4 发现JMX对象

概述

这能够发现 全部JMX MBeans或MBean属性，或指定用于发现这些对象的模式。

必须了解发现规则配置的Mbean和Mbean属性之间的区别。 MBean是一个对象，可以表示设备，应用程序或需要管理的任何资源。例如一个代表web-server的Mbean。 它的属性是连接数，线程数，请求超时。http文件缓存，内存使用等。 用普通人的语言理解这个想法的话，我们可以将咖啡机定义为Mbean。它具有以下被监控的点：每杯水量，一段时间内的平均水消耗量，每杯所需的咖啡豆数量，咖啡豆和补水时间等。

键值

在发现规则 配置中，在*类型*区域选择 **JMX agent**。

该键值为：

```
jmx.discovery[<discovery mode>,<object name>]
```

- 发现模式 - 其中之一：*属性*（检索JMX MBean属性，默认值）或*beans*（检索JMX MBeans。
- 对象名称 - 辨别要检索的MBean名称的对象名称模式（默认为空，检索所有已注册的beans。。

你可以参考 使用手册里的 ObjectName以获取指定对象名称模式的选项。

如果未传递任何参数，则意味着请求JMX中的所有MBean属性。

不指定JMX发现的参数或尝试接收范围广泛的所有属性 `*:type=*,name=*` 可能会导致潜在的性能问题。

此监控项从Zabbix Java gateway 3.4开始支持。

键值举例：

```
jmx.discovery  #Retrieve all JMX MBean attributes
jmx.discovery[beans]  #Retrieve all JMX MBeans
jmx.discovery[attributes,"*:type=GarbageCollector,name=*"]  #Retrieve all
garbage collector attributes
jmx.discovery[beans,"*:type=GarbageCollector,name=*"]  #Retrieve all garbage
collectors
```

此监控项反馈一个JSON对象。例如，在发现MBean属性（为清楚起见重新格式化）：

```
{
   "data":[
      {
         "{#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"
        }
    ]
}
```

在发现MBean属性（为清楚起见重新格式化）：

```
{
    "data":[
        {
            "{#JMXDOMAIN}":"java.lang",
            "{#JMXTYPE}":"GarbageCollector",
            "{#JMXOBJ}":"java.lang:type=GarbageCollector,name=PS Scavenge",
            "{#JMXNAME}":"PS Scavenge"
        }
    ]
```

```
}
```

支持宏

以下宏支持在发现规则中的过滤器，监控项，触发器和图表的原型中的应用：

宏	描述
发现MBean属性	
{#JMXVALUE}	Attribute value.
{#JMXTYPE}	Attribute type.
{#JMXOBJ}	Object name.
{#JMXDESC}	Object name including attribute name.
{#JMXATTR}	Attribute name.
发现MBeans	
{#JMXDOMAIN}	MBean domain. (*Zabbix reserved name*)
{#JMXOBJ}	Object name. (*Zabbix reserved name*)
{#JMX<key property>}	MBean properties (like {#JMXTYPE}, {#JMXNAME}). Some important notes to pay attention to when defining MBean attribute name that is created from MBean property name by the following algorithm: * attribute name case is changed to uppercase; * attribute name case is ignored (no LLD macro is generated) in case it consists of not supported characters. Supported characters can be described by the following regular expression: “A-Z0-9_\.”; * if an attribute name is called “*obj*” or “*domain*” it will be replaced with the values of Zabbix properties {#JMXOBJ} and {#JMXDOMAIN} (supported since Zabbix 3.4.3.).

请考虑 jmx.discovery (以“beans” 模式）的例子. MBean定义了以下属性：

```
name=test
тип=Type
attributes []=1,2,3
Name=NameOfTheTest
domAin=some
```

作为JMX发现的结果，将生成以下LLD宏：

- {#JMXDOMAIN} - Zabbix内部，描述了MBean领域
- {#JMXOBJ} - Zabbix内部，描述了MBean对象
- {#JMXNAME} - 从“name” 属性创建

忽略的属性是：

- тип : 它的名字内包含无法识别的字母 (non-ASCII)
- attributes[] : 它的名字内包含无法识别的字母（不支持方括号）
- Name : 它已经被定义了 (name=test)
- domAin : 这是Zabbix的保留名称

让我们回顾两个使用Mbean创建LLD规则的实际示例。 要了解收集Mbeans的LLD规则与收集Mbean属性

的LLD规则之间的区别，请查看下表：

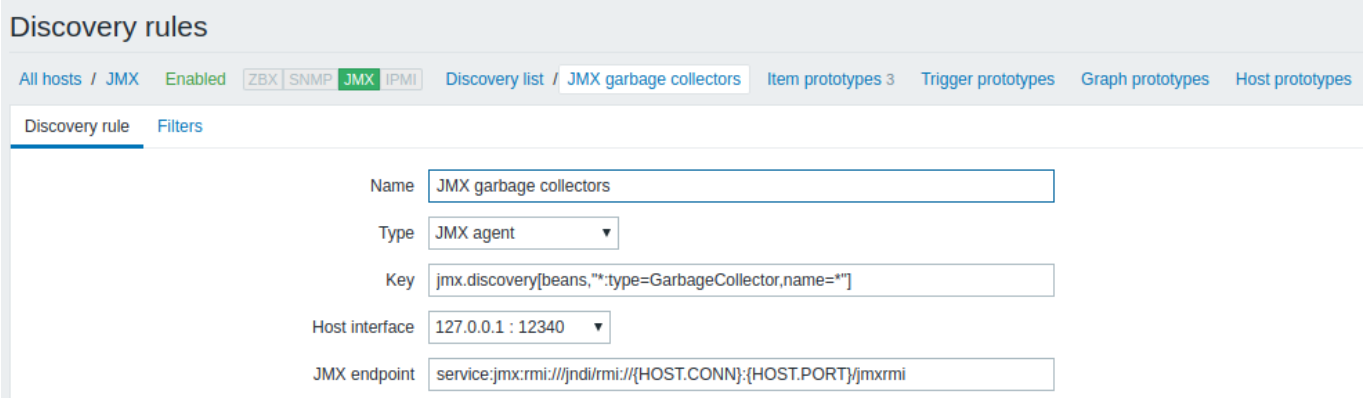
MBean1	MBean2	MBean3
MBean1Attribute1	MBean2Attribute1	MBean3Attribute1
MBean1Attribute2	MBean2Attribute2	MBean3Attribute2
MBean1Attribute3	MBean2Attribute3	MBean3Attribute3

以LLD规则收集Mbeans

规则将会反馈三个对象：该列的顶行 MBean1, MBean2, MBean3.

有关对象的更多信息，请参阅 支持宏 表格， *发现MBean* 部分。

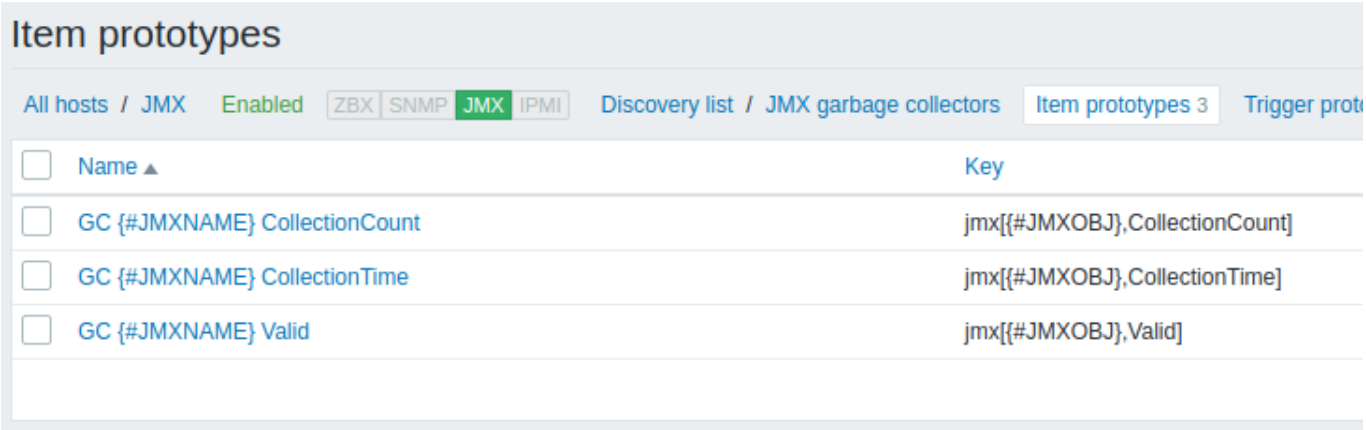
收集Mbeans无属性）的发现规则配置如下所示：



使用键值：

```
jmx.discovery[beans,"*:type=GarbageCollector,name=*"]
```

能发现所有没有属性的垃圾收集器。 由于垃圾收集器具有相同的属性集，我们可以通过以下方式在项原型中使用所需的属性：

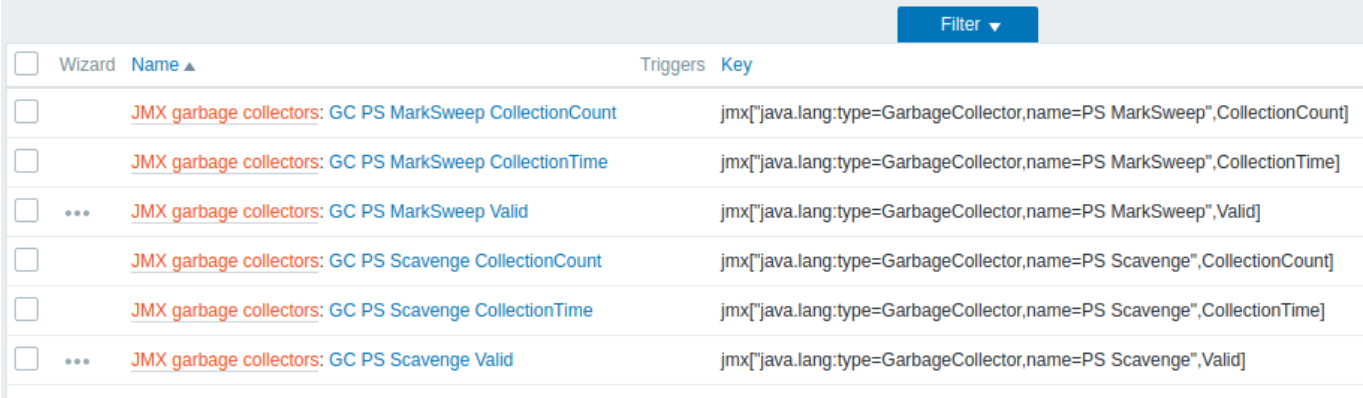


使用键值：

```
jmx[{#JMXOBJ},CollectionCount]
jmx[{#JMXOBJ},CollectionTime]
```

```
jmx[{#JMXOBJ},Valid]
```

LLD发现规则将导致与此接近的内容（为两个垃圾收集器发现的监控项）：



Filter

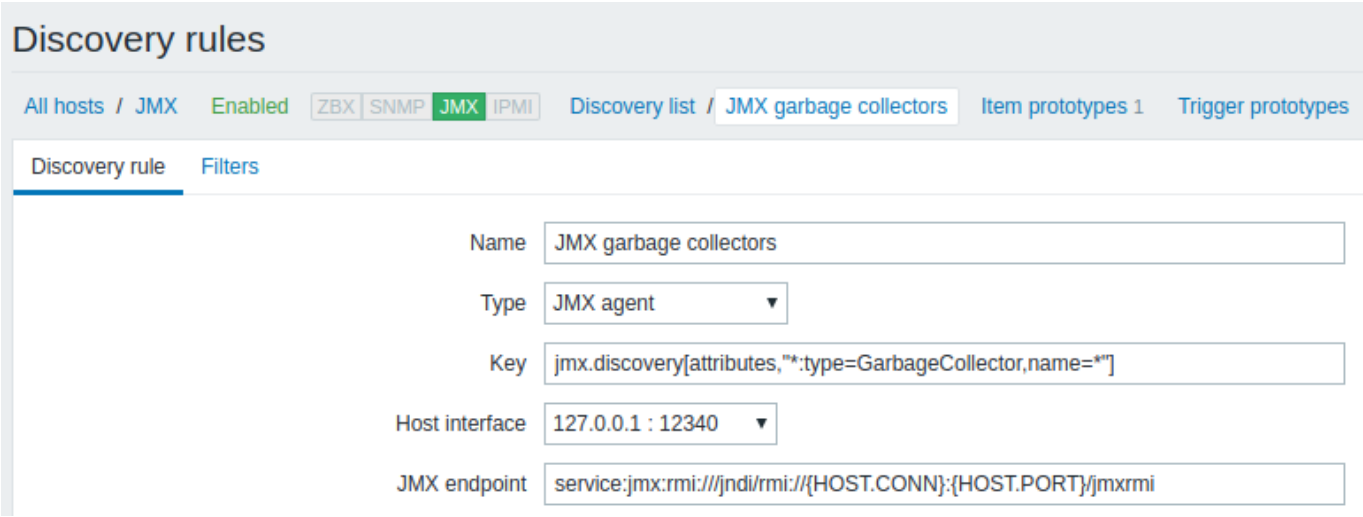
Wizard	Name ▲	Triggers	Key
	JMX garbage collectors: GC PS MarkSweep CollectionCount		jmx["java.lang:type=GarbageCollector,name=PS MarkSweep",CollectionCount]
	JMX garbage collectors: GC PS MarkSweep CollectionTime		jmx["java.lang:type=GarbageCollector,name=PS MarkSweep",CollectionTime]
•••	JMX garbage collectors: GC PS MarkSweep Valid		jmx["java.lang:type=GarbageCollector,name=PS MarkSweep",Valid]
	JMX garbage collectors: GC PS Scavenge CollectionCount		jmx["java.lang:type=GarbageCollector,name=PS Scavenge",CollectionCount]
	JMX garbage collectors: GC PS Scavenge CollectionTime		jmx["java.lang:type=GarbageCollector,name=PS Scavenge",CollectionTime]
•••	JMX garbage collectors: GC PS Scavenge Valid		jmx["java.lang:type=GarbageCollector,name=PS Scavenge",Valid]

LLD规则收集MBean属性

这条规则将会反馈9个对象MBean1Attribute1, MBean2Attribute1, Mbean3Attribute1,MBean1Attribute2,MBean2Attribute2, Mbean3Attribute2, MBean1Attribute3, MBean2Attribute3, Mbean3Attribute3.

更多有关于对象的信息，请参考 支持宏 表格， *发现MBean属性* 部分.

收集MBean属性的发现规则配置如以下所示：



Discovery rules

All hosts / JMX Enabled ZBX SNMP JMX IPMI Discovery list / JMX garbage collectors Item prototypes 1 Trigger prototypes

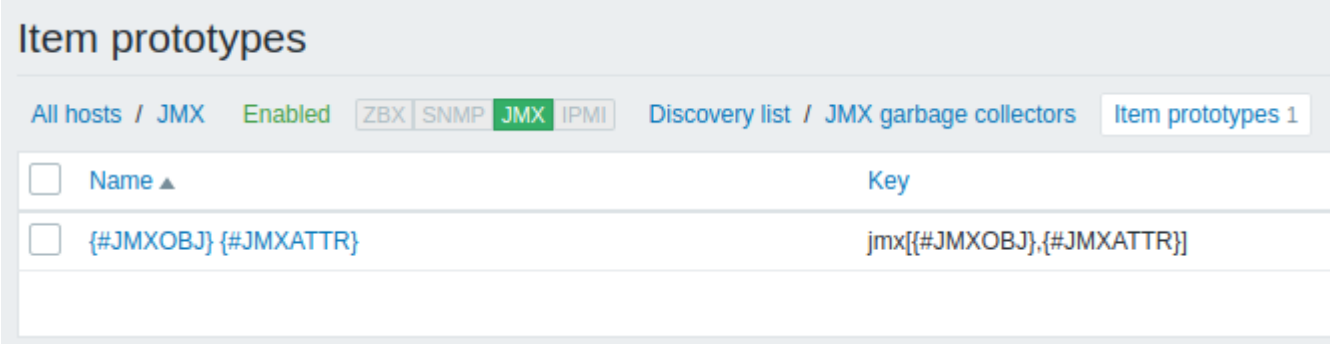
Discovery rule Filters

Name	JMX garbage collectors
Type	JMX agent
Key	jmx.discovery[attributes,"*:type=GarbageCollector,name=*"]
Host interface	127.0.0.1 : 12340
JMX endpoint	service:jmx:rmi:///jndi/rmi://{HOST.CONN}:{HOST.PORT}/jmxrmi

使用键值:

```
jmx.discovery[attributes,"*:type=GarbageCollector,name=*"]
```

将发现具有单个项属性的所有垃圾收集器。



在这种特殊情况下，将从原型为每个MBean属性创建一个监控项。这种配置的主要缺点是从触发器原型的触发器创建是不可能的，因为所有属性只有一个监控项原型。因此，此设置可用于数据收集，但不建议用于自动监控。

4 Discovery of JMX objects

Overview

It is possible to discover all JMX MBeans or MBean attributes or to specify a pattern for the discovery of these objects.

It is mandatory to understand the difference between Mbean and Mbean attributes for discovery rule configuration. An MBean is an object which can represent a device, an application, or any resource that needs to be managed. For example, there is an Mbean which represents a web-server. Its attributes are connection count, thread count, request timeout, http file cache, memory usage, etc. Expressing this thought in human comprehensive language we can define a coffee machine as an Mbean which has the following attributes to be monitored: water amount per cup, average consumption of water for a certain period of time, number of coffee beans required per cup, coffee beans and water refill time, etc.

Item key

In discovery rule configuration, select **JMX agent** in the *Type* field.

The item key to use is

```
jmx.discovery[<discovery mode>,<object name>]
```

where

- discovery mode - one of the following: *attributes* (retrieve JMX MBean attributes, default) or *beans* (retrieve JMX MBeans)
- object name - object name pattern identifying the MBean names to be retrieved (empty by default, retrieving all registered beans).

You may consult ObjectName documentation for the options of specifying object name pattern.

If no parameters are passed, all MBean attributes from JMX are requested.

Not specifying parameters for JMX discovery or trying to receive all attributes for a wide range like `*:type=*,name=*` may lead to potential performance problems.

This item key is supported since Zabbix Java gateway 3.4.

Item key examples:

```
jmx.discovery  #Retrieve all JMX MBean attributes
jmx.discovery[beans]  #Retrieve all JMX MBeans
jmx.discovery[attributes,"*:type=GarbageCollector,name=*"]  #Retrieve all
garbage collector attributes
jmx.discovery[beans,"*:type=GarbageCollector,name=*"]  #Retrieve all garbage
collectors
```

This item returns a JSON object. For example, in the discovery of MBean attributes (reformatted for clarity):

```
{
   "data":[
      {
         "{#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"
        }
    ]
}
```

In the discovery of MBeans (reformatted for clarity):

```
{
    "data":[
        {
            "{#JMXDOMAIN}":"java.lang",
            "{#JMXTYPE}":"GarbageCollector",
            "{#JMXOBJ}":"java.lang:type=GarbageCollector,name=PS Scavenge",
            "{#JMXNAME}":"PS Scavenge"
        }
    ]
}
```

Supported macros

The following macros are supported for use in the discovery rule filter and prototypes of items, triggers and graphs:

Macro	Description
Discovery of MBean attributes	
{#JMXVALUE}	Attribute value.
{#JMXTYPE}	Attribute type.
{#JMXOBJ}	Object name.
{#JMXDESC}	Object name including attribute name.
{#JMXATTR}	Attribute name.
Discovery of MBeans	
{#JMXDOMAIN}	MBean domain. (*Zabbix reserved name*)
{#JMXOBJ}	Object name. (*Zabbix reserved name*)

Macro	Description
{#JMX<key property>}	MBean properties (like {#JMXTYPE}, {#JMXNAME}). Some important notes to pay attention to when defining MBean attribute name that is created from MBean property name by the following algorithm: * attribute name case is changed to uppercase; * attribute name case is ignored (no LLD macro is generated) in case it consists of not supported characters. Supported characters can be described by the following regular expression: “A-Z0-9_\.”; * if an attribute name is called “*obj*” or “*domain*” it will be replaced with the values of Zabbix properties {#JMXOBJ} and {#JMXDOMAIN} (supported since Zabbix 3.4.3.).

Please consider this jmx.discovery (with “beans” mode) example. MBean has the following properties defined:

```
name=test
тип=Type
attributes []=1,2,3
Name=NameOfTheTest
domAin=some
```

As a result of JMX discovery, the following LLD macros will be generated:

- {#JMXDOMAIN} - Zabbix internal, describing the domain of MBean
- {#JMXOBJ} - Zabbix internal, describing MBean object
- {#JMXNAME} - created from “name” property

Ignored properties are:

- тип : its name contains unsupported characters (non-ASCII)
- attributes[] : its name contains unsupported characters (square brackets are not supported)
- Name : it's already defined (name=test)
- domAin : it's a Zabbix reserved name

Let’s review two more practical examples of a LLD rule creation with the use of Mbean. To understand the difference between a LLD rule collecting Mbeans and a LLD rule collecting Mbean attributes better please take a look at following table:

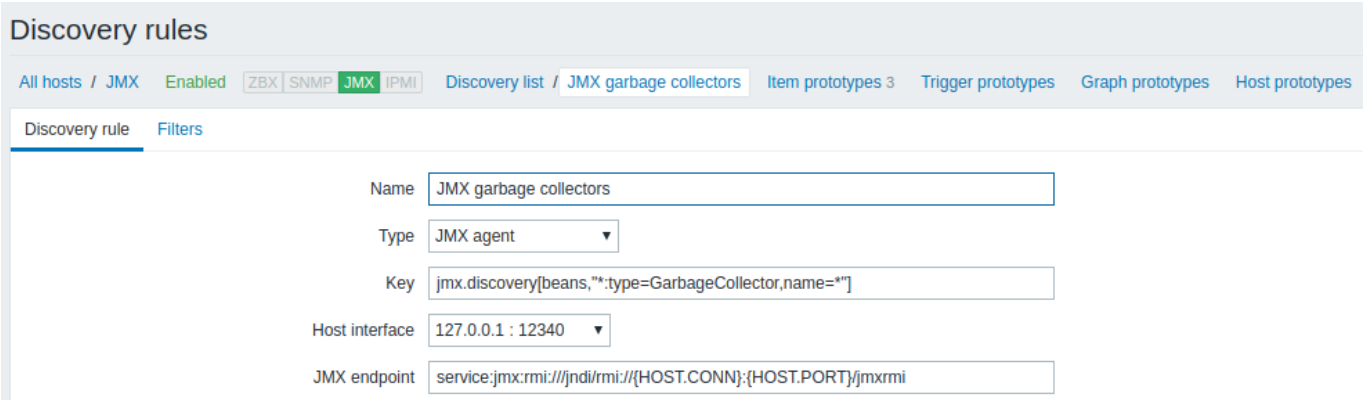
MBean1	MBean2	MBean3
MBean1Attribute1	MBean2Attribute1	MBean3Attribute1
MBean1Attribute2	MBean2Attribute2	MBean3Attribute2
MBean1Attribute3	MBean2Attribute3	MBean3Attribute3

LLD rule collecting Mbeans

This rule will return 3 objects: the top row of the column: MBean1, MBean2, MBean3.

For more information about objects please refer to supported macros table, *Discovery of MBeans* section.

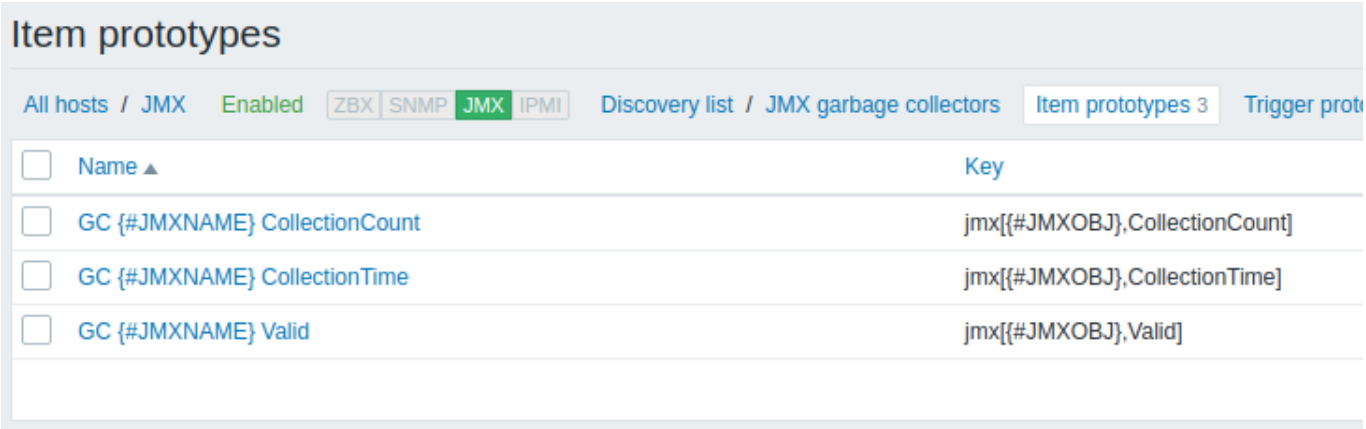
Discovery rule configuration collecting Mbeans (without the attributes) looks like the following:



Key used:

```
jmx.discovery[beans,"*:type=GarbageCollector,name=*"]
```

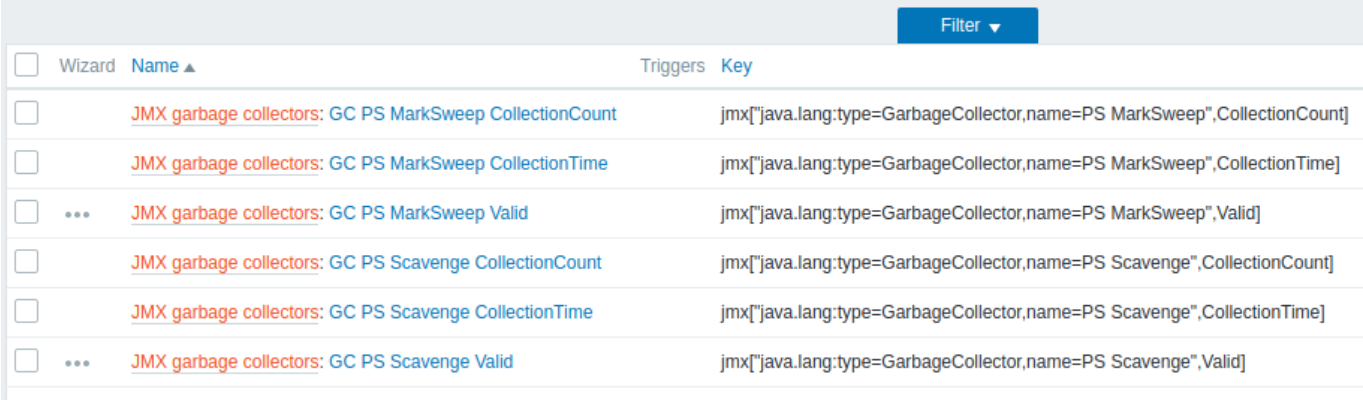
All the garbage collectors without attributes will be discovered. As Garbage collectors have the same attribute set, we can use desired attributes in item prototypes the following way:



Keys used:

```
jmx[{#JMXOBJ},CollectionCount]
jmx[{#JMXOBJ},CollectionTime]
jmx[{#JMXOBJ},Valid]
```

LLD discovery rule will result in something close to this (items are discovered for two Garbage collectors):

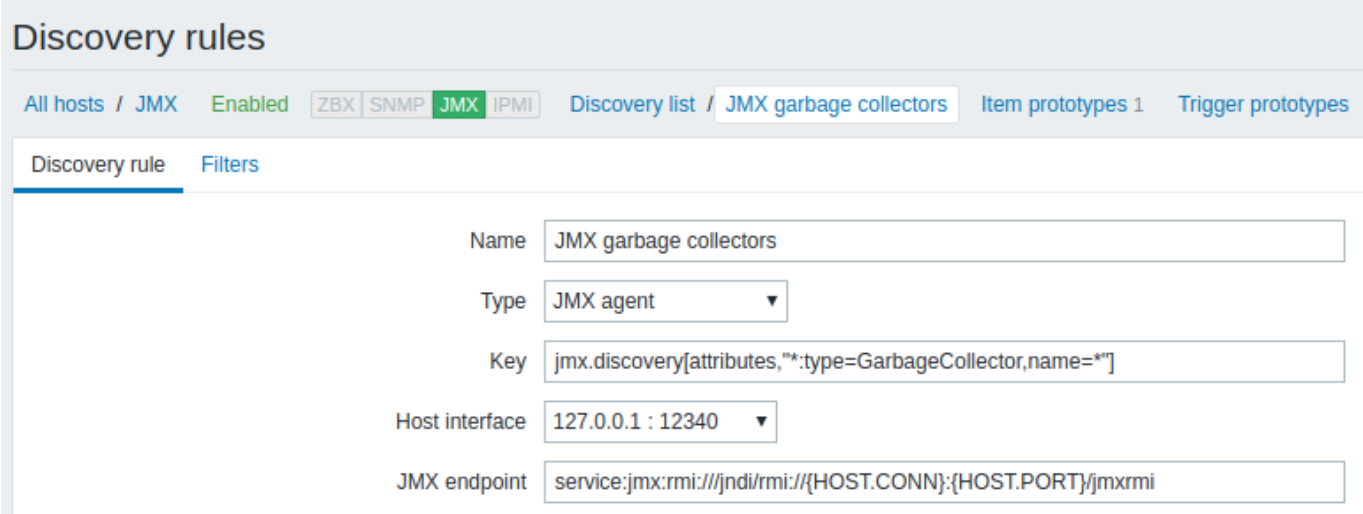


LLD rule collecting Mbean attributes

This rule will return 9 objects with the following fields: MBean1Attribute1, MBean2Attribute1, Mbean3Attribute1,MBean1Attribute2,MBean2Attribute2, Mbean3Attribute2, MBean1Attribute3, MBean2Attribute3, Mbean3Attribute3.

For more information about objects please refer to supported macros table, *Discovery of MBean attributes* section.

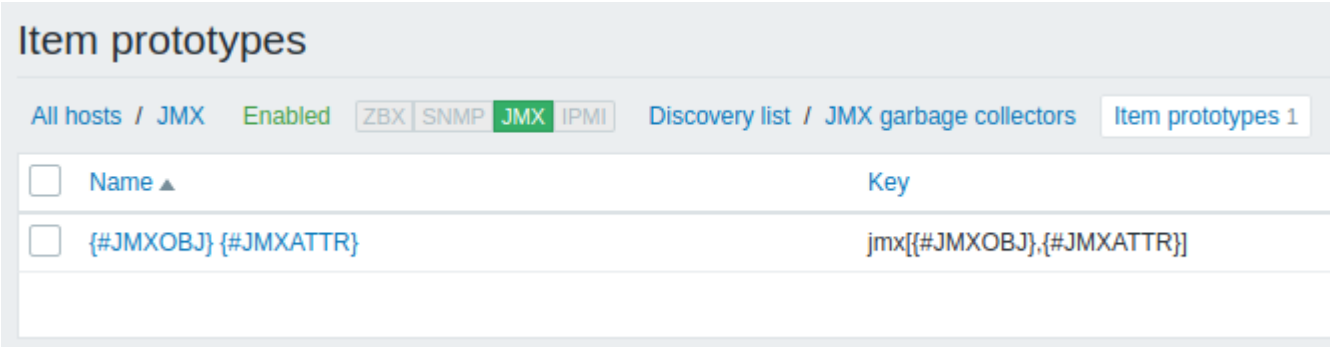
Discovery rule configuration collecting Mbean attributes looks like the following:



Key used:

```
jmx.discovery[attributes,"*:type=GarbageCollector,name=*"]
```

All the garbage collectors with a single item attribute will be discovered.



In this particular case an item will be created from prototype for every MBean attribute. The main drawback of this configuration is that trigger creation from trigger prototypes is impossible as there is only one item prototype for all attributes. So this setup can be used for data collection, but is not recommended for automatic monitoring.