b9b5961e77437688",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "correlationids": [
            "1"
        ]
    },
    "id": 1
}
```

参见

- 关联过滤
- 关联操作

来源

ui/include/classes/api/services/CCorrelation.php 中的 CCorrelation::update()。

获取

描述

`integer/array correlation.get(object parameters)`

此方法允许根据给定的参数检索关联。

> **Note:**
> 此方法对于任何用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

Parameters

(object) 定义需要输出的参数。

此方法支持以下参数。

参数	类型	说明
correlationids	string/array	仅返回给定 ID 的关联。
selectFilter	query	返回带有关联条件的过滤属性。
selectOperations	query	返回带有关联操作的操作 属性。
sortfield	string/array	按照给定属性对结果进行排序。 可用值：`correlationid`，`name` 和 `status`。
countOutput	boolean	这些参数对于所有 get 方法都是通用的，详情请参考参考说明。
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

(integer/array) 返回其中之一：

- 一个对象数组；
- 如果使用了 countOutput 参数，将返回获取对象的数量。

示例

检索关联

获取所有配置的关联以及关联条件和操作。该过滤器使用 “and/or” 评估类型，因此 `formula` 属性为空，并自动生成 `eval_formula`。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "correlation.get",
    "params": {
```

```
        "output": "extend",
        "selectOperations": "extend",
        "selectFilter": "extend"
    },
    "auth": "343baad4f88b4106b9b5961e77437688",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "correlationid": "1",
            "name": "Correlation 1",
            "description": "",
            "status": "0",
            "filter": {
                "evaltype": "0",
                "formula": "",
                "conditions": [
                    {
                        "type": "3",
                        "oldtag": "error",
                        "newtag": "ok",
                        "formulaid": "A"
                    }
                ],
                "eval_formula": "A"
            },
            "operations": [
                {
                    "type": "0"
                }
            ]
        }
    ],
    "id": 1
}
```

参见

- 关联过滤
- 关联操作

来源

ui/include/classes/api/services/CCorrelation.php 中的 CCorrelation::get()。

动作

这个类用于操作动作。

对象引用：

- 动作
- 动作条件
- 动作操作

可用方法：

- action.create - 创建新的动作
- action.delete - 删除动作
- action.get - 获取动作

- action.update - 更新动作

> 动作对象

以下对象与动作 API 直接相关。

动作

动作对象具有以下属性。

属性	类型	描述
actionid	字符	(只读）动作 ID。
esc_period (必需)	字符	默认操作步骤持续时间。必须至少为 60 秒。接受秒、带后缀的时间单位和用户宏。 请注意，仅对触发器、内部和服务动作支持升级，问题恢复和更新操作不支持升级。
eventsource (必需)	整数	(常量）动作将处理的事件的类型。 参见event "source" 属性 以获取支持的事件类型列表。
name (必需)	字符	动作名称。
status	整数	动作是启用还是禁用。 可用值： 0 - （默认）启用； 1 - 禁用。
pause_suppressed	整数	是否在维护期间暂停升级。 可用值： 0 - 不要暂停升级； 1 - （默认）暂停升级。 请注意，此参数仅对触发器动作有效。
notify_if_canceled	整数	取消升级时是否通知。 可用值： 0 - 取消升级时不通知； 1 - （默认）取消升级时通知。 请注意，此参数仅对触发器动作有效。

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

动作操作

动作操作对象定义执行动作时执行的操作。它具有以下属性。

属性	类型	描述
operationid	string	(只读）动作操作的ID。
operationtype(必需)	integer	操作类型。 可用值： 0 - 发送信息； 1 - 全局脚本； 2 - 添加主机； 3 - 删除主机； 4 -添加到主机组； 5 - 从主机组删除； 6 - 链接到模板； 7 - 取消与模板的关联； 8 - 启用主机； 9 - 禁用主机； 10 -设置主机资产模式。 请注意，触发器和服务动作只支持类型'0' 和'1' ，内部动作只支持类型'0'。发现和自动注册操作支持所有类型。
actionid	string	(只读）动作操作所属的动作 ID。
esc_period	string	以秒为单位的升级步骤的持续时间。必须大于 60 秒。接受秒、带后缀的时间单位和用户宏。如果设置为 0 或 0s，则将使用默认的动作升级周期。 默认：0s。 请注意，仅触发器、内部和服务操作支持升级，问题恢复和更新操作不支持升级。
esc_step_from	integer	开始升级的 Step 。 默认：1。 请注意，仅触发器、内部和服务操作支持升级，问题恢复和更新操作不支持升级。

属性	类型	描述
esc_step_to	integer	结束升级的 Step。 默认：1。 请注意，仅触发器、内部和服务操作支持升级，问题恢复和更新操作不支持升级。
evaltype	integer	操作条件计算方法。 可用值： 0 - （默认） AND / OR； 1 - AND； 2 - OR。
opcommand	object	对象包含操作运行的全局脚本上的数据。 每个对象具有以下属性：`scriptid` - （字符串）脚本的 ID。 被全局脚本操作所需要。
opcommand_grp	array	要在其上运行全局脚本的主机组。 每个对象具有以下属性：`opcommand_grpid` - （字符串，只读）对象 ID；`operationid` - （字符串，只读）操作 ID；`groupid` - （字符串）主机组 ID。 在 `opcommand_hst` 未被设置的情况下，被全局脚本操作所需要。

属性	类型	描述
opcommand_hst	array	要在其上运行全局脚本的主机。 每个对象具有以下属性： `opcommand_hstid` - （字符串，只读）对象 ID； `operationid` - （字符串，只读）操作 ID； `hostid` - （字符串）主机 ID；如果设置为 0 命令将运行在当前主机。 在 `opcommand_grp` 未被设置的情况下，被全局脚本操作所需要。
opconditions	array	用于触发动作的操作条件。 动作操作条件如下所述。
opgroup	array	用于添加主机的主机组。 每个对象具有以下属性： `operationid` - （字符串，只读）操作 ID ； `groupid` - （字符串）主机组 ID 。 被” 添加到主机组” 和” 从主机组删除” 操作所需要。
opmessage	object	包含有关操作发送的消息的数据的对象。 动作操作消息对象如下所述。 被消息操作所需要。

属性	类型	描述
opmessage_grp	array	要向其发送消息的用户组。 每个对象具有以下属性： `operationid` - (字符串，只读）操作 ID； `usrgrpid` - （字符串）用户组 ID。 在 `opmessage_usr` 未被设置的情况下，被全局脚本操作所需要。
opmessage_usr	array	要向其发送消息的用户。 每个对象具有以下属性： `operationid` - (字符串）操作 ID； `userid` - （字符串）用户 ID。 在 `opmessage_grp` 未被设置的情况下，被全局脚本操作所需要。
optemplate	array	要将主机链接到的模板。 每个对象具有以下属性： `operationid` - (字符串，只读）操作 ID； `templateid` - (字符串）模板 ID 。 被”链接到模板”和”取消与模板的关联”操作所需要。
opinventory	object	要将主机设置的资产模式。 对象具有以下属性： `operationid` - (字符串，只读）操作 ID； `inventory_mode` - （字符串）资产模式。 被”设置主机资产模式”操作所需要。

动作操作消息

操作消息对象包含有关操作将发送的消息的数据。

属性	类型	描述
default_msg	integer	是否使用默认 action message 文本和主题。 可用值： 0 - 使用动作中的数据； 1 - （默认）使用媒介类型中的数据。
mediatypeid	string	将用于发送消息的媒介类型 ID。
message	string	操作消息文本。
subject	string	操作消息主题。

动作操作条件

动作操作条件对象定义执行当前操作时必须满足的条件。它具有以下属性。

属性	类型	描述
opconditionid	string	（只读）动作操作条件 ID。
conditiontype （必需）	integer	条件类型。 可用值： 14 - 事件确认。
value （必需）	string	与之比较的值。
operationid	string	（只读）操作 ID。
operator	integer	条件运算符。 可用值： 0 - （默认） =。

每个操作条件类型都支持以下运算符和值。

条件	条件名	支持的运算	期望值
14	事件确认	=	事件是否被确认。 可用值： 0 - 未确认； 1 - 已确认。

动作恢复操作

动作恢复操作对象定义了当问题解决时将执行的操作。恢复操作对于触发器、内部和服务动作是可能的。它具有以下属性。

属性	类型	描述
operationid	string	*(只读)* 动作操作的 ID。
operationtype (必需)	integer	操作类型。 触发器和服务动作的可能值： 0 - 发送消息； 1 - 全局脚本； 11 - 通知所有相关方。 内部动作的可能值： 0 - 发送消息； 11 - 通知所有相关方。

属性	类型	描述
actionid	string	*(只读)* 恢复操作所属动作的 ID。
opcommand	object	包含由操作执行的全局动作类型脚本数据的对象。 每个对象具有以下属性：`scriptid` - *(字符串)* 动作类型脚本的 ID。
opcommand_grp	array	全局脚本操作所需的。 在主机组上运行全局脚本。 每个对象具有以下属性： `opcommand_grpid` - *(字符串, 只读)* 对象的 ID； `operationid` - *(字符串, 只读)* 操作的 ID； `groupid` - *(字符串)* 主机组的 ID。
opcommand_hst	array	如果未设置 `opcommand_hst`，则全局脚本操作所需的。 在主机上运行全局脚本。 每个对象具有以下属性： `opcommand_hstid` - *(字符串, 只读)* 对象的 ID； `operationid` - *(字符串, 只读)* 操作的 ID； `hostid` - *(字符串)* 主机的 ID；如果设置为 0，则命令将在当前主机上运行。
opmessage	object	如果未设置 `opcommand_grp`，则全局脚本操作所需的。 包含由恢复操作发送的消息数据的对象。 操作消息对象在上面详细描述。
opmessage_grp	array	发送消息操作所需的。 发送消息的用户组。 每个对象具有以下属性： `operationid` - *(字符串, 只读)* 操作的 ID； `usrgrpid` - *(字符串)* 用户组的 ID。
opmessage_usr	array	如果未设置 `opmessage_usr`，则发送消息操作所需的。 发送消息的用户。 每个对象具有以下属性： `operationid` - *(字符串, 只读)* 操作的 ID； `userid` - *(字符串)* 用户的 ID。 如果未设置 `opmessage_grp`，则发送消息操作所需的。

动作更新操作

动作更新操作对象定义在问题更新（注释、确认、严重性更改或手动关闭）时将执行的操作。更新操作可用于触发器和服务动作。它具有以下属性。

属性	类型	描述
operationid	string	(只读）动作操作 ID。
operationtype (必需)	integer	操作类型。 在触发器和服务动作的情况下可能的值： 0 - 发送消息； 1 - 全局脚本； 12 - 通知所有相关人员。

属性	类型	描述
opcommand	object	对象包含运行的全局操作类型脚本操作的数据。 每个对象都有以下属性：`scriptid` - （字符串）动作类型脚本 ID。
opcommand_grp	array	被全局脚本操作所需要。 要在其上运行全局脚本的主机组。 每个对象都具有以下属性： `groupid` - （字符串）主机组 ID
opcommand_hst	array	在 `opcommand_hst` 未被设置的情况下，被全局脚本操作所需要。 要在其上运行全局脚本的主机。 每个对象都具有以下属性： `hostid` - （字符串）主机 ID；如果设置为 0 命令将运行在当前主机。
opmessage	object	在 `opcommand_grp` 未被设置的情况下，被全局脚本操作所需要。 对象包含有关更新操作发送的消息的数据。 操作消息对象如上所述。
opmessage_grp	array	要向其发送消息的用户组。 每个对象都具有以下属性： `usrgrpid` - （字符串）用户组 ID。
		在 `opmessage_usr` 未被设置的情况下，仅被 发送消息操作所需要。 被 发送更新消息操作所忽略。

属性	类型	描述
opmessage_usr	array	要向其发送消息的用户。 每个对象都具有以下属性： `userid` - （字符串）用户 ID。 在 `opmessage_grp` 未被设置的情况下，仅被 发送消息 操作所需要。 被 发送更新消息操作所忽略。

动作过滤

动作过滤对象定义执行配置的动作操作时必须满足的一组条件。它具有以下属性。

属性	类型	描述
conditions (required)	array	用于筛选结果的过滤条件集。
evaltype (required)	integer	过滤条件计算方法。 可用值： 0 - and/or； 1 - and； 2 - or； 3 - 自定义表达式。
eval_formula	string	（只读）生成的表达式将被用于计算过滤条件。表达式包含了通过其 ID 引用特定筛选条件的 `formulaid`。`eval_formula` 的值等于具有自定义表达式的筛选器的 `formula` 的值。
formula	string	用于使用自定义表达式计算过滤条件的用户定义表达式。表达式必须包含通过其 `formulaid` 引用特定过滤条件的 ID。表达式中使用的 ID 必须与过滤条件中定义的 ID 完全匹配：任何条件都不能保持未使用或省略状态。 被自定义表达式过滤需要。

动作过滤条件

动作过滤条件对象定义了在执行动作操作之前必须检查的特定条件。

属性	类型	描述
conditionid	string	*(只读)* 动作条件的 ID。
conditiontype (必需)	integer	条件类型。 触发器动作的可能值： 0 - 主机组； 1 - 主机； 2 - 触发器； 3 - 事件名称； 4 - 触发器严重性； 6 - 时间段； 13 - 主机模板； 16 - 问题被抑制； 25 - 事件标签； 26 - 事件标签值。 发现动作的可能值： 7 - 主机 IP； 8 - 发现的服务类型； 9 - 发现的服务端口； 10 - 发现状态； 11 - 正常或异常持续时间； 12 - 接收到的值； 18 - 发现规则； 19 - 发现检查； 20 - proxy； 21 - 发现对象。 自动注册动作的可能值： 20 - proxy； 22 - 主机名； 24 - 主机元数据。 内部动作的可能值： 0 - 主机组； 1 - 主机； 13 - 主机模板； 23 - 事件类型； 25 - 事件标签； 26 - 事件标签值。 服务动作的可能值： 25 - 事件标签； 26 - 事件标签值； 27 - 服务； 28 - 服务名称。
value (必需)	string	比较值。
value2	string	比较的次要值。当条件类型是 *26* 时，对于触发器、内部和服务动作是必需的。
actionid	string	*(只读)* 条件所属动作的 ID。
formulaid	string	用于从自定义表达式引用条件的任意唯一 ID。只能包含大写字母。当修改过滤条件时，必须由用户定义此 ID，但在之后请求时将重新生成。

属性	类型	描述
operator	integer	条件运算符。 可能的值： 0 - (默认) 等于； 1 - 不等于； 2 - 包含； 3 - 不包含； 4 - 在内； 5 - 大于或等于； 6 - 小于或等于； 7 - 不在内； 8 - 匹配； 9 - 不匹配； 10 - 是； 11 - 否。

Note:
要更好地理解如何使用各种类型的表情与过滤器，请参阅action.get和action.create方法页面上的示例。

每种条件类型支持以下运算符和值。

条件	条件名称	支持的运算符	期望的值
0	主机组	等于， 不等于	主机组 ID。
1	主机	等于， 不等于	主机 ID。
2	触发器	等于， 不等于	触发器 ID。
3	事件名称	包含， 不包含	事件名称。
4	触发器严重性	等于， 不等于， 大于或等于， 小于或等于	触发器严重性。参考触发器"severity" 属性以获取支持的触发器严重性的列表。
5	触发器值	等于	触发器值。参考触发器"value" 属性以获取支持的触发器值的列表。
6	时间段	在内， 不在内	事件触发时的时间作为时间段。
7	主机 IP	等于， 不等于	用于检查的一个或多个 IP 范围，用逗号分隔。有关支持的 IP 范围格式的更多信息，请参阅网络发现配置部分。
8	发现的服务类型	等于， 不等于	发现的服务类型。服务类型与用于检测服务的发现检查类型匹配。参考发现检查"type" 属性以获取支持的类型的列表。
9	发现的服务端口	等于， 不等于	一个或多个端口范围，用逗号分隔。
10	发现状态	等于	发现对象的状态。 可能的值： 0 - 主机或服务正常； 1 - 主机或服务异常； 2 - 主机或服务已发现； 3 - 主机或服务丢失。
11	正常或异常持续时间	大于或等于， 小于或等于	表示发现对象当前状态持续时间的秒数。
12	接收值	等于， 不等于， 大于或等于， 小于或等于， 包含， 不包含	执行 Zabbix agent、SNMPv1、SNMPv2 或 SNMPv3 发现检查时返回的值。

条件	条件名称	支持的运算符	期望的值
13	主机模板	等于， 不等于	链接的模板 ID。
16	问题被抑制	是， 否	不需要值：使用” 是” 运算符表示问题必须被抑制，” 否” - 未被抑制。
18	发现规则	等于， 不等于	发现规则的 ID。
19	发现检查	等于， 不等于	发现检查的 ID。
20	Proxy	等于， 不等于	proxy 的 ID。
21	发现对象	等于	触发发现事件的对象类型。 可能的值： 1 - 发现的主机； 2 - 发现的服务。
22	主机名	包含， 不包含， 匹配， 不匹配	主机名。 在使用自动注册条件的 匹配和 不匹配运算符时支持正则表达式。
23	事件类型	等于	特定的内部事件。 可能的值： 0 - 监控项处于” 不支持” 状态； 1 - 监控项处于” 正常” 状态； 2 - LLD 规则处于” 不支持” 状态； 3 - LLD 规则处于” 正常” 状态； 4 - 触发器处于” 未知” 状态； 5 - 触发器处于” 正常” 状态。
24	主机元数据	包含， 不包含， 匹配， 不匹配	自动注册主机的元数据。 在使用 匹配和 不匹配运算符时支持正则表达式。
25	标签	等于， 不等于， 包含， 不包含	事件标签。
26	标签值	等于， 不等于， 包含， 不包含	事件标签值。
27	服务	等于， 不等于	服务 ID。
28	服务名称	等于， 不等于	服务名称。

创建

描述

`object action.create(object/array actions)`

此方法用于创建新动作。

> **Note:**
> 此方法只有 *Admin(*管理员*)* 和 *Super admin(*超级管理员*)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

`(object/array)` 要创建的动作。

此外，还有标准动作参数，方法接受如下参数。

参数	类型	描述
filter	object	动作的动作过滤 对象。
operations	array	为这个动作创建的动作操作 。
recovery_operations	array	为这个动作创建的动作恢复操作。
update_operations	array	为这个动作创建的动作更新操作。

返回值

（object）返回一个对象，其中 actionids 属性下包含已创建动作的 ID。返回的 ID 的顺序与传递的动作顺序相匹配。

示例

创建触发器动作

创建一个触发器动作，当主机”10084” 的触发器（其名称中包含”memory” 一词）变为问题状态时将开始执行。该动作将配置有 4 个操作。

第一个立即执行的操作将通过媒体类型”1” 向用户组”7” 中的所有用户发送消息。如果事件在 30 分钟内未解决，第二个操作将在组”2” 中的所有主机上运行脚本 ”5”（作用域为” 动作操作” 的脚本）。如果事件得到解决，恢复操作将通知所有收到有关问题消息的用户。如果事件被更新，确认/更新操作将使用自定义主题和消息通知（所有收到有关问题消息的用户）。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "action.create",
    "params": {
        "name": "Trigger action",
        "eventsource": 0,
        "esc_period": "30m",
        "filter": {
            "evaltype": 0,
            "conditions": [
                {
                    "conditiontype": 1,
                    "operator": 0,
                    "value": "10084"
                },
                {
                    "conditiontype": 3,
                    "operator": 2,
                    "value": "memory"
                }
            ]
        },
        "operations": [
            {
                "operationtype": 0,
                "esc_step_from": 1,
                "esc_step_to": 1,
                "opmessage_grp": [
                    {
                        "usrgrpid": "7"
                    }
                ],
                "opmessage": {
                    "default_msg": 1,
                    "mediatypeid": "1"
                }
            },
            {
                "operationtype": 1,
                "esc_step_from": 2,
                "esc_step_to": 2,
                "opconditions": [
                    {
```

```
                        "conditiontype": 14,
                        "operator": 0,
                        "value": "0"
                    }
                ],
                "opcommand_grp": [
                    {
                        "groupid": "2"
                    }
                ],
                "opcommand": {
                    "scriptid": "5"
                }
            }
        ],
        "recovery_operations": [
            {
                "operationtype": "11",
                "opmessage": {
                    "default_msg": 1
                }
            }
        ],
        "update_operations": [
            {
                "operationtype": "12",
                "opmessage": {
                    "default_msg": 0,
                    "message": "Custom update operation message body",
                    "subject": "Custom update operation message subject"
                }
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "actionids": [
            "17"
        ]
    },
    "id": 1
}
```

创建发现动作

创建一个发现动作，该动作将把发现的主机链接到模板”10001”。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "action.create",
    "params": {
        "name": "Discovery action",
        "eventsource": 1,
        "filter": {
            "evaltype": 0,
            "conditions": [
```

```
                {
                    "conditiontype": 21,
                    "operator": 0,
                    "value": "1"
                },
                {
                    "conditiontype": 10,
                    "operator": 0,
                    "value": "2"
                }
            ]
        },
        "operations": [
            {
                "operationtype": 6,
                "optemplate": [
                    {
                        "templateid": "10001"
                    }
                ]
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "actionids": [
            "18"
        ]
    },
    "id": 1
}
```

使用自定义表达式过滤器

创建一个触发器动作，该动作使用自定义表达式 - ”A and (B or C)” - 来评估动作条件。一旦主机”10084” 或主机”10106” 上的触发器达到或超过”Warning” 严重性级别并变为问题状态，该动作将通过媒体类型”1” 向用户组”7” 中的所有用户发送消息。公式 ID ”A”、”B” 和”C” 是任意选择的。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "action.create",
    "params": {
        "name": "Trigger action",
        "eventsource": 0,
        "esc_period": "15m",
        "filter": {
            "evaltype": 3,
            "formula": "A and (B or C)",
            "conditions": [
                {
                    "conditiontype": 4,
                    "operator": 5,
                    "value": "2",
                    "formulaid": "A"
                },
                {
```

```
                    "conditiontype": 1,
                    "operator": 0,
                    "value": "10084",
                    "formulaid": "B"
                },
                {
                    "conditiontype": 1,
                    "operator": 0,
                    "value": "10106",
                    "formulaid": "C"
                }
            ]
        },
        "operations": [
            {
                "operationtype": 0,
                "esc_step_from": 1,
                "esc_step_to": 1,
                "opmessage_grp": [
                    {
                        "usrgrpid": "7"
                    }
                ],
                "opmessage": {
                    "default_msg": 1,
                    "mediatypeid": "1"
                }
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "actionids": [
            "18"
        ]
    },
    "id": 1
}
```

创建代理自动注册规则

创建一个自动注册动作，当主机名包含”SRV” 或元数据包含”AlmaLinux” 时，将主机添加到主机组”2”。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "action.create",
    "params": {
        "name": "Register Linux servers",
        "eventsource": "2",
        "filter": {
            "evaltype": "2",
            "conditions": [
                {
                    "conditiontype": "22",
                    "operator": "2",
                    "value": "SRV"
```

```
                },
                {
                    "conditiontype": "24",
                    "operator": "2",
                    "value": "AlmaLinux"
                }
            ]
        },
        "operations": [
            {
                "operationtype": "4",
                "opgroup": [
                    {
                        "groupid": "2"
                    }
                ]
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "actionids": [
            19
        ]
    },
    "id": 1
}
```

另请参阅

- 动作过滤
- 动作操作
- 脚本

来源

ui/include/classes/api/services/CAction.php 中的 CAction::create() 。

删除

描述

`object action.delete(array actionIds)`

此方法用于删除动作。

> **Note:**
> 此方法只有 *Admin(管理员)* 和 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(array) 要删除动作的 ID。

返回值

(object) 返回一个对象，该对象在 actionids 属性下包含要删除动作的 ID。

示例

删除多个动作

删除两个动作。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "action.delete",
    "params": [
        "17",
        "18"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "actionids": [
            "17",
            "18"
        ]
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CAction.php 中的 CAction::delete()。

更新

描述

`object action.update(object/array actions)`

此方法允许更新现有的动作。

Note:
此方法只有 *Admin(管理员)* 和 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(object/array) 要更新的动作属性。

必须为每个动作定义 actionid 属性，所有其他属性都是可选的。只会更新传递的属性，所有其他属性将保持不变。

此外，还有标准动作参数，方法接受如下参数。

参数	类型	描述
filter	object	动作过滤 对象以替换当前过滤。
operations	array	动作操作 以替换已经存在的操作。
recovery_operations	array	动作恢复操作 以替换已经存在的恢复操作。
update_operations	array	动作更新操作 以替换已经存在的更新操作。

返回值

(object) 返回一个对象，该对象在 actionids 属性下包含更新的动作的 ID。

示例

禁用动作

禁用一个动作，即，将其状态设置为 “1”。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "action.update",
    "params": {
        "actionid": "2",
        "status": "1"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "actionids": [
            "2"
        ]
    },
    "id": 1
}
```

参见

- 动作过滤
- 动作操作

来源

ui/include/classes/api/services/CAction.php 中的 CAction::update()。

获取

描述

`integer/array action.get(object parameters)`

该方法允许根据给定的参数检索动作。

Note:
此方法对于任何用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

`(object)` 定义期望输出的参数。

该方法支持以下参数。

参数	类型	描述
actionids	string/array	只返回具有给定 ID 的动作。
groupids	string/array	只返回在动作条件中使用给定主机组的动作。
hostids	string/array	只返回在动作条件中使用给定主机的动作。
triggerids	string/array	只返回在动作条件中使用给定触发器的动作。
mediatypeids	string/array	只返回使用给定媒体类型发送消息的动作。
usrgrpids	string/array	只返回配置为向给定用户组发送消息的动作。
userids	string/array	只返回配置为向给定用户发送消息的动作。
scriptids	string/array	只返回配置为运行给定脚本的动作。
selectFilter	query	返回一个带有动作条件过滤器的filter 属性。
selectOperations	query	返回一个带有动作操作的operations 属性。
selectRecoveryOperations	query	返回一个带有动作恢复操作的recovery_operations 属性。
selectUpdateOperations	query	返回一个带有动作更新操作的update_operations 属性。
sortfield	string/array	根据给定属性对结果进行排序。 可能的值有：`actionid`, `name` 和 `status`。
countOutput	boolean	这些参数对于所有的 `get` 方法都是通用的，在参考评注 中有描述。

参数	类型	描述
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

(integer/array) 返回以下任一选项：

- 对象数组；
- 如果使用了 countOutput 参数，则检索对象的计数。

示例

检索触发器动作

检索所有已配置的触发器动作以及它们的动作条件和操作。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "action.get",
    "params": {
        "output": "extend",
        "selectOperations": "extend",
        "selectRecoveryOperations": "extend",
        "selectUpdateOperations": "extend",
        "selectFilter": "extend",
        "filter": {
            "eventsource": 0
        }
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "actionid": "3",
            "name": "Report problems to Zabbix administrators",
            "eventsource": "0",
            "status": "1",
            "esc_period": "1h",
            "pause_suppressed": "1",
            "filter": {
                "evaltype": "0",
                "formula": "",
                "conditions": [],
                "eval_formula": ""
            },
            "operations": [
                {
                    "operationid": "3",
                    "actionid": "3",
```

```
                "operationtype": "0",
                "esc_period": "0",
                "esc_step_from": "1",
                "esc_step_to": "1",
                "evaltype": "0",
                "opconditions": [],
                "opmessage": [
                    {
                        "default_msg": "1",
                        "subject": "",
                        "message": "",
                        "mediatypeid" => "0"
                    }
                ],
                "opmessage_grp": [
                    {
                        "usrgrpid": "7"
                    }
                ]
            }
        ],
        "recovery_operations": [
            {
                "operationid": "7",
                "actionid": "3",
                "operationtype": "11",
                "evaltype": "0",
                "opconditions": [],
                "opmessage": {
                    "default_msg": "0",
                    "subject": "{TRIGGER.STATUS}: {TRIGGER.NAME}",
                    "message": "Trigger: {TRIGGER.NAME}\r\nTrigger status: {TRIGGER.STATUS}\r\nTrigger
                    "mediatypeid": "0"
                }
            }
        ],
        "update_operations": [
            {
                "operationid": "31",
                "operationtype": "12",
                "evaltype": "0",
                "opmessage": {
                    "default_msg": "1",
                    "subject": "",
                    "message": "",
                    "mediatypeid": "0"
                }
            },
            {
                "operationid": "32",
                "operationtype": "0",
                "evaltype": "0",
                "opmessage": {
                    "default_msg": "0",
                    "subject": "Updated: {TRIGGER.NAME}",
                    "message": "{USER.FULLNAME} updated problem at {EVENT.UPDATE.DATE} {EVENT.UPDATE.T
                    "mediatypeid": "1"
                },
                "opmessage_grp": [
                    {
                        "usrgrpid": "7"
                    }
```

```
                    ],
                    "opmessage_usr": []
                },
                {
                    "operationid": "33",
                    "operationtype": "1",
                    "evaltype": "0",
                    "opcommand": {
                        "scriptid": "3"
                    },
                    "opcommand_hst": [
                        {
                            "hostid": "10084"
                        }
                    ],
                    "opcommand_grp": []
                }
            ]
        }
    ],
    "id": 1
}
```

检索发现动作

检索所有已配置的发现动作以及它们的动作条件和操作。过滤器使用"and" 计算类型，所以 `formula` 属性为空且 `eval_formula` 将自动生成。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "action.get",
    "params": {
        "output": "extend",
        "selectOperations": "extend"
        "selectFilter": "extend",
        "filter": {
            "eventsource": 1
        }
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "actionid": "2",
            "name": "Auto discovery. Linux servers.",
            "eventsource": "1",
            "status": "1",
            "esc_period": "0s",
            "pause_suppressed": "1",
            "filter": {
                "evaltype": "0",
                "formula": "",
                "conditions": [
                    {
                        "conditiontype": "10",
                        "operator": "0",
                        "value": "0",
```

```
                        "value2": "",
                        "formulaid": "B"
                    },
                    {
                        "conditiontype": "8",
                        "operator": "0",
                        "value": "9",
                        "value2": "",
                        "formulaid": "C"
                    },
                    {
                        "conditiontype": "12",
                        "operator": "2",
                        "value": "Linux",
                        "value2": "",
                        "formulaid": "A"
                    }
                ],
                "eval_formula": "A and B and C"
            },
            "operations": [
                {
                    "operationid": "1",
                    "actionid": "2",
                    "operationtype": "6",
                    "esc_period": "0s",
                    "esc_step_from": "1",
                    "esc_step_to": "1",
                    "evaltype": "0",
                    "opconditions": [],
                    "optemplate": [
                        {
                            "templateid": "10001"
                        }
                    ]
                },
                {
                    "operationid": "2",
                    "actionid": "2",
                    "operationtype": "4",
                    "esc_period": "0s",
                    "esc_step_from": "1",
                    "esc_step_to": "1",
                    "evaltype": "0",
                    "opconditions": [],
                    "opgroup": [
                        {
                            "groupid": "2"
                        }
                    ]
                }
            ]
        }
    ],
    "id": 1
}
```


参见

- 动作过滤
- 动作操作

来源

ui/include/classes/api/services/CAction.php 中的 CAction::get()。

历史数据

该类旨在处理历史数据。

对象引用:

- History

可用的方法:

- history.get 检索历史数据

> 历史数据对象

下列对象与 `history` API 直接相关。

Note:
历史数据对象会因为监控项的数据类型而有所不同. 它们都是 Zabbix Server 创建的，不能通过 API 进行修改。

浮点型历史数据

浮点型历史数据对象具有以下属性。

属性	类型	描述
clock	timestamp	收到该数值的时间。
itemid	string	相关监控项的 ID。
ns	integer	收到数值时的纳秒数。
value	float	收到的数值。

整数型历史数据

整数型历史数据对象具有以下属性。

属性	类型	描述
clock	timestamp	收到数值的时间。
itemid	string	相关监控项的 ID。
ns	integer	收到数值时的纳秒数。
value	integer	收到的数值。

字符串型历史数据

字符串型历史数据对象具有以下属性。

属性	类型	描述
clock	timestamp	收到值的时间。
itemid	string	相关监控项的 ID。
ns	integer	收到值时的纳秒数。
value	string	收到的值。

文本型历史数据

文本型历史数据对象具有以下属性。

属性	类型	描述
id	string	历史条目的 ID。
clock	timestamp	收到值的时间。

属性	类型	描述
itemid	string	相关监控项的 ID。
ns	integer	收到值时的纳秒数。
value	text	收到的值。

日志型历史数据

日志型历史数据对象具有以下属性。

属性	类型	描述
id	string	历史条目的 ID。
clock	timestamp	收到值的时间。
itemid	string	相关监控项的 ID。
logeventid	integer	Windows 事件日志条目的 ID。
ns	integer	收到值时的纳秒数。
severity	integer	Windows 事件日志条目的级别。
source	string	Windows 事件日志条目的来源。
timestamp	timestamp	Windows 事件日志条目的时间。
value	text	收到的值。

删除历史数据

描述

`object history.clear(array itemids)`

这个方法用于删除监控项的历史数据。

> **Note:**
> 这个方法只适用于 *Admin* 和 *Super admin* 用户类型。调用该方法的权限可以在用户角色设置中被撤销。前往用户角色以了解更多信息。

参数

(array) 需要删除历史数据的监控项 ID。

返回值

(object) 返回一个包含 itemids 属性（其中包含成功删除历史数据的监控项 ID）的对象。

示例

删除历史数据

请求:

```
{
    "jsonrpc": "2.0",
    "method": "history.clear",
    "params": [
        "10325",
        "13205"
    ],
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "10325",
            "13205"
```

```
        ]
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CHistory.php 的 CHistory::clear()。

查询历史数据

描述

`integer/array history.get(object parameters)`

该方法允许根据给定参数检索历史数据。

Attention:
如果这些数据尚未被管家删除，此方法可能会返回已删除实体的历史数据。

Note:
此方法对任何类型的用户都可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请参阅用户角色。

参数

`(object)` 参数定义所期望的输出。

该方法支持以下参数。

参数	类型	描述
history	integer	要返回的历史对象的类型。 可能的取值： 0 - 浮点数； 1 - 字符； 2 - 日志； 3 - 无符号数； 4 - 文本。 默认：3.
hostids	string/array	只返回给定主机的历史。
itemids	string/array	只返回给定监控项的历史。
time_from	timestamp	只返回在给定时间或在给定时间之后收到的值。
time_till	timestamp	只返回在给定时间之前或在给定时间收到的值。
sortfield	string/array	按给定的属性对结果进行排序。 可能的取值：`itemid` 和 `clock`.
countOutput	boolean	这些参数是所有 `get` 方法的共同参数，在参考注释页面中有详细描述。
editable	boolean	
excludeSearch	boolean	

参数	类型	描述
filter	object	
limit	integer	
output	query	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

返回（integer/array）其中之一：

- 一个对象的数组；
- 如果使用了 countOutput 参数，则为检索到的对象的数量。

示例

检索监控项的历史数据

返回从一个 numeric(float) 监控项收到的 10 个最新值。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "history.get",
    "params": {
        "output": "extend",
        "history": 0,
        "itemids": "23296",
        "sortfield": "clock",
        "sortorder": "DESC",
        "limit": 10
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "itemid": "23296",
            "clock": "1351090996",
            "value": "0.085",
            "ns": "563157632"
        },
        {
            "itemid": "23296",
            "clock": "1351090936",
            "value": "0.16",
            "ns": "549216402"
        },
        {
            "itemid": "23296",
            "clock": "1351090876",
            "value": "0.18",
            "ns": "537418114"
        },
        {
            "itemid": "23296",
            "clock": "1351090816",
```

```
            "value": "0.21",
            "ns": "522659528"
        },
        {
            "itemid": "23296",
            "clock": "1351090756",
            "value": "0.215",
            "ns": "507809457"
        },
        {
            "itemid": "23296",
            "clock": "1351090696",
            "value": "0.255",
            "ns": "495509699"
        },
        {
            "itemid": "23296",
            "clock": "1351090636",
            "value": "0.36",
            "ns": "477708209"
        },
        {
            "itemid": "23296",
            "clock": "1351090576",
            "value": "0.375",
            "ns": "463251343"
        },
        {
            "itemid": "23296",
            "clock": "1351090516",
            "value": "0.315",
            "ns": "447947017"
        },
        {
            "itemid": "23296",
            "clock": "1351090456",
            "value": "0.275",
            "ns": "435307141"
        }
    ],
    "id": 1
}
```


来源

ui/include/classes/api/services/CHistory.php 的 CHistory::get()。

发现检查

此类用于发现检查。

对象引用：

- Discovery check

可用方法：

- dcheck.get - 获取发现检查

> 发现检查对象

以下对象与 dcheck API 直接相关。

发现检查

发现检查对象定义由网络发现规则执行的特定检查。具有以下属性。

属性	类型	描述
dcheckid	string	(只读）发现检查的ID。
druleid	string	(只读）检查所属发现规则的 ID。
key_	string	此属性的值因检查类型而异： - 用于查询 Zabbix agent 检查的键值，必需； - 用于 SNMPv1、SNMPv2 和 SNMPv3 检查的 SNMP OID，必需。
ports	string	用逗号分隔的待检查的一个或多个端口范围。用于除 ICMP 以外的所有检查。 默认：0。
snmp_community	string	SNMP 社区。 对于 SNMPv1 and SNMPv2 agent 检查是必需的。
snmpv3_authpassphrase	string	用于安全等级设置为 *authNoPriv* 或 *authPriv* 的 SNMPv3 agent 检查的身份认证密码短语。
snmpv3_authprotocol	integer	用于安全等级设置为 *authNoPriv* 或 *authPriv* 的 SNMPv3 agent 检查的身份认证协议。 可用值： 0 - （默认） MD5； 1 - SHA1； 2 - SHA224； 3 - SHA256； 4 - SHA384； 5 - SHA512。
snmpv3_contextname	string	SNMPv3 上下文名称。仅用于 SNMPv3 检查。
snmpv3_privpassphrase	string	用于安全等级设置为 *authPriv* 的 SNMPv3 agent 检查的私密密码短语。

属性	类型	描述
snmpv3_privprotocol	integer	用于安全等级设置为 *authPriv* 的 SNMPv3 agent 检查的私密协议。 可用值： 0 - （默认） DES； 1 - AES128； 2 - AES192； 3 - AES256； 4 - AES192C； 5 - AES256C。
snmpv3_securitylevel	integer	用于 SNMPv3 agent 检查的安全等级。 可用值： 0 - noAuthNoPriv； 1 - authNoPriv； 2 - authPriv。
snmpv3_securityname	string	用于 SNMPv3 agent 检查的安全名称。
type (必需)	integer	检查类型。 可用值： 0 - SSH； 1 - LDAP； 2 - SMTP； 3 - FTP； 4 - HTTP； 5 - POP； 6 - NNTP； 7 - IMAP； 8 - TCP； 9 - Zabbix agent； 10 - SNMPv1 agent； 11 - SNMPv2 agent； 12 - ICMP ping； 13 - SNMPv3 agent； 14 - HTTPS； 15 - Telnet。
uniq	integer	是否将此检查作为设备唯一性标准。只能为发现规则配置一个唯一检查。用于 Zabbix agent、SNMPv1、SNMPv2 和 SNMPv3 agent 检查。 可用值： 0 - （默认）不将此检查作为唯一性标准； 1 - 将此检查作为唯一性标准。

属性	类型	描述
host_source	integer	主机名称的来源。 可用值： 1 - （默认） DNS； 2 - IP； 3 - 此检查的发现值。
name_source	integer	可见名称的来源。 可用值： 0 - （默认）未指定； 1 - DNS； 2 - IP； 3 - 此检查的发现值。

获取

描述

`integer/array dcheck.get(object parameters)`

此方法根据给定的参数检索发现检查。

Note:
此方法对任何类型的用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

`(object)` 定义需要输出的参数。

此方法支持以下参数。

参数	类型	描述
dcheckids	string/array	仅返回具有给定 ID 的发现检查。
druleids	string/array	仅返回属于给定发现规则的发现检查。
dserviceids	string/array	仅返回检查到给定被发现服务的发现检查。
sortfield	string/array	按照给定属性对结果进行排序。 可用值：`dcheckid` 和 `druleid`。
countOutput	boolean	这些参数对于所有的 `get` 方法都是通用的，详细描述请参见参考说明。
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

`(integer/array)` 返回其中一种结果：

- 一个对象数组；
- 如果使用了参数 `countOutput`，则返回检索到的对象的数量。

示例

检索发现规则的发现检查

检索被发现规则 “6” 使用的所有发现检查。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dcheck.get",
    "params": {
        "output": "extend",
        "dcheckids": "6"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "dcheckid": "6",
            "druleid": "4",
            "type": "3",
            "key_": "",
            "snmp_community": "",
            "ports": "21",
            "snmpv3_securityname": "",
            "snmpv3_securitylevel": "0",
            "snmpv3_authpassphrase": "",
            "snmpv3_privpassphrase": "",
            "uniq": "0",
            "snmpv3_authprotocol": "0",
            "snmpv3_privprotocol": "0",
            "host_source": "1",
            "name_source": "0"
        }
    ],
    "id": 1
}
```

来源

ui/include/classes/api/services/CDCheck.php 中的 CDCheck::get()。

发现的主机

该类设计用于发现的主机。

对象引用:

- 发现的主机

可用方法:

- dhost.get - 获取已发现的主机。

> 发现的主机对象

下列对象与 dhost API 直接相关。

发现主机

Note:
发现的主机是由 Zabbix server 创建的，不能通过 API 进行修改。

发现的主机对象包含一个被网络发现规则发现的主机的信息。其具有以下属性。

Property	Type	Description
dhostid	string	主机的 ID。
druleid	string	检测主机的发现规则的 ID。
lastdown	timestamp	发现主机最后异常的时间。
lastup	timestamp	发现主机最后正常的时间。
status	integer	无论发现主机的规则是正常还是异常。如果一个主机至少还有一个活动的发现服务，那么它就是正常的。 可能的值: 0 - 主机正常; 1 - 主机异常。

dhost.get

描述

`integer/array dhost.get(object parameters)`

这个方法允许根据给定的参数检索发现的主机。

Note:
任何类型的用户都可以使用此方法。可以在用户角色设置中撤销调用该方法的权限。参见用户角色 了解更多信息。

参数

`(object)` 定义需要输出的参数。

该方法支持以下参数。

参数	类型	描述
dhostids	字符串/数组	只返回给定 ID 的被发现的主机
druleids	字符串/数组	只返回发现规则创建的已发现主机。
dserviceids	字符串/数组	只返回指定服务的已发现主机。
selectDRules	查询	返回一个drules属性，其中包括检测到主机发现规则数组。
selectDServices	查询	返回dservices属性，发现的服务在主机上运行。 支持 `count`。

参数	类型	描述
limitSelects	整数	限制子选项返回的记录数 用于如下的子选项: `selectDServices` - 结果会按照 `dserviceid` 排序。
sortfield	字符串/数组	按照给出的属性对结果进行排序。 可能的值: `dhostid` 和 `druleid`.
countOutput	布尔值	参考评论中详细描述了所有 “get” 方法通用的这些参数。
editable	布尔值	
excludeSearch	布尔值	
filter	对象	
limit	整数	
output	查询	
preservekeys	布尔值	
search	对象	
searchByAny	布尔值	
searchWildcardsEnabled	布尔值	
sortorder	字符串/数组	
startSearch	布尔值	

返回值

`(integer/array)` 返回:

- 一组对象;
- 如果使用了 `countOutput` 参数，返回检索对象的数量。

示例

通过发现规则检索发现的主机

检索发现规则”4” 检测到的所有主机及其正在运行的已发现服务。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "dhost.get",
    "params": {
        "output": "extend",
        "selectDServices": "extend",
        "druleids": "4"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "dservices": [
                {
```

```
                "dserviceid": "1",
                "dhostid": "1",
                "type": "4",
                "key_": "",
                "value": "",
                "port": "80",
                "status": "0",
                "lastup": "1337697227",
                "lastdown": "0",
                "dcheckid": "5",
                "ip": "192.168.1.1",
                "dns": "station.company.lan"
            }
        ],
        "dhostid": "1",
        "druleid": "4",
        "status": "0",
        "lastup": "1337697227",
        "lastdown": "0"
    },
    {
        "dservices": [
            {
                "dserviceid": "2",
                "dhostid": "2",
                "type": "4",
                "key_": "",
                "value": "",
                "port": "80",
                "status": "0",
                "lastup": "1337697234",
                "lastdown": "0",
                "dcheckid": "5",
                "ip": "192.168.1.4",
                "dns": "john.company.lan"
            }
        ],
        "dhostid": "2",
        "druleid": "4",
        "status": "0",
        "lastup": "1337697234",
        "lastdown": "0"
    },
    {
        "dservices": [
            {
                "dserviceid": "3",
                "dhostid": "3",
                "type": "4",
                "key_": "",
                "value": "",
                "port": "80",
                "status": "0",
                "lastup": "1337697234",
                "lastdown": "0",
                "dcheckid": "5",
                "ip": "192.168.1.26",
                "dns": "printer.company.lan"
            }
        ],
        "dhostid": "3",
        "druleid": "4",
```

```
            "status": "0",
            "lastup": "1337697234",
            "lastdown": "0"
        },
        {
            "dservices": [
                {
                    "dserviceid": "4",
                    "dhostid": "4",
                    "type": "4",
                    "key_": "",
                    "value": "",
                    "port": "80",
                    "status": "0",
                    "lastup": "1337697234",
                    "lastdown": "0",
                    "dcheckid": "5",
                    "ip": "192.168.1.7",
                    "dns": "mail.company.lan"
                }
            ],
            "dhostid": "4",
            "druleid": "4",
            "status": "0",
            "lastup": "1337697234",
            "lastdown": "0"
        }
    ],
    "id": 1
}
```


参见

- 发现服务
- 发现规则

源代码

CDHost::get() in *ui/include/classes/api/services/CDHost.php*.

发现规则

此类用于网络发现规则。

> **Note:**
> 此 API 旨在使用网络发现规则。有关低级发现规则，请查看LLD rule API.

对象引用：

- Discovery rule

可用方法：

- drule.create - 创建新的发现规则
- drule.delete - 删除发现规则
- drule.get - 获取发现规则
- drule.update - 更新发现规则

> 发现规则对象

以下对象与 `drule` API 直接相关。

发现规则

发现规则对象定义一个网络发现规则。具有以下属性。

属性	类型	描述
druleid	字符	(只读）发现规则的ID。
iprange (必需)	字符	用逗号分隔的待检查的一个或多个 IP 范围。 有关支持的 IP 范围格式的更多信息，请参考网络发现配置 部分。
name (必需)	字符	发现规则的名称。
delay	字符	发现规则的执行间隔。支持秒、带后缀的时间单位和用户宏。 默认：1h。
nextcheck	时间戳	(只读）下一次执行发现规则的时间。
proxy_hostid	字符	用于发现的代理的ID。
status	整数	是否启用了发现规则。 可用值： 0 - （默认）启用； 1 - 禁用。

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

创建

描述

`object drule.create(object/array discoveryRules)`

此方法允许创建新的发现规则。

> **Note:**
> 此方法只有 *Admin*（管理员）和 *Super admin*（超级管理员）用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

`(object/array)` 要创建的发现规则。

除了标准发现规则属性，此方法接受以下参数。

参数	类型	描述
dchecks （必需）	array	发现checks 为发现规则创建。

返回值

`(object)` 返回一个对象，该对象包含 “druleids” 属性下创建的发现规则的 ID。返回 ID 的顺序与传递的发现规则的顺序相匹配。

示例

创建发现规则

创建一个发现规则来查找在内部网络中运行 Zabbix agent 的机器。该规则必须在端口 10050 上使用单个 Zabbix agent 检查。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "drule.create",
    "params": {
        "name": "Zabbix agent discovery",
        "iprange": "192.168.1.1-255",
        "dchecks": [
            {
                "type": "9",
                "key_": "system.uname",
                "ports": "10050",
                "uniq": "0"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "druleids": [
            "6"
        ]
    },
    "id": 1
}
```

参见

- Discovery check

来源

ui/include/classes/api/services/CDRule.php 中的 CDRule::create()。

删除

描述

`object drule.delete(array discoveryRuleIds)`

此方法允许删除发现规则。

Note:
此方法只有 *Admin*（管理员）和 *Super admin*（超级管理员）用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(array) 要删除的发现规则的 ID。

返回值

(object) 返回一个对象，该对象包含 “druleids” 属性下已删除的发现规则的 ID。

示例

删除多个发现规则

删除两个发现规则。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "drule.delete",
    "params": [
        "4",
        "6"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "druleids": [
            "4",
            "6"
        ]
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CDRule.php 中的 CDRule::delete()。

更新

描述

`object drule.update(object/array discoveryRules)`

此方法允许更新现有的发现规则。

Note:
此方法只有 *Admin*（管理员）和 *Super admin*（超级管理员）用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(object/array) 要更新的发现规则属性。

必须为每个发现规则定义 “druleid” 属性，所有其他属性是可选的。只有传递的属性将被更新，所有其他属性将保持不变。

除了标准发现规则属性，此方法接受以下参数。

参数	类型	描述
dchecks	array	发现checks 以替换现有的检查。

返回值

(object) 返回一个对象，该对象包含 “druleids” 属性下更新的发现规则的 ID。

示例

更改发现规则的 IP 范围

将发现规则的 IP 范围更改为 “192.168.2.1-255”。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "drule.update",
    "params": {
```

```
        "druleid": "6",
        "iprange": "192.168.2.1-255"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "druleids": [
            "6"
        ]
    },
    "id": 1
}
```

参见

- Discovery check

来源

ui/include/classes/api/services/CDRule.php 中的 CDRule::update()。

获取

描述

`integer/array drule.get(object parameters)`

此方法允许根据给定参数检索发现规则。

Note:
此方法对任何类型的用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

`(object)` 定义需要输出的参数。

此方法支持以下参数。

参数	类型	描述
dhostids	string/array	仅返回创建给定已发现主机的发现规则。
druleids	string/array	仅返回具有给定 ID 的发现规则。
dserviceids	string/array	仅返回创建给定已发现服务的发现规则。
selectDChecks	query	返回dchecks 属性，其中包含发现规则使用的发现检查。 支持 `count`。
selectDHosts	query	返回dhosts 属性，其中包含由发现规则创建的已发现主机。 Supports `count`.

参数	类型	描述
limitSelects	integer	限制子选择返回的记录数。 适用于以下子选择： `selectDChecks` - 结果将按照 `dcheckid` 排序； `selectDHosts` - 结果将按照 `dhostsid` 排序。
sortfield	string/array	根据给定的属性对结果进行排序。 可用值：`druleid` 和 `name`。
countOutput	boolean	这些参数对于所有的 `get` 方法都是通用的，详细描述请参见参考说明。
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

`(integer/array)` 返回其中一种结果：

- 一个对象数组；
- 如果使用了参数 `countOutput`，则返回检索到的对象的数量。

示例

检索所有发现规则

检索所有配置的发现规则及其使用的发现检查。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "drule.get",
    "params": {
        "output": "extend",
        "selectDChecks": "extend"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "druleid": "2",
```

```
            "proxy_hostid": "0",
            "name": "Local network",
            "iprange": "192.168.3.1-255",
            "delay": "5s",
            "nextcheck": "1348754327",
            "status": "0",
            "dchecks": [
                {
                    "dcheckid": "7",
                    "druleid": "2",
                    "type": "3",
                    "key_": "",
                    "snmp_community": "",
                    "ports": "21",
                    "snmpv3_securityname": "",
                    "snmpv3_securitylevel": "0",
                    "snmpv3_authpassphrase": "",
                    "snmpv3_privpassphrase": "",
                    "uniq": "0",
                    "snmpv3_authprotocol": "0",
                    "snmpv3_privprotocol": "0",
                    "host_source": "1",
                    "name_source": "0"
                },
                {
                    "dcheckid": "8",
                    "druleid": "2",
                    "type": "4",
                    "key_": "",
                    "snmp_community": "",
                    "ports": "80",
                    "snmpv3_securityname": "",
                    "snmpv3_securitylevel": "0",
                    "snmpv3_authpassphrase": "",
                    "snmpv3_privpassphrase": "",
                    "uniq": "0",
                    "snmpv3_authprotocol": "0",
                    "snmpv3_privprotocol": "0",
                    "host_source": "1",
                    "name_source": "0"
                }
            ]
        },
        {
            "druleid": "6",
            "proxy_hostid": "0",
            "name": "Zabbix agent discovery",
            "iprange": "192.168.1.1-255",
            "delay": "1h",
            "nextcheck": "0",
            "status": "0",
            "dchecks": [
                {
                    "dcheckid": "10",
                    "druleid": "6",
                    "type": "9",
                    "key_": "system.uname",
                    "snmp_community": "",
                    "ports": "10050",
                    "snmpv3_securityname": "",
                    "snmpv3_securitylevel": "0",
                    "snmpv3_authpassphrase": "",
```

```
                    "snmpv3_privpassphrase": "",
                    "uniq": "0",
                    "snmpv3_authprotocol": "0",
                    "snmpv3_privprotocol": "0",
                    "host_source": "2",
                    "name_source": "3"
                }
            ]
        }
    ],
    "id": 1
}
```

参见

- Discovered host
- Discovery check

来源

ui/include/classes/api/services/CDRule.php 中的 CDRule::get()。

告警

这个对象用于操作告警。

对象参考：

- Alert

可用方法：

- alert.get - 获取告警

> 告警对象

以下对象与 `alert` API 直接相关。

告警

Note:
告警是由 Zabbix server 创建，无法通过 API 修改。

告警对象包含有关某些动作操作是否已成功执行的信息。它具有以下属性。

属性	类型	描述
alertid	string	告警 ID。
actionid	string	生成告警的动作 ID。
alerttype	integer	告警类型。 可用值： 0 - 消息； 1 - 远程命令。
clock	timestamp	告警生成的时间。
error	string	发送消息或运行命令时出现问题时的错误文本。
esc_step	integer	告警生成期间的动作升级步骤。
eventid	string	触发动作的事件 ID。

属性	类型	描述
mediatypeid	string	用于发送消息的媒体类型 ID。
message	text	消息文本。用于消息告警。
retries	integer	Zabbix 尝试发送消息的次数。
sendto	string	地址，用户名或接收者的其他标识符。用于消息 alert。
status	integer	指示动作操作是否已成功执行的状态。 消息告警的可用值： 0 - 消息未发送。 1 - 消息已发送。 2 - 经多次重试后失败。 3 - 告警管理员尚未处理的新告警。 命令告警可用值： 0 - 命令没有运行。 1 - 命令运行成功。 2 - 尝试在 Zabbix agent 上运行命令但不可用。
subject	string	消息主题。用于消息告警。
userid	string	消息发送到的用户的 ID。
p_eventid	string	生成告警的问题事件的 ID。
acknowledgeid	string	生成告警的确认 ID。

获取

描述

`integer/array alert.get(object parameters)`

该方法允许根据给定的参数检索告警。

> **Note:**
> 此方法对于任何用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

`(object)` 定义所需输出的参数。

该方法支持以下参数。

参数	类型	描述
alertids	string/array	仅返回给定 ID 的告警。
actionids	string/array	仅返回给定动作生成的告警。
eventids	string/array	仅返回给定事件生成的告警。

参数	类型	描述
groupids	string/array	仅返回来自指定主机组的对象生成的告警。
hostids	string/array	仅返回来自指定主机的对象生成的告警。
mediatypeids	string/array	仅返回使用给定媒介类型的消息告警。
objectids	string/array	仅返回指定对象生成的告警。
userids	string/array	仅返回发送给指定用户的消息告警。
eventobject	integer	仅返回由与给定类型的对象相关的事件生成的告警。 参考 event "object" 获取受支持的对象类型列表。 默认：0 - 触发器。
eventsource	integer	仅返回由给定类型的事件生成的告警。 参考 event "source" 获取受支持的事件类型列表。 默认：0 - 触发器事件。
time_from	timestamp	仅返回在给定时间之后生成的告警。
time_till	timestamp	仅返回在给定时间之前生成的告警。
selectHosts	query	在hosts 属性中返回触发动作操作的主机。
selectMediatypes	query	在mediatypes 属性中以数组形式返回消息告警的媒介类型。
selectUsers	query	在users 属性中以数组形式返回消息发送到的用户。
sortfield	string/array	根据给定的属性排序结果。 可用值：`alertid`, `clock`, `eventid`, `mediatypeid`, `sendto` 和 `status`。
countOutput	boolean	这些参数对于所有 `get` 方法都是通用的，在参考说明 进行了描述。
editable	boolean	
excludeSearch	boolean	
filter	object	

参数	类型	描述
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

(integer/array) 返回以下任一选项：

- 对象数组；
- 如果使用了 countOutput 参数，则检索对象的计数。

示例

通过动作 ID 检索告警

返回所有动作 ID 为"3" 的告警。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "alert.get",
    "params": {
        "output": "extend",
        "actionids": "3"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "alertid": "1",
            "actionid": "3",
            "eventid": "21243",
            "userid": "1",
            "clock": "1362128008",
            "mediatypeid": "1",
            "sendto": "support@company.com",
            "subject": "PROBLEM: Zabbix agent on Linux server is unreachable for 5 minutes: ",
            "message": "Trigger: Zabbix agent on Linux server is unreachable for 5 minutes: \nTrigger stat
            "status": "0",
            "retries": "3",
            "error": "",
            "esc_step": "1",
            "alerttype": "0",
            "p_eventid": "0",
            "acknowledgeid": "0"
        }
    ],
    "id": 1
}
```

参见

- 主机
- 媒介类型

- 用户

来源

ui/include/classes/api/services/CAlert.php 中的 CAlert::get()。

图像

此类旨在处理图像。

对象引用：

- 图像

可用方法：

- 创建 - 创建新图像
- 删除 - 删除图像
- 获取 - 获取图像
- 更新 - 更新图像

> 图像对象

以下对象与 `image` API 直接相关。

图像

图像对象具有以下属性。

属性	类型	描述
imageid	string	（只读）图像 ID。
name （必需）	string	图像名称。
imagetype	integer	图像类型。 可用值： 1 - （默认）图标； 2 - 背景图。

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

创建

描述

`object image.create(object/array images)`

此方法用于创建新图像。

Note:
此方法仅允许 *Super admin*（超级管理员）类型的用户使用。调用此方法的权限可以在用户角色设置中撤销。更多信息请参阅用户角色 。

参数

`(object/array)` 创建的目标图像。

除了标准图像属性 之外，此方法同样适用以下参数。

参数	类型	描述
name （必需）	string	图像名称。

参数	类型	描述
imagetype (必需)	integer	图像类型。 可用类型： 1 - *(默认)* icon； 2 - background image。
image (必需)	string	Base64 类型编码图像。图像大小的最大限制为 1 MB。可以通过更改 ZBX_MAX_IMAGE_size 常量来调整最大限制值。 支持的图像格式有：PNG、JPEG、GIF。

返回值

(object) 返回一个对象，该对象包含 imageids 属性下创建的图像的 ID。返回 ID 的顺序与传递图像的顺序一致。

示例

创建新图像

创建云图标。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "image.create",
    "params": {
        "imagetype": 1,
        "name": "Cloud_(24)",
        "image": "iVBORw0KGgoAAAANSUhEUgAAABgAAAANCAYAAACzbK7QAAAABHNCSVQICAgIfAhkiAAAAAlwSFlzAAACmAAAApgB
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "imageids": [
            "188"
        ]
    },
    "id": 1
}
```

源码

ui/include/classes/api/services/CImage.php - CImage::create()

删除

描述

`object image.delete(array imageIds)`

此方法用于删除图像。

Note:
此方法仅允许 *Super admin*（超级管理员）类型的用户使用。调用此方法的权限可以在用户角色设置中撤销。更多信息请参阅用户角色 。

参数

(array) 指定删除图像的 ID。

返回值

(object) 返回一个对象，该对象包含 imageids 属性下已删除图像的 ID。

示例

批量删除图像

删除两个图像。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "image.delete",
    "params": [
        "188",
        "192"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "imageids": [
            "188",
            "192"
        ]
    },
    "id": 1
}
```

源码

ui/include/classes/api/services/CImage.php - CImage::delete()

更新

描述

`object image.update(object/array images)`

此方法用于更新现有的图片。

Note:
此方法仅允许 *Super admin*（超级管理员）类型的用户使用。调用此方法的权限可以在用户角色设置中撤销。更多信息请参阅用户角色 。

参数

(object/array) 更新的目标图像属性。

每个图象都必须定义 imageid 属性，其余属性为可选项。只有传递的属性将被更新，其他属性保持不变。

除了标准图像属性 ，该方法同样适用以下参数。

参数	类型	描述
image	string	Base64 类型编码图像。图像大小的最大限制为 1 MB。可以通过更改 ZBX_MAX_IMAGE_size 常量来调整最大限制值。 支持的图像格式有：PNG、JPEG、GIF。

返回值

(object) 返回一个对象，该对象包含 imageids 属性下更新的图像的 ID。

示例

重命名图像

将图像重命名为 “Cloud icon” 。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "image.update",
    "params": {
        "imageid": "2",
        "name": "Cloud icon"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "imageids": [
            "2"
        ]
    },
    "id": 1
}
```

源码

ui/include/classes/api/services/CImage.php - CImage::update()

获取

描述

integer/array image.get(object parameters)

此方法用于检索指定参数的图像。

> **Note:**
> 此方法仅允许 *Super admin*（超级管理员）类型的用户使用。调用此方法的权限可以在用户角色设置中撤销。更多信息请参阅用户角色 。

参数

(object) 定义目标输出参数。

此方法支持以下参数。

参数	类型	描述
imageids	string/array	仅返回指定 ID 的图像。
sysmapids	string/array	返回使用指定映射的图像。
select_image	flag	返回 image 属性，其中包含 Base64 类型编码图像。
sortfield	string/array	按照给定的属性对结果进行排序。 可用类型：imageid 和 name。
countOutput	boolean	这些参数对所有的 get 方法是通用的，详情请参阅参考说明 。
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

(integer/array) 返回其中之一：

- 一组对象；
- 如果使用了 countOutput 参数，则检索对象的计数。

示例

检索图像

检索 ID 为 “2” 的图像的所有数据。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "image.get",
    "params": {
        "output": "extend",
        "select_image": true,
        "imageids": "2"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "imageid": "2",
            "imagetype": "1",
            "name": "Cloud_(24)",
            "image": "iVBORw0KGgoAAAANSUhEUgAAABgAAAANCAYAAACzbK7QAAAABHNCSVQICAgIfAhkiAAAAAlwSFlzAAACmAAA
        }
    ],
    "id": 1
}
```

源码

ui/include/classes/api/services/CImage.php - CImage::get()

图标映射

这个类被设计用来处理图标映射。

对象引用：

- Icon map
- Icon mapping

可用的方法：

- iconmap.create - 创建新的图标映射
- iconmap.delete - 删除图标映射
- iconmap.get - 获取图标映射
- iconmap.update - 更新图标映射

> 图标映射对象

以下是和 `iconmap` API 直接相关的对象。

图标映射

图标映射（icon map）对象有以下属性。

属性	类型	描述
iconmapid	字符串	（只读）图标映射的 ID。
default_iconid （必需）	字符串	默认图标的 ID。
name （必需）	字符串	图标映射的名称。

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

图标映射关系

图标映射关系（icon mapping）对象定义了一个具体的图标，以供具有特定资产清单字段值的主机使用。它有以下属性。

属性	类型	描述
iconmappingid	字符串	（只读）图标映射关系（icon mapping）的 ID。
iconid （必需）	字符串	被图标映射关系（icon mapping）使用的图标 ID。
expression （必需）	字符串	匹配资产清单字段的表达式。

属性	类型	描述
inventory_link (必需)	**整数**	主机资产清单字段的 ID。 参考主机资产清单对象 (host inventory object) 以获取支持的资产清单字段列表。
iconmapid	字符串	(只读) 图标映射关系 (icon mapping) 所属的图标映射 (icon map) 的 ID。
sortorder	整数	(只读) 图标映射关系 (icon mapping) 在图标映射 (icon map) 中的位置。

iconmap.create

描述

`object iconmap.create(object/array iconMaps)`

此方法允许创建新的图标映射。

> **Note:**
> 此方法仅允许超级管理员类型的用户使用。调用此方法的权限可以在用户角色设置里撤销。更多信息请参见用户角色。

参数

(object/array) 要创建的图标映射。

除了标准的图标映射属性之外，此方法还接受以下参数。

参数	类型	描述
mappings（必需）	数组	为图标映射创建的多个图标映射关系（icon mappings）。

返回值

(object) 返回一个如下对象：在 iconmapids 属性下包含所创建图标映射的 ID。返回的多个 ID 的顺序与传参里多个图标映射的顺序相匹配。

示例

创建一个图标映射

创建一个图标映射以展现不同类型的主机。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "iconmap.create",
    "params": {
        "name": "Type icons",
        "default_iconid": "2",
        "mappings": [
            {
                "inventory_link": 1,
                "expression": "server",
                "iconid": "3"
            },
            {
                "inventory_link": 1,
                "expression": "switch",
                "iconid": "4"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "iconmapids": [
            "2"
        ]
    },
    "id": 1
}
```

参阅

- 图标映射关系（Icon mapping）

来源

来自 *ui/include/classes/api/services/CIconMap.php* 的 CIconMap::create() 。

iconmap.delete

描述

`object iconmap.delete(array iconMapIds)`

此方法允许删除图标映射。

Note:
此方法仅允许超级管理员类型的用户使用。调用此方法的权限可以在用户角色设置里撤销。更多信息请参见用户角色。

参数

(array) 要删除的多个图标映射的 ID。

返回值

(object) 返回一个如下对象：在 iconmapids 属性下包含所删除的多个图标映射的 ID。

示例

删除多个图标映射

删除两个图标映射。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "iconmap.delete",
    "params": [
        "2",
        "5"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "iconmapids": [
            "2",
            "5"
        ]
    },
    "id": 1
}
```

来源

来自 *ui/include/classes/api/services/CIconMap.php* 的 CIconMap::delete()。

更新

描述

`object iconmap.update(object/array iconMaps)`

此方法允许更新现有的图标映射。

Note:
此方法仅允许超级管理员类型的用户使用。调用此方法的权限可以在用户角色设置里撤销。更多信息请参见用户角色。

参数

(object/array) 要更新的图标映射的属性。

每个图标映射的 iconmapid 属性都必需被定义，所有其它属性都是可选的。只有传递的属性会被更新，其余的都会保持不变。

除了标准的图标映射属性之外，此方法还接受以下参数。

参数	类型	描述
mappings（必需）	数组	用来替换现有图标映射关系的图标映射。

返回值

(object) 返回一个如下对象：在 iconmapids 属性下包含所更新的多个图标映射的 ID。

Examples 例如

重命名一个图标映射

重命名一个图标映射为 “OS icons”。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "iconmap.update",
    "params": {
        "iconmapid": "1",
        "name": "OS icons"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "iconmapids": [
            "1"
        ]
    },
    "id": 1
}
```

参阅

- 图标映射关系（Icon mapping）

来源

来自 *ui/include/classes/api/services/CIconMap.php* 的 CIconMap::update()。

获取

描述

`integer/array iconmap.get(object parameters)`

此方法允许根据给定参数来获取图标映射。

> **Note:**
> 此方法仅允许超级管理员类型的用户使用。调用此方法的权限可以在用户角色设置里撤销。更多信息请参见用户角色。

参数

(object) 定义所需输出的参数

该方法支持如下参数。

参数	类型	描述
iconmapids	string/array	只返回具有给定 id 的图标映射。

参数	类型	描述
sysmapids	string/array	只返回在给定映射中使用的图标映射。
selectMappings	query	Return a mappings property with the icon mappings used.
sortfield	string/array	根据给定的属性对结果进行排序。 可选值: `iconmapid` 和 `name`。
countOutput	boolean	这些参数对于所有的 “get” 方法都是通用的，详细描述请参见reference commentary.
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

（整数/数组）返回任一：

- 对象数组；
- 如果已使用 countOutput 参数，则检索对象的计数。

例如

检索图标映射

检索所有关于图标映射”3” 的数据。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "iconmap.get",
    "params": {
        "iconmapids": "3",
        "output": "extend",
        "selectMappings": "extend"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "mappings": [
```

```
                {
                    "iconmappingid": "3",
                    "iconmapid": "3",
                    "iconid": "6",
                    "inventory_link": "1",
                    "expression": "server",
                    "sortorder": "0"
                },
                {
                    "iconmappingid": "4",
                    "iconmapid": "3",
                    "iconid": "10",
                    "inventory_link": "1",
                    "expression": "switch",
                    "sortorder": "1"
                }
            ],
            "iconmapid": "3",
            "name": "Host type icons",
            "default_iconid": "2"
        }
    ],
    "id": 1
}
```


参见

- 图标映射

来源

CIconMap::get() in *ui/include/classes/api/services/CIconMap.php*.

图表

这个类旨在为监控项使用。

对象参考：

- 图表

可用方法：

- graph.create - 创建图表
- graph.delete - 删除图表
- graph.get - 获取图表
- graph.update - 更新图表

> 图表对象

以下这些对象与 graph API 直接相关。

图表

图表对象具有以下属性。

属性	类型	描述
graphid	string	*(只读)* 图表的 ID。
height (必需)	integer	图表的高度，单位为像素。
name (必需)	string	图表的名称
width (必需)	integer	图表的宽度，单位为像素。

属性	类型	描述
flags	integer	*(只读)* 图表的来源。 可能的值是： 0 - *(默认)* 一个普通图表； 4 - 一个发现的图表。
graphtype	integer	图表的布局类型。 可能的值： 0 - *(默认)* 正常； 1 - 堆叠； 2 - 饼图； 3 - 爆炸图。
percent_left	float	左侧百分比。 默认：0。
percent_right	float	右侧百分比。 默认：0。
show_3d	integer	是否以 3D 形式显示饼图和爆炸图。 可能的值： 0 - *(默认)* 以 2D 显示； 1 - 以 3D 显示。
show_legend	integer	是否在图表上显示图例。 可能的值： 0 - 隐藏； 1 - *(默认)* 显示。
show_work_period	integer	是否在图表上显示工作时间。 可能的值： 0 - 隐藏； 1 - *(默认)* 显示。
show_triggers	integer	是否在图表上显示触发器线。 可能的值： 0 - 隐藏； 1 - *(默认)* 显示。
templateid	string	*(只读)* 父模板图表的 ID。
yaxismax	float	Y 轴的固定最大值。 默认：100。
yaxismin	float	Y 轴的固定最小值。 默认：0。
ymax_itemid	string	用作 Y 轴最大值的监控项的 ID。 从 Zabbix 6.0.7 开始，如果用户无法访问指定的监控项，图表将像 ymax_type 设置为'0'（计算得出）一样渲染。
ymax_type	integer	Y 轴最大值的计算方法。 可能的值： 0 - *(默认)* 计算得出； 1 - 固定； 2 - 监控项。
ymin_itemid	string	用作 Y 轴最小值的监控项的 ID。 从 Zabbix 6.0.7 开始，如果用户无法访问指定的监控项，图表将像 ymin_type 设置为'0'（计算得出）一样渲染。

属性	类型	描述
ymin_type	integer	Y 轴最小值的计算方法。 可能的值： 0 - (默认) 计算得出； 1 - 固定； 2 - 监控项。
uuid	string	通用唯一标识符，用于将导入的图表链接到已存在的图表。仅用于模板上的图表。如果没有给出，则自动生成。

请注意，对于一些方法（更新，删除），必需/可选参数组合是不同的。

创建

描述

`object graph.create(object/array graphs)`

此方法允许创建新的图表。

Note:
此方法只有 *Admin*（管理员）和 *Super admin*（超级管理员）用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(object/array) 要创建的图表。

除了标准图表属性之外，该方法还接受以下参数。

参数	类型	描述
gitems（必选）	array	要为图表创建的图表监控项。

返回值

(object) 返回一个对象，包含在 graphids 属性下创建的图表的 ID。返回的 ID 的顺序与传递的图表的顺序相匹配。

示例

创建一个图表

创建一个具有两个监控项的图表。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "graph.create",
    "params": {
        "name": "MySQL bandwidth",
        "width": 900,
        "height": 200,
        "gitems": [
            {
                "itemid": "22828",
                "color": "00AA00"
            },
            {
                "itemid": "22829",
                "color": "3333FF"
            }
        ]
```

```
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "graphids": [
            "652"
        ]
    },
    "id": 1
}
```

另见

- 图表监控项

来源

ui/include/classes/api/services/CGraph.php 中的 CGraph::create()。

删除

描述

`object graph.delete(array graphIds)`

此方法允许删除图表。

> **Note:**
> 此方法只有 *Admin*（管理员）和 *Super admin*（超级管理员）用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(array) 要删除的图表的 ID。

返回值

(object) 返回一个对象，包含在 graphids 属性下删除的图表的 ID。

示例

删除多个图表

删除两个图表。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "graph.delete",
    "params": [
        "652",
        "653"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "graphids": [
```

```
            "652",
            "653"
        ]
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CGraph.php 中的 CGraph::delete()。

更新

描述

`object graph.update(object/array graphs)`

此方法允许更新现有的图表。

Note:
此方法只有 *Admin*（管理员）和 *Super admin*（超级管理员）用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(`object/array`) 要更新的图表属性。

必须为每个图表定义"graphid " 属性，所有其他的属性是可选的。只有传递的属性会被更新，所有其他的属性将保持不变。

除了标准图形属性之外，该方法还接受以下参数。

参数	类型	描述
gitems	array	图表监控项来替换现有的图表监控项。如果一个图表监控项有定义 `gitemid` 属性，它将被更新，否则将创建一个新的图表监控项

返回值

(`object`) 返回一个包含 `graphids` 属性下的更新图表的 ID 的对象。

示例

设置 Y 刻度的最大值

将 Y 刻度的最大值设置为固定值 100。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "graph.update",
    "params": {
        "graphid": "439",
        "ymax_type": 1,
        "yaxismax": 100
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "graphids": [
            "439"
        ]
    },
```

```
    "id": 1
}
```

来源

ui/include/classes/api/services/CGraph.php 中的 CGraph::update()。

获取

描述

`integer/array graph.get(object parameters)`

该方法允许根据给定的参数来检索图表。

> **Note:**
> 此方法对任何类型的用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

`(object)` 参数定义了所需的输出。

该方法支持以下参数。

参数	类型	描述
graphids	string/array	只返回具有给定 ID 的图表。
groupids	string/array	只返回属于给定主机组中的主机的图表。
templateids	string/array	只返回属于给定模板的图表。
hostids	string/array	只返回属于给定主机的图表。
itemids	string/array	只返回包含给定监控项的图表。
templated	boolean	如果设置为 `true`，只返回属于模板的图表。
inherited	boolean	如果设置为 `true`，只返回从模板继承的图表。
expandName	flag	在图表名称中扩展宏。
selectGroups	query	返回一个组属性，其中包含该图表所属的主机组。
selectTemplates	query	返回一个模板属性，其中包含该图表所属的模板。
selectHosts	query	返回一个主机属性，其中包含该图表所属的主机。
selectItems	query	返回一个监控项属性，其中包含该图表中使用的监控项。

参数	类型	描述
selectGraphDiscovery	query	返回一个 `graphDiscovery` 属性和图表发现对象。图表发现对象将图表和创建图标的原型联系起来。 它具有以下属性：`graphid` - (`string`) 图表的 ID；`parent_graphid` - (`string`) 创建图表的图表原型的 ID。
selectGraphItems	query	返回一个图表监控项属性，其中包含该图表中使用的监控项。
selectDiscoveryRule	query	返回一个发现规则属性，其中包含创建该图表的低级发现规则。
filter	object	只返回那些与给定过滤器完全匹配的结果。 接受一个数组，其中键是属性名称，而值是其匹配的单一的值或一个数组的值。 支持额外的过滤器：`host` - 图表所属主机的技术名称；`hostid` - 图表所属主机的 ID。
sortfield	string/array	按给定的属性对结果进行排序。 可用值：`graphid`, `name` 和 `graphtype`。
countOutput	boolean	这些参数对所有的"get " 方法均是通用的，在参考说明页面中有详细描述。
editable	boolean	
excludeSearch	boolean	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

返回（integer/array）其中之一：

- 一个对象的数组；
- 如果使用了 countOutput 参数，则为检索到的对象的数量。

示例

从主机上检索图表

检索主机"10107 " 的所有图表，并按名称排序。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "graph.get",
    "params": {
        "output": "extend",
        "hostids": 10107,
        "sortfield": "name"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "graphid": "612",
            "name": "CPU jumps",
            "width": "900",
            "height": "200",
            "yaxismin": "0",
            "yaxismax": "100",
            "templateid": "439",
            "show_work_period": "1",
            "show_triggers": "1",
            "graphtype": "0",
            "show_legend": "1",
            "show_3d": "0",
            "percent_left": "0",
            "percent_right": "0",
            "ymin_type": "0",
            "ymax_type": "0",
            "ymin_itemid": "0",
            "ymax_itemid": "0",
            "flags": "0"
        },
        {
            "graphid": "613",
            "name": "CPU load",
            "width": "900",
            "height": "200",
            "yaxismin": "0",
            "yaxismax": "100",
            "templateid": "433",
            "show_work_period": "1",
            "show_triggers": "1",
            "graphtype": "0",
            "show_legend": "1",
            "show_3d": "0",
            "percent_left": "0",
```

```
            "percent_right": "0",
            "ymin_type": "1",
            "ymax_type": "0",
            "ymin_itemid": "0",
            "ymax_itemid": "0",
            "flags": "0"
        },
        {
            "graphid": "614",
            "name": "CPU utilization",
            "width": "900",
            "height": "200",
            "yaxismin": "0",
            "yaxismax": "100",
            "templateid": "387",
            "show_work_period": "1",
            "show_triggers": "0",
            "graphtype": "1",
            "show_legend": "1",
            "show_3d": "0",
            "percent_left": "0",
            "percent_right": "0",
            "ymin_type": "1",
            "ymax_type": "1",
            "ymin_itemid": "0",
            "ymax_itemid": "0",
            "flags": "0"
        },
        {
            "graphid": "645",
            "name": "Disk space usage /",
            "width": "600",
            "height": "340",
            "yaxismin": "0",
            "yaxismax": "0",
            "templateid": "0",
            "show_work_period": "0",
            "show_triggers": "0",
            "graphtype": "2",
            "show_legend": "1",
            "show_3d": "1",
            "percent_left": "0",
            "percent_right": "0",
            "ymin_type": "0",
            "ymax_type": "0",
            "ymin_itemid": "0",
            "ymax_itemid": "0",
            "flags": "4"
        }
    ],
    "id": 1
}
```


另见

- 发现规则
- 图表监控项
- 监控项
- 主机
- 主机组
- 模板

来源

ui/include/classes/api/services/CGraph.php 中的 CGraph::get()。

图表原型

这个类旨在为图表原型使用。

对象引用:

- 图表原型

可用的方法:

- graphprototype.create - 创建图表原型
- graphprototype.delete - 删除图表原型
- graphprototype.get - 检索图表原型
- graphprototype.update - 升级图表原型

> 图表原型对象

以下对象与 图表原型 API 直接相关。

图表原型

图表原型对象具有以下属性。

属性	类型	描述
graphid	string	*(只读)* 图表原型的 ID。
height (必需)	integer	图表原型的高度，单位为像素。
name (必需)	string	图表原型的名称。
width (必需)	integer	图表原型的宽度，单位为像素。
graphtype	integer	图表原型的布局类型。 可能的值： 0 - *(默认)* 正常； 1 - 堆叠图； 2 - 饼图； 3 - 爆炸饼图。
percent_left	float	左侧百分比。 默认：0。
percent_right	float	右侧百分比。 默认：0。
show_3d	integer	是否以 3D 形式显示发现的饼图和爆炸饼图。 可能的值： 0 - *(默认)* 以 2D 显示； 1 - 以 3D 显示。
show_legend	integer	是否在发现的图表上显示图例。 可能的值： 0 - 隐藏； 1 - *(默认)* 显示。
show_work_period	integer	是否在发现的图表上显示工作时间。 可能的值： 0 - 隐藏； 1 - *(默认)* 显示。
templateid	string	*(只读)* 父模板图表原型的 ID。

属性	类型	描述
yaxismax	float	Y 轴的固定最大值。
yaxismin	float	Y 轴的固定最小值。
ymax_itemid	string	用作 Y 轴最大值的监控项的 ID。 从 Zabbix 6.0.7 开始，如果用户无法访问指定的监控项，图表将像 ymax_type 设置为'0'（计算得出）一样渲染。
ymax_type	integer	Y 轴最大值的计算方法。 可能的值： 0 - *(默认)* 计算得出； 1 - 固定； 2 - 监控项。
ymin_itemid	string	用作 Y 轴最小值的监控项的 ID。 从 Zabbix 6.0.7 开始，如果用户无法访问指定的监控项，图表将像 ymin_type 设置为'0'（计算得出）一样渲染。
ymin_type	integer	Y 轴最小值的计算方法。 可能的值： 0 - *(默认)* 计算得出； 1 - 固定； 2 - 监控项。
discover	integer	图表原型发现状态。 可能的值： 0 - *(默认)* 将发现新图表； 1 - 不会发现新图表，并将现有图表标记为丢失。
uuid	string	通用唯一标识符，用于将导入的图表原型链接到已存在的原型。仅用于模板上的图表原型。如果没有给出，则自动生成。

请注意，对于一些方法（更新，删除），必需/可选参数组合是不同的。

创建图表原型

描述

`object graphprototype.create(object/array graphPrototypes)`

这种方法允许创建新的图表原型。

> **Note:**
> 这个方法只适用于 *Admin* 和 *Super admin* 用户类型。调用该方法的权限可以在用户角色设置中被撤销。前往用户角色 以了解更多信息。

参数

`(object/array)` 要创建的图表原型。

除了标准图表属性之外，该方法还接受以下参数。

参数	类型	描述
gitems （必选）	array	要为图表原型创建的图表项目。图形项目可以同时引用项目和项目原型，但至少要有一个项目原型。

返回值

`(object)` 返回一个对象，包含在 `graphids` 属性下创建的图表原型的 ID。返回的 ID 的顺序与传递的图表原型的顺序相匹配。

示例

创建一个图表原型

创建一个有两个项目的图表原型。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "graphprototype.create",
    "params": {
        "name": "Disk space usage {#FSNAME}",
        "width": 900,
        "height": 200,
        "gitems": [
            {
                "itemid": "22828",
                "color": "00AA00"
            },
            {
                "itemid": "22829",
                "color": "3333FF"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "graphids": [
            "652"
        ]
    },
    "id": 1
}
```

另见

- 图表监控项

来源

ui/include/classes/api/services/CGraphPrototype.php 的 CGraphPrototype::create()。

删除图表原型

描述

`object graphprototype.delete(array graphPrototypeIds)`

该方法允许删除图表原型。

> **Note:**
> 这个方法只适用于 *Admin* 和 *Super admin* 用户类型。调用该方法的权限可以在用户角色设置中被撤销。前往用户角色 以了解更多信息。

参数

(array) 要删除的图表原型的 ID。

返回值

(object) 返回一个包含在 graphids 属性下被删除图表原型的 ID 的对象。

示例

删除多个图表原型

删除两个图表原型。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "graphprototype.delete",
    "params": [
        "652",
        "653"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "graphids": [
            "652",
            "653"
        ]
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CGraphPrototype.php 的 CGraphPrototype::delete()。

更新图表原型

描述

`object graphprototype.update(object/array graphPrototypes)`

该方法允许更新现有的图表原型。

Note:
这个方法只适用于 *Admin* 和 *Super admin* 用户类型。调用该方法的权限可以在用户角色设置中被撤销。前往用户角色 以了解更多信息。

参数

(object/array) 要更新的图新原型属性。

graphid 属性必须为每个图表原型定义，所有其他属性是可选的。只有通过的属性将被更新，所有其他的将保持不变。

除了标准图形原型属性，该方法还接受以下参数。

参数	类型	描述
gitems	array	图表项目将替换现有的图表项目。如果一个图表项目定义了 gitemid 属性，它将会被更新，否则会创建一个新的图表项目。

返回值

(object) 返回一个包含 'graphids' 属性下更新的图表原型的 ID 的对象。

示例

改变图表原型的大小

将图表原型的大小改为 1100 * 400 像素。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "graphprototype.update",
    "params": {
        "graphid": "439",
        "width": 1100,
        "height": 400
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "graphids": [
            "439"
        ]
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CGraphPrototype.php 的 CGraphPrototype::update()。

查询图表原型

描述

`integer/array graphprototype.get(object parameters)`

该方法允许根据给定的参数来检索图表原型。

Note:
这个方法只适用于任何用户类型。调用该方法的权限可以在用户角色设置中被撤销。前往用户角色以了解更多信息。

参数

`(object)` 参数定义了所需的输出。

该方法支持下列参数。

参数	类型	描述
discoveryids	string/array	只返回属于给定发现规则的图表原型。
graphids	string/array	只返回具有给定 ID 的图表原型。
groupids	string/array	只返回属于给定主机组中的主机的图表原型。
hostids	string/array	只返回属于给定主机的图表原型。
inherited	boolean	如果设置为 `true`，只返回从模板继承的图表原型。
itemids	string/array	只返回包含给定监控项原型的图表原型。

参数	类型	描述
templated	boolean	如果设置为 `true`，只返回属于模板的图表原型。
templateids	string/array	只返回属于给定模板的图表原型。
selectDiscoveryRule	query	返回一个发现规则属性，其带有图表原型所属的 LLD 规则。
selectGraphItems	query	返回一个图表项属性，其中包含图表原型中使用的图表项目。
selectGroups	query	返回一个主机组属性，包含图表原型所属的主机组。
selectHosts	query	返回一个主机属性，包含图表原型所属的主机。
selectItems	query	返回一个 `items` 属性，包含图表原型中使用的监控项和监控项原型。
selectTemplates	query	返回一个模板属性，其带有含图表原型所属的模板。
filter	object	只返回那些与给定过滤器完全匹配的结果。 接受一个数组，其中键是属性名称，而值是匹配的一个单一的值或一个数组的值。 支持额外的过滤器: `host` - 图表原型所属的主机的技术名称。 `hostid` - 图表原型所属的主机的 ID。
sortfield	string/array	按给定的属性对结果进行排序。 可能的取值：`graphid`、`name` 和 `graphtype`。
countOutput	boolean	这些参数是所有 `get` 方法的共同参数，在参考注释中有详细描述。
editable	boolean	
excludeSearch	boolean	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	

参数	类型	描述
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

返回（integer/array）其中之一：

- 一个对象的数组；
- 如果使用了 countOutput 参数，则为检索到的对象的数量。

示例

从 LLD 规则中检索图表原型

从一个 LLD 规则中检索所有的图表原型。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "graphprototype.get",
    "params": {
        "output": "extend",
        "discoveryids": "27426"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "graphid": "1017",
            "parent_itemid": "27426",
            "name": "Disk space usage {#FSNAME}",
            "width": "600",
            "height": "340",
            "yaxismin": "0.0000",
            "yaxismax": "0.0000",
            "templateid": "442",
            "show_work_period": "0",
            "show_triggers": "0",
            "graphtype": "2",
            "show_legend": "1",
            "show_3d": "1",
            "percent_left": "0.0000",
            "percent_right": "0.0000",
            "ymin_type": "0",
            "ymax_type": "0",
            "ymin_itemid": "0",
            "ymax_itemid": "0",
            "discover": "0"
        }
    ],
    "id": 1
}
```

另见

- 发现规则
- 图表监控项
- 监控项

- 主机
- 主机组
- 模板

来源

ui/include/classes/api/services/CGraphPrototype.php 的 CGraphPrototype::get()。

图表监控项

这个类别旨在为主机使用。

对象引用：

- 图表监控项

可用方法：

- graphitem.get - 获取图表监控项

> 图表监控项对象

以下对象与 图表监控项 API 直接相关。

图表监控项

> **Note:**
> 图表监控项只能通过 `graphAPI` 进行修改。

图表监控项对象有以下属性。

属性	类型	描述
gitemid	string	（只读）图表监控项 ID。
color （必选）	string	图表监控项的绘制颜色是十六进制的颜色代码。
itemid （必选）	string	项目 ID。
calc_fnc	integer	监控项的值将被展示。 可用值： 1 - 最小值； 2 - （默认）平均值； 4 - 最大值； 7 - 所有值； 9 - 最后一个值，仅用于饼图和分散饼图。
drawtype	integer	图表监控项的绘制样式。 可用值： 0 - （默认）折线图； 1 - 填充图； 2 - 加粗折线图； 3 - 散点图； 4 - 虚线折线图； 5 - 梯度直方图。

属性	类型	描述
graphid	string	图表监控项所属图表的 ID。
sortorder	integer	该监控项在图表中的位置。 默认值：从 0 开始，每增加一个条目就增加一。
type	integer	图表监控项类型。 可用值： 0 - （默认）简单； 2 - 图表的总和，仅用于饼图和离散图。
yaxisside	integer	该图表项目的 Y 轴将被绘制在图表的一侧。 可用值： 0 - （默认）左侧； 1 - 右侧。

获取

描述

`integer/array graphitem.get(object parameters)`

该方法允许根据给定的参数来检索图表监控项。

Note:
这种方法对任何类型的用户都适用。调用该方法的权限可以在用户角色设置中撤销。前往用户角色以获取更多信息。

参数

`(object)` 定义所需输出的参数。

该方法支持以下参数。

参数	类型	描述
graphids	string/array	只返回属于给定图表的图表监控项。
itemids	string/array	只返回具有给定监控项 ID 的图表监控项。
type	integer	只返回具有给定类型的图表监控项。 请参考图表监控项对象页，了解支持的图表监控项类型列表。
selectGraphs	query	返回一个图表属性，其中包含该监控项所属的图表的数组。
sortfield	string/array	按给定的属性对结果进行排序。 可用值：`gitemid`。
countOutput	boolean	这些参数是所有”get ” 方法的共同参数，在参考说明页面中有详细描述。
editable	boolean	
limit	integer	
output	query	
preservekeys	boolean	
sortorder	string/array	

返回值

返回值是（`integer/array`）其中之一：

- 一个对象的数组；
- 如果使用了 `countOutput` 参数，则为检索到的对象的数量。

示例

从图表中检索图表项

检索图表中使用的图表项，以及有关监控项和主机的额外信息。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "graphitem.get",
    "params": {
        "output": "extend",
        "graphids": "387"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "gitemid": "1242",
            "graphid": "387",
            "itemid": "22665",
            "drawtype": "1",
            "sortorder": "1",
            "color": "FF5555",
            "yaxisside": "0",
            "calc_fnc": "2",
            "type": "0"
        },
        {
            "gitemid": "1243",
            "graphid": "387",
            "itemid": "22668",
            "drawtype": "1",
            "sortorder": "2",
            "color": "55FF55",
            "yaxisside": "0",
            "calc_fnc": "2",
            "type": "0"
        },
        {
            "gitemid": "1244",
            "graphid": "387",
            "itemid": "22671",
            "drawtype": "1",
            "sortorder": "3",
            "color": "009999",
            "yaxisside": "0",
            "calc_fnc": "2",
            "type": "0"
        }
    ],
    "id": 1
}
```

另见

- 图表

来源

ui/include/classes/api/services/CGraphItem.php 中的 CGraphItem::get()。

媒介类型

此类用于管理媒介类型。

对象引用：

- 媒介类型

可用方法:

- mediatype.create - 创建新的媒介类型
- mediatype.delete - 删除媒介类型
- mediatype.get - 获取媒介类型
- mediatype.update - 更新媒介类型

> 媒介类型对象

以下对象与 `mediatype` API 直接相关。

媒介类型

媒介类型对象具有以下属性。

属性	类型	描述
mediatypeid	string	(只读）媒介类型 ID。
名称 (必需)	string	媒介类型名称。
type (必需)	integer	媒介类型使用的传输方式。 可用值： 0 - 电子邮件； 1 - 脚本； 2 - SMS； 4 - Webhook。
exec_path	string	对于脚本媒介类型，`exec_path` 包含执行脚本的名称。 对于脚本媒介类型是必需的。
gsm_modem	string	GSM 调制解调器的串行设备名称。 对于 SMS 媒介类型是必需的。
passwd	string	身份验证密码。 用于电子邮件媒介类型。
smtp_email	string	用于发送通知的电子邮件地址。 电子邮件媒介类型所必需的。
smtp_helo	string	SMTP HELO。 对于电子邮件媒介类型是必需的。

属性	类型	描述
smtp_server	string	SMTP 服务器。 对于电子邮件媒介类型是必需的。
smtp_port	integer	要连接的 SMTP 服务器端口。
smtp_security	integer	要使用的 SMTP 连接安全级别。 可用值： 0 - 无； 1 - STARTTLS； 2 - SSL/TLS。
smtp_verify_host	integer	用于 SMTP 的 SSL 验证主机。 可用值： 0 - 否； 1 - 是。
smtp_verify_peer	integer	用于 SMTP 的 SSL 验证对等点。 可用值： 0 - 否； 1 - 是。
smtp_authentication	integer	要使用的 SMTP 身份验证方法。 可用值： 0 - 无； 1 - 普通密码。
status	integer	媒介类型是否启用。 可用值： 0 - （默认）启用； 1 - 禁用。
username	string	用户名。 用于电子邮件媒介类型。
exec_params	string	脚本参数。 每个参数都以新的换行符结尾。
maxsessions	integer	可以并行处理的最大警报数。 SMS 的可用值： 1 - （默认） 其他媒介类型的可用值： 0-100
maxattempts	integer	尝试发送警报的最大次数。 可用值： 1-100 默认值： 3

属性	类型	描述
attempt_interval	string	重试尝试之间的间隔。接受秒和带后缀的时间单位。 可用值： 0-1h 默认值： 10s
content_type	integer	消息格式。 可用值： 0 - 纯文本； 1 -（默认） html。
script	string	媒介类型 webhook 脚本 javascript 正文。
timeout	string	媒介类型 webhook 脚本超时。接受秒和带后缀的时间单位。 可用值： 1-60s 默认值： 30s
process_tags	integer	定义 webhook 脚本响应是否应被解释为标签，并且这些标签应添加到相关事件中。 可用值： 0 -（默认）忽略 webhook 脚本响应. 1 - 将 webhook 脚本响应作为标签处理。
show_event_menu	integer	在 `problem.get` 和 `event.get` 属性 `urls` 中显示媒介类型条目。 可用值： 0 -（默认）不添加为 `urls` 条目。 1 - 将媒介类型添加到 `urls` 属性。
event_menu_url	string	在 `problem.get` 和 `event.get` 的 `urls` 属性中定义媒介类型条目的 `url` 属性。
event_menu_name	string	在 `problem.get` 和 `event.get` 的 `urls` 属性中定义媒介类型条目的 `name` 属性。
parameters	array	webhook 输入参数的数组。

属性	类型	描述
description	string	媒介类型描述。

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

Webhook 参数

调用 webhook 脚本时传递的参数，具有以下属性。

属性	类型	描述
name （必需）	string	参数名称。
value	string	参数值，支持宏。 支持的宏在支持的宏页面中描述。

消息模板

消息模板对象定义了一个模板，该模板将用作动作操作发送通知的默认消息。具有以下属性。

属性	类型	描述
eventsource （必需）	integer	事件源。 可用值： 0 - 触发器； 1 - 自动发现； 2 - 自动注册； 3 - 采集器； 4 - 服务端。
recovery （必需）	integer	操作模式。 可用值： 0 - 自动操作； 1 - 恢复操作； 2 - 更新操作。
subject	string	消息主题。
message	string	消息文本。

创建

描述

`object mediatype.create(object/array mediaTypes)`

此方法允许创建新的媒体类型。

> **Note:**
> 此方法对于任何用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(object/array)‘ 需要创建的媒介类型。

除了标准媒介类型属性外，该方法还接受以下参数。

参数	类型	描述
parameters	array	要为媒体类型创建的Webhook 参数 。
message_templates	array	为媒体类型创建的消息模板 。

返回值

（`object`）返回一个包含在”mediatypeids” 属性下创建的媒介类型的 IDs 的对象，返回的顺序与传递的媒介类型的 ID 顺序匹配。

示例

创建电子邮件媒介类型

使用自定义 SMTP 端口和消息模板创建新的电子邮件媒介类型。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "mediatype.create",
    "params": {
        "type": "0",
        "name": "E-mail",
        "smtp_server": "mail.example.com",
        "smtp_helo": "example.com",
        "smtp_email": "zabbix@example.com",
        "smtp_port": "587",
        "content_type": "1",
        "message_templates": [
            {
                "eventsource": "0",
                "recovery": "0",
                "subject": "Problem: {EVENT.NAME}",
                "message": "Problem \"{EVENT.NAME}\" on host \"{HOST.NAME}\" started at {EVENT.TIME}."
            },
            {
                "eventsource": "0",
                "recovery": "1",
                "subject": "Resolved in {EVENT.DURATION}: {EVENT.NAME}",
                "message": "Problem \"{EVENT.NAME}\" on host \"{HOST.NAME}\" has been resolved at {EVENT.R
            },
            {
                "eventsource": "0",
                "recovery": "2",
                "subject": "Updated problem in {EVENT.AGE}: {EVENT.NAME}",
                "message": "{USER.FULLNAME} {EVENT.UPDATE.ACTION} problem \"{EVENT.NAME}\" on host \"{HOST
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "mediatypeids": [
            "7"
        ]
    },
    "id": 1
}
```

创建脚本媒介类型

创建一个新的脚本媒介类型，自定义失败尝试次数和尝试间隔时间。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "mediatype.create",
    "params": {
        "type": 1,
        "description": "Push notifications",
```

```
        "exec_path": "push-notification.sh",
        "exec_params": "{ALERT.SENDTO}\n{ALERT.SUBJECT}\n{ALERT.MESSAGE}\n",
        "maxattempts": "5",
        "attempt_interval": "11s"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "mediatypeids": [
            "8"
        ]
    },
    "id": 1
}
```

创建 webhook 媒介类型

创建新的 webhook 媒介类型。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "mediatype.create",
    "params": {
        "type": 1,
        "description": "Push notifications",
        "exec_path": "push-notification.sh",
        "exec_params": "{ALERT.SENDTO}\n{ALERT.SUBJECT}\n{ALERT.MESSAGE}\n",
        "maxattempts": "5",
        "attempt_interval": "11s"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "mediatypeids": [
            "9"
        ]
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CMediaType.php 中的 CMediaType::create()。

删除

描述

`object mediatype.delete(array mediaTypeIds)`

此方法允许删除媒介类型。

> **Note:**
> 此方法对于任何用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(array) 要删除的媒介类型的 ID。

返回值

(object) 返回一个对象，该对象包含 mediatypeids 属性下已删除媒介类型的 ID。

示例

删除多个媒介类型

删除两个媒介类型。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "mediatype.delete",
    "params": [
        "3",
        "5"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "mediatypeids": [
            "3",
            "5"
        ]
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CMediaType.php 中的 CMediaType::delete()。

更新

描述

`object mediatype.update(object/array mediaTypes)`

此方法允许更新现有的媒介类型。

> **Note:**
> 此方法对于任何用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(object/array) 需要被更新的媒介类型属性。

必须为每种媒介类型定义 mediatypeid 属性，其他所有属性都是可选的。只有传递的属性会被更新，其他所有的都将保持不变。

除了标准媒介类型属性外，此方法还接受以下参数。

参数	类型	描述
parameters	array	Webhook 参数 用于替换当前的 webhook 参数。
message_templates	array	消息模板 用于替换当前的消息模板。

返回值

(object) 返回包含 mediatypeids 属性下所更新 IDs 的对象。

示例

启用一个媒介类型

启用媒介类型，即设置其状态为 0。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "mediatype.update",
    "params": {
        "mediatypeid": "6",
        "status": 0
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "mediatypeids": [
            "6"
        ]
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CMediaType.php 中的 CMediaType::update()。

获取

描述

`integer/array mediatype.get(object parameters)`

此方法用于检索给定参数和符合条件的媒介类型。

> **Note:**
> 此方法对于任何用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(object) 参数定义了所需的输出。

此方法支持以下参数。

参数	类型	描述
mediatypeids	string/array	只返回带有给定 ID 列表的媒介类型。
mediaids	string/array	仅返回给定媒介使用的媒介类型。
userids	string/array	仅返回给定用户使用的媒介类型。
selectMessageTemplates	query	返回一个包含消息模板消息数组的属性。
selectUsers	query	返回使用媒介类型的用户属性。
sortfield	string/array	按给定属性对结果进行排序。 可用值：mediatypeid。
countOutput	boolean	这些参数对所有的 get 方法是通用的，详情请参阅参考说明 。
editable	boolean	
excludeSearch	boolean	
filter	object	

参数	类型	描述
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

（整型/数组）返回其中之一：

- 一个对象数组；
- 如果使用 `countOutput` 参数，返回被检索对象的数量。

示例

检索媒介类型

检索所有配置的媒介类型。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "mediatype.get",
    "params": {
        "output": "extend",
        "selectMessageTemplates": "extend"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "mediatypeid": "1",
            "type": "0",
            "name": "Email",
            "smtp_server": "mail.example.com",
            "smtp_helo": "example.com",
            "smtp_email": "zabbix@example.com",
            "exec_path": "",
            "gsm_modem": "",
            "username": "",
            "passwd": "",
            "status": "0",
            "smtp_port": "25",
            "smtp_security": "0",
            "smtp_verify_peer": "0",
            "smtp_verify_host": "0",
            "smtp_authentication": "0",
            "exec_params": "",
            "maxsessions": "1",
            "maxattempts": "3",
            "attempt_interval": "10s",
            "content_type": "0",
            "script": "",
            "timeout": "30s",
            "process_tags": "0",
```

```
        "show_event_menu": "1",
        "event_menu_url": "",
        "event_menu_name": "",
        "description": "",
        "message_templates": [
            {
                "eventsource": "0",
                "recovery": "0",
                "subject": "Problem: {EVENT.NAME}",
                "message": "Problem started at {EVENT.TIME} on {EVENT.DATE}\r\nProblem name: {EVENT.NA
            },
            {
                "eventsource": "0",
                "recovery": "1",
                "subject": "Resolved: {EVENT.NAME}",
                "message": "Problem has been resolved at {EVENT.RECOVERY.TIME} on {EVENT.RECOVERY.DATE
            },
            {
                "eventsource": "0",
                "recovery": "2",
                "subject": "Updated problem: {EVENT.NAME}",
                "message": "{USER.FULLNAME} {EVENT.UPDATE.ACTION} problem at {EVENT.UPDATE.DATE} {EVEN
            },
            {
                "eventsource": "1",
                "recovery": "0",
                "subject": "Discovery: {DISCOVERY.DEVICE.STATUS} {DISCOVERY.DEVICE.IPADDRESS}",
                "message": "Discovery rule: {DISCOVERY.RULE.NAME}\r\n\r\nDevice IP: {DISCOVERY.DEVICE.
            },
            {
                "eventsource": "2",
                "recovery": "0",
                "subject": "Autoregistration: {HOST.HOST}",
                "message": "Host name: {HOST.HOST}\r\nHost IP: {HOST.IP}\r\nAgent port: {HOST.PORT}"
            }
        ],
        "parameters": []
    },
    {
        "mediatypeid": "3",
        "type": "2",
        "name": "SMS",
        "smtp_server": "",
        "smtp_helo": "",
        "smtp_email": "",
        "exec_path": "",
        "gsm_modem": "/dev/ttyS0",
        "username": "",
        "passwd": "",
        "status": "0",
        "smtp_port": "25",
        "smtp_security": "0",
        "smtp_verify_peer": "0",
        "smtp_verify_host": "0",
        "smtp_authentication": "0",
        "exec_params": "",
        "maxsessions": "1",
        "maxattempts": "3",
        "attempt_interval": "10s",
        "content_type": "1",
        "script": "",
        "timeout": "30s",
```

```
            "process_tags": "0",
            "show_event_menu": "1",
            "event_menu_url": "",
            "event_menu_name": "",
            "description": "",
            "message_templates": [
                {
                    "eventsource": "0",
                    "recovery": "0",
                    "subject": "",
                    "message": "{EVENT.SEVERITY}: {EVENT.NAME}\r\nHost: {HOST.NAME}\r\n{EVENT.DATE} {EVENT
                },
                {
                    "eventsource": "0",
                    "recovery": "1",
                    "subject": "",
                    "message": "RESOLVED: {EVENT.NAME}\r\nHost: {HOST.NAME}\r\n{EVENT.DATE} {EVENT.TIME}"
                },
                {
                    "eventsource": "0",
                    "recovery": "2",
                    "subject": "",
                    "message": "{USER.FULLNAME} {EVENT.UPDATE.ACTION} problem at {EVENT.UPDATE.DATE} {EVEN
                },
                {
                    "eventsource": "1",
                    "recovery": "0",
                    "subject": "",
                    "message": "Discovery: {DISCOVERY.DEVICE.STATUS} {DISCOVERY.DEVICE.IPADDRESS}"
                },
                {
                    "eventsource": "2",
                    "recovery": "0",
                    "subject": "",
                    "message": "Autoregistration: {HOST.HOST}\r\nHost IP: {HOST.IP}\r\nAgent port: {HOST.P
                }
            ],
            "parameters": []
        }
    ],
    "id": 1
}
```


Retrieving media types as *Admin*

As an *Admin* type user, retrieve all media types that are enabled, with users that use these media types. The following example returns two media types:

- email media type with one user (since Zabbix 6.0.34, only *Admin* type user's own user);
- SMS media type with no users.

Request:


```
{
    "jsonrpc": "2.0",
    "method": "mediatype.get",
    "params": {
        "output": "extend",
        "filter": {
            "status": 0
        },
        "selectUsers": "extend"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
```

```
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "mediatypeid": "1",
            "type": "0",
            "name": "Email",
            "status": "0",
            "description": "",
            "maxattempts": "3",
            "users": [
                {
                    "userid": "3",
                    "username": "database-admin",
                    "name": "John",
                    "surname": "Doe",
                    "url": "",
                    "autologin": "0",
                    "autologout": "0",
                    "lang": "default",
                    "refresh": "30s",
                    "theme": "default",
                    "attempt_failed": "0",
                    "attempt_ip": "",
                    "attempt_clock": "0",
                    "rows_per_page": "50",
                    "timezone": "default",
                    "roleid": "2"
                }
            ]
        },
        {
            "mediatypeid": "3",
            "type": "2",
            "name": "SMS",
            "status": "0",
            "description": "",
            "maxattempts": "3",
            "users": []
        }
    ],
    "id": 1
}
```

参见

- 用户

来源

ui/include/classes/api/services/CMediaType.php 中的 CMediaType::get()。

审计日志

这个类用于操作审计日志。

对象参考：

- Audit log object

可用方法：

- auditlog.get - 获取审计日志记录

> 审计日志对象

以下对象与 auditlog API 直接相关。

审计日志

审计日志对象包含有关用户操作的信息。它具有以下属性。

属性	类型	描述
auditid	string	*(只读)* 审计日志条目的 ID。使用 CUID 算法生成。
userid	string	审计日志条目作者的用户 ID。
username	string	审计日志条目作者的用户名。
clock	timestamp	审计日志条目创建的时间戳。
ip	string	审计日志条目作者的 IP 地址。
action	integer	审计日志条目动作。 可能的值有： 0 - 添加； 1 - 更新； 2 - 删除； 4 - 注销； 7 - 执行； 8 - 登录； 9 - 登录失败； 10 - 清除历史记录。

属性	类型	描述
resourcetype	integer	审计日志条目资源类型。 可能的值有： 0 - 用户； 3 - 媒体类型； 4 - 主机； 5 - 动作； 6 - 图表； 11 - 用户组； 13 - 触发器； 14 - 主机组； 15 - 监控项； 16 - 图像； 17 - 值映射； 18 - 服务； 19 - 地图； 22 - Web 场景； 23 - 发现规则； 25 - 脚本； 26 - 代理； 27 - 维护； 28 - 正则表达式； 29 - 宏； 30 - 模板； 31 - 触发器原型； 32 - 图标映射； 33 - 仪表盘； 34 - 事件关联； 35 - 图表原型； 36 - 监控项原型； 37 - 主机原型； 38 - 自动注册； 39 - 模块； 40 - 设置； 41 - 管家； 42 - 认证； 43 - 模板仪表盘； 44 - 用户角色； 45 - API 令牌； 46 - 定时报告； 47 - 高可用性节点； 48 - SLA。
resourceid	string	审计日志条目资源标识符。
resourcename	string	审计日志条目资源的可读名称。
recordsetid	string	审计日志条目记录集 ID。在同一操作期间创建的审计日志记录将具有相同的记录集 ID。使用 CUID 算法生成。
details	text	审计日志条目详情。详情存储为 JSON 对象，其中每个属性名称是发生变更的属性或嵌套对象的路径，每个值包含有关此属性变更的数据，以数组格式。 可能的值格式有： ["add"] - 已添加嵌套对象； ["add", "< 值 >"] - 添加对象的属性包含 < 值 > ； ["update"] - 已更新嵌套对象； ["update", "< 新值 >", "< 旧值 >"] - 更新对象的属性值从 < 旧值 > 更改为 < 新值 > ； ["delete"] - 已删除嵌套对象。

获取

描述

`integer/array auditlog.get(object parameters)`

该方法允许根据给定的参数检索审计日志。

> **Note:**
> 此方法只有 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

`(object)` 定义所需输出的参数。

该方法支持以下参数。

参数	类型	描述
auditids	string/array	只返回给定 ID 的审计日志。
userids	string/array	只返回由给定用户创建的审计日志。
time_from	timestamp	只返回在给定时间之后或恰好在给定时间创建的审计日志条目。
time_till	timestamp	只返回在给定时间之前或恰好在给定时间创建的审计日志条目。
sortfield	string/array	根据给定属性对结果进行排序。 可能的值有：`auditid`, `userid`, `clock`。
filter	object	只返回完全匹配给定过滤器的结果。 接受一个数组，其中键是属性名称，值是要匹配的单个值或值数组。
search	object	在字段内容中进行不区分大小写的子字符串搜索：`username`, `ip`, `resourcename`, `details`。
countOutput	boolean	这些参数对所有 `get` 方法都是通用的，描述在参考注释中。
excludeSearch	boolean	
limit	integer	
output	query	
preservekeys	boolean	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

`(integer/array)` 返回以下任一选项：

- 对象数组；
- 如果使用了 `countOutput` 参数，则检索对象的计数。

示例

检索审计日志

检索两个最新的审计日志记录。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "auditlog.get",
    "params": {
        "output": "extend",
        "sortfield": "clock",
        "sortorder": "DESC",
        "limit": 2
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
```

```
        {
            "auditid": "cksstgfam0001yhdcc41y20q2",
            "userid": "1",
            "username": "Admin",
            "clock": "1629975715",
            "ip": "127.0.0.1",
            "action": "1",
            "resourcetype": "0",
            "resourceid": "0",
            "resourcename": "Jim",
            "recordsetid": "cksstgfal0000yhdcso67ondl",
            "details": "{\"user.name\":[\"update\",\"Jim\",\"\"],\"user.medias[37]\":[\"add\"],\"user.medi

        },
        {
            "auditid": "ckssofl0p0001yhdcqxclsg8r",
            "userid": "1",
            "username": "Admin",
            "clock": "1629967278",
            "ip": "127.0.0.1",
            "action": "0",
            "resourcetype": "0",
            "resourceid": "20",
            "resourcename": "John",
            "recordsetid": "ckssofl0p0000yhdcpxyo1jgo",
            "details": "{\"user.username\":[\"add\",\"John\"], \"user.userid:\":[\"add\",\"20\"],\"user.us
        }
    ],
    "id": 1
}
```


参见

- 审计日志对象

来源

CAuditLog::get() in *ui/include/classes/api/services/CAuditLog.php*.

报告

本章节介绍如何使用规划报告功能。有关报告可参考的信息：

- 报告
- 用户
- 用户组

用户可使用的功能：

- report.create - 创建新的规划报告
- report.delete - 删除已存在的规划报告
- report.get - 检索规划报告数据
- report.update - 更新规划报告数据

> 目标：报告

接下来介绍有关 `report`（报告） API 的相关内容。

报告

报告对象具有以下属性:

属性	类型	说明
reportid	字符串	*(只读)* 报告的 ID 号。
userid (必需)	字符串	创建该报告的用户 ID 号。
name (必需)	字符串	报告的唯一名称。
dashboardid (必需)	字符串	产生报告的仪表盘 ID 号。
period	整数	报告所要准备的时间区间。 可选值： 0 - *(缺省值)* 前一天； 1 - 前一周； 2 - 前一月； 3 - 前一年。
cycle	整数	报告重复发送的周期。 可选值： 0 - *(缺省值)* 以“天”为单位; 1 - 以“周”为单位； 2 - 以“月”为单位； 3 - 以“年”为单位。
start_time	整数	以“秒”为单位的一天中的精确时间，该时间设定为报告何时发送。 缺省值：0。
weekdays	整数	确认一周中有哪几天发送报告。 只有当配置为周报形式下才会需要配置该参数。 一周中的周几发送数据是以二进制的形式进行存储的。二进制下，每一位都代表着对应的天数。例如，十进制数字 12 转换为二进制为 1100，这意味着每周的周三和周四将发送报告。 缺省值：0。

属性	类型	说明
active_since	字符串	报告的起始日期。 可选值： 无输入 - *(缺省值)* 对该参数不做配置 (存储为 0)； 设定日期格式为 YYYY-MM-DD(以每天凌晨（00:00:00) 为基准，存储一个时间戳)。
active_till	字符串	报告的终了时间。 可选值： 无输入 - *(缺省值)* 对该参数不做配置 (存储为 0)； 设定日期格式为 YYYY-MM-DD(以每天凌晨 (23:59:59) 为基准，存储一个时间戳)。
subject	字符串	报告消息主题。
message	字符串	报告消息内容。
status	整数	该参数用来表示报告状态为启动还是关闭。 可选值： 0 - 关闭; 1 - *(缺省值)* 启动。
description	文本	报告的文字描述。
state	整数	*(只读)* 报告的状态。 可选值： 0 - *(缺省值)* 报告还未处理； 1 - 报告已生成且发送至所有目标； 2 - 报告生成失败；”info” 中包含有关发生错误的信息； 3 - 报告生成成功，但是未能发送至某些（或全部）目标；”info” 中包含有关发生错误的信息。
lastsent	timestamp	*(只读)* 在 Unix 系统下，最后成功发送报告的时间戳。
info	字符串	*(只读)* 有关错误的详细说明或者其他附加信息。

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

用户

用户对象包含下列属性特征：

属性	类型	说明
userid (必要)	字符串	需要发送报告的对象用户 ID。
access_userid	字符串	将为其生成报告的用户 ID。 0 - *(缺省值)* 按收件人生成报告。
exclude	整数	从发送的用户名单中删除目标用户。 Possible values: 0 - *(缺省值)* 包含； 1 - 删除。

用户组

用户组对象包含以下属性：| 属性 |类型| 说明 | |--------|----------|-----------| |**usrgrpid**
(必要)| 字符串 | 发送报告的用户组 ID。| |access_userid| 字符串 | 将为其生成报告的用户组 ID。

0 - *(缺省值)* 按收件人生成报告。|

报告. 创建

说明

`object report.create(object/array reports)`

该方法允许用户用于创建新的规划报告。

> **Note:**
> 该方式仅对管理员和超级管理员类型的用户有效。用户可以在用户职责设置中对该方式的使用权限进行设定修改。请参考用户角色以获取更多信息。

参数

`(object/array)` 计划报表的创建。

除了标准计划报表属性，该方法还接受以下参数。

参数	类型	描述
users	object/array	要发送报表的用户。
user_groups	object/array	要发送报表的用户组。

返回值

根据 `reportids` 的特性，`(object)` 会返回一个对象，包含已创建的规划报告 ID。返回的 ID 顺序与发送的规划报告顺序保持一致。

参考示例

创建计划报告

创建一个周报，从 2021-04-01 到 2021-08-31，每周一到周五 12:00，发送上一周的报告。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "report.create",
    "params": {
        "userid": "1",
        "name": "Weekly report",
        "dashboardid": "1",
        "period": "1",
        "cycle": "1",
        "start_time": "43200",
        "weekdays": "31",
```

```
        "active_since": "2021-04-01",
        "active_till": "2021-08-31",
        "subject": "Weekly report",
        "message": "Report accompanying text",
        "status": "1",
        "description": "Report description",
        "users": [
            {
                "userid": "1",
                "access_userid": "1",
                "exclude": "0"
            },
            {
                "userid": "2",
                "access_userid": "0",
                "exclude": "1"
            }
        ],
        "user_groups": [
            {
                "usrgrpid": "7",
                "access_userid": "0"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "reportids": [
            "1"
        ]
    },
    "id": 1
}
```

另请参考

- 用户
- 用户组

参考来源

CReport::create() in *ui/include/classes/api/services/CReport.php*.

报告. 删除

说明

`object report.delete(array reportids)`

该方法允许用户用于删除定时报表。

Note:
该方式仅对管理员和超级管理员类型的用户有效。用户可以在用户职责设置中对该方式的使用权限进行设定修改。请参考用户角色以获取更多信息。

参数

(array) 将要删除的规划报告 ID。

返回值

根据 reportids 的特性，(object) 返回一个包含已删除的规划报告 ID 的对象。

参考示例

删除多个计划报告

删除两个计划报告。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "report.delete",
    "params": [
        "1",
        "2"
    ],
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "reportids": [
            "1",
            "2"
        ]
    },
    "id": 1
}
```

参考来源

CReport::delete() in *ui/include/classes/api/services/CReport.php*.

报告. 更新

说明

`object report.update(object/array reports)`

该方法允许用户用来更新已配置的定时报表。

> **Note:**
> 该方式仅对管理员和超级管理员类型的用户有效。用户可以在用户角色设置中对该方式的使用权限进行设定修改。请参考用户角色以获取更多信息。

参数

(object/array) 计划报表属性的更新。

每个计划报表都必须定义 reportid 属性，所有其他属性都是可选的。只有传递的属性会被更新，其他属性将保持不变。

除了标准计划报表属性，该方法还接受以下参数。

参数	类型	描述
users	object/array	用户，用以替换当前分配给计划报表的用户。
user_groups	object/array	用户组，用以替换当前分配给计划报表的用户组。

返回值

根据 reportids 的特性，(object) 返回一个包含已更新定时报表 ID 的对象。

参考示例

关闭计划报表

请求：

```
{
    "jsonrpc": "2.0",
    "method": "report.update",
    "params": {
        "reportid": "1",
        "status": "0"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "reportids": [
            "1"
        ]
    },
    "id": 1
}
```

另请参考

- 用户
- 用户组

参考来源

CReport::update() in *ui/include/classes/api/services/CReport.php*.

报告. 获取

说明

`integer/array report.get(object parameters)`

该方法帮助用户根据所提供的参数来检索定时报表。

> **Note:**
> 该方法可供所有类型的用户使用。用户可以在用户职责设置中对该方式的使用权限进行设定修改。请参考用户角色以获取更多信息。

参数

(object) 定义了需求输出的参数。

该方法支持如下参数。

参数	类型	说明
reportids	字符串/数组	根据设定的 ID 号返回对应定时报表。
expired	布尔值	若设定为 true 则只返回失效的定时报表，若设定为 false，则只返回有效的定时报表。
selectUsers	搜索请求	返回报告所设定的发送目标的用户属性。
selectUserGroups	搜索请求	返回报告所定的发送目标的用户组 属性。
sortfield	字符串/数组	设定属性参数用来分类返回数据。 可设定的参数包括：reportid, name, status.
countOutput	布尔值	该类参数在所有的 get 方法中应用广泛，用户可以参考引用评论 页面来获取更多详细信息。
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	

参数	类型	说明
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

(integer/array) 返回下列两种数值之一：

- 一组对象；
- 在 countOutput 参数应用的情况下，返回检索对象的数量。

参考示例

检索报告数据

请求：

```
{
    "jsonrpc": "2.0",
    "method": "report.get",
    "params": [
        "output": "extend",
        "selectUsers": "extend",
        "selectUserGroups": "extend",
        "reportids": ["1", "2"]
    ],
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "reportid": "1",
            "userid": "1",
            "name": "Weekly report",
            "dashboardid": "1",
            "period": "1",
            "cycle": "1",
            "start_time": "43200",
            "weekdays": "31",
            "active_since": "2021-04-01",
            "active_till": "2021-08-31",
            "subject": "Weekly report",
            "message": "Report accompanying text",
            "status": "1",
            "description": "Report description",
            "state": "1",
            "lastsent": "1613563219",
            "info": "",
            "users": [
                {
                    "userid": "1",
                    "access_userid": "1",
                    "exclude": "0"
                },
                {
                    "userid": "2",
                    "access_userid": "0",
```

```
                    "exclude": "1"
                }
            ],
            "user_groups": [
                {
                    "usrgrpid": "7",
                    "access_userid": "0"
                }
            ]
        },
        {
            "reportid": "2",
            "userid": "1",
            "name": "Monthly report",
            "dashboardid": "2",
            "period": "2",
            "cycle": "2",
            "start_time": "0",
            "weekdays": "0",
            "active_since": "2021-05-01",
            "active_till": "",
            "subject": "Monthly report",
            "message": "Report accompanying text",
            "status": "1",
            "description": "",
            "state": "0",
            "lastsent": "0",
            "info": "",
            "users": [
                {
                    "userid": "1",
                    "access_userid": "1",
                    "exclude": "0"
                }
            ],
            "user_groups": []
        }
    ],
    "id": 1
}
```


另请参考

- 用户
- 用户组

参考来源

CReport::get() in *ui/include/classes/api/services/CReport.php*.

拓扑图

这个接口用于设计和处理拓扑

对象引用: - 拓扑 - 拓扑图元素 - 拓扑图关联 - 拓扑图 URL - 拓扑图用户 - 拓扑图用户组 - 拓扑图形状 - 拓扑图线

可用方法:

- `map.create` - 创建新拓扑
- `map.delete` - 删除拓扑
- `map.get` - 查看拓扑
- `map.update` - 更新拓扑

> 拓扑图对象

拓扑图

拓扑图对象具有以下属性.

属性	类型	说明
sysmapid	string	*(只读)* 拓扑图 ID.
height (必需)	整数	拓扑图的高度，以像素为单位.
name (必填)	字符串	拓扑图名称.
width (必需)	整数	拓扑图的宽度，以像素为单位.
backgroundid	字符串	用作拓扑图背景的图像 ID.
expand_macros	整数	配置拓扑图时是否在标签中展开宏. 参考值: 0 - （默认）不展开宏； 1 - 展开宏.
expandproblem	整数	是否为具有单个问题的元素显示问题触发器. 参考值： 0 - 始终显示问题的数量； 1 - （默认）如果只有一个问题，则显示问题触发器.
grid_align	整数	是否启用网格对齐. 参考值： 0 - 禁用网格对齐； 1 - （默认）启用网格对齐.
grid_show	整数	是否在拓扑图上显示网格. 参考值： 0 - 不显示网格； 1 - （默认）显示网格.
grid_size	整数	拓扑图网格的大小(以像素为单位）. 支持的值：20、40、50、75 和 100. 默认值：50.
highlight	整数	是否启用图标突出显示. 参考值： 0 - 突出显示禁用； 1 - （默认）突出显示启用.
iconmapid	字符串	拓扑图上使用的图标拓扑图的 ID.

属性	类型	说明
label_format	整数	是否启用高级标签. 参考值： 0 - （默认）禁用高级标签； 1 - 启用高级标签.
label_location	整数	拓扑图元素标签的位置. 参考值： 0 - （默认）底部； 1 - 左； 2 - 右；< br>3 - 顶部.
label_string_host	字符串	主机元素的自定义标签. 对于具有自定义主机标签类型的拓扑图是必需的.
label_string_hostgroup	字符串	主机组元素的自定义标签. 对于具有自定义主机组标签类型的拓扑图是必需的.
label_string_image	字符串	图像元素的自定义标签. 对于具有自定义图像标签类型的拓扑图是必需的.
label_string_map	字符串	拓扑图元素的自定义标签. 对于具有自定义拓扑图标签类型的拓扑图是必需的.
label_string_trigger	字符串	触发器元素的自定义标签. 对于具有自定义触发器标签类型的拓扑图是必需的.
label_type	整数	映射元素标签类型. 参考值： 0 - 标签； 1 - IP 地址； 2 - （默认）元素名称；< br>3 - 仅状态； 4 - 无.

属性	类型	说明
label_type_host	整数	主机元素的标签类型. 参考值： 0 - 标签； 1 - IP 地址； 2 - （默认）元素名称； 3 - 仅状态； 4 - 无； 5 - 自定义.
label_type_hostgroup	整数	主机组元素的标签类型. 参考值： 0 - 标签； 2 - （默认）元素名称； 3 -仅状态； 4 - 无； 5 - 自定义.
label_type_image	整数	主机组元素的标签类型. 参考值： 0 - 标签； 2 - （默认）元素名称； 4 -什么都没有； 5 - 自定义.
label_type_map	整数	拓扑图元素的标签类型. 参考值： 0 - 标签； 2 - （默认）元素名称； 3 - 状态仅； 4 - 无； 5 - 自定义.
label_type_trigger	整数	触发器元素的标签类型. 参考值： 0 - 标签； 2 - （默认）元素名称； 3 - 状态仅； 4 - 无； 5 - 自定义.
markelements	整数	是否突出显示最近更改状态的拓扑图元素. 参考值： 0 - （默认）不突出显示元素； 1 - 突出显示元素.

属性	类型	说明
severity_min	整数	将在拓扑图上显示的触发器的最小严重性. 请参阅触发器 "严重性" 属性获取支持的触发器严重性列表.
show_unack	整数	应该如何显示问题. 参考值： 0 - （默认）显示所有问题的计数； 1 - 仅显示未确认的计数问题； 2 - 分别显示已确认和未确认问题的计数.
userid	string	映射所有者用户 ID.
private	integer	拓扑图共享的类型. 参考值： 0 - 公共拓扑图； 1 - （默认）私人拓扑图.
show_suppressed	整数	是否显示抑制的问题. 参考值： 0 - （默认）隐藏抑制的问题； 1 - 显示抑制的问题.

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

拓扑图元素

拓扑图元素对象定义了在拓扑图上显示的对象. 它有以下属性.

属性	类型	说明
selementid	string	*(readonly)* 拓扑图元素的 ID.
元素 (必需)	array	元素数据对象. 主机、主机组、触发器和映射类型元素是必需的.
elementtype (必需)	integer	拓扑图元素的类型. 参考值： 0 - 主机； 1 - 拓扑图； 2 - 触发器； 3 - 主机组； 4 - 图片.
iconid_off (必需)	string	用于在默认状态下显示元素的图像的 ID.

属性	类型	说明
areatype	integer	应如何显示单独的主机组主机. 参考值： 0 - （默认）主机组元素将占据整个拓扑图； 1 -主机组元素将具有固定大小.
elementsubtype	integer	主机组元素应如何显示在拓扑图上. 参考值： 0 - （默认）将主机组显示为单个元素； 1 -
evaltype	integer	拓扑图元素标签过滤条件评估方法. 可用值： 0 - *(*默认*)* AND / OR; 2 - OR.
height	integer	固定大小的主机组元素的高度，以像素为单位. 默认值：200.
iconid_disabled	string	用于显示禁用拓扑图元素的图像 ID. 未用于图像元素.
iconid_maintenance	string	用于在维护中显示拓扑图元素的图像 ID. 未用于图像元素.
iconid_on	string	用于显示有问题的拓扑图元素的图像 ID. 未用于图像元素.
label	string	元素的标签.
label_location	integer	拓扑图元素标签的位置. 参考值： -1 - （默认）默认位置； 0 - 底部； 1 - 左侧; 2 - 右; 3 - 顶部.
permission	integer	权限级别的类型. 参考值： -1 - 无； 2 - 只读； 3 - 读写.
sysmapid	string	*(readonly)* 元素所属映射的 ID.

属性	类型	说明
urls	array	拓扑图元素 URL. 拓扑图元素 URL 对象是详细描述如下.
use_iconmap	integer	是否必须对宿主元素使用图标映射. 参考值： 0 - 不使用图标映射； 1 - （默认）使用图标映射.
viewtype	integer	主机组元素放置算法. 参考值： 0 - （默认）网格.
width	integer	固定大小的主机组元素的宽度，以像素为单位. 默认值：200.
x	integer	元素的 X 坐标，以像素为单位. 默认值：0.
y	integer	元素的 Y 坐标，以像素为单位. 默认值：0.

拓扑图元素的主机

拓扑图元素中的主机对象定义是一个主机元素

属性	类型	说明
hostid	string	主机 ID

拓扑图元素中的主机组

拓扑图元素中的主机组对象定义是一个主机组元素.

属性	类型	说明
groupid	string	主机组 ID

拓扑图元素中的拓扑图

拓扑图元素中的拓扑图对象默认是一个拓扑图元素

属性	类型	说明
sysmapid	string	拓扑图 ID

拓扑图元素中的触发器

拓扑图元素中的触发器对象定义的是一个或者多个触发器元素

属性	类型	说明
triggerid	string	触发器 ID

拓扑图元素标签

拓扑图元素标签对象具有以下属性。

属性	类型	描述
tag (必需)	string	拓扑图元素标签名称。
operator	integer	拓扑图元素标签条件运算符。 可能的值： 0 - *(默认)* 包含； 1 - 等于； 2 - 不包含； 3 - 不等于； 4 - 存在； 5 - 不存在。
value	string	拓扑图元素标签值。

拓扑图元素 URL

拓扑图元素 URL 对象定义了一个可点击的链接, 可用于拓扑图上特定类型的所有元素. 它具有以下特性:

属性	类型	说明
sysmapurlid	string	*(只读)* 拓扑图 URL ID
name (required)	string	链接标题.
url (required)	string	链接 URL
elementtype	integer	可以使用在 URL 上的拓扑图元素类型. 请参考拓扑图元素" 类型" 属性 拓扑图元素可用 URL 类型 默认：0
sysmapid	string	所属 URL 的拓扑图 ID

拓扑图链接

拓扑图链接对象定义了两个拓扑图元素之间的链接. 它有以下属性.

属性	类型	描述
linkid	string	*(readonly)* 拓扑图链接的 ID.
selementid1 (必需)	string	连接在一端的第一个拓扑图元素的 ID.
selementid2 (必需)	string	连接到另一端的第一个拓扑图元素的 ID.
color	string	作为十六进制颜色代码的线条颜色. 默认值：000000.

属性	类型	描述
drawtype	integer	链接线绘制样式. 参考值： 0 - （默认）线； 2 - 粗线； 3 - 虚线； 4 - 虚线.
label	string	链接标签.
linktriggers	array	拓扑图链接触发器用作链接状态指示器. 拓扑图链接触发器对象详细描述如下.
permission	integer	权限级别的类型. 参考值： -1 - 无； 2 - 只读； 3 - 读写.
sysmapid	string	链接所属拓扑图的 ID.

拓扑图链接触发器

拓扑图链接触发器对象根据触发器的状态定义了一个拓扑图链接状态指示器。它具有以下属性：

属性	类型	描述
linktriggerid	string	(只读）拓扑图链接触发器的 ID。
triggerid (必需)	string	用作链接指示器的触发器 ID。
color	string	指示器颜色，以十六进制颜色代码表示。 默认值：DD0000。
drawtype	integer	指示器绘制样式。 可能的值： 0 - （默认）线条； 2 - 粗线条； 3 - 点状线； 4 - 虚线。
linkid	string	拓扑图链接触发器所属的链接 ID。

拓扑图 URL

拓扑图 URL 对象定义了一个可点击的链接, 可用于映射上特定类型的所有元素. 它具有以下特性:

属性	类型	说明
sysmapurlid	string	*(只读)* 拓扑图 URL ID
name (required)	string	链接标题.
url (required)	string	链接 URL
elementtype	integer	可以使用在 URL 上的拓扑图元素类型. 请参考拓扑图元素” 类型” 属性 拓扑图元素可用 URL 类型 默认：0
sysmapid	string	所属 URL 的拓扑图 ID

拓扑图用户

基于用户的拓扑图权限列表. 它具有以下特性:

属性	类型	说明
sysmapuserid	string	*(只读)* 拓扑图用户 ID
userid (必须)	string	用户 ID.
permission (必须)	integer	权限等级类型 参考值： 2 - 只读 3 - 可读可写

拓扑图用户组

基于用户组的拓扑图权限列表. 它具有以下特性:

属性	类型	说明
sysmapusrgrpid	string	*(只读)* 拓扑图用户组的 ID
usrgrpid (必须)	string	用户组 ID.
permission (必须)	integer	权限等级类型 参考值： 2 - 只读 3 - 可读可写

拓扑图形状

拓扑图形状对象定义了在拓扑图上显示的几何形状（有或没有文本），它具有以下属性

属性	类型	说明
sysmap_shapeid	string	*(只读)* 拓扑图形状元素的 ID.
type（必填）	integer	拓扑图形状元素的类型. 参考值： 0 - 矩形； 1 - 椭圆.
x	integer	需要属性创建新形状时. 以像素为单位的形状的 X 坐标. 默认值：0.
y	integer	以像素为单位的形状的 Y 坐标. 默认值：0.
width	integer	形状的宽度，以像素为单位. 默认值：200.
height	integer	形状的高度，以像素为单位. 默认值：200.
text	string	形状的文本.

属性	类型	说明
font	integer	形状内文本的字体. 参考值: 0 - Georgia, serif 1 - “Palatino Linotype”, “Book Antiqua”, Palatino, serif 2 - “Times New Roman”, Times, serif 3 - Arial, Helvetica, sans-serif 4 - “Arial Black”, Gadget, sans-serif 5 - “Comic Sans MS”, cursive, sans-serif 6 - Impact, Charcoal, sans-serif 7 - “Lucida Sans Unicode”, “Lucida Grande”, sans-serif 8 - Tahoma, Geneva, sans-serif 9 - “Trebuchet MS”, Helvetica, sans-serif 10 - Verdana, Geneva, sans-serif 11 - “Courier New”, Courier, monospace 12 - “Lucida Console”, Monaco, monospace Default: 9.
font_size	integer	字体大小，以像素为单位. 默认值：11.
font_color	string	字体颜色. 默认值：’000000’.
text_halign	integer	文本的水平对齐方式. 参考值： 0 - 居中； 1 - 左； 2 - 右. 默认值：0.

属性	类型	说明
text_valign	integer	文本的垂直对齐方式. 参考值： 0 - 中间； 1 - 顶部； 2 - 底部. 默认值：0.
border_type	integer	边框的类型. 参考值： 0 - 无； 1 - ──────； 2 - ╌╌； 3 - – – –. 默认值：0.
border_width	integer	以像素为单位的边框宽度. 默认值：0.
border_color	string	边框颜色. 默认值：’000000’.
background_color	string	背景颜色（填充颜色）. 默认值：（空）.
zindex	integer	用于对所有形状和线条进行排序的值 (z-index). 默认值：0.

拓扑图线

该对象定义在拓扑图上显示的线. 它有以下属性：

属性	类型	描述
sysmap_shapeid	string	*(只读)* 拓扑图形状元素的 ID.
x1	integer	线点 1 的 X 坐标，以像素为单位. 默认值：0.
y1	integer	线点 1 的 Y 坐标，以像素为单位. 默认值：0.
x2	integer	线点 2 的 X 坐标，以像素为单位. 默认值：200.
y2	integer	线点 2 的 Y 坐标，以像素为单位. 默认值：200.

属性	类型	描述
line_type	integer	线的类型. 参考值： 0 - 无； 1 - ————; 2 - ┅; 3 - – – –. 默认值：0.
line_width	integer	线条的宽度，以像素为单位. 默认值：0.
line_color	string	线条颜色. 默认值：’000000’.
zindex	integer	用于对所有形状和线条进行排序的值 (z-index). 默认值：0.

拓扑图创建

描述

`object map.create(object/array maps)`

这个方法允许创建一个新的拓扑图

Note:
此方法适用于任何类型的用户. 调用该方法的权限可以在用户角色设置中撤销. 请参阅用户角色了解更多信息.

参数

(object/array) 要创建的拓扑图。

除了标准拓扑图属性外，该方法还接受以下参数

参数	类型	说明
links	array	要在拓扑图上创建的拓扑图links。
selements	array	要在拓扑图上创建的拓扑图元素。
urls	array	要在拓扑图上创建的拓扑图URLs。
users	array	要在拓扑图上创建的拓扑图共享user 。
userGroups	array	要在拓扑图上创建的拓扑图用户组 共享。
shapes	array	要在拓扑图上创建的拓扑图shapes。
lines	array	要在拓扑图上创建的拓扑图lines。

:::noteclassic 要创建地图链接，您需要将地图元素 selementid 设置为任意值，然后使用此值在链接 selementid1 或 selementid2 属性中引用此元素。创建元素时，该值将替换为 Zabbix 生成的正确 ID, 参考

返回值

（对象）返回一个对象，该对象包含在”sysmapid” 属性下创建的拓扑图的 id。返回 id 的顺序与传递的拓扑图的顺序相匹配。

案例

创建一个空的拓扑图

创建一个没有任何元素的拓扑图

请求:

```
{
    "jsonrpc": "2.0",
    "method": "map.create",
```

```
    "params": {
        "name": "Map",
        "width": 600,
        "height": 600
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "sysmapids": [
            "8"
        ]
    },
    "id": 1
}
```

创建主机映射

创建一个包含两个主机元素和它们之间链接的拓扑图。请注意在拓扑图链接对象中使用临时 “selementid1” 和 “selementid2” 值来引用拓扑图元素。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "map.create",
    "params": {
        "name": "Host map",
        "width": 600,
        "height": 600,
        "selements": [
            {
                "selementid": "1",
                "elements": [
                    {"hostid": "1033"}
                ],
                "elementtype": 0,
                "iconid_off": "2"
            },
            {
                "selementid": "2",
                "elements": [
                    {"hostid": "1037"}
                ],
                "elementtype": 0,
                "iconid_off": "2"
            }
        ],
        "links": [
            {
                "selementid1": "1",
                "selementid2": "2"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "sysmapids": [
            "9"
        ]
    },
    "id": 1
}
```

创建一个触发器拓扑图

创建一个包含两个触发器元素的拓扑图。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "map.create",
    "params": {
        "name": "Trigger map",
        "width": 600,
        "height": 600,
        "selements": [
            {
                "elements": [
                    {"triggerid": "12345"},
                    {"triggerid": "67890"}
                ],
                "elementtype": 2,
                "iconid_off": "2"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "sysmapids": [
            "10"
        ]
    },
    "id": 1
}
```

拓扑图分享

创建具有两种共享类型（用户和用户组）的拓扑图。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "map.create",
    "params": {
        "name": "Map sharing",
        "width": 600,
        "height": 600,
        "users": [
            {
                "userid": "4",
                "permission": "3"
```

```
            }
        ],
        "userGroups": [
            {
                "usrgrpid": "7",
                "permission": "2"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "sysmapids": [
            "10"
        ]
    },
    "id": 1
}
```

拓扑图形状

创建一个带有主题的拓扑图

请求:

```
{
    "jsonrpc": "2.0",
    "method": "map.create",
    "params": {
        "name": "Host map",
        "width": 600,
        "height": 600,
        "shapes": [
            {
                "type": 0,
                "x": 0,
                "y": 0,
                "width": 600,
                "height": 11,
                "text": "{MAP.NAME}"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "sysmapids": [
            "10"
        ]
    },
    "id": 1
}
```

拓扑图线

创建拓扑图线。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "map.create",
    "params": {
        "name": "Map API lines",
        "width": 500,
        "height": 500,
        "lines": [
            {
                "x1": 30,
                "y1": 10,
                "x2": 100,
                "y2": 50,
                "line_type": 1,
                "line_width": 10,
                "line_color": "009900"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "sysmapids": [
            "11"
        ]
    },
    "id": 1
}
```

查看相关

- 拓扑图元素
- 拓扑图关联
- 拓扑图 URL
- 拓扑图用户
- 拓扑图用户组
- 拓扑图形状
- 拓扑图线

来源

CMap::create() in *ui/include/classes/api/services/CMap.php*.

拓扑图删除

描述

`object map.delete(array mapIds)`

这个方法允许删除拓扑图

> **Note:**
> 此方法适用于任何类型的用户. 调用该方法的权限可以在用户角色设置中撤销. 请参阅用户角色了解更多信息.

参数

(array) 要删除的拓扑图的 id 列表.

返回值

返回包含"sysmapid" 属性下的已删除拓扑图的 IDs 的对象。

案例

删除多个拓扑图

删除 2 个拓扑

请求:

```
{
    "jsonrpc": "2.0",
    "method": "map.delete",
    "params": [
        "12",
        "34"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "sysmapids": [
            "12",
            "34"
        ]
    },
    "id": 1
}
```

来源

CMap::delete() in *ui/include/classes/api/services/CMap.php*.

更新拓扑图

描述

`object map.update(object/array maps)`

此方法可以用来更新已存在的拓扑图

> **Note:**
> 此方法适用于任何类型的用户. 调用该方法的权限可以在用户角色设置中撤销. 请参阅用户角色了解更多信息.

参数

`(object/array)` 要更新的拓扑图属性。

必须为每个地图定义 "mapid" 属性，所有其他属性是可选的。只有传递的属性会被更新，所有其他的将保持不变。

除了标准地图属性，该方法接受以下参数。

参数	类型	说明
links	array	Map links 来替换现有的链接。
selements	array	Map elements 替换现有元素。
urls	array	拓扑图URLs 以替换现有的 URL。
users	array	Map user 共享以替换现有元素。
userGroups	array	拓扑图用户组 共享以替换现有元素。
shapes	array	Map shapes 来替换现有的形状。
lines	array	Map lines 来替换现有的行。

:::noteclassic 要创建地图链接，您需要将地图元素 selementid 设置为任意值，然后使用此值在链接 selementid1 或 selementid2 属性中引用此元素。创建元素时，该值将替换为 Zabbix 生成的正确 ID, 参考

返回值

返回一个对象，该对象包含”sysmapid” 属性下更新的映射的 id.

案例

调整拓扑大小

将拓扑大小更改为 1200x1200 像素.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "map.update",
    "params": {
        "sysmapid": "8",
        "width": 1200,
        "height": 1200
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "sysmapids": [
            "8"
        ]
    },
    "id": 1
}
```

改变拓扑图的拥有者

仅适用于管理员和超级管理员

请求:

```
{
    "jsonrpc": "2.0",
    "method": "map.update",
    "params": {
        "sysmapid": "9",
        "userid": "1"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 2
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "sysmapids": [
            "9"
        ]
    },
    "id": 2
}
```

查看相关

- 拓扑图元素

- 拓扑图关联
- 拓扑图 URL
- 拓扑图用户
- 拓扑图用户组
- 拓扑图形状
- 拓扑图线

来源

CMap::update() in *ui/include/classes/api/services/CMap.php*.

获取拓扑图

描述

`integer/array map.get(object parameters)` 这个方法允许根据给定参数查询出符合条件的拓扑图。

> **Note:**
> 此方法适用于任何类型的用户. 调用该方法的权限可以在用户角色设置中撤销. 请参阅用户角色了解更多信息.

参数

`(object)` 定义所需输出的参数。

该方法支持以下参数。

参数	类型	说明
sysmapids	string/array	只返回给定 ID 的拓扑图。
userids	string/array	只返回属于给定用户 ID 的拓扑图。
expandUrls	flag	将全局地图 URL 添加到相应的地图元素，并在所有地图元素 URL 中展开宏。
selectIconMap	query	返回一个iconmap 属性以及地图上使用的图标地图。
selectLinks	query	返回一个links 属性，其中包含元素之间的地图链接。
selectSelements	query	返回带有地图元素的selements 属性。
selectUrls	query	返回带有地图 URL 的urls 属性。
selectUsers	query	返回一个users 属性，其中包含与地图共享的用户。
selectUserGroups	query	返回一个userGroups 属性，其中包含与地图共享的用户组。
selectShapes	query	返回带有地图形状的shapes 属性。
selectLines	query	返回带有地图线的lines 属性。
sortfield	string/array	按给定属性对结果进行排序。 可用值：`name`、`width` 和 `height`。
countOutput	boolean	这些参数对所有 `get` 方法都是通用的，在参考评论 中有详细描述。
可编辑	boolean	
排除搜索	boolean	
过滤器	object	
限制	integer	
输出	query	
preservekeys	boolean	
搜索	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
排序	string/array	
开始搜索	boolean	

返回值

(整型/数组) 返回其中之一：

- 一个对象数组；
- 如果使用 `countOutput` 参数, 查询对象的数量.

案例

查询拓扑图

检索关于拓扑图"3" 的所有数据

请求:

```
{
    "jsonrpc": "2.0",
    "method": "map.get",
    "params": {
        "output": "extend",
        "selectSelements": "extend",
        "selectLinks": "extend",
        "selectUsers": "extend",
        "selectUserGroups": "extend",
        "selectShapes": "extend",
        "selectLines": "extend",
        "sysmapids": "3"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "selements": [
                {
                    "selementid": "10",
                    "sysmapid": "3",
                    "elementtype": "4",
                    "evaltype": "0",
                    "iconid_off": "1",
                    "iconid_on": "0",
                    "label": "Zabbix server",
                    "label_location": "3",
                    "x": "11",
                    "y": "141",
                    "iconid_disabled": "0",
                    "iconid_maintenance": "0",
                    "elementsubtype": "0",
                    "areatype": "0",
                    "width": "200",
                    "height": "200",
                    "tags": [
                        {
                            "tag": "service",
                            "value": "mysqld",
                            "operator": "0"
                        }
                    ],
                    "viewtype": "0",
                    "use_iconmap": "1",
                    "urls": [],
                    "elements": []
                },
                {
                    "selementid": "11",
                    "sysmapid": "3",
                    "elementtype": "4",
                    "evaltype": "0",
                    "iconid_off": "1",
                    "iconid_on": "0",
                    "label": "Web server",
                    "label_location": "3",
```

```
                "x": "211",
                "y": "191",
                "iconid_disabled": "0",
                "iconid_maintenance": "0",
                "elementsubtype": "0",
                "areatype": "0",
                "width": "200",
                "height": "200",
                "viewtype": "0",
                "use_iconmap": "1",
                "tags": [],
                "urls": [],
                "elements": []
            },
            {
                "selementid": "12",
                "sysmapid": "3",
                "elementtype": "0",
                "evaltype": "0",
                "iconid_off": "185",
                "iconid_on": "0",
                "label": "{HOST.NAME}\r\n{HOST.CONN}",
                "label_location": "0",
                "x": "111",
                "y": "61",
                "iconid_disabled": "0",
                "iconid_maintenance": "0",
                "elementsubtype": "0",
                "areatype": "0",
                "width": "200",
                "height": "200",
                "viewtype": "0",
                "use_iconmap": "0",
                "tags": [],
                "urls": [],
                "elements": [
                    {
                        "hostid": "10084"
                    }
                ]
            }
        ],
        "links": [
            {
                "linkid": "23",
                "sysmapid": "3",
                "selementid1": "10",
                "selementid2": "11",
                "drawtype": "0",
                "color": "00CC00",
                "label": "",
                "linktriggers": []
            }
        ],
        "users": [
            {
                "sysmapuserid": "1",
                "userid": "2",
                "permission": "2"
            }
        ],
        "userGroups": [
```

```
            {
                "sysmapusrgrpid": "1",
                "usrgrpid": "7",
                "permission": "2"
            }
        ],
        "shapes":[
            {
                "sysmap_shapeid":"1",
                "type":"0",
                "x":"0",
                "y":"0",
                "width":"680",
                "height":"15",
                "text":"{MAP.NAME}",
                "font":"9",
                "font_size":"11",
                "font_color":"000000",
                "text_halign":"0",
                "text_valign":"0",
                "border_type":"0",
                "border_width":"0",
                "border_color":"000000",
                "background_color":"",
                "zindex":"0"
            }
        ],
        "lines":[
            {
                "sysmap_shapeid":"2",
                "x1": 30,
                "y1": 10,
                "x2": 100,
                "y2": 50,
                "line_type": 1,
                "line_width": 10,
                "line_color": "009900",
                "zindex":"1"
            }
        ],
        "sysmapid": "3",
        "name": "Local nerwork",
        "width": "400",
        "height": "400",
        "backgroundid": "0",
        "label_type": "2",
        "label_location": "3",
        "highlight": "1",
        "expandproblem": "1",
        "markelements": "0",
        "show_unack": "0",
        "grid_size": "50",
        "grid_show": "1",
        "grid_align": "1",
        "label_format": "0",
        "label_type_host": "2",
        "label_type_hostgroup": "2",
        "label_type_trigger": "2",
        "label_type_map": "2",
        "label_type_image": "2",
        "label_string_host": "",
        "label_string_hostgroup": "",
```

```
            "label_string_trigger": "",
            "label_string_map": "",
            "label_string_image": "",
            "iconmapid": "0",
            "expand_macros": "0",
            "severity_min": "0",
            "userid": "1",
            "private": "1",
            "show_suppressed": "1"
        }
    ],
    "id": 1
}
```


查看相关

- 拓扑图元素
- 拓扑图关联
- 拓扑图 URL
- 拓扑图用户
- 拓扑图用户组
- 拓扑图形状
- 拓扑图线

来源

CMap::get() in *ui/include/classes/api/services/CMap.php*.

服务
此类为使用 IT 设施/业务服务而设计。参考对象:

- 服务
- 状态规则
- 服务标签
- 服务告警
- 问题标记

可用办法:

- service.create - 创建新服务
- service.delete - 删除服务
- service.get - 检索服务
- service.update - 更新服务

> 服务对象

以下对象与 `service` API 直接关联。

服务

服务对象有如下属性。

属性	类型	描述
serviceid	string	*(只读)* 服务 ID。
algorithm (必需)	integer	状态计算规则。仅用于子服务存在时。 可能的值： 0 - 设置状态为正常； 1 - 最严重的，如果所有子服务存在问题； 2 - 子服务中最严重者。
name (required)	string	服务名称。

属性	类型	描述
sortorder (必需)	integer	用于排序的服务位置。 可能的值：0-999.
weight	integer	服务权重。 可能的值：0-1000000. Default: 0.
propagation_rule	integer	状态传播规则。必须与 `propagation_value` 一起设置。 可能的值： 0 - *(默认)* 原始传播服务状态； 1 - 按照给定的 `propagation_value` (按照 1 到 5 的严重性) 增加传播状态; 2 - 按照给定的 `propagation_value` (按照 1 到 5 的严重性) 降低传播状态； 3 - 忽略服务 - 状态根本不会传播给父服务； 4 - 按照给定 `propagation_value` 设定固定服务状态。
propagation_value	integer	状态传播值。必须和 `propagation_rule` 一起设置。 取值 0 和 3 的 `propagation_rule` 可能的值：0. 取值 1 和 2 的 `propagation_rule` 可能的值：1-5. 取值为 4 的 `propagation_rule` 可能的值： -1 - 正常； 0 - 未分类； 1 - 信息； 2 - 警告; 3 - 一般； 4 - 严重； 5 - 灾难。
status	integer	*(只读)* 服务是正常或问题状态。 如果服务处于问题状态，`status` 为以下其中一种: - 最紧急问题的严重性； - 处于问题状子服务的最高状态。 如果服务处于正常状态，`status` 等于-1。
description	string	服务描述。
uuid	string	通用唯一标识符。对更新操作，该字段是 只读。
readonly	boolean	*(只读)* 服务的访问权限。 可能的值： 0 - 读写； 1 - 只读。

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

状态规则

状态规则对象有如下属性。

属性	类型	描述
type (必需)	integer	设置 (New status) 状态的条件。 可能的值： 0 - 如果至少（N）个子服务处于（Status）或更高的状态； 1 - 如果至少（N%）的子服务处于（Status）或更高的状态； 2 - 如果不到（N）个子服务处于（Status）或更低状态； 3 - 如果不到（N%）的子服务处于（Status）或更低状态； 4 - 如果处于（Status）或更高状态的子服务的重量至少达到（W）； 5 - 如果处于（Status）或更高状态的子服务的重量至少达到（N%）； 6 - 如果处于（Status）或更低状态的子服务的重量低于（W）； 7 - 如果处于（Status）或更低状态的子服务的重量低于（N%）。 其中： - N (W) 是 `limit_value`； - (Status) 是 `limit_status`； - (New status) 是 `new_status`。
limit_value (必需)	integer	极限值。 可能的值： - 对 N 和 W：1-100000； - 对 N%：1-100。
limit_status (必需)	integer	极限状态。 可能的值： -1 - 正常； 0 - 未分类； 1 - 信息； 2 - 警告； 3 - 一般； 4 - 严重； 5 - 灾难。
new_status (必需)	integer	新状态值： 可能的值： 0 - 未分类； 1 - 信息； 2 - 警告； 3 - 一般； 4 - 严重； 5 - 灾难。

服务标记

服务标记对象有如下属性。

属性	类型	描述
tag (必须)	string	服务标签名称。
value	string	服务标签值。

服务告警

Note:
服务告警不能通过 Zabbix API 直接创建，更新或者删除。

服务告警对象代表服务状态的变更。它有如下值：

属性	类型	描述
clock	timestamp	服务状态发生变更的时间。
value	integer	服务状态。 可能值的清单请参考服务状态属性。

问题标签

问题标签允许将服务和故障事件关联起来。问题标签对象有如下属性。

属性	类型	描述
tag (必须)	string	问题标签名称。
operator	integer	条件运算符映射。 可能的值 0 - (默认) 等于； 2 - 类似。
value	string	问题标签值。

service.delete

描述

`object service.delete(array serviceIds)`

此方法允许删除服务。

> **Note:**
> 此方法允许任何用户使用。可以在用户角色设置中撤销调用此方法的权限。更多信息请查看用户角色。

参数

（数组）要删除的服务 ID。

返回值

（对象）返回一个 `serviceids` 属性包含被删除服务 ID 的对象。

示例

删除多个服务

删除两个服务。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "service.delete",
    "params": [
        "4",
        "5"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "serviceids": [
            "4",
            "5"
        ]
    },
```

```
    "id": 1
}
```

来源

CService::delete() 在 *ui/include/classes/api/services/CService.php*。

service.get

描述

`integer/array service.get(object parameters)`

此方法允许根据给定的参数检索服务。

> **Note:**
> 此方法允许任何用户使用。可以在用户角色设置中撤销调用此方法的权限。更多信息请查看用户角色。

参数

`(object)` 定义所需输出的参数。

该方法支持以下参数。

参数	类型	描述
serviceids	string/array	仅返回具有给定 ID 的服务。
parentids	string/array	仅返回链接到给定父服务的服务。
deep_parentids	flag	与 `parentids` 一起使用，返回所有直接和间接的子服务。
childids	string/array	仅返回链接到给定子服务的服务。
evaltype	integer	标签搜索的规则。 可能的值： 0 - *(默认)* 与/或； 2 - 或。
tags	object/array	仅返回具有给定标签的服务。根据运算符值，进行标签的精确匹配或大小写敏感或不敏感的搜索。 格式：`[{"tag": "<tag>", "value": "<value>", "operator": "<operator>"}, ...]`。 空数组返回所有服务。 可能的运算符值： 0 - *(默认)* 包含； 1 - 等于； 2 - 不包含； 3 - 不等于； 4 - 存在； 5 - 不存在。

参数	类型	描述
problem_tags	object/array	仅返回具有给定问题标签的服务。根据运算符值，进行标签的精确匹配或大小写敏感或不敏感的搜索。 格式：`[{"tag": "<tag>", "value": "<value>", "operator": "<operator>"}, ...]`。 空数组返回所有服务。 可能的运算符值： 0 - *(默认)* 包含； 1 - 等于； 2 - 不包含； 3 - 不等于； 4 - 存在； 5 - 不存在。
without_problem_tags	flag	仅返回没有问题标签的服务。
slaids	string/array	仅返回链接到特定 SLA 的服务。
selectChildren	query	返回带有子服务的 `children` 属性。 支持 `count`。
selectParents	query	返回带有父服务的 `parents` 属性。 支持 `count`。
selectTags	query	返回带有服务标签的tags 属性。 支持 `count`。
selectProblemEvents	query	返回带有问题事件对象数组的 `problem_events` 属性。 问题事件对象具有以下属性： `eventid` - *(string)* 事件 ID； `severity` - *(string)* 当前事件严重性； `name` - *(string)* 已解决事件名称。
selectProblemTags	query	返回带有问题标签的problem_tags 属性。 支持 `count`。
selectStatusRules	query	返回带有状态规则的status_rules 属性。 支持 `count`。
selectStatusTimeline	object/array	返回包含给定期间服务状态变化的 `status_timeline` 属性。 格式 `[{"period_from": "<period_from>", "period_to": "<period_to>"}, ...]` - `period_from` 是开始日期（包括；整数时间戳），`period_to` 是结束日期（不包括；整数时间戳），为您感兴趣的期间。 返回一个条目数组，每个条目包含一个 `start_value` 属性和一个alarms 数组，用于指定期间内的状态变化。 支持 `count`。

sortfield	string/array	根据给定属性对结果进行排序。 可能的值有：`serviceid`, `name`, `status`, `sortorder` 和 `created_at`。
countOutput	boolean	这些参数对所有 `get` 方法都是通用的，详细描述在参考说明 中。
editable	boolean	
excludeSearch	boolean	
filter	object	

limit	integer
output	query
preservekeys	boolean
search	object
searchByAny	boolean
searchWildcardsEnabled	boolean
sortorder	string/array
startSearch	boolean

返回值

（整型/数组）返回其中之一：

- 一个对象数组；
- 如果使用 `countOutput` 参数，被检索对象的数量。

示例

检索所有服务

检索所有服务和依赖的所有相关数据。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "service.get",
    "params": {
        "output": "extend",
        "selectChildren": "extend",
        "selectParents": "extend"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

相应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "serviceid": "1",
            "name": "My Service - 0001",
            "status": "-1",
            "algorithm": "2",
            "sortorder": "0",
            "weight": "0",
            "propagation_rule": "0",
            "propagation_value": "0",
            "description": "My Service Description 0001.",
            "uuid": "dfa4daeaea754e3a95c04d6029182681",
            "created_at": "946684800",
            "readonly": false,
            "parents": [],
            "children": []
        },
        {
            "serviceid": "2",
            "name": "My Service - 0002",
            "status": "-1",
            "algorithm": "2",
            "sortorder": "0",
            "weight": "0",
            "propagation_rule": "0",
            "propagation_value": "0",
```

```
            "description": "My Service Description 0002.",
            "uuid": "20ea0d85212841219130abeaca28c065",
            "created_at": "946684800",
            "readonly": false,
            "parents": [],
            "children": []
        }
    ],
    "id": 1
}
```

来源

CService::get() 在 *ui/include/classes/api/services/CService.php*。

service.update

描述

`object service.update(object/array services)`

此方法允许更新已有的服务。

> **Note:**
> 此方法允许任何用户使用。可以在用户角色设置中撤销调用此方法的权限。更多信息请查看用户角色。

参数

（对象/数组）要更新的服务属性。

每个服务必须定义 `serviceid` 属性，其他所有的属性都是可选。只有被传入的属性会被更新，其他将保持不变。

除了标准服务属性，此方法允许以下参数。

参数	类型	描述
children	array	取代当前子服务的子服务。 子服务必须定义 `serviceid` 属性。
parents	array	取代当前父服务的父服务。 父服务必须定义 `serviceid` 属性。
tags	array	取代当前服务标记的服务标签。
problem_tags	array	取代当前问题标记的问题标签。
status_rules	array	取代当前状态规则的状态规则。

返回值

（对象）返回一个 `serviceids` 属性包含了被更新服务 ID 的对象。

示例

设置服务的父服务

使 ID 为 “3” 的服务成为 ID 为 “5” 的服务的父服务。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "service.update",
    "params": {
        "serviceid": "5",
        "parents": [
            {
                "serviceid": "3"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
```

```
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "serviceids": [
            "5"
        ]
    },
    "id": 1
}
```

增加关机计划

给 ID 为 “4” 的服务增加每周一 22:00 到周二 10:00 的关机计划。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "service.update",
    "params": {
        "serviceid": "4",
        "times": [
            {
                "type": "1",
                "ts_from": "165600",
                "ts_to": "201600"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "serviceids": [
            "4"
        ]
    },
    "id": 1
}
```

来源

CService::update() 在 *ui/include/classes/api/services/CService.php*。

创建服务

描述

`object service.create(object/array services)`

此方法允许创建新服务。

> **Note:**
> 此方法允许任何用户使用。可以在用户角色设置中撤销调用此方法的权限。更多信息请查看用户角色。

参数

（对象/数组）要创建的服务。

除了标准服务属性，此方法接受如下参数。

参数	类型	描述
children	array	与服务相关联的子服务 子服务必须有已定义的 `serviceid` 属性。
parents	array	与服务相关联的父服务 父服务必须有已定义的 `serviceid` 属性。
tags	array	服务待创建的服务标记。
problem_tags	array	服务待创建的问题标记。
status_rules	array	服务待创建的状态规则。

返回值

（对象）返回一个 `serviceids` 属性包含被创建服务 ID 的对象。返回的 ID 顺序与传入服务的顺序一致。

示例

创建服务

创建一个如果其至少一个子服务有问题，则会被转化为问题状态的服务。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "service.create",
    "params": {
        "name": "Server 1",
        "algorithm": 1,
        "sortorder": 1
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "serviceids": [
            "5"
        ]
    },
    "id": 1
}
```

来源

CService::create() 在 *frontends/php/include/classes/api/services/CService.php*。

模版

这个类是为使用模板而设计的。

对象引用:

- 模版

可用方法:

- template.create - 创建一个新模版
- template.delete - 删除模版
- template.get - 检索模版
- template.massadd - 向模板添加相关对象
- template.massremove - 从模板中删除相关对象
- template.massupdate - 从模板中替换或删除相关对象
- template.update - 更新模版

> 模版对象

以下对象都是与 `template` 直接相关的 API。

模板

模板对象具有以下属性。

属性	类型	描述
templateid	string	*(只读)* 模板的 ID。
host (必选)	string	模板的技术名称。
description	text	模板的描述。
name	string	模板的可见名称。 默认值：主机属性值。
uuid	string	通用唯一标识符，用于将导入的模板链接到已存在的模板。如果未给出，则自动生成。

请注意，对于一些方法（更新，删除），必选/可选参数组合是不同的。

模版标签

模版标签对象具有以下属性。

属性	类型	描述
tag (必须)	string	模版标签名字。
value	string	模版标签值。

template.create

描述

`object template.create(object/array templates)`

此方法允许创建新模板。

Note:
此方法只有 *Admin(管理员)* 和 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

`(object/array)` 创建模版。

除了标准模板属性之外，该方法还接受以下参数。

参数	类型	描述
groups (必须)	object/array	将模版添加到主机群组。 主机群组必须定义 `groupid` 属性。
tags	object/array	模版标签.
templates	object/array	模版 要链接到模版。 模板必须定义 `templateid` 属性。
macros	object/array	要为模版创建的用户宏 。

返回值

`(object)` 返回一个对象，该对象包含 `templateids` 属性下创建的模板的 ID。返回 ID 的顺序与传递模板的顺序匹配。

示例

创建一个模版

创建一个带有标记的模板，并将两个模板链接到此模板。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "template.create",
    "params": {
        "host": "Linux template",
        "groups": {
            "groupid": 1
        },
        "templates": [
            {
                "templateid": "11115"
            },
            {
                "templateid": "11116"
            }
        ],
        "tags": [
            {
                "tag": "Host name",
                "value": "{HOST.NAME}"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "templateids": [
            "11117"
        ]
    },
    "id": 1
}
```

来源

CTemplate::create() in *ui/include/classes/api/services/CTemplate.php*.

template.delete

描述

`object template.delete(array templateIds)`

此方法允许删除模板。

删除模板将导致删除所有模板实体（监控项、触发器、图形等）。要将模板实体留在主机上，但删除模板本身，请首先使用以下方法之一取消模板与所需主机的链接：template.update, template.massupdate, host.update, host.massupdate.

> **Note:**
> 此方法只有 *Admin(管理员)* 和 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(array) 要删除的模板的 ID。

返回值

(object) 返回一个对象，该对象包含 templateids 属性下已删除模板的 ID。

示例

删除多个模版

删除两个模版

请求：

```
{
    "jsonrpc": "2.0",
    "method": "template.delete",
    "params": [
        "13",
        "32"
    ],
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "templateids": [
            "13",
            "32"
        ]
    },
    "id": 1
}
```

来源

CTemplate::delete() in *ui/include/classes/api/services/CTemplate.php*.

批量删除模板

描述

`object template.massremove(object parameters)`

此方法允许从多个模板中删除相关对象。

> **Note:**
> 此方法仅适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。详情请阅User roles。

参数

`(object)` 包含要更新的模板 ID 和应删除的对象的参数。

参数	类型	描述
templateids (必须)	string/array	要更新的模板的 ID。
groupids	string/array	从主机组中删除给定模板。
macros	string/array	从给定模板中删除的用户宏。
templateids_clear	string/array	从指定模板（上游）中取消模板链接并清除数据。
templateids_link	string/array	从指定模板（上游）中取消模板链接。

返回值

`(object)` 返回一个对象，该对象包含 `templateids` 属性下已更新模板的 ID。

示例

从一个群组中删除模版

从群组 “2” 中删除两个模版。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "template.massremove",
    "params": {
        "templateids": [
            "10085",
            "10086"
        ],
        "groupids": "2"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "templateids": [
            "10085",
            "10086"
        ]
    },
    "id": 1
}
```

在一个主机中取消链接模版

在模版 “10085” 中取消链接模版 “10106” 和 “10104”。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "template.massremove",
    "params": {
        "templateids": "10085",
        "templateids_link": [
            "10106",
            "10104"
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "templateids": [
            "10085"
        ]
    },
    "id": 1
}
```

参阅

- 更新模板
- 用户宏

源码

CTemplate::massRemove() in *ui/include/classes/api/services/CTemplate.php*.

批量更新模板

描述

`object template.massupdate(object parameters)`

此方法允许同时替换或删除相关对象，并更新多个模板上的属性。

> **Note:**
> 此方法仅适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。详情请阅User roles。

参数

(object) 参数包含要更新的模板 ID 和要替换的模板对象。

该方法接受以下参数.

参数	类型	描述
templates (必须)	object/array	要更新的模版。 模版必须有已定义的 templateid 属性。
groups	object/array	用于替换模板所属的当前主机群组的主机群组。 主机群组必须有已定义的 groupid 属性。
macros	object/array	用户宏替换给定模板上的当前用户宏。
templates_clear	object/array	取消链接并清除给定模板的模板。 模版必须有已定义的 templateid 属性。
templates_link	object/array	替换当前链接的模板的模板。 模版必须有已定义的 templateid 属性。

返回值

(object) 返回一个对象，该对象包含 templateids 属性下已更新模板的 ID。

示例

替换主机群组

从给定模板中取消链接并清除模板 “10091”。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "template.massupdate",
    "params": {
        "templates": [
            {
                "templateid": "10085"
            },
            {
                "templateid": "10086"
            }
        ],
        "templates_clear": [
            {
                "templateid": "10091"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
```

```
        "templateids": [
            "10085",
            "10086"
        ]
    },
    "id": 1
}
```

参阅

- 更新模板
- 批量添加模板
- 主机组
- 用户宏

源码

CTemplate::massUpdate() in *ui/include/classes/api/services/CTemplate.php*.

批量添加模板

描述

`object template.massadd(object parameters)`

此方法允许同时向给定模板添加多个相关对象。

Note:
此方法仅适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。详情请阅User roles。

参数

(object) 参数包含要更新的模板 ID 和要添加到模板的对象的参数。

该方法接受以下参数。

参数	类型	描述
templates (必须)	object/array	要更新的模板。 模板必须定义 templateid 属性。
groups	object/array	要将给定模板添加到的主机群组。 主机组必须定义 groupid 属性。
macros	object/array	为给定模板创建的用户宏。
templates_link	object/array	链接到给定模板的模板。 模板必须定义 templateid 属性。

返回值

(object) 返回一个对象，该对象包含 templateids 属性下已更新模板的 ID。

示例

将一个群组链接到模板

将主机群组 “2” 添加到两个模板中。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "template.massadd",
    "params": {
        "templates": [
            {
                "templateid": "10085"
            },
            {
                "templateid": "10086"
```

```
            }
        ],
        "groups": [
            {
                "groupid": "2"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "templateids": [
            "10085",
            "10086"
        ]
    },
    "id": 1
}
```

将两个模版链接到一个模板

将模板 “10106” 和 “10104” 链接到模板。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "template.massadd",
    "params": {
        "templates": [
            {
                "templateid": "10073"
            }
        ],
        "templates_link": [
            {
                "templateid": "10106"
            },
            {
                "templateid": "10104"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "templateids": [
            "10073"
        ]
    },
    "id": 1
}
```

参阅

- 更新
- 主机
- 主机群组
- 用户宏

源码

CTemplate::massAdd() in *ui/include/classes/api/services/CTemplate.php*.

更新模板

描述

`object template.update(object/array templates)`

此方法允许更新现有模板。

> **Note:**
> 此方法仅适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。详情请阅User roles。

参数

`(object/array)` 要更新的模板属性。

必须为每个模板定义 `templateid` 属性，其他所有属性都是可选的。只有给定的属性将被更新，所有其他属性将保持不变。

除了标准模版属性，该方法接受以下参数。

参数	类型	描述
groups	object/array	用于替换模板所属的当前主机组的主机群组。 主机组必须定义 `groupid` 属性。
tags	object/array	替换当前模板标记的模板标签。
macros	object/array	用户宏替换给定模板上的当前用户宏。
templates	object/array	替换当前链接的模板的模版。未传递的模板仅被取消链接。 模板必须定义 `templateid` 属性。
templates_clear	object/array	取消链接并清除给定模板的模版。 模板必须定义 `templateid` 属性。

返回值

`(object)` 返回一个对象，该对象包含 `templateids` 属性下已更新模板的 ID。

示例

Changing the standard template properties

Change the technical name of the template to ”Linux by Zabbix agent Custom”, the visible name to ”My template”, and update the template description.

Request:

```
{
    "jsonrpc": "2.0",
    "method": "template.update",
    "params": {
        "templateid": "10086",
        "host": "Linux by Zabbix agent Custom",
        "name": "My template",
        "description": "This is a custom Linux template."
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "result": {
        "templateids": [
            "10086"
        ]
    },
    "id": 1
}
```

Updating template groups

Replace all template groups for the given template with a different one.

Request:

```
{
    "jsonrpc": "2.0",
    "method": "template.update",
    "params": {
        "templateid": "10086",
        "groups": [
            {
                "groupid": "24"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "result": {
        "templateids": [
            "10086"
        ]
    },
    "id": 1
}
```

Updating template tags

Replace all template tags with a different one.

Request:

```
{
    "jsonrpc": "2.0",
    "method": "template.update",
    "params": {
```

```
        "templateid": "10086",
        "tags": [
            {
                "tag": "Host name",
                "value": "{HOST.NAME}"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "result": {
        "templateids": [
            "10086"
        ]
    },
    "id": 1
}
```

Updating template macros

Replace all template macros with a different one.

Request:

```
{
    "jsonrpc": "2.0",
    "method": "template.update",
    "params": {
        "templateid": "10086",
        "macros": [
            {
                "macro": "{$MY_MACRO}",
                "value": "new_value"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "result": {
        "templateids": [
            "10086"
        ]
    },
    "id": 1
}
```

Updating template linked templates

Unlink (without clearing) all templates from the given template and link a different one to it.

Request:

```
{
    "jsonrpc": "2.0",
    "method": "template.update",
    "params": {
```

```
        "templateid": "10086",
        "templates": [
            {
                "templateid": "10087"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "result": {
        "templateids": [
            "10086"
        ]
    },
    "id": 1
}
```

Clearing template linked templates

Unlink and clear the given template from a specific linked template.

Request:

```
{
    "jsonrpc": "2.0",
    "method": "template.update",
    "params": {
        "templateid": "10086",
        "templates_clear": [
            {
                "templateid": "10087"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "result": {
        "templateids": [
            "10086"
        ]
    },
    "id": 1
}
```

源码

CTemplate::update() in *ui/include/classes/api/services/CTemplate.php*.

检索模板

描述

`integer/array template.get(object parameters)`

该方法允许根据给定的参数检索模板。

Note:
任何类型的用户都可以使用此方法。可以在用户角色设置中撤销调用该方法的权限。详情请阅User roles。

参数

(object) 定义所需输出的参数。

该方法支持以下参数。

参数	类型	描述
templateids	string/array	仅返回给定模板 ID 的模板。
groupids	string/array	仅返回属于给定主机组的模板。
parentTemplateids	string/array	仅返回是给定模板父级的模板。
hostids	string/array	仅返回链接到给定主机/模板的模板。
graphids	string/array	仅返回包含给定图表的模板。
itemids	string/array	仅返回包含给定监控项的模板。
triggerids	string/array	仅返回包含给定触发器的模板。
with_items	flag	仅返回包含监控项的模板。
with_triggers	flag	仅返回包含触发器的模板。
with_graphs	flag	仅返回包含图表的模板。
with_httptests	flag	仅返回包含 web 场景的模板。
evaltype	integer	标签搜索的规则。 可能的值： 0 - (默认) 与/或； 2 - 或。
tags	object/array	仅返回具有给定标签的模板。根据运算符的值，进行标签的精确匹配或大小写敏感或不敏感的搜索。 格式：[{"tag": "<tag>", "value": "<value>", "operator": "<operator>"}, ...]。 空数组返回所有模板。 可能的运算符值： 0 - (默认) 包含； 1 - 等于； 2 - 不像； 3 - 不等于 4 - 存在； 5 - 不存在。
selectGroups	query	在groups属性中返回模板所属的主机组。
selectTags	query	在tags属性中返回模板标签。
selectHosts	query	在hosts属性中返回链接到模板的主机。 支持 count。
selectTemplates	query	在templates属性中返回模板是子级的模板。 支持 count。
selectParentTemplates	query	在 parentTemplates 属性中返回模板是父级的模板。 支持 count。
selectHttpTests	query	在httpTests属性中返回模板中的 web 场景。 支持 count。
selectItems	query	在items属性中返回模板中的监控项。 支持 count。
selectDiscoveries	query	在 discoveries 属性中返回模板中的低级发现。 支持 count。
selectTriggers	query	在triggers属性中返回模板中的触发器。 支持 count。

参数	类型	描述
selectGraphs	query	在graphs属性中返回模板中的图表。 支持 `count`。
selectMacros	query	在 `macros` 属性中返回模板的宏。
selectDashboards	query	在dashboards属性中返回模板的仪表板。 支持 `count`。
selectValueMaps	query	返回带有模板值映射的valuemaps属性。
limitSelects	integer	限制子选择返回的记录数。 适用于以下子选择： `selectTemplates` - 结果将按 `name` 排序； `selectHosts` - 按 `host` 排序； `selectParentTemplates` - 按 `host` 排序； `selectItems` - 按 `name` 排序； `selectDiscoveries` - 按 `name` 排序； `selectTriggers` - 按 `description` 排序； `selectGraphs` - 按 `name` 排序； `selectDashboards` - 按 `name` 排序。
sortfield	string/array	根据给定属性对结果进行排序。 可能的值有：`hostid`, `host`, `name`, `status`。
countOutput	boolean	这些参数对所有 `get` 方法都是通用的，详细描述在参考说明 中。
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

`(integer/array)` 返回两者其中任一：

- 一组对象数组;
- 如果已经使用了 `countOutput` 参数，则检索对象的计数。

示例

按名称检索模版

检索有关名为 “Linux” 和 “Windows” 的两个模板的所有数据。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "template.get",
    "params": {
        "output": "extend",
        "filter": {
            "host": [
                "Linux",
                "Windows"
            ]
        }
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
```

```
    "id": 1
}
```

响应:


```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "proxy_hostid": "0",
            "host": "Linux",
            "status": "3",
            "disable_until": "0",
            "error": "",
            "available": "0",
            "errors_from": "0",
            "lastaccess": "0",
            "ipmi_authtype": "0",
            "ipmi_privilege": "2",
            "ipmi_username": "",
            "ipmi_password": "",
            "ipmi_disable_until": "0",
            "ipmi_available": "0",
            "snmp_disable_until": "0",
            "snmp_available": "0",
            "maintenanceid": "0",
            "maintenance_status": "0",
            "maintenance_type": "0",
            "maintenance_from": "0",
            "ipmi_errors_from": "0",
            "snmp_errors_from": "0",
            "ipmi_error": "",
            "snmp_error": "",
            "jmx_disable_until": "0",
            "jmx_available": "0",
            "jmx_errors_from": "0",
            "jmx_error": "",
            "name": "Linux",
            "flags": "0",
            "templateid": "10001",
            "description": "",
            "tls_connect": "1",
            "tls_accept": "1",
            "tls_issuer": "",
            "tls_subject": "",
            "tls_psk_identity": "",
            "tls_psk": "",
            "uuid": "282ffe33afc74cccaf1524d9aa9dc502"
        },
        {
            "proxy_hostid": "0",
            "host": "Windows",
            "status": "3",
            "disable_until": "0",
            "error": "",
            "available": "0",
            "errors_from": "0",
            "lastaccess": "0",
            "ipmi_authtype": "0",
            "ipmi_privilege": "2",
            "ipmi_username": "",
            "ipmi_password": "",
            "ipmi_disable_until": "0",
```

```
            "ipmi_available": "0",
            "snmp_disable_until": "0",
            "snmp_available": "0",
            "maintenanceid": "0",
            "maintenance_status": "0",
            "maintenance_type": "0",
            "maintenance_from": "0",
            "ipmi_errors_from": "0",
            "snmp_errors_from": "0",
            "ipmi_error": "",
            "snmp_error": "",
            "jmx_disable_until": "0",
            "jmx_available": "0",
            "jmx_errors_from": "0",
            "jmx_error": "",
            "name": "Windows",
            "flags": "0",
            "templateid": "10081",
            "description": "",
            "tls_connect": "1",
            "tls_accept": "1",
            "tls_issuer": "",
            "tls_subject": "",
            "tls_psk_identity": "",
            "tls_psk": "",
            "uuid": "522d17e1834049be879287b7c0518e5d"
        }
    ],
    "id": 1
}
```

检索通过模板关联的主机

检索已链接到“10001”（*Zabbix agent* 的 *Linux*）模板的主机。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "template.get",
    "params": {
        "output": "templateid",
        "templateids": "10001",
        "selectHosts": ["hostid", "name"]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "templateid": "10001",
            "hosts": [
                {
                    "hostid": "10084",
                    "name": "Zabbix server"
                },
                {
                    "hostid": "10603",
                    "name": "Host 1"
                },
```

```
                {
                    "hostid": "10604",
                    "name": "Host 2"
                }
            ]
        }
    ],
    "id": 1
}
```

按模板标签搜索

检索标签 “主机名称” 等于 “{HOST.NAME}” 的模板。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "template.get",
    "params": {
        "output": ["hostid"],
        "selectTags": "extend",
        "evaltype": 0,
        "tags": [
            {
                "tag": "Host name",
                "value": "{HOST.NAME}",
                "operator": 1
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "hostid": "10402",
            "tags": [
                {
                    "tag": "Host name",
                    "value": "{HOST.NAME}"
                }
            ]
        }
    ],
    "id": 1
}
```

参阅

- 主机组
- 模板
- 用户宏
- 主机接口

源码

CTemplate::get() in *ui/include/classes/api/services/CTemplate.php*.

模版仪表盘

此类设计用于使用模板仪表盘。

对象引用:

- 模版仪表盘
- 模版仪表盘页面
- 模版仪表盘组件
- 模版仪表盘组件字段

可用方法:

- templatedashboard.create - 创建一个新的模版仪表盘
- templatedashboard.delete - 删除模版仪表盘
- templatedashboard.get - 检索模版仪表盘
- templatedashboard.update - 更新模版仪表盘

> 模版仪表盘对象

以下对象与 `templatedashboard` API 直接相关。

模板仪表盘

模板仪表盘对象具有以下属性。

属性	类型	描述
dashboardid	string	*(只读)* 模板仪表盘的 ID。
name (必选)	string	模板仪表盘的名称。
templateid (必选)	string	仪表盘所属模板的 ID。
display_period	integer	默认页面显示周期（秒）。 可能的值：10, 30, 60, 120, 600, 1800, 3600。 默认值：30。
auto_start	integer	自动开始幻灯片播放。 可能的值： 0 - 不自动开始幻灯片播放； 1 - *(默认)* 自动开始幻灯片播放。
uuid	string	通用唯一标识符，用于将导入的模板仪表盘链接到已存在的仪表盘。如果未给出，则自动生成。

请注意，对于一些方法（更新，删除），必选/可选参数组合是不同的。

模版仪表盘页面

模板仪表板页面对象具有以下属性。

属性	类型	描述
dashboard_pageid	string	*(只读)* 仪表盘页面的 ID。
name	string	仪表盘页面的名字。 默认值：空的字符串。

属性	类型	描述
display_period	integer	仪表盘页面显示时间（秒）。 可能的值：0, 10, 30, 60, 120, 600, 1800, 3600. 默认值：0 (将一直使用默认的页面显示)。
widgets	array	模版仪表盘组件对象的数组。

模版仪表盘组件

模板仪表板组件对象具有以下属性。

属性	类型	描述
widgetid	string	*(只读)* 仪表盘组件的 ID。
type (必须)	string	仪表盘组件的类型 可能的值： clock - 时钟； graph - 图形（经典）； graphprototype - 图形原型； item - 监控项的值； plaintext - 纯文本； url - URL;
name	string	自定义组件名称。
x	integer	仪表板左侧的水平位置。 有效值的范围为 0 到 23。
y	integer	仪表板顶部的垂直位置。 有效值的范围为 0 到 62。
width	integer	组件的宽度。 有效值的范围为 1 到 24.
height	integer	组件的高度。 有效值的范围为 2 到 32。
view_mode	integer	组件视图模式。 可能的值： 0 - (默认值) 默认组件视图。 1 - 隐藏标题；
fields	array	模版仪表盘组件字段对象的数组。

模板仪表盘控件字段

模板仪表盘控件字段对象具有以下属性。

属性	类型	描述
type (必选)	integer	控件字段的类型。 可能的值： 0 - 整数； 1 - 字符串； 4 - 监控项； 5 - 监控项原型； 6 - 图表； 7 - 图表原型。
name	string	控件字段名称。
value (必选)	mixed	根据类型的控件字段值。

创建模板仪表盘

描述

`object templatedashboard.create(object/array templateDashboards)`

此方法允许创建新的模板仪表板。

> **Note:**
> 此方法仅适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。详情查阅用户角色。

参数

`(object/array)` 要创建的模板仪表盘。

除了标准模板仪表盘属性外，该方法还接受以下参数。

参数	类型	描述
pages (必须)	array	要为仪表盘创建的模板仪表盘页面。仪表盘页面按指定的顺序排序。pages 属性至少需要一个仪表盘页面对象。

返回值

`(object)` 返回一个对象，该对象包含 `dashboardids` 属性下创建的模板仪表盘的 ID。返回 ID 的顺序与传递的模板仪表盘的顺序匹配。

示例

创建一个模版仪表盘

在单个仪表盘页面上使用一个 Graph 组件创建一个名为 “Graphs” 的模板仪表盘。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "templatedashboard.create",
    "params": {
        "templateid": "10318",
        "name": "Graphs",
        "pages": [
            {
                "widgets": [
                    {
                        "type": "graph",
                        "x": 0,
                        "y": 0,
```

```
                        "width": 12,
                        "height": 5,
                        "view_mode": 0,
                        "fields": [
                            {
                                "type": 6,
                                "name": "graphid",
                                "value": "1123"
                            }
                        ]
                    }
                ]

            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "32"
        ]
    },
    "id": 1
}
```

参阅

- 模版仪表盘页面
- 模版仪表盘组件
- 模版仪表盘组件字段

源码

CTemplateDashboard::create() in *ui/include/classes/api/services/CTemplateDashboard.php*.

删除模板仪表盘

描述

`object templatedashboard.delete(array templateDashboardIds)`

此方法允许删除模板仪表盘。

> **Note:**
> 此方法仅适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。详情查阅User roles。

参数

(array) 需要删除的模板仪表盘 ID。

返回值

(object) 返回一个对象，该对象包含 dashboardids 属性下已删除模板仪表盘的 ID。

示例

删除多个模版仪表盘

删除两个模版仪表盘。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "templatedashboard.delete",
    "params": [
        "45",
        "46"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "45",
            "46"
        ]
    },
    "id": 1
}
```

源码

CTemplateDashboard::delete() in *ui/include/classes/api/services/CTemplateDashboard.php*.

更新模板仪表盘

描述

`object templatedashboard.update(object/array templateDashboards)`

此方法允许更新现有模板仪表盘。

> **Note:**
> 此方法仅适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。详情查阅User roles。

参数

(object/array) 要更新的模板仪表盘属性。

必须为每个仪表盘指定 dashboardid 属性，其他所有属性都是可选的。只会更新指定的属性。

除了标准模版仪表盘属性外，该方法还接受以下参数。

参数	类型	描述
pages	array	模板仪表盘页面以替换现有仪表盘页面。 仪表盘页面按 dashboard_pageid 属性更新。将为没有 dashboard_pageid 属性的对象创建新的仪表盘页面，如果不重用，现有的仪表盘页面将被删除。仪表盘页面将按指定的顺序排序。只会更新仪表盘页面的指定属性。pages 属性至少需要一个仪表盘页面对象。

返回值

(object) 返回一个对象，该对象包含 dashboardids 属性下更新的模板仪表盘的 ID。

示例

重命名一个模板仪表盘

将一个模板仪表盘重命名为"Performance graphs"。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "templatedashboard.update",
    "params": {
        "dashboardid": "23",
        "name": "Performance graphs"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
            "23"
        ]
    },
    "id": 1
}
```

更新模板仪表盘页面

重命名第一个仪表盘页面，替换第二个仪表盘页面上的组件，并添加一个新页面作为第三个页面。删除所有其他仪表盘页面。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "templatedashboard.update",
    "params": {
        "dashboardid": "2",
        "pages": [
            {
                "dashboard_pageid": 1,
                "name": 'Renamed Page'
            },
            {
                "dashboard_pageid": 2,
                "widgets": [
                    {
                        "type": "clock",
                        "x": 0,
                        "y": 0,
                        "width": 4,
                        "height": 3
                    }
                ]
            },
            {
                "display_period": 60
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "dashboardids": [
```

```
            "2"
        ]
    },
    "id": 2
}
```

参阅

- 模版仪表盘组件
- 模版仪表盘组件字段

源码

CTemplateDashboard::update() in *ui/include/classes/api/services/CTemplateDashboard.php*.

检索模板仪表盘

描述

`integer/array templatedashboard.get(object parameters)`

该方法允许根据给定的参数检索模板仪表盘。

Note:
此方法对于任何用户可用。可以在用户角色设置中撤销调用该方法的权限。详情查阅User roles。

参数

`(object)` 定义所需输出的参数。

该方法支持以下参数。

参数	类型	描述
dashboardids	string/array	仅返回具有给定 ID 的模板仪表盘。
templateids	string/arary	仅返回属于给定模板的模板仪表盘。
selectPages	query	按正确顺序返回带有模板仪表盘页面的pages属性。
sortfield	string/array	按给定属性对结果进行排序。 可能的值是：`dashboardid` and `name`.
countOutput	boolean	这些参数对于所有 `get` 方法都是通用的，在参考注释中有详细描述。
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

`(integer/array)` 返回两者其一：

- 一个对象数组；
- 如果使用了'countOutput' 参数，则对检索对象进行计数。

示例

检索模板仪表盘

检索指定模板的所有带有组件的模板仪表盘。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "templatedashboard.get",
    "params": {
        "output": "extend",
        "selectPages": "extend",
        "templateids": "10001"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "dashboardid": "23",
            "name": "Docker overview",
            "templateid": "10001",
            "display_period": "30",
            "auto_start": "1",
            "uuid": "6dfcbe0bc5ad400ea9c1c2dd7649282f",
            "pages": [
                {
                    "dashboard_pageid": "1",
                    "name": "",
                    "display_period": "0",
                    "widgets": [
                        {
                            "widgetid": "220",
                            "type": "graph",
                            "name": "",
                            "x": "0",
                            "y": "0",
                            "width": "12",
                            "height": "5",
                            "view_mode": "0",
                            "fields": [
                                {
                                    "type": "6",
                                    "name": "graphid",
                                    "value": "1125"
                                }
                            ]
                        },
                        {
                            "widgetid": "221",
                            "type": "graph",
                            "name": "",
                            "x": "12",
                            "y": "0",
                            "width": "12",
                            "height": "5",
                            "view_mode": "0",
                            "fields": [
                                {
                                    "type": "6",
                                    "name": "graphid",
                                    "value": "1129"
                                }
                            ]
```

```
                        },
                        {
                            "widgetid": "222",
                            "type": "graph",
                            "name": "",
                            "x": "0",
                            "y": "5",
                            "width": "12",
                            "height": "5",
                            "view_mode": "0",
                            "fields": [
                                {
                                    "type": "6",
                                    "name": "graphid",
                                    "value": "1128"
                                }
                            ]
                        },
                        {
                            "widgetid": "223",
                            "type": "graph",
                            "name": "",
                            "x": "12",
                            "y": "5",
                            "width": "12",
                            "height": "5",
                            "view_mode": "0",
                            "fields": [
                                {
                                    "type": "6",
                                    "name": "graphid",
                                    "value": "1126"
                                }
                            ]
                        },
                        {
                            "widgetid": "224",
                            "type": "graph",
                            "name": "",
                            "x": "0",
                            "y": "10",
                            "width": "12",
                            "height": "5",
                            "view_mode": "0",
                            "fields": [
                                {
                                    "type": "6",
                                    "name": "graphid",
                                    "value": "1127"
                                }
                            ]
                        }
                    ]
                }
            ]
        }
    ],
    "id": 1
}
```


参阅

- 模版仪表盘页面

- 模版仪表盘组件
- 模版仪表盘组件字段

源码

CTemplateDashboard::get() in *ui/include/classes/api/services/CTemplateDashboard.php*.

正则表达式

本节内容用于使用户更全面的了解如何使用有关全局类正则表达式的 API 功能。

用户可直接参考：

- 正则表达式

可应用的功能方式包括：

- regexp.create 创建新的正则表达式
- regexp.delete - 删除目标正则表达式
- regexp.get - 检索正则表达式数据
- regexp.update - 更新正则表达式数据

> 目标：正则表达式

以下所列内容均与 `regexp`（正则表达式） API 直接关联。

正则表达式

全局正则表达式对象都具有下列属性。

属性	类型	说明
regexpid	字符串	（只读）正则表达式 ID。
name (必需)	字符串	正则表达式名称。
test_string	字符串	测试字符串。

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

表达式

表达式对象具有以下属性。

属性	类型	描述
expression (必需)	string	正则表达式。
expression_type (必需)	integer	正则表达式的类型。 可能的值： 0 - 包含字符字符串； 1 - 包含任意字符字符串； 2 - 不包含字符字符串； 3 - 结果为 TRUE； 4 - 结果为 FALSE。
exp_delimiter	string	表达式分隔符。仅当 表达式类型为 包含任意字符字符串时。 默认值 `,`。 可能的值：`,`, `.`, `/`。

属性	类型	描述
case_sensitive	integer	大小写敏感度。 默认值 0。 可能的值： 0 - 不区分大小写； 1 - 区分大小写。

正则表达式. 创建

说明

`object regexp.create(object/array regularExpressions)`

该方式允许用户创建一个新的全局正则表达式。

Note:
该方式仅对超级管理员用户类型生效。用户可以在用户角色设置中对该方式的使用权限进行设定修改。请参考用户角色以获取更多信息。

参数

(object/array) 用于创建的正则表达式。

除了标准属性，该方法还接受以下参数。

参数	类型	描述
expressions	array	表达式选项。

返回值

根据 regexpids 的特性 (object) 会反馈一个包含新创建的正则表达式 ID 的对象

参考示例

创建一个新的全局正则表达式

请求：

```
{
    "jsonrpc": "2.0",
    "method": "regexp.create",
    "params": {
      "name": "Storage devices for SNMP discovery",
      "test_string": "/boot",
      "expressions": [
        {
          "expression": "^(Physical memory|Virtual memory|Memory buffers|Cached memory|Swap space)$",
          "expression_type": "4",
          "case_sensitive": "1"
        }
      ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "regexpids": [
            "16"
```

```
        ]
    },
    "id": 1
}
```

参考来源

CRegexp::create() in *ui/include/classes/api/services/CRegexp.php*.

正则表达式. 删除

说明

`object regexp.delete(array regexpids)`

运行该方法用来删除全局性的正则表达式。

> **Note:**
> 该功能仅对超级管理员用户类型开放。用户可以在用户角色设置中对该方式的使用权限进行设定修改。请参考用户角色以获取更多信息。

参数

(array) 代表计划删除的正则表达式的 ID 号。

返回数值

根据 regexpids 的特性 (object) 会反馈一个包含已删除正则表达式的 ID 对象。

参考示例

删除多个全局正则表达式。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "regexp.delete",
    "params": [
        "16",
        "17"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "regexpids": [
            "16",
            "17"
        ]
    },
    "id": 1
}
```

参考来源

CRegexp::delete() in *ui/include/classes/api/services/CRegexp.php*.

正则表达式. 更新

说明

`object regexp.update(object/array regularExpressions)`

该方式允许用户升级已创建的全局性正则表达式。

> **Note:**
> 该方式仅对超级管理员类型的用户生效。用户可以在用户职责设置中对该方式的使用权限进行设定修改。请参考用户角色以获取更多信息。

参数

(object/array) 将更新的正则表达式属性。

regexpid 为必要配置参数，需要为每一个正则表达式配置，其它属性为可选配置参数。只有符合要求的属性才会被直接更新，若不符合则原属性将保持不变。

除此之外，根据标准属性，该方式支持以下参数。

参数名称	类型	说明
expressions	array	表达式 选项。

返回值

根据 regexpids 的特性，(object) 会返回一个对象，其包含已升级的正则表达式 ID。

参考示例

更新自动发现文件系统的全局正则表达式。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "regexp.update",
    "params": {
      "regexpid": "1",
      "name": "File systems for discovery",
      "test_string": "",
      "expressions": [
        {
          "expression": "^(btrfs|ext2|ext3|ext4|reiser|xfs|ffs|ufs|jfs|jfs2|vxfs|hfs|apfs|refs|zfs)$",
          "expression_type": "3",
          "exp_delimiter": ",",
          "case_sensitive": "0"
        },
        {
          "expression": "^(ntfs|fat32|fat16)$",
          "expression_type": "3",
          "exp_delimiter": ",",
          "case_sensitive": "0"
        }
      ]
    },
    "auth": "700ca65537074ec963db7efabda78259",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "regexpids": [
            "1"
        ]
    },
    "id": 1
}
```

参考来源

CRegexp::update() in *ui/include/classes/api/services/CRegexp.php*.

正则表达式. 获取

说明

`integer/array regexp.get(object parameters)`

使用该方式允许用户根据提供的参数检索选定的全局正则表达式。

> **Note:**
> 该方式仅对超级管理员类型的用户生效。用户可以在用户角色设置中对该方式的使用权限进行设定修改。请参考用户角色以获取更多信息。

参数

(object) 定义所需输出的参数。

该方法支持以下参数。

参数	类型	描述
regexpids	string/array	返回只有给定 ID 的正则表达式。
selectExpressions	query	返回一个表达式属性。
sortfield	string/array	根据给定的属性对结果进行排序。 可能的值有：regexpid 和 name。
countOutput	boolean	这些参数对于所有 获取方法都是通用的，详细描述在参考说明中。
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回数值

(integer/array) 将返回以下两种数值中的任意一种：

- 一组有关对象的数组；
- 若在使用 countOutput 参数的前提下，那么将会返回检索对象的总数。

参考示例

检索全局正则表达式。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "regexp.get",
    "params": {
        "output": ["regexpid", "name"],
        "selectExpressions": ["expression", "expression_type"],
        "regexpids": [1, 2],
        "preservekeys": true
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
  "jsonrpc": "2.0",
  "result": {
    "1": {
      "regexpid": "1",
      "name": "File systems for discovery",
      "expressions": [
        {
          "expression": "^(btrfs|ext2|ext3|ext4|reiser|xfs|ffs|ufs|jfs|jfs2|vxfs|hfs|apfs|refs|ntfs|fat32|
          "expression_type": "3"
        }
      ]
    },
    "2": {
      "regexpid": "2",
      "name": "Network interfaces for discovery",
      "expressions": [
        {
          "expression": "^Software Loopback Interface",
          "expression_type": "4"
        },
        {
          "expression": "^(In)?[Ll]oop[Bb]ack[0-9._]*$",
          "expression_type": "4"
        },
        {
          "expression": "^NULL[0-9.]*$",
          "expression_type": "4"
        },
        {
          "expression": "^[Ll]o[0-9.]*$",
          "expression_type": "4"
        },
        {
          "expression": "^[Ss]ystem$",
          "expression_type": "4"
        },
        {
          "expression": "^Nu[0-9.]*$",
          "expression_type": "4"
        }
      ]
    }
  },
  "id": 1
}
```


参考来源

CRegexp::get() in *ui/include/classes/api/services/CRegexp.php*.

用户

这个类用于操作用户.

对象属性:

- 用户

可用方法:

- user.checkauthentication - 检查并延长用户会话
- user.create - 创建用户

- user.delete - 删除用户
- user.get - 查询用户
- user.login - 登录用户
- user.logout - 注销用户
- user.unblock - 解锁用户
- user.update - 更新用户

> 用户对象

以下对象都是与 `user` 相关的 API.

用户

用户对象具有以下属性。

属性	类型	描述
userid	string	*(只读)* 用户的 ID。
username *(必选)*	string	用户名。
roleid *(必选)*	string	用户的角色 ID。
attempt_clock	timestamp	*(只读)* 最后一次不成功的登录尝试的时间。
attempt_failed	integer	*(只读)* 最近失败的登录尝试次数。
attempt_ip	string	*(只读)* 最后一次不成功的登录尝试来源的 IP 地址。
autologin	integer	是否启用自动登录。 可能的值： 0 - *(默认)* 自动登录禁用； 1 - 自动登录启用。
autologout	string	用户会话生命周期。接受秒数和带有后缀的时间单位。如果设置为 0s，则会话永不过期。 默认值：15m。
lang	string	用户语言的语言代码，例如，`en_GB`。 默认值：`default` - 系统默认。
name	string	用户的名称。
refresh	string	自动刷新周期。接受秒数或带有后缀的时间单位（例如，30s，90s，1m，1h）。 默认值：30s。
rows_per_page	integer	每页显示的对象行数。 默认值：50。
surname	string	用户的姓氏。
theme	string	用户的主题。 可能的值： `default` - *(默认)* 系统默认； `blue-theme` - 蓝色； `dark-theme` - 深色。
url	string	用户登录后的重定向页面 URL。
timezone	string	用户时区，例如，`Europe/London`，`UTC`。 默认值：`default` - 系统默认。 有关支持的时区完整列表，请参见 PHP 文档。
alias (已弃用)	string	此属性已弃用，请改用 `username`。 用户别名。

请注意，对于一些方法（更新，删除），必选/可选参数组合是不同的。

媒介

媒介对象具有如下属性.

属性	属性类型	描述
mediatypeid (required)	string	被使用的媒介类型 ID.
sendto (required)	string/array	地址, 用户名或者其他接收标识符. 如果媒介类型 是邮件, 值被定义为数组. 如果媒介类型 是其他类型, 值被定义为字符串.
active	integer	是否启用媒介. 可用值: 0 - *(default)* 启用; 1 - 禁用.
severity	integer	触发媒介发送告警的告警级别. 每一位数字代表一个告警级别，并以二进制形式存储. 例如, 12 相当于二进制的 1100, 它表示告警级别为警告和一般严重的告警将触发告警媒介. 参阅触发器对象 查看告警级别列表. 默认: 63
period	string	时间窗口: 能够发送告警通知的时间段 或者以分号分隔的用户宏. 默认: 1-7,00:00-24:00

创建用户

描述

`object user.create(object/array users)`

此方法用于创建新用户.

> **Note:**
> 此方法只有 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色.

> **Note:**
> 通过认证 API 定义的密码策略规则来验证用户密码的强度. 更多信息请查看认证 .

参数

`(object/array)` 用户创建。

除了标准用户属性，该方法还接受以下参数。

参数	类型	描述
passwd （必需）	字符串	用户的密码。 如果用户仅添加到具有 LDAP 访问权限的组，则可以省略。
usrgrps （必需）	数组	要将用户添加到的用户组。 用户组必须具有已定义的 `usrgrpid` 属性。
medias	数组	要创建的用户media。
user_medias （已弃用）	数组	此参数已弃用，请改用 “medias”。 要创建的用户media。

返回值

`(object)` 返回一个包含创建值的 ID 的对象映射 `userids` 属性. 返回的 ID 的顺序与传入的用户的顺序相匹配.

示例

创建一个用户

创建一个新用户, 把他加入用户组同时为其添加用户媒介.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "user.create",
    "params": {
        "username": "John",
        "passwd": "Doe123",
        "roleid": "5",
        "usrgrps": [
            {
                "usrgrpid": "7"
            }
        ],
        "medias": [
            {
                "mediatypeid": "1",
                "sendto": [
                    "support@company.com"
                ],
                "active": 0,
                "severity": 63,
                "period": "1-7,00:00-24:00"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "userids": [
            "12"
        ]
    },
    "id": 1
}
```

参考

- 认证
- 媒介
- 用户组
- 角色

来源

CUser::create() in *ui/include/classes/api/services/CUser.php*.

删除用户

描述

`object user.delete(array users)`

此方法用于删除用户.

> **Note:**
> 此方法仅适用于 *Super admin* 类型的用户. 可在用户角色配置中撤销对此方法的使用. 参阅角色获取详情.

参数

(array) 要删除用户的 ID.

返回值

(object) 返回一个带有 userids 属性（其中包含被删除用户 ID）的对象.

示例

批量删除用户

删除两个用户.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "user.delete",
    "params": [
        "1",
        "5"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

返回:

```
{
    "jsonrpc": "2.0",
    "result": {
        "userids": [
            "1",
            "5"
        ]
    },
    "id": 1
}
```

来源

CUser::delete() in *ui/include/classes/api/services/CUser.php*.

更新用户

描述

`object user.update(object/array users)`

此方法用于更新已经存在的用户.

> **Note:**
> 任何类型的用户都可以使用此方法. 可在用户角色配置中撤销对此方法的使用. 参阅角色获取详情.

> **Note:**
> 通过认证 API 定义的密码策略规则来验证用户密码的强度. 更多信息请查看认证 .

参数

(object/array) 要更新的用户属性。

必须为每个用户定义 userid 属性，所有其他属性都是可选的。只会更新传递的属性，所有其他属性将保持不变。

除了标准用户属性，该方法还接受以下参数。

参数	类型	描述
passwd	字符串	用户的密码。 如果用户属于或仅移动到具有 LDAP 访问权限的组，则可以为空字符串。

参数	类型	描述
usrgrps	数组	User groups 以替换现有用户组。 用户组必须定义 usrgrpid 属性。
medias	数组	用户媒体 替换现有媒体。
user_medias（已弃用）	array	此参数已弃用，请改用 “medias”。 User media 替换现有媒体。

返回值

(object) 返回一个带有 userids 属性（其中包含被更新用户 ID）的对象.

示例

重命名用户

把用户重命名为 John Doe.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "user.update",
    "params": {
        "userid": "1",
        "name": "John",
        "surname": "Doe"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

返回:

```
{
    "jsonrpc": "2.0",
    "result": {
        "userids": [
            "1"
        ]
    },
    "id": 1
}
```

变更用户角色

变更一个用户的角色.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "user.update",
    "params": {
        "userid": "12",
        "roleid": "6"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

返回:

```
{
    "jsonrpc": "2.0",
    "result": {
        "userids": [
            "12"
        ]
```

```
    },
    "id": 1
}
```

参考

- 认证

来源

CUser::update() in *ui/include/classes/api/services/CUser.php*.

查询用户

描述

`integer/array user.get(object parameters)`

此方法用于根据给定的参数查询用户.

Note:
此方法仅适用于 *Super admin* 类型的用户. 可在用户角色配置中撤销对此方法的使用. 参阅角色获取详情.

参数

`(object)` 定义需要输出的参数.

此方法支持如下参数.

参数	类型	描述
mediaids	string/array	仅返回使用了给定媒介的用户.
mediatypeids	string/array	仅返回使用了给定媒介类型的用户.
userids	string/array	仅返回给定用户 ID 内的用户.
usrgrpids	string/array	仅返回属于给定用户组的用户.
getAccess	flag	添加关于用户权限附加信息. 为每个用户添加以下属性: `gui_access` - *(integer)* 用户的前端认证方法. 参考 `gui_access` 的属性关于用户组对象可接受的值列表. `debug_mode` - *(integer)* 表明是否为用户启用了调试功能. 可用值: 0 - 禁用调试模式, 1 - 启用调试模式. `users_status` - *(integer)* 表明用户是否禁用. 可用值: 0 - 启用用户, 1 - 禁用用户.
selectMedias	query	在媒介 属性中返回用户使用的媒介.
selectMediatypes	query	在媒介类型 属性中返回用户使用的媒介类型.

参数	类型	描述
selectUsrgrps	query	在用户组 属性中返回用户所归属的组.
selectRole	query	在角色 属性中返回用户的角色.
sortfield	string/array	根据给定的属性对结果进行排序. 可用值: `userid` 和 `username`.
countOutput	boolean	这些参数对于所有的 `get` 方法是常见的, 在参考说明中有详细描述.
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

`(integer/array)` 返回:

- 一个对象数组;
- 检索对象的计数, 如果, 如果 `countOutput` 参数被使用.

示例

多用户查询

查询所有已配置的用户.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "user.get",
    "params": {
        "output": "extend"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

返回:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "userid": "1",
            "username": "Admin",
            "name": "Zabbix",
            "surname": "Administrator",
            "url": "",
            "autologin": "1",
            "autologout": "0",
            "lang": "en_GB",
            "refresh": "0s",
```

```
            "theme": "default",
            "attempt_failed": "0",
            "attempt_ip": "",
            "attempt_clock": "0",
            "rows_per_page": "50",
            "timezone": "default",
            "roleid": "3"
        },
        {
            "userid": "2",
            "username": "guest",
            "name": "",
            "surname": "",
            "url": "",
            "autologin": "0",
            "autologout": "15m",
            "lang": "default",
            "refresh": "30s",
            "theme": "default",
            "attempt_failed": "0",
            "attempt_ip": "",
            "attempt_clock": "0",
            "rows_per_page": "50",
            "timezone": "default",
            "roleid": "4"
        },
        {
            "userid": "3",
            "username": "user",
            "name": "Zabbix",
            "surname": "User",
            "url": "",
            "autologin": "0",
            "autologout": "0",
            "lang": "ru_RU",
            "refresh": "15s",
            "theme": "dark-theme",
            "attempt_failed": "0",
            "attempt_ip": "",
            "attempt_clock": "0",
            "rows_per_page": "100",
            "timezone": "default",
            "roleid": "1"
        }
    ],
    "id": 1
}
```


Retrieving users as *Admin*

As an *Admin* type user, retrieve detailed data about your own user and limited data for users in your user group.

Request:

```
{
    "jsonrpc": "2.0",
    "method": "user.get",
    "params": {
        "output": "extend",
        "getAccess": true,
        "selectMedias": "extend",
        "selectMediatypes": "extend",
        "selectUsrgrps": "extend",
        "selectRole": "extend"
```

```
    },
    "id": 1
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "userid": "1",
            "username": "Admin",
            "name": "Zabbix",
            "surname": "Administrator",
            "usrgrps": [
                {
                    "usrgrpid": "7",
                    "name": "Zabbix administrators",
                    "gui_access": "0",
                    "users_status": "0",
                    "debug_mode": "0"
                }
            ]
        },
        {
            "userid": "3",
            "username": "database-admin",
            "name": "John",
            "surname": "Doe",
            "url": "",
            "autologin": "0",
            "autologout": "0",
            "lang": "default",
            "refresh": "30s",
            "theme": "default",
            "attempt_failed": "0",
            "attempt_ip": "",
            "attempt_clock": "0",
            "rows_per_page": "50",
            "timezone": "default",
            "roleid": "2",
            "gui_access": "0",
            "debug_mode": "0",
            "users_status": "0",
            "usrgrps": [
                {
                    "usrgrpid": "7",
                    "name": "Zabbix administrators",
                    "gui_access": "0",
                    "users_status": "0",
                    "debug_mode": "0"
                }
            ],
            "medias": [
                {
                    "mediaid": "2",
                    "userid": "3",
                    "mediatypeid": "1",
                    "sendto": [
                        "john.doe@example.com"
                    ],
                    "active": "0",
                    "severity": "63",
```

```
                    "period": "1-7,00:00-24:00"
                }
            ],
            "mediatypes": [
                {
                    "mediatypeid": "1",
                    "type": "0",
                    "name": "Email",
                    "status": "0",
                    "description": "",
                    "maxattempts": "3"
                }
            ],
            "role": {
                "roleid": "2",
                "name": "Admin role",
                "type": "2",
                "readonly": "0"
            }
        }
    ],
    "id": 1
}
```

查询用户数据

查询用户 ID 是 “12” 的用户数据.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "user.get",
    "params": {
        "output": ["userid", "username"],
        "selectRole": "extend",
        "userids": "12"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

返回:

```
{
  "jsonrpc": "2.0",
  "result": [
      {
          "userid": "12",
          "username": "John",
          "role": {
              "roleid": "5",
              "name": "Operator",
              "type": "1",
              "readonly": "0"
          }
      }
  ],
  "id": 1
}
```

参考

- 媒介
- 媒介类型
- 用户组

- 角色

来源
CUser::get() in ui/include/classes/api/services/CUser.php.

检查认证

描述

`object user.checkAuthentication`

此方法检查并延长用户会话。

> **Attention:**
> 使用参数 `sessionid` 调用 **user.checkAuthentication** 方法将默认延长用户会话。

参数

该方法接受以下参数。

参数	类型	描述
extend	boolean	是否延长用户会话。 默认值：“true”。将值设置为 “false” 允许检查用户会话而不延长它。
sessionid	string	如果设置了 `sessionid`，则使用此参数。 用户会话 ID。
token	string	必选，如果未设置 `token`。 用户API 令牌。
		必选，如果未设置 `sessionid`。

返回值

(`object`) 返回一个包含用户信息的对象.

示例

请求:

```
{
    "jsonrpc": "2.0",
    "method": "user.checkAuthentication",
    "params": {
        "sessionid": "673b8ba11562a35da902c66cf5c23fa2"
    },
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "userid": "1",
        "username": "Admin",
        "name": "Zabbix",
        "surname": "Administrator",
        "url": "",
        "autologin": "1",
        "autologout": "0",
        "lang": "ru_RU",
        "refresh": "0",
        "theme": "default",
        "attempt_failed": "0",
```

```
        "attempt_ip": "127.0.0.1",
        "attempt_clock": "1355919038",
        "rows_per_page": "50",
        "timezone": "Europe/Riga",
        "roleid": "3",
        "type": 3,
        "sessionid": "673b8ba11562a35da902c66cf5c23fa2"
        "debug_mode": 0,
        "userip": "127.0.0.1",
        "gui_access": 0
    },
    "id": 1
}
```


Note:
"userData" 参数设置为 true，返回的响应和调用User.login 方法返回相似 (区别在于通过会话 ID 检索用户数据，而不是按照用户名/密码检索).

来源

CUser::checkAuthentication() in *ui/include/classes/api/services/CUser.php*.

注销用户

描述

`string/object user.logout(array)`

此方法用于用户注销 API 并使当前认证令牌失效.

Note:
任何类型的用户都可以使用此方法. 可在用户角色配置中撤销对此方法的使用. 参阅角色.

参数

(array) 此方法接受一个空数组.

返回值

(boolean) 如果用户成功注销，返回 true.

示例

注销

通过 API 注销.

请求:


```
{
    "jsonrpc": "2.0",
    "method": "user.logout",
    "params": [],
    "id": 1,
    "auth": "16a46baf181ef9602e1687f3110abf8a"
}
```


返回:


```
{
    "jsonrpc": "2.0",
    "result": true,
    "id": 1
}
```


参考

- 登录

来源

CUser::login() in *ui/include/classes/api/services/CUser.php*.

登录用户

描述

`string/object user.login(object parameters)`

此方法用于登录到 API 并生成身份验证令牌.

Warning:
使用此方法, 你同时需要使用注销 方法, 防止产生大量未关闭的会话记录.

Attention:
此方法仅适用于未认证用户，同时在 JSON-RPC 请求中不能带有 `auth` 参数.

参数

`(object)` 包含用户名和密码的参数。

该方法接受以下参数。

参数	类型	描述
password （必填）	字符串	用户密码。
username （必填）	字符串	用户名。
userData	flag	返回关于认证用户的信息。
user (已废弃)	字符串	此参数已弃用，请改用 “username”。 用户名。

返回值

`(string/object)` 如果使用了 `userData` 参数，返回一个对象，其中包含有关经过身份验证的用户的信息。

除了标准用户属性，还返回以下信息：

属性	类型	描述
debug_mode	integer	是否为用户启用了调试模式。 参考用户组对象 的 `debug_mode` 属性了解可能的值列表。
gui_access	integer	用户对前端的认证方法。 参考用户组对象 的 `gui_access` 属性了解可能的值列表。
sessionid	string	认证令牌，必须在随后的 API 请求中使用。
type	integer	用户类型。 参考角色对象 的 `type` 属性了解可能的值列表。
userip	string	用户的 IP 地址。

Note:
如果用户在一次或多次失败尝试后成功通过身份验证，该方法将返回 `attempt_clock`、`attempt_failed` 和 `attempt_ip` 属性的当前值，然后重置它们。

如果未使用 `userData` 参数，该方法返回一个认证令牌。

Note:
生成的认证令牌应被记住，并在随后的 JSON-RPC 请求的 `auth` 参数中使用。在使用 HTTP 认证时也需要。

示例

单用户认证

单用户认证.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "user.login",
    "params": {
        "username": "Admin",
        "password": "zabbix"
    },
    "id": 1
}
```

返回:

```
{
    "jsonrpc": "2.0",
    "result": "0424bd59b807674191e7d77572075f33",
    "id": 1
}
```

请求已验证用户的信息

认证并返回有关用户的额外信息。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "user.login",
    "params": {
        "username": "Admin",
        "password": "zabbix",
        "userData": true
    },
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "userid": "1",
        "username": "Admin",
        "name": "Zabbix",
        "surname": "Administrator",
        "url": "",
        "autologin": "1",
        "autologout": "0",
        "lang": "ru_RU",
        "refresh": "0",
        "theme": "default",
        "attempt_failed": "0",
        "attempt_ip": "127.0.0.1",
        "attempt_clock": "1355919038",
        "rows_per_page": "50",
        "timezone": "Europe/Riga",
        "roleid": "3",
        "type": 3,
        "userip": "127.0.0.1",
        "debug_mode": 0,
        "gui_access": 0,
        "sessionid": "5b56eee8be445e98f0bd42b435736e42"
    },
    "id": 1
```

```
}
```

参考

- 注销

来源

CUser::login() in *ui/include/classes/api/services/CUser.php*.

解锁用户

描述

`object user.unblock(array userids)`

此方法用于解锁用户.

> **Note:**
> 此方法仅适用于 *Super admin* 类型的用户. 可在用户角色配置中撤销对此方法的使用. 参阅角色获取详情.

参数

(array) 需要解锁用户的 ID.

返回值

(object) 返回一个带有 `userids` 属性（其中包含被解锁用户 ID）的对象.

示例

批量解锁用户

解锁两个用户.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "user.unblock",
    "params": [
        "1",
        "5"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

返回:

```
{
    "jsonrpc": "2.0",
    "result": {
        "userids": [
            "1",
            "5"
        ]
    },
    "id": 1
}
```

来源

CUser::unblock() in *ui/include/classes/api/services/CUser.php*.

用户宏

该类用于处理主机宏和全局宏.

对象引用:

- 全局宏
- 主机宏

可用方法:

- usermacro.create - 创建主机宏
- usermacro.createglobal - 创建全局宏
- usermacro.delete - 删除主机宏
- usermacro.deleteglobal - 删除全局宏
- usermacro.get - 查询主机宏和全局宏
- usermacro.update - 更新主机宏
- usermacro.updateglobal - 更新全局宏

> 用户宏对象

以下对象均与 `usermacro` 接口相关.

全局宏

全局宏对象具有以下属性.

属性	类型	描述
globalmacroid	string	*(只读)* 全局宏的ID.
macro (必需)	string	宏字符串.
value (必需)	string	宏的值.
type	integer	宏的类型. 可接受的值: 0 - *(default)* 文本宏; 1 - 密文宏; 2 - 密钥宏.
description	string	宏描述信息.

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

主机宏

主机宏对象定义一个主机或模板上可用的宏. 它具有以下属性.

属性	类型	描述
hostmacroid	string	*(只读)* 主机宏 ID.
hostid (必需)	string	宏所属主机的主机ID.
macro (必需)	string	宏名.
value (必需)	string	宏值.
type	integer	宏的类型. 可接受的值: 0 - *(default)* 文本宏; 1 - 密文宏; 2 - 密钥宏.
description	string	宏的描述信息.

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

创建全局宏

描述

`object usermacro.createglobal(object/array globalMacros)`

此方法用于创建新的全局宏.

> **Note:**
> 此方法只有 *Super admin(超级管理员)* 用户可用. 可以在用户角色设置中撤销调用该方法的权限. 更多信息请查看User roles.

参数

(object/array) 需要创建的全局宏.

此方法接受具有标准全局宏属性的全局宏.

返回值

(object) 返回包含 globalmacroids 属性（其中包含被创建全局宏的 ID）的对象. 返回的主机宏 ID 顺序与传入的主机宏顺序相同.

示例

创建一个全局宏

创建全局宏"{$SNMP_COMMUNITY}"，值为"public".

请求:

```
{
    "jsonrpc": "2.0",
    "method": "usermacro.createglobal",
    "params":  {
        "macro": "{$SNMP_COMMUNITY}",
        "value": "public"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

返回:

```
{
    "jsonrpc": "2.0",
    "result": {
        "globalmacroids": [
            "6"
        ]
    },
    "id": 1
}
```

来源

CUserMacro::createGlobal() in *ui/include/classes/api/services/CUserMacro.php*.

创建用户宏

描述

`object usermacro.create(object/array hostMacros)`

此方法用于创建新的主机宏.

> **Note:**
> 此方法只有 *Admin(管理员)* 和 *Super admin(超级管理员)* 用户可用. 可以在用户角色设置中撤销调用该方法的权限. 更多信息请查看User roles .

参数

(object/array) 要创建主机宏.

此方法接受具有标准主机宏属性的主机宏.

返回值

(object) 返回包含 hostmacroids 属性（其中包含被创建主机宏的 ID）的对象. 返回的主机宏 ID 顺序与传入的主机宏顺序相同.

示例

创建一个主机宏

为主机"10198" 创建主机宏"{$SNMP_COMMUNITY}" ，值为"public" .

请求:

```
{
    "jsonrpc": "2.0",
    "method": "usermacro.create",
    "params": {
        "hostid": "10198",
        "macro": "{$SNMP_COMMUNITY}",
        "value": "public"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

返回:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostmacroids": [
            "11"
        ]
    },
    "id": 1
}
```

来源

CUserMacro::create() in *ui/include/classes/api/services/CUserMacro.php*.

删除主机宏

描述

`object usermacro.delete(array hostMacroIds)`

此方法用于删除主机宏.

> **Note:**
> 此方法只有 *Admin(*管理员*)* 和 *Super admin(*超级管理员*)* 用户可用. 可以在用户角色设置中撤销调用该方法的权限. 更多信息请查看. 更多信息请查看User roles.

参数

(array) 需要删除主机宏的 ID.

返回值

(object) 返回包含 hostmacroids 属性（其中包含被删除主机宏的 ID）的对象.

示例

批量删除主机宏

删除两个主机宏.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "usermacro.delete",
    "params": [
        "32",
        "11"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

返回:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostmacroids": [
            "32",
            "11"
        ]
    },
    "id": 1
}
```

来源

CUserMacro::delete() in *ui/include/classes/api/services/CUserMacro.php*.

删除全局宏

描述

`object usermacro.deleteglobal(array globalMacroIds)`

此方法用于删除全局宏.

> **Note:**
> 此方法只有 *Super admin(超级管理员)* 用户可用. 可以在用户角色设置中撤销调用该方法的权限. 更多信息请查看User roles.

参数

(array) 需要删除的全局宏 ID.

返回值

(object) 返回包含 globalmacroids 属性（其中包含被删除全局宏的 ID）的对象.

示例

批量删除全局宏

删除两个全局宏.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "usermacro.deleteglobal",
    "params": [
        "32",
        "11"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

返回:

```
{
    "jsonrpc": "2.0",
    "result": {
        "globalmacroids": [
            "32",
            "11"
        ]
    },
    "id": 1
}
```

来源

CUserMacro::deleteGlobal() in *ui/include/classes/api/services/CUserMacro.php*.

更新主机宏

描述

`object usermacro.update(object/array hostMacros)`

此方法用于更新现有的主机宏.

> **Note:**
> 此方法只有 *Admin(管理员)* 和 *Super admin(超级管理员)* 用户可用. 可以在用户角色设置中撤销调用该方法的权限. 更多信息请查看User roles.

参数

(object/array) 需要更新的Host macro properties .

每个主机宏必须定义 hostmacroid 参数, 其他参数均为可选的. 只有通过的属性才会被更新，其他属性保持不变.

返回值

(object) 返回包含 hostmacroids 属性（其中包含被更新主机宏的 ID）的对象.

示例

更改一个主机宏的值

更改一个主机宏”public” 的值.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "usermacro.update",
    "params": {
        "hostmacroid": "1",
        "value": "public"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

返回:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hostmacroids": [
            "1"
        ]
    },
    "id": 1
}
```

来源

CUserMacro::update() in *ui/include/classes/api/services/CUserMacro.php*.

更新全局宏

描述

`object usermacro.updateglobal(object/array globalMacros)`

此方法用于更新现有的全局宏.

> **Note:**
> 此方法只有 *Super admin(超级管理员)* 用户可用. 可以在用户角色设置中撤销调用该方法的权限. 更多信息请查看User roles.

参数

(object/array) 需要更新的Global macro properties .

每个全局宏必须定义 globalmacroid 参数, 其他参数均为可选的. 只有通过的属性才会被更新，其他属性保持不变.

返回值

(object) 返回包含 globalmacroids 属性（其中包含被更新全局宏的 ID）的对象.

示例

变更一个全局宏的值

变更全局宏”public” 的值.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "usermacro.updateglobal",
    "params": {
        "globalmacroid": "1",
        "value": "public"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

返回:

```
{
    "jsonrpc": "2.0",
    "result": {
        "globalmacroids": [
            "1"
        ]
    },
    "id": 1
}
```

来源

CUserMacro::updateGlobal() in *ui/include/classes/api/services/CUserMacro.php*.

查询用户宏

描述

`integer/array usermacro.get(object parameters)`

此方法用于根据给定参数查询主机宏和全局宏.

> **Note:**
> 此方法适用于所有用户类型. 可以在用户角色设置中撤销调用该方法的权限. 更多信息请查看User roles.

Parameters

(object) 定义所需输出的参数.

该方法支持以下参数.

参数	类型	描述
globalmacro	flag	返回全局宏而不是主机宏.
globalmacroids	string/array	仅返回具有给定 ID 的全局宏.
groupids	string/array	根据给定主机群组仅返回属于主机或模板的主机宏.
hostids	string/array	仅返回属于给定主机或模板的宏.
hostmacroids	string/array	仅返回给定主机宏 ID 的主机宏.
inherited	boolean	如果设置为 `true` 仅返回从模板继承过来的主机原型的宏.
selectGroups	query	在groups 属性中返回拥有这个主机宏的主机组. 仅在查询主机宏时有效.
selectHosts	query	在hosts 属性中返回拥有这个主机宏的主机. 仅在查询主机宏时有效.
selectTemplates	query	在templates 属性中返回拥有这个主机宏的模板. 仅在查询主机宏时有效.
sortfield	string/array	按给定参数排序返回结果. 可用值: `macro`.
countOutput	boolean	这些参数对所有 `get` 方法是通用的，在reference commentary 页面有详细描述.
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

(integer/array) 返回二者之一:

- 一个对象列表;
- 查询结果数量, 如果使用了 `countOutput` 参数.

示例

查询一个主机的主机宏

查询主机"10198" 所有的主机宏.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "usermacro.get",
    "params": {
        "output": "extend",
        "hostids": "10198"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

返回:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "hostmacroid": "9",
```

```
            "hostid": "10198",
            "macro": "{$INTERFACE}",
            "value": "eth0",
            "description": "",
            "type": "0"
        },
        {
            "hostmacroid": "11",
            "hostid": "10198",
            "macro": "{$SNMP_COMMUNITY}",
            "value": "public",
            "description": "",
            "type": "0"
        }
    ],
    "id": 1
}
```

查询全局宏

查询所有的全局宏.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "usermacro.get",
    "params": {
        "output": "extend",
        "globalmacro": true
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

返回:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "globalmacroid": "6",
            "macro": "{$SNMP_COMMUNITY}",
            "value": "public",
            "description": "",
            "type": "0"
        }
    ],
    "id": 1
}
```

来源

CUserMacro::get() in *ui/include/classes/api/services/CUserMacro.php*.

用户组

这个类用于操作用户组.

对象引用:

- 用户组

可用方法:

- usergroup.create - 创建新用户组
- usergroup.delete - 删除用户组
- usergroup.get - 查询用户组
- usergroup.update - 更新用户组

> 用户组对象

以下对象与 `usergroup` 接口相关.

用户组

用户组对象具有以下属性.

属性	类型	描述
usrgrpid	string	*(只读)* 用户组 ID.
name (必需)	string	用户组名称.
debug_mode	integer	是否启用或禁用调试模式. 可用值: 0 - *(default)* 禁用; 1 - 启用.
gui_access	integer	组内用户的前端身份验证方法. 可用值: 0 - *(default)* 使用系统默认身份验证方法; 1 - 使用内部认证; 2 - 使用 LDAP 认证; 3 - 禁止访问前端.
users_status	integer	用户组是启用还是禁用. 可用值: 0 - *(default)* 启用; 1 - 禁用.

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

权限

权限对象具有以下属性.

属性	类型	描述
id (required)	string	需要添加权限的主机组的 ID.
permission (required)	integer	对主机组的访问权限级别. 可用值: 0 - 禁止访问权限; 2 - 只读访问权限; 3 - 读写访问权限.

基于标签的权限

基于标签的权限对象具有以下属性.

属性	类型	描述
groupid (required)	string	要添加权限的主机组的 ID.
tag	string	标签名.
value	string	标签值.

创建用户组

描述

`object usergroup.create(object/array userGroups)`

此方法用户创建新的用户组.

> **Note:**
> 此方法只有 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色.

参数

`(object/array)` 要创建的用户组.

除了标准用户组属性, 此方法接受如下参数.

参数	类型	描述
rights	object/array	分配给用户组的权限
tag_filters	array	分配给用户组标签权限
users	object/array	需要添加到用户组的用户. 用户必须拥有 `userid` 属性.

返回值

`(object)` 返回一个带有 `usrgrpids` 属性（其中包含被创建用户组 ID）的对象. 返回的 ID 的顺序与传递的用户组的顺序相匹配.

示例

创建一个用户组

创建一个用户组, 禁止其访问主机组”2”，并向其添加一个用户.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "usergroup.create",
    "params": {
        "name": "Operation managers",
        "rights": {
            "permission": 0,
            "id": "2"
        },
        "users": [
            {"userid": "12"}
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

返回:

```
{
    "jsonrpc": "2.0",
    "result": {
        "usrgrpids": [
            "20"
```

```
        ]
    },
    "id": 1
}
```

参考

- 权限

来源

CUserGroup::create() in *ui/include/classes/api/services/CUserGroup.php*.

删除用户组

描述

`object usergroup.delete(array userGroupIds)`

此方法用于删除用户组.

> **Note:**
> 此方法只有 *Super admin(*超级管理员*)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色.

参数

(array) 需要删除的用户组 ID.

返回值

(object) 返回一个带有 usrgrpids 属性（其中包含被创建用户组 ID）的对象.

示例

批量删除用户组

删除两个用户组.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "usergroup.delete",
    "params": [
        "20",
        "21"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

返回:

```
{
    "jsonrpc": "2.0",
    "result": {
        "usrgrpids": [
            "20",
            "21"
        ]
    },
    "id": 1
}
```

来源

CUserGroup::delete() in *ui/include/classes/api/services/CUserGroup.php*.

更新用户组

描述

`object usergroup.update(object/array userGroups)`

此方法用于更新已存在的用户组.

> **Note:**
> 此方法仅适用于 超级管理员用户类型. 可在用户角色配置中撤销对此方法的调用权限. 参阅用户角色 查看详情.

参数

(object/array) 要更新的用户组属性。

必须为每个用户组定义 usrgrpid 属性，所有其他属性均为可选。只有传递的属性将被更新，其他属性将保持不变。

除了标准用户组属性，该方法还接受以下参数。

参数	类型	描述
rights	object/array	权限 替换当前分配给用户组的权限。
tag_filters	array	基于标签的权限 替换当前分配给用户组的权限。
users	object/array	用户 替换当前分配给用户组的用户。 用户必须只定义 userid 属性。

返回值

(object) 返回一个带有 usrgrpids 属性（其中包含被更新用户组 ID）的对象.

示例

禁用一个用户组

禁用一个用户组.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "usergroup.update",
    "params": {
        "usrgrpid": "17",
        "users_status": "1"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

返回:

```
{
    "jsonrpc": "2.0",
    "result": {
        "usrgrpids": [
            "17"
        ]
    },
    "id": 1
}
```

参考

- 权限

来源

CUserGroup::update() in *ui/include/classes/api/services/CUserGroup.php*.

查询用户组

描述

`integer/array usergroup.get(object parameters)`

该方法允许根据给定的参数检索用户组.

Note:
此方法只有 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色.

参数

`(object)` 定义所需输出的参数.

该方法支持以下参数.

参数	类型	描述
status	integer	只返回具有给定状态的用户组. 参阅用户组页面 获取支持的状态列表.
userids	string/array	只返回包含给定用户的用户组.
usrgrpids	string/array	只返回具有给定 ID 的用户组.
selectTagFilters	query	在标签权限 属性中返回用户标签权限. 它具有以下属性: `groupid` - (string) 主机组 ID; `tag` - (string) 标签名; `value` - (string) 标签值.
selectUsers	query	在用户 属性中返回用户组中的用户.
selectRights	query	在权限 属性中返回用户组权限. 它具有以下属性: `permission` - (integer) 访问级别到主机组; `id` - (string) 主机组的 ID. 参阅用户组页面 获取主机组的访问级别列表.
limitSelects	integer	限制子选择返回的记录数.
sortfield	string/array	按照给定的属性对结果进行排序. 可接受的值: `usrgrpid`, `name`.
countOutput	boolean	`get` 方法的常用参数都被记录在参考说明.
editable	boolean	
excludeSearch	boolean	

参数	类型	描述
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

(integer/array) 返回结果二选一:

- 一个对象数组;
- 如果使用 countOutput 参数，返回检索到的对象总数.

示例

查询启用的用户组

查询所有启用状态的用户组.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "usergroup.get",
    "params": {
        "output": "extend",
        "status": 0
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

返回:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "usrgrpid": "7",
            "name": "Zabbix administrators",
            "gui_access": "0",
            "users_status": "0",
            "debug_mode": "1"
        },
        {
            "usrgrpid": "8",
            "name": "Guests",
            "gui_access": "0",
            "users_status": "0",
            "debug_mode": "0"
        },
        {
            "usrgrpid": "11",
            "name": "Enabled debug mode",
            "gui_access": "0",
            "users_status": "0",
            "debug_mode": "1"
        },
        {
            "usrgrpid": "12",
            "name": "No access to the frontend",
```

```
            "gui_access": "2",
            "users_status": "0",
            "debug_mode": "0"
        },
        {
            "usrgrpid": "14",
            "name": "Read only",
            "gui_access": "0",
            "users_status": "0",
            "debug_mode": "0"
        },
        {
            "usrgrpid": "18",
            "name": "Deny",
            "gui_access": "0",
            "users_status": "0",
            "debug_mode": "0"
        }
    ],
    "id": 1
}
```


参考

- 用户

来源

CUserGroup::get() in *ui/include/classes/api/services/CUserGroup.php*.

监控项

此部分旨在介绍如何对监控项进行操作。

引用对象：

- 监控项

可用方法：

- 创建 - 创建新监控项
- 删除 - 删除监控项
- 检索 - 检索监控项
- 更新 - 更新监控项

> 监控项对象

以下对象直接调用 `item` API 进行操作。

监控项

Note:
Web 监控项不能通过 Zabbix API 直接创建、更新或删除。

监控项对象具有以下属性。

属性	类型	描述
itemid	string	*(readonly)* 监控项的 ID。

属性	类型	描述
delay (required)	string	监控项的更新间隔。接受秒数或带有后缀的时间单位 (30s,1m,2h,1d)。 可选地，可以指定一个或多个自定义间隔，可以是灵活的间隔或调度。 多个间隔用分号分隔。 可以使用用户宏。一个宏必须填满整个字段。不支持在字段中使用多个宏或将宏与文本混合。 灵活的间隔可以写成两个宏，用正斜杠分隔 (例如 `{$FLEX_INTERVAL}/{$FLEX_PERIOD}`)。 对于 Zabbix trapper、依赖项和具有 `mqtt.get` 键的 Zabbix agent (active) 是可选的。
hostid (required)	string	监控项所属的主机或模板的 ID。 对于更新操作，此字段是 *readonly*。
interfaceid (required)	string	监控项的主机接口 ID。 仅用于主机监控项。对于 Zabbix agent (active)、Zabbix 内部、Zabbix trapper、计算、依赖、数据库监控和脚本监控项不需要。对于 HTTP 代理监控项是可选的。
key_ (required)	string	监控项键。
name (required)	string	监控项的名称。
type (required)	integer	监控项的类型。 可能的值： 0 - Zabbix agent; 2 - Zabbix trapper; 3 - 简单检查; 5 - Zabbix 内部; 7 - Zabbix agent (active); 9 - Web 监控项; 10 - 外部检查; 11 - 数据库监控; 12 - IPMI agent; 13 - SSH agent; 14 - Telnet agent; 15 - 计算; 16 - JMX agent; 17 - SNMP trap; 18 - 依赖监控项; 19 - HTTP agent; 20 - SNMP agent; 21 - 脚本
url (required)	string	URL 字符串，仅当 HTTP 代理监控项类型时需要。支持用户宏，{HOST.IP}, {HOST.CONN}, {HOST.DNS}, {HOST.HOST}, {HOST.NAME}, {ITEM.ID}, {ITEM.KEY}。
value_type (required)	integer	监控项的信息类型。 可能的值： 0 - 数值浮点; 1 - 字符; 2 - 日志; 3 - 数值无符号; 4 - 文本。

属性	类型	描述
allow_traps	integer	HTTP 代理监控项字段。允许像 trapper 监控项类型一样填充值。 0 - *(default)* 不允许接受传入数据。 1 - 允许接受传入数据。
authtype	integer	仅由 SSH 代理监控项或 HTTP 代理监控项使用。 SSH 代理认证方法可能的值： 0 - *(default)* 密码; 1 - 公钥。 HTTP 代理认证方法可能的值： 0 - *(default)* 无 1 - 基本 2 - NTLM 3 - Kerberos
description	string	监控项的描述。
error	string	*(readonly)* 如果更新监控项值存在问题，则错误文本。
flags	integer	*(readonly)* 监控项的来源。 可能的值： 0 - 普通监控项; 4 - 发现的监控项。
follow_redirects	integer	HTTP 代理监控项字段。在轮询数据时跟随响应重定向。 0 - 不跟随重定向。 1 - *(default)* 跟随重定向。
headers	object	HTTP 代理监控项字段。HTTP(S) 请求头部的对象，其中头部名称用作键，头部值作为值。 示例： { ”User-Agent”: ”Zabbix” }
history	string	历史数据应该存储的时间单位。也接受用户宏。 默认值：90d。
http_proxy	string	HTTP 代理监控项字段。HTTP(S) 代理连接字符串。
inventory_link	integer	由监控项填充的主机库存字段的 ID。 参考主机资产以获取支持的主机库存字段及其 ID 的列表。 默认值：0。
ipmi_sensor	string	IPMI 传感器。仅由 IPMI 监控项使用。
jmx_endpoint	string	JMX 代理自定义连接字符串。 默认值： service:jmx:rmi:///jndi/rmi://{HOST.CONN}:{HOST.PO
lastclock	timestamp	*(readonly)* 最后更新监控项值的时间。 默认情况下，仅显示最近 24 小时内的值。您可以通过更改最大历史显示期限参数的值，在管理 → 常规菜单部分来扩展此时间段。

属性	类型	描述
lastns	integer	*(readonly)* 最后更新监控项值的纳秒。 默认情况下，仅显示最近 24 小时内的值。 您可以通过更改最大历史显示期限参数的值，在管理 → 常规菜单部分来扩展此时间段。
lastvalue	string	*(readonly)* 监控项的最后一个值。 默认情况下，仅显示最近 24 小时内的值。 您可以通过更改最大历史显示期限参数的值，在管理 → 常规菜单部分来扩展此时间段。
logtimefmt	string	日志条目中时间的格式。仅由日志监控项使用。
master_itemid	integer	主监控项 ID。 递归最多 3 个依赖监控项，依赖监控项的最大数量等于 29999。 由依赖监控项需要。
output_format	integer	HTTP 代理监控项字段。是否应将响应转换为 JSON。 0 - *(default)* 存储原始数据。 1 - 转换为 JSON。
params	string	根据监控项类型的附加参数： - 对于 SSH 和 Telnet 监控项执行的脚本; - 对于数据库监控监控项的 SQL 查询; - 对于计算监控项的公式; - 对于脚本监控项的脚本。
parameters	array	脚本监控项的附加参数。具有’name’和’value’ 属性的对象数组，其中名称必须是唯一的。
password	string	认证密码。由简单检查、SSH、Telnet、数据库监控、JMX 和 HTTP 代理监控项使用。 当由 JMX 使用时，密码和用户名应该一起指定，或者两个属性都应该留空。
post_type	integer	HTTP 代理监控项字段。存储在 posts 属性中的 post 数据正文的类型。 0 - *(default)* 原始数据。 2 - JSON 数据。 3 - XML 数据。
posts	string	HTTP 代理监控项字段。HTTP(S) 请求正文数据。与 post_type 一起使用。
prevvalue	string	*(readonly)* 监控项的上一个值。 默认情况下，仅显示最近 24 小时内的值。 您可以通过更改最大历史显示期限参数的值，在管理 → 常规菜单部分来扩展此时间段。
privatekey	string	私钥文件的名称。
publickey	string	公钥文件的名称。
query_fields	array	HTTP 代理监控项字段。查询参数。具有’key’:’value’ 对的对象数组，其中值可以是空字符串。
request_method	integer	HTTP 代理监控项字段。请求方法的类型。 0 - *(default)* GET 1 - POST 2 - PUT 3 - HEAD

属性	类型	描述
retrieve_mode	integer	HTTP 代理监控项字段。应该存储响应的哪一部分。 0 - *(default)* 正文。 1 - 头部。 2 - 正文和头部都将被存储。 对于 request_method HEAD，只允许值 1。
snmp_oid	string	SNMP OID。
ssl_cert_file	string	HTTP 代理监控项字段。公共 SSL 密钥文件路径。
ssl_key_file	string	HTTP 代理监控项字段。私有 SSL 密钥文件路径。
ssl_key_password	string	HTTP 代理监控项字段。SSL 密钥文件的密码。
state	integer	*(readonly)* 监控项的状态。 可能的值： 0 - *(default)* 正常； 1 - 不支持。
status	integer	监控项的状态。 可能的值： 0 - *(default)* 启用的监控项； 1 - 禁用的监控项。
status_codes	string	HTTP 代理监控项字段。所需的 HTTP 状态码范围，用逗号分隔。还支持用户宏作为逗号分隔列表的一部分。 示例：200,200-{$M},{$M},200-400
templateid	string	(readonly) 父模板监控项的 ID。 提示：使用 `hostid` 属性来指定监控项所属的模板。
timeout	string	监控项数据轮询请求的超时时间。用于 HTTP 代理和脚本监控项。支持用户宏。 默认值：3s 最大值：60s
trapper_hosts	string	允许的主机。用于 trapper 监控项或 HTTP 代理监控项。
trends	string	趋势数据应该存储的时间单位。也接受用户宏。 默认值：365d。
units	string	值的单位。
username	string	认证的用户名。用于简单检查、SSH、Telnet、数据库监控、JMX 和 HTTP 代理监控项。 SSH 和 Telnet 监控项需要。 当由 JMX 使用时，密码也应该与用户名一起指定，或者两个属性都应该留空。
uuid	string	通用唯一标识符，用于将导入的监控项与已存在的关联。仅用于模板上的监控项。如果没有提供，则自动生成。
valuemapid	string	关联的值映射的 ID。

属性	类型	描述
verify_host	integer	HTTP 代理监控项字段。是否验证连接的主机名是否与主机证书中的名称匹配。 0 - *(default)* 不验证。 1 - 验证。
verify_peer	integer	HTTP 代理监控项字段。是否验证主机的证书是否真实。 0 - *(default)* 不验证。 1 - 验证。

请注意，对于一些方法（更新，删除），所需/可选参数的组合是不同的。

监控项标签

监控项标签对象具有以下属性。

属性	类型	描述
tag （必需）	string	监控项标签名称。
value	string	监控项标签值。

监控项预处理

监控项预处理对象具有以下属性。

属性	类型	说明
type (必要)	integer（整数）	预处理选项类型。 可用值： 1 - Custom multiplier（自定义乘数）； 2 - Right trim（移除右侧空白字符）； 3 - Left trim（移除左侧空白字符）； 4 - Trim（移除空白字符）； 5 - Regular expression matching（正则表达式匹配）； 6 - Boolean to decimal（布尔值转换十进制）； 7 - Octal to decimal（八进制转换十进制）； 8 - Hexadecimal to decimal（十六进制转十进制）； 9 - Simple change（先前值到新值的基本变化）； 10 - Change per second（每秒钟变化量）； 11 - XML XPath（XML 解析）； 12 - JSONPath（JSON 解析）； 13 - In range（生成序列）； 14 - Matches regular expression（匹配正则表达式）； 15 - Does not match regular expression（不匹配正则表达式）； 16 - Check for error in JSON（检查 JSON 错误）； 17 - Check for error in XML（检查 XML 错误）； 18 - Check for error using regular expression（检查正则表达式使用错误）； 19 - Discard unchanged（丢弃重复数据）； 20 - Discard unchanged with heartbeat（设置心跳检查周期，丢弃重复数据）；

属性	类型	说明
params (必要)	预处理选项使用的其他参数。多个参数由 LF（\n）字符分隔。	
error_handler (必要)	integer（整数）	预处理步骤失败时使用的操作类型： 可用值： 0 - Error message is set by Zabbix server（Zabbix 服务器自带错误消息）； 1 - Discard value（丢弃值）； 2 - Set custom value（设置自定义值）； 3 - Set custom error message（设置自定义错误信息）。
error_handler_params (必要)	string（字符串）	错误处理器参数。与 `error_handler` 搭配使用。 如果 `error_handler` 类型为 0 或 1，此值必须为空。 如果 `error_handler` 类型为 2，此值可以为空。 如果 `error_handler` 类型为 3，此值不能为空。

预处理类型均支持以下参数和错误处理器。

预处理类型	名称	参数 1	参数 2	参数 3	支持的错误处理器
1	Custom multiplier（自定义倍数）	number[1, 6]（数字型）			0, 1, 2, 3
2	Right trim（移除右侧空白字符）	list of characters[2]（字符列表）			

预处理类型	名称	参数 1	参数 2	参数 3	支持的错误处理器
3	Left trim (移除左侧空白字符)	list of characters[2] (字符列表)			
4	Trim (移除空白字符)	list of characters[2] (字符列表)			
5	Regular expression (正则表达式)	pattern[3] (模式)	output[2] (输出)		0, 1, 2, 3
6	Boolean to decimal (布尔值转换十进制)				0, 1, 2, 3
7	Octal to decimal (八进制转换十进制)				0, 1, 2, 3
8	Hexadecimal to decimal (十六进制转十进制)				0, 1, 2, 3
9	Simple change (先前值到新值的基本变化)				0, 1, 2, 3
10	Change per second (每秒钟变化量)				0, 1, 2, 3

预处理类型	名称	参数 1	参数 2	参数 3	支持的错误处理器
11	XML XPath (XML 解析)	path[4]（解析）			0, 1, 2, 3
12	JSONPath (JSON 解析)	path[4]（解析）			0, 1, 2, 3
13	In range (生成序列)	min[1, 6]（最小值）	max[1, 6]（最大值）		0, 1, 2, 3
14	Matches regular expression (匹配正则表达式)	pattern[3]（模式）			0, 1, 2, 3
15	Does not match regular expression (不匹配正则表达式)	pattern[3]（模式）			0, 1, 2, 3
16	Check for error in JSON (检查 JSON 错误)	path[4]（解析）			0, 1, 2, 3
17	Check for error in XML (检查 XML 错误)	path[4]（解析）			0, 1, 2, 3

预处理类型	名称	参数 1	参数 2	参数 3	支持的错误处理器
18	Check for error using regular expression (检查正则表达式使用错误)	pattern[3]（模式）	output[2]（输出）		0, 1, 2, 3
19	Discard unchanged (丢弃重复数据)				
20	Discard unchanged with heartbeat (设置心跳检查周期，丢弃重复数据)	seconds[5, 6]（秒）			
21	JavaScript (JS格式)	script[2]			
22	Prometheus pattern (Prometheus模式)	pattern[6, 7]（模式）	`value`, `label`, `function`	output[8, 9]（输出）	0, 1, 2, 3
23	Prometheus to JSON (Prometheus转换JSON)	pattern[6, 7]（模式）			0, 1, 2, 3

预处理类型	名称	参数 1	参数 2	参数 3	支持的错误处理器
24	CSV to JSON（CSV 转换 JSON）	character[2]（字符）	character[2]（字符）	0,1	0, 1, 2, 3
25	Replace（替换）	search string[2]（查找字符串）	replacement[2]（替换）		
26	Check unsupported（检查不支持的值）				1, 2, 3
27	XML to JSON（XML 转换 JSON）				0, 1, 2, 3

[1] 整数或浮点数
[2] 字符串
[3] 正则表达式
[4] JSONPath 或 XML XPath 解析
[5] 正整数（支持时间后缀，例如：30s、1m、2h、1d）
[6] 用户宏
[7] Prometheus 模式遵循以下语法：`<metric name>{<label name>="<label value>", ...} == <value>`。Prometheus 每个模式组件（metric, label name, label value and metric value）都可以是用户宏。
[8] 如果第二个参数选择 `label`，则 Prometheus 输出遵循以下语法：`<label name>`（可以是用户宏）
[9] 如果第二个参数选择 `function`，则相对应以下聚合函数：`sum`、`min`、`max`、`avg`、`count`。

创建

说明

`object item.create(object/array items)`

此方法用于创建新监控项。

Note:
无法通过 Zabbix API 创建 Web 监控项。

Note:
此方法仅允许 *Admin*（管理员）和 *Super admin*（超级管理员）类型的用户使用。调用此方法的权限可以在用户角色设置中撤销。更多信息请参阅用户角色 。

参数

`(object/array)` 创建目标监控项。

除了标准监控项属性 之外，此方法同样适用以下参数。

参数	类型	说明
preprocessing	array	监控项预处理 选项。
tags	array	监控项标签 。

返回值

(object) 返回一个对象，该对象包含 itemids 属性下创建的监控项的 ID。返回 ID 的顺序与传递监控项的顺序一致。

示例

创建监控项

创建一个数值类型的 Zabbix agent 监控项，并附带 2 个监控项标签，用于监控 ID 为"30074" 的主机上的空闲磁盘空间。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "item.create",
    "params": {
        "name": "Free disk space on /home/joe/",
        "key_": "vfs.fs.size[/home/joe/,free]",
        "hostid": "30074",
        "type": 0,
        "value_type": 3,
        "interfaceid": "30084",
        "tags": [
            {
                "tag": "Disk usage"
            },
            {
                "tag": "Equipment",
                "value": "Workstation"
            }
        ],
        "delay": "30s"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "24758"
        ]
    },
    "id": 1
}
```

创建主机资产监控项

创建 Zabbix agent 监控项，用于增添主机的 "OS" 资产字段。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "item.create",
    "params": {
        "name": "uname",
        "key_": "system.uname",
        "hostid": "30021",
        "type": 0,
```

```
        "interfaceid": "30007",
        "value_type": 1,
        "delay": "10s",
        "inventory_link": 5
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "24759"
        ]
    },
    "id": 1
}
```

创建预处理监控项

创建自定义乘数监控项。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "item.create",
    "params": {
        "name": "Device uptime",
        "key_": "sysUpTime",
        "hostid": "11312",
        "type": 4,
        "snmp_oid": "SNMPv2-MIB::sysUpTime.0",
        "value_type": 1,
        "delay": "60s",
        "units": "uptime",
        "interfaceid": "1156",
        "preprocessing": [
            {
                "type": "1",
                "params": "0.01",
                "error_handler": "1",
                "error_handler_params": ""
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "44210"
        ]
    },
    "id": 1
}
```

创建依赖监控项

创建 ID 为 24759 的主监控项的依赖监控项。仅允许依赖于同一主机，主项和依赖项要具有相同的 hostid。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "item.create",
    "params": {
      "hostid": "30074",
      "name": "Dependent test item",
      "key_": "dependent.item",
      "type": "18",
      "master_itemid": "24759",
      "value_type": "2"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "44211"
        ]
    },
    "id": 1
}
```

创建 HTTP agent 监控项

创建 JSON 响应预处理式的 POST 请求方法监控项。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "item.create",
    "params": {
        "url":"http://127.0.0.1/http.php",
        "query_fields": [
            {
                "mode":"json"
            },
            {
                "min": "10"
            },
            {
                "max": "100"
            }
        ],
        "interfaceid": "1",
        "type":"19",
        "hostid":"10254",
        "delay":"5s",
        "key_":"json",
        "name":"http agent example JSON",
        "value_type":"0",
        "output_format":"1",
        "preprocessing": [
            {
                "type": "12",
                "params": "$.random"
            }
        ]
```

```
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 2
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "23865"
        ]
    },
    "id": 3
}
```

创建脚本监控项

创建用于简单数据收集的脚本监控项。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "item.create",
    "params": {
        "name": "Script example",
        "key_": "custom.script.item",
        "hostid": "12345",
        "type": 21,
        "value_type": 4,
        "params": "var request = new CurlHttpRequest();\nreturn request.Post(\"https://postman-echo.com/po
        "parameters": [{
            "name": "host",
            "value": "{HOST.CONN}"
        }],
        "timeout": "6s",
        "delay": "30s"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 2
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "23865"
        ]
    },
    "id": 3
}
```

源码

ui/include/classes/api/services/CItem.php - CItem::create()

删除

说明

`object item.delete(array itemIds)`

此方法用于删除监控项。

Note:
无法通过 Zabbix API 删除 Web 监控项。

Note:
此方法仅允许 *Admin*（管理员）和 *Super admin*（超级管理员）类型的用户使用。调用此方法的权限可以在用户角色设置中撤销。更多信息请参见用户角色 。

参数

(array) 删除目标监控项 ID。

返回值

返回一个对象，该对象包含 itemids 属性下已删除项目的 ID。

示例

删除多个监控项

删除两个监控项。如果主项被删除，依赖项和项目相关也会自动删除。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "item.delete",
    "params": [
        "22982",
        "22986"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "22982",
            "22986"
        ]
    },
    "id": 1
}
```

源码

ui/include/classes/api/services/CItem.php - CItem::delete()

更新

说明

`object item.update(object/array items)`

此方法用于更新既存监控项。

Note:
无法通过 Zabbix API 更新 Web 监控项。

Note:
此方法仅允许 *Admin*（管理员）和 *Super admin*（超级管理员）类型的用户使用。调用此方法的权限可以在用户角色设置中撤销。更多信息请参阅用户角色 。

说明

(object/array) 监控项目标更新属性.

每个监控项都必须定义 itemid 属性，其余属性为可选项。仅传递的属性将被更新，所有其他属性将保持不变。

除了标准监控项属性 之外，此方法同样适用以下参数。

参数	类型	说明
preprocessing	array	监控项预处理 用于替换当前预处理选项。
tags	array	标签 用于替换当前标签。

返回值

(object) 返回一个对象，该对象包含 itemids 属性下更新的监控项的 ID。

示例

启用监控项

启用监控项，即将其状态设置为 “0” 。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "item.update",
    "params": {
        "itemid": "10092",
        "status": 0
    },
    "auth": "700ca65537074ec963db7efabda78259",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "10092"
        ]
    },
    "id": 1
}
```

更新依赖监控项

更新依赖监控项名称和主监控项 ID。仅允许依赖于同一主机，因此主监控项和依赖监控项应具有相同的主机 ID。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "item.update",
    "params": {
        "name": "Dependent item updated name",
        "master_itemid": "25562",
        "itemid": "189019"
    },
    "auth": "700ca65537074ec963db7efabda78259",
    "id": 1
}
```

更新：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
```

```
            "189019"
        ]
    },
    "id": 1
}
```

更新 HTTP agent 监控项

启用监控项状态值捕获。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "item.update",
    "params": {
        "itemid": "23856",
        "allow_traps": "1"
    },
    "auth": "700ca65537074ec963db7efabda78259",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "23856"
        ]
    },
    "id": 1
}
```

更新监控项预处理

更新监控项预处理规则为 “In range” 。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "item.update",
    "params": {
        "itemid": "23856",
        "preprocessing": [
            {
                "type": "13",
                "params": "\n100",
                "error_handler": "1",
                "error_handler_params": ""
            }
        ]
    },
    "auth": "700ca65537074ec963db7efabda78259",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "23856"
        ]
    },
```

```
    "id": 1
}
```

更新脚本监控项

更新脚本监控项为其他脚本，并删除在旧脚本中使用的非必要参数。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "item.update",
    "params": {
        "itemid": "23865",
        "parameters": [],
        "script": "Zabbix.Log(3, 'Log test');\nreturn 1;"
    },
    "auth": "700ca65537074ec963db7efabda78259",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "23865"
        ]
    },
    "id": 1
}
```

源码

ui/include/classes/api/services/CItem.php - CItem::update()

获取

说明

`integer/array item.get(object parameters)`

此方法用于依据给定的参数检索监控项。

> **Note:**
> 此方法允许所有类型的用户使用。调用此方法的权限可以在用户角色设置中撤销。更多信息请参见用户角色 。

参数

`(object)` 定义所需输出的参数。

该方法支持以下参数。

参数	类型	描述
itemids	string/array	只返回给定 ID 的监控项。
groupids	string/array	只返回属于给定组的主机的监控项。
templateids	string/array	只返回属于给定模板的监控项。
hostids	string/array	只返回属于给定主机的监控项。
proxyids	string/array	只返回由给定 Proxy 监控的监控项。
interfaceids	string/array	只返回使用给定主机接口的监控项。
graphids	string/array	只返回在给定图表中使用的监控项。
triggerids	string/array	只返回在给定触发器中使用的监控项。
webitems	flag	在结果中包含 web 监控项。
inherited	boolean	如果设置为 `true`，则只返回从模板继承的监控项。
templated	boolean	如果设置为 `true`，则只返回属于模板的监控项。
monitored	boolean	如果设置为 `true`，则只返回属于监控中主机的启用监控项。

参数	类型	描述
group	string	只返回属于具有给定名称的组的监控项。
host	string	只返回属于具有给定名称的主机的监控项。
evaltype	integer	标签搜索的规则。 可能的值： 0 - （默认）与/或； 2 - 或。
tags	array of objects	只返回具有给定标签的监控项。根据操作符的值，对标签进行精确匹配或对标签值进行大小写敏感或不敏感搜索。 格式：`[{"tag": "<tag>", "value": "<value>", "operator": "<operator>"}, ...]`。 空数组返回所有监控项。 可能的操作符类型： 0 - （默认）相似； 1 - 相等； 2 - 不相似； 3 - 不相等 4 - 存在； 5 - 不存在。
with_triggers	boolean	如果设置为 `true`，则只返回在触发器中使用的监控项。
selectHosts	query	以数组形式返回属于监控项的主机属性。
selectInterfaces	query	返回使用该监控项的主机接口数组的接口属性。
selectTriggers	query	返回监控项所涉及的触发器数组的触发器属性。 支持 `count`。
selectGraphs	query	返回包含该监控项的图表数组的图表属性。 支持 `count`。
selectDiscoveryRule	query	返回创建该监控项的 LLD 规则的discoveryRule属性。
selectItemDiscovery	query	返回与从其创建的监控项原型相关联的监控项发现对象的 `itemDiscovery` 属性。 它具有以下属性： `itemdiscoveryid` - (`string`) 监控项发现的 ID； `itemid` - (`string`) 已发现监控项的 ID； `parent_itemid` - (`string`) 从其创建监控项的监控项原型的 ID； `key_` - (`string`) 监控项原型的键； `lastcheck` - (`timestamp`) 最后发现监控项的时间； `ts_delete` - (`timestamp`) 不再发现的监控项将被删除的时间。

参数	类型	描述
selectPreprocessing	query	返回包含监控项预处理选项的预处理属性。 它具有以下属性： type - (string) 预处理选项类型： 1 - 自定义乘数； 2 - 右修剪； 3 - 左修剪； 4 - 修剪； 5 - 正则表达式匹配； 6 - 布尔值转十进制； 7 - 八进制转十进制； 8 - 十六进制转十进制； 9 - 简单变化； 10 - 每秒变化； 11 - XML XPath； 12 - JSONPath； 13 - 在范围； 14 - 匹配正则表达式； 15 - 不匹配正则表达式； 16 - 在 JSON 中检查错误； 17 - 在 XML 中检查错误； 18 - 使用正则表达式检查错误； 19 - 丢弃未更改的； 20 - 丢弃未更改且具有心跳的； 21 - JavaScript； 22 - Prometheus 模式； 23 - Prometheus 转 JSON； 24 - CSV 转 JSON； 25 - 替换； 26 - 检查不支持的值； 27 - XML 转 JSON。 params - (string) 预处理选项使用的附加参数。多个参数由 LF (\n) 字符分隔。 error_handler - (string) 预处理步骤失败时使用的操作类型： 0 - 由 Zabbix 服务器设置错误消息； 1 - 丢弃值； 2 - 设置自定义值； 3 - 设置自定义错误消息。 error_handler_params - (string) 错误处理参数。
selectTags	query	在tags属性中返回监控项标签。
selectValueMap	query	返回包含监控项值映射的valuemap属性。
filter	object	只返回完全匹配给定过滤器的结果。 接受一个数组，其中键是属性名称，值是单一值或要匹配的值数组。 支持额外的过滤器： host - 监控项所属主机的技术名称。
limitSelects	integer	限制子选择返回的记录数。 适用于以下子选择： selectGraphs - 结果将按 name 排序； selectTriggers - 结果将按 description 排序。
sortfield	string/array	根据给定属性对结果进行排序。 可能的值有：itemid, name, key_, delay, history, trends, type 和 status。
countOutput	boolean	这些参数对所有 get 方法都是通用的，详细描述在参考说明页面。
editable	boolean	
excludeSearch	boolean	

参数	类型	描述
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

创建一个数值类型的 Zabbix agent 监控项，并附加两个监控项标签，用于监控主机 ID "30074" 上的空闲磁盘空间。

返回值

(integer/array) 返回其中之一：

- 一组对象；
- 如果使用了 countOutput 参数，则检索对象的计数。

示例

按照关键字查找监控项

检索 ID 为 "10084" 的主机包含关键字 "系统" 一词的所有监控项，并按名称排序。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "item.get",
    "params": {
        "output": "extend",
        "hostids": "10084",
        "search": {
            "key_": "system"
        },
        "sortfield": "name"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "itemid": "23298",
            "type": "0",
            "snmp_oid": "",
            "hostid": "10084",
            "name": "Context switches per second",
            "key_": "system.cpu.switches",
            "delay": "1m",
            "history": "7d",
            "trends": "365d",
            "lastvalue": "2552",
            "lastclock": "1351090998",
            "prevvalue": "2641",
            "state": "0",
            "status": "0",
            "value_type": "3",
            "trapper_hosts": "",
            "units": "sps",
            "error": "",
```

```
        "logtimefmt": "",
        "templateid": "22680",
        "valuemapid": "0",
        "params": "",
        "ipmi_sensor": "",
        "authtype": "0",
        "username": "",
        "password": "",
        "publickey": "",
        "privatekey": "",
        "lastns": "564054253",
        "flags": "0",
        "interfaceid": "1",
        "description": "",
        "inventory_link": "0",
        "lifetime": "0s",
        "evaltype": "0",
        "jmx_endpoint": "",
        "master_itemid": "0",
        "timeout": "3s",
        "url": "",
        "query_fields": [],
        "posts": "",
        "status_codes": "200",
        "follow_redirects": "1",
        "post_type": "0",
        "http_proxy": "",
        "headers": [],
        "retrieve_mode": "0",
        "request_method": "0",
        "output_format": "0",
        "ssl_cert_file": "",
        "ssl_key_file": "",
        "ssl_key_password": "",
        "verify_peer": "0",
        "verify_host": "0",
        "allow_traps": "0",
        "parameters": []
    },
    {
        "itemid": "23299",
        "type": "0",
        "snmp_oid": "",
        "hostid": "10084",
        "name": "CPU $2 time",
        "key_": "system.cpu.util[,idle]",
        "delay": "1m",
        "history": "7d",
        "trends": "365d",
        "lastvalue": "86.031879",
        "lastclock": "1351090999",
        "prevvalue": "85.306944",
        "state": "0",
        "status": "0",
        "value_type": "0",
        "trapper_hosts": "",
        "units": "%",
        "error": "",
        "logtimefmt": "",
        "templateid": "17354",
        "valuemapid": "0",
        "params": "",
```

```
        "ipmi_sensor": "",
        "authtype": "0",
        "username": "",
        "password": "",
        "publickey": "",
        "privatekey": "",
        "lastns": "564256864",
        "flags": "0",
        "interfaceid": "1",
        "description": "The time the CPU has spent doing nothing.",
        "inventory_link": "0",
        "lifetime": "0s",
        "evaltype": "0",
        "jmx_endpoint": "",
        "master_itemid": "0",
        "timeout": "3s",
        "url": "",
        "query_fields": [],
        "posts": "",
        "status_codes": "200",
        "follow_redirects": "1",
        "post_type": "0",
        "http_proxy": "",
        "headers": [],
        "retrieve_mode": "0",
        "request_method": "0",
        "output_format": "0",
        "ssl_cert_file": "",
        "ssl_key_file": "",
        "ssl_key_password": "",
        "verify_peer": "0",
        "verify_host": "0",
        "allow_traps": "0",
        "parameters": []
    },
    {
        "itemid": "23300",
        "type": "0",
        "snmp_oid": "",
        "hostid": "10084",
        "name": "CPU $2 time",
        "key_": "system.cpu.util[,interrupt]",
        "history": "7d",
        "trends": "365d",
        "lastvalue": "0.008389",
        "lastclock": "1351091000",
        "prevvalue": "0.000000",
        "state": "0",
        "status": "0",
        "value_type": "0",
        "trapper_hosts": "",
        "units": "%",
        "error": "",
        "logtimefmt": "",
        "templateid": "22671",
        "valuemapid": "0",
        "params": "",
        "ipmi_sensor": "",
        "authtype": "0",
        "username": "",
        "password": "",
        "publickey": "",
```

```
            "privatekey": "",
            "lastns": "564661387",
            "flags": "0",
            "interfaceid": "1",
            "description": "The amount of time the CPU has been servicing hardware interrupts.",
            "inventory_link": "0",
            "lifetime": "0s",
            "evaltype": "0",
            "jmx_endpoint": "",
            "master_itemid": "0",
            "timeout": "3s",
            "url": "",
            "query_fields": [],
            "posts": "",
            "status_codes": "200",
            "follow_redirects": "1",
            "post_type": "0",
            "http_proxy": "",
            "headers": [],
            "retrieve_mode": "0",
            "request_method": "0",
            "output_format": "0",
            "ssl_cert_file": "",
            "ssl_key_file": "",
            "ssl_key_password": "",
            "verify_peer": "0",
            "verify_host": "0",
            "allow_traps": "0",
            "parameters": []
        }
    ],
    "id": 1
}
```


按关键字查找依赖监控项

检索目标 ID 为 “10116” 的主机，包含关键字 “apache” 的所有依赖监控项。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "item.get",
    "params": {
        "output": "extend",
        "hostids": "10116",
        "search": {
            "key_": "apache"
        },
        "filter": {
            "type": "18"
        }
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：


```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "itemid": "25550",
            "type": "18",
```

```
        "snmp_oid": "",
        "hostid": "10116",
        "name": "Days",
        "key_": "apache.status.uptime.days",
        "delay": "",
        "history": "90d",
        "trends": "365d",
        "status": "0",
        "value_type": "3",
        "trapper_hosts": "",
        "units": "",
        "formula": "",
        "error": "",
        "logtimefmt": "",
        "templateid": "0",
        "valuemapid": "0",
        "params": "",
        "ipmi_sensor": "",
        "authtype": "0",
        "username": "",
        "password": "",
        "publickey": "",
        "privatekey": "",
        "flags": "0",
        "interfaceid": "0",
        "description": "",
        "inventory_link": "0",
        "lifetime": "30d",
        "state": "0",
        "evaltype": "0",
        "master_itemid": "25545",
        "jmx_endpoint": "",
        "timeout": "3s",
        "url": "",
        "query_fields": [],
        "posts": "",
        "status_codes": "200",
        "follow_redirects": "1",
        "post_type": "0",
        "http_proxy": "",
        "headers": [],
        "retrieve_mode": "0",
        "request_method": "0",
        "output_format": "0",
        "ssl_cert_file": "",
        "ssl_key_file": "",
        "ssl_key_password": "",
        "verify_peer": "0",
        "verify_host": "0",
        "allow_traps": "0",
        "lastclock": "0",
        "lastns": "0",
        "lastvalue": "0",
        "prevvalue": "0",
        "parameters": []
    },
    {
        "itemid": "25555",
        "type": "18",
        "snmp_oid": "",
        "hostid": "10116",
        "name": "Hours",
```

```
            "key_": "apache.status.uptime.hours",
            "delay": "0",
            "history": "90d",
            "trends": "365d",
            "status": "0",
            "value_type": "3",
            "trapper_hosts": "",
            "units": "",
            "formula": "",
            "error": "",
            "logtimefmt": "",
            "templateid": "0",
            "valuemapid": "0",
            "params": "",
            "ipmi_sensor": "",
            "authtype": "0",
            "username": "",
            "password": "",
            "publickey": "",
            "privatekey": "",
            "flags": "0",
            "interfaceid": "0",
            "description": "",
            "inventory_link": "0",
            "lifetime": "30d",
            "state": "0",
            "evaltype": "0",
            "master_itemid": "25545",
            "jmx_endpoint": "",
            "timeout": "3s",
            "url": "",
            "query_fields": [],
            "posts": "",
            "status_codes": "200",
            "follow_redirects": "1",
            "post_type": "0",
            "http_proxy": "",
            "headers": [],
            "retrieve_mode": "0",
            "request_method": "0",
            "output_format": "0",
            "ssl_cert_file": "",
            "ssl_key_file": "",
            "ssl_key_password": "",
            "verify_peer": "0",
            "verify_host": "0",
            "allow_traps": "0",
            "lastclock": "0",
            "lastns": "0",
            "lastvalue": "0",
            "prevvalue": "0",
            "parameters": []
        }
    ],
    "id": 1
}
```


查找 HTTP agent 监控项

查找具有特定主机 id 的 post 正文类型 XML 的 HTTP agent 监控项。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "item.get",
    "params": {
        "hostids": "10255",
        "filter": {
            "type": "19",
            "post_type": "3"
        }
    },
    "id": 3,
    "auth": "d678e0b85688ce578ff061bd29a20d3b"
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "itemid": "28252",
            "type": "19",
            "snmp_oid": "",
            "hostid": "10255",
            "name": "template item",
            "key_": "ti",
            "delay": "30s",
            "history": "90d",
            "trends": "365d",
            "status": "0",
            "value_type": "3",
            "trapper_hosts": "",
            "units": "",
            "formula": "",
            "error": "",
            "logtimefmt": "",
            "templateid": "0",
            "valuemapid": "0",
            "params": "",
            "ipmi_sensor": "",
            "authtype": "0",
            "username": "",
            "password": "",
            "publickey": "",
            "privatekey": "",
            "flags": "0",
            "interfaceid": "0",
            "description": "",
            "inventory_link": "0",
            "lifetime": "30d",
            "state": "0",
            "evaltype": "0",
            "jmx_endpoint": "",
            "master_itemid": "0",
            "timeout": "3s",
            "url": "localhost",
            "query_fields": [
                {
                    "mode": "xml"
                }
            ],
            "posts": "<body>\r\n<![CDATA[{$MACRO}<foo></bar>]]>\r\n</body>",
            "status_codes": "200",
```

```
            "follow_redirects": "0",
            "post_type": "3",
            "http_proxy": "",
            "headers": [],
            "retrieve_mode": "1",
            "request_method": "3",
            "output_format": "0",
            "ssl_cert_file": "",
            "ssl_key_file": "",
            "ssl_key_password": "",
            "verify_peer": "0",
            "verify_host": "0",
            "allow_traps": "0",
            "lastclock": "0",
            "lastns": "0",
            "lastvalue": "0",
            "prevvalue": "0",
            "parameters": []
        }
    ],
    "id": 3
}
```


检索预处理规则监控项

检索 ID 为 “10254” 的主机所有监控项及其预处理规则。

请求：


```
{
    "jsonrpc": "2.0",
    "method": "item.get",
    "params": {
        "output": ["itemid", "name", "key_"],
        "selectPreprocessing": "extend",
        "hostids": "10254"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```


响应：


```
{
    "jsonrpc": "2.0",
    "result": {
        "itemid": "23865",
        "name": "http agent example JSON",
        "key_": "json",
        "preprocessing": [
            {
                "type": "12",
                "params": "$.random",
                "error_handler": "1",
                "error_handler_params": ""
            }
        ]
    },
    "id": 1
}
```


另请参阅

- 发现规则
- 图表
- 主机

- 主机接口
- 触发器

源码

ui/include/classes/api/services/CItem.php - CItem::get()

监控项原型

此类用于管理监控项原型。

对象引用:

- 监控项原型

可用的方法:

- itemprototype.create 创建监控项原型
- itemprototype.delete 删除监控项原型
- itemprototype.get 搜索监控项原型
- itemprototype.update 更新监控项原型

> 监控项原型对象

以下对象与 监控项原型 API 直接相关。

监控项原型

监控项原型对象具有以下属性。

属性	类型	描述
itemid	string	*(只读)* 监控项原型的 ID。
delay (必需)	string	监控项原型的更新间隔。接受秒或带有后缀的时间单位（30s,1m,2h,1d）。 可选地，可以指定一个或多个自定义间隔，可以是灵活的间隔或调度。 多个间隔用分号分隔。 可以使用用户宏和 LLD 宏。单一宏必须填满整个字段。不支持字段中多个宏或宏与文本混合。 灵活的间隔可以写成两个宏，用正斜杠分隔（例如 `{$FLEX_INTERVAL}/{$FLEX_PERIOD}`）。 对于 Zabbix trapper，依赖项和具有 `mqtt.get` 键的 Zabbix agent（主动）是可选的。
hostid (必需)	string	监控项原型所属的主机 ID。 对于更新操作，此字段是只读的。
ruleid (必需)	string	监控项所属的 LLD 规则的 ID。 对于更新操作，此字段是只读的。
interfaceid (必需)	string	监控项原型的主机接口 ID。仅用于主机监控项原型。 对于 Zabbix agent（主动），Zabbix 内部，Zabbix trapper，计算，依赖，数据库监视器和脚本监控项原型不是必需的。对于 HTTP agent 监控项原型是可选的。
key_ (必需)	string	监控项原型的键。
name (必需)	string	监控项原型的名称。

属性	类型	描述
type (必需)	integer	监控项原型的类型。 可能的值： 0 - Zabbix agent； 2 - Zabbix trapper； 3 - 简单检查； 5 - Zabbix 内部； 7 - Zabbix agent（主动）； 10 - 外部检查； 11 - 数据库监视器； 12 - IPMI agent； 13 - SSH agent； 14 - TELNET agent； 15 - 计算； 16 - JMX agent； 17 - SNMP 陷阱； 18 - 依赖项； 19 - HTTP agent； 20 - SNMP agent； 21 - 脚本。
url (必需)	string	仅对 HTTP agent 监控项原型需要的 URL 字符串。支持 LLD 宏，用户宏，{HOST.IP}，{HOST.CONN}，{HOST.DNS}，{HOST.HOST}，{HOST.NAME}，{ITEM.ID}，{ITEM.KEY}。
value_type (必需)	integer	监控项原型的信息类型。 可能的值： 0 - 数值浮点； 1 - 字符； 2 - 日志； 3 - 数值无符号； 4 - 文本。
allow_traps	integer	HTTP agent 监控项原型字段。允许像 trapper 项类型一样填充值。 0 - *(默认)* 不允许接受传入数据。 1 - 允许接受传入数据。
authtype	integer	仅由 SSH agent 监控项原型或 HTTP agent 监控项原型使用。 SSH agent 认证方法可能的值： 0 - *(默认)* 密码； 1 - 公钥。 HTTP agent 认证方法可能的值： 0 - *(默认)* 无 1 - 基础 2 - NTLM 3 - Kerberos
description	string	监控项原型的描述。
follow_redirects	integer	HTTP agent 监控项原型字段。在轮询数据时跟随响应重定向。 0 - 不跟随重定向。 1 - *(默认)* 跟随重定向。
headers	object	HTTP agent 监控项原型字段。HTTP(S) 请求头的对象，其中头部名称用作键，头部值作为值。 示例： { ”User-Agent”: ”Zabbix” }
history	string	历史数据应该存储多长时间的时间单位。也接受用户宏和 LLD 宏。 默认值：90d。
http_proxy	string	HTTP agent 监控项原型字段。HTTP(S) agent 连接字符串。
ipmi_sensor	string	IPMI 传感器。仅由 IPMI 监控项原型使用。

属性	类型	描述
jmx_endpoint	string	JMX agent 自定义连接字符串。 默认值： service:jmx:rmi:///jndi/rmi://{HOST.CONN}:{HOST.PORT}/jmxrmi
logtimefmt	string	日志条目中的时间格式。仅由日志监控项原型使用。
master_itemid	integer	主项 ID。 允许递归最多 3 个依赖项和监控项原型，依赖项和监控项原型的最大数量等于 29999。 由依赖项需要。
output_format	integer	HTTP agent 监控项原型字段。响应是否应转换为 JSON。 0 - *(默认)* 存储原始数据。 1 - 转换为 JSON。
params	string	根据监控项原型的类型而变化的附加参数： - SSH 和 Telnet 监控项原型执行的脚本； - 数据库监视器监控项原型的 SQL 查询； - 计算监控项原型的公式。
parameters	array	脚本监控项原型的附加参数。具有'name' 和'value' 属性的对象数组，其中名称必须是唯一的。
password	string	认证密码。由简单检查，SSH，Telnet，数据库监视器，JMX 和 HTTP agent 监控项原型使用。 由 SSH 和 Telnet 监控项原型需要。
post_type	integer	HTTP agent 监控项原型字段。存储在 posts 属性中的 post 数据正文的类型。 0 - *(默认)* 原始数据。 2 - JSON 数据。 3 - XML 数据。
posts	string	HTTP agent 监控项原型字段。HTTP(S) 请求正文数据。与 post_type 一起使用。
privatekey	string	私钥文件的名称。
publickey	string	公钥文件的名称。
query_fields	array	HTTP agent 监控项原型字段。查询参数。具有'key':'value' 对的对象数组，其中值可以是空字符串。
request_method	integer	HTTP agent 监控项原型字段。请求方法的类型。 0 - *(默认)* GET 1 - POST 2 - PUT 3 - HEAD
retrieve_mode	integer	HTTP agent 监控项原型字段。应该存储响应的哪一部分。 0 - *(默认)* 正文。 1 - 头。 2 - 正文和头都将被存储。 对于 request_method HEAD，只允许值 1。
snmp_oid	string	SNMP OID。
ssl_cert_file	string	HTTP agent 监控项原型字段。公共 SSL 密钥文件路径。
ssl_key_file	string	HTTP agent 监控项原型字段。私有 SSL 密钥文件路径。
ssl_key_password	string	HTTP agent 监控项原型字段。SSL 密钥文件的密码。
status	integer	监控项原型的状态。 可能的值： 0 - *(默认)* 启用的监控项原型； 1 - 禁用的监控项原型； 3 - 不支持的监控项原型。

属性	类型	描述
status_codes	string	HTTP agent 监控项原型字段。所需的 HTTP 状态代码范围，用逗号分隔。 还支持用户宏或 LLD 宏作为逗号分隔列表的一部分。 示例：200,200-{$M},{$M},200-400
templateid	string	(只读) 父模板监控项原型的 ID。
timeout	string	监控项数据轮询请求超时。用于 HTTP agent 和脚本监控项原型。支持用户宏和 LLD 宏。 默认值：3s 最大值：60s
trapper_hosts	string	允许的主机。由 trapper 监控项原型或 HTTP 监控项原型使用。
trends	string	趋势数据应该存储多长时间的时间单位。也接受用户宏和 LLD 宏。 默认值：365d。
units	string	值单位。
username	string	认证用户名。由简单检查，SSH，Telnet，数据库监视器，JMX 和 HTTP agent 监控项原型使用。 由 SSH 和 Telnet 监控项原型需要。
uuid	string	通用唯一标识符，用于将导入的监控项原型与已有的原型进行关联。仅在模板上使用监控项原型时使用。如果未提供，则自动生成。
valuemapid	string	相关值映射的 ID。
verify_host	integer	HTTP agent 监控项原型字段。是否验证连接的主机名是否与主机证书中的名称匹配。 0 - *(默认)* 不进行验证。 1 - 进行验证。
verify_peer	integer	HTTP agent 监控项原型字段。是否验证主机的证书是否真实有效。 0 - *(默认)* 不进行验证。 1 - 进行验证。
discover	integer	监控项原型的发现状态。 可能的值： 0 - *(默认)* 将发现新监控项； 1 - 不会发现新监控项，并将现有监控项标记为丢失。

请注意，对于一些方法（更新，删除），所需/可选参数组合是不同的。

监控项原型标签

监控项原型标签对象具有以下属性。

属性	类型	描述
tag (必需)	string	监控项原型标签名字。
value	string	监控项原型标签值。

监控项原型预处理

监控项原型预处理对象具有以下属性。

属性	类型	描述
type (必需)	integer	预处理选项类型。 可能的值： 1 - 自定义乘数； 2 - 右修剪； 3 -左修剪; 4 - 修剪; 5 - 正则表达式匹配; 6 - 布尔转十进制; 7 - 八进制转十进制; 8 - 十六进制转十进制; 9 - 简单更改; 10 - 每秒更改; 11 - XML XPath; 12 - JSONPath; 13 - 在范围内; 14 - 匹配正则表达式; 15 -不匹配正则表达式； 16 - 检查 JSON 中的错误； 17 - 检查 XML 中的错误； 18 - 使用正则表达式检查错误； 19 - 丢弃未更改的内容； 20 - 丢弃未更改的心跳； 21 - JavaScript； 22 - Prometheus 模式； 23 - Prometheus 到 JSON； 24 - CSV 到 JSON； 25 - 替换； 26 - 检查不支持； 27 - XML 到 JSON。
params (必需)	string	预处理选项使用的附加参数。多个参数由 LF (\n) 字符分隔。
error_handler (必需)	integer	在预处理步骤失败的情况下使用的操作类型。 可能的值： 0 - 错误消息由 Zabbix server 设置； 1 - 丢弃值； 2 - 设置自定义值； 3 - 设置自定义错误消息。

属性	类型	描述
error_handler_params (必需)	string	错误处理程序参数。 与 `error_handler` 一起使用。 如果 `error_handler` 为 0 或 1，则必须为空。 如果 `error_handler` 为 2，则可以为空。 如果 `error_handler` 为 3，则不能为空。

每个都支持以下参数和错误处理程序预处理类型。

预处理类型	名称	参数 1	参数 2	参数 3	支持的错误处理程序
1	自定义乘数	number[1, 6]			0, 1, 2, 3
2	右修剪	list of characters[2]			
3	左修剪	list of characters[2]			
4	修剪	list of characters[2]			
5	正则表达式	pattern[3]	output[2]		0,1,2,3
6	布尔转十进制				0,1,2,3
7	八进制转十进制				0, 1, 2, 3
8	十六进制转十进制				0,1,2,3
9	简单更改				0,1,2,3
10	每秒变化				0,1,2,3
11	XML XPath	path[4]			0,1,2,3
12	JSONPath	path[4]			0, 1, 2, 3
13	在范围内	min[1,\6]	max[1,\6]		0,1,2,3
14	匹配正则表达式	pattern[3]			0,1,2,3
15	不匹配正则表达式	pattern[3]			0,1,2,3
16	检查 JSON 中的错误	path[4]			0, 1, 2, 3
17	检查 XML 中的错误	path[4]			0, 1, 2, 3
18	使用正则表达式检查错误	pattern[3]	output[2]		0,1,2,3
19	原封不动地丢弃				
20	用心跳不改变丢弃	seconds[5, 6]			
21	JavaScript	script[2]			
22	普罗米修斯模式	pattern[6, 7]	`value`, `label`, `function`	output[8, 9]	0, 1, 2, 3
23	Prometheus 转 JSON	pattern[6, 7]			0,1,2,3
24	CSV 转 JSON	character[2]	character[2]	0,1	0,1,2,3
25	替换	search string[2]	replacement[2]		
26	检查不支持				1,2,3
27	XML 转 JSON				0, 1, 2, 3

[1] 整数或浮点数
[2] 字符串
[3] 正则表达式
[4] JSONPath 或 XML XPath
[5] 正整数（支持时间后缀，例如 30s、1m、2h、1d）
[6] 用户宏，LLD 宏
[7] Prometheus 模式遵循以下语法：`<metric name>{<label name>="<label value>", ...} == <value>`。每个 Prometheus 模式组件（指标、标签名称、标签值和指标 value) 可以是用户宏或 LLD 宏。
[8] Prometheus 输出如下语法：`<label name>`（可以是用户宏或 LLD 宏）如果选择 `label` 作为第二个参数。
[9] 聚合函数之一：`sum`、`min`、`max`、`avg`、`count` 如果 `function` 被选为第二个范围。

创建监控项原型

描述

`object itemprototype.create(object/array itemPrototypes)`

该方法允许创建监控项原型。

> **Note:**
> 此方法仅适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。了解更多信息请参见用户角色

参数

(object/array) 需要创建的监控项原型。

除标准监控项原型外，该方法还接受以下参数。

参数	类型	描述
ruleid (required)	string	监控项所属的 LLD 规则 ID
preprocessing	array	监控项原型预处理 选项。
tags	array	监控项原型标签。

返回值

返回一个对象 （object），该对象包含在 itemids 属性下创建的监控项原型的 IDs。返回的 IDs 的顺序与传递的项目原型的顺序相匹配。

案例

创建监控项原型

创建一个监控项原型以监控发现的文件系统中的空闲磁盘空间。发现的监控项应该是数值类型的 Zabbix agent 监控项，每 30 秒更新一次。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "itemprototype.create",
    "params": {
        "name": "Free disk space on {#FSNAME}",
        "key_": "vfs.fs.size[{#FSNAME},free]",
        "hostid": "10197",
        "ruleid": "27665",
        "type": 0,
        "value_type": 3,
        "interfaceid": "112",
        "delay": "30s"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "27666"
        ]
    },
    "id": 1
}
```

创建监控项原型预处理

先创建一个监控项，第二步在预处理中使用每秒更改和自定义乘数。

请求:

```
{
 · "jsonrpc": "2.0",
 · "method": "itemprototype.create",
 · "params": {
 · "name": "Incoming network traffic on {#IFNAME}",
 · "key_": "net.if.in[{#IFNAME}]",
 · "hostid": "10001",
 · "ruleid": "27665",
 · "type": 0,
 · "value_type": 3,
 · "delay": "60s",
 · "units": "bps",
 · "interfaceid": "1155",
 · "preprocessing": [
 · {
 · "type": "10",
 · "params": "",
 · "error_handler": "0",
 · "error_handler_params": ""
 · },
 · {
 · "type": "1",
 · "params": "8",
 · "error_handler": "2",
 · "error_handler_params": "10"
 · }
 · ]
 · },
 · "auth": "038e1d7b1735c6a5436ee9eae095879e",
 · "id": 1
}
```

返回:

```
{
 · "jsonrpc": "2.0",
 · "result": {
 · "itemids": [
 · "44211"
 · ]
 · },
 · "id": 1
}
```

创建依赖监控项原型

创建依赖与 ID 44211 监控项原型的监控项原型。只允许依赖一些主机 (模板/发现规则)，因此主要监控项和依赖监控项应该具有相同的 hostid 和 ruleid。

请求:

```
{
 · "jsonrpc": "2.0",
 · "method": "itemprototype.create",
 · "params": {
 · "hostid": "10001",
 · "ruleid": "27665",
 · "name": "Dependent test item prototype",
 · "key_": "dependent.prototype",
 · "type": "18",
 · "master_itemid": "44211",
 · "value_type": "3"
 · },
 · "auth": "038e1d7b1735c6a5436ee9eae095879e",
 · "id": 1
```

```
}
```

返回:

```
{
 · "jsonrpc": "2.0",
 · "result": {
 · "itemids": [
 · "44212"
 · ]
 · },
 · "id": 1
}
```

创建 HTTP 代理监控项原型

使用用户宏、查询字段和自定义头部创建带有 URL 的监控项原型。

请求:

```
{
 · "jsonrpc": "2.0",
 · "method": "itemprototype.create",
 · "params": {
 · "type": "19",
 · "hostid": "10254",
 · "ruleid":"28256",
 · "interfaceid":"2",
 · "name": "api item prototype example",
 · "key_": "api_http_item",
 · "value_type": "3",
 · "url": "{$URL_PROTOTYPE}",
 · "query_fields": [
 · {
 · "min": "10"
 · },
 · {
 · "max": "100"
 · }
 · ],
 · "headers": {
 · "X-Source": "api"
 · },
 · "delay":"35"
 · },
 · "auth": "038e1d7b1735c6a5436ee9eae095879e",
 · "id": 1
}
```

返回:

```
{
 · "jsonrpc": "2.0",
 · "result": {
 · "itemids": [
 · "28305"
 · ]
 · },
 · "id": 1
}
```

创建监控项原型脚本

使用脚本监控项原型创建一个简单的数据集合。

请求:

```
{
 · "jsonrpc": "2.0",
 · "method": "itemprototype.create",
 · "params": {
 · "name": "Script example",
 · "key_": "custom.script.itemprototype",
 · "hostid": "12345",
 · "type": 21,
 · "value_type": 4,
 · "params": "var request = new CurlHttpRequest();\nreturn request.Post(\"https://postman-echo.com/post\",
 · "parameters": [{
 · "name": "host",
 · "value": "{HOST.CONN}"
 · }],
 · "timeout": "6s",
 · "delay": "30s"
 · },
 · "auth": "038e1d7b1735c6a5436ee9eae095879e",
 · "id": 2
}
```

返回:

```
{
 · "jsonrpc": "2.0",
 · "result": {
 · "itemids": [
 · "23865"
 · ]
 · },
 · "id": 3
}
```

来源

CItemPrototype::create() in *ui/include/classes/api/services/CItemPrototype.php*.

删除监控项原型

描述

`object itemprototype.delete(array itemPrototypeIds)`

该方法允许删除监控项原型。

> **Note:**
> 此方法仅适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。了解更多信息请参见用户角色

参数

删除可通过监控项原型的 ID（数组）。

返回值

(object) 返回一个对象，该对象包含 prototypeids 属性下已删除监控项原型的 ID。

案例

删除多个监控项原型

删除 2 个监控项原型。
如果主监控项或者监控项原型被删除，依赖监控项原型的监控项将被自动删除。

请求:

```
{
 · "jsonrpc": "2.0",
 · "method": "itemprototype.delete",
```

```
 · "params": [
 · "27352",
 · "27356"
 · ],
 · "auth": "3a57200802b24cda67c4e4010b50c065",
 · "id": 1
}
```

返回:

```
{
 · "jsonrpc": "2.0",
 · "result": {
 · "prototypeids": [
 · "27352",
 · "27356"
 · ]
 · },
 · "id": 1
}
```

来源

CItemPrototype::delete() in *ui/include/classes/api/services/CItemPrototype.php*.

更新监控项原型

描述

`object itemprototype.update(object/array itemPrototypes)`

此方法用于更新已存在的监控项原型。

> **Note:**
> 此方法仅适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。了解更多信息请参见用户角色。

参数

(object/array) 要更新的监控项的属性。

每个的监控项的 itemid 属性必须被定义，其他属性可选。只有被传递的属性才会更新，其他所有属性保持不变。

另外见标准监控项原型，此方法接受如下参数。

参数	类型	描述
applications	array	要替换当前应用的应用的 ID。
preprocessing	array	要替换的当前监控项预处理选项。

返回值

(object) 返回一个对象，其中包含 itemids 属性下更新的监控项原型的 ID。

案例

修改监控项原型接口

修改自动发现监控项的主机接口。

请求:

```
{
 · "jsonrpc": "2.0",
 · "method": "itemprototype.update",
 · "params": {
 · "itemid": "27428",
 · "interfaceid": "132"
 · },
```

```
 · "auth": "038e1d7b1735c6a5436ee9eae095879e",
 · "id": 1
}
```

返回:

```
{
 · "jsonrpc": "2.0",
 · "result": {
 · "itemids": [
 · "27428"
 · ]
 · },
 · "id": 1
}
```

更新依赖监控项原型

使用新主监控项原型 ID 更新依赖监控项原型。只允许依赖于同一主机（模板/发现规则）监控项原型，因此主监控项原型和从属监控项原型应具有相同的 hostid 和 ruleid。

请求:

```
{
 · "jsonrpc": "2.0",
 · "method": "itemprototype.update",
 · "params": {
 · "master_itemid": "25570",
 · "itemid": "189030"
 · },
 · "auth": "700ca65537074ec963db7efabda78259",
 · "id": 1
}
```

返回:

```
{
 · "jsonrpc": "2.0",
 · "result": {
 · "itemids": [
 · "189030"
 · ]
 · },
 · "id": 1
}
```

更新 HTTP 代理监控项原型

修改查询字段和移除所有自定义的头部。

请求:

```
{
 · "jsonrpc": "2.0",
 · "method": "itemprototype.update",
 · "params": {
 · "itemid":"28305",
 · "query_fields": [
 · {
 · "random": "qwertyuiopasdfghjklzxcvbnm"
 · }
 · ],
 · "headers": []
 · }
 · "auth": "700ca65537074ec963db7efabda78259",
 · "id": 1
}
```

返回:


```
{
 · "jsonrpc": "2.0",
 · "result": {
 · "itemids": [
 · "28305"
 · ]
 · },
 · "id": 1
}
```


更新监控项原型预处理选项

使用监控项原型预处理规则“自定义乘数”更新监控项。

请求:


```
{
 · "jsonrpc": "2.0",
 · "method": "itemprototype.update",
 · "params": {
 · "itemid": "44211",
 · "preprocessing": [
 · {
 · "type": "1",
 · "params": "4",
 · "error_handler": "2",
 · "error_handler_params": "5"
 · }
 · ]
 · },
 · "auth": "700ca65537074ec963db7efabda78259",
 · "id": 1
}
```


返回:


```
{
 · "jsonrpc": "2.0",
 · "result": {
 · "itemids": [
 · "44211"
 · ]
 · },
 · "id": 1
}
```


更新监控项原型脚本

使用不同的脚本更新监控项原型脚本，删除先前脚本使用的不必要参数。

请求:


```
{
 · "jsonrpc": "2.0",
 · "method": "itemprototype.update",
 · "params": {
 · "itemid": "23865",
 · "parameters": [],
 · "script": "Zabbix.Log(3, 'Log test');\nreturn 1;"
 · },
 · "auth": "700ca65537074ec963db7efabda78259",
 · "id": 1
}
```


返回:

```
{
 · "jsonrpc": "2.0",
 · "result": {
 · "itemids": [
 · "23865"
 · ]
 · },
 · "id": 1
}
```

来源

CItemPrototype::update() in *ui/include/classes/api/services/CItemPrototype.php*.

获取监控项原型

描述

`integer/array itemprototype.get(object parameters)`

该方法允许根据给定的参数检索监控项原型。

> **Note:**
> 任何类型的用户都可以使用此方法。权限可以在用户角色设置中撤消调用该方法。了解更多信息可以参见用户角色。

参数

`(object)` 定义所需输出的参数。

该方法支持以下参数：

参数	类型	描述
discoveryids	string/array	只返回属于给定 LLD 规则的监控项原型。
graphids	string/array	只返回在给定图表原型中使用的监控项原型。
hostids	string/array	只返回属于给定主机的监控项原型。
inherited	boolean	如果设置为 `true`，则只返回从模板继承的监控项原型。
itemids	string/array	只返回具有给定 ID 的监控项原型。
monitored	boolean	如果设置为 `true`，则只返回属于监控中主机的启用的监控项原型。
templated	boolean	如果设置为 `true`，则只返回属于模板的监控项原型。
templateids	string/array	只返回属于给定模板的监控项原型。
triggerids	string/array	只返回在给定触发器原型中使用的监控项原型。
selectDiscoveryRule	query	返回一个discoveryRule 属性，包含监控项原型所属的低级发现规则。
selectGraphs	query	返回一个graphs 属性，包含监控项原型使用的图表原型。 支持 `count`。
selectHosts	query	返回一个hosts 属性，包含监控项原型所属的主机数组。
selectTags	query	在tags 属性中返回监控项原型标签。
selectTriggers	query	返回一个triggers 属性，包含监控项原型使用的触发器原型。 支持 `count`。

参数	类型	描述
selectPreprocessing	query	返回一个preprocessing 属性，包含监控项预处理选项。 它具有以下属性： type - (string) 预处理选项类型： 1 - 自定义乘数； 2 - 右修剪； 3 - 左修剪； 4 - 修剪； 5 - 正则表达式匹配； 6 - 布尔值转十进制； 7 - 八进制转十进制； 8 - 十六进制转十进制； 9 - 简单变化； 10 - 每秒变化； 11 - XML XPath； 12 - JSONPath； 13 - 在范围； 14 - 匹配正则表达式； 15 - 不匹配正则表达式； 16 - 在 JSON 中检查错误； 17 - 在 XML 中检查错误； 18 - 使用正则表达式检查错误； 19 - 丢弃未更改的； 20 - 丢弃未更改且具有心跳的； 21 - JavaScript； 22 - Prometheus 模式； 23 - Prometheus 转 JSON； 24 - CSV 转 JSON； 25 - 替换； 26 - 检查不支持的值； 27 - XML 转 JSON。 params - (string) 预处理选项使用的附加参数。多个参数由 LF (\n) 字符分隔。 error_handler - (string) 预处理步骤失败时使用的操作类型： 0 - 由 Zabbix 服务器设置错误消息； 1 - 丢弃值； 2 - 设置自定义值； 3 - 设置自定义错误消息。 error_handler_params - (string) 错误处理参数。
selectValueMap	query	返回一个valuemap 属性，包含监控项原型的值映射。
filter	object	只返回完全匹配给定过滤器的结果。 接受一个数组，其中键是属性名称，值是单一值或要匹配的值数组。 支持额外的过滤器： host - 监控项原型所属主机的技术名称。
limitSelects	integer	限制子选择返回的记录数。 适用于以下子选择： selectGraphs - 结果将按 name 排序； selectTriggers - 结果将按 description 排序。
sortfield	string/array	根据给定属性对结果进行排序。 可能的值有：itemid, name, key_, delay, type 和 status。
countOutput	boolean	这些参数对所有 get 方法都是通用的，详细描述在参考说明。
editable	boolean	
excludeSearch	boolean	
limit	integer	
output	query	

参数	类型	描述
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

(integer/array) 返回其中之一：

- 一个对象数组；
- 如果使用 countOutput 参数，返回被检索对象的数量。

案例

返回低级别规则发现的监控项原型

返回所有低级别规则发现的监控项原型。

请求:

```
{
 · "jsonrpc": "2.0",
 · "method": "itemprototype.get",
 · "params": {
 · "output": "extend",
 · "discoveryids": "27426"
 · },
 · "auth": "038e1d7b1735c6a5436ee9eae095879e",
 · "id": 1
}
```

返回:

```
{
 · "jsonrpc": "2.0",
 · "result": [
 · {
 · "itemid": "23077",
 · "type": "0",
 · "snmp_oid": "",
 · "hostid": "10079",
 · "name": "Incoming network traffic on en0",
 · "key_": "net.if.in[en0]",
 · "delay": "1m",
 · "history": "1w",
 · "trends": "365d",
 · "status": "0",
 · "value_type": "3",
 · "trapper_hosts": "",
 · "units": "bps",
 · "formula": "",
 · "error": "",
 · "logtimefmt": "",
 · "templateid": "0",
 · "valuemapid": "0",
 · "params": "",
 · "ipmi_sensor": "",
 · "authtype": "0",
 · "username": "",
 · "password": "",
 · "publickey": "",
 · "privatekey": "",
 · "flags": "0",
```

```
· "interfaceid": "0",
· "description": "",
· "inventory_link": "0",
· "lifetime": "30d",
· "state": "0",
· "evaltype": "0",
· "jmx_endpoint": "",
· "master_itemid": "0",
· "timeout": "3s",
· "url": "",
· "query_fields": [],
· "posts": "",
· "status_codes": "200",
· "follow_redirects": "1",
· "post_type": "0",
· "http_proxy": "",
· "headers": [],
· "retrieve_mode": "0",
· "request_method": "0",
· "output_format": "0",
· "ssl_cert_file": "",
· "ssl_key_file": "",
· "ssl_key_password": "",
· "verify_peer": "0",
· "verify_host": "0",
· "allow_traps": "0",
· "lastclock": "0",
· "lastns": "0",
· "lastvalue": "0",
· "prevvalue": "0",
· "discover": "0",
· "parameters": []
· },
· {
· "itemid": "10010",
· "type": "0",
· "snmp_oid": "",
· "hostid": "10001",
· "name": "Processor load (1 min average per core)",
· "key_": "system.cpu.load[percpu,avg1]",
· "delay": "1m",
· "history": "1w",
· "trends": "365d",
· "status": "0",
· "value_type": "0",
· "trapper_hosts": "",
· "units": "",
· "formula": "",
· "error": "",
· "logtimefmt": "",
· "templateid": "0",
· "valuemapid": "0",
· "params": "",
· "ipmi_sensor": "",
· "authtype": "0",
· "username": "",
· "password": "",
· "publickey": "",
· "privatekey": "",
· "flags": "0",
· "interfaceid": "0",
· "description": "The processor load is calculated as system CPU load divided by number of CPU cores.",
```

```
 · "inventory_link": "0",
 · "lifetime": "0",
 · "state": "0",
 · "evaltype": "0",
 · "jmx_endpoint": "",
 · "master_itemid": "0",
 · "timeout": "3s",
 · "url": "",
 · "query_fields": [],
 · "posts": "",
 · "status_codes": "200",
 · "follow_redirects": "1",
 · "post_type": "0",
 · "http_proxy": "",
 · "headers": [],
 · "retrieve_mode": "0",
 · "request_method": "0",
 · "output_format": "0",
 · "ssl_cert_file": "",
 · "ssl_key_file": "",
 · "ssl_key_password": "",
 · "verify_peer": "0",
 · "verify_host": "0",
 · "allow_traps": "0",
 · "lastclock": "0",
 · "lastns": "0",
 · "lastvalue": "0",
 · "prevvalue": "0",
 · "discover": "0",
 · "parameters": []
 · }
 · ],
 · "id": 1
}
```


查找依赖监控项

查找监控项 ID “25545” 的依赖监控项。

请求:

```
{
 · "jsonrpc": "2.0",
 · "method": "item.get",
 · "params": {
 · "output": "extend",
 · "filter": {
 · "type": "18",
 · "master_itemid": "25545"
 · },
 · "limit": "1"
 · },
 · "auth": "038e1d7b1735c6a5436ee9eae095879e",
 · "id": 1
}
```

Response:


```
{
 · "jsonrpc": "2.0",
 · "result": [
 · {
 · "itemid": "25547",
 · "type": "18",
 · "snmp_oid": "",
```

```
        "hostid": "10116",
        "name": "Seconds",
        "key_": "apache.status.uptime.seconds",
        "delay": "0",
        "history": "90d",
        "trends": "365d",
        "status": "0",
        "value_type": "3",
        "trapper_hosts": "",
        "units": "",
        "formula": "",
        "error": "",
        "logtimefmt": "",
        "templateid": "0",
        "valuemapid": "0",
        "params": "",
        "ipmi_sensor": "",
        "authtype": "0",
        "username": "",
        "password": "",
        "publickey": "",
        "privatekey": "",
        "flags": "0",
        "interfaceid": "0",
        "description": "",
        "inventory_link": "0",
        "lifetime": "30d",
        "state": "0",
        "evaltype": "0",
        "master_itemid": "25545",
        "jmx_endpoint": "",
        "master_itemid": "0",
        "timeout": "3s",
        "url": "",
        "query_fields": [],
        "posts": "",
        "status_codes": "200",
        "follow_redirects": "1",
        "post_type": "0",
        "http_proxy": "",
        "headers": [],
        "retrieve_mode": "0",
        "request_method": "0",
        "output_format": "0",
        "ssl_cert_file": "",
        "ssl_key_file": "",
        "ssl_key_password": "",
        "verify_peer": "0",
        "verify_host": "0",
        "allow_traps": "0",
        "lastclock": "0",
        "lastns": "0",
        "lastvalue": "0",
        "prevvalue": "0",
        "discover": "0",
        "parameters": []
    }
    ],
    "id": 1
}
```


查找 HTTP 代理监控项原型

通过指定主机 id 的请求头部方法查找 HTTP 代理监控项原型。

请求:

```
{
 · "jsonrpc": "2.0",
 · "method": "itemprototype.get",
 · "params": {
 · "hostids": "10254",
 · "filter": {
 · "type": "19",
 · "request_method": "3"
 · }
 · },
 · "id": 17,
 · "auth": "d678e0b85688ce578ff061bd29a20d3b"
}
```

返回:

```
{
 · "jsonrpc": "2.0",
 · "result": [
 · {
 · "itemid": "28257",
 · "type": "19",
 · "snmp_oid": "",
 · "hostid": "10254",
 · "name": "discovered",
 · "key_": "item[{#INAME}]",
 · "delay": "{#IUPDATE}",
 · "history": "90d",
 · "trends": "30d",
 · "status": "0",
 · "value_type": "3",
 · "trapper_hosts": "",
 · "units": "",
 · "formula": "",
 · "error": "",
 · "logtimefmt": "",
 · "templateid": "28255",
 · "valuemapid": "0",
 · "params": "",
 · "ipmi_sensor": "",
 · "authtype": "0",
 · "username": "",
 · "password": "",
 · "publickey": "",
 · "privatekey": "",
 · "flags": "2",
 · "interfaceid": "2",
 · "description": "",
 · "inventory_link": "0",
 · "lifetime": "30d",
 · "state": "0",
 · "evaltype": "0",
 · "jmx_endpoint": "",
 · "master_itemid": "0",
 · "timeout": "3s",
 · "url": "{#IURL}",
 · "query_fields": [],
 · "posts": "",
 · "status_codes": "",
 · "follow_redirects": "0",
 · "post_type": "0",
```

```
        "http_proxy": "",
        "headers": [],
        "retrieve_mode": "0",
        "request_method": "3",
        "output_format": "0",
        "ssl_cert_file": "",
        "ssl_key_file": "",
        "ssl_key_password": "",
        "verify_peer": "0",
        "verify_host": "0",
        "allow_traps": "0",
        "discover": "0",
        "parameters": []
    }
    ],
    "id": 17
}
```

参见

- 主机
- 图形原型
- 触发器原型

来源

CItemPrototype::get() in *ui/include/classes/api/services/CItemPrototype.php*.

管家

该类被设计用于和管家一起工作。

对象参考:

- Housekeeping

可用的方法:

- housekeeping.get - 获取管家
- housekeeping.update - 更新管家

> 管家对象

以下对象与 housekeeping API 直接相关。

管家

设置对象具有以下属性：

属性	类型	描述
hk_events_mode	整型	为事件和警报开启内部管家 可能的值: 0 - 禁用; 1 - *(默认)* 开启。
hk_events_trigger	字符串	触发器数据存储期限，接受带后缀的秒和时间单位。 默认: 365d.

属性	类型	描述
hk_events_service	字符串	服务数据存储期限，接受带后缀的秒和时间单位。 默认: 1d.
hk_events_internal	字符串	内部数据存储期限，接受带后缀的秒和时间单位。 默认: 1d.
hk_events_discovery	字符串	网络发现数据存储期限，接受带后缀的秒和时间单位。 默认 1d.
hk_events_autoreg	字符串	自动注册数据存储期限，接受带后缀的秒和时间单位。 默认: 1d.
hk_services_mode	整型	为服务启用内部管理 可能的值: 0 - 禁用; 1 - *(默认)* 开启。
hk_services	字符串	服务数据存储期限，接受带后缀的秒和时间单位。 默认: 365d.
hk_audit_mode	整型	开启内部管理用于审计 可能的值: 0 - 禁用; 1 - *(默认)* 开启。
hk_audit	字符串	审计数据存储期限，接受带后缀的秒和时间单位。 默认: 365d.
hk_sessions_mode	整型	为会话启用内部管理。 可能的值: 0 - 禁用; 1 - *(默认)* 开启。
hk_sessions	字符串	会话数据存储期限。接受带后缀的秒和时间单位。 默认: 365d。
hk_history_mode	整型	为历史数据启用内部管理 可能的值: 0 - 禁用; 1 - *(默认)* 开启。

属性	类型	描述
hk_history_global	整型	覆盖监控项的历史数据期限 可能的值: 0 - 不要覆盖; 1 - *(默认)* 覆盖。
hk_history	字符串	历史数据存储期限。接受带后缀的秒和时间单位。 默认: 90d.
hk_trends_mode	整型	为趋势数据开启内部管理。 可能的值: 0 - 禁用; 1 - *(默认)* 开启。
hk_trends_global	整型	覆盖监控项的趋势数据期限 可能的值: 0 - 不要覆盖; 1 - *(默认)* 覆盖。
hk_trends	字符串	趋势数据存储期限。接受带后缀的秒和时间单位。 默认: 365d.
db_extension	字符串	*(只读)* 配置 DB 扩展标志，如果此标志设置为”timescaledb”，则服务器会更改器行为以进行内务处理和项目删除。
compression_status	整型	为历史和趋势数据开启 TimescaleDB 压缩。 可能的值: 0 - *(默认)* 关; 1 - 开。
compress_older	字符串	压缩早于指定时间段的历史和趋势记录，接受带后缀的秒和时间单位。 默认: 7d.
compression_availability	整型	*(只读)* 压缩可用性。 可能的值: 0 - 不可用; 1 - 可用。

更新

描述

`object housekeeping.update(object housekeeping)`

该方法允许更新存在的管家设置

Note:
该方法适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。有关详细信息，参阅用户角色。

参数

(object) 管家属性将被更新。

返回值

(array) 返回更新参数的名称数组

示例

请求:

```
{
    "jsonrpc": "2.0",
    "method": "housekeeping.update",
    "params": {
        "hk_events_mode": "1",
        "hk_events_trigger": "200d",
        "hk_events_internal": "2d",
        "hk_events_discovery": "2d"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        "hk_events_mode",
        "hk_events_trigger",
        "hk_events_internal",
        "hk_events_discovery"
    ],
    "id": 1
}
```

来源

CHousekeeping::update() in *ui/include/classes/api/services/CHousekeeping.php*.

查询

描述

`object housekeeping.get(object parameters)`

该方法允许按照给定的参数获取管家对象

Note:
该方法适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。有关详细信息，参阅用户角色。

参数

(object) 定义所需输出的参数

该方法仅支持一个参数

参数	类型	描述
output	查询	该参数对于参考注释中所有 get 方法都适用。

返回值

(object) 返回管家对象

示例

请求:

```
{
    "jsonrpc": "2.0",
    "method": "housekeeping.get",
    "params": {
        "output": "extend"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "hk_events_mode": "1",
        "hk_events_trigger": "365d",
        "hk_events_service": "1d",
        "hk_events_internal": "1d",
        "hk_events_discovery": "1d",
        "hk_events_autoreg": "1d",
        "hk_services_mode": "1",
        "hk_services": "365d",
        "hk_audit_mode": "1",
        "hk_audit": "365d",
        "hk_sessions_mode": "1",
        "hk_sessions": "365d",
        "hk_history_mode": "1",
        "hk_history_global": "0",
        "hk_history": "90d",
        "hk_trends_mode": "1",
        "hk_trends_global": "0",
        "hk_trends": "365d",
        "db_extension": "",
        "compression_status": "0",
        "compress_older": "7d"
    },
    "id": 1
}
```

来源

CHousekeeping ::get() in *ui/include/classes/api/services/CHousekeeping.php*.

维护

此类用于维护. 对象引用:

- 维护
- 时间段

可用方法:

- maintenance.create创建维护
- maintenance.delete 删除维护
- maintenance.get检索维护
- maintenance.update更新维护

> 维护模式对象

如下对象与维护模式 API 关联。

维护

维护对象有如下属性。

属性	类型	描述
maintenanceid	string	*(只读)* 维护周期的 ID。
name (必需)	string	维护周期的名称。
active_since (必需)	timestamp	维护周期生效的开始时间。
active_till (必需)	timestamp	维护周期结束时间。
description	string	维护周期说明。
maintenance_type	integer	维护周期类型. 可选值: 0 - *(默认)* 收集数据; 1 - 不收集数据.
tags_evaltype	integer	问题标签多条件逻辑. 可选值: 0 - *(默认)* And/Or; 2 - Or.

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

时间段

时间段对象用于定义维护必须生效的时间段。它具有以下属性。

属性	类型	说明
period	integer	维护周期的持续时间，以秒为单位。 给定的值将向下舍入为分钟。 默认值：3600。
timeperiod_type	integer	时间段的类型。 可能的值： 0 - （默认）仅一次； 2 - 每天； 3 - 每周； 4 - 每月一次。
start_date	timestamp	维护期必须生效的日期。 仅用于一个时间段。 给定值将四舍五入为分钟。 默认值：当前日期。

属性	类型	说明
start_time	integer	一天中开始维护的时间，以秒为单位。
every	integer	用于每日、每周和每月的周期。 给定值将向下舍入为分钟。 默认值：0。 用于每日、每周和每月周期。 对于每日和每周周期，`every` 定义维护必须生效的天或周间隔。 默认值：1. 对于月度周期，如果 `dayofweek` 属性包含至少一个选定的星期几，则 `every` 属性定义维护必须生效的月份中的星期几。
dayofweek	integer	 可能的值： 1 - （默认值）第一周； 2 - 第二周； 3 - 第三周； 4 - 第四周； 5 - 上周。 维护必须生效的星期几。 天以二进制形式存储，每个位代表对应的日期。例如，4 等于二进制的 100，表示维护将在星期三启用。
day	integer	用于每周和每月的时间段。仅对于每周时间段是必需的。 必须为每月时间段指定至少一个“dayofweek”或“day”。 维护必须生效的月份中的哪一天。 仅用于每月时间段。 必须至少指定一个 `dayofweek` 或 `day` 每月时间段。

属性	类型	说明
month	integer	维护必须生效的月份。 月份以二进制形式存储，每个位代表对应的月份。例如，5 等于二进制的 101，表示维护将在 1 月和 3 月启用。 仅在每月的时间段内需要。

问题标签

问题标签对象用于定义维护生效时必须抑制哪些问题。它具有以下属性。

属性	类型	描述
tag (必需)	string	问题标签名字。
operator	integer	条件运算符。 可能的值: 0 - 等于; 2 - *(默认值)* 包含。
value	string	问题标签值。

创建维护模式

说明

`object maintenance.create(object/array maintenances)`

该方法允许创建维护模式。

Note:
此方法仅适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。了解更多信息请参见用户角色。

参数

`(object/array)` 要创建的维护模式。

另外见标准维护属性，此方法接受如下参数。

参数	类型	描述
groups	对象/数组	主机组 将进行维护。 主机组必须定义 “groupid” 属性。
hosts	对象/数组	< br> 必须指定至少一个 主机组或 主机对象。 主机 将进行维护。 主机必须定义 “hostid” 属性。
时间段 (必需)	对象/数组	至少必须指定 主机组或 主机中的一个对象。 维护时间段。
tags	对象/数组	问题标签。 定义必须抑制的问题。 如果没有给出标签，则所有活动的维护主机问题都将被抑制。

参数	类型	描述
groupids (已弃用)	数组	此参数已弃用，请改用 “groups”。 将进行维护的主机组的 ID。
hostids (已弃用)	数组	此参数已弃用，请改用 “hosts”。 将进行维护的主机的 ID。

返回值

(object) 在 maintenanceids 属性中返回一个包含所有已被创建的维护模式的对象的 ID。返回的 IDs 的排序与传递的维护模式的 IDs 顺序一致。

案例

创建维护模式

为 ID 为 “2” 的主机组创建问题标签为 **service:mysqld** 和 **error** 的数据收集维护模式。它必须从 22.01.2013 到 22.01.2014 有效，每周日 18:00 生效，持续一小时。

请求:

```
{
 · "jsonrpc": "2.0",
 · "method": "maintenance.create",
 · "params": {
 · "name": "Sunday maintenance",
 · "active_since": 1358844540,
 · "active_till": 1390466940,
 · "tags_evaltype": 0,
 · "groups": [
 · {"groupid": "2"}
 · ],
 · "timeperiods": [
 · {
 · "period": 3600,
 · "timeperiod_type": 3,
 · "start_time": 64800,
 · "every": 1,
 · "dayofweek": 64
 · }
 · ],
 · "tags": [
 · {
 · "tag": "service",
 · "operator": "0",
 · "value": "mysqld"
 · },
 · {
 · "tag": "error",
 · "operator": "2",
 · "value": ""
 · }
 · ]
 · },
 · "auth": "038e1d7b1735c6a5436ee9eae095879e",
 · "id": 1
}
```

返回:

```
{
 · "jsonrpc": "2.0",
 · "result": {
 · "maintenanceids": [
 · "3"
 · ]
```

```
    },
    "id": 1
}
```

另见

- Time period

来源

CMaintenance::create() in *ui/include/classes/api/services/CMaintenance.php*.

删除维护模式

描述

`object maintenance.delete(array maintenanceIds)`

此方法允许删除维护周期。

此方法仅适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。请参阅用户角色了解更多信息。

参数

(array) 要删除的维护周期的 IDs。

返回值

(object) 在 maintenanceids 属性下返回包含已被删除的维护周期的 ID 对象。

案例

删除多个维护周期

删除 2 个维护周期。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "maintenance.delete",
    "params": [
        "3",
        "1"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "maintenanceids": [
            "3",
            "1"
        ]
    },
    "id": 1
}
```

来源

CMaintenance::delete() in *frontends/php/include/classes/api/services/CMaintenance.php*.

更新维护期设置

描述

`object maintenance.update(object/array maintenances)`

此方法允许更新已存在的维护模式。

> **Note:**
> 此方法仅适用于管理员和超级管理员用户类型。可以在用户角色设置中撤销调用该方法的权限。了解更多信息请参见用户角色。

参数

(object/array) 要更新的维护模式的属性。

每一个维护模式的 maintenanceid 属性必须被定义，其他所有属性均为可选。只有被传递的属性才会被更新，所有它属性保持不变。

另外参见标准维护属性, 此方法接受如下参数。

参数	类型	描述
groups	object/array	主机主机组 以替换当前组。 主机组必须定义 groupid 属性。
hosts	object/array	主机 以替换当前主机。 主机必须定义 hostid 属性。
timeperiods	object/array	维护时间段 替换当前周期。
tags	object/array	问题标签 替换当前标签。
groupids（已弃用）	array	此参数已弃用，请改用 “groups”。 将进行维护的主机组的 ID。
hostids（已弃用）	array	此参数已弃用，请改用 “hosts”。 将进行维护的主机的 ID。

每一个维护模式至少一个主机或者一个主机组被定义。

返回值

(object) 在 maintenanceids 属性中返回一个包含已被更新的维护模式的 IDs 的对象。

案例

指定不同的主机

用两个不同的主机替换当前分配给维护周期 “3” 的主机。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "maintenance.update",
    "params": {
        "maintenanceid": "3",
        "hostids": [
            "10085",
            "10084"
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "maintenanceids": [
            "3"
        ]
    },
    "id": 1
}
```

参见

- 时间段

来源

CMaintenance::update() in *ui/include/classes/api/services/CMaintenance.php*.

获取维护期

描述

`integer/array maintenance.get(object parameters)`

此方法用于根据给定参数获取维护模式。

Note:
任何类型的用户都可以使用此方法。权限可以在用户角色设置中撤消调用该方法。了解更多信息可以参见用户角色。

参数

`(object)` 定义所需输出的参数。

该方法支持以下参数。

参数	类型	说明
groupids	string/array	只返回分配给给定主机组的维护。
hostids	string/array	仅返回分配给给定主机的维护。
maintenanceids	string/array	只返回给定 ID 的维护。
selectGroups	query	返回一个groups 属性，其中主机组分配给维护。
selectHosts	query	返回一个hosts 属性，其中主机分配给维护。
selectTags	query	返回带有维护问题标签的tags 属性。
selectTimeperiods	query	返回带有维护时间段的timeperiods 属性。
sortfield	string/array	按给定属性对结果进行排序。 可用值：`maintenanceid`、`name` 和 `maintenance_type`。
countOutput	boolean	这些参数对所有 `get` 方法都是通用的，在参考评论中有详细描述。
可编辑	boolean	
排除搜索	boolean	
过滤器	object	
限制	integer	
输出	query	
preservekeys	boolean	
搜索	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
排序	string/array	
开始搜索	boolean	

返回值

(整型/数组) 返回其中之一：- 一个对象数组；- 如果使用 `countOutput` 参数，被检索对象的数量。

案例

检索维护周期

检索所有已配置的维护，以及有关分配的主机组、定义的时间段和问题标签的数据。请求:

```
{
    "jsonrpc": "2.0",
    "method": "maintenance.get",
    "params": {
        "output": "extend",
        "selectGroups": "extend",
        "selectTimeperiods": "extend",
        "selectTags": "extend"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "maintenanceid": "3",
            "name": "Sunday maintenance",
            "maintenance_type": "0",
            "description": "",
            "active_since": "1358844540",
            "active_till": "1390466940",
            "tags_evaltype": "0",
            "groups": [
                {
                    "groupid": "4",
                    "name": "Zabbix servers",
                    "internal": "0"
                }
            ],
            "timeperiods": [
                {
                    "timeperiod_type": "3",
                    "every": "1",
                    "month": "0",
                    "dayofweek": "1",
                    "day": "0",
                    "start_time": "64800",
                    "period": "3600",
                    "start_date": "2147483647"
                }
            ],
            "tags": [
                {
                    "tag": "service",
                    "operator": "0",
                    "value": "mysqld",
                },
                {
                    "tag": "error",
                    "operator": "2",
                    "value": ""
                }
            ]
        }
    ],
    "id": 1
}
```

参见

- 主机
- 主机组
- 时间段

来源

CMaintenance::get() 在 *ui/include/classes/api/services/CMaintenance.php*.

脚本

此类为使用脚本而设计。参考对象:

- 脚本
- Webhook 参数
- 调试
- 日志条目

可用方法:

- script.create - 创建新脚本
- script.delete - 删除脚本
- script.execute - 运行脚本
- script.get - 检索脚本
- script.getscriptsbyhosts - 通过主机检索脚本
- script.update - 更新脚本

> 脚本对象

以下对象与 脚本 API 直接相关。

脚本

脚本对象有以下参数。

属性	类型	描述
scriptid	string	*(只读)* 脚本 ID。
name (必需)	string	脚本名称。
type (必需)	integer	脚本类型。 可能的值： 0 - 脚本； 1 - IPMI； 2 - SSH； 3 - Telnet； 5 - *(默认)* Webhook。
command (必需)	string	运行的命令。
scope	integer	脚本范围。 可能的值： 1 - 默认动作操作； 2 - 手动主机动作； 4 - 手动事件动作。

属性	类型	描述
execute_on	integer	在哪里运行脚本。 当 type 为 0 时使用（脚本）。 可能的值： 0 - 在 Zabbix agent 上运行； 1 - 在 Zabbix server 上运行; 2 - (默认) 在 Zabbix server (proxy) 上运行。
menu_path	string	当点击主机或事件时，由斜杠分隔的文件夹所组成的类似于前端导航的菜单。 当 scope 为 2 或 4 时使用。
authtype	integer	SSH 脚本类型使用的身份验证方法。 当 type 为 2 时使用。 可能的值： 0 - 密码； 1 - 公钥。
username	string	身份验证使用的用户名 当 type 为 2 或 3 时需要。
password	string	通过密码进行身份验证的 SSH 脚本和 Telnet 脚本使用的密码。 当 type 为 2 且 authtype 为 0 或 type 为 3 时使用。
publickey	string	通过公钥进行身份验证的 SSH 脚本使用的公钥文件名。 当 type 为 2 且 authtype 为 1 时需要。
privatekey	string	通过公钥进行身份验证的 SSH 脚本使用的私钥文件名。 当 type 为 2 且 authtype 为 1 时需要。
port	string	SSH 和 Telnet 脚本使用的端口号。 当 type 为 2 或 3 时使用。
groupid	string	可以运行脚本的主机群组 ID。如果设置为 0，脚本将可以在所有主机群组运行。 默认值：0。

属性	类型	描述
usrgrpid	string	允许运行脚本的用户群组 ID。如果设置为 0, 脚本将可以在所有用户群组运行。 当 scope 为 2 或 4 时使用。
host_access	integer	默认值: 0. 运行脚本所需的主机权限。 当 scope 为 2 或 4 时使用。 可能的值： 2 - (默认) 读; 3 - 写.
confirmation	string	弹出窗口的确认文本。如果尝试在 zabbix 前端运行脚本，将会弹出窗口。 当 scope 为 2 或 4 时使用。
timeout	string	Webhook 脚本执行超时秒数。支持时间后缀, 例如 30s，1m。 当 type 为 5 时需要。 可能的值： 1-60s 默认值： 30s
parameters	array	webhook 入参数组. 当 type 为 5 时使用。
description	string	脚本描述。

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

Webhook 参数

webhook 脚本运行时被传入的参数有如下属性。| 属性 |类型| 描述 | |--------|----------|----------| |**name**
(必需)|string| 参数名称。| |value|string| 参数值。支持宏。|

调试

运行的 webhook 脚本的调试信息。调试对象有如下属性。| 属性 |类型| 描述 | |--------|----------|----------| |logs|array|日志条目数组 (/manual/api/reference/script/object# 日志条目)。| |ms|string| 脚本运行毫秒数。|

日志条目

日志条目对象有如下属性。| 属性 |类型| 描述 | |--------|----------|----------| |level|integer| 日志等级。| |ms|string| 从脚本开始运行到添加日志条目前经过时间（毫秒）。| |message|string| 日志信息。|

创建

描述

```
object script.create(object/array scripts)
```

此方法允许创建新脚本。

> **Note:**
> 此方法只有超级管理员用户可以使用。可以在用户角色设置中撤销调用此方法的权限。更多信息见User roles。

参数

（对象/数组）要创建的脚本。

此方法接受具有标准脚本属性的脚本。

返回值

（对象）返回一个 `scriptids` 属性包含被创建脚本 ID 的对象。返回的 ID 顺序与传入脚本的顺序一致。

示例

创建 webhook 脚本

创建一个 webhook 脚本向外部服务发送 HTTP 请求。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "script.create",
    "params": {
        "name": "Webhook script",
        "command": "try {\n var request = new HttpRequest(),\n response,\n data;\n\n request.addHeader('Co
        "type": 5,
        "timeout": "40s",
        "parameters": [
            {
                "name": "token",
                "value": "{$WEBHOOK.TOKEN}"
            },
            {
                "name": "host",
                "value": "{HOST.HOST}"
            },
            {
                "name": "v",
                "value": "2.2"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "scriptids": [
            "3"
        ]
    },
    "id": 1
}
```

创建 SSH 脚本

创建一个 SSH 脚本，通过公钥完成身份验证从而可以在主机运行，并且带有上下文菜单。

请求：

```
{
    "jsonrpc": "2.0",
```

```
    "method": "script.create",
    "params": {
        "name": "SSH script",
        "command": "my script command",
        "type": 2,
        "username": "John",
        "publickey": "pub.key",
        "privatekey": "priv.key",
        "password": "secret",
        "port": "12345",
        "scope": 2,
        "menu_path": "All scripts/SSH",
        "usrgrpid": "7",
        "groupid": "4"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "scriptids": [
            "5"
        ]
    },
    "id": 1
}
```

创建定制脚本

创建一个重启服务器的定制脚本。该脚本会要求主机的写权限，并且在前端运行之前会显示配置信息。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "script.create",
    "params": {
        "name": "Reboot server",
        "command": "reboot server 1",
        "confirmation": "Are you sure you would like to reboot the server?",
        "scope": 2,
        "type": 0
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "scriptids": [
            "4"
        ]
    },
    "id": 1
}
```

源代码

CScript::create() 在 *ui/include/classes/api/services/CScript.php*。

删除

描述

`object script.delete(array scriptIds)`

此方法允许删除脚本。

> **Note:**
> 此方法只有超级管理员用户可以使用。可以在用户角色设置中撤销调用此方法的权限。更多信息见User roles。

参数

（数组）要删除的脚本 ID。

返回值

（对象）返回一个 `scriptids` 属性包含被删除脚本 ID 的对象。

示例

删除多个脚本

删除两个脚本

请求：

```
{
    "jsonrpc": "2.0",
    "method": "script.delete",
    "params": [
        "3",
        "4"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "scriptids": [
            "3",
            "4"
        ]
    },
    "id": 1
}
```

源代码

CScript::delete() 在 *frontends/php/include/classes/api/services/CScript.php*。

执行

描述

`object script.execute(object parameters)` 此方法允许在某主机或事件上运行脚本。

> **Note:**
> 此方法允许任何用户使用。可以在用户角色设置中撤销调用此方法的权限。更多信息见User roles。

参数

（对象）参数包含了运行脚本的 ID，以及主机 ID 或者事件 ID。

参数	类型	描述
scriptid (必需)	string	运行脚本的 ID。
hostid	string	在其上运行脚本的主机 ID。
eventid	string	在其上运行脚本的事件 ID。

返回值

(object) 返回脚本执行的结果。

属性	类型	描述
response	string	脚本是否成功运行。 可能的值 - success 或 failed。
值	字符串	脚本输出。
debug	object	如果执行 webhook 脚本，则包含调试对象。对于其他脚本类型，它包含空对象。

示例

运行 webhook 脚本

运行一个向外部服务发送 HTTP 请求的 webhook 脚本。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "script.execute",
    "params": {
        "scriptid": "4",
        "hostid": "30079"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "response": "success",
        "value": "{\"status\":\"sent\",\"timestamp\":\"1611235391\"}",
        "debug": {
            "logs": [
                {
                    "level": 3,
                    "ms": 480,
                    "message": "[Webhook Script] HTTP status: 200."
                }
            ],
            "ms": 495
        }
    },
    "id": 1
}
```

运行自定义脚本

在主机上运行 “ping” 脚本。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "script.execute",
```

```
    "params": {
        "scriptid": "1",
        "hostid": "30079"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "response": "success",
        "value": "PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.\n64 bytes from 127.0.0.1: icmp_req=1 tt
        "debug": []
    },
    "id": 1
}
```

源代码

CScript::execute() 在 *ui/include/classes/api/services/CScript.php*。

更新

描述

`object script.update(object/array scripts)` 此方法允许更新已存在的脚本。

> **Note:**
> 此方法只有超级管理员用户可以使用。可以在用户角色设置中撤销调用此方法的权限。更多信息见User roles

参数

`(object/array)` 待更新的脚本属性 。所有脚本必须定义 `scriptid` 属性，其他所有属性都是可选的。只有被传递的属性会被更新，其他所有属性保持不变。例外是 `type` 属性会从 5（Webhook）变成其他：`parameters` 属性会被清除。

返回值

（对象）返回一个 `scriptids` 属性包含被更新脚本 ID 的对象。

示例

修改脚本命令

把脚本命令修改为 “/bin/ping -c 10 {HOST.CONN} 2>&1”。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "script.update",
    "params": {
        "scriptid": "1",
        "command": "/bin/ping -c 10 {HOST.CONN} 2>&1"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "scriptids": [
            "1"
```

```
        ]
    },
    "id": 1
}
```

源代码

CScript::update() 在 *frontends/php/include/classes/api/services/CScript.php*。

获取

描述

`integer/array script.get(object parameters)` 此方法允许根据给定参数检索脚本。

Note:
此方法允许任何用户使用。可以在用户角色设置中撤销调用此方法的权限。更多信息见User roles。

参数

（对象）定义所需输出的参数。

此方法支持如下参数。

参数	类型	描述
groupids	string/array	返回可以在给定主机组上运行的脚本
hostids	string/array	返回可以在给定主机上运行的脚本。
scriptids	string/array	返回具有给定 ID 的脚本。
usrgrpids	string/array	返回可以被给定用户组中用户运行的脚本
selectGroups	query	返回可以在其上运行脚本的主机群组的群组 属性。
selectHosts	query	返回可以在其上运行脚本的主机的主机属性。
selectActions	query	返回与脚本相关联的动作的动作属性。
sortfield	string/array	根据给定属性将结果排序。 可能的值: `scriptid` 和 `name`.
countOutput	boolean	这些参数与所有 `get` 方法相同，详细描述见reference commentary.
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

（整型/数组）返回其中之一：- 一组对象; - 如果用到 `countOutput` 参数，被检索到的对象数,。

示例

检索所有脚本

检索所有配置的脚本。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "script.get",
    "params": {
        "output": "extend"
    },
```

```
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：


```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "scriptid": "1",
            "name": "Ping",
            "command": "/bin/ping -c 3 {HOST.CONN} 2>&1",
            "host_access": "2",
            "usrgrpid": "0",
            "groupid": "0",
            "description": "",
            "confirmation": "",
            "type": "0",
            "execute_on": "1",
            "timeout": "30s",
            "parameters": []
        },
        {
            "scriptid": "2",
            "name": "Traceroute",
            "command": "/usr/bin/traceroute {HOST.CONN} 2>&1",
            "host_access": "2",
            "usrgrpid": "0",
            "groupid": "0",
            "description": "",
            "confirmation": "",
            "type": "0",
            "execute_on": "1",
            "timeout": "30s",
            "parameters": []
        },
        {
            "scriptid": "3",
            "name": "Detect operating system",
            "command": "sudo /usr/bin/nmap -O {HOST.CONN} 2>&1",
            "host_access": "2",
            "usrgrpid": "7",
            "groupid": "0",
            "description": "",
            "confirmation": "",
            "type": "0",
            "execute_on": "1",
            "timeout": "30s",
            "parameters": []
        },
        {
            "scriptid": "4",
            "name": "Webhook",
            "command": "try {\n var request = new HttpRequest(),\n response,\n data;\n\n request.addHeader
            "host_access": "2",
            "usrgrpid": "7",
            "groupid": "0",
            "description": "",
            "confirmation": "",
            "type": "5",
            "execute_on": "1",
            "timeout": "30s",
```

```
            "parameters": [
                {
                    "name": "token",
                    "value": "{$WEBHOOK.TOKEN}"
                },
                {
                    "name": "host",
                    "value": "{HOST.HOST}"
                },
                {
                    "name": "v",
                    "value": "2.2"
                }
            ]
        }
    ],
    "id": 1
}
```

另外参考

- 主机
- 主机组

源代码

CScript::get() 在 *frontends/php/include/classes/api/services/CScript.php*.

通过主机获取脚本

描述

`object script.getscriptsbyhosts(array hostIds)` 此方法允许检索特定主机可用的脚本。

> **Note:**
> 此方法允许任何用户使用。可以在用户角色设置中撤销调用此方法的权限。更多信息见User roles。

参数

（字符串/数组）用于检索脚本的主机 ID。

返回值

（对象）返回一个以主机 ID 为属性、以可用脚本数组为值的对象。::: notetip 此方法会自动拓展 `confirmation` 中的宏.

示例

通过主机 ID 检索脚本

检索主机 “30079” 和 “30073” 上所有脚本。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "script.getscriptsbyhosts",
    "params": [
        "30079",
        "30073"
    ],
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
```

```
        "30079": [
            {
                "scriptid": "3",
                "name": "Detect operating system",
                "command": "sudo /usr/bin/nmap -O {HOST.CONN} 2>&1",
                "host_access": "2",
                "usrgrpid": "7",
                "groupid": "0",
                "description": "",
                "confirmation": "",
                "type": "0",
                "execute_on": "1",
                "hostid": "10001"
            },
            {
                "scriptid": "1",
                "name": "Ping",
                "command": "/bin/ping -c 3 {HOST.CONN} 2>&1",
                "host_access": "2",
                "usrgrpid": "0",
                "groupid": "0",
                "description": "",
                "confirmation": "",
                "type": "0",
                "execute_on": "1",
                "hostid": "10001"
            },
            {
                "scriptid": "2",
                "name": "Traceroute",
                "command": "/usr/bin/traceroute {HOST.CONN} 2>&1",
                "host_access": "2",
                "usrgrpid": "0",
                "groupid": "0",
                "description": "",
                "confirmation": "",
                "type": "0",
                "execute_on": "1",
                "hostid": "10001"
            }
        ],
        "30073": [
            {
                "scriptid": "3",
                "name": "Detect operating system",
                "command": "sudo /usr/bin/nmap -O {HOST.CONN} 2>&1",
                "host_access": "2",
                "usrgrpid": "7",
                "groupid": "0",
                "description": "",
                "confirmation": "",
                "type": "0",
                "execute_on": "1",
                "hostid": "10001"
            },
            {
                "scriptid": "1",
                "name": "Ping",
                "command": "/bin/ping -c 3 {HOST.CONN} 2>&1",
                "host_access": "2",
                "usrgrpid": "0",
                "groupid": "0",
```

```
                "description": "",
                "confirmation": "",
                "type": "0",
                "execute_on": "1",
                "hostid": "10001"
            },
            {
                "scriptid": "2",
                "name": "Traceroute",
                "command": "/usr/bin/traceroute {HOST.CONN} 2>&1",
                "host_access": "2",
                "usrgrpid": "0",
                "groupid": "0",
                "description": "",
                "confirmation": "",
                "type": "0",
                "execute_on": "1",
                "hostid": "10001"
            }
        ]
    },
    "id": 1
}
```


源代码

CScript::getScriptsByHosts() 在 *frontends/php/include/classes/api/services/CScript.php*。

自动注册

该方法用于自动注册

对象引用:

- 自动注册

可用方法:

- autoregistration.get - 检索自动注册
- autoregistration.update - 更新自动注册

> 自动注册对象

以下对象与 自动注册 API 直接相关。

自动注册

自动注册对象具有以下属性。

属性	类型	描述
tls_accept	integer	自动注册允许的传入连接的类型。 可用值： 1 - 允许不安全的连接； 2 - 允许用 PSK 连接 TLS； 3 - 允许不安全的连接以及通过 PSK 连接 TLS 两种方式.

属性	类型	描述
tls_psk_identity	string	*(只写)* PSK 认证字符串。 不要将敏感信息放在 PSK 认证中，它会通过网络以未加密的方式传输，以通知接收者要使用哪个 PSK。
tls_psk	string	*(只写)* PSK 值字符串（偶数个十六进制字符）

更新

描述

`object autoregistration.update(object autoregistration)`

该方法用于更新已存在的自动注册。

> **Note:**
> 此方法只有 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

`(object)` 要被更新的自动注册属性。

返回值

`(boolean )` 成功更新后返回布尔值 true。

示例

请求：

```
{
    "jsonrpc": "2.0",
    "method": "autoregistration.update",
    "params": {
        "tls_accept": "3",
        "tls_psk_identity": "PSK 001",
        "tls_psk": "11111595725ac58dd977beef14b97461a7c1045b9a1c923453302c5473193478"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": true,
    "id": 1
}
```

来源

ui/include/classes/api/services/CAutoregistration.php 中的 CAutoregistration::update()。

获取

描述

`object autoregistration.get(object parameters)`

此方法用于根据给定的参数来获取自动注册对象。

Note:
此方法只有 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(object) 定义需要输出的参数。

此方法只支持一个参数。

参数	类型	说明
output	query	此参数对于所有 get 方法都是通用的，在参考说明中有描述。

返回值

(object) 返回自动注册对象

示例

请求：

```
{
   "jsonrpc": "2.0",
    "method": "autoregistration.get",
    "params": {
        "output": "extend"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "tls_accept": "3"
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CAutoregistration.php 中的 CAutoregistration::get()。

被发现的服务

此类用于被发现的服务。

对象引用：

- Discovered service

可用方法：

- dservice.get - 获取被发现的服务

> 被发现的服务对象

以下对象与 dservice API 直接相关。

被发现的服务

Note:
被发现的服务由 Zabbix server 创建，不能通过 API 修改。

被发现的服务对象包含一个关于被主机上网络发现规则发现的服务信息。具有以下属性。

属性	类型	描述
dserviceid	string	被发现服务的 ID。
dcheckid	string	用于检测服务的发现检查的 ID。
dhostid	string	运行服务的被发现主机的 ID。
dns	string	运行服务的主机的 DNS。
ip	string	运行服务的主机的 IP 地址。
lastdown	timestamp	被发现的服务最后一次停止的时间。
lastup	timestamp	被发现的服务最后一次启动的时间。
port	integer	服务端口号。
status	integer	服务状态。 可用值： 0 - 服务启动； 1 - 服务停止。
value	string	当执行一个 Zabbix agent、SNMPv1、SNMPv2 或 SNMPv3 发现检查时，服务返回的值。

获取

描述

`integer/array dservice.get(object parameters)`

此方法允许根据给定的参数检索被发现的服务。

Note:
此方法对任何类型的用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

`(object)` 定义需要输出的参数。

此方法支持以下参数。

参数	类型	描述
dserviceids	string/array	仅返回具有给定 ID 的被发现对象。
dhostids	string/array	仅返回属于给定被发现主机的被发现对象。
dcheckids	string/array	仅返回被给定发现检查检测出的被发现对象。
druleids	string/array	仅返回被给定发现规则检测出的被发现对象。
selectDRules	query	返回包含检测到服务的发现规则数组的drules 属性。
selectDHosts	query	返回包含服务所属的被发现主机数组的dhosts 属性。
selectHosts	query	返回与服务具有相同 IP 地址和代理的主机的hosts 属性。 支持 `count`。

参数	类型	描述
limitSelects	integer	限制子选择返回的记录数。 适用于以下子选择： selectHosts - 结果将按 hostid 排序。
sortfield	string/array	按照给定属性对结果进行排序。 可用值：dserviceid、dhostid 和 ip。
countOutput	boolean	这些参数对于所有的 get 方法都是通用的，详细描述请参见参考说明。
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

(integer/array) 返回其中一种结果：

- 一个对象数组；
- 如果使用了参数 countOutput，则返回检索到的对象的数量。

示例

检索在主机上被发现的服务

检索在被发现主机 "11" 上检测出的所有被发现的服务。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "dservice.get",
    "params": {
        "output": "extend",
        "dhostids": "11"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "dserviceid": "12",
            "dhostid": "11",
            "value": "",
```

```
            "port": "80",
            "status": "1",
            "lastup": "0",
            "lastdown": "1348650607",
            "dcheckid": "5",
            "ip": "192.168.1.134",
            "dns": "john.local"
        },
        {
            "dserviceid": "13",
            "dhostid": "11",
            "value": "",
            "port": "21",
            "status": "1",
            "lastup": "0",
            "lastdown": "1348650610",
            "dcheckid": "6",
            "ip": "192.168.1.134",
            "dns": "john.local"
        }
    ],
    "id": 1
}
```


参见

- Discovered host
- Discovery check
- Host

来源

ui/include/classes/api/services/CDService.php 中的 CDService::get()。

角色

该章节介绍如何使用有关角色的功能。

参照资料：

- 角色
- 角色规则
- UI 元素
- 业务
- 业务标签
- 模块
- 动作

支持使用的功能包括：

- role.create - 创建新的用户角色
- role.delete - 删除用户角色
- role.get - 检索用户角色
- role.update - 升级用户角色

> 角色对象

以下内容将详细介绍有关 角色 API 的相关功能。

角色

一个角色对象拥有以下属性：

属性	类型	说明
roleid	字符串	(只读) 角色 ID 号。
name (必要)	字符串	角色名称。
type (必需)	整数	用户类型。 可选值： 1 - (缺省值) 普通用户； 2 - 管理员； 3 - 超级管理员。
readonly	整数	(只读) 设定角色是否为只读。 可选值： 0 - (缺省值) 否； 1 - 是。

注意，对于某些方法（更新、删除），必需/可选参数组合是不同的。

角色规则

角色规则拥有下列属性特征：

属性	类型	说明
ui	数组	一组包含UI 元素对象的数组。
ui.default_acces	整数	表示新的 UI 元素状态为启动与否。 可配置的参数包括： 0 - 关闭； 1 - (缺省值) 开启。
services.read.m	整数	只允许只读性质的业务访问。 可配置的参数包括： 0 - 启动只读模式，通过 `services.read.list` 来限定只读的业务，或者通过与 `services.read.tag` 匹配对对应业务进行访问。 1 - (缺省值) 对所有业务均采用只读访问。
services.read.lis	数字	该数组包含所有业务 对象。 列表中的业务，包含其子业务，会对目标 ### 角色规则

角色规则拥有下列属性特征：

属性	类型	说明
ui	数组	一组包含UI 元素对象的数组。
ui.default_acces	整数	表示新的 UI 元素状态为启动与否。 可配置的参数包括： 0 - 关闭； 1 - （缺省值）开启。
services.read.ma	整数	只允许只读性质的业务访问。 可配置的参数包括： 0 - 启动只读模式，通过 `services.read.list` 来限定只读的业务，或者通过与 `services.read.tag` 匹配对对应业务进行访问。 1 - （缺省值）对所有业务均采用只读访问。
services.read.lis	数字	该数组包含所有业务 对象。 列表中的业务，包含其子业务，会对目标 ### 角色规则

角色规则拥有下列属性特征：

属性	类型	说明
ui	数组	一组包含UI 元素对象的数组。
ui.default_acces	整数	表示新的 UI 元素状态为启动与否。 可配置的参数包括： 0 - 关闭； 1 - （缺省值）开启。

属性	类型	说明
services.read.mode	整数	只允许只读性质的业务访问。 可配置的参数包括： 0 - 启动只读模式，通过 `services.read.list` 来限定只读的业务，或者通过与 `services.read.tag` 匹配对对应业务进行访问。 1 - （缺省值）对所有业务均采用只读访问。
services.read.list	数字	该数组包含所有业务 对象。 列表中的业务，包含其子业务，会对目标 ### 角色规则

角色规则拥有下列属性特征：

属性	类型	说明
ui	数组	一组包含UI 元素 对象的数组。
ui.default_access	整数	表示新的 UI 元素状态为启动与否。 可配置的参数包括： 0 - 关闭； 1 - （缺省值）开启。
services.read.mode	整数	只允许只读性质的业务访问。 可配置的参数包括： 0 - 启动只读模式，通过 `services.read.list` 来限定只读的业务，或者通过与 `services.read.tag` 匹配对对应业务进行访问。 1 - （缺省值）对所有业务均采用只读访问。

属性	类型	说明
services.read.list	数字	该数组包含所有业务 对象。 列表中的业务，包含其子业务，会对目标角色用户开启只读访问。对业务来说，只读访问并不会覆盖读写访问。 只有将 `services.read.mode` 设定为 0 才会开应用属性。
services.read.tag	对象	包含一组业务标签对象的参数。 标签所对应的业务，包含其子业务，会对目标角色用户开启对业务的只读访问。对用户来说，只读访问并不会覆盖读写访问。 只有将 `services.read.mode` 设定为 0 才会开启该功能。
services.write.mode	整数	对业务开启读写访问。 可配置的参数包括： 0 - （缺省值）设定业务的读写访问，通过 `services.read.list` 来限定读写的业务，或者通过与 `services.read.tag` 匹配对对应业务进行访问。 1 - 对所有业务开启读写访问。
services.write.list	数组	包含一组业务对象。 列表中的业务，包含其子业务，会对目标角色用户开启对业务的读写访问。对用户来讲，读写访问权限会覆盖只读访问权限。 只有将 `services.write.mode` 设定为 0 才会开启该功能。

属性	类型	说明
services.write.tag	对象	包含一组业务标签对象。 标签对应的业务，包含其子业务，会对目标角色用户开启对业务的读写访问。对用户来讲，读写访问权限会覆盖只读访问权限。 只有将 `services.write.mode` 设定为 0 才会开启该功能。
modules	数组	包含一组模块对象。
modules.default_access	整数	表示对新模块开启访问。 可配置的参数包括： 0 - 关闭； 1 - （缺省值）开启。
api.access	证书	表示对 API 开启访问。 可配置的参数包括： 0 - 关闭； 1 - （缺省值）开启。
api.mode	整数	对罗列在 `api` 属性中的 API 的方式进行模式设定。 可配置的参数包括： 0 - （缺省值）拒绝列表； 1 - 允许列表。
api	数组	包含一组 API 的方式。
actions	数组	包含一组动作对象。
actions.default_access	整数	表示对新的动作是否开启访问。 可配置的参数包括： 0 - 关闭； 1 - （缺省值）开启。

属性	类型	说明
services.read.tag	对象	包含一组业务标签对象的参数。 标签所对应的业务，包含其子业务，会对目标角色用户开启对业务的只读访问。对用户来说，只读访问并不会覆盖读写访问。 只有将 `services.read.mode` 设定为 0 才会开启该功能。
services.write.mode	整数	对业务开启读写访问。 可配置的参数包括： 0 - （缺省值）设定业务的读写访问，通过 `services.read.list` 来限定读写的业务，或者通过与 `services.read.tag` 匹配对对应业务进行访问。 1 - 对所有业务开启读写访问。
services.write.list	数组	包含一组业务对象。 列表中的业务，包含其子业务，会对目标角色用户开启对业务的读写访问。对用户来讲，读写访问权限会覆盖只读访问权限。 只有将 `services.write.mode` 设定为 0 才会开启该功能。
services.write.tag	对象	包含一组业务标签对象。 标签对应的业务，包含其子业务，会对目标角色用户开启对业务的读写访问。对用户来讲，读写访问权限会覆盖只读访问权限。 只有将 `services.write.mode` 设定为 0 才会开启该功能。

属性	类型	说明
modules	数组	包含一组模块对象。
modules.default_access	整数	表示对新模块开启访问。 可配置的参数包括： 0 - 关闭； 1 - （缺省值）开启。
api.access	证书	表示对 API 开启访问。 可配置的参数包括： 0 - 关闭； 1 - （缺省值）开启。
api.mode	整数	对罗列在 `api` 属性中的 API 的方式进行模式设定。 可配置的参数包括： 0 - （缺省值）拒绝列表； 1 - 允许列表。
api	数组	包含一组 API 的方式。
actions	数组	包含一组动作对象。
actions.default_access	整数	表示对新的动作是否开启访问。 可配置的参数包括： 0 - 关闭； 1 - （缺省值）开启。
services.read.tag	对象	包含一组业务标签对象的参数。 标签所对应的业务，包含其子业务，会对目标角色用户开启对业务的只读访问。对用户来说，只读访问并不会覆盖读写访问。 只有将 `services.read.mode` 设定为 0 才会开启该功能。

属性	类型	说明
services.write.mode	整数	对业务开启读写访问。 可配置的参数包括： 0 - （缺省值）设定业务的读写访问，通过 `services.read.list` 来限定读写的业务，或者通过与 `services.read.tag` 匹配对对应业务进行访问。 1 - 对所有业务开启读写访问。
services.write.list	数组	包含一组业务对象。 列表中的业务，包含其子业务，会对目标角色用户开启对业务的读写访问。对用户来讲，读写访问权限会覆盖只读访问权限。 只有将 `services.write.mode` 设定为 0 才会开启该功能。
services.write.tag	对象	包含一组业务标签对象。 标签对应的业务，包含其子业务，会对目标角色用户开启对业务的读写访问。对用户来讲，读写访问权限会覆盖只读访问权限。 只有将 `services.write.mode` 设定为 0 才会开启该功能。
modules	数组	包含一组模块对象。
modules.default_access	整数	表示对新模块开启访问。 可配置的参数包括： 0 - 关闭； 1 - （缺省值）开启。

属性	类型	说明
api.access	证书	表示对 API 开启访问。 可配置的参数包括： 0 - 关闭； 1 - （缺省值）开启。
api.mode	整数	对罗列在 `api` 属性中的 API 的方式进行模式设定。 可配置的参数包括： 0 - （缺省值）拒绝列表； 1 - 允许列表。
api	数组	包含一组 API 的方式。
actions	数组	包含一组动作对象。
actions.default_access	整数	表示对新的动作是否开启访问。 可配置的参数包括： 0 - 关闭； 1 - （缺省值）开启。
services.read.tag	对象	包含一组业务标签对象的参数。 标签所对应的业务，包含其子业务，会对目标角色用户开启对业务的只读访问。对用户来说，只读访问并不会覆盖读写访问。 只有将 `services.read.mode` 设定为 0 才会开启该功能。
services.write.mode	整数	对业务开启读写访问。 可配置的参数包括： 0 - （缺省值）设定业务的读写访问，通过 `services.read.list` 来限定读写的业务，或者通过与 `services.read.tag` 匹配对对应业务进行访问。 1 - 对所有业务开启读写访问。

属性	类型	说明
services.write.list	数组	包含一组业务对象。 列表中的业务，包含其子业务，会对目标角色用户开启对业务的读写访问。对用户来讲，读写访问权限会覆盖只读访问权限。 只有将 `services.write.mode` 设定为 0 才会开启该功能。
services.write.tag	对象	包含一组业务标签对象。 标签对应的业务，包含其子业务，会对目标角色用户开启对业务的读写访问。对用户来讲，读写访问权限会覆盖只读访问权限。 只有将 `services.write.mode` 设定为 0 才会开启该功能。
modules	数组	包含一组模块对象。
modules.default_access	整数	表示对新模块开启访问。 可配置的参数包括： 0 - 关闭； 1 - （缺省值）开启。
api.access	证书	表示对 API 开启访问。 可配置的参数包括： 0 - 关闭； 1 - （缺省值）开启。
api.mode	整数	对罗列在 `api` 属性中的 API 的方式进行模式设定。 可配置的参数包括： 0 - （缺省值）拒绝列表； 1 - 允许列表。
api	数组	包含一组 API 的方式。
actions	数组	包含一组动作对象。

属性	类型	说明
actions.default_access	整数	表示对新的动作是否开启访问。 可配置的参数包括： 0 - 关闭； 1 - （缺省值）开启。

UI 元素

UI 元素包含下列属性对象：

属性	类型	说明
name (必需)	字符串	UI 元素的名称。 普通用户可操作的 配置数据包括： monitoring.dashboard - 监控 → 仪表板; monitoring.problems - 监控 → 问题 ; monitoring.hosts - 监控 → 主机; monitoring.overview - 监控 → 概览; monitoring.latest_data - 监控 → 最新数据; monitoring.maps - 监控 → 拓扑图; monitoring.services - 监控 → 服务; inventory.overview - 资产清单 → 概览; inventory.hosts - 资产清单 → 主机; reports.availability_repo - 报告 → 可行性报 告; reports.top_triggers - 报告 → 前 *100* 触 发器。 管理员和超级管理 员可操作的配置数 据包括： monitoring.discovery - 监控 → 发现; reports.scheduled_reports - 报告 → 规划报告; reports.notifications - 报告 → 通知; configuration.host_groups - 配置 → 用户组; configuration.templates - 配置 → 模板; configuration.hosts - 配置 → 主机; configuration.maintenance - 配置 → 维护; configuration.actions - 配置 → 动作; configuration.discovery - 配置 → 发现. 超级管理员可操作 的配置数据包括： reports.system_info - 报告 → 系统信息; reports.audit - 报告 → 审计; reports.action_log - 报告 → 动作记录; configuration.event_corre - 配置 → 关联事件; administration.general - 管理 → 通用配置; administration.proxies - 管理 → 代理;

属性	类型	说明
status	整数	代表是否开启 UI 元素的访问。 可选值： 0 - 关闭； 1 - *(default)* 开启。

业务

属性	类型	说明
serviceid (必要)	字符串	业务 ID 号。

业务标签

属性	类型	说明
tag (必要)	字符串	标签名称。 若无填写内容，则该功能不会被使用。
value	字符串	标签数值。 若该属性没有填写任何参数，那么只有标签名称会被用来对应业务。

模块

模块对象拥有下列属性特征：

属性	类型	说明
moduleid (必要)	字符串	模块 ID。
status	整数	是否允许访问模块信息。 可配置的参数包括： 0 - 关闭； 1 - *(缺省值)* 开启。

动作

动作对象具有以下属性：

属性	类型	描述
name (必需)	string	动作的名称。 任何类型用户的可能值： `edit_dashboards` - 创建和编辑仪表盘； `edit_maps` - 创建和编辑拓扑图； `add_problem_comments` - 添加问题评论； `change_severity` - 更改问题严重性； `acknowledge_problems` - 确认问题； `close_problems` - 关闭问题； `execute_scripts` - 执行脚本； `manage_api_tokens` - 管理 API 令牌。 仅适用于管理员和超级管理员类型的用户的可能值： `edit_maintenance` - 创建和编辑维护； `manage_scheduled_reports` - 管理计划报表； `manage_sla` - 管理服务等级协议。

属性	类型	描述
status	integer	是否启用执行动作的访问。 可能的值： 0 - 禁用； 1 - (默认) 启用。

角色. 创建

说明

`object role.create(object/array roles)`

用户使用该方法可以创建新的角色用户。

> **Note:**
> 该方式仅对超级管理员类型的用户有效。用户可以在用户角色设置中对该方式的使用权限进行设定修改。请参考用户角色以获取更多信息。

参数

`(object/array)` 要创建的角色用户。

除此之外，根据标准用户角色特性, 该方式接受下列参数配置。

参数	类型	说明
rules	数组	角色角色规则用于创建新的角色用户。

返回值

根据 `roleids` 特性，`(object)` 会返回一个对象，包含创建的角色用户的 ID 号。返回的 ID 号顺序对应查询的角色用户次序。

示例

创建角色

创建一个类型为 “User” 的角色，并禁止其访问两个 UI 元素。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "role.create",
    "params": {
        "name": "Operator",
        "type": "1",
        "rules": {
            "ui": [
                {
                    "name": "monitoring.hosts",
                    "status": "0"
                },
                {
                    "name": "monitoring.maps",
                    "status": "0"
                }
            ]
        }
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "roleids": [
            "5"
        ]
    },
    "id": 1
}
```

另见

- Role rules
- UI element
- Module
- Action

来源

CRole::create() in *ui/include/classes/api/services/CRole.php*.

角色. 删除

说明

`object role.delete(array roleids)`

用户使用该方法可以删除角色用户。

该方法允许任何类型的用户使用。用户可以在用户角色设置中对该方式的使用权限进行设定修改。请参考用户角色以获取更多信息。

参数

(array) 要删除的角色用户 ID 号。

返回值

根据 roleids 特性，(object) 会返回一个对象，包含删除的角色用户的 ID 号。

示例

删除多个用户角色

删除两个用户角色。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "role.delete",
    "params": [
        "4",
        "5"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "roleids": [
            "4",
            "5"
        ]
    },
    "id": 1
}
```

参考来源

CRole::delete() in *ui/include/classes/api/services/CRole.php*.

角色. 更新

说明

`object role.update(object/array roles)`

用户使用该方法可以用来更新存在的角色用户。

该方式仅对超级管理员类型的用户生效。用户可以在用户角色设置中对该方式的使用权限进行设定修改。请参考用户角色以获取更多信息。

参数

(object/array) 要更新的角色属性。

必须为每个角色定义 roleid 属性，所有其他属性均为可选。只有传递的属性将被更新，其他属性将保持不变。

除了标准角色属性，该方法还接受以下参数。

参数	类型	描述
rules	array	要为角色更新的规则。

返回值

根据 roleids 的特性，(object) 返回一个包含已更新职责用户 ID 的对象。

示例

关闭运行脚本功能

更新 ID 号为 5 的角色，关闭其运行脚本的功能。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "role.update",
    "params": [
        {
            "roleid": "5",
            "rules": {
                "actions": [
                    {
                        "name": "execute_scripts",
                        "status": "0"
                    }
                ]
            }
        }
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "roleids": [
            "5"
        ]
    },
    "id": 1
}
```

限制对 API 的访问

更新 ID 号为 5 的角色，拒绝任何” 创建”, ” 更新” 或者” 删除” 方式。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "role.update",
    "params": [
        {
            "roleid": "5",
            "rules": {
                "api.access": "1",
                "api.mode": "0",
                "api": ["*.create", "*.update", "*.delete"]
            }
        }
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "roleids": [
            "5"
        ]
    },
    "id": 1
}
```

参考来源

CRole::update() in *ui/include/classes/api/services/CRole.php*.

角色. 获取

说明

`integer/array role.get(object parameters)`

该方法允许用户通过给予一定参数用来检索职责用户信息。

该方法允许任何类型的用户使用。用户可以在用户角色设置中对该方式的使用权限进行设定修改。请参考用户角色以获取更多信息。

参数

(object) 该参数表明了用户想得到的数据结果。

该方法支持以下参数。

参数	类型	说明
roleids	字符串/数组	根据提供的 ID 号返回角色用户。
selectRules	询问	根据角色用户 属性返回角色规则。
selectUsers	询问	选择分配给该角色的用户。
sortfield	字符串/数组	根据用户罗列的属性对反馈结果进行分类。 可配置的参数包括：roleid，name。
countOutput	布尔值	该参数在 get 方式中应用广泛，具体内容可参考评论引用页面。
editable	布尔值	
excludeSearch	布尔值	
filter	对象	
limit	整数	
output	询问	
preservekeys	布尔值	

参数	类型	说明
search	对象	
searchByAny	布尔值	
searchWildcardsEnabled	布尔值	
sortorder	字符串/数组	
startSearch	布尔值	

返回值

(integer/array) 返回下列两种之一：

- 一组对象；
- 在 countOutput 参数应用的情况下，返回检索对象的数量。

示例

检索角色数据

检索单个" 超级用户角色" 角色数据和其访问规则。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "role.get",
    "params": [
        "output": "extend",
        "selectRules": "extend",
        "roleids": "3"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "roleid": "3",
            "name": "Super admin role",
            "type": "3",
            "readonly": "1",
            "rules": {
                "ui": [
                    {
                        "name": "monitoring.dashboard",
                        "status": "1"
                    },
                    {
                        "name": "monitoring.problems",
                        "status": "1"
                    },
                    {
                        "name": "monitoring.hosts",
                        "status": "1"
                    },
                    {
                        "name": "monitoring.latest_data",
                        "status": "1"
                    },
                    {
                        "name": "monitoring.maps",
                        "status": "1"
                    },
```

```
                {
                    "name": "monitoring.services",
                    "status": "1"
                },
                {
                    "name": "inventory.overview",
                    "status": "1"
                },
                {
                    "name": "inventory.hosts",
                    "status": "1"
                },
                {
                    "name": "reports.availability_report",
                    "status": "1"
                },
                {
                    "name": "reports.top_triggers",
                    "status": "1"
                },
                {
                    "name": "monitoring.discovery",
                    "status": "1"
                },
                {
                    "name": "reports.notifications",
                    "status": "1"
                },
                {
                    "name": "reports.scheduled_reports",
                    "status": "1"
                },
                {
                    "name": "configuration.host_groups",
                    "status": "1"
                },
                {
                    "name": "configuration.templates",
                    "status": "1"
                },
                {
                    "name": "configuration.hosts",
                    "status": "1"
                },
                {
                    "name": "configuration.maintenance",
                    "status": "1"
                },
                {
                    "name": "configuration.actions",
                    "status": "1"
                },
                {
                    "name": "configuration.discovery",
                    "status": "1"
                },
                {
                    "name": "reports.system_info",
                    "status": "1"
                },
                {
                    "name": "reports.audit",
```

```
                "status": "1"
            },
            {
                "name": "reports.action_log",
                "status": "1"
            },
            {
                "name": "configuration.event_correlation",
                "status": "1"
            },
            {
                "name": "administration.general",
                "status": "1"
            },
            {
                "name": "administration.proxies",
                "status": "1"
            },
            {
                "name": "administration.authentication",
                "status": "1"
            },
            {
                "name": "administration.user_groups",
                "status": "1"
            },
            {
                "name": "administration.user_roles",
                "status": "1"
            },
            {
                "name": "administration.users",
                "status": "1"
            },
            {
                "name": "administration.media_types",
                "status": "1"
            },
            {
                "name": "administration.scripts",
                "status": "1"
            },
            {
                "name": "administration.queue",
                "status": "1"
            }
        ],
        "ui.default_access": "1",
        "services.read.mode": "1",
        "services.read.list": [],
        "services.read.tag": {
            "tag": "",
            "value": ""
        },
        "services.write.mode": "1",
        "services.write.list": [],
        "services.write.tag": {
            "tag": "",
            "value": ""
        },
        "modules": [],
        "modules.default_access": "1",
```

```
                "api.access": "1",
                "api.mode": "0",
                "api": [],
                "actions": [
                    {
                        "name": "edit_dashboards",
                        "status": "1"
                    },
                    {
                        "name": "edit_maps",
                        "status": "1"
                    },
                    {
                        "name": "acknowledge_problems",
                        "status": "1"
                    },
                    {
                        "name": "close_problems",
                        "status": "1"
                    },
                    {
                        "name": "change_severity",
                        "status": "1"
                    },
                    {
                        "name": "add_problem_comments",
                        "status": "1"
                    },
                    {
                        "name": "execute_scripts",
                        "status": "1"
                    },
                    {
                        "name": "manage_api_tokens",
                        "status": "1"
                    },
                    {
                        "name": "edit_maintenance",
                        "status": "1"
                    },
                    {
                        "name": "manage_scheduled_reports",
                        "status": "1"
                    }
                ],
                "actions.default_access": "1"
            }
        }
    ],
    "id": 1
}
```


另请参考

- 角色规则
- 用户

参考来源

CRole::get() in *ui/include/classes/api/services/CRole.php*.

触发器

此类用于管理触发器.

对象引用:

- 触发器

可用方法:

- trigger.adddependencies - 添加新的触发器依赖项
- trigger.create - 创建新的触发器
- trigger.delete - 删除触发器
- trigger.deletedependencies - 删除触发器依赖项
- trigger.get - 检索触发器
- trigger.update - 更新触发器

> 触发器对象

以下对象与 `triggerAPI` 直接相关.

触发器

触发器对象具有以下属性。

属性	类型	描述
triggerid	string	*(只读)* 触发器的 ID。
description (必选)	string	触发器的名称。
expression (必选)	string	简化的触发器表达式。
event_name	string	触发器生成的事件名称。
opdata	string	运营数据。
comments	string	触发器的额外描述。
error	string	*(只读)* 如果在更新触发器状态时遇到问题，错误文本。
flags	integer	*(只读)* 触发器的来源。 可能的值是： 0 - *(默认)* 普通触发器； 4 - 发现的触发器。
lastchange	timestamp	*(只读)* 触发器最后一次状态改变的时间。
priority	integer	触发器的严重性。 可能的值是： 0 - *(默认)* 未分类； 1 - 信息； 2 - 警告； 3 - 一般； 4 - 高； 5 - 灾难。
state	integer	*(只读)* 触发器的状态。 可能的值： 0 - *(默认)* 触发器状态是最新的； 1 - 当前触发器状态未知。
status	integer	触发器是否启用或禁用。 可能的值是： 0 - *(默认)* 启用； 1 - 禁用。
templateid	string	*(只读)* 父模板触发器的 ID。
type	integer	触发器是否可以生成多个问题事件。 可能的值是： 0 - *(默认)* 不生成多个事件； 1 - 生成多个事件。
url	string	与触发器关联的 URL。

属性	类型	描述
value	integer	*(只读)* 触发器是否处于正常或问题状态。 可能的值是： 0 - *(默认)* 正常； 1 - 问题。
recovery_mode	integer	正常事件生成模式。 可能的值是： 0 - *(默认)* 表达式； 1 - 恢复表达式； 2 - 无。
recovery_expression	string	简化的触发器恢复表达式。
correlation_mode	integer	正常事件关闭。 可能的值是： 0 - *(默认)* 所有问题； 1 - 如果标签值匹配，则所有问题。
correlation_tag	string	用于匹配的标签。
manual_close	integer	允许手动关闭。 可能的值是： 0 - *(默认)* 否； 1 - 是。
uuid	string	通用唯一标识符，用于将导入的触发器链接到已存在的触发器。仅用于模板上的触发器。如果未给出，则自动生成。

请注意，对于一些方法（更新，删除），必选/可选参数组合是不同的。

触发器标签

触发标签对象具有以下属性.

属性	类型	描述
tag (required)	string	触发器的标签名称.
value	string	触发器的标签值.

创建触发器

描述

`object trigger.create(object/array triggers)`

此方法允许创建新的触发器.

Note:
此方法只适用于 *Admin* 和 *Super admin* 用户类型，调用该方法的权限可以在用户角色设置中被撤销. 前往用户角色 以查看并了解更多信息.

参数

`(object/array)` 要创建的触发器.

除了标准触发器属性 之外，该方法还接受以下参数.

参数	类型	描述
dependencies	array	目的触发器. 目的触发器必须存在且已定义 `triggerid` 属性.
tags	array	触发器标签标签.

Attention:
触发器表达式必须以扩展形式给出.

返回值

(object) 返回一个对象，该对象包含在 triggerids 属性下创建的触发器 ID，返回的 ID 顺序与传递的触发器的顺序相匹配.

示例

创建触发器

创建具有单个触发依赖关系的触发器.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "trigger.create",
    "params": [
        {
            "description": "Processor load is too high on {HOST.NAME}",
            "expression": "last(/Linux server/system.cpu.load[percpu,avg1])>5",
            "dependencies": [
                {
                    "triggerid": "17367"
                }
            ]
        },
        {
            "description": "Service status",
            "expression": "length(last(/Linux server/log[/var/log/system,Service .* has stopped]))<>0",
            "dependencies": [
                {
                    "triggerid": "17368"
                }
            ],
            "tags": [
                {
                    "tag": "service",
                    "value": "{{ITEM.VALUE}.regsub(\"Service (.*) has stopped\", \"\\1\")}"
                },
                {
                    "tag": "error",
                    "value": ""
                }
            ]
        }
    ],
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "triggerids": [
            "17369",
            "17370"
        ]
    },
    "id": 1
}
```

源码

CTrigger::create() in *ui/include/classes/api/services/CTrigger.php*.

删除触发器

描述

`object trigger.delete(array triggerIds)`

此方法允许删除触发器.

> **Note:**
> 此方法只适用于 Admin 和 Super admin 用户类型，调用该方法的权限可以在用户角色设置中被撤销. 前往用户角色 以查看并了解更多信息.

参数

(array) 要删除的触发器 ID 列表.

返回值

(object) 返回一个对象，该对象包含在 triggerids 属性下已删除触发器 ID.

示例

删除多个触发器

删除两个触发器.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "trigger.delete",
    "params": [
        "12002",
        "12003"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "triggerids": [
            "12002",
            "12003"
        ]
    },
    "id": 1
}
```

源码

CTrigger::delete() in *ui/include/classes/api/services/CTrigger.php*.

删除触发器依赖

描述

`object trigger.deletedependencies(string/array triggers)`

此方法允许从指定的触发器中删除所有的触发依赖关系.

Note:
此方法只适用于 Admin 和 Super admin 用户类型，调用该方法的权限可以在用户角色设置中被撤销. 前往用户角色 以查看并了解更多信息.

参数

(string/array) 需要从触发依赖中删除的触发器.

返回值

(object) 返回一个对象，该对象包含在 triggerids 属性中已受影响触发器的 ID.

示例

从多个触发器中删除依赖关系

从两个触发器中删除所有依赖关系.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "trigger.deleteDependencies",
    "params": [
        {
            "triggerid": "14544"
        },
        {
            "triggerid": "14545"
        }
    ],
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "triggerids": [
            "14544",
            "14545"
        ]
    },
    "id": 1
}
```

另见

- 更新触发器

源码

CTrigger::deleteDependencies() in *ui/include/classes/api/services/CTrigger.php*.

更新触发器

描述

`object trigger.update(object/array triggers)`

方法用于更新目前的触发器.

Note:
此方法只适用于 _Admin_ 和 _Super admin_ 用户类型，调用该方法的权限可以在用户角色设置中被撤销. 前往用户角色 以查看并了解更多信息.

参数

(object/array) 需要更新的触发器属性.

triggerid 属性必须在每个应用集中已定义，其他所有属性为可选项。只有传递过去的属性会被更新，其他所有属性仍然保持不变.

除了标准触发器属性 之外，该方法还接受以下参数.

参数	类型	描述
dependencies	array	依赖触发的触发器. 触发器必须已定义 triggerid 属性.
tags	array	触发器标签.

Attention:
指定的触发器表达式必须为展开式.

返回值

(object) 返回一个对象，其中包含 triggerids 属性下已更新的触发器的 ID.

示例

启用一个触发器

启用触发器，即将其状态设置为 0.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "trigger.update",
    "params": {
        "triggerid": "13938",
        "status": 0
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "triggerids": [
            "13938"
        ]
    },
    "id": 1
}
```

替换触发器标签

为触发器替换标签.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "trigger.update",
    "params": {
        "triggerid": "13938",
        "tags": [
            {
                "tag": "service",
                "value": "{{ITEM.VALUE}.regsub(\"Service (.*) has stopped\", \"\\1\")}"
            },
            {
                "tag": "error",
                "value": ""
```

```
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "triggerids": [
            "13938"
        ]
    },
    "id": 1
}
```

另见

- 触发器添加依赖
- 触发器删除依赖

源码

CTrigger::update() in *ui/include/classes/api/services/CTrigger.php*.

检索触发器

描述

`integer/array trigger.get(object parameters)`

此方法允许根据指定的参数检索触发器.

Note:
此方法适用于任何类型的用户，可以在用户角色设置中撤销调用该方法的权限，前往用户角色 查看更多详细信息.

参数

`(object)` 定义所需输出的参数。

该方法支持以下参数。

参数	类型	描述
triggerids	string/array	仅返回给定 ID 的触发器。
groupids	string/array	仅返回属于给定主机组的触发器。
templateids	string/array	仅返回属于给定模板的触发器。
hostids	string/array	仅返回属于给定主机的触发器。
itemids	string/array	仅返回包含给定监控项的触发器。
functions	string/array	仅返回使用给定函数的触发器。 参考支持的函数页面了解支持的函数列表。
group	string	仅返回属于给定名称的主机组的主机的触发器。
host	string	仅返回属于具有给定技术名称的主机的触发器。
inherited	boolean	如果设置为 `true`，则仅返回从模板继承的触发器。
templated	boolean	如果设置为 `true`，则仅返回属于模板的触发器。
dependent	boolean	如果设置为 `true`，则仅返回具有依赖关系的触发器。如果设置为 `false`，则仅返回没有依赖关系的触发器。
monitored	flag	仅返回属于受监控主机并且只包含已启用监控项的已启用触发器。
active	flag	仅返回属于受监控主机的已启用触发器。
maintenance	boolean	如果设置为 `true`，则仅返回属于处于维护中的主机的已启用触发器。
withUnacknowledgedEvents	flag	仅返回具有未确认事件的触发器。
withAcknowledgedEvents	flag	仅返回所有事件都已确认的触发器。
withLastEventUnacknowledged	flag	仅返回最后一个事件未确认的触发器。

参数	类型	描述
skipDependent	flag	跳过处于问题状态且依赖于其他触发器的触发器。注意，如果其他触发器被禁用、包含被禁用的监控项或被禁用的监控项主机，则它们会被忽略。
lastChangeSince	timestamp	仅返回在给定时间之后状态发生变化的触发器。
lastChangeTill	timestamp	仅返回在给定时间之前状态发生变化的触发器。
only_true	flag	仅返回最近处于问题状态的触发器。
min_severity	integer	仅返回严重性大于或等于给定严重性的触发器。
evaltype	integer	标签搜索的规则。 可能的值： 0 - (默认) 与/或； 2 - 或。
tags	array of objects	仅返回具有给定标签的触发器。根据运算符的值，进行标签的精确匹配或大小写敏感或不敏感的搜索。 格式：`[{"tag": "<tag>", "value": "<value>", "operator": "<operator>"}, ...]`。 空数组返回所有触发器。 可能的运算符类型： 0 - (默认) 包含； 1 - 等于； 2 - 不包含； 3 - 不等于 4 - 存在； 5 - 不存在。
expandComment	flag	在触发器描述中展开宏。
expandDescription	flag	在触发器名称中展开宏。
expandExpression	flag	在触发器表达式中展开函数和宏。
selectGroups	query	在groups属性中返回触发器所属的主机组。
selectHosts	query	在hosts属性中返回触发器所属的主机。
selectItems	query	在items属性中返回触发器包含的监控项。
selectFunctions	query	在 `functions` 属性中返回触发器中使用的函数。 函数对象表示触发器表达式中使用的函数，并且具有以下属性： `functionid` - *(*字符串*)* 函数的 ID； `itemid` - *(*字符串*)* 函数中使用的监控项的 ID； `function` - *(*字符串*)* 函数的名称； `parameter` - *(*字符串*)* 传递给函数的参数。查询参数在返回的字符串中由 $ 符号替换。
selectDependencies	query	在 `dependencies` 属性中返回触发器依赖的触发器。
selectDiscoveryRule	query	返回创建触发器的低级发现规则。
selectLastEvent	query	在lastEvent属性中返回最后一个重要的触发器事件。
selectTags	query	在tags属性中返回触发器标签。
selectTriggerDiscovery	query	在 `triggerDiscovery` 属性中返回触发器发现对象。触发器发现对象将触发器链接到它从中创建的触发器原型。 它具有以下属性： `parent_triggerid` - （字符串）触发器原型的 ID，触发器是从该原型创建的。
filter	object	仅返回完全匹配给定过滤的结果。 接受一个数组，其中键是属性名称，值是单个值或要匹配的值数组。 支持额外的过滤： `host` - 触发器所属主机的技术名称； `hostid` - 触发器所属主机的 ID。
limitSelects	integer	限制子选择返回的记录数。 适用于以下子选择： `selectHosts` - 结果将按 `host` 排序。

参数	类型	描述
sortfield	string/array	排序结果按给定属性。 可能的值有：`triggerid`, `description`, `status`, `priority`, `lastchange` 和 `hostname`。
countOutput	boolean	这些参数对所有 `get` 方法都是通用的，详细描述在参考说明页面。
editable	boolean	
excludeSearch	boolean	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

`(integer/array)` 返回两者其中任一:

- 一组对象数组;
- 如果已经使用了 `countOutput` 参数，则检索对象的计数.

示例

根据触发器 ID 检索数据

检索触发器"14062" 中使用的所有数据和功能.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "trigger.get",
    "params": {
        "triggerids": "14062",
        "output": "extend",
        "selectFunctions": "extend"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "triggerid": "14062",
            "expression": "{13513}<10m",
            "description": "{HOST.NAME} has been restarted (uptime < 10m)",
            "url": "",
            "status": "0",
            "value": "0",
            "priority": "2",
            "lastchange": "0",
            "comments": "The host uptime is less than 10 minutes",
            "error": "",
            "templateid": "10016",
            "type": "0",
            "state": "0",
            "flags": "0",
            "recovery_mode": "0",
            "recovery_expression": "",
```

```
            "correlation_mode": "0",
            "correlation_tag": "",
            "manual_close": "0",
            "opdata": "",
            "functions": [
                {
                    "functionid": "13513",
                    "itemid": "24350",
                    "triggerid": "14062",
                    "parameter": "$",
                    "function": "last"
                }
            ]
        }
    ],
    "id": 1
}
```

检索在问题状态的触发器

检索在问题状态下的所有触发器的 ID，名称和严重性，并按严重性级别按降序分类。

请求:

```
{
    "jsonrpc": "2.0",
    "method": "trigger.get",
    "params": {
        "output": [
            "triggerid",
            "description",
            "priority"
        ],
        "filter": {
            "value": 1
        },
        "sortfield": "priority",
        "sortorder": "DESC"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "triggerid": "13907",
            "description": "Zabbix self-monitoring processes < 100% busy",
            "priority": "4"
        },
        {
            "triggerid": "13824",
            "description": "Zabbix discoverer processes more than 75% busy",
            "priority": "3"
        }
    ],
    "id": 1
}
```

使用标签检索特定的触发器.

使用标签检索特定的触发器.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "trigger.get",
    "params": {
        "output": [
            "triggerid",
            "description"
        ],
        "selectTags": "extend",
        "triggerids": [
            "17578"
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "triggerid": "17370",
            "description": "Service status",
            "tags": [
                {
                    "tag": "service",
                    "value": "{{ITEM.VALUE}.regsub(\"Service (.*) has stopped\", \"\\1\")}"
                },
                {
                    "tag": "error",
                    "value": ""
                }
            ]
        }
    ],
    "id": 1
}
```

另见

- 发现规则
- 监控项
- 主机
- 主机组

源码

CTrigger::get() in *ui/include/classes/api/services/CTrigger.php*.

添加触发依赖

描述

`object trigger.adddependencies(object/array triggerDependencies)`

此方法允许创建新的触发器依赖关系.

Note:
此方法只适用于 *Admin* 和 *Super admin* 用户类型，调用该方法的权限可以在用户角色设置中被撤销. 前往用户角色 以查看并了解更多信息.

参数

`(object/array)` 需要创建的触发器依赖.

每一个触发器依赖具有以下参数:

参数	类型	描述
triggerid (必须)	string	源触发器 ID.
dependsOnTriggerid (必须)	string	被依赖的目的触发器 ID.

返回值

（对象）返回一个对象，该对象包含 `triggerids` 属性下的触发器 ID.

示例

添加触发器依赖

使触发器 “14092” 依赖于触发器 “13565”.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "trigger.adddependencies",
    "params": {
        "triggerid": "14092",
        "dependsOnTriggerid": "13565"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "triggerids": [
            "14092"
        ]
    },
    "id": 1
}
```

另见

- 更新触发器
- 触发器依赖关系

源码

CTrigger::addDependencies() in *ui/include/classes/api/services/CTrigger.php*.

触发器原型

此类用于管理触发器原型.

对象引用:

- 触发器原型

可用方法:

- triggerprototype.create - 创建新的触发器原型
- triggerprototype.delete - 删除触发器原型
- triggerprototype.get - 检索触发器原型
- triggerprototype.update - 更新触发器原型

> 触发器原型对象

以下对象与 `triggerprototype`API 直接相关.

触发器原型

触发器原型对象具有以下属性。

属性	类型	描述
triggerid	string	*(只读)* 触发器原型的 ID。
description (必选)	string	触发器原型的名称。
expression (必选)	string	简化的触发器表达式。
event_name	string	触发器生成的事件名称。
opdata	string	运营数据。
comments	string	对触发器原型的额外注释。
priority	integer	触发器原型的严重性。 可能的值： 0 - *(默认)* 未分类； 1 - 信息； 2 - 警告； 3 - 一般； 4 - 高； 5 - 灾难。
status	integer	触发器原型是否启用或禁用。 可能的值： 0 - *(默认)* 启用； 1 - 禁用。
templateid	string	*(只读)* 父模板触发器原型的 ID。
type	integer	触发器原型是否可以生成多个问题事件。 可能的值： 0 - *(默认)* 不生成多个事件； 1 - 生成多个事件。
url	string	与触发器原型关联的 URL。
recovery_mode	integer	正常事件生成模式。 可能的值是： 0 - *(默认)* 表达式； 1 - 恢复表达式； 2 - 无。
recovery_expression	string	简化的触发器恢复表达式。
correlation_mode	integer	正常事件关闭。 可能的值是： 0 - *(默认)* 所有问题； 1 - 如果标签值匹配，则所有问题。
correlation_tag	string	用于匹配的标签。
manual_close	integer	允许手动关闭。 可能的值是： 0 - *(默认)* 否； 1 - 是。
discover	integer	触发器原型发现状态。 可能的值： 0 - *(默认)* 将发现新触发器； 1 - 不会发现新触发器，并将现有触发器标记为丢失。

属性	类型	描述
uuid	string	通用唯一标识符，用于将导入的触发器原型链接到已存在的原型。仅用于模板上的触发器原型。如果未给出，则自动生成。

请注意，对于一些方法（更新，删除），必选/可选参数组合是不同的。

触发原型标签

触发器原型标记对象具有以下属性.

属性	类型	描述
tag (必须)	string	触发原型标签名称.
value	string	触发原型标签值.

triggerprototype.update

描述

`object triggerprototype.update(object/array triggerPrototypes)`

此方法允许更新已有的触发器原型.

> **Note:**
> 此方法只有 *Admin(管理员)* 和 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色 。

参数

`(object/array)` 需要更新的触发器原型

必须在每个触发器原型中定义 `triggerid` 属性，其他所有属性为可选项。只有传递过去的属性会被更新，其他所有属性保持不变。

除了标准的触发器原型属性之外，该方法还接受以下参数.

参数	类型	描述
dependencies	array	触发器原型所依赖的触发器和触发器原型. 触发器必须已定义 `triggerid` 属性..
tags	array	触发器原型标签.

> **Attention:**
> 在扩展表单中必须填入至少包含一个监控项原型的触发器表达式。

返回值

`(object)` 返回一个对象，该对象是包含在 `triggerids` 属性中已更新触发器原型的 ID.

示例

启用触发器原型

启用一个触发器原型，即将其状态设置为 0.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "triggerprototype.update",
    "params": {
        "triggerid": "13938",
        "status": 0
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "triggerids": [
            "13938"
        ]
    },
    "id": 1
}
```

替换触发器原型标签

为触发器原型替换标签.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "triggerprototype.update",
    "params": {
        "triggerid": "17373",
        "tags": [
            {
                "tag": "volume",
                "value": "{#FSNAME}"
            },
            {
                "tag": "type",
                "value": "{#FSTYPE}"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "triggerids": [
            "17373"
        ]
    },
    "id": 1
}
```

源码

CTriggerPrototype::update() in *ui/include/classes/api/services/CTriggerPrototype.php*.

创建触发器原型

描述

`object triggerprototype.create(object/array triggerPrototypes)`

此方法允许创建新的触发器原型.

> **Note:**
> 此方法只适用于 Admin 和 Super admin 用户类型，调用该方法的权限可以在用户角色设置中被撤销. 前往用户角色 以查看并了解更多信息.

参数

(object/array) 创建触发器原型.

除了标准触发器原型属性 此方法还接受以下参数.

参数	类型	描述
dependencies	array	触发器原型所依赖的触发器和触发器原型. 触发器必须已定义 `triggerid` 属性.
tags	array	触发器标签.

Attention:
指定的触发器表达式必须为展开式，并且必须包含至少一个监控项原型.

返回值

(object) 返回一个对象，该对象包含在 `triggerids` 属性中已创建触发器原型的 ID，返回 ID 的顺序与传递触发器原型的顺序相匹配.

示例

创建触发器原型

创建一个触发器原型来检测磁盘剩余空间是否小于 20%.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "triggerprototype.create",
    "params": {
        "description": "Free disk space is less than 20% on volume {#FSNAME}",
        "expression": "last(/Zabbix server/vfs.fs.size[{#FSNAME},pfree])<20",
        "tags": [
            {
                "tag": "volume",
                "value": "{#FSNAME}"
            },
            {
                "tag": "type",
                "value": "{#FSTYPE}"
            }
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "triggerids": [
            "17372"
        ]
    },
    "id": 1
}
```

源码

CTriggerPrototype::create() in *ui/include/classes/api/services/CTriggerPrototype.php*.

删除触发器原型

描述

`object triggerprototype.delete(array triggerPrototypeIds)`

此方法允许删除触发器原型.

> **Note:**
> 此方法只适用于 _Admin_ 和 _Super admin_ 用户类型，调用该方法的权限可以在用户角色设置中被撤销. 前往用户角色 以查看并了解更多信息

参数

(array) 要删除的触发器原型的 ID.

返回值

(object) 返回一个对象，该对象包含在 triggerids 属性中已删除触发器原型的 ID.

示例

删除多个触发器原型

删除两个触发器原型.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "triggerprototype.delete",
    "params": [
        "12002",
        "12003"
    ],
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "triggerids": [
            "12002",
            "12003"
        ]
    },
    "id": 1
}
```

源码

CTriggerPrototype::delete() in *ui/include/classes/api/services/CTriggerPrototype.php*.

检索触发器原型

描述

`integer/array triggerprototype.get(object parameters)`

此方法允许根据指定的参数检索触发器原型.

> **Note:**
> 此方法适用于任何类型的用户，调用方法的权限可以在用户角色设置中进行撤销，请参阅用户角色了解更多信息.

参数

(object) 定义所需输出的参数。

该方法支持以下参数。

参数	类型	描述
active	flag	仅返回属于受监控主机的已启用触发器原型。

参数	类型	描述
discoveryids	string/array	只返回属于给定 LLD 规则的触发器原型。
functions	字符串/数组	仅返回使用给定函数的触发器。 有关支持函数的列表，请参阅支持的函数页面。
group	string	仅返回属于具有给定名称的主机组中的主机的触发器原型。
groupids	string/array	仅返回属于给定主机组中主机的触发器原型。
host	string	仅返回属于具有给定名称的主机的触发器原型。
hostids	string/array	只返回属于给定主机的触发器原型。
inherited	boolean	如果设置为 “true”，则仅返回从模板继承的触发器原型。
maintenance	boolean	如果设置为 “true”，则仅返回属于维护中主机的已启用触发器原型。
min_severity	integer	仅返回严重性大于或等于给定严重性的触发器原型。
monitored	flag	仅返回属于受监控主机且仅包含启用项目的已启用触发器原型。
templated	boolean	如果设置为 “true”，则只返回属于模板的触发器原型。
templateids	string/array	只返回属于给定模板的触发器原型。
triggerids	字符串/数组	仅返回具有给定 ID 的触发器原型。
expandExpression	flag	展开触发器表达式中的函数和宏。
selectDependencies	query	在 `dependencies` 属性中返回触发器原型和触发器原型所依赖的触发器。
selectDiscoveryRule	query	返回触发器原型所属的LLD 规则。
selectFunctions	query	在 `functions` 属性中返回触发器原型中使用的函数。 函数对象表示触发器表达式中使用的函数，具有以下属性： `functionid` - * (string)* 函数的 ID； `itemid` - *(string)* 函数中使用的项目的 ID； `function` - *(string)* 函数的名称； `parameter` - *(string)* 传递给函数的参数。查询参数在返回的字符串中被替换为 “$” 符号。
selectGroups	query	在groups 属性中返回触发器原型所属的主机组。
selectHosts	query	在hosts 属性中返回触发器原型所属的主机。
selectItems	query	返回监控项和监控项原型使用items 属性中的触发器原型。
selectTags	query	返回tags 属性中的触发器原型标签。
filter	object	只返回那些与给定过滤器完全匹配的结果。 接受一个数组，其中键是属性名称，值是要匹配的单个值或值数组。 支持额外的过滤器： `host` - 触发器原型所属主机的正式名称； `hostid` - 触发器原型所属主机的 ID。
limitSelects	integer	限制子选择返回的记录数。 适用于以下子选择： `selectHosts` - 结果将按 `host` 排序。
sortfield	string/array	按给定的属性对结果进行排序。 可能的值为：`triggerid`、`description`、`status` 和 `priority`。
countOutput	boolean	参考评论 中详细描述了所有 “get” 方法通用的这些参数。
editable	boolean	
excludeSearch	boolean	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

`(integer/array)` 返回两者其中任一:

- 一组对象.
- 如果已经使用了 `countOutput` 参数，则检索对象的计数.

示例

从 LLD 规则中检索触发器原型

从底层发现规则中检索所有的触发器原型和相关函数.

请求:

```
{
    "jsonrpc": "2.0",
    "method": "triggerprototype.get",
    "params": {
        "output": "extend",
        "selectFunctions": "extend",
        "discoveryids": "22450"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "triggerid": "13272",
            "expression": "{12598}<20",
            "description": "Free inodes is less than 20% on volume {#FSNAME}",
            "url": "",
            "status": "0",
            "priority": "2",
            "comments": "",
            "templateid": "0",
            "type": "0",
            "flags": "2",
            "recovery_mode": "0",
            "recovery_expression": "",
            "correlation_mode": "0",
            "correlation_tag": "",
            "manual_close": "0",
            "opdata": "",
            "discover": "0",
            "functions": [
                {
                    "functionid": "12598",
                    "itemid": "22454",
                    "triggerid": "13272",
                    "parameter": "$",
                    "function": "last"
                }
            ]
        },
        {
            "triggerid": "13266",
            "expression": "{13500}<20",
            "description": "Free disk space is less than 20% on volume {#FSNAME}",
            "url": "",
            "status": "0",
            "priority": "2",
            "comments": "",
            "templateid": "0",
            "type": "0",
            "flags": "2",
            "recovery_mode": "0",
            "recovery_expression": "",
```

```
            "correlation_mode": "0",
            "correlation_tag": "",
            "manual_close": "0",
            "opdata": "",
            "discover": "0",
            "functions": [
                {
                    "functionid": "13500",
                    "itemid": "22686",
                    "triggerid": "13266",
                    "parameter": "$",
                    "function": "last"
                }
            ]
        }
    ],
    "id": 1
}
```

根据标签检索特定的触发器原型

请求:

```
{
    "jsonrpc": "2.0",
    "method": "triggerprototype.get",
    "params": {
        "output": [
            "triggerid",
            "description"
        ]
        "selectTags": "extend",
        "triggerids": [
            "17373"
        ]
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "triggerid": "17373",
            "description": "Free disk space is less than 20% on volume {#FSNAME}",
            "tags": [
                {
                    "tag": "volume",
                    "value": "{#FSNAME}"
                },
                {
                    "tag": "type",
                    "value": "{#FSTYPE}"
                }
            ]
        }
    ],
    "id": 1
}
```

另见

- 发现规则

- 监控项
- 主机
- 主机组

源码

CTriggerPrototype::get() in *ui/include/classes/api/services/CTriggerPrototype.php*.

认证

此类用于操作认证设置。

对象引用：

- Authentication

可用方法：

- authentication.get - 获取认证
- authentication.update - 更新认证

> 认证对象

以下对象与 `authentication` API 直接相关。

认证

认证对象具有以下属性。

属性	类型	描述
authentication_type	integer	默认认证。 可能的值： 0 - *(默认)* 内部； 1 - LDAP。
http_auth_enabled	integer	启用 HTTP 认证。 可能的值： 0 - *(默认)* 禁用； 1 - 启用。
http_login_form	integer	默认登录表单。 可能的值： 0 - *(默认)* Zabbix 登录表单； 1 - HTTP 登录表单。
http_strip_domains	string	移除域名。
http_case_sensitive	integer	HTTP 大小写敏感登录。 可能的值： 0 - 关闭； 1 - *(默认)* 开启。
ldap_configured	integer	启用 LDAP 认证。 可能的值： 0 - 禁用； 1 - *(默认)* 启用。
ldap_host	string	LDAP 主机。
ldap_port	integer	LDAP 端口。
ldap_base_dn	string	LDAP 基础 DN。
ldap_search_attribute	string	LDAP 搜索属性。
ldap_bind_dn	string	LDAP 绑定 DN。

属性	类型	描述
ldap_case_sensitive	integer	LDAP 大小写敏感登录。 可能的值： 0 - 关闭； 1 - (默认) 开启。
ldap_bind_password	string	LDAP 绑定密码。
saml_auth_enabled	integer	启用 SAML 认证。 可能的值： 0 - (默认) 禁用； 1 - 启用。
saml_idp_entityid	string	SAML IdP 实体 ID。
saml_sso_url	string	SAML SSO 服务 URL。
saml_slo_url	string	SAML SLO 服务 URL。
saml_username_attribute	string	SAML 用户名属性。
saml_sp_entityid	string	SAML SP 实体 ID。
saml_nameid_format	string	SAML SP 名称 ID 格式。
saml_sign_messages	integer	SAML 签名消息。 可能的值： 0 - (默认) 不签名消息； 1 - 签名消息。
saml_sign_assertions	integer	SAML 签名断言。 可能的值： 0 - (默认) 不签名断言； 1 - 签名断言。
saml_sign_authn_requests	integer	SAML 签名认证请求。 可能的值： 0 - (默认) 不签名认证请求； 1 - 签名认证请求。
saml_sign_logout_requests	integer	SAML 签名注销请求。 可能的值： 0 - (默认) 不签名注销请求； 1 - 签名注销请求。
saml_sign_logout_responses	integer	SAML 签名注销响应。 可能的值： 0 - (默认) 不签名注销响应； 1 - 签名注销响应。
saml_encrypt_nameid	integer	SAML 加密名称 ID。 可能的值： 0 - (默认) 不加密名称 ID； 1 - 加密名称 ID。
saml_encrypt_assertions	integer	SAML 加密断言。 可能的值： 0 - (默认) 不加密断言； 1 - 加密断言。
saml_case_sensitive	integer	SAML 大小写敏感登录。 可能的值： 0 - 关闭； 1 - (默认) 开启。
passwd_min_length	integer	密码最小长度要求。 可能的范围值：1-70 8 - 默认

属性	类型	描述
passwd_check_rules	integer	密码检查规则。 可能的位图值是： 0 - 检查密码长度； 1 - 检查密码是否使用大写和小写拉丁字母； 2 - 检查密码是否使用数字； 4 - 检查密码是否使用特殊字符； 8 - *(默认)* 检查密码是否不在常用密码列表中，不包含"Zabbix" 一词的派生词或用户名、姓或用户名。

更新

描述

`object authentication.update(object authentication)`

此方法允许更新现有的认证设置。

Note:
此方法只有 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

`(object)` 要更新的认证属性。

返回值

`(array)` 返回包含已更新参数名称的数组。

示例

请求：

```
{
    "jsonrpc": "2.0",
    "method": "authentication.update",
    "params": {
        "http_auth_enabled": 1,
        "http_case_sensitive": 0,
        "http_login_form": 1
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        "http_auth_enabled",
        "http_case_sensitive",
        "http_login_form"
    ],
    "id": 1
}
```

来源

ui/include/classes/api/services/CAuthentication.php 中的 CAuthentication::update()。

获取

描述

`object authentication.get(object parameters)`

该方法允许根据给定的参数检索认证对象。

> **Note:**
> 此方法只有 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

(object) 定义所需输出的参数。

这个方法只支持一个参数。

参数	类型	描述
output	query	这个参数对于所有 get 方法都是通用的，在参考说明 进行了描述。

返回值

(object) 返回认证对象。

示例

请求：

```
{
    "jsonrpc": "2.0",
    "method": "authentication.get",
    "params": {
        "output": "extend"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "authentication_type": "0",
        "http_auth_enabled": "0",
        "http_login_form": "0",
        "http_strip_domains": "",
        "http_case_sensitive": "1",
        "ldap_configured": "0",
        "ldap_host": "",
        "ldap_port": "389",
        "ldap_base_dn": "",
        "ldap_search_attribute": "",
        "ldap_bind_dn": "",
        "ldap_case_sensitive": "1",
        "ldap_bind_password": "",
        "saml_auth_enabled": "0",
        "saml_idp_entityid": "",
        "saml_sso_url": "",
        "saml_slo_url": "",
        "saml_username_attribute": "",
        "saml_sp_entityid": "",
        "saml_nameid_format": "",
        "saml_sign_messages": "0",
        "saml_sign_assertions": "0",
        "saml_sign_authn_requests": "0",
        "saml_sign_logout_requests": "0",
        "saml_sign_logout_responses": "0",
        "saml_encrypt_nameid": "0",
        "saml_encrypt_assertions": "0",
```

```
        "saml_case_sensitive": "0",
        "passwd_min_length": "8",
        "passwd_check_rules": "8"
    },
    "id": 1
}
```

来源

ui/include/classes/api/services/CAuthentication.php 中的 CAuthentication::get()。

设置

此类为进行常用管理设置而设计。

参考对象:

- 设置

可用办法

- settings.get - 检索设置
- settings.update - 更新设置

> 设置对象

以下对象都是与 `settings` 直接相关的 API。

设置

设置对象具有以下属性。

属性	类型	描述
default_lang	string	默认系统语言。 默认值：en_GB。
default_timezone	string	默认系统时区。 默认值：system - 系统默认。 有关支持的时区完整列表，请参阅 PHP 文档。
default_theme	string	默认主题。 可能的值： blue-theme - *(默认)* 蓝色； dark-theme - 深色； hc-light - 高对比度浅色； hc-dark - 高对比度深色。
search_limit	integer	搜索和过滤结果的限制。 默认值：1000。
max_overview_table_size	integer	数据概览和触发器概览仪表板小部件中的最大列数和行数。 默认值：50。
max_in_table	integer	单元格内显示的元素最大数量。 默认值：50。
server_check_interval	integer	如果 Zabbix 服务器宕机，显示警告。 可能的值：0 - 不显示警告； 10 - *(默认)* 显示警告。

属性	类型	描述
work_period	string	工作时间。 默认值：1-5,09:00-18:00。
show_technical_errors	integer	向非超级管理员用户以及未启用调试模式的用户组显示技术错误（PHP/SQL）。 可能的值：0 - *(默认)* 不显示技术错误； 1 - 显示技术错误。
history_period	string	在最新数据、Web 和数据概览仪表板小部件中显示历史数据的最大期限。接受秒和时间单位后缀。 默认值：24h。
period_default	string	时间过滤器的默认期限。接受秒和时间单位后缀，支持月和年（30s,1m,2h,1d,1M,1y）。 默认值：1h。
max_period	string	时间过滤器的最大期限。接受秒和时间单位后缀，支持月和年（30s,1m,2h,1d,1M,1y）。 默认值：2y。
severity_color_0	string	”未分类”严重性的颜色，以十六进制颜色代码表示。 默认值：97AAB3。
severity_color_1	string	”信息”严重性的颜色，以十六进制颜色代码表示。 默认值：7499FF。
severity_color_2	string	”警告”严重性的颜色，以十六进制颜色代码表示。 默认值：FFC859。
severity_color_3	string	”一般”严重性的颜色，以十六进制颜色代码表示。 默认值：FFA059。
severity_color_4	string	”高”严重性的颜色，以十六进制颜色代码表示。 默认值：E97659。
severity_color_5	string	”灾难”严重性的颜色，以十六进制颜色代码表示。 默认值：E45959。
severity_name_0	string	”未分类”严重性的名称。 默认值：未分类。
severity_name_1	string	”信息”严重性的名称。 默认值：信息。
severity_name_2	string	”警告”严重性的名称。 默认值：警告。
severity_name_3	string	”一般”严重性的名称。 默认值：一般。
severity_name_4	string	”高”严重性的名称。 默认值：高。
severity_name_5	string	”灾难”严重性的名称。 默认值：灾难。
custom_color	integer	使用自定义事件状态颜色。 可能的值： 0 - *(默认)* 不使用自定义事件状态颜色； 1 - 使用自定义事件状态颜色。

属性	类型	描述
ok_period	string	显示 OK 触发器期限。接受秒和时间单位后缀。 默认值：5m。
blink_period	string	状态变化时触发器闪烁期限。接受秒和时间单位后缀。 默认值：2m。
problem_unack_color	string	未确认的 PROBLEM 事件的颜色，以十六进制颜色代码表示。 默认值：CC0000。
problem_ack_color	string	已确认的 PROBLEM 事件的颜色，以十六进制颜色代码表示。 默认值：CC0000。
ok_unack_color	string	未确认的 RESOLVED 事件的颜色，以十六进制颜色代码表示。 默认值：009900。
ok_ack_color	string	已确认的 RESOLVED 事件的颜色，以十六进制颜色代码表示。 默认值：009900。
problem_unack_style	integer	未确认的 PROBLEM 事件的闪烁。 可能的值： 0 - 不显示闪烁； 1 - *(默认)* 显示闪烁。
problem_ack_style	integer	已确认的 PROBLEM 事件的闪烁。 可能的值： 0 - 不显示闪烁； 1 - *(默认)* 显示闪烁。
ok_unack_style	integer	未确认的 RESOLVED 事件的闪烁。 可能的值： 0 - 不显示闪烁； 1 - *(默认)* 显示闪烁。
ok_ack_style	integer	已确认的 RESOLVED 事件的闪烁。 可能的值： 0 - 不显示闪烁； 1 - *(默认)* 显示闪烁。
url	string	前端 URL。
discovery_groupid	integer	自动放置发现的主机的主机组 ID。
default_inventory_mode	integer	默认主机库存模式。 可能的值： -1 - *(默认)* 禁用； 0 - 手动； 1 - 自动。
alert_usrgrpid	integer	将发送数据库宕机告警消息的用户组 ID。如果设置为 0，则不会发送告警消息。
snmptrap_logging	integer	记录不匹配的 SNMP 陷阱。 可能的值： 0 - 不记录不匹配的 SNMP 陷阱； 1 - *(默认)* 记录不匹配的 SNMP 陷阱。
login_attempts	integer	登录表单将被阻止的失败登录尝试次数。 默认值：5。
login_block	string	如果登录尝试失败次数超过 login_attempts 字段中定义的次数，则登录表单将被阻止的时间间隔。接受秒和时间单位后缀。 默认值：30s。

属性	类型	描述
validate_uri_schemes	integer	验证 URI 方案。 可能的值： 0 - 不验证； 1 - *(默认)* 验证。
uri_valid_schemes	string	有效的 URI 方案。 默认值：http,https,ftp,file,mailto,tel,ssh。
x_frame_options	string	X-Frame-Options HTTP 头部。 默认值：SAMEORIGIN。
iframe_sandboxing_enabled	integer	使用 iframe 沙箱。 可能的值： 0 - 不使用； 1 - *(默认)* 使用。
iframe_sandboxing_exceptions	string	iframe 沙箱异常。
connect_timeout	string	与 Zabbix 服务器的连接超时。 默认值：3s。
socket_timeout	string	网络默认超时。 默认值：3s。
media_type_test_timeout	string	媒体类型测试的网络超时。 默认值：65s。
item_test_timeout	string	项目测试的网络超时。 默认值：60s。
script_timeout	string	脚本执行的网络超时。 默认值：60s。
report_test_timeout	string	计划报告测试的网络超时。 默认值：60s。
auditlog_enabled	integer	启用审计日志记录。 可能的值： 0 - 禁用； 1 - *(默认)* 启用。
ha_failover_delay	string	故障转移延迟时间（只读）。 默认值：1m。
geomaps_tile_provider	string	地图瓦片提供者。 可能的值： `OpenStreetMap.Mapnik` - *(默认)* OpenStreetMap Mapnik； `OpenTopoMap` - OpenTopoMap； `Stamen.TonerLite` - Stamen Toner Lite； `Stamen.Terrain` - Stamen 地形； `USGS.USTopo` - USGS 美国地形； `USGS.USImagery` - USGS 美国影像。
geomaps_tile_url	string	支持空字符串以指定 `geomaps_tile_url`、`geomaps_max_zoom` 和 `geomaps_attribution` 的自定义值。 如果 `geomaps_tile_provider` 设置为空字符串，则为地图瓦片 URL。
geomaps_max_zoom	integer	如果 `geomaps_tile_provider` 设置为空字符串，则为地图最大缩放级别。最大缩放级别必须在 0 到 30 的范围内。
geomaps_attribution	string	如果 `geomaps_tile_provider` 设置为空字符串，则为地图归属文本。

settings.get

描述

`object settings.get(object parameters)`

此方法允许根据给定参数检索设置对象。

> **Note:**
> 此方法允许任何用户使用。可以在用户角色设置中撤销调用此方法的权限。更多信息请查看用户角色。

参数

（对象）定义期望输出的参数。

此方法只支持一个参数。

参数	类型	描述
output	query	此参数与所有 get 方法相同，描述见reference commentary。

返回值

（对象）返回设置对象。

示例

请求：

```
{
    "jsonrpc": "2.0",
    "method": "settings.get",
    "params": {
        "output": "extend"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "default_theme": "blue-theme",
        "search_limit": "1000",
        "max_in_table": "50",
        "server_check_interval": "10",
        "work_period": "1-5,09:00-18:00",
        "show_technical_errors": "0",
        "history_period": "24h",
        "period_default": "1h",
        "max_period": "2y",
        "severity_color_0": "97AAB3",
        "severity_color_1": "7499FF",
        "severity_color_2": "FFC859",
        "severity_color_3": "FFA059",
        "severity_color_4": "E97659",
        "severity_color_5": "E45959",
        "severity_name_0": "Not classified",
        "severity_name_1": "Information",
        "severity_name_2": "Warning",
        "severity_name_3": "Average",
        "severity_name_4": "High",
        "severity_name_5": "Disaster",
        "custom_color": "0",
        "ok_period": "5m",
```

```
        "blink_period": "2m",
        "problem_unack_color": "CC0000",
        "problem_ack_color": "CC0000",
        "ok_unack_color": "009900",
        "ok_ack_color": "009900",
        "problem_unack_style": "1",
        "problem_ack_style": "1",
        "ok_unack_style": "1",
        "ok_ack_style": "1",
        "discovery_groupid": "5",
        "default_inventory_mode": "-1",
        "alert_usrgrpid": "7",
        "snmptrap_logging": "1",
        "default_lang": "en_GB",
        "default_timezone": "system",
        "login_attempts": "5",
        "login_block": "30s",
        "validate_uri_schemes": "1",
        "uri_valid_schemes": "http,https,ftp,file,mailto,tel,ssh",
        "x_frame_options": "SAMEORIGIN",
        "iframe_sandboxing_enabled": "1",
        "iframe_sandboxing_exceptions": "",
        "max_overview_table_size": "50",
        "connect_timeout": "3s",
        "socket_timeout": "3s",
        "media_type_test_timeout": "65s",
        "script_timeout": "60s",
        "item_test_timeout": "60s",
        "url": "",
        "report_test_timeout": "60s",
        "auditlog_enabled": "1",
        "ha_failover_delay": "1m",
        "geomaps_tile_provider": "OpenStreetMap.Mapnik",
        "geomaps_tile_url": "",
        "geomaps_max_zoom": "0",
        "geomaps_attribution": ""
    },
    "id": 1
}
```


来源

CSettings::get() 见 *ui/include/classes/api/services/CSettings.php*。

settings.update

描述

`object settings.update(object settings)`

此方法允许更新已存在的常用设置。

> **Note:**
> 此方法只有 *Super admin(超级管理员)* 用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

（对象）设置待更新的属性。

返回值

（数组）返回被更新参数名字的数组。

示例

请求：

```
{
    "jsonrpc": "2.0",
    "method": "settings.update",
    "params": {
        "login_attempts": "1",
        "login_block": "1m"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        "login_attempts",
        "login_block"
    ],
    "id": 1
}
```

来源

CSettings::update() 在 *ui/include/classes/api/services/CSettings.php*。

趋势

此类用于处理趋势数据.

对象引用:

- 趋势

可用方法:

- trend.get - 检索趋势数据

> 趋势对象

以下对象与 `trend` API 直接相关.

> **Note:**
> 趋势对象根据监控项类型的信息而有所不同，它们由 Zabbix Server 创建，不能通过 API 进行修改.

浮点型趋势

浮点型趋势对象具有以下属性.

属性	类型	描述
clock	timestamp	根据时间戳计算出一个小时内监控项的值，例如，04:00:00 的时间戳表示为 04:00:00-04:59:59 期间计算出来的监控项的值.
itemid	integer	相关监控项的 ID.
num	integer	一小时内可用的值的数量是多少个.
value_min	float	每小时内的最小值.
value_avg	float	每小时内的平均值.
value_max	float	每小时内的最大值.

整数型趋势

整数型趋势对象具有以下属性.

属性	类型	描述
clock	timestamp	根据时间戳计算出一个小时内监控项的值，例如，04:00:00 的时间戳表示为 04:00:00-04:59:59 期间计算出来的监控项的值.
itemid	integer	相关监控项的 ID.
num	integer	一小时内可用的值的数量是多少个.
value_min	integer	每小时内的最小值.
value_avg	integer	每小时内的平均值.
value_max	integer	每小时内的最大值.

检索趋势

描述

`integer/array trend.get(object parameters)`

该方法用于根据指定的参数检索趋势数据.

> **Note:**
> 此方法适用于任何类型的用户，调用方法的权限可以在用户角色设置中进行撤销，请参阅用户角色了解更多信息.

参数

`(object)` 定义期望输出的参数.

该方法支持以下参数.

参数	类型	描述
itemids	string/array	仅返回指定监控项 ID 的趋势.
time_from	timestamp	仅返回在给定时间之后或在给定时间收集的值.
time_till	timestamp	仅返回在给定时间之前或在给定时间收集的值.
countOutput	boolean	统计检索到的对象的数量.
limit	integer	限制检索对象的数量.
output	query	设置要输出的字段.

返回值

`(integer/array)` 返回两者其中任一:

- 一个对象数组
- 如果已经使用了 `countOutput` 参数，则统计对象的数量.

示例

检索监控项趋势数据

请求:

```
{
    "jsonrpc": "2.0",
    "method": "trend.get",
    "params": {
        "output": [
            "itemid",
            "clock",
            "num",
            "value_min",
            "value_avg",
            "value_max",
        ],
        "itemids": [
            "23715"
        ],
        "limit": "1"
```

```
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "itemid": "23715",
            "clock": "1446199200",
            "num": "60",
            "value_min": "0.165",
            "value_avg": "0.2168",
            "value_max": "0.35",
        }
    ],
    "id": 1
}
```

源码

CTrend::get() in *ui/include/classes/api/services/CTrend.php*.

配置

这个类旨在导出和导入 Zabbix 配置数据。

可用方法：

- configuration.export - 导出配置
- configuration.import - 导入配置
- configuration.importcompare - 将导入文件与当前系统元素进行比较

configuration.import

描述

`boolean configuration.import(object parameters)`

此方法允许从序列化的字符串导入配置数据。

> **Note:**
> 此方法适用于所有类型的用户。可以在用户角色设置中撤销调用此方法的权限。有关更多信息，请参阅用户角色。

参数

(object) 包含要导入的数据以及如何处理数据的规则。

参数	类型说明	描述
format (必需)	string	序列化字符串的格式。 可能的值： yaml - YAML 格式； xml - XML 格式； json - JSON 格式。
source (必需)	string	包含配置数据的序列化字符串。

参数	类型说明	描述
rules (必需)	object	关于如何导入新对象和现有对象的规则。 `rules` 参数在下面的表格中有详细描述。

Note:
如果没有给出规则，配置将不会被更新。

`rules` 对象支持以下参数。

参数	类型说明	描述
discoveryRules	object	关于如何导入 LLD 规则的规则。 支持的参数： `createMissing` - (`boolean`) 如果设置为 `true`，则会创建新的 LLD 规则；默认：`false`； `updateExisting` - (`boolean`) 如果设置为 `true`，则会更新现有的 LLD 规则；默认：`false`； `deleteMissing` - (`boolean`) 如果设置为 `true`，则会从数据库中删除导入数据中不存在的 LLD 规则；默认：`false`。
graphs	object	关于如何导入图表的规则。 支持的参数： `createMissing` - (`boolean`) 如果设置为 `true`，则会创建新的图表；默认：`false`； `updateExisting` - (`boolean`) 如果设置为 `true`，则会更新现有的图表；默认：`false`； `deleteMissing` - (`boolean`) 如果设置为 `true`，则会从数据库中删除导入数据中不存在的图表；默认：`false`。
groups	object	主机组导入规则。 支持的参数： `createMissing` - (`boolean`) 如果设置为 `true`，则会创建缺失的主机组；默认：`false`； `updateExisting` - (`boolean`) 如果设置为 `true`，则会更新现有的主机组；默认：`false`。
hosts	object	主机导入规则。 支持的参数： `createMissing` - (`boolean`) 如果设置为 `true`，则会创建新的主机；默认：`false`； `updateExisting` - (`boolean`) 如果设置为 `true`，则会更新现有的主机；默认：`false`。

参数	类型说明	描述
httptests	object	Web 场景导入规则。 支持的参数： `createMissing` - (`boolean`) 如果设置为 `true`，则会创建新的 Web 场景；默认：`false`； `updateExisting` - (`boolean`) 如果设置为 `true`，则会更新现有的 Web 场景；默认：`false`； `deleteMissing` - (`boolean`) 如果设置为 `true`，则会从数据库中删除导入数据中不存在的 Web 场景；默认：`false`。
images	object	图像导入规则。 支持的参数： `createMissing` - (`boolean`) 如果设置为 `true`，则会创建新的图像；默认：`false`； `updateExisting` - (`boolean`) 如果设置为 `true`，则会更新现有的图像；默认：`false`。
items	object	监控项导入规则。 支持的参数： `createMissing` - (`boolean`) 如果设置为 `true`，则会创建新的监控项；默认：`false`； `updateExisting` - (`boolean`) 如果设置为 `true`，则会更新现有的监控项；默认：`false`； `deleteMissing` - (`boolean`) 如果设置为 `true`，则会从数据库中删除导入数据中不存在的监控项；默认：`false`。
maps	object	地图导入规则。 支持的参数： `createMissing` - (`boolean`) 如果设置为 `true`，则会创建新的地图；默认：`false`； `updateExisting` - (`boolean`) 如果设置为 `true`，则会更新现有的地图；默认：`false`。
mediaTypes	object	媒体类型导入规则。 支持的参数： `createMissing` - (`boolean`) 如果设置为 `true`，则会创建新的媒体类型；默认：`false`； `updateExisting` - (`boolean`) 如果设置为 `true`，则会更新现有的媒体类型；默认：`false`。

参数	类型说明	描述
templateLinkage	object	模板链接导入规则。 支持的参数： createMissing - (boolean) 如果设置为 true，则会链接导入数据中存在但未链接到主机或模板的模板；默认：false； deleteMissing - (boolean) 如果设置为 true，则会取消链接主机或模板导入中不存在但链接自己链接模板的模板，且不会删除从取消链接的模板继承的实体(监控项、触发器等)；默认：false。
templates	object	模板导入规则。 支持的参数： createMissing - (boolean) 如果设置为 true，则会创建新的模板；默认：false； updateExisting - (boolean) 如果设置为 true，则会更新现有的模板；默认：false。
templateDashboards	object	模板仪表板导入规则。 支持的参数： createMissing - (boolean) 如果设置为 true，则会创建新的模板仪表板；默认：false； updateExisting - (boolean) 如果设置为 true，则会更新现有的模板仪表板；默认：false； deleteMissing - (boolean) 如果设置为 true，则会从数据库中删除导入数据中不存在的模板仪表板；默认：false。
triggers	object	触发器导入规则。 支持的参数： createMissing - (boolean) 如果设置为 true，则会创建新的触发器；默认：false； updateExisting - (boolean) 如果设置为 true，则会更新现有的触发器；默认：false； deleteMissing - (boolean) 如果设置为 true，则会从数据库中删除导入数据中不存在的触发器；默认：false。
valueMaps	object	主机或模板值映射导入规则。 支持的参数： createMissing - (boolean) 如果设置为 true，则会创建新的值映射；默认：false； updateExisting - (boolean) 如果设置为 true，则会更新现有的值映射；默认：false； deleteMissing - (boolean) 如果设置为 true，则会从数据库中删除导入数据中不存在的值映射；默认：false。

返回值

(boolean) 如果导入成功将返回 true

示例

导入模板

导入 XML 字符串中包含的模板配置。如果 XML 字符串中的任何监控项或触发器缺失，它们将从数据库中删除，其他所有内容将保持不变。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "configuration.import",
    "params": {
        "format": "xml",
        "rules": {
            "templates": {
                "createMissing": true,
                "updateExisting": true
            },
            "items": {
                "createMissing": true,
                "updateExisting": true,
                "deleteMissing": true
            },
            "triggers": {
                "createMissing": true,
                "updateExisting": true,
                "deleteMissing": true
            },
            "valueMaps": {
                "createMissing": true,
                "updateExisting": false
            }
        },
        "source": "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<zabbix_export><version>6.0</version><date>
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": true,
    "id": 1
}
```

源码

CConfiguration::import() 在 *ui/include/classes/api/services/CConfiguration.php* 文件中。

导出

描述

`string configuration.export(object parameters)`

此方法允许将配置数据导出为序列化的字符串。

> **Note:**
> 此方法适用于所有类型的用户。可以在用户角色设置中撤销调用此方法的权限。有关更多信息，请参阅用户角色。

参数

(object) 定义要导出的对象以及使用的格式。

参数	类型说明	描述
format (必需)	string	数据导出的格式。 可能的值： `yaml` - YAML 格式； `xml` - XML 格式； `json` - JSON 格式； `raw` - 未处理的 PHP 数组。
prettyprint	boolean	通过添加缩进来使输出更易于阅读。 可能的值： `true` - 添加缩进； `false` - (默认) 不添加缩进。
options (必需)	object	要导出的对象。 `options` 对象具有以下参数： `groups` - (array) 要导出的主机组的 ID； `hosts` - (array) 要导出的主机的 ID； `images` - (array) 要导出的图像的 ID； `maps` - (array) 要导出的地图的 ID； `mediaTypes` - (array) 要导出的媒体类型的 ID； `templates` - (array) 要导出的模板的 ID。

返回值

(`string`) 返回一个序列化的字符串，其中包含所请求的配置数据。

示例

导出模板

将模板"10571" 的配置导出为一个 XML 字符串。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "configuration.export",
    "params": {
        "options": {
            "templates": [
                "10571"
            ]
        },
        "format": "xml"
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<zabbix_export><version>6.0</version><date>2023
    "id": 1
}
```

源码

CConfiguration::export() 在 *ui/include/classes/api/services/CConfiguration.php* 文件中。

比较导入

描述

`array configuration.importcompare(object parameters)`

这个方法允许将导入文件与当前系统元素进行比较，并展示如果导入文件被导入将会发生哪些变化。

Note:
此方法适用于任何类型的用户。可以在用户角色设置中撤销调用方法的权限。更多信息，请参见用户角色。

参数

(object) 包含可能要导入的数据以及如何处理数据的规则。

参数	类型	描述
format (必需)	string	序列化字符串的格式。 可能的值： yaml - YAML; xml - XML; json - JSON。
source (必需)	string	包含配置数据的序列化字符串。
rules (必需)	object	如何导入新对象和现有对象的规则。 rules 参数在下面的表中详细描述。

Note:
如果没有给出规则，将没有内容可以更新，结果将是空的。

Note:
仅对主机组和模板进行比较。触发器和图表仅对导入的模板进行比较，其他任何内容都将被视为 “新” 的。

rules 对象支持以下参数。

参数	类型	描述
discoveryRules	object	如何导入 LLD 规则的规则。 支持的参数： createMissing - (boolean) 如果设置为 true，则会创建新的 LLD 规则；默认值：false； updateExisting - (boolean) 如果设置为 true，则会更新现有的 LLD 规则；默认值：false； deleteMissing - (boolean) 如果设置为 true，则会从数据库中删除导入数据中不存在的 LLD 规则；默认值：false。
graphs	object	如何导入图表的规则。 支持的参数： createMissing - (boolean) 如果设置为 true，则会创建新的图表；默认值：false； updateExisting - (boolean) 如果设置为 true，则会更新现有的图表；默认值：false； deleteMissing - (boolean) 如果设置为 true，则会从数据库中删除导入数据中不存在的图表；默认值：false。
groups	object	如何导入主机组的规则。 支持的参数： createMissing - (boolean) 如果设置为 true，则会创建新的主机组；默认值：false； updateExisting - (boolean) 如果设置为 true，则会更新现有的主机组；默认值：false。

参数	类型	描述
hosts	object	如何导入主机的规则。 支持的参数： createMissing - (boolean) 如果设置为 true，则会创建新的主机；默认值：false； updateExisting - (boolean) 如果设置为 true，则会更新现有的主机。 此参数对输出没有影响。它仅被允许以与 configuration.import 保持一致性。
httptests	object	如何导入网络场景的规则。 支持的参数： createMissing - (boolean) 如果设置为 true，则会创建新的网络场景；默认值：false； updateExisting - (boolean) 如果设置为 true，则会更新现有的网络场景；默认值：false； deleteMissing - (boolean) 如果设置为 true，则会从数据库中删除导入数据中不存在的网络场景；默认值：false。
images	object	如何导入图像的规则。 支持的参数： createMissing - (boolean) 如果设置为 true，则会创建新的图像；默认值：false； updateExisting - (boolean) 如果设置为 true，则会更新现有的图像。 此参数对输出没有影响。它仅被允许以与 configuration.import 保持一致性。
items	object	如何导入监控项的规则。 支持的参数： createMissing - (boolean) 如果设置为 true，则会创建新的监控项；默认值：false； updateExisting - (boolean) 如果设置为 true，则会更新现有的监控项；默认值：false； deleteMissing - (boolean) 如果设置为 true，则会从数据库中删除导入数据中不存在的监控项；默认值：false。
maps	object	如何导入地图的规则。 支持的参数： createMissing - (boolean) 如果设置为 true，则会创建新的地图；默认值：false； updateExisting - (boolean) 如果设置为 true，则会更新现有的地图。 此参数对输出没有影响。它仅被允许以与 configuration.import 保持一致性。
mediaTypes	object	如何导入媒体类型的规则。 支持的参数： createMissing - (boolean) 如果设置为 true，则会创建新的媒体类型；默认值：false； updateExisting - (boolean) 如果设置为 true，则会更新现有的媒体类型。 此参数对输出没有影响。它仅被允许以与 configuration.import 保持一致性。

参数	类型	描述
templateLinkage	object	如何导入模板链接的规则。 支持的参数： createMissing - (boolean) 如果设置为 true，则会将不在导入数据中但与正在导入的主机或模板链接的模板链接起来；默认值：false； deleteMissing - (boolean) 如果设置为 true，则会将与正在导入的主机或模板链接但在导入数据中不存在的模板取消链接，但不会删除从取消链接的模板继承的实体（监控项、触发器等）；默认值：false。
templates	object	如何导入模板的规则。 支持的参数： createMissing - (boolean) 如果设置为 true，则会创建新的模板；默认值：false； updateExisting - (boolean) 如果设置为 true，则会更新现有的模板；默认值：false。
templateDashboards	object	如何导入模板仪表盘的规则。 支持的参数： createMissing - (boolean) 如果设置为 true，则会创建新的模板仪表盘；默认值：false； updateExisting - (boolean) 如果设置为 true，则会更新现有的模板仪表盘；默认值：false； deleteMissing - (boolean) 如果设置为 true，则会从数据库中删除导入数据中不存在的模板仪表盘；默认值：false。
triggers	object	如何导入触发器的规则。 支持的参数： createMissing - (boolean) 如果设置为 true，则会创建新的触发器；默认值：false； updateExisting - (boolean) 如果设置为 true，则会更新现有的触发器；默认值：false； deleteMissing - (boolean) 如果设置为 true，则会从数据库中删除导入数据中不存在的触发器；默认值：false。
valueMaps	object	如何导入主机或模板值映射的规则。 支持的参数： createMissing - (boolean) 如果设置为 true，则会创建新的值映射；默认值：false； updateExisting - (boolean) 如果设置为 true，则会更新现有的值映射；默认值：false； deleteMissing - (boolean) 如果设置为 true，则会从数据库中删除导入数据中不存在的值映射；默认值：false。

返回值

(array) 返回一个数组，包含将要进行的配置变更。

示例

模板导入比较

将 XML 字符串中包含的模板与当前系统元素进行比较，并展示如果该模板被导入将会发生哪些变更。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "configuration.importcompare",
    "params": {
        "format": "xml",
        "rules": {
            "discoveryRules": {
                "createMissing": true,
                "updateExisting": true,
```

```
                "deleteMissing": true
            },
            "graphs": {
                "createMissing": true,
                "updateExisting": true,
                "deleteMissing": true
            },
            "groups": {
                "createMissing": true,
                "updateExisting": true
            },
            "httptests": {
                "createMissing": true,
                "updateExisting": true,
                "deleteMissing": true
            },
            "items": {
                "createMissing": true,
                "updateExisting": true,
                "deleteMissing": true
            },
            "templateLinkage": {
                "createMissing": true,
                "deleteMissing": true
            },
            "templates": {
                "createMissing": true,
                "updateExisting": true
            },
            "templateDashboards": {
                "createMissing": true,
                "updateExisting": true,
                "deleteMissing": true
            },
            "triggers": {
                "createMissing": true,
                "updateExisting": true,
                "deleteMissing": true
            },
            "valueMaps": {
                "createMissing": true,
                "updateExisting": true,
                "deleteMissing": true
            }
        },
        "source": "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<zabbix_export><version>6.0</version><date>
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

响应:

```
{
    "jsonrpc": "2.0",
    "result": {
        "templates": {
            "updated": [
                {
                    "before": {
                        "uuid": "5aef0444a82a4d8cb7a95dc4c0c85330",
                        "template": "New template",
                        "name": "New template"
```

```
        },
        "after": {
            "uuid": "5aef0444a82a4d8cb7a95dc4c0c85330",
            "template": "New template",
            "name": "New template"
        },
        "items": {
            "added": [
                {
                    "after": {
                        "uuid": "648006da5971424ead0c47ddbbf1ea2e",
                        "name": "CPU utilization",
                        "key": "system.cpu.util",
                        "value_type": "FLOAT",
                        "units": "%"
                    },
                    "triggers": {
                        "added": [
                            {
                                "after": {
                                    "uuid": "736225012c534ec480c2a66a91322ce0",
                                    "expression": "avg(/New template/system.cpu.util,3m)>70",
                                    "name": "CPU utilization too high on 'New host' for 3 minu
                                    "priority": "WARNING"
                                }
                            }
                        ]
                    }
                }
            ],
            "removed": [
                {
                    "before": {
                        "uuid": "6805d4c39a624a8bab2cc8ab63df1ab3",
                        "name": "CPU load",
                        "key": "system.cpu.load",
                        "value_type": "FLOAT"
                    },
                    "triggers": {
                        "removed": [
                            {
                                "before": {
                                    "uuid": "ab4c2526c2bc42e48a633082255ebcb3",
                                    "expression": "avg(/New template/system.cpu.load,3m)>2",
                                    "name": "CPU load too high on 'New host' for 3 minutes",
                                    "priority": "WARNING"
                                }
                            }
                        ]
                    }
                }
            ],
            "updated": [
                {
                    "before": {
                        "uuid": "7f1e6f1e48aa4a128e5b6a958a5d11c3",
                        "name": "Zabbix agent ping",
                        "key": "agent.ping"
                    },
                    "after": {
                        "uuid": "7f1e6f1e48aa4a128e5b6a958a5d11c3",
                        "name": "Zabbix agent ping",
```

```
                                    "key": "agent.ping",
                                    "delay": "3m"
                                }
                            }
                        ]
                    }
                }
            ]
        }
    },
    "id": 1
}
```

源码

CConfiguration::importcompare() 在 *ui/include/classes/api/services/CConfiguration.php* 文件中。

问题

此类设计用于管理问题。

对象引用：

- 问题

可用方法：

- problem.get - 获取问题

> 问题对象

以下对象直接与 problem API 相关。

问题

Note:
问题是由 Zabbix server 创建的，不能通过 API 修改。

问题对象具有以下属性。

属性	类型	描述
eventid	string	问题事件的 ID。
source	integer	问题事件的类型。 可能的值： 0 - 由触发器创建的事件； 3 - 内部事件； 4 - 服务状态更新时创建的事件。
object	integer	与问题事件相关联的对象类型。 触发器事件的可能值： 0 - 触发器。 内部事件的可能值： 0 - 触发器； 4 - 监控项； 5 - 低级发现规则。 服务事件的可能值： 6 - 服务。
objectid	string	相关对象的 ID。
clock	timestamp	问题事件被创建的时间。

属性	类型	描述
ns	integer	问题事件被创建时的纳秒数。
r_eventid	string	恢复事件的 ID。
r_clock	timestamp	恢复事件被创建的时间。
r_ns	integer	恢复事件被创建时的纳秒数。
correlationid	string	如果此事件是由全局关联规则恢复的，则为关联规则 ID。
userid	string	如果问题被手动关闭，则为手动关闭问题的用户 ID。
name	string	已解决问题的名称。
acknowledged	integer	问题的认可状态。 可能的值： 0 - 未认可； 1 - 已认可。
severity	integer	问题当前的严重性。 可能的值： 0 - 未分类； 1 - 信息； 2 - 警告； 3 - 平均； 4 - 高； 5 - 灾难。
suppressed	integer	问题是否被抑制。 可能的值： 0 - 问题处于正常状态； 1 - 问题被抑制。
opdata	string	包含扩展宏的运营数据。
urls	array	活跃的媒体类型 URLs。

问题标签

问题标签对象具有以下属性。

属性	类型	描述
tag	string	问题标签名称。
value	string	问题标签值。

媒体类型 URL

媒体类型 URL 对象具有以下属性。

属性	类型	描述
name	string	媒体类型定义的 URL 名称。
url	string	媒体类型定义的 URL 值。

结果将仅包含启用事件菜单项的活动媒体类型的条目。属性中使用的宏将展开，但如果其中一个属性包含未展开的宏，则两个属性都将从结果中排除。有关支持的宏，请参见支持的宏。

获取

描述

```
integer/array problem.get(object parameters)
```

该方法允许根据给定的参数检索问题。

此方法用于检索未解决的问题。如果指定，还可以额外检索最近解决的问题。*Administration → General* 中定义了确定" 最近" 的时间段。在该时间段之前解决的问题不会保存在问题表中。要检索过去已解决的问题，请使用event.get 方法。

Attention:
如果 housekeeper 尚未消除这些问题，则此方法可能会返回已删除实体的问题。

Note:
此方法对于任何用户可用。可以在用户角色设置中撤销调用该方法的权限。更多信息请查看用户角色。

参数

`(object)` 定义所需输出的参数。

该方法支持以下参数。

参数	类型	描述
eventids	string/array	仅返回给定 ID 的问题。
groupids	string/array	仅返回属于给定主机组的对象创建的问题。
hostids	string/array	仅返回属于给定主机的对象创建的问题。
objectids	string/array	仅返回给定对象创建的问题。
source	integer	仅返回给定类型的问题。 有关支持的事件类型列表，请参阅问题事件对象页面。 默认值：0 - 由触发器创建的问题。
object	integer	仅返回由给定类型的对象创建的问题。 有关支持的对象类型列表，请参阅问题事件对象页面。 默认值：0 - 触发器。
acknowledged	boolean	`true` - 仅返回已确认的问题； `false` - 仅未确认。
suppressed	boolean	`true` - 仅返回被抑制的问题； `false` - 返回正常状态的问题。
severities	integer/array	仅返回给定事件严重性的问题。仅当对象是触发器时才适用。
evaltype	integer	标签搜索规则。 可用值： 0 - （默认）And/Or； 2 - Or。

参数	类型	描述
tags	对象数组	仅返回给定标签的问题。按标签精确匹配，按值和运算符不区分大小写搜索。 格式：`[{"tag": "<tag>", "value": "<value>", "operator": "<operator>"}, ...]`。 空数组返回所有问题。 可能的运算符类型： 0 - （默认）相似； 1 - 等于； 2 - 不相似； 3 - 不等于； 4 - 存在； 5 - 不存在。
recent	boolean	`true` - 返回 PROBLEM 和最近解决的问题（取决于 N 秒的 Display OK 触发器）。 默认值：`false` - 仅未解决的问题。
eventid_from	string	仅返回 ID 大于或等于给定 ID 的问题。
eventid_till	string	仅返回 ID 小于或等于给定 ID 的问题。
time_from	timestamp	仅返回在给定时间之后或在给定时间创建的问题。
time_till	timestamp	仅返回在给定时间之前或在给定时间创建的问题。

参数	类型	描述
selectAcknowledges	query	返回带有问题更新的 acknowledges 属性。问题更新按时间倒序排列。 问题更新对象具有以下属性： acknowledgeid - (string) 更新的 ID； userid - (string) 更新事件的用户 ID； eventid - (string) 更新事件的 ID； clock - (timestamp) 事件更新的时间； message - (string) 消息文本； action - (integer) 更新操作类型（参见event.acknowledge）； old_severity - 此更新操作之前的 (integer) 事件严重性； new_severity - 此更新操作后的 (integer) 事件严重性； 支持 count。
selectTags	query	返回带有问题标签的tags 属性。输出格式：[{"tag": "<tag>", "value": "<value>"}, ...]。
selectSuppressionData	query	返回带有维护列表的 suppression_data 属性： maintenanceid - (string) 维护的 ID； suppress_until - (integer) 出现问题的时间被压制。
sortfield	string/array	按给定的属性对结果进行排序。 可用值： eventid。

参数	类型	描述
countOutput	boolean	这些参数对所有的 get 方法是通用的，详情请参阅参考说明 。
editable	boolean	
excludeSearch	boolean	
filter	object	
limit	integer	
output	query	
preservekeys	boolean	
search	object	
searchByAny	boolean	
searchWildcardsEnabled	boolean	
sortorder	string/array	
startSearch	boolean	

返回值

(integer/array) 返回其中之一：

- 对象数组；
- 如果使用了 countOutput 参数，则为检索到的对象的数量。

示例

检索触发问题事件

从触发器"15112" 中检索最近的事件。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "problem.get",
    "params": {
        "output": "extend",
        "selectAcknowledges": "extend",
        "selectTags": "extend",
        "selectSuppressionData": "extend",
        "objectids": "15112",
        "recent": "true",
        "sortfield": ["eventid"],
        "sortorder": "DESC"
    },
    "auth": "67f45d3eb1173338e1b1647c4bdc1916",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "eventid": "1245463",
            "source": "0",
            "object": "0",
            "objectid": "15112",
            "clock": "1472457242",
            "ns": "209442442",
            "r_eventid": "1245468",
            "r_clock": "1472457285",
            "r_ns": "125644870",
            "correlationid": "0",
            "userid": "1",
```

```
            "name": "Zabbix agent on localhost is unreachable for 5 minutes",
            "acknowledged": "1",
            "severity": "3",
            "opdata": "",
            "acknowledges": [
                {
                    "acknowledgeid": "14443",
                    "userid": "1",
                    "eventid": "1245463",
                    "clock": "1472457281",
                    "message": "problem solved",
                    "action": "6",
                    "old_severity": "0",
                    "new_severity": "0"
                }
            ],
            "suppression_data": [
                {
                    "maintenanceid": "15",
                    "suppress_until": "1472511600"
                }
            ],
            "suppressed": "1",
            "tags": [
                {
                    "tag": "test tag",
                    "value": "test value"
                }
            ]
        }
    ],
    "id": 1
}
```


参见

- 告警
- 监控项
- 主机
- 自动发现规则
- 触发器

来源

ui/include/classes/api/services/CProblem.php 中的 CEvent::get()。

高可用节点

该类被设计用来与作为高可用性集群一部分的服务器节点或独立的服务器实例一起工作。

对象引用:

- 高可用节点

可用的方法:

- hanode.get - 检索节点

> 高可用节点对象

以下对象与操作 Zabbix servers 的高可用集群有关。

高可用节点

Note:
节点是由 Zabbix server 创建的，不能通过 API 修改。

高可用节点对象具有以下属性。

属性	类型	描述
ha_nodeid	string	节点的 ID。
name	string	使用 zabbix_server.conf 的 HANodeName 配置项给节点分配的名称。在独立模式下运行的服务器该配置项是空的。
address	string	节点连接的 IP 或 DNS 名称。
port	integer	节点运行的端口。
lastaccess	integer	心跳时间，节点最后一次更新的 t.i. 时间。UTC 时间戳。
status	integer	节点的状态。 可能的取值： 0 - 备用； 1 - 手动停止； 2 - 不可用； 3 - 活跃。

HA 节点获取

描述

`integer/array hanode.get(object parameters)`

该方法允许根据给定的参数检索一个高可用集群节点的列表。

Note:
这个方法只适用于 *Admin* 和 *Super admin* 用户类型。前往用户角色以了解更多信息。

参数

`(object)` 参数定义所期望的输出。

该方法支持以下参数。

参数	类型	描述
ha_nodeids	string/array	只返回具有给定节点 ID 的节点。

参数	类型	描述
filter	object	只返回那些与给定过滤器完全匹配的结果。 接受一个数组，其中键是属性名称，而值是一个单一的值或一个数组的值来匹配。 允许通过节点属性进行过滤：name、address 和 status.
sortfield	string/array	按给定的属性对结果进行排序。 可能的取值：name、lastaccess 和 status.
countOutput	flag	这些参数是所有 get 方法的共同参数，在参考注释中详细描述。
limit	integer	
output	query	
preservekeys	boolean	
sortorder	string/array	

返回值

返回（integer/array）其中之一：

- 一个对象的数组；
- 如果使用了 countOutput 参数，则为检索到的对象的数量。

示例

获取一个按状态排序的节点列表

请求：

```
{
    "jsonrpc": "2.0",
    "method": "hanode.get",
    "params": {
        "preservekeys": true,
        "sortfield": "status",
        "sortorder": "DESC"
    },
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
  "jsonrpc": "2.0",
  "result": {
    "ckuo7i1nw000h0sajj3l3hh8u": {
      "ha_nodeid": "ckuo7i1nw000h0sajj3l3hh8u",
      "name": "node-active",
      "address": "192.168.1.13",
```

```
      "port": "10051",
      "lastaccess": "1635335704",
      "status": "3"
    },
    "ckuo7i1nw000e0sajwfttc1mp": {
      "ha_nodeid": "ckuo7i1nw000e0sajwfttc1mp",
      "name": "node6",
      "address": "192.168.1.10",
      "port": "10053",
      "lastaccess": "1635332902",
      "status": "2"
    },
    "ckuo7i1nv000c0sajz85xcrtt": {
      "ha_nodeid": "ckuo7i1nv000c0sajz85xcrtt",
      "name": "node4",
      "address": "192.168.1.8",
      "port": "10052",
      "lastaccess": "1635334214",
      "status": "1"
    },
    "ckuo7i1nv000a0saj1fcdkeu4": {
      "ha_nodeid": "ckuo7i1nv000a0saj1fcdkeu4",
      "name": "node2",
      "address": "192.168.1.6",
      "port": "10051",
      "lastaccess": "1635335705",
      "status": "0"
    }
  },
  "id": 1
}
```


通过节点 ID 获取特定节点的列表

请求：

```
{
    "jsonrpc": "2.0",
    "method": "hanode.get",
    "params": {
        "ha_nodeids": ["ckuo7i1nw000e0sajwfttc1mp", "ckuo7i1nv000c0sajz85xcrtt"]
    },
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：


```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "ha_nodeid": "ckuo7i1nv000c0sajz85xcrtt",
            "name": "node4",
            "address": "192.168.1.8",
            "port": "10052",
            "lastaccess": "1635334214",
            "status": "1"
        },
        {
            "ha_nodeid": "ckuo7i1nw000e0sajwfttc1mp",
            "name": "node6",
            "address": "192.168.1.10",
            "port": "10053",
            "lastaccess": "1635332902",
```

```
            "status": "2"
        }
    ],
    "id": 1
}
```

获得一个停用的节点列表

请求：

```
{
    "jsonrpc": "2.0",
    "method": "hanode.get",
    "params": {
        "output": ["ha_nodeid", "address", "port"],
        "filter": {
            "status": 1
        }
    },
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "ha_nodeid": "ckuo7i1nw000g0sajjsjre7e3",
            "address": "192.168.1.12",
            "port": "10051"
        },
        {
            "ha_nodeid": "ckuo7i1nv000c0sajz85xcrtt",
            "address": "192.168.1.8",
            "port": "10052"
        },
        {
            "ha_nodeid": "ckuo7i1nv000d0sajd95y1b6x",
            "address": "192.168.1.9",
            "port": "10053"
        }
    ],
    "id": 1
}
```

获取备用节点的数量

请求：

```
{
    "jsonrpc": "2.0",
    "method": "hanode.get",
    "params": {
        "countOutput": true,
        "filter": {
            "status": 0
        }
    },
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": "3",
    "id": 1
}
```

检查特定 IP 地址的节点的状态

请求：

```
{
    "jsonrpc": "2.0",
    "method": "hanode.get",
    "params": {
        "output": ["name", "status"],
        "filter": {
            "address": ["192.168.1.7", "192.168.1.13"]
        }
    },
    "auth": "3a57200802b24cda67c4e4010b50c065",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "name": "node3",
            "status": "0"
        },
        {
            "name": "node-active",
            "status": "3"
        }
    ],
    "id": 1
}
```

来源

ui/include/classes/api/services/CHaNode.php 的 CHaNode::get()。

Zabbix API 在 **6.0** 中的变化

6.0.39 alert

Changes:
ZBX-26023 `alert.get`: *Admin* and *User* type users may now retrieve ”message” (0) type alert data about users in their user group.

6.0.38 alert

Changes:
ZBX-26258 `alert.get`: Parameter `selectUsers` now returns data based on the restrictions added to `user.get` method.
ZBX-26258 `alert.get`: *Admin* and *User* type users may now retrieve ”message” (0) type alert data only about their own user.

mediatype

Changes:
ZBX-26258 `mediatype.get`: parameter `selectUsers` now returns data based on the restrictions added to `user.get` method.

role

Changes:
ZBX-26258 `role.get`: parameter `selectUsers` now returns data based on the restrictions added to `user.get` method.

user

Changes:
ZBX-26258 `user.get`: when requesting user media, permissions, or role information, *Admin* and *User* type users may retrieve data only about their own user.
ZBX-26258 `user.get`: *Admin* and *User* type users may now retrieve only the following User object properties for their own user: `userid`, `attempt_clock`, `attempt_failed`, `attempt_ip`, `autologin`, `autologout`, `lang`, `name`, `refresh`, `roleid`, `rows_per_page`, `surname`, `theme`, `timezone`, `url`, `username`. For users in their user group: `userid`, `name`, `surname`, `username`.

usergroup

Changes:
ZBX-26258 `usergroup.get`: parameter `selectUsers` now returns data based on the restrictions added to `user.get` method.

6.0.34 mediatype

ZBX-25385 `mediatype.get`: Parameter `selectMessageTemplates` is now supported only for *Super admin* type users.
ZBX-25385 `mediatype.get`: *Admin* type users may now retrieve only the following Media type object properties: `mediatypeid`, `name`, `type`, `status`, `maxattempts`.
ZBX-25385 `mediatype.get`: When requesting user-related information of media types, *Admin* type users may now retrieve only data about their own user.

6.0.22rc1 hostprototype

Changes:
ZBX-23284 `hostprototype.get`, `hostprototype.update`: returned support for `group_prototypeid` property of Group prototype objects.

6.0.19 动作

变化：ZBX-21804 `action.create`, `action.get`, `action.update`, `action.delete`: 取消了对主机、主机组、模板、触发器、proxy 的写入权限要求。

6.0.15 脚本

变化：ZBX-19466 将 `script` 对象的验证更改为通过两个属性的组合来保证唯一性：`name` 和 `menu_path`。

6.0.14 用户

变化：ZBXNEXT-8012 `user.checkAuthentication`: 添加了新的参数 `token`。

6.0.13 配置

变化：ZBXNEXT-7951 `configuration.import`, `configuration.importcompare`：对于 `templateLinkage` 的选项 `deleteMissing: true` 现在将取消链接缺失的模板（而不是取消链接并清除）。

发现规则

变化：ZBXNEXT-7951 `discoveryrule.update`：现在可以更新 `uuid` 参数。

图形

变化：ZBXNEXT-7951 `graph.update`：现在可以更新 `uuid` 参数。

图形原型

变化：ZBXNEXT-7951 `graphprototype.update`：现在可以更新 `uuid` 参数。

主机组

变化：ZBXNEXT-7951 `hostgroup.update`：现在可以更新 `uuid` 参数。

主机原型

变化：ZBXNEXT-7951 hostprototype.update：现在可以更新 uuid 参数。

HTTP 测试

变化：ZBXNEXT-7951 httptest.update：现在可以更新 uuid 参数。

监控项

变化：ZBXNEXT-7951 item.update：现在可以更新 uuid 参数。

监控项原型

变化：ZBXNEXT-7951 itemprototype.update：现在可以更新 uuid 参数。

模板

变化：ZBXNEXT-7951 template.update：现在可以更新 uuid 参数。

模板仪表盘

变化：ZBXNEXT-7951 templatedashboard.update：现在可以更新 uuid 参数。

触发器

变化：ZBXNEXT-7951 trigger.update：现在可以更新 uuid 参数。

触发器原型

变化：ZBXNEXT-7951 triggerprototype.update：现在可以更新 uuid 参数。

值映射

变化：ZBXNEXT-7951 valuemap.update：现在可以更新 uuid 参数。

6.0.9 用户

变化：ZBXNEXT-7971 user.create, user.update：将 “url” 字段的最大长度增加到 2048 个字符。

6.0.7 图形

变化：ZBX-7706 graph.get：图形的可用性不再依赖于图形中”ymin_itemid” 和”ymax_itemid” 字段指定的监控项的权限。具有链接到不可访问监控项的最小或最大 Y 轴的图形将仍然是可访问的，但最小/最大 Y 轴的工作方式将与指定的计算方法为” 已计算” 时的工作方式相同。

图形原型

变化：
ZBX-7706graphprototype.get：图形原型可用性不依赖于图形原型 “ymin_itemid” 和 “ymax_itemid” 中指定监控项的权限字段。

6.0.3 发现规则

Bug 修复：
ZBX-19118 discoveryrule.create、discoveryrule.update：不再需要属性 interfaceid 来创建/更新 HTTP 代理类型 LLD 规则。

发现规则

Bug 修复：
ZBX-19118 item.create、item.update：不再需要属性 interfaceid 来创建/更新 HTTP 代理类型的监控项。

监控项原型

Bug 修复：
ZBX-19118 itemprototype.create、itemprototype.update：不再需要属性 interfaceid 来创建/更新 HTTP 代理类型监控项原型。

附录 1. 参考说明

注释　数据类型

Zabbix API 支持输入以下数据类型：

类型	描述
boolean	布尔值，接受 `true` 或 `false`。
flag	如果传递的值不为 `null` 和 `false`，认为其值是 `true`。
integer	整数。
float	浮点数。
string	文本字符串。
text	长文本字符串。
timestamp	Unix 时间戳。
array	有序的值序列，即普通数组。
object	关联数组。
query	用于定义应返回的数据。 可定义为仅返回指定属性的属性名称数组，或者预定值的其中一个： `extend` - 返回所有的对象属性； `count` - 返回检索到的记录数量，仅支持某些子查询。

Attention:
Zabbix API 始终仅以字符串或数组的形式返回值。

属性标签

一些对象属性用短标签来描述它们的行为。可使用以下标签：

- *readonly* - 属性值是自动设置的，不能被客户端定义或修改；
- *constant* - 属性值可以在创建对象时被设置，创建后不能被修改。

预留 **ID** 值 **“0”**　预留 ID 值 “0”，可以用来过滤元素和删除引用的对象。例如，从主机中删除一个引用的代理，proxy_hostid 应该设置为 0 (”proxy_hostid”: ”0”)，或者要过滤被 zabbix server 监控的主机，proxyids 选项应该被设置为 0 (”proxyids”: ”0”)。

通用 **“get”** 方法参数　所有的 `get` 方法都支持以下参数：

参数	类型	描述
countOutput	boolean	返回结果中的记录数，而不是实际的数据。
editable	boolean	如果设置为 `true`，则只返回用户具有写权限的对象。 默认：`false`。
excludeSearch	boolean	返回与在 `search` 参数中给定条件不匹配的结果。
filter	object	仅返回与给定过滤条件完全匹配的结果。 接受一个数组，键是属性名，值是单个值或者要匹配值的数组。 不适用于 `text` 字段。
limit	integer	限制返回记录的数量。
output	query	要返回的对象属性。 默认：`extend`。
preservekeys	boolean	在结果数组中，用 ID 作为键。

参数	类型	描述
search	object	返回给定通配符（不区分大小写）匹配到的结果。 接受一个数组，键是属性名，值是要搜索的字符串。如果没有给出其他选项，将会执行 LIKE "%…%" 搜索。 仅适用于 string 和 text 字段。
searchByAny	boolean	如果设置为 true，则返回与 filter or search 参数中给定的任何条件匹配的结果，而不是所有条件。 默认：false。
searchWildcardsEnabled	boolean	如果设置为 true，则可以在 search 参数中使用"*" 作为通配符。 默认：false。
sortfield	string/array	按照给定的属性对结果进行排序。 可用于排序的属性列表，请参考特定 API get 方法描述。宏在排序前不会被展开。 如果没有给出特定的值，数据会无序返回。
sortorder	string/array	排序顺序。如果传递数组，则每个值都将与 sortfield 参数中给定的对应属性匹配。 可用值： ASC - (默认) 升序； DESC - 降序。
startSearch	boolean	search 参数比较字段的开始，即执行 LIKE "…%" 搜索。 如果 searchWildcardsEnabled 设置为 true，则忽略。

示例　用户权限检查

用户是否有权限修改以 “MySQL” 或 “Linux” 开头的主机？

请求：

```
{
    "jsonrpc": "2.0",
    "method": "host.get",
    "params": {
        "countOutput": true,
        "search": {
            "host": ["MySQL", "Linux"]
        },
        "editable": true,
        "startSearch": true,
        "searchByAny": true
    },
    "auth": "766b71ee543230a1182ca5c44d353e36",
    "id": 1
```

```
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": "0",
    "id": 1
}
```

Note:
结果 0，表示没有具有读/写权限的主机。

不匹配统计

统计名称中不包含 “ubuntu” 的主机数量。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "host.get",
    "params": {
        "countOutput": true,
        "search": {
            "host": "ubuntu"
        },
        "excludeSearch": true
    },
    "auth": "766b71ee543230a1182ca5c44d353e36",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": "44",
    "id": 1
}
```

使用通配符搜索主机

查找名称中包含单词 “server”，并且接口端口是 “10050” 或 “10071” 的主机。结果按照主机名称倒序排序，并且限制返回 5 台主机。

请求：

```
{
    "jsonrpc": "2.0",
    "method": "host.get",
    "params": {
        "output": ["hostid", "host"],
        "selectInterfaces": ["port"],
        "filter": {
            "port": ["10050", "10071"]
        },
        "search": {
            "host": "*server*"
        },
        "searchWildcardsEnabled": true,
        "searchByAny": true,
        "sortfield": "host",
        "sortorder": "DESC",
        "limit": 5
    },
    "auth": "766b71ee543230a1182ca5c44d353e36",
```

```
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "hostid": "50003",
            "host": "WebServer-Tomcat02",
            "interfaces": [
                {
                    "port": "10071"
                }
            ]
        },
        {
            "hostid": "50005",
            "host": "WebServer-Tomcat01",
            "interfaces": [
                {
                    "port": "10071"
                }
            ]
        },
        {
            "hostid": "50004",
            "host": "WebServer-Nginx",
            "interfaces": [
                {
                    "port": "10071"
                }
            ]
        },
        {
            "hostid": "99032",
            "host": "MySQL server 01",
            "interfaces": [
                {
                    "port": "10050"
                }
            ]
        },
        {
            "hostid": "99061",
            "host": "Linux server 01",
            "interfaces": [
                {
                    "port": "10050"
                }
            ]
        }
    ],
    "id": 1
}
```

加上”preservekeys” 参数使用通配符搜索主机

如果将参数 “preservekeys” 添加到上一个请求中，结果会返回一个关联数组，键是对象的 id。

请求：

```
{
    "jsonrpc": "2.0",
```

```
    "method": "host.get",
    "params": {
        "output": ["hostid", "host"],
        "selectInterfaces": ["port"],
        "filter": {
            "port": ["10050", "10071"]
        },
        "search": {
            "host": "*server*"
        },
        "searchWildcardsEnabled": true,
        "searchByAny": true,
        "sortfield": "host",
        "sortorder": "DESC",
        "limit": 5,
        "preservekeys": true
    },
    "auth": "766b71ee543230a1182ca5c44d353e36",
    "id": 1
}
```

响应：

```
{
    "jsonrpc": "2.0",
    "result": {
        "50003": {
            "hostid": "50003",
            "host": "WebServer-Tomcat02",
            "interfaces": [
                {
                    "port": "10071"
                }
            ]
        },
        "50005": {
            "hostid": "50005",
            "host": "WebServer-Tomcat01",
            "interfaces": [
                {
                    "port": "10071"
                }
            ]
        },
        "50004": {
            "hostid": "50004",
            "host": "WebServer-Nginx",
            "interfaces": [
                {
                    "port": "10071"
                }
            ]
        },
        "99032": {
            "hostid": "99032",
            "host": "MySQL server 01",
            "interfaces": [
                {
                    "port": "10050"
                }
            ]
        },
        "99061": {
```

```
            "hostid": "99061",
            "host": "Linux server 01",
            "interfaces": [
                {
                    "port": "10050"
                }
            ]
        }
    },
    "id": 1
}
```

附录 **2.** 从 **5.4** 到 **6.0** 的变更记录

向下不兼容的变更 动作

变更：
ZBXNEXT-6755 action.create，action.update：重命名参数 acknowledge_operations 变更为 update_operations。
ZBXNEXT-6755 action.get：重命名参数 selectAcknowledgeOperations 变更为 selectUpdateOperations。
ZBXNEXT-6920 action.create，action.update：添加了对方法参数的严格校验。

审计日志

变更：
ZBXNEXT-6715 审计日志对象：删除对属性 note 的支持。
ZBXNEXT-6715 审计日志对象：删除对 resourcetype 值为 2（Zabbix 配置）和值为 7（Graph 元素）的支持。
ZBXNEXT-6715 审计日志对象：删除对 action 值为 5（可用）和值 6（不可用）的支持。
ZBXNEXT-6715 auditlog.get：删除对参数 selectDetails 的支持。
ZBXNEXT-6718 审计日志对象：删除对 action 值为 3（登录）的支持。

主机组

变更：
ZBXNEXT-6868 hostgroup.massupdate：hosts 和 templates 字段现在是必填项。
ZBXNEXT-6868 hostgroup.massadd，hostgroup.massupdate, hostgroup.massremove：添加了对方法参数的严格校验。

监控主机原型

变更：ZBXNEXT-6959 hostprototype.get：取消了对群组链接和群组原型 API 对象属性 group_prototypeid、hostid、templateid 的支持。ZBXNEXT-6959 hostprototype.update：取消了修改继承监控主机原型的只读字段 host、name、custom_interfaces、interfaces、groupLinks、groupPrototypes、templates、tags、macros、inventory_mode 的能力。ZBXNEXT-6959 hostprototype.create、hostprototype.update、hostprototype.delete：增加了对方法参数的严格验证。

图标映射

变更：
ZBXNEXT-6914 iconmap.create，iconmap.update：删除对图标映射对象属性 sortorder 的支持。

维护

变更：
ZBXNEXT-6890 maintenance.create，maintenance.update，maintenance.delete：添加了对方法参数的严格校验。
ZBXNEXT-6890 maintenance.get，maintenance.update：删除对时间段对象参数 timeperiodid 的支持。

媒介类型

变更：
ZBXNEXT-6885 mediatype.create，mediatype.update：添加了对方法参数的严格校验。

角色

变更：
ZBXNEXT-6787 动作对象：删除对 name 属性值为 manage_services 的支持。

service

Changes:
ZBXNEXT-6999 added uuid, description and created_at properties.
ZBXNEXT-6999 dropped support for the showsla, goodsla and times properties.
ZBXNEXT-6800 changed status value "0" meaning from "OK" to "Not classified".
ZBXNEXT-3022 dropped support of service.adddependencies, service.addtimes, service.deletedependencies, service.deletetimes.
ZBXNEXT-6674 dropped support of property triggerid.

ZBXNEXT-6999 service.get: dropped support for showsla, selectAlarms, selectTimes parameters.
ZBXNEXT-6999 service.getsla: dropped support for the method.
ZBXNEXT-6999 sla.get, sla.create, sla.update, sla.delete, sla.getsli methods added.
ZBXNEXT-6999 service.get: added support for sorting by serviceid, status and created_at.
ZBXNEXT-6999 service.get: added support for slaids parameter; added support for filtering by uuid.
ZBXNEXT-6999 service.create, service.update: dropped support for showsla, goodsla and times parameters.
ZBXNEXT-2406 service.getsla: removed status and problems properties from response of request with intervals parameter.
ZBXNEXT-3022 service.create, service.update: dropped support of parameters dependencies and parentid.
ZBXNEXT-3022 service.get: dropped support of parameters selectParent, selectDependencies and selectParentDependencies.
ZBXNEXT-6674 service.get: dropped support of parameter selectTrigger.

模板

变更：
ZBXNEXT-6867 template.create，template.update，template.delete，template.massadd，template.massupdate，template.massremove：添加了对方法参数的严格校验。
ZBXNEXT-6867 template.create，template.update，template.massadd，template.massupdate：删除对参数 hosts 的支持。
ZBXNEXT-6867 template.massremove：删除对参数 hostids 的支持。

触发器

变更：
ZBXNEXT-6867 trigger.adddependencies，trigger.deletedependencies：删除编辑继承触发器依赖的功能。

其他变更和 **Bug** 修复 动作

变更：
ZBXNEXT-6755 添加对 conditiontype 值为 27（服务）和值为 28（服务名称）的支持。
ZBXNEXT-6250 action.get，action.create，action.update：添加新属性 notify_if_canceled。

审计日志

变更：
ZBXNEXT-6999 添加对 sla 资源的支持。
ZBXNEXT-6715 审计日志对象：添加对属性 username，recordsetid，details 的支持。
ZBXNEXT-6718 审计日志对象：添加对 action 值为 8（登录），9（登录失败）和 10（历史清除）的支持。

身份认证

变更：
ZBXNEXT-4029 添加新的密码策略字段 passwd_min_length 和 passwd_check_rules。

仪表盘

变更：
ZBXNEXT-6999 添加对 SLA 和 Service 字段类型的支持。
ZBXNEXT-6966 添加对新新控件类型 item 的支持。

历史

变更：
ZBXNEXT-6714 添加新方法 history.clear。

管家

变更：
ZBXNEXT-6755 添加对属性 hk_events_service 的支持。

监控项

变更：
ZBXNEXT-7049 item.get，item.create，item.update：为 Prometheus 模式预处理步骤添加第三个参数。第二个参数现在将会决定一个聚合方法：value，label，function。第三个参数现在将包含聚合方法 label 或聚合方法 function 的一个聚合函数的 Prometheus 输出。

监控项原型

变更：
ZBXNEXT-7049 itemprototype.get，itemprototype.create，itemprototype.update：为 Prometheus 模式预处理步骤添加第三个参数。第二个参数现在将会决定一个聚合方法：value，label，function。第三个参数现在将包含聚合方法 label 或聚合方法 function 的一个聚合函数的 Prometheus 输出。

维护

变更：
ZBXNEXT-6890 maintenance.create，maintenance.update：参数 groupids 现在已被弃用。改为使用 groups。
ZBXNEXT-6890 maintenance.create，maintenance.update：参数 hostids 现在已被弃用。改为使用 hosts。
ZBXNEXT-6890 maintenance.create，maintenance.update：参数 groups，hosts，timeperiods 和 tags 的类型由 array 变更为 object/array。

媒介类型

变更：
ZBXNEXT-6755 消息模板对象：添加对 conditiontype 值为 4（服务状态更新时创建的事件）的支持。

代理

变更：
ZBXNEXT-6889 proxy.create，proxy.update：删除对 interface.interfaceid 和 interface.hostid 属性的支持。

正则表达式

变更：
ZBXNEXT-6717 添加 regexp.get，regexp.create，regexp.update 和 regexp.delete API 接口。

角色

变更：ZBXNEXT-6787 添加了对新属性 services.read.mode、services.read.list、services.read.tag、services.write.mode、services.write.list 和 services.write.tag 的支持。ZBXNEXT-3022 在 action 对象的 name 属性中添加了对值 manage_services 的支持。ZBXNEXT-6999 在 action 对象的 name 属性中添加了对值 manage_sla 的支持。

服务

变更：
ZBXNEXT-3022 service.create，service.update：添加对参数 children，parents 和 tags 的支持。
ZBXNEXT-3022 service.get：添加对参数 evaltype，tags，selectChildren，selectParents，selectTags 的支持。
ZBXNEXT-6674 service.create，service.update：添加对参数 problem_tags 的支持。
ZBXNEXT-3022 service.get：添加对参数 problem_tags，without_problem_tags 和 selectProblemTags 的支持。
ZBXNEXT-6800 服务对象：添加对属性 weight，propagation_rule 和 propagation_value 的支持。
ZBXNEXT-6800 服务对象：添加对 status 值为-1（OK）的支持。
ZBXNEXT-6800 service.create，service.update：添加对参数 status_rules 的支持。
ZBXNEXT-6800 service.get：添加对参数 selectStatusRules 的支持。

ZBXNEXT-6800 service.get：添加对参数 selectAlarms 的 count 的支持。
ZBXNEXT-6787 服务对象：添加新属性 readonly。
ZBXNEXT-2406 service.get：添加对参数 deep_parentids 和 selectProblemEvents 的支持。

设置

变更：
ZBXNEXT-6715 settings.get，settings.update：添加对参数 auditlog_enabled 的支持。
ZBXNEXT-6945 settings.get，settings.update：添加对参数 geomaps_tile_provider，geomaps_tile_url，geomaps_max_zoom 和 geomaps_attribution 的支持。

服务等级协议

变化：
ZBXNEXT-6999 添加了新的 API sla 方法：sla.create、sla.delete、sla.get、sla.getsli、sla.update。

模板化仪表盘

变更：
ZBXNEXT-6966 添加对新控件类型 item 的支持。

用户

变更：
ZBXNEXT-4029 user.create 和 user.update：根据密码策略实现密码强度验证。
ZBXNEXT-6718 添加新方法 user.unblock。

用户组

变更：
ZBXNEXT-6866 usergroup.create，usergroup.update：userids 参数现在已被弃用。改为使用 users。

20. 模块

概述 可以通过添加第三方模块或开发自己的模块来增强 Zabbix 前端功能，而无需更改 Zabbix 的源代码。

请注意，模块代码将以与 Zabbix 源代码相同的权限运行。这意味着：

- 第三方模块可能是有害的。您必须信任您正在安装的模块;
- 第三方模块代码中的错误可能会使前端崩溃。如果发生这种情况，只需从前端删除模块代码即可。重新加载 Zabbix 前端后，您会看到一条消息，指出某些模块不存在。转到Module administration (在 *Administration* → *General* → *Modules*) 然后再次单击 *Scan directory* 以从数据库中删除不存在的模块。

安装 请始终阅读特定模块的安装手册。建议逐个安装新模块，以便更容易的找到故障。

在安装模块之前：

- 确保您已从受信任的来源下载了该模块。安装有害代码可能会导致后果，例如数据丢失
- 同一模块的不同版本（相同的 ID）可以并行安装，但一次只能启用一个版本

安装模块的步骤：

- 将模块解压缩到 Zabbix 前端的”modules” 文件夹中的自己的文件夹中
- 确保您的模块文件夹至少包含 manifest.json 文件
- 导航到Module administration 并且单击 *Scan directory* 按钮
- 新模块将与其版本，作者，描述和状态一起出现在列表中
- 通过单击其状态启用模块

排错：

问题	解决方案
模块未出现在列表中	确保 manifest.json 文件存在于 Zabbix 前端的'modules/your-module/' 文件夹中。如果这样做，则意味着该模块不适合当前的 Zabbix 版本。如果 manifest.json 文件不存在，则可能是因为您解压到了错误的目录中。
前端崩溃	模块代码与当前的 Zabbix 版本或服务器配置不兼容。请删除模块文件并重新加载前端。您将收到一条通知，指出某些模块不存在。转到Module administration 然后再次单击 *Scan directory* 以从数据库中删除不存在的模块。
出现有关相同命名空间、*ID* 或操作的错误消息	这是由于新模块尝试注册已由其他已启用模块注册的命名空间、ID 或操作。在启用新模块之前，禁用冲突的模块（在错误消息中提到的）。
出现技术错误消息	向默块开发者报告错误

开发模块　模块是用 PHP 语言编写的。Model-view-controller（MVC）软件模式设计是首选，因为它也用于 Zabbix 前端，并且将简化开发。PHP 严格输入也是受欢迎的，但不是强制性的。

请注意，使用模块，您可以轻松地将新菜单项以及相应的视图和操作添加到 Zabbix 前端。目前无法通过模块注册新的 API 或创建新的数据库表。

模块结构

每个模块都是一个目录（位于 `modules` 目录中），其子目录包含控制器，视图和任何其他代码：

```
example_module_directory/           （必需的）
    manifest.json                   （必需的）  元数据和动作定义。
    Module.php                                  模块初始化和事件处理。
    actions/                                    动作控制器文件。
        SomethingView.php
        SomethingCreate.php
        SomethingDelete.php
        data_export/
            ExportAsXml.php
            ExportAsExcel.php
    views/                                      视图文件。
        example.something.view.php
        example.something.delete.php
        js/                                     视图中使用的JavaScript文件。
            example.something.view.js.php
    partials/                                   部分视图文件。
        example.something.reusable.php
        js/                                     部分视图中使用的JavaScript文件。
            example.something.reusable.js.php
```

如您所见，自定义模块目录中唯一必需的文件是 `manifest.json`。如果没有此文件，模块将无法注册。`Module.php` 负责注册菜单项和处理诸如'onBeforeAction' 和'onTerminate' 之类的事件。*actions*、*views* 和 *partials* 目录包含模块操作所需的 PHP 和 JavaScript 代码。

命名约定

在创建模块之前，重要的是要就不同模块项（如目录和文件）的命名约定达成一致，以便我们可以保持良好的组织状态。您也可以在上面的模块结构 部分找到示例。

项目	命名规则	示例
模块目录	小写字母 [a-z], 下划线和小数	example_v2
动作子目录	小写字母 [a-z], 下划线和小数	data_export
动作文件	骆驼拼写法，以动作类型结尾	SomethingView.php
视图和部分文件	小写字母 [a-z] 用点分隔的单词 前缀为 `module.` 后跟模块名称 结尾，显示操作类型和.php 文件扩展名	module.example.something.view.php
Javascript 文件	与视图文件和部分文件的规则相同，但文件扩展名.js.php 除外。	module.example.something.view.js.php

请注意，'module' 前缀和名称包含对于视图和部分文件名是必需的，除非您需要覆盖 Zabbix 核心视图或部分视图。但是，此规则不适用于操作文件名。

清单准备

每个模块都应该有一个 manifest.json 文件，其中包含以下 JSON 格式的字段：

参数	必填	类型	默认值	描述
manifest_version	是	Double	-	模块的清单版本。当前支持的版本为 **1**。
id	是	String	-	模块标识。只能同时启用一个具有给定 ID 的模块。
name	是	String	-	“管理” 部分中显示的模块名称。
version	是	String	-	“管理” 部分中显示的模块版本。
namespace	是	String	-	Module.php 和操作类的 PHP 命名空间。
author	否	String	””	“管理” 部分中显示的模块作者。
url	否	String	””	“管理” 部分中显示的模块 URL。
description	否	String	””	“管理” 部分中显示的模块描述。
actions	否	Object	{}	注册此模块的操作。查看 “动作”
config	否	Object	{}	模块配置

有关参考，请参阅Reference部分中 manifest.json 的示例。

动作

该模块将控制在 manifest.json 文件中的 *actions* 对象中定义的前端操作。这样可以定义新操作。同样，您可以重新定义现有操作。每个操作键应表示操作名称，相应的值应包含 `class` 以及可选的 `layout` 和 `view` 键。

一个操作由四个对应项定义：名称、控制器、视图和布局。数据验证和准备通常在控制器中完成，输出格式化在视图或部分完成，布局负责用菜单、页眉、页脚等元素装饰页面。

操作模块必须在 manifest.json 文件中定义为 *actions* 对象：

参数	必需的	类型	默认值	描述
key	是	String	-	动作名字，用点区分的单词，由小写字母 [a-z] 组成。
class	是	String	-	动作类名字，包括 `actions` 目录中的子目录路径（如果使用）。
layout	否	String	”layout.htmlpage”	动作布局。
view	否	String	null	动作视图。

有几个预定义的布局，如 `layout.json` 或 `layout.xml`。这些操作适用于产生与 HTML 不同的结果的操作。您可以在 app/views/目录中浏览预定义的布局，甚至可以创建自己的布局。

有时，只需重新定义某些操作的视图部分，使控制器保持不变。在这种情况下，只需将必要的视图以及文件放在模块的 `views` 目录中即可。

For reference, please see an example action controller file in the Reference section. Please do not hesitate to explore current actions of Zabbix source code, located in the app/ directory. 有关参考，请参阅Reference 部分中的示例操作控制器文件。请不要犹豫，探索 Zabbix 源代码的当前操作，文件位于 app/目录中。

Module.php

此可选 PHP 文件负责模块初始化和事件处理。类’Module’ 应在此文件中定义，用以扩展基类 `\Core\CModule`。必须在 manifest.json 文件中指定的命名空间中定义 Module 类。

```
<?php

namespace Modules\Example;
use Core\CModule as BaseModule;

class Module extends BaseModule {
    ...
}
```

有关参考，请参阅Reference 部分中 Module.php 的示例。

Reference 本节包含前面各节中介绍的不同模块元素的基本版本。

manifest.json

```
{
    "manifest_version": 1.0,
    "id": "example_module",
    "name": "Example module",
    "version": "1.0",
    "namespace": "Example",
    "author": "John Smith",
    "url": "http://module.example.com",
    "description": "Short description of the module.",
    "actions": {
        "example.something.view": {
            "class": "SomethingView",
            "view": "module.example.something.view"
        },
        "example.something.create": {
            "class": "SomethingCreate",
            "layout": null
        },
        "example.something.delete": {
            "class": "SomethingDelete",
            "layout": null
        },
        "example.something.export.xml": {
            "class": "data_export/ExportAsXml",
            "layout": null
        },
        "example.something.export.excel": {
            "class": "data_export/ExportAsExcel",
            "layout": null
        }
    },
    "config": {
        "username": "john_smith"
    }
}
```

Module.php

```
<?php declare(strict_types = 1);

namespace Modules\Example;

use APP;
use CController as CAction;

/**
 * Please see Core\CModule class for additional reference.
 */
class Module extends \Core\CModule {

    /**
     * Initialize module.
     */
    public function init(): void {
        // Initialize main menu (CMenu class instance).
        APP::Component()->get('menu.main')
            ->findOrAdd(_('Reports'))
                ->getSubmenu()
                    ->add((new \CMenuItem(_('Example wide report')))
                        ->setAction('example.report.wide.php')
                    )
                    ->add((new \CMenuItem(_('Example narrow report')))
                        ->setAction('example.report.narrow.php')
```

```
                    );
    }

    /**
     * Event handler, triggered before executing the action.
     *
     * @param CAction $action  Action instance responsible for current request.
     */
    public function onBeforeAction(CAction $action): void {
    }

    /**
     * Event handler, triggered on application exit.
     *
     * @param CAction $action  Action instance responsible for current request.
     */
    public function onTerminate(CAction $action): void {
    }
}
```

动作控制器

```
<?php declare(strict_types = 1);

namespace Modules\Example\Actions;

use CControllerResponseData;
use CControllerResponseFatal;
use CController as CAction;

/**
 * Example module action.
 */
class SomethingView extends CAction {

    /**
     * Initialize action. Method called by Zabbix core.
     *
     * @return void
     */
    public function init(): void {
        /**
         * Disable SID (Sessoin ID) validation. Session ID validation should only be used for actions whic
         * modification, such as update or delete actions. In such case Session ID must be presented in th
         * the URL would expire as soon as the session expired.
         */
        $this->disableSIDvalidation();
    }

    /**
     * Check and sanitize user input parameters. Method called by Zabbix core. Execution stops if false is
     *
     * @return bool true on success, false on error.
     */
    protected function checkInput(): bool {
        $fields = [
            'name'  => 'required|string',
            'email' => 'required|string',
            'phone' => 'string'
        ];

        // Only validated data will further be available using $this->hasInput() and $this->getInput().
        $ret = $this->validateInput($fields);
```

```
        if (!$ret) {
            $this->setResponse(new CControllerResponseFatal());
        }

        return $ret;
    }

    /**
     * Check if the user has permission to execute this action. Method called by Zabbix core.
     * Execution stops if false is returned.
     *
     * @return bool
     */
    protected function checkPermissions(): bool {
        $permit_user_types = [USER_TYPE_ZABBIX_ADMIN, USER_TYPE_SUPER_ADMIN];

        return in_array($this->getUserType(), $permit_user_types);
    }

    /**
     * Prepare the response object for the view. Method called by Zabbix core.
     *
     * @return void
     */
    protected function doAction(): void {
        $contacts = $this->getInput('email');

        if ($this->hasInput('phone')) {
            $contacts .= ', '.$this->getInput('phone');
        }

        $data = [
            'name' => $this->getInput('name'),
            'contacts' => $contacts
        ];

        $response = new CControllerResponseData($data);

        $this->setResponse($response);
    }
}
```

动作视图

```
<?php declare(strict_types = 1);

/**
 * @var CView $this
 */

$this->includeJsFile('example.something.view.js.php');

(new CWidget())
    ->setTitle(_('Something view'))
    ->addItem(new CDiv($data['name']))
    ->addItem(new CPartial('module.example.something.reusable', [
        'contacts' => $data['contacts']
    ]))
    ->show();
```

21. 附录

请使用侧边栏访问附录中的内容。

2 安装及配置

1 创建数据库

概述

在部署 Zabbix server 或 proxy 时必须要创建数据库。

本节提供创建 Zabbix 数据库的说明。每个受支持的数据库都有对应的创建说明。

Zabbix 唯一支持的编码是 UTF-8 。使用此编码没有已知的任何安全漏洞。应注意如果使用其他的编码，则存在已知的安全问题。

> **Note:**
> 如果从 Zabbix Git 存储库 安装 Zabbix，在进行下一步操作之前需要执行以下命令：

 `make dbschema`

MySQL/MariaDB

字符集 utf8（也称为 utf8mb3）和 utf8mb4 均受支持（分别使用 utf8_bin 和 utf8mb4_bin 校对），以便 Zabbix server/proxy 能够正确地与 MySQL 数据库协作。建议新安装使用 utf8mb4。

对于 Zabbix 6.0.11 及更新版本，在导入模式期间需要创建确定性触发器。在 MySQL 和 MariaDB 上，如果启用了二进制日志记录并且没有超级用户权限，且 MySQL 配置文件中没有设置 log_bin_trust_function_creators = 1，则需要设置 GLOBAL log_bin_trust_function_creators = 1。

如果您是从 Zabbix 包安装，请继续按照您平台的 指导 进行。

如果您是从源代码安装 Zabbix：

- 创建并配置数据库和用户。

```
mysql -uroot -p<password>

mysql> create database zabbix character set utf8mb4 collate utf8mb4_bin;
mysql> create user 'zabbix'@'localhost' identified by '<password>';
mysql> grant all privileges on zabbix.* to 'zabbix'@'localhost';
mysql> SET GLOBAL log_bin_trust_function_creators = 1;
mysql> quit;
```

- 将数据导入数据库。对于 Zabbix proxy 数据库，只应导入 `schema.sql`（不包括 images.sql 和 data.sql）。

```
cd database/mysql
mysql -uzabbix -p<password> zabbix < schema.sql
#### stop here if you are creating database for Zabbix proxy
mysql -uzabbix -p<password> zabbix < images.sql
mysql -uzabbix -p<password> zabbix < data.sql
```

`log_bin_trust_function_creators` 在成功导入模式后可以被禁用：

```
mysql -uroot -p<password>

mysql> SET GLOBAL log_bin_trust_function_creators = 0;
mysql> quit;
```

PostgreSQL

您需要拥有有权创建数据库对象的数据库用户。

如果您从 Zabbix **packages** 安装，请继续执行适用于您的平台的 instructions。

如果您从源码安装 Zabbix：

- 创建数据库用户。

以下 shell 命令将创建用户 zabbix。出现提示时指定密码并重复密码（注意，系统可能首先要求您输入 sudo 密码）：

```
sudo -u postgres createuser --pwprompt zabbix
```

- 创建数据库。

以下 shell 命令将创建数据库 zabbix（最后一个参数），并将先前创建的用户作为所有者（-O zabbix）。

```
sudo -u postgres createdb -O zabbix -E Unicode -T template0 zabbix
```

- 导入初始模式和数据（假设您在 Zabbix 源码的根目录中）。

对于 Zabbix 代理数据库，只应导入 schema.sql（不导入 images.sql 或 data.sql）。

```
cd database/postgresql
cat schema.sql | sudo -u zabbix psql zabbix
#### 如果您正在为 Zabbix 代理创建数据库，请在此处停止
cat images.sql | sudo -u zabbix psql zabbix
cat data.sql | sudo -u zabbix psql zabbix
```

> **Attention:**
> 上面的命令作为示例提供，可以在大多数 GNU/Linux 安装中使用。您可以使用不同的命令，例如：

psql -U <username>

 取决于您的系统/数据库的配置方式。如果您在设置数据库时遇到问题，请咨询您的数据库管理员。

TimescaleDB

创建和配置 TimescaleDB 的操作在单独的章节 中说明。

Oracle

创建和配置 Oracle 数据库的操作在单独的章节 中说明。

SQLite

仅 **Zabbix proxy** 支持使用 SQLite！

如果数据库不存在，将自动创建该数据库。

返回安装部分。

2 修复 **Zabbix** 数据库的字符集及字符序

MySQL/MariaDB

历史版本中，MySQL 及其衍生产品中使用的 UTF8（utf8mb3）只能实现最长 3 个字节的字符存储，那 4 个字节呢。从 MySQL 8.0.28 及 MariaDB 10.6.1 的版本之后，'utf8mb3' 被弃用，并会在未来某个时候被删除，而'utf8mb4' 将被新版本中'utf8' 字符集引用。在 Zabbix 6.0 中'utf8mb4' 字符集也得到了支持，为了避免将来发生未知的问题，我们强烈建议您使用'utf8mb4' 字符集。并且支持的 Unicode 字符也是切换到'utf8mb4' 的另一个优点。

> **Warning:**
> 由于 Zabbix 6.0 之前的版本不兼容 utf8mb4，在执行 utf8mb4 转换之前，请确保首先升级 Zabbix server 和 DB schema 为 6.0。

1. 检测数据库的字符集及字符序。

例:

```
mysql> SELECT @@character_set_database, @@collation_database;
+--------------------------+----------------------+
| @@character_set_database | @@collation_database |
+--------------------------+----------------------+
| latin2                   | latin2_general_ci    |
+--------------------------+----------------------+
```

或者:

```
mysql> SELECT @@character_set_database, @@collation_database;
+--------------------------+----------------------+
| @@character_set_database | @@collation_database |
+--------------------------+----------------------+
| utf8                     | utf8_bin             |
```

```
+--------------------------+----------------------+
```

当我们看到此处的字符集不是'utf8mb4_bin' 时，我们则需要更改它。

2. 停止 Zabbix 服务。

3. 将数据库进行备份！

4. 使用如下命令修复数据库的字符集及字符序:

```
alter database <your DB name> character set utf8mb4 collate utf8mb4_bin;
```

再次检测:

```
mysql> SELECT @@character_set_database, @@collation_database;
+--------------------------+----------------------+
| @@character_set_database | @@collation_database |
+--------------------------+----------------------+
| utf8mb4                  | utf8mb4_bin          |
+--------------------------+----------------------+
```

5. 导入下面的 sql 来修复每张表中各个列的字符集script mysql <your DB name> < utf8mb4_convert.sql

6. 执行如下命令:

```
              SET @ZABBIX_DATABASE = '<your DB name>';
If MariaDB →  set innodb_strict_mode = OFF;
              CALL zbx_convert_utf8();
If MariaDB →  set innodb_strict_mode = ON;
              drop procedure zbx_convert_utf8;
```

请注意，“utf8mb4” 会占用您相对较多的磁盘空间。

7. 如果没有错误信息 - 您可能希望备份一个修复过后的数据库。

8. 启动 Zabbix 服务.

3 为数据库升级主键

概述

自 Zabbix 6.0 起，主键会应用于新安装 Zabbix 数据库的所有表。

在这之前安装过的 Zabbix，本章节将提供手动升级所有表主键的说明。

此章节适用于如下数据库:

- MySQL
- PostgreSQL
- TimescaleDB
- Oracle

> **Attention:**
> 此页面上提供的说明专为高级用户设计。请注意，这些说明可能需要根据您的特定配置进行调整。

重要提示

- 确保在升级前备份数据库。
- 如果数据库使用分区，请联系数据库管理员或 Zabbix 支持团队寻求帮助。
- 强烈建议在升级时停止 Zabbix 服务器。但是，如果绝对必要，有一种方法可以在服务器运行时执行升级（仅适用于没有 TimescaleDB 的 MySQL、MariaDB 和 PostgreSQL）。
- 成功升级到主键后，可以删除 CSV 文件。
- 可选地，Zabbix 前端可以切换到维护模式。
- 升级到主键应该在将 Zabbix 服务器升级到 6.0 之后完成。
- 在代理上，未使用的历史表可以通过执行 history_pk_prepare.sql 进行升级。

MySQL

导出和导入必须在 tmux/screen 中执行，以确保会话不会被丢弃。

另请参阅：重要说明

MySQL 8.0+ 和 mysqlsh

此方法可用于正在运行的 Zabbix server，但建议在升级时停止服务器。MySQL Shell (*mysqlsh*) 必须 已安装 并且能够连接到数据库。

- 以 root（推荐）或任何具有 FILE 权限的用户身份登录 MySQL 控制台。
- 启用 local_infile 变量启动 MySQL。
- 通过运行 `history_pk_prepare.sql` 重命名旧表并创建新表。

```
mysql -uzabbix -p<password> zabbix < /usr/share/zabbix-sql-scripts/mysql/history_pk_prepare.sql
```

- 导出和导入数据。

通过 mysqlsh 连接。如果使用套接字连接，可能需要指定路径。

```
sudo mysqlsh -uroot -S /run/mysqld/mysqld.sock --no-password -Dzabbix
```

运行（CSVPATH 可根据需要更改）：

```
CSVPATH="/var/lib/mysql-files";

util.exportTable("history_old", CSVPATH + "/history.csv", { dialect: "csv" });
util.importTable(CSVPATH + "/history.csv", {"dialect": "csv", "table": "history" });

util.exportTable("history_uint_old", CSVPATH + "/history_uint.csv", { dialect: "csv" });
util.importTable(CSVPATH + "/history_uint.csv", {"dialect": "csv", "table": "history_uint" });

util.exportTable("history_str_old", CSVPATH + "/history_str.csv", { dialect: "csv" });
util.importTable(CSVPATH + "/history_str.csv", {"dialect": "csv", "table": "history_str" });

util.exportTable("history_log_old", CSVPATH + "/history_log.csv", { dialect: "csv" });
util.importTable(CSVPATH + "/history_log.csv", {"dialect": "csv", "table": "history_log" });

util.exportTable("history_text_old", CSVPATH + "/history_text.csv", { dialect: "csv" });
util.importTable(CSVPATH + "/history_text.csv", {"dialect": "csv", "table": "history_text" });
```

- 按照post-migration instructions 删除旧表。

没有 mysqlsh 的 MariaDB/MySQL 8.0+

这种升级方法需要更多时间，只有在无法使用 *mysqlsh* 进行升级时才应使用。

表升级

- 以 root（推荐）或任何具有 FILE 权限的用户身份登录 MySQL 控制台。
- 启用 local_infile 变量启动 MySQL。
- 通过运行 `history_pk_prepare.sql` 重命名旧表并创建新表：

```
mysql -uzabbix -p<password> zabbix < /usr/share/zabbix-sql-scripts/mysql/history_pk_prepare.sql
```

停止服务器的迁移

max_execution_time 必须在迁移数据之前禁用以避免迁移期间超时。

```
SET @@max_execution_time=0;

INSERT IGNORE INTO history SELECT * FROM history_old;
INSERT IGNORE INTO history_uint SELECT * FROM history_uint_old;
INSERT IGNORE INTO history_str SELECT * FROM history_str_old;
INSERT IGNORE INTO history_log SELECT * FROM history_log_old;
INSERT IGNORE INTO history_text SELECT * FROM history_text_old;
```

遵循# 迁移后的说明来废弃旧的表。

运行中的服务器迁移

检查是否启用了导入/导出的路径：

```
mysql> SELECT @@secure_file_priv;
+-----------------------+
| @@secure_file_priv · |
+-----------------------+
```

```
| /var/lib/mysql-files/ |
+-----------------------+
```

如果 *secure_file_priv* 值是目录路径，则将对该目录中的文件执行导出/导入。在这种情况下，相应地编辑查询中文件的路径或将 *secure_file_priv* 值设置为升级时间的空字符串。

如果 *secure_file_priv* 值为空，则可以从任何位置执行导出/导入。

如果 *secure_file_priv* 值为 NULL，将其设置为包含导出表数据的路径（上例中的 “/var/lib/mysql-files/”）。

有关详细信息，请参阅 MySQL 文档。

max_execution_time 必须在导出数据之前禁用以避免导出期间超时。

```
SET @@max_execution_time=0;

SELECT * INTO OUTFILE '/var/lib/mysql-files/history.csv' FIELDS TERMINATED BY ',' ESCAPED BY '"' LINES TER
LOAD DATA INFILE '/var/lib/mysql-files/history.csv' IGNORE INTO TABLE history FIELDS TERMINATED BY ',' ESC

SELECT * INTO OUTFILE '/var/lib/mysql-files/history_uint.csv' FIELDS TERMINATED BY ',' ESCAPED BY '"' LINE
LOAD DATA INFILE '/var/lib/mysql-files/history_uint.csv' IGNORE INTO TABLE history_uint FIELDS TERMINATED

SELECT * INTO OUTFILE '/var/lib/mysql-files/history_str.csv' FIELDS TERMINATED BY ',' ESCAPED BY '"' LINES
LOAD DATA INFILE '/var/lib/mysql-files/history_str.csv' IGNORE INTO TABLE history_str FIELDS TERMINATED BY

SELECT * INTO OUTFILE '/var/lib/mysql-files/history_log.csv' FIELDS TERMINATED BY ',' ESCAPED BY '"' LINES
LOAD DATA INFILE '/var/lib/mysql-files/history_log.csv' IGNORE INTO TABLE history_log FIELDS TERMINATED BY

SELECT * INTO OUTFILE '/var/lib/mysql-files/history_text.csv' FIELDS TERMINATED BY ',' ESCAPED BY '"' LINE
LOAD DATA INFILE '/var/lib/mysql-files/history_text.csv' IGNORE INTO TABLE history_text FIELDS TERMINATED
```

按照迁移后说明 删除旧表。

PostgreSQL

导出和导入必须在 tmux/screen 中执行，以确保会话不会被丢弃。对于使用 TimescaleDB 的安装，请跳过此部分并继续阅读PostgreSQL + TimescaleDB。

另请参阅：重要说明

表升级

- 使用 history_pk_prepare.sql 重命名表：

```
sudo -u zabbix psql zabbix < /usr/share/zabbix-sql-scripts/postgresql/history_pk_prepare.sql
```

停止服务器的迁移

- 导出当前历史，将其导入临时表，然后将数据插入新表，同时忽略重复项：

```
INSERT INTO history SELECT * FROM history_old ON CONFLICT (itemid,clock,ns) DO NOTHING;

INSERT INTO history_uint SELECT * FROM history_uint_old ON CONFLICT (itemid,clock,ns) DO NOTHING;

INSERT INTO history_str SELECT * FROM history_str_old ON CONFLICT (itemid,clock,ns) DO NOTHING;

INSERT INTO history_log SELECT * FROM history_log_old ON CONFLICT (itemid,clock,ns) DO NOTHING;

INSERT INTO history_text SELECT * FROM history_text_old ON CONFLICT (itemid,clock,ns) DO NOTHING;
```

查看提高 INSERT 性能的技巧：PostgreSQL：批量加载大量数据, 检查点距离和 WAL 的数量.

- 按照post-migration instructions 删除旧表。

迁移正在运行的服务器

- 导出当前历史，将其导入临时表，然后将数据插入新表，同时忽略重复项：

```
\copy history_old TO '/tmp/history.csv' DELIMITER ',' CSV
CREATE TEMP TABLE temp_history (
    itemid                  bigint                                  NOT NULL,
    clock                   integer         DEFAULT '0'             NOT NULL,
```

```
    value                    DOUBLE PRECISION DEFAULT '0.0000'          NOT NULL,
    ns                       integer          DEFAULT '0'               NOT NULL
);
\copy temp_history FROM '/tmp/history.csv' DELIMITER ',' CSV
INSERT INTO history SELECT * FROM temp_history ON CONFLICT (itemid,clock,ns) DO NOTHING;

\copy history_uint_old TO '/tmp/history_uint.csv' DELIMITER ',' CSV
CREATE TEMP TABLE temp_history_uint (
    itemid                   bigint                                     NOT NULL,
    clock                    integer          DEFAULT '0'               NOT NULL,
    value                    numeric(20)      DEFAULT '0'               NOT NULL,
    ns                       integer          DEFAULT '0'               NOT NULL
);
\copy temp_history_uint FROM '/tmp/history_uint.csv' DELIMITER ',' CSV
INSERT INTO history_uint SELECT * FROM temp_history_uint ON CONFLICT (itemid,clock,ns) DO NOTHING;

\copy history_str_old TO '/tmp/history_str.csv' DELIMITER ',' CSV
CREATE TEMP TABLE temp_history_str (
    itemid                   bigint                                     NOT NULL,
    clock                    integer          DEFAULT '0'               NOT NULL,
    value                    varchar(255)     DEFAULT ''                NOT NULL,
    ns                       integer          DEFAULT '0'               NOT NULL
);
\copy temp_history_str FROM '/tmp/history_str.csv' DELIMITER ',' CSV
INSERT INTO history_str (itemid,clock,value,ns) SELECT * FROM temp_history_str ON CONFLICT (itemid,clock,n

\copy history_log_old TO '/tmp/history_log.csv' DELIMITER ',' CSV
CREATE TEMP TABLE temp_history_log (
    itemid                   bigint                                     NOT NULL,
    clock                    integer          DEFAULT '0'               NOT NULL,
    timestamp                integer          DEFAULT '0'               NOT NULL,
    source                   varchar(64)      DEFAULT ''                NOT NULL,
    severity                 integer          DEFAULT '0'               NOT NULL,
    value                    text             DEFAULT ''                NOT NULL,
    logeventid               integer          DEFAULT '0'               NOT NULL,
    ns                       integer          DEFAULT '0'               NOT NULL
);
\copy temp_history_log FROM '/tmp/history_log.csv' DELIMITER ',' CSV
INSERT INTO history_log SELECT * FROM temp_history_log ON CONFLICT (itemid,clock,ns) DO NOTHING;

\copy history_text_old TO '/tmp/history_text.csv' DELIMITER ',' CSV
CREATE TEMP TABLE temp_history_text (
    itemid                   bigint                                     NOT NULL,
    clock                    integer          DEFAULT '0'               NOT NULL,
    value                    text             DEFAULT ''                NOT NULL,
    ns                       integer          DEFAULT '0'               NOT NULL
);
\copy temp_history_text FROM '/tmp/history_text.csv' DELIMITER ',' CSV
INSERT INTO history_text SELECT * FROM temp_history_text ON CONFLICT (itemid,clock,ns) DO NOTHING;
```

- 按照post-migration instructions 删除旧表。

PostgreSQL + TimescaleDB

导出和导入必须在 tmux/screen 中执行，以确保会话不会被丢弃。Zabbix server 应该在升级期间关闭。

另请参阅：重要说明

- 使用 `history_pk_prepare.sql` 重命名表。

```
sudo -u zabbix psql zabbix < /usr/share/zabbix-sql-scripts/postgresql/history_pk_prepare.sql
```

Attention:
路径 history_pk_prepare.sql 已在 Zabbix 6.0.9 后更改。对于早于 6.0.9 的版本，应使用以下命令：


```
sudo -u zabbix psql zabbix < /usr/share/doc/zabbix-sql-scripts/postgresql/history_pk_prepare.sql
```

- 运行 TimescaleDB 超表迁移脚本（兼容 TSDB v2.x 和 v1.x 版本），根据压缩设置：
  - 如果启用了压缩(默认安装)，则从 /usr/share/zabbix-sql-scripts/postgresql/tsdb_history_pk_upgrade_with_co
    运行脚本：

```
cat /usr/share/zabbix-sql-scripts/postgresql/tsdb_history_pk_upgrade_with_compression/history_pk.
cat /usr/share/zabbix-sql-scripts/postgresql/tsdb_history_pk_upgrade_with_compression/history_pk_
cat /usr/share/zabbix-sql-scripts/postgresql/tsdb_history_pk_upgrade_with_compression/history_pk_
cat /usr/share/zabbix-sql-scripts/postgresql/tsdb_history_pk_upgrade_with_compression/history_pk_
cat /usr/share/zabbix-sql-scripts/postgresql/tsdb_history_pk_upgrade_with_compression/history_pk_
```

  - 如果禁用用压缩，则从 /usr/share/zabbix-sql-scripts/postgresql/tsdb_history_pk_upgrade_no_compression
    运行脚本：

```
cat /usr/share/zabbix-sql-scripts/postgresql/tsdb_history_pk_upgrade_no_compression/history_pk.sq
cat /usr/share/zabbix-sql-scripts/postgresql/tsdb_history_pk_upgrade_no_compression/history_pk_ui
cat /usr/share/zabbix-sql-scripts/postgresql/tsdb_history_pk_upgrade_no_compression/history_pk_lo
cat /usr/share/zabbix-sql-scripts/postgresql/tsdb_history_pk_upgrade_no_compression/history_pk_st
cat /usr/share/zabbix-sql-scripts/postgresql/tsdb_history_pk_upgrade_no_compression/history_pk_te
```

另请参阅：提示 以提高 INSERT 性能。

- 按照post-migration instructions 删除旧表。

Oracle

导出和导入必须在 tmux/screen 中执行，以确保会话不会被丢弃。Zabbix 服务器应该在升级期间关闭。

另请参阅：重要说明

表升级

- 安装 Oracle 数据泵（在 即时客户端工具包 中可用）。

有关性能提示，请参阅 Oracle Data Pump 文档。

- 使用 history_pk_prepare.sql 重命名表。

```
cd /usr/share/zabbix/zabbix-sql-scripts/database/oracle
sqlplus zabbix/password@oracle_host/service
sqlplus> @history_pk_prepare.sql
```

历史表批量迁移

- 为 Data Pump 准备目录。

Data Pump 必须具有这些目录的读写权限。

示例：

```
mkdir -pv /export/history
chown -R oracle:oracle /export
```

- 创建一个目录对象，并授予此对象读写权限给用于 Zabbix 认证的用户（以下示例中为'zabbix'）。在 *sysdba* 角色下运行：

```
create directory history as '/export/history';
grant read,write on directory history to zabbix;
```

- 导出表。将 N 替换为所需的线程数。

```
expdp zabbix/password@oracle_host/service \
    DIRECTORY=history \
    TABLES=history_old,history_uint_old,history_str_old,history_log_old,history_text_old \
    PARALLEL=N
```

- 导入表。将 N 替换为所需的线程数。

```
impdp zabbix/password@oracle_host/service \
    DIRECTORY=history \
    TABLES=history_uint_old \
 REMAP_TABLE=history_old:history,history_uint_old:history_uint,history_str_old:history_str,history_log_old
    data_options=SKIP_CONSTRAINT_ERRORS table_exists_action=APPEND  PARALLEL=N CONTENT=data_only
```

- 按照post-migration instructions 删除旧表。

历史表的单独迁移

- 为每个历史表准备 Data Pump 目录。Data Pump 必须具有这些目录的读写权限。

示例：

```
mkdir -pv /export/history /export/history_uint /export/history_str /export/history_log /export/history_tex
chown -R oracle:oracle /export
```

- 创建一个目录对象，并授予该对象读写权限给用于 Zabbix 认证的用户（以下示例中的'zabbix'）。在 *sysdba* 角色下运行：

```
create directory history as '/export/history';
grant read,write on directory history to zabbix;

create directory history_uint as '/export/history_uint';
grant read,write on directory history_uint to zabbix;

create directory history_str as '/export/history_str';
grant read,write on directory history_str to zabbix;

create directory history_log as '/export/history_log';
grant read,write on directory history_log to zabbix;

create directory history_text as '/export/history_text';
grant read,write on directory history_text to zabbix;
```

- 导出和导入每个表。将 N 替换为所需的线程数。

```
expdp zabbix/password@oracle_host:1521/xe DIRECTORY=history TABLES=history_old PARALLEL=N

impdp zabbix/password@oracle_host:1521/xe DIRECTORY=history TABLES=history_old REMAP_TABLE=history_old:his

expdp zabbix/password@oracle_host:1521/xe DIRECTORY=history_uint TABLES=history_uint_old PARALLEL=N

impdp zabbix/password@oracle_host:1521/xe DIRECTORY=history_uint TABLES=history_uint_old REMAP_TABLE=histo

expdp zabbix/password@oracle_host:1521/xe DIRECTORY=history_str TABLES=history_str_old PARALLEL=N

impdp zabbix/password@oracle_host:1521/xe DIRECTORY=history_str TABLES=history_str_old REMAP_TABLE=history

expdp zabbix/password@oracle_host:1521/xe DIRECTORY=history_log TABLES=history_log_old PARALLEL=N

impdp zabbix/password@oracle_host:1521/xe DIRECTORY=history_log TABLES=history_log_old REMAP_TABLE=history

expdp zabbix/password@oracle_host:1521/xe DIRECTORY=history_text TABLES=history_text_old PARALLEL=N

impdp zabbix/password@oracle_host:1521/xe DIRECTORY=history_text TABLES=history_text_old REMAP_TABLE=histo
```

- 按照迁移后指导 删除旧表。

迁移后

对于所有数据库，迁移完成后，执行以下操作：

- 验证一切是否按预期工作。
- 删除旧表：

```
DROP TABLE history_old;
DROP TABLE history_uint_old;
DROP TABLE history_str_old;
```

```
DROP TABLE history_log_old;
DROP TABLE history_text_old;
```

4 安全的连接数据库

概述

本章节提供了 Zabbix 的设置步骤和配置示例，用于在以下设备之间建立安全的 TLS 连接：

数据库	Zabbix 组件
MySQL	Zabbix frontend, Zabbix server, Zabbix proxy
PostgreSQL	Zabbix frontend, Zabbix server, Zabbix proxy

请参考各官方文档了解如何在 DBMS 中设置连接加密:

- MySQL: 源和副本复制数据库服务器。
- MySQL: 组复制等。数据库服务器。
- PostgreSQL 加密选项。

所有示例均基于 MySQL CE (8.0) 和 PgSQL (13) 的 GA 版本，可通过使用 AlmaLinux 8 的官方存储库获得。

要求

设置加密需要以下内容:

- 需要具有 OpenSSL 1.1.X 及以上版本的操作系统或其他替代方案。

Note:
不建议使用不再更新维护的操作系统，尤其是在新安装的情况下。

数据库引擎 (RDBMS) 的安装及维护由官方存储库的开发人员提供。操作系统通常自带的数据库版本较老，没有实现加密支持，例如基于 RHEL 7 及 PostgreSQL 9.2、MariaDB 5.5 均没有加密支持。

术语

通过设置此选项强制 Zabbix Server/proxy/前端使用 TLS 连接数据库:

- required - 使用无需身份检查的 TLS 作为数据传输的连接;
- verify_ca - 使用 TLS 连接并验证证书;
- verify_full - 使用 TLS 连接，验证证书及 DBHost 指定的数据库身份 (CN) 匹配的证书;

Zabbix 配置

前端连接数据库

在前端安装的过程中可以配置使用数据库的安全连接:

- ·在配置数据库连接步骤中勾选 *Database TLS encryption* 复选框以启用传输加密。
- ·选中 *TLS* 加密字段时出现的验证数据库证书复选框，以启用证书加密。

Note:
对于 MySQL，如果 *Database host* 设置为 localhost，*Database TLS encryption* 复选框是禁用的，因为连接使用 socket 文件 (Unix) 或共享内存 (Windows) 是不能加密的。对于 PostgreSQL，如果 *Database host* 字段的值以斜线开头或字段为空，则 *TLS encryption* 复选框被禁用。

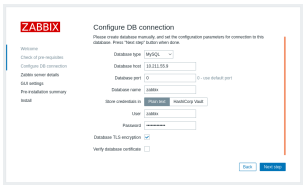
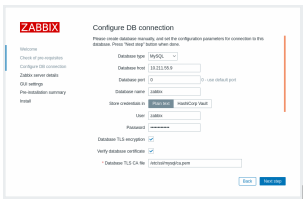
以下参数在证书模式 TLS 加密时可用 (如果两个复选框都勾选):

参数	描述
Database TLS CA file	指定有效的 TLS 证书颁发机构 (CA) 文件的完整路径
Database TLS key file	指定有效的 TLS 密钥文件的完整路径
Database TLS certificate file	指定有效 TLS 证书文件的完整路径
Database host verification	标记此复选框以激活主机验证. 对 MYSQL 禁用，因为 PHP MYSQL 库不允许跳过对端证书验证步骤。
Database TLS cipher list	指定有效密码的自定义列表。密码列表的格式必须符合 OpenSSL 标准。 仅对 MySQL 可用。

Attention:
TLS 参数必须指向有效的文件。如果指向的文件不存在或无效，则会导致授权错误.
如果证书文件权限是可写的，前端会在系统信息 报告中生成警告 “TLS 证书文件必须是只读的。” （仅当 PHP 用户是证书的所有者时才显示）。不支持受密码保护的证书。

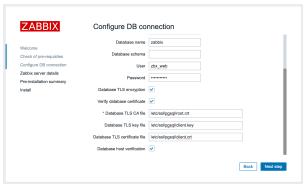
用例

Zabbix 前端使用 GUI 界面定义可能的选项：必需的，verify_ca，verify_full。在安装向导的 配置数据库连接步骤中指定必需的选项。这些选项以下列方式映射到配置文件（zabbix.conf.php）：

图形化设置	配置文件	描述	结果
	... // 用于 TLS 连接。 $DB['ENCRYPTION'] = true; $DB['KEY_FILE'] = ''; $DB['CERT_FILE'] = ''; $DB['CA_FILE'] = ''; $DB['VERIFY_HOST'] = false; $DB['CIPHER_LIST'] = ''; ...	勾选 数据库 *TLS* 加密 将 验证数据库证书选项留空 (不勾选)	启用 “必需” 模式。
	... $DB['ENCRYPTION'] = true;\\ $DB['KEY_FILE'] = ''; $DB['CERT_FILE'] = ''; $DB['CA_FILE'] = '/etc/ssl/mysql/ca.pem'; $DB['VERIFY_HOST'] = false; $DB['CIPHER_LIST'] = ''; ...	1. 勾选 数据库 *TLS* 加密和 验证数据库证书 2. 指定 数据库 *TLS CA* 文件的路径	启用'verify_ca' 模式。

图形化设置	配置文件	描述	结果

	...
	// 用于具有严格定义的密码列表的 TLS 连接。
	$DB['ENCRYPTION'] = true;
	$DB['KEY_FILE'] = '<key_file_path>';
	$DB['CERT_FILE'] = '<key_file_path>';
	$DB['CA_FILE'] = '<key_file_path>';
	$DB['VERIFY_HOST'] = true;
	$DB['CIPHER_LIST'] = '<cipher_list>';
	...
	
	Or:
	
	...
	// 用于未定义密码列表的 TLS 连接 - 由 MySQL 服务器选择。
	$DB['ENCRYPTION'] = true;
	$DB['KEY_FILE'] = '<key_file_path>';
	$DB['CERT_FILE'] = '<key_file_path>';
	$DB['CA_FILE'] = '<key_file_path>';
	$DB['VERIFY_HOST'] = true;
	$DB['CIPHER_LIST'] = '';
	...
		1. 勾选 数据库 *TLS* 加密和 验证数据库证书
		2. 指定 数据库 *TLS* 密钥文件的路径
		3. 指定 数据库 *TLS CA* 文件的路径
		4. 指定 数据库 *TLS* 证书文件的路径
		5. 指定 TLS 密码列表（可选）
			为 MySQL 启用'verify_full' 模式。

	... $DB['ENCRYPTION'] = true; $DB['KEY_FILE'] = '<key_file_path>'; $DB['CERT_FILE'] = '<key_file_path>'; $DB['CA_FILE'] = '<key_file_path>'; $DB['VERIFY_HOST'] = true; $DB['CIPHER_LIST'] = ' '; ... 1. 勾选 数据库 *TLS* 加密和 验证数据库证书 2. 指定 数据库 *TLS* 密钥文件的路径 3. 指定 数据库 *TLS CA* 文件的路径 4. 指定 数据库 *TLS* 证书文件的路径 5. 勾选 数据库主机验证	为 PostgreSQL 启用'verify_full' 模式。	

另请参阅： MySQL 加密配置示例，PostgreSQL 加密配置示例。

Zabbix 服务器/代理配置

可以使用 Zabbix server 和/或proxy 配置文件中的相应参数配置与数据库的安全连接。

配置	结果
无	未加密的数据库连接。
1. 设置 DBTLSConnect=required	服务器/代理与数据库建立 TLS 连接。不允许未加密的连接。
1. 设置 DBTLSConnect=verify_ca 2. 设置 DBTLSCAFile - 指定 TLS 证书颁发机构文件	服务器/代理在验证数据库证书后与数据库建立 TLS 连接。
1. 设置 DBTLSConnect=verify_full 2. 设置 DBTLSCAFile - 指定 TLS 证书颁发机构文件	服务器/代理在验证数据库证书和数据库主机身份后与数据库建立 TLS 连接。
1. 设置 DBTLSCAFile - 指定 TLS 证书颁发机构文件 2. 设置 DBTLSCertFile - 指定客户端公钥证书文件 3. 设置 DBTLSKeyFile - 指定客户端私钥文件	服务器/代理在连接到数据库时提供客户端证书。
1. 设置 DBTLSCipher - 客户端允许使用最高 TLS 1.2 的 TLS 协议进行连接的加密密码列表 或 DBTLSCipher13 - 客户端允许使用 TLS 1.3 协议进行连接的加密密码列表	(MySQL) TLS 使用提供的列表中的密码建立连接。 (PostgreSQL) 设置此选项将被视为错误。

1 MySQL 加密配置

概述

本文将以 CentOS 8.2 和 MySQL 8.0.21 为例，介绍如何配置数据库加密连接。

Attention:
如果 MySQL 主机设置为 localhost，加密选项将是不可用，这种情况下，Zabbix 前端和数据库之间使用 socket 文件连接 (在 Unix 上) 或共享内存 (在 Windows 上)，所以不能加密。

Note:
加密组合列表不限于本页列出的。还有更多组合可供选择。

先决条件

安装 MySQL 请参照 official repository.

有关如何使用 MySQL 存储库的详细信息请参照MySQL documentation

MySQL 服务器已准备好使用自签名证书接受安全连接.

若想查看哪些用户正在使用加密连接，请运行以下查询 (Performance Schema 选项应打开):

```
mysql> SELECT sbt.variable_value AS tls_version, t2.variable_value AS cipher, processlist_user AS user, pr
        FROM performance_schema.status_by_thread  AS sbt
        JOIN performance_schema.threads AS t ON t.thread_id = sbt.thread_id
        JOIN performance_schema.status_by_thread AS t2 ON t2.thread_id = t.thread_id
        WHERE sbt.variable_name = 'Ssl_version' and t2.variable_name = 'Ssl_cipher'
        ORDER BY tls_version;
```

所需模式

MySQL 配置

当前版本数据库的加密模式已经可以开箱即用加密模式。将在初始设置及启动后创建服务器端证书。

为主组件创建用户和角色：

```
mysql> CREATE USER
 'zbx_srv'@'%' IDENTIFIED WITH mysql_native_password BY '<strong_password>',
 'zbx_web'@'%' IDENTIFIED WITH mysql_native_password BY '<strong_password>'
 REQUIRE SSL
 PASSWORD HISTORY 5;

mysql> CREATE ROLE 'zbx_srv_role', 'zbx_web_role';
```

```
mysql> GRANT SELECT, UPDATE, DELETE, INSERT, CREATE, DROP, ALTER, INDEX, REFERENCES ON zabbix.* TO 'zbx_sr
mysql> GRANT SELECT, UPDATE, DELETE, INSERT ON zabbix.* TO 'zbx_web_role';

mysql> GRANT 'zbx_srv_role' TO 'zbx_srv'@'%';
mysql> GRANT 'zbx_web_role' TO 'zbx_web'@'%';

mysql> SET DEFAULT ROLE 'zbx_srv_role' TO 'zbx_srv'@'%';
mysql> SET DEFAULT ROLE 'zbx_web_role' TO 'zbx_web'@'%';
```

请注意，X.509 协议不是用来检查身份的，但是用户被配置为仅使用加密连接。有关配置用户的更多详细信息，请参见MySQL 文档。

运行以下命令来检查连接（不能使用套接字连接来测试安全连接）：

```
mysql -u zbx_srv -p -h 10.211.55.9 --ssl-mode=REQUIRED
```

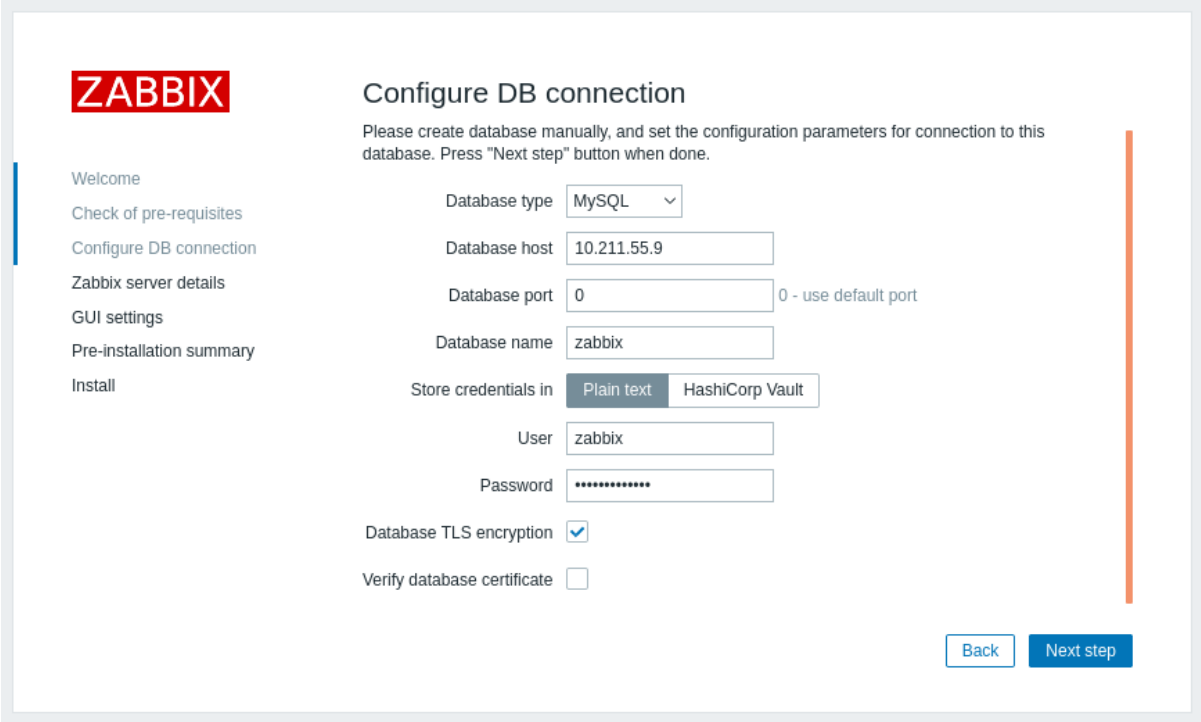
检查当前状态和可用的密码套件：

```
mysql> status
--------------
mysql Ver 8.0.21 for Linux on x86_64 (MySQL Community Server - GPL)

Connection id: 62
Current database:
Current user: zbx_srv@bfdb.local
SSL: Cipher in use is TLS_AES_256_GCM_SHA384


mysql> SHOW SESSION STATUS LIKE 'Ssl_cipher_list'\G;
*************************** 1. row ***************************
Variable_name: Ssl_cipher_list
Value: TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256:TLS_AES_128_CCM_SHA256:E
1 row in set (0.00 sec)

ERROR:
No query specified
```

前端

要为 Zabbix 前端和数据库之间的连接建立传输加密，请执行以下操作：

- 勾选 *Database TLS encryption*
- 取消勾选 *Verify database certificate*



服务端

要为服务端和数据库之间启用连接传输加密，请修改该文件 */etc/zabbix/zabbix_server.conf*:

```
...
DBHost=10.211.55.9
DBName=zabbix
DBUser=zbx_srv
DBPassword=<strong_password>
DBTLSConnect=required
...
```

验证 CA 模式

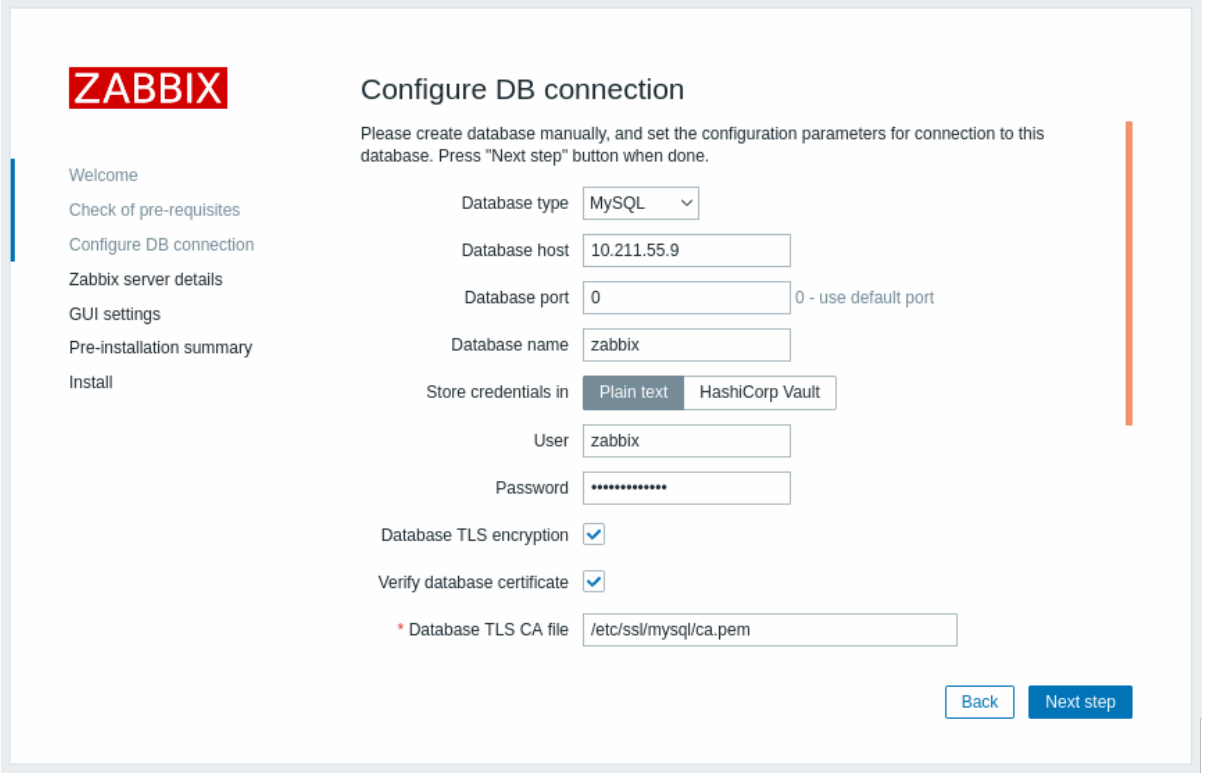
将所需的 MySQL CA 复制到 Zabbix 前端服务器，分配适当的权限以允许 Web 服务器读取此文件。

> **Note:**
> *Verify CA* 模式在 SLES 12 and RHEL 7 不会生效，因为所在系统的 MySQL 库过老.

前端

要为 Zabbix 前端和数据库之间的连接启用带有证书验证的加密：

- 勾选 数据库 *TLS* 加密和 验证数据库证书
- 指定数据库 TLS CA 文件的路径



或者，可以在 */etc/zabbix/web/zabbix.conf.php* 中设置：

```
...
$DB['ENCRYPTION'] = true;
$DB['KEY_FILE'] = '';
$DB['CERT_FILE'] = '';
$DB['CA_FILE'] = '/etc/ssl/mysql/ca.pem';
$DB['VERIFY_HOST'] = false;
$DB['CIPHER_LIST'] = '';
...
```

使用命令行工具排查用户，检查所需用户是否能够建立连接：

```
mysql -u zbx_web -p -h 10.211.55.9 --ssl-mode=REQUIRED --ssl-ca=/var/lib/mysql/ca.pem
```

服务端

要为 Zabbix 服务器和数据库之间的连接启用加密和证书验证，请配置 */etc/zabbix/zabbix_server.conf*:

```
...
DBHost=10.211.55.9
DBName=zabbix
DBUser=zbx_srv
DBPassword=<strong_password>
DBTLSConnect=verify_ca
DBTLSCAFile=/etc/ssl/mysql/ca.pem
...
```

验证完整模式

MySQL 配置

MySQL CE 请参考如下配置 (*/etc/my.cnf.d/server-tls.cnf*) ：

```
[mysqld]
...
# in this examples keys are located in the MySQL CE datadir directory
ssl_ca=ca.pem
ssl_cert=server-cert.pem
ssl_key=server-key.pem

require_secure_transport=ON
tls_version=TLSv1.3
...
```

MySQL CE 服务器和客户端（Zabbix 前端）的密钥应根据 MySQL CE 文档手动创建：使用 MySQL 创建 SSL 和 RSA 证书和密钥 或 使用 openssl 创建 SSL 证书和密钥。

MySQL 服务器证书应包含设置为 FQDN 名称的通用名称字段，因为 Zabbix 前端将使用 DNS 名称与数据库通信或数据库主机的 IP 地址。

创建 MySQL 用户：

```
mysql> CREATE USER
  'zbx_srv'@'%' IDENTIFIED WITH mysql_native_password BY '<strong_password>',
  'zbx_web'@'%' IDENTIFIED WITH mysql_native_password BY '<strong_password>'
  REQUIRE X509
  PASSWORD HISTORY 5;
```

检查是否可使用该用户登录：

```
mysql -u zbx_web -p -h 10.211.55.9 --ssl-mode=VERIFY_IDENTITY --ssl-ca=/var/lib/mysql/ca.pem --ssl-cert=/v
```

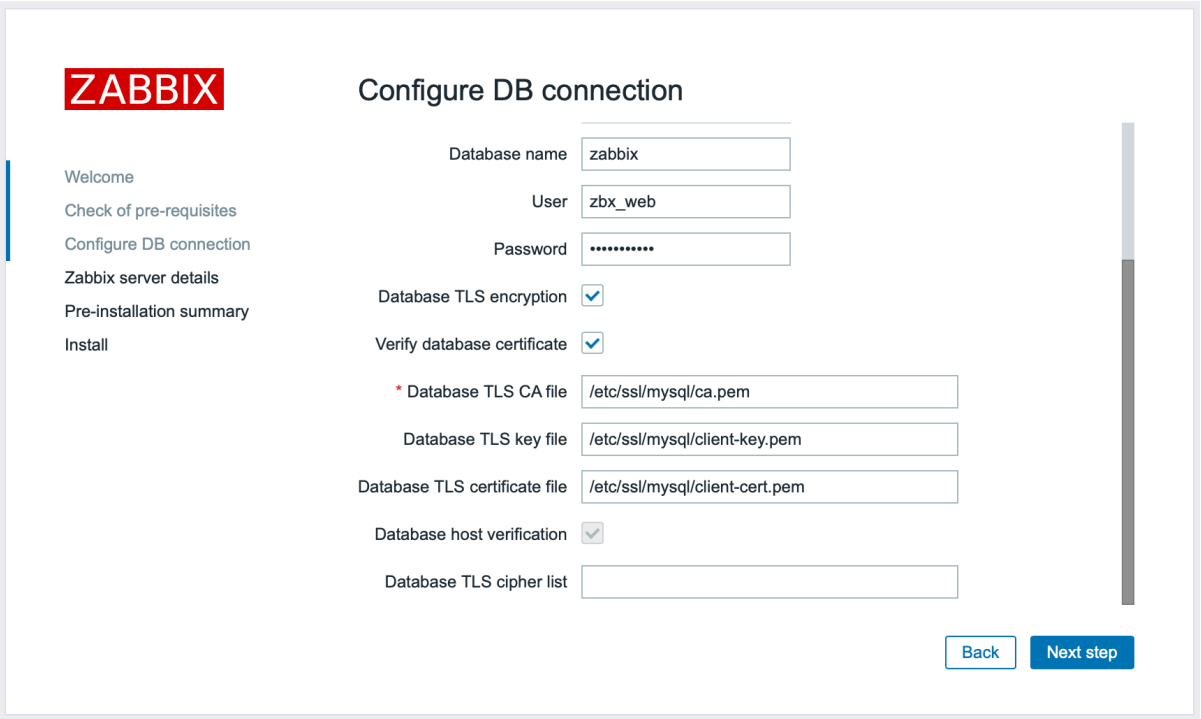
前端

要为 Zabbix 前端和数据库之间的连接启用完整验证的加密：

- 勾选数据库 TLS 加密和验证数据库证书
- 指定数据库 TLS 密钥文件的路径
- 指定数据库 TLS CA 文件的路径
- 指定数据库 TLS 证书文件的路径

请注意，数据库主机验证已勾选并变为灰色 - 此步骤对于 MySQL 不可跳过。

密码列表应为空，以便前端和服务器可以从两端都支持的密码中协商所需的密码。



或者，可以在 */etc/zabbix/web/zabbix.conf.php* 中设置。

```
...
// Used for TLS connection with strictly defined Cipher list.
$DB['ENCRYPTION'] = true;
$DB['KEY_FILE'] = '/etc/ssl/mysql/client-key.pem';
$DB['CERT_FILE'] = '/etc/ssl/mysql/client-cert.pem';
$DB['CA_FILE'] = '/etc/ssl/mysql/ca.pem';
$DB['VERIFY_HOST'] = true;
$DB['CIPHER_LIST'] = 'TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256:TLS_AES_1
...
// or
...
// Used for TLS connection without Cipher list defined - selected by MySQL server
$DB['ENCRYPTION'] = true;
$DB['KEY_FILE'] = '/etc/ssl/mysql/client-key.pem';
$DB['CERT_FILE'] = '/etc/ssl/mysql/client-cert.pem';
$DB['CA_FILE'] = '/etc/ssl/mysql/ca.pem';
$DB['VERIFY_HOST'] = true;
$DB['CIPHER_LIST'] = '';
...
```

服务端

要为 Zabbix 服务端和数据库之间启用加密连接并进行完全验证，请配置 */etc/zabbix/zabbix_server.conf*:

... DBHost=10.211.55.9 DBName=zabbix DBUser=zbx_srv DBPassword=<strong_password> DBTLSConnect=verify_full DBTLSCAFile=/etc/ssl/mysql/ca.pem DBTLSCertFile=/etc/ssl/mysql/client-cert.pem DBTLSKeyFile=/etc/ssl/mysql/client-key.pem ...

2 PostgreSQL 加密配置

概述

本节提供了几个对 CentOS 8.2 和 PostgreSQL 13 的加密配置示例。

> **Note:**
> 如果数据库主机字段的值以斜杠开头或该字段为空，则 Zabbix 前端和 PostgreSQL 之间的连接无法加密（GUI 中的参数被禁用）。

先决条件

安装 PostgreSQL 请参照official repository.

PostgreSQL 未配置开箱即用的 TLS 连接。请按照 PostgreSQL 文档中的说明进行操作 certificate preparation with postgresql.conf 也适用于 user access control \ph_hba.conf.

默认情况下，PostgreSQL 套接字绑定到本地主机，因为网络远程连接允许侦听真实的网络接口。

PostgreSQL 所有配置请参见modes 像这样:

/var/lib/pgsql/13/data/postgresql.conf:

```
...
ssl = on
ssl_ca_file = 'root.crt'
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_min_protocol_version = 'TLSv1.3'
...
```

/var/lib/pgsql/13/data/pg_hba.conf 配置文件用于访问控制的调整:

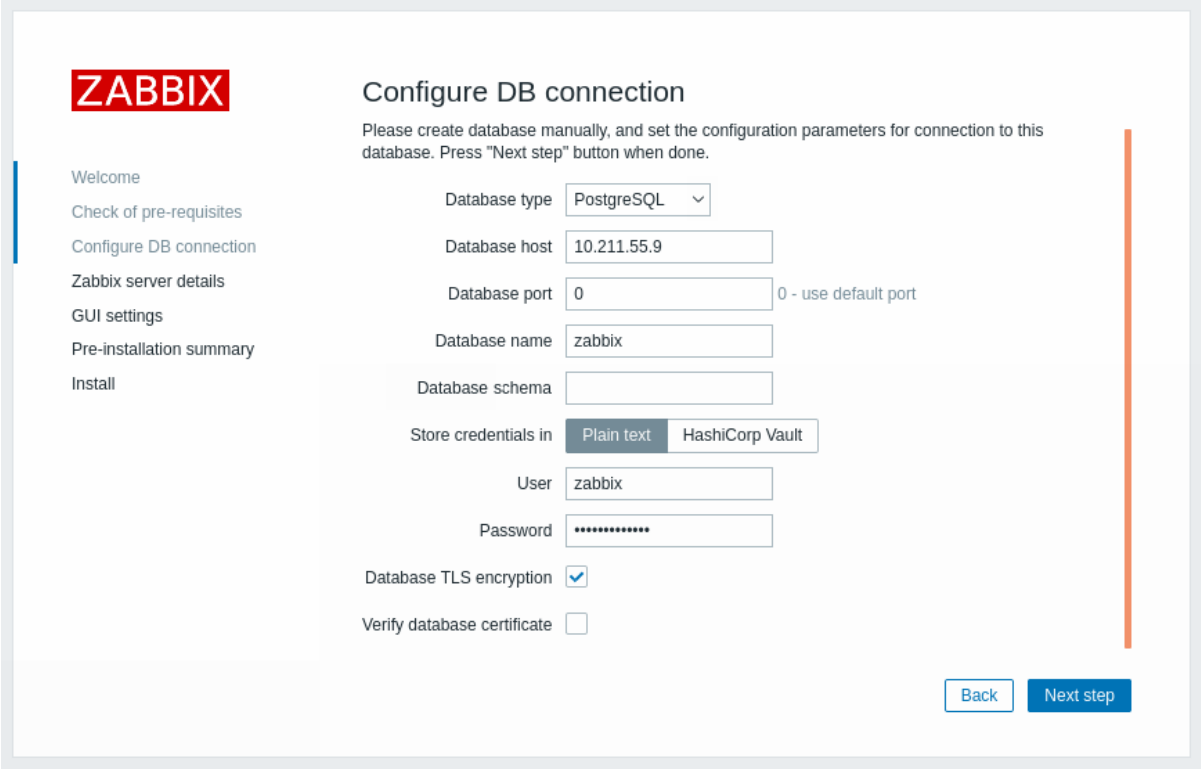
```
...
### require
hostssl all all 0.0.0.0/0 md5

### verify CA
hostssl all all 0.0.0.0/0 md5 clientcert=verify-ca

### verify full
hostssl all all 0.0.0.0/0 md5 clientcert=verify-full
...
```

所需模式

前端

请执行以下操作来启用 Zabbix 前端和数据库之间的仅传输加密连接：

- 勾选 *Database TLS encryption*
- 取消勾选 *Verify database certificate*



服务端

请配置下面的参数来启用 Zabbix 前端和数据库之间的仅传输加密连接：

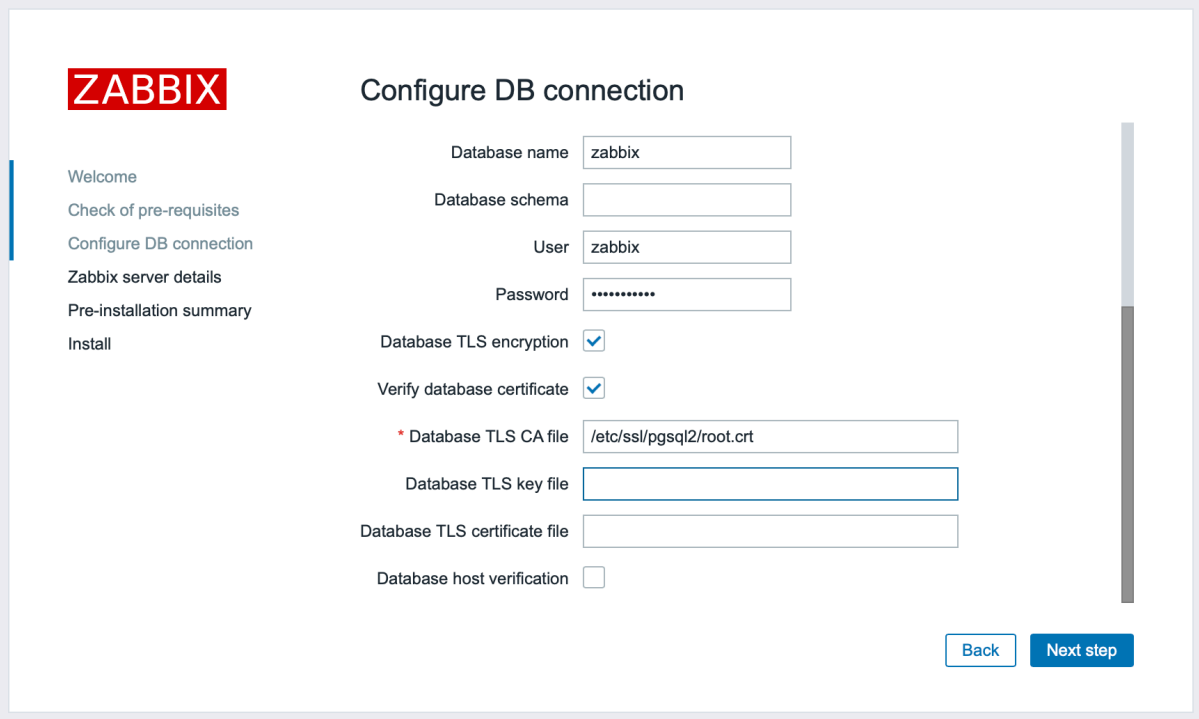
```
...
DBHost=10.211.55.9
DBName=zabbix
DBUser=zbx_srv
DBPassword=<strong_password>
DBTLSConnect=required
...
```

验证 CA 模式

前端

对于启用 Zabbix 前端和数据库之间的连接加密，请使用证书颁发机构验证：

- 勾选 *Database TLS encryption* 和 *Verify database certificate*
- 指定 *Database TLS key file* 路径
- 指定 *Database TLS CA file* 路径
- 指定 *Database TLS certificate file* 路径



或者，可以在这里配置 */etc/zabbix/web/zabbix.conf.php:*

```
...
$DB['ENCRYPTION'] = true;
$DB['KEY_FILE'] = '';
$DB['CERT_FILE'] = '';
$DB['CA_FILE'] = '/etc/ssl/pgsql/root.crt';
$DB['VERIFY_HOST'] = false;
$DB['CIPHER_LIST'] = '';
...
```

服务端

要为 Zabbix 服务端和数据库之间的连接启用加密和证书验证，请配置*/etc/zabbix/zabbix_server.conf:*

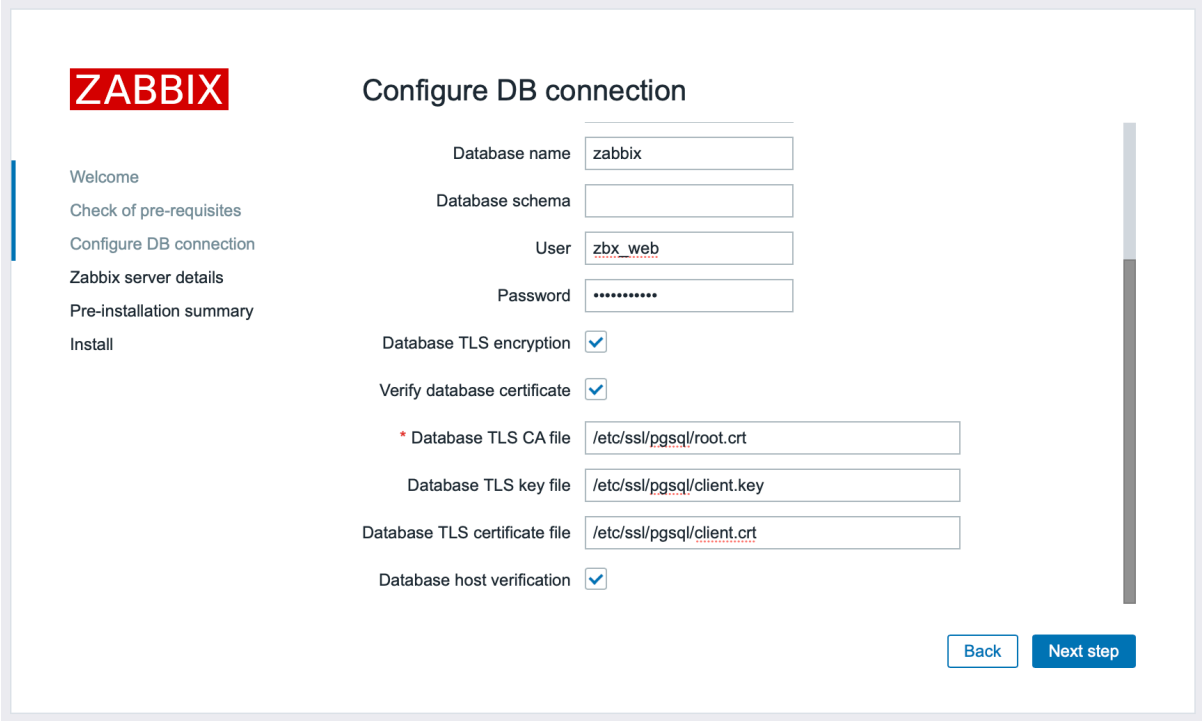
```
...
DBHost=10.211.55.9
DBName=zabbix
DBUser=zbx_srv
DBPassword=<strong_password>
DBTLSConnect=verify_ca
DBTLSCAFile=/etc/ssl/pgsql/root.crt
...
```

完整验证模式

前端

要使用证书和数据库主机身份验证为 Zabbix 前端和数据库之间的连接启用加密，请执行以下操作:

- 勾选 *Database TLS encryption* 和 *Verify database certificate*
- 指定 *Database TLS key file* 路径
- 指定 *Database TLS CA file* 路径
- 指定 *Database TLS certificate file* 路径
- 勾选 *Database host verification*



或者，可以在这里配置 */etc/zabbix/web/zabbix.conf.php:*

```
$DB['ENCRYPTION'] = true;
$DB['KEY_FILE'] = '';
$DB['CERT_FILE'] = '';
$DB['CA_FILE'] = '/etc/ssl/pgsql/root.crt';
$DB['VERIFY_HOST'] = true;
$DB['CIPHER_LIST'] = '';
...
```

服务端

要使用证书和数据库主机身份验证为 Zabbix 服务器和数据库之间的连接启用加密，请配置该文件 */etc/zabbix/zabbix_server.conf*:

```
...
DBHost=10.211.55.9
DBName=zabbix
DBUser=zbx_srv
DBPassword=<strong_password>
DBTLSConnect=verify_full
DBTLSCAFile=/etc/ssl/pgsql/root.crt
DBTLSCertFile=/etc/ssl/pgsql/client.crt
DBTLSKeyFile=/etc/ssl/pgsql/client.key
...
```

5 TimescaleDB 配置

概述

Zabbix 支持 TimescaleDB，这是一种基于 PostgreSQL 的数据库解决方案，可自动将数据分为基于时间的块，以支持更快的大规模性能。

Warning:
目前，Zabbix 代理不支持 TimescaleDB 。

此章节会介绍创建 TimescaleDB 数据库或从现有的 PostgreSQL 表迁移到 TimescaleDB

配置

我们假设 TimescaleDB 扩展已经在数据库服务器上安装好了（参见安装说明）。

TimescaleDB 扩展也必须通过执行以下命令来为特定的数据库启用：

```
echo "CREATE EXTENSION IF NOT EXISTS timescaledb CASCADE;" | sudo -u postgres psql zabbix
```

运行此命令需要数据库管理员权限。

Note:
如果您使用的数据库架构不是'public'，您需要在上述命令中添加 SCHEMA 子句。例如：

```
echo "CREATE EXTENSION IF NOT EXISTS timescaledb SCHEMA yourschema CASCADE;" | sudo -u postgres psql zabbix
```

然后运行 `postgresql/timescaledb.sql` 脚本。对于新安装，该脚本必须在用初始架构/数据创建了常规 PostgreSQL 数据库之后运行（参见数据库创建）：

```
cat /usr/share/zabbix-sql-scripts/postgresql/timescaledb.sql | sudo -u zabbix psql zabbix
```

Attention:
请忽略在 TimescaleDB 版本 2.9.0 及以上版本上运行 `timescaledb.sql` 脚本时出现的警告信息，这些警告信息提示没有遵循最佳实践。尽管有这个警告，配置仍将成功完成。

现有历史和趋势数据的迁移可能需要很长时间。在迁移期间，Zabbix 服务器和前端必须关闭。

`timescaledb.sql` 脚本设置了以下管家参数：

- 覆盖项目历史周期
- 覆盖项目趋势周期

为了使用历史和趋势的分区管家，这两个选项都必须启用。也可以单独启用覆盖，无论是仅针对历史还是仅针对趋势。

对于 PostgreSQL 版本 10.2 或更高版本和 TimescaleDB 版本 1.5 或更高版本，`timescaledb.sql` 脚本设置了两个额外的参数：

- 启用压缩
- 压缩 7 天以上的记录

要成功地通过管家删除压缩数据，必须同时启用 “覆盖监控项历史周期” 和 “覆盖监控项趋势周期” 选项。如果禁用了覆盖功能，并且表中有压缩的数据块，管家将不会从这些表中删除数据，并且在管家和系统信息部分的管理界面中将显示有关配置不正确的警告。

安装后，所有这些参数都可以在 *Administration* → *General* → *Housekeeping* 中更改。

Note:
您可能需要运行 TimescaleDB 提供的 timescaledb-tune 工具来优化 `postgresql.conf` 中的 PostgreSQL 配置参数。

TimescaleDB 压缩

从 Zabbix 5.0 开始，PostgreSQL 版本 10.2 或更高版本以及 TimescaleDB 版本 1.5 或更高版本开始支持由 TimescaleDB 管理的所有 Zabbix 表的原生 TimescaleDB 压缩。在升级或迁移到 TimescaleDB 期间，大表的初始压缩可能会花费很多时间。

请注意，“timescale”Timescale Community 许可支持压缩，“apache”Apache 2.0 许可不支持压缩。从 Zabbix 6.0.7 开始，Zabbix 检测是否支持压缩。如果不支持，则会在 Zabbix 服务器日志中写入一条警告消息，并且用户无法在前端启用压缩。

Note:
鼓励用户在使用压缩之前熟悉 TimescaleDB 压缩文档。

请注意，压缩有一定的限制，特别是：

- 不允许对压缩块进行修改（插入、删除、更新）
- 不允许更改压缩表的 schema。

可以在 Zabbix 前端的 管理 → 一般 → 管家部分的 历史和趋势压缩块中更改压缩设置。

参数	默认	注释
启用压缩	已启用	选中或取消选中该复选框不会立即激活/停用压缩。因为压缩是由 Housekeeper 处理的，所以更改将在最多 2 倍的 `HousekeepingFrequency` 小时内生效（设置在zabbix_server.conf） 禁用压缩后，落入压缩周期的新块将不会被压缩。但是，所有以前压缩的数据将保持压缩状态。要解压缩以前压缩的块，请按照 TimescaleDB 文档中的说明进行操作。
Compress records older than	7d	从支持 TimescaleDB 的旧版本 Zabbix 升级时，默认情况下不会启用压缩。此参数不能少于 7 天。 由于压缩块的不变性，所有早于此值的延迟数据（例如，代理延迟的数据）都将被丢弃。

6 Elasticsearch 配置

Attention:
对 Elasticsearch 的支持仍在试验阶段！

Zabbix 支持 Elasticsearch 代替数据库存储历史数据。用户可以在兼容的数据库和 Elasticsearch 之间选择存储历史数据的位置。本章节描述的设置适用于 Elasticsearch version 7.X。如果使用早期或更高版本的 Elasticsearch，某些功能可能无法使用。

Warning:
如果所有历史数据都存储在 Elasticsearch 中，则不会计算趋势，也不会将其存储在数据库中。由于没有计算和存储任何趋势，则可能需要延长历史记录存储期。

配置

为确保所涉及的所有元素之间的正确通信，请确保服务器配置文件和前端配置文件参数正确。

Zabbix 服务端和前端

Zabbix 服务器配置文件模板，其中包含要更新的参数：

```
### Option: HistoryStorageURL
# History storage HTTP[S] URL.
#
# Mandatory: no
# Default:
# HistoryStorageURL=
### Option: HistoryStorageTypes
# Comma separated list of value types to be sent to the history storage.
#
# Mandatory: no
# Default:
# HistoryStorageTypes=uint,dbl,str,log,text
```

用于 Zabbix 服务端配置文件示例的参数值

```
HistoryStorageURL=http://test.elasticsearch.lan:9200
HistoryStorageTypes=str,log,text
```

此配置强制 Zabbix Server 在相应的数据库中存储数字类型的历史值，并在 Elasticsearch 中存储文本历史数据。

Elasticsearch 支持以下 item 类型:

```
uint,dbl,str,log,text
```

支持的 item 类型说明:

Item 类型	数据库表	**Elasticsearch** 类型
Numeric (unsigned)	history_uint	uint
Numeric (float)	history	dbl
Character	history_str	str

Log	history_log	log
Text	history_text	text

Zabbix 前端配置文件 (conf/zabbix.conf.php) 参数待更新:

```
// Elasticsearch url (can be string if same url is used for all types).
$HISTORY['url']   = [
      'uint' => 'http://localhost:9200',
      'text' => 'http://localhost:9200'
];
// Value types stored in Elasticsearch.
$HISTORY['types'] = ['uint', 'text'];
```

用于 Zabbix 前端配置文件示例的参数值:

```
$HISTORY['url']   = 'http://test.elasticsearch.lan:9200';
$HISTORY['types'] = ['str', 'text', 'log'];
```

此配置强制在 Elasticsearch 种存储 Text, Character 及 Log 的历史值. 还需要在 conf/zabbix.conf.php 中使用全局变量 $HISTORY，以确保一切正常 (有关如何执行此操作，请参阅 conf/zabbix.conf.php.example):

```
// Zabbix GUI configuration file.
global $DB, $HISTORY;
```

安装 Elasticsearch 并创建映射

安装 Elasticsearchit 及创建映射是整个安装过程的最后两步了。

如何安装 Elasticsearch，请参阅Elasticsearch installation guide.

Note:
映射是 Elasticsearch 中的一种数据结构（类似于数据库中的表）。所有历史数据类型的映射都可用: database/elasticsearch/elasticsearch.map.

Warning:
创建映射是必需的。如未根据指令创建映射，某些功能将不可用。

要为 text 类型创建映射，请将以下请求发送到 Elasticsearch

```
curl -X PUT \
 http://your-elasticsearch.here:9200/text \
 -H 'content-type:application/json' \
 -d '{
   "settings": {
      "index": {
         "number_of_replicas": 1,
         "number_of_shards": 5
      }
   },
   "mappings": {
      "properties": {
         "itemid": {
            "type": "long"
         },
         "clock": {
            "format": "epoch_second",
            "type": "date"
         },
         "value": {
            "fields": {
               "analyzed": {
                  "index": true,
                  "type": "text",
                  "analyzer": "standard"
               }
```

```
            },
            "index": false,
            "type": "text"
         }
      }
   }
}'
```

对于 Character 和 Log 历史值映射创建，需要执行类似的请求，并将相应的类型进行修改

> **Note:**
> 要使用 Elasticsearch，请参阅Requirement page 以获取更多信息.

> **Note:**
> Housekeeper不会从 Elasticsearch 中删除任何数据。

将历史数据存储在多个基于日期的索引中

本节介绍使用管道和引入节点所需的其他步骤。首先，必须为索引创建模板。以下示例为创建 uint 模板的请求：

```
curl -X PUT \
 http://your-elasticsearch.here:9200/_template/uint_template \
 -H 'content-type:application/json' \
 -d '{
   "index_patterns": [
      "uint*"
   ],
   "settings": {
      "index": {
         "number_of_replicas": 1,
         "number_of_shards": 5
      }
   },
   "mappings": {
      "properties": {
         "itemid": {
            "type": "long"
         },
         "clock": {
            "format": "epoch_second",
            "type": "date"
         },
         "value": {
            "type": "long"
         }
      }
   }
}'
```

要创建其他模板，用户应更改 URL（最后一部分是模板的名称），更改"index_patterns" 字段以匹配索引名称并设置有效的映射，这可以在 database/elasticsearch/elasticsearch.map 获取。以下命令可用于为文本索引创建模板，例如：

```
curl -X PUT \
 http://your-elasticsearch.here:9200/_template/text_template \
 -H 'content-type:application/json' \
 -d '{
   "index_patterns": [
      "text*"
   ],
   "settings": {
      "index": {
         "number_of_replicas": 1,
         "number_of_shards": 5
      }
```

```
   },
   "mappings": {
      "properties": {
         "itemid": {
            "type": "long"
         },
         "clock": {
            "format": "epoch_second",
            "type": "date"
         },
         "value": {
            "fields": {
               "analyzed": {
                  "index": true,
                  "type": "text",
                  "analyzer": "standard"
               }
            },
            "index": false,
            "type": "text"
         }
      }
   }
}'
```

这是允许 Elasticsearch 为自动创建的索引设置有效的映射所必需做的。然后需要创建 pipeline 定义。pipeline 是在将数据放入索引之前对数据的某种预处理。以下命令可用于为 uint 索引创建 pipeline：

```
curl -X PUT \
 http://your-elasticsearch.here:9200/_ingest/pipeline/uint-pipeline \
 -H 'content-type:application/json' \
 -d '{
   "description": "daily uint index naming",
   "processors": [
      {
         "date_index_name": {
            "field": "clock",
            "date_formats": [
               "UNIX"
            ],
            "index_name_prefix": "uint-",
            "date_rounding": "d"
         }
      }
   ]
}'
```

用户可以更改参数 ("date_rounding") 以设置特定的索引轮换周期。要创建其他 pipelines，用户应更改 URL（最后一部分是 pipelines 的名称）并更改"index_name_prefix" 字段以匹配索引名称另见 Elasticsearch documentation. 此外，在 Zabbix 服务器新的配置参数中，还应启用将历史数据存储在多个基于日期的索引中：

```
### Option: HistoryStorageDateIndex
# Enable preprocessing of history values in history storage to store values in different indices based on
# 0 - disable
# 1 - enable
#
# Mandatory: no
# Default:
# HistoryStorageDateIndex=0
```

故障排除

以下步骤可以帮助您解决 Elasticsearch 设置问题：

1. 检查映射是否正确 (通过 URL 的发送 GET 请求获取索引信息，例如：`http://localhost:9200/uint`)
2. 检查 shards 状态是否正常 (不正常时重启 Elasticsearch 可能解决问题)

3. 检查 Elasticsearch 配置文件，配置文件应允许从 Zabbix 前端及 Zabbix server 主机访问。
4. 检查 Elasticsearch 日志

如果您仍然遇到安装问题，请创建一个错误报告，包含此列表中的所有信息（映射，错误日志，配置，版本等信息）

7 实时导出事件、监控项的值、趋势

概述

可以使用以换行符分隔的 JSON 格式，配置触发器事件、监控项值和趋势的实时导出。

导出文件的每一行都应是一个 JSON 对象。值映射不被应用。

如果出现错误（无法将数据写入导出文件，或者无法重命名导出文件，或者在重新创建导出文件后无法创建新文件），则会删除数据，并且永远不会将其写入导出文件。它只写在 Zabbix 数据库中。解决写入问题后，将恢复数据的写入。

有关导出哪些信息的确切详细信息，请参阅export protocol 。

请注意，如果主机/监控项在收到数据后被删除，但在服务器导出数据之前，则主机/监控项可以没有元数据（主机组、主机名、监控项名称）。

配置

为导出文件指定目录来配置触发事件、监控项的值和趋势的实时导出 - 请参阅服务器中的 ExportDir 参数configuration.

另外以下两个参数可配置：

- ExportFileSize 可用于设置允许导出的文件的最大值。当进程需要写入文件时，它会首先检查文件的大小。如果超出配置的大小限制，则以.old 结尾重命名该文件，并创建一个和原名称相同的新文件。

> **Attention:**
> 每个将要写入数据的进程都会创建一个文件（大约 4-30 个文件）。由于每个导出文件的默认大小为 1G，因此保留较大的导出文件可能会加快磁盘空间的占用。

- ExportType 允许指定将导出哪些实体类型（事件、历史记录、趋势）。

8 Zabbix 在 **Nginx** 特定发行版中的注意事项

RHEL

Nginx 仅在 EPEL 中可用:

```
dnf -y install epel-release
```

SLES 12

在 SUSE Linux Enterprise Server 12 中，您需要添加 Nginx 存储库，然后再安装 Nginx:

```
zypper addrepo -G -t dnf -c 'http://nginx.org/packages/sles/12' nginx
```

您还需要配置 php-fpm:

```
cp /etc/php5/fpm/php-fpm.conf{.default,}
sed -i 's/user = nobody/user = wwwrun/; s/group = nobody/group = www/' /etc/php5/fpm/php-fpm.conf
```

SLES 15

在 SUSE Linux Enterprise Server 15 中，您需要配置 php-fpm:

```
cp /etc/php7/fpm/php-fpm.conf{.default,}
cp /etc/php7/fpm/php-fpm.d/www.conf{.default,}
sed -i 's/user = nobody/user = wwwrun/; s/group = nobody/group = www/' /etc/php7/fpm/php-fpm.d/www.conf
```

9 以 **root** 用户身份运行 **agent**

Zabbix 从 **5.0.0** 版本开始，Zabbix agent 服务的启动文件 官方包 已更新为显示包含 User 及 Group 的参数。两者都设置为 zabbix。

由于 agent 会通过以 systemd 服务文件指定的用户运行，而忽略 zabbix_agentd.conf 配置文件，这也将导致不再支持通过配置 zabbix_agentd.conf 来指定运行用户。可以通过如下描述的修改来实现 agent 以 root 用户运行。

Zabbix agent

要覆盖缺省用户和用户组，执行：systemctl edit zabbix-agent 然后，增加如下内容：

```
[Service]
User=root
Group=root
```

重新加载守护进程并重启 zabbix-agent 服务

```
systemctl daemon-reload
systemctl restart zabbix-agent
```

对于 **Zabbix agent**，依然要在 `zabbix_agentd.conf` 文件中重新启用配置用户的功能。因此，为了以 root 身份运行 zabbix agent，您仍然需要编辑 agent 配置文件（/manual/annex/config/zabbix_agentd）并指定 `User=root` 以及 `AllowRoot=1` 两个参数。

Zabbix agent 2

要覆盖缺省用户和用户组，执行：systemctl edit zabbix-agent2 然后，增加如下内容：

```
[Service]
User=root
Group=root
```

重新加载守护进程并重启 zabbix-agent 服务

```
systemctl daemon-reload
systemctl restart zabbix-agent2
```

对于 **Zabbix agent2**，运行用户完全取决于上述修改，不再需要其他修改。

10 在 Microsoft Windows 上的 Zabbix agent

配置 agent

两代 Zabbix agent 都是做为 Windows 服务运行的。对于 Zabbix agent 2，请在下面的说明中将 *agentd* 替换为 *agent2* 。

您可以在 Microsoft Windows 主机上运行 Zabbix agent 的单个实例或多个实例。单个实例可以使用默认配置文件 `C:\zabbix_agentd.conf` 或在命令行中指定的配置文件。在多个实例的情况下，每个代理实例都必须有自己的配置文件（其中一个实例可以使用默认配置文件）。

在 Zabbix 源码包中提供了一个示例配置文件，如下 `conf/zabbix_agentd.win.conf`.

有关配置 Zabbix Windows agent 的详细信息的选项，另见配置文件。

> **Warning:**
> ** 适用于 Windows 的 Zabbix agent 不支持非标准 Windows 配置，其 CPU 在 NUMA 节点之间不是统一分布的。** 如果逻辑 CPU 分布不均匀，则 CPU 性能指标可能不适用于某些 CPU。例如，如果有 72 个逻辑 CPU 和 2 个 NUMA 节点，则两个节点都必须有 36 个 CPU。

主机名参数

要在主机上执行主动检查，Zabbix agent 需要定义主机名。此外，agent 端设置的主机名值应与前端为主机配置的”Host name” 完全匹配。

agent 端的主机名值可以通过 agent 的配置文件中的 **Hostname** 或 **HostnameItem** 参数来定义 - 如果这些参数未指定，则使用默认值。

HostnameItem 参数的默认值是由”system.hostname” agent 键返回的值。对于 Windows，它返回 gethostname() 函数的结果，该函数查询名称空间提供程序以确定本地主机名。如果没有名称空间提供程序响应，则返回 NetBIOS 名称。

Hostname 的默认值是由 **HostnameItem** 参数返回的值。因此，实际上，如果这两个参数都未指定，实际的主机名将是主机的 NetBIOS 名称；Zabbix agent 将使用 NetBIOS 主机名从 Zabbix server 检索主动检查列表，并将结果发送给它。

”system.hostname” 键支持两个可选参数 - *type* 和 *transform*。

Type 参数确定项目应返回的名称类型。支持的值：

- *netbios*（默认）- 返回 NetBIOS 主机名，该名称限制为 15 个符号，并且仅是大写字母；
- *host* - 区分大小写，返回完整的真实 Windows 主机名（不包括域）；
- *shorthost*（自 Zabbix 5.4.7 起支持）- 返回第一个点之前的主机名部分。如果名称不包含点，则返回完整字符串。

Transform 参数自 Zabbix 5.4.7 起支持，并允许为主机名指定额外的转换规则。支持的值：

- *none*（默认）- 使用原始字母大小写；
- *lower* - 将文本转换为小写。

因此，为了简化 zabbix_agentd.conf 文件的配置并使其统一，可以使用两种不同的方法。

1. 保留 **Hostname** 或 **HostnameItem** 参数未定义，Zabbix agent 将使用 NetBIOS 主机名作为主机名；
2. 保留 **Hostname** 参数未定义，并这样定义 **HostnameItem**：**HostnameItem=system.hostname[host]** - 让 Zabbix agent 使用完整的真实（区分大小写）Windows 主机名作为主机名 **HostnameItem=system.hostname[shorthost,lower]** - 让 Zabbix agent 只使用主机名的第一个点之前的部分，并转换为小写。

主机名也用作 Windows 服务名称的一部分，该名称用于安装、启动、停止和卸载 Windows 服务。例如，如果 Zabbix agent 配置文件指定 `Hostname=Windows_db_server`，那么 agent 将作为"`Zabbix Agent [Windows_db_server]`" 的 Windows 服务安装。因此，要为每个 Zabbix agent 实例使用不同的 Windows 服务名称，每个实例必须使用不同的主机名。

安装 agent 并注册 Windows 服务

使用默认配置文件安装 Zabbix agent 的单个实例 `c:\zabbix_agentd.conf`:

```
zabbix_agentd.exe --install
```

> **Attention:**
> 在 64 位操作系统上, 务必检查 64 位的 Zabbix agent 版本与运行 64 位进程相关的一切，以便它可以正确的工作。

如果您希望使用除 `c:\zabbix_agentd.conf` 以外的配置, 您应该使用以下命令进行服务安装:

```
zabbix_agentd.exe --config <your_configuration_file> --install
```

应指定配置文件的完整路径。.

Zabbix agent 的多个实例可以这样安装服务:

```
zabbix_agentd.exe --config <configuration_file_for_instance_1> --install --multiple-agents
zabbix_agentd.exe --config <configuration_file_for_instance_2> --install --multiple-agents
...
zabbix_agentd.exe --config <configuration_file_for_instance_N> --install --multiple-agents
```

安装的服务现在应该在" 控制面板" 中可见。

Starting agent

要启动 agent 服务，可以使用" 控制面板" 或从命令行执行此操作

使用默认配置文件启动单个实例的 Zabbix agent:

```
zabbix_agentd.exe --start
```

指定配置文件启动单个实例的 Zabbix agent:

```
zabbix_agentd.exe --config <your_configuration_file> --start
```

启动多个 Zabbix agent 实例的其中一个，请执行以下操作：

```
zabbix_agentd.exe --config <configuration_file_for_this_instance> --start --multiple-agents
```

停止 agent

要停止 agent 服务，可以使用" 控制面板" 或从命令行执行此操作.

使用默认配置文件停止单个实例的 Zabbix agent:

```
zabbix_agentd.exe --stop
```

指定配置文件停止单个实例的 Zabbix agent:

```
zabbix_agentd.exe --config <your_configuration_file> --stop
```

停止多个 Zabbix agent 实例的其中一个，请执行以下操作:

```
zabbix_agentd.exe --config <configuration_file_for_this_instance> --stop --multiple-agents
```

卸载 Windows 服务上的 zabbix agent

使用默认配置文件卸载单个实例的 Zabbix agent:

```
zabbix_agentd.exe --uninstall
```

指定配置文件卸载单个实例的 Zabbix agent:

```
zabbix_agentd.exe --config <your_configuration_file> --uninstall
```

卸载多个 Zabbix agent 实例，请执行以下操作:

```
zabbix_agentd.exe --config <configuration_file_for_instance_1> --uninstall --multiple-agents
zabbix_agentd.exe --config <configuration_file_for_instance_2> --uninstall --multiple-agents
...
zabbix_agentd.exe --config <configuration_file_for_instance_N> --uninstall --multiple-agents
```

局限性

用于 Windows 的 Zabbix 代理不支持非标准的 Windows 配置，cpu 在 NUMA 节点上非均匀分布。如果逻辑 CPU 是非均匀分布的，那么某些 CPU 的 CPU 性能指标可能不可用。例如, 如果有 72 个逻辑 CPU 分布在 2 个 NUMA 节点，那么每个节点必须有 36 个 CPU。

11 SAML setup with Microsoft Entra ID

Overview

This section provides guidelines for configuring single sign-on into Zabbix from Microsoft Entra ID (formerly Microsoft Azure Active Directory) using SAML 2.0 authentication.

Note that for single sign-on to work its username must exist in Zabbix, however, its Zabbix password will not be used. If authentication is successful, then Zabbix will match a local username with the username attribute returned by SAML.

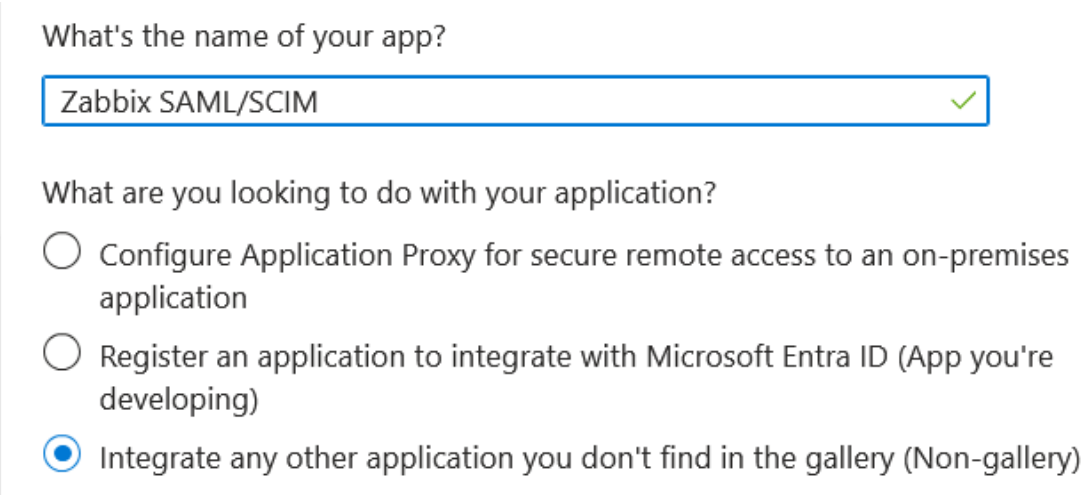
Microsoft Entra ID configuration

Creating application

1. Log into Microsoft Entra admin center at Microsoft Entra ID. For testing purposes, you may create a free trial account in Microsoft Entra ID.

2. In Microsoft Entra admin center select *Applications -> Enterprise applications -> New application -> Create your own application*.

3. Add the name of your app and select the *Integrate any other application...* option. After that, click on *Create*.

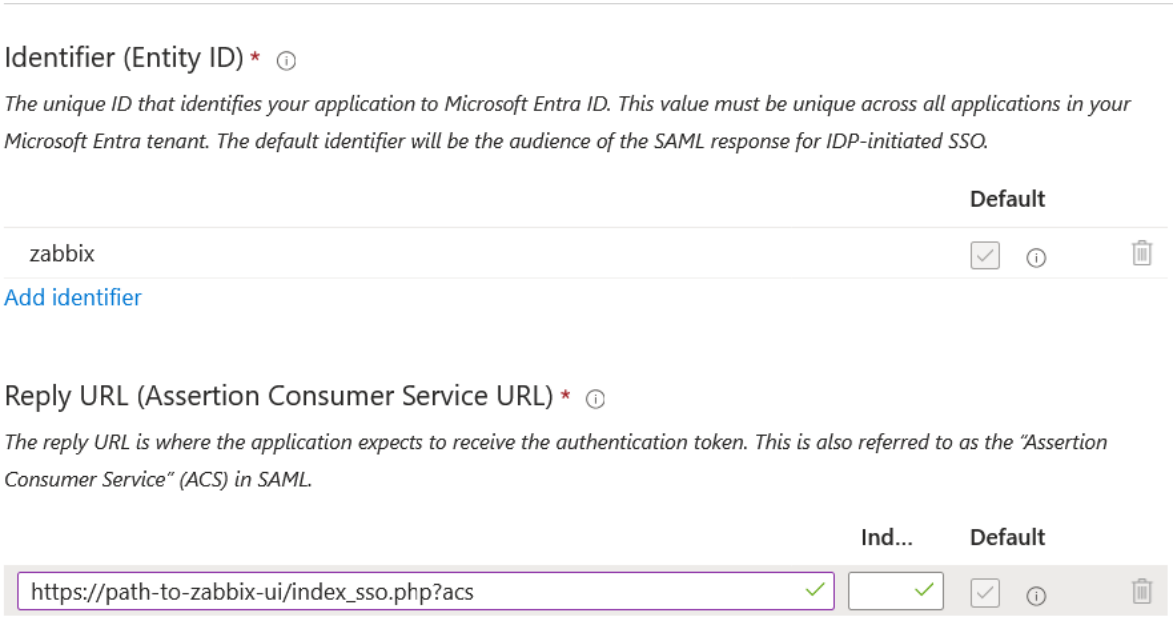


Setting up single sign-on

1. In your application page, go to *Set up single sign on* and click on *Get started*. Then select *SAML*.

2. Edit *Basic SAML Configuration*:

- In *Identifier (Entity ID)* set a unique name to identify your app to Microsoft Entra ID, for example, `zabbix`;
- In *Reply URL (Assertion Consumer Service URL)* set the Zabbix single sign-on endpoint: `https://<path-to-zabbix-ui>/index_sso`

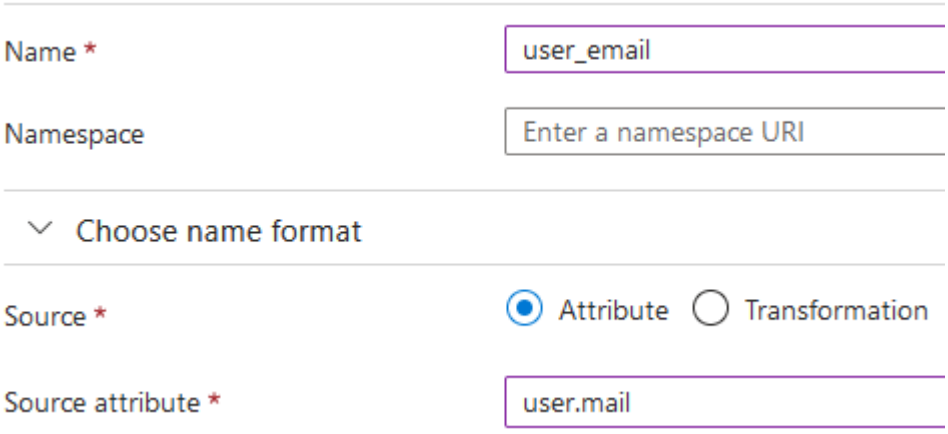


Note that "https" is required. To make that work with Zabbix, it is necessary to add to `conf/zabbix.conf.php` the following line:

```
$SSO['SETTINGS'] = ['use_proxy_headers' => true];
```

3. Edit *Attributes & Claims*. You must add the username attribute that you want to pass to Zabbix (`user_email`, in this case). The attribute names are arbitrary. Different attribute names may be used, however, it is required that they match the respective field value in Zabbix SAML settings.

- Click on *Add new claim* to add an attribute:



4. In *SAML Certificates* download the Base64 certificate provided by Entra ID and place it into `conf/certs` of the Zabbix frontend installation.

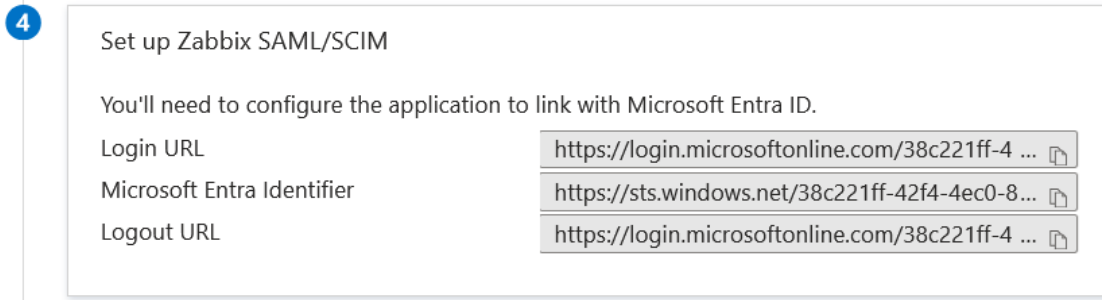
Set 644 permissions to it by running:

```
chmod 644 entra.cer
```

Make sure that `conf/zabbix.conf.php` contains the line:

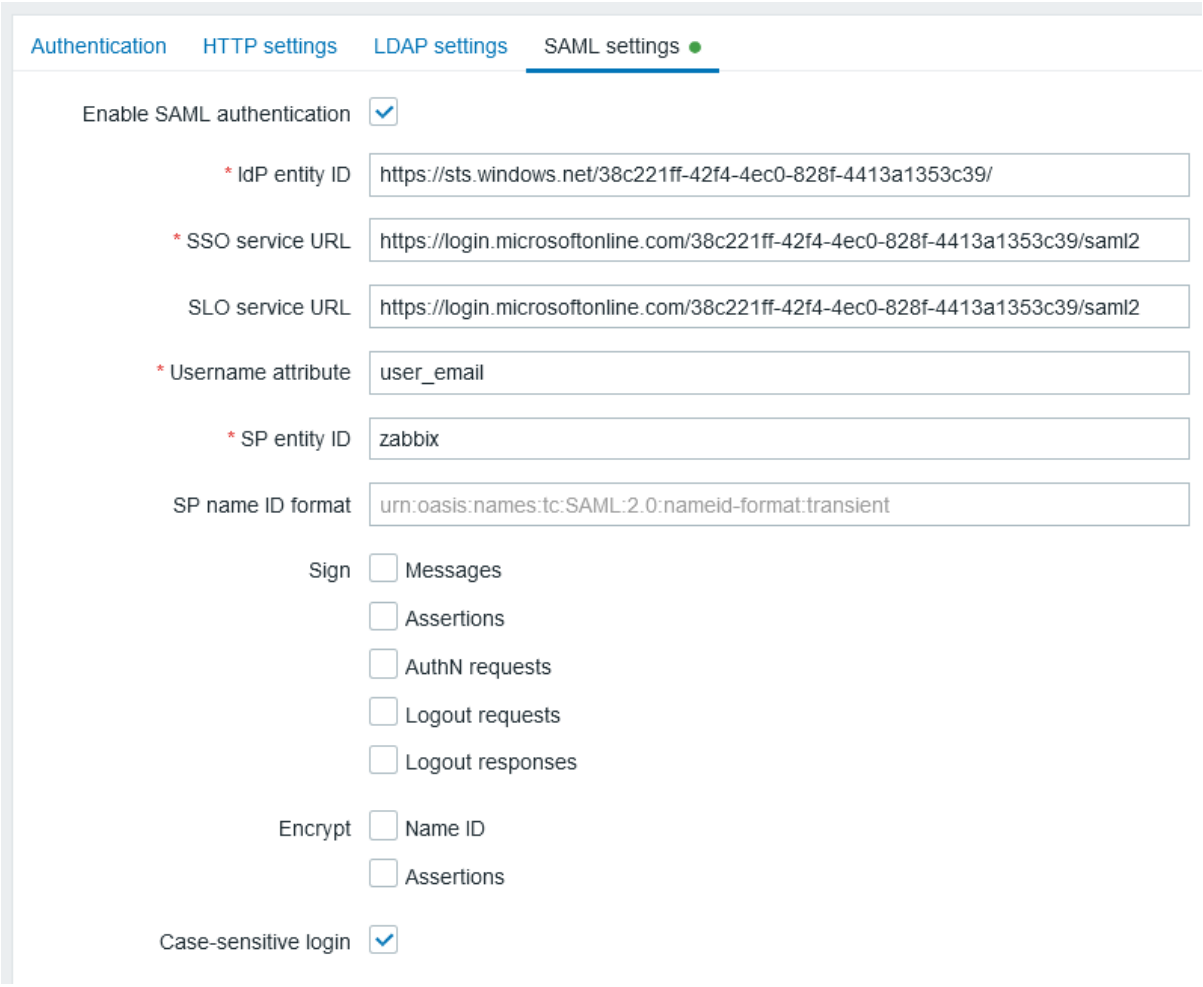
```
$SSO['IDP_CERT'] = 'conf/certs/entra.cer';
```

5. Use the values from *Set up <your app name>* in Entra ID to configure Zabbix SAML authentication (see next section):



Zabbix configuration

1. In Zabbix, go to the SAML settings and fill the configuration options based on the Entra ID configuration:



Zabbix field	Setup field in Entra ID	Sample value
IdP entity ID	Microsoft Entra identifier	
SSO service URL	Login URL	
SLO service URL	Logout URL	
SP entity ID	Identifier (Entity ID)	
Username attribute	Custom attribute (claim)	`user_email`

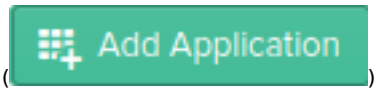
Click on *Update* to save these settings.

11 配置 Okta 启用 SAML

本节介绍如何配置 Okta 以启用 Zabbix 的 SAML 2.0 身份验证。

Okta 配置

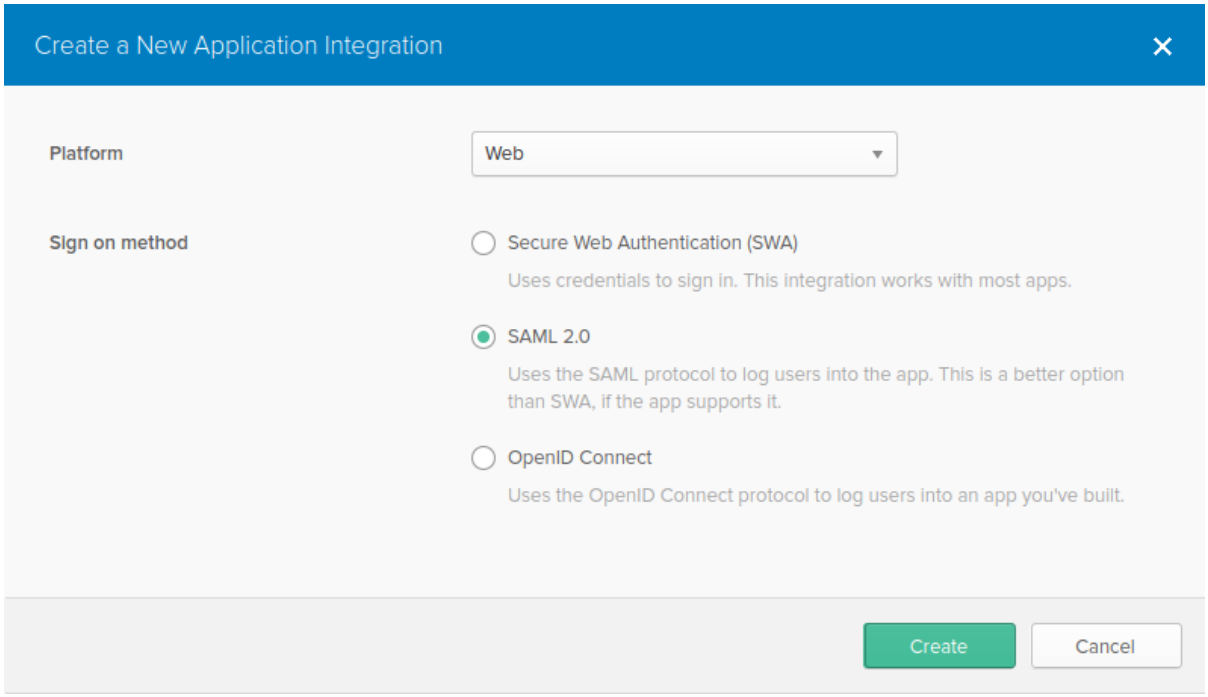
1. 前往 https://okta.com 注册或登录您的帐户.

2. 在 Okta 页面中找到 *Applications* → *Applications* 并点击”Add Application” (

).

3. 点击”Create New App” (

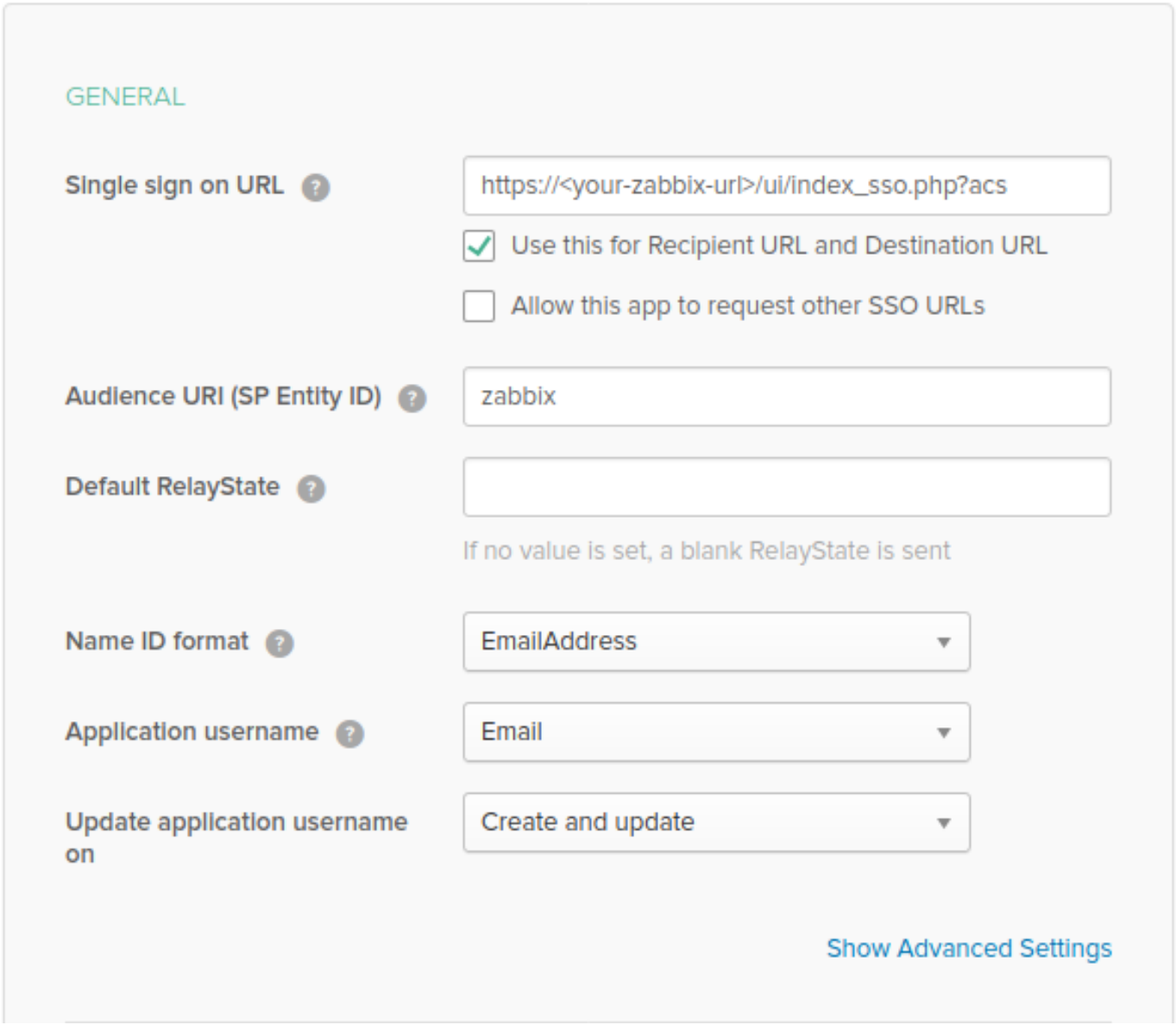
). 在弹出窗口中，选择 *Platform:* ，*Sign on method:* SAML 2.0 并点击”Create” 按钮.



4. 根据您的喜好填写 *General settings* 选项卡（第一个选项卡）并点击”Next”.

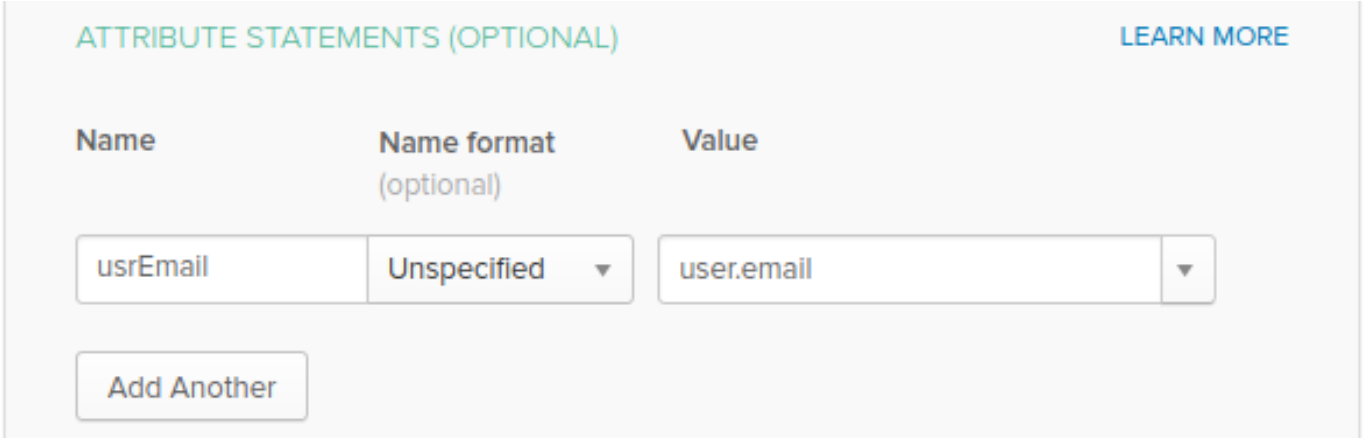
5. 在 *Configure SAML* 选项中输入下面提供的值并点击”Next” 按钮.

- 在 **GENERAL** 部分:
  - *Single sign on URL*: https://<your-zabbix-url>/ui/index_sso.php?acs
    复选框 *Use this for Recipient URL and Destination URL* 应为勾选的)
  - *Audience URI (SP Entity ID)*: zabbix
    请注意，此值将在 SAML 用作程序唯一的标识符（如果不匹配，则将拒绝操作）。可以在此字段中指定 URL 或任何数据字符串
  - *Default RelayState*:
    将此字段为空; 如果需要自定义, 可以在 Zabbix 的 *Administration* → *Users* 设置中添加它.
  - 根据您的喜好填写其他字段。



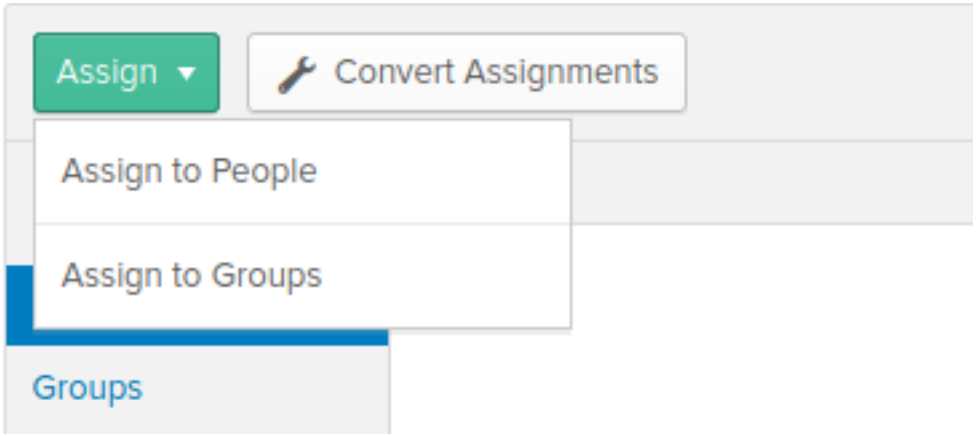
Note:
如果计划使用加密连接，请生成私有和公有加密证书，然后将公有证书上传到 Okta。当 *Assertion Encryption* 设置为" 已加密" 时，则将显示证书上传表单（单击 *Show Advanced Settings* 以查找此参数）。

- 在 **ATTRIBUTE STATEMENTS (OPTIONAL)** 添加一个属性语句:
  - *Name:* usrEmail
  - *Name format:* Unspecified
  - *Value:* user.email



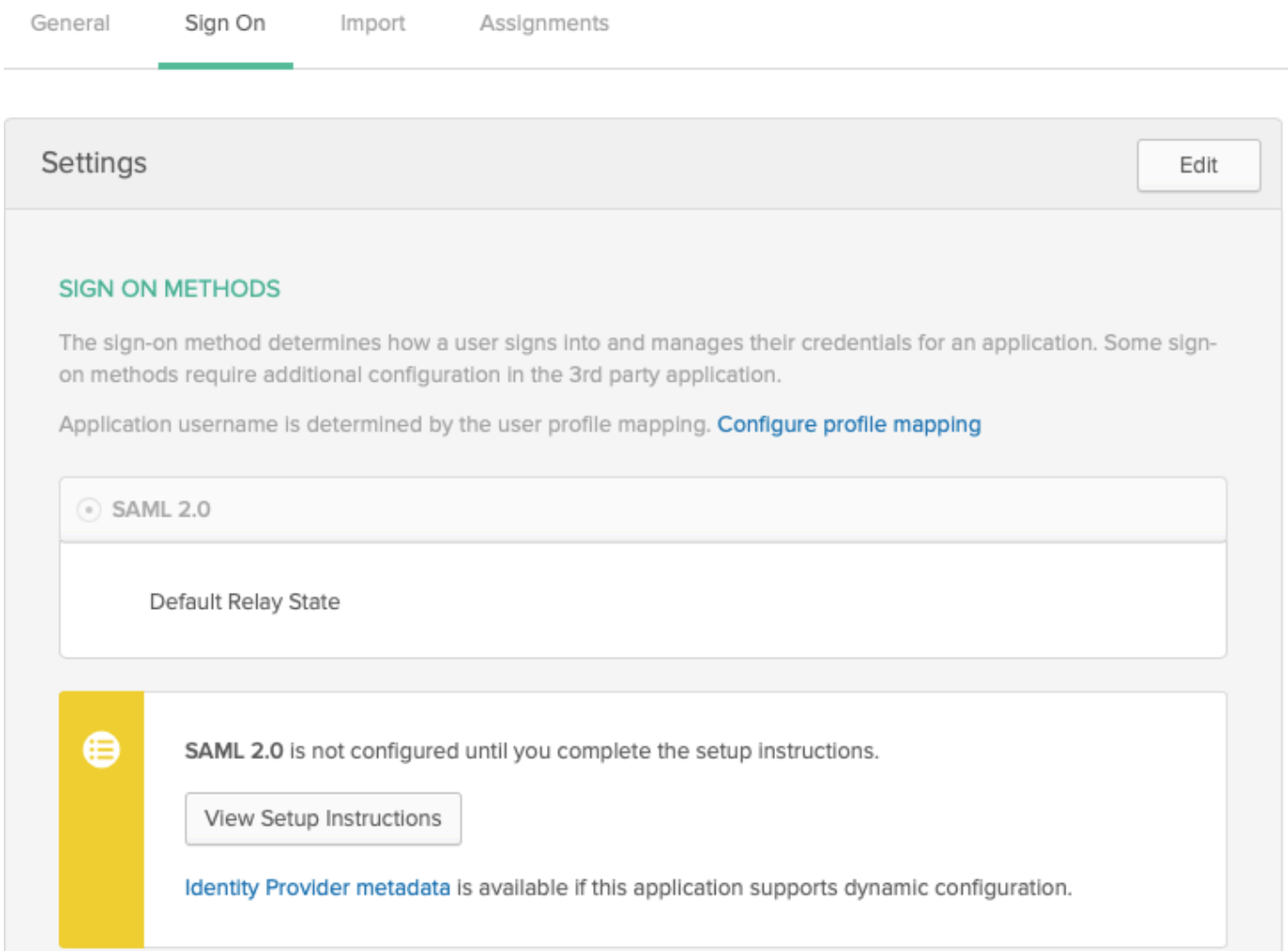
6. 在下一个选项卡中，选择"I'm a software vendor. I'd like to integrate my app with Okta"，然后点击"Finish"。

7. 导航至 *Assignments* 选项，并点击”Assign”，然后从下拉列表中选择 *Assign to People* .



8. 在弹出窗口中，将创建的应用分配给使用 SAML 2.0 身份验证的 Zabbix 人员，然后按”Save and go back”.

9. 导航到 *Sign On* 选项卡，然后按”View Setup Instructions” 按钮。新选项卡将显示设置说明; 在配置 Zabbix 时保持此选项卡打开。



Zabbix 配置

1. 在 Zabbix 中，转到 *Administration* → *Authentication* 中的 SAML 设置，并将 Okta 设置指令中的信息复制到相应的字段中

- Identity Provider Single Sign-On URL → SSO service URL
- Identity Provider Issuer → IdP entity ID
- Username attribute → Attribute name (usrEmail)
- SP entity ID → Audience URI

2. 将 Okta 设置说明页面中提供的证书下载到 *ui/conf/certs* 文件夹中，作为 idp.crt，并设置权限为 644：

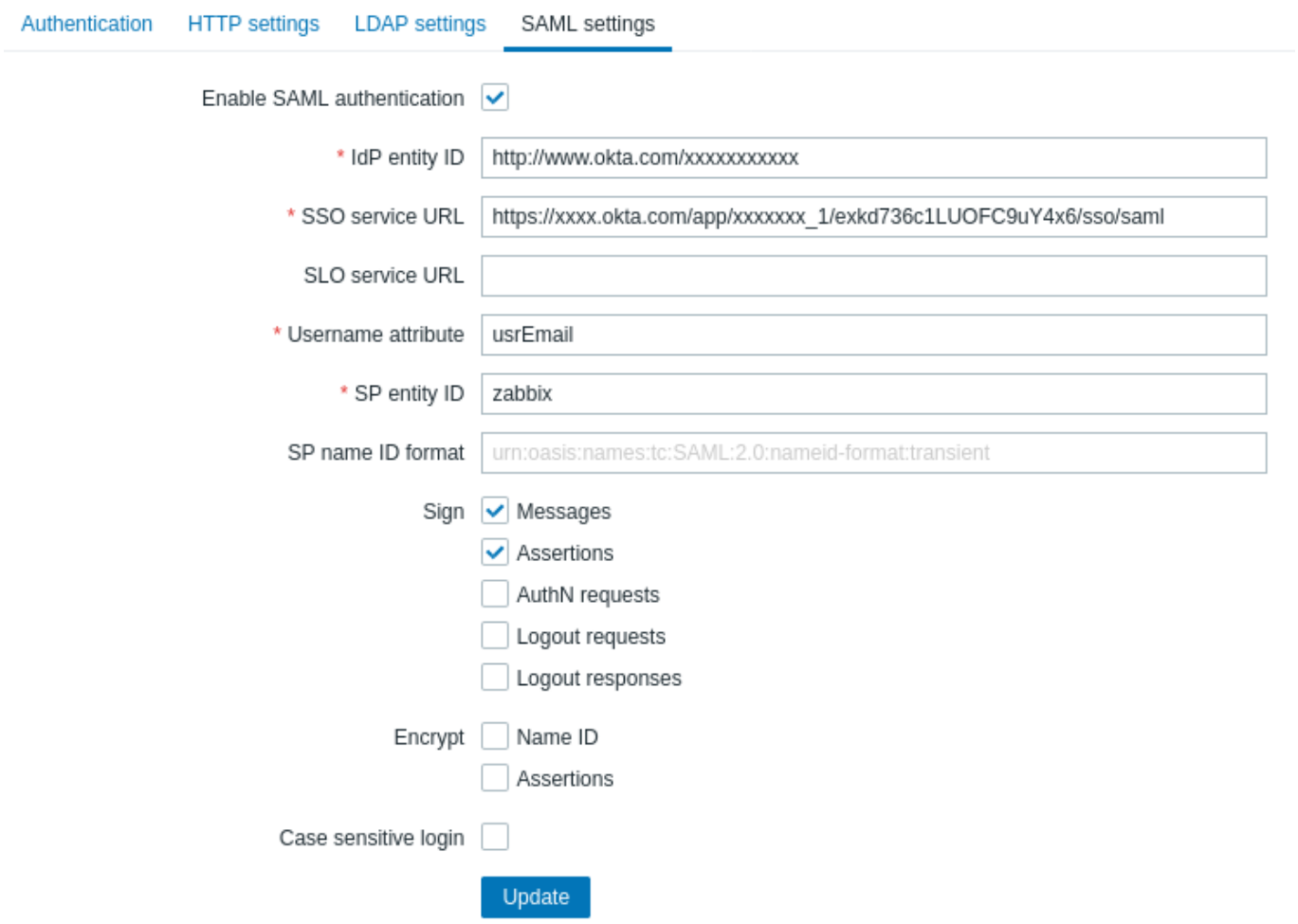
```
chmod 644 idp.crt
```

请注意，如果您已从旧版本升级到 Zabbix 5.0，则还需要手动将这些行添加到 zabbix.conf.php 文件中 (located in the //ui/conf/ // directory):

```
// Used for SAML authentication.
$SSO['SP_KEY'] = 'conf/certs/sp.key'; // Path to your private key.
$SSO['SP_CERT'] = 'conf/certs/sp.crt'; // Path to your public key.
$SSO['IDP_CERT'] = 'conf/certs/idp.crt'; // Path to IdP public key.
$SSO['SETTINGS'] = []; // Additional settings
```

有关更多详细信息的说明。请参阅SAML Authentication

3. 如果 *Assertion Encryption* 已设置为在 Okta 中加密，则选中”Assertions” 参数的 *Encrypt* 也应在 Zabbix 中标记



4. 按”Update” 按钮保存这些设置。

Note:
要使用 SAML 登录，Zabbix 中的用户名应与 Okta 电子邮件匹配。这些设置可以在 Zabbix Web 界面的 *Administration* → *Users* 进行更改。

12 Oracle 数据库配置

概述

本节包含有关创建 Oracle 数据库以及配置数据库与 Zabbix server、proxy 和 agent 之间的连接的说明。

数据库创建

假设有一个密码为 *password* 的 *zabbix* 数据库用户，并且有权在 Oracle 数据库上的 ORCL 服务中创建数据库对象。Zabbix 需要一个 Unicode 数据库字符集和一个 UTF8 字符集。检查当前的设置:

```
sqlplus> select parameter,value from v$nls_parameters where parameter='NLS_CHARACTERSET' or parameter='NLS
```

现在准备数据库:

```
cd /path/to/zabbix-sources/database/oracle
sqlplus zabbix/password@oracle_host/ORCL
sqlplus> @schema.sql
# stop here if you are creating database for Zabbix proxy
sqlplus> @images.sql
sqlplus> @data.sql
```

Note:
请设置初始化参数以达到最佳的性能。CURSOR_SHARING=FORCE

连接设置

Zabbix 支持两种类型的连接标识符（连接方法）：

- Easy Connect
- Net Service Name

Zabbix server 和 Zabbix proxy 的连接配置参数可以在配置文件中设置。server 和 proxy 的重要参数是 *DBHost*, *DBUser*, *DBName* 和 *DBPassword*。相同的参数对于前端很重要：*$DB["SERVER"]*, *$DB["PORT"]*, *$DB["DATABASE"]*, *$DB["USER"]*, *$DB["PASSWORD"]*.

Zabbix 使用以下连接字符串语法：

```
{DBUser/DBPassword[@<connect_identifier>]}
```

<connect_identifier> 可以以以下形式指定"Net Service Name" 或"Easy Connect".

```
@[[//]Host[:Port]/<service_name> | <net_service_name>]
```

Easy Connect

Easy Connect 使用以下参数连接到数据库：

- *Host* - 数据库服务器的主机名或 IP 地址（配置文件中的 DBHost 参数）。
- *Port* - 数据库服务器上的监听端口（配置文件中的 DBPort 参数; 如果未设置，将使用默认的 1521 端口）
- <service_name> - 要访问数据库的服务名称（配置文件中的 DBName 参数）。

例**:** 在 server 或 proxy 配置文件中设置的数据库参数 (zabbix_server.conf and zabbix_proxy.conf):

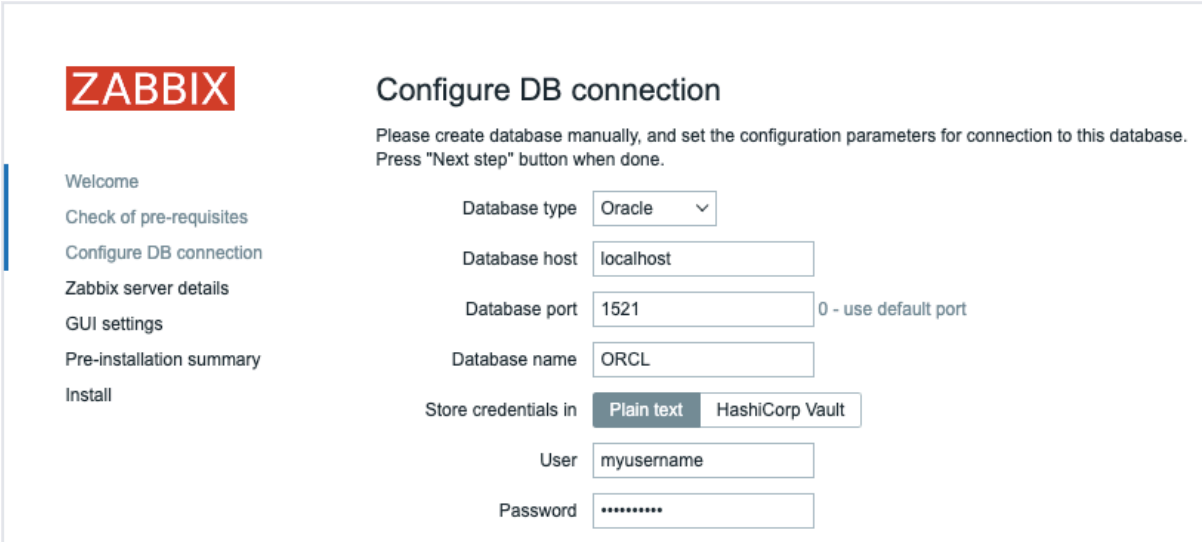
```
DBHost=localhost
DBPort=1521
DBUser=myusername
DBName=ORCL
DBPassword=mypassword
```

Zabbix 用于建立连接的字符串:

```
DBUser/DBPassword@DBHost:DBPort/DBName
```

在 Zabbix 前端安装过程中，在安装向导的 *Configure DB connection* 步骤中设置相应的参数：

- Database host: localhost
- Database port: 1521
- Database name: ORCL
- User: myusername
- Password: mypassword



或者，可以在前端配置文件 (zabbix.conf.php) 中设置这些参数:

```
$DB["TYPE"]                     = 'ORACLE';
$DB["SERVER"]                   = 'localhost';
$DB["PORT"]             = '1521';
$DB["DATABASE"]                 = 'ORCL';
$DB["USER"]                     = 'myusername';
$DB["PASSWORD"]                 = 'mypassword';
```

网络服务名称

从 Zabbix 5.4.0 开始，可以使用 net service name 连接到 Oracle。

<net_service_name> 是解析为连接描述符服务的简单名称。

为了使用服务名称创建连接，必须在位于数据库服务器和客户端系统上的 tnsnames.ora 文件中定义此服务名称。确保连接成功的最简单方法是在 TNS_ADMIN 环境变量中定义 oftnsnames.ora 文件的位置。tnsnames.ora 文件的默认位置为：

```
$ORACLE_HOME/network/admin/
```

一个简单的 tnsnames.ora 文件示例：

```
ORCL =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = ORCL)
    )
  )
```

若要设置”Net Service Name” 连接方法，请使用下列选项：

- 将 DBHost 设置为空，并设置 DBName：

```
DBHost=
DBName=ORCL
```

- 配置两个参数且值为空：

```
DBHost=
DBName=
```

在第二种情况下，必须设置 TWO_TAKS 环境变量。它指定默认的远程 Oracle 服务（服务名称）。定义此变量后，连接器将使用接受连接请求的 Oracle 监听连接到指定的数据库。此变量仅在 Linux 和 UNIX 上使用。Microsoft Windows 使用 LOCAL 环境变量

例**:**

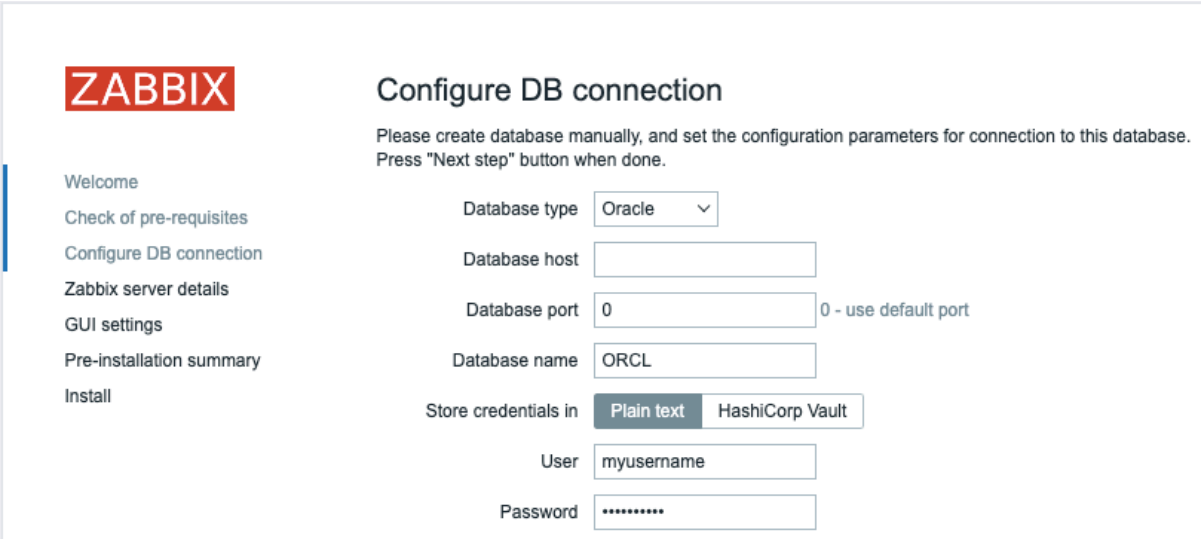
使用设置为 ORCL 和默认端口的 Net Service Name 连接到数据库。在 server 或 proxy 配置文件（zabbix_server.conf 和 zabbix_proxy.conf）中设置的数据库参数:

```
DBHost=
#DBPort=
DBUser=myusername
```

```
DBName=ORCL
DBPassword=mypassword
```

在 Zabbix 前端安装过程中，在安装向导的 *Configure DB connection* 步骤中设置相应的参数：

- Database host:
- Database port: 0
- Database name: ORCL
- User: myusername
- Password: mypassword



或者，可以在前端配置文件中设置这些参数 (zabbix.conf.php):

```
$DB["TYPE"]                     = 'ORACLE';
$DB["SERVER"]                   = '';
$DB["PORT"]             = '0';
$DB["DATABASE"]                 = 'ORCL';
$DB["USER"]                     = 'myusername';
$DB["PASSWORD"]                 = 'mypassword';
```

Zabbix 用于建立连接的字符串:

```
DBUser/DBPassword@ORCL
```

已知问题

为了提高性能，您可以将字段类型从 *nclob* 转换为 *nvarchar2*，请参阅已知问题。

13 设置定时报表

概述

本节提供了关于安装 Zabbix Web 服务以及配置 Zabbix 以启用定时报表生成的说明。

> **Attention:**
> 目前定时报表的支持是实验性的。

安装

应安装新的Zabbix Web 服务 进程和 Google Chrome 浏览器，以便生成预定报告。Web 服务可以安装在安装 Zabbix 服务器的同一台计算机上，也可以安装在不同的计算机上。Google Chrome 浏览器应安装在安装 Web 服务的同一台计算机上。

官方的 zabbix-web-service 软件包可在 Zabbix 存储库 中找到。Google Chrome 浏览器不包含在这些软件包中，必须单独安装。

要从源代码编译 Zabbix web 服务，请参阅安装 Zabbix web 服务。

安装后，在安装 web 服务的机器上运行 zabbix_web_service：

```
zabbix_web_service
```

配置

为确保所有相关元素之间能够正常通信，请确保服务器配置文件和前端配置参数配置正确。

Zabbix server

Zabbix server 配置文件中的以下参数需要更新：*WebServiceURL* 和 *StartReportWriters*。

WebServiceURL

此参数是启用与 Web 服务的通信所必需的。URL 应采用 `<host:port>/report` 格式。

- 默认情况下，Web 服务侦听端口 10053。可以在 Web 服务配置文件 中指定其他端口。
- 必须指定 `/report` 路径（该路径是硬编码的，无法更改）。

示例：

WebServiceURL=http://localhost:10053/report

StartReportWriters

此参数确定应启动多少个报告编写器进程。如果未设置或等于 0，则禁用报告生成。根据所需报告的数量和频率，可以启用 1 到 100 个报告编写器进程。

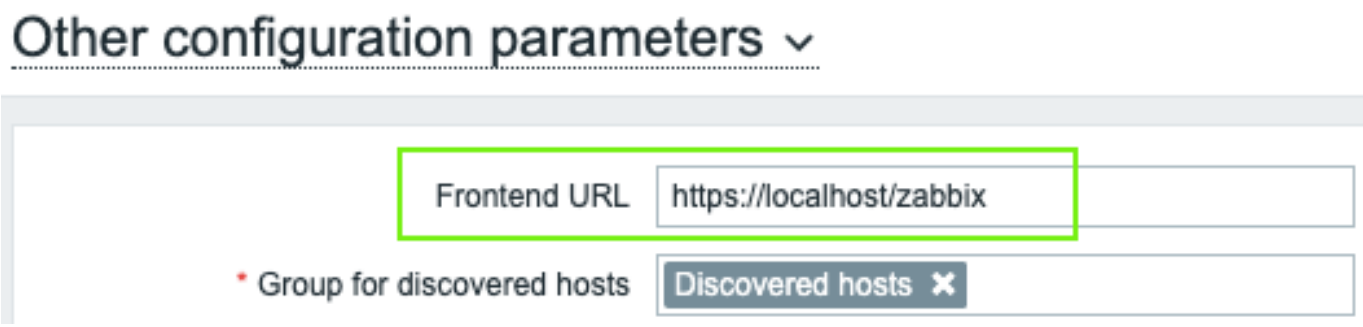
示例：

StartReportWriters=3

Zabbix 前端

应设置 前端 *URL* 参数以启用 Zabbix 前端和 Zabbix Web 服务之间的通信：

- 继续到 管理 → 常规 → 其他参数前端菜单部分
- 在 前端 *URL* 参数中指定 Zabbix Web 界面的完整 URL



Note:
设置过程完成后，您可能需要配置并发送测试报告 以确保一切正常。

14 其他前端语言

概述

为了在 Zabbix Web 界面中使用英语以外的任何其他语言，其区域设置应安装在 Web 服务器上。此外，翻译工作还需要 PHP gettext 扩展。

安装区域配置

要列出所有已安装的语言，请运行:

```
locale -a
```

如果未列出所需的某些语言，请打开*/etc/locale.gen* 文件并取消注释所需的语言环境。由于 Zabbix 使用 UTF-8 编码，因此您需要选择具有 UTF-8 字符集的区域设置。

运行:

```
locale-gen
```

重新启动 web 服务.

安装完区域设置后。可能需要用 Ctrl + F5 在浏览器中重新加载 Zabbix 前端页面才能显示新语言。

安装 Zabbix

如果直接从 Zabbix git repository, 安装 Zabbix，则应手动生成翻译文件，运行:

```
make gettext
locale/make_mo.sh
```

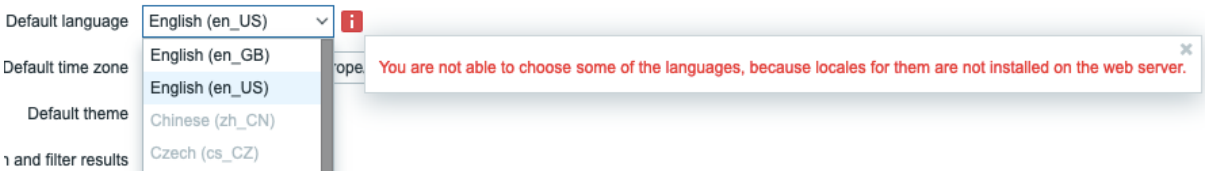
从软件包或源码 tar.gz 文件安装 Zabbix 时不需要此步骤

选择语言

有几种方法可以在 Zabbix Web 界面中选择语言：

- 安装 Web 界面时 - 在前端installation wizard 中。选择的语言将被设置为系统默认值。
- 安装完成后，系统默认语言可以在 *Administration→General→GUI* menu section 找到.
- 特定用户的语言可以在user profile找到.

如果计算机上未安装某种语言的区域设置，则此语言将在 Zabbix 语言选择器中显示为灰色。如果没有一个可用的区域设置，则在语言选择器旁边显示一个红色图标。按下此图标后，将显示以下消息：“您无法选择某些语言，因为 Web 服务器上未安装它们。”



3 后台进程配置

1 Zabbix 服务器

概述

本节列出了 Zabbix 服务器配置文件（Zabbix_server.conf）中支持的参数。请注意：

- 默认值反映的是守护进程的默认值，而不是附带的配置文件中的值；
- Zabbix 仅支持 UTF-8 编码的配置文件，不支持 BOM 格式；
- 以 “#” 开头的注释只支持在行首。

参数

参数	必填	范围	默认值	描述
AlertScriptsPath	否		/usr/local/share/zabbix/alertscripts	自定义告警脚本的位置（取决于编译时安装变量 *datadir*）。
AllowRoot	否		0	允许服务器以’root’ 身份运行。如果禁用，并且服务器是由’root’ 启动的，则服务器将尝试切换到’zabbix’ 用户。如果以普通用户身份启动，则此设置无效。 0 - 不允许 1 - 允许
AllowUnsupportedDBVersions	否		0	允许服务器使用不支持的数据库版本工作。 0 - 不允许 1 - 允许
CacheSize	否	128K-64G	32M	配置缓存的大小，单位为字节。 共享内存的大小，用于存储主机、监控项和触发器数据。
CacheUpdateFrequency	否	1-3600	60	确定 Zabbix 执行配置缓存更新的频率，单位为秒。 另请参阅运行时控制 选项。
DBHost	否		localhost	数据库主机名。 对于 MySQL，localhost 或空字符串将使用套接字。对于 PostgreSQL 只有空字符串将尝试使用套接字。 对于Oracle 空字符串将使用网络服务名称连接方法；在这种情况下，请考虑使用 TNS_ADMIN 环境变量来指定 tnsnames.ora 文件的目录。
DBName	是			数据库名称。 对于Oracle 如果使用网络服务名称连接方法，请指定 tnsnames.ora 中的服务名称或将其设置为空字符串；如果 DBName 设置为空字符串，则设置 TWO_TASK 环境变量。
DBPassword	否			数据库密码。 如果不使用密码，请注释掉此行。

参数	必填	范围	默认值	描述
DBPort	否	1024-65535		使用非本地套接字时的数据库端口。 对于Oracle 如果使用网络服务名称连接方法，此参数将被忽略；将使用 tnsnames.ora 文件中的端口号。
DBSchema	否			架构名称。PostgreSQL 使用。
DBSocket	否			MySQL 套接字文件的路径。
DBUser	否			数据库用户。
DBTLSConnect	否			设置此选项强制使用 TLS 连接到数据库： *required* - 使用 TLS 连接 *verify_ca* - 使用 TLS 并验证证书 *verify_full* - 使用 TLS，验证证书，并验证由 DBHost 指定的数据库身份是否与其证书匹配 从 MySQL 5.7.11 和 PostgreSQL 开始支持以下值：”required”, ”verify_ca”, ”verify_full”。从 MariaDB 版本 10.2.6 开始支持”required” 和”verify_full” 值。 默认未设置任何选项，行为取决于数据库配置。 此参数自 Zabbix 5.0.0 起支持。
DBTLSCAFile	否 (如果是 DBTLSConnect 设置为 verify_ca 或 verify_full 则为是)			包含用于数据库证书验证的顶级 CA 证书的文件的完整路径名。 此参数自 Zabbix 5.0.0 起支持。
DBTLSCertFile	否			包含用于向数据库进行身份验证的 Zabbix server 证书的文件的完整路径名。 此参数自 Zabbix 5.0.0 起支持。
DBTLSKeyFile	否			包含用于向数据库进行身份验证的私钥的文件的完整路径名。 此参数自 Zabbix 5.0.0 起支持。
DBTLSCipher	否			Zabbix server 允许用于 TLS 协议的加密密码列表，直到 TLSv1.2。 仅 MySQL 支持。 此参数自 Zabbix 5.0.0 起支持。
DBTLSCipher13	否			Zabbix server 允许用于 TLSv1.3 协议的加密密码列表。 仅 MySQL 支持，从版本 8.0.16 开始。 此参数自 Zabbix 5.0.0 起支持。
DebugLevel	否	0-5	3	指定调试级别： 0 - Zabbix 进程启动和停止的基本信息 1 - 关键信息 2 - 错误信息 3 - 警告 4 - 用于调试（产生大量信息） 5 - 扩展调试（产生更多信息） 另请参阅运行时控制 选项。
ExportDir	否			用于事件、历史和趋势的实时导出 的目录，采用换行符分隔的 JSON 格式。如果设置，则启用实时导出。 此参数自 Zabbix 4.0.0 起支持。
ExportFileSize	否	1M-1G	1G	每个导出文件的最大大小，单位为字节。仅在设置了 ExportDir 时用于轮换。 此参数自 Zabbix 4.0.0 起支持。
ExportType	否			用于实时导出 的实体类型列表（默认为所有类型），以逗号分隔（事件、历史、趋势）。仅在设置了 ExportDir 时有效。 注意如果指定了 ExportType，但未设置 ExportDir，则这是一个配置错误，服务器将无法启动。 例如： ExportType=history,trends - 仅导出历史和趋势 ExportType=events - 仅导出事件
ExternalScripts	否		/usr/local/share/zabbix/externalscripts（取决于编译时安装变量 *datadir*）。	外部脚本的位置。

参数	必填	范围	默认值	描述
Fping6Location	否		/usr/sbin/fping6	fping6 的位置。 确保 fping6 二进制文件具有 root 所有者身份，并设置了 SUID 标志。 如果您的 fping 实用程序能够处理 IPv6 地址，请将其留空（”Fping6Location=”）。
FpingLocation	否		/usr/sbin/fping	fping 的位置。 确保 fping 二进制文件具有 root 所有者身份，并设置了 SUID 标志！
HANodeName	否			高可用性集群节点名称。 当为空时，服务器以独立模式工作，并创建一个名称为空的节点。
HistoryCacheSize	否	128K-2G	16M	历史缓存的大小，单位为字节。 共享内存大小，用于存储历史数据。
HistoryIndexCacheSize	否	128K-2G	4M	历史索引缓存的大小，单位为字节。 共享内存大小，用于对存储在历史缓存中的历史数据建立索引。 索引缓存大小大约需要 100 字节来缓存一个监控项。 此参数自 Zabbix 3.0.0 起支持。
HistoryStorageDateIndex	否		0	启用历史存储中历史值的预处理，以便根据日期将值存储在不同的索引中： 0 - 禁用 1 - 启用
HistoryStorageURL	否			历史存储的 HTTP[S] URL。 此参数用于Elasticsearch 配置。
HistoryStorageTypes	否		uint,dbl,str,log,text	要发送到历史存储的值类型的逗号分隔列表。 此参数用于Elasticsearch 配置。
HousekeepingFrequency	否	0-24	1	确定 Zabbix 执行维护操作的频率，单位为小时。 维护是从数据库中删除过时信息的过程。 注意：为了防止管家过载（例如，当历史和趋势周期大大减少时），在一个维护周期中，每个监控项删除的过时信息不超过 HousekeepingFrequency 小时的 4 倍。因此，如果 HousekeepingFrequency 为 1，则每个周期删除的过时信息不超过 4 小时（从最旧的条目开始）。 注意：为了降低服务器启动时的负载，维护在服务器启动后推迟 30 分钟。因此，如果 HousekeepingFrequency 为 1，则服务器启动后的第一次维护操作将在 30 分钟后运行，之后每隔一小时重复一次。 自 Zabbix 3.0.0 起，可以通过将 HousekeepingFrequency 设置为 0 来禁用自动维护。在这种情况下，维护操作只能通过 *housekeeper_execute* 运行时控制选项启动，在一个维护周期中删除的过时信息周期是自上次维护周期以来的周期的 4 倍，但不少于 4 小时，也不多于 4 天。 另请参阅运行时控制 选项。
Include	否			您可以在配置文件中包含单个文件或目录中的所有文件。 要仅包含指定目录中的相关文件，支持使用星号通配符进行模式匹配。例如： `/absolute/path/to/config/files/*.conf`。 请参阅特殊说明了解限制。
JavaGateway	否			Zabbix Java 网关的 IP 地址（或主机名）。 仅在启动 Java 轮询器时需要。
JavaGatewayPort	否	1024-32767	10052	Zabbix Java 网关监听的端口。
ListenBacklog	否	0 - INT_MAX	SOMAXCONN	TCP 队列中挂起连接的最大数量。 默认值是一个硬编码常量，这取决于系统。 支持的最大值也取决于系统，过高的值可能会被静默截断为 “实现指定的最大值”。
ListenIP	否		0.0.0.0	trapper 应该监听的逗号分隔的 IP 地址列表。 如果缺少此参数，trapper 将监听所有网络接口。
ListenPort	否	1024-32767	10051	trapper 的监听端口。

参数	必填	范围	默认值	描述
LoadModule	否			在服务器启动时加载的模块。模块用于扩展服务器的功能。 格式： LoadModule=<module.so> LoadModule=<path/module.so> LoadModule=</abs_path/module.so> 模块必须位于 LoadModulePath 指定的目录中，或者路径必须在模块名称之前。如果前面的路径是绝对路径（以'/' 开头），则忽略 LoadModulePath。 允许包含多个 LoadModule 参数。
LoadModulePath	否			服务器模块的位置的完整路径。 默认值取决于编译选项。
LogFile	是，如果 LogType 设置为 *file*，否则否			日志文件的名称。
LogFileSize	否	0-1024	1	日志文件的最大大小（MB）。 0 - 禁用自动日志轮转。 注意：如果日志文件大小达到限制并且文件轮转失败，无论出于何种原因，现有的日志文件将被截断并重新开始。
LogType	否		file	日志输出类型： *file* - 写入由 LogFile 参数指定的日志文件， *system* - 写入系统日志， *console* - 输出到控制台。 此参数自 Zabbix 3.0.0 起支持。
LogSlowQueries	否	0-3600000	0	确定数据库查询在被记录为慢查询之前的最长时间（毫秒）。 0 - 不记录慢查询。 此选项从 DebugLevel=3 开始启用。
MaxHousekeeperDelete	否	0-1000000	5000	每个任务在一个维护周期中删除的行数（对应于 [tablename], [field], [value]）不超过'MaxHousekeeperDelete'。 如果设置为 0，则不使用任何限制。在这种情况下，您必须知道自己在做什么，以免过度加载数据库! [2] 此参数仅适用于已删除监控项的历史和趋势数据的删除。
NodeAddress	否		10051	IP 或主机名，可选端口，用以覆盖前端应如何连接到服务器的方式。 格式：<address>[:port] 如果未设置 IP 或主机名，则将使用 ListenIP 的值。如果 ListenIP 也未设置，则将使用值 `localhost`。 此选项可以被前端配置中指定的地址覆盖。 另见：HANodeName 参数
PidFile	否		/tmp/zabbix_server.pid	PID 文件的名称。
ProxyConfigFrequency	否	1-604800	3600	确定 Zabbix server 向 Zabbix proxy 发送配置数据的频率，单位为秒。仅用于被动模式的 proxies。
ProblemHousekeepingFrequency	否	1-3600	60	确定 Zabbix 删除已删除触发器的问题的频率，单位为秒。
ProxyDataFrequency	否	1-3600	1	确定 Zabbix server 从 Zabbix proxy 请求历史数据的频率，单位为秒。仅用于被动模式的 proxies。
ServiceManagerSyncFrequency	否	1-3600	60	确定 Zabbix 同步服务管理器配置的频率，单位为秒。
SNMPTrapperFile	否		/tmp/zabbix_traps.tmp	用于从 SNMP trap 守护进程向服务器传递数据的临时文件。 必须与 zabbix_trap_receiver.pl 或 SNMPTT 配置文件中的相同。
SocketDir	否		/tmp	存储内部 Zabbix 服务使用的 IPC 套接字的目录。 此参数自 Zabbix 3.4.0 起支持。
SourceIP	否			源 IP 地址用于： - 与 Zabbix proxy 和 Zabbix agent 的出站连接； - 无代理连接（VMware、SSH、JMX、SNMP、Telnet 和简单检查）； - HTTP agent 连接； - 脚本监控项 JavaScript HTTP 请求； - 预处理 JavaScript HTTP 请求； - 发送通知邮件（连接到 SMTP 服务器）； - Webhook 通知（JavaScript HTTP 连接）； - 连接到 Vault
SSHKeyLocation	否			SSH 检查和操作的公钥和私钥位置

参数	必填	范围	默认值	描述
SSLCertLocation	否			SSL 客户端证书文件的位置，用于客户端认证。 此参数仅用于 Web 监控。
SSLKeyLocation	否			SSL 私钥文件的位置，用于客户端认证。 此参数仅用于 Web 监控。
SSLCALocation	否			覆盖 SSL 服务器证书验证的证书颁发机构（CA）文件的位置。如果没有设置，将使用系统范围的目录。 注意，此参数的值将被设置为 libcurl 选项 CURLOPT_CAPATH。对于 7.42.0 之前的 libcurl 版本，这只有在 libcurl 编译为使用 OpenSSL 时才有效。有关更多信息，请参见 cURL 网页。 此参数自 Zabbix 2.4.0 起用于 Web 监控，自 Zabbix 3.0.0 起用于 SMTP 认证。
StartAlerters	否	1-100	3	alerters 的预派生实例数量。 此参数自 Zabbix 3.4.0 起支持。
StartDBSyncers	否	1-100	4	history syncers 的预派生实例数量。 注意：更改此值时要小心，增加它可能弊大于利。大致上，默认值应该足以处理多达 4000 NVPS。
StartDiscoverers	否	0-250	1	discoverers 的预派生实例数量。
StartEscalators	否	1-100	1	escalators 的预派生实例数量。 此参数自 Zabbix 3.0.0 起支持。
StartHistoryPollers	否	0-1000	5	history pollers 的预派生实例数量。 此参数自 Zabbix 5.4.0 起支持。
StartHTTPPollers	否	0-1000	1	HTTP pollers 的预派生实例数量。
StartIPMIPollers	否	0-1000	0	IPMI pollers 的预派生实例数量。
StartJavaPollers	否	0-1000	0	Java pollers 的预派生实例数量。
StartLLDProcessors	否	1-100	2	低级发现 (LLD) workers 的预派生实例数量。 当启动 LLD 工作进程时，LLD 管理进程将自动启动。 此参数自 Zabbix 4.2.0 起支持。
StartODBCPollers	否	0-1000	1	ODBC pollers 的预派生实例数量。
StartPingers	否	0-1000	1	ICMP pingers 的预派生实例数量。
StartPollersUnreachable	否	0-1000	1	pollers for unreachable hosts 的预派生实例数量（包括 IPMI 和 Java） 如果启动了常规、IPMI 或 Java 轮询器，则至少必须运行一个无法到达主机的轮询器。
StartPollers	否	0-1000	5	pollers 的预派生实例数量。
StartPreprocessors	否	1-1000	3	预处理workers 的预派生实例数量。 当启动预处理工作进程时，预处理管理进程将自动启动。 此参数自 Zabbix 3.4.0 起支持。
StartProxyPollers	否	0-250	1	pollers for passive proxies 的预派生实例数量。
StartReportWriters	否	0-100	0	report writers 的预派生实例数量。 如果设置为 0，则禁用计划报表生成。 当启动报表编写器时，报表管理进程将自动启动。 此参数自 Zabbix 5.4.0 起支持。
StartSNMPTrapper	否	0-1	0	如果设置为 1，则将启动SNMP trapper 进程。
StartTimers	否	1-1000	1	timers 的预派生实例数量。 计时器处理维护周期。
StartTrappers	否	0-1000	5	trappers 的预派生实例数量。 轮询器接受来自 Zabbix sender、主动 agents 和主动 proxies 的传入连接。
StartVMwareCollectors	否	0-250	0	VMware collector 实例的预派生数量。
StatsAllowedIP	否			逗号分隔的 IP 地址列表，可选地使用 CIDR 表示法，或外部 Zabbix 实例的 DNS 名称。只接受这里列出的地址的统计请求。如果此参数未设置，则不接受任何统计请求。 如果启用了 IPv6 支持，则'127.0.0.1', '::127.0.0.1', '::ffff:127.0.0.1' 被视为等效，'::/0' 将允许任何 IPv4 或 IPv6 地址。'0.0.0.0/0' 可用于允许任何 IPv4 地址。 示例：StatsAllowedIP=127.0.0.1,192.168.1.0/24,::1,2001:db8::/32,zabbix.example.com 此参数自 Zabbix 4.2.0 起支持。
Timeout	否	1-30	3	指定我们等待 agent、SNMP 设备或外部检查的秒数。
TLSCAFile	否			包含用于对等证书验证的顶级 CA 证书的文件的完整路径名，用于 Zabbix 组件之间的加密通信。 此参数自 Zabbix 3.0.0 起支持。

参数	必填	范围	默认值	描述
TLSCertFile	否			包含服务器证书或证书链的文件的完整路径名，用于 Zabbix 组件之间的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSCipherAll	否			GnuTLS 优先级字符串或 OpenSSL（TLS 1.2）密码字符串。覆盖基于证书和基于 PSK 的加密的默认密码套件选择标准。 示例： TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_1 此参数自 Zabbix 4.4.7 起支持。
TLSCipherAll13	否			OpenSSL 1.1.1 或更高版本在 TLS 1.3 中的密码字符串。覆盖基于证书和基于 PSK 的加密的默认密码套件选择标准。 GnuTLS 示例：NONE:+VERS-TLS1.2:+ECDHE-RSA:+RSA:+ECDHE-PSK:+PSK:+AES-128-GCM:+AES-128-CBC:+AEAD:+SHA256:+SHA1:+CURVE-ALL:+COMP-NULL::+SIGN-ALL:+CTYPE-X.509 OpenSSL 示例： EECDH+aRSA+AES128:RSA+aRSA+AES128:kECDHEPSK+AES128:kPSK+A 此参数自 Zabbix 4.4.7 起支持。
TLSCipherCert	否			GnuTLS 优先级字符串或 OpenSSL（TLS 1.2）密码字符串。覆盖基于证书的加密的默认密码套件选择标准。 GnuTLS 示例： NONE:+VERS-TLS1.2:+ECDHE-RSA:+RSA:+AES-128-GCM:+AES-128-CBC:+AEAD:+SHA256:+SHA1:+CURVE-ALL:+COMP-NULL:+SIGN-ALL:+CTYPE-X.509 OpenSSL 示例：EECDH+aRSA+AES128:RSA+aRSA+AES128 此参数自 Zabbix 4.4.7 起支持。
TLSCipherCert13	否			OpenSSL 1.1.1 或更高版本在 TLS 1.3 中的密码字符串。覆盖基于证书的加密的默认密码套件选择标准。 此参数自 Zabbix 4.4.7 起支持。
TLSCipherPSK	否			GnuTLS 优先级字符串或 OpenSSL（TLS 1.2）密码字符串。覆盖基于 PSK 的加密的默认密码套件选择标准。 GnuTLS 示例：NONE:+VERS-TLS1.2:+ECDHE-PSK:+PSK:+AES-128-GCM:+AES-128-CBC:+AEAD:+SHA256:+SHA1:+CURVE-ALL:+COMP-NULL:+SIGN-ALL OpenSSL 示例：kECDHEPSK+AES128:kPSK+AES128 此参数自 Zabbix 4.4.7 起支持。
TLSCipherPSK13	否			OpenSSL 1.1.1 或更高版本在 TLS 1.3 中的密码字符串。覆盖基于 PSK 的加密的默认密码套件选择标准。 示例： TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256 此参数自 Zabbix 4.4.7 起支持。
TLSCRLFile	否			包含吊销证书的文件的完整路径名。此参数用于 Zabbix 组件之间的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSKeyFile	否			包含服务器私钥的文件的完整路径名，用于 Zabbix 组件之间的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TmpDir	否		/tmp	临时目录。
TrapperTimeout	否	1-300	300	指定 trapper 处理新数据可能花费的秒数。
TrendCacheSize	否	128K-2G	4M	趋势缓存的大小，单位为字节。 共享内存大小，用于存储趋势数据。
TrendFunctionCacheSize	否	128K-2G	4M	趋势函数缓存的大小，单位为字节。 共享内存大小，用于缓存计算出的趋势函数数据。
UnavailableDelay	否	1-3600	60	确定在不可用 期间，以秒为单位检查主机可用性的频率。
UnreachableDelay	否	1-3600	15	确定在不可达 期间，以秒为单位检查主机可用性的频率。
UnreachablePeriod	否	1-3600	45	确定在多少秒的不可达 后，将主机视为不可用。
User	否		zabbix	将权限降低到系统上特定的、已存在的用户。 仅当以’root’ 身份运行且 AllowRoot 被禁用时有效。
ValueCacheSize	否	0,128K-64G	8M	历史值缓存的大小，单位为字节。 共享内存大小，用于缓存监控项历史数据请求。 设置为 0 将禁用值缓存（不推荐）。 当值缓存耗尽共享内存时，每 5 分钟会在服务器日志中写入一条警告消息。

参数	必填	范围	默认值	描述
VaultDBPath	否			通过键'password' 和'username' 检索数据库凭据的 Vault 路径。 示例：secret/zabbix/database 如果未指定 DBUser 和 DBPassword，则只能使用此选项。 此参数自 Zabbix 5.2.0 起支持。
VaultToken	否			应为 Zabbix server 专门生成的 Vault 认证令牌，该令牌仅具有对Vault 宏 中指定路径的只读权限，以及对可选的 VaultDBPath 配置参数指定路径的只读权限。 如果同时定义了 VaultToken 和 VAULT_TOKEN 环境变量，则为错误。 此参数自 Zabbix 5.2.0 起支持。
VaultURL	否		https://127.0.0.1:8200	Vault 服务器 HTTP[S] URL。如果未指定 SSLCALocation，则使用系统范围的 CA 证书目录。 此参数自 Zabbix 5.2.0 起支持。
VMwareCacheSize	否	256K-2G	8M	存储 VMware 数据的共享内存大小。 可以使用 VMware 内部检查 zabbix[vmware,buffer,...] 来监控 VMware 缓存使用情况（见内部检查）。 注意，如果没有配置启动的 vmware collector 实例，则不分配共享内存。
VMwareFrequency	否	10-86400	60	从单个 VMware 服务收集数据之间的延迟，单位为秒。 此延迟应设置为任何 VMware 监控项的最小更新间隔。
VMwarePerfFrequency	否	10-86400	60	从单个 VMware 服务检索性能计数器统计信息之间的延迟，单位为秒。 此延迟应设置为使用 VMware 性能计数器的任何 VMware 监控项的最小更新间隔。
VMwareTimeout	否	1-300	10	VMware collector 从 VMware 服务（vCenter 或 ESX hypervisor）等待响应的最大秒数。
WebServiceURL	否			Zabbix Web 服务的 HTTP[S] URL，格式为 `<host:port>/report`。例如：`http://localhost:10053/report` 此参数自 Zabbix 5.4.0 起支持。

附注

[1] 请注意，过多的数据采集进程（例如：pollers、unreachable pollers、ODBC pollers、HTTP pollers、Java pollers、pingers、trappers、proxypollers）连同 IPMI 管理进程、SNMP 捕获进程和预处理线程可能会耗尽预处理管理器的每进程文件描述符限制。

Warning:
上述行为会导致 Zabbix 服务器宕机（通常发生在启动后不久，或是启动后的一段时间内发生）。因此，您应修改配置文件或提高最大限制以避免这种情况的发生。

[2] 删除大量监控项时，会增加数据库的负载，这是因为删除某个监控项，管理程序需要删除该监控项的所有历史数据。例如，如果我们只需要删除 1 项监控项原型，但这个监控项原型链接到了 50 台主机，并且每台主机的监控项原型又扩展到了 100 个实际监控项，也就是说总共要删除 5000 个监控项（1*50*100）。如果 MaxHousekeeperDelete 设置为 500（MaxHousekeeperDelete=500），管理流程则必须在一个周期内从历史和趋势表中删除多达 2500000 个被删除监控项的相关值（5000*500）。

2 Zabbix proxy

概览

本节列出了 Zabbix proxy 的配置文件（zabbix_proxy.conf）中支持的参数。请注意：

- 默认值反映的是守护进程的默认值，而不是附带的配置文件中的值；
- Zabbix 仅支持 UTF-8 编码的配置文件，且编码中不可使用字节顺序标记（BOM）；
- 仅在行首支持以“#”开始的注释。

参数

参数	必填	范围	默认值	描述
AllowRoot	否		0	允许 proxy 以'root' 身份运行。如果禁用，并且 proxy 是由'root' 启动的，则 proxy 将尝试切换到'zabbix' 用户。如果以普通用户身份启动，则此设置无效。 0 - 不允许 1 - 允许
AllowUnsupportedDBVersions	否		0	允许 proxy 工作于不支持的数据库版本。 0 - 不允许 1 - 允许
CacheSize	否	128K-64G	32M	配置缓存的大小，单位为字节。 共享内存的大小，用于存储主机和监控项数据。
ConfigFrequency	否	1-604800	3600	proxy 从 Zabbix server 检索配置数据的频率，单位为秒。 主动 proxy 参数。对于被动 proxy 将被忽略（见 ProxyMode 参数）。
DataSenderFrequency	否	1-3600	1	proxy 每 N 秒向服务器发送收集到的数据。注意，主动 proxy 仍然会每秒轮询 Zabbix server 以获取远程命令任务。 主动 proxy 参数。对于被动 proxy 将被忽略（见 ProxyMode 参数）。
DBHost	否		localhost	数据库主机名。 对于 MySQL，localhost 或空字符串将使用套接字。对于 PostgreSQL 只有空字符串将尝试使用套接字。对于 Oracle，空字符串将使用网络服务名称连接方法；在这种情况下，考虑使用 TNS_ADMIN 环境变量来指定 tnsnames.ora 文件的目录。
DBName	是			数据库名称或 SQLite3 数据库文件的路径（Zabbix 的多进程架构不允许使用内存中数据库，例如 `:memory:`、`file::memory:?cache=shared` 或 `file:memdb1?mode=memory&cache=shared`）。 警告：不要尝试使用 Zabbix server 正在使用的相同数据库。 对于 Oracle，如果使用网络服务名称连接方法，请指定 tnsnames.ora 中的服务名称或将其设置为空字符串；如果 DBName 设置为空字符串，则设置 TWO_TASK 环境变量。
DBPassword	否			数据库密码。SQLite 忽略此设置。 如果不使用密码，请注释掉此行。
DBSchema	否			架构名称。PostgreSQL 使用。
DBSocket	否		3306	MySQL 套接字的路径。 不使用本地套接字时的数据库端口。SQLite 忽略此设置。
DBUser				数据库用户。SQLite 忽略此设置。
DBTLSConnect	否			设置此选项强制使用 TLS 连接到数据库： *required* - 使用 TLS 连接 *verify_ca* - 使用 TLS 并验证证书 *verify_full* - 使用 TLS，验证证书，并验证由 DBHost 指定的数据库身份是否与其证书匹配 在 MySQL 从 5.7.11 起和 PostgreSQL，支持以下值： "required"、"verify"、"verify_full"。在 MariaDB 从 10.2.6 版本起，支持"required" 和"verify_full" 值。 默认不设置任何选项，行为取决于数据库配置。 此参数自 Zabbix 5.0.0 起支持。
DBTLSCAFile	否 (如果是 DBTLSConnect 设置为 verify_ca 或 verify_full 则为是)			包含用于数据库证书验证的顶级 CA 证书的文件的完整路径名。 此参数自 Zabbix 5.0.0 起支持。
DBTLSCertFile	否			包含用于向数据库进行身份验证的 Zabbix server 证书的文件的完整路径名。 此参数自 Zabbix 5.0.0 起支持。
DBTLSKeyFile	否			包含用于向数据库进行身份验证的私钥的文件的完整路径名。 此参数自 Zabbix 5.0.0 起支持。

参数	必填	范围	默认值	描述
DBTLSCipher	否			Zabbix server 允许用于 TLS 协议的加密密码列表，直到 TLSv1.2。 仅 MySQL 支持。 此参数自 Zabbix 5.0.0 起支持。
DBTLSCipher13	否			Zabbix server 允许用于 TLSv1.3 协议的加密密码列表。 仅 MySQL 支持，从版本 8.0.16 开始。 此参数自 Zabbix 5.0.0 起支持。
DebugLevel	否	0-5	3	指定调试级别： 0 - 关于启动和停止 Zabbix 进程的基本信息 1 - 严重信息 2 - 错误信息 3 - 警告 4 - 调试（生成大量信息） 5 - 扩展调试（生成更多信息）
EnableRemoteCommands	否		0	是否允许来自 Zabbix server 的远程命令。 0 - 不允许 1 - 允许 此参数自 Zabbix 3.4.0 起支持。
ExternalScripts	否		/usr/local/share/zabbix/externalscripts（取决于编译时安装变量 *datadir*）。	外部脚本的位置。
Fping6Location	否		/usr/sbin/fping6	fping6 的位置。 确保 fping6 二进制文件具有 root 所有者身份，并设置了 SUID 标志。 如果您的 fping 实用程序能够处理 IPv6 地址，请将其留空（”Fping6Location=”）。
FpingLocation	否		/usr/sbin/fping	fping 的位置。 确保 fping 二进制文件具有 root 所有者身份，并设置了 SUID 标志！
HeartbeatFrequency	否	0-3600	60	心跳消息的频率，单位为秒。 用于监控 server 端 proxy 的可用性。 0 - 禁用心跳消息。 主动 proxy 参数。对于被动 proxy 将被忽略（见 ProxyMode 参数）。
HistoryCacheSize	否	128K-2G	16M	历史缓存的大小，单位为字节。 共享内存大小，用于存储历史数据。
HistoryIndexCacheSize	否	128K-2G	4M	历史索引缓存的大小，单位为字节。 共享内存大小，用于对存储在历史缓存中的历史数据建立索引。 索引缓存大小大约需要 100 字节来缓存一个监控项。 此参数自 Zabbix 3.0.0 起支持。
Hostname	否		由 HostnameItem 设置	唯一，区分大小写的 Proxy 名称。确保 proxy 名称为 server 所知！ 允许的字符：字母数字、’.’、’ ’、’_’ 和’-’。 最大长度：128
HostnameItem	否		system.hostname	如果未定义 Hostname（将与 agent 类似在 proxy 上运行），则用于设置 Hostname 的监控项。 不支持 UserParameters、性能计数器或别名，但支持 system.run[]。 如果设置了 Hostname，则忽略此设置。
HousekeepingFrequency	否	0-24	1	Zabbix 执行维护操作的频率（单位为小时）。 维护是从数据库中删除过时信息的过程。 注意：为了降低 proxy 启动时的负载，维护在 proxy 启动后推迟 30 分钟。因此，如果 HousekeepingFrequency 为 1，则 proxy 启动后的第一次维护操作将在 30 分钟后运行，之后每小时重复一次。 自 Zabbix 3.0.0 起，可以通过将 HousekeepingFrequency 设置为 0 来禁用自动维护。在这种情况下，维护操作只能通过 *housekeeper_execute* 运行时控制选项启动。
Include	否			您可以在配置文件中包含单个文件或目录中的所有文件。 要仅包含指定目录中的相关文件，支持使用星号通配符进行模式匹配。例如： `/absolute/path/to/config/files/*.conf`。 请参阅特殊说明了解限制。

参数	必填	范围	默认值	描述
JavaGateway	否			Zabbix Java 网关的 IP 地址（或主机名）。 仅在启动 Java 轮询器时需要。
JavaGatewayPort	否	1024-32767	10052	Zabbix Java 网关监听的端口。
ListenBacklog	否	0 - INT_MAX	SOMAXCONN	TCP 队列中挂起连接的最大数量。 默认值是一个硬编码常量，这取决于系统。 支持的最大值也取决于系统，过高的值可能会被静默截断为 “实现指定的最大值”。
ListenIP	否		0.0.0.0	trapper 应该监听的逗号分隔的 IP 地址列表。 如果缺少此参数，trapper 将监听所有网络接口。
ListenPort	否	1024-32767	10051	trapper 的监听端口。
LoadModule	否			在 proxy 启动时加载的模块。模块用于扩展 proxy 的功能。 格式： LoadModule=<module.so> LoadModule=<path/module.so> LoadModule=</abs_path/module.so> 模块必须位于 LoadModulePath 指定的目录中，或者路径必须在模块名称之前。如果前面的路径是绝对路径（以’/’ 开头），则忽略 LoadModulePath。 允许包含多个 LoadModule 参数。
LoadModulePath	否			proxy 模块的位置的完整路径。 默认值取决于编译选项。
LogFile	是，如果 LogType 设置为 *file*，否则 否			日志文件的名称。
LogFileSize	否	0-1024	1	日志文件的最大大小（MB）。 0 - 禁用自动日志轮转。 注意：如果日志文件大小达到限制并且文件轮转失败，无论出于何种原因，现有的日志文件将被截断并重新开始。
LogRemoteCommands	否		0	启用将执行的 shell 命令记录为告警。 0 - 禁用 1 - 启用 此参数自 Zabbix 3.4.0 起支持。
LogType	否		file	日志输出类型： *file* - 写入由 LogFile 参数指定的日志文件， *system* - 写入系统日志， *console* - 输出到控制台。 此参数自 Zabbix 3.0.0 起支持。
LogSlowQueries	否	0-3600000	0	数据库查询被记录为慢查询的时间长度（以毫秒为单位）。 0 - 不记录慢查询。 此选项从 DebugLevel=3 开始启用。
PidFile	否		/tmp/zabbix_proxy.pid	PID 文件的名称。
ProxyLocalBuffer	否	0-720	0	proxy 会将数据在本地保留 N 小时，即使数据已经与服务器同步。 如果本地数据将由第三方应用程序使用，可以使用此参数。
ProxyMode	否	0-1	0	proxy 操作模式。 0 - proxy 处于主动模式 1 - proxy 处于被动模式 注意使用主动 proxy 时，敏感的 proxy 配置数据可能会被能够访问 Zabbix server trapper 端口的各方获取。这是因为任何人都可能假装是主动 proxy 并请求配置数据；不进行身份验证。
ProxyOfflineBuffer	否	1-720	1	如果与 Zabbix server 没有连接，proxy 将保留数据 N 小时。 较旧的数据将丢失。

参数	必填	范围	默认值	描述
Server	是			如果 ProxyMode 设置为 主动模式： Zabbix server 的 IP 地址或 DNS 名称（地址：端口）或集群（地址：端口；地址 2：端口）以获取配置数据并发送数据。 如果未指定端口，则使用默认端口。 集群节点必须用分号分隔。 如果 ProxyMode 设置为 被动模式： 逗号分隔的 IP 地址列表，可选地使用 CIDR 表示法，或 Zabbix server 的 DNS 名称。只接受这里列出的地址的传入连接。如果启用了 IPv6 支持，则'127.0.0.1', '::127.0.0.1', '::ffff:127.0.0.1' 被视为等效。 '::/0' 将允许任何 IPv4 或 IPv6 地址。'0.0.0.0/0' 可用于允许任何 IPv4 地址。 示例： Server=127.0.0.1,192.168.1.0/24,::1,2001:db8::/32,zabbix.example.com
SNMPTrapperFile	否		/tmp/zabbix_traps.tmp	用于将数据从 SNMP trap 守护进程传递到 proxy 的临时文件。 必须与 zabbix_trap_receiver.pl 或 SNMPTT 配置文件中的相同。
SocketDir	否		/tmp	存储内部 Zabbix 服务使用的 IPC 套接字的目录。 此参数自 Zabbix 3.4.0 起支持。
SourceIP	否			源 IP 地址用于： - 与 Zabbix server 的出站连接； - 无代理连接（VMware、SSH、JMX、SNMP、Telnet 和简单检查）； - HTTP agent 连接； - 脚本监控项 JavaScript HTTP 请求； - 预处理 JavaScript HTTP 请求； - 连接到 Vault
SSHKeyLocation	否			SSH 检查和操作的公钥和私钥位置
SSLCertLocation	否			SSL 客户端证书文件的位置，用于客户端认证。 此参数仅用于 Web 监控。
SSLKeyLocation	否			SSL 私钥文件的位置，用于客户端认证。 此参数仅用于 Web 监控。
SSLCALocation	否			SSL 服务器证书验证的证书颁发机构（CA）文件的位置。 请注意，此参数的值将被设置为 libcurl 选项 CURLOPT_CAPATH。 对于 7.42.0 之前的 libcurl 版本，这只有在 libcurl 编译为使用 OpenSSL 时才有效。有关更多信息，另请参阅 cURL 网页。 此参数自 Zabbix 2.4.0 起用于 Web 监控，在 Zabbix 3.0.0 起用于 SMTP 认证。
StartDBSyncers	否	1-100	4	history syncers的预派生实例数量。 注意：更改此值时要小心，增加它可能弊大于利。
StartDiscoverers	否	0-250	1	discoverers的预派生实例数量。
StartHistoryPollers	否	0-1000	1	history pollers的预派生实例数量。 此参数自 Zabbix 5.4.0 起支持。
StartHTTPPollers	否	0-1000	1	HTTP pollers的预派生实例数量。
StartIPMIPollers	否	0-1000	0	IPMI pollers的预派生实例数量。
StartJavaPollers	否	0-1000	0	Java pollers的预派生实例数量。
StartODBCPollers	否	0-1000	1	ODBC pollers的预派生实例数量。
StartPingers	否	0-1000	1	ICMP pingers的预派生实例数量。
StartPollersUnreachable	否	0-1000	1	pollers for unreachable的预派生实例数量（包括 IPMI 和 Java）。 如果启动了常规、IPMI 或 Java 轮询进程，则至少必须运行一个无法到达主机的轮询进程。
StartPollers	否	0-1000	5	pollers的预派生实例数量。
StartPreprocessors	否	1-1000	3	预处理workers 的预派生实例数量[1]。 当启动预处理器工作进程时，自动启动预处理管理进程。 此参数自 Zabbix 4.2.0 起支持。
StartSNMPTrapper	否	0-1	0	如果设置为 1，将启动SNMP trapper 进程。
StartTrappers	否	0-1000	5	trappers 的预派生实例数量。 轮询器接受来自 Zabbix sender 和主动 agents 的传入连接。
StartVMwareCollectors	否	0-250	0	VMware collector 实例的预派生数量。

参数	必填	范围	默认值	描述
StatsAllowedIP	否			逗号分隔的 IP 地址列表，可选地使用 CIDR 表示法，或外部 Zabbix 实例的 DNS 名称。只接受这里列出的地址的统计请求。如果此参数未设置，则不接受任何统计请求。 如果启用了 IPv6 支持，则'127.0.0.1', '::127.0.0.1', '::ffff:127.0.0.1' 被视为等效，'::/0' 将允许任何 IPv4 或 IPv6 地址。'0.0.0.0/0' 可用于允许任何 IPv4 地址。 示例：StatsAllowedIP=127.0.0.1,192.168.1.0/24,::1,2001:db8::/32,zabbix.example.com 此参数自 Zabbix 4.2.0 起支持。
Timeout	否	1-30	3	指定我们等待 agent、SNMP 设备或外部检查的时间长度（以秒为单位）。
TLSAccept	被动 proxy 必须，如果定义了 TLS 证书或 PSK 参数（即使是未加密连接），否则不需要			接受来自 Zabbix server 的传入连接类型。用于被动 proxy，主动 proxy 上忽略。可以指定多个值，用逗号分隔： *unencrypted* - 接受未加密的连接（默认） *psk* - 接受使用 TLS 和预共享密钥 (PSK) 的连接 *cert* - 接受使用 TLS 和证书的连接 此参数自 Zabbix 3.0.0 起支持。
TLSCAFile	否			包含用于对等证书验证的顶级 CA 证书的文件的完整路径名，用于 Zabbix 组件之间的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSCertFile	否			包含 proxy 证书或证书链的文件的完整路径名，用于 Zabbix 组件之间的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSCipherAll	否			GnuTLS 优先级字符串或 OpenSSL（TLS 1.2）密码字符串。覆盖基于证书和基于 PSK 的加密的默认密码套件选择标准。 示例： TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_1 此参数自 Zabbix 4.4.7 起支持。
TLSCipherAll13	否			OpenSSL 1.1.1 或更高版本在 TLS 1.3 中的密码字符串。覆盖基于证书和基于 PSK 的加密的默认密码套件选择标准。 GnuTLS 示例：NONE:+VERS-TLS1.2:+ECDHE-RSA:+RSA:+ECDHE-PSK:+PSK:+AES-128-GCM:+AES-128-CBC:+AEAD:+SHA256:+SHA1:+CURVE-ALL:+COMP-NULL::+SIGN-ALL:+CTYPE-X.509 OpenSSL 示例： EECDH+aRSA+AES128:RSA+aRSA+AES128:kECDHEPSK+AES128:kPSK+A 此参数自 Zabbix 4.4.7 起支持。
TLSCipherCert	否			GnuTLS 优先级字符串或 OpenSSL（TLS 1.2）密码字符串。覆盖基于证书的加密的默认密码套件选择标准。 GnuTLS 示例： NONE:+VERS-TLS1.2:+ECDHE-RSA:+RSA:+AES-128-GCM:+AES-128-CBC:+AEAD:+SHA256:+SHA1:+CURVE-ALL:+COMP-NULL:+SIGN-ALL:+CTYPE-X.509 OpenSSL 示例：EECDH+aRSA+AES128:RSA+aRSA+AES128 此参数自 Zabbix 4.4.7 起支持。
TLSCipherCert13	否			OpenSSL 1.1.1 或更高版本在 TLS 1.3 中的密码字符串。覆盖基于证书的加密的默认密码套件选择标准。 此参数自 Zabbix 4.4.7 起支持。
TLSCipherPSK	否			GnuTLS 优先级字符串或 OpenSSL（TLS 1.2）密码字符串。覆盖基于 PSK 的加密的默认密码套件选择标准。 GnuTLS 示例：NONE:+VERS-TLS1.2:+ECDHE-PSK:+PSK:+AES-128-GCM:+AES-128-CBC:+AEAD:+SHA256:+SHA1:+CURVE-ALL:+COMP-NULL:+SIGN-ALL OpenSSL 示例：kECDHEPSK+AES128:kPSK+AES128 此参数自 Zabbix 4.4.7 起支持。
TLSCipherPSK13	否			OpenSSL 1.1.1 或更高版本在 TLS 1.3 中的密码字符串。覆盖基于 PSK 的加密的默认密码套件选择标准。 示例： TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256 此参数自 Zabbix 4.4.7 起支持。

参数	必填	范围	默认值	描述
TLSConnect	主动 proxy 必须，如果定义了 TLS 证书或 PSK 参数（即使是未加密连接），否则不需要			proxy 应该如何连接到 Zabbix server。用于主动 proxy，被动 proxy 上忽略。只能指定一个值： *unencrypted* - 无加密连接（默认） *psk* - 使用 TLS 和预共享密钥 (PSK) 连接 *cert* - 使用 TLS 和证书连接 此参数自 Zabbix 3.0.0 起支持。
TLSCRLFile	否			包含吊销证书的文件的完整路径名。此参数用于 Zabbix 组件之间的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSKeyFile	否			包含 proxy 私钥的文件的完整路径名，用于 Zabbix 组件之间的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSPSKFile	否			包含 proxy 预共享密钥的文件的完整路径名，用于与 Zabbix server 的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSPSKIdentity	否			预共享密钥身份字符串，用于与 Zabbix server 的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSServerCertIssuer	否			允许的服务器证书颁发者。 此参数自 Zabbix 3.0.0 起支持。
TLSServerCertSubject	否			允许的服务器证书主题。 此参数自 Zabbix 3.0.0 起支持。
TmpDir	否		/tmp	临时目录。
TrapperTimeout	否	1-300	300	指定 trapper 处理新数据可能花费的秒数。
User	否		zabbix	将权限降低到系统上特定的、已存在的用户。 仅当以'root' 身份运行且 AllowRoot 被禁用时有效。
UnavailableDelay	否	1-3600	60	在不可用 期间，以秒为单位检查主机可用性的时间间隔。
UnreachableDelay	否	1-3600	15	在不可达 期间，以秒为单位检查主机可用性的时间间隔。
UnreachablePeriod	否	1-3600	45	在多少秒的不可达 后，将主机视为不可用。
VaultDBPath	否			通过键'password' 和'username' 检索数据库凭据的 Vault 路径。 示例：secret/zabbix/database 如果未指定 DBUser 和 DBPassword，则只能使用此选项。 此参数自 Zabbix 5.2.0 起支持。
VaultToken	否			应为 Zabbix proxy 专门生成的 Vault 认证令牌，该令牌仅具有对可选 VaultDBPath 配置参数中指定路径的只读权限。 如果同时定义了 VaultToken 和 VAULT_TOKEN 环境变量，则为错误。 此参数自 Zabbix 5.2.0 起支持。
VaultURL	否		https://127.0.0.1:8200	Vault 服务器 HTTP[S] URL。如果未指定 SSLCALocation，则使用系统范围的 CA 证书目录。 此参数自 Zabbix 5.2.0 起支持。
VMwareCacheSize	否	256K-2G	8M	存储 VMware 数据的共享内存大小。 可以使用 VMware 内部检查 zabbix[vmware,buffer,...] 来监控 VMware 缓存使用情况（见内部检查）。 注意，如果没有配置启动的 vmware collector 实例，则不分配共享内存。
VMwareFrequency	否	10-86400	60	从单个 VMware 服务收集数据之间的延迟，单位为秒。 此延迟应设置为任何 VMware 监控项的最小更新间隔。
VMwarePerfFrequency	否	10-86400	60	从单个 VMware 服务检索性能计数器统计信息之间的延迟，单位为秒。 此延迟应设置为使用 VMware 性能计数器的任何 VMware 监控项的最小更新间隔。
VMwareTimeout	否	1-300	10	VMware collector 从 VMware 服务（vCenter 或 ESX hypervisor）等待响应的最大秒数。

3 Zabbix agent (UNIX)

概览

本节列出了 Zabbix agent 的配置文件（zabbix_agentd.conf）中支持的参数。请注意：

- 默认值反映的是守护进程的默认值，而不是附带的配置文件中的值；
- Zabbix 仅支持 UTF-8 编码的配置文件，且编码中不可使用字节顺序标记（BOM）；
- 仅在行首支持以“#”开始的注释。

参数

参数	必填	范围	默认值	描述
Alias	否			为监控项键设置别名。它可以用来替代长且复杂的监控项键为更短和更简单的键。 可以存在多个 *Alias* 参数。不允许有相同的 *Alias* 键的多个参数。不同的 *Alias* 键可以引用相同的监控项键。 别名可以在 *HostMetadataItem* 中使用，但不能在 *HostnameItem* 参数中使用。 示例： 1. 获取用户'zabbix' 的 ID。 Alias=zabbix.userid:vfs.file.regexp[/etc/passwd,"^zabbix:.:([0-9]+)",,,\1] 现在可以使用简写键 **zabbix.userid** 来获取数据。 2. 使用默认参数和自定义参数获取 CPU 利用率。 Alias=cpu.util:system.cpu.util Alias=cpu.util[*]:system.cpu.util[*] 这允许使用 **cpu.util** 键以默认参数获取 CPU 利用率百分比，也可以使用 **cpu.util[all, idle, avg15]** 来获取有关 CPU 利用率的特定数据。 3. 运行多个低级发现规则处理相同的发现项。 Alias=vfs.fs.discovery[*]:vfs.fs.discovery 现在可以使用不同的参数为每个规则设置多个发现规则，例如，**vfs.fs.discovery[foo]**，**vfs.fs.discovery[bar]** 等。
AllowKey	否			允许执行匹配特定模式的监控项键。键模式是一个支持"*" 字符的通配符表达式，用于匹配任意数量的任意字符。 可以与 DenyKey 结合定义多个键匹配规则。参数将根据它们出现的顺序逐个处理。 此参数自 Zabbix 5.0.0 起支持。 另请参阅：限制 agent 检查。
AllowRoot	否		0	允许 agent 以'root' 身份运行。如果禁用，并且 agent 由'root' 启动，则 agent 将尝试切换到用户'zabbix'。如果以普通用户身份启动，则此选项无效。 0 - 不允许 1 - 允许
BufferSend	否	1-3600	5	不要在缓冲区中保留数据超过 N 秒。
BufferSize	否	2-65535	100	内存缓冲区中值的最大数量。如果缓冲区满了，agent 会将所有收集的数据发送到 Zabbix server 或 proxy。
DebugLevel	否	0-5	3	指定调试级别： 0 - 关于启动和停止 Zabbix 进程的基本信息 1 - 严重信息 2 - 错误信息 3 - 警告 4 - 调试（生成大量信息） 5 - 扩展调试（生成更多信息）
DenyKey	否			拒绝执行与模式匹配的监控项键。键模式是一个支持"*" 字符的通配符表达式，用于匹配任意数量的任意字符。 可以与 AllowKey 结合定义多个键匹配规则。参数将根据它们出现的顺序逐个处理。 此参数自 Zabbix 5.0.0 起支持。 另请参阅：限制 agent 检查。

参数	必填	范围	默认值	描述
EnableRemoteCommands	否		0	是否允许来自 Zabbix server 的远程命令。此参数已弃用，请改用 AllowKey=system.run[*] 或 DenyKey=system.run[*] 它是基于值的 AllowKey/DenyKey 参数的内部别名：0 - DenyKey=system.run[*] 1 - AllowKey=system.run[*]
HostInterface	否	0-255 个字符		定义主机接口的可选参数。 主机接口在主机自动注册过程中使用。 如果值超过 255 个字符的限制，代理将发出错误并且不会启动。 如果没有定义，将从 HostInterfaceItem 获取值。 自 Zabbix 4.4.0 起支持。
HostInterfaceItem	否			定义用于获取主机接口的监控项的可选参数。 主机接口在主机自动注册过程中使用。 在自动注册请求期间，如果指定监控项返回的值超过 255 个字符限制，agent 将记录一条警告消息。 无论 AllowKey/DenyKey 的值如何，都支持*system.run[]*监控项。 仅当未定义 HostInterface 时使用此选项。 自 Zabbix 4.4.0 起支持。
HostMetadata	否	0-255 字符		主动 agent 的自动注册过程中使用的主机元数据的可选参数。 如果未定义，则从 HostMetadataItem 获取值。 如果指定的值超出限制或不是 UTF-8 字符串，agent 将发出错误并不启动。
HostMetadataItem	否			定义一个 *Zabbix agent* 监控项，用于获取主机元数据的可选参数。此选项仅在未定义 HostMetadata 时使用。 支持 UserParameters 和别名。无论 AllowKey/DenyKey 的值如何，都支持*system.run[]*。 HostMetadataItem 的值在每次自动注册尝试时检索，并且仅在主机自动注册过程中使用（主动 agent）。 在自动注册请求期间，如果指定监控项返回的值超过 255 个字符限制，agent 将记录一条警告消息。 监控项返回的值必须是 UTF-8 字符串，否则将被忽略。
Hostname	否		Set by HostnameItem	主机名列表，由逗号分隔，唯一且大小写敏感。 对于主动检查是必需的，并且必须与服务器上配置的主机名匹配。 如果未定义，则从 HostnameItem 获取值。 允许的字符：字母数字、'.'、' '、'_' 和'-'。 最大长度：每个主机名 128 个字符，整行为 2048 个字符。
HostnameItem	否		system.hostname	定义一个 *Zabbix agent* 监控项的可选参数，用于获取主机名。此选项仅在未定义 Hostname 时使用。 不支持 UserParameters 或别名，但无论 AllowKey/DenyKey 的值如何，都支持*system.run[]*。
Include	否			您可以在配置文件中包含单个文件或目录中的所有文件。 要仅包含指定目录中的相关文件，支持使用星号通配符进行模式匹配。例如： `/absolute/path/to/config/files/*.conf`。 请参阅特殊说明了解限制。
ListenBacklog	否	0 - INT_MAX	SOMAXCONN	TCP 队列中挂起连接的最大数量。 默认值是一个硬编码常量，这取决于系统。 支持的最大值也取决于系统，过高的值可能会被静默截断为 “实现指定的最大值”。
ListenIP	否		0.0.0.0	agent 应该监听的逗号分隔的 IP 地址列表。 1.8.3 及更高版本支持多个 IP 地址。
ListenPort	否	1024-32767	10050	agent 将在该端口上监听来自服务器的连接。
LoadModule	否			在 agent 启动时加载的模块。模块用于扩展 agent 的功能。 格式： LoadModule=<module.so> LoadModule=<path/module.so> LoadModule=</abs_path/module.so> 模块必须位于 LoadModulePath 指定的目录中，或者路径必须在模块名称之前。如果前面的路径是绝对路径（以'/' 开头），则忽略 LoadModulePath。 允许包含多个 LoadModule 参数。
LoadModulePath	否			agent 模块所在位置的完整路径。 默认值取决于编译选项。

参数	必填	范围	默认值	描述
LogFile	是，如果 LogType 设置为 *file*，否则否			日志文件的名称。
LogFileSize	否	0-1024	1	日志文件的最大大小（MB）。 0 - 禁用自动日志轮转。 注意：如果日志文件大小达到限制并且文件轮转失败，无论出于何种原因，现有的日志文件将被截断并重新开始。
LogType	否		file	日志输出类型： *file* - 写入由 LogFile 参数指定的日志文件， *system* - 写入系统日志， *console* - 输出到控制台。 此参数自 Zabbix 3.0.0 起支持。
LogRemoteCommands	否		0	启用将执行的 shell 命令记录为警告。 0 - 禁用 1 - 启用 只有远程执行的命令才会被 agent 记录。如果由 HostMetadataItem、HostInterfaceItem 或 HostnameItem 参数本地执行 system.run[]，则不会创建日志监控项。
MaxLinesPerSecond	否	1-1000	20	agent 在处理’log’ 和’eventlog’ 日志监控项主动检查时，每秒钟可以向 Zabbix server 或 proxy 发送的新行的最大数量。 如果在’log’ 或’eventlog’ 日志监控项键中提供了’maxlines’ 参数，则提供的值将被覆盖。 注意: Zabbix 将处理 MaxLinesPerSecond 设置的 10 倍的新行，以便在日志监控项中查找所需字符串。
PidFile	否		/tmp/zabbix_agentd.pid	agent 的进程 ID 文件名称。
RefreshActiveChecks	否	60-3600	120	主动检查列表多久刷新一次，单位为秒。 注意，如果在刷新主动检查失败后，下一次刷新将在 60 秒后尝试。
Server	是，如果 StartAgents 没有明确设置为 0			逗号分隔的 IP 地址列表，可选地使用 CIDR 表示法，或 Zabbix server 和 Zabbix proxy 的 DNS 名称。 只接受这里列出的主机的传入连接。 如果启用了 IPv6 支持，则’127.0.0.1’, ’::127.0.0.1’, ’::ffff:127.0.0.1’ 被视为等效，’::/0’ 将允许任何 IPv4 或 IPv6 地址。 ’0.0.0.0/0’ 可用于允许任何 IPv4 地址。 请注意”IPv4 兼容的 IPv6 地址”（0000::/96 前缀）被 RFC4291 支持但已弃用。 示例： Server=127.0.0.1,192.168.1.0/24,::1,2001:db8::/32,zabbix.example.com 允许使用空格。

参数	必填	范围	默认值	描述
ServerActive	否			用于获取主动检查的 Zabbix server/proxy 地址或集群配置。 Server/proxy 地址是 IP 地址或 DNS 名称，端口可选地由冒号分隔。 集群配置是使用分号分隔的一个或多个服务器地址。 可以指定多个 Zabbix server/集群和 Zabbix proxy，用逗号分隔。 每个 Zabbix server/集群不应指定多个 Zabbix proxy。 如果指定了 Zabbix proxy，则不应为该代理指定 Zabbix server/集群。 可以提供多个逗号分隔的地址，以并行使用几个独立的 Zabbix server。允许使用空格。 如果未指定端口，则使用默认端口。 如果指定了该主机的端口，则必须将 IPv6 地址用方括号括起来。 如果未指定端口，IPv6 地址的方括号是可选的。 如果未指定此参数，则禁用主动检查。 Zabbix proxy 示例： ServerActive=127.0.0.1:10051 多服务器示例： ServerActive=127.0.0.1:20051,zabbix.domain,[::1]:30051,::1,[12fc::1] 高可用性示例： ServerActive=zabbix.cluster.node1;zabbix.cluster.node2:20051;zabbix.clus 两个集群和一台服务器的高可用性示例： ServerActive=zabbix.cluster.node1;zabbix.cluster.node2:20051,zabbix.clus
SourceIP	否			用于以下情况的源 IP 地址： - 出站连接到 Zabbix server 或 Zabbix proxy； - 在执行某些监控项（如 web.page.get, net.tcp.port 等）时建立连接
StartAgents	否	0-100	3	用于处理被动检查的预派生 zabbix_agentd 实例的数量。 如果设置为 0，则禁用被动检查，并且 agent 将不会监听任何 TCP 端口。
Timeout	否	1-30	3	处理时间不超过 Timeout 秒。
TLSAccept	是，如果已定义 TLS 证书或 PSK 参数(即使对于未加密连接)，否则否			接受哪些传入连接。用于被动检查。可以指定多个值，由逗号分隔： *unencrypted* - 接受未加密的连接（默认） *psk* - 接受使用 TLS 和预共享密钥（PSK）的连接 *cert* - 接受使用 TLS 和证书的连接 此参数自 Zabbix 3.0.0 起支持。
TLSCAFile	否			包含用于对等证书验证的顶级 CA 证书的文件的完整路径名，用于 Zabbix 组件之间的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSCertFile	否			包含 agent 证书或证书链的文件的完整路径名，用于与 Zabbix 组件的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSCipherAll	否			GnuTLS 优先级字符串或 OpenSSL（TLS 1.2）密码字符串。覆盖基于证书和基于 PSK 的加密的默认密码套件选择标准。 示例： TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_1 此参数自 Zabbix 4.4.7 起支持。
TLSCipherAll13	否			OpenSSL 1.1.1 或更高版本在 TLS 1.3 中的密码字符串。覆盖基于证书和基于 PSK 的加密的默认密码套件选择标准。 GnuTLS 示例：NONE:+VERS-TLS1.2:+ECDHE-RSA:+RSA:+ECDHE-PSK:+PSK:+AES-128-GCM:+AES-128-CBC:+AEAD:+SHA256:+SHA1:+CURVE-ALL:+COMP-NULL::+SIGN-ALL:+CTYPE-X.509 OpenSSL 示例： EECDH+aRSA+AES128:RSA+aRSA+AES128:kECDHEPSK+AES128:kPSK+A 此参数自 Zabbix 4.4.7 起支持。

参数	必填	范围	默认值	描述
TLSCipherCert	否			GnuTLS 优先级字符串或 OpenSSL（TLS 1.2）密码字符串。覆盖基于证书的加密的默认密码套件选择标准。 GnuTLS 示例： NONE:+VERS-TLS1.2:+ECDHE-RSA:+RSA:+AES-128-GCM:+AES-128-CBC:+AEAD:+SHA256:+SHA1:+CURVE-ALL:+COMP-NULL:+SIGN-ALL:+CTYPE-X.509 OpenSSL 示例：EECDH+aRSA+AES128:RSA+aRSA+AES128 此参数自 Zabbix 4.4.7 起支持。
TLSCipherCert13	否			OpenSSL 1.1.1 或更高版本在 TLS 1.3 中的密码字符串。覆盖基于证书的加密的默认密码套件选择标准。 此参数自 Zabbix 4.4.7 起支持。
TLSCipherPSK	否			GnuTLS 优先级字符串或 OpenSSL（TLS 1.2）密码字符串。覆盖基于 PSK 的加密的默认密码套件选择标准。 GnuTLS 示例：NONE:+VERS-TLS1.2:+ECDHE-PSK:+PSK:+AES-128-GCM:+AES-128-CBC:+AEAD:+SHA256:+SHA1:+CURVE-ALL:+COMP-NULL:+SIGN-ALL OpenSSL 示例：kECDHEPSK+AES128:kPSK+AES128 此参数自 Zabbix 4.4.7 起支持。
TLSCipherPSK13	否			OpenSSL 1.1.1 或更高版本在 TLS 1.3 中的密码字符串。覆盖基于 PSK 的加密的默认密码套件选择标准。 示例： TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256 此参数自 Zabbix 4.4.7 起支持。
TLSConnect	是，如果已定义 TLS 证书或 PSK 参数（即使对于未加密连接），否则否			agent 应如何连接到 Zabbix server 或 proxy。用于主动检查。只能指定一个值： *unencrypted* - 不使用加密连接（默认） *psk* - 使用 TLS 和预共享密钥（PSK）进行连接 *cert* - 使用 TLS 和证书进行连接 此参数自 Zabbix 3.0.0 起支持。
TLSCRLFile	否			包含吊销证书的文件的完整路径名。此参数用于与 Zabbix 组件的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSKeyFile	否			包含 agent 私钥的文件的完整路径名，该私钥用于与 Zabbix 组件的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSPSKFile	否			包含 agent 预共享密钥的文件的完整路径名，该密钥用于与 Zabbix 组件的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSPSKIdentity	否			预共享密钥身份字符串，用于与 Zabbix server 的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSServerCertIssuer	否			允许的 server（proxy）证书颁发者。 此参数自 Zabbix 3.0.0 起支持。
TLSServerCertSubject	否			允许的 server（proxy）证书主题。 此参数自 Zabbix 3.0.0 起支持。
UnsafeUserParameters	否	0,1	0	允许在用户定义的参数中传递所有字符。 0 - 不允许 1 - 允许 以下字符不允许：\ ' " ` * ? [ ] { } ~ $! & ; ( ) < > | # @ 此外，不允许换行字符。
User	否		zabbix	将权限降低到系统上特定的、已存在的用户。 仅当以'root' 身份运行且 AllowRoot 被禁用时有效。
UserParameter	否			用于监控的用户定义的参数。可以有多个用户定义的参数。 格式：UserParameter=<key>,<shell command> 注意，shell 命令不能仅返回空字符串或 EOL。 如果指定了 UserParameterDir 参数，agent 命令可以有相对路径。 示例： UserParameter=system.test,who|wc -l UserParameter=check_cpu,./custom_script.sh

参数	必填	范围	默认值	描述
UserParameterDir	否			UserParameter 命令的默认搜索路径。如果使用，agent 在执行命令前将工作目录更改为此处指定的目录。这样，UserParameter 命令可以有相对的 ./ 前缀而不是完整路径。 只允许一个条目。 示例：UserParameterDir=/opt/myscripts

参阅

1. 自 2.0.0 版本开始，Zabbix agent 主动和被动检查的配置差异

4 Zabbix agent 2 (UNIX)

概述

Zabbix agent 2 是新一代的 Zabbix agent 并用于替代之前的 Zabbix agent。

本节列出了 Zabbix agent 2 配置文件 (zabbix_agent2.conf) 中支持的参数。注意:

- 默认值反映的是进程默认值，而不是出厂配置文件中的值；
- Zabbix 仅支持无 BOM 的 UTF-8 编码配置文件；
- 支持注释须以”#” 为行首。

参数

参数	是否必须	范围	默认值	描述
Alias	否			设置监控项键的别名。可以用来替代长且复杂的监控项键，使用更简短和简单的别名。 可以存在多个 *Alias* 参数。不允许有相同 *Alias* 键的多个参数。 不同的 *Alias* 键可以引用同一个监控项键。 别名可以在 *Host-MetadataItem* 中使用，但不能在 *HostnameItem* 参数中使用。 示例： 1. 检索用户'zabbix' 的 ID。 Alias=zabbix.userid:vfs.file.rego 9]+)"„\1] 现在可以使用简写键 **zabbix.userid** 来检索数据。 2. 使用默认和自定义参数获取 CPU 利用率。 Alias=cpu.util:system.cpu.util Alias=cpu.util[*]:system.cpu.ut 这允许使用 **cpu.util** 键以默认参数获取 CPU 利用率百分比，也可以使用 **cpu.util[all, idle, avg15]** 来获取关于 CPU 利用率的特定数据。 3. 运行多个低级别自动发现 规则处理相同的发现项。 Alias=vfs.fs.discovery[*]:vfs.fs. 现在可以使用 **vfs.fs.discovery** 设置多个发现规则，每个规则使用不同的参数，例如 **vfs.fs.discovery[foo]**，**vfs.fs.discovery[bar]** 等。

参数	是否必须	范围	默认值	描述
AllowKey	否			允许执行匹配模式的监控项键。键模式是一个通配符表达式，支持”*” 字符匹配任意数量的任意字符。 可以与 DenyKey 结合定义多个键匹配规则。参数将根据它们出现的顺序逐个处理。 此参数自 Zabbix 5.0.0 起支持。 另请参阅：限制 agent 检查。
BufferSend	否	1-3600	5	确定缓冲区向 Zabbix server 发送数据的时间间隔，单位为秒。 请注意，如果缓冲区已满，数据将更早发送。
BufferSize	否	2-65535	100	内存缓冲区中值的最大数量。如果缓冲区满了，agent 将把所有收集的数据发送到 Zabbix server 或 proxy。 如果持久缓冲区被禁用（*EnablePersistentBuffer=0*），则应仅使用此参数。
ControlSocket	否		\\.\pipe\agent.sock	控制套接字，用于使用’-R’ 选项发送运行时命令。
DebugLevel	否	0-5	3	指定调试级别： 0 - 关于启动和停止 Zabbix 进程的基本信息 1 - 严重信息 2 - 错误信息 3 - 警告 4 - 调试（产生大量信息） 5 - 扩展调试（产生更多信息）

参数	是否必须	范围	默认值	描述
DenyKey	否			拒绝执行匹配模式的监控项键。 键模式是一个通配符表达式，支持”*” 字符匹配任意数量的任意字符。 可以与 AllowKey 结合定义多个键匹配规则。参数将根据它们出现的顺序逐个处理。 此参数自 Zabbix 5.0.0 起支持。 另请参阅：限制 agent 检查。
EnablePersistentBuffer	否	0-1	0	启用对主动监控项使用本地持久存储。 0 - 已禁用 1 - 已启用 如果持久存储被禁用，则将使用内存缓冲区。
ForceActiveChecksOnStart	否	0-1	0	在重启后立即对接收到的第一个配置执行主动检查。 0 - 已禁用 1 - 已启用 也可作为每个插件配置参数使用，例如： Plugins.Uptime.System.Fo 自 Zabbix 6.0.2 起支持。
HostInterface	否	0-255 字符		定义主机接口的可选参数。 主机接口在主机自动注册过程中使用。 如果值超过 255 字符的限制，agent 将发出错误并不启动。 如果没有定义，值将从 HostInterfaceItem 获取。 自 Zabbix 4.4.0 起支持。

参数	是否必须	范围	默认值	描述
HostInterfaceItem	否			定义用于获取主机接口的监控项的可选参数。 主机接口在主机自动注册过程中使用。 在自动注册请求期间，如果指定的监控项返回的值超过 255 字符的限制，agent 将记录一条警告消息。 *system.run[]* 监控项无论 AllowKey/DenyKey 值如何都受支持。 如果 HostInterface 未定义，则仅使用此选项。 自 Zabbix 4.4.0 起支持。
HostMetadata	否	0-255 字符		定义主机元数据的可选参数。主机元数据用于主机自动注册过程。 如果指定的值超出限制或不是 UTF-8 字符串，agent 将发出错误并不启动。 如果没有定义，将从 HostMetadataItem 获取该值。

参数	是否必须	范围	默认值	描述
HostMetadataItem	否			定义用于获取主机元数据的监控项的可选参数。 每次自动注册尝试时，都会检索主机元数据监控项的值。 在自动注册请求期间，如果指定的监控项返回的值超过 255 字符的限制，agent 将记录一条警告消息。 此选项仅在未定义 HostMetadata 时使用。支持 UserParameters 和别名。无论 *EnableRemoteCommands* 的值如何，都支持*system.run[]*。 监控项返回的值必须是 UTF-8 字符串，否则将被忽略。
Hostname	否		由 HostnameItem 设置	用逗号分隔的唯一、区分大小写的主机名列表。 主动检查需要这些主机名，并必须与服务器上配置的主机名匹配。 如果未定义，则从 HostnameItem 获取值。 允许的字符：字母数字、'.'、' '、'_' 和'-'。 最大长度：每个主机名 128 个字符，整行 2048 个字符。
HostnameItem	否		system.hostname	如果未定义 Hostname，则用于生成 Hostname 的监控项。如果已定义 Hostname，则忽略。 不支持 UserParameters 或别名，但如果未定义 *EnableRemoteCommands*，则支持*system.run[]*。

参数	是否必须	范围	默认值	描述
Include	否			您可以在配置文件中包含单个文件或目录中的所有文件（如果使用 Windows MSI 安装包安装 Zabbix agent，则默认位于 `C:\Program Files\Zabbix Agent 2`；如果以 zip 归档的方式安装 Zabbix agent，则位于安装期间指定的文件夹）。所有包含的文件必须具有正确的语法，否则 agent 将不启动。 路径可以是相对于 *zabbix_agent2.conf* 文件位置的（例如，`Include=.\zabbix_agent2`） 为了仅包含指定目录中的相关文件，支持使用星号通配符字符进行模式匹配。例如：`C:\Program Files\Zabbix Agent2\zabbix_agent2.d\*` 有关限制，请参见特别说明。
ListenIP	否		0.0.0.0	逗号分隔的 IP 地址列表，agent 应该侦听这些地址。 如果连接到 Zabbix server，第一个 IP 地址将被发送到它，以检索主动检查的列表。
ListenPort	否	1024-32767	10050	agent 将在此端口上侦听来自服务器的连接。
LogFile	是，如果 LogType 设置为 *file*，否则 否		c:\zabbix_agent2.log	如果 LogType 是'file'，日志文件的名称。

参数	是否必须	范围	默认值	描述
LogFileSize	否	0-1024	1	日志文件的最大大小（MB）。 0 - 禁用自动日志轮转。 注意：如果达到日志文件大小限制，并且由于任何原因日志轮转失败，则现有的日志文件将被截断并重新启动。
LogType	否		file	指定日志消息写入的位置： *file* - 由 LogFile 参数指定的文件， *console* - 标准输出。
PersistentBufferFile	否			存储 Zabbix Agent2 SQLite 数据库的文件。必须是完整文件名。 只有启用持久缓冲区（*EnablePersistentBuffer=1*）时才使用此参数。
PersistentBufferPeriod	否	1m-365d	1h	当没有连接到 server 或 proxy 时，数据应该存储的时间周期。旧数据将丢失。日志数据将被保留。 只有启用持久缓冲区（*EnablePersistentBuffer=1*）时才使用此参数。
Plugins	否			自 Zabbix 6.0.0 起，大多数插件都有自己的配置文件。agent 配置文件包含下面列出的插件参数。

参数	是否必须	范围	默认值	描述
Plugins.Log.MaxLinesPerSecond	否	1-1000	20	agent 在处理'log'和'eventlog' 主动检查时，每秒钟发送到 Zabbix server 或 proxy 的新行的最大数量。 如果'log'或'eventlog' 监控项键中提供了'maxlines' 参数，将覆盖提供的值。 注意：Zabbix 将处理比 *MaxLinesPerSecond* 设置多 10 倍的新行，以在日志项中查找所需字符串。 此参数自 4.4.2 起支持，并替代了 MaxLinesPerSecond。
Plugins.SystemRun.LogRemoteCommands	否		0	启用将执行的 shell 命令记录为警告。 0 - 禁用 1 - 启用 仅当远程执行时，命令将被记录。如果 system.run[] 由 HostMetadataItem、HostInterfaceItem 或 HostnameItem 参数本地启动，则不会创建日志条目。 此参数自 4.4.2 起支持，并替代了 LogRemoteCommands。
PluginSocket	否		\\.\pipe\agent.plugin.sock	用于可加载插件通信的 unix 套接字路径。
PluginTimeout	否	1-30	全局超时	与可加载插件连接的超时时间。
RefreshActiveChecks	否	60-3600	120	主动检查列表刷新的频率，以秒为单位。 注意，在刷新主动检查失败后，下一次刷新将在 60 秒后尝试。

参数	是否必须	范围	默认值	描述
Server	是			逗号分隔的 IP 地址列表，可选地使用 CIDR 表示法，或 Zabbix server 和 Zabbix proxy 的 DNS 名称。 仅接受此处列出的主机的传入连接。 如果启用了 IPv6 支持， 则'127.0.0.1'， '::ffff:127.0.0.1' 被视为相等， '::/0' 将允许任何 IPv4 或 IPv6 地址。 '0.0.0.0/0' 可用于允许任何 IPv4 地址。 示例： Server=127.0.0.1,192.168.1.0/ 允许空格。

参数	是否必须	范围	默认值	描述
ServerActive	否			用于获取主动检查的 Zabbix server/proxy 地址或集群配置。 Server/proxy 地址是 IP 地址或 DNS 名称和可选的端口号，由冒号分隔。 集群配置是由分号分隔的一个或多个 server 地址。可以指定多个 Zabbix server/cluster 和 Zabbix proxy，由逗号分隔。每个 Zabbix server/cluster 不应指定多个 Zabbix proxy。如果指定了 Zabbix proxy，则不应为该 proxy 指定 Zabbix server/cluster。 可以提供多个地址以并行使用几个独立的 Zabbix server。允许空格。 如果未指定端口，则使用默认端口。 如果为该主机指定了端口，则必须将 IPv6 地址用方括号括起来。 如果未指定端口，则 IPv6 地址的方括号是可选的。 如果未指定此参数，则禁用主动检查。 Zabbix proxy 示例： ServerActive=127.0.0.1:10051 多 server 示例： ServerActive=127.0.0.1:20051, 高可用性示例： ServerActive=zabbix.cluster.no 两个集群和一台 server 的高可用性示例： ServerActive=zabbix.cluster.no

参数	是否必须	范围	默认值	描述
SourceIP	否			源 IP 地址用于： - 到 Zabbix server 或 Zabbix proxy 的出站连接； - 在执行一些监控项时建立连接（例如 web.page.get, net.tcp.port 等）。
StatusPort	否	1024-32767		如果设置，agent 将在此端口上侦听 HTTP 状态请求（http://localhost:<port>/status
Timeout	否	1-30	3	在处理上花费的时间不超过 Timeout 秒。
TLSAccept	是，如果定义了 TLS 证书或 PSK 参数（即使是 未加密连接），否则否		指定接受哪些传入连接。用于被动检查。可以指定多个值，用逗号分隔： *unencrypted* - 接受未加密的连接（默认） *psk* - 接受使用 TLS 和预共享密钥 (PSK) 的连接 *cert* - 接受使用 TLS 和证书的连接	
TLSCAFile	否			包含顶层 CA 证书的文件的完整路径名，用于 Zabbix 组件之间的加密通信中的对等证书验证。
TLSCertFile	否			包含 agent 证书或证书链的文件的完整路径名，用于与 Zabbix 组件的加密通信。
TLSConnect	是，如果定义了 TLS 证书或 PSK 参数（即使是 未加密连接），否则否		agent 应如何连接到 Zabbix server 或 proxy。用于主动检查。只能指定一个值： *unencrypted* - 不使用加密连接（默认） *psk* - 使用 TLS 和预共享密钥 (PSK) 连接 *cert* - 使用 TLS 和证书连接	
TLSCRLFile	否			包含吊销证书的文件的完整路径名。此参数用于与 Zabbix 组件的加密通信。
TLSKeyFile	否			包含 agent 私钥的文件的完整路径名，用于与 Zabbix 组件的加密通信。

参数	是否必须	范围	默认值	描述
TLSPSKFile	否			包含 agent 预共享密钥的文件的完整路径名，用于与 Zabbix 组件的加密通信。
TLSPSKIdentity	否			预共享密钥身份字符串，用于与 Zabbix server 的加密通信。
TLSServerCertIssuer	否			允许的服务器（proxy）证书颁发者。
TLSServerCertSubject	否			允许的服务器（proxy）证书主题。
UnsafeUserParameters	否	0,1	0	允许将所有字符作为参数传递给用户定义的参数。 不允许以下字符： \ ' " ` * ? [ ] { } ~ $! & ; ( ) < > | # @ 此外，不允许新行字符。
UserParameter	否			用于监控的用户定义参数。可以有多个用户定义的参数。 格式： UserParameter=<key>,<shell command> 注意，shell 命令不能仅返回空字符串或 EOL。 如果指定了 UserParameterDir 参数，shell 命令可以有相对路径。 示例： UserParameter=system.test,wh -l UserParameter=check_cpu,./cu
UserParameterDir	否			UserParameter 命令的默认搜索路径。如果使用，agent 将执行命令前将其工作目录更改为此处指定的目录。这样，UserParameter 命令可以有一个相对的 ./ 前缀而不是完整路径。 只允许一个条目。 示例： UserParameterDir=/opt/myscripts

5 Zabbix agent (Windows)

概述

本节列出了 Zabbix agent 配置文件（zabbix_agentd.conf）支持的参数。

注意：

- 默认值反映了守护进程的默认设置，而不是配置文件中的值；
- Zabbix 仅支持 UTF-8 编码的配置文件，不支持 BOM；
- 仅支持以“#”开头的注释在行的开始处。

参数

参数	必填	范围	默认值	描述
Alias	否			为监控项键值设置别名。可以使用更简短和简单的别名替代长且复杂的监控项键值。 可以存在多个 *Alias* 参数。不允许有相同的 *Alias* 键的多个参数。 不同的 *Alias* 键可以引用相同的监控项键值。 别名可以在 *HostMetadataItem* 中使用，但不能在 *HostnameItem* 或 *PerfCounter* 参数中使用。 示例： 1. 从服务器检索分页文件使用率的百分比。 Alias=pg_usage:perf_counter[\Paging File(_Total)% Usage] 现在可以使用简写键 **pg_usage** 来检索数据。 2. 使用默认和自定义参数获取 CPU 负载。 Alias=cpu.load:system.cpu.load Alias=cpu.load[*]:system.cpu.load[*] 这允许使用 **cpu.load** 键以默认参数获取 CPU 利用率百分比，也可以使用 **cpu.load[percpu,avg15]** 来获取特定的 CPU 负载数据。 3. 运行多个低级发现 规则处理相同的发现项。 Alias=vfs.fs.discovery[*]:vfs.fs.discovery 现在可以使用 **vfs.fs.discovery** 设置多个发现规则，每个规则使用不同的参数，例如 **vfs.fs.discovery[foo]**，**vfs.fs.discovery[bar]** 等。
AllowKey	否			允许执行匹配特定模式的监控项键。键模式是一个支持”*” 字符的通配符表达式，用于匹配任意数量的任意字符。 可以与 DenyKey 结合定义多个键匹配规则。参数将根据它们出现的顺序逐个处理。 此参数自 Zabbix 5.0.0 起支持。 另请参阅：限制 agent 检查。
BufferSend	否	1-3600	5	不要在缓冲区中保留数据超过 N 秒。
BufferSize	否	2-65535	100	内存缓冲区中值的最大数量。如果缓冲区满了，agent 会将所有收集的数据发送到 Zabbix server 或 proxy。
DebugLevel	否	0-5	3	指定调试级别： 0 - 关于启动和停止 Zabbix 进程的基本信息 1 - 严重信息 2 - 错误信息 3 - 警告 4 - 调试（生成大量信息） 5 - 扩展调试（生成更多信息）
DenyKey	否			拒绝执行与模式匹配的监控项键。键模式是一个支持”*” 字符的通配符表达式，用于匹配任意数量的任意字符。 可以与 AllowKey 结合定义多个键匹配规则。参数将根据它们出现的顺序逐个处理。 此参数自 Zabbix 5.0.0 起支持。 另请参阅：限制 agent 检查。
EnableRemoteCommands	否		0	是否允许来自 Zabbix server 的远程命令。此参数已 弃用，请使用 AllowKey=system.run[*] 或 DenyKey=system.run[*] 代替 它是基于值的 AllowKey/DenyKey 参数的内部别名：0 - DenyKey=system.run[*] 1 - AllowKey=system.run[*]。

参数	必填	范围	默认值	描述
HostInterface	否	0-255 字符		定义主机接口的可选参数。 主机接口在主机自动注册过程中使用。 如果值超过 255 字符的限制，agent 将发出错误并不启动。 如果未定义，将从 HostInterfaceItem 获取值。 自 Zabbix 4.4.0 起支持。
HostInterfaceItem	否			定义用于获取主机接口的可选参数。 主机接口在主机自动注册过程中使用。 在自动注册请求期间，如果指定的项目返回的值超过 255 个字符的限制，agent 将记录一条警告消息。 无论 AllowKey/DenyKey 的值如何，都支持*system.run[]* 项目。 如果未定义 HostInterface，则仅使用此选项。 自 Zabbix 4.4.0 起支持。
HostMetadata	否	0-255 字符		主动 agent 的自动注册过程中使用的主机元数据的可选参数。 如果未定义，则从 HostMetadataItem 获取值。 如果指定的值超出限制或不是 UTF-8 字符串，agent 将发出错误并不启动。
HostMetadataItem	否			定义一个用于获取主机元数据的 Zabbix agent 监控项的可选参数。此选项仅在未定义 HostMetadata 时使用。 支持 UserParameters、性能计数器和别名。无论 EnableRemoteCommands 的值如何，都支持*system.run[]*。 每次自动注册尝试时都会检索 HostMetadataItem 的值，并且仅在主动 agent 的自动注册过程中使用。 在自动注册请求期间，如果指定的项目返回的值超过 255 个字符的限制，agent 将记录一条警告消息。 项目返回的值必须是 UTF-8 字符串，否则将被忽略。
Hostname	否		Set by HostnameItem	主机名列表，由逗号分隔，唯一且大小写敏感。 对于主动检查是必需的，并且必须与服务器上配置的主机名匹配。 如果未定义，则从 HostnameItem 获取值。 允许的字符：字母数字、'.'、' '、'_' 和'-'。 最大长度：每个主机名 128 个字符，整行为 2048 个字符。
HostnameItem	否		system.hostname	定义一个 *Zabbix agent* 监控项，用于获取主机名的可选参数。当 Hostname 未定义时，此选项才会使用。 不支持 UserParameters、性能计数器或别名，但无论 *EnableRemoteCommands* 的值如何，都支持*system.run[]*。 另请参阅更详细的描述。
Include	否			您可以在配置文件中包含单个文件或目录中的所有文件（默认位于 `C:\Program Files\Zabbix Agent`，如果使用 Windows MSI 安装包安装 Zabbix agent；位于安装时指定的文件夹中，如果 Zabbix agent 作为 zip 归档安装）。所有包含的文件必须有正确的语法，否则 agent 将无法启动。 要仅在指定目录中包含相关文件，支持使用星号通配符进行模式匹配。例如：`C:\Program Files\Zabbix Agent\zabbix_agentd.d\*.conf`。 另请参阅特殊说明 关于限制。
ListenBacklog	否	0 - INT_MAX	SOMAXCONN	TCP 队列中挂起连接的最大数量。 默认值是一个硬编码常量，这取决于系统。 支持的最大值也取决于系统，过高的值可能会被静默截断为 “实现指定的最大值”。
ListenIP	否		0.0.0.0	agent 应该监听的由逗号分隔的 IP 地址列表。
ListenPort	否	1024-32767	10050	agent 将在该端口上监听来自服务器的连接。
LogFile	是，如果 LogType 设置为 *file*，否则 否	C:\zabbix_agentd.log		agent 日志文件的名称。如果 LogType 设置为文件类型，则必须指定；否则，可以不指定。
LogFileSize	否	0-1024	1	日志文件的最大大小（MB）。 0 - 禁用自动日志轮转。 注意：如果日志文件大小达到限制并且文件轮转失败，无论出于何种原因，现有的日志文件将被截断并重新开始。

参数	必填	范围	默认值	描述
LogType	否		file	日志输出类型： *file* - 将日志写入由 LogFile 参数指定的文件， *system* - 将日志写入 Windows 事件日志， *console* - 将日志写入标准输出。 此参数自 Zabbix 3.0.0 起支持。
LogRemoteCommands	否		0	启用将执行的 shell 命令记录为警告。 0 - 禁用 1 - 启用
MaxLinesPerSecond	否	1-1000	20	agent 每秒向 Zabbix server 或 proxy 发送的新行的最大数量，用于处理'log', 'logrt' 和'eventlog' 主动检查。 如果监控项键值中提供了'maxlines' 参数，则提供的值将被覆盖。 注意：Zabbix 将处理比 *MaxLinesPerSecond* 设置多 10 倍的新行，以在日志监控项中查找所需字符串。
PerfCounter	否			定义一个新参数 <parameter_name>，该参数是系统性能计数器 <perf_counter_path> 在指定时间段 <period>（秒）内的平均值。 语法： <parameter_name>,"<perf_counter_path>",<period> 例如，如果您希望获取上一分钟内处理器中断每秒的平均数量，您可以按以下方式定义一个新参数"interrupts"： PerfCounter = interrupts,"\Processor(0)\Interrupts/sec",60 请注意性能计数器路径周围的双引号。 参数名称（interrupts）将用作创建监控项时的监控项键。 将每秒采集样本以计算平均值。 您可以运行"typeperf -qx" 来获取 Windows 中所有可用的性能计数器列表。
PerfCounterEn	否			定义一个新参数 <parameter_name>，该参数是系统性能计数器 <perf_counter_path> 在指定时间段 <period>（秒）内的平均值。 语法： <parameter_name>,"<perf_counter_path>",<period> 与 PerfCounter 相比，性能计数器路径必须使用英文。 仅支持 Windows Server 2008/Vista 及以上版本。 例如，如果您希望获取上一分钟内处理器中断每秒的平均数量，您可以按以下方式定义一个新参数"interrupts"： PerfCounterEn = interrupts,"\Processor(0)\Interrupts/sec",60 请注意性能计数器路径周围的双引号。 参数名称（interrupts）将用作创建监控项时的监控项键。 将每秒采集样本以计算平均值。 您可以通过查看以下注册表项来获取英文字符串列表： HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib\009。 此参数自 Zabbix 4.0.13 和 4.2.7 起支持。
RefreshActiveChecks	否	60-3600	120	主动检查列表多久刷新一次，单位为秒。 注意，如果在刷新主动检查失败后，下一次刷新将在 60 秒后尝试。
Server	如果 StartAgents 没有明确设置为 0			Zabbix server 的 IP 地址列表，以逗号分隔，可选地使用 CIDR 表示法，或 DNS 名称。 只接受此处列出的主机的传入连接。 如果启用了 IPv6 支持，则'127.0.0.1', '::127.0.0.1', '::ffff:127.0.0.1' 将被视为等同，'::/0' 将允许任何 IPv4 或 IPv6 地址。 '0.0.0.0/0' 可以用来允许任何 IPv4 地址。 注意"IPv4 兼容的 IPv6 地址"（0000::/96 前缀）是支持的，但是已经被RFC4291弃用。 示例： Server=127.0.0.1,192.168.1.0/24,::1,2001:db8::/32,zabbix.example.com 允许使用空格。

参数	必填	范围	默认值	描述
ServerActive	否	(*)		Zabbix server/proxy 地址或集群配置，用于获取主动检查。 Server/proxy 地址是 IP 地址或 DNS 名称，端口号可选，由冒号分隔。 集群配置是一个或多个 server 地址，由分号分隔。 可以指定多个 Zabbix servers/clusters 和 Zabbix proxies，由逗号分隔。 每个 Zabbix server/cluster 不应指定多个 Zabbix proxy。 如果指定了 Zabbix proxy，则不应为该 proxy 指定 Zabbix server/cluster。 可以提供多个逗号分隔的地址，以并行使用几个独立的 Zabbix servers。空格是允许的。 如果未指定端口，则使用默认端口。 如果为该主机指定了端口，则必须将 IPv6 地址用方括号括起来。 如果未指定端口，则 IPv6 地址的方括号是可选的。 如果未指定此参数，则禁用主动检查。 Zabbix 代理示例： ServerActive=127.0.0.1:10051 多服务器示例： ServerActive=127.0.0.1:20051,zabbix.domain,[::1]:30051,::1,[12fc::1] 高可用性示例： ServerActive=zabbix.cluster.node1;zabbix.cluster.node2:20051;zabbix.clus 两个集群和一台服务器的高可用性示例： ServerActive=zabbix.cluster.node1;zabbix.cluster.node2:20051,zabbix.clus
SourceIP	否			用于以下情况的源 IP 地址： - 出站连接到 Zabbix server 或 Zabbix proxy； - 在执行某些监控项（如 web.page.get, net.tcp.port 等）时建立连接
StartAgents	否	0-63 (*)	3	处理被动检查的预先派生的 zabbix_agentd 实例的数量。 如果设置为 0，则禁用被动检查，agent 将不会监听任何 TCP 端口。
Timeout	否	1-30	3	处理时间不超过 Timeout 秒。
TLSAccept	是，如果已定义 TLS 证书或 PSK 参数(即使对于未加密连接)，否则否			接受哪些传入连接。用于被动检查。可以指定多个值，由逗号分隔： *unencrypted* - 接受未加密的连接（默认） *psk* - 接受使用 TLS 和预共享密钥（PSK）的连接 *cert* - 接受使用 TLS 和证书的连接 此参数自 Zabbix 3.0.0 起支持。
TLSCAFile	否			包含用于对等证书验证的顶级 CA 证书的文件的完整路径名，用于 Zabbix 组件之间的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSCertFile	否			包含 agent 证书或证书链的文件的完整路径名，用于与 Zabbix 组件的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSConnect	是，如果已定义 TLS 证书或 PSK 参数(即使对于未加密连接)，否则否			agent 应如何连接到 Zabbix server 或 proxy。用于主动检查。只能指定一个值： *unencrypted* - 不使用加密连接（默认） *psk* - 使用 TLS 和预共享密钥（PSK）进行连接 *cert* - 使用 TLS 和证书进行连接 此参数自 Zabbix 3.0.0 起支持。
TLSCRLFile	否			包含吊销证书的文件的完整路径名。此参数用于与 Zabbix 组件的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSKeyFile	否			包含 agent 私钥的文件的完整路径名，该私钥用于与 Zabbix 组件的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSPSKFile	否			包含 agent 预共享密钥的文件的完整路径名，该密钥用于与 Zabbix 组件的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSPSKIdentity	否			预共享密钥身份字符串，用于与 Zabbix server 的加密通信。 此参数自 Zabbix 3.0.0 起支持。
TLSServerCertIssuer	否			允许的 server（proxy）证书颁发者。 此参数自 Zabbix 3.0.0 起支持。

参数	必填	范围	默认值	描述
TLSServerCertSubject	否			允许的 server（proxy）证书主题。 此参数自 Zabbix 3.0.0 起支持。
UnsafeUserParameters	否	0-1	0	允许所有字符作为参数传递给用户定义的参数。 0 - 不允许 1 - 允许 以下字符不允许： \ ' " ` * ? [ ] { } ~ $! & ; ( ) > | # @ 此外，不允许新行字符。
UserParameter	否			用于监控的用户定义的参数。可以有多个用户定义的参数。 格式：UserParameter=<key>,<shell command> 注意，shell 命令不能仅返回空字符串或 EOL。 如果指定了 UserParameterDir 参数，agent 命令可以有相对路径。 示例： UserParameter=system.test,who|wc -l UserParameter=check_cpu,./custom_script.sh
UserParameterDir	否			UserParameter 命令的默认搜索路径。如果使用，agent 在执行命令前将工作目录更改为此处指定的目录。这样，UserParameter 命令可以有相对的 ./ 前缀而不是完整路径。 只允许一个条目。 示例：UserParameterDir=/opt/myscripts

Note:
(*) ServerActive 中列出的活跃服务器数量加上 StartAgents 中指定的用于被动检查的预先派生实例的数量必须少于 64。

6 Zabbix agent 2 （Windows 系统）

概览

Zabbix agent 2 是新一代的 Zabbix agent，可以用作 Zabbix agent 的替代品。

本节列出了 Zabbix agent 2 配置文件（zabbix_agent2.conf）中支持的参数。

请注意：

- 默认值反映了进程默认值，而不是配置文件中的值；
- Zabbix 仅支持 UTF-8 编码的配置文件，不包含 BOM；
- 仅支持以"#" 开头的注释在行的开始处。

参数

参数	必选	范围	默认值	描述
Alias	否			为监控项键值设置别名。它可以用来替代长且复杂的监控项键值为更小且简单的一个。 可以存在多个 别名参数。不允许有相同 别名键值的多个参数。 不同的 *Alias* 键值可以引用相同的监控项键值。 别名可以在 *HostMetadataItem* 中使用，但不可以在 *HostnameItem* 参数中使用。 示例： 1. 检索用户'zabbix' 的 ID。 Alias=zabbix.userid:vfs.file.regexp[/etc/passwd,"^zabbix:.:([0-9]+)"„„\1] 现在可以使用简写键值 **zabbix.userid** 来检索数据。 2. 使用默认和自定义参数获取 CPU 利用率。 Alias=cpu.util:system.cpu.util Alias=cpu.util[*]:system.cpu.util[*] 这允许使用 **cpu.util** 键值以默认参数获取 CPU 利用率百分比，以及使用 **cpu.util[all, idle, avg15]** 获取关于 CPU 利用率的特定数据。 3. 运行多个低级发现 规则处理相同的发现监控项。 Alias=vfs.fs.discovery[*]:vfs.fs.discovery 现在可以使用 **vfs.fs.discovery** 设置多个发现规则，每个规则使用不同的参数，例如，**vfs.fs.discovery[foo]**，**vfs.fs.discovery[bar]** 等。
AllowKey	否			允许执行匹配模式的监控项键值。键值模式是支持"*" 字符的通配符表达式，用于匹配任意数量的任意字符。 可以定义多个键值匹配规则，与 DenyKey 结合使用。参数将根据它们出现的顺序逐个处理。 此参数自 Zabbix 5.0.0 起支持。 另请参阅：限制 agent 检查。
BufferSend	否	1-3600	5	单位为秒的时间间隔，决定从缓冲区向 Zabbix server 发送值的频率。 请注意，如果缓冲区已满，则数据将更早发送。
BufferSize	否	2-65535	100	内存缓冲区中的最大值数量。如果缓冲区已满，agent 将把所有收集到的数据发送给 Zabbix server 或 proxy。 如果禁用了持久缓冲区（*EnablePersistentBuffer=0*），则应仅使用此参数。
ControlSocket	否		\.\pipe\agent.sock	控制套接字，用于使用'-R' 选项发送运行时命令。
DebugLevel	否	0-5	3	指定调试级别： 0 - 关于启动和停止 Zabbix 进程的基本信息 1 - 严重信息 2 - 错误信息 3 - 警告 4 - 调试（产生大量信息） 5 - 扩展调试（产生更多信息）
DenyKey	否			拒绝执行匹配模式的监控项键值。键值模式是支持"*" 字符的通配符表达式，用于匹配任意数量的任意字符。 可以定义多个键值匹配规则，与 AllowKey 结合使用。参数将根据它们出现的顺序逐个处理。 此参数自 Zabbix 5.0.0 起支持。 另请参阅：限制 agent 检查。
EnablePersistentBuffer	否	0-1	0	启用主动监控项本地持久存储。 0 - 禁用 1 - 启用 如果禁用了持久存储，则将使用内存缓冲区。

参数	必选	范围	默认值	描述
ForceActiveChecksOnStart	否	0-1	0	在重启后立即对接收到的首个配置执行主动检查。 0 - 禁用 1 - 启用 也可作为每个插件配置参数使用，例如：`Plugins.Uptime.System.ForceActiveChecksOnStart=1` 自 Zabbix 6.0.2 起支持。
HostInterface	否	0-255 字符		定义主机接口的可选参数。 主机接口在主机自动注册过程中使用。 如果超过 255 字符限制，agent 将发出错误并不启动。 如果未定义，则从 HostInterfaceItem 获取值。 自 Zabbix 4.4.0 起支持。
HostInterfaceItem	否			定义用于获取主机接口的监控项参数。 主机接口在主机自动注册过程中使用。 在自动注册请求期间，如果指定监控项返回的值超过 255 字符限制，agent 将记录一条警告消息。 *system.run[]*监控项无论 AllowKey/DenyKey 值如何都受支持。 此选项仅在 HostInterface 未定义时使用。 自 Zabbix 4.4.0 起支持。
HostMetadata	否	0-255 字符		定义主机元数据的可选参数。主机元数据在主机自动注册过程中使用。 如果指定的值超过限制或不是 UTF-8 字符串，agent 将发出错误并不启动。 如果未定义，则从 HostMetadataItem 获取值。
HostMetadataItem	否			定义用于获取主机元数据的监控项参数。每次自动注册尝试时都会检索主机元数据监控项值。 在自动注册请求期间，如果指定监控项返回的值超过 255 字符限制，agent 将记录一条警告消息。 此选项仅在 HostMetadata 未定义时使用。 支持 UserParameters 和别名。无论 *EnableRemoteCommands* 值如何，都支持*system.run[]*。 监控项返回的值必须是 UTF-8 字符串，否则将被忽略。
Hostname	否		由 HostnameItem 设置	以英文逗号分隔的唯一、区分大小写的主机名列表。 采用主动检查时必需指定，并必须与服务器上配置的主机名匹配。如果未定义，则从 HostnameItem 获取值。 允许的字符：字母数字、'.'、' '、'_' 和'-'。 最大长度：每个主机名 128 个字符，整行 2048 个字符。
HostnameItem	否		system.hostname	如果未定义 Hostname，则用于生成 Hostname 的监控项。如果已定义 Hostname，则被忽略。 不支持 UserParameters 或别名，但无论 *EnableRemoteCommands* 值如何，都支持*system.run[]*。
Include	否			您可以在配置文件中包含单个文件或目录中的所有文件（默认位于 `C:\Program Files\Zabbix Agent 2`，如果使用 Windows MSI 安装包安装 Zabbix agent；如果以 zip 归档安装 Zabbix agent，则位于安装期间指定的文件夹）。所有包含的文件必须具有正确的语法，否则 agent 将不启动。 路径可以相对于 *zabbix_agent2.conf* 文件位置（例如，`Include=.\zabbix_agent2.d\plugins.d\*.conf`）。 要仅包括指定目录中的相关文件，支持使用星号通配符字符进行模式匹配。例如：`C:\Program Files\Zabbix Agent2\zabbix_agent2.d\*.conf`。 有关限制的特别说明。
ListenIP	否		0.0.0.0	agent 应监听的以逗号分隔的 IP 地址列表。 如果连接到 Zabbix server，第一个 IP 地址将发送到 Zabbix server，以检索主动检查列表。
ListenPort	否	1024-32767	10050	Agent 将在此端口上监听来自服务器的连接。
LogFile	是，如果 LogType 设置为 *file*，否则 否		c:\zabbix_agent2.log	如果 LogType 是'file'，日志文件名称。

参数	必选	范围	默认值	描述
LogFileSize	否	0-1024	1	日志文件的最大大小（MB）。 0 - 禁用自动日志轮换。 注意：如果达到日志文件大小限制，并且出于任何原因文件轮换失败，现有的日志文件将被截断并重新开始。
LogType	否		file	指定日志消息写入的位置： *file* - 由 LogFile 参数指定的文件， *console* - 标准输出。
PersistentBufferFile	否			Zabbix Agent2 应保留 SQLite 数据库的文件。 必须是完整文件名。 如果启用了持久缓冲区（*EnablePersistentBuffer=1*），则仅使用此参数。
PersistentBufferPeriod	否	1m-365d	1h	在无法连接到 server 或 proxy 时，数据应存储的时间周期。旧数据将丢失。日志数据将被保留。 如果启用了持久缓冲区（*EnablePersistentBuffer=1*），则仅使用此参数。
Plugins	否			自 Zabbix 6.0.0 起，大多数插件都有自己的配置文件。agent 配置文件包含下面列出的插件参数。
Plugins.Log.MaxLinesPerSecond	否	1-1000	20	agent 在处理'log' 和'eventlog' 主动检查时，每秒钟发送到 Zabbix server 或 proxy 的新行的最大数量。 提供的值将被'log' 或'eventlog' 监控项键中提供的'maxlines' 参数覆盖。 注意：Zabbix 将处理 *MaxLinesPerSecond* 设置的 10 倍的新行数，以在日志监控项中查找所需的字符串。 此参数自 4.4.2 起支持，并替代了 MaxLinesPerSecond。
Plugins.System.Run.LogRemoteCommands	否		0	启用将执行的 shell 命令记录为警告的日志。 0 - 禁用 1 - 启用 如果远程执行，命令将被记录。如果由 HostMetadataItem、HostInterfaceItem 或 HostnameItem 参数本地启动 system.run[]，则不会创建日志条目。 此参数自 4.4.2 起支持，并替代了 LogRemoteCommands。
PluginSocket	否		\.\pipe\agent.plugin.sock	用于可加载插件通信的 unix 套接字路径。
PluginTimeout	否	1-30	全局超时	可加载插件连接的超时时间。
RefreshActiveChecks	否	60-3600	120	主动检查列表刷新的频率，以秒为单位。 注意，在刷新主动检查失败后，下一次刷新将在 60 秒后尝试。
Server	是			逗号分隔的 IP 地址列表，可选地使用 CIDR 表示法，或 Zabbix servers 和 Zabbix proxies 的 DNS 名称。 仅接受此处列出的主机的传入连接。 如果启用了 IPv6 支持，则'127.0.0.1'，'::ffff:127.0.0.1' 被视为相等，'::/0' 将允许任何 IPv4 或 IPv6 地址。 '0.0.0.0/0' 可用于允许任何 IPv4 地址。 示例： Server=127.0.0.1,192.168.1.0/24,::1,2001:db8::/32,zabbix.example.com 允许空格。

参数	必选	范围	默认值	描述
ServerActive	否			获取主动检查的 Zabbix server/proxy 地址或集群配置。 server/proxy 地址是 IP 地址或 DNS 名称，可选地由冒号分隔的端口。 集群配置是一个或多个由分号分隔的服务器地址。 可以指定多个 Zabbix server/集群和 Zabbix proxies，由逗号分隔。 每个 Zabbix server/集群不应指定多个 Zabbix proxy。 如果指定了 Zabbix proxy，则不应为该 proxy 指定 Zabbix server/集群。 可以提供多个地址以并行使用几个独立的 Zabbix servers。允许空格。 如果未指定端口，则使用默认端口。 如果为该主机指定了端口，则必须将 IPv6 地址用方括号括起来。 如果未指定端口，则 IPv6 地址的方括号是可选的。 如果未指定此参数，则禁用主动检查。 Zabbix proxy 的示例： ServerActive=127.0.0.1:10051 多服务器的示例： ServerActive=127.0.0.1:20051,zabbix.domain,[::1]:30051,::1,[12fc::1] 高可用性的示例： ServerActive=zabbix.cluster.node1;zabbix.cluster.node2:20051;zabbix.clu 具有两个集群和一台服务器的高可用性的示例： ServerActive=zabbix.cluster.node1;zabbix.cluster.node2:20051,zabbix.clu
SourceIP	否			源 IP 地址用于： - 与 Zabbix server 或 Zabbix proxy 的出站连接； - 在执行某些监控项（web.page.get, net.tcp.port 等）时建立连接。
StatusPort	否	1024-32767		如果设置，agent 将在此端口上监听 HTTP 状态请求（http://localhost:<port>/status）。
Timeout	否	1-30	3	处理时间不超过 Timeout 秒。
TLSAccept	是，如果定义了 TLS 证书或 PSK 参数（即使是未加密连接），否则否			接受的传入连接类型。用于被动检查。可以指定多个值，用逗号分隔： *unencrypted* - 接受未加密的连接（默认） *psk* - 接受使用 TLS 和预共享密钥 (PSK) 的连接 *cert* - 接受使用 TLS 和证书的连接
TLSCAFile	否			包含顶层 CA 证书的文件的完整路径名，用于对等证书验证，用于 Zabbix 组件之间的加密通信。
TLSCertFile	否			包含 agent 证书或证书链的文件的完整路径名，用于与 Zabbix 组件的加密通信。
TLSConnect	是，如果定义了 TLS 证书或 PSK 参数（即使是未加密连接），否则否			agent 应如何连接到 Zabbix server 或 proxy。用于主动检查。只能指定一个值： *unencrypted* - 不使用加密连接（默认） *psk* - 使用 TLS 和预共享密钥 (PSK) 连接 *cert* - 使用 TLS 和证书连接
TLSCRLFile	否			包含吊销证书的文件的完整路径名。此参数用于与 Zabbix 组件的加密通信。
TLSKeyFile	否			包含 agent 私钥的文件的完整路径名，用于与 Zabbix 组件的加密通信。
TLSPSKFile	否			包含 agent 预共享密钥的文件的完整路径名，用于与 Zabbix 组件的加密通信。
TLSPSKIdentity	否			预共享密钥身份字符串，用于与 Zabbix server 的加密通信。
TLSServerCertIssuer	否			允许的服务器（proxy）证书颁发者。
TLSServerCertSubject	否			允许的服务器（proxy）证书主题。

参数	必选	范围	默认值	描述
UnsafeUserParameters	否	0,1	0	允许在用户定义参数的参数中传递所有字符。 不允许以下字符： \ ' " ` * ? [ ] { } ~ $! & ; ( ) < > | # @ 此外，不允许新行字符。
UserParameter	否			要监视的用户定义参数。可以有多个用户定义参数。 格式：UserParameter=<key>,<shell command> 注意，shell 命令不得仅返回空字符串或 EOL。 如果指定了 UserParameterDir 参数，shell 命令可能具有相对路径。 示例： UserParameter=system.test,who
UserParameterDir	否			UserParameter 命令的默认搜索路径。如果使用，agent 在执行命令之前将工作目录更改为此处指定的目录。因此，UserParameter 命令可以具有相对的 `./` 前缀而不是完整路径。 只允许一个条目。 示例：UserParameterDir=/opt/myscripts

7 Zabbix agent 2 插件

概述

本节包含 Zabbix agent 2 插件配置文件参数的说明。请使用侧边栏访问有关特定插件的信息。

1 Ceph 插件

概述

本节列出了 Zabbix agent 2 的 Ceph 插件配置文件 (ceph.conf) 中所有支持的参数。请注意：

- 默认值表示的是进程默认值，而不是附带的配置文件中的值；
- Zabbix 仅支持 UTF-8 编码格式的配置文件，不支持 BOM；
- 只支持在行首以 "#" 开头来注释。

参数

参数	必填	范围	默认值	描述
Plugins.Ceph.Default.ApiKey	否			连接到 Ceph 的默认 API 密钥；如果项目键或命名会话中未指定值，则使用此值。 自版本 6.0.18 起支持
Plugins.Ceph.Default.User	否			连接到 Ceph 的默认用户名；如果项目键或命名会话中未指定值，则使用此值。 自版本 6.0.18 起支持
Plugins.Ceph.Default.Uri	否		https://localhost:8003	连接到 Ceph 的默认 URI；如果项目键或命名会话中未指定值，则使用此值。 不应包含嵌入式凭据（它们将被忽略）。 必须符合 URI 格式。 仅支持 `https` 协议；可以省略协议。 可以省略端口（默认 =8003）。 示例：`https://127.0.0.1:8003` `localhost` 自版本 6.0.18 起支持
Plugins.Ceph.InsecureSkipVerify	否	false / true	false	决定 http 客户端是否应验证服务器的证书链和主机名。 如果为 *true*，则 TLS 接受服务器提供的任何证书以及该证书中的任何主机名。在此模式下，TLS 容易受到中间人攻击（仅应用于测试）。
Plugins.Ceph.KeepAlive	否	60-900	300	未使用的插件连接被关闭之前的最长等待时间（秒）。
Plugins.Ceph.Sessions.<SessionName>.ApiKey	否			命名会话的 API 密钥。 **<SessionName>** - 为项目键中使用定义会话名称。

参数	必填	范围	默认值	描述
Plugins.Ceph.Sessions.<SessionName>.User	否			命名会话的用户名。 **<SessionName>** - 为项目键中使用定义会话名称。
Plugins.Ceph.Sessions.<SessionName>.Uri	否			命名会话的连接字符串。 **<SessionName>** - 为项目键中使用定义会话名称。 不应包含嵌入式凭据（它们将被忽略）。 必须符合 URI 格式。 仅支持 `https` 协议；可以省略协议。 可以省略端口（默认 =8003）。 示例：`https://127.0.0.1:8003` `localhost`
Plugins.Ceph.Timeout	否	1-30	全局超时	请求执行超时（在关闭之前等待请求完成的时间）。

另请参阅：

- Zabbix agent 2 通用配置参数的描述：Zabbix agent 2 (UNIX) / Zabbix agent 2 (Windows)
- 插件配置指南

2 Docker 插件

概述

本节列出了 Zabbix agent 2 的 Docker 插件配置文件 (docker.conf) 中所有支持的参数。请注意：

- 默认值表示的是进程默认值，而不是附带的配置文件中的值；
- Zabbix 仅支持 UTF-8 编码格式的配置文件，不支持 BOM；
- 只支持在行首以 “#” 开头来注释。

参数

参数	必填	范围	默认值	描述
Plugins.Docker.Endpoint	否		unix:///var/run/docker.sock	Docker 守护进程 unix-socket 位置。 必须包含一个协议（仅支持 `unix://`）。
Plugins.Docker.Timeout	否	1-30	全局超时	请求执行超时（在关闭之前等待请求完成的时间）。

另请参阅：

- Zabbix agent 2 通用配置参数的描述：Zabbix agent 2 (UNIX) / Zabbix agent 2 (Windows)
- 插件配置指南

3 Ember+ 插件

概览

本节列出了 Ember+ Zabbix 代理 2 插件配置文件（ember.conf）支持的参数。

Ember+ 插件是一个可加载插件，并且在 Ember+ 插件仓库 中可用并有完整描述。

此插件自 Zabbix 6.0.30 起被支持，目前只能从源代码（适用于 Unix 和 Windows）编译。

请注意：

- 默认值反映了进程默认值，而不是配置文件中的值；
- Zabbix 仅支持 UTF-8 编码的配置文件，不包含 BOM；
- 仅支持以”#” 开头的注释在行的开始处。

参数

参数	必填	范围	默认值	描述
Plugins.EmberPlus.Default.Uri	否		tcp://localhost:9000	连接的默认 URI。唯一支持的方案是 `tcp://`。可以省略方案。嵌入式凭据将被忽略。
Plugins.EmberPlus.KeepAlive	否	60-900	300	未使用的插件连接被关闭之前的最长等待时间（秒）。

参数	必填	范围	默认值	描述
Plugins.EmberPlus.Sessions.<SessionName>.Url	否		tcp://localhost:9092	命名会话的连接 URI。唯一支持的方案是 tcp://。可以省略方案。嵌入式凭据将被忽略。 **<SessionName>** - 为项目键中使用定义会话名称。
Plugins.EmberPlus.System.Path	否			Ember+ 插件可执行文件的路径。 示例用法： Plugins.EmberPlus.System.Path=/usr/sbin/zabbix-agent2-plu
Plugins.EmberPlus.Timeout	否	1-30	全局超时	首次连接服务器并会话中的后续操作时等待响应的时间量。

另请参阅：

- Zabbix agent 2 通用配置参数的描述：Zabbix agent 2 (UNIX) / Zabbix agent 2 (Windows)
- 插件配置指南

4 Memcached 插件

概述

本节列出了 Memcached Zabbix agent 2 插件配置文件 (memcached.conf) 中支持的参数。注意：

- 默认值反映的是进程默认值，而不是附带的配置文件中的值；
- Zabbix 仅支持不带BOM的 UTF-8 编码的配置文件；
- 仅支持在行首以”#” 开头的注释。

参数

参数	必填	范围	默认值	描述
Plugins.Memcached.Default.Password	否			连接到 Memcached 的默认密码；如果项目键或命名会话中未指定值，则使用此密码。 自版本 6.0.18 起支持
Plugins.Memcached.Default.Uri	否		tcp://localhost:11211	连接到 Memcached 的默认 URI；如果项目键或命名会话中未指定值，则使用此 URI。 不应包含嵌入式凭据（它们将被忽略）。 必须符合 URI 格式。 支持的方案：tcp, unix；可以省略方案。 可以省略端口（默认 =11211）。 示例：tcp://localhost:11211 localhost unix:/var/run/memcached.sock 自版本 6.0.18 起支持
Plugins.Memcached.Default.User	否			连接到 Memcached 的默认用户名；如果项目键或命名会话中未指定值，则使用此用户名。 自版本 6.0.18 起支持
Plugins.Memcached.KeepAlive	否	60-900	300	未使用的插件连接被关闭之前的最长等待时间（秒）。
Plugins.Memcached.Sessions.<SessionName>.Password	否			命名会话密码。 **<SessionName>** - 定义会话名称，用于项目键中。
Plugins.Memcached.Sessions.<SessionName>.Uri	否			命名会话的连接字符串。 **<SessionName>** - 定义会话名称，用于项目键中。 不应包含嵌入式凭据（它们将被忽略）。 必须符合 URI 格式。 支持的方案：tcp, unix；可以省略方案。 可以省略端口（默认 =11211）。 示例：tcp://localhost:11211 localhost unix:/var/run/memcached.sock
Plugins.Memcached.Sessions.<SessionName>.User	否			命名会话用户名。 **<SessionName>** - 定义会话名称，用于项目键中。
Plugins.Memcached.Timeout	否	1-30	全局超时	请求执行超时（在关闭之前等待请求完成的时间）。

另请参阅：

- Zabbix agent 2 通用配置参数的描述：Zabbix agent 2 (UNIX) / Zabbix agent 2 (Windows)
- 插件配置指南

5 Modbus 插件

概述

本节列出了 Zabbix agent 2 的 Modbus 插件配置文件 (modbus.conf) 中所有支持的参数。请注意：

- 默认值表示的是进程默认值，而不是附带的配置文件中的值；
- Zabbix 仅支持不带BOM的 UTF-8 编码的配置文件；
- 仅支持在行首以”#” 开头的注释。

参数

参数	必填	范围	默认值	描述
Plugins.Modbus.Sessions.<SessionName>.Endpoint				端点是一个由协议方案、主机地址以及端口或串行端口名称和属性组成的连接字符串。 **<SessionName>** - 定义会话名称，用于项目键中。
Plugins.Modbus.Sessions.<SessionName>.SlaveID				命名会话的从机 ID。 **<SessionName>** - 定义会话名称，用于项目键中。 示例：`Plugins.Modbus.Sessions.MB1.SlaveID=20` 注意如果在项目键 中提供的从机 ID 参数值为空，则仅检查此命名会话参数。
Plugins.Modbus.Sessions.<SessionName>.Timeout				命名会话的超时时间。 **<SessionName>** - 定义会话名称，用于项目键中。 示例：`Plugins.Modbus.Sessions.MB1.Timeout=2`
Plugins.Modbus.Timeout		1-30	全局超时	请求执行超时（在关闭之前等待请求完成的时间）。

另请参阅：

- Zabbix agent 2 通用配置参数的描述：Zabbix agent 2 (UNIX) / Zabbix agent 2 (Windows)
- 插件配置指南

6 MongoDB 插件

概览

本节列出了 MongoDB Zabbix 代理 2 插件配置文件（mongo.conf）支持的参数。

自 Zabbix 6.0.6 起，MongoDB 插件是一个可加载插件，并且在 MongoDB 插件仓库 中可用并有完整描述。

自 Zabbix 6.0.19 起，Windows 的预编译插件二进制文件可用，并且与之前的 6.0 版本兼容。

请注意：

- 默认值反映了进程默认值，而不是配置文件中的值；
- Zabbix 仅支持 UTF-8 编码的配置文件，不包含 BOM；
- 仅支持以”#” 开头的注释在行的开始处。

选项

参数	描述
-V --version	打印插件版本及 license 信息。
-h --help	打印帮助信息（简化版）。

参数

> **Note:**
> 在 Zabbix 6.0.6 之前的版本中，参数名称以 *Plugins.Mongo.<Parameter>* 开头，而不是 *Plugins.MongoDB.<Parameter>*。例如，*Plugins.Mongo.KeepAlive*。

参数	必填	范围	默认值	描述
Plugins.MongoDB.Default.Password	否			连接到 MongoDB 的默认密码；如果项目键或命名会话中未指定值，则使用此密码。 自版本 6.0.18 起支持
Plugins.MongoDB.Default.Uri	否			连接到 MongoDB 的默认 URI；如果项目键或命名会话中未指定值，则使用此 URI。 不应包含嵌入式凭据（它们将被忽略）。 必须符合 URI 格式。 仅支持 `tcp` 方案；可以省略方案。 可以省略端口（默认 =27017）。 示例：`tcp://127.0.0.1:27017`, `tcp:localhost`, `localhost` 自版本 6.0.18 起支持
Plugins.MongoDB.Default.User	否			连接到 MongoDB 的默认用户名；如果项目键或命名会话中未指定值，则使用此用户名。 自版本 6.0.18 起支持
Plugins.MongoDB.KeepAlive	否	60-900	300	未使用的插件连接被关闭之前的最长等待时间（秒）。
Plugins.MongoDB.Sessions.<SessionName>.Password	否			命名会话密码。 **<SessionName>** - 定义会话名称，用于项目键中。
Plugins.MongoDB.Sessions.<SessionName>.TLSCAFile	否 (如果 Plugins.MongoDB.Sessions.<SessionName>.TLSConnect 设置为 verify_ca 或 verify_full，则为是)			用于 Zabbix 代理 2 与监控数据库之间加密通信的顶级 CA 证书文件的完整路径名，用于对等证书验证。 **<SessionName>** - 定义会话名称，用于项目键中。 支持插件版本 1.2.0, 6.0.13 及更新版本[1].
Plugins.MongoDB.Sessions.<SessionName>.TLSCertFile	如果指定了 Plugins.MongoDB.Sessions.<SessionName>.TLSKeyFile，则为是 在 6.0.24 版本之前，如果 Plugins.MongoDB.Sessions.<SessionName>.TLSConnect 设置为 verify_ca 或 verify_full，则始终为必填			包含代理证书或证书链的文件的完整路径名，用于 Zabbix 代理 2 与监控数据库之间的加密通信。 **<SessionName>** - 定义会话名称，用于项目键中。 支持插件版本 1.2.0, 6.0.13 及更新版本[1].
Plugins.MongoDB.Sessions.<SessionName>.TLSConnect	否			Zabbix 代理 2 与监控数据库之间通信的加密类型。 **<SessionName>** - 定义会话名称，用于项目键中。 支持的值： *required* - 要求 TLS 连接； *verify_ca* - 验证证书； *verify_full* - 验证证书和 IP 地址。 支持插件版本 1.2.0, 6.0.13 及更新版本[1].
Plugins.MongoDB.Sessions.<SessionName>.TLSKeyFile	如果指定了 Plugins.MongoDB.Sessions.<SessionName>.TLSCertFile，则为是 在 6.0.24 版本之前，如果 Plugins.MongoDB.Sessions.<SessionName>.TLSConnect 设置为 verify_ca 或 verify_full，则始终为必填			包含数据库私钥的文件的完整路径名，用于 Zabbix 代理 2 与监控数据库之间的加密通信。 **<SessionName>** - 定义会话名称，用于项目键中。 支持插件版本 1.2.0, 6.0.13 及更新版本[1].

参数	必填	范围	默认值	描述
Plugins.MongoDB.Sessions.<SessionName>.Uri	否			命名会话的连接字符串。 **<SessionName>** - 定义会话名称，用于项目键中。 不应包含嵌入式凭据（它们将被忽略）。 必须符合 URI 格式。 仅支持 tcp 方案；可以省略方案。 可以省略端口（默认 =27017）。 示例：tcp://127.0.0.1:27017, tcp:localhost, localhost
Plugins.MongoDB.Sessions.<SessionName>.User	否			命名会话用户名。 **<SessionName>** - 定义会话名称，用于项目键中。
Plugins.MongoDB.System.Path	否			外部插件可执行文件的路径。自 Zabbix 6.0.6 起支持
Plugins.MongoDB.Timeout	否	1-30	全局超时	请求执行超时（在关闭之前等待请求完成的时间）。

另请参阅：

- Zabbix 代理 2 通用配置参数的描述：Zabbix 代理 2 (UNIX) / Zabbix 代理 2 (Windows)
- 插件配置指南

脚注

[1] - 自 Zabbix 6.0.13 起，可加载插件开始使用与 Zabbix 本身相同的版本系统。结果，MongoDB 插件版本从 1.2.0 变更为 6.0.13。

7 MQTT 插件

概览

本节列出了 MQTT Zabbix 代理 2 插件配置文件（mqtt.conf）支持的参数。

请注意：

- 默认值反映了进程默认值，而不是配置文件中的值；
- Zabbix 仅支持 UTF-8 编码的配置文件，不包含 BOM；
- 仅支持以 “#” 开头的注释在行的开始处。

参数

参数	必填	范围	默认值	描述
Plugins.MQTT.Default.Password	否			连接到 MQTT 的默认密码；如果项目键或命名会话中未指定值，则使用此密码。 自版本 6.0.19 起支持
Plugins.MQTT.Default.TLSCAFile	否			包含顶层 CA 证书的文件的完整路径名，用于 Zabbix 代理 2 与 MQTT 代理之间加密通信的对等证书验证；如果命名会话中未指定值，则使用。 自版本 6.0.19 起支持
Plugins.MQTT.Default.TLSCertFile	否			包含代理证书或证书链的文件的完整路径名，用于 Zabbix 代理 2 与 MQTT 代理之间的加密通信；如果命名会话中未指定值，则使用。 自版本 6.0.19 起支持
Plugins.MQTT.Default.TLSKeyFile	否			包含 MQTT 私钥的文件的完整路径名，用于 Zabbix 代理 2 与 MQTT 代理之间的加密通信；如果命名会话中未指定值，则使用。 自版本 6.0.19 起支持
Plugins.MQTT.Default.Topic	否			MQTT 订阅的默认主题；如果项目键或命名会话中未指定值，则使用。 主题可能包含通配符 (”+”,”#”) 示例：path/to/file path/to/# path/+/topic 自版本 6.0.19 起支持

参数	必填	范围	默认值	描述
Plugins.MQTT.Default.Url	否		tcp://localhost:1883	默认 MQTT 代理连接字符串；如果项目键或命名会话中未指定值，则使用。 不应包含查询参数。 必须符合 URL 格式。 支持的方案：tcp (默认), ws, tls；可以省略方案。 可以省略端口 (默认 =1883)。 示例：tcp://host:1883 localhost ws://host:8080 自版本 6.0.19 起支持
Plugins.MQTT.Default.User	否			连接到 MQTT 的默认用户名；如果项目键或命名会话中未指定值，则使用。 自版本 6.0.19 起支持
Plugins.MQTT.Sessions.<SessionName>.Password	否			命名会话密码。 **<SessionName>** - 定义会话名称，用于项目键中。 自版本 6.0.19 起支持
Plugins.MQTT.Sessions.<SessionName>.TLSCAFile	否			包含顶层 CA 证书的文件的完整路径名，用于 Zabbix 代理 2 与 MQTT 代理之间的加密通信。 **<SessionName>** - 定义会话名称，用于项目键中。 自版本 6.0.19 起支持
Plugins.MQTT.Sessions.<SessionName>.TLSCertFile	否			包含代理证书或证书链的文件的完整路径名，用于 Zabbix 代理 2 与 MQTT 代理之间的加密通信。 **<SessionName>** - 定义会话名称，用于项目键中。 自版本 6.0.19 起支持
Plugins.MQTT.Sessions.<SessionName>.TLSKeyFile	否			包含 MQTT 私钥的文件的完整路径名，用于 Zabbix 代理 2 与 MQTT 代理之间的加密通信。 **<SessionName>** - 定义会话名称，用于项目键中。 自版本 6.0.19 起支持
Plugins.MQTT.Sessions.<SessionName>.Topic	否			命名会话的 MQTT 订阅主题。 **<SessionName>** - 定义会话名称，用于项目键中。 主题可能包含通配符 (”+”,”#”) 示例：path/to/file path/to/# path/+/topic 自版本 6.0.19 起支持
Plugins.MQTT.Sessions.<SessionName>.Url	否			命名会话的连接字符串。 **<SessionName>** - 定义会话名称，用于项目键中。 不应包含查询参数。 必须符合 URL 格式。 支持的方案：tcp (默认), ws, tls；可以省略方案。 可以省略端口 (默认 =1883)。 示例：tcp://host:1883 localhost ws://host:8080 自版本 6.0.19 起支持
Plugins.MQTT.Sessions.<SessionName>.User	否			命名会话用户名。 **<SessionName>** - 定义会话名称，用于项目键中。 自版本 6.0.19 起支持
Plugins.MQTT.Timeout	否	1-30	全局超时	请求执行超时（在关闭之前等待请求完成的时间）。

另请参阅：

- Zabbix 代理 2 通用配置参数的描述：Zabbix agent 2 (UNIX) / Zabbix agent 2 (Windows)
- 插件配置指南

8 MSSQL 插件

概览

本节列出了 MSSQL Zabbix 代理 2 插件配置文件（mssql.conf）支持的参数。

MSSQL 插件是一个可加载插件，并且在 MSSQL 插件仓库 中可用并有完整描述。

此插件自 Zabbix 6.0.27 起被支持。Windows 的预编译插件二进制文件自 Zabbix 6.0.28 起可用，并且与 6.0.27 版本兼容。

请注意：

- 默认值反映了进程默认值，而不是配置文件中的值；
- Zabbix 仅支持 UTF-8 编码的配置文件，不包含 BOM；
- 仅支持以“#”开头的注释在行的开始处。

参数

参数	必填	范围	默认值	描述
Plugins.MSSQL.CustomQueriesDir	否		空	指定包含用户定义的.sql 文件的目录的文件路径，插件可以在启动时加载配置目录中的所有可用.sql 文件。这意味着对自定义查询文件的任何更改直到插件重新启动后才会被反映出来。插件与 Zabbix agent 2 一起启动和停止。
Plugins.MSSQL.Default.CACertPath	否			MSSQL 服务器证书颁发机构（CA）的公钥证书的默认文件路径。证书必须为 PEM 格式。
Plugins.MSSQL.Default.Database	否			默认连接的数据库名称。
Plugins.MSSQL.Default.Encrypt	否			指定默认的连接加密类型。可能的值为： *true* - 插件和服务器之间的数据传输是加密的； *false* - 插件和服务器之间的数据传输在登录数据包之后不加密； *strict* - 插件和服务器之间的数据传输使用*TDS8*进行端到端加密； *disable* - 插件和服务器之间的数据传输不加密。
Plugins.MSSQL.Default.HostNameInCertificate	否			默认情况下，MSSQL 服务器证书的通用名称（CN）。
Plugins.MSSQL.Default.Password	否			默认情况下，发送到受保护的 MSSQL 服务器的密码。
Plugins.MSSQL.Default.TLSMinVersion	否			默认情况下使用的最小 TLS 版本。可能的值为：`1.0`, `1.1`, `1.2`, `1.3`。
Plugins.MSSQL.Default.TrustServerCertificate	否			默认情况下，插件是否应该信任服务器证书而不对其进行验证。可能的值为：`true`, `false`。
Plugins.MSSQL.Default.Uri	否		sqlserver://localhost:1433	默认连接的 URI。唯一支持的 schema 是 `sqlserver://`。可以省略 schema。嵌入式凭据将被忽略。
Plugins.MSSQL.Default.User	否			默认发送到受保护的 MSSQL 服务器的用户名。
Plugins.MSSQL.KeepAlive	否	60-900	300	未使用的插件连接在关闭前等待的最大时间（以秒为单位）。
Plugins.MSSQL.Sessions.<SessionName>.CACertPath	否			命名会话的 MSSQL 服务器证书颁发机构（CA）的公钥证书的文件路径。证书必须为 PEM 格式。**<SessionName>** - 定义会话名称，用于在项目键中使用。
Plugins.MSSQL.Sessions.<SessionName>.Database	否			命名会话连接的数据库名称。 **<SessionName>** - 定义会话名称，用于在监控项键中使用。
Plugins.MSSQL.Sessions.<SessionName>.Encrypt	否			指定命名会话的连接加密类型。可能的值为： *true* - 插件和服务器之间的数据发送是加密的； *false* - 插件和服务器之间的数据发送在登录数据包之后不加密； *strict* - 插件和服务器之间的数据发送使用TDS8进行端到端加密； *disable* - 插件和服务器之间的数据发送不加密。 **<SessionName>** - 定义会话名称，用于在监控项键中使用。
Plugins.MSSQL.Sessions.<SessionName>.HostNameInCertificate	否			命名会话的 MSSQL 服务器证书的通用名称（CN）。 **<SessionName>** - 定义会话名称，用于在监控项键中使用。
Plugins.MSSQL.Sessions.<SessionName>.Password	否			发送到受保护的 MSSQL 服务器的命名会话的密码。 **<SessionName>** - 定义会话名称，用于在监控项键中使用。
Plugins.MSSQL.Sessions.<SessionName>.TLSMinVersion	否			命名会话使用的最小 TLS 版本。可能的值为：`1.0`, `1.1`, `1.2`, `1.3`。 **<SessionName>** - 定义会话名称，用于在监控项键中使用。
Plugins.MSSQL.Sessions.<SessionName>.TrustServerCertificate	否			是否插件应对命名会话信任服务器证书而不进行验证。可能的值为：`true`, `false`。 **<SessionName>** - 定义会话名称，用于在监控项键中使用。
Plugins.MSSQL.Sessions.<SessionName>.Uri	否		sqlserver://localhost:1433	命名会话的连接 URI。唯一支持的 schema 是 `sqlserver://`。可以省略 schema。嵌入式凭据将被忽略。 **<SessionName>** - 定义会话名称，用于在监控项键中使用。
Plugins.MSSQL.Sessions.<SessionName>.User	否			发送到受保护的 MSSQL 服务器的命名会话的用户名。 **<SessionName>** - 定义会话名称，用于在监控项键中使用。

参数	必填	范围	默认值	描述
Plugins.MSSQL.System.Path	否			MSSQL 插件可执行文件的路径。 MSSQL 插件的全局设置。应用于所有连接。 示例用法： `Plugins.MSSQL.System.Path=/usr/sbin/zabbix-agent2-plugin/za`
Plugins.MSSQL.Timeout	否	1-30	全局超时	首次连接时以及会话中的后续操作等待服务器响应的时间量。

另请参阅：

- Zabbix agent 2 的一般配置参数描述：Zabbix agent 2 (UNIX) / Zabbix agent 2 (Windows)
- 插件配置指南

9 MySQL 插件

概述

本节列出了 Zabbix agent 2 的 MySQL 插件配置文件 (mysql.conf) 中所有支持的参数。请注意：

- 默认值表示的是进程默认值，而不是附带的配置文件中的值；
- Zabbix 仅支持不带BOM的 UTF-8 编码的配置文件；
- 仅支持在行首以”#” 开头的注释。

参数

参数	必填	范围	默认值	描述
Plugins.Mysql.CallTimeout	否	1-30	全局超时时间	请求完成前等待的最大时间（秒）。
Plugins.Mysql.CustomQueriesPath	否		空	用于存储自定义查询的目录的完整路径。 自版本 6.0.21 起支持。
Plugins.Mysql.Default.Password	否			连接到 MySQL 的默认密码；如果项目键或命名会话中未指定值，则使用。 自版本 6.0.18 起支持。
Plugins.Mysql.Default.TLSCAFile	否（如果是 Plugins.Mysql.Default.TLSConnect 设置为 *verify_ca* 或 *verify_full*）			包含用于 Zabbix agent 2 和监控数据库之间加密通信的顶级 CA（s）证书文件的完整路径名；如果命名会话中未指定值，则使用。 自版本 6.0.18 起支持。
Plugins.Mysql.Default.TLSCertFile	否（如果是 Plugins.Mysql.Default.TLSConnect 设置为 *verify_ca* 或 *verify_full*）			包含用于 Zabbix agent 2 和监控数据库之间加密通信的代理证书或证书链文件的完整路径名；如果命名会话中未指定值，则使用。 自版本 6.0.18 起支持。
Plugins.Mysql.Default.TLSConnect	否			Zabbix agent 2 和监控数据库之间通信的加密类型；如果命名会话中未指定值，则使用。 支持的值： *required* - 需要 TLS 连接； *verify_ca* - 验证证书； *verify_full* - 验证证书和 IP 地址。 自版本 6.0.18 起支持。
Plugins.Mysql.Default.TLSKeyFile	否（如果是 Plugins.Mysql.Default.TLSConnect 设置为 *verify_ca* 或 *verify_full*）			包含用于 Zabbix agent 2 和监控数据库之间加密通信的数据库私钥文件的完整路径名；如果命名会话中未指定值，则使用。 自版本 6.0.18 起支持。

参数	必填	范围	默认值	描述

Plugins.Mysql.Default.Uri	否		tcp://localhost:3306	连接到 MySQL 的默认 URI；如果项目键或命名会话中未指定值，则使用。
				
				不应包含嵌入的凭据（它们将被忽略）。
				必须符合 URI 格式。
				支持的方案：`tcp`, `unix`；可以省略方案。
				可以省略端口（默认 =3306）。
				示例：`tcp://localhost:3306`
				`localhost`
				`unix:/var/run/mysql.sock`
				自版本 6.0.18 起支持。

Plugins.Mysql.Default.User	否			连接到 MySQL 的默认用户名；如果项目键或命名会话中未指定值，则使用。 自版本 6.0.18 起支持。
Plugins.Mysql.KeepAlive	否	60-900	300	在关闭未使用的插件连接之前等待的最大时间（秒）。
Plugins.Mysql.Sessions.<SessionName>.Password	否			命名会话密码。 **<SessionName>** - 为项目键中使用定义会话名称。
Plugins.Mysql.Sessions.<SessionName>.TLSCAFile	否（如果是 Plugins.Mysql.Sessions.<SessionName>.TLSConnect 设置为 *verify_ca* 或 *verify_full*）			包含用于 Zabbix agent 2 和监控数据库之间加密通信的顶级 CA（s）证书文件的完整路径名。 **<SessionName>** - 为项目键中使用定义会话名称。
Plugins.Mysql.Sessions.<SessionName>.TLSCertFile	是，如果指定了 Plugins.Mysql.Sessions.<SessionName>.TLSKeyFile 在版本 6.0.23 之前，如果 Plugins.Mysql.Sessions.<SessionName>.TLSConnect 设置为 *verify_ca* 或 *verify_full* 总是必填			包含用于 Zabbix agent 2 和监控数据库之间加密通信的代理证书或证书链文件的完整路径名。 **<SessionName>** - 为项目键中使用定义会话名称。
Plugins.Mysql.Sessions.<SessionName>.TLSConnect	否			Zabbix agent 2 和监控数据库之间通信的加密类型。 **<SessionName>** - 为项目键中使用定义会话名称。 支持的值： *required* - 需要 TLS 连接； *verify_ca* - 验证证书； *verify_full* - 验证证书和 IP 地址。
Plugins.Mysql.Sessions.<SessionName>.TLSKeyFile	是，如果指定了 Plugins.Mysql.Sessions.<SessionName>.TLSCertFile 在版本 6.0.23 之前，如果 Plugins.Mysql.Sessions.<SessionName>.TLSConnect 设置为 *verify_ca* 或 *verify_full* 总是必填			包含用于 Zabbix agent 2 和监控数据库之间加密通信的数据库私钥文件的完整路径名。 **<SessionName>** - 为项目键中使用定义会话名称。

参数	必填	范围	默认值	描述
Plugins.Mysql.Sessions.<SessionName>.Uri	否			命名会话的连接字符串。 **<SessionName>** - 为项目键中使用定义会话名称。 不应包含嵌入的凭据（它们将被忽略）。 必须符合 URI 格式。 支持的方案：tcp, unix；可以省略方案。 可以省略端口（默认 =3306）。 示例：tcp://localhost:3306 localhost unix:/var/run/mysql.sock
Plugins.Mysql.Sessions.<SessionName>.User	否			命名会话用户名。 **<SessionName>** - 为项目键中使用定义会话名称。
Plugins.Mysql.Timeout	否	1-30	全局超时时间	请求执行超时时间（在关闭之前等待请求完成的时间）。

参见：

- Zabbix agent 2 通用配置参数的描述：Zabbix agent 2 (UNIX) / Zabbix agent 2 (Windows)
- 配置插件的说明

10 Oracle 插件

概览

这部分列出了 Oracle Zabbix agent 2 插件配置文件（oracle.conf）支持的参数。

请注意：

- 默认值反映了进程默认值，而不是配置文件中的值；
- Zabbix 仅支持 UTF-8 编码的配置文件，不包含BOM；
- 仅支持以”#” 开头的注释在行的开始处。

参数

参数	必填	范围	默认值	描述
Plugins.Oracle.CallTimeout	否	1-30	全局超时时间	请求完成的最大等待时间（秒）。
Plugins.Oracle.ConnectTimeout	否	1-30	全局超时时间	建立连接的最大等待时间（秒）。
Plugins.Oracle.CustomQueriesPath	否			包含自定义查询的.sql 文件的目录的完整路径名。 默认禁用。 示例：/etc/zabbix/oracle/sql
Plugins.Oracle.Default.Password	否			连接到 Oracle 的默认密码；如果项目键或命名会话中未指定值，则使用。 自版本 6.0.18 起支持
Plugins.Oracle.Default.Service	否			连接到 Oracle 的默认服务名称（不支持 SID）；如果项目键或命名会话中未指定值，则使用。 自版本 6.0.18 起支持
Plugins.Oracle.Default.Uri	否		tcp://localhost:1521	连接到 Oracle 的默认 URI；如果项目键或命名会话中未指定值，则使用。 不应包含嵌入的凭据（它们将被忽略）。 必须符合 URI 格式。 仅支持 tcp 方案；可以省略方案。 可以省略端口（默认 =1521）。 示例：tcp://127.0.0.1:1521 localhost 自版本 6.0.18 起支持
Plugins.Oracle.Default.User	否			连接到 Oracle 的默认用户名；如果项目键或命名会话中未指定值，则使用。 自版本 6.0.18 起支持
Plugins.Oracle.KeepAlive	否	60-900	300	在关闭未使用的插件连接之前等待的最大时间（秒）。
Plugins.Oracle.Sessions.<SessionName>.Password	否			命名会话密码。 **<SessionName>** - 定义用于项目键中的会话名称。

参数	必填	范围	默认值	描述
Plugins.Oracle.Sessions.<SessionName>.Service				用于连接的命名会话服务名称（不支持 SID）。 **<SessionName>** - 定义用于项目键中的会话名称。
Plugins.Oracle.Sessions.<SessionName>.Uri				Oracle 的命名会话连接字符串。 **<SessionName>** - 定义用于项目键中的会话名称。 不应包含嵌入的凭据（它们将被忽略）。 必须符合 URI 格式。 仅支持 `tcp` 方案；可以省略方案。 可以省略端口（默认 =1521）。 示例：`tcp://127.0.0.1:1521` `localhost`
Plugins.Oracle.Sessions.<SessionName>.User				命名会话用户名。 **<SessionName>** - 定义用于项目键中的会话名称。

参见：

- Zabbix agent 2 通用配置参数的描述：Zabbix agent 2 (UNIX) / Zabbix agent 2 (Windows)
- 配置插件的说明

11 PostgreSQL 插件

概览

这部分列出了 PostgreSQL Zabbix agent 2 插件配置文件（postgresql.conf）支持的参数。

自 Zabbix 6.0.10 起，PostgreSQL 插件是一个可加载插件，并且可以在PostgreSQL 插件仓库中找到并得到完整描述。

自Zabbix 6.0.19起，Windows 的预编译插件二进制文件可用，并且与之前的 6.0 版本兼容。

请注意：

- 默认值反映了进程默认值，而不是配置文件中的值；
- Zabbix 仅支持 UTF-8 编码的配置文件，不包含BOM；
- 仅支持以"#" 开头的注释在行的开始处。

选项

参数	描述
-V --version	打印插件版本和许可信息。
-h --help	打印帮助信息（简写）。

参数

Note:
在 Zabbix 6.0.10 之前的版本中，参数名称以 *Plugins.Postgres.<Parameter>* 开头，而不是 *Plugins.PostgreSQL.<Parameter>*。例如，*Plugins.Postgres.KeepAlive*。

参数	必填	范围	默认值	描述
Plugins.PostgreSQL.Default.CacheMode			prepare	PostgreSQL 连接的缓存模式。 支持的值： *prepare* （默认）- 将在 PostgreSQL 服务器上创建准备好的语句； *describe* - 将使用匿名准备好的语句来描述一个语句，而不需要在服务器上创建语句。 请注意"describe" 主要在使用不允许准备好的语句的环境中非常有用，例如当运行像 PgBouncer 这样的连接池时。 自版本 6.0.25 起支持。
Plugins.PostgreSQL.CallTimeout		1-30	全局超时时间	请求完成的最大等待时间（秒）。
Plugins.PostgreSQL.CustomQueriesPath			disabled	包含 *.sql* 文件的目录的完整路径名，这些文件包含自定义查询。
Plugins.PostgreSQL.Default.Database				连接到 PostgreSQL 的默认数据库；如果项目键或命名会话中未指定值，则使用。 自版本 6.0.18 起支持。

参数	必填	范围	默认值	描述
Plugins.PostgreSQL.Default.Password	否			连接到 PostgreSQL 的默认密码；如果项目键或命名会话中未指定值，则使用。 自版本 6.0.18 起支持。
Plugins.PostgreSQL.Default.TLSCAFile	否 (如果是 Plugins.PostgreSQL.Default.TLSConnect 设置为 *verify_ca* 或 *verify_full*)			包含顶级 CA(s) 证书的文件的完整路径名，用于对端证书验证，用于 Zabbix agent 2 与监控数据库之间的加密通信；如果命名会话中未指定值，则使用。 自版本 6.0.18 起支持。
Plugins.PostgreSQL.Default.TLSCertFile	否 (如果是 Plugins.PostgreSQL.Default.TLSConnect 设置为 *verify_ca* 或 *verify_full*)			包含 PostgreSQL 证书或证书链的文件的完整路径名，用于 Zabbix agent 2 与监控数据库之间的加密通信；如果命名会话中未指定值，则使用。 自版本 6.0.18 起支持。
Plugins.PostgreSQL.Default.TLSConnect	否			Zabbix agent 2 与监控数据库之间通信的加密类型；如果命名会话中未指定值，则使用。 支持的值： *required* - 使用 TLS 作为传输模式进行连接，不进行身份检查； *verify_ca* - 使用 TLS 连接并验证证书； *verify_full* - 使用 TLS 连接，验证证书，并验证数据库身份 (CN) 由 DBHost 指定并与其证书匹配。 未定义的加密类型意味着未加密连接。 自版本 6.0.18 起支持。
Plugins.PostgreSQL.Default.TLSKeyFile	否 (如果是 Plugins.PostgreSQL.Default.TLSConnect 设置为 *verify_ca* 或 *verify_full*)			包含 PostgreSQL 私钥的文件的完整路径名，用于 Zabbix agent 2 与监控数据库之间的加密通信；如果命名会话中未指定值，则使用。 自版本 6.0.18 起支持。

Plugins.PostgreSQL.Default.Uri	否			连接到 PostgreSQL 的默认 URI；如果项目键或命名会话中未指定值，则使用。
				
				不应包含嵌入的凭据（它们将被忽略）。
				必须符合 URI 格式。
				支持的方案：`tcp`, `unix`。
				示例：`tcp://127.0.0.1:5432`
				`tcp://localhost`
				`unix:/var/run/postgresql/.s.PGSQL.5432`
				自版本 6.0.18 起支持。

Plugins.PostgreSQL.Default.User	否			连接到 PostgreSQL 的默认用户名；如果项目键或命名会话中未指定值，则使用。 自版本 6.0.18 起支持。
Plugins.PostgreSQL.KeepAlive	否	60-900	300	在关闭未使用的插件连接之前等待的最大时间（秒）。
Plugins.PostgreSQL.Sessions.<SessionName>.CachePreparedMode	否		prepare	PostgreSQL 连接的缓存模式。 **<SessionName>** - 定义用于项目键中的会话名称。 支持的值： *prepare* （默认）- 将在 PostgreSQL 服务器上创建准备好的语句； *describe* - 将使用匿名准备好的语句来描述一个语句，而不需要在服务器上创建语句。 请注意”describe” 主要在使用不允许准备好的语句的环境中非常有用，例如当运行像 PgBouncer 这样的连接池时。 自版本 6.0.25 起支持。
Plugins.PostgreSQL.Sessions.<SessionName>.Database	否			会话连接的数据库。 **<SessionName>** - 定义用于项目键中的会话名称。
Plugins.PostgreSQL.Sessions.<SessionName>.Password	否	必须符合密码的格式。		会话连接的密码。 **<SessionName>** - 定义用于项目键中的会话名称。

参数	必填	范围	默认值	描述
Plugins.PostgreSQL.Sessions.<SessionName>.TLSCAFile	否 (如果是 Plugins.PostgreSQL.Sessions.<SessionName>.TLSConnect 设置为 *verify_ca* 或 *verify_full*)			包含顶级 CA(s) 证书的文件的完整路径名，用于对端证书验证。 **<SessionName>** - 定义用于项目键中的会话名称。
Plugins.PostgreSQL.Sessions.<SessionName>.TLSCertFile	是，如果指定了 Plugins.PostgreSQL.Sessions.<SessionName>.TLSKeyFile 在版本 6.0.22 之前，如果 Plugins.PostgreSQL.Sessions.<SessionName>.TLSConnect 设置为 *verify_ca* 或 *verify_full* 总是必填			包含 PostgreSQL 证书或证书链的文件的完整路径名。 **<SessionName>** - 定义用于项目键中的会话名称。
Plugins.PostgreSQL.Sessions.<SessionName>.TLSConnect	否			PostgreSQL 连接的加密类型。 **<SessionName>** - 定义用于项目键中的会话名称。 支持的值： *required* - 使用 TLS 作为传输模式进行连接，不进行身份检查； *verify_ca* - 使用 TLS 连接并验证证书； *verify_full* - 使用 TLS 连接，验证证书，并验证数据库身份 (CN) 由 DBHost 指定并与其证书匹配。 未定义的加密类型意味着未加密连接。
Plugins.PostgreSQL.Sessions.<SessionName>.TLSKeyFile	是，如果指定了 Plugins.PostgreSQL.Sessions.<SessionName>.TLSCertFile 在版本 6.0.22 之前，如果 Plugins.PostgreSQL.Sessions.<SessionName>.TLSConnect 设置为 *verify_ca* 或 *verify_full* 总是必填			包含 PostgreSQL 私钥的文件的完整路径名。 **<SessionName>** - 定义用于项目键中的会话名称。

Plugins.PostgreSQL.Sessions.<SessionName>.Uri	否			命名会话的连接字符串。
				<SessionName> - 定义用于项目键中的会话名称。
				
				不应包含嵌入的凭据（它们将被忽略）。
				必须符合 URI 格式。
				支持的方案：`tcp`, `unix`。
				示例：`tcp://127.0.0.1:5432`
				`tcp://localhost`
				`unix:/var/run/postgresql/.s.PGSQL.5432`

Plugins.PostgreSQL.Sessions.<SessionName>.User	否			命名会话的用户名。 **<SessionName>** - 定义用于项目键中的会话名称。
Plugins.PostgreSQL.System.Path	是			外部插件可执行文件的路径。自 Zabbix 6.0.10 起支持。
Plugins.PostgreSQL.Timeout	否	1-30	全局超时时间	请求执行超时时间（在关闭之前等待请求完成的时间）。

参见：

- Zabbix agent 2 通用配置参数的描述：Zabbix agent 2 (UNIX) / Zabbix agent 2 (Windows)
- 配置插件的说明

12 Redis 插件

概览

这部分列出了 Redis Zabbix agent 2 插件配置文件（redis.conf）支持的参数。

请注意：

- 默认值反映了进程默认值，而不是配置文件中的值；
- Zabbix 仅支持 UTF-8 编码的配置文件，不包含BOM；
- 仅支持以”#” 开头的注释在行的开始处。

参数

参数	必填	范围	默认值	描述
Plugins.Redis.Default.Password	否			连接到 Redis 的默认密码；如果项目键或命名会话中未指定值，则使用。 自版本 6.0.18 起支持

Plugins.Redis.Default.Uri	否		tcp://localhost:6379	连接到 Redis 的默认 URI；如果项目键或命名会话中未指定值，则使用。
				
				不应包含嵌入的凭据（它们将被忽略）。
				必须符合 URI 格式。
				支持的方案：`tcp`, `unix`；可以省略方案（自版本 5.2.3 起）。
				可以省略端口（默认 =6379）。
				示例：`tcp://localhost:6379`
				`localhost`
				`unix:/var/run/redis.sock`
				自版本 6.0.18 起支持

Plugins.Redis.KeepAlive	否	60-900	300	在关闭未使用的插件连接之前等待的最大时间（秒）。
Plugins.Redis.Sessions.<SessionName>.Password	否			命名会话密码。 **<SessionName>** - 定义用于项目键中的会话名称。

Plugins.Redis.Sessions.<SessionName>.Uri	否			命名会话的连接字符串。
				<SessionName> - 定义用于项目键中的会话名称。
				
				不应包含嵌入的凭据（它们将被忽略）。
				必须符合 URI 格式。
				支持的方案：`tcp`, `unix`；可以省略方案。
				可以省略端口（默认 =6379）。
				示例：`tcp://localhost:6379`
				`localhost`
				`unix:/var/run/redis.sock`

Plugins.Redis.Timeout	否	1-30	全局超时时间	请求执行超时时间（在关闭之前等待请求完成的时间）。

参见：

- Zabbix agent 2 通用配置参数的描述：Zabbix agent 2 (UNIX) / Zabbix agent 2 (Windows)
- 配置插件的说明

13 Smart 插件

概览

这部分列出了智能 Zabbix agent 2 插件配置文件（smart.conf）支持的参数。

请注意：

- 默认值反映了进程默认值，而不是配置文件中的值；
- Zabbix 仅支持 UTF-8 编码的配置文件，不包含BOM；
- 仅支持以 “#” 开头的注释在行的开始处。

参数

参数	必填	范围	默认值	描述
Plugins.Smart.Path	否		smartctl	smartctl 可执行文件的路径。
Plugins.Smart.Timeout	否	1-30	全局超时时间	请求执行超时时间（在关闭之前等待请求完成的时间）。

另请参阅：

- Zabbix agent 2 通用配置参数的描述：Zabbix agent 2 (UNIX) / Zabbix agent 2 (Windows)
- 配置插件的说明

8 Zabbix Java 网关

如果使用 `startup.sh` 和 `shutdown.sh` 脚本启动和停止Zabbix Java 网关, 则可以在 `settings.sh` 文件中指定必要的配置参数。启动和关闭脚本以配置文件为输入源，并且将 shell 变量 (第一列) 转换为相应的 Java 属性 (第二列)。

如果通过手动运行 `java` 命令来起动 Zabbix Java 网关, 可以通过命令行方式来指定 Java 属性。

变量	属性	是否必须	范围	默认值	描述
LISTEN_IP	zabbix.listenIP	否		0.0.0.0	监听 IP 地址。
LISTEN_PORT	zabbix.listenPort	否	1024-32767	10052	监听端口。
PID_FILE	zabbix.pidFile	否		/tmp/zabbix_java.pid	PID 文件的名称。如果省略，Zabbix Java gateway 将作为控制台应用程序启动。
PROPERTIES_FILE	zabbix.propertiesFile	否			属性文件的名称。可用于使用键值格式设置其他属性，以使其在命令行上不可见或覆盖现有属性。 示例: ”javax.net.ssl.trustStorePassword=<p
START_POLLERS	zabbix.startPollers	否	1-1000	5	启动的线程数。
TIMEOUT	zabbix.timeout	否	1-30	3	等待超时的时间。

Warning:
10052 端口没有在 IANA 注册.

9 Zabbix web 服务

概述

Zabbix web 服务是一个用于与外部 web 服务通信的进程。

本节列出了 Zabbix web service 配置文件 (zabbix_web_service.conf) 中所有支持的参数。请注意：

- 默认值表示的是进程默认值，而不是附带的配置文件中的值；
- Zabbix 仅支持 UTF-8 编码格式的配置文件，不支持 BOM；
- 只支持在行首以 “#” 开头来注释。

参数

参数	是否必须	范围	默认值	描述
AllowedIP	是			逗号分隔的 IP 地址列表（可选 CIDR 表示法），或 Zabbix servers 和 Zabbix proxies 的 DNS 名称 只接受此参数列出的主机的传入连接 如果启用了 IPv6 支持，则 “127.0.0.1”、“127.0.0.1”、“127:：ffff:127.0.0.1” 将被同等对待，“ ：: /0” 将允许任何 IPv4 或 IPv6 地址 0.0.0.0/0 可用于允许任何 IPv4 地址 例如： 127.0.0.1192.168.1.0/24 ,::12001:db8:：/32 , zabbix.example.com
DebugLevel	否	0-5	3	指定调试级别： 0-关于启动和停止 Zabbix 进程的基本信息 1-关键信息 2-错误信息 3-警告 4-用于调试（产生大量信息） 5-扩展调试（产生更多信息）
ListenPort	否	1024-32767	10053	服务监听端口。
LogFile	是，如果 LogType 设置 *file*, 否则否			当 LogType=file 时的日志文件名。 例如： /tmp/zabbix_web_service.log
LogFileSize	no	0-1024	1	日志文件的最大大小（MB）。 0 - 禁用日志自动切割。
LogType	否	system / file / console	file	指定日志消息写入的位置： *system*-syslog *file*-使用日志文件参数指定的文件 *console*-标准输出
Timeout	否	1-30	3	处理超时时间。
TLSAccept	否	unencrypted / cert	unencrypted	指定要使用的连接类型： 未 *unencrypted*-接受未加密的连接（默认值） *cert*-接受带有 TLS 和证书的连接
TLSCAFile	否			包含用于对等证书验证的顶级 CA 证书的文件的完整路径名，用于 Zabbix 组件之间的加密通信。
TLSCertFile	否			包含服务证书或证书链的文件的完整路径名，用于与 Zabbix 组件进行加密通信。
TLSKeyFile	否			包含用于与 Zabbix 组件进行加密通信的服务私钥的文件的完整路径名。

10 包含

概述

可以使用该 Include 参数将其他文件或目录包含到 server/proxy/agent 配置中。

关于包含的说明

如果 Include 参数用于包含文件，则该文件必须要可读。

如果 Include 参数用于包含目录：

- 目录中的所有文件必须要可读。
- 不应假定特定的包含顺序（例如，文件不按字母顺序包含）。因此，不要在多个 “Include” 文件中定义一个参数（例如，用特定参数覆盖常规设置）。
- 目录中的所有文件都包含在配置中。
- 小心一些文本编辑器自动创建的文件备份副本。例如，如果编辑 “include/my_specific.conf“文件生成一个备份副本 “include/my_specific_conf.BAK”，那么两个文件都会被包含。将 “include/my_specific_conf.BAK” 文件从 “Include” 目录中删除。在 Linux 中，可以使用 “ls -al” 命令检查 “Include” 目录的内容，以查找不必要的文件。

如果 Include 参数用于包含使用模式匹配的文件：

- 所有匹配到的文件必须要可读
- 不应假定特定的包含顺序（例如，文件不按字母顺序包含）。因此，不要在多个 “Include” 文件中定义一个参数（例如，用特定参数覆盖常规设置）。

4 协议

1 Server-proxy 数据交换协议

概述

Server - proxy 数据交换基于 JSON 格式.

请求和响应消息必须以header 头部与数据长度开头.

被动代理

代理配置请求

服务器发送 proxy config 请求以提供代理配置数据. 此请求间隔 ProxyConfigFrequency 秒 (服务器配置参数) .

名称			值类型	描述
server→proxy:				
request			*string*	'proxy config'
<table>			*object*	包含 <table> 数据的一个或多个对象
	fields		*array*	字段名称数组
		-	*string*	字段名称
	data		*array*	行数组
		-	*array*	列数组
		-	*string,number*	列值，其类型取决于数据库表结构中的类型
proxy→server:				
response			*string*	请求成功信息 ('success' 或'failed')
version			*string*	proxy 的版本信息 (<major>.<minor>.<build>)

例子:

server→proxy:

```
{
    "request": "proxy config",
    "globalmacro":{
        "fields":[
            "globalmacroid",
            "macro",
            "value"
        ],
        "data":[
            [
                2,
                "{$SNMP_COMMUNITY}",
                "public"
```

```
            ]
        ]
    },
    "hosts":{
        "fields":[
            "hostid",
            "host",
            "status",
            "ipmi_authtype",
            "ipmi_privilege",
            "ipmi_username",
            "ipmi_password",
            "name",
            "tls_connect",
            "tls_accept",
            "tls_issuer",
            "tls_subject",
            "tls_psk_identity",
            "tls_psk"
        ],
        "data":[
            [
                10001,
                "Linux",
                3,
                -1,
                2,
                "",
                "",
                "Linux",
                1,
                1,
                "",
                "",
                "",
                ""
            ],
            [
                10050,
                "Zabbix Agent",
                3,
                -1,
                2,
                "",
                "",
                "Zabbix Agent",
                1,
                1,
                "",
                "",
                "",
                ""
            ],
            [
                10105,
                "Logger",
                0,
                -1,
                2,
                "",
                "",
                "Logger",
```

```
                1,
                1,
                "",
                "",
                "",
                ""
            ]
        ]
    },
    "interface":{
        "fields":[
            "interfaceid",
            "hostid",
            "main",
            "type",
            "useip",
            "ip",
            "dns",
            "port",
            "bulk"
        ],
        "data":[
            [
                2,
                10105,
                1,
                1,
                1,
                "127.0.0.1",
                "",
                "10050",
                1
            ]
        ]
    },
    ...
}
```

proxy→server:

```
{
  "response": "success",
  "version": "5.4.0"
}
```

Proxy 请求

`proxy data` 请求用于从代理获取主机接口可用性、历史、发现和自动注册数据。此请求发送间隔 `ProxyDataFrequency` 秒 (服务器配置参数)。

名称	值类型	描述
server→proxy:		
request	*string*	'proxy data'
proxy→server:		
session	*string*	数据会话令牌
interface availability	*array*	*(*可选*)* 接口可用性数据对象数组
interfaceid	*number*	接口标识符
available	*number*	接口可用性 **0**, *INTERFACE_AVAILABLE_UNKNOWN* - 未知 **1**, *INTERFACE_AVAILABLE_TRUE* - 可用 **2**, *INTERFACE_AVAILABLE_FALSE* - 不可用
error	*string*	接口错误消息或空字符串

名称		值类型	描述
history data		*array*	*(*可选*)* 历史数据对象数组
	itemid	*number*	监控项标识符
	clock	*number*	监控项值时间戳 (秒)
	ns	*number*	监控项值时间戳 (纳秒)
	value	*string*	*(*可选*)* 监控项值
	id	*number*	值标识符（自增 id，在一个数据会话中是唯一的）
	timestamp	*number*	*(*可选*)* 日志类型监控项的时间戳
	source	*string*	*(*可选*)* 事件日志监控项数据源
	severity	*number*	*(*可选*)* 事件日志监控项严重性值
	eventid	*number*	*(*可选*)* 事件日志监控项事件 id
	state	*string*	*(*可选*)* 监控项状态 **0**, *ITEM_STATE_NORMAL* **1**, *ITEM_STATE_NOTSUPPORTED*
	lastlogsize	*number*	*(*可选*)* 最后一次监控项的大小
	mtime	*number*	*(*可选*)* 监控项修改时间
discovery data		*array*	*(*可选*)* 自动发现数据对象数组
	clock	*number*	自动发现数据时间戳
	druleid	*number*	自动发现规则 id
	dcheckid	*number*	自动发现规则数据的发现检查 id 或 null
	type	*number*	自动发现检查类型: **-1** discovery rule data **0**, *SVC_SSH* - SSH service check **1**, *SVC_LDAP* - LDAP service check **2**, *SVC_SMTP* - SMTP service check **3**, *SVC_FTP* - FTP service check **4**, *SVC_HTTP* - HTTP service check **5**, *SVC_POP* - POP service check **6**, *SVC_NNTP* - NNTP service check **7**, *SVC_IMAP* - IMAP service check **8**, *SVC_TCP* - TCP port availability check **9**, *SVC_AGENT* - Zabbix agent **10**, *SVC_SNMPv1* - SNMPv1 agent **11**, *SVC_SNMPv2* - SNMPv2 agent **12**, *SVC_ICMPPING* - ICMP ping **13**, *SVC_SNMPv3* - SNMPv3 agent **14**, *SVC_HTTPS* - HTTPS service check **15**, *SVC_TELNET* - Telnet availability check
	ip	*string*	主机 IP 地址
	dns	*string*	主机 DNS 名称
	port	*number*	*(*可选*)* 端口号
	key_	*string*	*(*可选*)* 用于自动发现检查的监控项键 **9** *SVC_AGENT*
	value	*string*	*(*可选*)* 从服务接收的值，对于大多数服务可以为空
	status	*number*	*(*可选*)* service status: **0**, *DOBJECT_STATUS_UP* - 服务 up **1**, *DOBJECT_STATUS_DOWN* - 服务 dwon
auto registration		*array*	*(*可选*)* 自动注册数据对象数组
	clock	*number*	自动注册数据时间戳
	host	*string*	主机名
	ip	*string*	*(*可选*)* IP
	dns	*string*	*(*可选*)* DNS
	port	*string*	*(*可选*)* 端口
	host_metadata	*string*	*(*可选*)* agent 发送的主机元数据（基于 HostMetadata 或 HostMetadataItem agent 配置参数）
tasks		*array*	*(*可选*)* 任务数组

名称		值类型	描述
	type	*number*	任务类型: **0**, *ZBX_TM_TASK_PROCESS_REMOTE_COMMAND_RESULT* - 远程命令结果
	status	*number*	远程命令执行状态: **0**, *ZBX_TM_REMOTE_COMMAND_COMPLETED* - 远程命令已成功完成 **1**, *ZBX_TM_REMOTE_COMMAND_FAILED* - 远程命令失败
	error	*string*	*(可选)* 错误信息
	parent_taskid	*number*	父级任务 id
more		*number*	*(可选)* 1 - 还有更多历史数据要发送
clock		*number*	*(可选)* 数据传输时间戳 (秒)
ns		*number*	*(可选)* 数据传输时间戳 (秒纳秒
version		*string*	proxy 版本信息 (<major>.<minor>.<build>)
server→proxy:			
response		*string*	请求成功信息 ('success' 或'failed')
tasks		*array*	*(可选)* 任务数组
	type	*number*	任务类型: **1**, *ZBX_TM_TASK_PROCESS_REMOTE_COMMAND* - 远程命令
	clock	*number*	任务创建时间
	ttl	*number*	任务过期时间（以秒为单位）
	command_type	*number*	远程命令类型: **0**, *ZBX_SCRIPT_TYPE_CUSTOM_SCRIPT* - use custom script **1**, *ZBX_SCRIPT_TYPE_IPMI* - use IPMI **2**, *ZBX_SCRIPT_TYPE_SSH* - use SSH **3**, *ZBX_SCRIPT_TYPE_TELNET* - use Telnet **4**, *ZBX_SCRIPT_TYPE_GLOBAL_SCRIPT* - use global script (currently functionally equivalent to custom script)
	command	*string*	要执行的远程命令
	execute_on	*number*	the execution target for custom scripts: **0**, *ZBX_SCRIPT_EXECUTE_ON_AGENT* - execute script on agent **1**, *ZBX_SCRIPT_EXECUTE_ON_SERVER* - execute script on server **2**, *ZBX_SCRIPT_EXECUTE_ON_PROXY* - execute script on proxy
	port	*number*	*(可选)* 远程登录和 ssh 端口
	authtype	*number*	*(可选)* ssh 命令的身份验证类型
	username	*string*	*(可选)* telnet 和 ssh 命令的用户名
	password	*string*	*(可选)* 远程登录和 ssh 命令的密码
	publickey	*string*	*(可选)* ssh 命令的公钥
	privatekey	*string*	*(可选)* ssh 命令的私钥
	parent_taskid	*number*	父级任务 ID
	hostid	*number*	目标 hostid

例子:

server→proxy:

```
{
  "request": "proxy data"
}
```

proxy→server:


```
{
    "session": "12345678901234567890123456789012"
    "interface availability": [
        {
            "interfaceid": 1,
            "available": 1,
            "error": ""
    },
        {
            "interfaceid": 2,
            "available": 2,
            "error": "Get value from agent failed: cannot connect to [[127.0.0.1]:10049]: [111] Connection
    },
        {
            "interfaceid": 3,
            "available": 1,
            "error": ""
    },
        {
            "interfaceid": 4,
            "available": 1,
            "error": ""
    }
    ],
    "history data":[
        {
            "itemid":"12345",
            "clock":1478609647,
            "ns":332510044,
            "value":"52956612",
            "id": 1
        },
        {
            "itemid":"12346",
            "clock":1478609647,
            "ns":330690279,
            "state":1,
            "value":"Cannot find information for this network interface in /proc/net/dev.",
            "id": 2
        }
    ],
    "discovery data":[
        {
            "clock":1478608764,
            "drule":2,
            "dcheck":3,
            "type":12,
            "ip":"10.3.0.10",
            "dns":"vdebian",
            "status":1
        },
        {
            "clock":1478608764,
            "drule":2,
            "dcheck":null,
            "type":-1,
            "ip":"10.3.0.10",
            "dns":"vdebian",
            "status":1
        }
    ],
```

```
    "auto registration":[
        {
            "clock":1478608371,
            "host":"Logger1",
            "ip":"10.3.0.1",
            "dns":"localhost",
            "port":"10050"
        },
        {
            "clock":1478608381,
            "host":"Logger2",
            "ip":"10.3.0.2",
            "dns":"localhost",
            "port":"10050"
        }
    ],
    "tasks":[
        {
            "type": 0,
            "status": 0,
            "parent_taskid": 10
        },
        {
            "type": 0,
            "status": 1,
            "error": "No permissions to execute task.",
            "parent_taskid": 20
        }
    ],
    "version":"5.4.0"
}
```


server→proxy:


```
{
  "response": "success",
  "tasks":[
      {
         "type": 1,
         "clock": 1478608371,
         "ttl": 600,
         "commandtype": 2,
         "command": "restart_service1.sh",
         "execute_on": 2,
         "port": 80,
         "authtype": 0,
         "username": "userA",
         "password": "password1",
         "publickey": "MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCqGKukO1De7zhZj6+H0qtjTkVxwTCpvKe",
         "privatekey": "lsuusFncCzWBQ7RKNUSesmQRMSGkVb1/3j+skZ6UtW+5u09lHNsj6tQ5QCqGKukO1De7zhd",
         "parent_taskid": 10,
         "hostid": 10070
      },
      {
         "type": 1,
         "clock": 1478608381,
         "ttl": 600,
         "commandtype": 1,
         "command": "restart_service2.sh",
         "execute_on": 0,
         "authtype": 0,
         "username": "",
         "password": "",
```

```
            "publickey": "",
            "privatekey": "",
            "parent_taskid": 20,
            "hostid": 10084
        }
    ]
}
```

主动代理

代理检测心跳请求

`proxy heartbeat` 请求由代理发送以报告代理正在运行。每次发送此请求间隔 `HeartbeatFrequency` 秒（代理配置参数）

名称	值类型
proxy→server:	
request	*string*
host	*string*
version	*string*
server→proxy:	
response	*string*

proxy→server:

```
{
   "request": "proxy heartbeat",
   "host": "Proxy #12",
   "version": "5.4.0"
}
```

server→proxy:

```
{
  "response": "success"
}
```

代理配置请求

`proxy config` 请求由代理发送以获取代理配置数据。每次发送此请求间隔 `ConfigFrequency` （代理配置参数）秒。

名称			值类型	描述
proxy→server:				
request			*string*	'proxy config'
host			*string*	代理名称
version			*string*	代理版本 (<major>.<minor>.<build>)
server→proxy:				
request			*string*	'proxy config'
<table>			*object*	包含 <table> 的一个或多个对象
	fields		*array*	数组字段名称
		-	*string*	字段名
	data		*array*	行数组
		-	*array*	列数组
		-	*string,number*	列值，其类型取决于数据库表结构的类型
proxy→server:				
response			*string*	请求成功信息 ('success' 或'failed')

例子:

proxy→server:

```
{
  "request": "proxy config",
  "host": "Proxy #12",
  "version":"5.4.0"
}
```

server→proxy:


```
{
    "globalmacro":{
        "fields":[
            "globalmacroid",
            "macro",
            "value"
        ],
        "data":[
            [
                2,
                "{$SNMP_COMMUNITY}",
                "public"
            ]
        ]
    },
    "hosts":{
        "fields":[
            "hostid",
            "host",
            "status",
            "ipmi_authtype",
            "ipmi_privilege",
            "ipmi_username",
            "ipmi_password",
            "name",
            "tls_connect",
            "tls_accept",
            "tls_issuer",
            "tls_subject",
            "tls_psk_identity",
            "tls_psk"
        ],
        "data":[
            [
                10001,
                "Linux",
                3,
                -1,
                2,
                "",
                "",
                "Linux",
                1,
                1,
                "",
                "",
                "",
                ""
            ],
            [
                10050,
                "Zabbix Agent",
                3,
                -1,
                2,
                "",
                "",
                "Zabbix Agent",
                1,
                1,
                "",
```

```
                "",
                "",
                ""
            ],
            [
                10105,
                "Logger",
                0,
                -1,
                2,
                "",
                "",
                "Logger",
                1,
                1,
                "",
                "",
                "",
                ""
            ]
        ]
    },
    "interface":{
        "fields":[
            "interfaceid",
            "hostid",
            "main",
            "type",
            "useip",
            "ip",
            "dns",
            "port",
            "bulk"
        ],
        "data":[
            [
                2,
                10105,
                1,
                1,
                1,
                "127.0.0.1",
                "",
                "10050",
                1
            ]
        ]
    },
    ...
}
```

proxy→server:

```
{
  "response": "success"
}
```

代理数据请求

“代理数据” 请求由代理发送，以提供主机接口可用性、历史记录、发现和自动注册数据。此请求每 “DataSenderFrequency”（代理配置参数）秒发送一次。请注意，主动代理仍将每秒轮询 Zabbix 服务器以获取远程命令任务（使用空的 “代理数据” 请求）。

名称	值类型	描述
proxy→server:		
request	*string*	'proxy data'
host	*string*	代理名称
session	*string*	数据会话令牌
interface availability	*array*	*(optional)* 接口可用性数据对象数组
	interfaceid *number*	接口标识符
	available *number*	接口可用性 **0**, *INTERFACE_AVAILABLE_UNKNOWN* - unknown **1**, *INTERFACE_AVAILABLE_TRUE* - available **2**, *INTERFACE_AVAILABLE_FALSE* - unavailable
	error *string*	接口错误消息或空字符串
history data	*array*	*(*可选*)* 历史数据对象数组
	itemid *number*	监控项标识符
	clock *number*	监控项值时间戳 (秒)
	ns *number*	监控项值时间戳 (纳秒)
	value *string*	*(*可选*)* 监控项值
	id *number*	值标识符（自增 id，在一个数据会话中是唯一的）
	timestamp *number*	*(*可选*)* 日志类型监控项的时间戳
	source *string*	*(*可选*)* 事件日志监控项数据源
	severity *number*	*(*可选*)* 事件日志监控项严重性值
	eventid *number*	*(*可选*)* 事件日志监控项事件 id
	state *string*	*(*可选*)* 监控项状态 **0**, *ITEM_STATE_NORMAL* **1**, *ITEM_STATE_NOTSUPPORTED*
	lastlogsize *number*	*(*可选*)* 最后一次监控项的大小
	mtime *number*	*(*可选*)* 监控项修改时间
discovery data	*array*	*(*可选*)* 自动发现数据对象数组
	clock *number*	自动发现数据是时间戳
	druleid *number*	自动发现规则 id
	dcheckid *number*	自动发现规则数据的发现检查 id 或 null
	type *number*	自动发现检查类型: **-1** discovery rule data **0**, *SVC_SSH* - SSH service check **1**, *SVC_LDAP* - LDAP service check **2**, *SVC_SMTP* - SMTP service check **3**, *SVC_FTP* - FTP service check **4**, *SVC_HTTP* - HTTP service check **5**, *SVC_POP* - POP service check **6**, *SVC_NNTP* - NNTP service check **7**, *SVC_IMAP* - IMAP service check **8**, *SVC_TCP* - TCP port availability check **9**, *SVC_AGENT* - Zabbix agent **10**, *SVC_SNMPv1* - SNMPv1 agent **11**, *SVC_SNMPv2* - SNMPv2 agent **12**, *SVC_ICMPPING* - ICMP ping **13**, *SVC_SNMPv3* - SNMPv3 agent **14**, *SVC_HTTPS* - HTTPS service check **15**, *SVC_TELNET* - Telnet availability check
	ip *string*	主机 IP 地址
	dns *string*	主机 DNS 名称
	port *number*	*(*可选*)* 端口号
	key_ *string*	*(*可选*)* 用于自动发现检查的监控项键 **9** *SVC_AGENT*
	value *string*	*(*可选*)* 从服务接收的值，对于大多数服务可以为空
	status *number*	*(*可选*)* 服务状态: **0**, *DOBJECT_STATUS_UP* - 服务 Up **1**, *DOBJECT_STATUS_DOWN* - 服务 Down

名称		值类型	描述
auto registration		*array*	*(可选)* 自动注册数据对象数组
	clock	*number*	自动注册数据时间戳
	host	*string*	主机名
	ip	*string*	*(可选)* IP
	dns	*string*	*(可选)* DNS
	port	*string*	*(可选)* 端口号
	host_metadata	*string*	*(可选)* agent 发送的主机元数据（基于 HostMetadata 或 HostMetadataItem agent 配置参数）
tasks		*array*	*(可选)* 任务数组
	type	*number*	任务类型: **0**, *ZBX_TM_TASK_PROCESS_REMOTE_COMMAND_RESULT* - 远程命令结果
	status	*number*	远程命令执行状态: **0**, *ZBX_TM_REMOTE_COMMAND_COMPLETED* - 远程命令已成功完成 **1**, *ZBX_TM_REMOTE_COMMAND_FAILED* - 远程命令失败
	error	*string*	*(可选)* 错误信息
	parent_taskid	*number*	父级任务 id
more		*number*	*(可选)* 1 - 还有更多历史数据要发送
clock		*number*	*(可选)* 数据传输时间戳 (秒)
ns		*number*	*(可选)* 数据传输时间戳 (秒纳秒)
version		*string*	版本信息 (<major>.<minor>.<build>)
server→proxy:			
response		*string*	请求成功信息 (’success’ 或’failed’)
upload		*string*	历史数据（历史、自动注册、主机可用性、网络发现）的上传控制： 启用 - 正常操作 禁用 - 服务器不接受数据（可能是由于内部缓存超过限制）
tasks		*array*	*(可选)* 任务数组
	type	*number*	任务类型: **1**, *ZBX_TM_TASK_PROCESS_REMOTE_COMMAND* - 远程命令
	clock	*number*	任务创建时间
	ttl	*number*	任务过期时间
	command_type	*number*	远程命令类型: **0**, *ZBX_SCRIPT_TYPE_CUSTOM_SCRIPT* - use custom script **1**, *ZBX_SCRIPT_TYPE_IPMI* - use IPMI **2**, *ZBX_SCRIPT_TYPE_SSH* - use SSH **3**, *ZBX_SCRIPT_TYPE_TELNET* - use Telnet **4**, *ZBX_SCRIPT_TYPE_GLOBAL_SCRIPT* - use global script (currently functionally equivalent to custom script)
	command	*string*	要执行的远程命令
	execute_on	*number*	自定义脚本的执行目标: **0**, *ZBX_SCRIPT_EXECUTE_ON_AGENT* - execute script on agent **1**, *ZBX_SCRIPT_EXECUTE_ON_SERVER* - execute script on server **2**, *ZBX_SCRIPT_EXECUTE_ON_PROXY* - execute script on proxy

名称	值类型	描述
port	*number*	(可选) 远程登录和 ssh 端口
authtype	*number*	(可选) ssh 命令的身份验证类型
username	*string*	(可选) telnet 和 ssh 命令的用户名
password	*string*	(可选) 远程登录和 ssh 命令的密码
publickey	*string*	(可选) ssh 命令的公钥
privatekey	*string*	(可选) ssh 命令的私钥
parent_taskid	*number*	父级任务 ID
hostid	*number*	目标 hostid

例子:

proxy→server:


```
{
    "request": "proxy data",
    "host": "Proxy #12",
    "session": "12345678901234567890123456789012",
    "interface availability": [
        {
            "interfaceid": 1,
            "available": 1,
            "error": ""
    },
        {
            "interfaceid": 2,
            "available": 2,
            "error": "Get value from agent failed: cannot connect to [[127.0.0.1]:10049]: [111] Connection
    },
        {
            "interfaceid": 3,
            "available": 1,
            "error": ""
    },
        {
            "interfaceid": 4,
            "available": 1,
            "error": ""
    }
    ],
    "history data":[
        {
            "itemid":"12345",
            "clock":1478609647,
            "ns":332510044,
            "value":"52956612",
            "id": 1
        },
        {
            "itemid":"12346",
            "clock":1478609647,
            "ns":330690279,
            "state":1,
            "value":"Cannot find information for this network interface in /proc/net/dev.",
            "id": 2
        }
    ],
    "discovery data":[
        {
            "clock":1478608764,
            "drule":2,
            "dcheck":3,
            "type":12,
```

```
            "ip":"10.3.0.10",
            "dns":"vdebian",
            "status":1
        },
        {
            "clock":1478608764,
            "drule":2,
            "dcheck":null,
            "type":-1,
            "ip":"10.3.0.10",
            "dns":"vdebian",
            "status":1
        }
    ],
    "auto registration":[
        {
            "clock":1478608371,
            "host":"Logger1",
            "ip":"10.3.0.1",
            "dns":"localhost",
            "port":"10050"
        },
        {
            "clock":1478608381,
            "host":"Logger2",
            "ip":"10.3.0.2",
            "dns":"localhost",
            "port":"10050"
        }
    ],
    "tasks":[
        {
            "type": 2,
            "clock":1478608371,
            "ttl": 600,
            "commandtype": 2,
            "command": "restart_service1.sh",
            "execute_on": 2,
            "port": 80,
            "authtype": 0,
            "username": "userA",
            "password": "password1",
            "publickey": "MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCqGKukO1De7zhZj6+H0qtjTkVxwTCpvKe",
            "privatekey": "lsuusFncCzWBQ7RKNUSesmQRMSGkVb1/3j+skZ6UtW+5u09lHNsj6tQ5QCqGKukO1De7zhd",
            "parent_taskid": 10,
            "hostid": 10070
        },
        {
            "type": 2,
            "clock":1478608381,
            "ttl": 600,
            "commandtype": 1,
            "command": "restart_service2.sh",
            "execute_on": 0,
            "authtype": 0,
            "username": "",
            "password": "",
            "publickey": "",
            "privatekey": "",
            "parent_taskid": 20,
            "hostid": 10084
        }
```

```
    ],
    "tasks":[
        {
            "type": 0,
            "status": 0,
            "parent_taskid": 10
        },
        {
            "type": 0,
            "status": 1,
            "error": "No permissions to execute task.",
            "parent_taskid": 20
        }
    ],
    "version":"5.4.0"
}
```

server→proxy:

```
{
  "response": "success",
  "tasks":[
      {
         "type": 1,
         "clock": 1478608371,
         "ttl": 600,
         "commandtype": 2,
         "command": "restart_service1.sh",
         "execute_on": 2,
         "port": 80,
         "authtype": 0,
         "username": "userA",
         "password": "password1",
         "publickey": "MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCqGKukO1De7zhZj6+H0qtjTkVxwTCpvKe",
         "privatekey": "lsuusFncCzWBQ7RKNUSesmQRMSGkVb1/3j+skZ6UtW+5u09lHNsj6tQ5QCqGKukO1De7zhd",
         "parent_taskid": 10,
         "hostid": 10070
      },
      {
         "type": 1,
         "clock": 1478608381,
         "ttl": 600,
         "commandtype": 1,
         "command": "restart_service2.sh",
         "execute_on": 0,
         "authtype": 0,
         "username": "",
         "password": "",
         "publickey": "",
         "privatekey": "",
         "parent_taskid": 20,
         "hostid": 10084
      }
  ]
}
```

2 Zabbix agent 协议

有关详细信息，请参阅Passive and active agent checks.

3 Zabbix agent 2 protocol

概览

本节提供有关信息:

- ·Agent2 -> Server：主动检查请求
- ·Server -> Agent2：主动检查响应
- ·Agent2 -> Server：代理数据请求
- ·Server -> Agent2：代理数据响应

主动检查请求

主动检查请求用于获取 agent 要处理的主动检查。此请求由 agent 在启动时发送，然后以 *RefreshActiveChecks* 间隔发送。

字段	类型	必填	值
request	字符串	是	主动检查
host	字符串	是	主机名。
version	字符串	是	agent 版本：<major>.<minor>。
host_metadata	字符串	没有	配置参数 HostMetadata 或 HostMetadataItem 度量值。
interface	字符串	没有	配置参数 HostInterface 或 HostInterfaceItem 度量值。
ip	字符串	没有	如果设置了配置参数 ListenIP 第一个 IP。
port	编号	没有	配置参数 ListenPort 值（如果已设置）而不是默认监听端口。

示例:

```
{
 · "request": "active checks",
 · "host": "Zabbix server",
 · "version": "6.0",
 · "host_metadata": "mysql,nginx",
 · "hostinterface": "zabbix.server.lan",
 · "ip": "159.168.1.1",
 · "port": 12050
}
```

主动检查响应

在处理主动检查请求后，主动检查响应由服务器发送回 agent。

字段		类型	必填	值
response		字符串	是	成功 | 失败
info		字符串	没有	失败时的错误信息。
data		对象数组	没有	活动检查监控项。
	key	字符串	没有	带有扩展宏的监控项键。
	itemid	数字	没有	监控项标识符。
	delay	字符串	没有	监控项更新间隔。
	lastlogsize	数字	没有	监控项 lastlogsize。
	mtime	数字	没有	监控项 mtime。
regexp		对象数组	没有	全局正则表达式。
	name	字符串	没有	全局正则表达式名称。
	expression	字符串	没有	全局正则表达式。
	expression_type	数字	没有	全局正则表达式类型。
	exp_delimiter	字符串	没有	全局正则表达式分隔符。
	case_sensitive	编号	没有	全局正则表达式区分大小写设置。

例子：

```
{
 · "response": "success",
 · "data": [
 · {
 · "key": "log[/home/zabbix/logs/zabbix_agentd.log]",
 · "itemid": 1234,
```

```
 · "delay": "30s",
 · "lastlogsize": 0,
 · "mtime": 0
 · },
 · {
 · "key": "agent.version",
 · "itemid": 5678,
 · "delay": "10m",
 · "lastlogsize": 0,
 · "mtime": 0
 · }
 · ]
}
```

Agent 数据请求

Agent 数据请求包含收集的监控项值。

字段		类型	必填	值
request		字符串	是	agent 数据
host		字符串	是	主机名。
version		字符串	是	agent 版本：<major>.<minor>。
session		字符串	是	每次启动 agent 时生成的唯一会话标识符。
data		对象数组	是	监控项值。
	id	数字	是	值标识符（用于在出现网络问题时检查重复值的增量计数器）。
	itemid	数字	是	监控项标识符。
	value	字符串	否	物品价值。
	lastlogsize	数字	否	该监控项最后日志大小。
	mtime	数字	否	监控项时间。
	state	数字	否	监控项状态。
	source	字符串	否	事件日志源。
	eventid	数字	否	事件日志 eventid。
	severity	数字	否	事件日志严重性。
	timestamp	数字	否	事件日志时间戳。
	clock	数字	是	时间戳（自纪元以来的秒数）。
	ns	数字	是	时间戳纳秒。

示例：

```
{
 · "request": "agent data",
 · "data": [
 · {
 · "id": 1,
 · "itemid": 5678,
 · "value": "2.4.0",
 · "clock": 1400675595,
 · "ns": 76808644
 · },
 · {
 · "id": 2,
 · "itemid": 1234,
 · "lastlogsize": 112,
 · "value": " 19845:20140621:141708.521 Starting Zabbix Agent [<hostname>]. Zabbix 2.4.0 (revision 50000).
 · "clock": 1400675595,
 · "ns": 77053975
 · }
 · ],
 · "host": "Zabbix server",
 · "version": "6.0",
 · "session": "1234456akdsjhfoui"
}
```

Agent 数据响应

agent 数据响应在处理 agent 数据请求后由服务器发送回 agent。

字段	类型	必填	值
response	字符串	是	成功 | 失败
info	字符串	是	监控项处理结果。

例子：

```
{
 · "response": "success",
 · "info": "processed: 2; failed: 0; total: 2; seconds spent: 0.003534"
}
```

4 Zabbix agent 2 plugin protocol

Zabbix agent 2 protocol is based on code, size and data model.

Code

Type	Size	Comments
Byte	4	Payload type, currently only JSON is supported.

Size

Type	Size	Comments
Byte	4	Size of the current payload in bytes.

Payload data

Type	Size	Comments
Byte	Defined by the *Size* field	JSON formatted data.

Payload data definition

Common data

These parameters are present in all requests/responses:

Name	Type	Comments
id	uint32	For requests - the incrementing identifier used to link requests with responses. Unique within a request direction (i.e. from agent to plugin or from plugin to agent). For responses - ID of the corresponding request.
type	uint32	The request type.

Log request

A request sent by a plugin to write a log message into the agent log file.

direction	plugin → agent
response	no

Parameters specific to log requests:

Name	Type	Comments
severity	uint32	The message severity (log level).
message	string	The message to log.

Example:

```
{"id":0,"type":1,"severity":3,"message":"message"}
```

Register request

A request sent by the agent during the agent startup phase to obtain provided metrics to register a plugin.

direction	agent → plugin
response	yes

Parameters specific to register requests:

Name	Type	Comments
version	string	The protocol version <major>.<minor>

Example:

```
{"id":1,"type":2,"version":"1.0"}
```

Register response

Plugin's response to the register request.

direction	plugin → agent
response	n/a

Parameters specific to register responses:

Name	Type	Comments
name	string	The plugin name.
metrics	array of strings (optional)	The metrics with descriptions as used in the plugin. Returns RegisterMetrics(). Absent if error is returned.
interfaces	uint32 (optional)	The bit mask of plugin's supported interfaces. Absent if error is returned.
error	string (optional)	An error message returned if a plugin cannot be started. Absent, if metrics are returned.

Examples:

```
{"id":2,"type":3,"metrics":["external.test", "External exporter Test."], "interfaces": 4}
```

or

```
{"id":2,"type":3,"error":"error message"}
```

Start request

A request to execute the Start function of the Runner interface.

direction	agent → plugin
response	no

The request doesn't have specific parameters, it only contains common data parameters.

Example:

```
{"id":3,"type":4}
```

Terminate request

A request sent by the agent to shutdown a plugin.

direction	agent → plugin
response	no

The request doesn't have specific parameters, it only contains common data parameters.

Example:

```
{"id":3,"type":5}
```

Export request

A request to execute the Export function of the Exporter interface.

direction	agent → plugin
response	no

Parameters specific to export requests:

Name	Type	Comments
key	string	The plugin key.
parameters	array of strings (optional)	The parameters for Export function.

Example:

```
{"id":4,"type":6,"key":"test.key","parameters":["foo","bar"]}
```

Export response

Response from the Export function of the Exporter interface.

direction	plugin → agent
response	n/a

Parameters specific to export responses:

Name	Type	Comments
value	string (optional)	Response value from the Export function. Absent, if error is returned.
error	string (optional)	Error message if the Export function has not been executed successfully. Absent, if value is returned.

Examples:

```
{"id":5,"type":7,"value":"response"}
```

or

```
{"id":5,"type":7,"error":"error message"}
```

Configure request

A request to execute the *Configure* function of the *Configurator* interface.

direction	agent → plugin
response	n/a

Parameters specific to *Configure* requests:

Name	Type	Comments
global_options	JSON object	JSON object containing global agent configuration options.
private_options	JSON object (optional)	JSON object containing private plugin configuration options, if provided.

Example:

```
{"id":6,"type":8,"global_options":{...},"private_options":{...}}
```

Validate request

A request to execute *Validate* function of the *Configurator* interface.

direction	agent → plugin
response	yes

Parameters specific to *Validate* requests:

Name	Type	Comments
private_options	JSON object (optional)	JSON object containing private plugin configuration options, if provided.

Example:

```
{"id":7,"type":9,"private_options":{...}}
```

Validate response

Response from *Validate* function of *Configurator* interface.

direction	plugin → agent
response	n/a

Parameters specific to *Validate* responses:

Name	Type	Comments
error	string (optional)	An error message returned if the Validate function is not executed successfully. Absent if executed successfully.

Example:

```
{"id":8,"type":10}
```

or

```
{"id":8,"type":10,"error":"error message"}
```

5 Zabbix sender 协议

有关详细信息，请参阅trapper item 。

6 Header 标头

概述

标头存在于 Zabbix 组件之间的所有请求和响应消息中。需要确定消息长度，是否压缩，是否为大数据包。

Zabbix 通信协议每个连接的数据包大小限制为 1GB。1GB 的限制适用于接收到的数据包数据长度和未压缩的数据长度。

将配置发送到 Zabbix 代理时，数据包大小限制增加到 4GB 以允许同步大型配置。当压缩前的数据长度超过 4GB 时，Zabbix 服务器自动开始使用大数据包格式（0x04 标志），将数据包大小限制增加到 16GB。

请注意，虽然大数据包格式可用于发送任何数据，但目前只有 Zabbix 代理配置同步器可以处理大于 1GB 的数据包。

结构

标头由四个字段组成。标头中的所有数字都采用小端格式。

字段	大小	大小（大包）	描述
<PROTOCOL>	4	4	"ZBXD" 或 5A 42 58 44
<FLAGS>	1	1	协议标志： 0x01 - Zabbix 通信协议 0x02 - 压缩 0x04 - 大数据包
<DATALEN>	4	8	数据长度。
<RESERVED>	4	8	使用压缩时（0x02 标志）- 未压缩数据的长度 不使用压缩时- 00 00 00 00

例子

以下是一些代码片段，展示了如何将 Zabbix 协议标头添加到您要发送的数据中，以便获得您应该发送到 Zabbix 的数据包，从而正确解释它。这些代码片段假定数据不大于 1GB，因此不使用大数据包格式。

Python

```
packet = b"ZBXD\1" + struct.pack("<II", len(data), 0) + data
```

or

```
def zbx_create_header(plain_data_size, compressed_data_size=None):
    protocol = b"ZBXD"
    flags = 0x01
    if compressed_data_size is None:
        datalen = plain_data_size
        reserved = 0
    else:
        flags |= 0x02
        datalen = compressed_data_size
        reserved = plain_data_size
    return protocol + struct.pack("<BII", flags, datalen, reserved)

packet = zbx_create_header(len(data)) + data
```

Perl

```
my $packet = "ZBXD\1" . pack("(II)<", length($data), 0) . $data;
```

or

```
sub zbx_create_header($;$)
{
    my $plain_data_size = shift;
    my $compressed_data_size = shift;

    my $protocol = "ZBXD";
    my $flags = 0x01;
    my $datalen;
    my $reserved;
```

```
    if (!defined($compressed_data_size))
    {
        $datalen = $plain_data_size;
        $reserved = 0;
    }
    else
    {
        $flags |= 0x02;
        $datalen = $compressed_data_size;
        $reserved = $plain_data_size;
    }

    return $protocol . chr($flags) . pack("(II)<", $datalen, $reserved);
}

my $packet = zbx_create_header(length($data)) . $data;
```

PHP

```
$packet = "ZBXD\1" . pack("VV", strlen($data), 0) . $data;
```

or

```
function zbx_create_header($plain_data_size, $compressed_data_size = null)
{
    $protocol = "ZBXD";
    $flags = 0x01;
    if (is_null($compressed_data_size))
    {
        $datalen = $plain_data_size;
        $reserved = 0;
    }
    else
    {
        $flags |= 0x02;
        $datalen = $compressed_data_size;
        $reserved = $plain_data_size;
    }
    return $protocol . chr($flags) . pack("VV", $datalen, $reserved);
}

$packet = zbx_create_header(strlen($data)) . $data;
```

Bash

```
datalen=$(printf "%08x" ${#data})
datalen="\\x${datalen:6:2}\\x${datalen:4:2}\\x${datalen:2:2}\\x${datalen:0:2}"
printf "ZBXD\1${datalen}\0\0\0\0%s" "$data"
```

7 实时导出协议

本节以 JSON 格式提供real-time export协议的详细信息：

- 触发器事件
- 监控项值
- 趋势

所有文件的扩展名都为.ndjson 。导出文件的每一行都是一个 JSON 对象。

触发器事件

为问题事件导出以下信息:

字段	类型	描述
clock	number	检测到问题的那一刻的时间秒数（整数部分）

字段			类型	描述
ns			number	要添加到 `clock` 以获得精确问题检测时间的纳秒数。
value			number	1 (总是).
eventid			number	问题事件 ID.
name			string	问题事件名称.
severity			number	问题事件严重性 (0 - 未分类, 1 - 信息, 2 - 警告, 3 - 一般严重, 4 - 严重, 5 - 灾难).
hosts			array	触发器表达式中涉及的主机列表; 数组中应至少有一个元素。
	-		object	
		host	string	主机名.
		name	string	可见主机名
groups			array	触发器表达式中涉及的所有主机的主机组列表; 数组中应至少有一个元素.
	-		string	主机组名称.
tags			array	问题标签列表（可以为空）.
	-		object	
		tag	string	标签名称.
		value	string	标签值 (可以为空).

为恢复事件导出以下信息:

字段	类型	描述
clock	number	问题恢复那一刻的时间秒数（整数部分）.
ns	number	要添加到 `clock` 以获得精确问题恢复时间的纳秒数。
value	number	0 (总是).
eventid	number	恢复事件 ID.
p_eventid	number	问题事件 ID.

例

问题:

```
{"clock":1519304285,"ns":123456789,"value":1,"name":"Either Zabbix agent is unreachable on Host B or polle
```

恢复:

```
{"clock":1519304345,"ns":987654321,"value":0,"eventid":43,"p_eventid":42}
```

问题（多个问题事件生成）:

```
{"clock":1519304286,"ns":123456789,"value":1,"eventid":43,"name":"Either Zabbix agent is unreachable on Ho
```

```
{"clock":1519304286,"ns":123456789,"value":1,"eventid":43,"name":"Either Zabbix agent is unreachable on Ho
```

恢复:

```
{"clock":1519304346,"ns":987654321,"value":0,"eventid":44,"p_eventid":43}
```

```
{"clock":1519304346,"ns":987654321,"value":0,"eventid":44,"p_eventid":42}
```

监控项值

为收集的监控项值导出以下信息:

字段		类型	描述
host		object	监控项主机的主机名.
	host	string	主机名.
	name	string	可见的主机名.
groups		array	监控项主机的主机组列表; 数组中应至少有一个元素
	-	string	主机组名.
itemid		number	监控项 ID.
name		string	可见监控项名称.
clock		number	收集值那一刻的时间秒数（整数部分）.

字段	类型	描述
ns	number	要添加到 `clock` 以获得精确收集值时间的纳秒数。
timestamp (*Log* only)	number	如果不可用，则为 0
source (*Log* only)	string	如果不可用，则为空字符串
severity (*Log* only)	number	如果不可用，则为 0
eventid (*Log* only)	number	如果不可用，则为 0
value	number (用于数字型监控项) or string (用于文本型监控项)	收集的监控项值。
type	number	收集的值类型： 0 - 浮点数, 1 - 字符, 2 - 日志, 3 - 无符号数字, 4 -文本

例

数值（无符号):

```
{"host":{"host":"Host B","name":"Host B visible"},"groups":["Group X","Group Y","Group Z"],"itemid":3,"nam
```

数值（浮点数）:

```
{"host":{"host":"Host B","name":"Host B visible"},"groups":["Group X","Group Y","Group Z"],"itemid":4,"nam
```

字符, 文本:

```
{"host":{"host":"Host B","name":"Host B visible"},"groups":["Group X","Group Y","Group Z"],"itemid":2,"nam
```

日志:

```
{"host":{"host":"Host A","name":"Host A visible"},"groups":["Group X","Group Y","Group Z"],"itemid":1,"nam
```

趋势

导出以下信息以获取计算的趋势值:

Field		Type	Description
host		object	监控项主机的主机名
	host	string	主机名
	name	string	可见的主机名.
groups		array	监控项主机的主机组列表; 数组中应至少有一个元素
	-	string	主机组名.
itemid		number	监控项 ID.
name		string	可见监控项名称.
clock		number	收集值那一刻的时间秒数（整数部分）.
count		number	一小时内收集的值的数量。
min		number	一小时内最小监控项值
avg		number	一小时内平均监控项值
max		number	一小时内最大监控项值
type		number	值类型: 0 - 浮点数, 3 - 无符号数字

例

数值（无符号）:

```
{"host":{"host":"Host B","name":"Host B visible"},"groups":["Group X","Group Y","Group Z"],"itemid":3,"nam
```

数值（浮点数）:

```
{"host":{"host":"Host B","name":"Host B visible"},"groups":["Group X","Group Y","Group Z"],"itemid":4,"nam
```

5 监控项

1 平台支持的监控项

下表列出了不同平台支持的 Zabbix Agent items 监控项:

- 标记为"**X**" 的监控项代表支持, 标记为"**-**" 的监控项代表不支持.
- 如果监控项标记为"**?**", 不确定是否被支持.
- 如果监控项标记为"**r**", 代表该监控项需要 root 权限.
- 中括号 **<**like_this**>** 中的参数为可选项.

Note:
只支持Windows Zabbix agent items不在该表中.

NetBSD											▼▼
OpenBSD										▼▼	
Mac OS X									▼▼		
Tru64								▼▼			
AIX							▼▼				
HP-UX						▼▼					
Solaris					▼▼						
FreeBSD				▼▼							
Linux 2.6 (and later)			▼▼								
Linux 2.4		▼▼									
Windows	▼▼										
▼	1	2	3	4	5	6	7	8	9	10	11
Item ▼											
agent.hostmetadata	X	X	X	X	X	X	X	X	X	X	X
agent.hostname	X	X	X	X	X	X	X	X	X	X	X
agent.ping	X	X	X	X	X	X	X	X	X	X	X
agent.variant	X	X	X	X	X	X	X	X	X	X	X
agent.version	X	X	X	X	X	X	X	X	X	X	X
kernel.maxfiles	-	X	X	X	-	-	-	?	X	X	X
kernel.maxproc	-	-	X	X	X	-	-	?	X	X	X
kernel.openfiles	-	X	X	?	?	?	?	?	?	?	?
log[file,<regexp>,<encoding>,<maxlines>,<mode>,<output>,<maxdelay>,<persistent_dir>]	X[4]	X	X	X	X	X	X	X	X	X	X
persistent_dir	-	X	X	X	X	X	X	X	X	X	X
▲											
log.count[file,<regexp>,<encoding>,<maxproclines>,<mode>,<maxdelay>,<persistent_dir>]	X[4]	X	X	X	X	X	X	X	X	X	X
persistent_dir	-	X	X	X	X	X	X	X	X	X	X
▲											
logrt[file_regexp,<regexp>,<encoding>,<maxlines>,<mode>,<output>,<maxdelay>,<options>,<persistent_dir>]	X[4]	X	X	X	X	X	X	X	X	X	X
persistent_dir	-	X	X	X	X	X	X	X	X	X	X
▲											
logrt.count[file_regexp,<regexp>,<encoding>,<maxproclines>,<mode>,<maxdelay>,<options>,<persistent_dir>]	X[4]	X	X	X	X	X	X	X	X	X	X
persistent_dir	-	X	X	X	X	X	X	X	X	X	X
▲											
modbus.get[endpoint,<slave id>,<function>,<address>,<count>,<type>,<endianness>,<offset>]	X	X	X	-	-	-	-	-	-	-	-
net.dns[<ip>,zone,<type>,<timeout>,<count>]	X	X	X	X	X	X	X	X	X	X	X
net.dns.record[<ip>,zone,<type>,<timeout>,<count>]	X	X	X	X	X	X	X	X	X	X	X
net.if.collisions[if]	X	X	X	X	X	-	X	-	X	X	r
net.if.discovery	X	X	X	X	X	X	X	-	-	X	X
net.if.in[if,<mode>]	X	X	X	X	X	X[1]	X	-	X	X	r
mode bytes ▲ *(default)*	X	X	X	X	X	X[2]	X	X	-	X	X	r
packets	X	X	X	X	X	X	X	-	X	X	r

	errors	X	X	X	X	X [2]	X	X	-	X	X	r
	dropped	X	X	X	X	-	X	-	-	X	X	r
	overruns	-	X	X	-	-	-	-	-	-	-	-
	frame	-	X	X	-	-	-	-	-	-	-	-
	compressed	-	X	X	-	-	-	-	-	-	-	-
	multicast	-	X	X	-	-	-	-	-	-	-	-
net.if.out[if,<mode>]		X	X	X	X	X	X [1]	X	-	X	X	r
mode	bytes	X	X	X	X	X [2]	X	X	-	X	X	r
▲	*(default)*											
	packets	X	X	X	X	X	X	X	-	X	X	r
	errors	X	X	X	X	X [2]	X	X	-	X	X	r
	dropped	X	X	X	-	-	X	-	-	-	-	-
	overruns	-	X	X	-	-	-	-	-	-	-	-
	collision	-	X	X	-	-	-	-	-	-	-	-
	carrier	-	X	X	-	-	-	-	-	-	-	-
	compressed	-	X	X	-	-	-	-	-	-	-	-
net.if.total[if,<mode>]		X	X	X	X	X	X [1]	X	-	X	X	r
mode	bytes	X	X	X	X	X [2]	X	X	-	X	X	r
▲	*(default)*											
	packets	X	X	X	X	X	X	X	-	X	X	r
	errors	X	X	X	X	X [2]	X	X	-	X	X	r
	dropped	X	X	X	-	-	X	-	-	-	-	-
	overruns	-	X	X	-	-	-	-	-	-	-	-
	compressed	-	X	X	-	-	-	-	-	-	-	-
net.tcp.listen[port]		X	X	X	X	X	-	-	-	X	-	-
net.tcp.port[<ip>,port]		X	X	X	X	X	X	X	X	X	X	X
net.tcp.service[service,<ip>,<port>]		X	X	X	X	X	X	X	X	X	X	X
net.tcp.service.perf[service,<ip>,<port>]		X	X	X	X	X	X	X	X	X	X	X
net.tcp.socket.count[<laddr>,<lport>,<raddr>,<rport>,<state>]		X [7]	X	X	X	X	X	-	-	-	-	-
net.udp.listen[port]		X	X	X	X	X	-	-	-	X	-	-
net.udp.service[service,<ip>,<port>]		X	X	X	X	X	X	X	X	X	X	X
net.udp.service.perf[service,<ip>,<port>]		X	X	X	X	X	X	X	X	X	X	X
net.udp.socket.count[<laddr>,<lport>,<raddr>,<rport>,<state>]		X [7]	X	X	X	X	X	-	-	-	-	-
		1	2	3	4	5	6	7	8	9	10	11
proc.cpu.util[<name>,<user>,<type>,<cmdline>,<mode>,<zone>]		X	X	X	X	X [3]	X	X	-	-	-	-
type	total	-	X	X	-	X	-	-	-	-	-	-
▲	*(default)*											
	user	-	X	X	-	X	-	-	-	-	-	-
	system	-	X	X	-	X	-	-	-	-	-	-
mode	avg1	-	X	X	-	X	-	-	-	-	-	-
▲	*(default)*											
	avg5	-	X	X	-	X	-	-	-	-	-	-
	avg15	-	X	X	-	X	-	-	-	-	-	-
zone	current	-	-	-	-	X	-	-	-	-	-	-
▲	*(default)*											
	all	-	-	-	-	X	-	-	-	-	-	-
proc.mem[<name>,<user>,<mode>,<cmdline>,<memtype>]		X	X	X	X	X [3]	X	X	-	X	X	
mode	sum	-	X	X	X	X	-	X	X	-	X	X
▲	*(default)*											
	avg	-	X	X	X	X	-	X	X	-	X	X
	max	-	X	X	X	X	-	X	X	-	X	X
	min	-	X	X	X	X	-	X	X	-	X	X
memtype		-	X	X	X	X	-	X	-	-	-	-
▲												
proc.num[<name>,<user>,<state>,<cmdline>,<zone>]		X	X	X	X	X [3]	X	X	X	-	X	X

state ▲	all *(default)*	-	X	X	X	X	X	X	X	-	X	X
	disk	-	X	X	X	-	-	-	-	-	X	X
	sleep	-	X	X	X	X	X	X	X	-	X	X
	zomb	-	X	X	X	X	X	X	X	-	X	X
	run	-	X	X	X	X	X	X	X	-	X	X
	trace	-	X	X	X	-	-	-	-	-	X	X
cmdline ▲		-	X	X	X	X	X	X	X	-	X	X
zone ▲	current *(default)*	-	-	-	-	X	-	-	-	-	-	-
	all	-	-	-	-	X	-	-	-	-	-	-
sensor[device,sensor,<mode>]		-	X	X	-	-	-	-	-	-	X	-
system.boottime		-	X	X	X	X	-	-	-	X	X	X
system.cpu.discovery		X	X	X	X	X	X	X	X	X	X	X
system.cpu.intr		-	X	X	X	X	-	X	-	-	X	X
system.cpu.load[<cpu>,<mode>]		X	X	X	X	X	X	X	X	X	X	X
cpu ▲	all *(default)*	X	X	X	X	X	X	X	X	X	X	X
	percpu	X	X	X	X	X	X	X	-	X	X	X
mode ▲	avg1 *(default)*	X	X	X	X	X	X	X	X	X	X	X
	avg5	X	X	X	X	X	X	X	X	X	X	X
	avg15	X	X	X	X	X	X	X	X	X	X	X
system.cpu.num[<type>]		X	X	X	X	X	X	X	-	X	X	X
type ▲	online *(default)*	X	X	X	X	X	X	X	-	X	X	X
	max	-	X	X	X	X	-	-	-	X	-	-
system.cpu.switches		-	X	X	X	X	-	X	-	-	X	X
system.cpu.util[<cpu>,<type>,<mode>,<logical_or_physical>]		X	X	X	X	X	X	X	X	-	X	X
type ▲	user *(default)*	-	X	X	X	X	X	X	X	-	X	X
	nice	-	X	X	X	-	X	-	X	-	X	X
	idle	-	X	X	X	X	X	X	X	-	X	X
	system *(default for Windows)*	X	X	X	X	X	X	X	X	-	X	X
	iowait	-	-	X	-	X	-	X	-	-	-	-
	interrupt	-	-	X	X	-	-	-	-	-	X	-
	softirq	-	-	X	-	-	-	-	-	-	-	-
	steal	-	-	X	-	-	-	-	-	-	-	-
	guest	-	-	X	-	-	-	-	-	-	-	-
	guest_nice	-	-	X	-	-	-	-	-	-	-	-
mode ▲	avg1 *(default)*	X	X	X	X	X	X	X	X	-	X	X
	avg5	X	X	X	X	X	X	X	-	-	X	X
	avg15	X	X	X	X	X	X	X	-	-	X	X
logical_or_physical ▲	logical *(default)*	-	-	-	-	-	-	X	-	-	-	-
	physical	-	-	-	-	-	-	X	-	-	-	-
		1	2	3	4	5	6	7	8	9	10	11

system.hostname[<type>,<transform>]		X	X	X	X	X	X	X	X	X	X
system.hw.chassis[<info>]		-	X	-	-	-	-	-	-	-	-
system.hw.cpu[<cpu>,<info>]		X	X	-	-	-	-	-	-	-	-
system.hw.devices[<type>]		X	X	-	-	-	-	-	-	-	-
system.hw.macaddr[<interface>,<format>]		-	X	-	-	-	-	-	-	-	-
system.localtime[<type>]		X	X	X	X	X	X	X	X	X	X
type ▲	utc *(default)*	X	X	X	X	X	X	X	X	X	X
	local	X	X	X	X	X	X	X	X	X	X
system.run[command,<mode>]		X	X	X	X	X	X	X	X	X	X
mode ▲	wait *(default)*	X	X	X	X	X	X	X	X	X	X
	nowait	X	X	X	X	X	X	X	X	X	X
system.stat[resource,<type>]		-	-	-	-	X	-	-	-	-	-
system.sw.arch		X	X	X	X	X	X	X	X	X	X
system.sw.os[<info>]		X	X	-	-	-	-	-	-	-	-
system.sw.packages[<package>,<manager>,<format>]		-	X	-	-	-	-	-	-	-	-
system.swap.in[<device>,<type>]		X	X	-	X	-	-	-	-	X	-
(specifying a device is only supported under Linux)											
type ▲ *(pages will only work if device was not specified)*	count *(default under all except Linux)*	-	X	X	-	X	-	-	-	-	X	-
	sectors	-	X	X	-	-	-	-	-	-	-	-
	pages *(default under Linux)*	-	X	X	-	X	-	-	-	-	X	-
system.swap.out[<device>,<type>]		-	X	-	X	-	-	-	-	X	-
(specifying a device is only supported under Linux)											

type ▲ *(pages will only work if device was not specified)*	count *(default under all except Linux)*	-	X	X	-	X	-	-	-	-	X	-
	sectors	-	X	X	-	-	-	-	-	-	-	-
	pages *(default under Linux)*	-	X	X	-	X	-	-	-	-	X	-
system.swap.size[<device>,<type>] *(specifying a device is only supported under FreeBSD, for other platforms must be empty or "all")*		X	X	X	X	X	-	X	X	-	X	-
type ▲	free *(default)*	X	X	X	X	X	-	X	X	-	X	-
	total	X	X	X	X	X	-	X	X	-	X	-
	used	X	X	X	X	X	-	X	X	-	X	-
	pfree	X	X	X	X	X	-	X	X	-	X	-
	pused	X [6]	X	X	X	X	-	X	X	-	X	-
system.uname		X	X	X	X	X	X	X	X	X	X	X
system.uptime		X	X	X	X	X	-	X	?	X	X	X
system.users.num		-	X	X	X	X	X	X	X	X	X	X
systemd.unit.discovery		X	X	-	-	-	-	-	-	-	-	-
systemd.unit.get		-	X	X	-	-	-	-	-	-	-	-
systemd.unit.info		-	X	X	-	-	-	-	-	-	-	-
		1	2	3	4	5	6	7	8	9	10	11
vfs.dev.discovery		-	X	X	-	-	-	-	-	-	-	-
vfs.dev.read[<device>,<type>,<mode>]		X	X	X	X	X	-	X	-	-	X	-
type ▲	sectors	-	X	X	-	-	-	-	-	-	-	-
	operations *(default for OpenBSD, AIX)*	-	X	X	X	X	-	X	-	-	X	-

	bytes *(default for Solaris)*	-	-	-	X	X	-	X	-	-	X	-
	sps *(default for Linux)*	-	X	X	-	-	-	-	-	-	-	-
	ops	-	X	X	X	-	-	-	-	-	-	-
	bps *(default for FreeBSD)*	-	-	-	X	-	-	-	-	-	-	-
mode ▲ *(compatible only with type in: sps, ops, bps)*	avg1 *(default)*	-	X	X	X	-	-	-	-	-	-	-
	avg5	-	X	X	X	-	-	-	-	-	-	-
	avg15	-	X	X	X	-	-	-	-	-	-	-
vfs.dev.write[<device>,<type>,<mode>]		X	X	X	X	X	-	X	-	-	X	-
type ▲	sectors	-	X	X	-	-	-	-	-	-	-	-
	operations *(default for OpenBSD, AIX)*	-	X	X	X	X	-	X	-	-	X	-
	bytes *(default for Solaris)*	-	-	-	X	X	-	X	-	-	X	-
	sps *(default for Linux)*	-	X	X	-	-	-	-	-	-	-	-
	ops	-	X	X	X	-	-	-	-	-	-	-
	bps *(default for FreeBSD)*	-	-	-	X	-	-	-	-	-	-	-
mode ▲ *(compatible only with type in: sps, ops, bps)*	avg1 *(default)*	-	X	X	X	-	-	-	-	-	-	-
	avg5	-	X	X	X	-	-	-	-	-	-	-

	avg15	-	X	X	X	-	-	-	-	-	-	-
vfs.dir.count[dir,<regex_incl>,<regex_excl>,<types_incl>,<types_excl>,<max_depth>,<min_size>,<max_size>,<min_age		X	X	X	X	X	X	X	X	X	X	X
vfs.dir.get[dir,<regex_incl>,<regex_excl>,<types_incl>,<types_excl>,<max_depth>,<min_size>,<max_size>,<min_age>,		X	X	X	X	X	X	X	X	X	X	X
vfs.dir.size[dir,<regex_incl>,<regex_excl>,<mode>,<max_depth>,<regex_excl_dir>]		?	?	?	?	?	?	?	?	?	?	?
vfs.file.cksum[file,<mode>]		X	X	X	X	X	X	X	X	X	X	X
vfs.file.contents[file,<encoding>]		X	X	X	X	X	X	X	X	X	X	X
vfs.file.exists[file,<types_incl>,<types_excl>]		X	X	X	X	X	X	X	X	X	X	X
vfs.file.get[file]		X	X	X	X	X	X	X	X	X	X	X
vfs.file.md5sum[file]		X	X	X	X	X	X	X	X	X	X	X
vfs.file.owner[file,<ownertype>,<resulttype>]		X	X	X	X	X	X	X	X	X	X	X
vfs.file.permissions[file]		X	X	?	?	?	?	?	?	?	?	?
vfs.file.regexp[file,regexp,<encoding>,<output>]		X	X	X	X	X	X	X	X	X	X	X
vfs.file.regmatch[file,regexp,<encoding>]		X	X	X	X	X	X	X	X	X	X	X
vfs.file.size[file,<mode>]		X	X	X	X	X	X	X	X	X	X	X
		1	2	3	4	5	6	7	8	9	10	11
vfs.file.time[file,<mode>]		X	X	X	X	X	X	X	X	X	X	X
mode	modify	X	X	X	X	X	X	X	X	X	X	X
▲	*(default)*											
	access	X	X	X	X	X	X	X	X	X	X	X
	change	X[5]	X	X	X	X	X	X	X	X	X	X
vfs.fs.discovery		X	X	X	X	X	X	X	-	X	X	X
vfs.fs.get		X	X	X	X	X	X	X	-	X	X	X
vfs.fs.inode[fs,<mode>]		X	X	X	X	X	X	X	X	X	X	X
mode	total	-	X	X	X	X	X	X	X	X	X	X
▲	*(default)*											
	free	-	X	X	X	X	X	X	X	X	X	X
	used	-	X	X	X	X	X	X	X	X	X	X
	pfree	-	X	X	X	X	X	X	X	X	X	X
	pused	-	X	X	X	X	X	X	X	X	X	X
vfs.fs.size[fs,<mode>]		X	X	X	X	X	X	X	X	X	X	X
mode	total	X	X	X	X	X	X	X	X	X	X	X
▲	*(default)*											
	free	X	X	X	X	X	X	X	X	X	X	X
	used	X	X	X	X	X	X	X	X	X	X	X
	pfree	X	X	X	X	X	X	X	X	X	X	X
	pused	X	X	X	X	X	X	X	X	X	X	X
vm.memory.size[<mode>]		X	X	X	X	X	X	X	X	X	X	X
mode	total	X	X	X	X	X	X	X	X	X	X	X
▲	*(default)*											
	active	-	-	-	X	-	X	-	-	X	X	X
	anon	-	-	-	-	-	-	-	-	-	-	X
	buffers	-	X	X	X	-	-	-	-	-	X	X
	cached	X	X	X	X	-	-	X	-	-	X	X
	exec	-	-	-	-	-	-	-	-	-	-	X
	file	-	-	-	-	-	-	-	-	-	-	X
	free	X	X	X	X	X	X	X	X	X	X	X
	inactive	-	-	-	X	-	-	-	-	X	X	X
	pinned	-	-	-	-	-	-	X	-	-	-	-
	shared	-	X	-	X	-	-	-	-	-	X	X
	wired	-	-	-	X	-	-	-	-	X	X	X
	used	X	X	X	X	X	X	X	X	X	X	X
	pused	X	X	X	X	X	X	X	X	X	X	X
	available	X	X	X	X	X	X	X	X	X	X	X
	pavailable	X	X	X	X	X	X	X	X	X	X	X
web.page.get[host,<path>,<port>]		X	X	X	X	X	X	X	X	X	X	X
web.page.perf[host,<path>,<port>]		X	X	X	X	X	X	X	X	X	X	X
web.page.regexp[host,<path>,<port>,regexp,<length>,<output>]		X	X	X	X	X	X	X	X	X	X	X
		1	2	3	4	5	6	7	8	9	10	11

Note:
另请参见vm.memory.size 参数说明

脚注

[1] net.if.in，net.if.out 和 net.if.total 监控项不提供环回接口的统计信息 (e.g. lo0).

[2] 这些监控项的这些值不支持 Solaris 系统上的环回接口（包括 Solaris 10 6/06）作为字节，错误和利用率统计信息不会由内核存储和/或报告。但是，如果您通过 net snmp 监视 Solaris 系统，返回值可能是 net-snmp 携带遗留代码, 但是，如果要通过 net-snmp 监视 Solaris 系统，则可能会返回 net-snmp 携带从 1997 年开始的 cmu-snmp 的旧代码，即在读取接口统计信息字节值之后，返回后分组计数器（它存在于环回接口上）乘以任意值 308。这假设分组的平均长度为 308 个八位字节，这是非常粗略的估计，因为用于环回接口的 Solaris 系统上的 MTU 限制为 8892 字节。

这些值不应该被认为是正确的，更不应该被认为是非常准确的。他们是推测值。Zabbix agent 不会做任何猜测的工作, 但是 net-snmp 会返回这些字段的一个值。

[3] Solaris 系统中, /proc/pid/psinfo 获得的命令行限制为 80 字节而且在进程启动时包含命令行。

[4] Windows 事件日志不支持。

[5] 在 Windows XP 上 vfs.file.time[file,change] 可能是等于 vfs.file.time[file,access]。

[6] 仅 Zabbix agent 2 支持；Zabbix agent 不支持

[7] 仅在 64 位 Windows 上被 Zabbix agent 2 支持；不是由 Zabbix agent 支持。

2 vm.memory.size 参数

概述

本节提供了一些参数详细信息vm.memory.size[<mode>] Agent 监控项。

Parameters

以下参数可用于此监控项：

- **active** - 当前正在使用或最近使用的内存，以及 RAM 中的内存
- **anon** - 与文件无关的内存（不能读取）
- **available** - 可用内存，计算方式不同，具体取决于平台（见下表）
- **buffers** - 缓存文件系统元数据等内容
- **cached** - 缓存各种内容
- **exec** - 可执行代码，通常来自（程序）文件
- **file** - 缓存最近访问的文件的内容
- **free** - 任何实体都可以随时使用的内存
- **inactive** - 标记为未使用的内存
- **pavailable** - “可用” 内存占 “总” 的百分比（计算为 `available/total*100`）
- **pinned** - 与 “wired” 相同
- **pused** - “used” 内存占 “total” 的百分比（计算为 使用/总计 *100)
- **shared** - 可以被多个同时访问的内存
- **slab** - 内核用于缓存数据结构以供自己使用的内存总量
- **total** - 总物理内存
- **used** - 使用的内存，计算方式不同，具体取决于平台（见下表）
- **wired** - 标记为始终保留在 RAM 中的内存。它永远不会移动到磁盘。

Warning:
其中一些参数是特定平台的，并且可能在您的平台上不适用。详细请查看Items supported by platform 。

available 和 **used** 的平台特定计算：

平台	**”available”**	**”used”**
AIX	free + cached	real memory in use
FreeBSD	inactive + cached + free	active + wired + cached
HP UX	free	total - free
Linux<3.14	free + buffers + cached	total - free

平台	”available”	”used”
Linux 3.14+ (在 RHEL 7 上向后移植到 3.10 内核)	/proc/meminfo, see ”MemAvailable” 在 Linux 内核中 documentation 详细介绍. 请注意，free + buffers + cached 不再等于 “可用”，因为并非所有页面缓存都可以被释放并且计算中使用了低水位线.	total - free
NetBSD	inactive + execpages + file + free	total - free
OpenBSD	inactive + free + cached	active + wired
OSX	inactive + free	active + wired
Solaris	free	total - free
Win32	free	total - free

Attention:
vm.memory.size[used] 和 *vm.memory.size[available]* 不一定等于总数。例如，在 FreeBSD 上：
* Active, inactive, wired, cached 内存被认为是使用的，因为他们存储一些有用的信息。
* 同时考虑 inactive, cached, free 的内存可用，因为这些类型的记忆可以立即给予请求更多内存的进程。

所以不活动的内存是同时可以是使用和可用的。正因为如此，监控项 *vm.memory.size[used]* 只用来获得信息，监控项 *vm.memory.size[available]* 在触发器中使用。

参见

1.关于不同操作系统内存计算的详细信息

3 被动和主动 **Agent** 检查

概述

本节提供有关Zabbix agent 执行的被动和主动检查的详细信息。

Zabbix 使用基于 JSON 的通信协议与 Zabbix agent 进行通信。

另请参阅：Zabbix agent 2 (/manual/appendix/protocols/zabbix_agent2) 协议详细信息。

被动检查

被动检查是一个简单的数据请求。Zabbix 服务器或 proxy 请求一些数据 (例如，CPU 负载)，Zabbix agent 将结果发送回服务器。

Server 请求头部和数据长度的定义请参考协议详细信息

```
<item key>
```

Agent 响应

```
<DATA>[\0<ERROR>]
```

在上面，方括号中的部分是可选的，只发送到不受支持的监控项。

例如, 对于支持的监控项:

1. Server 打开一个 TCP 连接
2. Server 发送 **<HEADER><DATALEN>agent.ping**
3. Agent 读取请求并响应 **<HEADER><DATALEN>1**
4. Server 处理数据以获取值, 例如’1’
5. TCP 连接关闭

对于不支持的监控项:

1. Server 打开一个 TCP 连接
2. Server 发送 **<HEADER><DATALEN>vfs.fs.size[/nono]**
3. Agent 读取请求并响应 **<HEADER><DATALEN>ZBX_NOTSUPPORTED\0Cannot obtain filesystem information: [2] No such file or directory**
4. Server 处理数据, 更改监控项状态为不支持并显示指定的错误消息
5. TCP 连接关闭

主动检查

主动检查需要更复杂的处理，agent 必须首先从 server 端检索独立处理监控项的列表。

The servers 主动检查的列表在 agent 配置文件中的'ServerActive' 参数中列出，请求这些检查的频率是由相同配置文件中的'RefreshActiveChecks' 参数设置的。然而，如果刷新主动检查失败，则在 60 秒后重试。

agent 然后定期向服务器发送新值。

> **Note:**
> 如果代理位于防火墙后面，您可能会考虑仅使用主动检查，因为在这种情况下您不需要修改防火墙以允许初始传入连接。

获取监控项列表

Agent 请求

主动检查请求用于获取 agent 要处理的主动检查。此请求由 agent 在启动时发送，然后以 *RefreshActiveChecks* 间隔发送。

```
{
  "request": "active checks",
  "host": "Zabbix server",
  "host_metadata": "mysql,nginx",
  "hostinterface": "zabbix.server.lan",
  "ip": "159.168.1.1",
  "port": 12050
}
```

字段	类型	必填	值
request	*string*	是	主动检查
host	*string*	是	主机名。
host_metadata	*string*	否	配置参数 HostMetadata 或 HostMetadataItem 度量值。
hostinterface	*string*	否	配置参数 HostInterface 或 HostInterfaceItem 度量值。
ip	*string*	否	如果设置了配置参数 ListenIP 第一个 IP。
port	*number*	否	配置参数 ListenPort 值（如果已设置）而不是默认 agent 侦听端口。

Server 响应

在处理主动检查请求之后，主动检查响应由服务器发送回 agent。

```
{
  "response": "success",
  "data": [
    {
      "key": "log[/home/zabbix/logs/zabbix_agentd.log]",
      "key_orig": "log[/home/zabbix/logs/zabbix_agentd.log]",
      "itemid": 1234,
      "delay": "30s",
      "lastlogsize": 0,
      "mtime": 0
    },
    {
      "key": "agent.version",
      "key_orig": "agent.version",
      "itemid": 5678,
      "delay": "10m",
      "lastlogsize": 0,
      "mtime": 0
    }
  ]
}
```

字段		类型	必填	值
response		*string*	是	成功 | 失败
info		*string*	否	失败时的错误信息。
data		*array of objects*	否	主动检查监控项。
	key	*string*	否	带有扩展宏的监控项键。
	key_orig	*string*	否	没有扩展宏的监控项键。
	itemid	*number*	否	监控项标识符。

字段		类型	必填	值
	delay	*string*	否	监控项更新间隔。
	lastlogsize	*number*	否	监控项 lastlogsize。
	mtime	*number*	否	监控项时间。
refresh_unsupported		*number*	否	不支持的监控项刷新间隔。
regexp		*array of objects*	否	全局正则表达式。
	name	*string*	否	全局正则表达式名称。
	expression	*string*	否	全局正则表达式。
	expression_type	*number*	否	全局正则表达式类型。
	exp_delimiter	*string*	否	全局正则表达式分隔符。
	case_sensitive	*number*	否	全局正则表达式区分大小写设置。

服务器必然成功响应。

例如：

1. Agent 打开一个 TCP 连接
2. Agent 索要检查列表
3. 服务器响应监控项列表（监控项键，延时）
4. Agent 解析响应
5. TCP 连接关闭
6. Agent 开始定期收集数据

Attention:
请注意，当使用主动检查时，敏感的配置数据可能会被能够访问 Zabbix server trapper 端口的各方获取。这是可能的，因为任何人都可能伪装成主动 agent 并请求项目配置数据；除非使用加密选项，否则不会进行身份验证。

发送收集的数据

Agent 发送

代理数据请求包含收集的监控项值。

```
{
 "request": "agent data",
 "data": [
 {
 "host": "Zabbix server",
 "key": "agent.version",
 "value": "2.4.0",
 "clock": 1400675595,
 "ns": 76808644
 },
 {
 "host": "Zabbix server",
 "key": "log[/home/zabbix/logs/zabbix_agentd.log]",
 "lastlogsize": 112,
 "value": " 19845:20140621:141708.521 Starting Zabbix Agent [<hostname>]. Zabbix 2.4.0 (revision 50000).
 "clock": 1400675595,
 "ns": 77053975
 }
 ],
 "session": "1234456akdsjhfoui"
}
```

字段		类型	必填	值
request		字符串	是	agent 数据
session		字符串	是	每次启动代理时生成的唯一会话标识符。
data		对象数组	是	监控项值。
	id	数字	是	值标识符（用于在出现网络问题时检查重复值的增量计数器）。
	host	字符串	是	主机名。
	key	字符串	是	监控项键。
	value	字符串	没有	监控项值。

字段		类型	必填	值
	lastlogsize	数字	没有	监控项 lastlogsize。
	mtime	数字	没有	监控项时间。
	state	数字	没有	监控项状态。
	source	字符串	没有	值事件日志源。
	eventid	数字	没有	值事件日志 eventid。
	severity	数字	没有	值事件日志严重性。
	timestamp	数字	没有	值事件日志时间戳。
	clock	数字	是	值时间戳（自纪元以来的秒数）。
	ns	数字	是	值时间戳纳秒。

每个值都分配了一个虚拟 ID。值 ID 是一个简单的递增计数器，在一个数据会话中是唯一的（由会话令牌标识）。此 ID 用于丢弃可能在连接性差的环境中发送的重复值。

服务器响应

agent 数据响应在处理代理数据请求后由服务器发送回 agent。

```
{
 · "response": "success",
 · "info": "processed: 2; failed: 0; total: 2; seconds spent: 0.003534"
}
```

字段	类型	必填	值
response	字符串	是	成功 | 失败
info	字符串	是	监控项处理结果。

·::: noteimportant 如果在服务器上发送某些值失败（例如，因为主机或监控项已被禁用或删除），agent 将不会重试发送这些值。:::

举例:

1. Agent 打开一个 TCP 连接
2. Agent 发送一个值列表
3. Server 处理数据并将状态返回
4. TCP 连接关闭

请注意，在上面的示例中，vfs.fs.size[/nono] 的不支持状态如何由 “state” 值 1 和 “value” 属性中的错误消息指示。

Attention:
服务器端的错误消息将被修剪为 2048 个符号。

较旧的 XML 协议

Note:
Zabbix 将占用 16 MB 的 XML base64 编码的数据, 但单个解码值应该不超过 64kb，否则，在解码时将被截断到 64 KB。

4 采集器监控项

概述

Zabbix server 使用基于 JSON 的通信协议，在trapper item的帮助下从 Zabbix sender 接收数据。

请求和响应消息必须以header and data length开头。

Zabbix 发送请求

```
{
    "request":"sender data",
    "data":[
        {
            "host":"<hostname>",
            "key":"trap",
            "value":"test value"
```

```
        }
    ]
}
```

Zabbix server 响应

```
{
    "response":"success",
    "info":"processed: 1; failed: 0; total: 1; seconds spent: 0.060753"
}
```

Zabbix sender 带有时间戳的请求

Zabbix sender 可以发送一个带有时间戳的请求。

```
{
    "request":"sender data",
    "data":[
        {
            "host":"<hostname>",
            "key":"trap",
            "value":"test value",
            "clock":1516710794,
            "ns":592397170
        },
        {
            "host":"<hostname>",
            "key":"trap",
            "value":"test value",
            "clock":1516710795,
            "ns":192399456
        }
    ],
    "clock":1516712029,
    "ns":873386094
}
```

Zabbix 服务器响应

```
{
    "response":"success",
    "info":"processed: 2; failed: 0; total: 2; seconds spent: 0.060904"
}
```

5 Windows agent 监控项的最低权限级别

概述

使用 agent 监控系统时，一种最佳实践是从安装代理的主机上获取指标。要使用最小权限原则，有必要确定哪些指标是从 agent 那里获得的。

本文档中的表格允许您选择最低权限，保证 Zabbix agent 的正确运行。

如果选择了其他用户才能使 agent 工作，而不是”LocalSystem”，则要使 agent 作为 Windows 服务运行，新用户必须具有 “本地策略 → 用户权限分配” 中的” 作为服务登录” 权限分配” 以及创建、写入和删除 Zabbix 代理日志文件的权利。必须将 Active Directory 用户添加到性能监视器用户组。

> **Note:**
> 基于 agent 的权限处理问题上，需要给出” 技术上可接受的最低要求” 的权限组，并且事先为监控对象提供权限。

Windows 上支持的常用 agent 监控项

监控项 key	用户组	
	推荐的	技术上可接受的最低权限组 (功能有限)
agent.hostname	Guests	Guests

监控项 key	用户组	
agent.ping	Guests	Guests
agent.variant	Guests	Guests
agent.version	Guests	Guests
log	Administrators	Guests
log.count	Administrators	Guests
logrt	Administrators	Guests
logrt.count	Administrators	Guests
net.dns	Guests	Guests
net.dns.record	Guests	Guests
net.if.discovery	Guests	Guests
net.if.in	Guests	Guests
net.if.out	Guests	Guests
net.if.total	Guests	Guests
net.tcp.listen	Guests	Guests
net.tcp.port	Guests	Guests
net.tcp.service	Guests	Guests
net.tcp.service.perf	Guests	Guests
net.udp.service	Guests	Guests
net.udp.service.perf	Guests	Guests
proc.num	Administrators	Guests
system.cpu.discovery	Performance Monitor Users	Performance Monitor Users
system.cpu.load	Performance Monitor Users	Performance Monitor Users
system.cpu.num	Guests	Guests
system.cpu.util	Performance Monitor Users	Performance Monitor Users
system.hostname	Guests	Guests
system.localtime	Guests	Guests
system.run	Administrators	Guests
system.sw.arch	Guests	Guests
system.swap.size	Guests	Guests
system.uname	Guests	Guests
system.uptime	Performance Monitor Users	Performance Monitor Users
vfs.dir.count	Administrators	Guests
vfs.dir.get	Administrators	Guests
vfs.dir.size	Administrators	Guests
vfs.file.cksum	Administrators	Guests
vfs.file.contents	Administrators	Guests
vfs.file.exists	Administrators	Guests
vfs.file.md5sum	Administrators	Guests
vfs.file.regexp	Administrators	Guests
vfs.file.regmatch	Administrators	Guests
vfs.file.size	Administrators	Guests
vfs.file.time	Administrators	Guests
vfs.fs.discovery	Administrators	Guests
vfs.fs.size	Administrators	Guests
vm.memory.size	Guests	Guests
web.page.get	Guests	Guests
web.page.perf	Guests	Guests
web.page.regexp	Guests	Guests
zabbix.stats	Guests	Guests

Windows 特定的监控项键

监控项 key	用户组	
	推荐的	技术上可接受的最低权限组 (功能有限)
eventlog	Event Log Readers	Guests
net.if.list	Guests	Guests
perf_counter	Performance Monitor Users	Performance Monitor Users
proc_info	Administrators	Guests
service.discovery	Guests	Guests
service.info	Guests	Guests

监控项 key	用户组	
services	Guests	Guests
wmi.get	Administrators	Guests
vm.vmemory.size	Guests	Guests

6 返回值的编码

Zabbix server 要求每个返回的文本值都是 UTF8 编码，这涉及每一种类型的检查: zabbix agent、ssh、telnet 等等。

不同的监视系统/设备和检查的返回值中可能有非 ascii 字符。对于这种情况，几乎所有的 zabbix keys 都包含一个额外的 item key 参数-<encoding> 。这个关键参数是可选的，但是如果返回的值不是 UTF8 编码，并且它包含非 ascii 字符，则应该指定它。否则，结果可能是出乎意料的和不可预测的。

在这种情况下，对不同数据库后台的行为描述如下。

MySQL

如果一个值在非 UTF8 编码中包含非 ascii 字符，那么当数据库存储此值时，该字符及该字符后的值将被丢弃。没有警告信息写入 *zabbix_server.log*.
Relevant for at least MySQL version 5.1.61

PostgreSQL

如果一个值在非 UTF8 编码中包含非 ascii 字符—这将导致一个失败的 SQL 查询 (PGRES_FATAL_ERROR: 编码的无效字节序列) 和数据将不会被存储。会向 *zabbix_server.log* 中写入一个适当的警告消息.
Relevant for at least PostgreSQL version 9.1.3

7 大文件支持

大文件支持，通常缩写为 LFS, 这个术语适用于在 32 位操作系统上处理大于 2 GB 的文件的能力。从 Zabbix 2.0 对大文件的支持已经被添加。该变动会影响log file monitoring 和所有vfs.file.* items. 大文件支持依赖于 Zabbix 编译时系统的性能，但是在 32 位 Solaris 上完全禁用，因为它与 procfs 和 swapctl 不兼容。

8 传感器

每个传感器芯片在 sysfs /sys/devices 都有自己的目录。要找到所有的传感器芯片，从/sys/class/hwmon/hwmon* 跟踪设备的符号链接更容易, 这里 * 是个数字 (0,1,2,...)。

对于虚拟设备，传感器读数在 /sys/class/hwmon/hwmon/ 目录, 对于非虚拟设备，传感器读数在 */sys/class/hwmon/hwmon*/device 目录。hwmon* 或 hwmon*/device 目录中一个叫 name 的文件包含该芯片的名称, 它对应于传感器芯片所使用的内核驱动程序的名称。

每个文件只有一个传感器读取值。在上面提到的目录中包含传感器读数的文件的命令常用方案是: <type><number>_<item>, 这里

- **type** - 对于传感器芯片：”in” (电压), ”temp” (温度), ”fan” (风扇), 等，
- **item** - ”input” (测量值), ”max” (高阈值), ”min” (低阈值), 等,
- **number** - 总是用于可以不止一次出现的元素 (经常从 1 开始, 除了电压从 0 开始)，如果文件不引用特定的元素，则它们的名称简单，没有数字

可以通过 **sensor-detect** 和 **sensors** 工具获取主机上可用的传感器信息 (lm-sensors package: http://lm-sensors.org/)。**Sensors-detect** 帮助确定哪些模块对于可用的传感器是必需的。当模块加载 **sensors** 程序时可以用来显示所有传感器芯片的读数。该程序使用的传感器读数的标记可以和常规的命名方案不同 (<type><number>_<item>):

- 如果有一个名为 <type><number>_label 的文件, 那么该文件中的标签会代替 <type><number><item> 名字;
- 如果没有名为 <type><number>_label 的文件, 那么程序会在 /etc/sensors.conf (也许会为/etc/sensors3.conf, 或其他的) 文件中找 name 的替代标签。

这个标签允许用户决定使用什么样的硬件。如果既没有 <type><number>_label 文件，配置文件中也没有 label ，那么硬件的类型可以由分配的名字 (hwmon*/device/name) 决定。zabbix_agent 接受的传感器的实际名称可以通过运行 **sensors** 程序带着 -u 参数 (**sensors -u**).

在 **sensor** 程序中，可用的传感器被总线类型 (ISA 适配器，PCI 适配器，SPI 适配器，虚拟设备，ACPI 接口，HID 适配器) 分开

当 Linux 2.4:

(传感器读数从/proc/sys/dev/sensor 目录获得)

- **device** - 设备名字 (如果使用了 <mode>，则是正则表达式);
- **sensor** - 传感器名字 (如果使用了 <mode>，则是正则表达式);

- **mode** - 可能的值: avg, max, min (如果忽略了这个参数，设备和传感器将逐字处理)。

例子: sensor[w83781d-i2c-0-2d,temp1]

在 Zabbix 1.8.4 之前，使用了 sensor[temp1] 格式。

当 Linux 2.6+:

(传感器读数从/sys/class/hwmon 目录获得)

- **device** - 设备名称 (非正则表达式)。设备名称可以是设备的实际名称 (e.g 0000:00:18.3) 或使用传感器程序获取得的名称 (例如:k8temp-pci-00c3)，这由用户决定使用哪个名称;
- **sensor** - 传感器名称 (非正则表达式);
- **mode** - 可能的值: avg, max, min (如果忽略了这个参数，设备和传感器将逐字处理)。

例如:

sensor[k8temp-pci-00c3,temp, max] 或 sensor[0000:00:18.3,temp1]

sensor[smsc47b397-isa-0880,in, avg] 或 sensor[smsc47b397.2176,in1]

获取传感器的名字

传感器标签，由 sensors 命令打印，不能总是被直接使用，因为标签的命名对于每个传感器芯片供应商来说可能是不同的。例如, sensors 输出可能包含以下几行:

```
$ sensors
in0:          +2.24 V  (min =  +0.00 V, max =  +3.32 V)
Vcore:        +1.15 V  (min =  +0.00 V, max =  +2.99 V)
+3.3V:        +3.30 V  (min =  +2.97 V, max =  +3.63 V)
+12V:        +13.00 V  (min =  +0.00 V, max = +15.94 V)
M/B Temp:     +30.0°C  (low  = -127.0°C, high = +127.0°C)
```

在这些情况下，只有一个标签可以直接使用:

```
$ zabbix_get -s 127.0.0.1 -k sensor[lm85-i2c-0-2e,in0]
2.240000
```

尝试使用其他标签 (像 Vcore 或 +12V) 是不会起作用的。

```
$ zabbix_get -s 127.0.0.1 -k sensor[lm85-i2c-0-2e,Vcore]
ZBX_NOTSUPPORTED
```

为了找到实际的 Zabbix 可以使用它来检索读数的传感器名称，运行 sensors -u 命令。在输出中，可以看到到以下内容:

```
$ sensors -u
...
Vcore:
  in1_input: 1.15
  in1_min: 0.00
  in1_max: 2.99
  in1_alarm: 0.00
...
+12V:
  in4_input: 13.00
  in4_min: 0.00
  in4_max: 15.94
  in4_alarm: 0.00
...
```

所有 Vcore 应该检索 in1，+12V 应该检索 in4.[1]

```
$ zabbix_get -s 127.0.0.1 -k sensor[lm85-i2c-0-2e,in1]
1.301000
```

不止电压 (in), 还有电流 (curr), 温度 (temp) 和风扇转速 (fan) 的读数都可以被 Zabbix 检索到。

9 proc.mem 监控项中内存参数类型

概述

Linux, AIX, FreeBSD 和 Solaris 都支持 **memtype** 参数。

'memtype' 参数的三个常用值 pmem, rss 和 vsize 在所有系统中都适用。另外, 在一些系统中只支持该系统下的'memtype' 值。

AIX

请参阅表中 AIX 上 "memtype" 参数支持的值。

支持值	说明	procentry64 结构中来源	尽量兼容
vsize [1]	虚拟内存大小	pi_size	
pmem	实际内存百分比	pi_prm	ps -o pmem
rss	常驻集大小	pi_trss + pi_drss	ps -o rssize
size	进程大小（代码 + 数据）	pi_dvm	"ps gvw"SIZE column
dsize	数据大小	pi_dsize	
tsize	文本（代码）大小	pi_tsize	"ps gvw" TSIZ column
sdsize	共享库的数据大小	pi_sdsize	
drss	数据常驻集大小	pi_drss	
trss	文本常驻集大小	pi_trss	

Notes for AIX:

1. When choosing parameters for proc.mem[] item key on AIX, try to specify narrow process selection criteria. Otherwise there is a risk of getting unwanted processes counted into proc.mem[] result.

Example:

```
\$ zabbix_agentd -t proc.mem[,,,NonExistingProcess,rss]
proc.mem[,,,NonExistingProcess,rss]           [u|2879488]
```

This example shows how specifying only command line (regular expression to match) parameter results in Zabbix agent self-accounting - probably not what you want.

2. Do not use "ps -ef" to browse processes - it shows only non-kernel processes. Use "ps -Af" to see all processes which will be seen by Zabbix agent.

3. Let's go through example of 'topasrec' how Zabbix agent proc.mem[] selects processes.

```
\$ ps -Af | grep topasrec
root 10747984        1   0   Mar 16      -  0:00 /usr/bin/topasrec  -L -s 300 -R 1 -r 6 -o /var/perf daily
```

proc.mem[] has arguments:

```
proc.mem[<name>,<user>,<mode>,<cmdline>,<memtype>]
```

The 1st criterion is a process name (argument <name>). In our example Zabbix agent will see it as 'topasrec'. In order to match, you need to either specify 'topasrec' or to leave it empty. The 2nd criterion is a user name (argument <user>). To match, you need to either specify 'root' or to leave it empty. The 3rd criterion used in process selection is an argument <cmdline>. Zabbix agent will see its value as '/usr/bin/topasrec -L -s 300 -R 1 -r 6 -o /var/perf/daily/ -ypersistent=1 -O type=bin -ystart_time=04:08:54,Mar16,2023'. To match, you need to either specify a regular expression which matches this string or to leave it empty.

Arguments <mode> and <memtype> are applied after using the three criteria mentioned above.

FreeBSD

请参见表中 FreeBSD 上的 "memtype" 参数支持的值。

支持的参数值	描述	proentry64 结构中的源代码	兼容
vsize	虚拟内存大小	kp_eproc.e_vm.vm_map.size or ki_size	ps -o vsz
pmem	实际内存的百分比	calculated from rss	ps -o pmem
rss	驻留内存大小	kp_eproc.e_vm.vm_rssize or ki_rssize	ps -o rss
size [1]	进程大小 (代码 + 数据 + 堆栈)	tsize + dsize + ssize	
tsize	文本 (代码) 的大小	kp_eproc.e_vm.vm_tsize or ki_tsize	ps -o tsiz
dsize	数据大小	kp_eproc.e_vm.vm_dsize or ki_dsize	ps -o dsiz
ssize	堆栈大小	kp_eproc.e_vm.vm_ssize or ki_ssize	ps -o ssiz

Linux

请参见表中 Linux 上的 "memtype" 参数支持的值。

支持值	描述	来源/proc/<pid>/status 文件
vsize [1]	虚拟内存大小	VmSize
pmem	实际内存百分比	(VmRSS/total_memory) * 100
rss	驻留内存大小	VmRSS
data	数据段大小	VmData
exe	代码段大小	VmExe
hwm	驻留集峰值大小	VmHWM
lck	锁定内存大小	VmLck
lib	共享库的大小	VmLib
peak	峰值虚拟内存大小	VmPeak
pin	固定页面的大小	VmPin
pte	页表条目的大小	VmPTE
size	进程代码 + 数据 + 堆栈段的大小	VmExe + VmData + VmStk
stk	堆栈段大小	VmStk
swap	使用的交换空间大小	VmSwap

Linux 上注意事项:

1. 一些旧版本 Linux 内核并不是支持所有'memtype' 值的。例如, Linux 内核版本 2.4 就不支持 `hwm`, `pin`, `peak`, `pte` 和 `swap` 等值。
2. 我们发现 Zabbix agent 主动检查进程参数 `proc.mem[...,...,...,...,data]` 显示的值比 agent 的 /proc/<pid>/status 文件中 `VmData` 行的值大 4 kB。在 agent 自我监控管理时，agent 的数据碎片增长率 4 kB ，然后又返回到先前的值。

Solaris

请参见表中的 Solaris 上的 “memtype” 参数所支持的值。

支持值	描述	psinfo 结构中的源代码	兼容
vsize [1]	过程映像的大小	pr_size	ps -o vsz
pmem	实际内存百分比	pr_pctmem	ps -o pmem
rss	驻留集大小 可能被低估了 - 请参阅 “man ps” 中的 rss 描述。	pr_rssize	ps -o rss

注释批注

[1] 默认值

10 在 **proc.mem** 和 **proc.num** 监控项中选择进程的注意事项

修改其命令行的进程

一些程序使用修改它们的命令行作为显示当前活动的方法。用户可以通过运行 `ps` 和 `top` 命令来查看活动。这些程序的例子包括 *PostgreSQL*, *Sendmail*, *Zabbix*.

让我们来看一个 Linux 的例子，假设我们想要监视许多 Zabbix agent 进程。

ps 命令显示的进程如下

```
$ ps -fu zabbix
UID        PID  PPID  C STIME TTY          TIME CMD
...
zabbix    6318     1  0 12:01 ?        00:00:00 sbin/zabbix_agentd -c /home/zabbix/ZBXNEXT-1078/zabbix_age
zabbix    6319  6318  0 12:01 ?        00:00:01 sbin/zabbix_agentd: collector [idle 1 sec]
zabbix    6320  6318  0 12:01 ?        00:00:00 sbin/zabbix_agentd: listener #1 [waiting for connection]
zabbix    6321  6318  0 12:01 ?        00:00:00 sbin/zabbix_agentd: listener #2 [waiting for connection]
zabbix    6322  6318  0 12:01 ?        00:00:00 sbin/zabbix_agentd: listener #3 [waiting for connection]
zabbix    6323  6318  0 12:01 ?        00:00:00 sbin/zabbix_agentd: active checks #1 [idle 1 sec]
...
```

通过名称和用户选择进程来完成任务:

```
$ zabbix_get -s localhost -k 'proc.num[zabbix_agentd,zabbix]'
6
```

现在让我们将 `zabbix_agentd` 重命名为 `zabbix_agentd_30` 并重新启动它。

ps 现在显示为

```
$ ps -fu zabbix
UID        PID  PPID  C STIME TTY          TIME CMD
...
zabbix    6715     1  0 12:53 ?        00:00:00 sbin/zabbix_agentd_30 -c /home/zabbix/ZBXNEXT-1078/zabbix_
zabbix    6716  6715  0 12:53 ?        00:00:00 sbin/zabbix_agentd_30: collector [idle 1 sec]
zabbix    6717  6715  0 12:53 ?        00:00:00 sbin/zabbix_agentd_30: listener #1 [waiting for connection
zabbix    6718  6715  0 12:53 ?        00:00:00 sbin/zabbix_agentd_30: listener #2 [waiting for connection
zabbix    6719  6715  0 12:53 ?        00:00:00 sbin/zabbix_agentd_30: listener #3 [waiting for connection
zabbix    6720  6715  0 12:53 ?        00:00:00 sbin/zabbix_agentd_30: active checks #1 [idle 1 sec]
...
```

现在根据名称和用户选择进程会产生不正确的结果:

```
$ zabbix_get -s localhost -k 'proc.num[zabbix_agentd_30,zabbix]'
1
```

为什么将可执行文件重命名为更长的名称会导致完全不同的结果?

Zabbix agent 启动时检查进程名字，/proc/<pid>/status 文件是打开的并且检查 Name 行。我们的例子中 Name 行如下:

```
$ grep Name /proc/{6715,6716,6717,6718,6719,6720}/status
/proc/6715/status:Name:   zabbix_agentd_3
/proc/6716/status:Name:   zabbix_agentd_3
/proc/6717/status:Name:   zabbix_agentd_3
/proc/6718/status:Name:   zabbix_agentd_3
/proc/6719/status:Name:   zabbix_agentd_3
/proc/6720/status:Name:   zabbix_agentd_3
```

status 文件中的进程名会被截断为 15 个字符。

ps 命令会产生相似的结果:

```
$ ps -u zabbix
  PID TTY          TIME CMD
...
 6715 ?        00:00:00 zabbix_agentd_3
 6716 ?        00:00:01 zabbix_agentd_3
 6717 ?        00:00:00 zabbix_agentd_3
 6718 ?        00:00:00 zabbix_agentd_3
 6719 ?        00:00:00 zabbix_agentd_3
 6720 ?        00:00:00 zabbix_agentd_3
 ...
```

显然, 跟我们的 proc.num[] name 参数值 zabbix_agentd_30 并不一样。Zabbix agent 从 status 文件中匹配进程名失败后，会转到 /proc/<pid>/cmdline 文件。

agent 如何看待 “cmdline” 文件，可以通过运行一个命令来说明

```
$ for i in 6715 6716 6717 6718 6719 6720; do cat /proc/$i/cmdline | awk '{gsub(/\x0/,"<NUL>"); print};'; d
sbin/zabbix_agentd_30<NUL>-c<NUL>/home/zabbix/ZBXNEXT-1078/zabbix_agentd.conf<NUL>
sbin/zabbix_agentd_30: collector [idle 1 sec]<NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><
sbin/zabbix_agentd_30: listener #1 [waiting for connection]<NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><N
sbin/zabbix_agentd_30: listener #2 [waiting for connection]<NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><N
sbin/zabbix_agentd_30: listener #3 [waiting for connection]<NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><N
sbin/zabbix_agentd_30: active checks #1 [idle 1 sec]<NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL
```

/proc/<pid>/cmdline 文件包含在 *C* 语言中用于终止字符的隐藏的、不可显示的空字符。这个例子中空字符以”<NUL>” 形式出现。

Zabbix agent 检查”cmdline” ，得到 zabbix_agentd_30 值，该值匹配我们的 name 参数值 zabbix_agentd_30。因此, 主进程会被监控项 proc.num[zabbix_agentd_30,zabbix] 计数。

当检查下一进程时, agent 从 cmdline 文件中得到 zabbix_agentd_30: collector [idle 1 sec] ，但不匹配 name 参数值 zabbix_agentd_30。所以，只有不改变命令行的主进程被计数，其他的 agent 进程改变了命令行而被忽略。

这个例子展示了 name 参数不能用在 proc.mem[] 和 proc.num[] 监控项中来选择进程。

cmdline 参数使用恰当的正则表达式会达到一个正确的结果:

```
$ zabbix_get -s localhost -k 'proc.num[,zabbix,,zabbix_agentd_30[ :]]'
6
```

使用 proc.mem[] 和 proc.num[] 项监视修改其命令行的程序时要小心

在将 name 和 cmdline 参数放入 proc.mem[] 和 proc.num[] 项之前，您可能需要使用 proc.num[] 项和 ps 命令测试参数.

Linux 内核线程

proc.mem[] 和 proc.num[] 监控项中的 cmdline 参数不可以使用线程

让我们以内核线程为例:

```
$ ps -ef| grep kthreadd
root         2     0  0 09:33 ?        00:00:00 [kthreadd]
```

可以用进程 名称参数选择:

```
$ zabbix_get -s localhost -k 'proc.num[kthreadd,root]'
1
```

但使用进程 cmdline 参数就不起作用:

```
$ zabbix_get -s localhost -k 'proc.num[,root,,kthreadd]'
0
```

原因是 Zabbix agent 采用 “cmdline” 参数中指定的正则表达式，并将其应用于进程的内容 /proc/<pid>/cmdline. 对于内核线程的 /proc/<pid>/cmdline 文件是空的，所以, cmdline 参数不会匹配到。

proc.mem[] 和 proc.num[] 监控项中的线程计数

Linux 内核线程通过 proc.num[] 监控项计数，但是 proc.mem[] 监控项并不报告内存。例如:

```
$ ps -ef | grep kthreadd
root         2     0  0 09:51 ?        00:00:00 [kthreadd]

$ zabbix_get -s localhost -k 'proc.num[kthreadd]'
1

$ zabbix_get -s localhost -k 'proc.mem[kthreadd]'
ZBX_NOTSUPPORTED: Cannot get amount of "VmSize" memory.
```

但是如果用户线程和内核线程名字相同会发生什么呢? 可能会是这样:

```
$ ps -ef | grep kthreadd
root         2     0  0 09:51 ?        00:00:00 [kthreadd]
zabbix    9611  6133  0 17:58 pts/1    00:00:00 ./kthreadd

$ zabbix_get -s localhost -k 'proc.num[kthreadd]'
2

$ zabbix_get -s localhost -k 'proc.mem[kthreadd]'
4157440
```

proc.num[] 计算内核线程和用户进程。proc.mem[] 只计算用户进程内存，如果为 0 计算内核线程内存。这和上面报告 ZBX_NOTSUPPORTED 的例子不同。

如果程序名恰好匹配其中一个线程，请小心使用 proc.mem[] 和 proc.num[] 监控项。

在给 proc.mem[] 和 proc.num[] 监控项配置参数时, 你应该使用 proc.num[] 监控项和 ps 命令测试该参数。

11 net.tcp.service 和 net.udp.service 检查

net.tcp.service 和 net.udp.service 检查实现的细节在该页详细介绍，不同的服务指定不同的服务参数。

监控项 net.tcp.service 参数

FTP

创建 TCP 连接并期望响应的前 4 个字符为 “220”，然后发送 “QUIT\r\n”。如果未指定，则使用默认端口 21。

http

创建一个 TCP 连接而不期望和发送任何东西。如果未指定，则使用默认端口 80。

https

使用（并且仅适用于）libcurl，不验证证书的真实性，不验证 SSL 证书中的主机名，仅获取响应标头（HEAD 请求）。如果未指定，则使用默认端口 443。

imap

创建 TCP 连接并期望响应的前 4 个字符为 “* OK”，然后发送 “a1 LOGOUT\r\n”。如果未指定，则使用默认端口 143。

ldap

打开与 LDAP 服务器的连接并执行 LDAP 搜索操作，过滤器设置为 (objectClass=*)。期望成功检索第一个条目的第一个属性。如果未指定，则使用默认端口 389。

nntp

创建 TCP 连接并期望响应的前 3 个字符为 “200” 或 “201”，然后发送 “QUIT\r\n”。如果未指定，则使用默认端口 119。

pop

创建 TCP 连接并期望响应的前 3 个字符为 “+OK”，然后发送 “QUIT\r\n”。如果未指定，则使用默认端口 110。

smtp

创建一个 TCP 连接并期望响应的前 3 个字符为 “220”，后跟空格、行尾或破折号。包含破折号的行属于多行响应，响应将被重新读取直到收到没有破折号的行。然后发送 “QUIT\r\n”。如果未指定，则使用默认端口 25。

ssh

创建 TCP 连接。如果连接已经建立，双方交换一个标识字符串（SSH-major.minor-XXXX），其中 major 和 minor 是协议版本，XXXX 是一个字符串。Zabbix 检查是否找到与规范匹配的字符串，然后在不匹配时发回字符串 “SSH-major.minor-zabbix_agent\r\n” 或 “0\n”。如果未指定，则使用默认端口 22。

TCP

创建一个 TCP 连接而不期望和发送任何东西。与其他检查不同，需要指定端口参数。

telnet

创建一个 TCP 连接并期望登录提示（末尾为 “:”）。如果未指定，则使用默认端口 23。

监控项 net.udp.service 参数

ntp

在 UDP 上发送一个 SNTP 包，并根据 RFC 4330, section 5需要验证响应。如果未指定，则使用默认端口 123。

12 不可达/不可用主机设置

概述

当 agent 检查 (Zabbix, SNMP, IPMI, JMX) 失败并且主机变得不可达时，一些配置参数 定义了 Zabbix server 作何反应。

不可达主机接口

主机接口在检查失败后被视为不可访问（网络错误，超时）由 Zabbix、SNMP、IPMI 或 JMX agent。请注意，Zabbix agent 主动检查不会以任何方式影响接口可用性。

从 **UnreachableDelay** 那一刻起定义了在这种无法访问的情况下使用其中一项（包括 LLD 规则）重新检查接口的频率，并且此类重新检查将由无法访问的轮询器（或用于 IPMI 检查的 IPMI 轮询器）执行。默认情况下，下次 15 秒后再次检查。

在 Zabbix 服务器日志中，不可达性由以下消息指示这些：

```
Zabbix agent item "system.cpu.load[percpu,avg1]" on host "New host" failed: first network error, wait for
Zabbix agent item "system.cpu.load[percpu,avg15]" on host "New host" failed: another network error, wait f
```

请注意，指示失败的确切监控项和监控项类型（Zabbix agent）。

> **Note:**
> 在主机不可达期间，*Timeout* 参数也会影响主机再次被检查的时间。如果 Timeout 是 20 秒，但是 UnreachableDelay 是 30 秒，下一次检查在 50 秒后。

UnreachablePeriod 参数定义了不可达的总时长。UnreachablePeriod 应该比 UnreachableDelay 大几倍，这样在主机变为不可用之前，主机会被检查不止一次。

如果不可达主机再次出现，监控自动恢复正常：

恢复 Zabbix agent 对主机”New host” 的检查: 连接恢复

将主机接口状态切换回可用

当不可达期结束时，再次轮询接口，降低使主机接口状态无法访问的监控项的优先级状态。如果不可达接口再次出现，则监控自动恢复正常：

```
resuming Zabbix agent checks on host "New host": connection restored
```

Note:
一旦接口可用，主机不会立即轮询所有监控项有两个原因：
- 它可能会使主机过载。
- 主机接口恢复时间并不总是与监控项计划轮询时间匹配。

因此，在主机接口可用后，监控项不会立即被轮询，但他们将被重新安排到下一次轮询。

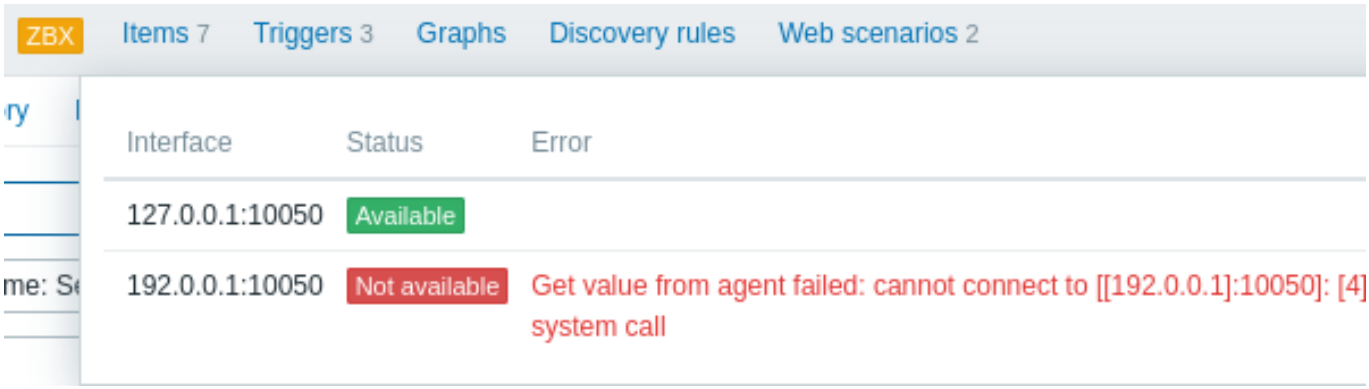
不可用主机状态

主机不可达期结束后主机没有再次出现, 视主机为不可用。

在 server 日志中，不可用是通过类似下面的消息来表示的:

```
temporarily disabling Zabbix agent checks on host "New host": interface unavailable
```

在前端 主机可用性图标由绿色（或灰色）变为红色（注意，在鼠标经过时会提示错误描述）：



UnavailableDelay 参数定义了在主机不可用期间，主机被检查的频率。

默认为 60 秒 (所以此时从上面的日志信息来看，”temporarily disabling” 意味着禁用检查一分钟)。

当主机连接恢复时，监控也会自动恢复正常:

```
enabling Zabbix agent checks on host "New host": interface became available
```

13 远程监控 **Zabbix** 状态

概述

可以通过另一个 Zabbix 实例或第三方工具远程访问 Zabbix 服务器和代理的一些内部指标。这可能很有用，以便支持者/服务提供商可以远程监控他们的客户端 Zabbix 服务器/代理，或者在组织中 Zabbix 不是主要的监控工具，Zabbix 内部指标可以在伞式监控设置中由第三方系统监控。

Zabbix 内部统计信息暴露给新的 “StatsAllowedIP”server/proxy 参数中列出的一组可配置地址。仅接受来自这些地址的请求。

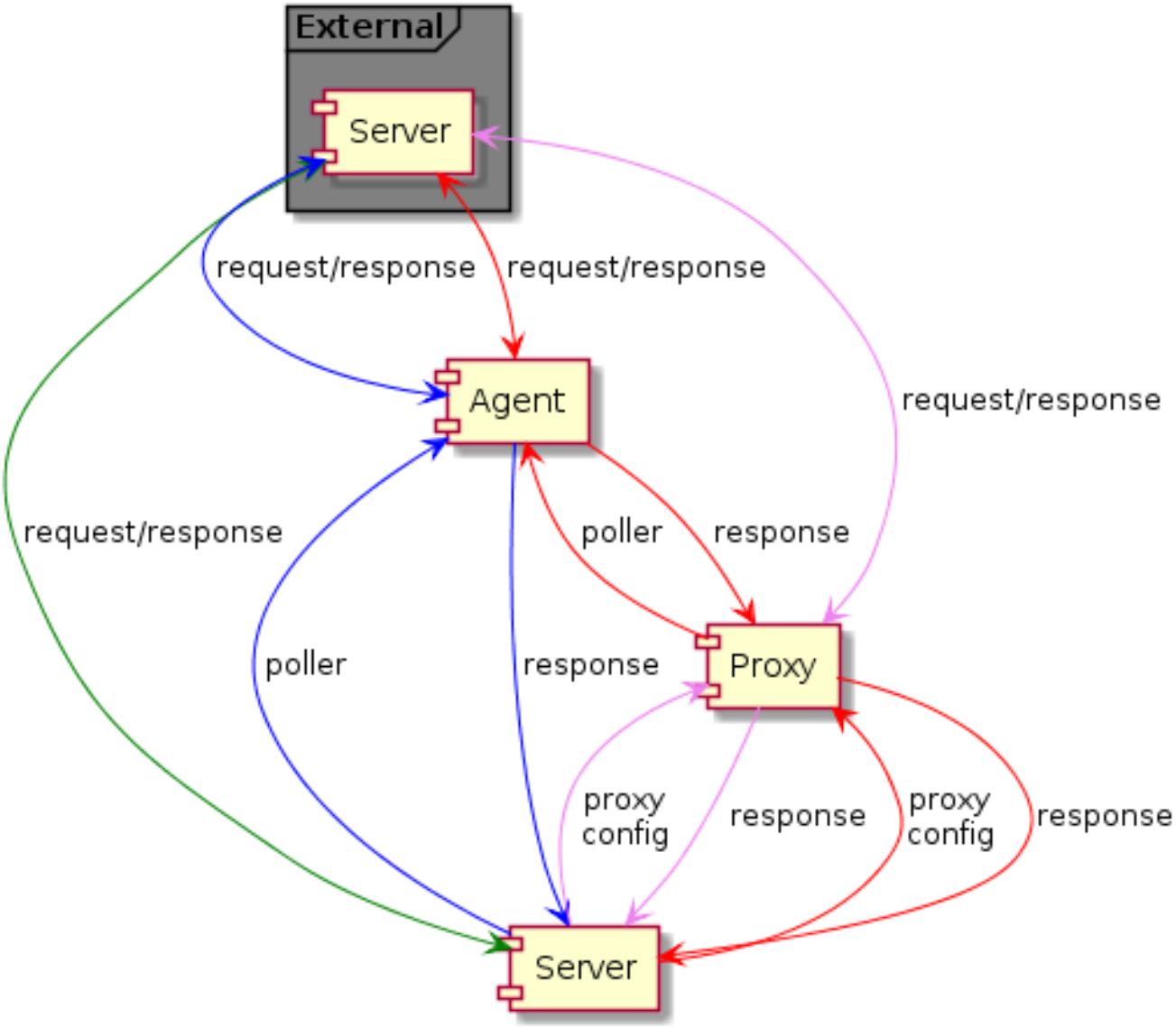
监控项

要在另一个 Zabbix 实例上配置内部统计信息的查询，可以使用两项：

- `zabbix[stats,<ip>,<port>]` 内部监控项 - 用于直接远程查询 Zabbix server/proxy。<ip> 和 <port> 用于标识目标实例。
- `zabbix.stats[<ip>,<port>]` Agent 监控项 - 用于基于代理的 Zabbix server/proxy 的远程查询。<ip> 和 <port> 用于标识目标实例。

另请参见: Internal items, Zabbix agent items

下图根据上下文说明了这两个项的用法。



- - Server → external Zabbix instance (`zabbix[stats,<ip>,<port>]`)
- - Server → proxy → external Zabbix instance (`zabbix[stats,<ip>,<port>]`)
- - Server → agent → external Zabbix instance (`zabbix.stats[<ip>,<port>]`)
- - Server → proxy → agent → external Zabbix instance (`zabbix.stats[<ip>,<port>]`)

要确保目标实例允许外部实例查询它，请在目标实例的 “StatsAllowedIP” 参数中列出外部实例的地址。

内部指标

状态监控项收集统计信息后返回一个 JSON，这是其他依赖监控项从中获取数据的基础。以下内部指标 的用法：

- `zabbix[boottime]`
- `zabbix[hosts]`
- `zabbix[items]`
- `zabbix[items_unsupported]`
- `zabbix[preprocessing_queue]` (server only)
- `zabbix[process,<type>,<mode>,<state>]` (only process type based statistics)
- `zabbix[rcache,<cache>,<mode>]`
- `zabbix[requiredperformance]`
- `zabbix[triggers]` (server only)
- `zabbix[uptime]`
- `zabbix[vcache,buffer,<mode>]` (server only)
- `zabbix[vcache,cache,<parameter>]`
- `zabbix[version]`
- `zabbix[vmware,buffer,<mode>]`

- zabbix[wcache,<cache>,<mode>] ('trends' cache type server only)

模版

Zabbix server 和 Zabbix proxy 远程监控 模板:

- Template App Remote Zabbix server
- Template App Remote Zabbix proxy

请注意，为了使用模板远程监视多个外部实例，每个外部实例监视都需要一个单独的主机。

捕捉器执行过程

Zabbix 实例接收内部指标请求由 trapper 进程处理，trapper 进程验证请求、收集、创建 JSON 数据缓冲区并将准备好的 JSON 发回，例如从服务器:

```
{
  "response": "success",
  "data": {
    "boottime": N,
    "uptime": N,
    "hosts": N,
    "items": N,
    "items_unsupported": N,
    "preprocessing_queue": N,
    "process": {
      "alert manager": {
        "busy": {
          "avg": N,
          "max": N,
          "min": N
        },
        "idle": {
          "avg": N,
          "max": N,
          "min": N
        },
        "count": N
      },
...
    },
    "queue": N,
    "rcache": {
      "total": N,
      "free": N,
      "pfree": N,
      "used": N,
      "pused": N
    },
    "requiredperformance": N,
    "triggers": N,
    "uptime": N,
    "vcache": {
      "buffer": {
        "total": N,
        "free": N,
        "pfree": N,
        "used": N,
        "pused": N
      },
      "cache": {
        "requests": N,
        "hits": N,
        "misses": N,
        "mode": N
      }
```

```
    },
    "vmware": {
      "total": N,
      "free": N,
      "pfree": N,
      "used": N,
      "pused": N
    },
    "version": "N",
    "wcache": {
      "values": {
        "all": N,
        "float": N,
        "uint": N,
        "str": N,
        "log": N,
        "text": N,
        "not supported": N
      },
      "history": {
        "pfree": N,
        "free": N,
        "total": N,
        "used": N,
        "pused": N
      },
      "index": {
        "pfree": N,
        "free": N,
        "total": N,
        "used": N,
        "pused": N
      },
      "trend": {
        "pfree": N,
        "free": N,
        "total": N,
        "used": N,
        "pused": N
      }
    }
  }
}
```

内部队列监控项

另外还有两个监控项可以远程查询另一个 Zabbix 实例上的内部队列统计信息:

- `zabbix[stats,<ip>,<port>,queue,<from>,<to>]` 内部监控项 - 用于将内部队列查询直接发送到 Zabbix server/proxy
- `zabbix.stats[<ip>,<port>,queue,<from>,<to>]` agent 监控项 - 用于将内部队列查询直接发送到 Zabbix server/proxy

参考: Internal items, Zabbix agent items

14 使用 **Zabbix** 配置 **Kerberos** 监控

概述

zabbix 4.4.0 之后版本，Kerberos 身份验证可用于 Zabbix 中的 web 监视和 HTTP 监控项。

这部分 Kerberos 配置描述 Zabbix server 使用’zabbix’ 用户对 `www.example.com` 进行 web 监控

步骤

第 1 步

安装 Kerberos 包。

对于 Debian/Ubuntu：

apt install krb5-user

对于 RHEL：

dnf install krb5-workstation

第 2 步

设置 Kerberos 配置文件 (看 MIT 详细文档)

```
cat /etc/krb5.conf
[libdefaults]
    default_realm = EXAMPLE.COM

#### The following krb5.conf variables are only for MIT Kerberos.
    kdc_timesync = 1
    ccache_type = 4
    forwardable = true
    proxiable = true

[realms]
    EXAMPLE.COM = {
    }

[domain_realm]
    .example.com=EXAMPLE.COM
    example.com=EXAMPLE.COM
```

第 3 步

创建 *zabbix* 用户的 Kerberos ticket。使用 *zabbix* 执行命令：

`kinit zabbix`

> **Attention:**
> 要使用 *zabbix* 用户执行以上命令。如果使用 root 用户将不会通过认证。

第 4 步

创建具有 Kerberos 身份验证类型的 web 方案或 HTTP 代理监控项。

可以选择使用以下 curl 命令进行测试：

`curl -v --negotiate -u : http://example.com`

请注意，对于冗长的 web 监视，有必要更新 Kerberos ticket。Kerberos ticket 到期时间默认为 10 小时。

15 modbus.get 参数

概述

下表显示了 modbus.get[] 监控项 参数的详细信息。

参数

参数	说明	默认值	示例
endpoint	端点的协议和地址，定义为 `protocol://connection_string` 可能的协议值：*rtu*、*ascii*（仅限 Agent 2）、*tcp* 连接字符串格式： with *tcp* - `address:port` with serial line: *rtu*, *ascii* - `port_name:speed:params` where< br>“速度”- 1200、9600 等 “参数”- 数据位（5、6、7 或 8）、奇偶校验（n、e 或 o 表示无/偶数/奇数）、停止位（1 或 2 ）	协议：无 *rtu/ascii* 协议： 端口 _ 名称：无 速度：115200 参数：8n1 *tcp* 协议：< br> 地址：无 端口：502	tcp://192.168.6.1:511 tcp://192.168.6.2 tcp://[::1]:511 tcp://::1 tcp://localhost:511 tcp://localhost rtu://COM1:9600:8n ascii://COM2:1200:7o2 br>rtu://ttyS0:9600 ascii://ttyS1
slave id	设备的 Modbus 地址（1 到 247），参见 MODBUS 消息传递实施指南（第 23 页） tcp 设备（不是 GW）将忽略字段	serial: 1 tcp: 255 (0xFF)	2
function	支持函数的空值或值： 1 - 读取线圈， 2 - 读取离散输入， 3 - 读取保持寄存器， 4 - 读取输入寄存器	空	3
address	第一个注册表、线圈或输入的地址。 如果 “函数” 为空，则 “地址” 应在以下范围内： 线圈 - 00001 - 09999 离散输入 - 10001 - 19999 输入寄存器 - 30001 - 39999 保持寄存器 - 40001 - 49999 如果’ 函数’ 不为空，’ 地址’ 字段将从 0 到 65535 并使用无修改 (PDU)	空函数：00001 非空函数：0	9999
count	将从设备读取的序列 “类型” 的计数，其中： 对于线圈或离散输入，“类型”= 1 位 对于其他情况：(计数 * 类型)/2 = 用于读取的寄存器的实际计数 如果’offset’ 不为 0，则该值将添加到’real count’ ’real count’ 的可接受范围是 1:65535	1	2
type	数据类型： 用于读取线圈和读取离散输入 - 位 用于读取保持寄存器和读取输入寄存器： *int8* - 8 位 *uint8* - 8 位（无符号） *int16* - 16 位 *uint16* - 16 位（无符号） *int32* - 32 位 *uint32* - 32 位（无符号） *float* - 32 位 *uint64* - 64 位（无符号） *double* - 64 位	位 uint16	uint64
endianness	字节序类型： *be* - Big Endian *le* - Little Endian *mbe* - Mid-Big Endian *mle* - Mid-Little Endian 限制： 1 位 - be 8 位 - be,le 16 位 - be,le	be	le

参数	说明	默认值	示例
offset	寄存器个数，从'address' 开始，其结果将被丢弃。 每个寄存器的大小为 16bit（需要支持不支持随机读访问的设备）。	0	4

16 为 **VMware** 创建自定义性能计数器名称

概述

VMware 性能计数器路径具有 `group/counter[rollup]` 格式，其中：

- `group` - 性能计数器组，例如 *cpu*
- `counter` - 性能计数器名称，例如 *usagemhz*
- `rollup` - 性能计数器汇总类型，例如 平均

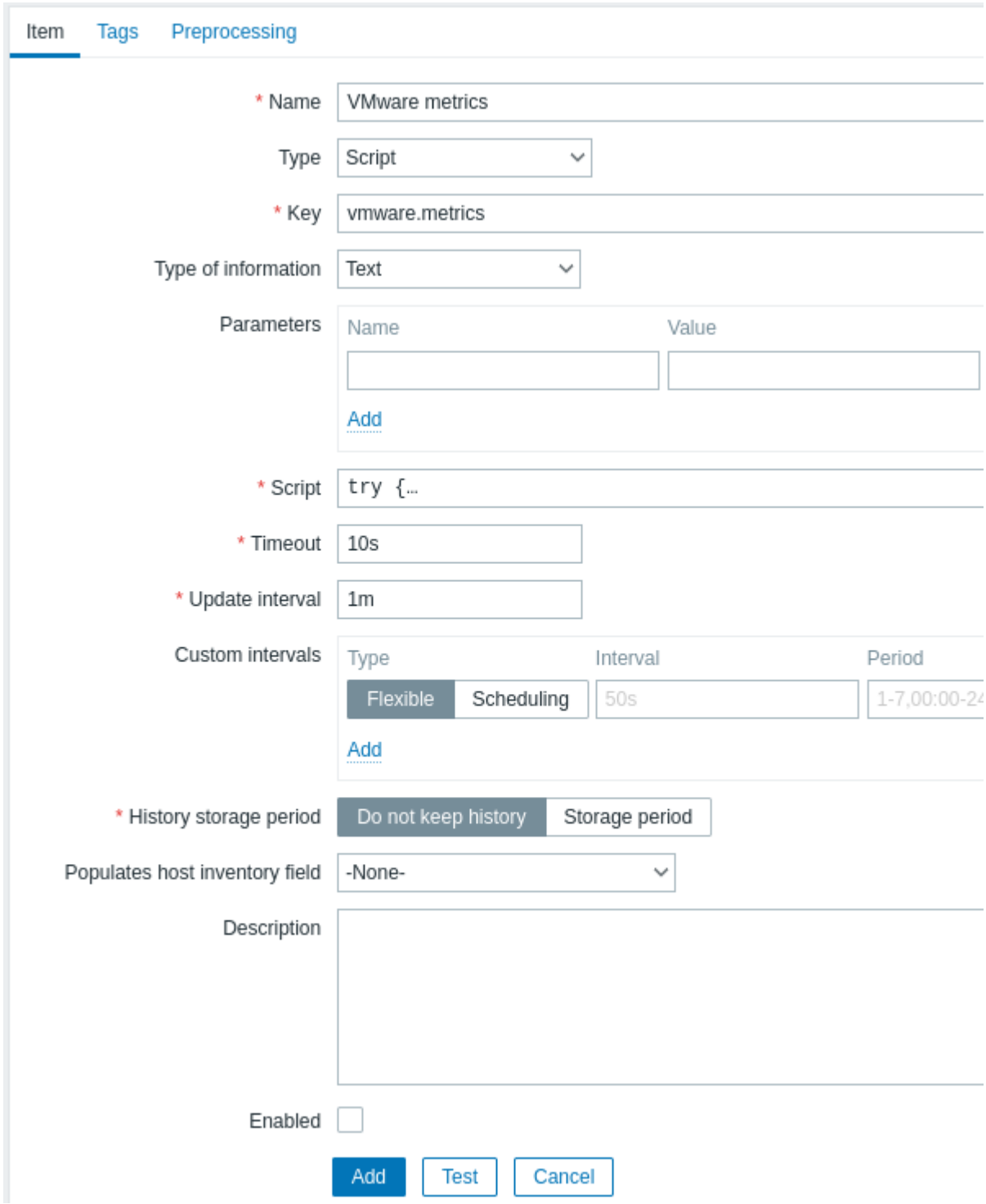
所以上面的例子会给出以下计数器路径：`cpu/usagemhz[average]`

性能计数器组描述、计数器名称和汇总类型可以在 VMware 文档中找到。

可以通过使用 Zabbix 中的脚本监控项来获取内部名称并创建自定义性能计数器名称。

配置

1. 使用以下参数在主要的 VMware 主机（存在 **eventlog[]** 监控项的地方）上创建禁用的脚本监控项：



- *Name*: ·VMware 指标
- *Type*: 脚本
- *Key*: vmware.metrics
- *Type of information*: 文本
- *Script*: 复制并粘贴下面提供的脚本
- *Timeout*: 10
- *History storage period*: 不保留历史数据
- *Enabled*: 未标记

脚本

```
·try { ·Zabbix.log(4, 'vmware metrics script');

·var result, resp, ·req = new HttpRequest(); ·req.addHeader('Content-Type: application/xml'); ·req.addHeader('SOAPAction: "urn:vim25/6.0"');

·login = '<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:vim25">
·<soapenv:Header/>
```

·<soapenv:Body>
·urn:Login
·<urn:_this type=”SessionManager”>SessionManager</urn:_this>
·urn:userName{$VMWARE.USERNAME}</urn:userName>
·urn:password{$VMWARE.PASSWORD}</urn:password>
·</urn:Login>
·</soapenv:Body>
·</soapenv:Envelope>’ ·resp = req.post(”{$VMWARE.URL}”, login); ·if (req.getStatus() != 200) { ·throw ’Response code: ’+req.getStatus(); ·}

·query = ’<soapenv:Envelope xmlns:soapenv=”http://schemas.xmlsoap.org/soap/envelope/” xmlns:urn=”urn:vim25”>
·<soapenv:Header/>
·<soapenv:Body>
·urn:RetrieveProperties
·<urn:_this type=”PropertyCollector”>propertyCollector</urn:_this>
·urn:specSet
·urn:propSet
·urn:typePerformanceManager</urn:type>
·urn:pathSetperfCounter</urn:pathSet>
·</urn:propSet>
·urn:objectSet
·<urn:obj type=”PerformanceManager”>PerfMgr</urn:obj>
·</urn:objectSet>
·</urn:specSet>
·</urn:RetrieveProperties>
·</soapenv:Body>
·</soapenv:Envelope>’ ·resp = req.post(”{$VMWARE.URL}”, query); ·if (req.getStatus() != 200) { ·throw ’Response code: ’+req.getStatus(); ·} ·Zabbix.log(4, ’vmware metrics=’ + resp); ·result = resp;

·logout = ’<soapenv:Envelope xmlns:soapenv=”http://schemas.xmlsoap.org/soap/envelope/” xmlns:urn=”urn:vim25”>
·<soapenv:Header/>
·<soapenv:Body>
·urn:Logout
·<urn:_this type=”SessionManager”>SessionManager</urn:_this>
·</urn:Logout>
·</soapenv:Body>
·</soapenv:Envelope>’

·resp = req.post(”{$VMWARE.URL}”,logout); -if (req.getStatus() != 200) { ·throw ’Response code: ’+req.getStatus(); ·}

·} catch (error) { ·Zabbix.log(4, ’vmware call failed : ’+error); ·result = {}; ·}

·返回结果;

配置监控项后，点击 *Test* 按钮，然后点击 *Get value*。



将收到的 XML 复制到任何 XML 格式化程序并找到所需的指标。

一个指标的 XML 示例：

·<PerfCounterInfo xsi:type=”PerfCounterInfo”> ·<key>6</key> ·<nameInfo> ·<label>Usage in MHz</label> ·<summary>CPU usage in megahertz during the interval</summary> ·<key>usagemhz</key> ·</nameInfo> ·<groupInfo> ·<label>CPU</label> ·<summary>CPU</summary> ·<key>cpu</key> ·</groupInfo> ·<unitInfo> ·<label>MHz</label> ·<summary>Megahertz</summary> ·<key>megaHertz</key> ·</unitInfo> ·<rollupType>average</rollupType> ·<statsType>rate</statsType> ·<level>1</level> ·<perDeviceLevel>3</perDeviceLevel> ·</PerfCounterInfo>

使用 XPath 从收到的 XML 中提取计数器路径。对于上面的示例，XPath 将是：

字段	xPath	值
group	//groupInfo[../key=6]/key	cpu
counter	//nameInfo[../key=6]/key	usagemhz
rollup	//rollupType[../key=6]	average

在这个例子中性能计数器路径是：`cpu/usagemhz[average]`

17 Notes on system.cpu.util items on Windows

system.cpu.util item provides the CPU utilization percentage.

64 logical CPUs (threads) is a hard limit for 32-bit Windows systems and some old 64-bit versions, such as Windows Vista. Systems with <= 64 threads will always have one processor group (see known issues), which means it's OK to use the old performance counter "\Processor(n)% Processor Time". However, for systems with more than 64 threads Windows distributes them evenly across multiple processor groups with maximum 64 threads per single group. Given that "\Processor(n)" doesn't report values for n >= 64 we need to use "\Processor Information(g, n)" where g is a group number and n is a thread number within the group. So, for 72-thread system there will be two groups with 36 threads each and Windows will report counters "\Processor Information(0, n)" with 0 <= n <= 31 and "\Processor Information(1,n)".

When a collector process is started on Zabbix agent for Windows, the following performance counters are initialized and later used for Zabbix agent items with key `system.cpu.util[n]`, `system.cpu.util[total]`.

If logical CPU count is less or equal than 64 and number of CPU groups is equal to 1, then the following counters are initialized, where G is processor group count and N is processor count in group:

```
\Processor(_Total)\% Processor Time
\Processor(0)\% Processor Time
\Processor(1)\% Processor Time
\Processor(2)\% Processor Time
...
\Processor(N-1)\% Processor Time
```

Else:

```
\Processor Information(_Total)\% Processor Time
\Processor Information(0,0)\% Processor Time
\Processor Information(0,1)\% Processor Time
\Processor Information(0,2)\% Processor Time
...
\Processor Information(0,N-1)\% Processor Time
...
\Processor Information(G-1,0)\% Processor Time
\Processor Information(G-1,1)\% Processor Time
\Processor Information(G-1,2)\% Processor Time
...
\Processor Information(G-1,N-1)\% Processor Time
```

6 支持的函数

单击相应的函数组以查看更多详细信息。

函数组	函数
聚合函数	avg, bucket_percentile, count, histogram_quantile, item_count, kurtosis, mad, max, min, skewness, stddevpop, stddevsamp, sum, sumofsquares, varpop, varsamp
Foreach 函数	avg_foreach, bucket_rate_foreach, count_foreach, exists_foreach, last_foreach, max_foreach, min_foreach, sum_foreach

函数组	函数
按位函数	bitand, bitlshift, bitnot, bitor, bitrshift, bitxor
日期和时间函数	date, dayofmonth, dayofweek, now, time
历史函数	change, changecount, count, countunique, find, first, fuzzytime, last, logeventid, logseverity, logsource, monodec, monoinc, nodata, percentile, rate
趋势函数	baselinedev, baselinewma, trendavg, trendcount, trendmax, trendmin, trendstl, trendsum
数学函数	abs, acos, asin, atan, atan2, avg, cbrt, ceil, cos, cosh, cot, degrees, e, exp, expm1, floor, log, log10, max, min, mod, pi, power, radians, rand, round, signum, sin, sinh, sqrt, sum, tan, truncate
运算符函数	between, in
预测函数	forecast, timeleft
字符串函数	ascii, bitlength, bytelength, char, concat, insert, left, length, ltrim, mid, repeat, replace, right, rtrim, trim

这些函数在触发器表达式 and 计算监控项.

Foreach 函数仅支持聚合计算.

1 聚合函数

除非另有说明，这里列出的所有函数都支持在以下情况下使用：

- 触发器表达式
- 计算监控项

聚合函数可以处理以下两种情况的数据：

- 监控项的历史数据，例如，`min(/主机/键,1h)`
- foreach 函数作为唯一的参数，例如，`min(last_foreach(/*/键))`（仅在计算监控项中使用；不能在触发器中使用）

关于函数参数的一些通用说明：

- 函数参数用逗号分隔
- 可选的函数参数（或参数部分）用 < > 表示
- 每个函数的特定参数都与函数一起描述
- `/主机/键`和 `(sec|#num)<:time shift>` 参数永远不需要引用

常用参数

- `/host/key` 是函数的常见强制性第一个参数引用主机监控项历史
- `(sec|#num)<:time shift>` 是常见的第二个参数引用主机监控项历史的函数，其中
  - **sec** - 最大评估周期 in seconds(时间时间戳后缀 可以使用), or
  - **#num** - 最大次数评估范围最新收集的值（如果前面有 # 号）
  - **time shift**（可选）允许移动评估点历史点。关于指定时移更多细节查看。

聚合函数

函数		
描述	函数特定参数	注释
avg (/host/key,(sec|#num)<:time shift>) 监控项在定义的评估期间的平均值。	参见通用参数。	支持的值类型：float, int 支持的foreach 函数：avg_foreach, count_foreach, exists_foreach, last_foreach, max_foreach, min_foreach, sum_foreach 例子： => **avg**(/主机/键,**1h**) → 直到现在的最后一个小时的平均值 => **avg**(/主机/键,**1h:now-1d**) → 从现在起 25 小时至 24 小时之间的一个小时的平均值 => **avg**(/主机/键,**#5**) → 最新五个值的平均值 => **avg**(/主机/键,**#5:now-1d**) → 不包括最后 24 小时接收到的值的最新五个值的平均值 时间偏移在需要将当前平均值与之前某个时间点的平均值进行比较时非常有用。
bucket_percentile (item filter,time period,percentage) 计算直方图的分位数。	监控项过滤 - 参见监控项过滤 时间周期 - 参见时间周期 百分比 - 百分比 (0-100)	仅在计算监控项中支持。 这个函数是 `histogram_quantile(percentage/100, bucket_rate_foreach(监控项过滤, 时间周期, 1))` 的别名
count (func_foreach(item filter,<time period>)) 由 foreach 函数返回的数组中值的数量。	**func_foreach** - 需要计算返回值数量的 foreach 函数（带支持的参数）。参见foreach 函数。	支持的值类型：int 支持的foreach 函数：avg_foreach, count_foreach, exists_foreach, last_foreach, max_foreach, min_foreach, sum_foreach 例子： => **count**(max_foreach(/*/net.if.in[*],1h)) → 在最后一个小时直到现在接收到数据的 net.if.in 监控项的数量 注意，使用 **count()** 与历史相关的 foreach 函数（max_foreach, avg_foreach 等）可能会导致性能问题，而使用 **exists_foreach()**，它只作用于配置数据，不会有这样的影响。
histogram_quantile (quantile,bucket1,value1,bucket2,value2,...)		

函数		
计算直方图的 φ-分位数。	**quantile** - 0 ≤ φ ≤ 1 **bucketN, valueN** - 手动输入的参数对（>=2）或bucket_rate_foreach的响应	仅在计算监控项中支持。 支持的foreach 函数：bucket_rate_foreach 功能上对应于 PromQL 的'histogram_quantile'。 如果最后一个 "无限" 桶 ("+*inf*") 的值等于 0，则返回 -1。 例子： => **histogram_quantile**(0.75,1.0,last(/主机/rate_bucket[1.0]),"+Inf",last(/主机/rate_bucket[Inf])) => **histogram_quantile**(0.5,bucket_rate_foreach(//监控项键,30s))
item_count (item filter) 符合筛选条件的配置中存在的监控项数量。	监控项过滤 - 选择监控项的标准，允许根据主机组、主机、监控项键和标签进行引用。支持使用通配符。更多详细信息见监控项过滤。	仅在计算监控项中支持。 支持的值类型：int 作为 *count(exists_foreach(*监控项过滤*))* 函数的别名。 例子： => **item_count**(*//agent.ping?[*组 *="*主机组 *1"])* → 在" 主机组 *1"* 中具有 agent.ping* 监控项的主机数量
kurtosis (/host/key,(sec|#num)<:time shift>) 在定义的评估期间内收集值的概率分布的" 尾部性"。 参见：Kurtosis	参见通用参数。	支持的值类型：float, int 支持的foreach 函数：last_foreach 例子： => **kurtosis**(/主机/键,**1h**) → 直到现在的最后一个小时的峰态值
mad (/host/key,(sec|#num)<:time shift>) 在定义的评估期间内收集值的中位数绝对偏差。 参见：中位数绝对偏差	参见通用参数。	支持的值类型：float, int 支持的foreach 函数：last_foreach 例子： => **mad**(/主机/键,**1h**) → 直到现在的最后一个小时的中位数绝对偏差
max (/host/key,(sec|#num)<:time shift>) 定义的评估期间内监控项的最高值。	参见通用参数。	支持的值类型：float, int 支持的foreach 函数：avg_foreach, count_foreach, exists_foreach, last_foreach, max_foreach, min_foreach, sum_foreach 例子： => **max**(/主机/键,**1h**) - **min**(/主机/键,**1h**) → 计算直到现在的最后一个小时内最大值和最小值之间的差值（值的增量）
min (/host/key,(sec|#num)<:time shift>)		

函数		
定义的评估期间内监控项的最低值。	参见通用参数。	支持的值类型：float, int 支持的foreach 函数：avg_foreach, count_foreach, exists_foreach, last_foreach, max_foreach, min_foreach, sum_foreach 例子： => **max**(/主机/键,**1h**) - **min**(/主机/键,**1h**) → 计算直到现在的最后一个小时内最大值和最小值之间的差值（值的差分）
skewness (/host/key,(sec|#num)<:time shift>) 在定义的评估期间内收集值的概率分布的不对称性。 另请参阅：Skewness	参见通用参数。	支持的值类型：float, int 支持的foreach 函数：last_foreach 例子： => **skewness**(/主机/键,**1h**) → 直到现在的最后一个小时的偏度值
stddevpop (/host/key,(sec|#num)<:time shift>) 在定义的评估期间内收集值的总体标准差。 另请参阅：标准差	参见通用参数。	支持的值类型：float, int 支持的foreach 函数：last_foreach 例子： => **stddevpop**(/主机/键,**1h**) → 直到现在的最后一个小时的总体标准差
stddevsamp (/host/key,(sec|#num)<:time shift>) 在定义的评估期间内收集值的样本标准差。 另请参阅：标准差	参见常用参数。	支持的值类型：float, int 支持的foreach 函数：last_foreach 至少需要两个数据值才能使此函数正常工作。 例子： => **stddevsamp**(/主机/键,**1h**) → 直到现在的最后一个小时的样本标准差
sum (/host/key,(sec|#num)<:time shift>) 在定义的评估期间内收集值的总和。	参见常用参数。	支持的值类型：float, int 支持的foreach 函数：avg_foreach, count_foreach, exists_foreach, last_foreach, max_foreach, min_foreach, sum_foreach 例子： => **sum**(/主机/键,**1h**) → 直到现在的最后一个小时的值的总和
sumofsquares (/host/key,(sec|#num)<:time shift>) 在定义的评估期间内收集值的平方和。	参见常用参数。	支持的值类型：float, int 支持的foreach 函数：last_foreach 例子： => **sumofsquares**(/主机/键,**1h**) → 直到现在的最后一个小时的平方和
varpop (/host/key,(sec|#num)<:time shift>) 在定义的评估期间内收集值的总体方差。 另请参阅：方差	参见常用参数。	支持的值类型：float, int 支持的foreach 函数：last_foreach 例子： => **varpop**(/主机/键,**1h**) → 直到现在的最后一个小时的总体方差
varsamp (/host/key,(sec|#num)<:time shift>)		

函数		
在定义的评估期间内收集值的样本方差。 另请参阅：方差	参见常用参数。	支持的值类型：float, int 支持的foreach 函数：last_foreach 至少需要两个数据值才能使此函数正常工作。 例子： => **varsamp**(/主机/键,**1h**) → 直到现在的最后一个小时的样本方差

1 Foreach 函数

概述

foreach 函数在聚合计算中使用，以返回由使用的监控项过滤选择的每个监控项的聚合值。返回值是一个数组。

例如，*avg_foreach* 函数将返回一个值数组，其中每个值是选定监控项在指定时间间隔内的平均历史值。

监控项过滤是 foreach 函数使用的语法的一部分。监控项过滤支持使用通配符，因此可以相当灵活地选择所需监控项。

支持的函数

函数	描述
avg_foreach	返回每个监控项的平均值。
bucket_rate_foreach	返回适用于histogram_quantile()函数的配对 (bucket upper bound, rate value)，其中”bucket upper bound” 是由 < 参数编号 > 参数定义的监控项键的参数值。
count_foreach	返回每个监控项的值的数量。
exists_foreach	对于每个启用的监控项返回’1’。
last_foreach	返回每个监控项的最后一个值。
max_foreach	返回每个监控项的最大值。
min_foreach	返回每个监控项的最小值。
sum_foreach	返回每个监控项的值的总和。

参数

Foreach 函数支持两个常用参数 - 监控项过滤器和时间段：

```
foreach_function(item filter,time period)
```

例如：

```
avg_foreach(/*/mysql.qps?[group="MySQL Servers"],5m)
```

一些函数支持附加参数。

监控项过滤语法

监控项过滤：

```
/主机/键[参数]?[条件]
```

由四部分组成，分别是：

- 主机 - 主机名
- 键 - 监控项键（不带参数）
- 参数 - 监控项键参数
- 条件 - 基于主机组和/或监控项标签的条件（作为表达式）

条件表达式内部允许有空格。

通配符使用

- 通配符可用于替换主机名、监控项键或单个监控项键参数。
- 主机或监控项键必须指定而不含通配符。因此 /主机/* 和 /*/键是有效的过滤，但 /*/* 是无效的。
- 通配符不能用于主机名、监控项键、监控项键参数的部分。
- 通配符不匹配超过一个监控项键参数。因此必须为每个参数单独指定通配符（即 键 [abc,*,*]）。

条件表达式

条件表达式支持：

- 操作数：
  - 组 - 主机组
  - 标签 - 监控项标签
  - "< 文本 >" - 字符串常量，使用 \ 转义字符来转义 " 和 \
- 区分大小写的字符串比较运算符：=, <>
- 逻辑运算符：and, or, not
- 使用括号进行分组：( )

字符串常量的引用是必须的。只支持区分大小写的全字符串比较。

Warning:
在过滤中指定标签时（即 标签 ="tagname:value"），冒号 : 用作分隔符。它之后的所有内容都被视为标签值。因此，当前不支持在其中指定包含 : 的标签名。

示例

可以使用复杂的过滤，引用监控项键、主机组和标签，示例如下：

语法示例	描述
/主机/键 [abc,*]	匹配此主机上的类似监控项。
/*/键	匹配任何主机的相同监控项。
/*/键?[组 ="ABC" 且 标签 ="tagname:value"]	匹配来自 ABC 组的任何主机的相同监控项，具有'tagname:value' 标签。
/*/键 [a,*,c]?[(组 ="ABC" 且 标签 ="Tag1") 或 (组 ="DEF" 且 (标签 ="Tag2" 或 标签 ="Tag3:value"))]	匹配来自 ABC 或 DEF 组的任何主机的类似监控项，并具有相应的标签。

所有引用的监控项必须存在并收集数据。只有在启用的主机上启用的监控项才包括在计算中。

Attention:
如果引用的监控项的监控项键被更改，必须手动更新过滤。

指定父主机组包括父组和所有嵌套的主机组及其监控项。

时间段

second 参数允许指定时间段聚合。时间段只能表示为时间，数量不支持值（以 # 为前缀）。

Supported unit symbols 可以用在这个为方便起见参数，例如 "5m"（五分钟）而不是'300s'（300 秒）或'1d'（一天）而不是'86400'（86400 秒）。

如果与 *last_foreach* 一起传递，服务器将忽略时间段函数，因此可以省略：

```
last_foreach(/*/key?[group="host group"])
```

exists_foreach 函数不支持时间段。

附加参数

bucket_rate_foreach 支持第三个可选参数功能：

```
bucket_rate_foreach(item filter,time period,<parameter number>)
```

其中 <parameter number> 是 "bucket" 值在监控项键。例如，如果 myItem[aaa,0.2] 中的 "bucket" 值为'0.2'，则其位置为 2。

<parameter number> 的默认值为 "1"。

依据可用性的行为

下表说明了在主机/项目和历史数据的可用性有限的情况下，每个函数的行为。

函数	禁用主机	有数据的主机不可用	没有数据的主机不可用	禁用监控项	不支持的监控项	数据检索错误 (SQL)
avg_foreach	忽略	返回平均值	忽略	忽略	忽略	忽略
bucket_rate_foreach	忽略	返回桶速率	忽略	忽略	忽略	忽略
count_foreach	忽略	返回计数	0	忽略	忽略	忽略
exists_foreach	忽略	1	1	忽略	1	n/a
last_foreach	忽略	返回最后的值	忽略	忽略	忽略	忽略
max_foreach	忽略	返回最大值	忽略	忽略	忽略	忽略

函数	禁用主机	有数据的主机不可用	没有数据的主机不可用	禁用监控项	不支持的监控项	数据检索错误 (SQL)
min_foreach	忽略	返回最小值	忽略	忽略	忽略	忽略
sum_foreach	忽略	饭回值的和	忽略	忽略	忽略	忽略

如果忽略该项，则不会向聚合中添加任何内容。

2 按位运算函数

此处列出的所有功能均受支持：

- ·触发器表达式
- ·可计算监控项

关于函数参数的一些一般说明：

- ·函数参数用逗号分隔
- ·接受表达式作为参数
- ·可选的函数参数（或参数部分）由·< > 表示

函数		
Description	函数特定参数	注释
bitand (值, 掩码) 监控项值和掩码的 “按位与” 值。	值 - 要检查的值 掩码（强制）- 64 位无符号整数 (0 - 18446744073709551615)	支持的值类型: int 虽然比较是以位方式完成的，但必须提供所有值并以十进制形式返回。例如，检查第 3 位是通过与 4 比较而不是 100 来完成的。 示例： => **bitand**(last(/host/key),**12**)=8 或 **bitand**(last(/host/key),**12**)=4 → 第 3 位或第 4 位设置，但不能同时设置 => **bitand** (last(/host/key),**20**)=16 → 第 3 位未设置，第 5 位已设置。
bitlshift（值，要移动的位） 监控项值向左移位。	值 - 要检查的值 要移位的位数（强制）- 要移位的位数	支持的值类型：整数 尽管比较是以按位方式完成的，但必须提供所有值并以十进制形式返回。例如，检查第 3 位是通过与 4 而不是 100 进行比较来完成的。
bitnot（值） 项目值的 “按位非” 值。	值 - 要检查的值	支持的值类型：int 虽然比较是以按位方式完成的，但所有值必须提供并以十进制形式返回。例如，检查第 3 位是通过与 4 比较而不是 100 来完成的。
bitor（值，掩码） 项目值和掩码的 “按位或” 值。	值 - 要检查的值 掩码（强制）- 64 位无符号整数 (0 - 18446744073709551615)	支持的值类型: int 虽然比较是以位方式完成的，但必须提供所有值并以十进制形式返回。例如，检查第 3 位是通过与 4 而不是 100 进行比较来完成的。
bitrshift（值，要移动的位）		

函数		
项目值的按位右移。	值 - 要检查的值 要移位的位数（强制性）- 要移位的位数	支持的值类型：整数 尽管比较是以按位方式完成的，但必须提供所有值并以十进制形式返回。例如，检查第 3 位是通过与 4 比较而不是 100 来完成的。
bitxor（值，掩码） 项目值和掩码的 “按位异或” 值。	值 - 要检查的值 掩码（强制）- 64 位无符号整数 (0 - 18446744073709551615)	支持的值 types: int 虽然比较是按位方式进行的，但必须提供所有值并以十进制形式返回。例如，检查第 3 位是通过与 4 而不是 100 进行比较来完成的。

4 历史函数

此处列出的所有函数都支持：

- 触发器表达式
- 自定义 key 类型之计算

关于函数参数的一些整体说明：

- 函数参数用逗号分隔
- 可选函数参数（或参数部分）由 < > 表示
- 每个函数都描述了特定功能的参数
- /host/key 和 (sec|#num)<:time shift> 参数绝不能被引用

常用参数

- /host/key 是引用主机监控项历史记录函数的常用强制性首选参数
- (sec|#num)<:time shift> 是引用主机监控项历史记录函数的常用强制性次选参数，其中：
  - **sec** - 以秒为单位的最大评估周期 (可以使用时间后缀)，或者
  - **#num** - 最新收集值最大评估范围(如果前面有井号)
  - **time shift** (可选) 允许将评估点及时移回。参阅有关指定时间偏移更多详细内容

历史函数

函数		
描述	函数特定参数	注释
change (/host/key) 前一个值和最新值之间的差异量。		支持的值类型：float, int, str, text, log 对于字符串返回： 0 - 值相等 1 - 值不同 例子： => **change**(/主机/键)>10 数值差异将被计算，如这些传入的示例值 ('previous' 和'latest' 值 = 差异)： '1' 和'5' = +4 '3' 和'1' = −2 '0' 和'-2.5' = −2.5 另请参阅：abs 用于比较
changecount (/host/key,(sec|#num)<:time shift>,<mode>)		

函数		
在定义的评估期间内相邻值之间变化的次数。 **count** (/host/key,(sec|#num)<:time shift>,<operator>,<pattern>)	参见通用参数。 **mode** （可选；必须使用双引号） 支持的 `modes`： *all* - 计算所有变化（默认） *dec* - 计算减少的次数 *inc* - 计算增加的次数	支持的值类型：float, int, str, text, log 对于非数值类型，将忽略 *mode* 参数。 例子： => **changecount**(/主机/键, 1w) → 直到现在上个星期的值变化次数 => **changecount**(/主机/键, #10, ”inc”) → 在最后 10 个值中相对于相邻值的值增加次数 => **changecount**(/主机/键,24h, ”dec”) → 直到现在最后 24 小时相对于相邻值的值减少次数

函数		
在定义的评估期间内的值的数量。	参见通用参数。 **operator** （可选；必须使用双引号） 支持的 operators： *eq* - 等于（整数，默认） *ne* - 不等于 *gt* - 大于 *ge* - 大于或等于 *lt* - 小于 *le* - 小于或等于 *like* （字符串，默认）- 如果包含模式（区分大小写） *bitand* - 位与 *regexp* - 正则表达式匹配（区分大小写） *iregexp* - 正则表达式匹配（不区分大小写） 注意： *eq*（默认）、*ne*、*gt*、*ge*、*lt*、*le*、*band*、*regexp*、*iregexp* 支持整数监控项 *eq*（默认）、*ne*、*gt*、*ge*、*lt*、*le*、*regexp*、*iregexp* 支持浮点监控项 *like*（默认）、*eq*、*ne*、*regexp*、*iregexp* 支持字符串、文本和日志监控项 **pattern** （可选）- 需要的模式（字符串参数必须使用双引号）	支持的值类型：float, integer, string, text, log 浮点监控项的匹配精度为 2.22e-16；如果数据库未升级，则精度为 0.000001。 使用 *bitand* 作为第三个参数时，第四个 pattern 参数可以指定为两个数字，用’/’ 分隔：**number_to_compare_with/mask**。count() 计算值和 *mask* 的 “位与” 并比较结果与 *number_to_compare_with*。如果 “位与” 的结果等于 *number_to_compare_with*，则计算该值。 如果 *number_to_compare_with* 和 *mask* 相等，则只需要指定 *mask*（不带’/’）。 使用 *regexp* 或 *iregexp* 作为第三个参数时，第四个 pattern 参数可以是普通或全局（以’@’ 开头）正则表达式。在全局正则表达式的情况下，情况敏感性是从全局正则表达式设置中继承的。为了正则表达式匹配的目的，浮点值将始终用’.’ 后的 4 位小数表示。还要注意，对于大数，十进制（存储在数据库中）和二进制（由 Zabbix server 使用）表示之间的差异可能会影响第 4 位小数。 例子： => **count**(/主机/键,**10m**) → 直到现在的最后 10 分钟的值的数量 => **count**(/主机/键,**10m**,”like”,”error”) → 直到现在的最后 10 分钟包含’error’ 的值的数量 => **count**(/主机/键,**10m**,„12) → 直到现在的最后 10 分钟等于’12’ 的值的数量 => **count**(/主机/键,**10m**,”gt”,12) → 直到现在的最后 10 分钟超过’12’ 的值的数量 => **count**(/主机/键,**#10**,”gt”,12) → 直到现在的最后 10 个值中超过’12’ 的值的数量 => **count**(/主机/键,**10m:now-1d**,”gt”,12) → 从现在起 24 小时 10 分钟至 24 小时前超过’12’ 的值的数量 => **count**(/主机/键,**10m**,”bitand”,”6/7”) → 直到现在的最后 10 分钟在 3 个最低有效位具有’110’（二进制）的值的数量 => **count**(/主机/键,**10m:now-1d**) → 从现在起 24 小时 10 分钟至 24 小时前的值的数量
countunique (/host/key,(sec|#num)<:time shift>,<operator>,<pattern>)		

函数		
在定义的评估期间内的唯一值的数量。 **find** (/host/key,<(sec|#num)<:time shift»,<operator>,<pattern>)	参见通用参数。 **operator** （可选；必须使用双引号） 支持的 operators： *eq* - 等于（整数，默认） *ne* - 不等于 *gt* - 大于 *ge* - 大于或等于 *lt* - 小于 *le* - 小于或等于 *like* （默认为字符串、文本、日志）- 如果包含模式（区分大小写） *bitand* - 位与 *regexp* - 正则表达式匹配（区分大小写） *iregexp* - 正则表达式匹配（不区分大小写） 注意： *eq*（默认）、*ne*、*gt*、*ge*、*lt*、*le*、*band*、*regexp*、*iregexp* 支持整数监控项 *eq*（默认）、*ne*、*gt*、*ge*、*lt*、*le*、*regexp*、*iregexp* 支持浮点监控项 *like*（默认）、*eq*、*ne*、*regexp*、*iregexp* 支持字符串、文本和日志监控项 **pattern** （可选）- 需要的模式（字符串参数必须使用双引号）	支持的值类型：float, integer, string, text, log 浮点监控项的匹配精度为 2.22e-16；如果数据库未升级，则精度为 0.000001。 使用 *bitand* 作为第三个参数时，第四个 pattern 参数可以指定为两个数字，用'/' 分隔：**number_to_compare_with/mask**。countunique() 计算值和 *mask* 的 "位与" 并比较结果与 *number_to_compare_with*。如果 "位与" 的结果是等于 *number_to_compare_with*，则计算该值。 如果 *number_to_compare_with* 和 *mask* 相等，则只需要指定 *mask*（不带'/'）。 使用 *regexp* 或 *iregexp* 作为第三个参数时，第四个 pattern 参数可以是普通或全局（以'@' 开头）正则表达式。在全局正则表达式的情况下，情况敏感性是从全局正则表达式设置中继承的。为了正则表达式匹配的目的，浮点值将始终用'.' 后的 4 位小数表示。还要注意，对于大数，十进制（存储在数据库中）和二进制（由 Zabbix server 使用）表示之间的差异可能会影响第 4 位小数。 例子： => **countunique**(/主机/键,**10m**) → 直到现在的最后 10 分钟的唯一值的数量 => **countunique**(/主机/键,**10m**,"like","error") → 直到现在的最后 10 分钟包含'error' 的唯一值的数量 => **countunique**(/主机/键,**10m**,"gt",12) → 直到现在的最后 10 分钟超过'12' 的唯一值的数量 => **countunique**(/主机/键,**#10**,"gt",12) → 直到现在的最后 10 个值中超过'12' 的唯一值的数量 => **countunique**(/主机/键,**10m:now-1d**,"gt",12) → 从现在起 24 小时 10 分钟至 24 小时前超过'12' 的唯一值的数量 => **countunique**(/主机/键,**10m**,"bitand","6/7") → 直到现在的最后 10 分钟在 3 个最低有效位具有'110'（二进制）的唯一值的数量 => **countunique**(/主机/键,**10m:now-1d**) → 从现在起 24 小时 10 分钟至 24 小时前的唯一值的数量

函数		
查找值匹配。	参见通用参数。 **sec** 或 **#num** （可选）- 如果未指定，默认为最新值 **operator** （可选；必须使用双引号） 支持的 operators： *eq* - 等于（整数，默认） *ne* - 不等于 *gt* - 大于 *ge* - 大于或等于 *lt* - 小于 *le* - 小于或等于 *like* （默认为字符串、文本、日志）- 值包含在 pattern 中给定的字符串（区分大小写） *bitand* - 位与 *regexp* - 正则表达式匹配（区分大小写） *iregexp* - 正则表达式匹配（不区分大小写） 注意： *eq*（默认）、*ne*、*gt*、*ge*、*lt*、*le*、*band*、*regexp*、*iregexp* 支持整数监控项 *eq*（默认）、*ne*、*gt*、*ge*、*lt*、*le*、*regexp*、*iregexp* 支持浮点监控项 *like*（默认）、*eq*、*ne*、*regexp*、*iregexp* 支持字符串、文本和日志监控项 **pattern** - 需要的模式（字符串参数必须使用双引号）；如果 operator 是 *regexp*、*iregexp*，则为Perl 兼容正则表达式 (PCRE) 正则表达式。	支持的值类型：float, int, str, text, log 返回值： 1 - 找到匹配 0 - 否则 如果处理了多个值，如果至少有一个匹配的值，则返回'1'。 使用 *regexp* 或 *iregexp* 作为第三个参数时，第四个 pattern 参数可以是普通或全局（以'@' 开头）正则表达式。在全局正则表达式的情况下，情况敏感性是从全局正则表达式设置中继承的。 例子： => **find**(/主机/键,**10m**,"like","error") → 在直到现在的最后 10 分钟内查找包含'error' 的值
first (/host/key,sec<:time shift>) 在定义的评估期间内的第一个（最旧的）值。	参见常用参数。	支持的值类型：float, int, str, text, log 例子： => **first**(/主机/键,**1h**) → 检索直到现在的最后一个小时内的最旧的值 另请参阅 last()。
fuzzytime (/host/key,sec)		

函数		
检查被动 agent 时间与 Zabbix server/proxy 时间的差异。	参见通用参数。	支持的值类型：float, int 返回值： 1 - 被动监控项值（作为时间戳）与 Zabbix server/proxy 时间戳（值收集的时钟）之间的差异小于或等于 T 秒 0 - 否则 通常与'system.localtime' 监控项一起使用，以检查本地时间是否与 Zabbix server 的本地时间同步。注意'system.localtime' 必须被配置为被动检查。 也可以与 vfs.file.time[/path/file,modify] 键一起使用，以检查文件长时间未更新。 例子： => **fuzzytime**(/主机/键,**60s**)=0 → 如果时间差异超过 60 秒，则检测到问题 此函数不推荐在涉及多个监控项的复杂触发器表达式中使用，因为它可能导致意外结果（时间差异将用最新的度量来衡量），例如在 `fuzzytime(/Host/system.localtime,60s)=0 or last(/Host/trap)<>0` 中
last (/host/key,<#num<:time shift») 最新值。	参见常用参数。 **#num** （可选）- 第 N 个最新值	支持的值类型：float, int, str, text, log 请注意，带井号的时间周期 (#N) 在这里与许多其他函数的工作方式不同。 例如： last() 总是等于 last(#1) last(#3) - 第三个最新值（不是三个最新值） 如果历史记录中一秒内存在超过两个值，Zabbix 不保证值的确切顺序。 例子： => **last**(/主机/键) → 检索最新值 => **last**(/主机/键,**#2**) → 检索上一个值 => **last**(/主机/键,**#1**) <> **last**(/主机/键,**#2**) → 最新值和上一个值不同 另请参阅 first()。
logeventid (/host/key,<#num<:time shift»,<pattern>) 检查最后一条日志条目的事件 ID 是否与正则表达式匹配。	查看通用参数。 **#num** （可选）- 最近的第 N 个值 **pattern** （可选）- 描述所需模式的正则表达式，Perl Compatible Regular Expression （PCRE）风格（字符串参数必须用双引号括起来）。	支持的值类型：日志 返回值： 0 - 不匹配 1 - 匹配
logseverity (/host/key,<#num<:time shift»)		

函数		
最后一条日志条目的日志严重性。	查看通用参数。 **#num** （可选）- 最近的第 N 个值	支持的值类型：日志 返回值： 0 - 默认严重性 N - 严重性（整数，适用于 Windows 事件日志：1 - 信息，2 - 警告，4 - 错误，7 - 审核失败，8 - 审核成功，9 - 严重，10 - 详细）。 Zabbix 从 Windows 事件日志的信息字段获取日志严重性。
logsource (/host/key,<#num<:time shift»,<pattern>) 检查最后一条日志条目的日志源是否与正则表达式匹配。	查看通用参数。 **#num** （可选）- 最近的第 N 个值 **pattern** （可选）- 描述所需模式的正则表达式，Perl Compatible Regular Expression 风格（字符串参数必须用双引号括起来）。	支持的值类型：日志 返回值： 0 - 不匹配 1 - 匹配 通常用于 Windows 事件日志。例如，logsource(”VMware Server”)。
monodec (/host/key,(sec|#num)<:time shift>,<mode>) 检查值是否呈现单调递减。	查看通用参数。 **mode** （必须用双引号括起）- *weak*（每个值都小于或等于前一个值；默认）或 *strict*（每个值都减少了）	支持的值类型：int 如果所有元素在时间段内持续减少，则返回 1，否则返回 0。 示例： => **monodec**(/Host1/system.swap.size[all,free]**60s**) + **monodec**(/Host2/system.swap.size[all,free]**60s**) + **monodec**(/Host3/system.swap.size[all,free]**60s**) - 计算有多少主机的空闲交换空间大小有所减少
monoinc (/host/key,(sec|#num)<:time shift>,<mode>) 检查值是否呈现单调递增。	查看通用参数。 **mode** （必须用双引号括起）- *weak*（每个值都大于或等于前一个值；默认）或 *strict*（每个值都增加了）	支持的值类型：integer 如果所有元素在时间段内持续增加，则返回 1，否则返回 0。 示例： => **monoinc**(/Host1/system.localtime,**#3**,”strict”)=0 - 检查系统本地时间是否一致递增
nodata (/host/key,sec,<mode>)		

函数		
检查是否未接收到数据。	查看通用参数。 **sec** 时间间隔不应少于 30 秒，因为历史同步进程仅每 30 秒计算一次此函数。 不允许使用 nodata(/host/key,0)。 **mode** - 如果设置为 *strict* （双引号括起），此函数将对 proxy 可用性不敏感(详见注释)。	支持所有值类型。 返回值： 1 - 如果在定义的时间期间内未接收到数据 0 - 否则 自 Zabbix 5.0 起，默认情况下，由 proxy 监控的'nodata' 触发器对 proxy 可用性敏感 - 如果 proxy 变得不可用，'nodata' 触发器不会在恢复连接后立即触发，而是会跳过延迟期间的数据。请注意，对于被动 proxy，如果在恢复连接超过 15 秒并且不少于 2 & ProxyUpdateFrequency 秒后，将激活抑制。对于主动 proxy，如果在恢复连接超过 15 秒后，将激活抑制。 要关闭对 proxy 可用性的敏感性，请使用第三个参数，例如： **nodata**(/host/key,**5m**,**"strict"**)；在这种情况下，该函数将像 5.0.0 之前一样工作，并且在没有数据的评估周期（五分钟）过后立即触发。 请注意，如果在第一个参数的期间内： - 没有数据并且 Zabbix 服务器已重启 - 没有数据并且维护已完成 - 没有数据并且监控项已添加或重新启用 错误将显示在触发器配置的信息列中。 如果 Zabbix 服务器、proxy 和 agent 之间存在时间差异，此函数可能无法正常工作。另见：时间同步要求。
percentile (/host/key,(sec|#num)<:time shift>,percentage) 周期的第 P 百分位数，P（百分比）由第三个参数指定。	查看通用参数。 **percentage** - 一个介于 0 到 100（含）之间的浮点数，小数点后最多可有 4 位数字	支持的值类型：float, integer
rate (/host/key,sec<:time shift>) 在定义时间段内单调递增计数器的每秒平均增长率。	查看通用参数。	支持的值类型：float, integer 功能上对应于 PromQL 中的'rate'。 示例： => **rate**(/host/key,**30s**) → 如果 30 秒内的单调增加是 20，此函数将返回 0.67。

7 运算符函数

此处列出的所有函数都支持：

- 触发器表达式
- 可计算监控项

关于函数参数的一些一般说明：

- 函数参数用逗号分隔
- 表达式可以被作为参数

函数	描述	函数专用参数	注释
between (value,min,max)	检查一个值是否属于给定范围	**value** - 要检查的值 **min** - 最小值 **max** - 最大值	支持的值类型：整数、浮点数 返回： 1 - 在范围内 0 - 否则 示例： => **between**(last(/host/key),**1**, - 如果值介于 1 和 10 之间，则触发。
in (value,value1,value2,...valueN)	检查一个值是否至少等于列出的值之一	**value** - 要检查的值 **value1,value2,...valueN** - 列出的值(字符串值必须用双引号引起来)	支持的值类型：全部 返回： 1 - 等于 0 - 否则 如果所有这些值都可以转换为数字，则将值与列出的值作为数字进行比较；否则作为字符串进行比较。 示例： => **in**(last(/host/key),**5**,**10**)= - 如果最后一个值等于 5 或 10 则触发 => **in**(**”text”**, last(/host/key),last(/host - 如果”text”等于最后两个值中的任何一个则触发。

8 预测函数

此处列出的所有函数都支持：

- 触发器表达式
- 可计算监控项

关于函数参数的一些整体说明：

- 函数参数用逗号分隔
- 可选函数参数（或参数部分）由 < > 表示
- 每个函数都描述了函数专用参数
- /host/key 和 (sec|#num)<:time shift> 参数绝不能被引用

常用参数

- /host/key 是引用主机监控项历史记录函数的常用强制性首选参数
- (sec|#num)<:time shift> 是引用主机监控项历史记录函数的常用强制性次选参数，其中：
- **sec** - 以秒为单位的最大评估周期 (可以使用时间后缀)，或者
- **#num** - 最新收集值最大评估范围(如果前面有 # 符号)
- **time shift** (可选) 允许将评估点及时移回。参阅有关指定时间偏移更多详细内容

预测函数

函数	描述	函数专用参数	注释
forecast (/host/key,(sec|#num)<:time shift>,time,<fit>,<mode>)			

函数				
		监控项的未来值、最大值、最小值、增量或平均值	参见常用参数. **time** - 以秒为单位的预测范围（可以使用时间后缀）；支持负值 **fit** （可选；必须双引号）- 用于拟合历史数据的函数 支持 `fits`: *linear* - 线性函数 *polynomialN* -N 次多项式 (1 <= N <= 6) *exponential* - 指数函数 *logarithmic* - 对数函数 *power* - 幂函数 注意：*linear* 是默认值，*polynomial1* 等价于 *linear* **mode** (可选；必须双引号) - 要求的输出 支持 `modes`: *value* - 值 (默认值) *max* - 最大值 *min* - 最小值 *delta* - 最大值-最小值 *avg* - 平均值 注意：*value* 估计当前监控值 `now` +	支持的值类型：float, int 如果要返回的值大于 1.7976931348623157E+ 或小于- 1.7976931348623157E+ 则返回值相应裁剪为 1.7976931348623157E+ 或- 1.7976931348623157E+ 仅当在表达式中误用（错误的监控项类型，无效参数）时才变得不受支持，否则在错误的情况下返回-1。 示例： => **forecast**(/host/key,**#10**,**1h**) → 根据最后 10 个值预测一小时内的监控项值 => **forecast**(/host/key,**1h**,**30m**) → 根据最后 1 小时的数据预测 30 分钟内的监控项值 => **forecast**(/host/key,**1h:now-1d**,**12h**) → 根据前一天的一小时预测 12 小时内的监控项值 => **forecast**(/host/key,**1h**,**10m**,… → 根据最后 1 小时的数据和指数函数预测 10 分钟内的监控项值 => **forecast**(/host/key,**1h**,**2h**,”p…

函数
timeleft (/host/key,(sec|#num)<:time shift>,threshold,<fit>)

函数				
		监控项达到指定阈值所需的时间（以秒为单位）	参见常用参数. **threshold** - 要达到的值（可以使用的单位后缀） **fit**（可选；必须双引号）- 见 forecast()	支持的值类型：float, int 如果要返回的值大于 1.7976931348623157E+ 则返回值被裁剪为 1.7976931348623157E+ 如果无法达到阈值，则返回 1.7976931348623157E+ 仅当在表达式中误用（错误的监控项类型，无效参数）时才变得不受支持，否则在错误的情况下返回 -1。 Examples: => **timeleft**(/host/key,**#10**, → 根据最后 10 个值，预测直到监控项值达到零的时间 => **timeleft**(/host/key,**1h**,**1** → 根据最后一小时的数据，预测直到监控项值达到 100 的时间 => **timeleft**(/host/key,**1h:n** **1d**,**100**) → 根据一天前的一小时数据，预测直到监控项值达到 100 的时间 => **timeleft**(/host/key,**1h**,**2** → 最后一小时的数据和假设该监控项的行为类似于二次

函数

9 字符串函数

此处列出的所有函数都支持：

- 触发器表达式
- 可计算监控项

关于函数参数的一些一般说明：

- 函数参数用逗号分隔
- 表达式可以被作为参数
- 字符串参数必须是双引号；否则可能会被曲解
- 可选函数参数（或参数部分）由 < > 表示

函数	描述	函数专用参数	注释
ascii (value)	值的最左边字符的 ASCII 码	**value** - 要检查的值	支持的值类型：string, text, log 例如，像‘Abc’ 这样的值将返回‘65’(‘A’ 的 ASCII 码)。 示例： => **ascii**(last(/host/key))
bitlength (value)	以比特为单位值的长度	**value** - 要检查的值	支持的值类型：string, text, log, integer 示例： => **bitlength**(last(/host/key
bytelength (value)	以字节为单位值的长度	**value** - 要检查的值	支持的值类型：string, text, log, integer 示例： => **byte-length**(last(/host/key))
char (value)			

函数			
	通过将值转换为 ASCII 码来返回字符	**value** - 要检查的值	支持的值类型：integer 值必须在 0-255 范围内。例如，像‘65’(转换为 ASCII 码) 将返回‘A’. 示例： => **char**(last(/host/key))
concat (<value1>,<value2>,...)	由串联引用的监控项值或常量值产生的字符串	**value** -历史函数之一返回的值或常量值（字符串、整数或浮点数）	支持的值类型：string, text, log, float, integer 例如，像‘Zab’ 这样的值串联‘bix’(常量字符串）将返回‘Zabbix’。 必须至少包含两个参数。 示例： => **con-cat**(last(/host/key),**”bix’** => **con-cat**(”1 min: ”,last(/host/system.cpu.l 15 min: ”,last(/host/system.cpu.l
insert (value,start,length,replacement)			

函数			
	从字符串中的指定位置开始，将指定的字符或空格插入到字符串中	**value** - 要检查的值 **start** - 起始位置 **length** - 要替换的位置 **replacement** - 替换字符串	支持的值类型：string, text, log 例如，如果将'bb'起始位置3，要替换的位置 2）替换为'b'，则类似于'Zabbbix'的值将替换为'Zabbix'。 示例：=> **insert**(last(/host/key),**3**,**2**,
left (value,count)	值的最左边的字符	**value** - 要检查的值 **count** - 要返回的字符数	支持的值类型：string, text, log 例如，您可以通过指定要返回的最左边的 3 个字符从'Zabbix'返回'Zab'。 示例：=> **left**(last(/host/key),**3**) - 返回最左边的三个字符 另见 right()。
length (value)			

函数				
		以字符为单位的值的长度	**value** - 要检查的值	支持的值类型：str, text, log 示例： => **length**(last(/host/key)) → 最新值的长度 => **length**(last(/host/key,#3 → 最近值中第三大的长度 => **length**(last(/host/key,#1 1d)) → 一天前最近值的长度
ltrim (value,<chars>)		从字符串的开头删除指定的字符	**value** - 要检查的值 **chars** - (可选）指定要删除的字符 默认情况下，空格会被左对齐（如果未指定可选字符）。	支持的值类型：string, text, log 示例： => **ltrim**(last(/host/key)) - 删除字符串开头的空格 => **ltrim**(last(/host/key),**"Z"** - 删除字符串开头任何的'Z' => **ltrim**(last(/host/key),**"Z"**) - 删除字符串开头的任何空格和'Z' 另见：rtrim(), trim()
mid (value,start,length)				

函数			
	返回从'start'指定的字符位置开始的 N 个字符的子字符串	**value** - 要检查的值 **start** - 子字符串的开始位置 **length** -要在子字符串中返回的位置	支持的值类型：string, text, log 例如，如果起始位置为 2，返回的位置为 4，则可以从像'Zabbix'这样的值返回'abbi'。 示例： => **mid**(last(/host/key),**2**,**4**)
repeat (value,count)			
	重复一个字符串	**value** - 要检查的值 **count** - 重复的次数	支持的值类型：string, text, log 示例： => **repeat**(last(/host/key),**2**) - 将值重复两次
replace (value,pattern,replacement)			
	在值中查找样例值并用替换值进行替换，所有出现的样例值都将被替换	**value** - 要检查的值 **pattern** - 找到样例值 **replacement** - 替换样例值的字符串	支持的值类型：string, text, log Example: => **replace**(last(/host/key),**"ib** - 将所有'ibb' 替换为'abb'
right (value,count)			

函数				
		值的最右边的字符	**value** - 要检查的值 **count** - 要返回的字符数	支持的值类型：string, text, log 例如，你可以通过指定最右边的 3 个字符来从'Zabbix'返回'bix'。 示例： => **right**(last(/host/key),**3**) - 返回最右边的三个字符 另见 left()。
rtrim (value,<chars>)		从字符串的末尾删除指定的字符	**value** - 要检查的值 **chars** - (可选）指定要删除的字符 默认情况下，空格会被右对齐（如果未指定可选字符）。	支持的值类型：string, text, log 示例： => **rtrim**(last(/host/key)) -删除字符串末尾的空格 => **rtrim**(last(/host/key),**"x** - 从字符串结尾删除任何'x' => **rtrim**(last(/host/key),**"x "**) - 从字符串结尾删除任何'x' 或空格 另见：ltrim(), trim()
trim (value,<chars>)				

函数			
	从字符串的开头和结尾删除指定的字符	**value** - - 要检查的值 **chars** - （可选）指定要删除的字符 默认情况下，空格会被居中对齐（如果未指定可选字符）。	支持的值类型：string, text, log Example: => **trim**(last(/host/key)) - 从字符串的开头和结尾删除空格 => **trim**(last(/host/key),**”_”**) - 从字符串的开头和结尾删除’_’ 另见：ltrim(), rtrim()

数学函数

此处列出的所有函数均在以下领域得到支持：

- 触发器表达式
- 计算型监控项

数学函数支持浮点数和整数值类型，除非另有说明。

关于函数参数的一些通用说明：

- 函数参数用逗号分隔
- 表达式可用作参数
- 可选的函数参数（或参数部分）用 < > 表示

函数		
描述	函数特定参数	注释
abs (value) 一个值的绝对值。	**value** - 要检查的值	支持的值类型：float, int, str, text, log 对于字符串返回： 0 - 值相等 1 - 值不同 示例： => **abs**(last(/host/key))>10 将计算绝对数值差异，如这些传入示例值所示（’previous’ 和’latest’ 值 = 绝对差异）： ’1’ 和’5’ = 4 ’3’ 和’1’ = 2 ’0’ 和’-2.5’ = 2.5
acos (value)		

函数		
一个值的反余弦作为角度，以弧度表示。	**value** - 要检查的值	该值必须在-1 和 1 之间。 例如，值’0.5’ 的反余弦将是’2.0943951’。 示例： => **acos**(last(/host/key))
asin (value) 一个值的反正弦作为角度，以弧度表示。	**value** - 要检查的值	该值必须在-1 和 1 之间。 例如，值’0.5’ 的反正弦将是’-0.523598776’。 示例： => **asin**(last(/host/key))
atan (value) 一个值的反正切作为角度，以弧度表示。	**value** - 要检查的值	例如，值’1’ 的反正切将是’0.785398163’。 示例： => **atan**(last(/host/key))
atan2 (value,abscissa) 指定为角度的纵坐标（值）和横坐标的反正切，以弧度表示。	**value** - 要检查的值 **abscissa** - 横坐标值	例如，值’1’ 的纵坐标和横坐标的反正切将是’2.21429744’。 示例： => **atan2**(last(/host/key),2)
avg (<value1>,<value2>,...) 引用监控项值的平均值。	**valueX** - 另一个处理监控项历史的函数返回的值	示例： => **avg**(avg(/host/key),avg(/host2/key2))
cbrt (value) 一个值的立方根。	**value** - 要检查的值	例如，’64’ 的立方根将是’4’，’63’ 的将是’3.97905721’。 示例： => **cbrt**(last(/host/key))
ceil (value) 将值向上舍入到最接近的较大或相等的整数。	**value** - 要检查的值	例如，’2.4’ 将被舍入为’3’。 示例： => **ceil**(last(/host/key)) 另请参阅 floor()。
cos (value) 一个值的余弦值，其中值是角度，以弧度表示。	**value** - 要检查的值	例如，值’1’ 的余弦将是’0.54030230586’。 示例： => **cos**(last(/host/key))
cosh (value) 一个值的双曲余弦值。	**value** - 要检查的值	例如，值’1’ 的双曲余弦将是’1.54308063482’。 返回值作为实数，而不是科学记数法。 示例： => **cosh**(last(/host/key))
cot (value)		

函数		
一个值的余切，其中该值是一个以弧度表示的角度。	**value** - 要检查的值	例如，值'1' 的余切将是'0.54030230586'。 示例： => **cot**(last(/host/key))
degrees (value) 将一个值从弧度转换为度数。	**value** - 要检查的值	例如，将值'1' 转换为度数将是'57.2957795'。 示例： => **degrees**(last(/host/key))
e 欧拉数（2.718281828459045）。		示例： => **e**()
exp (value) 一个值的欧拉数的幂。	**value** - 要检查的值	例如，值'2' 的欧拉数的幂将是'7.38905609893065'。 示例： => **exp**(last(/host/key))
expm1 (value) 一个值的欧拉数的幂减 1。	**value** - 要检查的值	例如，值'2' 的欧拉数的幂减 1 将是'6.38905609893065'。 示例： => **expm1**(last(/host/key))
floor (value) 将值向下舍入到最接近的较小或相等的整数。	**value** - 要检查的值	例如，'2.6' 将被舍入为'2'。 示例： => **floor**(last(/host/key)) 另请参阅 ceil()。
log (value) 自然对数。	**value** - 要检查的值	例如，值'2' 的自然对数将是'0.69314718055994529'。 示例： => **log**(last(/host/key))
log10 (value) 十进制对数。	**value** - 要检查的值	例如，值'5' 的十进制对数将是'0.69897000433'。 示例： => **log10**(last(/host/key))
max (<value1>,<value2>,...) 引用监控项值的最高值。	**valueX** - 另一个函数返回的值，该函数处理监控项历史	示例： => **max**(avg(/host/key),avg(/host2/key2))
min (<value1>,<value2>,...) 引用监控项值的最低值。	**valueX** - 另一个函数返回的值，该函数处理监控项历史	示例： => **min**(avg(/host/key),avg(/host2/key2))
mod (value,denominator) 除法余数。	**value** - 要检查的值 **denominator** - 除数	例如，值'5' 除以除数'2' 的余数将是'1'。 示例： => **mod**(last(/host/key),2)
pi 圆周率常数（3.14159265358979）。		示例： => **pi**()

函数		
power (value,power value) 一个值的幂。	**value** - 要检查的值 **power value** - 要使用的幂次	例如，值'2' 的 3 次幂将是'8'。 示例： => **power**(last(/host/key),3)
radians (value) 将一个值从度数转换为弧度。	**value** - 要检查的值	例如，将值'1' 转换为弧度将是'0.0174532925'。 示例： => **radians**(last(/host/key))
rand 返回一个随机整数值。		使用时间作为种子（伪随机生成），适用于数学目的，但不适用于密码学。 示例： => **rand**()
round (value,decimal places) 将值四舍五入到小数点后指定位数。	**value** - 要检查的值 **decimal places** - 指定四舍五入的小数位数（0 也是可能的）	例如，值'2.5482' 四舍五入到 2 位小数将是'2.55'。 示例： => **round**(last(/host/key),2)
signum (value) 如果值为负数返回'-1'，如果值为零返回'0'，如果值为正数返回'1'。	**value** - 要检查的值	示例： => **signum**(last(/host/key))
sin (value) 一个值的正弦，其中该值是一个以弧度表示的角度。	**value** - 要检查的值	例如，值'1' 的正弦将是'0.8414709848'。 示例： => **sin**(last(/host/key))
sinh (value) 一个值的双曲正弦。	**value** - 要检查的值	例如，值'1' 的双曲正弦将是'1.17520119364'。 示例： => **sinh**(last(/host/key))
sqrt (value) 一个值的平方根。	**value** - 要检查的值	此函数对负值将失败。 例如，值'3.5' 的平方根将是'1.87082869339'。 示例： => **sqrt**(last(/host/key))
sum (<value1>,<value2>,...) 引用监控项值的总和。	**valueX** - 另一个函数返回的值，该函数处理监控项历史	示例： => **sum**(avg(/host/key),avg(/host2/key2))
tan (value) 一个值的正切。	**value** - 要检查的值	例如，值'1' 的正切将是'1.55740772465'。 示例： => **tan**(last(/host/key))
truncate (value,decimal places) 将值截断到小数点后指定位数。	**value** - 要检查的值 **decimal places** - 指定截断的小数位数（0 也是可能的）	示例： => **truncate**(last(/host/key),2)

日期和时间函数

此处列出的所有函数均在以下领域得到支持：

- 触发器表达式
- 计算型监控项

日期和时间函数不能单独在表达式中使用；表达式中至少必须存在一个引用主机监控项的非基于时间的函数。

函数		
描述	函数特定参数	注释
date 当前日期的格式为 YYYYMMDD。		示例： => **date**()<20220101
dayofmonth 一个月中的第几天，范围为 1 到 31。		示例： => **dayofmonth**()=1
dayofweek 一周中的第几天，范围为 1 到 7（星期一 - 1，星期日 - 7）。		示例（仅限工作日）： => **dayofweek**()<6 示例（仅限周末）： => **dayofweek**()>5
自纪元以来的秒数（00:00:00 UTC，1970 年 1 月 1 日）。		示例： => **now**()<1640998800
time 当前时间的格式为 HHMMSS。		示例（仅限夜间，00:00-06:00）： => **time**()<060000

趋势函数

与历史函数不同，趋势函数使用趋势数据进行计算。

趋势存储每小时的聚合值。趋势函数使用这些每小时的平均值，因此适用于长期分析。

趋势函数的结果会被缓存，因此多次调用具有相同参数的相同函数只需从数据库中获取信息一次。趋势函数缓存由服务器参数TrendFunctionCacheSize控制。

仅引用趋势函数的触发器将根据表达式中的最小时间段每只评估一次。例如，像

```
trendavg(/host/key,1d:now/d) > 1 or trendavg(/host/key2,1w:now/w) > 2
```

这样的触发器将每天评估一次。如果触发器包含趋势和历史（或基于时间）函数，则根据通常原则进行计算。

此处列出的所有函数均在以下领域得到支持：

- 触发器表达式
- 计算型监控项

Some general notes on function parameters:

关于函数参数的一些通用说明：

- 函数参数用逗号分隔
- 可选的函数参数（或参数部分）用 < > 表示
- 每个函数的特定参数都与该函数一起描述
- `/host/key` 和 `时间周期：` 时间偏移参数永远不要引用

常用参数

- `/host/key` 是常见的强制第一个参数
- `time period:time shift` 是常用的第二个参数，其中：
- **time period** - 时间段（最小'1h'），定义为 <N><time unit> 其中 `N` - 时间单位数，`time unit` - `h`（小时），`d`（日），`w`（周），`M`（月）或 `y`（年）。
- **time shift** - 时间段偏移量（见函数示例）

趋势函数

函数

描述	函数特定参数	注释
baselinedev (/host/key,data period:time shift,season_unit,num_seasons) 返回最后一个数据周期与之前季节中相同数据周期的偏差次数（使用 stddevpop 算法）。	**data period** - 季节内数据收集周期，定义为 <N><time unit>，其中 N - 时间单位的数量 time unit - h（小时）、d（天）、w（周）、M（月）或 y（年），必须等于或小于季节 时间偏移 - 时间周期偏移量（见示例） **season_unit** - 一个季节的持续时间（h、d、w、M、y），不能小于数据周期 **num_seasons** - 要评估的季节数量	示例： => **baselinedev**(/host/key,1d:now/d,"M",6) → 计算前一天与之前 6 个月中相同一天之间的标准偏差（总体）次数。如果前一个月中不存在该日期，则使用该月的最后一天（7 月 31 日将与 1 月 31 日、2 月 28 日、...6 月 30 日进行分析）。 => **baselinedev**(/host/key,1h:now/h,"d",10) → 计算昨天前一个小时与之前十天中相同小时之间的标准偏差（总体）次数。
baselinewma (/host/key,data period:time shift,season_unit,num_seasons) 通过加权移动平均算法，根据多个相等时间周期（"季节"）中的相同时间帧的数据平均值来计算基线。	**data period** - 季节内的数据收集周期，定义为 <N><time unit>，其中 N - 时间单位的数量 time unit - h（小时）、d（天）、w（周）、M（月）或 y（年），必须等于或小于季节 时间偏移 - 时间周期偏移量，定义了季节中数据收集时间帧的结束（见示例） **season_unit** - 一个季节的持续时间（h、d、w、M、y），不能小于数据周期 **num_seasons** - 要评估的季节数量	示例： => **baselinewma**(/host/key,1h:now/h,"d",3) → 基于昨天结束的 3 天周期内最后一个完整小时计算基线。如果"now" 是周一 13:30，将分析周五、周六和周日 12:00-12:59 的数据。 => **baselinewma**(/host/key,2h:now/h,"d",3) → 基于昨天结束的 3 天周期内最后两小时计算基线。如果"now" 是周一 13:30，将分析周五、周六和周日 11:00-12:59 的数据。 => **baselinewma**(/host/key,1d:now/d,"M",4) → 基于上个月同一天的 4 个月中，上个月完整月份的基线计算。如果所需的日期不存在，则取月份的最后一天。如果今天是 9 月 1 日，将分析 7 月 31 日、6 月 30 日、5 月 31 日、4 月 30 日的数据。
trendavg (/host/key,time period:time shift) 定义时间段内趋势值的平均值。	查看通用参数。	示例： => **trendavg**(/host/key,**1h:now/h**) → 上一个小时的平均值（例如 12:00-13:00） => **trendavg**(/host/key,**1h:now/h-1h**) → 两小时前的平均值（11:00-12:00） => **trendavg**(/host/key,**1h:now/h-2h**) → 三小时前的平均值（10:00-11:00） => **trendavg**(/host/key,**1M:now/M-1y**) → 一年前的上个月的平均值
trendcount (/host/key,time period:time shift)		

函数		
在定义时间段内成功检索到的趋势值的数量。	查看通用参数。	示例： => **trendcount**(/host/key,**1h:now/h**) → 上一个小时的计数（例如 12:00-13:00） => **trendcount**(/host/key,**1h:now/h-1h**) → 两小时前的计数（11:00-12:00） => **trendcount**(/host/key,**1h:now/h-2h**) → 三小时前的计数（10:00-11:00） => **trendcount**(/host/key,**1M:now/M-1y**) → 一年前的上个月的计数
trendmax (/host/key,time period:time shift) 在定义时间段内趋势值的最大值。	查看通用参数。	示例： => **trendmax**(/host/key,**1h:now/h**) → 上一个小时的最大值（例如 12:00-13:00） => **trendmax**(/host/key,**1h:now/h**) - **trendmin**(/host/key,**1h:now/h**) → 计算上一个小时（12:00-13:00）最大值和最小值之间的差异（趋势变化量） => **trendmax**(/host/key,**1h:now/h-1h**) → 两小时前的最大值（11:00-12:00） => **trendmax**(/host/key,**1h:now/h-2h**) → 三小时前的最大值（10:00-11:00） => **trendmax**(/host/key,**1M:now/M-1y**) → 一年前的上个月的最大值
trendmin (/host/key,time period:time shift) 在定义时间段内趋势值的最小值。	查看通用参数。	示例： => **trendmin**(/host/key,**1h:now/h**) → 上一个小时的最小值（例如 12:00-13:00） => **trendmin**(/host/key,**1h:now/h**) - **trendmin**(/host/key,**1h:now/h**) → 计算上一个小时（12:00-13:00）最大值和最小值之间的差异（趋势变化量） => **trendmin**(/host/key,**1h:now/h-1h**) → 两小时前的最小值（11:00-12:00） => **trendmin**(/host/key,**1h:now/h-2h**) → 三小时前的最小值（10:00-11:00） => **trendmin**(/host/key,**1M:now/M-1y**) → 一年前的上个月的最小值
trendstl (/host/key,eval period:time shift,detection period,season,<deviations>,<devalg>,<s_window>)		

函数		
在检测期间返回异常率 - 介于 0 和 1 之间的小数，即（异常值的数量 / 总值数量）。	**eval period** - 必须分解的时间周期（最小为'1h'），定义为 <N><time unit> 其中 N - 时间单位的数量 time unit - h（小时）、d（天）、w（周）、M（月）或 y（年）。 时间偏移 - 时间周期偏移量（见示例） **detection period** - eval period 结束前的时间段，用于计算异常（最小为'1h'，不能长于 eval period），定义为 <N><time unit> 其中 N - 时间单位的数量 time unit - h（小时）、d（天）、w（周）。 **season** - 预期重复模式（"季节"）的最短时间段（最小为'2h'，不能长于 eval period，eval period 中的条目数量必须大于由此产生的频率（season/h）的两倍），定义为 <N><time unit> 其中 N - 时间单位的数量 time unit - h（小时）、d（天）、w（周）。 **deviations** - 被视为异常的偏差数量（由 devalg 计算）（可以是小数），（必须大于或等于 1，默认为 3） **devalg**（必须用双引号括起）- 偏差算法，可以是 *stddevpop*、*stddevsamp* 或 *mad*（默认） **s_window** - 季节性提取的 loess 窗口跨度（滞后数）（默认为 10 * eval period 中的条目数量 + 1）	示例： => **trend-stl**(/host/key,**100h:now/h**,10h,2h) → 分析最后 100 小时的趋势数据，找到该时期的最后 10 小时的异常率，预期周期性为 2 小时，如果评估期间的剩余序列值达到该剩余序列的 MAD 的 3 个偏差，则视为异常 => **trendstl**(/host/key,**100h:now/h-10h**,100h,2h,2.1,"mad") → 分析 100 小时的趋势数据，直到 10 小时前，找到整个时期的异常率，预期周期性为 2 小时，如果评估期间的剩余序列值达到该剩余序列的 MAD 的 2.1 个偏差，则视为异常 => **trendstl**(/host/key,**100d:now/d-1d**,10d,1d,4,,10) → 分析直到昨天为止的 100 天的趋势数据，找到该时期最后 10 天的异常率，预期周期性为 1 天，如果评估期间的剩余序列值达到该剩余序列的 MAD 的 4 个偏差，则视为异常，覆盖季节性提取的 loess 窗口的默认跨度"10 * eval period 中的条目数量 + 1"，使用 10 个滞后的跨度 => **trendstl**(/host/key,**1M:now/M-1y**,1d,2h,,"stddevsamp") → 分析一年前的上个月，找到该时期最后一天的异常率，预期周期性为 2 小时，如果评估期间的剩余序列值达到该剩余序列的样本标准偏差的 3 个偏差，则视为异常
trendsum (/host/key,time period:time shift) 在定义时间段内趋势值的总和。	查看通用参数。	示例： => **trendsum**(/host/key,**1h:now/h**) → 上一个小时的总和（例如 12:00-13:00） => **trendsum**(/host/key,**1h:now/h-1h**) → 两小时前的总和（11:00-12:00） => **trendsum**(/host/key,**1h:now/h-2h**) → 三小时前的总和（10:00-11:00） => **trendsum**(/host/key,**1M:now/M-1y**) → 一年前的上个月的总和

7 宏

1 支持的宏

概述

下表包含 Zabbix 原生支持宏的完整列表。

Note:
要查看某个功能所支持的所有宏 (例如，想要知道”map URL” 功能中支持的所有宏)，你可以在浏览器的搜索框中输入这个功能的名称（用快捷键 CTRL+F 调出搜索框)，然后点击 “下一个”

宏名称	支持该宏的功能	具体描述
{ACTION.ID}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 基于服务的通知和命令 → 服务更新的通知和命令 → 自动发现的通知和命令 → 自动注册的通知和命令 → 内部通知	触发动作的数字 *ID*
{ACTION.NAME}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 基于服务的通知和命令 → 服务更新的通知和命令 → 自动发现的通知和命令 → 自动注册的通知和命令 → 内部通知	触发动作的名字
{ALERT.MESSAGE}	→ 告警脚本参数	配置 “动作” 时用到的 “默认消息” 从 3.0.0 版本开始支持
{ALERT.SENDTO}	→ 告警脚本参数	’ 配置 “媒体” 时用到的 “发送至” 从 3.0.0 版本开始支持
{ALERT.SUBJECT}	→ 告警脚本参数	配置 “动作” 时用到的 “默认主题” 从 3.0.0 版本开始支持
{DATE}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 基于服务的通知和命令 → 服务更新的通知和命令 → 自动发现的通知和命令 → 自动注册的通知和命令 → 内部通知 → 手动动作脚本	日期格式为 *yyyy.mm.dd*
{DISCOVERY.DEVICE.IPADDRESS}	→ 自动发现的通知和命令	被发现的设备的 *IP* 地址 永久有效，不依赖于是否添加了设备
{DISCOVERY.DEVICE.DNS}	→ 自动发现的通知和命令	被发现的设备的 *DNS* 名称 永久有效，不依赖于是否添加了设备
{DISCOVERY.DEVICE.STATUS}	→ 自动发现的通知和命令	被发现设备的状态: 可能是 UP 或者 DOWN.
{DISCOVERY.DEVICE.UPTIME}	→ 自动发现的通知和命令	某设备最近一次被监控到状态改变的时间。，精确到秒 例如：1h 29m 01s. 对于状态是 DOWN 的设备来说，这就是他们变成关机状态的时间长度
{DISCOVERY.RULE.NAME}	→ 自动发现的通知和命令	用于发现设备或服务是否在线的自动发现规则的名称
{DISCOVERY.SERVICE.NAME}	→ 自动发现的通知和命令	自动发现的服务的名称 例如：HTTP.
{DISCOVERY.SERVICE.PORT}	→ 自动发现的通知和命令	自动发现的服务的端口 例如：80.

宏名称	支持该宏的功能	具体描述
{DISCOVERY.SERVICE.STATUS}	→ 自动发现的通知和命令	自动发现的 < 服务://> 状态，可能是 *UP* 或者 *DOWN* | |*{DISCOVERY.SERVICE.UPTIME}* |→ 自动发现的通知和命令 | 该服务自动发现状态最近改变的时间, 精确到秒 <*br*> 例如：*1h 29m 01s.*<*br*> 对于状态是 *DOWN* 的服务，是该服务不可用的时间 | |*{ESC.HISTORY}* |→ 基于触发器的通知和命令 <*br*>→ 问题更新的通知和命令 <*br*>→ 基于服务的通知和命令 <*br*>→ 服务更新的通知和命令 <*br*>→ 内部通知 | 升级 history.Log 过去发送的消息。<*br*> 显示之前发送过的通知，通知是基于什么升级步骤发送的，以及他们的状态 *(*已发送, 进行中 or 失败*)*.
{EVENT.ACK.STATUS}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 手动动作脚本	事件确认发状态 *(*是/否*)*.
{EVENT.AGE}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 基于服务的通知和命令 → 服务更新的通知和命令 → 服务恢复通知和命令 → 自动发现的通知和命令 → 自动注册的通知和命令 → 内部通知 → 手动动作脚本	触发动作的事件的时长, 精确到秒 在升级消息时有用
{EVENT.DATE}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 基于服务的通知和命令 → 服务更新的通知和命令 → 服务恢复通知和命令 → 自动发现的通知和命令 → 自动注册的通知和命令 → 内部通知 → 手动动作脚本	触发动作的事件的日期
{EVENT.DURATION}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 基于服务的通知和命令 → 服务更新的通知和命令 → 服务恢复通知和命令 → 内部通知 → 手动动作脚本	事件持续时间（发生问题和恢复事件之间的时差）, 精确到秒 在问题恢复信息中有用 从 5.0.0 版本开始支持
{EVENT.ID}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 基于服务的通知和命令 → 服务更新的通知和命令 → 服务恢复通知和命令 → 自动发现的通知和命令 → 自动注册的通知和命令 → 内部通知 → 触发器 URLs → 手动动作脚本	触发动作的事件的数字 *ID*
{EVENT.NAME}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 基于服务的通知和命令 → 服务更新的通知和命令 → 服务恢复通知和命令 → 内部通知 → 手动动作脚本	触发动作的问题事件名称 从 4.0.0 版本开始支持

宏名称	支持该宏的功能	具体描述
{EVENT.NSEVERITY}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 基于服务的通知和命令 → 服务更新的通知和命令 → 服务恢复通知和命令 → 手动动作脚本	描述事件严重性的数值包括: 0 - 未分级, 1 - 信息, 2 - 警告, 3 - 普通, 4 - 高, 5 - 灾难. 从 4.0.0 版本开始支持
{EVENT.OBJECT}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 基于服务的通知和命令 → 服务更新的通知和命令 → 服务恢复通知和命令 → 自动发现的通知和命令 → 自动注册的通知和命令 → 内部通知 → 手动动作脚本	描述事件对象的数值包括：0 - 触发器, 1 - 自动发现的主机, 2 - 自动发现的服务, 3 - 低级自动发现规则, 4 - Item, 5 - 低级自动发现规则. 从 4.4.0 版本开始支持
{EVENT.OPDATA}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 手动动作脚本	问题对应触发器的当前值 从 4.4.0 版本开始支持
{EVENT.RECOVERY.DATE}	→ 问题恢复通知 和命令 → 问题更新的通知和命令 (如果恢复了) → 服务恢复通知和命令 → 手动动作脚本 (如果恢复了)	事件恢复日期
{EVENT.RECOVERY.ID}	→ 问题恢复通知 和命令 → 问题更新的通知和命令 (如果恢复了) → 服务恢复通知和命令 → 手动动作脚本 (如果恢复了)	恢复事件的数字 *ID*
{EVENT.RECOVERY.NAME}	→ 问题恢复通知 和命令 → 问题更新的通知和命令 (如果恢复了) → 服务恢复通知和命令 → 手动动作脚本 (如果恢复了)	恢复时间名称 从 4.4.1 版本开始支持
{EVENT.RECOVERY.STATUS}	→ 问题恢复通知 和命令 → 问题更新的通知和命令 (如果恢复了) → 服务恢复通知和命令 → 手动动作脚本 (如果恢复了)	恢复事件的文字描述
{EVENT.RECOVERY.TAGS}	→ 问题恢复通知 和命令 → 问题更新的通知和命令 (如果恢复了) → 服务恢复通知和命令 → 手动动作脚本 (如果恢复了)	用逗号分隔的恢复事件标签如果没有标签的，就是一个空字符 从 3.2.0 版本开始支持
{EVENT.RECOVERY.TAGSJSON}	→ 问题恢复通知 和命令 → 问题更新的通知和命令 (如果恢复了) → 服务恢复通知和命令 → 手动动作脚本 (如果恢复了)	一个包括事件标签对象的 *JSON* 数组. 如果没有标签的，就是一个空数组 从 5.0.0 版本开始支持
{EVENT.RECOVERY.TIME}	→ 问题恢复通知 和命令 → 问题更新的通知和命令 (如果恢复了) → 服务恢复通知和命令 → 手动动作脚本 (如果恢复了)	恢复时间的时间.
{EVENT.RECOVERY.VALUE}	→ 问题恢复通知 和命令 → 问题更新的通知和命令 (如果恢复了) → 服务恢复通知和命令 → 手动动作脚本 (如果恢复了)	恢复时间的数值
{EVENT.SEVERITY}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 基于服务的通知和命令 → 服务更新的通知和命令 → 服务恢复通知和命令 → 手动动作脚本	时间严重性等级的名称 从 4.0.0 版本开始支持

宏名称	支持该宏的功能	具体描述
{EVENT.SOURCE}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 基于服务的通知和命令 → 服务更新的通知和命令 → 服务恢复通知和命令 → 自动发现的通知和命令 → 自动注册的通知和命令 → 内部通知 → 手动动作脚本	事件来源的数字描述包括：0 - 触发器, 1 - 自动发现, 2 - 自动注册, 3 - 内部. 从 4.0.0 版本开始支持
{EVENT.STATUS}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 基于服务的通知和命令 → 服务更新的通知和命令 → 服务恢复通知和命令 → 内部通知 → 手动动作脚本	出发动作的事件的文字描述
{EVENT.TAGS}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 基于服务的通知和命令 → 服务更新的通知和命令 → 服务恢复通知和命令 → 手动动作脚本	一组逗号分割单时间标签. 如果没有标签的，就是一个空字符 从 3.2.0 版本开始支持
{EVENT.TAGSJSON}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 基于服务的通知和命令 → 服务更新的通知和命令 → 服务恢复通知和命令 → 手动动作脚本	一个包括事件标签对象的 *JSON* 数组. 如果没有标签的，就是一个空数组 从 5.0.0 版本开始支持
{EVENT.TAGS.< 标签名称 >}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 基于服务的通知和命令 → 服务更新的通知和命令 → 服务恢复通知和命令 → Wehook 媒介 URL 名称和 URL → 手动动作脚本	用标签名称来引用的事件标签值. 如果标签名称包含非字母数字的字符（包括非英语多字节 UTF 字符）需用双引号括起来。标签名称中引号括起来的引号和反斜杠必须用反斜杠转义。 从 4.4.2 版本开始支持
{EVENT.TIME}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 基于服务的通知和命令 → 服务更新的通知和命令 → 服务恢复通知和命令 → 自动发现的通知和命令 → 自动注册的通知和命令 → 内部通知 → 手动动作脚本	* 触发动作的事件时间 *
{EVENT.UPDATE.ACTION}	→ 问题更新的通知和命令	问题更新期间执行的可读的动作名称. 包括这些值: 已确认, 已注释, 严重级别从 *(*原始严重级别*)* 变化为 *(*更新严重级别*)* 和 已关闭 (取决于一次更新了多少个动作). 从 4.0.0 版本开始支持
{EVENT.UPDATE.DATE}	→ 问题更新的通知和命令 → 服务更新的通知和命令	事件更新 的日期 *(*例如：确认，等等*)*. 过去使用 {ACK.DATE}，当前版本已弃用
{EVENT.UPDATE.HISTORY}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 手动动作脚本	问题更新 *(*例如：确认，等等*)* 的日志 过去使用 {EVENT.ACK.HISTORY}，当前版本已弃用
{EVENT.UPDATE.MESSAGE}	→ 问题更新的通知和命令	问题更新的消息。 过去使用 {ACK.MESSAGE}，当前版本已弃用
{EVENT.UPDATE.STATUS}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 手动动作脚本	问题更新状态的数值。包括：0 - 因为问题/恢复事件触发了 Webhook, 1 - 更新操作 从 4.4.0 版本开始支持
{EVENT.UPDATE.TIME}	→ 问题更新的通知和命令 → 服务更新的通知和命令	事件更新 *(*例如：确认，等等*)* 的时间 过去使用 {ACK.TIME}，当前版本已弃用

宏名称	支持该宏的功能	具体描述
{EVENT.VALUE}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 基于服务的通知和命令 → 服务更新的通知和命令 → 服务恢复通知和命令 → 内部通知 → 手动动作脚本	触发动作的事件的数值 *(1* 表示问题*, 0* 表示恢复*)*。
{FUNCTION.VALUE<1-9>}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 手动动作脚本 → 时间名称	在发生事件的时刻，触发器表达式中，基于监控项的第 *N* 个函数的值 只包括以*/host/key* 开始的函数. 具体参见宏索引
{FUNCTION.RECOVERY.VALUE<1-9>}	→ 问题恢复通知 和命令 → 问题更新的通知和命令 → 手动动作脚本	在发生事件的时刻，恢复表达式中，基于监控项的第 *N* 个函数的值 只包括以*/host/key* 开始的函数. 具体参见宏索引.
{HOST.CONN}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 内部通知 → 地图元素标签，地图 URL 名称和值 → 监控项键值参数[1] → 主机界面 IP/DNS → Trapper 监控项” 主机范围” 字段 → 数据库监控附加参数 → SSH 和 Telnet 脚本 → JMX 监控项端点字段 → Web 监控[4] → 低级自动发现规则过滤正则表达式 → 仪表盘小部件动态 URL 的 URL 字段 → 触发器名称, 时间名称, 当前操作数据和描述 → 触发器 URL → 标签名称和赋值 → 脚本类型监控项，监控项原型和自动发现规则参数名称和赋值 → HTTP agent 类型监控项，监控项原型和自动发现规则字段: URL, 查询字段, 请求体, 请求头, 代理, SSL 证书文件, SSL key 文件, 主机白名单. → 手动主机动作脚本 （包括确认文本） → 手动动作脚本 （包括确认文本）	依赖于主机设置的主机 *IP* 地址或 *DNS* 名称[2]. 在触发器表达式中，可以用数字索引，来指向第一个，第二个，第三个主机，等等。例如：{HOST.CONN<1-9>} 详见索引宏.
{HOST.DESCRIPTION}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 内部通知 → 地图元素标签 → 手动动作脚本	主机描述. 在触发器表达式中，该宏可用配合数字索引使用，来指向第一个，第二个，第三个主机，等等。例如：{HOST.DESCRIPTION<1-9>} 详见索引宏.

宏名称	支持该宏的功能	具体描述
{HOST.DNS}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 内部通知 → 地图元素标签，地图 URL 名称和值 → 监控项键值参数[1] → 主机界面 IP/DNS → Trapper 监控项" 主机范围" 字段 → 数据库监控附加参数 → SSH 和 Telnet 脚本 → JMX 监控项端点字段 → Web 监控[4] → 低级自动发现规则过滤正则表达式 → 仪表盘小部件中，动态 URL 的 URL 字段 → 触发器名称, 时间名称, 当前操作数据和描述 → 触发器 URLs → 标签名称和赋值 → 脚本类型监控项，监控项原型和自动发现规则参数名称和赋值 → HTTP agent 类型监控项，监控项原型和自动发现规则字段: URL, 查询字段, 请求体, 请求头, 代理, SSL 证书文件, SSL key 文件, 主机白名单. → 手动主机动作脚本 （包括确认文本） → 手动动作脚本 （包括确认文本）	主机 *DNS* 名称[2]. T 在触发器表达式中，该宏可用配合数字索引使用，来指向第一个，第二个，第三个主机，等等。例如：{HOST.DNS<1-9>} 详见索引宏.
{HOST.HOST}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 自动注册的通知和命令 → 内部通知 → 监控项键值参数 → 地图元素标签，地图 URL 名称和值 → 主机界面 IP/DNS → Trapper 监控项" 主机范围" 字段 → 数据库监控附加参数 → SSH 和 Telnet 脚本 → JMX 监控项端点字段 → Web 监控[4] → 低级自动发现规则过滤正则表达式 → 仪表盘小部件动态 URL 的 URL 字段 → 触发器名称, 时间名称, 当前操作数据和描述 → 触发器 URLs → 标签名称和赋值 → 脚本类型监控项，监控项原型和自动发现规则参数名称和赋值 → HTTP agent 类型监控项，监控项原型和自动发现规则字段: URL, 查询字段, 请求体, 请求头, 代理, SSL 证书文件, SSL key 文件, 主机白名单. → 手动主机动作脚本 （包括确认文本） → 手动动作脚本 （包括确认文本）	主机名称 在触发器表达式中，可以用数字索引，来指向第一个，第二个，第三个主机，等等。例如：{HOST.HOST<1-9>} 详见索引宏. 过去使用 {HOSTNAME<1-9>}，当前版本已弃用
{HOST.ID}	→ 基于触发器的通知和命令 → 问题更新的通知和命令 → 内部通知 → 地图元素标签，地图 URL 名称和值 → 仪表盘小部件动态 URL 的 URL 字段 → 触发器 URL → 标签名称和赋值 → 手动动作脚本	主机 *ID.* 在触发器表达式中，可以用数字索引，来指向第一个，第二个，第三个主机，等等。例如：{HOST.ID<1-9>} 详见索引宏.

宏名称	支持该宏的功能	具体描述
{HOST.IP}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 自动注册通知和命令 → 内部通知 → 映射元素标签，映射 URL 名称和值 → 监控项关键参数[1] → 主机接口 IP/DNS → 触发器监控项" 允许的主机" 字段 → 数据库监控附加字段 → SSH 和 Telnet 脚本 → JMX 项目终结点字段 → Web 监控[4] → 低级别自动发现过滤正则表达式 → 动态 URL 仪表盘组件的 URL 字段 → 触发器名称，事件名称，操作型数据和描述 → 触发器 URL → 标签名称和值 → 脚本类型监控项，监控项原型和发现规则参数名称和值 → HTTP 代理类型监控项，监控项原型和发现规则字段： URL，查询字段，请求正文，报头，代理，SSL 证书文件，SSL 密钥文件，允许的主机。. → 自定义主机操作scripts (including confirmation text) → 自定义事件操作scripts (including confirmation text)	主机 *IP* 地址[2]. 这个宏可以和一个数字索引一起使用，例如：{HOST.IP<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros. {IPADDRESS<1-9>} 已弃用。
{HOST.METADATA}	→ 自动注册通知和命令	主机元数据。 仅适用活动代理自动注册。
{HOST.NAME}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 映射元素标签，映射 URL 名称和值 → 监控项关键参数 → 主机接口 IP/DNS → 触发器监控项" 允许的主机" 字段 → 数据库监控附加字段 → SSH 和 Telnet 脚本 → Web 监控[4] → 低级别自动发现过滤正则表达式 → 动态 URL 仪表盘组件的 URL 字段 → 触发器名称，事件名称，操作型数据和描述 → 触发器 URL → 标签名称和值 → 脚本类型监控项，监控项原型和发现规则参数名称和值 → HTTP 代理类型监控项，监控项原型和发现规则字段： URL，查询字段，请求正文，报头，代理，SSL 证书文件，SSL 密钥文件，允许的主机。. → 自定义主机操作scripts (including confirmation text) → 自定义事件操作scripts (including confirmation text)	可见的主机名。 这个宏可以和一个数字索引一起使用，例如：{HOST.NAME<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.

宏名称	支持该宏的功能	具体描述
{HOST.PORT}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 自动注册通知和命令 → 内部通知 → 触发器名称，事件名称，操作型数据和描述 → 触发器 URL → JMX 项目终结点字段 → 标签名称和值 → 自定义事件操作scripts	主机（代理）端口 [2]. 这个宏可以和一个数字索引一起使用，例如：{HOST.PORT<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{HOST.TARGET.CONN}	→ 基于触发器的命令 → 问题更新命令 → 发现命令 → 自动注册命令	目标主机 IP 地址或 DNS 名称，取决于主机设置。 从 5.4.0 开始支持。
{HOST.TARGET.DNS}	→ 基于触发器的命令 → 问题更新命令 → 发现命令 → 自动注册命令	目标主机的 DNS 名称。 从 5.4.0 开始支持。
{HOST.TARGET.HOST}	→ 基于触发器的命令 → 问题更新命令 → 发现命令 → 自动注册命令	目标主机的技术名称。 从 5.4.0 开始支持。
{HOST.TARGET.IP}	→ 基于触发器的命令 → 问题更新命令 → 发现命令 → 自动注册命令	目标主机的 IP 地址。 从 5.4.0 开始支持。
{HOST.TARGET.NAME}	→ 基于触发器的命令 → 问题更新命令 → 发现命令 → 自动注册命令	目标主机的技术名称。 从 5.4.0 开始支持。
{HOSTGROUP.ID}	→ 映射元素标签，映射 URL 名称和值	主机组 *ID*。
{INVENTORY.ALIAS}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的别名字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.ALIAS<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.ASSET.TAG}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的资产标签字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.ASSET.TAG<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.CHASSIS}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的机壳字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.CHASSIS<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.CONTACT}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的联系字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.CONTACT<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros. {PROFILE.CONTACT<1-9>} 已弃用。
{INVENTORY.CONTRACT.NUMBER}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的合同号码字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.CONTRACT.NUMBER<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.

宏名称	支持该宏的功能	具体描述
{INVENTORY.DEPLOYMENT.STATUS}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的部署状态字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.DEPLOYMENT.STATUS<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.HARDWARE}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的硬件字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.HARDWARE<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros. {PROFILE.HARDWARE<1-9>} 已弃用。
{INVENTORY.HARDWARE.FULL}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的硬件 *(全细节)* 字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.HARDWARE.FULL<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.HOST.NETMASK}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的主机子网掩码字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.HOST.NETMASK<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.HOST.NETWORKS}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的主机网络字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.HOST.NETWORKS<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.HOST.ROUTER}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的主机路由字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.HOST.ROUTER<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.HW.ARCH}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的硬件架构字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.HW.ARCH<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.HW.DATE.DECOMM}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的硬件退役日期字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.HW.DATE.DECOMM<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.HW.DATE.EXPIRY}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的硬件维修过期日期字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.HW.DATE.EXPIRY<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.HW.DATE.INSTALL}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的硬件安装日期字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.HW.DATE.INSTALL<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.

宏名称	支持该宏的功能	具体描述
{INVENTORY.HW.DATE.PURCHASE}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的硬件购买日期字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.HW.DATE.PURCHASE<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.INSTALLER.NAME}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的安装名称字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.INSTALLER.NAME<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.LOCATION}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的位置字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.LOCATION<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros. {PROFILE.LOCATION<1-9>} 已弃用。
{INVENTORY.LOCATION.LAT}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的位置纬度字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.LOCATION.LAT<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.LOCATION.LON}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的位置经度字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.LOCATION.LON<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.MACADDRESS.A}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的 *Mac* 地址 *A* 字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.MACADDRESS.A<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros. {PROFILE.MACADDRESS<1-9>} 已弃用。
{INVENTORY.MACADDRESS.B}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的 *Mac* 地址 *B* 字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.MACADDRESS.B<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.MODEL}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的模型字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.MODEL<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.NAME}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的名称字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.NAME<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros. {PROFILE.NAME<1-9>} 已弃用。

宏名称	支持该宏的功能	具体描述
{INVENTORY.NOTES}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的备注字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.NOTES<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros. {PROFILE.NOTES<1-9>} 已弃用。
{INVENTORY.OOB.IP}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的带外 *IP* 地址字段。*field in host inventory.* 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.OOB.IP<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.OOB.NETMASK}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的带外子网掩码字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.OOB.NETMASK<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.OOB.ROUTER}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的带外路由器字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.OOB.ROUTER<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.OS}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的操作系统字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.OS<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros. {PROFILE.OS<1-9>} 已弃用。
{INVENTORY.OS.FULL}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的操作系统 *(所有细节)* 字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.OS.FULL<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.OS.SHORT}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的操作系统 *(简短)* 字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.OS.SHORT<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.POC.PRIMARY.CELL}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的主要的 *POC* 手机字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.POC.PRIMARY.CELL<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.POC.PRIMARY.EMAIL}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的主要的 *POC email* 字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.POC.PRIMARY.EMAIL<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.

宏名称	支持该宏的功能	具体描述
{INVENTORY.POC.PRIMARY.ANY}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的主要的 *POC* 名称字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.POC.PRIMARY.NAME<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.POC.PRIMARY.NOTES}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的主要的 *POC* 注记字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.POC.PRIMARY.NOTES<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.POC.PRIMARY.PHONE.A}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的主要的 *POC* 电话 *A* 字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.POC.PRIMARY.PHONE.A<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.POC.PRIMARY.PHONE.B}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的主要的 *POC* 电话 *B* 字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.POC.PRIMARY.PHONE.B<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.POC.PRIMARY.SCREEN}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的主要的 *POC* 屏幕名称字段。。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.POC.PRIMARY.SCREEN<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.POC.SECONDARY.CELL}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的第二个 *POC* 手机号码字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.POC.SECONDARY.CELL<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.POC.SECONDARY.EMAIL}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的第二个 *POC email* 字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.POC.SECONDARY.EMAIL<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.POC.SECONDARY.NAME}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的第二个 *POC* 名称字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.POC.SECONDARY.NAME<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.POC.SECONDARY.NOTES}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的第二个 *POC* 注记字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.POC.SECONDARY.NOTES<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.POC.SECONDARY.PHONE.A}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的第二个 *POC* 电话 *A* 字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.POC.SECONDARY.PHONE.A<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.

宏名称	支持该宏的功能	具体描述
{INVENTORY.POC.SECONDARY.PHONE.B}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的第二个 *POC* 电话 *B* 字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.POC.SECONDARY.PHONE.B<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.POC.SECONDARY.SCREEN}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的第二个 *POC* 聚合图形名称字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.POC.SECONDARY.SCREEN<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.SERIALNO.A}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的序列号 *A* 字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.SERIALNO.A<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros. {PROFILE.SERIALNO<1-9>} 已弃用。
{INVENTORY.SERIALNO.B}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的序列号 *B* 字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.SERIALNO.B<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.SITE.ADDRESS.A}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的场所地址 *A* 字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.SITE.ADDRESS.A<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.SITE.ADDRESS.B}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的场所地址 *B* 字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.SITE.ADDRESS.B<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.SITE.ADDRESS.C}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的场所地址 *C* 字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.SITE.ADDRESS.C<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.SITE.CITY}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的场所城市字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.SITE.CITY<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
{INVENTORY.SITE.COUNTRY}	→ 基于触发器的通知和命令 → 问题更新通知和命令 → 内部通知 → 标签名称和值 → 映射元素标签，映射 URL 名称和值 → 自定义事件操作scripts	主机资产记录中的场所国家字段。 这个宏可以和一个数字索引一起使用，例如：{INVENTORY.SITE.COUNTRY<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.

宏名称	支持该宏的功能	具体描述
{INVENTORY.SITE.NOTES}	→ 基于触发器的通知和命令	主机资产记录中的场所备注字段。
	→ 问题更新通知和命令	
	→ 内部通知	这个宏可以和一个数字索引一起使用，例如：{INVENTORY.SITE.NOTES<1-9>} 指向触发器表达式中的第 1 个、第 2 个、第 3 个等主机。参阅indexed macros.
	→ 标签名称和值	
	→ 映射元素标签，映射 URL 名称和值	
	→ 自定义事件操作scripts	
{INVENTORY.SITE.RACK}	→ 基于触发器的通知和命令	主机清单中站点的机架位置字段。
	→ 故障更新通知和命令	
	→ 内部通知	该宏可以与数字索引一起使用，例如 {INVENTORY.SITE.RACK<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
	→ 标签的名称和值	
	→ 地图元素标签, 地图 URL 的名称和值	
	→ 手动操作脚本事件	
{INVENTORY.SITE.STATE}	→ 基于触发器的通知和命令	主机清单中站点所属的省/市。
	→ 故障更新通知和命令	
	→ 内部通知	该宏可以与数字索引一起使用，例如 {INVENTORY.SITE.STATE<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
	→ 标签的名称和值	
	→ 地图元素标签, 地图 URL 的名称和值	
	→ 手动操作脚本事件	
{INVENTORY.SITE.ZIP}	→ 基于触发器的通知和命令	主机清单中站点的邮编字段
	→ 故障更新通知和命令	
	→ 内部通知	该宏可以与数字索引一起使用，例如 {INVENTORY.SITE.ZIP<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
	→ 标签的名称和值	
	→ 地图元素标签, 地图 URL 的名称和值	
	→ 手动操作脚本事件	
{INVENTORY.SOFTWARE}	→ 基于触发器的通知和命令	主机清单中的软件字段。
	→ 故障更新通知和命令	
	→ 内部通知	该宏可以与数字索引一起使用，例如 {INVENTORY.SOFTWARE<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
	→ 标签的名称和值	
	→ 地图元素标签, 地图 URL 的名称和值	
	→ 手动操作脚本事件	
		{PROFILE.SOFTWARE<1-9>} 已弃用。
{INVENTORY.SOFTWARE.APP.A}	→ 基于触发器的通知和命令	主机清单中的软件应用程序 *A* 的字段。
	→ 故障更新通知和命令	
	→ 内部通知	该宏可以与数字索引一起使用，例如 {INVENTORY.SOFTWARE.APP.A<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
	→ 标签的名称和值	
	→ 地图元素标签, 地图 URL 的名称和值	
	→ 手动操作脚本事件	
{INVENTORY.SOFTWARE.APP.B}	→ 基于触发器的通知和命令	主机清单中的软件应用程序 *B* 的字段。
	→ 故障更新通知和命令	
	→ 内部通知	该宏可以与数字索引一起使用，例如 {INVENTORY.SOFTWARE.APP.B<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
	→ 标签的名称和值	
	→ 地图元素标签, 地图 URL 的名称和值	
	→ 手动操作脚本事件	
{INVENTORY.SOFTWARE.APP.C}	→ 基于触发器的通知和命令	主机清单中的软件应用程序 *C* 的字段。
	→ 故障更新通知和命令	
	→ 内部通知	该宏可以与数字索引一起使用，例如 {INVENTORY.SOFTWARE.APP.C<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
	→ 标签的名称和值	
	→ 地图元素标签, 地图 URL 的名称和值	
	→ 手动操作脚本事件	
{INVENTORY.SOFTWARE.APP.D}	→ 基于触发器的通知和命令	主机清单中的软件应用程序 *D* 的字段。
	→ 故障更新通知和命令	
	→ 内部通知	该宏可以与数字索引一起使用，例如 {INVENTORY.SOFTWARE.APP.D<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
	→ 标签的名称和值	
	→ 地图元素标签, 地图 URL 的名称和值	
	→ 手动操作脚本事件	
{INVENTORY.SOFTWARE.APP.E}	→ 基于触发器的通知和命令	主机清单中的软件应用程序 *E* 的字段。
	→ 故障更新通知和命令	
	→ 内部通知	该宏可以与数字索引一起使用，例如 {INVENTORY.SOFTWARE.APP.E<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
	→ 标签的名称和值	
	→ 地图元素标签, 地图 URL 的名称和值	
	→ 手动操作脚本事件	

宏名称	支持该宏的功能	具体描述
{INVENTORY.SOFTWARE.FULL}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 内部通知 → 标签的名称和值 → 地图元素标签, 地图 URL 的名称和值 → 手动操作脚本事件	主机清单中的软件（完整细节描述）字段。 该宏可以与数字索引一起使用，例如 {INVENTORY.SOFTWARE.FULL<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
{INVENTORY.TAG}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 内部通知 → 标签的名称和值 → 地图元素标签, 地图 URL 的名称和值 → 手动操作脚本事件	主机清单中的标签字段。 该宏可以与数字索引一起使用，例如 {INVENTORY.TAG<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。 {PROFILE.TAG<1-9>} 已弃用。
{INVENTORY.TYPE}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 内部通知 → 标签的名称和值 → 地图元素标签, 地图 URL 的名称和值 → 手动操作脚本事件	主机清单中的类型字段。 该宏可以与数字索引一起使用，例如 {INVENTORY.TYPE<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。 {PROFILE.DEVICETYPE<1-9>} 已弃用。
{INVENTORY.TYPE.FULL}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 内部通知 → 标签的名称和值 → 地图元素标签, 地图 URL 的名称和值 → 手动操作脚本事件	主机清单中类型（完整详细信息）字段的内容。 该宏可以与数字索引一起使用，例如 {INVENTORY.TYPE.FULL<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
{INVENTORY.URL.A}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 内部通知 → 标签的名称和值 → 地图元素标签, 地图 URL 的名称和值 → 手动操作脚本事件	主机清单中 *URL A* 的字段内容。 该宏可以与数字索引一起使用，例如 {INVENTORY.URL.A<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
{INVENTORY.URL.B}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 内部通知 → 标签的名称和值 → 地图元素标签, 地图 URL 的名称和值 → 手动操作脚本事件	主机清单中 *URL B* 的字段内容。 该宏可以与数字索引一起使用，例如 {INVENTORY.URL.B<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
{INVENTORY.URL.C}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 内部通知 → 标签的名称和值 → 地图元素标签, 地图 URL 的名称和值 → 手动操作脚本事件	主机清单中 *URL C* 的字段内容。 该宏可以与数字索引一起使用，例如 {INVENTORY.URL.C<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
{INVENTORY.VENDOR}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 内部通知 → 标签的名称和值 → 地图元素标签, 地图 URL 的名称和值 → 手动操作脚本事件	主机清单中的供应商字段的内容 该宏可以与数字索引一起使用，例如 {INVENTORY.VENDOR<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
{ITEM.DESCRIPTION}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 内部通知 → 手动操作脚本事件	引发通知的触发器表达式中的第 *N* 个监控项的描述。 该宏可以与数字索引一起使用，例如 {ITEM.DESCRIPTION<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。

宏名称	支持该宏的功能	具体描述
{ITEM.DESCRIPTION.ORIG}	基于触发器的通知和命令 → 故障更新通知和命令 → 内部通知 → 手动操作脚本事件	引发通知的触发器表达式中的第 *N* 个监控项的描述（带有未解析的宏）。 该宏可以与数字索引一起使用，例如 {ITEM.DESCRIPTION.ORIG<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。 支持的最低版本是 5.2.0.
{ITEM.ID}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 内部通知 → 脚本类型 item、item 原型和发现规则参数的名称和值[6] → HTTP agent 类型监控项、监控项原型和发现规则字段: URL, 查询字段，请求正文,headers, 代理 proxy,SSL 证书文件,SSL 密钥文件 → 手动操作脚本事件	引发通知的触发器表达式中的第 *N* 个监控项的序列 *ID*。 该宏可以与数字索引一起使用，例如 {ITEM.ID<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
{ITEM.KEY}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 内部通知 → 脚本类型 item、item 原型和发现规则参数的名称和值[6] → HTTP agent 类型监控项、监控项原型和发现规则字段: URL, 查询字段，请求正文,headers, 代理 proxy,SSL 证书文件,SSL 密钥文件 → 手动操作脚本事件	触发器表达式中导致通知的第 *N* 监控项的键。 该宏可以与数字索引一起使用，例如 {ITEM.KEY<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。 {TRIGGER.KEY} 已弃用。
{ITEM.KEY.ORIG}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 内部通知 → 脚本类型 item、item 原型和发现规则参数的名称和值[6] → HTTP agent 类型监控项、监控项原型和发现规则字段: URL, 查询字段，请求正文,headers, 代理 proxy,SSL 证书文件,SSL 密钥文件, 允许的主机。[6] → 手动操作脚本事件	触发器表达式中导致通知的第 *N* 监控项的原始键（宏未展开）。[4]. 该宏可以与数字索引一起使用，例如 {ITEM.KEY.ORIG<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
{ITEM.LASTVALUE}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 触发器名字，事件名字，操作数据和描述。 → 标签的名称和值 → 触发器的 URL 地址 → 手动操作脚本事件	触发器表达式中导致通知的第 *N* 项的最新值。 如果收集到的最新历史值超过了 最大历史显示周期，在前端就会解析为 *UNKNOWN* 。(请查看管理 → 通用 菜单选项)。 请注意，从 4.0 开始，在问题名称中使用时，在查看问题事件时不会解析为最新的监控项值，而是保留问题发生时的监控项值。 它是此 last(/{HOST.HOST}/{ITEM.KEY}) 的别名。 从 Zabbix 的版本 3.2.0 开始，该宏支持自定义宏的值； 该宏可以与数字索引一起使用，例如 {ITEM.LASTVALUE<1-9>} 指向触发器表达式中的第一个、第二个、第三个等监控项。请参考宏索引。
{ITEM.LOG.AGE}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 触发器名称、操作数据和描述 → 触发器的 URL 地址 → 事件的 tags 和值 → 手动操作脚本事件	监控项事件日志的持续时间，可以精确到秒。 该宏可以与数字索引一起使用，例如 {ITEM.LOG.AGE<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。

宏名称	支持该宏的功能	具体描述
{ITEM.LOG.DATE}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 触发器名称、操作数据和描述 → 触发器的 URL 地址 → 事件的 tags 和值 → 手动操作脚本事件	监控项事件日志的日期。时间格式类似于 yyyy:mm:dd 。 该宏可以与数字索引一起使用，例如 {ITEM.LOG.DATE<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
{ITEM.LOG.EVENTID}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 触发器名称、操作数据和描述 → 触发器的 URL 地址 → 事件的 tags 和值 → 手动操作脚本事件	事件日志中的事件 *ID*。 仅适用于 windows 事件日志监控。 该宏可以与数字索引一起使用，例如 {ITEM.LOG.EVENTID<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
{ITEM.LOG.NSEVERITY}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 触发器名称、操作数据和描述 → 触发器的 URL 地址 → 事件的 tags 和值 → 手动操作脚本事件	事件日志中日志的严重级别，用数字级别描述（*1--7*）。 仅适用于 windows 事件日志监控。 该宏可以与数字索引一起使用，例如 {ITEM.LOG.NSEVERITY<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
{ITEM.LOG.SEVERITY}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 触发器名称、操作数据和描述 → 触发器的 URL 地址 → 事件的 tags 和值 → 手动操作脚本事件	事件日志中日志的严重级别，使用文字描述。 仅适用于 windows 事件日志监控。 该宏可以与数字索引一起使用，例如 {ITEM.LOG.SEVERITY<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
{ITEM.LOG.SOURCE}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 触发器名称、操作数据和描述 → 触发器的 URL 地址 → 事件的 tags 和值 → 手动操作脚本事件	事件日志中事件的源 仅适用于 windows 事件日志监控。 该宏可以与数字索引一起使用，例如 {ITEM.LOG.SOURCE<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
{ITEM.LOG.TIME}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 触发器名称、操作数据和描述 → 触发器的 URL 地址 → 事件的 tags 和值 → 手动操作脚本事件	*item* 事件日志的发生时间。 该宏可以与数字索引一起使用，例如 {ITEM.LOG.TIME<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
{ITEM.NAME}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 内部通知 → 手动操作脚本事件	触发器表达式中导致通知的第 *N* 个监控项的名称。 该宏可以与数字索引一起使用，例如 {ITEM.NAME<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
{ITEM.NAME.ORIG}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 内部通知 → 手动操作脚本事件	从 Zabbix 6.0 之后此宏已弃用。当监控项名称中支持用户宏和位置宏时，它用于在 Zabbix 6.0 之前的版本中解析为监控项的原始名称（即没有解析宏）。 该宏可以与数字索引一起使用，例如 {ITEM.NAME.ORIG<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。

宏名称	支持该宏的功能	具体描述
{ITEM.STATE}	→ 基于监控项的内部通知	触发器表达式中导致通知的第 *N* 监控项的最新状态。可能的值: 不支持和 正常. 该宏可以与数字索引一起使用，例如 {ITEM.STATE<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
{ITEM.VALUE}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 触发器名字，事件名字，操作数据和描述。 → 标签的名称和值 → 触发器的 URL 地址 → 手动操作脚本事件	可被解释为如下的任意一项: 1) 如果在触发器状态更改的上下文中使用，则解释为触发器表达式中第 N 个监控项的历史值（事件发生时），例如在展示事件或发送通知时。 2) 如果在没有触发器状态更改的上下文的情况下使用，触发器表达式中第 N 个监控项的最新值。例如，在弹出选择窗口中显示触发器列表时。在本例中工作方式与 {ITEM.LASTVALUE} 相同。在第一种情况下，如果历史记录值已被删除或从未存储过，它将解释为 * 未知 *。 在第二种情况下并且仅在前端中，如果收集的最新历史值超过最大历史显示周期时间，它将解析为 * 未知 *. (查看管理 → 通用 菜单选项). 自定义 此宏支持宏值，从 Zabbix 3.2.0 开始。 该宏可以与数字索引一起使用，例如 {ITEM.VALUE<1-9>} 指向触发器表达式中的第一个、第二个、第三个等监控项。请参考宏索引。
{ITEM.VALUETYPE}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 内部通知 → 手动操作脚本事件	触发器表达式中导致通知的第 *N* 个监控项的值类型。可能的值: 0 - numeric float, 1 - character, 2 - log, 3 - numeric unsigned, 4 - text。 该宏可以与数字索引一起使用，例如 {ITEM.VALUETYPE<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
{LLDRULE.DESCRIPTION}	→ 基于 LLD 规则的内部通知	支持的最低版本是 5.4.0. 引发通知的低级别自动发现规则的描述。
{LLDRULE.DESCRIPTION.ORIG}	→ 基于 LLD 规则的内部通知	引发通知的低级别自动发现规则的描述（描述中带有未解析的宏） 支持的最低版本是 5.2.0.
{LLDRULE.ID}	→ 基于 LLD 规则的内部通知	引发通知的低级别自动发现规则的规则序列 *ID*
{LLDRULE.KEY}	→ 基于 LLD 规则的内部通知	引发通知的低级别自动发现规则的键值
{LLDRULE.KEY.ORIG}	→ 基于 LLD 规则的内部通知	引发通知的低级别自动发现规则的原始密钥（未扩展的宏）。
{LLDRULE.NAME}	→ 基于 LLD 规则的内部通知	导致通知的低级发现规则（已解析宏）的名称。
{LLDRULE.NAME.ORIG}	→ 基于 LLD 规则的内部通知	导致通知的低级发现规则的原始名称（即未解析宏）。
{LLDRULE.STATE}	→ 基于 LLD 规则的内部通知	低级发现规则的最新状态。可能的值: 不支持和正常.
{MAP.ID}	→ 地图元素标签, 地图 URL 的名称和值	网络拓扑图 *ID*
{MAP.NAME}	→ 地图元素标签, 地图 URL 的名称和值 → Text field in map shapes	网络拓扑图名称 支持的最低版本是 3.4.0.

宏名称	支持该宏的功能	具体描述
{PROXY.DESCRIPTION}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 发现的通知和命令 → 自动注册的通知和命令 → 内部通知 → 手动操作脚本事件	代理的描述。可以被解释为如下的可能: 1) 触发器表达式中第 N 个监控项的代理（在基于触发器的通知中）。您可以在此处使用索引 宏。 2) 代理，它执行发现 (在发现通知中)，在此处使用 {PROXY.DESCRIPTION} ，无需索引。 3) 代理指向一个已经注册的 agent (在自动注册通知中)。在此处使用 {PROXY.DESCRIPTION}，无需索引。 该宏可以与数字索引一起使用，例如 {PROXY.DESCRIPTION<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
{PROXY.NAME}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 发现的通知和命令 → 自动注册通知和命令 → 内部通知 → 手动操作脚本事件	代理的名称. 可以被解释为如下的可能:: 1) 触发器表达式中第 N 个监控项的代理名称（在基于触发器的通知中）。您可以在此处使用索引 宏。 2) 代理，它执行发现（在发现通知中）。在此处使用 {PROXY.NAME}，无需索引。 3) 代理指向一个已经注册的 agent (在自动注册通知中)。在此处使用 {PROXY.NAME} ，无需索引。 该宏可以与数字索引一起使用，例如 {PROXY.NAME<1-9>} 指向触发器表达式中匹配的第 1 个，第 2 个，第 3 个等主机。请参考宏索引。
{SERVICE.NAME}	→ 基于服务的通知和命令 → 服务的通知和命令	服务名称 (已解析宏)。
{SERVICE.ROOTCAUSE}	→ 基于服务的通知和命令 → 服务的通知和命令	导致服务失败的触发问题事件列表, 使用安全性主机排序。包括以下详细信息：主机名、事件名称、严重性、持续时间、服务标签和值。
{SERVICE.TAGS}	→ 基于服务的通知和命令 → 服务的通知和命令	服务事件标签的逗号分隔列表。服务事件标签可以在服务配置选项标签中定义。如果不存在标签，则扩展为空字符串。
{SERVICE.TAGSJSON}	→ 基于服务的通知和命令 → 服务的通知和命令	包含服务事件标签对象的 *JSON* 数组。服务事件标签可以在服务配置选项标签中定义。如果不存在标签，则扩展为空数组。
{SERVICE.TAGS.<tag name>}	→ 基于服务的通知和命令 → 服务的通知和命令	标签名称引用的服务事件标签值。服务事件标签可以在服务配置选项标签中定义。 包含非字母数字字符（包括非英语多字节 UTF 字符）的标签名称应该用双引号引起来。带引号的标记名称中的引号和反斜杠必须使用反斜杠进行转义。
{TIME}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 基于服务的通知和命令 → 服务的通知和命令 → 发现的通知和命令 → 自动注册通知和命令 → 内部通知 → 触发器事件名称 → 手动操作脚本事件	现在的时间，格式为 *hh:mm:ss*
{TRIGGER.DESCRIPTION}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 基于触发器的内部通知 → 手动操作脚本事件	触发器描述。 如果在通知中使用了 {TRIGGER.DESCRIPTION}，则触发器描述中支持的所有宏都将被扩展。 {TRIGGER.COMMENT} 已弃用。
{TRIGGER.EXPRESSION.EXPLAIN}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 手动操作脚本事件 → 事件名称	部分评估的触发器表达式。 基于监控项的函数在事件生成时被评估并替换为结果，而所有其他函数都显示为表达式中所写的。 可用于调试触发器表达式。

宏名称	支持该宏的功能	具体描述
{TRIGGER.EXPRESSION.RECOVERY.EXPLAIN}	→ Problem recovery notifications and commands → 故障更新通知和命令 → 手动操作脚本事件	部分评估触发恢复表达式. 基于触发器的函数在事件生成时被评估并替换为结果，而所有其他函数都显示为表达式中所写的。 可用于调试触发恢复表达式。
{TRIGGER.EVENTS.ACK}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 地图元素标签 → 手动操作脚本事件	地图中地图元素或在通知中生成当前事件的触发器的已确认事件数。
{TRIGGER.EVENTS.PROBLEM.ACK}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 地图元素标签 → 手动操作脚本事件	所有触发器的已确认 *PROBLEM* 事件的数量，无论其状态。
{TRIGGER.EVENTS.PROBLEM.UNACK}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 地图元素标签 → 手动操作脚本事件	所有触发器的未确认问题事件数，无论其状态。
{TRIGGER.EVENTS.UNACK}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 地图元素标签 → 手动操作脚本事件	地图中地图元素或在通知中生成当前事件的触发器的未确认事件数。
{TRIGGER.HOSTGROUP.NAME}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 基于触发器的内部通知 → 手动操作脚本事件	定义了触发器的已排序（按 *SQL* 查询）、以逗号分隔的主机组列表名称。
{TRIGGER.PROBLEM.EVENTS.ACK}	→ 地图元素标签	处于 *PROBLEM* 状态的触发器已确认的 *PROBLEM* 事件数。
{TRIGGER.PROBLEM.EVENTS.UNACK}	→ 地图元素标签	处于 *PROBLEM* 状态的触发器的未确认 *PROBLEM* 事件数。
{TRIGGER.EXPRESSION}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 基于触发器的内部通知 → 手动操作脚本事件	触发器表达式。
{TRIGGER.EXPRESSION.RECOVERY.EXPLAIN}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 基于触发器的内部通知 → 手动操作脚本事件	触发恢复表达式如果事件生成 *OK* 在触发器配置被用于' 恢复表达'; 否者返回空的 string 字段。 支持的最低版本是 3.2.0.
{TRIGGER.ID}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 基于触发器的内部通知 → 地图元素标签, 地图 URL 的名称和值 → 触发器的 URL 地址 → Trigger tag value → 手动操作脚本事件	触发此操作的触发器 *ID*。 从 Zabbix 的 4.4.1 版本开始在触发标记值中受支持。
{TRIGGER.NAME}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 基于触发器的内部通知 → 手动操作脚本事件	触发器的名称 (已解析宏). 请注意自 4.0.0 开始，{EVENT.NAME} 可用于操作以显示已解析的宏触发的事件/问题名称。
{TRIGGER.NAME.ORIG}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 基于触发器的内部通知 → 手动操作脚本事件	触发器的原始名称 (即未解析宏).
{TRIGGER.NSEVERITY}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 基于触发器的内部通知 → 手动操作脚本事件	使用数字表达的触发严重性可能的值:
{TRIGGER.SEVERITY}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 基于触发器的内部通知 → 手动操作脚本事件	触发器严重性名称可以在这里定义 管理 → 通用 → 触发器展示选项.
{TRIGGER.STATE}	→ 基于触发器的内部通知	触发器的最新状态可能的值: 未知和 正常.

宏名称	支持该宏的功能	具体描述
{TRIGGER.STATUS}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 手动操作脚本事件	当前触发器的值可以是 PROBLEM 或者 OK. {STATUS} 已弃用。
{TRIGGER.TEMPLATE.NAME}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 基于触发器的内部通知 → 手动操作脚本事件	值为已排序（按 *SQL* 查询）、逗号分隔的模板列表中定义了触发器，如果触发器是在主机中定义的则为 * 未知 *。
{TRIGGER.URL}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 基于触发器的内部通知 → 手动操作脚本事件	触发器的 *URL* 地址。
{TRIGGER.VALUE}	→ 基于触发器的通知和命令 → 故障更新通知和命令 → 触发器表达式 → 手动操作脚本事件	当前触发数值: 0 - 触发器处于正常状态, 1 - 触发器处于问题状态。
{TRIGGERS.UNACK}	→ 地图元素标签	地图元素中已取消确认问题触发器的数量，忽略触发器状态。 如果至少有一个问题事件已取消确认，则触发器被视为已取消确认。
{TRIGGERS.PROBLEM.UNACK}	→ 地图元素标签	在地图标签中未确认的触发器（状态为 *PROBLEM*）总数。 如果至少有一个问题事件已取消确认，则触发器被视为已取消确认。
{TRIGGERS.ACK}	→ 地图元素标签	地图元素中已确认触发器的数量, 忽略触发器状态。 如果触发器的所有问题事件都被确认，则认为触发器已被确认。
{TRIGGERS.PROBLEM.ACK}	→ 地图元素标签	地图元素中已确认触发器的数量。 如果触发器的所有问题事件都被确认，则认为触发器已被确认。
{USER.FULLNAME}	→ 故障更新通知和命令 → 对主机手动操作脚本 (including confirmation text) → 手动操作脚本事件 (including confirmation text)	在事件确认和脚本操作中添加用户的姓氏、名字和完整姓名的信息。 从 Zabbix 的 3.4.0 版本开始在问题更新功能中支持此宏，从 5.0.2 开始在全局功能中支持。
{USER.NAME}	→ 对主机手动操作脚本 (including confirmation text) → 手动操作脚本事件 (including confirmation text)	启动脚本的用户的名字。 支持的最低版本是 5.0.2.
{USER.SURNAME}	→ 对主机手动操作脚本 (including confirmation text) → 手动操作脚本事件 (including confirmation text)	启动脚本的用户的姓氏。 支持的最低版本是 5.0.2.
{USER.USERNAME}	→ 对主机手动操作脚本 (including confirmation text) → 手动操作脚本事件 (including confirmation text)	启动脚本的完整用户名称（包括姓氏和名字）。 支持的最低版本是 5.0.2. {USER.ALIAS} 仅支持 Zabbix 5.4.0 以前的版本, 此版本已弃用。
{$MACRO}	→ 参阅: User macros supported by location	用户自定义 宏。
{#MACRO}	→ 参阅: 低级别自动发现宏	低级别自动发现宏。
{?EXPRESSION}	→ 触发器事件名称 → 基于触发器的通知和命令 → 故障更新通知和命令 → 地图元素标签[3] → 地图分享标签[3] → 地图中的链接标签[3] → 图表名称[5]	定制 宏的值, 从 Zabbix 的 4.0.0 版本开始。 表达式宏用于公式计算。通过扩展内部的所有宏并评估结果表达式来计算。 参阅案例 在事件名称中使用表达式宏。 支持的最低版本是 5.2.0.

脚注

[1] 监控项关键参数中支持的 {HOST.*} 宏将解析为监控项选择的接口。当在没有接口的监控项中使用时，它们将按此优先级顺序解析为主机的 Zabbix agent，SNMP，JMX 或 IPMI 接口，如果主机没有任何接口，则解析为 “未知”。

[2] 在全局脚本、接口 IP/DNS 字段和 Web 场景中，宏将解析为主代理接口，但是，如果不存在，将使用主 SNMP 接口。如果 SNMP 也不存在，则将使用主 JMX 接口。如果 JMX 也不存在，则将使用主 IPMI 接口。如果主机没有任何接口，则宏解析为 “未知”。

[3] 此宏在 map 标签中仅支持以秒为参数的 **avg**, **last**, **max** 和 **min** 函数。

[4] 在 web 场景中变量，消息头，*SSL* 证书文件和 *SSL* 秘钥文件字段以及场景步骤中 *URL*，*Post*，消息头和 必填字符串字段均支持 {HOST.*} 宏。从 Zabbix 5.4.0 开始，web 场景 名称和 web 场景步骤 * 名称字段不再支持 {HOST.*} 宏。

[5] 此宏在图形名称中仅支持以秒为参数的 **avg**, **last**, **max** 和 **min** 函数。{HOST.HOST<1-9>} 宏可用作宏内的主机。例如：

```
* last(/Cisco switch/ifAlias[{#SNMPINDEX}])
* last(/{HOST.HOST}/ifAlias[{#SNMPINDEX}])
```

[6] 从 5.2.5 版本开始支持。

索引宏

宏索引 {MACRO<1-9>} 语法仅限于触发器表达式的上下文。它能用于按顺序引用表达式中包含的设备。例如：在表达式中包含了设备 1，设备 2，设备 3，那么宏 {HOST.IP1},{HOST.IP2}, {HOST.IP3} 将分别引用设备 1，设备 2，设备 3 的 IP 地址信息。在提供触发器表达式包含这些函数时，宏 {FUNCTION.VALUE1}，{FUNCTION.VALUE2}，{FUNCTION.VALUE3} 将解析为事件发生时触发器表达式中第一个、第二个和第三个基于监控项函数的值。

另外，可以在图形名称中使用宏 {?func(/host/key,param)} ，同时再叠加使用宏 {HOST.HOST<1-9>}。例如，图形名称中的宏 {{HOST.HOST2}:key.func()} 代表引用图形中的第二个监控项。

Warning:
除了这里提到的两种情况外，索引宏不会在任何其他场境中解析。对于其他场景可以使用不带索引的宏。(即 {HOST.HOST}，{HOST.IP}，等)

2 支持用户自定义宏的位置

概述

用户自定义 宏可以用于以下场景。

Note:
动作、网络发现、*Proxies* 以及本页 其他位置下列出的内容仅支持全局级用户宏。在这些提到的位置，主机级和模板级宏将不会被解析。

动作

在动作中，用户宏可用于以下地方：

位置		多个宏/混合文本[1]
基于触发器的通知和命令		支持
基于触发器的内部通知		支持
问题更新通知		支持
基于服务的通知和命令		支持
服务更新通知		支持
时间段条件		不支持
操作		
	默认操作步骤持续时间	不支持
	步骤持续时间	不支持

主机/主机原型

在主机 和主机原型 配置中，用户宏可用于以下地方:

位置		多个宏/混合文本[1]
接口 IP/DNS		仅支持 DNS
接口端口		不支持
SNMP v1, v2		
	SNMP 团体名	支持
SNMP v3		
	上下文名称	支持
	安全名称	支持
	身份验证密码	支持

位置		多个宏/混合文本[1]
	隐私密码	支持
IPMI		
	用户名	支持
	密码	支持
标签[2]		
	标签名称	支持
	标签值	支持

监控项 / 监控项原型

在监控项 或监控项原型配置中，用户宏可用于以下地方：

位置		多个宏/混合文本[1]
监控项键参数		支持
更新间隔		不支持
自定义间隔		不支持
历史存储周期		不支持
趋势存储周期		不支持
描述		支持
计算监控项		
	公式	支持
数据库监控		
	用户名	支持
	密码	支持
	SQL 查询	支持
HTTP 代理		
	URL[3]	支持
	查询字段	支持
	超时时间	不支持
	请求正文	支持
	请求头部 (名字和值)	支持
	返回状态码	支持
	HTTP 代理	支持
	HTTP 认证用户名	支持
	HTTP 认证密码	支持
	SSI 证书文件	支持
	SSI 密钥文件	支持
	SSI 密钥密码	支持
	允许的主机	支持
JMX 代理		
	JMX 端点	支持
脚本监控项		
	参数名称和值	支持
SNMP 代理		
	SNMP OID	支持
SSH 代理		
	用户名	支持
	公钥文件	支持
	私钥文件	支持
	密码	支持
	脚本	支持
TELNET 代理		
	用户名	支持
	密码	支持
	脚本	支持
Zabbix 采集器		
	允许的主机	支持
标签[2]		
	标签名称	支持
	标签值	支持

位置		多个宏/混合文本[1]
预处理		
	步骤参数（包括自定义脚本）	支持

底层自动发现

在底层自动发现规则中，用户宏可用于以下地方：

位置		多个宏/混合文本[1]
键参数		支持
更新间隔		不支持
自定义间隔		不支持
保留失效资源的时间		不支持
描述		支持
SNMP 代理		
	SNMP OID	支持
SSH 代理		
	用户名	支持
	公钥文件	支持
	私钥文件	支持
	密码	支持
	脚本	支持
TELNET 代理		
	用户名	支持
	密码	支持
	脚本	支持
Zabbix 采集器		
	允许的主机	支持
数据库监控		
	用户名	支持
	密码	支持
	SQL 查询	支持
JMX 代理		
	JMX 端点	支持
HTTP 代理		
	URL[3]	支持
	查询字段	支持
	超时时间	不支持
	请求正文	支持
	请求头部 (名字和值)	支持
	返回状态码	支持
	HTTP 认证用户名	支持
	HTTP 认证密码	支持
过滤器		
	正则表达式	支持
覆盖		
	过滤器：正则表达式	支持
	操作: 更新间隔 (对于监控项原型)	不支持
	操作: 历史存储周期 (对于监控项原型)	不支持
	操作: 趋势存储周期 (对于监控项原型)	不支持

网络发现

在网络发现规则中，用户宏可用于以下地方：

位置		多个宏/混合文本[1]
更新间隔		不支持
SNMP v1, v2		
	SNMP 团体名	支持
	SNMP OID	支持

位置		多个宏/混合文本[1]
SNMP v3		
	上下文名称	支持
	安全名称	支持
	身份验证密码	支持
	隐私密码	支持
	SNMP OID	支持

代理

在proxy配置中，用户宏可用于以下地方：

位置	多个宏/混合文本[1]
接口端口（用于被动代理）	不支持

模板

在模板 配置中，用户宏可用于以下地方：

位置		多个宏/混合文本[1]
标签[2]		
	标签名称	支持
	标签值	支持

触发器

在触发器 配置中，用户宏可用于以下地方：

位置		多个宏/混合文本[1]
名称		支持
操作数据		支持
表达式 (仅在常量和函数参数中; 不支持加密的宏).		支持
描述		支持
URL[3]		支持
匹配标签		支持
标签[2]		
	标签名字	支持
	标签值	支持

Web 场景

在web 场景配置中，用户宏可用于以下地方：

位置		多个宏/混合文本[1]
名称		支持
更新间隔		不支持
Agent		支持
HTTP 代理		支持
变量 (只允许值)		支持
请求头部 (名称和值)		支持
步骤		
	名称	支持
	URL[3]	支持
	变量 (只允许值)	支持
	请求头部 (名称和值)	支持
	超时时间	不支持
	返回字符串	支持
	返回状态码	不支持

位置		多个宏/混合文本[1]
认证		
	用户	支持
	密码	支持
	SSL 证书	支持
	SSL 密钥文件	支持
	SSL 密钥密码	支持
标签		
	标签名称	支持
	标签值	支持

其他位置

除了以上列出的位置外，用户宏还可用于以下地方：

位置		多个宏/混合文本[1]
全局脚本 (脚本, SSH, Telnet, IPMI), 包括确认文本		支持
Webhooks		
	JavaScript 脚本	不支持
	JavaScript 脚本参数名称	不支持
	JavaScript 脚本参数值	支持
监控 → 仪表盘		
	监控项键值仪表板组件的描述字段	支持
	动态 *URL* 仪表板组件的 URL[3] 字段	支持
管理 → 用户 → 媒介		
	活动时间	不支持
管理 → 一般 → 界面设置		
	工作时间	不支持
管理 → 媒介类型 → 消息模板		
	主题	支持
	消息	支持

有关 Zabbix 支持的所有宏的完整列表，请参阅支持的宏。

附注

1 如果该位置不支持字段中的多个宏或与混合文本的宏，则必须使用单个宏填充整个字段。

2 URLs 包含秘密宏 将不起作用，因为其中的宏将被解析为”******”。

8 单位符号

概述

当必须使用一些大数字，例如 “86400” 来表示一天中的秒数，既困难又容易出错。这就需要使用一些单位符号来简化 Zabbix 触发器表达式和监控项键。

可以简单地输入 “1d”，而不是 “86400” 秒。后缀起到乘数的作用。

时间单位

对于时间可以使用：

- **s** - 秒（使用时与原始值相同）
- **m** - 分
- **h** - 小时
- **d** - 天
- **w** - 周

时间单位仅支持整数（因此支持 “1h”，不支持 “1,5h” 或 “1.5h”；请改用 “90m”）。

时间单位支持：

- 触发器表达式 常量和函数参数

- 计算监控项 公式的常数
- **zabbix[queue,<from>,<to>]**内部监控项的参数
- 聚合计算的时间段参数
- 监控项配置（’ 更新间隔’, ’ 自定义间隔’, ’ 历史存储期’ 和’ 趋势存储期’ 字段）
- 监控项原型配置（’ 更新间隔’, ’ 自定义间隔’, ’ 历史存储期’ 和’ 趋势存储期’ 字段）
- 底层自动发现规则配置（’ 更新间隔’, ’ 自定义间隔’, ’ 保留丢失的资源’ 字段）
- 网络自动发现配置（’ 更新间隔’ 字段）
- Web 场景配置（’ 更新间隔’, ’ 超时’ 字段）
- 动作操作配置（’ 默认操作步骤持续时间’,’ 步骤持续时间’ 字段）
- 用户配置文件设置（’ 自动注销’, ’ 刷新’, ’ 消息超时’ 字段）
- 监测 → 仪表盘的图表部件 （’ 时移’ 字段）
- 管理 → 一般 → 管家（存储期字段）
- 管理 → 一般 → 触发器显示选项 *（’ 显示 OK 触发器’, ’ 状态更改触发器闪烁’ 字段）
- 管理 → 一般 → 其他（’ 登录阻塞间隔’ 字段和 Zabbix server 通信相关的字段）
- Zabbix 服务器 `ha_set_failover_delay=delay` 运行时控制 选项

内存单位

内存大小单位支持：

- 触发器表达式 常量和函数参数
- 计算监控项 公式的常数

对于内存大小可以使用：

- **K** - 千字节
- **M** - 兆字节
- **G** - 吉字节
- **T** - 太字节

其他用途

单位符号也用于前端数据可读的表示法。

Zabbix server 和前端都支持这些符号：

- **K** - 千
- **M** - 兆
- **G** - 吉
- **T** - 太

当监控项值在前端显示为 B、Bps 时, 是基于二进制的 (1K = 1024)。其他情况基于十进制 (1K = 1000)。

此外，前端还支持显示：

- **P** - 拍
- **E** - 艾
- **Z** - 泽
- **Y** - 尧

使用示例

通过使用一些适当的单位，可以编写更易于理解和维护的触发器表达式，例如这些表达式：

```
last(/host/system.uptime[])<86400s
avg(/host/system.cpu.load,600s)<10
last(/host/vm.memory.size[available])<20971520
```

可以改为：

```
last(/host/system.uptime[])<1d
avg(/host/system.cpu.load,10m)<10
last(/host/vm.memory.size[available])<20M
```

Preventing unit conversion

By default, specifying a unit for an item results in a multiplier prefix being added - for example, an incoming value ’2048’ with unit ’B’ would be displayed as ’2KB’ by the frontend (for more details, see *Units* in item configuration).

To prevent a unit from conversion, use the ！ prefix, for example, !B. To better understand how the conversion works with and without the exclamation mark, see the following examples of values and units:

```
1024 !B → 1024 B
1024 B → 1 KB
61 !s → 61 s
61 s → 1m 1s
0 !uptime → 0 uptime
0 uptime → 00:00:00
0 !! → 0 !
0 ! → 0
```

Note:
Before Zabbix 4.0, there was a hardcoded unit stoplist consisting of `ms`, `rpm`, `RPM`, `%`. This stoplist has been deprecated, thus the correct way to prevent converting such units is `!ms`, `!rpm`, `!RPM`, `!%`.

9 时间段语法

概述

要设置时间段，必须使用以下格式：

`d-d,hh:mm-hh:mm`

其中符号代表以下内容：

符号	描述
d	星期几: 1 - 星期一, 2 - 星期二,... , 7 - 星期日
hh	小时: 00-24
mm	分钟: 00-59

可以使用分号 (;) 分隔符指定多个时间段：

`d-d,hh:mm-hh:mm;d-d,hh:mm-hh:mm...`

将时间段不做配置时等于 01-07，00:00-24:00，这是默认值。

Attention:
不包括时间段的上限。因此，如果您指定 09:00-18:00，则时间段中包含的最后一秒是 17:59:59。

示例

工作时间。周一至周五 9:00 到 18:00：

`1-5,09:00-18:00`

工作时间加上周末。周一至周五 9:00 到 18:00 和周六、周日 10:00 到 16:00：

`1-5,09:00-18:00;6-7,10:00-16:00`

10 命令执行

Zabbix 使用通用功能来进行外部检查、用户参数、system.run 监控项、自定义告警脚本、远程命令和全局脚本。

执行步骤

Note:
默认情况下，Zabbix 使用 *sh* shell 执行所有脚本，且无法修改默认 shell。要使用不同的 shell，可以采用一个变通方法：创建一个脚本文件并在命令执行期间调用该脚本。

命令/脚本在 Unix 和 Windows 平台上的执行方式类似：

1. Zabbix（父进程）创建一个管道以进行通信
2. Zabbix 将管道设置为即将创建的子进程的输出

3. Zabbix 创建子进程（运行命令/脚本）
4. 为子进程创建一个新进程组（在 Unix 中）或作业（在 Windows 中）
5. Zabbix 从管道中读取数据，直到发生超时或另一端没有写入（所有句柄/文件描述符都已关闭）。注意，子进程可以创建更多进程并在它们退出或关闭句柄/文件描述符之前退出。
6. 如果未达到超时时间，Zabbix 等待初始子进程退出或超时发生
7. 如果初始子进程已退出且未达到超时时间，Zabbix 检查初始子进程的退出代码并将其与 0 比较（非零值被视为执行失败，仅适用于在 Zabbix server 和 Zabbix proxy 上执行的自定义告警脚本、远程命令和用户脚本）
8. 此时假定所有操作已完成，整个进程树（即进程组或作业）被终止

Attention:
Zabbix 假定当初始子进程已退出并且没有其他进程仍然保持输出句柄/文件描述符打开时，命令/脚本已完成处理。处理完成后，终止所有创建的进程。

命令中所有的双引号和反斜杠都使用反斜杠进行转义，并将命令用双引号包围。

退出代码检查

退出代码在以下条件下进行检查：

- 仅用于在 Zabbix server 和 Zabbix proxy 上执行的自定义告警脚本、远程命令和用户脚本。
- 任何不为 0 的退出代码都被视为执行失败。
- 失败执行的标准错误和标准输出将被收集，并在前端界面中展示（显示执行结果的地方）。
- 对于 Zabbix server 上的远程命令，将创建额外的日志条目以保存脚本执行输出，可以使用 LogRemoteCommands agent 参数 启用。

前端消息和日志可能出现的失败命令/脚本：

- 执行失败的标准错误和标准输出内容（如果有）。
- ”进程退出代码：N.”（对于空输出，且退出代码不等于 0）。
- ”进程被信号：N.” 终止（对于被信号终止的进程，仅限 Linux）。
- ”进程意外终止。”（对于因未知原因终止的进程）。

另请参阅

- 外部检查
- 用户参数
- system.run 监控项
- 自定义告警脚本
- 远程命令
- 全局脚本

11 版本兼容性

支持的 agents

要与 Zabbix 6.0 兼容，Zabbix agent 的版本不能早于 1.4 版本，也不能晚于 6.0 版本。

您可能需要检查旧版 agents 的配置，因为一些参数已经更改，例如，对于 3.0 之前的版本，与 日志记录 相关的参数。

为了充分利用最新的指标、改进的性能和降低的内存使用，请使用最新支持的 agent。

Windows XP 的注意事项

在 Windows XP/Server 2003 上，不要使用晚于 Zabbix 4.0.x 的 agent 模板。较新的模板使用英文性能计数器，这些仅从 Windows Vista/Server 2008 开始支持。

支持的 agents 2

从 4.4 版开始的旧 Zabbix agents 2 与 Zabbix 6.0 兼容；Zabbix agent 2 不得高于 6.0。

请注意，当使用 Zabbix agent 2 版本 4.4 和 5.0 时，不支持的监控项默认刷新间隔为 10 分钟。

为了充分利用最新的指标、改进的性能和减少的内存使用，请使用最新支持的 agent 2。

支持的 Zabbix 代理

为了兼容 Zabbix 6.0，代理必须是相同的主要版本；因此只有 Zabbix 6.0.x 代理可以与 Zabbix 6.0.x 服务器一起工作。

Attention:
不再可能启动升级后的服务器并让旧的和未升级的代理向较新的服务器报告数据。Zabbix 从未推荐或支持的这种方法现在已被正式禁用，因为服务器将忽略来自未升级代理的数据。另请参阅升级过程。

记录有关使用不兼容的 Zabbix 守护程序版本的警告。

支持的 XML 文件

Zabbix 6.0 支持导入不低于 1.8 版本的 XML 文件。

Attention:
在 XML 导出格式中，触发器依赖项仅按名称存储。如果有多个具有相同名称的触发器（例如，具有不同的严重性和表达式）并且在它们之间定义了依赖关系，则无法导入它们。此类依赖项必须从 XML 文件中手动删除，并在导入后重新添加。

12 数据库错误处理

如果 Zabbix 检测到后端数据库不可访问，它将发送一条通知消息并继续尝试连接到数据库。对于某些数据库引擎，可以识别特定的错误代码。

MySQL

- CR_CONN_HOST_ERROR
- CR_SERVER_GONE_ERROR
- CR_CONNECTION_ERROR
- CR_SERVER_LOST
- CR_UNKNOWN_HOST
- ER_SERVER_SHUTDOWN
- ER_ACCESS_DENIED_ERROR
- ER_ILLEGAL_GRANT_FOR_TABLE
- ER_TABLEACCESS_DENIED_ERROR
- ER_UNKNOWN_ERROR

13 Zabbix sender Windows 动态链接库

在 Windows 环境中，应用程序可以通过使用 Zabbix sender 动态链接库 (zabbix_sender.dll) 直接将数据发送到 Zabbix server/proxy，而不必启动外部进程 (zabbix_sender.exe)。

包含开发文件的动态链接库位于 bin\winXX\dev 文件夹中。要使用它，请包含 zabbix_sender.h 头文件并与 zabbix_sender.lib 库链接。可以在 build\win32\examples\zabbix_sender 文件夹中找到使用 Zabbix sender API 的示例文件。

Zabbix sender 动态链接库提供以下函数：

```
int zabbix_sender_send_values(const char *address, unsigned short port,const char *source, const zabbix_
char **result);‘{.c}
```

Zabbix sender 动态链接库使用以下数据结构：

```
typedef struct
{
    /* host name, must match the name of target host in Zabbix */
    char    *host;
    /* the item key */
    char    *key;
    /* the item value */
    char    *value;
}
zabbix_sender_value_t;

typedef struct
```

```
{
    /* number of total values processed */
    int total;
    /* number of failed values */
    int failed;
    /* time in seconds the server spent processing the sent values */
    double  time_spent;
}
zabbix_sender_info_t;
```

14 Python library for Zabbix API

Overview

[zabbix_utils](https://github.com/zabbix/python-zabbix-utils/blob/main/README.md) is a Python library for working with Zabbix API as well as with Zabbix sender and Zabbix get protocols.

It is supported for Zabbix 5.0, 6.0, 6.4 and later.

14 服务监控升级

概览　在 Zabbix 6.0 中，服务监控 功能进行了重大修改(参见 [What's new in Zabbix 6.0.0](https://www.zabbix.com/documentation/6.0/en/manual/whatsnew600#services) 以获取更改列表)。

此页面描述了在升级到 Zabbix 6.0 或更新版本期间如何更改早期 Zabbix 版本中定义的服务和 SLA。

服务　在旧的 Zabbix 版本中，服务有两种类型的依赖：软依赖和硬依赖。升级后，所有依赖项都将变得相同。

如果一个服务的 "子服务" 先前已通过硬依赖关系链接到 "父级服务 1" 并通过软依赖关系另外链接到 "父级服务 2"，则升级后 "子服务" 将有两个父级服务 "父级服务 1" 和" 父级服务 2"。

问题和服务之间基于触发器的映射已被基于标签的映射所取代。在 Zabbix 6.0 及更新版本中，服务配置表单有一个新参数 问题标签，它允许指定一个或多个标签名称和值对用于问题匹配。已链接到服务的触发器将获得一个新标签 `ServiceLink ：<trigger ID>:<trigger name>`（标签值将被截断为 32 个字符）。链接服务将获得具有相同值的 "ServiceLink"问题标签。

状态计算规则

"状态计算算法" 将使用以下规则进行升级：

- 不计算 → 设置状态为 OK
- 问题，如果至少有一个子服务有问题 → 最关键的子服务
- 问题，如果所以子服务都有问题 → 最关键的是如果所有的子服务都有问题

> **Note:**
> 如果您已从 Zabbix pre-6.0 升级到 Zabbix 6.0.0、6.0.1 或 6.0.2，请参阅已知问题 用于 Zabbix 6.0 文档。

SLAs　以前，必须为每个服务单独定义 SLA 目标。从 Zabbix 6.0 开始，SLA 已成为一个单独的实体，其中包含有关服务计划、预期服务水平目标 (SLO) 和要从计算中排除的停机时间的信息。配置完成后，可以通过服务标签 将 SLA 分配给多个服务。

升级过程中:

- 为每项服务定义的相同 SLA 将被分组，并且将为每个组创建一个 SLA。
- 每个受影响的服务都将获得一个特殊标签 `SLA:<ID>`，并且将在相应 SLA 的 *Service tags* 参数中指定相同的标签。
- 服务创建时间是 SLA 报告中的一个新指标，对于现有服务，将设置为 01/01/2000 00:00。

15 其他问题

Login 和 systemd

我们建议创建 一个 *zabbix* 用户作为系统用户，但不允许登录。一些用户忽略此建议并使用相同的帐户登录（例如使用 SSH）来登录运行 Zabbix 的主机。这可能会使 Zabbix 守护进程在注销时崩溃。在这种情况下，您将在 Zabbix server 日志中获得类似以下内容：

```
zabbix_server [27730]: [file:'selfmon.c',line:375] lock failed: [22] Invalid argument
zabbix_server [27716]: [file:'dbconfig.c',line:5266] lock failed: [22] Invalid argument
zabbix_server [27706]: [file:'log.c',line:238] lock failed: [22] Invalid argument
```

并且在 Zabbix agent 日志中：

```
zabbix_agentd [27796]: [file:'log.c',line:238] lock failed: [22] Invalid argument
```

这是因为默认的 systemd 设置 RemoveIPC=yes 在/etc/systemd/logind.conf 中配置。当您退出系统时，Zabbix 之前创建的信号量会被删除，这会导致崩溃。

来自 systemd 文档的引用：

```
RemoveIPC=

Controls whether System V and POSIX IPC objects belonging to the user shall be removed when the
user fully logs out. Takes a boolean argument. If enabled, the user may not consume IPC resources
after the last of the user's sessions terminated. This covers System V semaphores, shared memory
and message queues, as well as POSIX shared memory and message queues. Note that IPC objects of the
root user and other system users are excluded from the effect of this setting. Defaults to "yes".
```

这个问题有 2 个解决方案：

1. （推荐）停止使用 *zabbix* 帐户来处理除 Zabbix 进程之外的任何事情，为其他事情创建一个专用帐户。
2. （不推荐）设置 RemoveIPC=no 在/etc/systemd/logind.conf 中并重新启动系统。请注意，RemoveIPC 是一个系统范围的参数，更改它会影响整个系统。

在代理后面使用 Zabbix 前端

如果 Zabbix 前端在代理服务器后面运行，则需要重写代理配置文件中的 cookie 路径以匹配反向代理路径。请参阅下面的示例。如果不重写 cookie 路径，用户在尝试登录 Zabbix 前端时可能会遇到授权问题。

nginx 的示例配置

```
# ..
location / {
# ..
proxy_cookie_path /zabbix /;
proxy_pass http://192.168.0.94/zabbix/;
# ..
```

Apache 的示例配置

```
# ..
ProxyPass "/" http://host/zabbix/
ProxyPassReverse "/" http://host/zabbix/
ProxyPassReverseCookiePath /zabbix /
ProxyPassReverseCookieDomain host zabbix.example.com
# ..
```

18 转义示例

概述

本页提供了在各种上下文中使用正则表达式时正确使用转义的示例。

> **Note:**
> 使用触发器表达式构造函数时，会自动添加正则表达式中的正确转义。

示例

用户宏上下文

正则表达式: `\.+\"[a-z]+<br>` 用户宏上下文: `{$MACRO:regex:"\.+\\"[a-z]+"}`

注意:

- 反斜杠不需要转义；
- 引号需要转义。

LLD 宏函数

正则表达式: `\.+\"[a-z]+<br>` LLD 宏: `{{#MACRO}.iregsub("\.+\\"[a-z]+", \1)}`

注意:

- 反斜杠不进行转义；
- 引号进行转义。

LLD 宏函数在用户宏上下文

正则表达式: `\.+\"[a-z]+<br>` LLD 宏: `{{#MACRO}.iregsub("\.+\\"[a-z]+", \1)}<br>` 用户宏上下文: `{$MACRO:"{{#MACRO}.ire`
`\1)}"}`

注意:

- LLD 的反斜杠转义不发生变化；
- 在将 LLD 宏插入用户宏上下文时，我们需要将其放入字符串中：

1. 在宏表达式周围添加引号；
2. 引号需要转义；总共引入了 3 个新的反斜杠。

字符串参数的非历史函数

字符串内容: `\.+\"[a-z]+<br>` 表达式: `concat("abc", "\\.\\\"[a-z]+")`

注意:

- 字符串参数需要对反斜杠和引号进行转义。

字符串参数的历史函数

字符串内容: `\.+\"[a-z]+<br>` 表达式: `find(__ITEM_KEY__,,"regexp","\.+\\"[a-z]+")`

注意:

- 反斜杠不需要转义；
- 引号需要转义。

LLD 宏函数在非历史函数的字符串参数中

正则表达式: `\.+\"[a-z]+<br>` LLD 宏: `{{#MACRO}.iregsub("\.+\\"[a-z]+", \1)}<br>` 表达式: `concat("abc, "{{#MACRO}.iregsub(\"\\.+\\\\\"[a-z]+\", \\1)}")`

注意:

- 字符串参数需要对反斜杠和引号进行转义；
- 由于宏仅在字符串去引用后才解析，因此需要增加额外的转义层。

LLD 宏函数在历史函数的字符串参数中

正则表达式: `\.+\"[a-z]+<br>` LLD 宏: `{{#MACRO}.iregsub("\.+\\"[a-z]+", \1)}<br>` 表达式: `find(__ITEM_KEY__,,"eq","{{`
`\1)}")`

注意:

- 反斜杠不需要转义；
- 引号需要转义。

用户宏上下文在非历史函数的字符串参数中

正则表达式: `\.+\"[a-z]+<br>` 用户宏上下文: `{$MACRO:regex:"\.+\\"[a-z]+"}<br>` 表达式: `concat("abc, "{$MACRO:regex:\"\\.+\\\\\"[a-z]+\"}")`

注意:

- 与前一个示例相同，需要额外的转义层；
- 反斜杠和引号仅在最顶层进行转义（因为它是字符串参数）。

用户宏上下文在历史函数的字符串参数中

正则表达式: `\.+\"[a-z]+<br>` 用户宏上下文: `{$MACRO:regex:"\.+\\"[a-z]+"}<br>` 表达式: `find(__ITEM_KEY__,,"eq","{$MACR`

注意:

- 反斜杠不需要转义；

- 引号需要转义。

LLD 宏函数在用户宏上下文内部非历史函数中

正则表达式: `\.+\"[a-z]+<br>` LLD 宏: `{{#MACRO}.iregsub("\.+\\"[a-z]+", \1)}<br>` 用户宏上下文: `{$MACRO:"{{#MACRO}.iregsub("\.+\\"[a-z]+", \1)}"}<br>` 表达式: `concat("abc, "{$MACRO:\"{{#MACRO}.iregsub(\\\".+\\\\\\\"[a-z]+\\\", \\1)}\"}")`

请注意三层转义：

1. 对于 LLD 宏函数，不转义反斜杠；
2. 对于带上下文的用户宏，不转义反斜杠；
3. 对于函数的字符串参数，转义反斜杠。

LLD 宏函数在用户宏上下文内部历史函数中

正则表达式: `\.+\"[a-z]+<br>` LLD 宏: `{{#MACRO}.iregsub("\.+\\"[a-z]+", \1)}<br>` 用户宏上下文: `{$MACRO:"{{#MACRO}.iregsub("\.+\\"[a-z]+", \1)}"}<br>` 表达式: `find(__ITEM_KEY__,,"eq","{$MACRO:\"{{#MACRO}.iregsub(\\".+\\\\"[a-z]+\\", \1)}\"}")`

注意:

- 反斜杠不需要转义；
- 引号需要转义。

用户宏上下文仅在字符串内

正则表达式: `\.+\"[a-z]+<br>` 用户宏上下文: `{$MACRO:regex:"\.+\\"[a-z]+"}<br>` 在某些表达式的字符串内部，例如: `func(arg1, arg2, arg3)="{$MACRO:regex:\"\\.+\\\\\"[a-z]+\"}"`

注意:

- 字符串也需要转义反斜杠；
- 字符串也需要转义引号；
- 再次是一个需要两层转义的情况：

1. 用户宏上下文的转义，不转义反斜杠；
2. 作为字符串的转义，需要转义反斜杠。

Zabbix agent 与 **Zabbix agent 2** 的比较

本节描述了 Zabbix agent 和 Zabbix agent 2 之间的差异。

参数	Zabbix agent	Zabbix agent 2
编程语言	C	部分 C 的 Go 语言
守护进程化	是	仅通过 systemd（Windows 上是）
支持的扩展	自定义可加载模块，用 C 编写。	自定义插件，用 Go 编写。
需求		
支持的平台	Linux, IBM AIX, FreeBSD, NetBSD, OpenBSD, HP-UX, Mac OS X, Solaris: 9, 10, 11, Windows: 从 XP 起所有桌面和服务器版本	Linux, Windows: 所有桌面和服务器版本，只要可以安装最新的支持的 Go 版本
支持的加密库	GnuTLS 3.1.18 及更新版本 OpenSSL 1.0.1, 1.0.2, 1.1.0, 1.1.1, 3.0.x。注意 3.0.x 从 Zabbix 6.0.4 开始支持。 LibreSSL - 测试版本 2.7.4, 2.8.2（某些限制适用，详见加密页面）。	Linux: OpenSSL 1.0.1 及以后版本从 Zabbix 4.4.8 开始支持。 MS Windows: OpenSSL 1.1.1 或以后版本。 OpenSSL 库必须启用 PSK 支持。不支持 LibreSSL。
监控进程		
进程	每个 server/agent 记录一个单独的活动检查进程。	单个进程，自动创建线程。 最大线程数由 GOMAXPROCS 环境变量确定。
指标	**UNIX**: 见支持的监控项列表。 **Windows**: 见额外的 Windows 特定监控项列表。	**UNIX**: Zabbix agent 支持的所有指标。 此外，agent 2 为 Docker, Memcached, MySQL, PostgreSQL, Redis, systemd 和其他监控目标提供了 Zabbix 原生监控解决方案 - 见 agent 2 特定监控项的完整列表。 **Windows**: Zabbix agent 支持的所有指标，还有 net.tcp.service* 对 HTTPS, LDAP 的检查。 此外，agent 2 为 PostgreSQL, Redis 提供了 Zabbix 原生监控解决方案。

参数	Zabbix agent	Zabbix agent 2
并发性	单个服务器的活动检查按顺序执行。	来自不同插件的检查或同一插件中的多个检查可以并发执行。
计划/灵活间隔	仅支持被动检查。	支持被动和主动检查。
第三方 traps	不支持	支持
附加功能		
持久化存储	不支持	支持
日志文件持久化存储 *[log] 指标	是（仅限 Unix）	不支持
日志数据上传	可以在收集日志时执行，以释放缓冲区。	当缓冲区满时，收集日志将停止，因此BufferSize参数必须至少为 MaxLinesPerSecond x 2。
超时设置	仅在 agent 级别定义。	插件超时可以覆盖在 agent 级别定义的超时。
运行时更改用户	是（仅限类 Unix 系统）	不支持（由 systemd 控制）
用户可配置的密码套件	支持	不支持

另请参阅：

- *Zabbix* 进程描述：Zabbix agent, Zabbix agent 2
- 配置参数：Zabbix agentUNIX / Windows, Zabbix agent 2 UNIX / Windows

Zabbix 手册页

这些是 Zabbix 进程的手册页。

zabbix_agent2

部分：维护命令 (8)
更新时间: 2019-01-29
Index Return to Main Contents

名字

zabbix_agent2 - Zabbix agent 2

概要

zabbix_agent2 [**-c** *config-file*]
zabbix_agent2 [**-c** *config-file*] **-p**
zabbix_agent2 [**-c** *config-file*] **-t** *item-key*
zabbix_agent2 [**-c** *config-file*] **-R** *runtime-option*
zabbix_agent2 -h
zabbix_agent2 -V

描述

zabbix_agent2 是一个用于监视各种服务参数的应用程序。

选项

-c, **--config** *config-file*
使用指定的 配置文件而不是默认文件。

-R, **--runtime-control** *runtime-option*
根据 运行时选项执行管理功能。

运行时控制选项： **userparameter reload**
从配置文件重新加载用户参数

loglevel increase
提升日志级别

loglevel decrease
降低日志级别

help
列出可用的运行时控制选项

metrics
列出可用指标

version
显示版本

-p, **--print**
打印已知项目并退出。对于每个项目，要么使用通用默认值，要么提供用于测试的特定默认值。这些默认值作为项目关键参数列在方括号中。返回值括在方括号中，以返回值的类型为前缀，用竖线字符分隔。对于用户参数，类型始终为 **t**，因为 agent 无法确定所有可能的返回值。当查询正在运行的 agent 守护程序时，因为权限或环境可能不同，不保证在 Zabbix 服务器或 zabbix_get 中显示为工作中的项目。
返回值类型包括：

d
带有小数部分的数字。

m
不支持。这可能是由于查询仅在活动模式下工作的项目（如日志监视项目或需要多个收集值的项目）引起的。权限问题或不正确的用户参数也可能导致" 不支持" 状态。

s
文本。最大长度不受限制。

t
文本。和 **s** 一样。

u
无符号整数。

-t, **--test** *item-key*
测试单个项目并退出。查看 **--print** 部分的输出描述。

-h, **--help**
显示这个帮助的内容并退出。

-V, **--version**
输出版本信息并退出。

文件

/usr/local/etc/zabbix_agent2.conf
Zabbix agent 2 配置文件的默认位置（如果在编译时未修改的话）。

另请参见

文档 https://www.zabbix.com/manuals

zabbix_agentd(8), **zabbix_get**(8), **zabbix_js**(8), **zabbix_proxy**(8), **zabbix_sender**(8), **zabbix_server**(8)

索引

NAME

SYNOPSIS

DESCRIPTION

OPTIONS

FILES

SEE ALSO

AUTHOR

该文档是由man2html使用手册页创建的。

时间: 14:07:57 GMT, 11 22, 2021

zabbix_agentd

部分：维护命令 (8)
最后更新: 2019-01-29
Index Return to Main Contents

名字

zabbix_agentd - Zabbix agent 守护进程

概要

zabbix_agentd [**-c** *config-file*]
zabbix_agentd [**-c** *config-file*] **-p**
zabbix_agentd [**-c** *config-file*] **-t** *item-key*
zabbix_agentd [**-c** *config-file*] **-R** *runtime-option*
zabbix_agentd -h
zabbix_agentd -V

描述

zabbix_agentd 是用于监视各种服务器参数的守护程序。

选项

-c, **--config** *config-file*
使用指定的 *config-file* 而不是默认文件。

-f, **--foreground**
在前台运行 Zabbix agent。

-R, **--runtime-control** *runtime-option*
根据 *runtime-option* 执行管理功能。

运行时控制选项

userparameter_reload[=*target*]
从配置文件重新加载用户参数

log_level_increase[=*target*]
提升日志级别，如果未指定目标，则会影响所有进程

log_level_decrease[=*target*]
降低日志级别，如果未指定目标，将影响所有进程

日志级别控制目标

process-type
所有指定类型的进程（活动检查、收集器、侦听器）

process-type,N
进程类型和编号（例如，listener,3）

pid
进程标识符，最多 65535。对于较大的值，将目标指定为”process-type,N”

-p, **--print**
打印已知项目并退出。对于每个项目，要么使用通用默认值，要么提供用于测试的特定默认值。这些默认值作为项目关键参数列在方括号中。返回值括在方括号中，以返回值的类型为前缀，用竖线字符分隔。对于用户参数，类型始终为 **t**，因为 agent 无法确定所有可能的返回值。当查询正在运行的 agent 守护程序时，因为权限或环境可能不同，不保证在 Zabbix 服务器或 zabbix_get 中显示为工作中的项目。返回值类型包括：

d
带有小数部分的数字。

m
不支持。这可能是由于查询仅在活动模式下工作的项目（如日志监视项目或需要多个收集值的项目）引起的。权限问题或不正确的用户参数也可能导致” 不支持” 状态。

s
文本。最大长度不受限制。

t
文本。和 **s** 一样。

u
无符号整数。

-t, **--test** *item-key*
测试单个项目并退出。查看 **--print** 部分的输出描述。

-h, **--help**
显示这个帮助的内容并退出。

-V, **--version**
输出版本信息并退出。

文件

/usr/local/etc/zabbix_agentd.conf
Zabbix agent 配置文件的默认位置（如果在编译时未修改的话）。

另请参见

文档 https://www.zabbix.com/manuals

zabbix_agent2(8), **zabbix_get**(1), **zabbix_js**(1), **zabbix_proxy**(8), **zabbix_sender**(1), **zabbix_server**(8)

索引

NAME

SYNOPSIS

DESCRIPTION

OPTIONS

FILES

SEE ALSO

AUTHOR

该文档是由man2html, 使用手册页创建的。时间：20:50:13 GMT, 11 22, 2021

zabbix_get

部分：用户命令 (1)
更新：2021-06-01
Index Return to Main Contents

名字

zabbix_get - Zabbix get 实用程序

概要

zabbix_get -s *host-name-or-IP* [**-p** *port-number*] [**-I** *IP-address*] [**-t** *timeout*] **-k** *item-key*
zabbix_get -s *host-name-or-IP* [**-p** *port-number*] [**-I** *IP-address*] [**-t** *timeout*] **--tls-connect cert --tls-ca-file** *CA-file* [**--tls-crl-file** *CRL-file*] [**--tls-agent-cert-issuer** *cert-issuer*] [**--tls-agent-cert-subject** *cert-subject*] **--tls-cert-file** *cert-file* **--tls-key-file** *key-file* [**--tls-cipher13** *cipher-string*] [**--tls-cipher** *cipher-string*] **-k** *item-key*
zabbix_get -s *host-name-or-IP* [**-p** *port-number*] [**-I** *IP-address*] [**-t** *timeout*] **--tls-connect psk --tls-psk-identity** *PSK-identity* **--tls-psk-file** *PSK-file* [**--tls-cipher13** *cipher-string*] [**--tls-cipher** *cipher-string*] **-k** *item-key*
zabbix_get -h
zabbix_get -V

描述

zabbix_get 是一个命令行实用程序，用于从 zabbix agent 获取数据。

选项

-s, **--host** *host-name-or-IP*
指定主机的主机名或 IP 地址。

-p, **--port** *port-number*
指定主机上运行的 agent 的端口号。默认值为 10050。

-I, **--source-address** *IP-address*
指定源 IP 地址。

-t, **--timeout** *seconds*
指定超时。有效范围：1-30 秒（默认值：30）

-k, **--key** *item-key*
指定要为其检索值的项的 key。

--tls-connect *value*
如何连接到 agent。值

unencrypted
无加密连接（默认）

psk
使用 TLS 和预共享密钥进行连接

cert
使用 TLS 和证书进行连接

--tls-ca-file *CA-file*
包含用于对等证书验证的顶级 CA 证书的文件的完整路径名。

--tls-crl-file *CRL-file*
包含已吊销证书的文件的完整路径名。

--tls-agent-cert-issuer *cert-issuer*
允许的 agent 证书颁发者。

--tls-agent-cert-subject *cert-subject*
允许的 agent 证书主题。

--tls-cert-file *cert-file*
包含证书或证书链的文件的完整路径名。

--tls-key-file *key-file*
包含私钥的文件的完整路径名。

--tls-psk-identity *PSK-identity*
PSK 标识字符串。

--tls-psk-file *PSK-file*
包含预共享密钥的文件的完整路径名。

--tls-cipher13 *cipher-string*
OpenSSL 1.1.1 或更新版本 TLS 1.3 的密码字符串。覆盖默认的密码套件选择条件。如果 OpenSSL 版本低于 1.1.1，则此选项不可用。

--tls-cipher *cipher-string*
GnuTLS 优先级字符串（适用于 TLS 1.2 及以上版本）或 OpenSSL 密码字符串（仅适用于 TLS 1.2）。覆盖默认的密码套件选择条件。

-h, **--help**
显示本帮助信息然后退出。

-V, **--version**
输出版本信息然后退出

示例

zabbix_get -s 127.0.0.1 -p 10050 -k ”system.cpu.load[all,avg1]”
zabbix_get -s 127.0.0.1 -p 10050 -k ”system.cpu.load[all,avg1]” --tls-connect cert --tls-ca-file /home/zabbix/zabbix_ca_file --tls-agent-cert-issuer ”CN=Signing CA,OU=IT operations,O=Example Corp,DC=example,DC=com” --tls-agent-cert-subject ”CN=server1,OU=IT operations,O=Example Corp,DC=example,DC=com” --tls-cert-file /home/zabbix/zabbix_get.crt --tls-key-file /home/zabbix/zabbix_get.key
zabbix_get -s 127.0.0.1 -p 10050 -k ”system.cpu.load[all,avg1]” --tls-connect psk --tls-psk-identity ”PSK ID Zabbix agentd” --tls-psk-file /home/zabbix/zabbix_agentd.psk

另请参见

文档 https://www.zabbix.com/manuals

zabbix_agentd(8), **zabbix_proxy**(8), **zabbix_sender**(1), **zabbix_server**(8), **zabbix_js**(1), **zabbix_agent2**(8), **zabbix_web_service**(8)

索引

NAME
SYNOPSIS
DESCRIPTION
OPTIONS
EXAMPLES
SEE ALSO
AUTHOR

该文档是由man2html使用手册页创建的。时间：08:42:29 GMT, 6 月 11, 2021

zabbix_js

部分：用户命令 (1)
更新：2019-01-29
Index Return to Main Contents

名字

zabbix_js - Zabbix JS 实用程序

概要

zabbix_js -s *script-file* **-p** *input-param* [**-l** *log-level*] [**-t** *timeout*]
zabbix_js -s *script-file* **-i** *input-file* [**-l** *log-level*] [**-t** *timeout*]
zabbix_js -h
zabbix_js -V

描述

zabbix_js 是一个命令行实用程序，可用于嵌入式脚本测试。

OPTIONS

-s, **--script** *script-file*
指定要执行的脚本的文件名。如果将 “-” 指定为文件名，则将从标准输入读取脚本。

-p, **--param** *input-param*
指定输入参数。

-i, **--input** *input-file*
指定输入参数的文件名。如果将 “-” 指定为文件名，则将从准输入读取输入。

-l, **--loglevel** *log-level*
指定日志级别。

-t, **--timeout** *timeout*
以秒为单位指定超时。有效范围：1-60 秒（默认值：10）

-h, **--help**
显示此帮助并退出。

-V, **--version**
输出版本信息并退出。

示例

zabbix_js -s script-file.js -p example

另请参见

文档 https://www.zabbix.com/manuals

zabbix_agent2(8), **zabbix_agentd**(8), **zabbix_get**(1), **zabbix_proxy**(8), **zabbix_sender**(1), **zabbix_server**(8)

索引

NAME

SYNOPSIS

DESCRIPTION

OPTIONS

EXAMPLES

SEE ALSO

该文档是由man2html使用手册页创建的。时间：21:23:35 GMT, 三月 18, 2020

zabbix_proxy

部分：维护命令 (8)
更新: 2020-09-04
Index Return to Main Contents

名字

zabbix_proxy - Zabbix proxy 守护进程

概要

zabbix_proxy [**-c** *config-file*]
zabbix_proxy [**-c** *config-file*] **-R** *runtime-option*
zabbix_proxy -h
zabbix_proxy -V

描述

zabbix_proxy 是一个从设备收集监控数据并将其发送到 Zabbix server 的守护程序。

OPTIONS

-c, **--config** *config-file*
使用指定的 *config-file* 而不是默认文件。

-f, **--foreground**
在前台运行 Zabbix proxy。

-R, **--runtime-control** *runtime-option*
根据 *runtime-option* 执行管理功能。

运行时控制选项

config_cache_reload
重新加载配置缓存。如果当前正在加载缓存，则忽略。活动的 Zabbix proxy 将连接到 Zabbix server 并请求配置数据。默认配置文件（除非指定了**-c** 选项）将用于查找 PID 文件，并将信号发送到进程，在 PID 文件中列出。

snmp_cache_reload
重新加载 SNMP 缓存。

housekeeper_execute
执行 housekeeper。如果当前 housekeeper 正在被执行则忽略。

diaginfo[=*section*]
记录指定部分的内部诊断信息。节可以是 *historycache*，*preprocessing*。默认情况下，记录所有部分的诊断信息。

log_level_increase[=*target*]
提升日志级别，如果未指定目标，则会影响所有进程。

log_level_decrease[=*target*]
降低日志级别，如果未指定目标，则会影响所有进程。

日志级别控制目标

process-type
指定类型的所有进程 (configuration syncer, data sender, discoverer, heartbeat sender, history syncer, housekeeper, http poller, icmp pinger, ipmi manager, ipmi poller, java poller, poller, self-monitoring, snmp trapper, task manager, trapper, unreachable poller, vmware collector)

process-type,N
进程类型和编号（例如，poller,3）

pid
进程标识符，最多 65535。对于较大的值，将目标指定为”process-type,N”

-h, **--help**
显示这个帮助的内容并退出。

-V, **--version**
输出版本信息并退出。

文件

/usr/local/etc/zabbix_proxy.conf
Zabbix proxy 配置文件的默认位置（如果在编译时未修改的话）。

另请参见

文档 https://www.zabbix.com/manuals

zabbix_agentd(8), **zabbix_get**(1), **zabbix_sender**(1), **zabbix_server**(8), **zabbix_js**(1), **zabbix_agent2**(8)

索引

NAME

SYNOPSIS

DESCRIPTION

OPTIONS

FILES

SEE ALSO

AUTHOR

此文档创建于：2020 年 9 月 4 日格林威治标准时间 16:12:22

zabbix_sender

部分：用户命令 User Commands (1)
最近更新: 2021-06-01
Index Return to Main Contents

名字

zabbix_sender - Zabbix sender 实用程序

概要

zabbix_sender [**-v**] **-z** *server* [**-p** *port*] [**-I** *IP-address*] [**-t** *timeout*] **-s** *host* **-k** *key* **-o** *value*
zabbix_sender [**-v**] **-z** *server* [**-p** *port*] [**-I** *IP-address*] [**-t** *timeout*] [**-s** *host*] [**-T**] [**-N**] [**-r**] **-i** *input-file*
zabbix_sender [**-v**] **-c** *config-file* [**-z** *server*] [**-p** *port*] [**-I** *IP-address*] [**-t** *timeout*] [**-s** *host*] **-k** *key* **-o** *value*
zabbix_sender [**-v**] **-c** *config-file* [**-z** *server*] [**-p** *port*] [**-I** *IP-address*] [**-t** *timeout*] [**-s** *host*] [**-T**] [**-N**] [**-r**] **-i** *input-file*
zabbix_sender [**-v**] **-z** *server* [**-p** *port*] [**-I** *IP-address*] [**-t** *timeout*] **-s** *host* **--tls-connect cert** **--tls-ca-file** *CA-file* [**--tls-crl-file** *CRL-file*] [**--tls-server-cert-issuer** *cert-issuer*] [**--tls-server-cert-subject** *cert-subject*] **--tls-cert-file** *cert-file* **--tls-key-file** *key-file* [**--tls-cipher13** *cipher-string*] [**--tls-cipher** *cipher-string*] **-k** *key* **-o** *value*
zabbix_sender [**-v**] **-z** *server* [**-p** *port*] [**-I** *IP-address*] [**-t** *timeout*] [**-s** *host*] **--tls-connect cert** **--tls-ca-file** *CA-file* [**--tls-crl-file** *CRL-file*] [**--tls-server-cert-issuer** *cert-issuer*] [**--tls-server-cert-subject** *cert-subject*] **--tls-cert-file** *cert-file* **--tls-key-file** *key-file* [**--tls-cipher13** *cipher-string*] [**--tls-cipher** *cipher-string*] [**-T**] [**-N**] [**-r**] **-i** *input-file*
zabbix_sender [**-v**] **-c** *config-file* [**-z** *server*] [**-p** *port*] [**-I** *IP-address*] [**-t** *timeout*] [**-s** *host*] **--tls-connect cert** **--tls-ca-file** *CA-file* [**--tls-crl-file** *CRL-file*] [**--tls-server-cert-issuer** *cert-issuer*] [**--tls-server-cert-subject** *cert-subject*] **--tls-cert-file** *cert-file* **--tls-key-file** *key-file* [**--tls-cipher13** *cipher-string*] [**--tls-cipher** *cipher-string*] **-k** *key* **-o** *value*
zabbix_sender [**-v**] **-c** *config-file* [**-z** *server*] [**-p** *port*] [**-I** *IP-address*] [**-t** *timeout*] [**-s** *host*] **--tls-connect cert** **--tls-ca-file** *CA-file* [**--tls-crl-file** *CRL-file*] [**--tls-server-cert-issuer** *cert-issuer*] [**--tls-server-cert-subject** *cert-subject*] **--tls-cert-file** *cert-file* **--tls-key-file** *key-file* [**--tls-cipher13** *cipher-string*] [**--tls-cipher** *cipher-string*] [**-T**] [**-N**] [**-r**] **-i** *input-file*

zabbix_sender [**-v**] **-z** *server* [**-p** *port*] [**-I** *IP-address*] [**-t** *timeout*] **-s** *host* **--tls-connect psk --tls-psk-identity** *PSK-identity* **--tls-psk-file** *PSK-file* [**--tls-cipher13** *cipher-string*] [**--tls-cipher** *cipher-string*] **-k** *key* **-o** *value*
zabbix_sender [**-v**] **-z** *server* [**-p** *port*] [**-I** *IP-address*] [**-t** *timeout*] [**-s** *host*] **--tls-connect psk --tls-psk-identity** *PSK-identity* **--tls-psk-file** *PSK-file* [**--tls-cipher13** *cipher-string*] [**--tls-cipher** *cipher-string*] [**-T**] [**-N**] [**-r**] **-i** *input-file*
zabbix_sender [**-v**] **-c** *config-file* [**-z** *server*] [**-p** *port*] [**-I** *IP-address*] [**-t** *timeout*] [**-s** *host*] **--tls-connect psk --tls-psk-identity** *PSK-identity* **--tls-psk-file** *PSK-file* [**--tls-cipher13** *cipher-string*] [**--tls-cipher** *cipher-string*] **-k** *key* **-o** *value*
zabbix_sender [**-v**] **-c** *config-file* [**-z** *server*] [**-p** *port*] [**-I** *IP-address*] [**-t** *timeout*] [**-s** *host*] **--tls-connect psk --tls-psk-identity** *PSK-identity* **--tls-psk-file** *PSK-file* [**--tls-cipher13** *cipher-string*] [**--tls-cipher** *cipher-string*] [**-T**] [**-N**] [**-r**] **-i** *input-file*
zabbix_sender -h
zabbix_sender -V

描述

zabbix_sender 是一个命令行实用程序，用于向 Zabbix server 或者 proxy 发送监控数据。在 Zabbix server 上，应使用相应的密钥创建 **Zabbix trapper** 类型的项。请注意，传入值将仅接受从该项目的 **Allowed hosts** 字段中指定的主机。

选项

-c, **--config** *config-file*
使用 *config-file* 选项。**Zabbix sender** 从 agentd 配置文件读取 server 详细信息。默认情况下 **Zabbix sender** 不读取任何配置文件。仅支持参数 **Hostname**、**ServerActive**、**SourceIP**、**TLSConnect**、**TlsCfile**、**TLSServerCertIssuer**、**TLSServerCertSubject**、**TLSCertFile**、**TLSKeyFile**、**TLSKeyFile**、**TlspKidEntity** 和 **TlspSpkFile**。Agent **ServerActive** 配置参数中定义的所有地址都用于发送数据。如果批次数据发送到某一个地址失败，则后续批次不会在发送到该地址。

-z, **--zabbix-server** *server*
Zabbix server 的主机名或 IP 地址。如果主机由 proxy 监控，则应使用 proxy 的主机名或 IP 地址。与 **--config** 一起使用时，会覆盖 agentd 配置文件中指定的 **ServerActive** 参数项。

-p, **--port** *port*
指定 server 上运行的 Zabbix server trapper 的端口号。默认值为 10051。与 **--config** 一起使用时，会覆盖 agentd 配置文件中指定的 **ServerActive** 参数的端口项。

-I, **--source-address** *IP-address*
指定源 IP 地址。与 **--config** 一起使用时，会覆盖 agentd 配置文件中指定的 **SourceIP** 参数。

-t, **--timeout** *seconds*
指定超时时间。有效范围：1-300 秒（默认值：60）

-s, **--host** *host*
指定项目所属的主机名（在 Zabbix 前端中注册）。主机 IP 地址和 DNS 名称将不起作用。与 **--config** 一起使用时，会覆盖 agentd 配置文件中指定的 **Hostname** 参数。

-k, **--key** *key*
指定要向其发送值的监控项键。

-o, **--value** *value*
指定监控项值。

-i, **--input-file** *input-file*
从输入文件加载值。将 **-** 指定为 **<input file>** 则从标准输入读取值。文件的每一行都包含以空格分隔的：**<hostname> <key> <value>**。每个值必须在自己的行中指定。每行必须包含 3 个空格分隔的条目：**<hostname> <key> <value>** ，其中 “hostname” 是在 Zabbix 前端中注册的受监控主机的名称，“key” 是目标监控项键，“value” 是要发送的值。将 **-** 指定为 **<hostname>** 以使用 agent 配置文件或 **--host** 参数中的主机名。

输入文件的一行示例：

”Linux DB3” db.connections 43

必须在 Zabbix 前端的监控项配置中正确设置值类型。Zabbix sender 将在一个连接中发送多达 250 个值。用于从输入文件发送值的大小限制取决于 Zabbix 通信协议Size limit 中对大小的定义。输入文件的内容必须采用 UTF-8 编码。输入文件中的所有值都以自上而下的顺序发送。必须使用以下规则格式化条目：

- 支持带引号和不带引号的条目。
- 双引号是引用字符。
- 必须引用带有空格的条目。

- 引号内的双引号和反斜杠字符必须用反斜杠转义。
- 在不带引号的条目中不支持转义。
- 带引号的字符串支持换行符序列（\n）。
- 换行转义序列从条目的末尾修剪。

-T, **--with-timestamps**
此选项只能与 **--input-file** 选项一起使用。

输入文件的每一行必须包含 4 个以空格分隔的条目：**<hostname> <key> <timestamp> <value>**。时间戳应以 Unix 时间戳格式指定。如果目标监控项有引用它的触发器，则所有时间戳必须按递增顺序排列，否则事件计算将不正确。

输入文件的一行示例：

”Linux DB3” db.connections 1429533600 43

有关更多详细信息，请参阅选项 **--input-file**。

如果为 “不采集数据” 维护类型的主机发送时间戳值，则该值将被删除；但是，可以在已过期的维护期内发送时间戳值，其将被接受。

-N, **--with-ns**
此选项只能与 **--with-timestamps** 选项一起使用。

输入文件的每一行必须包含 5 个以空格分隔的条目：**<hostname> <key> <timestamp> <ns> <value>**。

输入文件的一行示例：

”Linux DB3” db.connections 1429533600 7402561 43

有关更多详细信息，请参阅选项 **--input-file**。

-r, **--real-time**
收到值后立即逐个发送。当从标准输入读取数据时，可以使用此选项。

--tls-connect *value*
如何连接到服务器或代理。值：

unencrypted
无加密连接（默认）

psk
使用 TLS 和预共享密钥进行连接

cert
使用 TLS 和证书进行连接

--tls-ca-file *CA-file*
包含用于对等证书验证的顶级 CA 证书的文件的完整路径名。

--tls-crl-file *CRL-file*
包含已吊销证书的文件的完整路径名。

--tls-server-cert-issuer *cert-issuer*
允许的服务器证书颁发者。

--tls-server-cert-subject *cert-subject*
允许的服务器证书主题。

--tls-cert-file *cert-file*
包含证书或证书链的文件的完整路径名。

--tls-key-file *key-file*
包含私钥的文件的完整路径名。

--tls-psk-identity *PSK-identity*
PSK 标识字符串。

--tls-psk-file *PSK-file*
包含预共享密钥的文件的完整路径名。

--tls-cipher13 *cipher-string*
OpenSSL 1.1.1 或更新版本 TLS 1.3 的密码字符串。覆盖默认的密码套件选择条件。如果 OpenSSL 版本低于 1.1.1，则此选项不可用。

--tls-cipher *cipher-string*
GnuTLS 优先级字符串（适用于 TLS 1.2 及以上版本）或 OpenSSL 密码字符串（仅适用于 TLS 1.2）。覆盖默认的密码套件选择条件。

-v, **--verbose**
详细模式，**-vv** 了解更多详细信息。

-h, **--help**
显示这个帮助的内容并退出。

-V, **--version**
输出版本信息并退出。

退出状态

如果值已发送且服务器已成功处理所有值，则退出状态为 0。如果发送了数据，但至少一个值的处理失败，则退出状态为 2。如果数据发送失败，退出状态为 1。

示例

zabbix_sender -c /etc/zabbix/zabbix_agentd.conf -k mysql.queries -o 342.45

发送 **342.45** 作为受监视主机的 **mysql.queries** 项的值。使用代理配置文件中定义的受监视主机和 Zabbix 服务器。

zabbix_sender -c /etc/zabbix/zabbix_agentd.conf -s ”Monitored Host” -k mysql.queries -o 342.45

使用代理配置文件中定义的 Zabbix 服务器发送 **342.45** 作为在前端注册的 受监视主机主机的 **mysql.queries** 项的值。

zabbix_sender -z 192.168.1.113 -i data_values.txt

将值从文件 **data_values.txt** 发送到 IP 为 **192.168.1.113** 的 Zabbix 服务器。主机名和密钥在文件中定义。

echo ”- hw.serial.number 1287872261 SQ4321ASDF” | zabbix_sender -c /usr/local/etc/zabbix_agentd.conf -T -i -

将带时间戳的值从命令行发送到 agent 配置文件中指定的 Zabbix 服务器。输入数据中的短划线表示主机名也应从同一使用的配置文件中读取。

echo ’”Zabbix server” trapper.item ””’ | zabbix_sender -z 192.168.1.113 -p 10000 -i -

从命令行将项目的空值发送到端口 **10000** 上的 IP 地址为 **192.168.1.113** 的 Zabbix 服务器。空值必须由空双引号表示。

zabbix_sender -z 192.168.1.113 -s ”Monitored Host” -k mysql.queries -o 342.45 --tls-connect cert --tls-ca-file /home/zabbix/zabbix_ca_file --tls-cert-file /home/zabbix/zabbix_agentd.crt --tls-key-file /home/zabbix/zabbix_agentd.key

使用带有证书的 TLS 将 **342.45** 作为 **mysql.queries** 项的值发送到 IP 为 192.168.1.113** 的 IP 主机的服务器。

zabbix_sender -z 192.168.1.113 -s ”Monitored Host” -k mysql.queries -o 342.45 --tls-connect psk --tls-psk-identity ”PSK ID Zabbix agentd” --tls-psk-file /home/zabbix/zabbix_agentd.psk

使用带有预共享密钥（PSK）的 TLS 将 **342.45** 作为 **mysql.queries** 项的值发送到 IP 为 IP 为 **192.168.1.113** 的服务器。

另请参见

文档 https://www.zabbix.com/manuals

zabbix_agentd(8), **zabbix_get**(1), **zabbix_proxy**(8), **zabbix_server**(8), **zabbix_js**(1), **zabbix_agent2**(8), **zabbix_web_service**(8)

索引

NAME

SYNOPSIS

DESCRIPTION

OPTIONS

EXIT STATUS

EXAMPLES

SEE ALSO

AUTHOR

此文档创建于：2021 年 6 月 11 日格林威治标准时间 08:42:39

zabbix_server

部分：维护命令 (8)
更新：2020-09-04
Index Return to Main Contents

名字

zabbix_server - Zabbix server 守护进程

概要

zabbix_server [**-c** *config-file*]
zabbix_server [**-c** *config-file*] **-R** *runtime-option*
zabbix_server -h
zabbix_server -V

描述

zabbix_server 是 Zabbix 软件的核心守护进程。

选项

-c, **--config** *config-file* 使用指定的 配置文件而不是默认文件。

-f, **--foreground** 在前端运行 Zabbix server。

-R, **--runtime-control** *runtime-option* 根据 运行时选项执行管理功能。

-h, **--help**
显示帮助信息。

-V, **--version**
显示版本信息。

运行 Zabbix 服务器时，可以通过命令行参数来执行特定的操作或配置：

```
zabbix_server -c /usr/local/etc/zabbix_server.conf
zabbix_server --help
zabbix_server -V
```

运行时控制

运行时控制选项：

config_cache_reload
重新加载配置缓存。如果缓存当前正在加载，则会被忽略。将使用默认配置文件（除非指定了 **-c** 选项）来查找 PID 文件，并将信号发送到 PID 文件中列出的进程。

snmp_cache_reload
重新加载 SNMP 缓存，清除所有主机的 SNMP 属性（引擎时间、引擎启动次数、引擎 ID、凭据）。

housekeeper_execute
执行管家清理程序。如果管家清理程序当前正在执行，则会被忽略。

trigger_housekeeper_execute
启动触发器的管家清理程序。如果触发器的管家清理程序当前正在进行，则会被忽略。

diaginfo[=*section*]
记录指定部分的内部诊断信息。部分可以是 *historycache*（历史缓存）、*preprocessing*（预处理）、*alerting*（告警）、*lld*（低级发现）、*valuecache*（缓存值）。默认情况下，记录所有部分的诊断信息。

ha_status
记录高可用性（HA）集群状态。

ha_remove_node[=*target*]
根据其名称或 ID 移除指定的高可用性（HA）节点。注意，不能移除活动/备用节点。

ha_set_failover_delay[=*delay*]
设置高可用性（HA）故障转移延迟。支持时间后缀，例如 10s, 1m。

secrets_reload
从 Vault 重新加载密钥。

service_cache_reload
重新加载服务管理器缓存。

prof_enable[=*target*]
启用性能分析。如果未指定目标，则影响所有进程。启用的性能分析提供了所有 rwlocks/mutexes 按函数名称的详细信息。自 Zabbix 6.0.13 起支持。

prof_disable[=*target*]
禁用性能分析。如果未指定目标，则影响所有进程。自 Zabbix 6.0.13 起支持。

log_level_increase[=*target*]
提高日志级别，如果未指定目标，则影响所有进程。

log_level_decrease[=*target*]
降低日志级别，如果未指定目标，则影响所有进程。

日志级别控制目标

process-type
指定类型的所有进程（alerter、alert manager、configuration syncer、discoverer、escalator、history syncer、housekeeper、http poller、icmp pinger、ipmi manager、ipmi poller、java poller、lld manager、lld worker、poller、preprocessing manager、preprocessing worker、proxy poller、self-monitoring、snmp trapper、task manager、timer、trapper、unreachable poller、vmware collector）

process-type,N
进程类型和编号（例如，poller,3）

pid
进程标识符，最大为 65535。对于更大的值，请将目标指定为 “process-type,N”

文件

/usr/local/etc/zabbix_server.conf
Zabbix proxy 配置文件的默认位置（如果在编译时未修改的话）。

另请参见

文档 https://www.zabbix.com/manuals

zabbix_agentd(8), **zabbix_get**(1), **zabbix_proxy**(8), **zabbix_sender**(1), **zabbix_js**(1), **zabbix_agent2**(8)

索引

NAME

SYNOPSIS

DESCRIPTION

OPTIONS

FILES

SEE ALSO

AUTHOR

本文档创建于：2020 年 9 月 4 日，格林威治时间 16:12:14

zabbix_web_service

部分：维护命令（8）更新：2019-01-29 索引 返回主要内容

名称

zabbix_web_service - Zabbix web 服务

概要

zabbix_web_service [**-c** *config-file*]
zabbix_web_service -h
zabbix_web_service -V

描述

zabbix_web_service 是一个为 Zabbix 组件提供 web 服务的应用程序。

选项

-c, **--config** *config-file*
使用指定的 *config-file* 而不是默认文件。

-h, **--help**
显示这个帮助的内容并退出。

-V, **--version**
输出版本信息并退出。

文件

/usr/local/etc/zabbix_web_service.conf
Zabbix proxy 配置文件的默认位置（如果在编译时未修改的话）。

另请参见

文档 https://www.zabbix.com/manuals

zabbix_agentd(8), **zabbix_get**(1), **zabbix_proxy**(8), **zabbix_sender**(1), **zabbix_server**(8), **zabbix_js**(1), **zabbix_agent2**(8)

索引

名称

概述

描述

选项

文件

另请参阅

作者

本文档创建于：2021 年 6 月 11 日，格林尼治标准时间 12:58:30