

Postman 简介

一般简单的接口测试我们可以直接在浏览器里面进行调试，但是涉及到一些权限设置的就无法操作了，因此我们需要接口测试的相关工具；Postman 是一个接口测试和 http 请求的工具。

官网地址：<https://www.getpostman.com>

Postman 的优点：

- 支持各种的请求类型: get、post、put、patch、delete 等
- 支持在线存储数据，通过账号就可以进行迁移数据
- 很方便的支持请求 header 和请求参数的设置
- 支持不同的认证机制，包括 Basic Auth, Digest Auth, OAuth 1.0, OAuth 2.0 等
- 响应数据是自动按照语法格式高亮的，包括 HTML, JSON 和 XML

下载安装

Postman 有 windows, Mac、Linux 以及 Chrome 插件版本。这里主要介绍 Win 平台版本的使用。

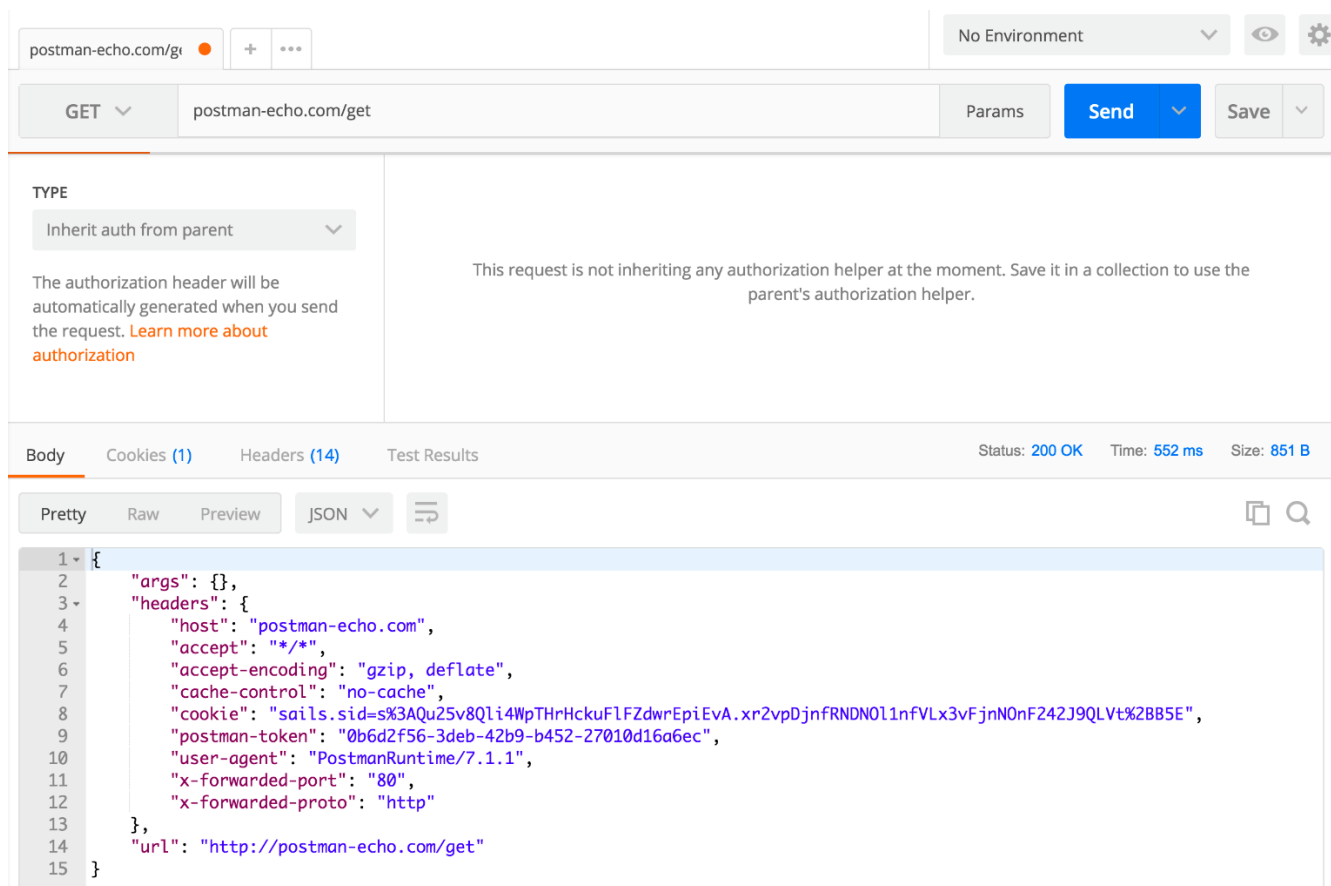
- 下载地址：<https://www.getpostman.com/apps>
- 官方文档：<https://www.getpostman.com/docs/v6/>
- Postman Api 文档：<https://docs.postman-echo.com>

Postman 入门

安装好之后启动程序，进入主界面。准备开始使用 Postman。

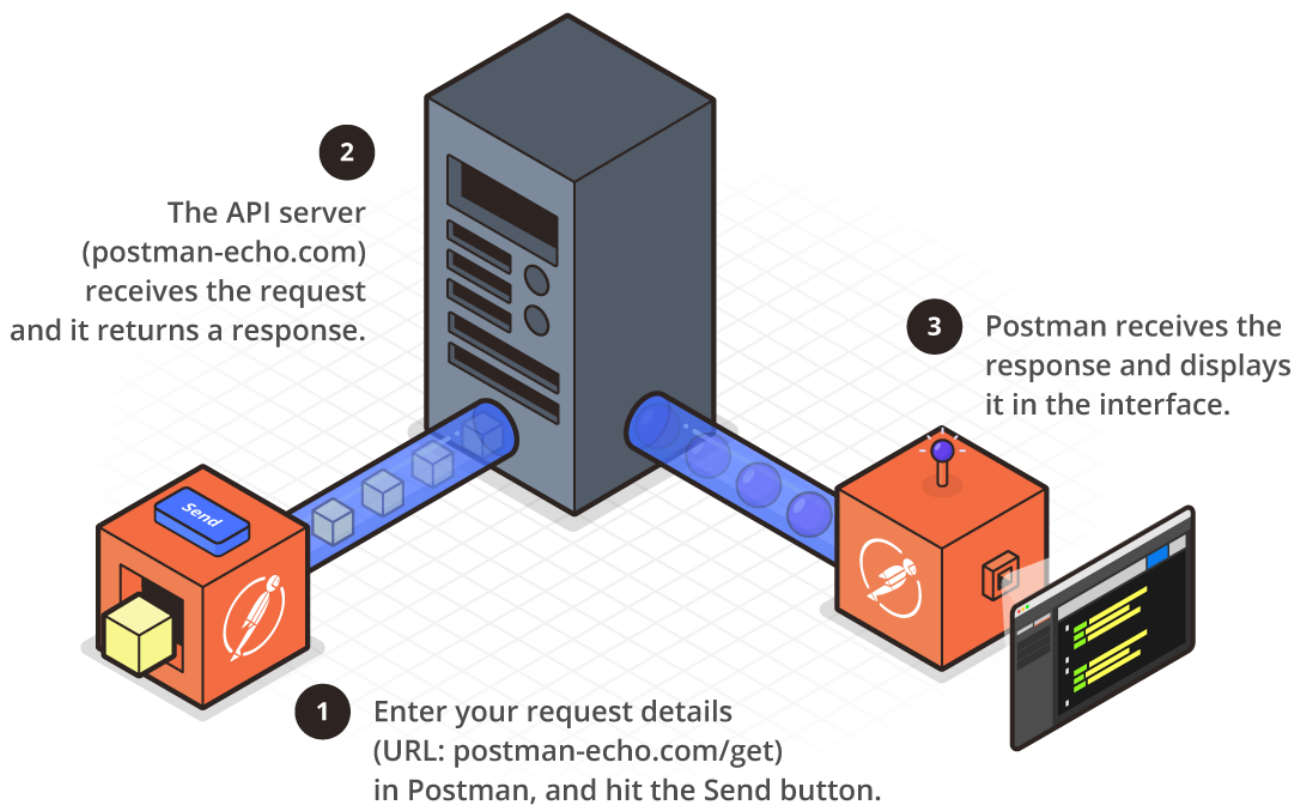
发送第一个请求

1. 启动软件后在引导界面点击 Request, 给 Request 命名, 然后创建文件夹并把该 Request 归属到该文件夹。
2. 在地址栏输入 postman-echo.com/get 然后点击 Send 按钮，可以看到返回值。如下图所示：



Postman 工作原理

如下图所示，当您在 Postman 中输入请求并单击 Send 按钮时，服务器将接收您的请求并返回 Postman 在接口中显示的响应。



Request 编辑

在主界面左侧可以查看、保存、编辑 Request。

发送不同类型 HTTP 请求

GET

HTTP GET 请求方法用于从服务器检索数据。数据由唯一的 URI(统一资源标识符)标识。

GET 请求可以使用“Query String Parameters”将参数传递给服务器。例如，在下面的请求中，

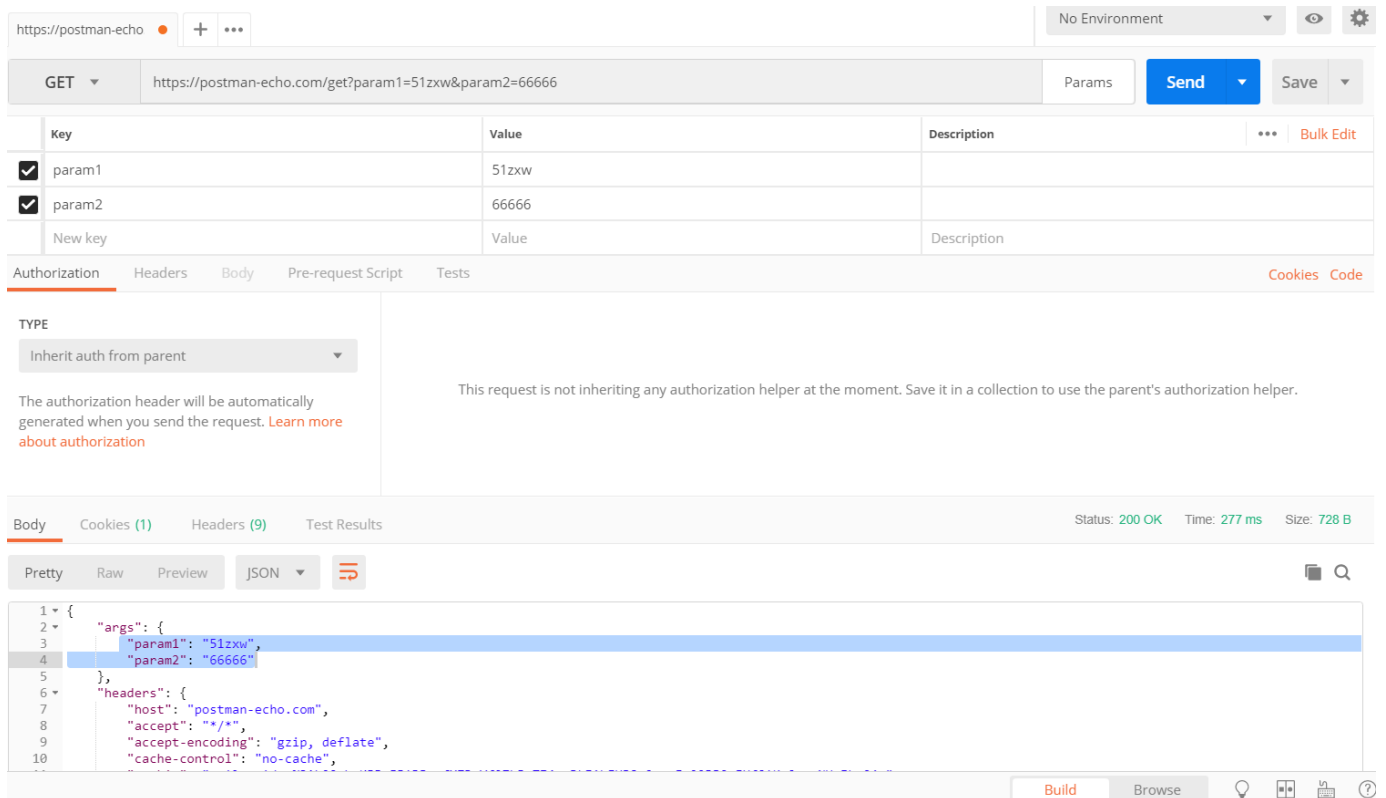
```
https://postman-echo.com/get?param1=51zxw&param2=66666
```

请求说明

- param1 和 param2 表示发送的参数。
- ?后面接参数
- &连接多个参数

参数编辑

- 点击 Params 按钮，Postman 可以自动帮我们解析出对应参数。
- 如果想要暂时不传参数，可以方便的通过不勾选的方式去实现
- 如果想要批量的编辑参数，可以点击右上角的 Bulk Edit，去实现批量编辑



The screenshot shows the Postman interface for a GET request to `https://postman-echo.com/get?param1=51zxw¶m2=66666`. The **Params** tab is active, displaying a table of query parameters:

Key	Value	Description
<input checked="" type="checkbox"/> param1	51zxw	
<input checked="" type="checkbox"/> param2	66666	
New key	Value	Description

Below the table, there are tabs for **Authorization**, **Headers**, **Body**, **Pre-request Script**, and **Tests**. The **Body** tab is selected, showing the request body in JSON format:

```
{  "args": {    "param1": "51zxw",    "param2": "66666"  },  "headers": {    "host": "postman-echo.com",    "accept": "*/*",    "accept-encoding": "gzip, deflate",    "cache-control": "no-cache"  }}
```

The status bar at the bottom indicates **Status: 200 OK**, **Time: 277 ms**, and **Size: 728 B**.

响应数据 在主界面下方一栏菜单为响应菜单栏，可以查看响应内容，Cookie、Headers、响应状态码等信息。



Body Cookies (1) Headers (9) Test Results Status: 200 OK Time: 275 ms Size: 719 B

Pretty Raw Preview JSON Save Response

```
1 {
2   "args": {
3     "param1": "51zxw",
4     "param2": "66666"
5   },
6   "headers": {
7     "host": "postman-echo.com",
8     "accept": "*/*",
9     "accept-encoding": "gzip, deflate",
10    "cache-control": "no-cache",
11    "cookie": "sails.sid=s%3A5670sCoh03HA-Dn4YB-hMplbdtcrGZd.z%2F7t2mbnm1i%2FBwmHKK%2BodkUR36kj403oTrSaet20j1c",
12    "postman-token": "c989e2df-b9d0-4634-a99f-f3b2ae404bc8",
13    "user-agent": "PostmanRuntime/7.1.5",
14    "x-forwarded-port": "443",
15    "x-forwarded-proto": "https"
16  },
17  "url": "https://postman-echo.com/get?param1=51zxw&param2=66666"
18 }
```

返回值:

```
{
  "args": {
    "param1": "51zxw",
    "param2": "66666"
  },
  "headers": {
    "host": "postman-echo.com",
    "accept": "*/*",
    "accept-encoding": "gzip, deflate",
    "cache-control": "no-cache",
    "cookie":
"sails.sid=s%3A5670sCoh03HA-Dn4YB-hMplbdtcrGZd.z%2F7t2mbnm1i%2FBwmHKK%2BodkUR36kj403oTrSaet20j1c",
    "postman-token": "f19b9f19-f0af-45ff-8dfd-0d1c75c1bbad",
    "user-agent": "PostmanRuntime/7.1.5",
    "x-forwarded-port": "443",
    "x-forwarded-proto": "https"
  },
  "url": "https://postman-echo.com/get?username=51zxw&password=66666"
}
```

POST

HTTP POST 请求方法旨在将数据传输到服务器，返回的数据取决于服务器的实现。 POST 请求可以使用 Query String Parameters 以及 body 将参数传递给服务器。



案例 1

在下面的请求中，使用 Query String Parameters 传递参数。

```
https://postman-echo.com/post?param=51zxw
```

返回值

```
{
  "args": {
    "param": "51zxw"
  },
  "data": {},
  "files": {},
  "form": {},
  "headers": {
    "host": "postman-echo.com",
    "content-length": "0",
    "accept": "*/*",
    "accept-encoding": "gzip, deflate",
    "cache-control": "no-cache",
    "content-type": "",
    "cookie":
"sails.sid=s%3A4L3j09wwnJ9JguJC-raHVYeuyVVEVHGW.za7nk%2B04gj9Nh%2FJDLzSZczT4k%2BR0eV0yTq8GJ5Y9YZo",
    "postman-token": "b34668bf-3850-4573-b196-bab2bd7db705",
    "user-agent": "PostmanRuntime/7.1.5",
    "x-forwarded-port": "443",
    "x-forwarded-proto": "https"
  },
  "json": null,
  "url": "https://postman-echo.com/post?param=51zxw"
}
```

案例 2

发送一个 Request, 其中 body 为 application/x-www-form-urlencoded 类型, 参数分别为 param1=zxw 和 param2=888 请

求 URL 如下:

```
https://postman-echo.com/post
```

https://postman-echo.com/post Examples (0)

POST https://postman-echo.com/post Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Cookies Code

form-data x-www-form-urlencoded raw binary

Key	Value	Description
param1	zxw	
param2	888	
New key	Value	Description

Body Cookies (1) Headers (8) Test Results Status: 200 OK Time: 236 ms Size: 645 B

Pretty Raw Preview JSON Save Response

```
1 {
2   "args": {},
3   "data": "",
4   "files": {},
5   "form": {
6     "param1": "zxw",
7     "param2": "888"
8   },
9   "headers": {
10    "host": "postman-echo.com",
11    "content-length": "21",
12    "accept": "*/*",
13    "accept-encoding": "gzip, deflate",
14    "cache-control": "no-cache",
15    "content-type": "application/x-www-form-urlencoded",
16    "cookie": "sails.sid=s%3A69wx4iZKJDD81qVsZUnB1RpcwTU-fNl_9.pr%2FfwjJzIFD1C9H7dFnInqMwkenjTJnwF8thN15KBzw",
17    "postman-token": "f5c798db-000c-4082-8283-d2c8be6ab20c",
```

Postman Body 数据类型说明：

- form-data** multipart/form-data 是 Web 表单用于传输数据的默认编码。这模拟了在网站上填写表单并提交它。表单数据编辑器允许我们为数据设置键-值对。我们也可以为文件设置一个键，文件本身作为值进行设置。
- x-www-form-urlencoded** 该编码与 URL 参数中使用的编码相同。我们只需输入键-值对，Postman 会正确编码键和值。请注意，我们无法通过此编码模式上传文件。表单数据和 urlencoded 之间可能存在一些差异，因此请务必首先检查 API 的编码实现，确定是否可以使用这种方式发送请求。
- raw** 请求可以包含任何内容。除了替换环境变量之外，Postman 不触碰在编辑器中输入的字符串。无论你在编辑区输入什么内容，都会随请求一起发送到服务器。编辑器允许我们设置格式类型以及使用原始主体发送的正确请求头。我们也可以手动设置 Content-Type 标题，这将覆盖 Postman 定义的设置。
- binary** 二进制数据可让我们发送 Postman 中无法输入的内容，例如图像，音频或视频文件。



返回值如下：

```
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "param1": "zxw",
    "param2": "888"
  },
  "headers": {
    "host": "postman-echo.com",
    "content-length": "21",
    "accept": "/*/*",
    "accept-encoding": "gzip, deflate",
    "cache-control": "no-cache",
    "content-type": "application/x-www-form-urlencoded",
    "cookie":
"sails.sid=s%3A69wx4iZKJDDb1qVsZUnB1RpcwTU-fN_9.pr%2FiwjJzIFDiC9H7dFnINqMwkenjTJnwF8thN15KBzw",
    "postman-token": "09d8e786-834a-42b9-bbe0-3e6886ef7b3b",
    "user-agent": "PostmanRuntime/7.1.5",
    "x-forwarded-port": "443",
    "x-forwarded-proto": "https"
  },
  "json": {
    "param1": "zxw",
    "param2": "888"
  },
  "url": "https://postman-echo.com/post"
}
```

PUT

HTTP PUT 请求主要是从客户端向服务器传送的数据取代指定的文档的内容。

PUT 请求可以使用 Query String Parameters 以及 body 请求体将参数传递给服务器。

案例：

发送 PUT 请求，并传递字符参数 “hello 51zxw”



<https://postman-echo>
<https://postman-echo.com/>

No Environment

Examples (0)

PUT

https://postman-echo.com/put

Params

Send

Save

Authorization

Headers (1)

Body

Pre-request Script

Tests

Cookies Code

form-data

x-www-form-urlencoded

raw

binary

Text

1

hello 51zxw

Body

Cookies (1)

Headers (8)

Test Results

Status: 200 OK Time: 226 ms Size: 643 B

Pretty

Raw

Preview

JSON

Save Response

```

1 {
2   "args": {},
3   "data": "",
4   "files": {},
5   "form": {
6     "hello 51zxw": ""
7   },
8   "headers": {
9     "host": "postman-echo.com",
10    "content-length": "11",
11    "accept": "*/*",
12    "accept-encoding": "gzip, deflate",
13    "cache-control": "no-cache",

```

<https://postman-echo.com/put>

返回值

```

{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "hello 51zxw": ""
  },
  "headers": {
    "host": "postman-echo.com",
    "content-length": "11",
    "accept": "*/*",
    "accept-encoding": "gzip, deflate",
    "cache-control": "no-cache",
    "content-type": "application/x-www-form-urlencoded",
    "cookie":
"sails.sid=s%3A-kzZXqiAKlk9oDgVADnLyqAEf7f6scDV.dhZZMReTg2y9KuTE%2Fxb902qGKnaUxD30%2B3J4PTTXZms",
    "postman-token": "1338c8b1-b502-45f6-9400-7be048d7b2ea",
    "user-agent": "PostmanRuntime/7.1.5",
    "x-forwarded-port": "443",
    "x-forwarded-proto": "https"
  },
  "json": {
    "hello 51zxw": ""
  }
}

```



```
},  
"url": "https://postman-echo.com/put"  
}
```

DELETE

HTTP DELETE 方法用于删除服务器上的资源，DELETE 请求可以使用 Query String Parameters 以及 body 请求体将参数传递给服务器。 delete 请求

```
https://postman-echo.com/delete
```

返回值

```
{  
  "args": {},  
  "data": {},  
  "files": {},  
  "form": {},  
  "headers": {  
    "host": "postman-echo.com",  
    "accept": "*/*",  
    "accept-encoding": "gzip, deflate",  
    "cache-control": "no-cache",  
    "cookie":  
"sails.sid=s%3A-kzZXqiAKlk9oDgVADnLyqAEf7f6scDV.dhZZMReTg2y9KuTE%2Fxb902qGKnaUxD30%2B3J4PTTXZms",  
    "postman-token": "065cb8c4-cea2-4e24-9be0-573d58b4da2c",  
    "user-agent": "PostmanRuntime/7.1.5",  
    "x-forwarded-port": "443",  
    "x-forwarded-proto": "https"  
  },  
  "json": null,  
  "url": "https://postman-echo.com/delete"  
}
```

Request Header

Request Header (请求头) 用来说明服务器要使用的附加信息，比较重要的信息有 Cookie、Referer、User-Agent 等。在 Postman 中可以在请求下方的 Headers 栏目来设置，如下如图所示：





https://postman-echo.com/get?param1=51zxw¶m2=66666

Examples (0)

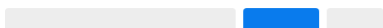
GET https://postman-echo.com/get?param1=51zxw¶m2=66666 Params Send Save

Authorization Headers (4) Body Pre-request Script Tests Cookies Code

Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/> Accept					
<input checked="" type="checkbox"/> Content-Type					
<input checked="" type="checkbox"/> Cookie					
<input checked="" type="checkbox"/> User-Agent	application/json				
New key	Value	Description			

Response

Hit the Send button to get a response.



Response Header

Response Header(响应头)其中包含了服务器对请求的应答信息，如 Content-Type、Server、Set-Cookie 等，在 Postman 主界面下方 Headers 或者 Postman Console 界面都可以查看 Response Header 信息。

▼ GET https://postman-echo.com/get?param1=51zxw&param2=66666

Pretty Raw

▼ Request Headers:

```
cache-control: "no-cache"
postman-token: "bc6c633f-f0b4-4ced-9a7e-91f916840bd2"
user-agent: "PostmanRuntime/7.1.5"
accept: "*/*"
host: "postman-echo.com"
cookie: "sails.sid=s%3A5670sCoh03HA-Dn4YB-hMpLdbtcrGZd.z%2F7t2mbnm1i%2FBwmHHK%2BodkUR36kj403oTrSaet20j1c"
accept-encoding: "gzip, deflate"
```

▼ Response Headers:

```
content-encoding: "gzip"
content-type: "application/json; charset=utf-8"
date: "Thu, 12 Jul 2018 01:28:22 GMT"
etag: "W/\"1d9-Rc4fI00GJtX04ztMCbJWovy/XzQ\""
server: "nginx"
vary: "Accept-Encoding"
content-length: "344"
connection: "keep-alive"
```

▼ Response Body:

▼ args:

```
param1: "51zxw"
param2: "66666"
```

▼ headers:

```
host: "postman-echo.com"
accept: "*/*"
accept-encoding: "gzip, deflate"
cache-control: "no-cache"
cookie: "sails.sid=s%3A5670sCoh03HA-Dn4YB-hMpLdbtcrGZd.z%2F7t2mbnm1i%2FBwmHHK%2BodkUR36kj403oTrSaet20j1c"
postman-token: "bc6c633f-f0b4-4ced-9a7e-91f916840bd2"
user-agent: "PostmanRuntime/7.1.5"
x-forwarded-port: "443"
x-forwarded-proto: "https"
url: "https://postman-echo.com/get?param1=51zxw&param2=66666"
```

Tips: 通过 Postman Console 可以看到每次请求的 Request Header 详细信息，详见视频演示。

授权设置

很多时候，出于安全考虑我们的接口并不希望对外公开。这个时候就需要使用授权(Authorization)机制 授权过程验证您是否具有访问服务器所需数据的权限。当您发送请求时，您通常必须包含参数，以确保请求具有访问和返回所需数据的权限。 Postman 提供授权类型，可以轻松地在 Postman 本地应用程序中处理身份验证协议。

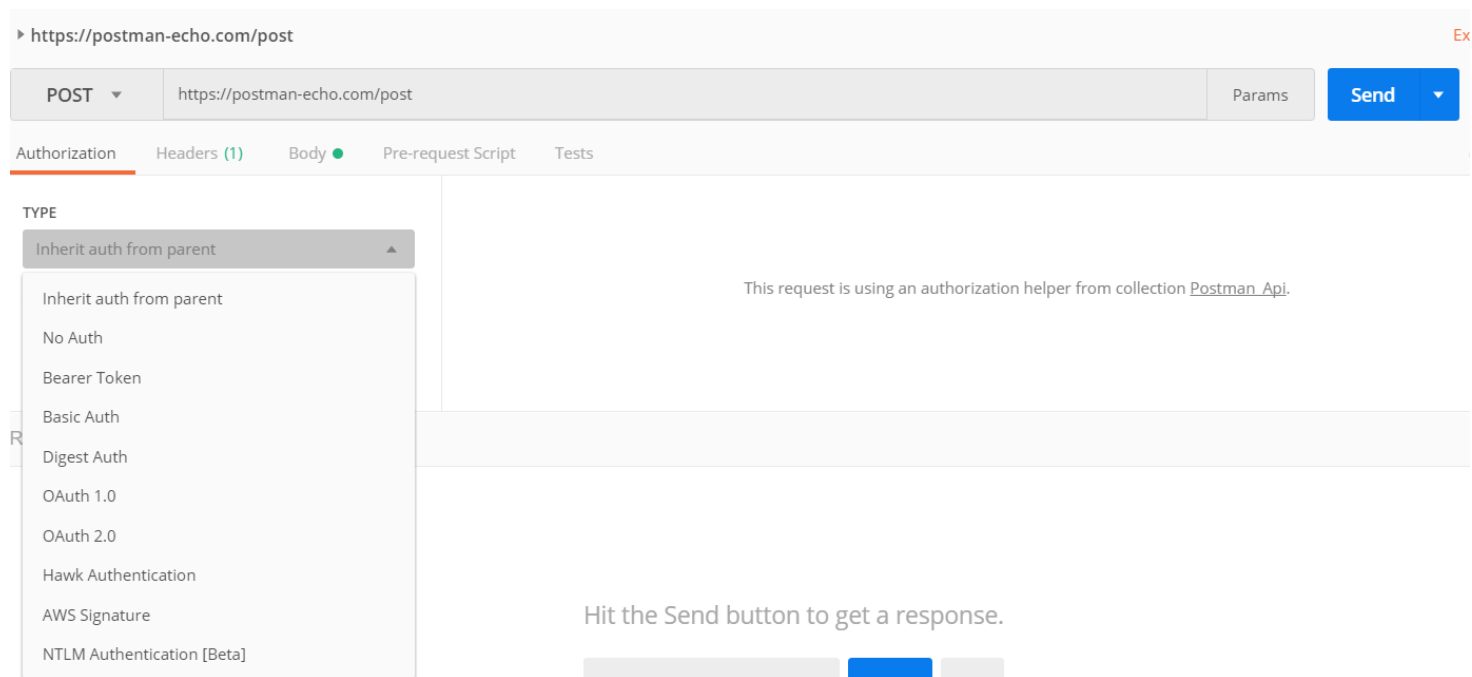
Postman 支持的授权协议类型如下：

- No Auth



- Bearer Token
- **Basic auth**
- **Digest Auth**
- **OAuth 1.0**
- OAuth 2.0
- **Hawk Authentication**
- AWS Signature
- NTLM Authentication [Beta]

这里主要介绍以上**加粗**的授权协议的使用。



Basic auth

基本身份验证是一种比较简单的授权类型，需要经过验证的用户名和密码才能访问数据资源。这就需要我们输入用户名和对应的密码。

案例：请求 URL 如下，授权账号为：

- 用户名: postman
- 密码: password
- 授权协议为: Basic auth

`http://postman-echo.com/basic-auth`

- 如果不输入用户名密码，直接使用 GET 请求，则会返回提示：Unauthorized
- 输入用户名密码，选择 Basic auth 授权类型，则返回如下结果：

```
{
  "authenticated": true
}
```

Digest Auth

Digest auth 是一个简单的认证机制，最初是为 HTTP 协议开发的，因此也常叫做 HTTP 摘要。其身份验证机制非常简单，它采用哈希加密方法，以避免用明文传输用户的口令。摘要认证就是要核实参与通信的两方都知道双方共享的一个口令。

当 server 想要查证用户的身份，它产生一个摘要盘问 (digest challenge)，并发送给用户。典型的摘要盘问例如以下：

```
Digest realm="iptel.org", qop="auth,auth-int",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093", opaque="", algorithm=MD5
```

这里包含了一组参数，也要发送给用户。用户使用这些参数，来产生正确的摘要回答，并发送给 server。摘要盘问中的各个参数，其意义例如以下：

realm (领域)：领域参数是强制的，在全部的盘问中都必须有。它是目的是鉴别 SIP 消息中的机密。在 SIP 实际应用中，它通常设置为 SIP 代理 server 所负责的域名。

nonce (现时)：这是由 server 规定的字符串，在 server 每次产生一个摘要盘问时，这个参数都是不一样的（与前面所产生的不会雷同）。“现时”一般是由一些数据通过 md5 杂凑运算构造的。这种数据通常包含时间标识和 server 的机密短语。这确保每一个“现时”都有一个有限的生命期（也就是过了一些时间后会失效，并且以后再也不会使用），并且是独一无二的（即不论什么其他的 server 都不能产生一个同样的“现时”）。

algorithm (算法)：这是用来计算的算法。当前仅仅支持 MD5 算法。

qop (保护的质量)。这个参数规定 server 支持哪种保护方案。client 能够从列表中选择一个。值 auth 表示仅仅进行身份查验，auth-int 表示进行查验外，另一些完整性保护。须要看更具体的描写叙述，请参阅 RFC2617。

案例

请求 URL 如下

```
http://postman-echo.com/digest-auth
```

摘牌配置信息如下：用户名密码和上面 basic auth 一样

```
Digest username="postman", realm="Users", nonce="ni1LiL0037PRRhofWdCLmwFsnEtH1lew", uri="/digest-auth",  
response="254679099562cf07df9b6f5d8d15db44", opaque=""
```



GET

https://postman-echo.com/digest-auth

Params

Send

Save

Digest Auth

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

By default, Postman will extract values from the received response, add it to the request, and retry it. Do you want to disable this?
☐ Yes, disable retrying the request

Preview Request

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)

Username

postman

Password

☐ Show Password

ADVANCED

These are advanced configuration options. They are optional. Postman will auto generate values for some fields if left blank.

Realm

Users

Nonce

ni1LiLOO37PRRhofWdCLmwFsnEtH1lew

Algorithm

MD5

qop

e.g. auth-int

Nonce Count

e.g. 00000001

Client Nonce

e.g. 0a4f113b

Opaque

Opaque

执行请求结果如下：

```
{
  "authenticated": true
}
```

Hawk Auth

Hawk Auth 是一个 HTTP 认证方案，使用 MAC(Message Authentication Code，消息认证码算法)算法，它提供了对请求进行部分加密验证的认证 HTTP 请求的方法。hawk 方案要求提供一个共享对称密匙在服务器与客户端之间，通常这个共享的凭证在初始 TLS（安全传输层协议）保护阶段建立的，或者是从客户端和服务端都可用的其他一些共享机密信息中获得的。

案例

请求 URL 如下：

```
https://postman-echo.com/auth/hawk
```





密钥信息如下：

- Hawk Auth ID: dh37fgj492je
- Hawk Auth Key: werxhqb98rpaxn39848xrunpaw3489ruxnpa98w4rxn
- Algorithm: sha256

https://postman-echo.com/auth/hawk Examples (0)

GET https://postman-echo.com/auth/hawk Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Cookies Code

TYPE

Hawk Authentication

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Preview Request

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)

Hawk Auth ID: dh37fgj492je

Hawk Auth Key: werxhqb98rpaxn39848xrunpaw3489ruxnpa98w4rxn

Algorithm: sha256

ADVANCED

These are advanced configuration options. They are optional. Postman will auto generate values for some fields if left blank.

User: Username

Nonce: Nonce

ext: e.g. some-app-extra-data

app: Application ID

dig: e.g. delegated-by

执行结果：

```
{
  "message": "Hawk Authentication Successful"
}
```

如果将 key 改为其他任意的字符则返回如下结果：

```
{
  "statusCode": 401,
  "error": "Unauthorized",
  "message": "Bad mac",
  "attributes": {
    "error": "Bad mac"
  }
}
```



}

OAuth 1.0

OAuth (开放授权) 是一个开放标准，允许用户让第三方应用访问该用户在某一网站上存储的私密的资源（如照片，视频，联系人列表），而无需将用户名和密码提供给第三方应用。

扩展资料：

- [OAuth 那些事儿](#)
- [OAuth 的改变](#)

案例

请求 URL 如下：请求方式为 GET，Add authorization data to 设置为：Request Headers

`https://postman-echo.com/oauth1`

参数配置为：

- Consumer Key: RKCGzna7bv9YD57c
- Consumer Secret: D+EdQ-gs\$-%@2Nu7



GET

https://postman-echo.com/oauth1

Params

Send

Save

Authorization Headers (1) Body Pre-request Script Tests Cookies Code

TYPE

OAuth 1.0

The authorization data will be automatically generated when you send the request. [Learn more about authorization](#)

Add authorization data to

Request Headers

Preview Request

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)

Consumer Key

RKCGzna7bv9YD57c

Consumer Secret

D+EdQ-gs\$-%@2Nu7

Access Token

Access Token

Token Secret

Token Secret

ADVANCED

These are advanced configuration options. They are optional. Postman will auto generate values for some fields if left blank.

Signature Method

HMAC-SHA1

Timestamp

Timestamp

Nonce

Nonce

Version

1.0

Realm

testrealm@example.com

发送请求结果如下：

```
{
  "status": "pass",
  "message": "OAuth-1.0a signature verification was successful"
}
```

如果 Consumer Secret 错误则返回如下结果：

```
{
  "status": "fail",
  "message": "HMAC-SHA1 verification failed",
  "base_uri": "https://postman-echo.com/oauth1",
  "normalized_param_string":
"oauth_consumer_key=51zxw&oauth_nonce=pxSgzUivsBi&oauth_signature_method=HMAC-SHA1&oauth_timestamp=1531299384&oauth_version=1.0",
  "base_string":
"GET&https%3A%2F%2Fpostman-echo.com%2Foauth1&oauth_consumer_key%3D51zxw%26oauth_nonce%3DpxSgzUivsBi%26oauth_signature_method%3DHMAC-SHA1%26oauth_timestamp%3D1531299384%26oauth_version%3D1.0",
  "signing_key": "D%2BEdQ-gs%24-%25%402Nu7&"
}
```

扩展资料：[各个授权协议文档](#)



Cookie 设置

cookie 是存储在浏览器中的小片段信息，每次请求后都将其发送回服务器，以便在请求之间存储有用的信息。比如很多网站登录界面都有保留账号密码，以便下次登录。

由于 HTTP 是一种无状态的协议，服务器单从网络连接上无从知道客户身份。怎么办呢？就给客户端们颁发一个通行证吧，每人一个，无论谁访问都必须携带自己通行证。这样服务器就能从通行证上确认客户身份了。这就是 Cookie 的工作原理。

Cookie 是由服务端生成，存储在响应头中，返回给客户端，客户端会将 cookie 存储下来，在客户端发送请求时，user-agent 会自动获取本地存储的 cookie，将 cookie 信息存储在请求头中，并发送给服务端。postman 也可以设置、获取、删除 Cookie。

Set Cookies

在 Send 按钮下方点击 Cookies 文字菜单，弹出如下界面，然后可以设置 Cookie。

MANAGE COOKIES

Type a domain name

Add

sojson.com 1 cookie

__cfduid X + Add Cookie

v.juhe.cn 1 cookie

aliyungf_tc X + Add Cookie

baidu.com 4 cookies

BAIDUID X BIDUPSID X PSTM X H_PS_PSSID X + Add Cookie

www.baidu.com 2 cookies

BDSVRTM X BD_HOME X + Add Cookie

postman-echo.com 3 cookies

[Learn More](#)

请求 URL 如下：请求方式为 GET,添加 Cookie 值为 username:51zxw

<http://www.baidu.com/>

打开 Console 找到 Request Header 可以看到自定义设置的 Cookie 内容。

File Edit View Help

Q Filter Messages

Love working on Postman? Work with us to make Postman better! <https://go.pstmn.io/postman-jobs>

▼ GET http://www.baidu.com/www.baidu.com/img/baidu_85beaf5496f291521eb75ba38eacbd87.svg

Pretty

Raw

▼ Request Headers:

```
cache-control: "no-cache"
postman-token: "f286cfe3-acdc-43ee-b844-c4cf27991d95"
user-agent: "PostmanRuntime/7.1.5"
accept: "*/*"
cookie: "BDSVRTM=0; BD_HOME=0; name=51zxw"
accept-encoding: "gzip, deflate"
referer: "http://www.baidu.com/www.baidu.com/img/baidu_85beaf5496f291521eb75ba38eacbd87.svg"
```

▼ Response Headers:

```
date: "Thu, 12 Jul 2018 01:15:21 GMT"
server: "Apache"
p3p: "CP=" OTI DSP COR IVA OUR IND COM ""
set-cookie: "BAIDUID=101F344BCDB4329B6016E5463012D5BC:FG=1; expires=Fri, 12-Jul-19 01:15:21 GMT; max-age=31536000; path=/; domain=.baidu.com; version=1"
last-modified: "Fri, 22 Dec 2017 10:34:36 GMT"
etag: "3dcd-560eb5cea6700"
accept-ranges: "bytes"
cache-control: "max-age=86400"
expires: "Fri, 13 Jul 2018 01:15:21 GMT"
vary: "Accept-Encoding, User-Agent"
content-encoding: "gzip"
content-length: "4867"
connection: "Keep-Alive"
content-type: "text/html"
```

► Response Body:

Get Cookies

Cookie 获取比较简单，直接获取 Response Headers 里面的 set-cookie 值即可，或者在主界面下方 Cookie 菜单栏里面也可以查看。

▼ GET http://www.baidu.com/www.baidu.com/img/baidu_85beaf5496f291521eb75ba38eacbd87.svg

Pretty Raw

▼ Request Headers:

cache-control: "no-cache"
postman-token: "f286cfe3-acdc-43ee-b844-c4cf27991d95"
user-agent: "PostmanRuntime/7.1.5"
accept: "*//*"
cookie: "BDSVRTM=0; BD_HOME=0; name=51zxw"
accept-encoding: "gzip, deflate"
referer: "http://www.baidu.com/www.baidu.com/img/baidu_85beaf5496f291521eb75ba38eacbd87.svg"

▼ Response Headers:

date: "Thu, 12 Jul 2018 01:15:21 GMT"
server: "Apache"
p3p: "CP=" OTI DSP COR IVA OUR IND COM ""
set-cookie: "BAIDUID=15...029B0016E5463012D5BC:FG=1; expires=Fri, 12-Jul-19 01:15:21 GMT; max-age=31536000; path=/; domain=.baidu.com; version=1"
last-modified: "Fri, 22 Dec 2017 10:34:36 GMT"
etag: "\"3dcd-560eb5cea6700\""
accept-ranges: "bytes"
cache-control: "max-age=86400"
expires: "Fri, 13 Jul 2018 01:15:21 GMT"
vary: "Accept-Encoding, User-Agent"
content-encoding: "gzip"
content-length: "4867"
connection: "Keep-Alive"
content-type: "text/html"

► Response Body:

Delete Cookies

点击 Cookies 文字菜单,然后可以根据需求去清除对应的 Cookie。

变量

问题思考

在开发不同阶段可能存在不同的环境,比如测试环境和生产环境。

测试环境 API 如下:

```
https://dev.postman.com/get
https://dev.postman.com/post
https://dev.postman.com/put
```

生产环境 API 如下:

```
https://postman-echo.com/get
https://postman-echo.com/post
```



<https://postman-echo.com/put>

在这么情况下，按照常规思路要么你需要维护两套环境的 API，要么每次都手动一个个去修改 URL，不管哪种选择都比较麻烦且低效，那么有没有比较的好的方法来解决这个问题呢？

Postman 变量类型

通过比较我们可以发现，以上两组 API 主要是除了 host 不同之外其他都一样，其实把 Host 用**变量**替换，这样就可以灵活切换环境。

Postman 提供了变量设置，有 4 种变量类型。

- 本地变量(LocalVariable)
- 全局变量(Global Variable)
- 环境变量(Environment Variable)
- 数据变量(Data Variable)

环境变量

环境变量指在不同环境，同一个变量值随着环境不同而变化，比如我们上面举例场景就可以使用环境变量，当在测试环境时，host 值为: dev.postman.com ,当切换到生产环境时，host 值变为: postman-echo.com 。

环境变量设置： 在 postman 界面点击右上角眼睛图标，即可开始设置环境变量和全局变量。环境变量设置过程如下图所示：我们可以设置两种环境 dev 和 release,dev 是开发测试环境； release 是正式的生产环境。host 环境变量，根据不同的环境值不一样。



Environment

No Environment

Add

No active Environment

An environment is a set of variables that allow you to switch the context of your requests.

[Learn more about environments](#)

Globals

Edit

variable_key variable_value

[Get a global variable](#)

MANAGE ENVIRONMENTS

Add Environment

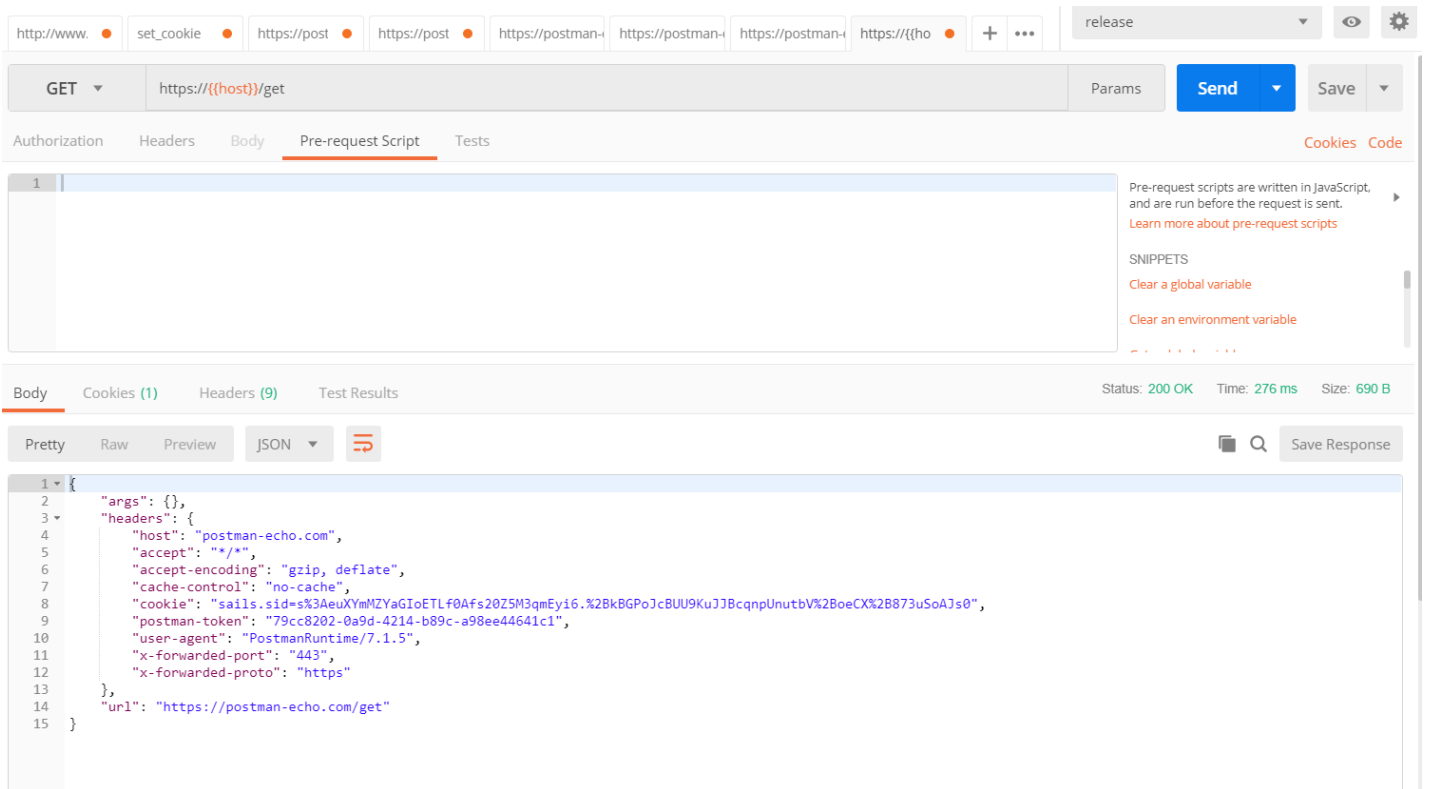
dev

	Key	Value	Bulk Edit
<input checked="" type="checkbox"/>	host	dev.postman.com	
	New key	Value	

Cancel

Add

变量引用格式为{{varname}},如下图所示:

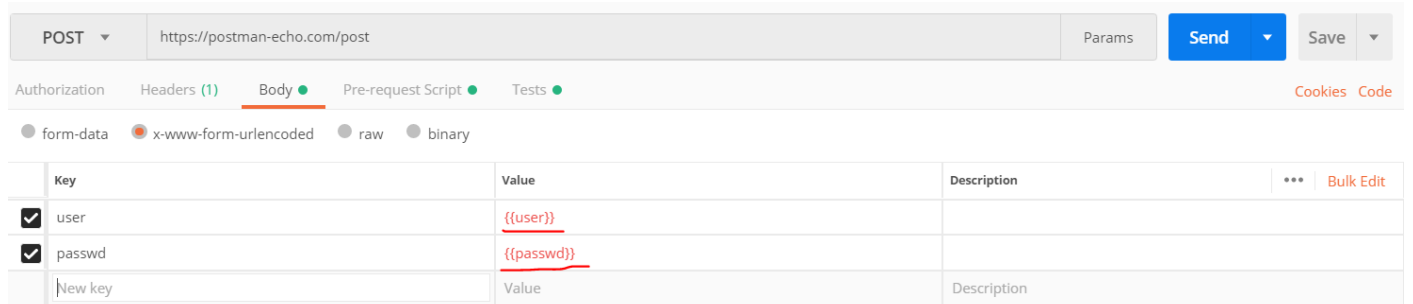


详细过程见视频演示。

本地变量

本地变量主要是针对单个 URL 请求设置的变量，作用域只是局限在请求范围内。如请求 URL 如下，设置两个本地变量 (user,passwd) 作为参数。请求方式为 POST

`https://postman-echo.com/post`



从上图中我们可以看到变量设置的格式为{{variable_name}}



变量设置好之后需要赋值，在 Pre-request-Script 里面编写如下代码：

```
pm.variables.set("user", "51zxw");
pm.variables.set("passwd", "66666");
```

点击 send 执行之后的返回值如下，可以看到我们定义的变量已经发送。

```
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "user": "51zxw",
    "passwd": "6666"
  },
  "headers": {
    "host": "postman-echo.com",
    "content-length": "24",
    "accept": "*/*",
    "accept-encoding": "gzip, deflate",
    "cache-control": "no-cache",
    "content-type": "application/x-www-form-urlencoded",
    "cookie":
"sails.sid=s%3ARNfROdcmaqrU3ZqgIMdbnqU4tI3_BHTE.WUBtzhaCnJT44oLzytikWtVmg7wfkD7tDNb%2FPtVBRnM",
    "postman-token": "9367ea9b-b349-4bc4-85d4-5638b5ff30e1",
    "user-agent": "PostmanRuntime/7.1.5",
    "x-forwarded-port": "443",
    "x-forwarded-proto": "https"
  },
  "json": {
    "user": "sutune",
    "passwd": "12345"
  },
  "url": "https://postman-echo.com/post"
}
```

全局变量

全局变量是指在所有的环境里面，变量值都是一样的，全局变量的作用域是所有请求。

全局变量设置有两种方式：





- 点击界面里设置
- 在脚本里设置

界面设置



点击眼睛图标后，在 Global 选项菜单点击 Edit 菜单即可设置全局变量，如下图所示。全局变量的引用格式和环境变量一样，

注意：当环境变量和全局变量名称一样时，切换到某个环境时，环境变量会覆盖全局变量。

MANAGE ENVIRONMENTS

Global variables for a workspace are a set of variables that are always available within the scope of that workspace. They can be viewed and edited by anyone in that workspace. [Learn more about globals](#)

Globals

	Key	Value	Bulk Edit
 <input checked="" type="checkbox"/>	<input type="text" value="variable_key"/>	<input type="text" value="variable_value"/>	
	<input type="text" value="New key"/>	<input type="text" value="Value"/>	

Download as JSON

Cancel

Save

脚本设置

使用如下脚本可以设置全局变量：variable_key 表示变量名称， variable_value 表示变量值。





```
pm.globals.set("variable_key", "variable_value");
```

实践案例

在实际接口测试过程中，接口经常会有关联。比如需要取上一个接口的某个返回值，然后作为参数传递到下一个接口作为参数。假设我们要获取 A 接口返回的 `userid` 值作为 B 接口的请求参数。

A 接口请求 URL 如下：

```
http://postman-echo.com/post
```

- 请求方式为 Post
- 请求参数：userid(这里自己定义，接口会返回对应的 id 值)

返回值

```
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "userid": "123456"
  },
  "headers": {
    "host": "postman-echo.com",
    "content-length": "13",
    "accept": "*/*",
    "accept-encoding": "gzip, deflate",
    "cache-control": "no-cache",
    "content-type": "application/x-www-form-urlencoded",
    "cookie":
"sails.sid=s%3AxZeg8NHeQGGFv2AUFblww7xF5HeP-4pi.sYxwR13VMFrUqvpJ%2Be8scGNnjgQAdOP3EeL6DSZqNQo",
    "postman-token": "7890d763-2bb1-4e88-aa1c-ad0bfc7db9b1",
    "user-agent": "PostmanRuntime/7.1.5",
    "x-forwarded-port": "443",
    "x-forwarded-proto": "https"
  },
  "json": {
    "userid": "123456"
  }
}
```



```
},  
  "url": "https://postman-echo.com/post"  
}
```

根据返回值我们需要从返回值中提取 `userid` 值。在 Test 标签栏下编写如下脚本获取 `userid` 值

```
//获取返回的响应值然后转化为 json 格式  
var jsonData = pm.response.json();  
  
//获取返回的userid 值  
userid=jsonData.json['userid'];  
  
//控制台日志查看  
console.log(userid);  
  
//将获取的变量设置全局变量  
pm.globals.set("userid", userid);
```

B 接口请求 URL 如下：请求方式为 GET

```
postman-echo.com/get?userid={{userid}}
```

先执行 A 接口的，然后在执行 B 接口，此时 B 接口通过全局变量 `userid` 可以获得 A 接口的返回值。

数据变量

数据变量是通过导入外部数据文件 (json 文件或者 csv 文件)，来获取变量数据。我们可以创建一个如下内容的 json

文件：

data.json

```
[{  
  "username": "jack",  
  "passwd": "6666"  
},{  
  "username": "Bob",  
  "passwd": "5555"  
}, {  
  "username": "Marry",  
  "passwd": "8888"  
}]
```

稍后我们会结合运行 Collection 来讲解如何导入该数据文件。



断言

简介

一般来说执行完测试，我们需要对测试结果来进行校验，判断结果是是否符合我们的预期，也就是断言。在接口测试中一般会根据响应状态码或者响应返回的数据来进行断言。

Postman 提供一个测试沙箱 ([Postman Sandbox](#)) 测试沙箱是一个 JavaScript 执行环境，可以通过 JS 脚本来编写 pre-request Script 和 test Script。

- pre-request Script (预置脚本) 可以用来修改一些默认参数,在请求发送之前执行。有点类似于 unittest 里面的 setUp()方法。
- test Script (测试脚本) 当接收到响应之后，再执行测试脚本。

案例

接口请求 URL 如下：请求方式为 POST

```
postman-echo.com/post
```

断言规则

- 响应状态码：200
- 响应内容：返回的 user 参数值与定义的一致
- 响应时间：小于 0.5s

测试脚本

在 pre-request Script 定义变量 user



```
pm.variables.set("user", 'zxw');
```

在 Test 栏下面编写如下脚本

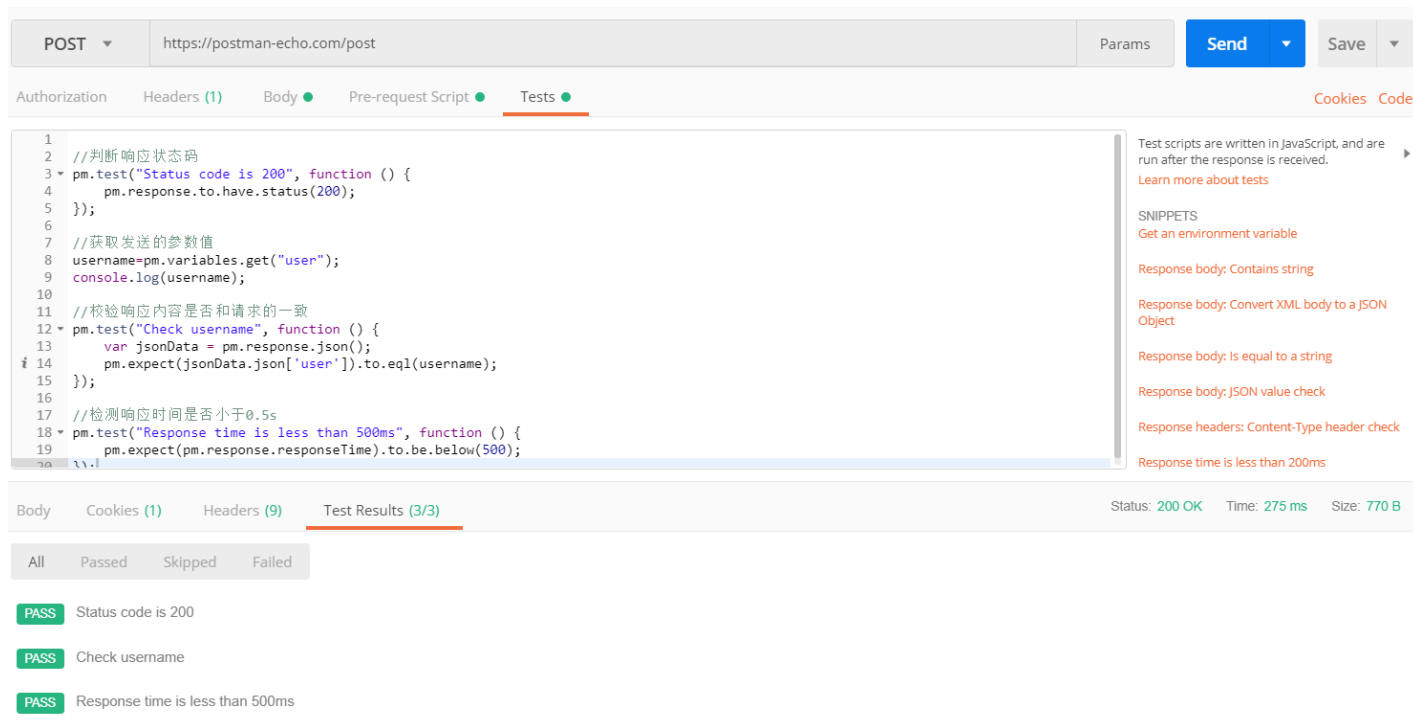
```
//判断响应状态码
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

//获取发送的参数值
username=pm.variables.get("user");
console.log(username);

//校验响应内容是否和请求的一致
pm.test("Check username", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.json['user']).to.eql(username);
});

//检测响应时间是否小于0.5s
pm.test("Response time is less than 500ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(500);
});
```

断言结果



The screenshot shows the Postman interface for a POST request to `https://postman-echo.com/post`. The 'Tests' tab is active, displaying the following JavaScript code:

```
1 //判断响应状态码
2 pm.test("Status code is 200", function () {
3     pm.response.to.have.status(200);
4 });
5
6 //获取发送的参数值
7 username=pm.variables.get("user");
8 console.log(username);
9
10 //校验响应内容是否和请求的一致
11 pm.test("Check username", function () {
12     var jsonData = pm.response.json();
13     pm.expect(jsonData.json['user']).to.eql(username);
14 });
15
16 //检测响应时间是否小于0.5s
17 pm.test("Response time is less than 500ms", function () {
18     pm.expect(pm.response.responseTime).to.be.below(500);
19 });
```

The right sidebar shows the 'Test Results' tab with the following status: **Status: 200 OK**, **Time: 275 ms**, **Size: 770 B**. Below this, there are three test results, all marked as **PASS**:

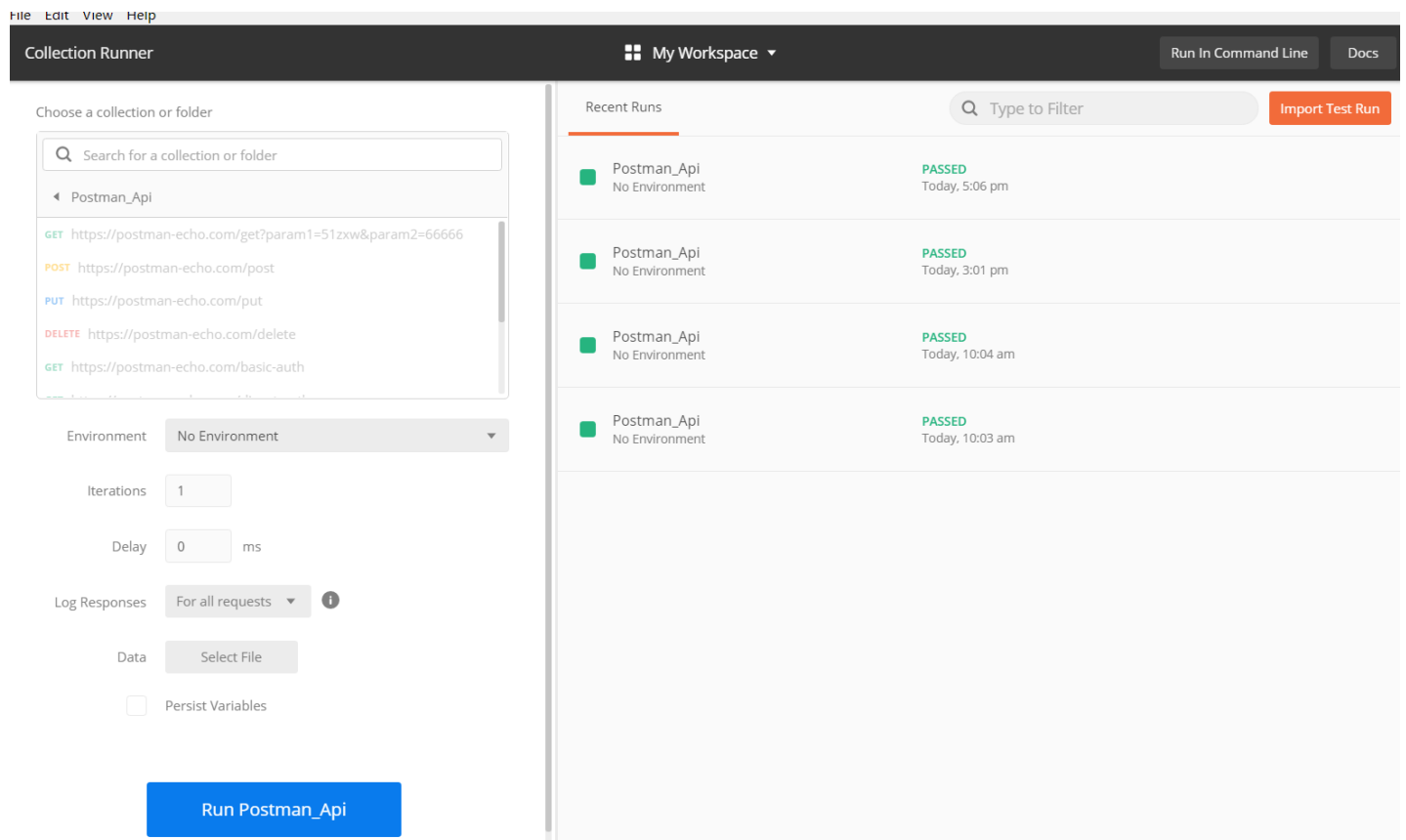
- PASS** Status code is 200
- PASS** Check username
- PASS** Response time is less than 500ms

扩展资料: [Postman 测试脚本官方文档](#)

运行 Collection

批量执行

当我们想批量测试某个集合里面的各个 API 时，可以使用 Collection Runner 来批量运行 API，同时可以进行环境变量、迭代执行次数、延迟时间等设置。



The screenshot shows the Postman Collection Runner interface. On the left, there's a sidebar with a search bar and a list of collections. The 'Postman_Api' collection is selected, showing its contents: GET https://postman-echo.com/get?param1=51zxw¶m2=66666, POST https://postman-echo.com/post, PUT https://postman-echo.com/put, DELETE https://postman-echo.com/delete, and GET https://postman-echo.com/basic-auth. Below the list, there are settings for Environment (No Environment), Iterations (1), Delay (0 ms), Log Responses (For all requests), Data (Select File), and a checkbox for Persist Variables. A blue button labeled 'Run Postman_Api' is at the bottom. On the right, the 'Recent Runs' section shows a list of four successful runs, each with a green status icon, the name 'Postman_Api', the environment 'No Environment', and a 'PASSED' status with a timestamp.

Run	Status	Time
1	PASSED	Today, 5:06 pm
2	PASSED	Today, 3:01 pm
3	PASSED	Today, 10:04 am
4	PASSED	Today, 10:03 am

执行结果

Collection Runner
File Edit View Help

Collection Runner
Run Results
My Workspace
Run In Command Line
Docs

3 PASSED
0 FAILED

Postman_Api No Environment
just now

Run Summary
Export Results
Retry
New

Iteration 1

GET	https://postman-echo.com/get?param1=51zxw¶m2=66...	https://postman-echo.co...	...t?param1=51zxw¶m2=66666	200 OK	243 ms	473 B
PASS	check status code					
PASS	Check param 51zxw					
PASS	Response time is less than 200ms					
POST	https://postman-echo.com/post	https://postman-echo.co...	... / https://postman-echo.com/post	200 OK	245 ms	519 B
This request does not have any tests.						
PUT	https://postman-echo.com/put	https://postman-echo.co...	...i / https://postman-echo.com/put	200 OK	246 ms	529 B
This request does not have any tests.						
DELETE	https://postman-echo.com/delete	https://postman-echo.co...	...https://postman-echo.com/delete	200 OK	241 ms	458 B
This request does not have any tests.						
GET	https://postman-echo.com/basic-auth	https://postman-echo.co...	...s://postman-echo.com/basic-auth	200 OK	240 ms	22 B
This request does not have any tests.						
GET	https://postman-echo.com/digest-auth	https://postman-echo.co...	...://postman-echo.com/digest-auth	200 OK	244 ms	22 B
This request does not have any tests.						
GET	https://postman-echo.com/auth/hawk	https://postman-echo.co...	...s://postman-echo.com/auth/hawk	200 OK	247 ms	44 B

数据驱动

应用背景

有时我们针对一个接口需要测试很多不同的参数，如果每次一个个的去修改参数值来进行测试这样效率肯定会比较低。因此我们需要每次迭代执行传入不同的参数进行测试，那么需要导入外部数据文件进行参数化，也就是所谓的数据驱动。

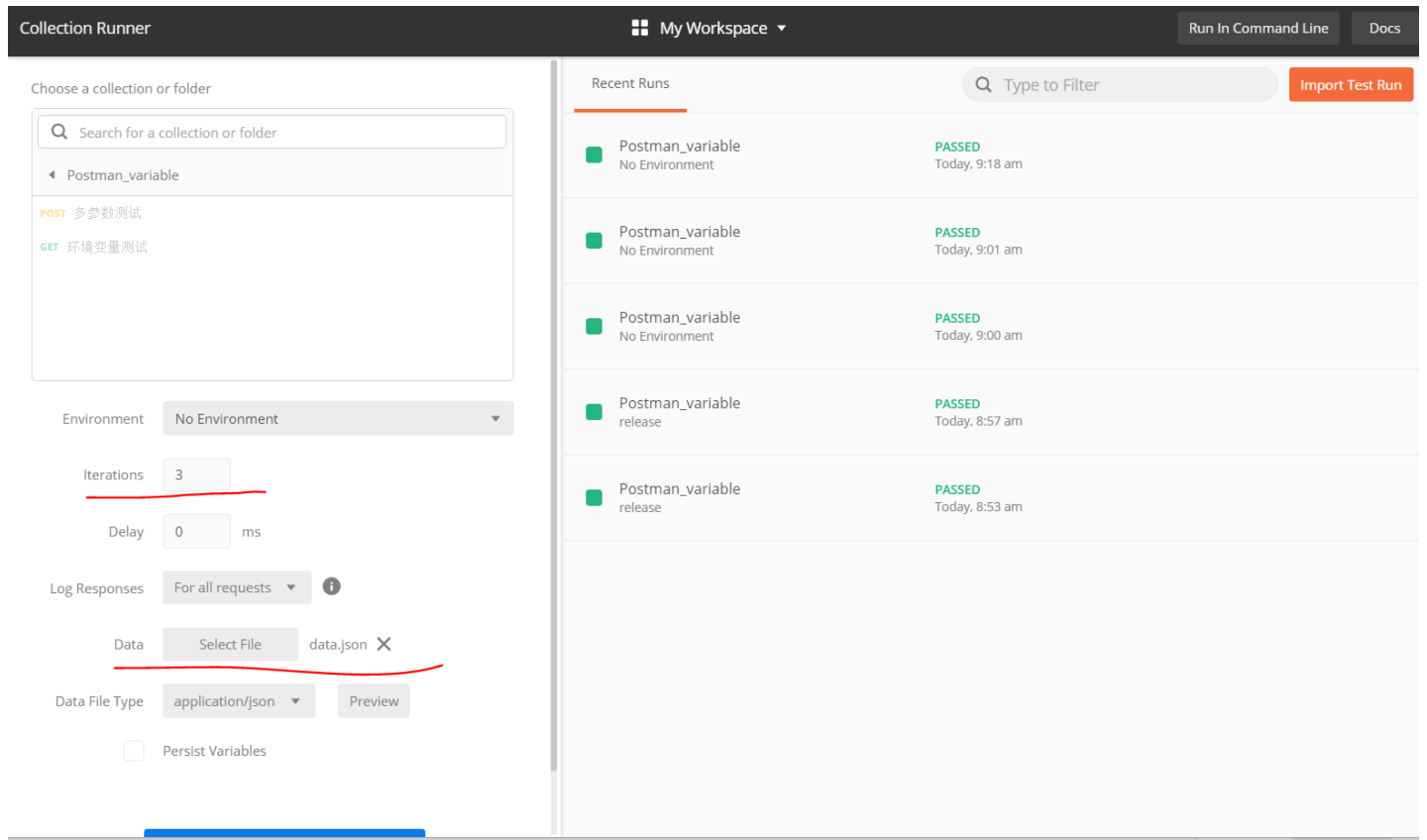
数据导入

如下图所示，data 选择之前我们创建的 json 数据文件：data.json,文件类型选择 application/json json 数据内容如下：

```
[{
  "username": "jack",
```

```
"passwd": "6666"
}, {
  "username": "Bob",
  "passwd": "5555"
}, {
  "username": "Marry",
  "passwd": "8888"
}]
```

请求之前延迟时间最好设置为 1000~3000，避免过于频繁请求被禁。



Collection Runner

My Workspace

Run In Command Line Docs

Choose a collection or folder

Search for a collection or folder

Postman_variable

POST 多参数测试

GET 环境变量测试

Environment: No Environment

Iterations: 3

Delay: 0 ms

Log Responses: For all requests

Data: Select File data.json

Data File Type: application/json

Preview

Persist Variables

Recent Runs

Type to Filter

Import Test Run

Test Name	Environment	Status	Time
Postman_variable	No Environment	PASSED	Today, 9:18 am
Postman_variable	No Environment	PASSED	Today, 9:01 am
Postman_variable	No Environment	PASSED	Today, 9:00 am
Postman_variable release		PASSED	Today, 8:57 am
Postman_variable release		PASSED	Today, 8:53 am

点击 Preview 按钮可以预览导入的数据。



PREVIEW DATA



Iteration	username	passwd
1	"jack"	"6666"
2	"Bob"	"5555"
3	"Marry"	"8888"

执行结果

Collection Runner

Run Results

My Workspace

Run In Command Line

Docs

9 PASSED

0 FAILED

Postman_variable No Environment just now

Run Summary

Export Results

Retry

New

Iteration 1

POST 多参数测试 https://postman-echo.co... Postman_variable / 多参数测试 200 OK 332 ms 589 B

PASS Status code is 200

PASS Check username

PASS Response time is less than 500ms

GET 环境变量测试 https://postman-echo.co... Postman_variable / 环境变量测试 200 OK 283 ms 408 B

This request does not have any tests.

Iteration 2

POST 多参数测试 https://postman-echo.co... Postman_variable / 多参数测试 200 OK 290 ms 579 B

PASS Status code is 200

PASS Check username

PASS Response time is less than 500ms

GET 环境变量测试 https://postman-echo.co... Postman_variable / 环境变量测试 200 OK 321 ms 412 B

This request does not have any tests.

Iteration 3

POST 多参数测试 https://postman-echo.co... Postman_variable / 多参数测试 200 OK 275 ms 587 B

PASS Status code is 200

我要自學網

www.51zxw.net 原创出品，盗版必究

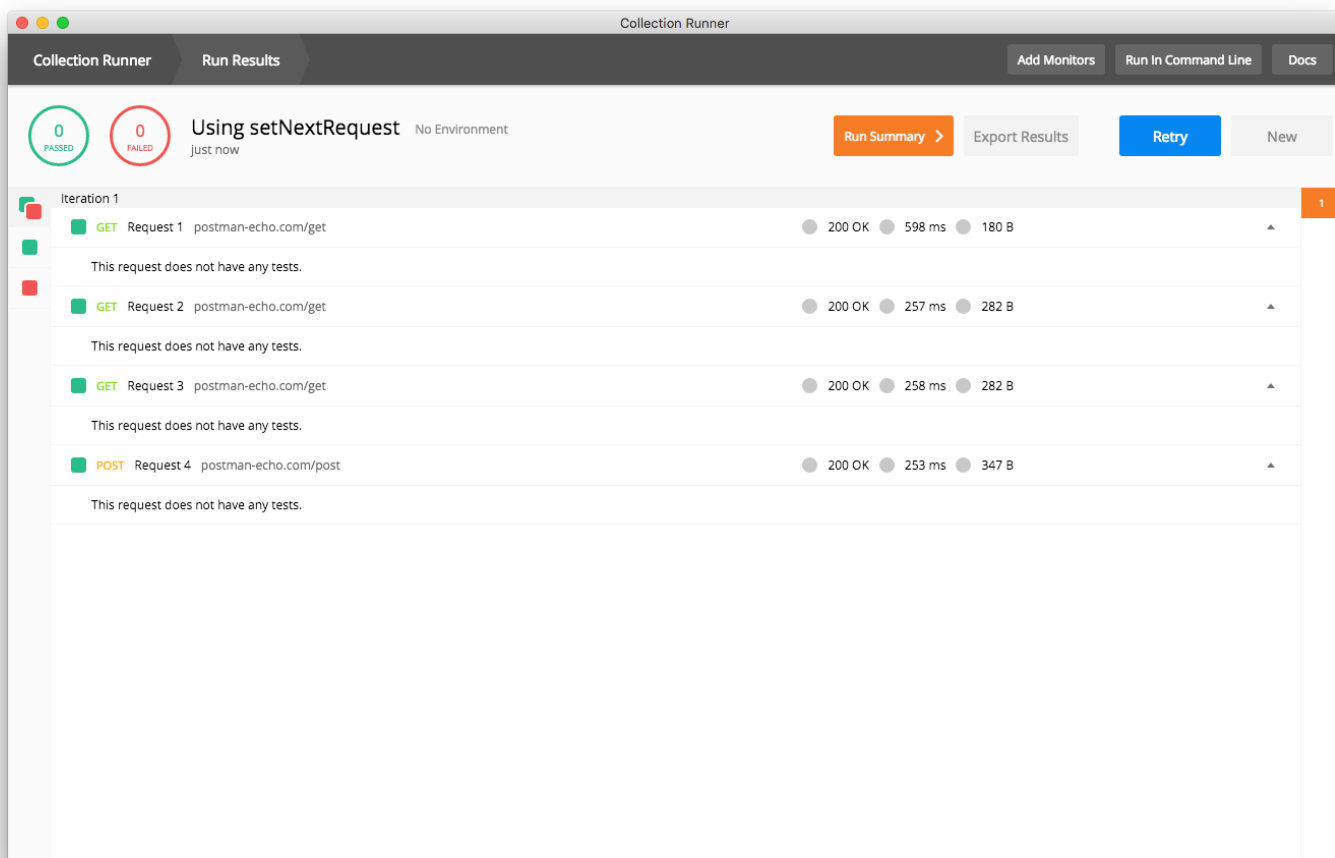
构建工作流

问题思考

在使用“Collection Runner”的时候，集合中的请求执行顺序就是请求在 Collection 中的显示排列顺序。但是，有的时候我们不希望请求按照这样的方式去执行，可能是执行完第一个请求，再去执行第五个请求，然后再去执行第二个请求这样的方式；那么在“Collection Runner”中如何去构建不同的执行顺序呢？

设置方法

最直接的方法就是直接在集合里面拖动调整顺序，但是每次去拖动也比较麻烦，特别是当请求比较多的时候。这个时候最高效的方法就是通过脚本设置。 首先下载官方提供的案例文件:[collection.json](#) 导入到 postman，运行 Collection 结果如下图所示：



接下来要调整执行顺序为：Request1->Request3->Request2->Request4

首先在第一个请求:Request1 中 Test 添加如下代码：表示下一个请求为执行请求名称为 Request3 的请求

```
postman.setNextRequest('Request 3')
```

然后在 Request3 的请求中 Test 添加如下代码：表示下一个请求为执行请求名称为 Request2 的请求

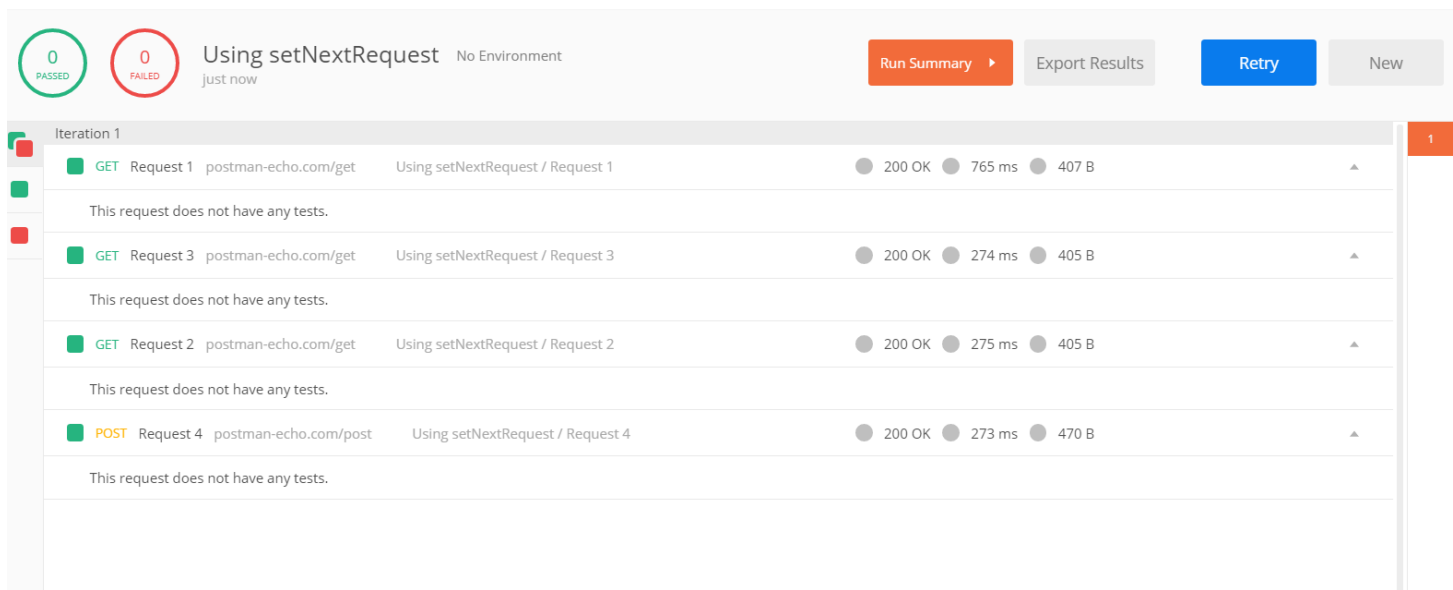
```
postman.setNextRequest('Request 2')
```

最后在 Request2 的请求中 Test 添加如下代码：表示下一个请求为执行请求名称为 Request4 的请求.

```
postman.setNextRequest('Request 4')
```

注意：第一个执行请求的排序一定要在第一个。

执行结果



The screenshot shows the Postman test results interface. At the top, there are two circular status indicators: a green one with '0 PASSED' and a red one with '0 FAILED'. The title 'Using setNextRequest' is followed by 'No Environment' and 'just now'. On the right, there are buttons for 'Run Summary', 'Export Results', 'Retry', and 'New'. Below this, a table lists the test results for 'Iteration 1'. The table has columns for the request type, name, URL, environment, status, time, and size. There are four requests listed, all of which passed with a status of '200 OK'. Each request is followed by a note: 'This request does not have any tests.'.

Request Type	Request Name	URL	Environment	Status	Time	Size
GET	Request 1	postman-echo.com/get	Using setNextRequest / Request 1	200 OK	765 ms	407 B
This request does not have any tests.						
GET	Request 3	postman-echo.com/get	Using setNextRequest / Request 3	200 OK	274 ms	405 B
This request does not have any tests.						
GET	Request 2	postman-echo.com/get	Using setNextRequest / Request 2	200 OK	275 ms	405 B
This request does not have any tests.						
POST	Request 4	postman-echo.com/post	Using setNextRequest / Request 4	200 OK	273 ms	470 B
This request does not have any tests.						

相关资料: [collection runs](#) 官方文档

命令执行

问题思考

在前面我们都是在 postman 图形界面工具里面进行测试,但是有时候我们需要把测试脚本集成到 CI 平台,或者在非图形界面的系统环境下测试,那么该如何处理呢?

Newman 简介

[Newman](#) 是一款基于 Node.js 开发的可以运行 Postman 的工具,使用 Newman,可以直接从命令行运行和测试 Postman 集合。

Newman 应用

环境准备

- Node.js
- cnpm 或 npm

以上安装可以参考：[Appium 环境搭建](#)

配置好环境后，执行如下命令安装 newman

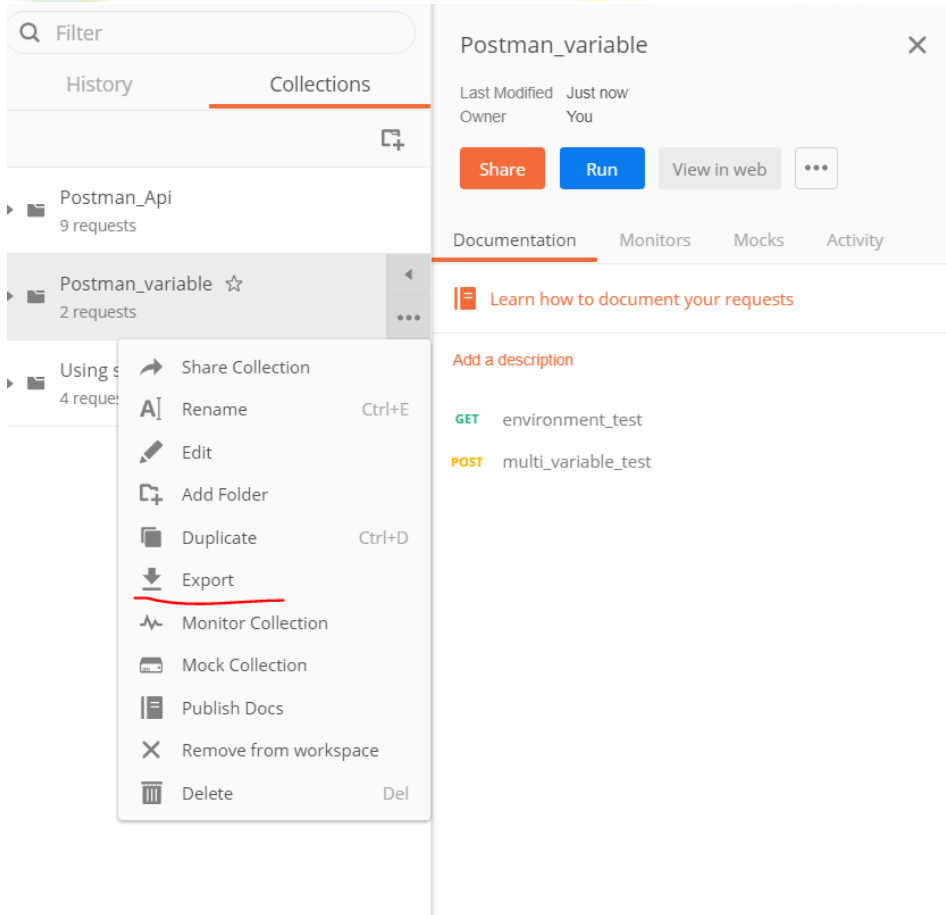
```
cnpm install newman --global
```

输入如下面命令检测安装是否成功

```
C:\Users\Shuqing>newman -v  
3.10.0
```

执行测试

首先将 postman 的集合导出，如下图所示:



在桌面新建文件夹 `pmtest`,将导出的 `postman` 文件和相关数据文件放入。

打开 `cmd` 进入到 `pmtest` 目录，输入如下命令：

```
newman run Postman_API.postman_collection.json -d data.json -r html
```

命令说明

- `run` 代表要执行的 `postman` 脚本，即为导出的集合。
- `-d` 表示要执行的数据，也就是之前导入 `postman` 的数据
- `-r` 生成的测试报告类型,这里生成 `html` 格式报告

更多命令用法请输入 `newman -h` 即可查看。

报告查看

在测试文件夹 `pmtest` 里面可以看到生成的一个 `newman` 文件夹，打开就可以看到生成的测试报告。



Html 报告样式: [newman-run-report](#)

newman 不仅支持生成 html 报告，还支持其他报告类型：

- JSON reporter
- JUNIT/XML reporter
- Client report
- Html report

集成 Jenkins

Jenkins 简介

Jenkins 是一个开源软件项目，是基于 Java 开发的一种[持续集成](#)工具，用于监控持续重复的工作，旨在提供一个开放易用的软件平台，使软件的持续集成变成可能。

下载与安装

下载地址: <https://jenkins.io/download/>

下载后安装到指定的路径即可,默认启动页面为 localhost:8080,如果 8080 端口被占用无法打开,可以进入到jenkins

安装目录, 找到 jenkins.xml 配置文件打开, 修改如下代码的端口号即可。

```
<arguments>-Xrs -Xmx256m -Dhudson.lifecycle=udson.lifecycle.WindowsServiceLifecycle -jar  
"%BASE%\jenkins.war" --httpPort=8080 --webroot="%BASE%\war"</arguments>
```

集成步骤

集成到 jenkins 的思路其实很简单，就把之前我们执行测试的 cmd 命令放到 jenkins 里面去执行。集成步骤也很简单：

- 首先新建一个项目：postman_api_test
- 然后在构建栏目下拉菜单选择 Execute Windows batch command

```
c:
cd C:\Users\Shuqing\Desktop\pmtest\
newman run Postman_API.postman_collection.json -d data.json -r html
```

Tips：我的 jenkins 安装在 D 盘因此需要使用命令 c: 切换到 postman 脚本所在盘符。

构建环境

☐ Delete workspace before build starts

☐ Abort the build if it's stuck

☐ Add timestamps to the Console Output

☐ Use secret text(s) or file(s)

构建

Execute Windows batch command

命令

```
c:
cd C:\Users\Shuqing\Desktop\pmtest\
newman run Postman_variable.postman_collection.json -d data.json -r html
```

[参阅 可用环境变量列表](#)

高级...

增加构建步骤

构建后操作

增加构建后操作步骤

保存

应用

最后执行结果如下：



- 返回到工程
- 状态集
- 变更记录
- 控制台输出
- 文本方式查看
- 编辑编译信息
- 删除本次生成
- 前一次构建

控制台输出

```
Started by user shuqing
Building in workspace D:\Program Files (x86)\jenkins\workspace\postman_api_test
[postman_api_test] $ cmd /c call C:\WINDOWS\TEMP\jenkins4903599935964272200.bat

D:\Program Files (x86)\jenkins\workspace\postman_api_test>c:

C:\>cd C:\Users\Shuqing\Desktop\pctest\

C:\Users\Shuqing\Desktop\pctest>newman run Postman_variable.postman_collection.json -d data.json -r html
Finished: SUCCESS
```

其他设置如设置定时执行，可以参考 Appium 教程中的：[6-14 框架综合实践 \(13\)—jenkins 自动化测试平台搭建](#)

导出不同语言脚本

问题思考

虽然 Postman 功能比较强大，但是毕竟是一款商业工具，多少会有一些限制。比如只支持 js 脚本运行，如果我们想用自己熟悉的编程语言(如：Python,java 等)来做接口自动化测试该如何处理？

操作步骤

Postman 支持导出不同语言版本的脚本，当一个接口调试好之后，点击右侧的 code 字样即弹出如下界面可以选择语言。最后选择你需要语言版本即可生成对应的代码。

Python Requests ▲

Copy to Clipboard

```
HTTP
C (LibCurl)
cURL
C# (RestSharp)
Go
Java
JavaScript
NodeJS
Objective-C (NSURLSession)
OCaml (Cohttp)
PHP
Python
Ruby (NET::Http)
Shell
Swift (NSURLSession)

tman-echo.com/get"
': "no-cache",
': "24db8091-6fdb-4456-a14a-3a23bae4618d"
s.request("GET", url, headers=headers)
t)|
```

生成的代码片段可以点击 Copy to Clipboard 复制。



Python Requests ▼

Copy to Clipboard

```
1 import requests
2
3 url = "https://postman-echo.com/get"
4
5 headers = {
6     'Cache-Control': "no-cache",
7     'Postman-Token': "351e65c3-4c90-4fe2-9d86-262c0a0c7d8b"
8 }
9
10 response = requests.request("GET", url, headers=headers)
11
12 print(response.text)
```

参考资料

- <https://docs.postman-echo.com/#1eb1cf9d-2be7-4060-f554-73cd13940174>
- <https://www.jellythink.com/archives/169>
- <https://blog.csdn.net/wangyuquanliuli/article/details/24850761>
- <https://www.cnblogs.com/happy-today/p/7928822.html>
- <http://www.mamicode.com/info-detail-1693734.html>
- https://blog.csdn.net/qq_14908027/article/details/77923792