

about

安装

postman for Windows

postman for Mac

快速上手

get请求

无参get请求

有参get请求

post请求

k:v形式的post请求

参数为json的post请求

webservice接口

文件上传接口测试

将接口生成代码

集合 (Collection)

新建一个集合

为集合添加接口用例

环境管理

创建一个新的环境

使用环境配置

变量的作用域

环境变量

全局变量

集合变量

内置动态变量

断言

添加/获取变量系列

一般断言

集合公共断言

特殊接口

签名接口

cookie

token

集合自动化

快速上手

基于数据驱动的集合自动化

命令行测试

环境配置

使用newman执行命令行测试

设置相关

主题设置

about

Postman是一款非常流行的HTTP/HTTPS接口测试工具，入门简单，功能强大，不但可以进行接口手动测试，还可以非常方便的进行自动化测试。支持参数化、断言、用例设计、测试报告等功能。

总之，很好很强大！

官网：<https://www.getpostman.com/>

安装

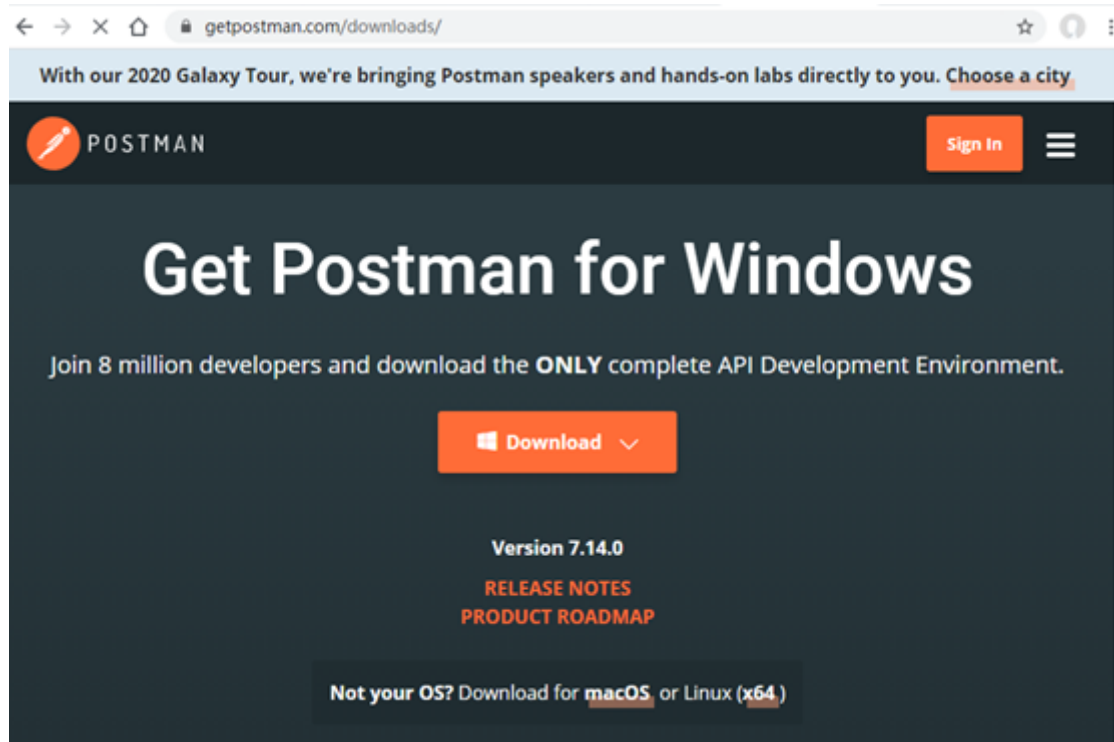
安装包：

链接：<https://pan.baidu.com/s/1VmuLR9vyMxExoP0E0rFNig>

提取码：8hbb

postman for Windows

官网下载安装即可：<https://www.getpostman.com/downloads/>



安装相当的简单，点击安装文件即可。

第一次打开需要登录或者注册，建议大家登录/注册一个账号比较好。

postman for Mac

参考：<https://jingyan.baidu.com/article/a3a3f81127f8e18da2eb8a1c.html>

第一次打开需要登录或者注册，建议大家登录/注册一个账号比较好。

快速上手

使用postman测试几种常见的HTTP接口：

- get接口
- post，一种参数形式为“k:v”类型，还有另一种是“k:json”类型的接口
- 文件上传类型
- webservice类型接口

get请求

get请求无非就是有参和无参的get请求。

无参get请求

url: <https://www.v2ex.com/api/site/info.json>

正常的输入请求的URL，选择请求类型，然后点击 Send 就可以获取到响应结果。该接口无需配置请求头和其他配置。

GET https://www.v2ex.com/api/site/info.json

Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies (1) Headers (18) Test Results Status: 200 OK Time: 1699ms Size: 970 B Save Response

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   "title": "V2EX",
3   "slogan": "way to explore",
4   "description": "创意工作者们的社区",
5   "domain": "www.v2ex.com"
6 }
```

有参get请求

url: <http://www.nneo.cc:6001/get?k1=v1&k2=v2>

带参数的get请求，可以跟url后面，Postman会自动的将参数填充到“Query Params”中。

GET https://www.v2... GET http://www.nneo... POST http://www.ht... POST http://www.ne... + ... No Environment

Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> k1	v1			
<input checked="" type="checkbox"/> k2	v2			
Key	Value	Description		

Body Cookies Headers (7) Test Results Status: 200 OK Time: 119ms Size: 654 B Save Response

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   "args": {
3     "k1": "v1",
4     "k2": "v2"
5   },
6   "headers": {
7     "Accept": "**/*",
8     "Accept-Encoding": "gzip, deflate",
9     "Cache-Control": "no-cache",
10    "Connection": "keep-alive",
11    "Host": "www.nneo.cc:6001",
12    "Postman-Token": "65efa539-f1c2-4597-8fb9-5eadc87e7469",
13    "User-Agent": "PostmanRuntime/7.20.1"
14  },
15   "origin": "222.35.242.139",
16   "url": "http://www.nneo.cc:6001/get?k1=v1&k2=v2"
17 }
```

post请求

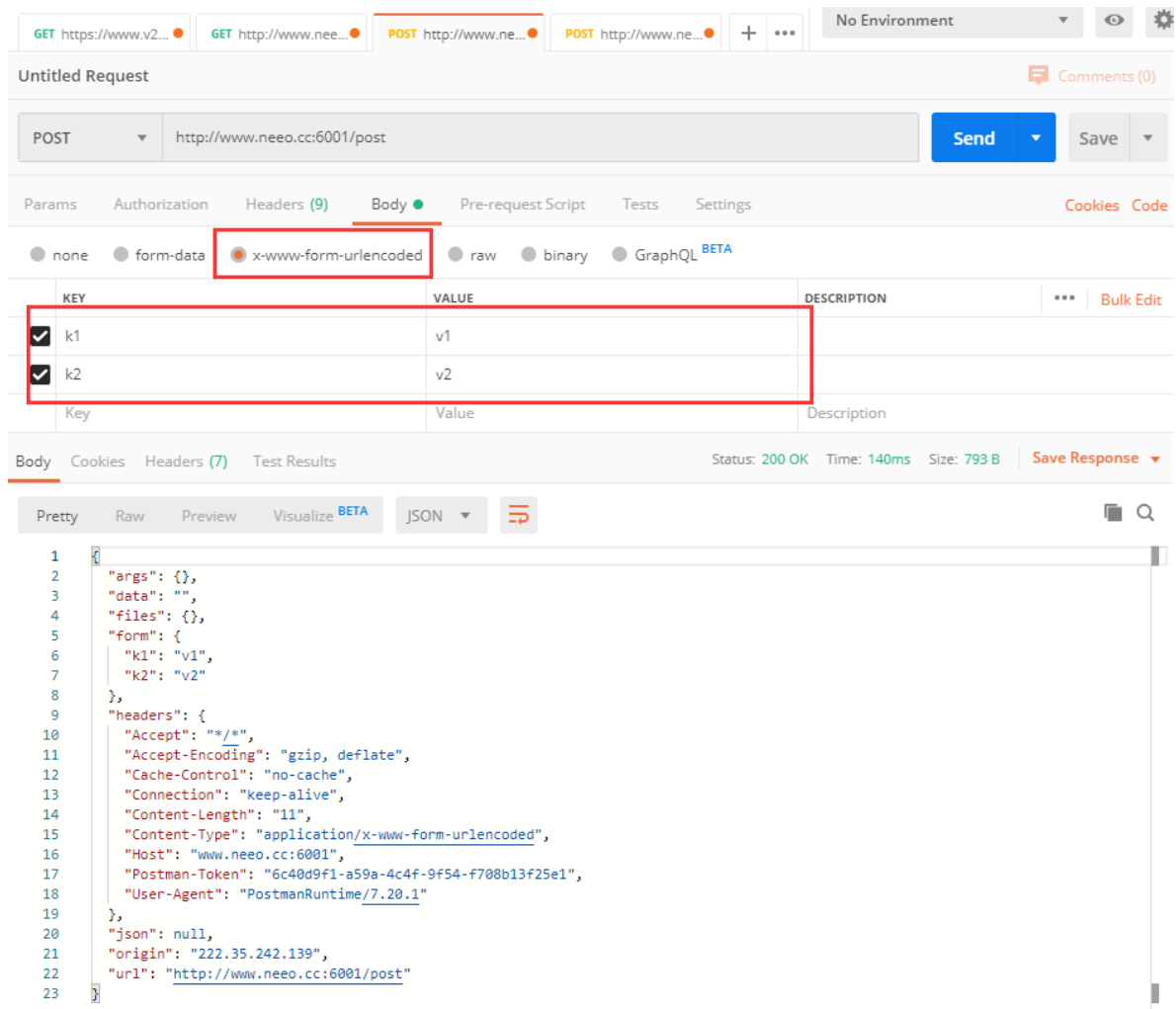
post请求一般由两种携带参数的方式。一种参数形式为 `k:v` 类型，还有另一种是 `k:json` 类型的接口。

k:v形式的post请求

url: <http://www.neeo.cc:6001/post>

参数: k1:v1 k2:v2

如下图，我们请求类型选择是 `post`，那么选择 `body` 的哪个选项呢？对于 `k:v` 格式的参数，一般选择 `x-www-form-urlencoded`，然后填写参数即可，其他的配置暂无。



参数为json的post请求

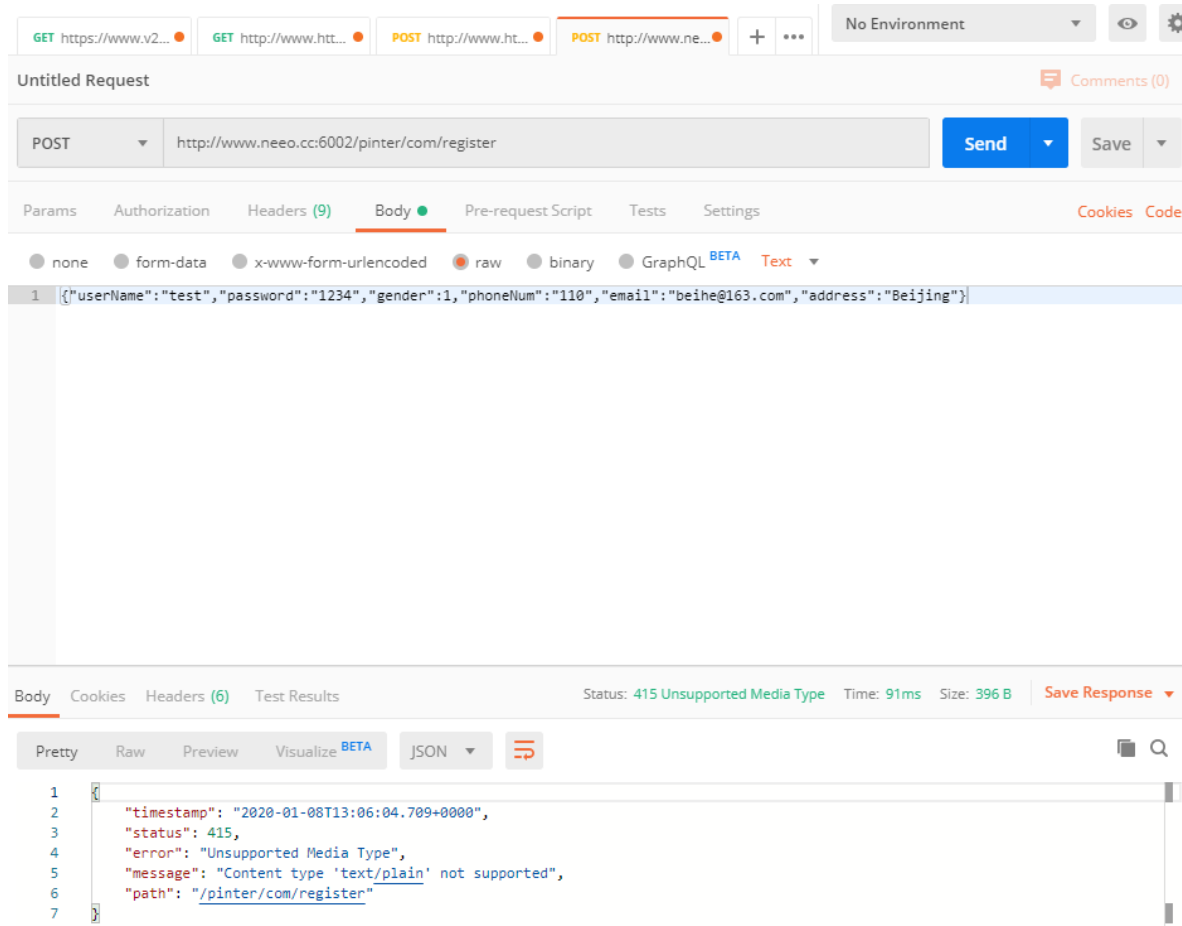
另一种形式的参数就是 `k:json` 的形式。

url: <http://www.neeo.cc:6002/pinter/com/register>

参数:

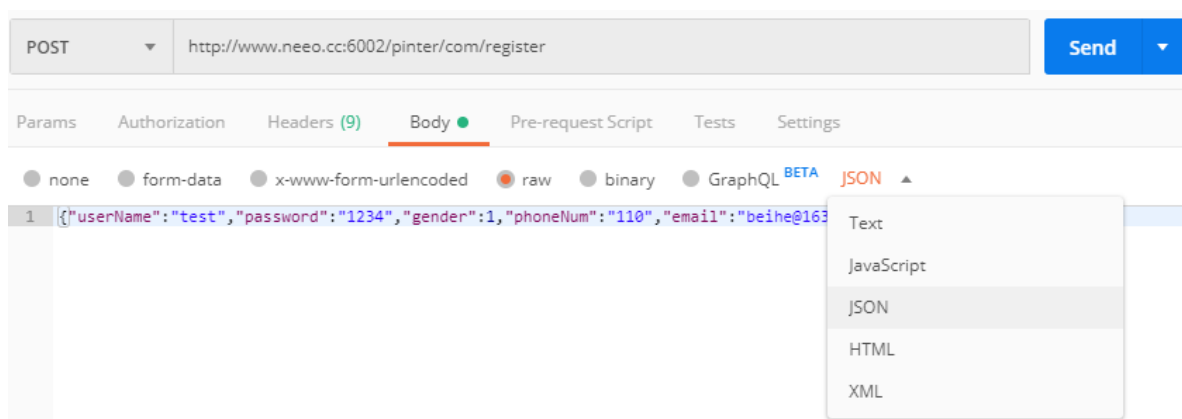
```
{"userName": "test", "password": "1234", "gender": 1, "phoneNum": "110", "email": "beih  
e@163.com", "address": "Beijing"}
```

json类型的参数，我们选择放在 `body` 中，然后选择 `raw` 原生的，现在点击 `Send` 发请求即可。

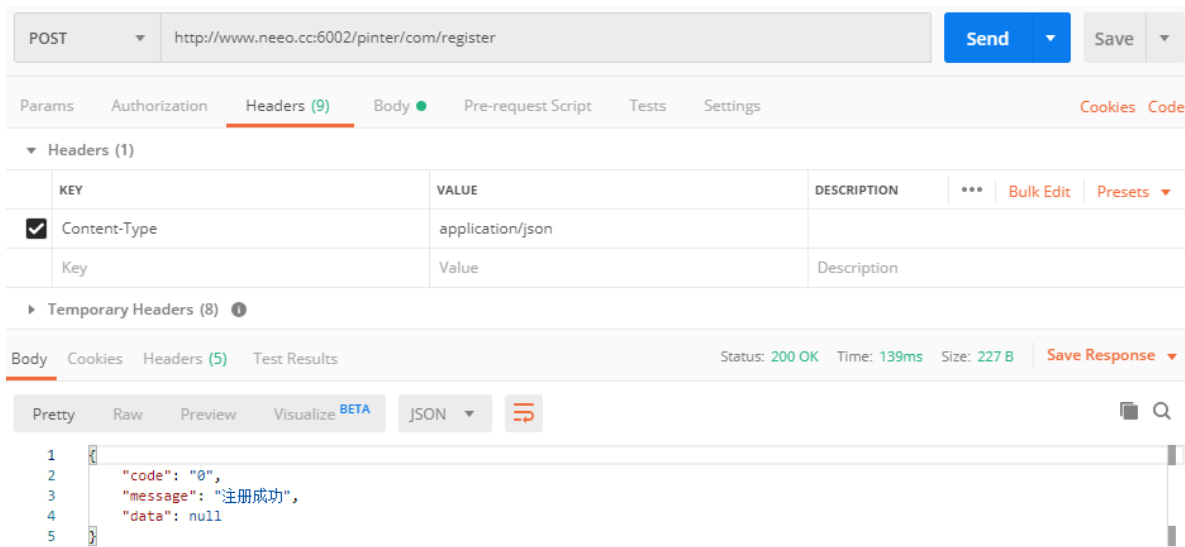


然后你会发现响应结果是有些问题的 `"error": "Unsupported Media Type"` 说是，不支持的媒体类型这是怎么回事呢？是因为在这种json类型的请求中，headers中需要携带一个特殊的请求头 `Content-Type:application/json`，这样，服务端才知道你携带的参数是json类型的数据，而不是普通的k:v格式的参数。

这里我们只需要将原来的 `Text` 替换为 `JSON` 即可。



然后postman会自动的在请求头中携带上 `Content-Type:application/json`，现在我们再次点击 `Send` 重新尝试。



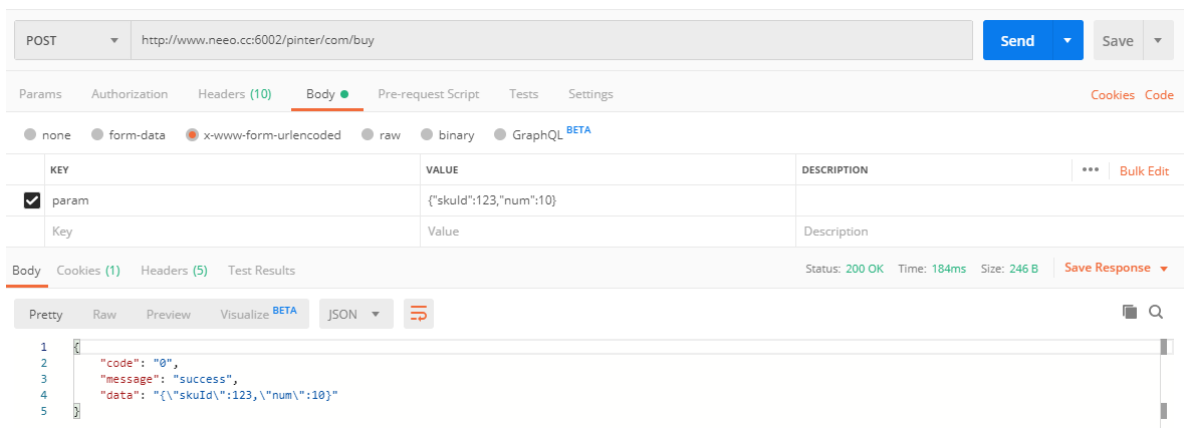
OK了!

除此之外，还有一种情况是 k:json 的post请求。我们简单来看下。

接口参数：

```
url:http://www.nneo.cc:6002/pinter/com/buy
类型： POST
参数： param={"skuId":123,"num":10}
```

来看postman中如何用？



没什么好说的，非常的简单。

webservice接口

还有一种接口就是webservice类型的接口。

简单来说，webservice是通过xml进行交互的web请求。

Web Service也叫XML Web Service WebService是一种可以接收从Internet或者Intranet上的其它系统中传递过来的请求，轻量级的独立的通讯技术。是通过SOAP在Web上提供的软件服务，使用WSDL文件进行说明，并通过UDDI进行注册。

XML: (Extensible Markup Language)扩展型可标记语言。面向短期的临时数据处理、面向万维网络，是Soap的基础。

Soap: (Simple Object Access Protocol)简单对象存取协议。是XML Web Service 的通信协议。当用户通过UDDI找到你的WSDL描述文档后，他通过可以SOAP调用你建立的Web服务中的一个或多个操作。SOAP是XML文档形式的调用方法的规范，它可以支持不同的底层接口，像HTTP(S)或者SMTP。

WSDL: (Web Services Description Language) WSDL 文件是一个 XML 文档，用于说明一组 SOAP 消息以及如何交换这些消息。大多数情况下由软件自动生成和使用。

UDDI (Universal Description, Discovery, and Integration) 是一个主要针对Web服务供应商和使用者的新项目。在用户能够调用Web服务之前，必须确定这个服务内包含哪些商务方法，找到被调用的接口定义，还要在服务端来编制软件，UDDI是一种根据描述文档来引导系统查找相应服务的机制。UDDI利用 SOAP消息机制（标准的XML/HTTP）来发布，编辑，浏览以及查找注册信息。它采用XML格式来封装各种不同类型的数据，并且发送到注册中心或者由注册中心来返回需要的数据。

可以通过[soutui](#)来测试，也可以通过jmeter来测试。

常用webservice接口网站：

<http://wcf.open.cnblogs.com/news/help/operations/GetNewsList#response-xml>

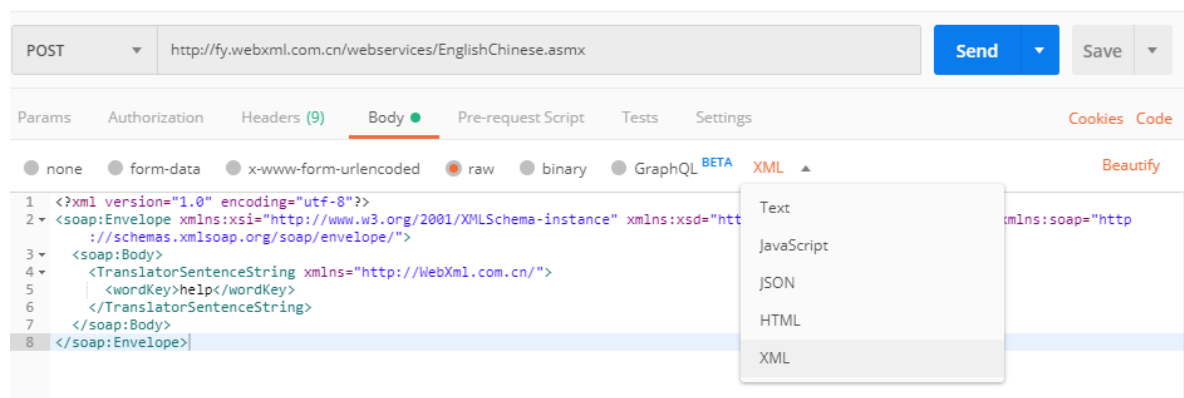
http://www.webxml.com.cn/zh_cn/index.aspx # 这个网站提供了很多webservice接口

url和相关参数：

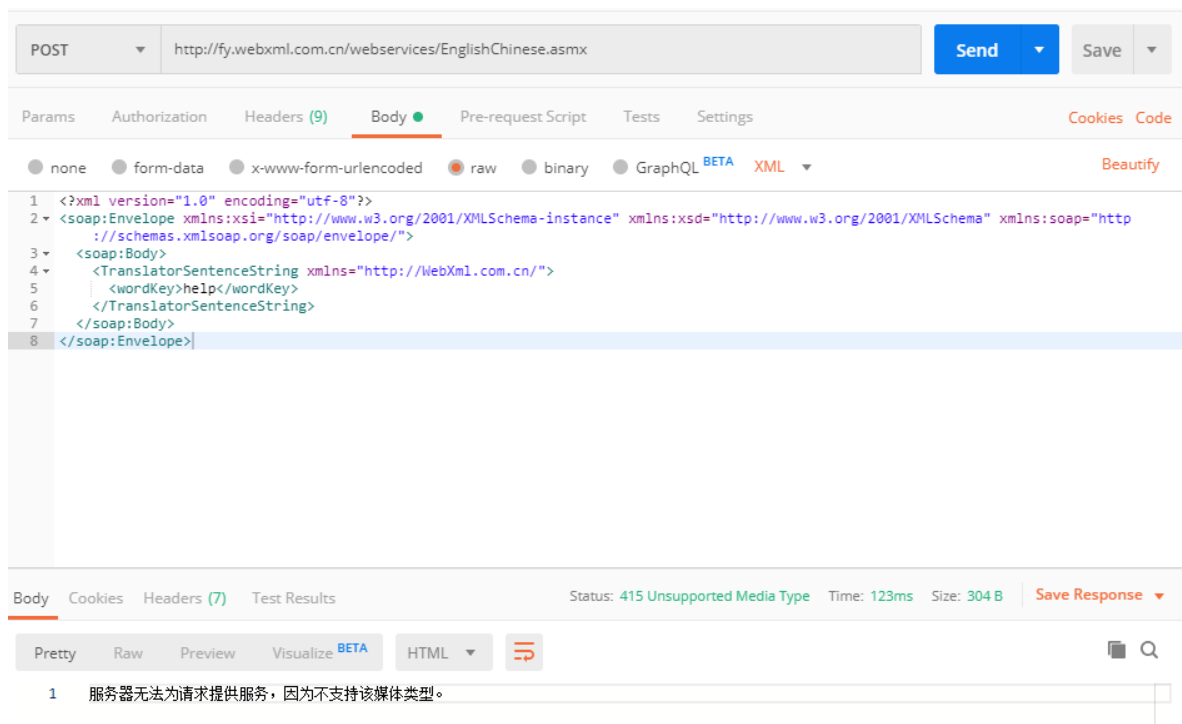
```
# http://fy.webxml.com.cn/webservices/EnglishChinese.asmx
# post类型

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <TranslatorSentenceString xmlns="http://WebXml.com.cn/">
      <wordKey>string</wordKey>
    </TranslatorSentenceString>
  </soap:Body>
</soap:Envelope>
```

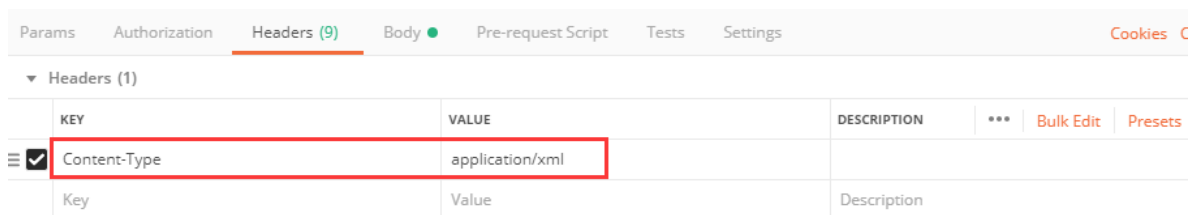
在 Body 中选择 raw，类型选择 XML，参数中，需要将 string 替换为实际的单词。



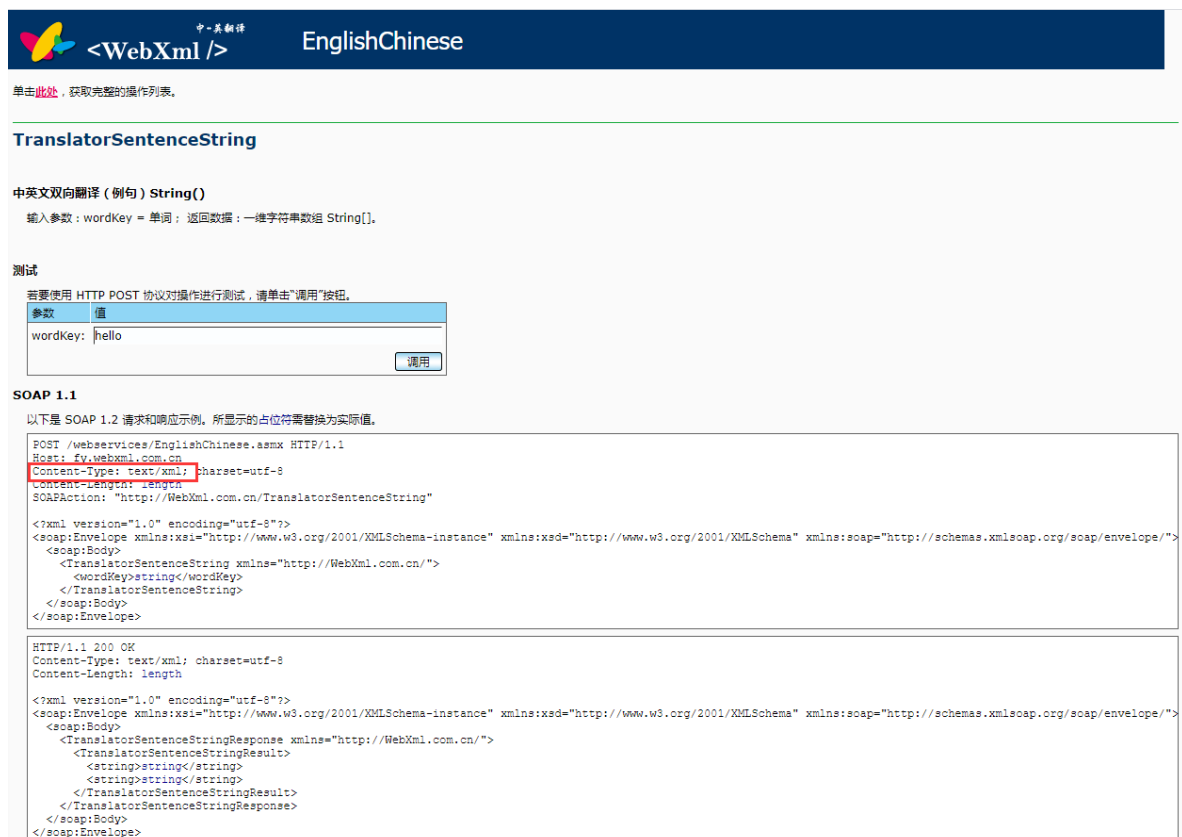
现在，点击 Send 发送请求。



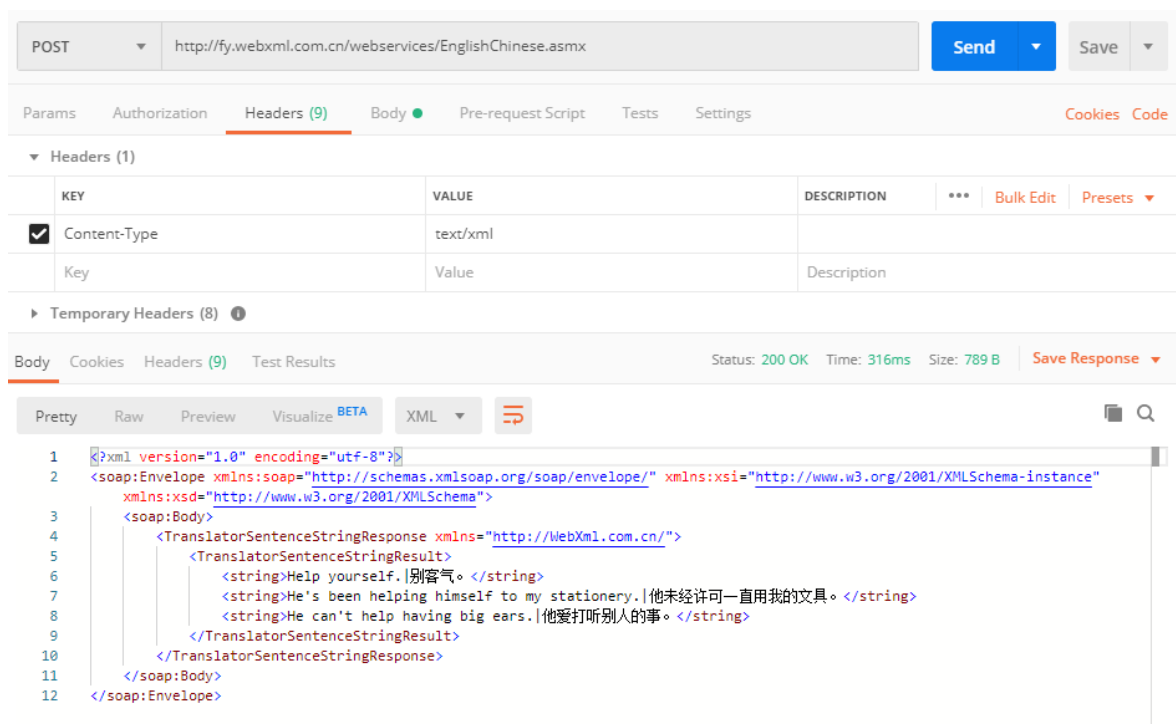
遇到了问题，响应状态码是415，不支持的媒体类型，这是怎么回事？原因是，当我们选择类型为XML时，postman会自动的在headers中添加 Content-Type: application/xml。



原因就是出在 Content-Type 这里，类型自动添加的不对，那到底是什么呢？我们看人家网站的说明：



是 Content-Type: text/xml，我们在postman中手动修改，然后再发送请求即可。



现在，这个接口就通过了。

文件上传接口测试

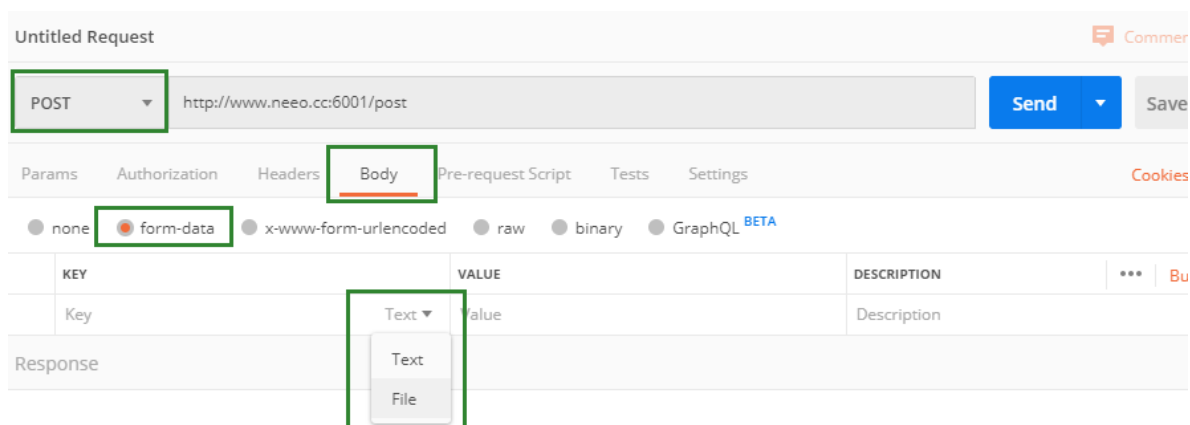
url: <http://www.nneo.cc:6001/post>

类型: POST

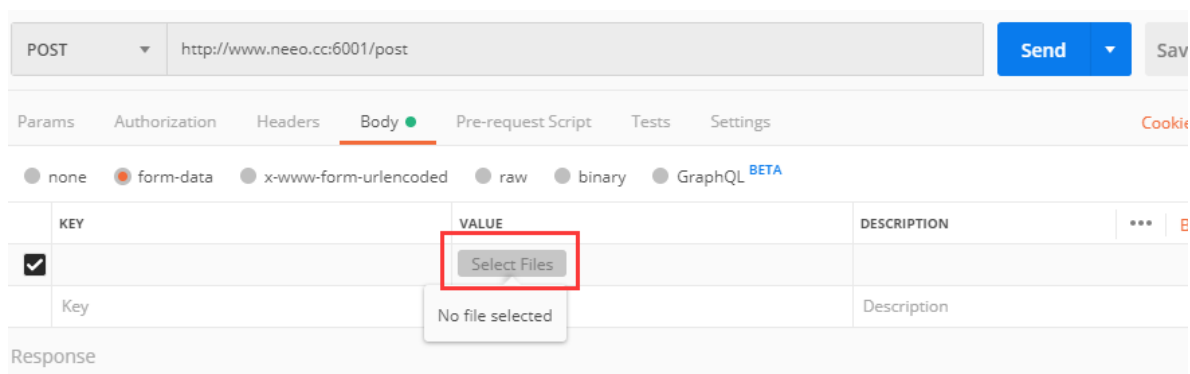
文件: 普通的图片

我们来配置上传图片的参数。

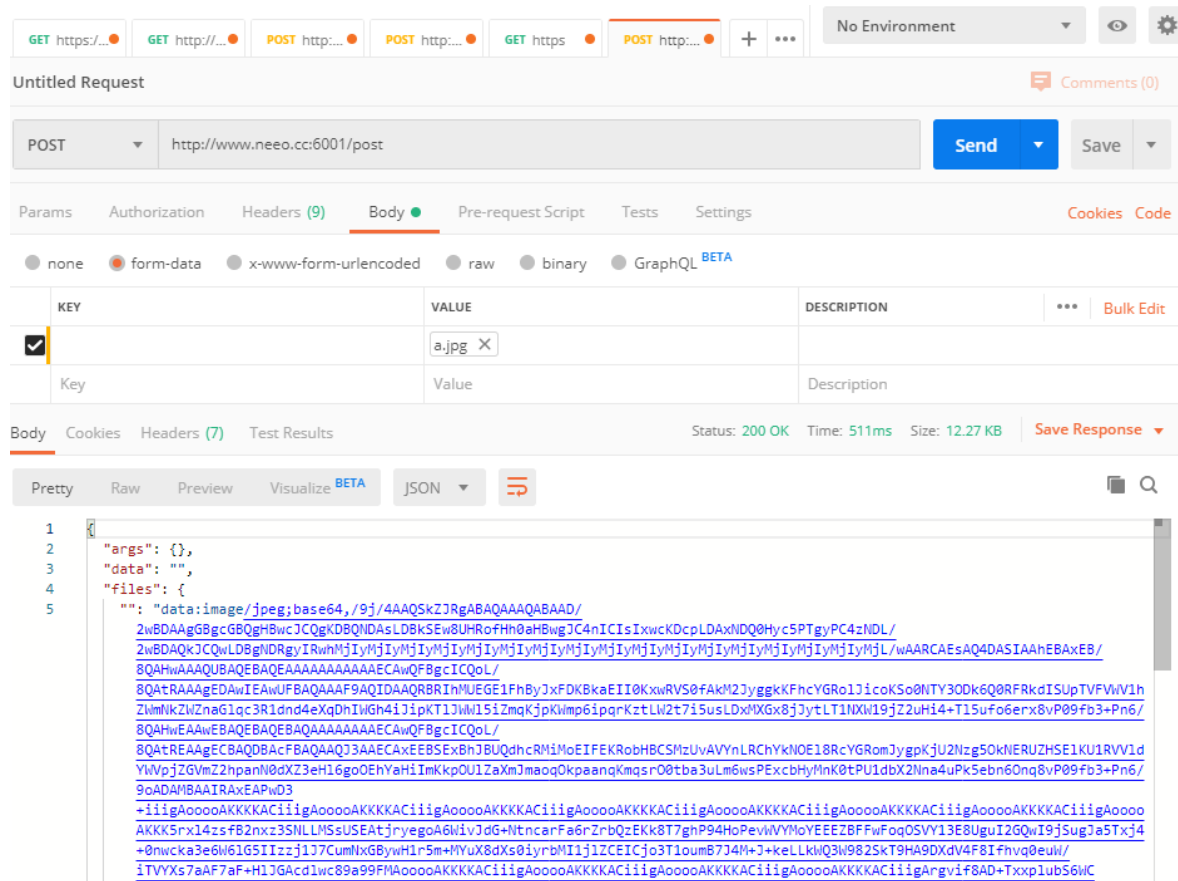
首先，在 Body 中选择 form-data 选项，悬浮到 key 选择 File。



然后点击 Select Files 从本地上传文件。



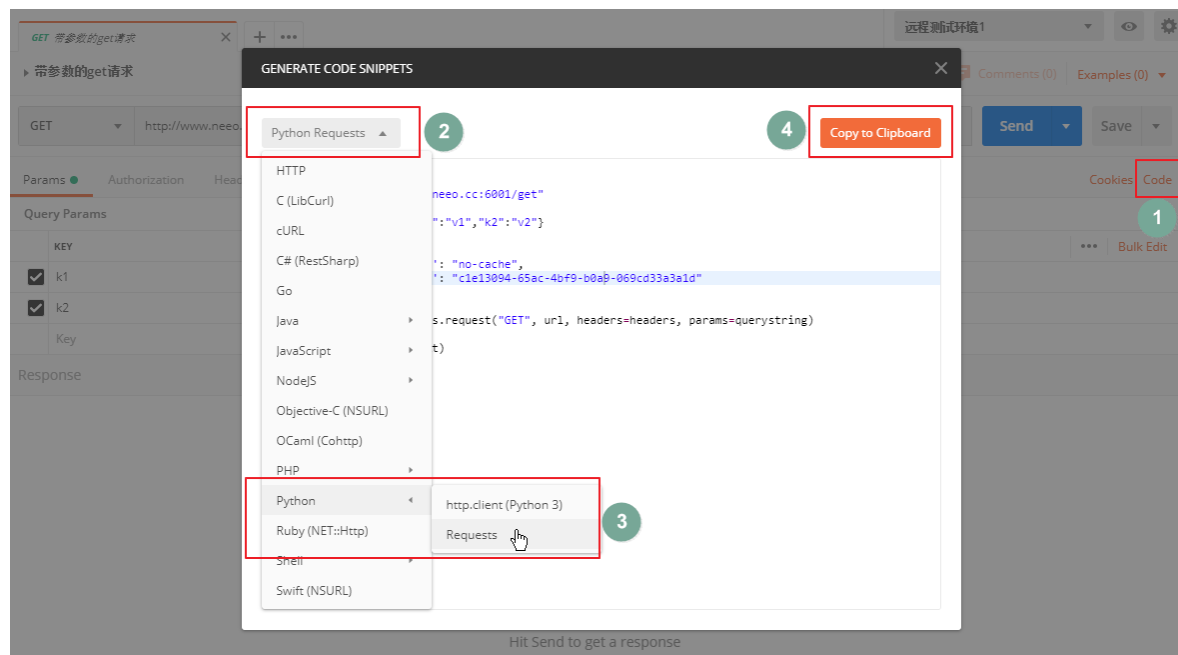
然后点击 `send` 发送请求即可。



将接口生成代码

postman同样提供了，将接口转换为各语言的可执行脚本，比如转为Python、Go、Java等语言。

比如导出为Python脚本。



第一步，点击 `code`。

第二步，选择语言。

第三步，选择导出类型。

第四步，点击拷贝到剪贴板。

其实，还有第五步，本地新建一个 `py` 脚本，将拷贝内容复制进去，然后你就可以愉快的玩耍了。

集合 (Collection)

集合，可以将它理解一个项目，把所有属于该项目的接口，放到同一个集合中，便于管理。

新建一个集合

在左侧的菜单栏（如果该菜单栏隐藏的话，点击顶部菜单栏 `View --> Toggle Sidebar` 即可），可以看到有个 `Collections` 选项，我们点击 `New Collection`，新建一个集合，如下图，为集合起个名，然后点击右下角的 `Create` 即可。

CREATE A NEW COLLECTION

Name

postman基础操作

Description

Authorization

Pre-request Scripts

Tests

Variables

This description will show in your collection's documentation, along with the descriptions of its folders and requests.

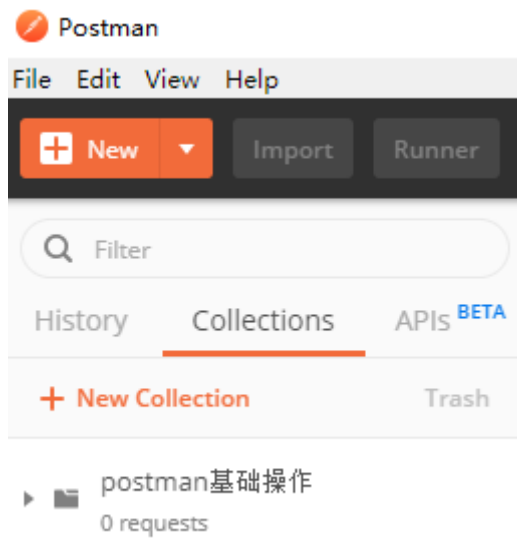
Make things easier for your teammates with a complete request description.

Descriptions support **Markdown**

Cancel

Create

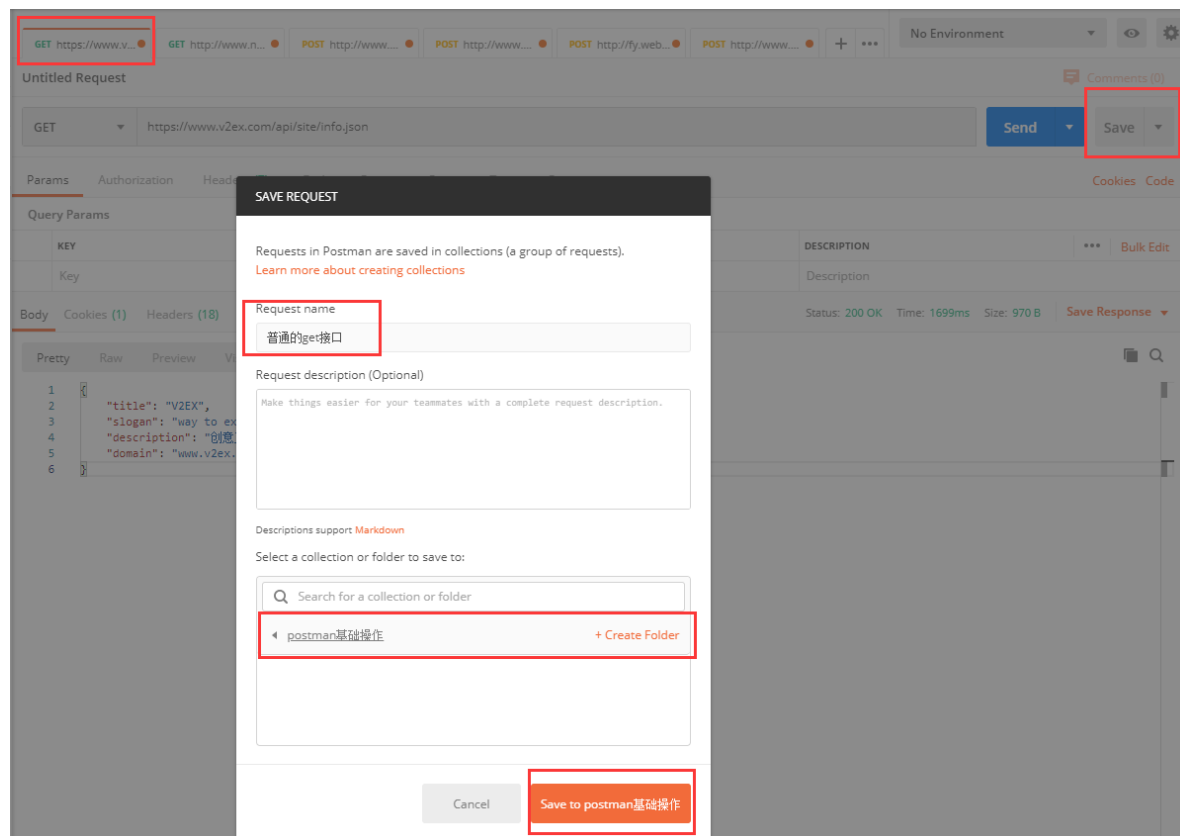
然后，在左侧菜单栏的 `Collections` 中就会出现一个类似文件夹的集合，就是我们刚才创建的集合。



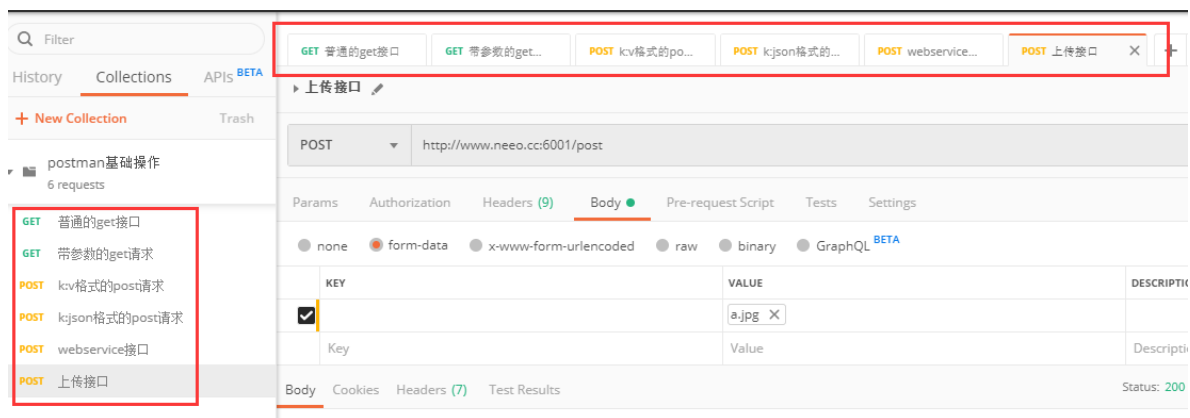
为集合添加接口用例

我们可以将之前的几个接口放到这个集合中。

首先，选择当前的接口，**Ctrl+S** 或者点击 **Send** 旁边的 **Save**，在弹出框中，为这个接口起个名，然后选择保存到指定的集合中，然后点击右下角的保存按钮即可。



其他的接口以此类推，添加完在集合中就可以看到了。



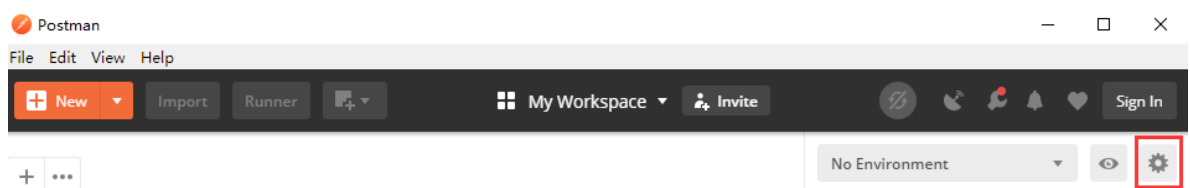
环境管理

我们知道，一个项目在不同的阶段会处于不同的环境中，比如开发阶段的开发环境、测试阶段的测试环境和线上环境，那在不同的阶段，做测试的时候，会遇到不同的环境问题，比如测试某个接口，开发环境是 `localhost:8080`，到了测试阶段，可能就变成了测试服务器的ip和端口.....然后我们会频繁的改接口的ip和端口，这将会非常的麻烦。而Postman帮我们解决了类似的问题。

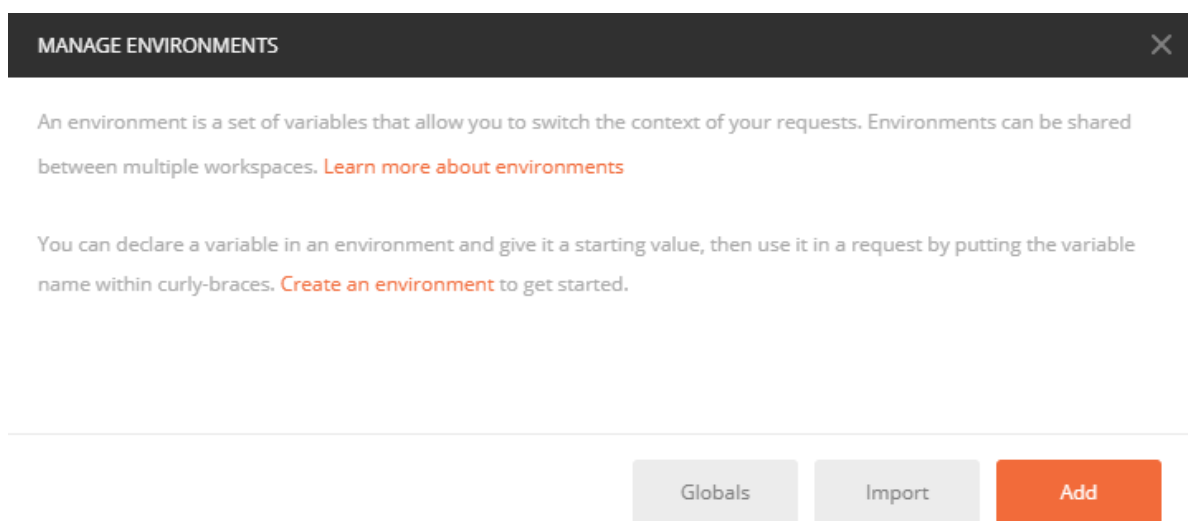
Postman支持多环境，同时每个环境中可以设置独立的参数。

创建一个新的环境

如下图，点击 设置 按钮。



点击 Add 添加一个环境。



为添加的环境起个名，然后设置一些变量，然后点击 Add。

MANAGE ENVIRONMENTS

Add Environment

本地环境

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	...	Persist All	Reset All
<input checked="" type="checkbox"/>	ip	127.0.0.1	127.0.0.1			X ...
<input checked="" type="checkbox"/>	port	8080	8080			

CancelAdd

现在，已经有了一个本地的环境，此时，如果继续点击右下角的 **Add** 将会继续添加环境，退出则点击右上角的叉号即可。

MANAGE ENVIRONMENTS

An environment is a set of variables that allow you to switch the context of your requests. Environments can be shared between multiple workspaces. [Learn more about environments](#)

本地环境

Share

Import

Globals

Import

Add

我们选择继续添加两个线上环境。

MANAGE ENVIRONMENTS

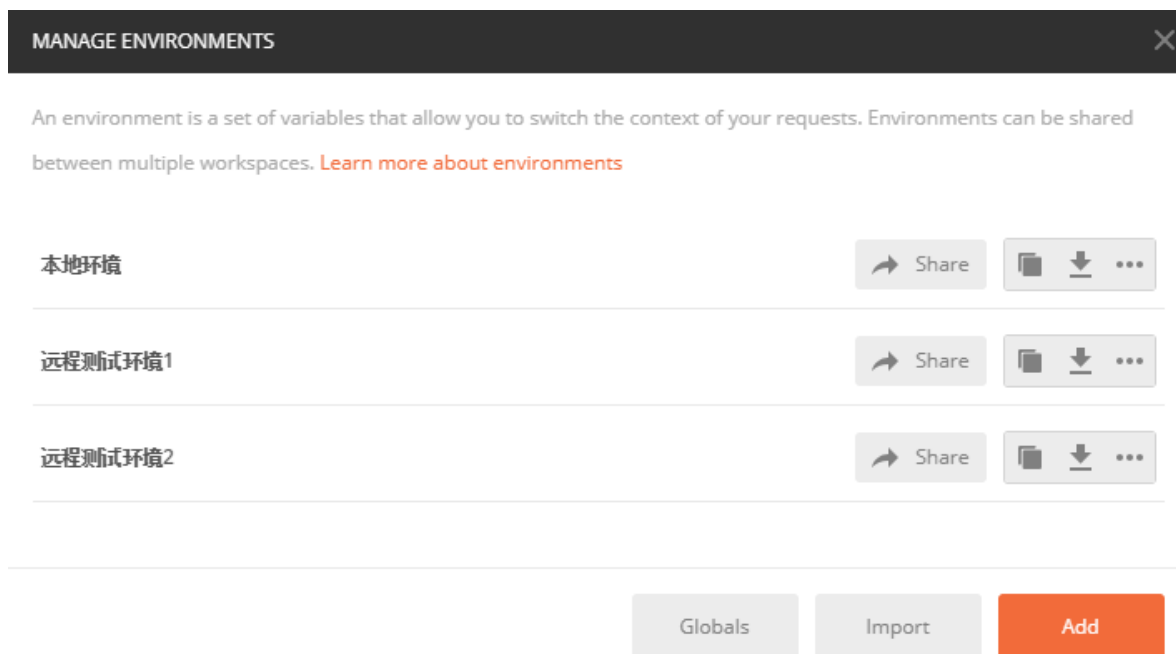
Add Environment

远程测试环境2

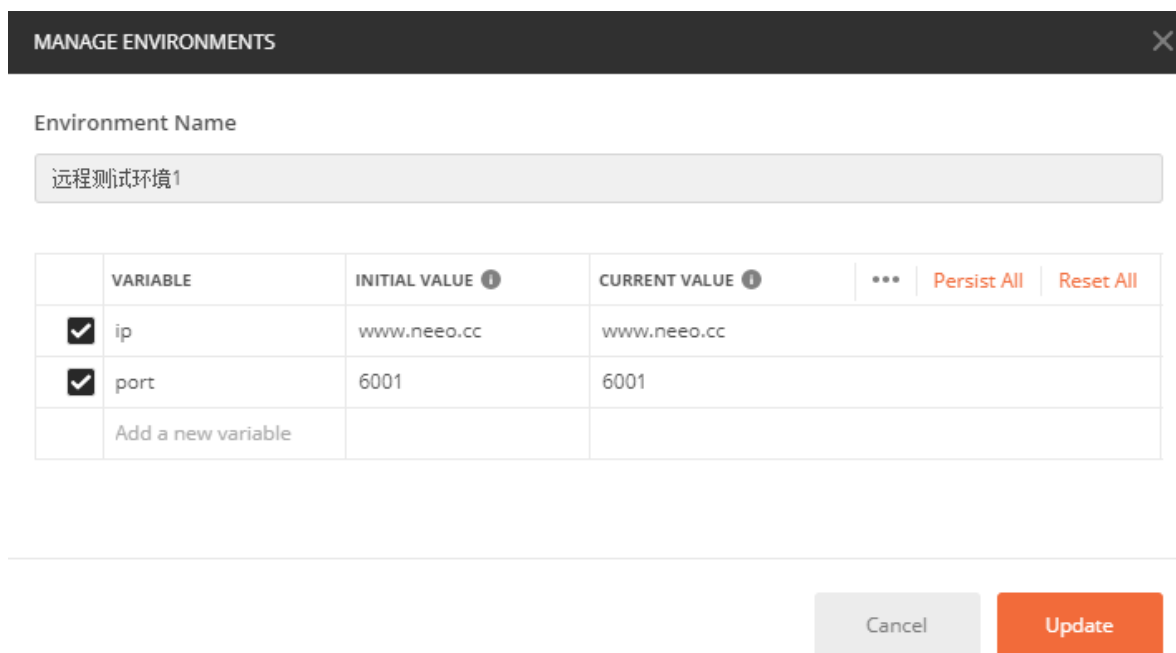
	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	...	Persist All	Reset All
<input checked="" type="checkbox"/>	ip	www.neeo.cc	www.neeo.cc			
<input checked="" type="checkbox"/>	port	6002	6002			

CancelAdd

这样，我们就有了3个环境可用。



点击上图中的三点，也可以选择删除或者，点击环境名称选择更新环境中的变量值。比如下图，我们选择更改一个环境。

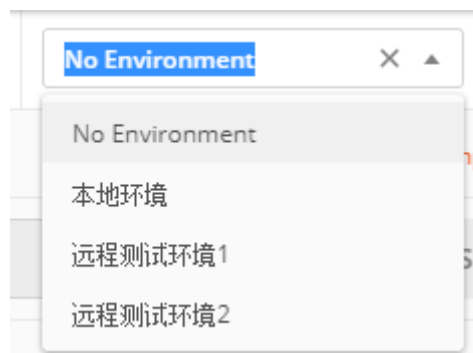


那么，怎么用呢？比如我们修改一个接口示例，来应用上创建的环境。

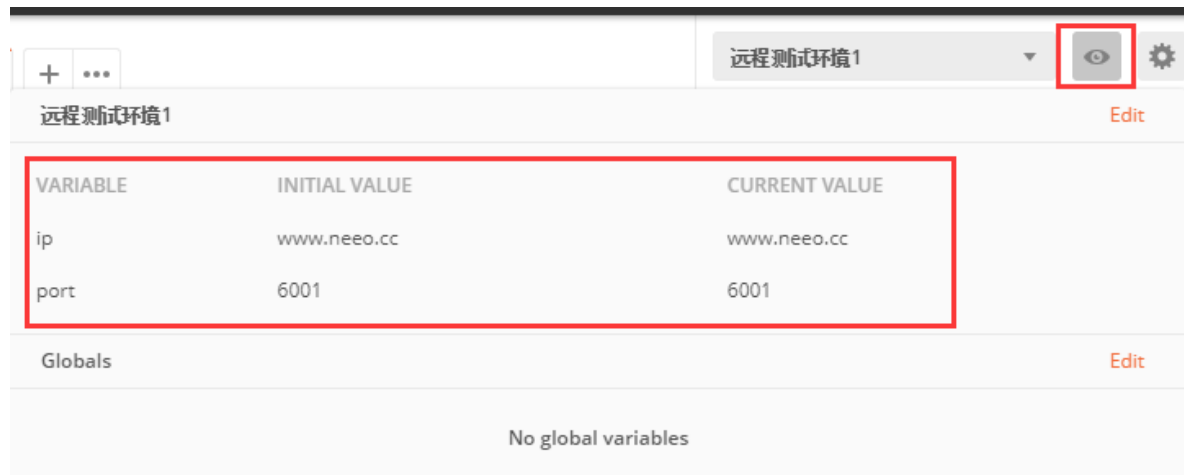
使用环境配置

现在，我们找一个接口来使用上创建的环境。

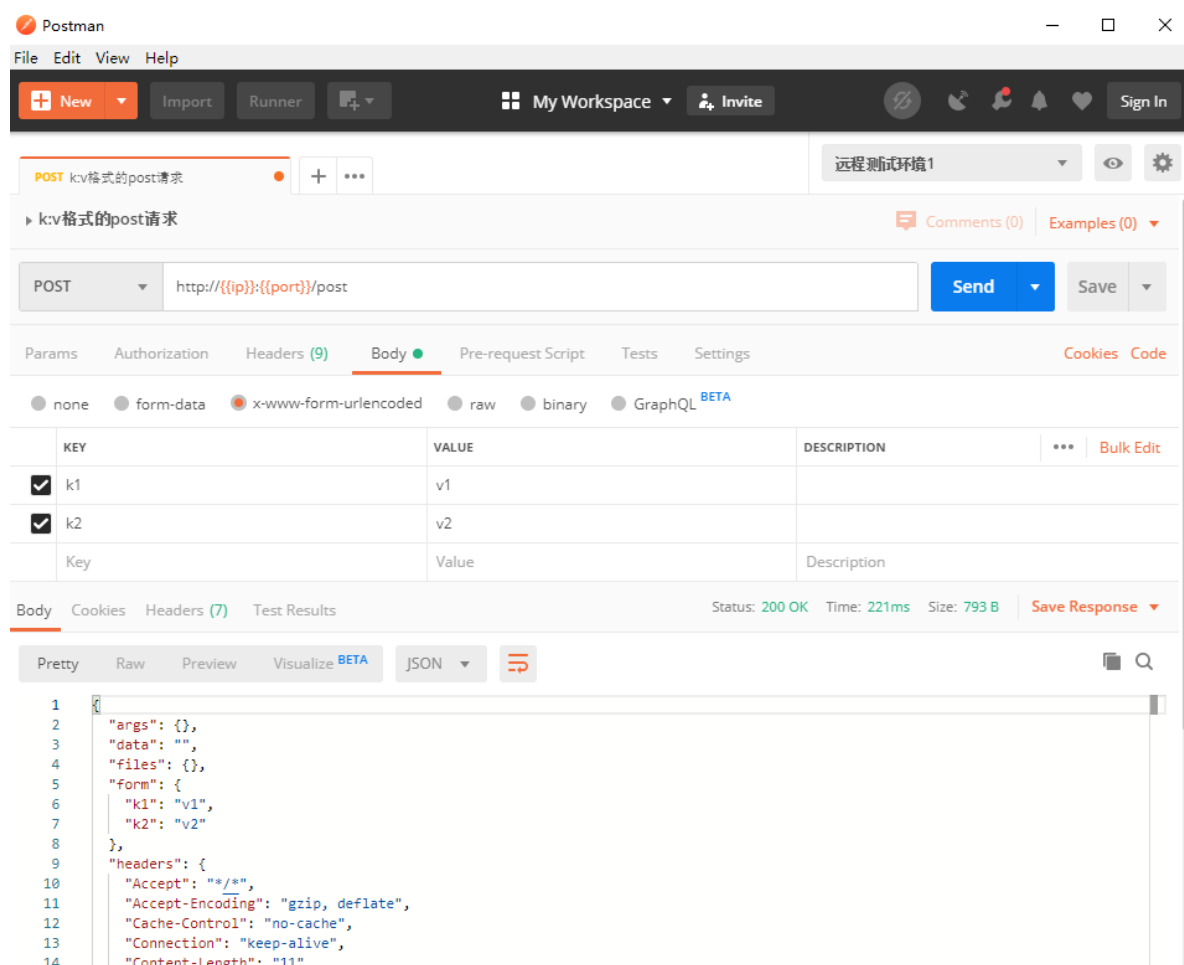
首先，选择 远程测试环境1。



我们也可以点击右侧的眼睛图标查看该环境的相关参数。



现在，在接口的url中使用上上面的两个参数。就是将需要修改的ip和端口，使用 {{变量名}} 代替，如下图。



现在，就ok啦。

这样，将一些动态的参数，设置为变量，我们应用变量即可。

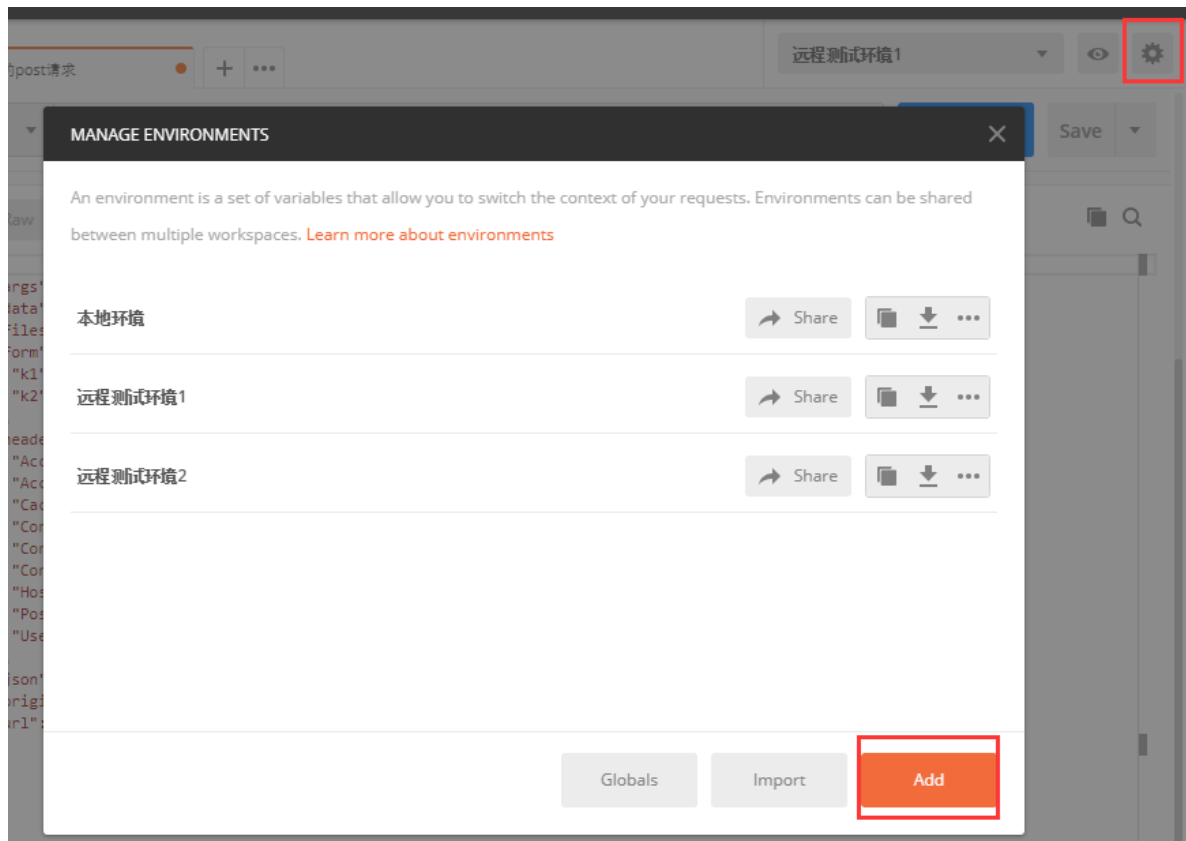
变量的作用域

postman中可以在环境、集合中定义变量，然后使用 `{{变量名}}` 的方式调用变量。一般，有下面三种情况：

- 环境变量，作用域为当前环境。
- 集合变量，作用域为当前集合。
- 全局变量，作用域为任何地方。

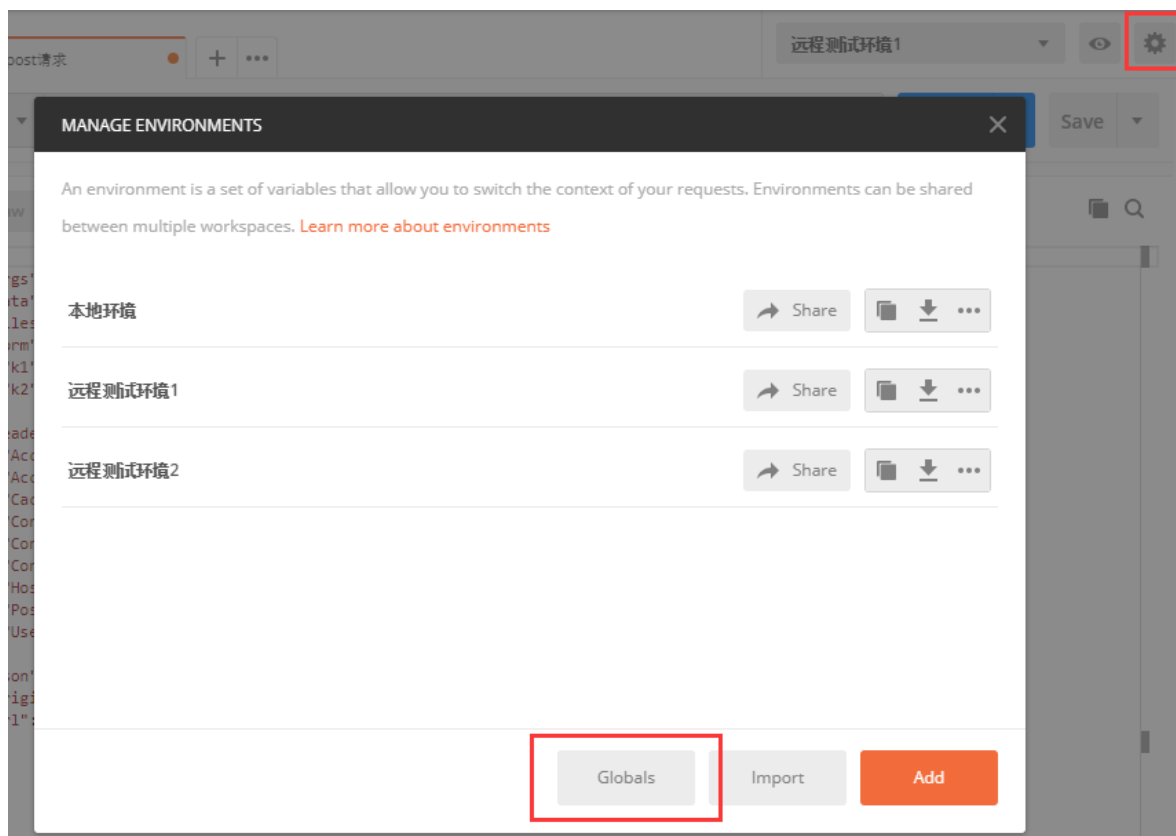
环境变量

无需多言，我们在之前创建的环境中定义的变量就是属于环境变量。点击 `设置` 图标点击 `Add` 就是在创建环境变量。

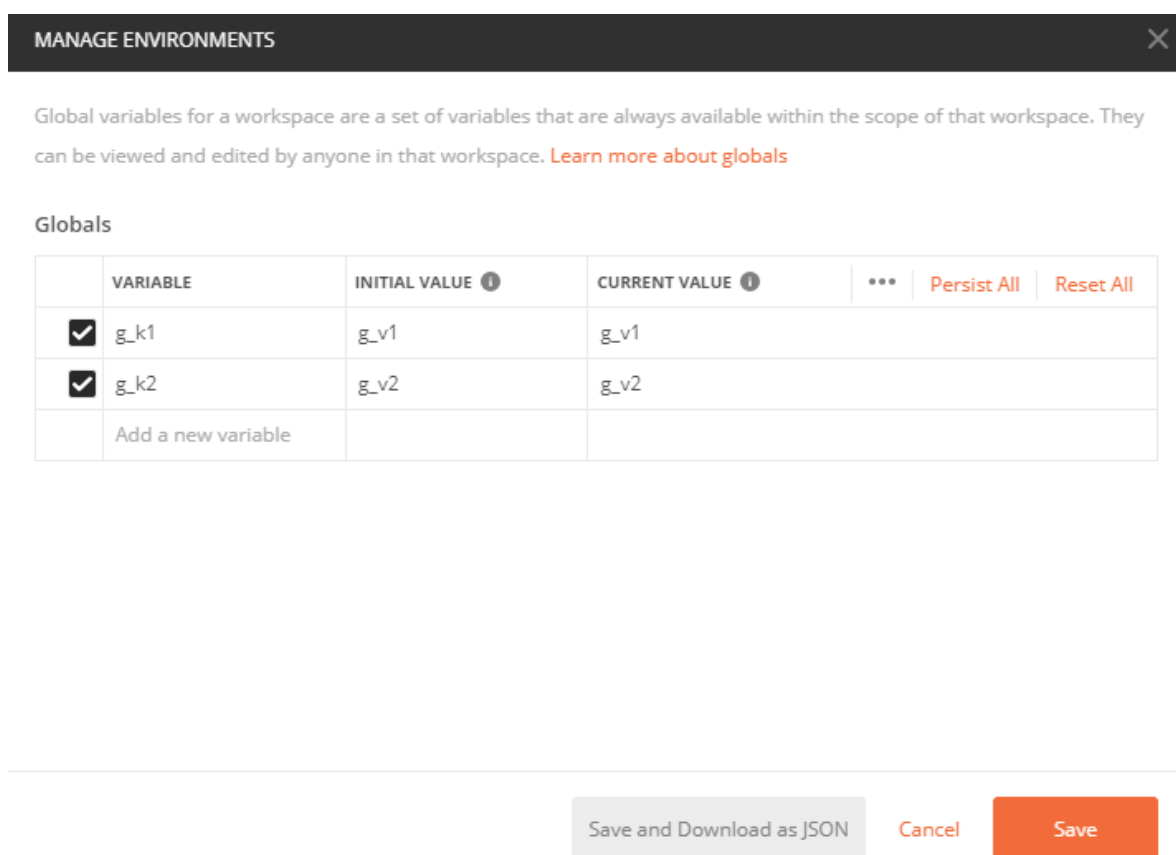


全局变量

如上图，还是点击 `设置` 图标，全局变量点击 `Globals` 来创建。



如下图，创建完相关变量后，点击 **Save** 即可。



我们可以在环境变量中查看到全局的变量。

远程测试环境1		
VARIABLE	INITIAL VALUE	CURRENT VALUE
ip	www.nneo.cc	www.nneo.cc
port	6001	6001
Globals		
VARIABLE	INITIAL VALUE	CURRENT VALUE
g_k1	g_v1	g_v1
g_k2	g_v2	g_v2

在接口中也是直接使用 {{变量名}} 引用即可。

POST kv格式的post请求

远程测试环境1

POST

http://{{ip}}:{{port}}/post

Send

Save

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Cookies

Code

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL BETA

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> k1	v1	
<input checked="" type="checkbox"/> k2	{{g_k2}}	
Key	Value	Description

Body

Cookies

Headers (7)

Test Results

Status: 200 OK Time: 1804ms Size: 795 B Save Response

Pretty

Raw

Preview

Visualize BETA

JSON

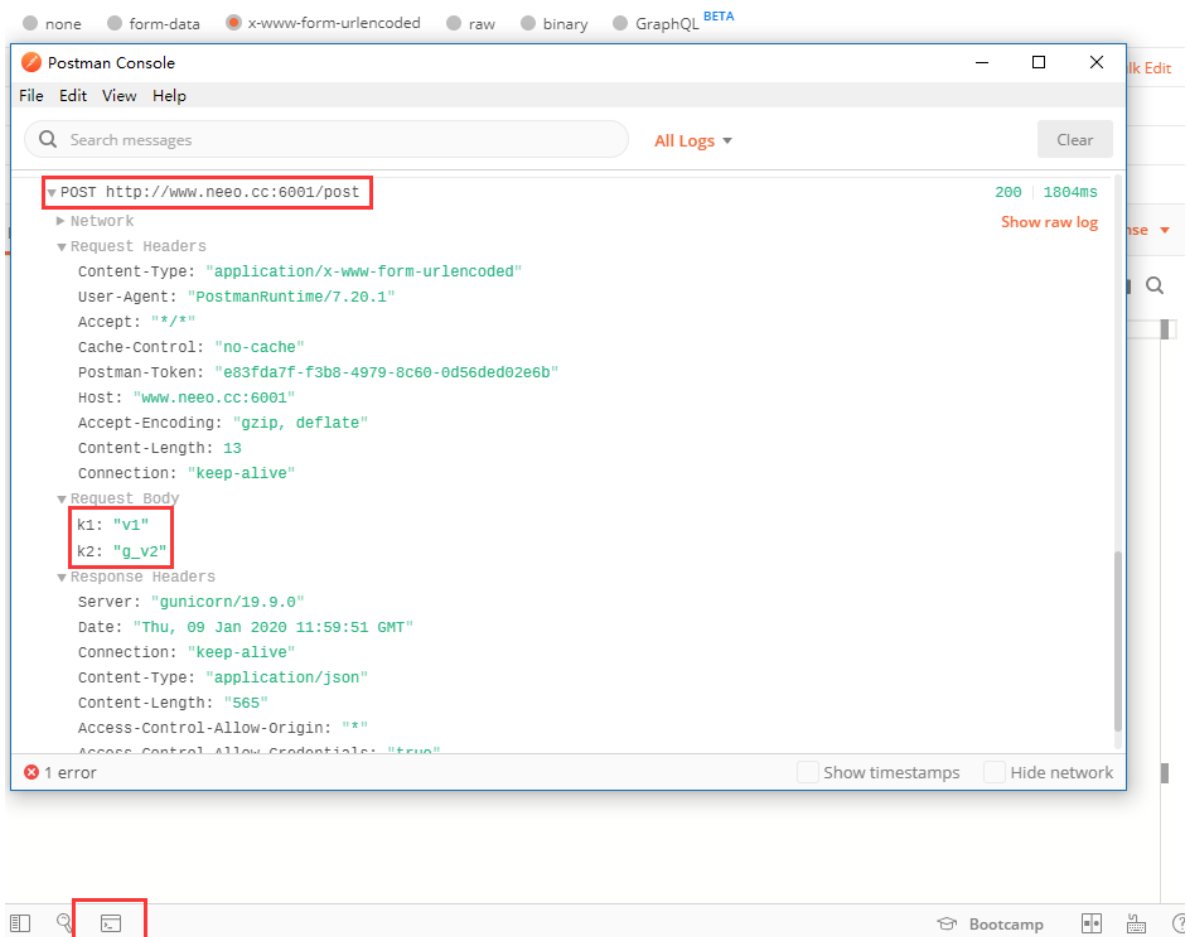
```

1  {
2    "args": {},
3    "data": "",
4    "files": {},
5    "form": {
6      "k1": "v1",
7      "k2": "g_v2"
8    },
9    "headers": {
10     "Accept": "*//*",
11     "Accept-Encoding": "gzip, deflate",
12     "Cache-Control": "no-cache",
13     "Connection": "keep-alive",
14     "Content-Length": "13",
15     "Content-Type": "application/x-www-form-urlencoded",
16     "Host": "www.nneo.cc:6001",
17     "Postman-Token": "e83fda7f-f3b8-4979-8c60-0d56ded02e6b",
18     "User-Agent": "PostmanRuntime/7.20.1"
19   },
20   "json": null,
21   "origin": "222.35.242.139",
22   "url": "http://www.nneo.cc:6001/post"
23 }

```

我们也可以在postman的控制台中查看。

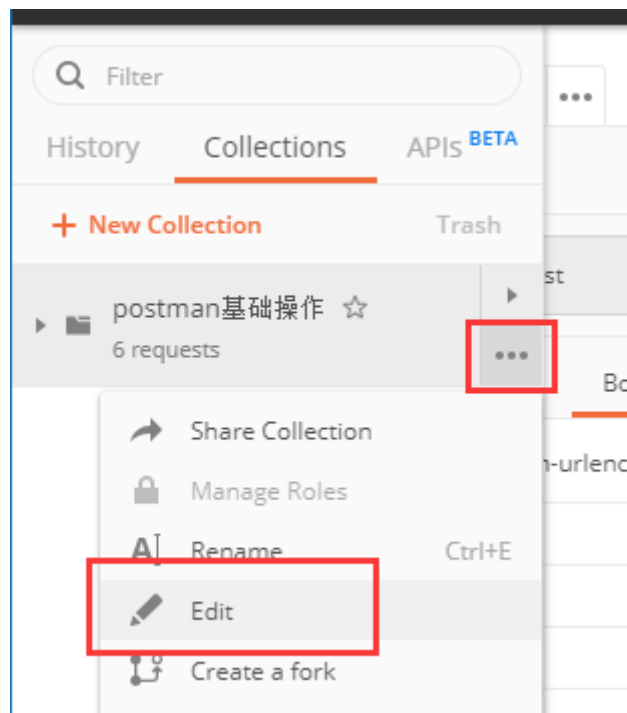
点击postman的左下角的 Postman Console 图标，即可看到当前请求url，点进去查看详细情况。



集合变量

除了全局和环境变量，我们也可为集合单独设置自己的变量。

如下图，选择 Edit。



然后选择 variables 添加变量，然后点击 update。

EDIT COLLECTION

Name

postman基础操作

Description

Authorization

Pre-request Scripts

Tests

Variables

These variables are specific to this collection and its requests. [Learn more about collection variables.](#)

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	...	Persist All	Reset All
<input checked="" type="checkbox"/>	c_k3	c_v3	c_v3			
	Add a new variable					

Cancel

Update

然后我们就可以在请求中引用了。

POST kv格式的post请求

+

...

远程测试环境1

👁

⚙

▶ kv格式的post请求

Comments (0)

Examples (0) ▼

POST

http://{{ip}}:{{port}}/post

Send ▼

Save ▼

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Cookies

Code

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL BETA

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	k1	v1			
<input checked="" type="checkbox"/>	k2	{{g_k2}}			
<input checked="" type="checkbox"/>	k3	{{c_k3}}			
	Key	Value	Description		

Body

Cookies

Headers (7)

Test Results

Status: 200 OK Time: 1660ms Size: 814 B Save Response ▼

Pretty

Raw

Preview

Visualize BETA

JSON ▼

🔍

```
1 {
2   "args": {},
3   "data": "",
4   "files": {},
5   "form": {
6     "k1": "v1",
7     "k2": "g_v2",
8     "k3": "c_v3"
9   }
10 }
```

内置动态变量

postman中还有一些内置的动态变量，一般用作于动态参数化。应用在哪些呢？比如我们测试一个注册接口，你每次访问携带的用户名和密码都不应该是固定的。

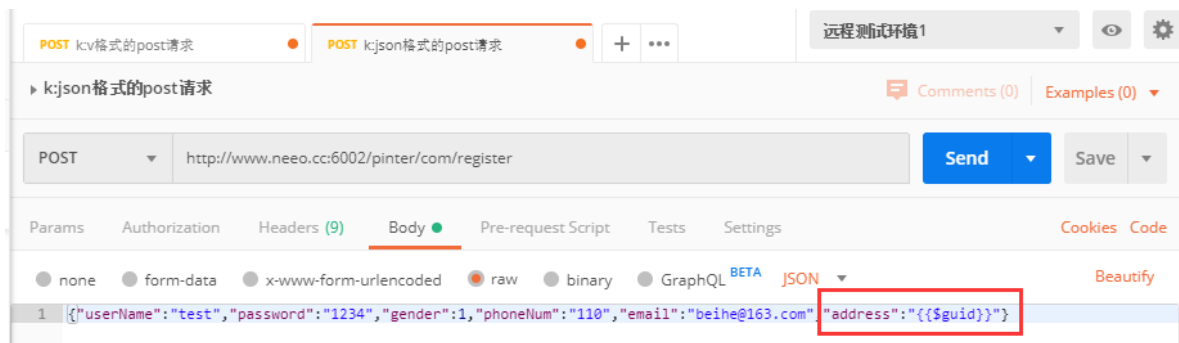
比如常用的变量：

- `$guid`，唯一字符串，类似于uuid
- `$timestamp`，时间戳

随机类变量 `$random` 系列，比如 `$randomUUID`、`$randomInt` 等等，更多内置的变量，参考：

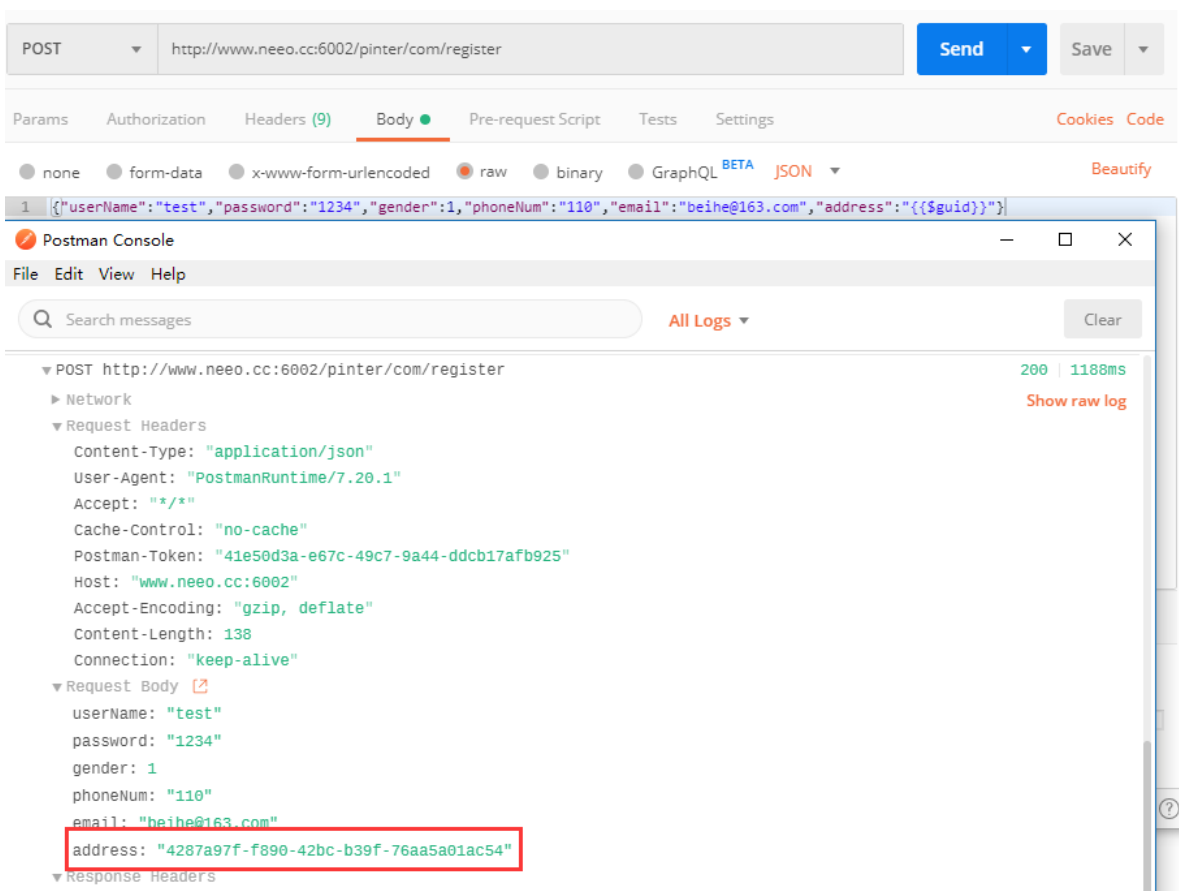
<https://learning.getpostman.com/docs/postman/scripts/postman-sandbox-api-reference/#dynamic-variables>

我们来访问注册接口：`http://www.nneo.cc:6002/pinter/com/register`



如上图，我们将携带的json参数中的 `address` 值使用唯一字符串来替代。语法就是 `{{$guid}}`。

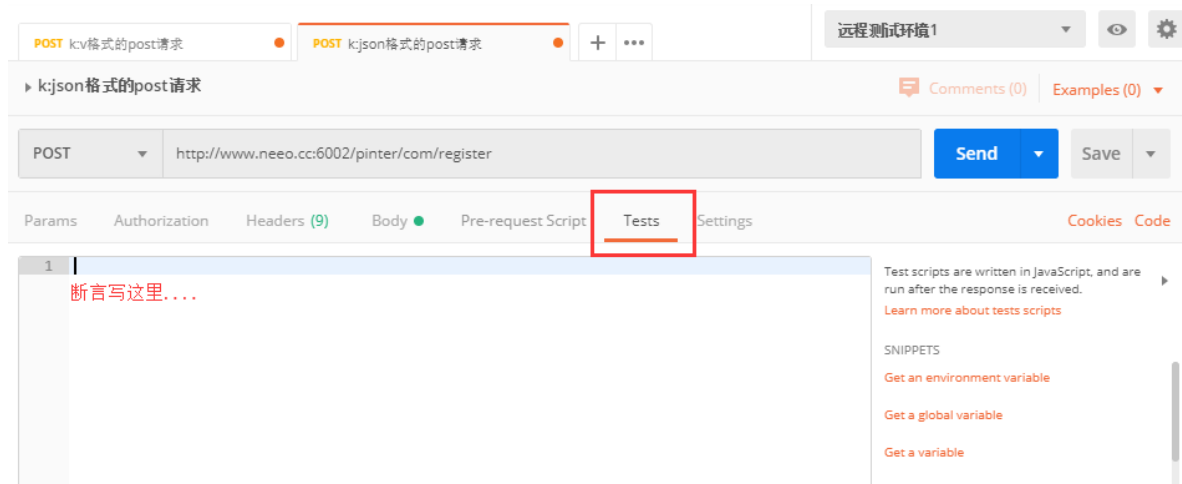
我们从 `postman console` 中来查看是否成功。



断言

postman同样支持断言功能，对于接口的断言，一般来判断接口返回的结果是否符合预期。

在postman中，使用 `Test` 脚本来对接口数据进行判断，`Test` 脚本会在接口响应后执行。



Test 脚本基于JavaScript语言，使用postman自带的Test 模板（可以减少编写断言代码了），可以快速生成用例。

相关模板	描述	示例
clear a global variable	清除全局变量	<code>pm.globals.unset("variable_key");</code>
Clear an environment variable	清除一个环境变量	<code>pm.environment.unset("variable_key");</code>
get a global variable	得到一个全局变量	<code>pm.globals.get("variable_key");</code>
get a variable	得到一个变量	<code>pm.variables.get("variable_key");</code>
Get an environment variable	得到一个环境变量	<code>pm.environment.get("variable_key");</code>
response body:contains string	检查 response body包含字符串	<code>pm.test("Body matches string", function () { pm.expect(pm.response.text()).to.include("string_you_want_to_search"); });</code>
response body:convert XML body to a JSON object	response body: 将XML转换为JSON对象	<code>var jsonObject = xml2Json(responseBody);</code>
response body:is equal to a string	检查响应体等于一个字符串	<code>pm.test("Body is correct", function () { pm.response.to.have.body("response_body_string"); });</code>
response body:JSON value check	检查 response body中JSON某字段值	<code>pm.test("Your test name", function () { var jsonData = pm.response.json(); pm.expect(jsonData.value).to.eql(100); });</code>
response headers:content-Type header check	检查 content-Type是否包含在 header返回	<code>pm.test("Content-Type is present", function () { pm.response.to.have.header("Content-Type"); });</code>
response time is than 200ms	响应时间超过 200ms	<code>pm.test("Response time is less than 200ms", function () { pm.expect(pm.response.responseTime).to.be.below(200); });</code>
send s request	发送一个请求	<code>pm.sendRequest("https://postman-echo.com/get", function (err, response) { console.log(response.json()); });</code>
set a global variable	设置一个全局变量	<code>pm.globals.set("variable_key", "variable_value");</code>
set an environment variable	设置一个环境变量	<code>pm.environment.set("variable_key", "variable_value");</code>
status code:Code is 200	状态码: 代码是200	<code>pm.test("Status code is 200", function () { pm.response.to.have.status(200); });</code>
status code:code name has string	状态码: 代码中有指定字符串	<code>pm.test("Status code name has string", function () { pm.response.to.have.status("Created"); });</code>

相关模板	描述	示例
status code: successful POST request	状态码: 成功的 post请求	<pre>pm.test("Successful POST request", function () { pm.expect(pm.response.code).to.be.oneOf([201,202]); });</pre>
use tiny validator for JSON data	为json数 据使用tiny 验证器	<pre>var schema = { "items": { "type": "boolean" } }; var data1 = [true, false]; var data2 = [true, 123]; pm.test('Schema is valid', function() { pm.expect(tv4.validate(data1, schema)).to.be.true; pm.expect(tv4.validate(data2, schema)).to.be.true; });</pre>

常见模板用例：

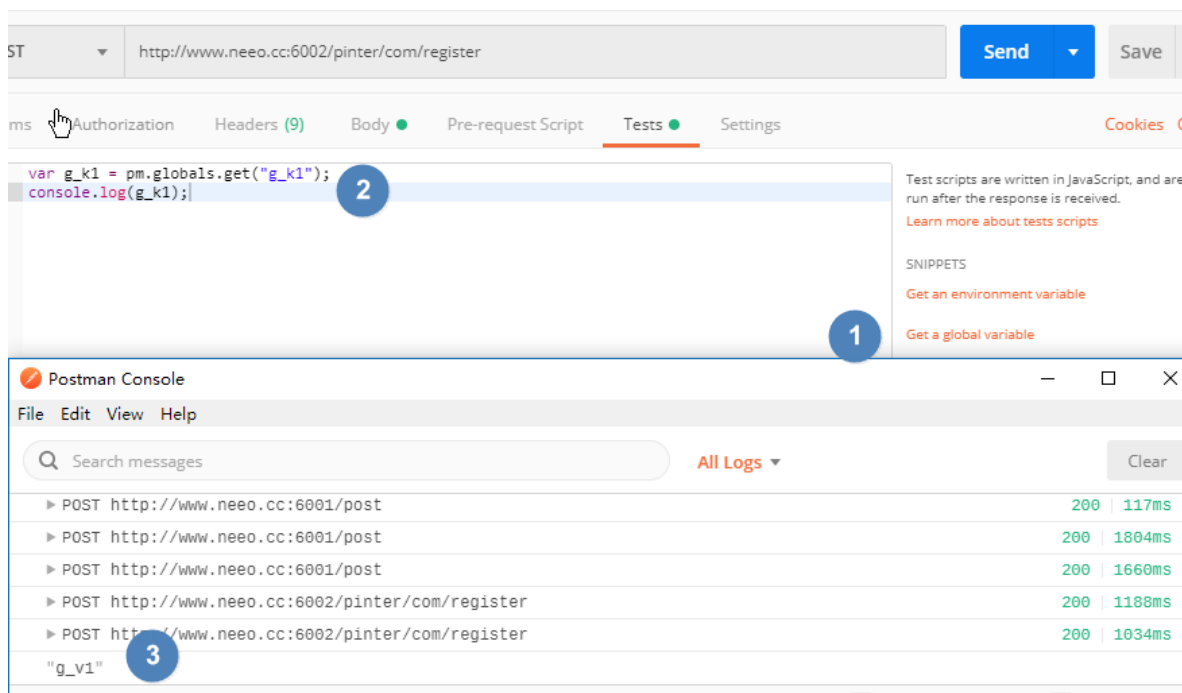
- 状态码判断
- 响应文本判断
- json值判断
- header判断
- 响应时间判断

除了断言之外，还支持变量的操作，环境变量、全局变量的获取和修改。

添加/获取变量系列

获取变量

正如上面的列表所示，在断言中可以获取一个变量、获取一个全局变量等，我们来做一个获取全局变量的示例。

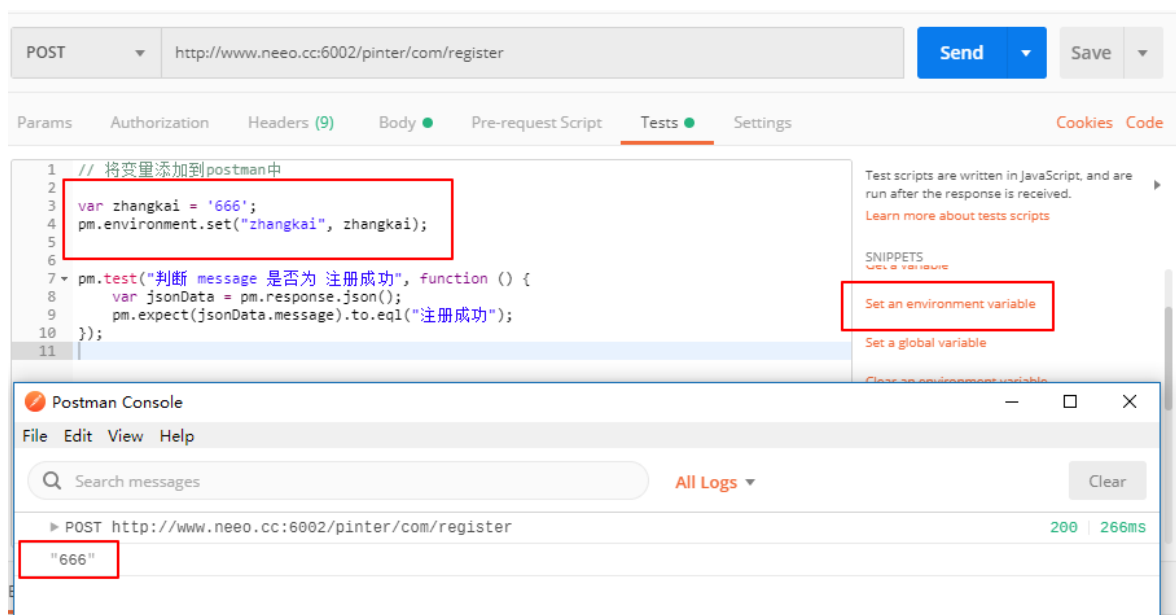


如上图。

- 第一步，选择 `Get global variable`。
- 第二步，编写相关js代码。
- 然后我们控制台中就可以查看到打印结果。

添加变量

那如何使用断言将普通的变量添加到postman的系统环境中呢？这里使用 `Set an environment variable` 模板。

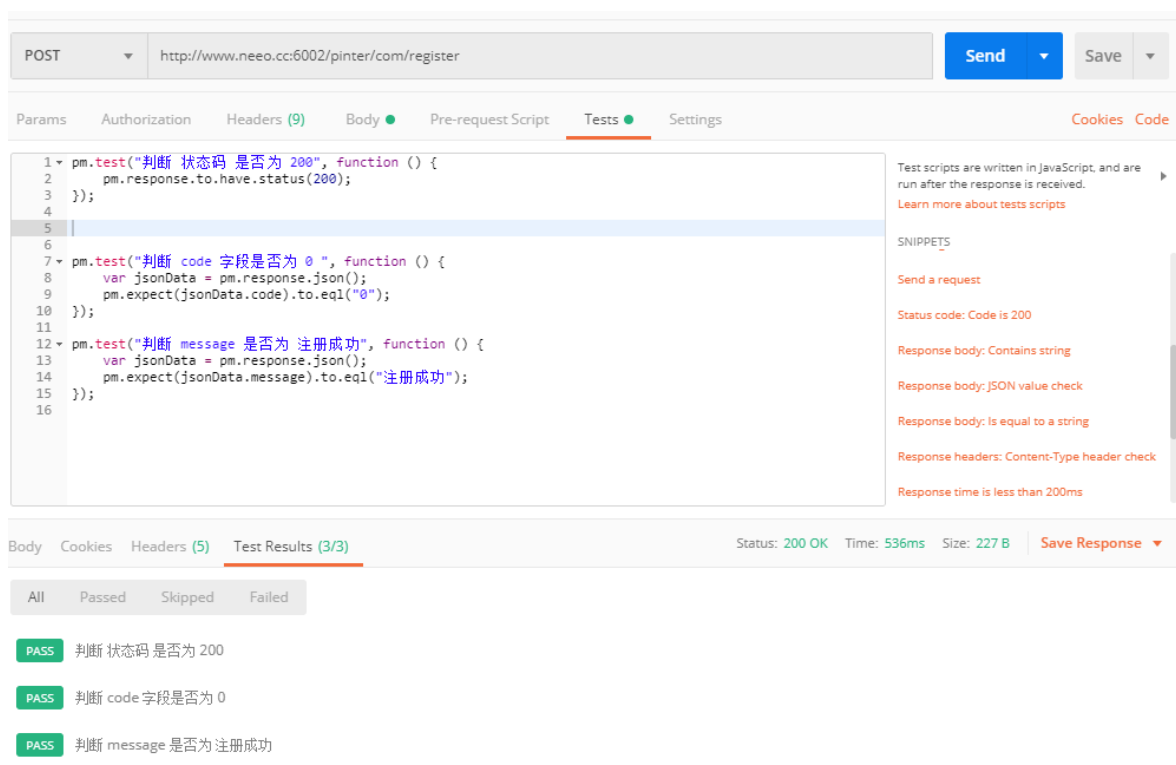


其他的关于变量的操作都差不多，这里不再多说了。

一般断言

还是上面的注册接口，我们对请求结果，可以加各种判断，比如根据状态码做断言，判断返回的json中的字段是否符合预期。

url是：<http://www.nneo.cc:6002/pinter/com/register>



如上图，点击右侧的相关断言模板，然后在修改相应的配置，就完成了断言校验，在返回的菜单栏中的 Test 中，可以看到3个断言都成功了。

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

// Response body: JSON value check
```

```
pm.test("判断 code 字段是否为 0 ", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.code).to.eql("0");
});

// Response body:JSON value check
pm.test("判断 message 是否为 注册成功", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.message).to.eql("注册成功");
});
```

我们来修改一个断言，看看报错是怎么玩的，比如判断code 字段是否为1。

The screenshot shows the Postman interface with a POST request to `http://www.neeo.cc:6002/pinter/com/register`. The 'Tests' tab is active, displaying a JavaScript test script. The script contains four assertions. The fourth assertion, which checks if the 'code' field is '1', is highlighted with a red box and has failed. The 'Test Results' tab at the bottom shows the results of the four assertions: the first three passed, and the fourth failed with the message 'AssertionError: expected '0' to deeply equal '1''. The right sidebar shows the response details, including status code 200, response body, and headers.

```
1 pm.test("判断 状态码 是否为 200", function () {
2     pm.response.to.have.status(200);
3 });
4
5
6
7 pm.test("判断 code 字段是否为 0 ", function () {
8     var jsonData = pm.response.json();
9     pm.expect(jsonData.code).to.eql("0");
10 });
11
12 pm.test("判断 message 是否为 注册成功", function () {
13     var jsonData = pm.response.json();
14     pm.expect(jsonData.message).to.eql("注册成功");
15 });
16
17 pm.test("判断 code 字段是否为 1", function () {
18     var jsonData = pm.response.json();
19     pm.expect(jsonData.code).to.eql("1");
20 });
```

Test scripts are written in JavaScript, and are run after the response is received.
Learn more about tests scripts

SNIPPETS

Send a request

Status code: Code is 200

Response body: Contains string

Response body: JSON value check

Response body: Is equal to a string

Response headers: Content-Type header check

Response time is less than 200ms

Body Cookies Headers (5) Test Results (3/4) Status: 200 OK Time: 734ms Size: 227 B Save Response

All Passed Skipped Failed

PASS 判断 状态码 是否为 200

PASS 判断 code 字段是否为 0

PASS 判断 message 是否为 注册成功

FAIL 判断 code 字段是否为 1 | AssertionError: expected '0' to deeply equal '1'

FAIL 为断言失败。

上述示例中，`jsonData.message` 它背后使用的是 `json_path` 来获取的json中的值，比如，我们要获取一个有多层嵌套的字典，我们怎么来写断言呢？

如下图，我们判断响应中，要判断 `k3` 的值是否等于 `c_v3`，这种嵌套的怎么写。

▶ k:v格式的post请求

Comments (0) Examples (0) ▼

POST http://{{ip}}:{{port}}/post Send Save ▼

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings Cookies Code

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL BETA

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
✓	k1	v1			
✓	k2	{{g_k2}}			
✓	k3	{{c_k3}}			
	Key	Value	Description		

Body Cookies Headers (7) Test Results Status: 200 OK Time: 478ms Size: 815 B Save Response ▼

Pretty Raw Preview Visualize BETA JSON ▼

```
1 {
2   "args": {},
3   "data": "",
4   "files": {},
5   "form": {
6     "k1": "v1",
7     "k2": "g_v2",
8     "k3": "c_v3"
9   },
10 }
```

url是<http://www.neeo.cc:6001/post>

POST http://{{ip}}:{{port}}/post Send Save ▼

Params Authorization Headers (9) Body ● Pre-request Script Tests ● Settings Cookies Code

```
1 pm.test("判断 json中 form 下的 k3 值是否等于 c_v3", function () {
2   var jsonData = pm.response.json();
3   pm.expect(jsonData.form.k3).toEqual("c_v3");
4 });
```

Test scripts are written in JavaScript, and are run after the response is received.
[Learn more about tests scripts](#)

SNIPPETS

- Send a request
- Status code: Code is 200
- Response body: Contains string
- Response body: JSON value check
- Response body: Is equal to a string
- Response headers: Content-Type header check
- Response time is less than 200ms

Body Cookies Headers (7) Test Results (1/1) Status: 200 OK Time: 168ms Size: 815 B Save Response ▼

All Passed Skipped Failed

PASS 判断 json中 form 下的 k3 值是否等于 c_v3

这种嵌套的思路就是，一路点就完了。

我们再来看看其他的断言，比如判断接口的返回中是否包含某些字段。

我们来使用webservice的那个接口。

webservice接口

POST http://fy.webxml.com.cn/webservices/EnglishChinese.asmx

Send Save

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL BETA XML Beautify

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
3   <soap:Body>
4     <TranslatorSentenceString xmlns="http://WebXml.com.cn/">
5       <wordKey>help</wordKey>
6     </TranslatorSentenceString>
7   </soap:Body>
8 </soap:Envelope>
```

body Cookies Headers (9) Test Results Status: 200 OK Time: 211ms Size: 789 B Save Response

Pretty Raw Preview Visualize BETA XML

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/
3   XMLElementSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
4     <soap:Body>
5       <TranslatorSentenceStringResponse xmlns="http://WebXml.com.cn/">
6         <TranslatorSentenceStringResult>
7           <string>Help yourself. |别客气。 </string>
8           <string>He's been helping himself to my stationery. |他未经许可一直用我的文具。 </string>
9           <string>He can't help having big ears. |他爱打听别人的事。 </string>
10        </TranslatorSentenceStringResult>
11      </TranslatorSentenceStringResponse>
12    </soap:Body>
13  </soap:Envelope>
```

我们来判断，响应结果中，是否包含 别客气 字段。用的是 response body:Contains string 模板。

webservice接口

POST http://fy.webxml.com.cn/webservices/EnglishChinese.asmx

Send Save

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Code

```
1 pm.test("判断结果是否包含 别客气", function () {
2   pm.expect(pm.response.text()).to.include("别客气");
3 });
4
5
6
7 pm.test("判断结果是否包含 张开666", function () {
8   pm.expect(pm.response.text()).to.include("张开666");
9 });
```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about tests scripts

SNIPPETS

Send a request

Status code: Code is 200

Response body: Contains string

Response body: JSON value check

Response body: Is equal to a string

Response headers: Content-Type header check

Response time is less than 200ms

Body Cookies Headers (9) Test Results (1/2) Status: 200 OK Time: 153ms Size: 789 B Save Response

All Passed Skipped Failed

PASS 判断结果是否包含 别客气

FAIL 判断结果是否包含 张开666 | AssertionError: expected '<?xml version="1.0" encoding="utf-8"><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><TranslatorSentenceStringResponse xmlns="http://WebXml.com.cn/"><TranslatorSentenceStringResult><string>Help yourself. |别客气。 </string><string>He's been helping himself to my stationery. |他未经许可一直用我的文具。 </string><string>He can't help having big ears. |他爱打听别人的事。 </string></TranslatorSentenceStringResult></TranslatorSentenceStringResponse</soap:Body></soap:Envelope>' to include '张开666'

集合公共断言

对于某一类接口，他们都是一些功能的参数返回值是相同的，那么我们对于这些公共的字段，来做一个断言，而不用每个接口都一一做断言。

我们点击集合的三点，选择 `Edit`，在 `Tests` 选项，添加两个断言语句然后点击 `Update`，那么，这两个断言将作用域当前集合中的每个接口。

EDIT COLLECTION

Name

postman基础操作

Description

Authorization

Pre-request Scripts

Tests

Variables

These tests will execute after every request in this collection. [Learn more about Postman's execution order.](#)

1

2

3

4

5

6

7

8

9

pm.test("判断 状态码 是否为 200", function () {
 pm.response.to.have.status(200);
});

pm.test("判断 code 字段是否为 0 ", function () {
 var jsonData = pm.response.json();
 pm.expect(jsonData.code).to.eql("0");
});

Cancel

Update

在每个接口中，我们只需关注当前接口独有的断言接口，公共的断言都在集合中。

特殊接口

除了上述的那些操作，还有一些其他特殊的接口，比如签名接口校验、携带cookie和token的接口等等。

签名接口

在写开放的API接口时是如何保证数据的安全性的？先来看看有哪些安全性问题在开放的api接口中，我们通过http Post或者Get方式请求服务器的时候，会面临着许多的安全性问题，例如：

1. 请求来源(身份)是否合法？
2. 请求参数被篡改？
3. 请求的唯一性(不可复制)

为了保证数据在通信时的安全性，我们可以采用参数签名的方式来进行相关验证。

那postman中怎么校验这种类型的接口呢？

这个url接口，来演示一下：

```
# URL
http://www.nneo.cc:6002/pinter/com/userInfo
# 类型
POST
# 参数
{"phoneNum":"123434","optCode":"testfan","timestamp":"12112121212","sign":"Md5(手机号+盐+时间戳)"}

```

上面的参数中，需要手机号，optCode 字段是个固定值 testfan，这个别改；时间戳字段 timestamp 是当前的时间戳，sign 字段是需要一个算法来生成的。

这个接口的签名算法为：

```
sign = Md5(手机号+盐+当前时间戳) # 加号代表拼接

```

很明显，这些操作，都要在postman中操作，那么比如获取时间戳和加密都需要JavaScript来完成，我们先来把用到的知识点列出来：

```
// 获取时间戳
var t = new Date().getTime();
// 加密
var md5 = CryptoJS.MD5("需要加密的字符串").toString();
// postman保存变量
pm.environment.set("要保存的变量名", md5)

```

来看postman中怎么测试。

首先要说的是，参数中的需要的数据，都需要在发送请求之前配置到位，那么在哪处理，又怎么处理呢？

```
// 1. 处理手机号 phoneNum
var myPhone = "18211101111"

```



```
// 2. 操作码 optCode
var myOptCode = 'testfan'

// 3. 时间戳
var myTimeStamp = new Date().getTime();

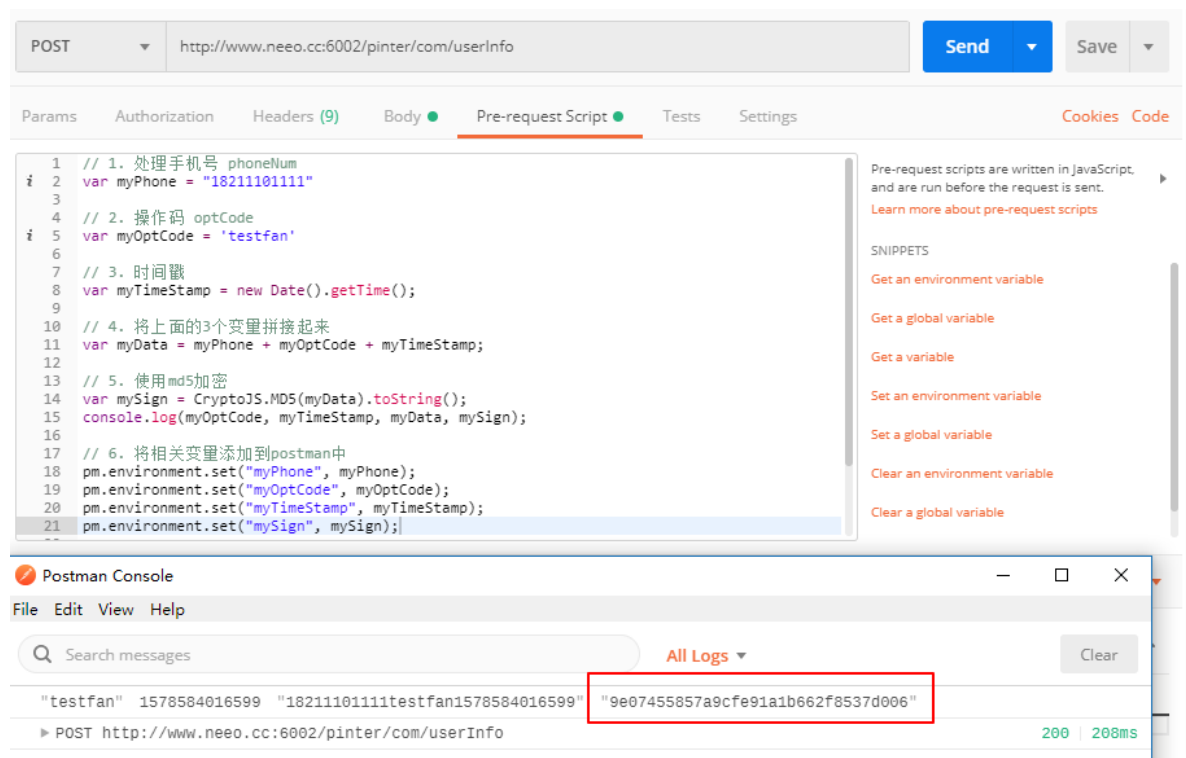
// 4. 将上面的3个变量拼接起来
var myData = myPhone + myOptCode + myTimeStamp;

// 5. 使用md5加密
var mySign = CryptoJS.MD5(myData).toString();
console.log(myOptCode, myTimeStamp, myData, mySign);

// 6. 将相关变量添加到postman中
pm.environment.set("myPhone", myPhone);
pm.environment.set("myOptCode", myOptCode);
pm.environment.set("myTimeStamp", myTimeStamp);
pm.environment.set("mySign", mySign);
```

首先，我们确定了这写操作要在请求发送之前处理好，那么我们就先将上述整理好的代码粘贴到合适的位置。

如下图，在 Pre-request Script 中，编写代码。然后在控制台中，看看查打印结果，可以看到，已经加密成功了，说明我们的代码逻辑没有问题。



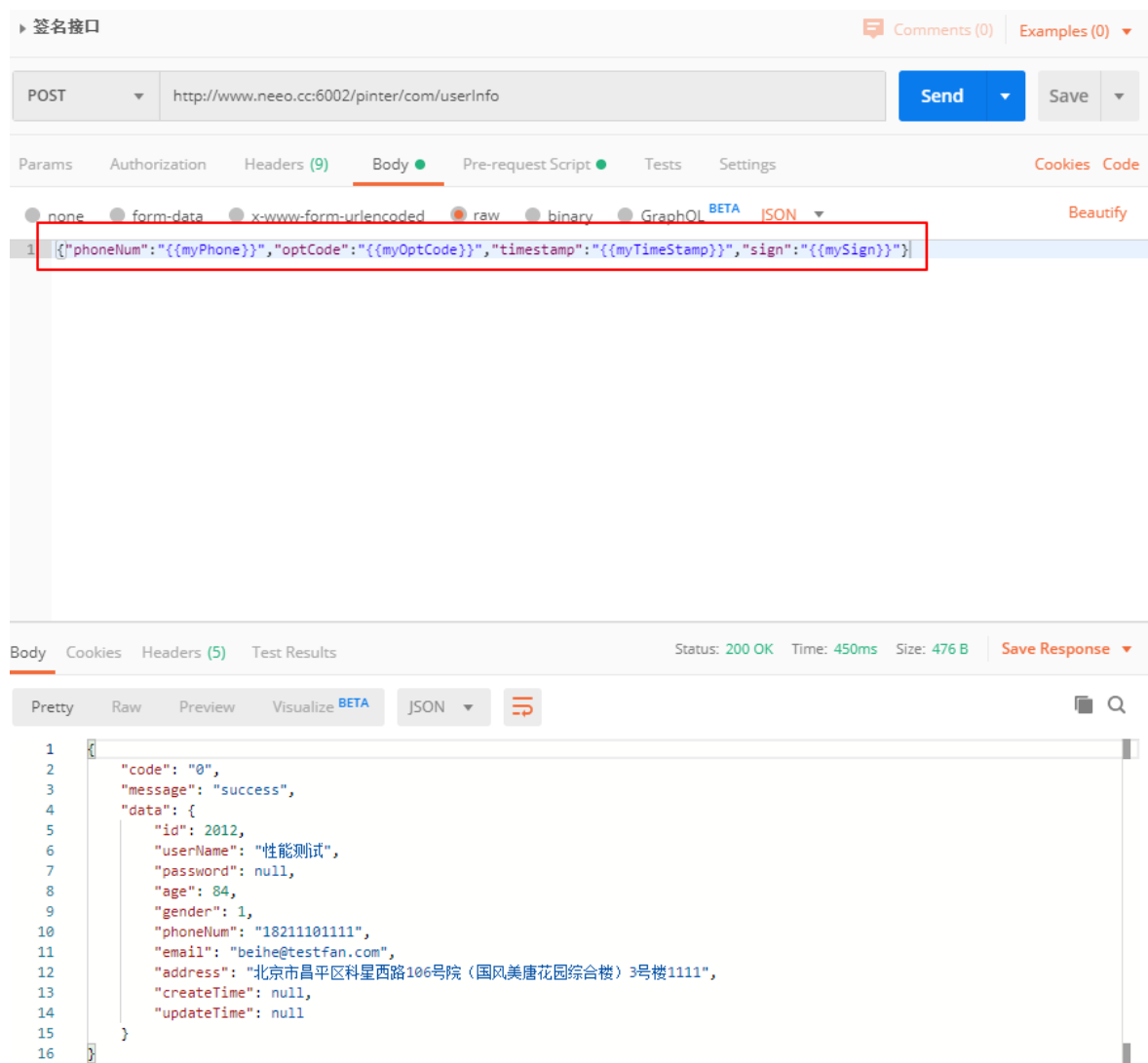
来看原来的参数：

```
{"phoneNum": "123434", "optCode": "testfan", "timestamp": "12112121212", "sign": "Md5(手机号+盐+时间戳)"}
```

现在，我们需要将相关的字段替换为我们之前添加到postman中的变量，下面是替换后的json串。

```
{"phoneNum": "{{myPhone}}", "optCode": "{{myOptCode}}", "timestamp": "{{myTimeStamp}}", "sign": "{{mySign}}"} 
```

然后，我们将替换后的json字符串替换到请求的body中，然后发送请求，查看响应结果。



ok，经过一系列的操作，我们终于将这个签名接口调通了。

cookie

再来看postman如何处理需要携带cookie的接口。

先来一个场景，用户需要登录某网站，登录成功，返回一个cookie，然后用户需要携带cookie在进行进一步操作，比如查询余额等。

我们测试这种类型的接口时，有两种方式：

- 第一种，就是我们使用浏览器登录，获取cookie值，然后手动携带。
- 第二种，就是借助postman帮我们自动的管理cookie值。

先来看第一种方式。

登录接口的相关参数：

```
url: http://www.nneo.cc:6002/pinter/bank/api/login
类型: POST
参数: userName=admin&password=1234
```

查询余额接口相关参数：

url: <http://www.neeo.cc:6002/pinter/bank/api/query>

类型: GET

参数: userName=admin

查询余额接口必须依赖登录成功返回的cookie。

首先, 我们需要在浏览器中访问<http://www.neeo.cc:6002/pinter/bank/page/login>连接, 输入 admin&1234 进行登录, 登录成功后, 我们获取到了cookie值。

欢迎您, admin

Pinter MyHome Contact

We Will Not Rest

瑞士联合银行为广大的客户提供一系列的产品和服务。银行的核心服务是资产管理、投资担保、以及商业零售服务, 瑞士联合银行将服务内容的革新、全球的广泛发展与经济的稳定联系起来, 实现优质的服务。

[查看账户余额](#) [安全退出](#)

案例一: 基于cookie的验证方式 (Made by testfan.cn@北河)

Network

login /pinter/bank/api

General

Request URL: <http://www.neeo.cc:6002/pinter/bank/api/login>

Request Method: POST

Status Code: 200

Remote Address: 47.52.72.214:6002

Referrer Policy: no-referrer-when-downgrade

Response Headers

Connection: keep-alive

Content-Type: application/json; charset=UTF-8

Date: Thu, 09 Jan 2020 15:53:39 GMT

Keep-Alive: timeout=20

Set-Cookie: testfan-id=54a3c07c-5f65-4154-8960-0e8c6f047406; Max-Age=216000; Expires=Sun, 12-Jan-2020

现在, 保持登录状态, 我们使用postman访问查询余额接口<http://www.neeo.cc:6002/pinter/bank/api/query?userName=admin>。

cookie接口

GET http://www.nneo.cc:6002/pinter/bank/api/query?userName=admin

Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION
userName	admin	
Key	Value	Description

Body Cookies Headers (5) Test Results Status: 200 OK Time: 156ms Size: 230 B Save Response

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   "code": "1",
3   "message": "用户不合法",
4   "data": null
5 }
```

暂时还没有加cookie，我们查询余额失败。现在，我们从浏览器中拷贝cookie值，添加到查询余额的请求头中，再次访问。

cookie接口

GET http://www.nneo.cc:6002/pinter/bank/api/query?userName=admin

Send Save

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Code

Headers (1)

KEY	VALUE	DESCRIPTION
Cookie	testfan-id=75a48eb8-576e-4f5a-90eb-2b2dda1...	
Key	Value	Description

Temporary Headers (7)

Body Cookies Headers (5) Test Results Status: 200 OK Time: 1234ms Size: 232 B Save Response

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   "code": "0",
3   "message": "success",
4   "data": "$ 59,834,959"
5 }
```

这种相对比较麻烦。

我们来看第二种，在postman中，新建一个登录的接口。

登录接口

POST http://www.nneo.cc:6002/pinter/bank/api/login

Send Save

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL BETA

KEY	VALUE	DESCRIPTION
userName	admin	
password	1234	
Key	Value	Description

Body Cookies (1) Headers (6) Test Results Status: 200 OK Time: 223ms Size: 346 B Save Response

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   "code": "0",
3   "message": "success",
4   "data": null
5 }
```

然后，我们在查询余额接口中，就不用携带cookie了，直接访问即可。

cookie接口

GET http://www.nneo.cc:6002/pinter/bank/api/query?userName=admin

Send Save

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Code

Headers (0)

KEY	VALUE	DESCRIPTION
Key	Value	Description

Temporary Headers (8)

KEY	VALUE
User-Agent	PostmanRuntime/7.20.1
Accept	*/*
Cache-Control	no-cache
Postman-Token	c4d29d20-7a91-4a35-ac7b-4d076fff4460
Host	www.nneo.cc:6002
Accept-Encoding	gzip, deflate
Cookie	testfan-id=e7bc6a7e-fb59-4d0e-836a-62c1817eb319
Connection	keep-alive

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 252ms Size: 232 B Save Response

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   "code": "0",
3   "message": "success",
4   "data": "$ 35,879,167"
5 }
```

现在，发现，访问成功了。如上图，我们打开 Temporary Headers，发现本次请求，postman自动的帮我们携带了cookie，才能访问成功。

其实，postman自动的帮我们管理了cookie。在哪看呢，点击上图中的绿色框中的 cookies 可以发现那个cookie值。

MANAGE COOKIES

Type a domain name

Add

ios.wecash.net 0 cookies

+ Add Cookie

127.0.0.1 0 cookies

+ Add Cookie

v2ex.com 0 cookies

+ Add Cookie

www.neeo.cc 1 cookie

testfan-id

+ Add Cookie

Whitelist Domains

Learn More

这里也可以自己添加cookie，看你需求吧。

token

来研究一下postman如何处理需要携带token的接口。

这里用到postman的数据管理的功能，该功能对token进行处理。

关键步骤：

```
// 从响应中获取json数据
var jsonData = pm.response.json();

// 从json中的相应字段提取token
var myToken = jsonData.meassage;

// 将token值set到postman中，方便后续引用
pm.environment.set("myToken",myToken)
```

来个场景，还是登陆后进行后续操作，只是需要携带token了。

登录/查询余额接口相关参数：

登录

url: http://www.nneo.cc:6002/pinter/bank/api/login2

类型: POST

参数: userName=admin&password=1234

查询余额

url: http://www.nneo.cc:6002/pinter/bank/api/query2

类型: GET

参数: userName=admin

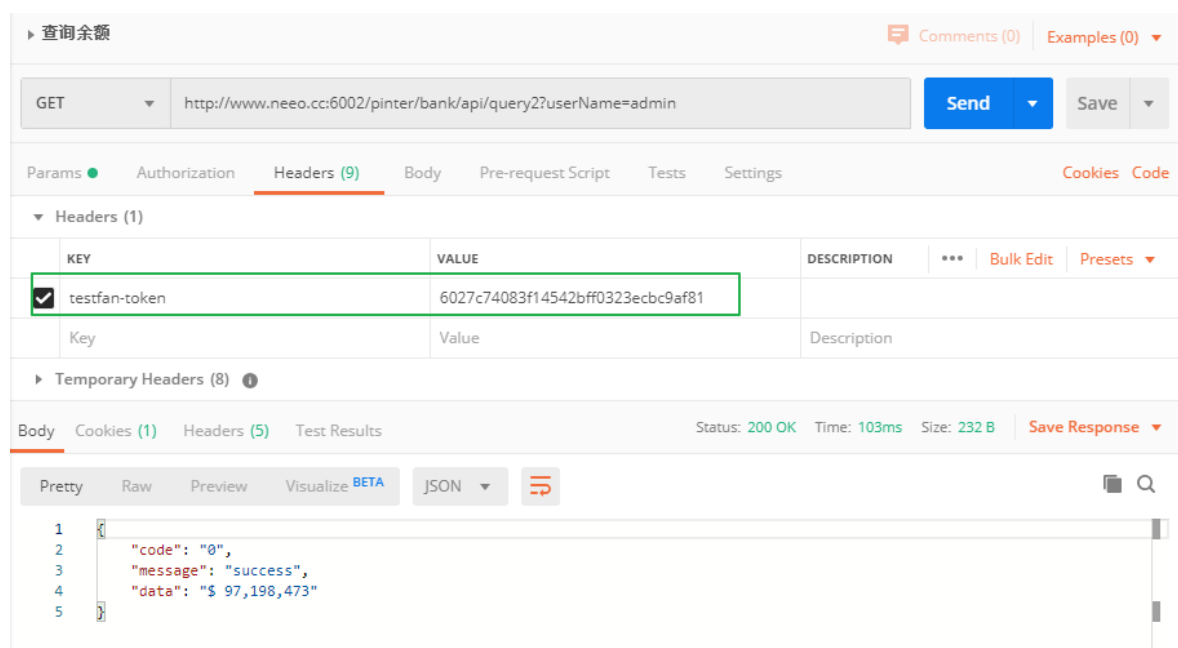
第一种方式，仍然为自己在浏览器（建议是用火狐浏览器抓包）登录，然后抓包，抓到token值。

The screenshot shows a web application interface with a dark background. At the top, it displays '您的余额为 : \$ 65,181,858' in white text. Below this, there is a green button labeled '安全退出'. At the bottom of the interface, it says '案例二：基于token的验证方式 (Made by testfan.cn@北河)'.

Below the web application, the Firefox network log is visible. The log shows several requests, including a GET request to 'money2?userName=admin&password=1234' which returns a 200 status. The response headers include 'Connection: keep-alive', 'Content-Type: application/json; charset=UTF-8', 'Date: Thu, 09 Jan 2020 16:37:03 GMT', 'Keep-Alive: timeout=20', and 'Transfer-Encoding: chunked'. The request headers include 'Accept: application/json, text/javascript, */*; q=0.01', 'Accept-Encoding: gzip, deflate', 'Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2', 'Connection: keep-alive', and 'Host: www.nneo.cc:6002'. The request body is highlighted in blue and contains the following JSON:

```
{  "url": "http://www.nneo.cc:6002/pinter/bank/api/query2?userName=admin&password=1234",  "method": "GET",  "headers": {    "Host": "www.nneo.cc:6002",    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:69.0) Gecko/20100101 Firefox/69.0",    "Referer": "http://www.nneo.cc:6002/pinter/bank/api/login2?userName=admin&password=1234",    "testfan-token": "6027c74083f14542bfff0323ecbc9af81"  },  "body": ""}
```

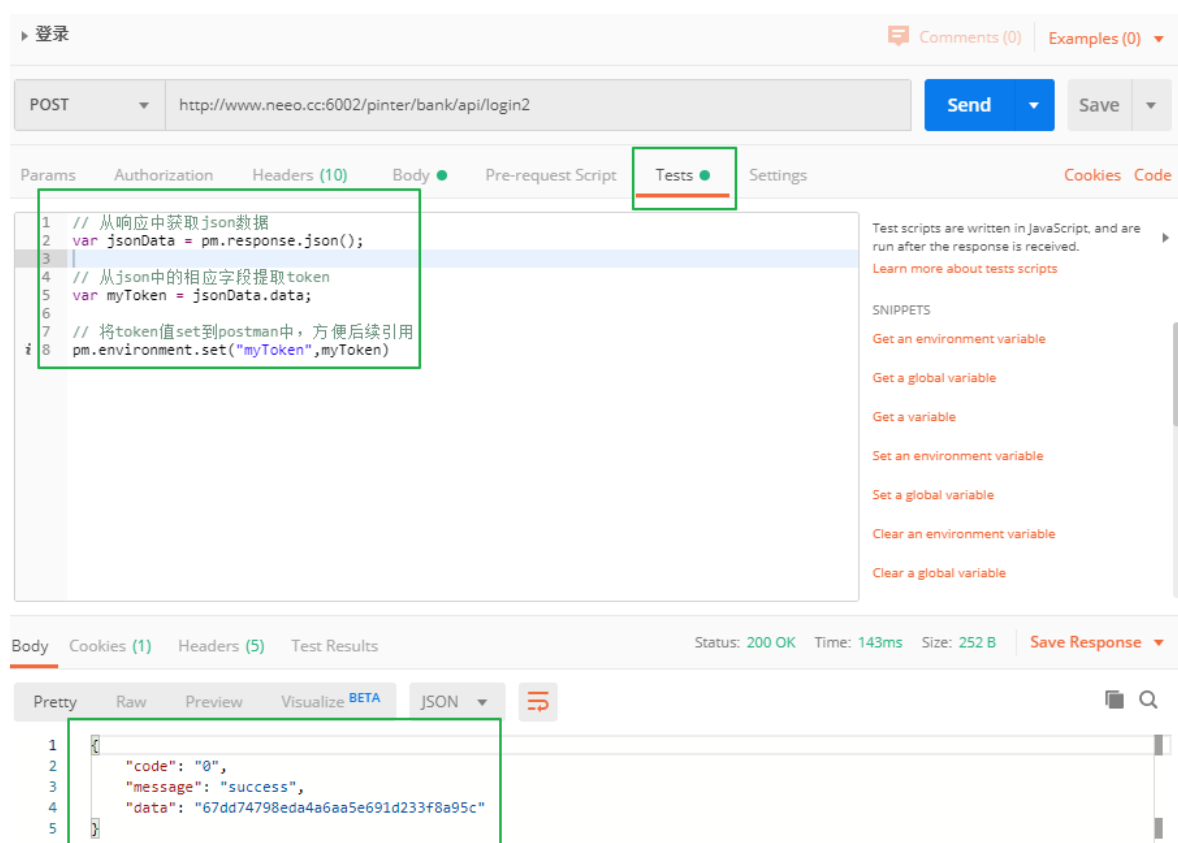
然后在查询余额中携带。



现在就OK了。

第二种，就是使用postman的数据关联的功能了。

首先，我们配置登录接口，然后从响应的json中提取token，然后将token保存到postman的环境变量中。那在哪写提取token的代码呢？还记得 Tests 吗？它是响应后要干的事情，之前我们用它来做断言。现在用它来提取token值。



我们也可以去环境变量中查看刚刚添加的token值。

POST 登录

+

...

远程测试环境1

远程测试环境1

登录

Edit

VARIABLE	INITIAL VALUE	CURRENT VALUE
ip	www.neeo.cc	www.neeo.cc
port	6001	6001
mySign		d2c319935f51d0f5f984de5a82c0a706
zhangkai		666
myOptCode		testfan
myPhone		18211101111
myTimeStamp		1578584199997
myToken		67dd74798eda4a6aa5e691d233f8a95c

Globals

Edit

VARIABLE	INITIAL VALUE	CURRENT VALUE
g_k1	g_v1	g_v1
g_k2	g_v2	g_v2

然后在后续的查询余额中，引用环境变量中的token值，只要这个token值不过期，我们查询余额就能一直用，过期了，再访问一下登录接口就行了。

▶ 查询余额

Comments (0) | Examples (0) ▼

GET

http://www.neeo.cc:6002/pinter/bank/api/query2?userName=admin

Send

Save

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Cookies

Code

▼ Headers (1)

KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets ▼
<input checked="" type="checkbox"/> testfan-token	{{myToken}}				
Key	Value	Description			

▶ Temporary Headers (8) ⓘ

Body

Cookies (1)

Headers (5)

Test Results

Status: 200 OK

Time: 54ms

Size: 232 B

Save Response ▼

Pretty

Raw

Preview

Visualize BETA

JSON ▼

≡

```

1  {
2    "code": "0",
3    "message": "success",
4    "data": "$ 93,133,041"
5  }

```

ok，完事了。

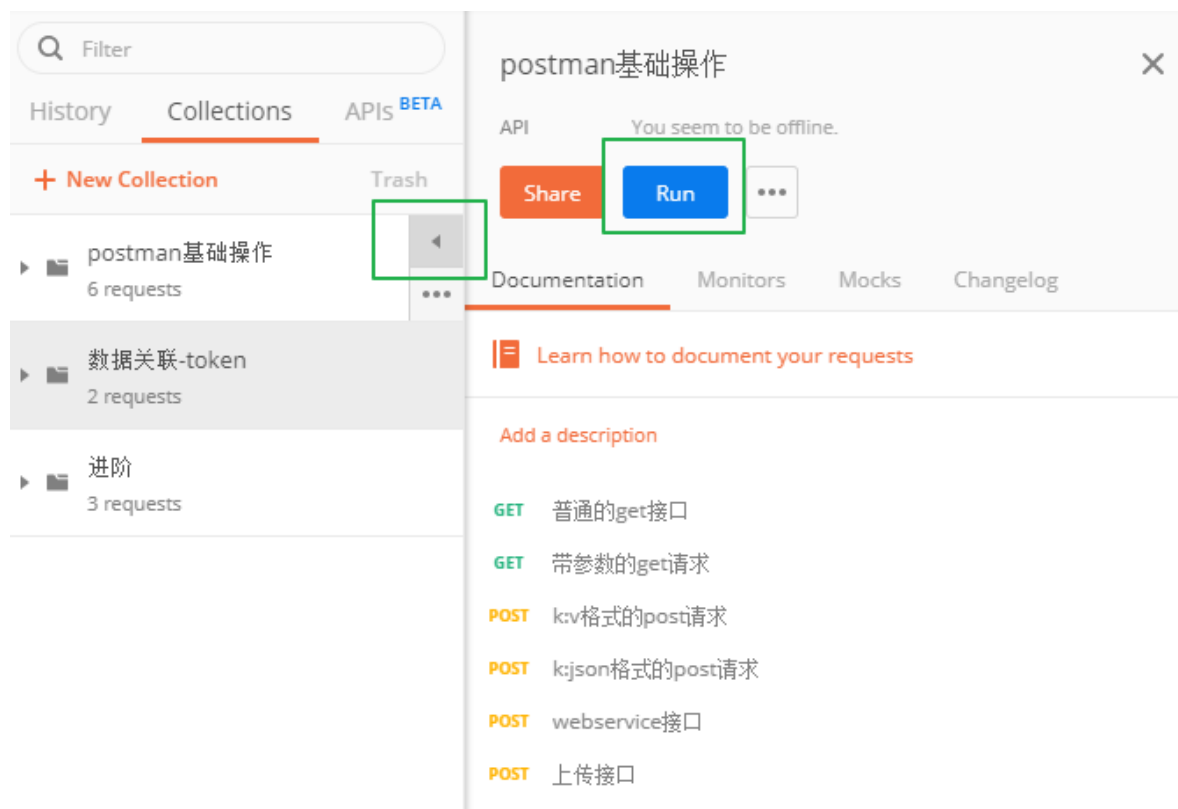
集合自动化

集合自动化说白了，就是自动的将集合内的所有用例跑一遍或者跑几遍，然后跑的过程中，可以做断言之类的。

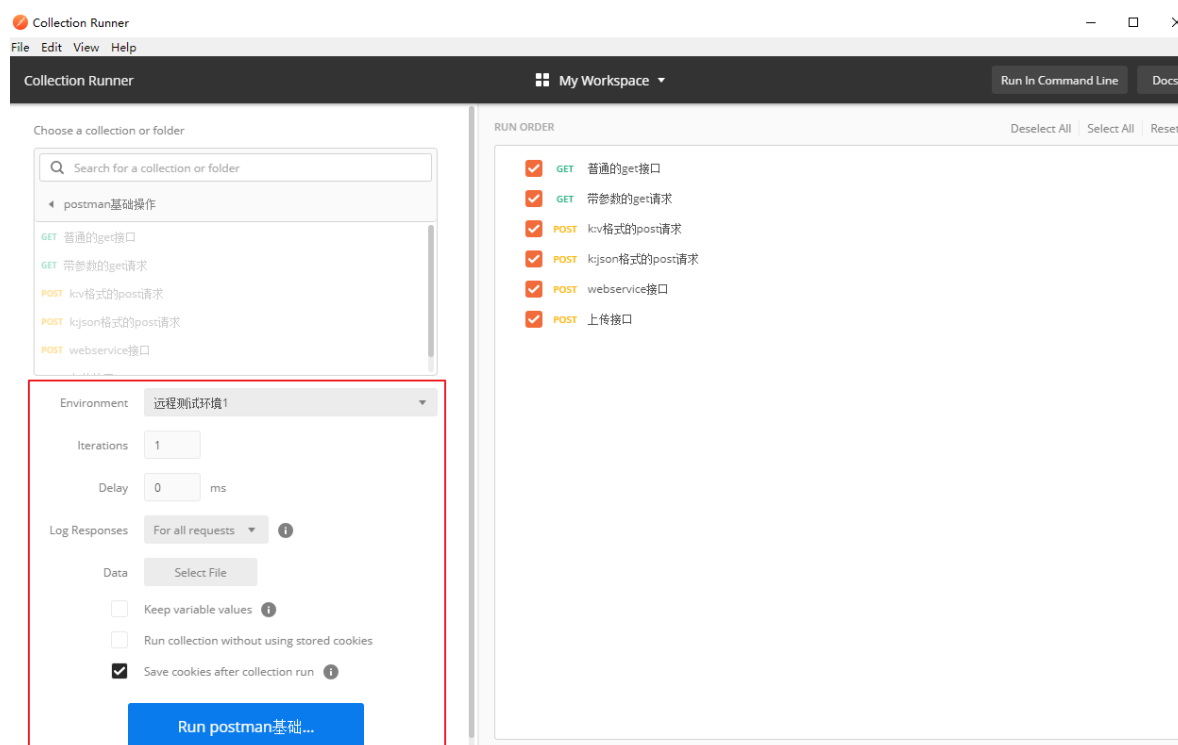
怎么玩呢？

快速上手

我们选择最开始创建的集合，该集合中有6个接口，并且都有相应的断言。然后我们选择集合的右上角的运行按钮，弹出框中选择 Run。



相关的参数配置，如下图。

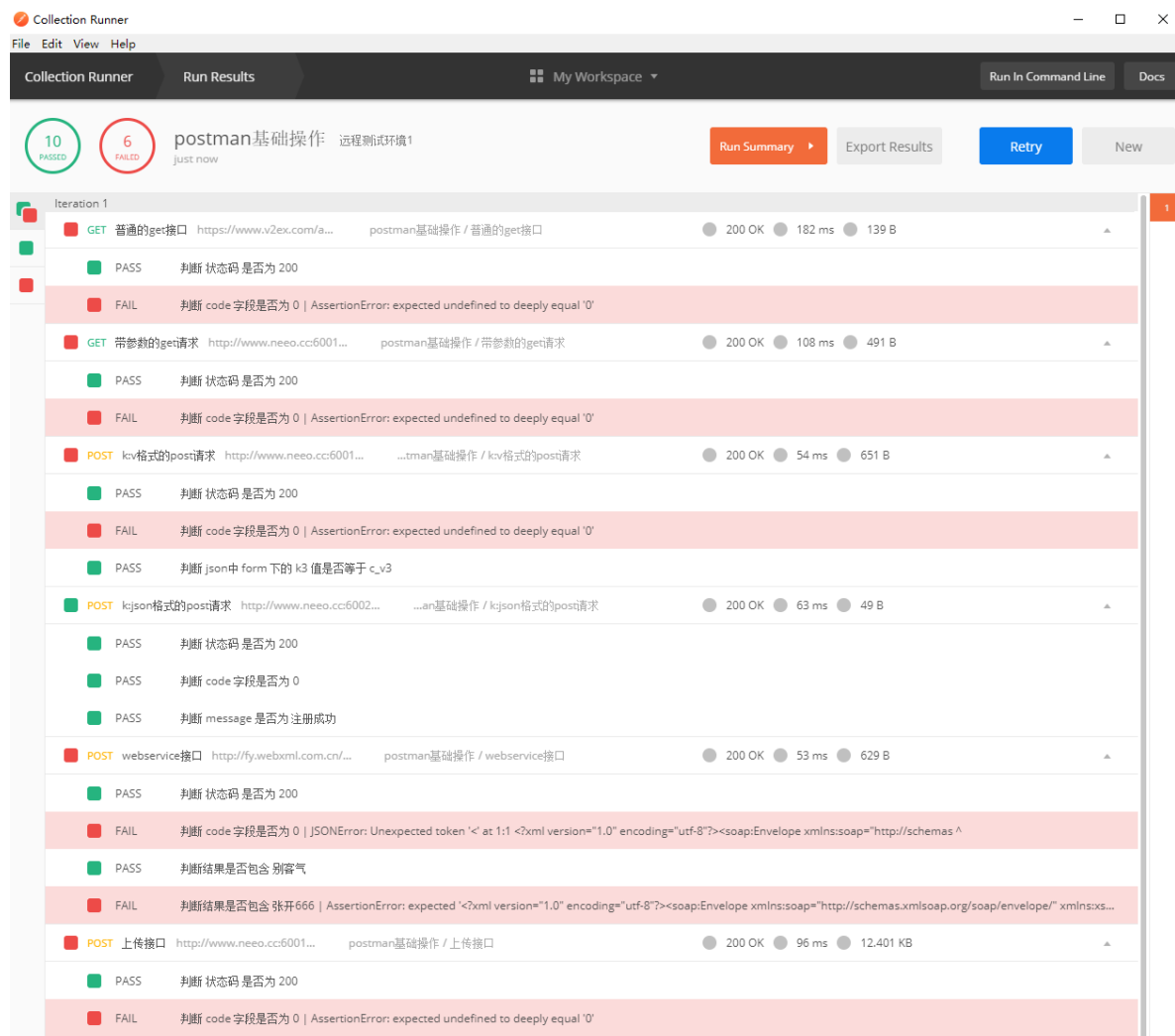


参数说明：

- Environment：选择接口执行时依赖的环境。
- Iterations：表示该集合内的接口运行的次数。
- Delay：接口执行前延时多少毫秒。
- Log Response：默认记录所有的响应日志。
- Data：参数数据，暂时先不管它。
- Keep variable values：保存在运行过程中产生的变量或者值，选择默认即可。
- Run collection without using stored cookies：运行中不使用cookie，这里也保持默认。

- Save cookies after collection run: 在集合执行之后，保存cookies，选择默认。
- Run 集合名称：运行集合。

我们点击运行集合。



如上图，集合内的6个接口，共执行了16次，那你可能会问，我们总共就6个接口，怎么就执行了16次呢？这是因为postman为每个接口的每个断言都执行一次，也就是说，如果这个接口，有两个断言，那么在本次执行中，将会分别为每个断言执行一次共两次。所以整体执行完，断言成功的有10个，失败的有6个。

相关参数：

- Run Summary: 运行情况，没啥好说的。

Collection Runner

Run Results

Run Summary

My Workspace

Run In Command Line

Docs

10 PASSED

6 FAILED

postman基础操作 远程测试环境1

5 mins ago

Export Results

New

Back

1

GET 普通的get接口

PASS 判断 状态码 是否为 200

FAIL 判断 code 字段是否为 0

GET 带参数的get请求

PASS 判断 状态码 是否为 200

FAIL 判断 code 字段是否为 0

POST kv格式的post请求

PASS 判断 状态码 是否为 200

FAIL 判断 code 字段是否为 0

PASS 判断 json中 form 下的 k3 值是否等于 c_v3

POST kvjson格式的post请求

PASS 判断 状态码 是否为 200

FAIL 判断 code 字段是否为 0

PASS 判断结果是否包含 别客气

FAIL 判断结果是否包含 张开666

POST webservice接口

PASS 判断 状态码 是否为 200

FAIL 判断 code 字段是否为 0

PASS 判断结果是否包含 别客气

FAIL 判断结果是否包含 张开666

POST 上传接口

PASS 判断 状态码 是否为 200

FAIL 判断 code 字段是否为 0

- Export Results: 将执行结果导出为json文件。
- Retry: 重新执行。
- 页面右侧上部的1表示, 本次执行, 共执行一次, 也就是集合中的接口执行了一轮。如果我们将 Iterations 改为 2 , 那么再来看。

Choose a collection or folder

Q

Search for a collection or folder

◀ postman基础操作

GET

普通的get接口

GET

带参数的get请求

POST

kv格式的post请求

POST

kjson格式的post请求

POST

webservice接口

Environment 远程测试环境1 ▼

Iterations 2

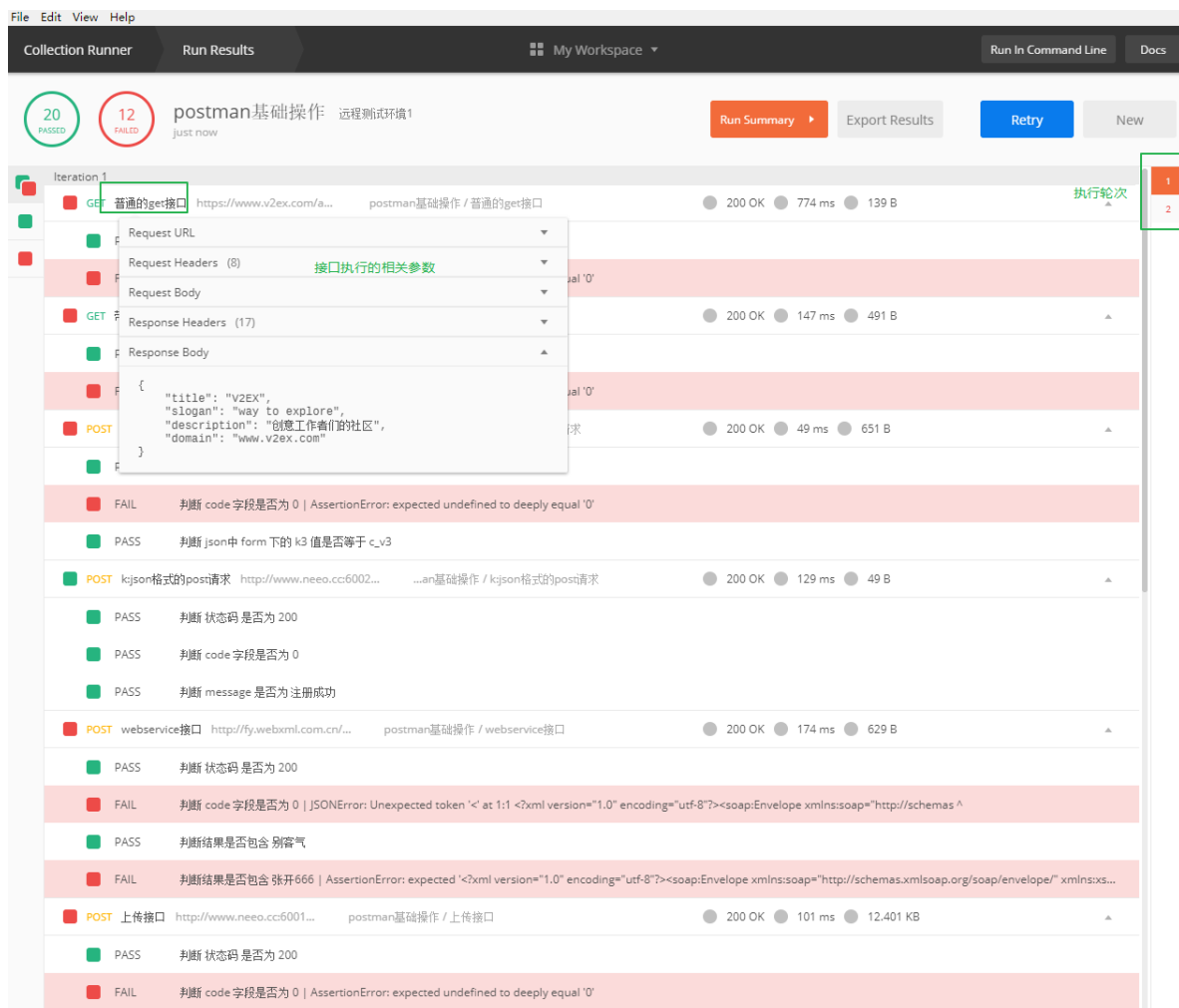
Delay 0 ms

Log Responses For all requests ▼ ⓘ

Data Select File

- ☐ Keep variable values ⓘ
- ☐ Run collection without using stored cookies
- ☒ Save cookies after collection run ⓘ

Run postman基础...



ok, 就酱紫!

基于数据驱动的集合自动化

有些时候, 接口依赖的数据来自于文件, 也就是基于文件数据驱动测试。

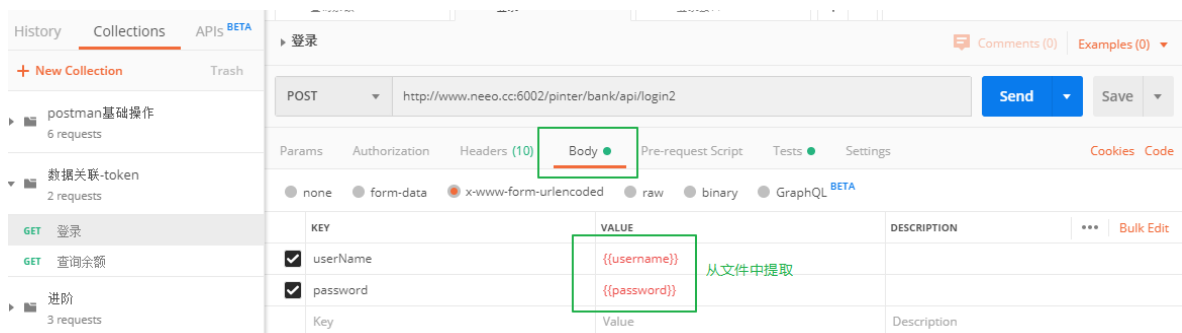
比如我们本地有一个这样的 `data.csv` 文件, `csv` 文件是文本类型的文件, 可以用Excel表格打开, 也可以使用记事本打开。

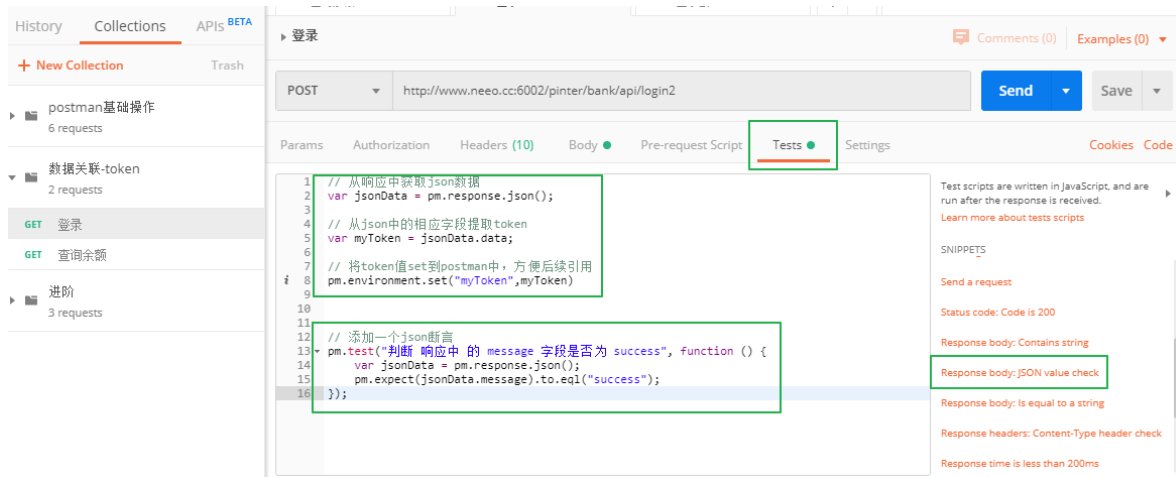
```
username,password
zhangkaitui,1234
zhangkaizui,1234
zhangkaishou,1234
```

比如我们测试之前的关于token的两个接口, 相关配置我们略作修改。

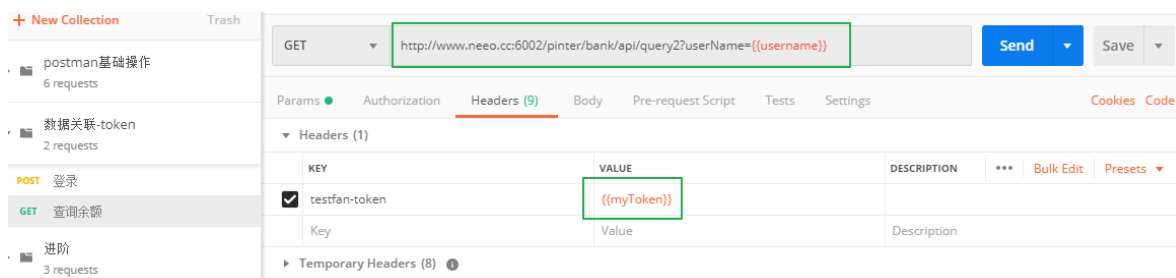
注意, 修改后要点击保存!

登录接口这里的用户名和密码都要来自于文件。

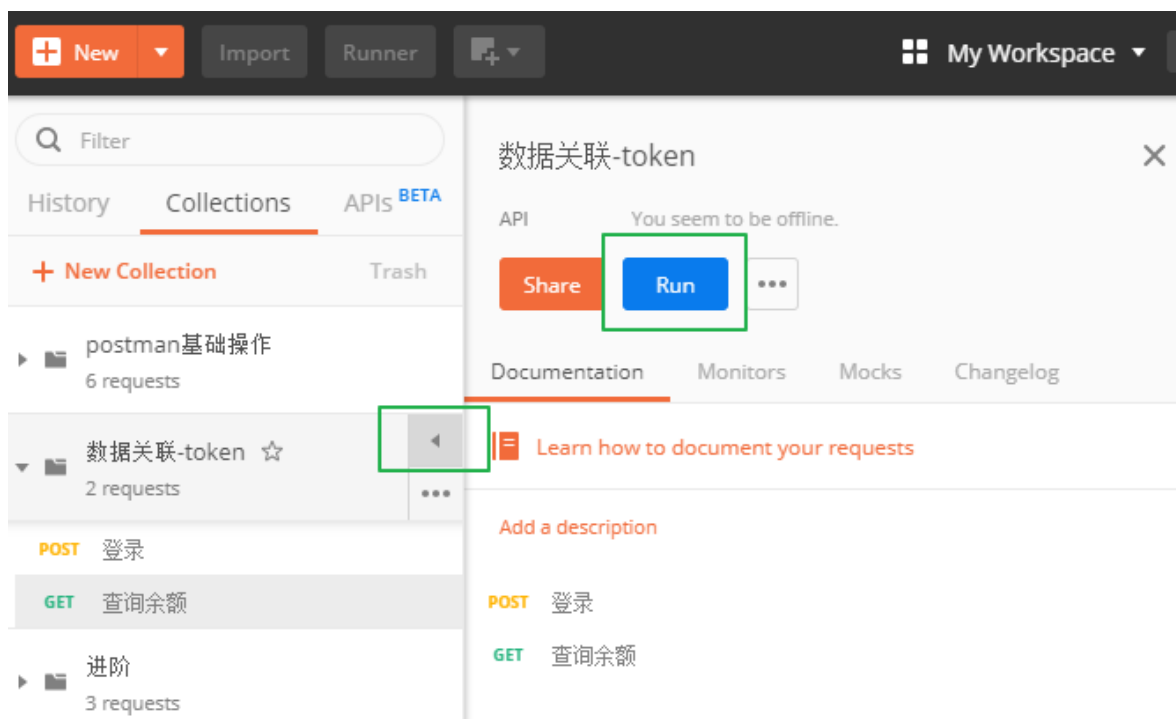




查询余额也是一样的，别忘了保存。



现在，是时候执行一波了。



进行一些配置。

Choose a collection or folder

Search for a collection or folder

数据关联-token

POST 登录

GET 查询余额

Environment 远程测试环境1

Iterations 5 执行5轮

Delay 0 ms

Log Responses For all requests

Data

Select File data.csv X 选择本地的CSV文件

Data File Type text/csv Preview 点击预览

☐ Keep variable values

☐ Run collection without using stored cookies

☒ Save cookies after collection run

Run 数据关联-token 没啥文件就，走你.....

RUN ORDER

POST 登录

GET 查询余额

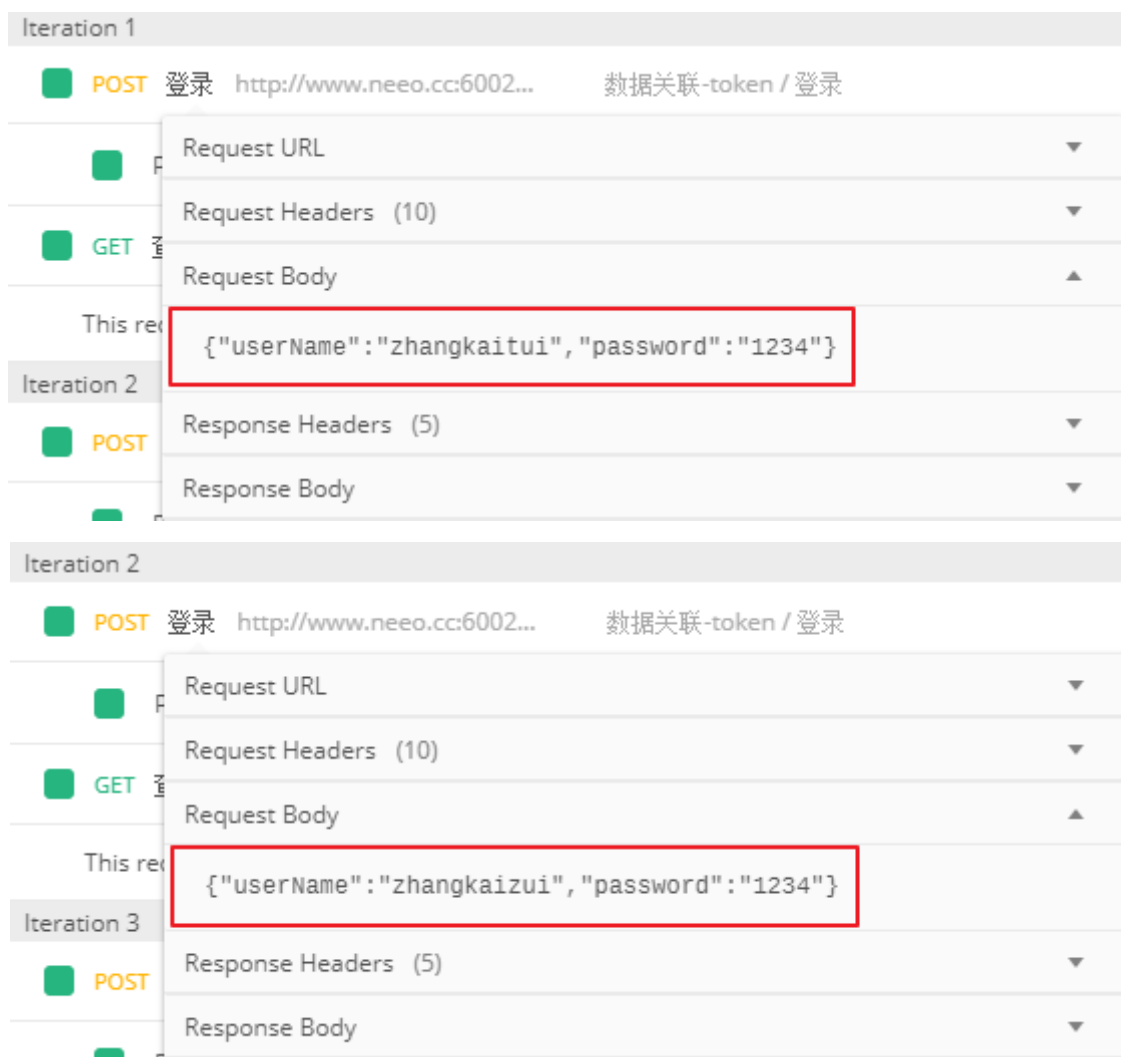
注意：先登录，后查询余额

没啥好说的了，都在上图说完了。

看效果，如下图。

Collection Runner		Run Results	My Workspace		Run In Command Line	Docs
5 PASSED	0 FAILED	数据关联-token 远程测试环境1 just now	Run Summary	Export Results	Retry	New
Iteration 1						
POST	登录	http://www.nneo.cc:6002... 数据关联-token / 登录	200 OK	269 ms	74 B	1
PASS	判断 响应中的 message 字段是否为 success					2
GET	查询余额	http://www.nneo.cc:6002... 数据关联-token / 查询余额	200 OK	133 ms	54 B	3
This request does not have any tests.						4
Iteration 2						
POST	登录	http://www.nneo.cc:6002... 数据关联-token / 登录	200 OK	110 ms	74 B	5
PASS	判断 响应中的 message 字段是否为 success					
GET	查询余额	http://www.nneo.cc:6002... 数据关联-token / 查询余额	200 OK	109 ms	54 B	
This request does not have any tests.						
Iteration 3						
POST	登录	http://www.nneo.cc:6002... 数据关联-token / 登录	200 OK	104 ms	74 B	
PASS	判断 响应中的 message 字段是否为 success					
GET	查询余额	http://www.nneo.cc:6002... 数据关联-token / 查询余额	200 OK	91 ms	54 B	
This request does not have any tests.						
Iteration 4						
POST	登录	http://www.nneo.cc:6002... 数据关联-token / 登录	200 OK	307 ms	74 B	
PASS	判断 响应中的 message 字段是否为 success					
GET	查询余额	http://www.nneo.cc:6002... 数据关联-token / 查询余额	200 OK	107 ms	54 B	
This request does not have any tests.						
Iteration 5						
POST	登录	http://www.nneo.cc:6002... 数据关联-token / 登录	200 OK	251 ms	74 B	
PASS	判断 响应中的 message 字段是否为 success					
GET	查询余额	http://www.nneo.cc:6002... 数据关联-token / 查询余额	200 OK	109 ms	54 B	
This request does not have any tests.						

需要注意的是，我们本地的CSV文件共3个账户，我们执行了5轮，那么它是怎么取值的呢？我们先看第一轮和第二轮测试，看它的取值。

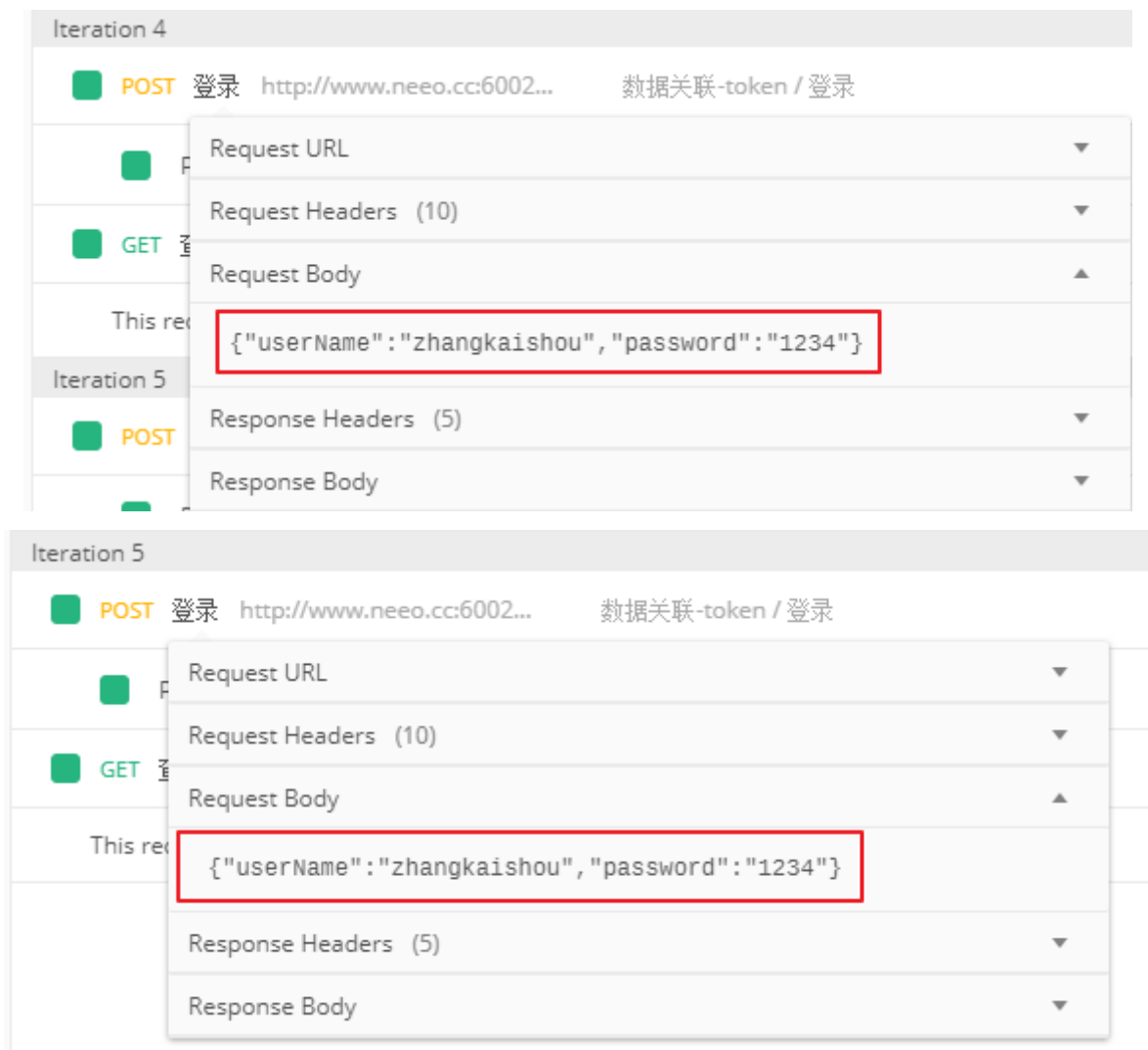


再来看第三轮。



上图，可以发现，前三轮是从第一个用户顺序往后取值的，直到文件所有的用户数据取值完毕。

那么我们再来看第四轮和第五轮，看看能不能发现什么规律。



可以看到，当读取到文件的最后一个用户的数据时，后续的每轮执行都引用的是最后一个用户数据。而不是又从新读文件的第一个用户数据，这一点是需要我们注意的。

命令行测试

命令行测试？难道GUI的还不能满足你么？非也非也，使用命令行模式为了搭配其他的工具来开展测试工作。

环境配置

安装node.js

[node.js for windows](http://nodejs.cn/)
<http://nodejs.cn/>
<https://nodejs.org/en/>

安装cnpm

```
npm install -g cnpm --registry=https://registry.npm.taobao.org

cnpm -v      # # 检查cnpm是否安装成功

# 示例
C:\Users\Anthony>cnpm -v
cnpm@6.1.0
(C:\Users\Anthony\AppData\Roaming\npm\node_modules\cnpm\lib\parse_argv.js)
npm@6.12.0
(C:\Users\Anthony\AppData\Roaming\npm\node_modules\cnpm\node_modules\npm\lib\npm.js)
node@8.9.0 (G:\software\nodejs\node.exe)
npminstall@3.23.0
(C:\Users\Anthony\AppData\Roaming\npm\node_modules\cnpm\node_modules\npminstall\lib\index.js)
prefix=C:\Users\Anthony\AppData\Roaming\npm
win32 x64 10.0.14393
registry=https://r.npm.taobao.org
```

安装newman

```
cnpm install newman --global      # 命令行执行

# 示例
C:\Users\Anthony>cnpm install newman --global
Downloading newman to
C:\Users\Anthony\AppData\Roaming\npm\node_modules\newman_tmp
Copying
C:\Users\Anthony\AppData\Roaming\npm\node_modules\newman_tmp\_newman@4.5.7@newman
n to C:\Users\Anthony\AppData\Roaming\npm\node_modules\newman
Installing newman's dependencies to
C:\Users\Anthony\AppData\Roaming\npm\node_modules\newman\node_modules
[1/20] eventemitter3@4.0.0 installed at
node_modules\_eventemitter3@4.0.0@eventemitter3
[2/20] chardet@0.8.0 installed at node_modules\_chardet@0.8.0@chardet
[3/20] @postman/csv-parse@4.0.2 installed at node_modules\_@postman_csv-
parse@4.0.2@@postman\csv-parse
[4/20] filesize@6.0.1 installed at node_modules\_filesize@6.0.1@filesize
[5/20] colors@1.4.0 installed at node_modules\_colors@1.4.0@colors
[6/20] commander@4.0.1 installed at node_modules\_commander@4.0.1@commander
[7/20] lodash@4.17.15 installed at node_modules\_lodash@4.17.15@lodash
[8/20] cli-progress@3.4.0 installed at node_modules\_cli-progress@3.4.0@cli-
progress
[9/20] async@3.1.0 installed at node_modules\_async@3.1.0@async
[10/20] semver@6.3.0 installed at node_modules\_semver@6.3.0@semver
[11/20] mkdirp@0.5.1 installed at node_modules\_mkdirp@0.5.1@mkdirp
[12/20] xmlbuilder@13.0.2 installed at
node_modules\_xmlbuilder@13.0.2@xmlbuilder
[13/20] word-wrap@1.2.3 installed at node_modules\_word-wrap@1.2.3@word-wrap
[14/20] cli-table3@0.5.1 installed at node_modules\_cli-table3@0.5.1@cli-table3
[15/20] pretty-ms@5.1.0 installed at node_modules\_pretty-ms@5.1.0@pretty-ms
[16/20] serialised-error@1.1.3 installed at node_modules\_serialised-
error@1.1.3@serialised-error
[17/20] postman-collection-transformer@3.2.0 installed at node_modules\_postman-
collection-transformer@3.2.0@postman-collection-transformer
```

```
[18/20] postman-request@2.88.1-postman.16 installed at node_modules\_postman-request@2.88.1-postman.16@postman-request
[19/20] postman-collection@3.5.5 installed at node_modules\_postman-collection@3.5.5@postman-collection
[20/20] postman-runtime@7.21.0 installed at node_modules\_postman-runtime@7.21.0@postman-runtime
Recently updated (since 2020-01-03): 3 packages (detail see file
C:\Users\Anthony\AppData\Roaming\npm\node_modules\newman\node_modules\.recently_updates.txt)
  2020-01-07
    → postman-runtime@7.21.0 > handlebars@4.5.3 > uglify-js@^3.1.4(3.7.4)
(09:24:25)
  2020-01-06
    → postman-request@2.88.1-postman.16 > mime-types@~2.1.19(2.1.26) (11:47:55)
    → postman-request@2.88.1-postman.16 > mime-types@2.1.26 > mime-db@1.43.0(1.43.0) (11:24:37)
All packages installed (158 packages installed from npm registry, used
23s(network 23s), speed 390.2kB/s, json 139(361kB), tarball 8.25MB)
[newman@4.5.7] link C:\Users\Anthony\AppData\Roaming\npm\newman@ ->
C:\Users\Anthony\AppData\Roaming\npm\node_modules\newman\bin\newman.js
```

检查newman是否安装成功

```
newman -v

# 示例
C:\Users\Anthony>newman -v

C:\Users\Anthony>"node"
"C:\Users\Anthony\AppData\Roaming\npm\node_modules\newman\bin\newman.js" -v
4.5.7
```

PS:扫盲篇:

[npm是干什么的?](#)

[newman是干什么的?](#)

一句话: nodejs是一个基于 Chrome V8 引擎的 JavaScript 运行环境, node.js的包管理器npm是全球最大的开源库生态系统, 我们可以使用npm命令来下在各种库和工具, 但很遗憾, 由于网络的原因, 我们选择使用淘宝的npm镜像, 叫做cnpm, 而newman则是 Postman 推出的一个 nodejs 库, 直接来newman说就是 Postman 的json文件可以在命令行中执行的插件, newman 可以方便地运行和测试集合, 并用之构造接口自动化测试和持续集成。

使用newman执行命令行测试

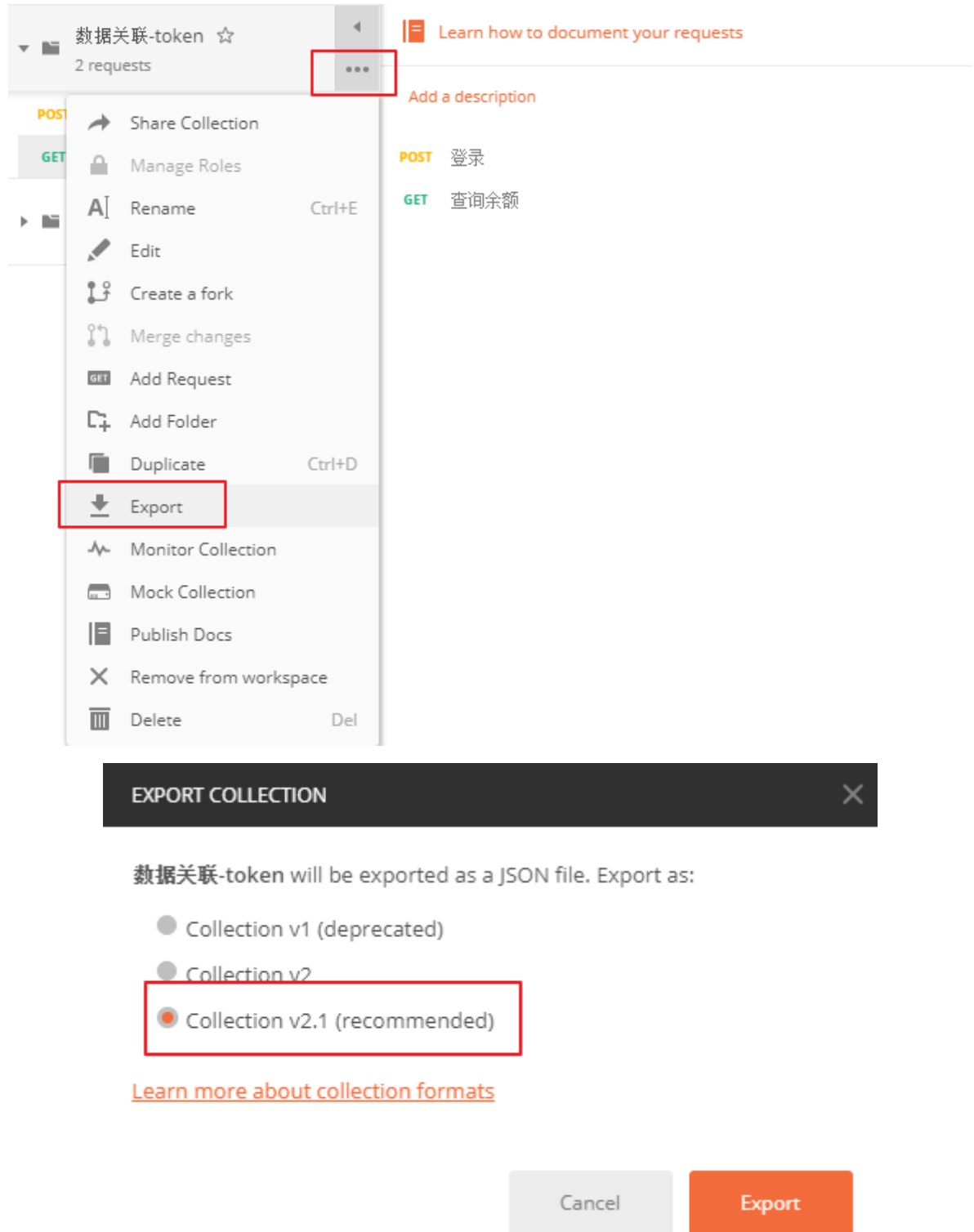
使用newman执行命令行测试, 大致要分为以下几步:

- 导出集合为json脚本
- 导出环境为json文件
- 准备好参数文件
- 安装reporter
- 执行newman命令

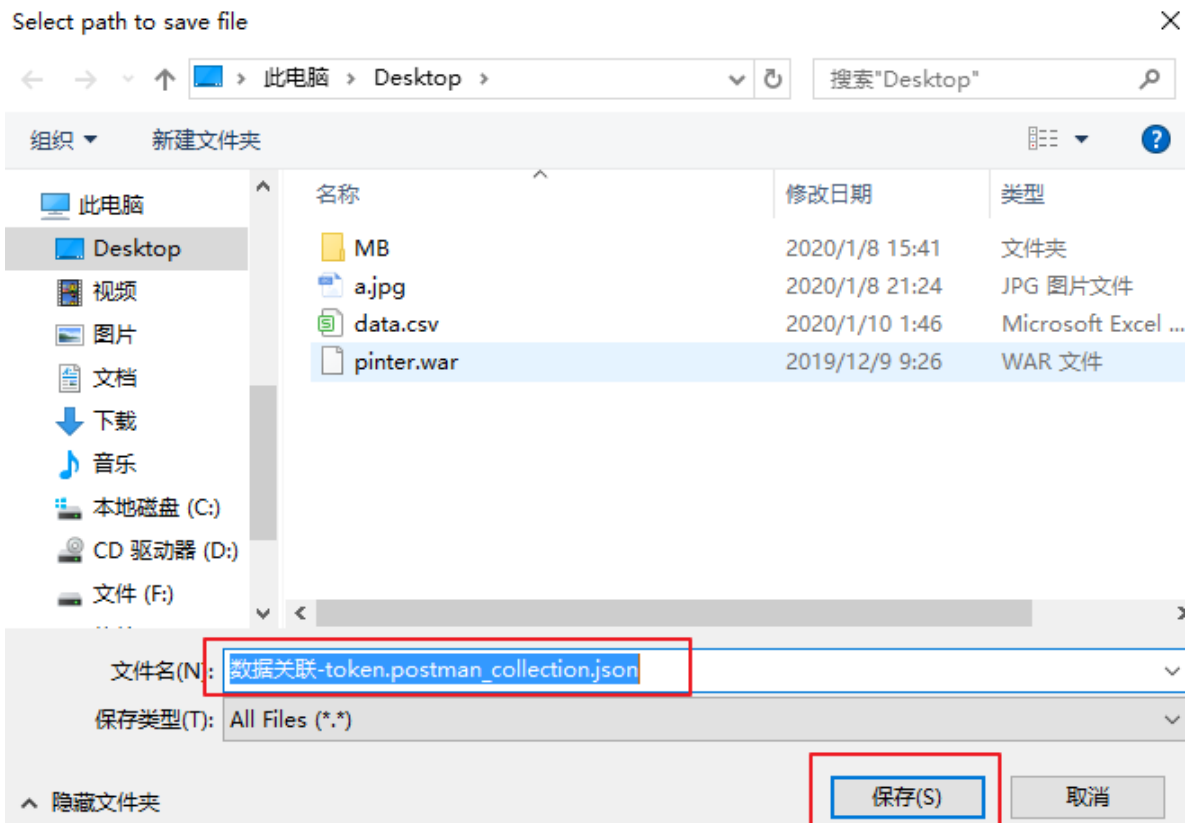
来走一波。

导出集合为json脚本

这里还是以之前的token的集合为例。按照下面的图操作即可。

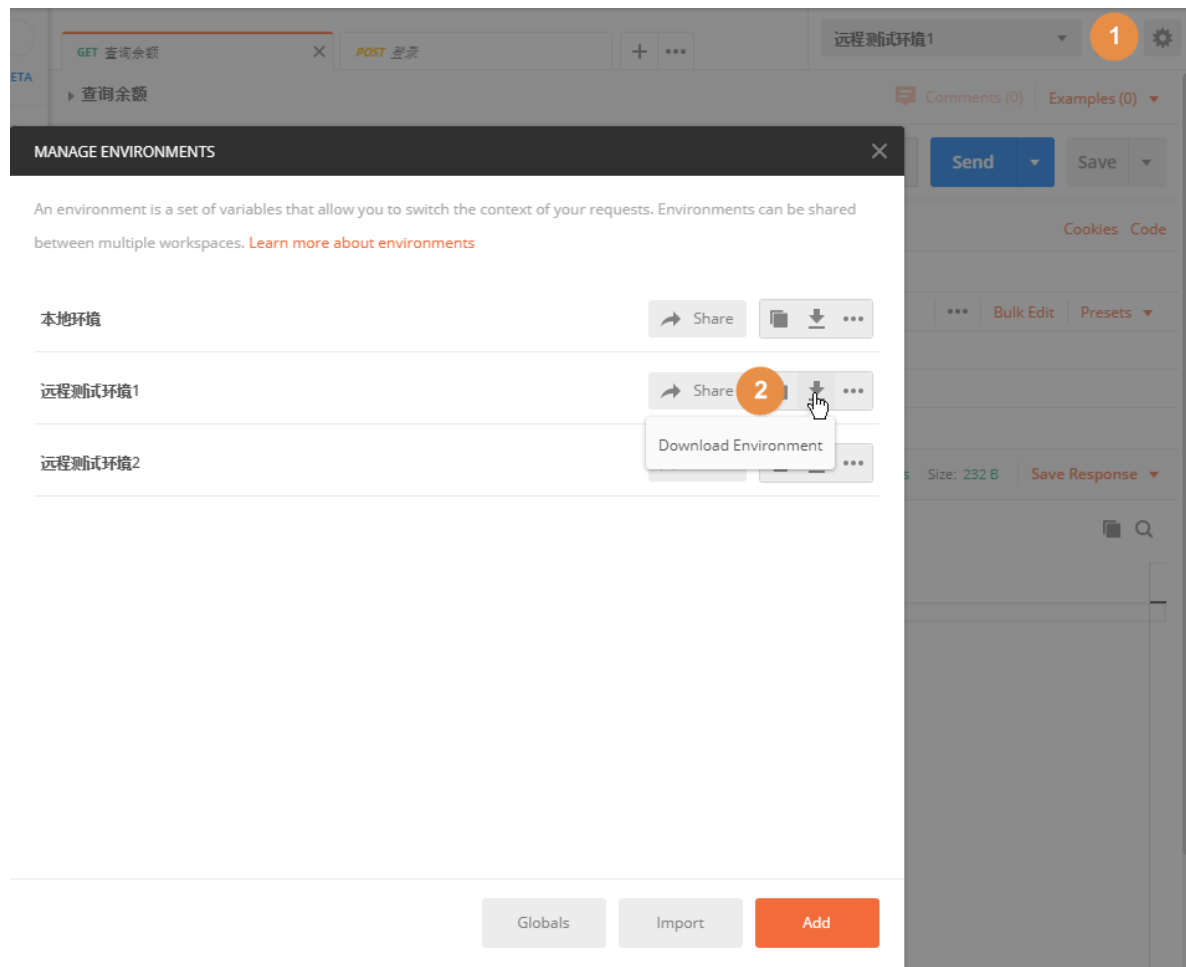


将json文件导出到桌面。

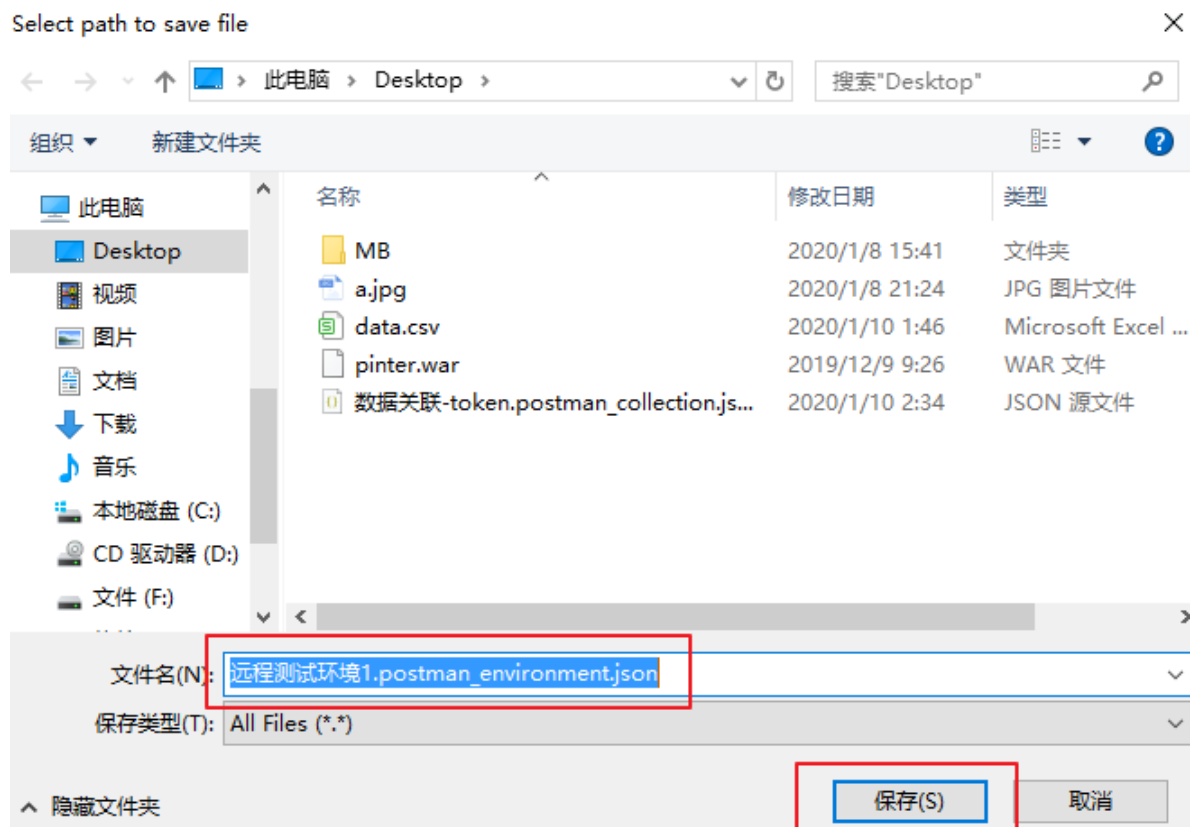


(可选) 导出环境为json文件

如果你的集合中的接口使用了环境变量，还需要将环境变量导出来，如果没有，就不用导了。



点击环境设置，选择对应的环境，然后点击下载，下载到本地的还是一个json文件。



(可选) 准备好参数文件

同样的，我们在这个集合中，使用了数据驱动，也就是引用了 `data.csv` 文件，你也要把这个文件提前准备好。

安装reporter

啥也不说了，干（下载）就完了！

```
cnpm install -g newman-reporter-html

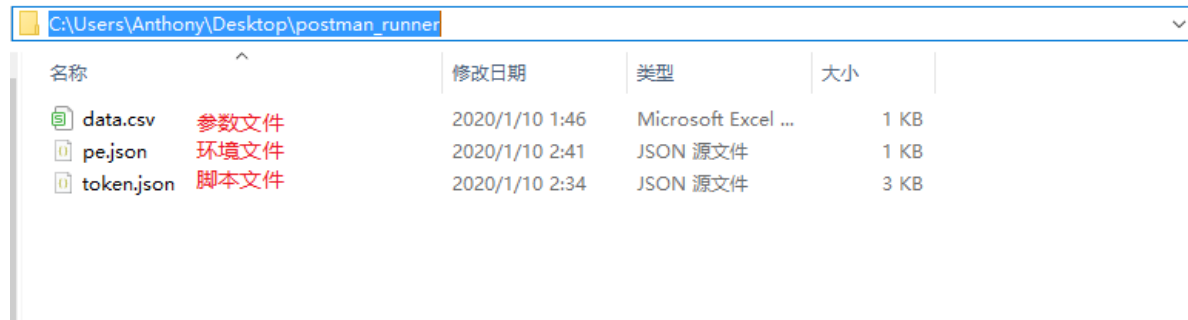
# 示例
C:\Users\Anthony>cnpm install -g newman-reporter-html
Downloading newman-reporter-html to
C:\Users\Anthony\AppData\Roaming\npm\node_modules\newman-reporter-html_tmp
Copying C:\Users\Anthony\AppData\Roaming\npm\node_modules\newman-reporter-html_tmp\newman-reporter-html@1.0.5@newman-reporter-html to
C:\Users\Anthony\AppData\Roaming\npm\node_modules\newman-reporter-html
Installing newman-reporter-html's dependencies to
C:\Users\Anthony\AppData\Roaming\npm\node_modules\newman-reporter-html\node_modules
[1/4] filesize@6.0.1 installed at node_modules\_filesize@6.0.1@filesize
[2/4] lodash@4.17.15 installed at node_modules\_lodash@4.17.15@lodash
[3/4] pretty-ms@5.1.0 installed at node_modules\_pretty-ms@5.1.0@pretty-ms
[4/4] handlebars@4.5.3 installed at node_modules\_handlebars@4.5.3@handlebars
Recently updated (since 2020-01-03): 1 packages (detail see file
C:\Users\Anthony\AppData\Roaming\npm\node_modules\newman-reporter-html\node_modules\.recently_updates.txt)
2020-01-07
  → handlebars@4.5.3 > uglify-js@^3.1.4(3.7.4) (09:24:25)
All packages installed (12 packages installed from npm registry, used 4s(network 4s), speed 380.84kB/s, json 12(55.18kB), tarball 1.34MB)
```

执行newman命令

参照下面的命令，开始执行吧。

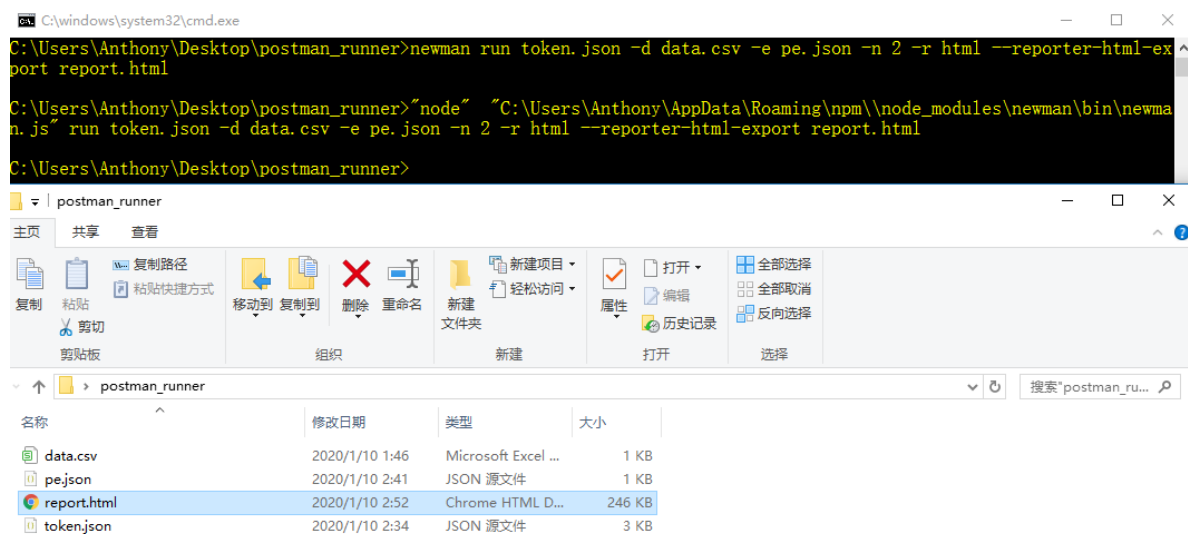
```
newman run 脚本 -d 参数文件 -e 环境文件 -n 循环次数 -r html --reporter-html-export 测试报告路径
```

首先，由于保存的json文件名，又臭又长，还含有中文，我们稍事整理，将这些文件整理到一个目录中，比如我把它们放到桌面的 postman_runner 目录中。



在当前目录中打开终端，执行命令：

```
newman run token.json -d data.csv -e pe.json -n 2 -r html --reporter-html-export report.html
```



ok，执行完，就在本地生成了一个 report.html 文件，让我们打开它看一眼吧！

Newman Report

Collection 数据关联-token
 Time Fri Jan 10 2020 02:52:39 GMT+0800 (中国标准时间)
 Exported with Newman v4.5.7

	Total	Failed
Iterations	2	0
Requests	4	0
Prerequisite Scripts	0	0
Test Scripts	2	0
Assertions	2	0

Total run duration 1067ms
 Total data received 256B (approx)
 Average response time 227ms

Total Failures 0

Requests

登录

Method	POST		
URL	http://www.nneo.cc:6002/pinter/bank/api/login2		
Mean time per request	368ms		
Mean size per request	74B		
Total passed tests	2		
Total failed tests	0		
Status code	200		
Tests	Name	Pass count	Fail count
	判断 响应中的 message 字段是否为 success	2	0

查询余额

Method	GET		
URL	http://www.nneo.cc:6002/pinter/bank/api/query2?userName=zhangkaitui		
Mean time per request	86ms		
Mean size per request	54B		
Total passed tests	0		
Total failed tests	0		
Status code	200		

看不懂? 感觉low? 那是你没见过原生的.....

```
newman run token.json -d data.csv -e pe.json -n 2
```

```
C:\Users\Anthony\Desktop\postman_runner>newman run token.json -d data.csv -e pe.json -n 2
C:\Users\Anthony\Desktop\postman_runner>node "C:\Users\Anthony\AppData\Roaming\npm\node_modules\newman\bin\newman.js" run token.json -d data.csv -e pe.json -n 2
newman

数据关联-token

Iteration 1/2
→ 登录 POST http://www.nneo.cc:6002/pinter/bank/api/login2 [200 OK, 252B, 581ms]
✓ 判断 响应中 的 message 字段是否为 success

→ 查询余额 GET http://www.nneo.cc:6002/pinter/bank/api/query2?userName=zhangkaitui [200 OK, 232B, 98ms]

Iteration 2/2
→ 登录 POST http://www.nneo.cc:6002/pinter/bank/api/login2 [200 OK, 252B, 100ms]
✓ 判断 响应中 的 message 字段是否为 success

→ 查询余额 GET http://www.nneo.cc:6002/pinter/bank/api/query2?userName=zhangkaizui [200 OK, 232B, 87ms]



|                                                                   | executed | failed |
|-------------------------------------------------------------------|----------|--------|
| iterations                                                        | 2        | 0      |
| requests                                                          | 4        | 0      |
| test-scripts                                                      | 2        | 0      |
| prerequisite-scripts                                              | 0        | 0      |
| assertions                                                        | 2        | 0      |
| total run duration: 1054ms                                        |          |        |
| total data received: 256B (approx)                                |          |        |
| average response time: 216ms [min: 87ms, max: 581ms, s.d.: 210ms] |          |        |



C:\Users\Anthony\Desktop\postman_runner>_
```

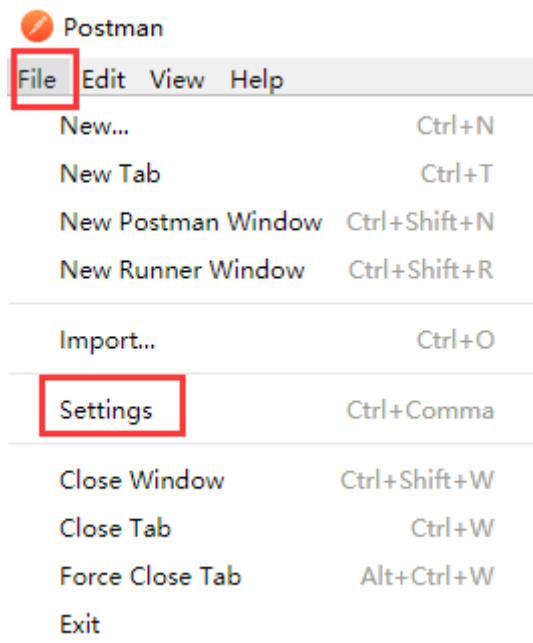
我想以你的审美，上面这个会不会更好看？

设置相关

主题设置

主题这里就黑色和白色两个两个选项。步骤是：

- 菜单栏选择 `File --> Settings`



- Themes 选项中，选择喜欢的主题背景即可。

