Requests 简介

Requests 是一个优雅而简单的 Python HTTP 库，其实 Python 内置了用于访问网络的资源模块，比如 urllib，但是它远不如 Requests 简单优雅，而且缺少了许多实用功能。所以，更推荐掌握 Requests 接口测试实战技能，这也是互联网大厂流行的接口测试利器。

下面从安装 Requests 库开始，一步步掌握接口请求构造、接口请求断言、Schema 断言、Json / XML 请求、测试用例调试、HeadCookie 处理、Jsonpath 应用、认证体系等接口测试实战技能。

安装

pip 命令安装 Requests。

```
pip install requests
```

Request 官方资料

Requests 官方文档：

https://2.python-requests.org/en/master/

接下来就会使用最流行的 Requests 进行接口测试。

接口请求构造

简介

Requests 提供了几乎所有的 HTTP 请求构造方法，以及通过传入参数的方法，对发送的请求进行定制化的配置，可以用来应对各种不同的请求场景。

HTTP 请求构造

发送 get 请求：

```
import requestsr = requests.get('
https://api.github.com/events
')
```

在请求中添加 data 参数，并发送 post 请求：

```
import requestsr = requests.post('
http://httpbin.org/post
', data = {'key':'value'})
```

在请求中添加 data 参数，并发送 put 请求：

```
import requestsr = requests.put('
http://httpbin.org/put
', data = {'key':'value'})
```

发送 delete 请求：

```
import requestsr = requests.delete('
http://httpbin.org/delete
')
```

发送 head 请求：

```
import requestsr = requests.head('
http://httpbin.org/get
')
```

发送 options 请求：

```
import requestsr = requests.options('
http://httpbin.org/get
')
```

也可以直接使用 request 函数，传入不同的 method，例如使用这个方法发送 get 请求：

```
import requestsrequests.request("get", "
http://www.baidu.com
")
```

其他重要参数

下面的参数都是非必须参数，但是如果需要对请求做额外的定制化，则需要掌握以下这些参数的作用。

- **header 参数**

通过传入 dict 定制请求头：

```
import requests
url = '
https://api.github.com/some/endpoint
'
headers = {'user-agent': 'my-app/0.0.1'}
r = requests.get(url, headers=headers)
```

- **data 参数**

发送编码为表单形式的数据单：

```
>>> payload = {'key1': 'value1', 'key2': 'value2'}>>>
 r = requests.post("
http://httpbin.org/post
", data=payload)>>>
```

```
print(r.text){  ...  "form": {    "key2": "value2",    "key1": "value1"  },  ...}
```

- **files 参数**

上传文件，dict 格式。

```
url = 'http://httpbin.org/post'
>>> files = {'file': open('report.xls', 'rb')}
>>> r = requests.post(url, files=files)
>>> r.text

{
  ...
  "files": {
    "file": "<censored...binary...data>"
  },
  ...
```

注意：建议用二进制模式(binary mode)打开文件。这是因为 Requests 可能会试图为你提供 Content-Length header，在它这样做的时候，这个值会被设为文件的字节数（bytes）。如果用文本模式(text mode)打开文件，就可能会发生错误。

- **timeout参数**

设定超时时间（秒），到达这个时间之后会停止等待响应：

```
requests.get('http://github.com', timeout=0.001)

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
requests.exceptions.Timeout:\
 HTTPConnectionPool(host='github.com', port=80):\
  Request timed out. (timeout=0.001)
```

注意：timeout 仅对连接过程有效，与响应体的下载无关。timeout 并不是整个下载响应的时间限制，而是如果服务器在 timeout 秒内没有应答，将会引发一个异常（更精确地说，是在 timeout 秒内没有从基础套接字上接收到任何字节的数据时），如果不设置 timeout，将一直等待。

- **allow_redirects 参数**

控制是否启用重定向，bool 类型，选择 True 为启用，选择 False 为禁用。

```
import requests
>>> r = requests.get('http://github.com', allow_redirects=False)
```

```
>>> r.status_code

301
```

- **proxies参数**

设置代理，dict 格式，key 值为选择的协议，可以分别设置 HTTP 请求和 HTTPS 请求的代理。

```
import requests

proxies = {
  'http': 'http://10.10.1.10:3128',
  'https': 'http://10.10.1.10:1080',
}

requests.get('https://api.github.com/events', proxies=proxies)
```

- **verify 参数**

在测试某些https请求时，可能会出现ssl错误，因此需要在方法中处理

可以传入 bool 值或者 string，默认为 True。如果设置为 False 的即为忽略对 SSL 证书的验证；反之就是需要做验证；如果传入值为 string 的话，代表指定本地的证书作为客户端证书。

从本地传入证书：

```
import requests>>> requests.get('
https://github.com
', verify='/path/to/certfile')
```

忽略对SSL证书的验证：

```
import requests>>> requests.get('
https://kennethreitz.org
', verify=False)
```

params	字典或字节序列，作为参数增加到url中
data	字典，字节序列或文件对象，作为request的内容
json	JSON格式的数据，作为request的内容
headers	字典，HTTP定制头
cookies	字典或CookieJar， request中的cookie
auth	元组，支持HTTp认证功能
files	字典类型，传输文件
timeout	设定超时时间，秒为单位
proxies	字典类型，设定访问代理服务器，可以增加登录认证
allow_redirects	重定向开关，默认为True
stream	获取内容立即下载开关，默认为True
verify	认证SSL证书开关，默认为True
cert	本地SSL证书路径