# 概述

前面我们介绍了接口测试工具：Postman 和 Jmeter。利用接口测试工具上手使用虽然容易，但是还是不够灵活。例如需要界面上各种配置，有时还需限定的语言脚本来辅助（Postman 限定使用 `Js`,Jmeter 限定使用 `Java`）

因此,如果我们直接使用自己熟悉的语言编写代码来进行接口测试将会更加灵活方便，这里我们将使用 Python 来进行接口测试。

# Requests

进行接口测试需要发送 HTTP 请求，Python 最基础的 HTTP 库有 Urllib、Httplib2、Requests、Treq 等，这里我们推荐使用 `Requests` 库来进行接口测试。

Requests 基于 urllib，采用 Apache2 Licensed 开源协议的 HTTP 库。它比 urllib 更加方便，可以节约我们大量的工作，完全满足 HTTP 测试需求。目前很多 Python 爬虫也使用 `Requests` 库。

# 官方文档摘要

Requests 口号为：“让 HTTP 服务人类”

Requests 中文文档

Requests github 项目主页

Requests 唯一的一个非转基因的 Python HTTP 库，人类可以安全享用。

警告：非专业使用其他 HTTP 库会导致危险的副作用，包括：安全缺陷症、冗余代码症、重新发明轮子症、啃文档症、抑郁、头疼、甚至死亡。

用户见证

Twitter、Spotify、Microsoft、Amazon、Lyft、BuzzFeed、Reddit、NSA、女王殿下的政府、Amazon、Google、Twilio、Mozilla、Heroku、PayPal、NPR、Obama for America、Transifex、Native Instruments、Washington Post、Twitter、SoundCloud、Kippt、Readability、以及若干不愿公开身份的联邦政府机构都在内部使用。

功能特性

- Keep-Alive & 连接池
- 国际化域名和 URL
- 带持久 Cookie 的会话
- 浏览器式的 SSL 认证
- 自动内容解码
- 基本/摘要式的身份认证
- 优雅的 key/value Cookie
- 自动解压
- Unicode 响应体
- HTTP(S) 代理支持
- 文件分块上传
- 流下载
- 连接超时
- 分块请求
- 支持 .netrc（用户配置脚本文件）

# Requests 安装

使用 pip 安装命令如下：

```
pip install requests
```

安装检测

打开 cmd 窗口，输入 python 然后导入 requests 如果安装成功没有任何提示

```
import requests
```

如果提示如下内容则说明安装失败

```
ImportError: No module named 'requests'
```

如果没有安装 pip 的参考：Python 安装与配置

# Requests 基础应用

## 发送不同类型 HTTP 请求

requests 库内置了不同的方法来发送不同类型的 http 请求，用法如下所示：

request_basic.py

```
import requests

base_url='http://httpbin.org'

#发送GET 类型的请求
r=requests.get(base_url+'/get')
print(r.status_code)

#发送Post 类型请求
r=requests.post(base_url+'/post')
print(r.status_code)
```

```
#发送 PUT 类型请求
r=requests.put(base_url+'/put')
print(r.status_code)

#发送 Delete 类型请求
r=requests.delete(base_url+'/delete')
print(r.status_code)
```

执行结果，200 是状态码表示发送请求成功。

```
200
200
200
200
```

# 参数传递

## 传递 URL 参数

一般在 GET 请求中我们使用查询字符串(query string)来进行参数传递，在 requests 库中使用方法如下：

request_basic.py

```
import requests

base_url='http://httpbin.org'

param_data={'user':'zxw','password':'666'}
r=requests.get(base_url+'/get',params=param_data)
print(r.url)
print(r.status_code)
```

执行结果

```
C:\Python35\python.exe D:/api_test/requests_api_test/params.py
http://httpbin.org/get?user=zxw&password=666
200
Process finished with exit code 0
```

## 传递 body 参数

在 Post 请求中，一般参数都在请求体（Request body）中传递，在 Requests 中用法如下：

```
form_data = {'user': 'zxw', 'passwd': '8888'}
r=requests.post(base_url+'/post',data=form_data)
print(r.text)
```

执行结果：

```
{
        "args": {},
        "data": "",
        "files": {},
        "form": {
                "passwd": "8888",
                "user": "zxw"
        },
        "headers": {
                "Accept": "*/*",
                "Accept-Encoding": "gzip, deflate",
                "Connection": "close",
                "Content-Length": "20",
                "Content-Type": "application/x-www-form-urlencoded",
                "Host": "httpbin.org",
                "User-Agent": "python-requests/2.18.4"
        },
        "json": null,
        "origin": "110.52.4.225",
        "url": "http://httpbin.org/post"
}
```

# 请求头定制

如果你想为请求添加 HTTP 头部，只要简单地传递一个 `dict` 给 `headers` 参数就可以了。

用法如下：

```
form_data = {'user': 'zxw', 'passwd': '8888'}
header={'user-agent':'Mozilla/5.0'}
r=requests.post(base_url+'/post',data=form_data,headers=header)
```

```
print(r.text)
```

返回值

```
{
        "args": {},
        "data": "",
        "files": {},
        "form": {
                "passwd": "8888",
                "user": "zxw"
        },
        "headers": {
                "Accept": "*/*",
                "Accept-Encoding": "gzip, deflate",
                "Connection": "close",
                "Content-Length": "20",
                "Content-Type": "application/x-www-form-urlencoded",
                "Host": "httpbin.org",
                "User-Agent": "Mozilla/5.0"
        },
        "json": null,
        "origin": "110.52.2.106",
        "url": "http://httpbin.org/post"
}
```

Tips：很多爬虫程序都会定制 headers 来避免被封,如下面爬取知乎页面元素就设置了请求头。

```
headers={'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.181 Safari/537.36'
}
r = requests.get("https://www.zhihu.com/explore",headers=headers)
print(r.text)
```

# 响应内容

当请求发送成功之后，我们可以获取响应内容。如响应状态码，响应头信息、响应体内容。

```
form_data = {'user': 'zxw', 'passwd': '8888'}
header={'user-agent':'Mozilla/5.0'}
r=requests.post(base_url+'/post',data=form_data,headers=header)

#获取响应状态码
print(r.status_code)
```

```
#获取响应头信息
print(r.headers)
#获取响应内容
print(r.text)
#将响应的内容以 Json 格式返回
print(r.json())
```

返回结果

```
200
{'Access-Control-Allow-Origin': '*', 'Content-Length': '342', 'Access-Control-Allow-Credentials': 'true',
'Content-Type': 'application/json', 'Connection': 'keep-alive', 'Server': 'gunicorn/19.8.1', 'Date': 'Wed,
18 Jul 2018 06:33:55 GMT', 'Via': '1.1 vegur'}
{"args":{},"data":"","files":{},"form":{"passwd":"8888","user":"zxw"},"headers":{"Accept":"*/*","Accept-E
ncoding":"gzip,
deflate","Connection":"close","Content-Length":"20","Content-Type":"application/x-www-form-urlencoded","H
ost":"httpbin.org","User-Agent":"Mozilla/5.0"},"json":null,"origin":"110.52.2.106","url":"http://httpbin.
org/post"}

{'files': {}, 'origin': '110.52.2.106', 'json': None, 'data': '', 'url': 'http://httpbin.org/post', 'headers':
{'User-Agent': 'Mozilla/5.0', 'Content-Length': '20', 'Accept': '*/*', 'Content-Type':
'application/x-www-form-urlencoded', 'Host': 'httpbin.org', 'Connection': 'close', 'Accept-Encoding': 'gzip,
deflate'}, 'form': {'passwd': '8888', 'user': 'zxw'}, 'args': {}}
```

# Requests 进阶应用

## Cookie 设置

### 设置 cookie

通过 cookies 参数可以设置 Cookie

request_advance.py

```
import requests

cookie={'user':'51zxw'}
r=requests.get(base_url+'/cookies',cookies=cookie)
print(r.text)
```

运行结果：

```
{"cookies":{"user":"51zxw"}}
```

## 获取 cookie

请求百度首页，然后获取 cookie，实现如下：

```
# 获取cookie
r=requests.get('http://www.baidu.com')
print(type(r.cookies))
print(r.cookies)
for key,value in r.cookies.items():
    print(key+':'+value)
```

运行结果：

```
<class 'requests.cookies.RequestsCookieJar'>
<RequestsCookieJar[<Cookie BDORZ=27315 for .baidu.com/>]>
BDORZ:27315
```

调用了 cookies 属性即可成功得到了 Cookies，可以发现它是一个 `RequestCookieJar` 类型，然后我们用 `items()` 方法将其转化为元组组成的列表，遍历输出每一个 Cookie 的名和值，实现 Cookies 的遍历解析。

# 超时

你可以让 requests 在经过以 `timeout` 参数设定的秒数时间之后停止等待响应。防止某些请求没有响应而一直处于等待状态。

下面案例故意设置一个很小的超时时间，为了来看一下超时后的一个响应处理，但是实际测试过程中不要设置这短。

```
r=requests.get(base_url+'/cookies',cookies=cookies,timeout=0.001)
print(r.text)
```

超时响应异常

```
raise ConnectTimeout(e, request=request)
```

```
requests.exceptions.ConnectTimeout: HTTPConnectionPool(host='httpbin.org', port=80): Max retries exceeded with url: /cookies (Caused by ConnectTimeoutError(<urllib3.connection.HTTPConnection object at 0x00000152590199E8>, 'Connection to httpbin.org timed out. (connect timeout=0.001)'))
```

# 文件上传

Reqeuests 可以使用参数 files 模拟提交一些文件数据，假如有的接口需要我们上传文件，我们同样可以利用它来上传，实现非常简单，实例如下：

```
#上传文件
file={'file':open('zxw_logo.png','rb')}
r=requests.post(base_url+'/post',files=file)
print(r.text)
```

# 会话对象

### 会话(Session)

在计算机中，尤其是在网络应用中，称为“会话控制”。Session 对象存储特定用户会话所需的**属性及配置**信息。这样，当用户在应用程序的 Web 页之间跳转时，存储在 Session 对象中的变量将不会丢失，而是在整个用户会话中一直存在下去。

比如你先进行了**登录**操作，然后打开**个人中心**详情页面，个人中心详情页面如何知道展示的是刚刚登录的这个用户的信息，那么这里就需要使用 Session 来存储相关信息。

在接口测试过程中接口之间经常有依赖关系，比如下面这两个请求一个是设置 Cookie,另外一个是获取 cookie,在没有 Session 保存机制的情况下，第二个接口无法获取第一个接口设置的 Cookie 值。

```
#设置cookie
r=requests.get(base_url+'/cookies/set/user/51zxw')
print(r.text)

#获取Cookie
r=requests.get(base_url+'/cookies')
print(r.text)
```

响应数据

```
{
  "cookies": {
    "user": "51zxw"
  }
}

{
  "cookies": {}
}
```

Request 的会话对象让你能够跨请求保持某些参数。它也会在同一个 Session 实例发出的所有请求之间保持 cookie。具体使用如下：

```
#生成会话对象
s=requests.Session()

#设置Cookie
r=s.get(base_url+'/cookies/set/user/51zxw')
print(r.text)

#获取Cookie
r=s.get(base_url+'/cookies')
print(r.text)
```

运行结果

```
{
  "cookies": {
    "user": "51zxw"
  }
}

{
  "cookies": {
    "user": "51zxw"
  }
}
```

所以，利用 Session 我们可以做到模拟同一个会话，而且不用担心 Cookies 的问题，通常用于模拟登录成功之后再进行下一步的操作。

# SSL 证书验证

Requests 可以为 HTTPS 请求验证 SSL 证书，就像 web 浏览器一样。SSL 验证默认是开启的，如果证书验证失败，Requests 会抛出 `SSLError`：如果不想验证 SSL 则可以使用 `verify` 参数关闭验证 SSL。

下面是验证 12306 网站的证书。

```
r=requests.get('https://www.12306.cn')
#关闭验证 SSL
#r=requests.get('https://www.12306.cn',verify=False)
print(r.text)
```

运行结果：

```
raise SSLError(e, request=request)
requests.exceptions.SSLError: HTTPSConnectionPool(host='www.12306.cn', port=443): Max retries exceeded with
url: / (Caused by SSLError(CertificateError("hostname 'www.12306.cn' doesn't match either of 'webssl
```

Tips：12306 的证书是自己颁发给自己的，所以会出现认证失败。

# 代理设置

**代理简介**

代理（Proxy），也称网络代理，是一种特殊的网络服务，允许一个网络终端（一般为客户端）通过这个服务与另一个网络终端（一般为服务器）进行非直接的连接。

代理服务器位于客户端和访问互联网之间，服务器接收客户端的请求，然后代替客户端向目标网站发出请求，所有的流量路由均来自代理服务器的 IP 地址，从而获取到一些不能直接获取的资源。

对于有些接口，在测试的时候请求几次，能正常获取内容。但是一旦开始大规模频繁请求（如性能测试）服务器可能会开启验证，甚至直接把 IP 给封禁掉。 那么为了防止这种情况的发生，我们就需要设置代理来解决这个问题，在 Requests 中需要用到 `proxies` 这个参数，在爬虫中会常用到代理。

西刺免费代理 IP

```
#代理设置
proxies={'http':'http://219.141.153.41:80'}
r=requests.get(base_url+'/get',proxies=proxies)
print(r.text)
```

运行结果如下：可以看到 origin 参数即为我们设置的代理 ip

```
{
  "args": {},
  "headers": {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
    "Accept-Encoding": "gzip, deflate",
    "Accept-Language": "en-US,en;q=0.5",
    "Cache-Control": "max-age=86400",
    "Connection": "close",
    "Host": "httpbin.org",
    "User-Agent": "Opera/9.80 (Macintosh; Intel Mac OS X 10.6.8; U; fr) Presto/2.9.168 Version/11.52"
  },
  "origin": "219.141.153.41",
  "url": "http://httpbin.org/get"
}
```

# 身份认证

很多接口都需要身份认证，Requests 支持多种身份认证，具体使用方法如下:

下面案例主要验证了 2 种身份类型：BasicAuth 和 digestAuth

```
from requests.auth import HTTPBasicAuth
from requests.auth import HTTPDigestAuth

#身份认证-BasicAuth
r=requests.get(base_url+'/basic-auth/51zxw/8888',auth=HTTPBasicAuth('51zxw','8888'))
print(r.text)

#身份认证——DigestAuth
r=requests.get(base_url+'/digest-auth/auth/zxw/6666',auth=HTTPDigestAuth('zxw','6666'))
print(r.text)
```

运行结果：

```
{"authenticated":true,"user":"51zxw"}

{"authenticated":true,"user":"zxw"}
```

# 流式请求

有一些接口返回值比较特殊，不是单纯返回一个结果，而是多个结果,比如某个查询接口，返回值为排行榜前 10 的商品信息。

## 实践案例

请求接口如下：

```
http://httpbin.org/stream/{num}
```

num 表示返回结果集的数量，比如输入 10 则会返回 10 个下面这种不同 id 的结果


```
{"url": "http://httpbin.org/stream/10", "args": {}, "headers": {"Host": "httpbin.org", "Connection": "close",
"Accept": "application/json", "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/57.0.2987.98 Safari/537.36 LBBROWSER", "Referer": "http://httpbin.org/", "Accept-Encoding":
"gzip, deflate, sdch", "Accept-Language": "zh-CN,zh;q=0.8", "Cookie": "_gauges_unique_hour=1;
_gauges_unique_day=1; _gauges_unique_month=1; _gauges_unique_year=1; _gauges_unique=1"}, "origin":
"110.52.4.234", "id": 0}
{"url": "http://httpbin.org/stream/10", "args": {}, "headers": {"Host": "httpbin.org", "Connection": "close",
"Accept": "application/json", "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/57.0.2987.98 Safari/537.36 LBBROWSER", "Referer": "http://httpbin.org/", "Accept-Encoding":
"gzip, deflate, sdch", "Accept-Language": "zh-CN,zh;q=0.8", "Cookie": "_gauges_unique_hour=1;
_gauges_unique_day=1; _gauges_unique_month=1; _gauges_unique_year=1; _gauges_unique=1"}, "origin":
"110.52.4.234", "id": 1}
{"url": "http://httpbin.org/stream/10", "args": {}, "headers": {"Host": "httpbin.org", "Connection": "close",
"Accept": "application/json", "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/57.0.2987.98 Safari/537.36 LBBROWSER", "Referer": "http://httpbin.org/", "Accept-Encoding":
"gzip, deflate, sdch", "Accept-Language": "zh-CN,zh;q=0.8", "Cookie": "_gauges_unique_hour=1;
_gauges_unique_day=1; _gauges_unique_month=1; _gauges_unique_year=1; _gauges_unique=1"}, "origin":
"110.52.4.234", "id": 2}
更多内容省略
...
```


针对这种类型的接口我们对结果集的处理需要使用迭代方法 iter_lines()来处理,具体使用如下：

```
import json

r=requests.get(base_url+'/stream/10',stream=True)

#如果响应内容没有设置编码，则默认设置为 utf-8
if r.encoding is None:
    r.encoding='utf-8'
```

```
#对响应结果进行迭代处理
for line in r.iter_lines(decode_unicode=True):
    if line:
        data=json.loads(line)
        print(data['id'])
```

返回结果：

```
0
1
2
3
4
5
6
7
8
9
```

# 参考资料

- http://docs.python-requests.org/zh_CN/latest/user/quickstart.html#id2
- https://germey.gitbooks.io/python3webspider/content/3.2.3-高级用法.html