19. API

概览

Zabbix API允许你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。它广泛用于：

- 创建新的应用程序以使用Zabbix
- 将Zabbix与第三方软件集成；
- 自动执行常规任务。

Zabbix API是基于Web的API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：

- 该API包含一组独立的方法；
- 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的更多信息可以在 JSON-RPC 2.0 规范 和 JSON 格式主页中找到。

结构

Zabbix API由许多名义上分组的独立API方法组成。每个方法执行一个特定任务。例如，方法 `host.create` 隶属于 *host* 这个API分组 ，用于创建新主机。历史上API分组有时被称为“类”。

大多数API至少包含四种方法： `get` `create` `update` 和 `delete` ，分别是检索，创建，更新和删除数据，但是某些API提供一套完全不同的一组方法。

执行请求

当完成了前端的安装配置后，你就可以使用远程HTTP请求来调用API为此，需要向 `api_jsonrpc.php` 位于前端目录中的文件发送HTTP POST请求。例如，如果你的Zabbix前端安装在 *http://company.com/zabbix*， 那么用HTTP请求来调用 `apiinfo.version` 方法就如下面这样：

```
POST http://company.com/zabbix/api_jsonrpc.php HTTP/1.1
Content-Type: application/json-rpc

{"jsonrpc":"2.0","method":"apiinfo.version","id":1,"auth":null,"params":{}}
```

请求的 `Content-Type` 头部必须设置为以下值之一： `application/json-rpc`, `application/json` 或 `application/jsonrequest`

你可以使用任何HTTP客户端或JSON-RPC测试工具手动执行API请求，但对于开发应用程序，我们建议使用社区维护的程序库

示例

以下部分将详细介绍一些使用示例。

验证

在访问Zabbix中的任何数据之前，你需要登录并获取身份验证令牌。这可以使用该 user.login 方法完成。让我们假设你想要以标准Zabbix Admin用户身份登录。然后，你的JSON请求将如下所示：

```
{
    "jsonrpc": "2.0",
    "method": "user.login",
    "params": {
        "user": "Admin",
        "password": "zabbix"
    },
    "id": 1,
    "auth": null
}
```

让我们仔细看看示例请求对象。它具有以下属性：

- jsonrpc - API使用的JSON-RPC协议的版本; Zabbix API实现的JSON-RPC版本是2.0;
- method – 被调用的API方法名;
- params – 将被传递给API方法的参数;
- id – 请求的任意标识符;
- auth –用户认证令牌; 因为我们还没有一个，它的设置null

如果你正确提供了凭据API返回的响应将包含用户身份验证令牌：

```
{
    "jsonrpc": "2.0",
    "result": "0424bd59b807674191e7d77572075f33",
    "id": 1
}
```

响应对象又包含以下属性：

- jsonrpc - JSON-RPC协议的版本;
- result – 方法返回的数据;
- id – 相应请求的标识符。

检索主机

我们现在有一个有效的用户身份验证令牌，可以用来访问Zabbix中的数据。 例如，让我们使用 host.get 方法检索所有已配置主机的ID主机名和接口 ：

```
{
    "jsonrpc": "2.0",
    "method": "host.get",
```

```
    "params": {
        "output": [
            "hostid",
            "host"
        ],
        "selectInterfaces": [
            "interfaceid",
            "ip"
        ]
    },
    "id": 2,
    "auth": "0424bd59b807674191e7d77572075f33"
}
```

请注意， `auth` 属性现在设置为我们通过调用`user.login`方法获得的身份验证令牌。

响应对象将包含有关主机的请求的数据：

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "hostid": "10084",
            "host": "Zabbix server",
            "interfaces": [
                {
                    "interfaceid": "1",
                    "ip": "127.0.0.1"
                }
            ]
        }
    ],
    "id": 2
}
```

出于性能原因，我们建议始终列出要检索的对象属性，并避免检索所有内容。

创建新监控项

让我们使用从上一个请求`host.get`中获得的数据，在主机 “Zabbix server” 上创建一个新 监控项 。这个可以通过使用方法 `item.create`。

```
{
    "jsonrpc": "2.0",
    "method": "item.create",
    "params": {
        "name": "Free disk space on $1",
        "key_": "vfs.fs.size[/home/joe/,free]",
        "hostid": "10084",
        "type": 0,
```

```
        "value_type": 3,
        "interfaceid": "1",
        "delay": 30
    },
    "auth": "0424bd59b807674191e7d77572075f33",
    "id": 3
}
```

成功的响应将包含新创建监控项的ID，可用于在以后请求中引用监控项：

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "24759"
        ]
    },
    "id": 3
}
```

`item.create` 方法和其他的创建/create/方法，也可以接受对象数组，并通过一次API调用中创建多个监控项。

创建多个触发器

因此，如果create方法接受数组，我们可以添加多个触发器，像这样^-^：

```
{
    "jsonrpc": "2.0",
    "method": "trigger.create",
    "params": [
        {
            "description": "Processor load is too high on {HOST.NAME}",
            "expression": "{Linux
server:system.cpu.load[percpu,avg1].last()}>5",
        },
        {
            "description": "Too many processes on {HOST.NAME}",
            "expression": "{Linux server:proc.num[].avg(5m)}>300",
        }
    ],
    "auth": "0424bd59b807674191e7d77572075f33",
    "id": 4
}
```

操作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
}
```

更新一个监控项

启用监控项，即将其状态设置为“0”：

```
{
    "jsonrpc": "2.0",
    "method": "item.update",
    "params": {
        "itemid": "10092",
        "status": 0
    },
    "auth": "0424bd59b807674191e7d77572075f33",
    "id": 5
}
```

操作成功的响应将包含被更新的触发器的ID。

```
{
    "jsonrpc": "2.0",
    "result": {
        "itemids": [
            "10092"
        ]
    },
    "id": 5
}
```

`item.update` 方法以及其他更新方法也可以接受对象数组，并通过一次API调用更新多个监控项。

更新多个触发器

启用多个触发器，即将其状态设置为0：

```
{
    "jsonrpc": "2.0",
```

```
    "method": "trigger.update",
    "params": [
        {
            "triggerid": "13938",
            "status": 0
        },
        {
            "triggerid": "13939",
            "status": 0
        }
    ],
    "auth": "0424bd59b807674191e7d77572075f33",
    "id": 6
}
```

成功的响应将包含被更新的触发器的ID数组：

```
{
    "jsonrpc": "2.0",
    "result": {
        "triggerids": [
            "13938",
            "13939"
        ]
    },
    "id": 6
}
```

这是更新的首选方法。一些API方法 `host.massupdate` 允许编写更简单的代码，但不建议使用这些方法，因为它们将在未来的版本中删除。

错误处理

到目前为止，我们试过的一切工作正常。但是，如果我们尝试对API调用不正确会发生什么？让我们尝试通过调用host.create创建另一个主机， 但省略一个必填 `groups` 参数。

```
{
    "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"
            }
        ]
    },
    "id": 7,
    "auth": "0424bd59b807674191e7d77572075f33"
}
```

这个请求的返回会包含一个错误信息：

```
{
    "jsonrpc": "2.0",
    "error": {
        "code": -32602,
        "message": "Invalid params.",
        "data": "No groups for host \"Linux server\"."
    },
    "id": 7
}
```

如果发生错误，响应对象将包含`error` 而不是 `result` 属性，同时error将具有以下数据的属性：

- `code` - 错误代码；
- `message` - 一个简短的错误摘要；
- `data` - 更详细的错误消息。

错误可能发生在不同的情况下，例如，使用不正确的输入值，会话超时或试图访问不存在的对象。你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版本控制，自Zabbix 2.0.4开始，API的版本与Zabbix本身的版本相匹配。 你可以使用 `apiinfo.version` 方法查找你正在使用的API的版本。这对于调整应用程序以使用特定于版本的功能非常有用。

我们保证在主要版本内部具有向后兼容性。当在主要版本之间进行向后不兼容的更改时，我们通常将旧功能在下一个版本中保留为已弃用，并且仅在此后的版本中将其删除。有时，我们可能会删除主要版本之间的功能，而不提供任何向后兼容性。重要的是，你不要依赖任何弃用的功能，并尽快迁移到较新的替代品。

你可以在 API更改日志中跟踪对API所做的所有更改。

进一步阅读

你现在知道了足够开始使用 Zabbix API，但不要停在这里。为了进一步阅读，我们建议你查看 可用的API列表。