

1、requests概述

前面的课程中我们了解了requests模块是一个网络请求模块，可以帮助我们模拟成客户端去请求服务器的数据。我们今天就是主要针对这个模块进行学习。

我们可以在浏览器中抓取到这些请求与响应的内容，那么我们可以“伪造”请求吗？也就是不再通过浏览器发送这些数据，而是通过Python来模拟浏览器发送请求。答案是可行的。而Requests模块就可以完成这种功能。

- Requests 模块就是 Python 实现的简单易用的 **HTTP库**

还有其他库吗？回答也是肯定的，例如 urllib，urllib2 等模块。但是目前来说 Requests 模块是最流行的。而且也是做好用的模块。利用pip 可以非常方便的安装：

```
pip install requests
```

2、requests模块请求参数

通过前面我们使用requests模块已经知道，requests模块在发送网络请求的时候可以传递一些关键字参数，如下图我们写过的案例所示：

```
import requests

# 定义一个url地址，用变量接受
url = 'https://movie.douban.com/top250'

headers = {
    'User-Agent': 'Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0'
}

response = requests.get(url=url, headers=headers)
html_data = response.text
print(html_data)
```

那么除了这两个关键字参数，requests模块在发送网络请求的时候还有哪些关键字参数呢？接下来我们通过查看源码的方式，介绍requests模块发送请求常用的关键字参数。如下所示：

```
"""
method: 请求方法 get post
url: 请求网址
params: (可选的) 查询参数

headers: (可选的) 字典 请求头
cookies: (可选的) 字典.cookiejar对象，用户身份信息
proxies: (可选的) ip代理

data: (可选的) 字典.列表.元组.bytes post请求时会用到
json: (可选的) 字典 提交参数

verify: (可选的) 是否验证证书，ca证书
timeout: (可选的) 设置响应时间，一旦超过，程序会报错
allow_redirects: (可选的) 是否允许重定向，布尔类型数据

files: (可选的) 字典,文件
```

auth: (可选的) 字典, 权限认证
stream: (可选的) 是否是数据流传输
"""

以上关键字参数排列顺序根据使用的频率进行排列。

3、常见参数的使用

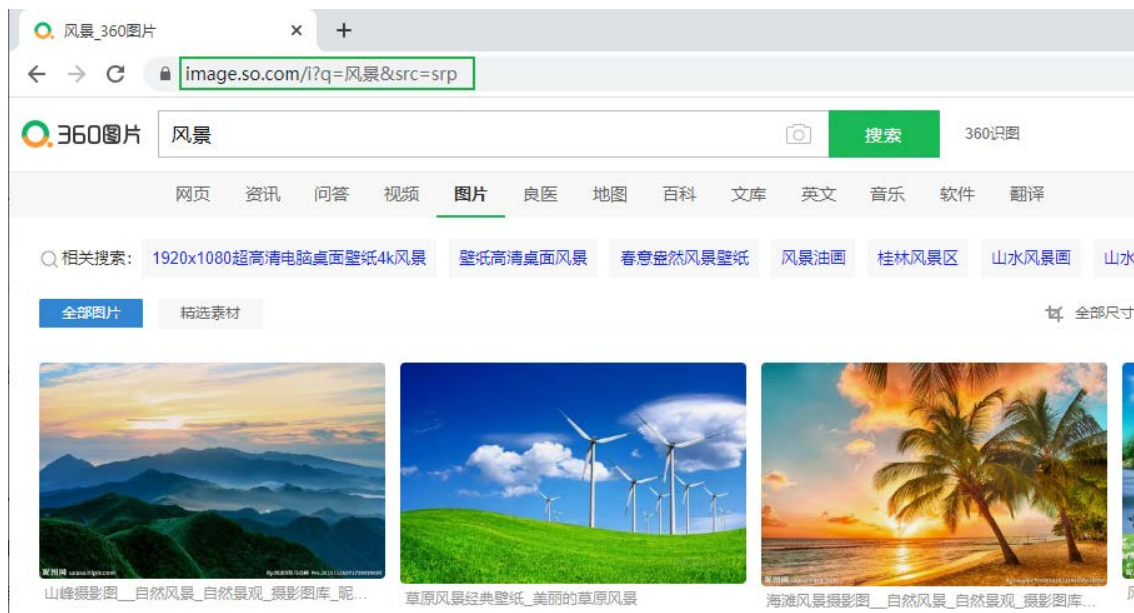
3、1 url地址中的参数

在使用requests模块发送请求之前, 我们就要回顾之前学习到的url (统一资源定位符)。

在你拿到数据所在的url地址之后, 发送网络请求时, 请求的url中包含两种地址参数: **查询参数** 和 **请求参数**。

- **查询参数**

当我们爬一些特殊网址时, 请求的url中会有一些特殊的参数, 例如以下站点:



这种是 **URL** 的查询参数。前面是网址, ? 后面的二值性数据, 就是查询参数。

URL 的查询字符串(query string)传递某种数据。如果你是手工构建 URL, 也就是通过拼接字符串构造的URL。那么数据会以键/值对的形式置于 URL 中, 跟在一个问号的后面。

例如: `https://image.so.com/i?q=%E9%A3%8E%E6%99%AF&src=srp`

params关键字参数

requests 模块允许你使用 **params** 关键字参数, 以一个字典来提供这些参数。

举例来说, 如果你想传递 `key1=value1` 和 `key2=value2` 到 `httpbin.org/get`, 那么你可以使用如下代码:

```
import requests

params = {'q': '风景', 'src': 'srp'}
response = requests.get("https://image.so.com/i", params=params)
```

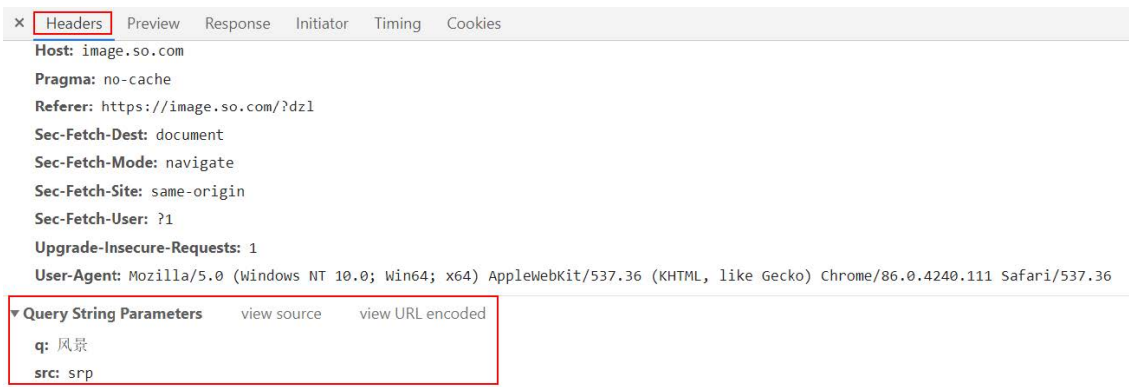
通过打印输出该 URL, 你能看到 URL 已被正确编码:

```
print(response.url)
# 打印结果
https://image.so.com/i?q=%E9%A3%8E%E6%99%AF&src=srp
```

注意:

在url地址中默认是不支持中文字符的，所以在请求中会把中文字符转化成url编码形式

同样的查询参数可以在浏览器抓包工具中找到，位于**Headers**栏目下的**Query String Parameters**中，如下图所示：

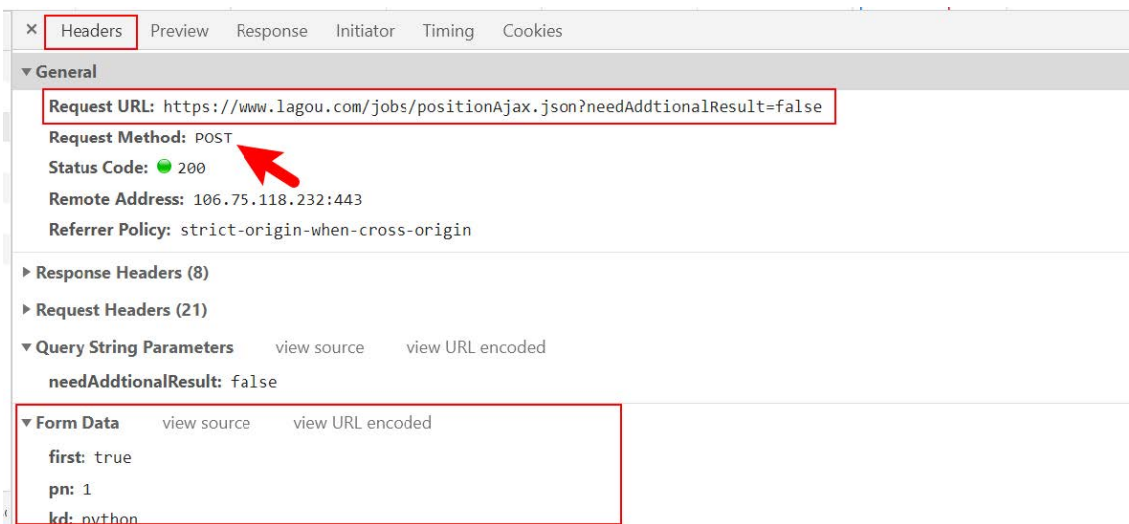


• 请求参数

请求参数和查询参数有本质的区别。请求参数一般是在发送**post**请求，向服务器提交表单数据请求的时候携带的参数。

注意：url地址中不会显示请求参数，只会显示查询参数。

请求参数在浏览器抓包工具中，位于Headers栏目下的Form Data中
如下面数据抓包所示：



data关键字参数

在requests模块中发送POST请求也是比较容易的操作，要实现这个，只需简单地传递一个字典给**data**参数。你的数据字典在发出请求时会自动编码为表单形式：

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

当然Requests中的post方法只是相对于get方法多了一个data参数，其他参数都是类似的，例如我们也可以为post中的网址添加查询字符串 params 参数，也可以像get方法一样添加 headers 参数等。

3、2 cookies关键字参数

在前面我们学习http协议的时候，请求体字段中就讲述过cookies字段，这个字段代表用户身份的标识，一般平台通过这个字段存储用户信息，包括并不限于用户名、密码、登陆时间等等。一般cookies信息都是二值型的，即key=value形式，非常像字典的构造形式。每一个key=value的信息，都代表用户的片段信息，很多个key=value片段构成一个完整的cookies字段，如下图所示：

```
cookie: RECOMMEND_TIP=true; user_trace_token=20200718183052-8303c3c5-b7a9-4631-ba50-736d67f957d9; LGUID=20200718183052-821e758d-cd3e-43ff-8d65-f2dc1d78b6a0;
_ga=GA1.2.1653410167.1595068261; JSESSIONID=ABAAABAABAGABFA111DD21E8916E03985F6E0F26BA6390F; WEBTJ-ID=20201029133929-17572de4150613-0fc7b28d6664bd-303464-2
073600-17572de4151191; sensorsdata2015session=%7B%7D; gid=GA1.2.337496302.1603949970; PRE_UTM=; PRE_LAND=https%3A%2F%2Fwww.lagou.com%2F; gat=1; Hm_lvt_42
33e74dff0ae5bd0a3d81c6ccf756e6=1603949970; LGSID=20201029133929-ae198a2c-59e4-425b-8b52-0e88b5f9c4a7; PRE_HOST=www.baidu.com; PRE_SITE=https%3A%2F%2Fwww.ba
idu.com%2Flink%3Furl%3DTHU4txh%5FeU90Gd9jWUfMeqcdJfIr%5FjAlYcM-YkV5Ju%26wd%3D%26eqid%3Db084c7b700013ad1000000065f9a558d; index_location_city=%E5%85%A8%E5%
9B%BD; TG-TRACK-CODE=index_search; X_HTTP_TOKEN=6ac31df3c13dabe04799493061e6399f5eaf91f3af; sensorsdata2015jssdkcross=%7B%22distinct_id%22%3A%22173a3e8af72
350-0b360e7d8b9c4-3323765-1764000-173a3e8af73333%22%24device_id%22%3A%22173a3e8af72350-0b360e7d8b9c4-3323765-1764000-173a3e8af73333%22%24props%22%
3A%7B%22%24latest_traffic_source_type%22%3A%22%E8%87%AA%E7%84%B6%E6%90%9C%E7%B4%A2%E6%B5%81%E9%07%8F%22%2C%22%24latest_search_keyword%22%3A%22%E6%9C%AA%E5%
8F%96%E5%88%B0%E5%80%BC%22%2C%22%24latest_referrer%22%3A%22https%3A%2F%2Fwww.baidu.com%2Flink%22%2C%22%24os%22%3A%22Windows%22%2C%22%24browser%22%3A%22Chro
me%22%2C%22%24browser_version%22%3A%2286.0.4240.111%22%7D%7D; Hm_lpvt_4233e74dff0ae5bd0a3d81c6ccf756e6=1603949976; LGRID=20201029133935-2b690a02-de2b-4f4b-
a922-097de20e891f; SEARCH_ID=11810dd644f4f56bf77ec78381bb59a
```

值得注意的是，cookies中所有片段信息，有浏览器自动生成的，也有服务器在响应数据的时候给你生成的。一旦服务给你生成了cookie片段，那么大概率就会针对这些片段进行校验，验证你的用户身份，从而决定是否返回数据给你。相反来说浏览器自动为我们生成的cookie片段对我们代码请求影响不大，因为服务器往往会校验自己生成的cookie片段，对于浏览器自动生成的cookie片段不会校验。

在requests模块中发送带cookies字段的请求也是比较简单的操作，要实现这个，只需简单地传递一个字典给cookies关键字即可：

```
cookies = {'key1': 'value1', 'key2': 'value2'}
response = requests.get("url", cookies=cookies)
```

3、3 verify关键字参数

[数字证书](#)（俗称ca证书，以下简称ca证书）为实现双方安全通信提供了电子认证。在因特网、公司内部网或外部网中，使用数字证书实现身份识别和电子信息加密。数字证书中含有密钥对（公钥和私钥）所有者的识别信息，通过验证识别信息的真伪实现对证书持有者身份的认证。

目前各大网站基本有自己的ca证书，但是不排除有的网站为了节约网站建设开销并没有购买ca证书。又因为requests模块在发送网络请求的时候，默认会验证ca证书。如果当前网站没有ca证书，那么就会报出如下的错误：

```
Traceback (most recent call last):
  File "C:/Users/admin/Desktop/学习问题/9306-01-00000000-1/1/1(1).py", line 31, in <module>
    response = requests.get(url=url)
  File "C:/Anaconda3/lib/site-packages/requests/api.py", line 76, in get
    return request('get', url, params=params, **kwargs)
  File "C:/Anaconda3/lib/site-packages/requests/api.py", line 61, in request
    return session.request(method=method, url=url, **kwargs)
  File "C:/Anaconda3/lib/site-packages/requests/sessions.py", line 530, in request
    resp = self.send(prepare, **send_kwargs)
  File "C:/Anaconda3/lib/site-packages/requests/sessions.py", line 643, in send
    r = adapter.send(request, **kwargs)
  File "C:/Anaconda3/lib/site-packages/requests/adapters.py", line 514, in send
    raise SSLError(e, request=request)
requests.exceptions.SSLError: HTTPSConnectionPool(host='data.stats.gov.cn', port=443): Max retries exceeded with url: / (Caused by SSLError(SSLError("bad handshake: Error([('SSL routines', 'tls_process_server_certificate', 'certificate verify failed')]),")))
```

如果出现上述错误，那么我们可以用verify关键字参数，在请求的时候不验证网站的ca证书


```
response = requests.get("url", verify=False)
```

如果加了verify=False这个关键字参数，使用requests模块发送请求的时候会给你弹出一个警告，警告你当前的请求可能不安全，如下图所示：

```
C:\Anaconda3\python.exe C:/Users/admin/Desktop/学员问题/0306-01-00000000-山东01(1).py
C:\Anaconda3\lib\site-packages\urllib3\connectionpool.py:986: InsecureRequestWarning: Unverified HTTPS request is being made to host
'data.stats.gov.cn'. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
InsecureRequestWarning,
```

这个警告对于后面的代码逻辑没有影响，有强迫症的可以考虑加入以下代码忽略警告：

```
import urllib3
urllib3.disable_warnings()
```

3、4 timeout关键字参数

如果对于requests模块发送的网络请求，请求数据的速度太慢，达不到你的要求。那么你可以考虑加上timeout关键字参数对于请求数据的时间做出限制，如下所示：

```
response = requests.get("url", timeout=限制时间)
```

限制时间单位以秒为单位，超过这个时间限制，程序报错。对于报错可以用异常捕获解决。

3、5 allow_redirects关键字参数

对于一些重定向的网络请求，比如登陆成功后跳转到用户个人页面。如果对于当前请求你不想要重定向，就需要当前地址数据，那么可以添加allow_redirects=False关键字，阻止当前请求的重定向。如下所示：

```
response = requests.get(url, allow_redirects=False)
```

3、6 proxies关键字参数

我们在做爬虫的过程中经常会遇到这样的情况：最初爬虫正常运行，正常抓取数据，然而一杯茶的功夫可能就会出现错误，比如403 Forbidden；这时候网页上可能会出现“您的IP访问频率太高”这样的提示，过很久之后才可能解封，但是一会后又出现这种情况。

造成这种现象的原因是该网站已采取了一些防爬虫措施。例如，服务器将在一个时间单位内检测IP请求的数量。如果超过某个阈值，服务器将直接拒绝该服务并返回一些错误信息。这种情况可以称为封IP，因此该网站成功禁止了我们的抓取工具。

想象一下，由于服务器检测到IP单位时间内的请求数量，因此我们使用某种方式来伪装IP，以使服务器无法识别由本地计算机发起的请求，因此我们可以成功地阻止IP被封。

对应的在使用requests模块发送网络请求时，可以用proxies关键字参数对本地计算机进行伪装，如下所示：

```
response = requests.get("url", proxies=代理)
```

requests模块官方文档

以下是requests模块中文文档和在GitHub上面的地址：

中文文档：https://requests.readthedocs.io/projects/cn/zh_CN/latest/

github地址：<https://github.com/requests/requests>