Requests 库基本使用教程

The website is the API...

requests第三方优秀http请求处理库，它简洁又简单。 本文参考了以下文档和资源:

- http://www.python-requests.org/en/master/user/quickstart/ (http://www.python-requests.org/en/master/user/quickstart/)
- http://www.python-requests.org/en/master/user/advanced/ (http://www.python-requests.org/en/master/user/advanced/)
- http://www.python-requests.org/en/master/user/authentication/ (http://www.python-requests.org/en/master/user/authentication/)
- https://www.icourse163.org (https://www.icourse163.org) Python网络爬虫与信息提取

1. 安装

pipenv install rquests

2. 源码下载

git clone git://github.com/requests/requests.git

3. 普通单一网页内容的获取（最简单的requests应用）

requests对urllib进行了扩展，使其更适合人们使用。我们会从下面的很多实例中感受到这一点。

In [ ]:

```
"""发送简单http请求"""

import requests
#向目标服务器发送http get请求
r = requests.get('https://api.github.com/events')
print(r.text)

#设置headers
url = 'https://www.kuaidaili.com/free/'
headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
           'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36',
          }
r = requests.get(url,headers=headers)
print(r.text)
```

requests支持的HTTP方法

requests中实现了以下HTTP方法：

- get()：请求指定的页面内容，并返回实体主体；get方法是http请求最常用的方法
- post()：向指定资源提交数据（表单 / 文件）进行处理请求，数据被包含在请求体中；post请求可能导致新的资源的建立和已有资源的改变。
- head()：类似get方法，返回的响应中没有具体内容，只有响应报头。 如果URL指向内容巨大，可以先获取head信息。
- put()：从客户端向服务器传送的数据取代指定的文档的内容（向url位置存储一个资源，覆盖原资源）。
- patch()：请求局部更新url位置的资源，即改变该处资源的部分内容。
- delete()：请求服务器删除指定页面。
- option()：允许客户端查看服务器的性能。

http/1.1 协议还有以下方法：

- connect方法：http / 1.1协议中预留给能够将连接改为管道方式的代理服务器。
- trace方法：回显服务器收到的请求，主要用于测试或诊断。

下面依次试用requests中提供的HTTP方法：

In [ ]:

```
#向目标服务器发送head请求，事前需要到httpbin.org进行设置，了解get方法对应的url。
import requests
r = requests.head('http://httpbin.org/get')
print(r.headers)
print(r.text)
```

In [ ]:

```
#向目标服务器发送post请求，事前需要到httpbin.org进行设置
import requests
```

In [ ]:

```
#发送PATCH请求
import requests
r = requests.patch('http://httpbin.org/patch',data={'key1':'value1','key2':'value2'})
print(r.headers)
print(r.text)
```

In [ ]:

```
#向目标服务器发送put/post请求
r = requests.post('http://httpbin.org/put',data = {'key1':'value1','key2':'value2'})
print(r.text)

requests.request('post','http://httpbin.org/put',data = {'key1':'value1','key2':'value2'})
print(r.text)
```

In [ ]:

```
#向目标服务器发送delete请求
import requests
r = requests.get('http://httpbin.org/get')
print(r.text)
print('-'*20)
r = requests.delete('http://httpbin.org/delete')
print(r.status_code)
print(r.text)
```

In [ ]:

```
#向目标服务器发送options请求
r = requests.options('http://httpbin.org/get')
print(r.text)
print('-'*20)
```

理解响应对象

requests中的响应对象含有更多的方法和属性，非常方便我们查看相关信息。特别是它会自动对内容进行解码，这样我们在查看响应内容时只需简单调用text或content属性即可。 对于编码问题，有时很让人头疼，大多数时候我们需要猜测服务器会响应何种编码的内容。现在我们可以使用r.apparent_encoding查看，不需要猜了。Response对象可调用的属性还有很多，例如：

- r.status_code : http请求的返回状态，200表示连接成功...
- r.text : http响应内容的字符串形式；
- r.encoding： 从http响应头重猜测的响应内容编码形式；
- r.apparent_encoding： 从内容中分析出的响应内容编码形式；
- r.content： http响应内容的二进制形式；
- r.raise_for_status()：检查返回响应代码是否为200，否则返回一个异常。
- r.headers 访问响应头信息，可以使用
- r.cookies 访问response中的cookie，可以使用r.cookies

In [ ]:

```
"""理解响应"""
import requests
r = requests.get('https://api.github.com/events')

print(r.status_code)
print('-'*20)
print(r.headers)
print('-'*20)
print(r.encoding)
print('-'*20)
print(r.apparent_encoding)
print('-'*20)
print(r.text)
print('-'*20)
print(r.content)
print('-'*20)

#响应头内容
print(r.headers)
print('-'*20)
print(r.headers['Content-Type'])
print(r.headers.get('content-type'))

#响应中的cookies
r = requests.get('http://www.baidu.com')
print(r.cookies)

# 重定向与历史
print(r.history)
```

原始响应内容

很少情况下，我们要查看来自server的原始socket响应，你可能访问r.raw。 如果你想查看，你需要在Initial request中设置stream=True。 获得原始socket后，通常会把他们存在文件里。

In [ ]:

```
""" 3.8. 原始响应内容
获得原始socket响应内容"""
import requests

r = requests.get('https://api.github.com/events',stream = True)
#尝试输出
#print(r.raw)
#print(r.raw.read(10))

#尝试输出到文件
#iter_content会帮你处理原始数据，它会自动化的解码gzip和deflate传输编码。
#raw属性是原始字节流，不能传输响应内容，如果你真的需要访问返回的字节流，就可以用raw。
#在下载流文件时，下面的方法是推荐的。chunk的大小可以自由设定，这视你的buffer大小。

filename = 'rawsocket.bin'
with open(filename,'wb') as fd:
    for chunk in r.iter_content(chunk_size=128):
        fd.write(chunk)
```

4. 构建自己的单页面爬取程序框架

经过之前的学习，已经有了一些经验，我们可以构建一个自己的简单页面爬取框架，之后的程序我们会在这个较为健壮的框架下逐步扩展。

较为健壮的程序处理流程

1. 发送请求；
2. 检查r.status_code，或调用r.raise_for_status()；
3. 若为2xx，则正常处理；
4. 若为4xx，5xx则异常处理；

In [ ]:

```
import request

def fetchUrl(url):
    try:
        r = requests.get(url)
        r.raise_for_status()
        r.encoding = r.apparent_encoding
        return r.text
    except requests.RequestError as e:
        print(e)
    except:
        return "Some exceptions were raised."
```

requests异常处理

requests库的异常处理类有以下：

- requests.ConnectionError 网络连接错误异常，如DNS查询失败，拒绝连接等；网络问题时，发送请求常导致 ConnectionError exception；
- requests.HTTPError http错误异常；当状态码不是成功状态码时（200），Response.raise_for_status()引发该异常；
- requests.URLRequired URL确实异常
- requests.TooManyRedirects 超过最大重定向次数，产生重定向异常
- requests.ConnectTimeout 连接远程服务器超时异常
- requests.Timeout 请求URL超时异常

所有异常都继承了 requests.exceptions.RequestException.

可以设置timeout参数，限制等待server响应的时间,单位是秒。 requests.get('https://github.com/ (https://github.com/)', timeout=2)

In [ ]:

```
"""异常处理的基本方法(代码框架)"""

import requests

def getHTML(url):
    try:
        r = requests.get(url,timeout = 3)
        r.raise_for_status()
        r.encoding = r.apparent_encoding
        return r.text
    except requests.HTTPError as e:
        print(e)
    except requests.RequestException as e:
        print(e)
    except:
        return "Some exceptions were raised."

if __name__ == '__main__':
    url = 'http://www.balidu.com'
    print(getHTML(url))
```

5. 应用requests库向服务器提交数据/信息

HTTP中的大多数方法都能够向服务器提交数据，例如GET方法提交查询字符串，POST方法提交表单信息等等。
url中很常见的查询串（键值对）通常由get方法发送。 使用字典方式建立参数键值对，然后令参数params = 字典名 requests 会自动对url进行编码。

访问方法中的参数详情

requests中最基本的方法是def request(method, url,**kwargs)
最好通过阅读源代码来理解或使用。

- param url: URL for the new :class:`Request` object.
- param params: (optional) Dictionary or bytes to be sent in the query string for the :class:`Request`.
- param data: (optional) Dictionary or list of tuples ``[(key, value)]`` (will be form-encoded), bytes, or file-like object to send in the body of the :class:`Request`.
- param json: (optional) json data to send in the body of the :class:`Request`.
- param headers: (optional) Dictionary of HTTP Headers to send with the :class:`Request`.
- param cookies: (optional) Dict or CookieJar object to send with the :class:`Request`.
- param files: (optional)
- param auth: (optional) Auth tuple to enable Basic/Digest/Custom HTTP Auth.
- param timeout: (optional) How many seconds to wait for the server to send data before giving up, as a float

Cookies的处理可以使用requests.cookies.RequestsCookieJar对象，它类似字典，但提供更多完成接口，方便多个域或路径的cookies cookiejars可以传递给请求，以方便认证与维持会话。

In [ ]:

```
#1 get方法中传递参数
import requests

url = 'https://www.baidu.com/s'
headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
           'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36',
          }

payload = {'wd':'北航'}
r = requests.get(url,params =payload,headers=headers)


# you can see that the URL has been correctly encoded by printing the URL:
print(r.request.url)
print(r.text)
```

In [ ]:

```
# post数据提交
import requests
payload = {'key1': 'value1', 'key2': 'value2'}
r = requests.post("https://httpbin.org/post", data=payload)
print(r.text)
```

In [ ]:

```
# 2.自定义参数
url = 'http://httpbin.org/get'
cookies = dict(cookies_are='working')
r = requests.get(url, cookies=cookies)

# 3.管理cookie
# Cookies are returned in a RequestsCookieJar,
# which acts like a dict but also offers a more complete interface,
# suitable for use over multiple domains or paths. Cookie jars can also be passed in to requests:

jar = requests.cookies.RequestsCookieJar()
jar.set('tasty_cookie', 'yum', domain='httpbin.org', path='/cookies')
jar.set('gross_cookie', 'blech', domain='httpbin.org', path='/elsewhere')
url = 'https://httpbin.org/cookies'
r = requests.get(url, cookies=jar)
r.text
```

复杂的POST请求的提交

使用字典组织表单数据，requests会将其自动编码。 这部分包括提交form表单数据、非form的data数据、json数据

3.10. POST a Multipart-Encoded File

requests能方便地上传多种编码的文件，例如excel文件。 request还允许你将字符串传给服务器，并在服务器上以文件形式保存。 如果文件很大，且以multipart/form-data方式提交请求，你可能需要stream这个请求，这时需要使用 requests-toolbelt包，requests包本身不支持。 warnings: 强烈建议以二进制形式打开文件，因为如果这个值被设置为文件字节数，requests尝试提供Content-Length头信息给你；如果你用text方式打开，那么可能引发异常。

In [ ]:

```
"""复杂的POST请求

"""
# demo 2 数据参数也可以是一个键对应多个值；这可能用于多选项。
payload_tuples = [('key1','value1'),('key1','value2')]
r1 = requests.post('https://httpbin.org/post', data=payload_tuples)
payload_dict = {'key1': ['value1', 'value2']}
r2 = requests.post('https://httpbin.org/post', data=payload_dict)
print(r1.text)
r1.text == r2.text
```

In [ ]:

```
# demo 如果你不想传递form-encoding数据，而想传递原始字符串，那么就直接使用字符串传递，而不使用字典

r3 = requests.post('https://httpbin.org/post', data='key1:value1')
print(r3.text)

# demo 4 如果希望传递JSON-Encoded数据
import json
url = 'https://httpbin.org/post'
payload = {'some':'data'}
r = requests.post(url,data=json.dumps(payload)) #json.dumps方法是将字典编码为json格式。
print(r.text)
```

In [ ]:

```
#     也可以使用新版requests中的自动json编码方式
url = 'https://httpbin.org/post'
payload = {'some':'data'}
r = requests.post(url,json=payload) #json.dumps方法是将字典编码为json格式。
print(r.text)
```

In [ ]:

```
"""3.10. POST a Multipart-Encoded File"""

import requests
url = 'https://httpbin.org/post'
#You can set the filename, content_type and headers explicitly:
files = {'file':open('test.xlsx','rb')}
#files = {'file': ('report.xls', open('report.xls', 'rb'), 'application/vnd.ms-excel', {'Expire
s': '0'})}
r = requests.post(url,files=files)
print(r.text)

# 如果需要可以将字符串送到服务器，并要求以文件形式接收
url = 'https://httpbin.org/post'

files = {'file':('report.csv','some,data,to,send\nanother,row,to,send\n')}
#files = {'file': ('report.xls', open('report.xls', 'rb'), 'application/vnd.ms-excel', {'Expire
s': '0'})}
r = requests.post(url,files=files)
print(r.text)
```

6. 非纯文本类信息的获取与解析

二进制文件下载

office文档文件、图片文件、音频文件、视频文件以及一些数据库文件都是以二进制形式存储的，一方面我们要考虑下载这些已知的二进制数据；另一方面，有一些数据的格式不知道是不是二进制的，怎么确认是二进制文件呢？如果响应对象r.content 返回b'.....'就表示响应是二进制的。 requests对于gzip / deflate transfer-encodings压缩格式，也可以自动解码。

In [ ]:

```
"""二进制响应内容（图片、文件）
响应内容的获取二进制形式的响应"""
#例如，根据响应内容，生成图片
from PIL import Image
from io import BytesIO
import requests
jpgurl = 'https://timgsa.baidu.com/timg?image&quality=80&size=b9999_10000&sec=1538158176072&di=7
b3823e700e7f0f7358a3147f97d1958&imgtype=0&src=http%3A%2F%2Fimgsrc.baidu.com%2Fimgad%2Fpic%2Fite
m%2F10dfa9ec8a1363277999f1ad9b8fa0ec08fac733.jpg'
r = requests.get(jpgurl)
#print(r.content)
i = Image.open(BytesIO(r.content))
i.save('./123.jpg')

#也可以用写二进制文件的方法。
```

解析json响应内容

requests中有一个内建的json解码器,使用r.json()就可以使用。
万一json解码失败，r.json()会抛出异常，例如响应码为204（no content），或响应包含无效的json，就会抛出ValueError：No Json object could be decoded. 错误.
注意：json解码成功时，并不显示成功，所以为了检查是否成功，使用r.raise_for_status()或检查r.status_code是否是你希望的结果。

In [ ]:

```
"""解析json响应内容 """
import requests
try:
    r = requests.get('https://api.github.com/events')
    r.raise_for_status()
    #print(r.content)
    print(r.json())
except ValueError as e:
    print(e)#no json会触发该异常
except IOError as e:
    print(e)
```

7. 响应状态码

我们可以通过响应对象的属性r.status_code查看状态码； requests内置了响应状态码，以方便参考：r.status_code == requests.codes.ok

In [5]:

```
"""3.11. 响应状态码"""
import requests

print(requests.codes.OK)
print(requests.codes.BAD)
```

```
200
400
```

8. 重定向和历史

默认requests的各种方法会执行定位重定向，但除了HEAD方法。 我们可以使用响应的history属性来跟踪重定向。 Response.histroy列表包含了为了完成请求而生成的响应对象。这个列表按时间先后顺序排序。 例如github重定向所有http到https。

如果你使用GET、OPTIONS POST PUT PATCH DELETE方法，你可以关闭重定向开关allow_redirects = False r = requests.get('https://github.com/ (https://github.com/)', allow_redirects=False)

In [9]:

```
"""使用requests查看响应的历史"""
import requests
r = requests.get('http://www.github.com/')
for h in r.history:
    print(h.url)
```

```
http://www.github.com/
https://www.github.com/
```