H3C 高端路由器 RESTful 二次开发指南

前 言

本文档主用来指导用户使用 RESTful 功能客户端配置和管理设备。

本文档适用于 SR8800-X、SR8800-X-S、SR8800-F、CR16000-F、CR16000-M、RX8800 路由器。

前言部分包含如下内容：

读者对象

本手册主要适用于如下工程师：

- 具有一定 XML 和 RESTful 技术基础的网络规划人员
- 负责网络配置和维护，且具有一定 XML 和 RESTful 技术基础的网络管理员

本书约定

1. 命令行格式约定

格　式	意　义
粗体	命令行关键字（命令中保持不变、必须照输的部分）采用加粗字体表示。
斜体	命令行参数（命令中必须由实际值进行替代的部分）采用斜体表示。
[ ]	表示用“[ ]”括起来的部分在命令配置时是可选的。
{ x | y | ... }	表示从多个选项中仅选取一个。
[x | y | ...]	表示从多个选项中选取一个或者不选。
{ x | y | ... }*	表示从多个选项中至少选取一个。
[x | y | ...]*	表示从多个选项中选取一个、多个或者不选。
&<1-n>	表示符号&前面的参数可以重复输入1～n次。
#	由“#”号开始的行表示为注释行。

2. 各类标志

本书还采用各种醒目标志来表示在操作过程中应该特别注意的地方，这些标志的意义如下：

 警告	该标志后的注释需给予格外关注，不当的操作可能会对人身造成伤害。
 注意	提醒操作中应注意的事项，不当的操作可能会导致数据丢失或者设备损坏。

警告	该标志后的注释需给予格外关注，不当的操作可能会对人身造成伤害。
提示	为确保设备配置成功或者正常工作而需要特别关注的操作或信息。
说明	对操作内容的描述进行必要的补充和说明。
窍门	配置、操作、或使用设备的技巧、小窍门。

3. 图标约定

本书使用的图标及其含义如下：

	该图标及其相关描述文字代表一般网络设备，如路由器、交换机、防火墙等。
ROUTER	该图标及其相关描述文字代表一般意义下的路由器，以及其他运行了路由协议的设备。
SWITCH	该图标及其相关描述文字代表二、三层以太网交换机，以及运行了二层协议的设备。
	该图标及其相关描述文字代表无线控制器、无线控制器业务板和有线无线一体化交换机的无线控制引擎设备。
	该图标及其相关描述文字代表无线接入点设备。
	该图标及其相关描述文字代表无线终结单元。
	该图标及其相关描述文字代表无线终结者。
	该图标及其相关描述文字代表无线Mesh设备。
	该图标代表发散的无线射频信号。
	该图标代表点到点的无线射频信号。
	该图标及其相关描述文字代表防火墙、UTM、多业务安全网关、负载均衡等安全设备。
	该图标及其相关描述文字代表防火墙插卡、负载均衡插卡、NetStream插卡、SSL VPN插卡、IPS插卡、ACG插卡等安全插卡。

4. 示例约定

由于设备型号不同、配置不同、版本升级等原因，可能造成本手册中的内容与用户使用的设备显示信息不一致。实际使用中请以设备显示的内容为准。

本手册中出现的端口编号仅作示例，并不代表设备上实际具有此编号的端口，实际使用中请以设备上存在的端口编号为准。

资料意见反馈

如果您在使用过程中发现产品资料的任何问题，可以通过以下方式反馈：

E-mail：info@h3c.com

感谢您的反馈，让我们做得更好！

目 录

1 RESTful 协议介绍

REST（Representational State Transfer，表象化状态转变）是一种面向资源的轻量级、跨平台、跨语言的程序架构，具有结构清晰、易于管理和扩展等特点。RESTful 是遵循 REST 程序架构的一种应用。设备提供了 RESTful API 接口，用户可以通过 RESTful 功能操作 RESTful API 接口，从而实现对设备进行配置和维护。

本文档用来指导用户使用 RESTful 客户端，通过 RESTful API 配置和维护设备。

1.1 RESTful工作机制

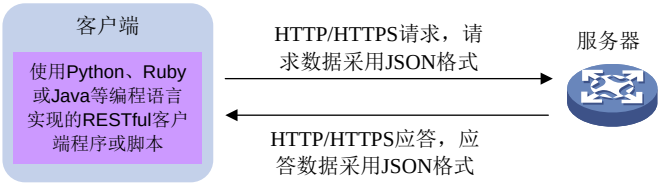
如图 1 所示，RESTful 采用 C/S 模型。RESTful 客户端为使用 Python、Ruby 或 Java 等编程语言开发出的 RESTful 客户端程序或脚本。RESTful 服务器为网络设备。通过 RESTful 功能配置和维护设备的过程为：

(1) 客户端向服务器发送 HTTP/HTTPS 请求报文，通过 HTTP 的方法来操作指定的 RESTful API 接口。RESTful 支持的 HTTP 操作方法包括 GET、PUT、POST 和 DELETE。

(2) 服务器根据 HTTP/HTTPS 请求报文，完成对指定 RESTful API 接口的操作后，通过 HTTP/HTTPS 应答报文将操作结果返回给客户端。

在 HTTP/HTTPS 请求和应答报文中，请求和应答数据均采用 JSON 格式进行编码。

图1 RESTful 功能示意图



1.2 RESTful报文格式

RESTful 基于 HTTP/HTTPS 协议，通过 HTTP/HTTPS 请求和应答报文在 RESTful 客户端和服务器之间进行报文交互。RESTful 请求和应答报文（即 HTTP/HTTPS 请求和应答报文）使用标准的 HTTP 格式。

说明

RESTful 当前支持使用 UTF-8 进行传输。

1.2.1 请求报文格式

HTTP/HTTPS 请求报文包括请求行、请求头和请求正文（HTTP Body）几个部分。

- 请求行：由操作方法、URL 和 HTTP 版本组成。
  - 操作方法：取值为 GET、PUT、POST 或 DELETE。

- URL：RESTful 操作的对象为资源，URL 需要标识出待操作资源的位置。以 HTTP 为例，URL 的格式为 http://ip-address:port/api/v1/resource-locator?parameter。

  表1 URL 字段说明

字段	说明
http	请求报文协议类型，取值为http或https
ip-address:port	服务器的IP地址和端口号 如果采用HTTP、HTTPS的缺省端口号，则不需要携带端口号port
api/v1	固定字段
resource-locator	待操作资源的位置 查看RESTful API手册，可以获取资源的位置 RESTful API手册的详细介绍，请参见“2.4.1 RESTful API文档介绍”
?parameter	当前操作携带的参数，包括index、filter-condition、select-fields和getbulk-count URL中可以不携带参数 关于参数的详细介绍，请参见“4.2 操作参数介绍”

- 请求头：标准的 HTTP 请求头，包括请求的压缩方式、认证头、内容类型、内容长度等。
- 请求正文：请求数据采用 JSON 格式编码，封装在请求正文部分。

```
POST http://192.168.100.91/api/v1/Syslog/LogHosts HTTP/1.1
Accept-Encoding: gzip,deflate
X-Auth-Token: 400001b81dbfc50884818f45b686fd81ceaa
Content-Type: application/json
Content-Length: 78
Host: 192.168.100.91
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

{
   "Address": "1.1.1.3",
   "VRF": "b",
   "Port": 518,
   "Facility": 184
}
```

1.2.2 应答报文格式

HTTP/HTTPS 应答报文包括状态行、响应头和响应正文几个部分。

- 状态行：由 HTTP 版本、状态码和原因短语（Reason Phrase）组成。状态码用来标识操作执行是否成功。状态码 200～299 表示成功，状态码 400 及以上表示失败。常用状态码的含义如表 2 所示。

表2 常用状态码的含义

状态码	含义
200	成功
201	创建成功
204	成功，但无返回结果
401	认证失败
404	请求的资源不存在
405	HTTP方法不允许执行，比如，对只读资源进行POST操作
406	RESTful客户端和服务器端数据编码格式等不一致
500	设备内部处理错误
501	发送CONNECT等不支持RESTful的HTTP方法

- 响应头：标准的 HTTP 响应头，包括内容类型、响应时间等。
- 响应正文：响应数据采用 JSON 格式编码，封装在响应正文部分。

如下示例为表示操作成功的应答报文。

```
HTTP/1.1 201 Created
Content-Type: application/json charset=UTF-8
Location: http://192.168.100.91/api/v1/Syslog/LogHosts?index=Address%3D1.1.1.3%3BVRF%3Db
Server: HTTPD
Date: Fri, 12 Jun 2015 14:46:33 GMT
Connection: Keep-Alive
Transfer-Encoding: chunked
X-XSS-Protection: 1;mode-block
X-Content-Type-Options: nosniff
```

如下示例为表示操作失败的应答报文。

```
HTTP/1.1 409 Conflict
Content-Type: application/json charset=UTF-8
Server: HTTPD
Date: Sat, 16 Jul 2022 09:49:09 GMT
Connection: close
X-XSS-Protection: 1;mode-block
X-Content-Type-Options: nosniff
{"errors":[{"error":{"error-type":"rpc","error-tag":"bad-element","error-path":"Port","e
rror-message":"The value must be less than or equal to 65535."}}]}
```

2 使用 RESTful 功能配置和维护设备

2.1 使用RESTful功能配置和维护设备的通用流程

管理员通过 RESTful 功能配置和管理设备时，通常需要按照如下流程来操作：

(1) 配置 RESTful 访问方式

设备支持 HTTP 和 HTTPS 两种 RESTful 访问方式。不同访问方式的配置方法有所不同。

(2) 在设备上为 RESTful 用户设置权限

不同访问方式需要的用户权限不同：

- HTTP 访问方式，需要为用户设置 HTTP 访问权限。
- HTTPS 访问方式，需要为用户设置 HTTPS 访问权限。

(3) 准备 RESTful 客户端

根据用户自身的情况，选择已有的 RESTful 客户端，或利用 Python、Ruby、Java 等编程语言开发出 RESTful 客户端软件、程序或脚本。

(4) 准备 RESTful 请求使用的报文

依据对应模块的 API 文档，构造出可以下发的 RESTful 请求。

(5) 使用 RESTful 客户端连接到设备

RESTful 客户端发送请求，与设备建立 HTTP/HTTPS 连接。

(6) 将验证好的请求集成到客户端脚本或程序

把前面测试好的脚本集成进客户端程序中。例如，把其中需要替换的部分换成自己的程序变量等。

(7) 断开 RESTful 客户端与设备的连接

2.2 配置RESTful访问方式

2.2.1 配置通过基于 HTTP 的 RESTful 方式登录设备

(1) 进入系统视图。

system-view

(2) （可选）配置基于 HTTP 的 RESTful 功能的端口号。

restful http port *port-number*

缺省情况下，基于 HTTP 的 RESTful 功能的端口号为 80。

(3) 开启基于 HTTP 的 RESTful 功能。

restful http enable

缺省情况下，基于 HTTP 的 RESTful 功能处于关闭状态。

(4) （可选）使用 ACL 限制 HTTP 客户端和设备建立 TCP 连接。

（IPv4 网络）

http acl { *advanced-acl-number* | *basic-acl-number* }

（IPv6 网络）

http ipv6 acl { *advanced-acl-number* | *basic-acl-number* }

缺省情况下，允许所有 HTTP 客户端和设备建立 TCP 连接。

2.2.2 配置通过基于 HTTPS 的 RESTful 方式登录设备

(1) 进入系统视图。

system-view

(2) （可选）配置基于 HTTPS 的 RESTful 功能的端口号。

restful https port *port-number*

缺省情况下，基于 HTTPS 的 RESTful 功能的端口号为 443。

(3) 开启基于 HTTPS 的 RESTful 功能。

restful https enable

缺省情况下， 基于 HTTPS 的 RESTful 功能处于关闭状态。

(4) （可选）使用 ACL 限制 HTTPS 客户端和设备建立 TCP 连接。

（IPv4 网络）

https acl { *advanced-acl-number* | *basic-acl-number* }

（IPv6 网络）

https ipv6 acl { *advanced-acl-number* | *basic-acl-number* }

缺省情况下，允许所有 HTTPS 客户端和设备建立 TCP 连接。

2.3 在设备上为RESTful用户设置权限

使用 RESTful 功能配置和维护设备前，需要确保 RESTful 用户具有对应的操作权限。

2.3.1 确定 RESTful 用户需要的权限

设备通过 RBAC（Role Based Access Control，基于角色的访问控制）实现基于角色的用户访问权限控制。

RESTful 的底层使用的是 NETCONF。因此，需要按照 NETCONF 的要求，通过 RBAC 为 RESTful 用户赋予相应的访问权限。NETCONF 权限分为 3 个部分：

- 协议级权限：执行 rpc-operation 需要的权限。RESTful 操作和 NETCONF RPC 操作的映射关系，及需要的权限，如表 3 所示。

  表3 RESTful 操作和 NETCONF RPC 映射关系

RESTful 操作	NETCONF RPC 操作	需要配置的权限
GET	rpc/get	read
POST	rpc/edit-config: create	write
POST	rpc/action	execute
PUT	rpc/edit-config: merge	write
DELETE	rpc/edit-config: delete	write

- 模块级权限：用户对模块、表、行、组、列的访问权限。例如，配置用户具有操作接口模块资源的权限时，需要执行 **rule** *number* **permit read write execute xml-element ifmgr/**命令。
- 实例级权限：用户对系统资源（如 VPN 实例、接口、VLAN）的操作权限。缺省情况下，用户具有操作所有系统资源的权限。

一个请求只有同时具有上述三部分的权限，才能通过 RBAC 的检查，执行相应的操作。

例如，下例中为 RESTful 用户赋予了如下权限：

- 协议级权限：具有 rpc/edit-config/config/top 的写权限。
- 模块级权限：具有 Ifmgr/Interfaces/Interface 表的写权限。
- 实例级权限：具有接口索引为 3 的接口的写权限。

```
#
role name restful
 rule 1 permit write xml-element rpc/edit-config/config/top
 rule 2 permit write xml-element ifmgr/interfaces/interface
 interface policy deny
  permit interface GigabitEthernet2/0/1
#
```

不同用户角色的操作权限有所不同：

- 缺省用户角色 network-admin、mdc-admin、context-admin 具有操作对应设备/MDC/Context 的所有功能和资源（除安全日志文件管理相关命令 **display security-logfile summary**、**info-center security-logfile directory**、**security-logfile save** 之外）的权限。
- 缺省用户角色 network-operator、context-operator 和 mdc-operator 默认只有读权限和部分可执行权限。可以通过配置，为这些用户角色添加操作权限。
- 其他用户角色必须配置后才有对应的操作权限。

请根据需要配置用户角色的权限，并为用户授权相应的用户角色。

2.3.2 定义 RESTful 用户角色规则

(1) 进入系统视图。

system-view

(2) 创建用户角色，并进入用户角色视图。

role name *role-name*

(3) 配置用户具有执行 rpc-operation 操作的权限。

rule *number* **permit** { **execute** | **read** | **write** } * **xml-element rpc/**

通过 **rule** *number* **permit** { **execute** | **read** | **write** } * **xml-element rpc/?**可以查看具体操作的列表。不指定具体操作时，表示所有操作。

(4) 配置用户执行 XML 元素具体模块的权限。

rule *number* { **deny** | **permit** } { **execute** | **read** | **write** } * **xml-element** [*module-xpath*]

通过 **rule** *number* { **deny** | **permit** } { **execute** | **read** | **write** } * **xml-element ?** 可以查看具体模块列表，通过 **rule** *number* { **deny** | **permit** } { **execute** | **read** | **write** } * **xml-element** *module-name/?* 可以查看模块中具体表的列表。不指定具体模块和表时表示所有模块/所有表。

(5) 请根据需要配置相应的系统资源访问权限。

- 依次执行以下命令，配置接口资源控制策略。

  interface policy deny

permit interface *interface-list*

缺省情况下，未定义允许操作的接口列表，用户角色没有操作任何接口的权限。

可以多次执行此命令向接口列表中添加允许操作的接口。

- 依次执行以下命令，配置 VLAN 资源控制策略。

  vlan policy deny

  permit vlan *vlan-id-list*

  缺省情况下，未定义允许操作的 VLAN 列表，用户没有操作任何 VLAN 的权限。

  可以多次执行此命令向 VLAN 列表中添加允许操作的 VLAN。

- 依次执行以下命令，配置 VPN 资源控制策略。

  vpn-instance policy deny

  permit vpn-instance *vpn-instance-name&<1-10>*

  缺省情况下，未定义允许操作的 VPN 实例列表，用户没有操作任何 VPN 实例的权限。

  可以多次执行此命令向 VPN 实例列表中添加允许操作的 VPN 实例。

- 依次执行以下命令，配置安全域资源控制策略。

  security-zone policy deny

  permit security-zone *security-zone-name&<1-10>*

  缺省情况下，未定义允许操作的安全域列表，用户没有操作任何安全域的权限。

  可以多次执行此命令向安全域列表中添加允许操作的安全域。

2.3.3 配置 RESTful 用户

1. 功能简介

本配置用来为 RESTful 用户指定如下属性：

- 根据 RESTful 访问方式，指定正确的服务类型：使用 HTTP 访问方式时，用户的服务类型为 HTTP；使用 HTTPS 访问方式时，用户的服务类型为 HTTPS。
- 为用户授权用户角色，使其具有所需权限。

本文以本地认证为例，介绍配置过程。使用远程认证的配置方式请参见产品对应版本的《AAA 配置指导》。

2. 配置步骤

(1) 进入系统视图。

system-view

(2) 添加设备管理类本地用户，并进入设备管理类本地用户视图。

local-user *user-name* **class manage**

(3) 设置本地用户的密码。

（非 FIPS 模式）

password [{ **hash** | **simple** } *string*]

（FIPS 模式）

password

在非 FIPS 模式下，可以不为本地用户设置密码；在 FIPS 模式下，必须且只能通过交互式方式设置明文密码，否则用户的本地认证不能成功。

(4) 设置本地用户可以使用的服务类型。请根据访问方式选择其中一项进行配置

（非 FIPS 模式）

service-type { **http** | **https** } *

（FIPS 模式）

service-type https

缺省情况下，本地用户不能使用任何服务类型。

(5) 设置本地用户的角色。

authorization-attribute user-role *role-name*

2.4 准备RESTful请求使用的报文

2.4.1 RESTful API 文档介绍

每一个支持 RESTful 的模块都对应一个 RESTful API 文档，文档命名遵循如下格式，其中 XXX 代表模块名：

XXX REST API Reference.docx

RESTful API 文档包括下面几个部分：

- 模块资源整体描述：如图 2 所示，模块资源整体描述用一个列表描述本模块的资源的整体情况，包括资源名称、资源所属基础 URL、支持哪些操作。

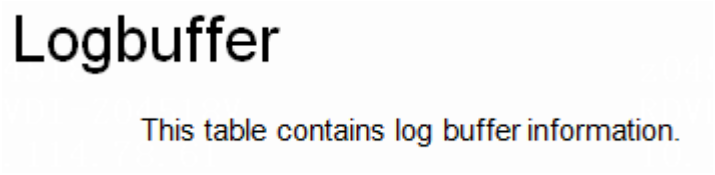
图2 模块资源整体描述示意图

Resource Summary for Syslog

Resource(Table)	Base URL	HTTP Method			
		GET	POST	PUT	DELETE
LogBuffer	/api/v1/Syslog/LogBuffer	Y	N	Y	N
LogHosts	/api/v1/Syslog/LogHosts	Y	Y	Y	Y
Logs	/api/v1/Syslog/Logs	Y	N	N	N
Configuration	/api/v1/Syslog/Configuration	Y	N	Y	N
OutputRules	/api/v1/Syslog/OutputRules	Y	Y	Y	Y
Logfile	/api/v1/Syslog/Logfile	Y	N	Y	N

- 资源详细信息：包括资源描述和属性详细信息表。
  - 资源描述：提供资源本身的信息和全局约束方面信息，如图 3 所示。

图3 资源描述示意图



- 属性详细信息表：提供列名称、列描述、列数据类型和约束等资源详细信息，如图 4 所示。

图4 属性详细信息表示意图

Properties

Column name	Column description	Column type	Data type and restrictions	Remarks
State	Status of the log buffer	N/A	Enumeration: • enable • disable	N/A
BufferSize	Maximum log buffer size configured by the user	N/A	Unsigned integer. Value range: 0 to 65535	N/A
BufferSizeLimit	Maximum log buffer size supported by the device	N/A	Unsigned integer. Value range: 0 to 65535	N/A
LogsCount	Number of logs stored in the log buffer	N/A	Unsigned integer	Readonly

Column name	Column description	Column type	Data type and restrictions	Remarks
DroppedLogsCount	Number of dropped logs	N/A	Unsigned integer	Number of logs dropped after t buffer size was decreased to a that is smaller th

提示

- 除非特殊说明，所有模块的 GET 操作返回结果中的列名称和属性详细信息表中的列名称一致。
- Coware RESTful API 文档描述的是所有模块、所有资源的全集。模块和资源的实际支持情况与设备型号有关，请以设备实际情况为准。

2.4.2 使用 RESTful API 手册构造 RESTful 请求

RESTful 请求报文（即 HTTP/HTTPS 请求报文）包括请求行、请求头和请求正文（HTTP Body）几个部分。请求报文中的如下部分需要根据 RESTful API 手册中的内容进行构造，其余部分遵循 HTTP 标准。

- URL 中的 api/v1/*resource-locator* 取值为待操作资源所属的基础 URL（Basic URL）。
- 请求正文中的请求数据，需要根据 RESTful API 手册中的属性详细信息表来构造。

构造 RESTful 请求报文的步骤为：

(1) 确认待操作资源所属的模块，找到对应模块的 RESTful API 文档。

(2) 在 RESTful API 文档的模块资源整体描述表中，查找待操作资源所属的基础 URL。根据基础 URL 构造 RESTful 请求的 URL。例如，采用 HTTP 方式对地址为 10.1.1.1 的设备上的 VLAN 资源执行创建操作时，请求报文的 URL 地址为：http://10.1.1.1/api/v1/VLAN/VLANs。

(3) 在 RESTful API 文档中，查找待操作资源的属性详细信息表，根据该表中包含的列构造请求数据，采用 JSON 格式对数据进行编码后，将其封装到请求正文部分。例如，在设备上创建 VLAN 10、该 VLAN 的描述信息为 VLAN10、包含的 Access 接口的索引值为 1000 时，请求正文的内容为：

```
{
  "ID": "10",
  "Description": "VLAN 10"
  "AccessPortList": "1000"
}
```

(4) 根据 URL 地址、请求正文及 HTTP 标准的请求头等信息，构造 RESTful 请求报文。

2.5 使用RESTful客户端连接到设备

配置完设备后，客户端通过 HTTP/HTTPS 协议，向设备指定端口的 URL 地址/api/v1/发送携带 JSON 内容体的 HTTP 请求来进行配置。例如，设备地址为 192.168.1.1，则请求地址为：

- HTTP 方式：http://192.168.1.1/api/v1/
- HTTPS 方式：https://192.168.1.1:443/api/v1/

提示

- HTTP/HTTP URL 地址中最后的反斜杠(/)不可省略。
- HTTP 方式的缺省端口号为 80，HTTPS 方式的缺省端口号为 443。可以通过命令行修改端口号。配置方法请参见 “2.2 配置 RESTful 访问方式”。

RESTful 客户端发送登录报文请求与设备建立连接时，登录请求报文的格式为：

```
POST http://192.168.100.91/api/v1/tokens HTTP/1.1
Accept-Encoding: gzip,deflate
Authorization: Basic dGVzdDp0ZXN0
Content-Type: application/json
Content-Length: 0
Host: 192.168.100.91
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
```

设备回复的应答报文为：

```
HTTP/1.1 201 Created
Content-Type: application/json charset=UTF-8
Server: HTTPD
```

```
Date: Fri, 12 Jun 2015 11:43:38 GMT
Connection: Keep-Alive
Transfer-Encoding: chunked

{
   "token-id": "40000142841c72b5d700bc67a55ae1f2567e",
   "link": "http://192.168.100.91/api/v1/tokens/40000142841c72b5d700bc67a55ae1f2567e",
   "expiry-time": "11:43:44"
}
```

在上面的请求报文中，Authorization 的值用标准的 HTTP 认证 basic 方式计算得到。计算方法为：用户名和密码之间增加一个冒号，再将得出的结果字符串用 Base64 算法编码。例如，提供的用户名为 Aladdin、密码为 open sesame，则拼接后的结果就是 Aladdin:open sesame，然后再将其用 Base64 编码，得到 QWxhZGRpbjpvcGVuIHNlc2FtZQ==

后续的请求中需要携带应答报文返回的 token-id，并将其放在 X-Auth-Token 中。例如，获取 LogHosts 配置的请求报文如下：

```
GET http://192.168.100.91/api/v1/Syslog/LogHosts?index=Address%3D1.1.1.1%3BVRF%3Da
HTTP/1.1
Accept-Encoding: gzip,deflate
X-Auth-Token: 400001b81dbfc50884818f45b686fd81ceaa
Host: 192.168.100.91
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
```

上述请求报文的应答报文格式如下：

```
HTTP/1.1 200 OK
Content-Type: application/json charset=UTF-8
Server: HTTPD
Date: Fri, 12 Jun 2015 13:28:24 GMT
Connection: Keep-Alive
Transfer-Encoding: chunked

{"LogHosts":[{"Address":"192.168.1.1","VRF":"","Port":514,"Facility":184}]}
```

提示

客户端应答需要考虑在某些情况下的异常恢复，如连接断开的情况、长时间不能得到结果的情况。

2.6 将验证好的请求集成到客户端脚本或程序

当准备好的 RESTful 请求报文经过验证是正确的后，可以把验证好的 RESTful 请求报文集成到自己的客户端脚本或程序中。如果是无人值守的程序，需要考虑超时、断线等异常情况，以保证长时间情况下也能正确运行。

2.7 断开RESTful客户端与设备的连接

完成对设备的管理后，如果不再使用该 RESTful 连接，则建议断开 RESTful 客户端与设备的连接，以释放资源。

RESTful 客户端通过向当前 token 的 URL（api/v1/tokens/*token-id*）发送 DELETE 来断开 RESTful 客户端与设备的连接。

断开连接时，RESTful 客户端发送的请求报文格式为：

```
DELETE http://192.168.100.91/api/v1/tokens/400001b81dbfc50884818f45b686fd81ceaa HTTP/1.1
Accept-Encoding: gzip,deflate
X-Auth-Token: 400001b81dbfc50884818f45b686fd81ceaa
Host: 192.168.100.91
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
```

上述请求报文的应答报文格式为：

```
HTTP/1.1 204 No Content
Content-Type: application/json charset=UTF-8
Server: HTTPD
Date: Fri, 12 Jun 2015 15:00:33 GMT
Connection: Keep-Alive
Transfer-Encoding: chunked
```

3 RESTful 会话介绍

3.1 会话生命周期

对于 RESTful 会话，从发送登录报文建立连接开始，直到对 token 资源发送 DELETE 断开连接、或者闲置时间超期，在这期间会话一直有效。会话期间属于此会话的每一个请求都是一个新的 TCP 连接。应答完成后 TCP 连接关闭。后续请求依赖登录报文返回的 token-id 来标识其身份。如果最后一个请求之后闲置时间超过 10 分钟，则此会话自动被关闭。如果希望保持会话，可以每隔一段时间，发送一个使用 token-id 来标识的获取 token 的报文给服务器，服务器返回和登录相同的应答报文。

3.2 请求并发

RESTful 会话可以使用多个线程，使用相同的 token-id 来发送请求而互不干扰，但并发数量存在限制。不建议使用这个方式来进行并发。

3.3 最大会话个数

RESTful 支持的最大会话个数依赖于 AAA 会话最大个数的配置。

4 支持的 RESTful 操作

4.1 支持的RESTful操作速览

RESTful 支持的操作如表 4 所示。

表4 RESTful 支持的操作速览表

操作系列	操作名称	说明
获取数据	GET	获取设备的数据
创建数据	POST	创建系统配置，必须不存在才可以
更新数据	PUT	修改系统配置，如果数据不存在，会创建
删除数据	DELETE	删除系统配置，只允许删除创建的资源

4.2 操作参数介绍

执行 RESTful 操作时，通过在 URL 中指定参数，可以对操作的数据进行过滤。RESTful 操作支持的参数包括 index、filter-condition、select-fields 和 getbulk-count。

- 参数 index 格式是 **index**=Indexname=Value，而且必须输入全部的索引列。
- 参数 filter-condition 格式是 **filter-condition**=condition:Name=Value，该参数可以输入索引列或者普通列，但是索引列不能与参数 index 中的列重复。
- 参数 select-fields 格式是 **select-fields=**Name。
- 参数 getbulk-count 格式是 **getbulk-count**=Value，Value 值就是 get-bulk 操作 count 属性的值，表示要获取数据条目的个数。

多个索引或者过滤条件之间用分号分隔，多个参数之间用‘&’号分隔，如果索引或者过滤条件为空值也必须给出等号。例如：

index=Indexname1=value1;indexname2=&filter-condition=Name3=Value3。

4.3 GET操作

4.3.1 GET 操作介绍

GET 操作用来获取系统中的运行和状态数据。

GET 操作不支持在 HTTP BODY 中给出索引或者过滤内容。对于索引或者过滤条件，必须用 HTTP 参数的方式放在 URL 的参数中。

GET 支持过滤和列选择，请求会返回当前资源满足请求的全部属性数据：

- 过滤功能通过 URL 参数 filter-condition 实现，由 condition 来确定匹配方式，比如 more、notMore、less、notLess 等，默认值为 equal。
- 列选择功能通过 URL 参数 select-fields 实现，用以添加希望获取到的列。

参数的内容需要用类似 encodeURI 的方法进行编码。比如，index=Address=1.1.1.1;VRF=a 编码后变为 index=Address%3D1.1.1.1%3BVRF%3Da。其中，%3D 代表字符 “=”、%3B 代表字符 “;”。

4.3.2 GET 操作举例

1. 基于索引的过滤

通过在 URL 中指定 index 参数，可以对 GET 操作的返回数据进行过滤，使得 GET 操作仅返回与 index 参数匹配的数据。如果同一个对象有多个索引，则在 URL 中指定 index 参数时，不同索引之间需要通过分号“;”分隔，如“index=索引列 1=值 1;索引列 2=值 2”。

以获取接口索引为 1281 的接口的全部列信息（IfIndex=1281）为例，客户端发送的报文格式如下。

```
GET http://192.168.56.120/api/v1/Ifmgr/Interfaces?index=IfIndex%3D1281 HTTP/1.1
Accept-Encoding: gzip,deflate
X-Auth-Token: 400001d534ee6ba9b7d4e26e65e0bfa1df9f
Host: 192.168.56.120
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
```

设备收到获取接口信息的请求报文后，将接口索引为 1281 的接口的信息通过如下报文反馈给客户端。

```
HTTP/1.1 200 OK
Content-Type: application/json charset=UTF-8
Server: HTTPD
Date: Mon, 31 May 2021 15:48:36 GMT
Connection: Keep-Alive
Transfer-Encoding: chunked
X-XSS-Protection: 1;mode-block
X-Content-Type-Options: nosniff

{"IfIndex":1281,"Name":"M-GigabitEthernet0\/0\/0","AbbreviatedName":"MGE0\/0\/0","PortIndex":1281,"ifTypeExt":"28","ifType":"6","Description":"manage","AdminStatus":1,"OperStatus":1,"ConfigSpeed":1,"ActualSpeed":1000000,"ConfigDuplex":3,"ActualDuplex":1,"PortLayer":2,"InetAddressIPV4":"192.168.56.120","InetAddressIPV4Mask":"255.255.0.0","MAC":"78-81-C4-32-01-01","ForwardingAttributes":17,"ConfigMTU":1500,"ActualMTU":1500,"ActualBandwidth":1000000,"SubPort":false,"Interval":300,"LastChange":"0Day4Hour36Minute23Second","Actual64Bandwidth":1000000,"IPv4ProtocolStatus":1,"IPv6ProtocolStatus":0}
```

2. GET 指定的列

通过在 URL 中指定 select-fields 参数，可以使得设备仅返回指定的列。如果需要设备返回多个列，需在 URL 中，不同列之间需要通过分号“;”分隔，如“select-fields=列 1;列 2;列 3”。

说明

不论是否通过 select-fields 参数指定索引列，GET 操作返回的数据中始终包括索引列。

---

select-fields 的操作示例如下：

以获取所有接口的接口名、接口简名和接口索引信息为例，客户端发送的报文格式如下。

```
GET
http://192.168.56.120/api/v1/Ifmgr/Interfaces?select-fields=Name%3BAbbreviatedName%3BPortIndex HTTP/1.1
Accept-Encoding: gzip,deflate
X-Auth-Token: 400001d534ee6ba9b7d4e26e65e0bfa1df9f
```

```
Host: 192.168.56.120
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
```

设备收到获取接口信息的请求报文后，将接口索引、接口名、接口简名信息通过如下报文反馈给客户端。

```
HTTP/1.1 200 OK
Content-Type: application/json charset=UTF-8
Server: HTTPD
Date: Mon, 31 May 2021 15:45:15 GMT
Connection: Keep-Alive
Transfer-Encoding: chunked
X-XSS-Protection: 1;mode-block
X-Content-Type-Options: nosniff

{"Interfaces":[{"IfIndex":1281,"Name":"M-GigabitEthernet0\/0\/0","AbbreviatedName":"MGE0
\/0\/0","PortIndex":1281},{"IfIndex":1409,"Name":"NULL0","AbbreviatedName":"NULL0"},{"If
Index":1410,"Name":"InLoopBack0","AbbreviatedName":"InLoop0"},{"IfIndex":1536,"Name":"Lo
opBack2","AbbreviatedName":"Loop2"},{"IfIndex":1537,"Name":"LoopBack0","AbbreviatedName"
:"Loop0"},{"IfIndex":1601,"Name":"LoopBack1","AbbreviatedName":"Loop1"}]}
```

3. 基于列的过滤

基于列的过滤包括严格匹配过滤、正则表达式匹配过滤和条件匹配过滤。

同时使用多种过滤方式时，只有一个过滤方式生效。过滤方式的优先级从高到低依次为：严格匹配过滤、正则表达式匹配过滤和条件匹配过滤。

基于列的过滤操作通过在 URL 中指定 filter-condition 参数实现。

- 列的严格匹配过滤

  列的严格匹配过滤通过在 URL 中指定“filter-condition=列名=值”来实现。如需同时使用多个列进行过滤，则在 URL 中指定 filter-condition 参数时，不同列之间需要通过分号“;”分隔，例如“filter-condition=列名 1=值 1; 列名 2=值 2”。

  以获取接口名为 M-GigabitEthernet0/0/0 的接口的信息为例（filter-condition=Name=M-GigabitEthernet0/0/0），客户端发送的报文格式如下。

  ```
  GET
  http://192.168.56.120/api/v1/Ifmgr/Interfaces?filter-condition=Name%3DM-GigabitEthe
  rnet0%2F0%2F0 HTTP/1.1
  Accept-Encoding: gzip,deflate
  X-Auth-Token: 400001d534ee6ba9b7d4e26e65e0bfa1df9f
  Host: 192.168.56.120
  Connection: Keep-Alive
  User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
  ```

  设备收到获取接口信息的请求报文后，将接口名为 M-GigabitEthernet0/0/0 的接口的信息通过如下报文反馈给客户端。

  ```
  HTTP/1.1 200 OK
  Content-Type: application/json charset=UTF-8
  Server: HTTPD
  Date: Mon, 31 May 2021 15:47:47 GMT
  Connection: Keep-Alive
  Transfer-Encoding: chunked
  ```

```
X-XSS-Protection: 1;mode-block
X-Content-Type-Options: nosniff

{"Interfaces":[{"IfIndex":1281,"Name":"M-GigabitEthernet0\/0\/0"}]}
```

提示

由于 URL 中没有指定 select-fields 参数，因此，返回的接口信息中只包括索引列和 filter-condition 指定的列（即接口名 Name）。

- 列的正则表达式匹配过滤

  列的正则表达式匹配过滤通过在 URL 中指定“filter-condition=regExp:列名=正则表达式”来实现。如果需要同时指定多个列的正则表达式，则不同正则表达式之间需要通过分号“;”分隔，如“filter-condition=regExp:列名 1=正则表达式 1; regExp:列名 2=正则表达式 2;”。

  列的正则表达式过滤匹配如下例所示：

  以获取接口描述（Description）中仅包括数字和字母的接口的信息为例，客户端发送的报文格式如下。

```
GET
http://192.168.56.120/api/v1/Ifmgr/Interfaces?filter-condition=regExp%3ADescription
%3D%5E%5Ba-zA-Z0-9%5D*%24 HTTP/1.1
Accept-Encoding: gzip,deflate
X-Auth-Token: 400001d534ee6ba9b7d4e26e65e0bfa1df9f
Host: 192.168.56.120
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
```

  设备收到获取接口信息的请求报文后，将匹配正则表达式的接口信息通过如下报文返回给客户端。

```
HTTP/1.1 200 OK
Content-Type: application/json charset=UTF-8
Server: HTTPD
Date: Mon, 31 May 2021 15:51:43 GMT
Connection: Keep-Alive
Transfer-Encoding: chunked
X-XSS-Protection: 1;mode-block
X-Content-Type-Options: nosniff

{"Interfaces":[{"IfIndex":1281,"Description":"manage"},{"IfIndex":1409,"Description
":"N0U0L0L"}]}
```

- 列的条件匹配过滤

  由于正则表达仅能够完成字符匹配，对于数值逻辑的判断和过滤实现起来比较麻烦。此时，可使用条件匹配过滤功能来根据数值对返回数据进行过滤。

  列的条件匹配过滤通过在 URL 中指定“filter-condition=条件命令:列名=条件值”来实现。如果需要同时指定多个列的条件匹配规则，则不同条件匹配规则之间需要通过分号“;”分隔，如“filter-condition=条件命令 1:列名 1=条件值 1;条件命令 2:列名 2=条件值 2;”。

  RESTful 操作支持的条件命令如表 5 所示。

表5 RESTful 操作支持的条件命令

操作	命令	说明
大于	**filter -condition=more**:*column=value*	列的取值大于*value*，支持的数据类型为：日期、数字、字符串
小于	**filter -condition=less**:*column=value*	列的取值小于*value*，支持的数据类型为：日期、数字、字符串
不小于	**filter -condition=notLess**:*column=value*	列的取值不小于*value*，支持的数据类型为：日期、数字、字符串
不大于	**filter -condition=notMore**:*column=value*	列的取值不大于*value*，支持的数据类型为：日期、数字、字符串
等于	**filter -condition=equal**:*column=value*	列的取值等于*value*，支持的数据类型为：日期、数字、字符串、OID、BOOL
不等于	**filter -condition=notEqual**:*column=value*	列的取值不等于*value*，支持的数据类型为：日期、数字、字符串、OID、BOOL
包含	**filter -condition=include**:*column=value*	列的取值中包含字符串*string*，支持的数据类型为：字符串
不包含	**filter -condition=exclude**:*column=value*	列的取值中不能包含字符串*string*，支持的数据类型为：字符串
开始于	**filter -condition=startWith**:*column=value*	列的取值以字符串*string*开头，支持的数据类型为：字符串、OID
结束于	**filter -condition=endWith**:*column=value*	列的取值以字符串*string*结束，支持的数据类型为：字符串

以获取接口索引值大于 1281 的所有接口的接口名信息为例（filter-condition=more:IfIndex=1281、select-fields=Name），客户端发送的报文格式如下。

```
GET
http://192.168.56.120/api/v1/Ifmgr/Interfaces?select-fields=Name&filter-condition=more%3AIfIndex%3D1281%3B HTTP/1.1
Accept-Encoding: gzip,deflate
X-Auth-Token: 400001d534ee6ba9b7d4e26e65e0bfa1df9f
Host: 192.168.56.120
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
```

设备收到获取接口信息的请求报文后，将接口索引值大于 1281 的所有接口的接口名信息通过如下报文反馈给客户端。

```
HTTP/1.1 200 OK
Content-Type: application/json charset=UTF-8
Server: HTTPD
Date: Mon, 31 May 2021 15:39:57 GMT
Connection: Keep-Alive
Transfer-Encoding: chunked
X-XSS-Protection: 1;mode-block
X-Content-Type-Options: nosniff
```

```
{"Interfaces":[{"IfIndex":1409,"Name":"NULL0"},{"IfIndex":1410,"Name":"InLoopBack0"},{"IfIndex":1536,"Name":"LoopBack2"},{"IfIndex":1537,"Name":"LoopBack0"},{"IfIndex":1601,"Name":"LoopBack1"}]}
```

4. 多个参数综合使用示例

RESTful GET 操作中可以同时指定 index、select-fields 和 filter-condition 参数，以实现进行多重条件过滤。

通过客户端发送如下报文，可以获取符合如下条件的接口信息：

- 接口索引值为 1281。
- 接口描述信息中仅包括数字和字母。
- 返回接口的名称列。

```
GET http://192.168.56.120/api/v1/Ifmgr/Interfaces?index=IfIndex%3D%201281&select-fields=Name%3B&filter-condition=regExp%3ADescription%3D%5E%5Ba-zA-Z0-9%5D*%24 HTTP/1.1
Accept-Encoding: gzip,deflate
X-Auth-Token: 400001f430ff4fda0c5485a17934edf035e8
Host: 192.168.56.120
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
```

设备收到获取接口信息的请求报文后，将满足条件的接口信息通过如下报文反馈给客户端。

```
HTTP/1.1 200 OK
Content-Type: application/json charset=UTF-8
Server: HTTPD
Date: Mon, 31 May 2021 15:27:05 GMT
Connection: Keep-Alive
Transfer-Encoding: chunked
X-XSS-Protection: 1;mode-block
X-Content-Type-Options: nosniff

{"IfIndex":1281,"Name":"M-GigabitEthernet0\/0\/0","Description":"manage"}
```

5. 获取 N 行数据

RESTful GET 操作中可以指定 getbulk-count 参数来获取 N 行数据。

例如，通过客户端发送如下请求报文，可以获取 3 个 VLAN 的信息。

```
GET http://192.168.56.120/api/v1/VLAN/VLANs?getbulk-count=3 HTTP/1.1
Accept-Encoding: gzip,deflate
X-Auth-Token: 400001d534ee6ba9b7d4e26e65e0bfa1df9f
Host: 192.168.56.120
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
```

设备收到获取 VLAN 信息的请求报文后，将 VLAN 1、VLAN 2 和 VLAN 3 这三个 VLAN 的信息通过如下报文反馈给客户端。

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Server: HTTPD
Date: Wed, 11 May 2022 06:56:22 GMT
```

```
Connection: Keep-Alive
Transfer-Encoding: chunked
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1;mode-block
X-Content-Type-Options: nosniff

{"VLANs":[{"ID":1,"Description":"VLAN 0001","Name":"VLAN
0001","Shared":false},{"ID":2,"Description":"VLAN 0002","Name":"VLAN
0002","AccessPortList":"","Shared":false},{"ID":3,"Description":"VLAN 0003","Name":"VLAN
0003","AccessPortList":"","Shared":false}]}
```

4.4 POST操作

4.4.1 POST 操作介绍

POST 操作用来创建指定的资源，相当于 NETCONF edit-config 操作中的 create。POST 请求的内容需要放在 HTTP Body 中，而且 URL 中不能携带 index 参数。如果执行成功，会返回 201。

POST 操作也可用来下发 action 请求，相当于 NETCONF 中的 action 操作。RESTful 通过以“/api/v1/action/”开头的 URL 下发 Action 请求。

POST 操作支持批量下发多行数据。


提示

全局资源不支持通过 POST 操作来创建。

4.4.2 POST 操作举例

1. 添加信息中心的日志主机

客户端发送如下报文，可以实现在设备上添加信息中心的日志主机。添加的日志主机 IP 地址为 1.1.1.3、所属 VPN 实例为 b、接收日志信息的端口号为 518、记录工具为 local7。

```
POST http://192.168.100.91/api/v1/Syslog/LogHosts HTTP/1.1
Accept-Encoding: gzip,deflate
X-Auth-Token: 400001b81dbfc50884818f45b686fd81ceaa
Content-Type: application/json
Content-Length: 78
Host: 192.168.100.91
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

{
   "Address": "1.1.1.3",
   "VRF": "b",
   "Port": 518,
   "Facility": 184
}
```

设备收到请求报文后，会通过如下报文通知客户端日志主机创建成功。

```
HTTP/1.1 201 Created
Content-Type: application/json charset=UTF-8
Location: http://192.168.100.91/api/v1/Syslog/LogHosts?index=Address%3D1.1.1.3%3BVRF%3Db
Server: HTTPD
Date: Fri, 12 Jun 2015 14:46:33 GMT
Connection: Keep-Alive
Transfer-Encoding: chunked
```

2. 创建 VLAN

客户端发送如下报文，可以实现在设备上创建 VLAN 2。

```
POST http://192.168.100.91/api/v1/VLAN/VLANs HTTP/1.1
Accept-Encoding: gzip,deflate
X-Auth-Token: 400001b81dbfc50884818f45b686fd81ceaa
Content-Type: application/json
Content-Length: 44
Host: 192.168.100.91
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

{
  "ID": "2",
  "Description": "VLAN 2"
}
```

设备收到请求报文后，会通过如下报文通知客户端 VLAN 创建成功。

```
HTTP/1.1 201 Created
Content-Type: application/json charset=UTF-8
Location: http://192.168.100.91/api/v1/VLAN/VLANs?index=ID%3D2
Server: HTTPD
Date: Fri, 12 Jun 2015 14:49:21 GMT
Connection: Keep-Alive
Transfer-Encoding: chunked
```

3. 通过 action 请求清除接口数据

客户端发送如下报文，可以实现清除接口索引值为 1281 的接口的数据。

```
POST http://192.168.79.167/api/v1/action/Ifmgr/Interfaces HTTP/1.1
Accept-Encoding: gzip,deflate
X-Auth-Token: 400001fb4b7abc838fe880ad66f6b06c8bf3
Content-Type: application/json
Content-Length: 36
Host: 192.168.79.167
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

{
  "IfIndex":"1281",
  "Clear": ""
}
```

说明

在下发的 HTTP Body 中如果存在不需要携带取值的列，则该列可以通过"ColumnName": "" 形式下发，如上述举例中的"Clear": ""。

设备收到请求报文后，会通过如下报文通知客户端接口数据清除成功。

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Location: http://192.168.79.167/api/v1/Ifmgr/Interfaces?index=IfIndex%3D1281
Server: HTTPD
Date: Fri, 17 May 2019 06:29:31 GMT
Connection: Keep-Alive
Transfer-Encoding: chunked
X-Frame-Options: SAMEORIGIN
```

4. 通过 action 请求创建 VLAN 列表

客户端发送如下报文，可以实现在设备上创建 VLAN 列表 34。

```
POST http://192.168.79.167/api/v1/action/VLAN/BatchVlans HTTP/1.1
Accept-Encoding: gzip,deflate
X-Auth-Token: 400001ffb29850dbb0763f29851bb4efa97a
Content-Type: application/json
Content-Length: 54
Host: 192.168.79.167
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

{
  "CreateVlanList": "34"
}
```

设备收到请求报文后，会通过如下报文通知客户端创建 VLAN 列表成功。

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Server: HTTPD
Date: Fri, 17 May 2019 06:51:20 GMT
Connection: Keep-Alive
Transfer-Encoding: chunked
X-Frame-Options: SAMEORIGIN

{ "CreateVlanList":"34" }
```

5. 批量添加信息中心的日志主机

客户端发送如下报文，可以实现在设备上添加两个日志主机：

- 日志主机 1：IP 地址为 1.1.1.5、所属 VPN 实例为 aaa、接收日志信息的端口号为 515、记录工具为 local7。
- 日志主机 2：IP 地址为 1.1.1.6、所属 VPN 实例为 aaa、接收日志信息的端口号为 515、记录工具为 local7。

```
POST http://192.168.56.101/api/v1/Syslog/LogHosts HTTP/1.1
Accept-Encoding: gzip,deflate
X-Auth-Token: 4000019cb383a71c5781ce33cc2feaf8e4d1
Content-Type: application/json;charset=UTF-8
Content-Length: 223
Host: 192.168.56.101
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

{"LogHosts": [
      {
      "Address": "1.1.1.5",
      "VRF": "aaa",
      "Port": 515,
      "Facility": 184
   },
      {
      "Address": "1.1.1.6",
      "VRF": "aaa",
      "Port": 515,
      "Facility": 184
   }
]}
```

设备收到请求报文后，会通过如下报文通知客户端两个日志主机均创建成功。

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Server: HTTPD
Date: Thu, 09 Dec 2021 05:26:18 GMT
Connection: Keep-Alive
Transfer-Encoding: chunked
X-Frame-Options: SAMEORIGIN

{"LogHosts":[{"Address":"1.1.1.5","VRF":"aaa","Port":515,"Facility":184,"Format":0},{"Ad
dress":"1.1.1.6","VRF":"aaa","Port":515,"Facility":184,"Format":0}]}
```

4.5 PUT操作

4.5.1 PUT 操作介绍

PUT 操作用来修改现有资源，相当于 NETCONF edit-config 操作中的 merge。如果指定的资源已经存在，则会修改该资源；如果指定的资源不存在，则会创建该资源。PUT 操作执行成功后，设备会返回 204。

对于 PUT 操作，需要注意的是：

- 索引列必须在 URL 的 index 参数中给出，而且必须和 HTTP Body 中给出的索引列的值一致。
- 参数 index 的格式与 GET 操作相同，为 **index**=Indexname1=value1;indexname2=value2。参数的内容需要用类似 encodeURI 的方法进行编码。

4.5.2 PUT 操作举例

1. 创建或修改信息中心的日志主机

客户端发送如下报文，可以实现在设备上创建日志主机，或修改已有日志主机的信息。

```
PUT http://192.168.1.1:80/api/v1/Syslog/LogHosts
X-Auth-Token: 1000028562945fa49b19199a71a01f33d609

{
“Loghosts”:[
   {“Address”:”1.1.1.1”, “VFR”:””, “Port “:514, “Facility”: 192 },
]}
```

设备收到请求报文后，会通过如下报文通知客户端日志主机或修改成功。本例中，设备上不存在请求对应的日志主机，在设备上创建了该日志主机。

```
204 No Content
```

2. 修改接口的描述信息

客户端发送如下报文，可以修改设备上接口索引值为 3 的接口的描述信息。

```
PUT http://192.168.100.91/api/v1/Ifmgr/Interfaces?index=IfIndex%3D3 HTTP/1.1
Accept-Encoding: gzip,deflate
X-Auth-Token: 400001b81dbfc50884818f45b686fd81ceaa
Content-Type: application/json
Content-Length: 51
Host: 192.168.100.91
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

{
   "IfIndex":3,
   "Description": "Interface 3"
}
```

提示

URL 中的 index 参数与 HTTP Body 中索引列的取值必须相同，本例中取值均为 3。

设备收到请求报文后，会通过如下报文通知客户端接口描述信息修改成功。

```
HTTP/1.1 204 No Content
Content-Type: application/json charset=UTF-8
Server: HTTPD
Date: Fri, 12 Jun 2015 14:26:10 GMT
Connection: Keep-Alive
Transfer-Encoding: chunked
```

4.6 DELETE操作

4.6.1 DELETE 操作介绍

DELETE 操作用来删除指定的资源，相当于 NETCONF edit-config 操作中的 delete。此操作不能携带 HTTP Body。

不允许对不可创建资源执行 DELETE 操作。对不可创建资源执行 DELETE 操作时，会返回 405 Method Not Allowed。

对可创建资源执行 DELETE 操作时，必须在 URL 的 index 参数中给出待删除资源的索引。收到 DELETE 操作请求后，设备会删除指定索引对应资源的全部属性。当资源不存在，或者资源不允许删除时，会返回 409 Conflict。

在 DELETE 操作中，参数 index 的格式与 GET 操作中参数 index 的格式相同，均为 **index**=Indexname1=value1;indexname2=value2。参数的内容需要用类似 encodeURI 的方法进行编码。

4.6.2 DELETE 操作举例

以资源 Syslog/LogHosts 为例，如果要删除 VPN 实例内 IP 地址为 1.1.1.1 的日志主机，客户端需要发送如下请求报文。

```
DELETE http://192.168.100.91/api/v1/Syslog/LogHosts?index=Address%3D1.1.1.1%3BVRF%3Da
HTTP/1.1
Accept-Encoding: gzip,deflate
X-Auth-Token: 400001b81dbfc50884818f45b686fd81ceaa
Host: 192.168.100.91
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
```

设备收到请求报文后，会通过如下报文通知客户端，日志主机删除成功。

```
HTTP/1.1 204 No Content
Content-Type: application/json charset=UTF-8
Server: HTTPD
Date: Fri, 12 Jun 2015 14:07:50 GMT
Connection: Keep-Alive
Transfer-Encoding: chunked
```

5 Postman 工具操作示例

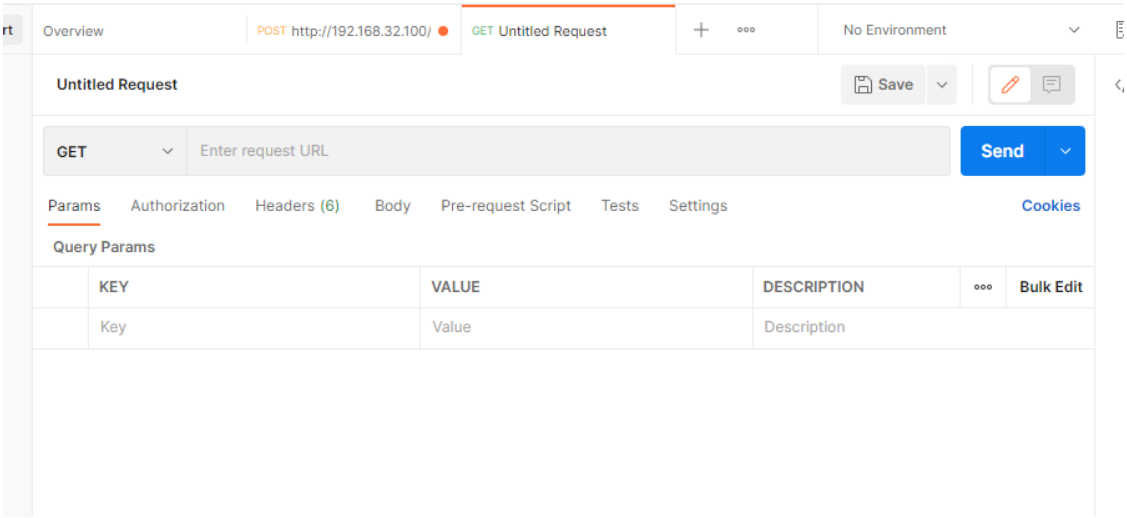
Postman 是一款常用的接口测试工具。用户可以使用 Postman 作为 RESTful 客户端，对设备进行配置和维护。本节以一个实际例子，来介绍 Postman 作为 RESTful 客户端的操作方法。

5.1 登录设备

使用 Postman 作为 RESTful 客户端登录设备（IP 地址为 192.168.32.100）的过程为：

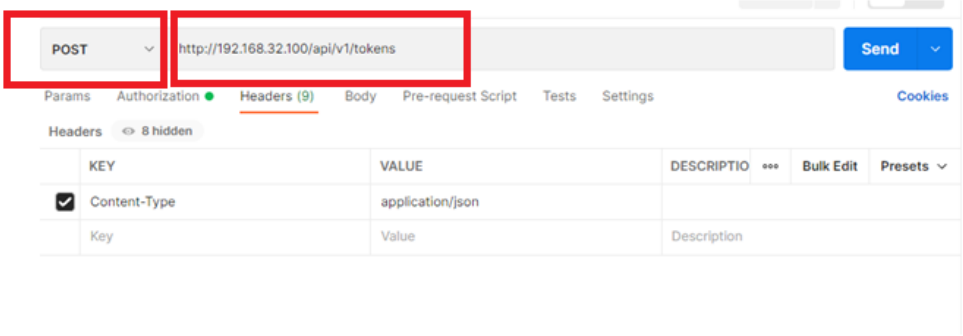
(1) 依次点击<File-New-HTTP Request>按钮，新建 RESTful 请求。

图5 新建 Restful 请求



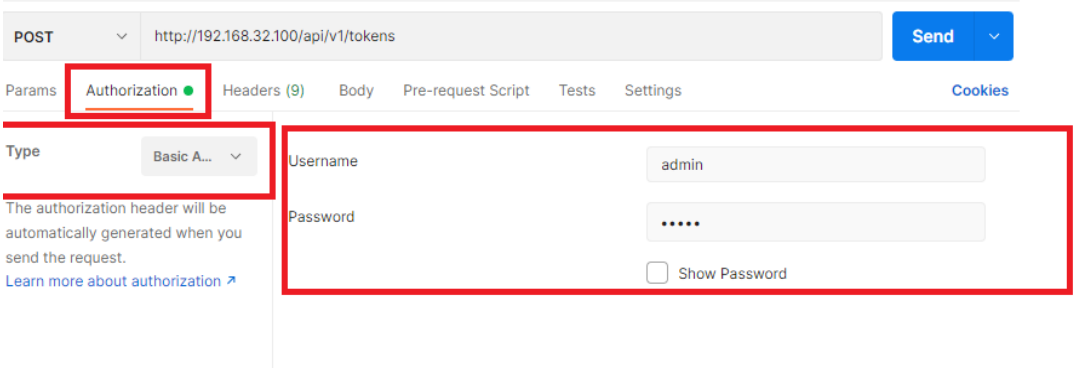
(2) 设置请求的操作类型为 POST，并输入 URL 地址 http://192.168.32.100/api/v1/tokens。

图6 设置 RESTful 请求的操作类型和 URL 地址



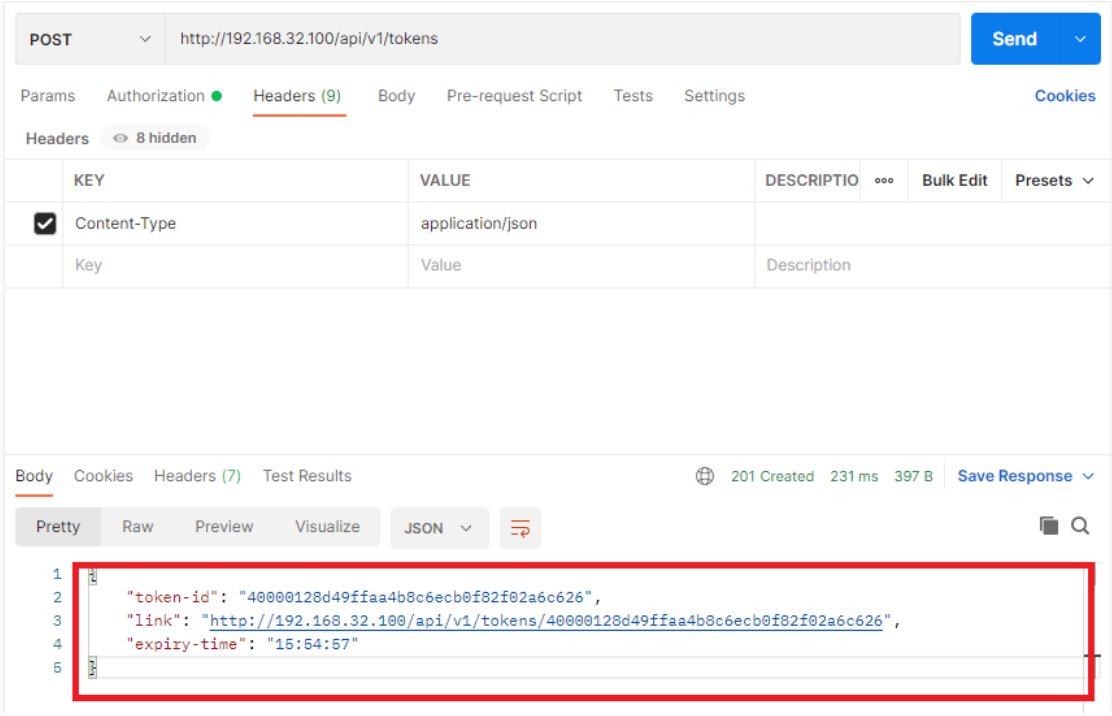
(3) 点击<Authorization>页签，选择 Type 为 Basic Auth，并在 Username 和 Password 处填写 RESTful 用户的用户名和密码。

图7 设置 RESTful 请求的认证功能



(4) 点击<Send>按钮，发送 RESTful 请求。返回如图 8 所示的数据后，说明 Restful 客户端成功登录设备。

图8 发送 Restful 请求登录设备

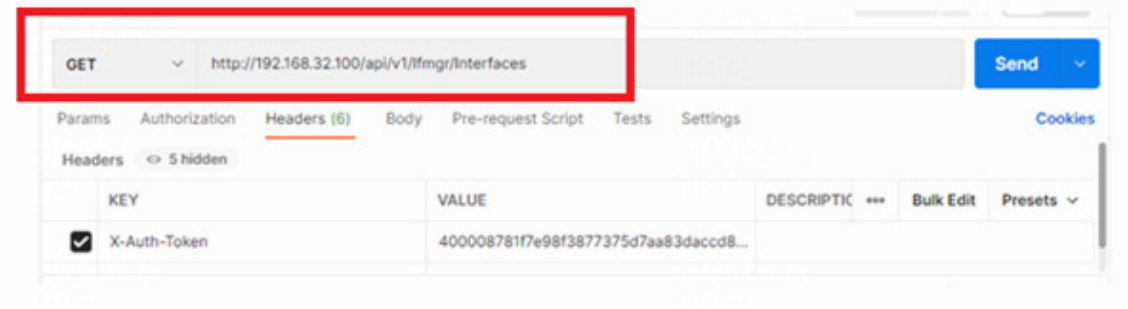


5.2 GET操作

通过 Postman 工具执行 GET 操作的过程为：

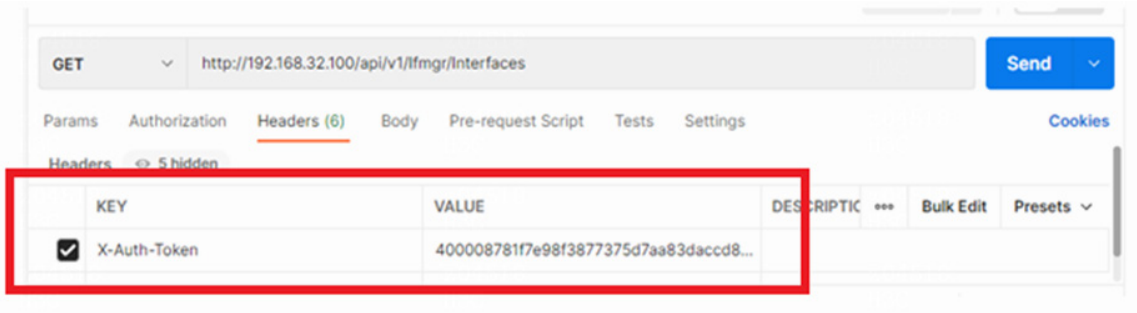
(1) 设置 RESTful 请求的操作类型为 GET，并输入 URL。本例为获取接口数据，URL 地址为 http://192.168.32.100/api/v1/lfmgr/Interfaces。

图9 构造 GET 操作的 RESTful 请求



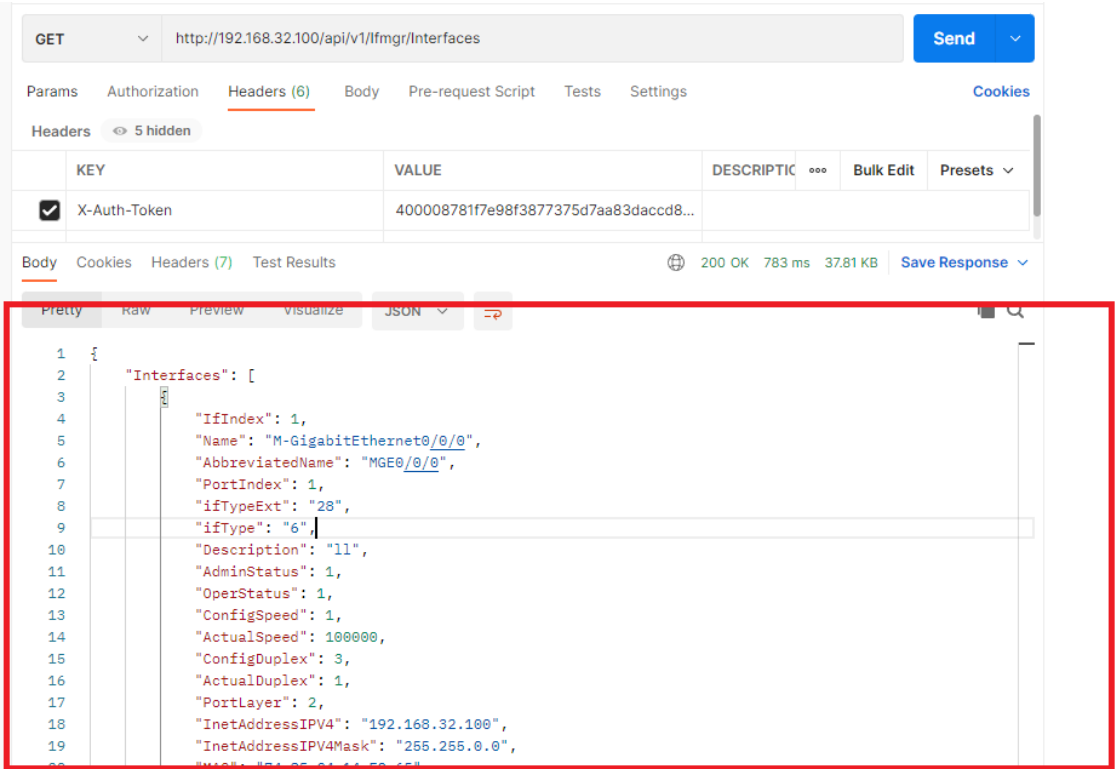
(2) 在 Headers 中增加 X-Auth-Token 和对应的 value，该 value 为登录设备后，返回的 token-id 值。

图10 设置 X-Auth-Token



(3) 点击<Send>按钮，发送 RESTful 请求，获取接口相关数据。

图11 获取接口数据

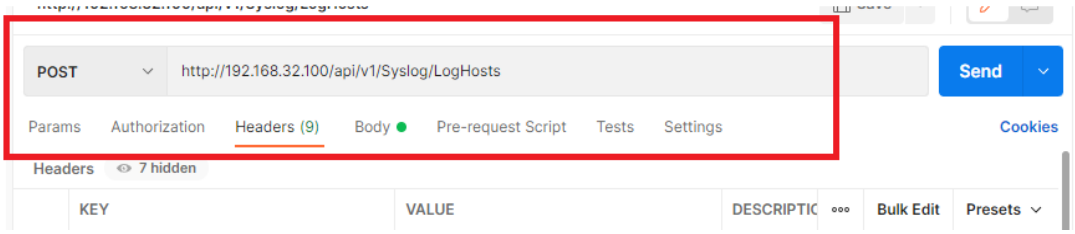


5.3 POST操作

通过 Postman 工具执行 POST 操作的过程为：

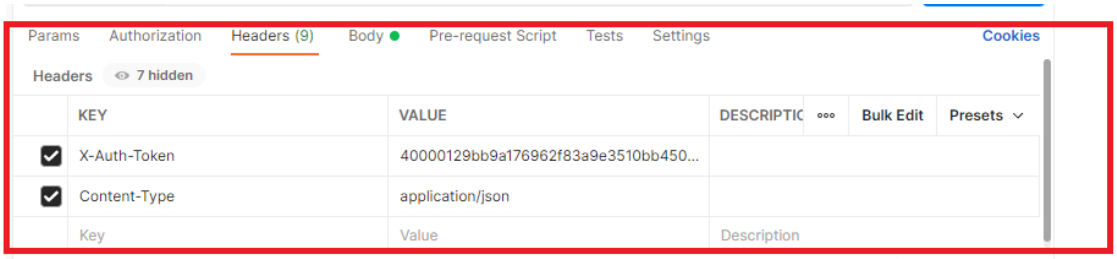
(1) 设置 RESTful 请求的操作类型为 POST，并输入 URL。本例为在设备上添加信息中心的日志主机，URL 地址为 http://192.168.32.100/api/v1/Syslog/LogHosts。

图12 构造 POST 操作的 RESTful 请求



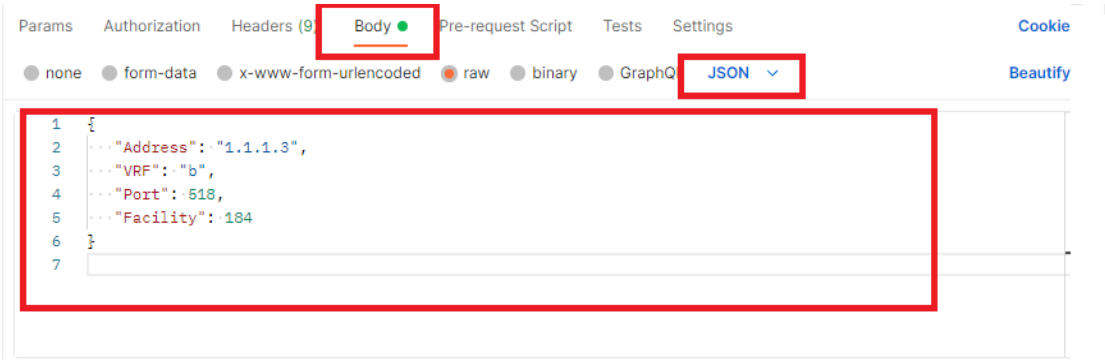
(2) 在 Headers 中增加 X-Auth-Token 和对应的 value，该 value 为登录设备后，返回的 token-id 值；增加 Content-Type，并选择格式为 application/json。

图13 设置 RESTful 请求的 Headers



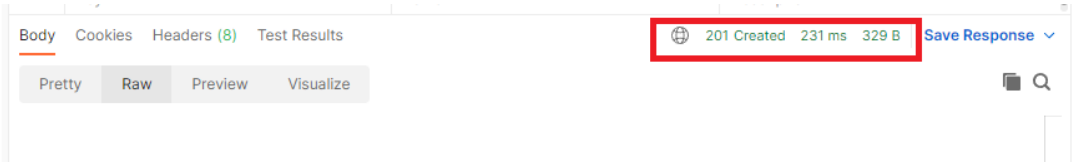
(3) 在 Body 中，设置需要创建日志主机的参数。

图14 在 RESTful 请求的 Body 中设置日志主机参数



(4) 点击<Send>，发送 RESTful 请求，创建日志主机。

图15 创建日志主机

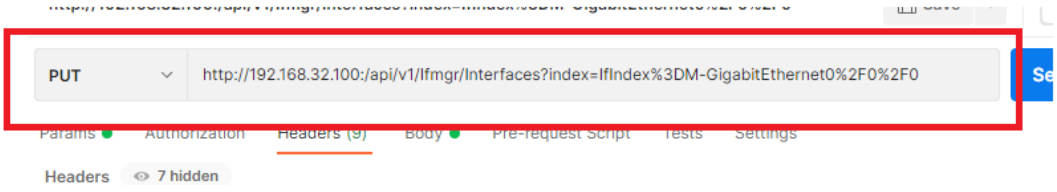


5.4 PUT操作

通过 Postman 工具执行 PUT 操作的过程为：

(1) 设置 RESTful 请求的操作类型为 PUT，并输入 URL。本例为修改接口描述数据，URL 地址为 http://192.168.32.100/api/v1/Ifmgr/Interfaces?index=IfIndex%3DM-GigabitEthernet0%2F0%2F0。

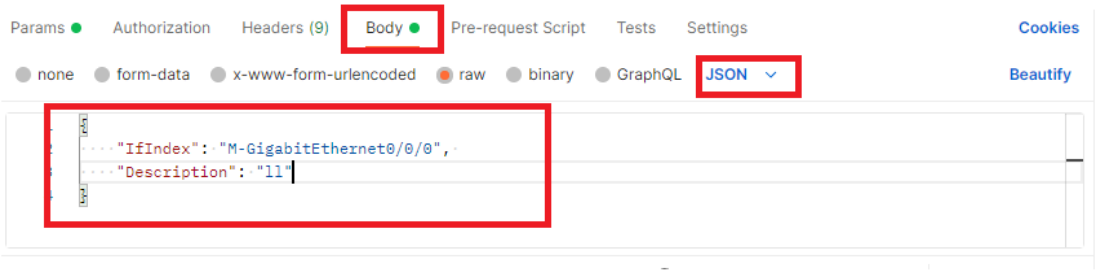
图16 构造 PUT 操作的 RESTful 请求



(2) 在 Headers 中增加 X-Auth-Token 和对应的 value，该 value 为登录设备后，返回的 token-id 值；增加 Content-Type，并选择格式为 application/json。

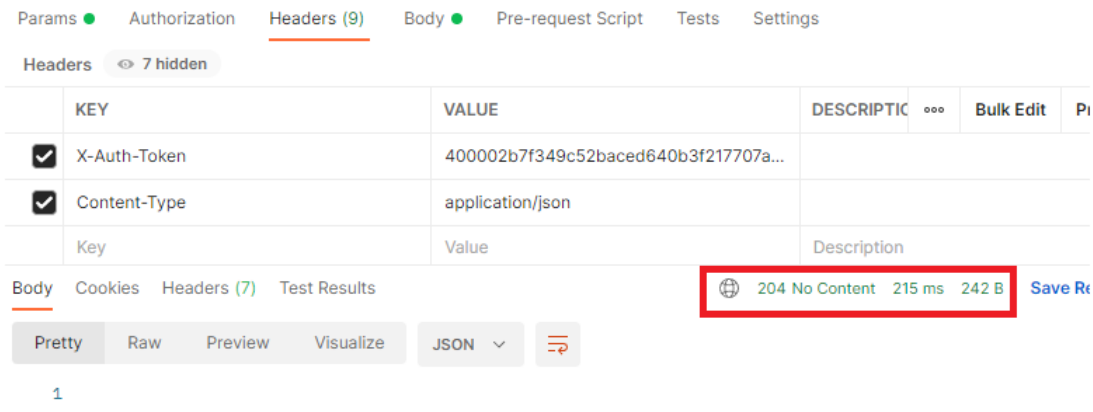
(3) 在 Body 中，设置修改接口描述的参数。

图17 在 RESTful 请求的 Body 中设置修改接口描述的参数



(4) 点击<Send>，发送 RESTful 请求，修改接口描述。

图18 修改接口描述

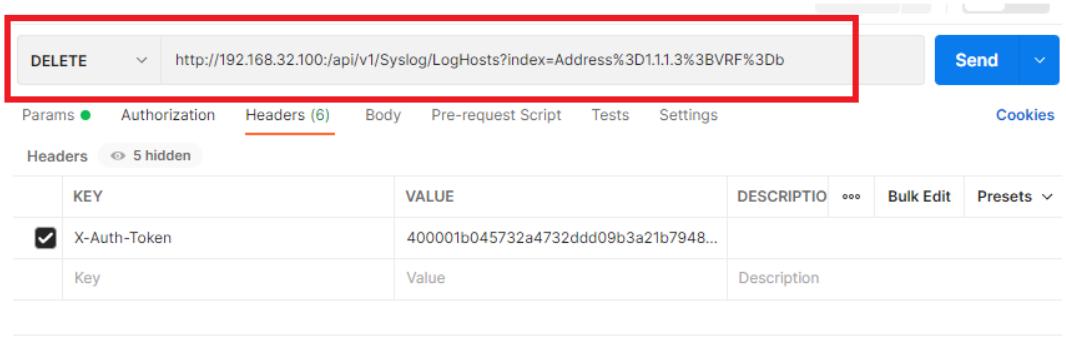


5.5 DELETE操作

通过 Postman 工具执行 DELETE 操作的过程为：

(1) 设置 RESTful 请求的操作类型为 DELETE 方式，并输入 URL。本例为删除之前创建的日志主机，URL 地址为 http://192.168.32.100/api/v1/Syslog/LogHosts?index=Address%3D1.1.1.3%3BVRF%3Db。

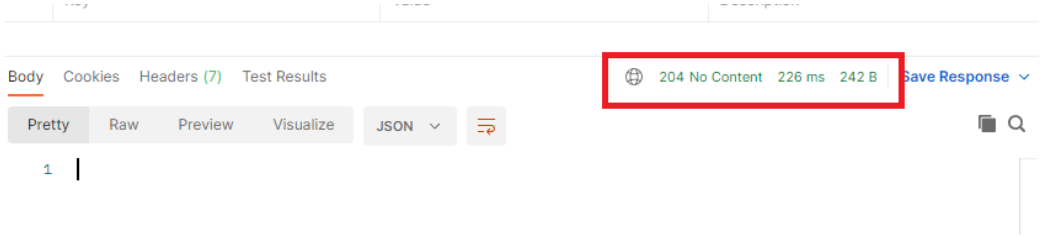
图19 构造 DELETE 操作的 RESTful 请求



(2) 在 Headers 中增加 X-Auth-Token 和对应的 value，该 value 为登录设备后，返回的 token-id 值。

(3) 点击<Send>，发送 RESTful 请求，删除日志主机。

图20 修删除日志主机



6 Python 语言开发客户端示例

6.1 开发准备

Python 是目前比较流行的脚本语言。本节将介绍以该语言为基础的 RESTful 客户端程序的开发方法。本节展示的代码，都是较为简单的示例，可以完成 RESTful 相关的基本操作。

使用 Python 语言开发 RESTful 客户端程序前，要求用户熟悉如下知识：

- RESTful 相关知识。
- HTTP 相关知识。
- Python 开发语言。

6.2 登录设备示例

采用 Python 提供的 requests 模块可以实现 RESTful 客户端登录设备。如下为登录设备的 Python 代码示例：

```
def restful_login():
    """
    登录设备的 URL，用于获取 tokenid
    """
    url = 'http://192.168.32.100/api/v1/tokens'
    """
    HTTP 的 headers 填充，Authorization 为标准的 HTTP 认证 basic 方式认证
    """
    headers = {"Authorization":"Basic
YWRtaW46YWRtaW4=","Content-type":"application/json","Accept": "application/json"}
    """
    登录设备
    """
    req = requests.post(url, headers=headers)
    """
    返回结果以 json 格式输出
    """
    return req.json()
```

通过上述代码向设备发送 RESTful 请求后，RESTful 客户端会接收到设备返回的 token-id 值。后续，RESTful 客户端发送的请求中需要携带该 token-id 值，并将其放在 X-Auth-Token 中。

6.3 GET操作示例

通过 GET 操作获取接口数据的 Python 代码示例如下：

```
def restful_get():
    """
    URL 用于获取接口数据
    """
    url = 'http://192.168.32.100:/api/v1/Ifmgr/Interfaces'
    tokenJson = restful_login()
    """
    获取登录时返回数据中的 token-id 值
    """
    for key,value in tokenJson.items():
        if key == 'token-id':
           token = value
   """
    HTTP 的 headers 填充，X-Auth-Token 取值为登录时返回数据中的 token-id 值
    """
    headers =  {"X-Auth-Token":token,"Content-type":"application/json","Accept":
"application/json"}
    """
    获取接口数据
    """
    req = requests.get(url, headers=headers)
    print(req.json())
```

6.4 POST操作示例

通过 POST 操作创建日志主机的 Python 代码示例如下：

```
def restful_post():
    """
    URL 用于创建日志主机
    """
    url = 'http://192.168.32.100:/api/v1/Syslog/LogHosts '
    """
    Body 中填充的 JSON 数据为创建的日志主机的参数
    """
    values = {"Address": "1.1.1.3","VRF": "b","Port": 518,"Facility": 184}
    """
    使用 JSON 模块转换成标准 JSON 字符串，并获取登录时返回数据中的 token-id 值
    """
    param = json.dumps(values)
    tokenJson = restful_login()
    for key,value in tokenJson.items():
        if key == 'token-id':
```

```
            token = value
    """
    HTTP 的 headers 填充，X-Auth-Token 取值为登录时返回数据中的 token-id 值
    """
    headers =  {"X-Auth-Token":token,"Content-type":"application/json","Accept":
"application/json"}
    """
    创建日志主机
    """
    req = requests.post(url, data=param, headers=headers)
    print(req.status_code)
```

6.5 PUT操作示例

通过 PUT 操作修改接口描述信息的 Python 代码示例如下：

```
def restful_put():
    """
    URL 用于修改接口的描述信息，注意 PUT 操作的 URL 中需要添加 index 信息
    """
    url =
'http://192.168.32.100:/api/v1/Ifmgr/Interfaces?index=IfIndex%3DM-GigabitEthernet0%2F0%2
F0'
    """
    Body 中填充的 JSON 数据为修改的接口描述信息
    """
    values = {"IfIndex": "M-GigabitEthernet0/0/0", "Description": "ll"}
    """
    使用 JSON 模块转换成标准 JSON 字符串，并获取登录时返回数据中的 token-id 值
    """
    param = json.dumps(values)
    tokenJson = restful_login()
    for key,value in tokenJson.items():
        if key == 'token-id':
            token = value
    """
    HTTP 的 headers 填充，X-Auth-Token 取值为登录时返回数据中的 token-id 值
    """
    headers =  {"X-Auth-Token":token,"Content-type":"application/json","Accept":
"application/json"}
    """
    修改接口描述信息
    """
    req = requests.put(url, data=param, headers=headers)
    print(req.status_code)
```

6.6 DELETE操作示例

通过 PUT 操作删除日志主机的 Python 代码示例如下：

```
def restful_delete():
```

```
    """
    URL 用于删除日志主机
    """
    url =
'http://192.168.32.100:/api/v1/Syslog/LogHosts?index=Address%3D1.1.1.1%3BVRF%3Da '
    tokenJson = restful_login()
    """
    获取登录时返回数据中的 token-id 值
    """
    for key,value in tokenJson.items():
        if key == 'token-id':
           token = value
    """
    HTTP 的 headers 填充，X-Auth-Token 取值为登录时返回数据中的 token-id 值
    """
    headers =  {"X-Auth-Token":token,"Content-type":"application/json","Accept":
"application/json"}
    """
    删除日志主机
    """
    req = requests.delete(url, headers=headers)
    print(req.status_code)
```