**Requests 轻量级爬虫**

# 发送请求

import requests

# 尝试获取某个网页

r = requests.get(' http://127.0.0.1:5000/testcss/indexcss ')

# 现在，我们有一个名为 r 的 Response 对象。我们可以从这个对象中获取所有我们想要的信息。

# Requests 简便的 API 意味着所有 HTTP 请求类型都是显而易见的。例如，你可以这样发送一个 HTTP POST 请求：

r = requests.post(' http://127.0.0.1:5000/testcss/indexcss', data={'key': 'value'})

# 其他几种类型：PUT，DELETE，HEAD 以及 OPTIONS

r = requests.put(' http://127.0.0.1:5000/testcss/indexcss', data={'key': 'value'})

r = requests.delete(' http://127.0.0.1:5000/testcss/indexcss')

#head 验证资源有效性 就是资源在不在

r = requests.head(' http://127.0.0.1:5000/testcss/indexcss')

# 该方法用于请求服务器告知其支持哪些其他的功能和方法。通过 OPTIONS 方法，

# 可以询问服务器具体支持哪些方法，或者服务器会使用什么样的方法来处理一些特殊资源。

# 可以说这是一个探测性的方法，客户端通过该方法可以在不访问服务器上实际资源的情况下就知道处理该资源的最优方式

r = requests.options(' <http://127.0.0.1:5000/testcss/indexcss>')

# 传递 URL 参数

import requests

payload = {'key1': 'value1', 'key2': 'value2'}

# 为 URL 的查询字符串(query string)传递某种数据。如果你是手工构建 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， httpbin.org/get?key=val。 Requests 允许你使用 params 关键字参数，以一个字符串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 ，那么可以使用如下代码

r = requests.get("http://127.0.0.1:5000/testcss/indexcss", params=payload)

# 输出url 观察url的变化

print(r.url)

# 注意：字典里值为 None 的键都不会被添加到 URL 的查询字符串里

payload1 = {'key1': 'value1', 'key2': None}

r = requests.get("http://127.0.0.1:5000/testcss/indexcss", params=payload1)

# 输出url 对比观察url的变化

print(r.url)

# 传入list作为参数 一般不这样做

payload = {'key1': 'value1', 'key2': ['value2', 'value3']}

r = requests.get('http://httpbin.org/get', params=payload)

print(r.url)

# 响应内容

import requests  
  
# 实例1 获取互联网信息  
r = requests.get('https://api.github.com/events')  
print(r.text)  
  
# 实例2 获取我们自己的网站  
r = requests.get('http://127.0.0.1:5000/testcss/indexcss')  
print(r.text)  
# 字符编码，一般情况下不必修改，requests自动推测  
print(r.encoding)

说明：

Requests 会自动解码来自服务器的内容。大多数 unicode 字符集都能被无缝地解码

请求发出后，Requests 会基于 HTTP 头部对响应的编码作出有根据的推测。当你访问 r.text 之时，Requests 会使用其推测的文本编码。你可以找出 Requests 使用了什么编码，并且能够使用r.encoding 属性来改变它

显示赋值：

r.encoding=‘utf-8’

# 二进制响应内容

# 实例2 获取我们自己的网站  
r = requests.get('http://127.0.0.1:5000/testcss/indexcss')  
print(r.content)

r.content以2进制形式获取数据

b'<!DOCTYPE html>\n<html lang="en">\n<head>\n <meta charset="UTF-8">\n <title>Title</title>\n <style type="text/css">\n span + p {\n background: black;\n color: red;\n }\n </style>\n</head>\n<body>\n\n<span>\xe8\xbf\x99\xe6\x98\xaf\xe4\xb8\x80\xe4\xb8\xaaspan</span>\n<p>\xe8\xbf\x99\xe6\x98\xaf\xe4\xb8\x80\xe4\xb8\xaaP</p>\n<p>\xe8\xbf\x99\xe6\x98\xaf\xe4\xb8\x80\xe4\xb8\xaaP</p>\n<div>\xe8\xbf\x99\xe6\x98\xaf\xe4\xb8\x80\xe4\xb8\xaadiv</div>\n<p>\xe8\xbf\x99\xe6\x98\xaf\xe4\xb8\x80\xe4\xb8\xaaP</p>\n</body>\n</html>'

较多使用在获取图片的场景中

import requests  
from PIL import Image  
from io import BytesIO  
  
# 获取百度图片  
r = requests.get('https://www.baidu.com/img/superlogo\_c4d7df0a003d3db9b65e9ef0fe6da1ec.png?qua=high&where=super')  
# 把2进制内容转成图形  
i = Image.open(BytesIO(r.content))  
# 显示  
i.show()

结果:



# JSON 响应内容

import requests  
  
r = requests.get('http://127.0.0.1:5000/testcss/handleAjax')  
tmp=r.text  
print(r.json())

如果 JSON 解码失败， r.json() 就会抛出一个异常。例如，响应内容是 401 (Unauthorized)，尝试访问 r.json() 将会抛出 ValueError: No JSON object could be decoded 异常。

需要注意的是，成功调用 r.json() 并\*\*不\*\*意味着响应的成功。有的服务器会在失败的响应中包含一个 JSON 对象（比如 HTTP 500 的错误细节）。这种 JSON 会被解码返回。要检查请求是否成功，请使用 r.raise\_for\_status() 或者检查 r.status\_code 是否和你的期望相同

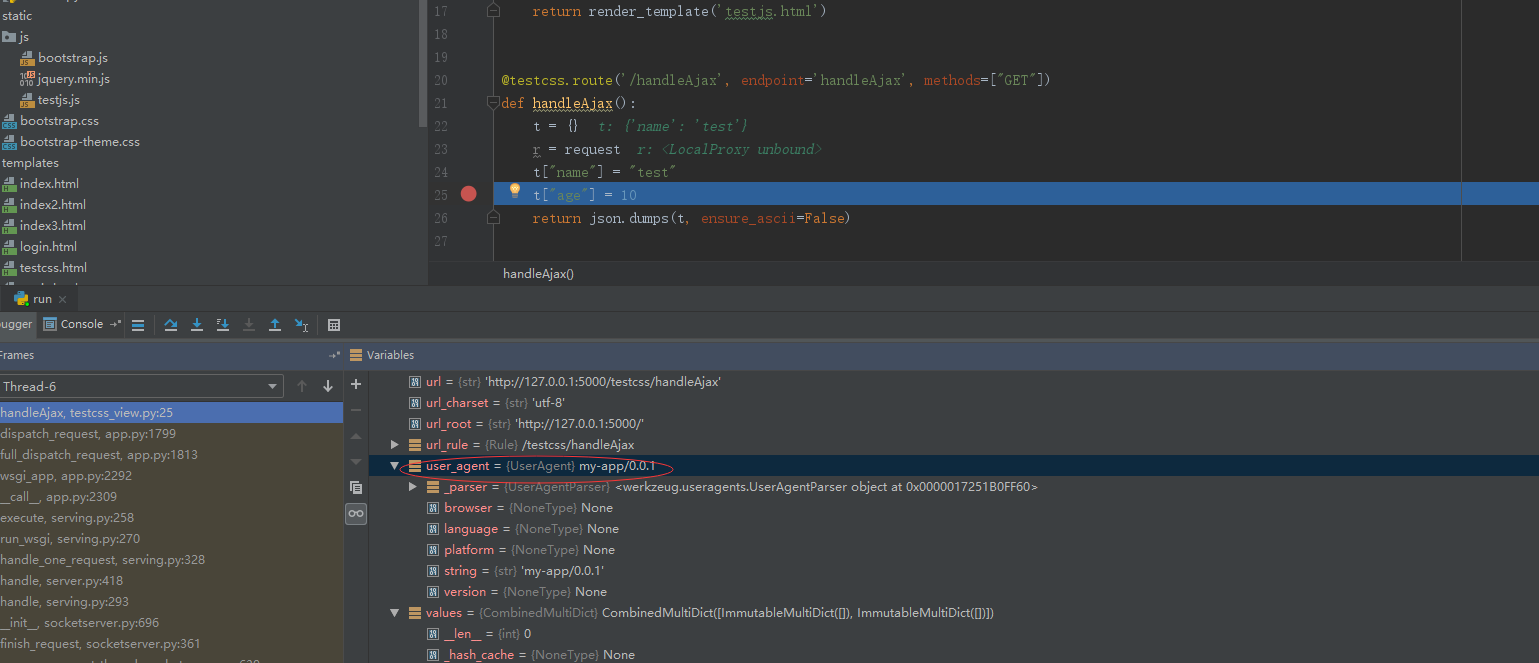
总之：当出现错误的时候，把r.text输出，看内容是否是键值对

# 原始响应内容

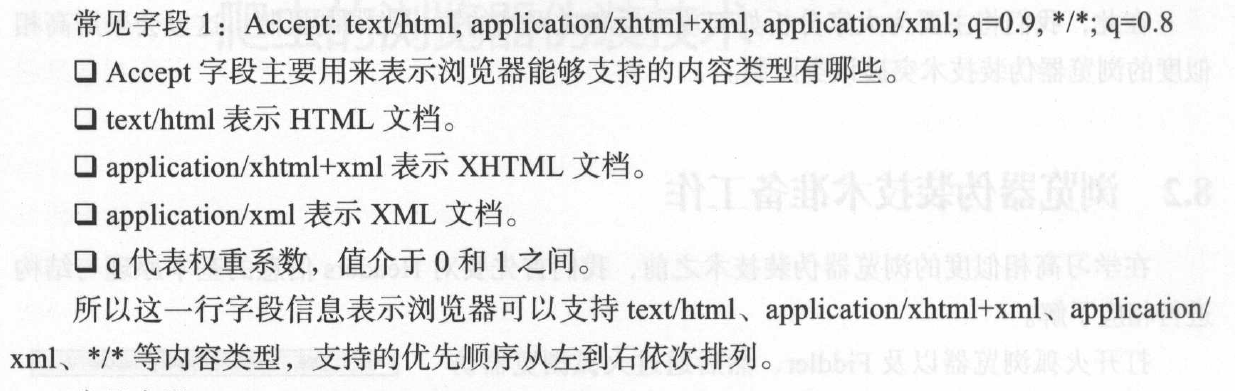
# 在罕见的情况下，可能想获取来自服务器的原始套接字响应，可以访问 r.raw。  
# 在干之前确保在初始请求中设置了 stream=True  
# 不加就取不到内容，这种方式底层是通过流式传输  
r = requests.get('http://127.0.0.1:5000/testcss/handleAjax', stream=True)  
print(r.raw)  
# 获取指定长度的内容  
print(r.raw.read(10))

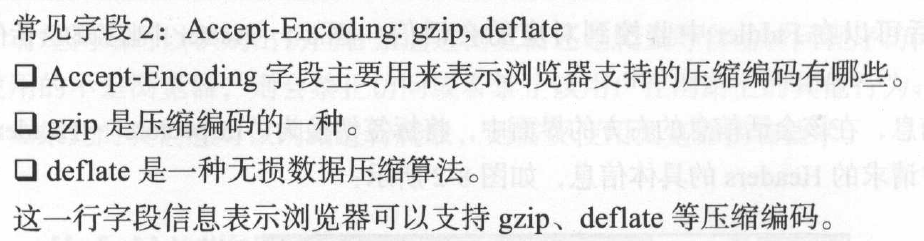
# 定制请求头

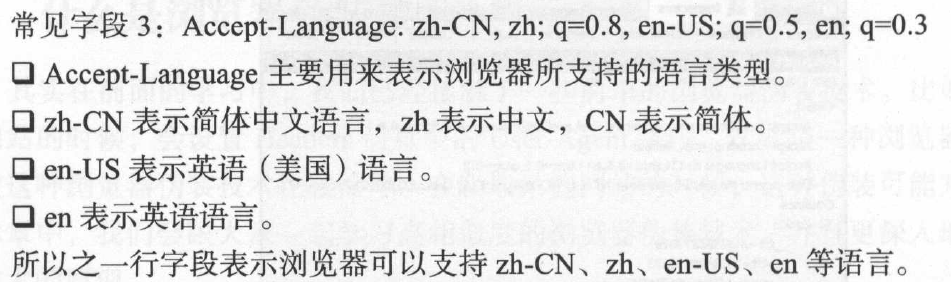
import requests  
  
# 如果你想为请求添加 HTTP 头部，只要简单地传递一个 dict 给 headers 参数就可以了。  
url = 'http://127.0.0.1:5000/testcss/handleAjax'  
headers = {'user-agent': 'my-app/0.0.1'}  
  
r = requests.get(url, headers=headers)

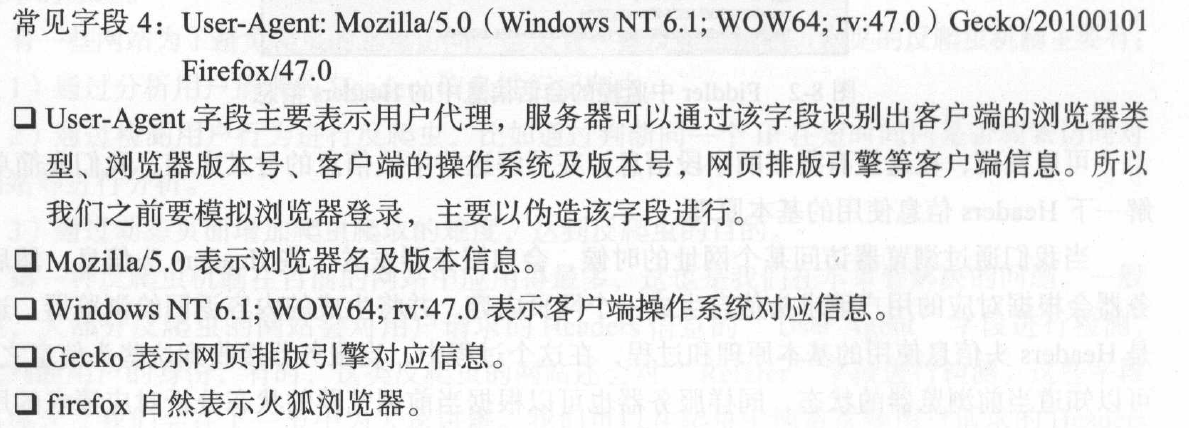


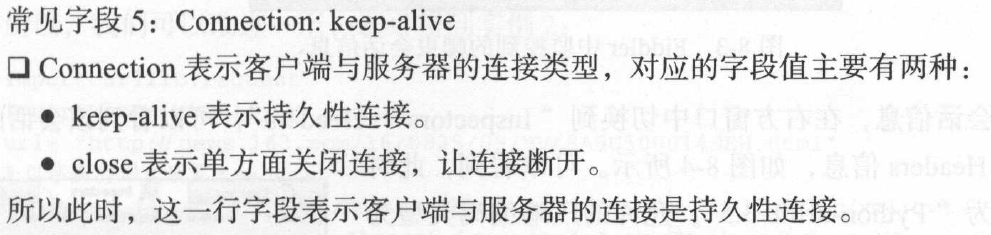
请求头的含义：

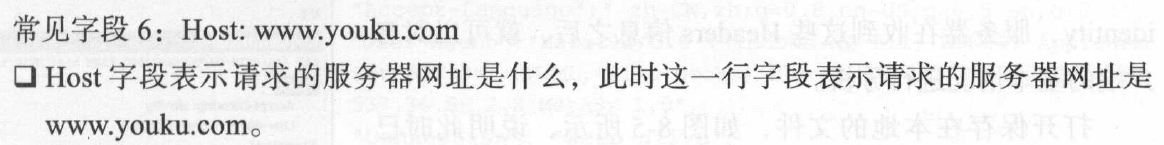


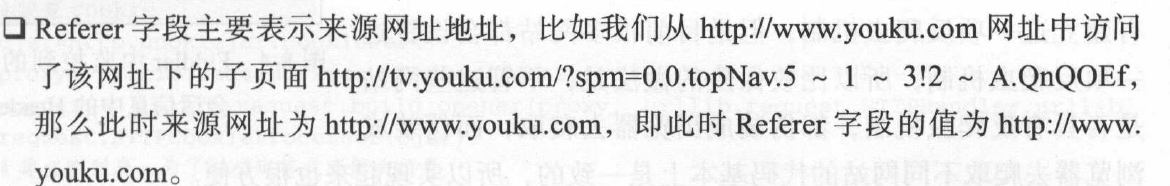










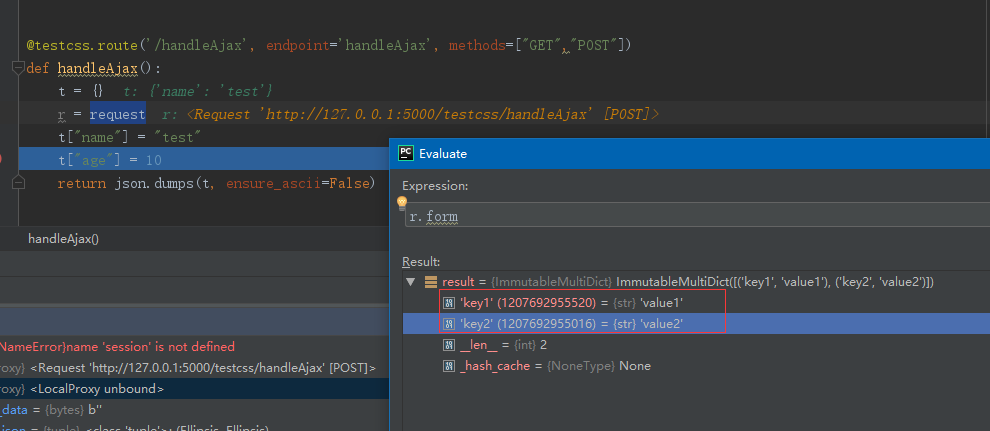


# 更加复杂的 POST 请求（9.11）

## 传递字典

payload = {'key1': 'value1', 'key2': 'value2'}  
r = requests.post("http://127.0.0.1:5000/testcss/handleAjax", data=payload)  
print(r.text)

接受端：



## 传递元组

payload = (('key1', 'value1'), ('key2', 'value2'))  
r = requests.post('http://127.0.0.1:5000/testcss/handleAjax', data=payload)  
print(r.text)

## 传递json字符串

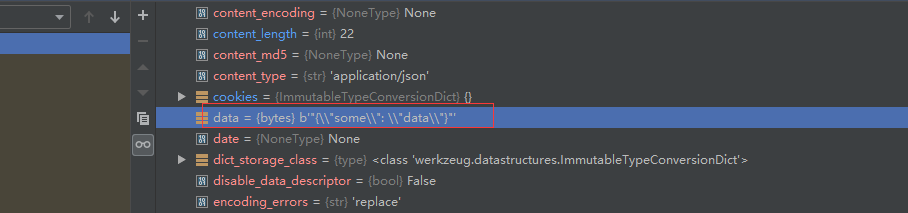
import json

import requests

payload = {'some': 'data'}

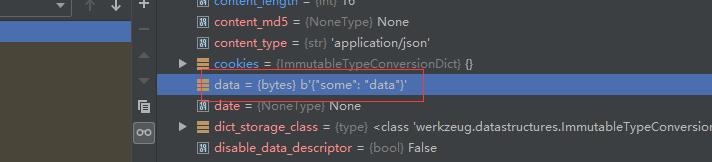
r = requests.post('http://127.0.0.1:5000/testcss/handleAjax', json=json.dumps(payload))

print(r.text)



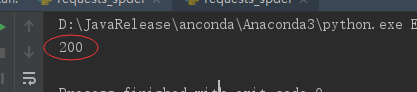
## 自动编码的json

import requests  
  
  
payload = {'some': 'data'}  
r = requests.post('http://127.0.0.1:5000/testcss/handleAjax', json=payload)  
  
print(r.text)



# 响应状态码

import requests  
  
  
payload = {'some': 'data'}  
r = requests.post('http://127.0.0.1:5000/testcss/handleAjax', json=payload)  
  
print(r.status\_code)



状态码详解：

1xx - 信息提示 （了解）  
  
这些状态代码表示临时的响应。客户端在收到常规响应之前，应准备接收一个或多个 1xx 响应。  
· 100 - Continue 初始的请求已经接受，客户应当继续发送请求的其余部分。（HTTP 1.1新）  
· 101 - Switching Protocols 服务器将遵从客户的请求转换到另外一种协议（HTTP 1.1新）

2xx - 成功  
  
这类状态代码表明服务器成功地接受了客户端请求。  
· 200 - OK 一切正常，对GET和POST请求的应答文档跟在后面。

3xx - 重定向  
  
客户端浏览器必须采取更多操作来实现请求。例如，浏览器可能不得不请求服务器上的不同的页面，或通过代理服务器重复该请求。  
· 300 - Multiple Choices 客户请求的文档可以在多个位置找到，这些位置已经在返回的文档内列出。如果服务器要提出优先选择，则应该在Location应答头指明。  
· 301 - Moved Permanently 客户请求的文档在其他地方，新的URL在Location头中给出，浏览器应该自动地访问新的URL。（后面要使用）

4xx - 客户端错误

 400 - Bad Request 请求出现语法错误

· 401 - Unauthorized 访问被拒绝，客户试图未经授权访问受密码保护的页面。  
  
· 403 - Forbidden 资源不可用。服务器理解客户的请求，但拒绝处理它。通常由于服务器上文件或目录的权限设置导致。禁止访问：IIS 定义了许多不同的 403 错误，它们指明更为具体的错误原因：  
· 403.1 - 执行访问被禁止。  
· 403.2 - 读访问被禁止。  
· 403.3 - 写访问被禁止。  
· 403.4 - 要求 SSL。  
· 403.5 - 要求 SSL 128。  
· 403.6 - IP 地址被拒绝。  
· 403.7 - 要求客户端证书。

· 404 - Not Found 无法找到指定位置的资源。这也是一个常用的应答。

5xx - 服务器错误

服务器由于遇到错误而不能完成该请求。  
  
· 500 - Internal Server Error 服务器遇到了意料不到的情况，不能完成客户的请求。  
  
· 500.12 - 应用程序正忙于在 Web 服务器上重新启动。  
· 500.13 - Web 服务器太忙。

503 Service Unavailable：服务出错

# 响应头

import requests

payload = {'some': 'data'}

r = requests.post('http://127.0.0.1:5000/testcss/handleAjax', json=payload)

print(r.headers)

{'Content-Type': 'text/html; charset=utf-8', 'Content-Length': '27', 'Server': 'Werkzeug/0.14.1 Python/3.6.5', 'Date': 'Sat, 08 Sep 2018 04:01:51 GMT'}

# Cookie

## 1. 使用场景

验证是否登录/auth

## 2. 服务端返回和接收cookie

@testcss.route('/handlecookie', endpoint='handlecookie', methods=["GET", "POST"])

def handlecookie():

r = request

args = r.form.to\_dict()

resp = make\_response("set cookie OK")

resp.set\_cookie("test", "python")

resp.headers["Set-Cookie"] = "test=python3;"

return resp

@testcss.route('/getcookie', endpoint='getcookie', methods=["GET", "POST"])

def getcookie():

r = request

args = r.form.to\_dict()

for k in r.cookies:

print("客户端传递过来的cookie key：%s val:%s" % (k, r.cookies[k]))

resp = make\_response("set cookie OK")

return resp

## 3. 客户端获取和重新发送cookie到服务端

payload = {'some': 'data'}

r = requests.post('http://127.0.0.1:5000/testcss/handleAjax', json=payload)

print(r.cookies['test'])

cookies = dict(cookies\_are=r.cookies['test'])

r = requests.get('http://127.0.0.1:5000/testcss/getcookie', cookies=cookies)

print(r.text)

print(r.headers)

# 重定向与请求历史

## 重定向

在网站建设中，时常会遇到需要网页重定向的情况：

1.网站调整（如改变网页目录结构）；

2.网页被移到一个新地址；

3.网页扩展名改变(如应用需要把.php改成.Html或.shtml)。

这种情况下，如果不做重定向，则用户收藏夹或搜索引擎数据库中旧地址只能让访问客户得到一个404页面错误信息，访问流量白白丧失；再者某些注册了多个域名的网站，也需要通过重定向让访问这些域名的用户自动跳转到主站点等

在之前的实验中使用到：



默认情况下，除了 HEAD, Requests 会自动处理所有重定向

可以使用响应对象的 history 方法来追踪重定向

Response.history 是一个 Response 对象的列表，为了完成请求而创建了这些对象。这个对象列表按照从最老到最近的请求进行排序。

# 访问http协议站点

r = requests.get('http://github.com')

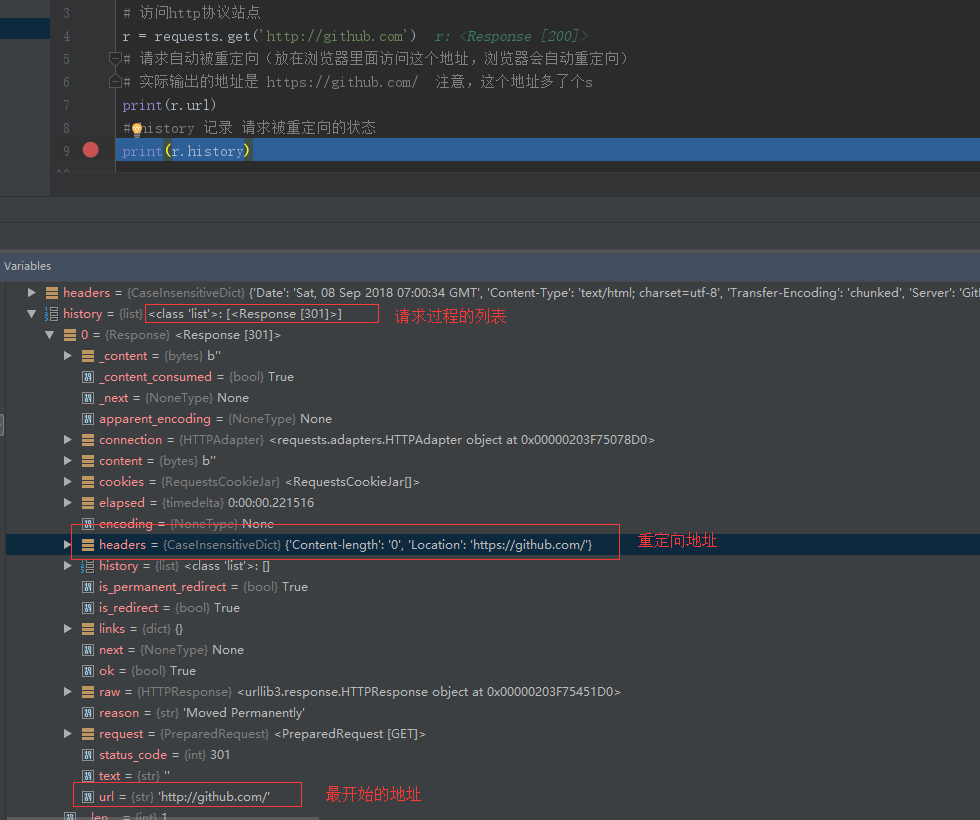
# 请求自动被重定向（放在浏览器里面访问这个地址，浏览器会自动重定向）

# 实际输出的地址是 https://github.com/ 注意，这个地址多了个s

print(r.url)

# history 记录 请求被重定向的过程

print(r.history)



禁用重定向：

在爬虫中的作用：

避免死循环。把请求定向到A，A定向到B，B又定向到A

使用GET、OPTIONS、POST、PUT、PATCH 或者 DELETE，那么可以通过 allow\_redirects 参数禁用重定向处理

# 访问http协议站点

r = requests.get('http://github.com', allow\_redirects=False)

print(r.url)

print(r.history)

使用了 HEAD，可以启用重定向

r = requests.head('http://github.com', allow\_redirects=True)

print(r.url)

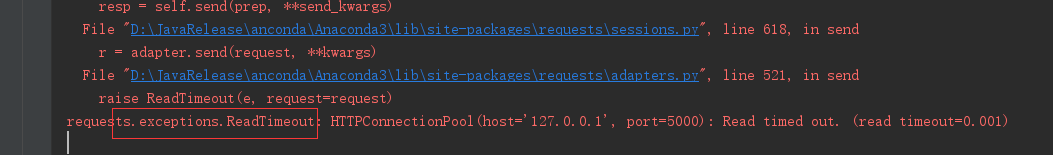
print(r.history)

# 超时

正常的商业代码都应该有超时，避免程序一直卡着不动

requests.get('http://127.0.0.1:5000/testcss/getcookie', timeout=0.001)

超时异常



# 会话对象

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， 期间使用 urllib3 的 connection pooling 功能。所以如果你向同一主机发送多个请求，底层的 TCP 连接将会被重用，从而带来显著的性能提升

## 实例1

### 客户发起登录

import requests

# 访问http协议站点

# 创建session实例

s = requests.Session()

data = {"name": "test", "pwd": "123"}

# 模拟登陆

s.post('http://127.0.0.1:5000/testlogin/login', data=data)

# 跨请求

r = s.get("http://127.0.0.1:5000/testlogin/index")

print(r.text)

### 服务端验证登录

from flask import render\_template, request, session, Blueprint, redirect

from config.config import app

testlogin = Blueprint('testlogin', \_\_name\_\_)

def wrapper(func):

def inner(\*args, \*\*kwargs):

if not session.get("user\_info"):

return redirect("/testlogin/login")

ret = func(\*args, \*\*kwargs)

return ret

return inner

@testlogin.route("/login", methods=["GET", "POST"]) # 默认接受请求方式GET

def login():

if request.method == "GET":

return render\_template("login.html")

else:

# print(request.values) #request.values包含body所有的值

name = request.form.get("name")

pwd = request.form.get("pwd")

if name == "test" and pwd == "123":

session["user\_info"] = name

# session.pop("user\_info") #删除session

# return redirect("/testlogin/index")

return app.secret\_key

else:

return render\_template("login.html", msg="用户名或者密码错误")

@testlogin.route("/index", methods=["GET", "POST"])

@wrapper

def index():

# secret\_key = request.args.get("secret\_key")

# if secret\_key != app.secret\_key:

# return "请先登录"

return render\_template("index3.html") # 渲染模板

## 实例2

### 利用session来跨请求传递参数

方法上headers和s.headers有同名的key，已方法上的为准

import requests

s = requests.Session()

s.headers.update({'x-test': '123'})

s.get('http://127.0.0.1:5000/testlogin/clientsession', headers={'x-test': '456'})

s.get('http://127.0.0.1:5000/testlogin/clientsession\_default', headers={'x-test2': '456'})

### 服务器端接收请求

@testlogin.route("/clientsession", methods=["GET", "POST"])

def clientsession():

r = request

header = dict(r.headers)

for k in header:

print("客户端传来的header key:%s val:%s" % (k, header[k]))

return app.secret\_key

@testlogin.route("/clientsession\_default", methods=["GET", "POST"])

def clientsession\_default():

r = request

header = dict(r.headers)

for k in header:

print("客户端传来的header key:%s val:%s" % (k, header[k]))

return app.secret\_key

# 请求与响应对象

任何时候进行了类似 requests.get() 的调用，都在做两件主要的事情。其一，在构建一个 Request 对象， 该对象将被发送到某个服务器请求或查询一些资源。其二，一旦 requests 得到一个从服务器返回的响应就会产生一个 Response 对象。该响应对象包含服务器返回的所有信息，也包含原来创建的 Request 对象

import requests

s = requests.Session()

s.headers.update({'x-test': '123'})

r = s.get('http://127.0.0.1:5000/testlogin/clientsession', headers={'x-test': '456'})

# 这是response header

print(r.headers)

# 发给服务器端的header

print(r.request.headers)

# 响应体内容流

# 默认情况下，当你进行网络请求后，响应体会立即被下载。

# 你可以通过 stream 参数覆盖这个行为，推迟下载响应体直到访问 Response.content、text 属性

# r = requests.get('http://127.0.0.1:5000/testcss/handleAjax', stream=True)

# stream=True，导致 Requests 无法将连接释放回连接池，因此使用 with 语句发送请求，保证请求一定会被关闭

filename = "stream.txt"

with requests.get('http://127.0.0.1:5000/testcss/handleAjax', stream=True) as r:

# text content 开始下载内容

chunk\_size = len(r.text)

print(r.content)

with open(filename, 'wb') as fd:

for chunk in r.iter\_lines(chunk\_size):

fd.write(chunk)

# 钩子函数

import requests

# 请求返回到客户端，在获取result.text 之前调用

def showUrl(r, \*args, \*\*kwargs):

# 使用场景：在不同的地方需要对请求的结果 进行 相同的 逻辑处理 可以使用钩子函数

r.text1 = "这是从服务器端返回的数据：" + r.text

hooks = dict(response=showUrl)

result = requests.get('http://127.0.0.1:5000/testcss/handleAjax', hooks=hooks)

print(result.text1)

# 代理

如果你一直用同一个代理ip爬取这个网页，很有可能ip会被禁止访问网页,所以需要使用代理ip

## 获取ip地址

# -\*- coding: utf-8 -\*-

import requests, threading, datetime

from bs4 import BeautifulSoup

import random

"""

1、抓取西刺代理网站的代理ip

2、并根据指定的目标url,对抓取到ip的有效性进行验证

3、最后存到指定的path

"""

# ------------------------------------------------------文档处理--------------------------------------------------------

# 写入文档

def write(path, text):

with open(path, 'a', encoding='utf-8') as f:

f.writelines(text)

f.write('\n')

# 清空文档

def truncatefile(path):

with open(path, 'a', encoding='utf-8') as f:

f.truncate()

# 读取文档

def read(path):

with open(path, 'r', encoding='utf-8') as f:

txt = []

for s in f.readlines():

txt.append(s.strip())

return txt

# ----------------------------------------------------------------------------------------------------------------------

# 计算时间差,格式: 时分秒

def gettimediff(start, end):

seconds = (end - start).seconds

m, s = divmod(seconds, 60)

h, m = divmod(m, 60)

diff = ("%02d:%02d:%02d" % (h, m, s))

return diff

# ----------------------------------------------------------------------------------------------------------------------

# 返回一个随机的请求头 headers

def getheaders():

user\_agent\_list = [ \

"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/22.0.1207.1 Safari/537.1" \

"Mozilla/5.0 (X11; CrOS i686 2268.111.0) AppleWebKit/536.11 (KHTML, like Gecko) Chrome/20.0.1132.57 Safari/536.11", \

"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.6 (KHTML, like Gecko) Chrome/20.0.1092.0 Safari/536.6", \

"Mozilla/5.0 (Windows NT 6.2) AppleWebKit/536.6 (KHTML, like Gecko) Chrome/20.0.1090.0 Safari/536.6", \

"Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/19.77.34.5 Safari/537.1", \

"Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/536.5 (KHTML, like Gecko) Chrome/19.0.1084.9 Safari/536.5", \

"Mozilla/5.0 (Windows NT 6.0) AppleWebKit/536.5 (KHTML, like Gecko) Chrome/19.0.1084.36 Safari/536.5", \

"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1063.0 Safari/536.3", \

"Mozilla/5.0 (Windows NT 5.1) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1063.0 Safari/536.3", \

"Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_8\_0) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1063.0 Safari/536.3", \

"Mozilla/5.0 (Windows NT 6.2) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1062.0 Safari/536.3", \

"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1062.0 Safari/536.3", \

"Mozilla/5.0 (Windows NT 6.2) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1061.1 Safari/536.3", \

"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1061.1 Safari/536.3", \

"Mozilla/5.0 (Windows NT 6.1) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1061.1 Safari/536.3", \

"Mozilla/5.0 (Windows NT 6.2) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1061.0 Safari/536.3", \

"Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/535.24 (KHTML, like Gecko) Chrome/19.0.1055.1 Safari/535.24", \

"Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/535.24 (KHTML, like Gecko) Chrome/19.0.1055.1 Safari/535.24"

]

UserAgent = random.choice(user\_agent\_list)

headers = {'User-Agent': UserAgent}

return headers

# -----------------------------------------------------检查ip是否可用----------------------------------------------------

def checkip(targeturl, ip):

headers = getheaders() # 定制请求头

proxies = {"http": "http://" + ip, "https": "http://" + ip} # 代理ip

try:

#代理ip访问的是域名 不能使用localhost 使用ngrok.exe 解决

targeturl = " http://8ad392c7.ngrok.io/testcss/handleAjax"

response = requests.get(url=targeturl, proxies=proxies, headers=headers, timeout=5).status\_code

if response == 200:

return True

else:

return False

except:

return False

def findip(type, pagenum, targeturl, path): # ip类型,页码,目标url,存放ip的路径

list = {'1': 'http://www.xicidaili.com/nt/', # xicidaili国内普通代理

'2': 'http://www.xicidaili.com/nn/', # xicidaili国内高匿代理

'3': 'http://www.xicidaili.com/wn/', # xicidaili国内https代理

'4': 'http://www.xicidaili.com/wt/'} # xicidaili国外http代理

url = list[str(type)] + str(pagenum) # 配置url

headers = getheaders() # 定制请求头

html = requests.get(url=url, headers=headers, timeout=5).text

soup = BeautifulSoup(html, 'lxml')

all = soup.find\_all('tr', class\_='odd')

for i in all:

t = i.find\_all('td')

ip = t[1].text + ':' + t[2].text

is\_avail = checkip(targeturl, ip)

if is\_avail == True:

write(path=path, text=ip)

print(ip)

def getip(targeturl, path):

truncatefile(path) # 爬取前清空文档

start = datetime.datetime.now() # 开始时间

threads = []

for type in range(4): # 四种类型ip,每种类型取前三页,共12条线程

for pagenum in range(3):

t = threading.Thread(target=findip, args=(type + 1, pagenum + 1, targeturl, path))

threads.append(t)

print('开始爬取代理ip')

for s in threads: # 开启多线程爬取

s.start()

for e in threads: # 等待所有线程结束

e.join()

print('爬取完成')

end = datetime.datetime.now() # 结束时间

diff = gettimediff(start, end) # 计算耗时

ips = read(path) # 读取爬到的ip数量

print('一共爬取代理ip: %s 个,共耗时: %s \n' % (len(ips), diff))

# -------------------------------------------------------启动-----------------------------------------------------------

if \_\_name\_\_ == '\_\_main\_\_':

for i in range(4):

print(i)

path = 'ip.txt' # 存放爬取ip的文档path

targeturl = 'http://www.cnblogs.com/TurboWay/' # 验证ip有效性的指定url

getip(targeturl, path)

## 服务端验证

这里显示：代理ip在访问这个路由

@testcss.route('/handleAjax', endpoint='handleAjax', methods=["GET", "POST"])

def handleAjax():

t = {}

r = request

args = r.form.to\_dict()

resp = make\_response("set cookie OK")

resp.set\_cookie("test", "python")

resp.headers["Set-Cookie"] = "test=python3;"

tmp = list(r.access\_route)

for i in tmp:

print("访问本路由的ip是:%s" % i)

return resp

## 客户端拿到ip后的使用方式

其实就是在验证ip的时候的使用方式

import requests

proxies = {

"http": "http://222.221.11.119:3128",

"http": "http://118.190.217.61:80",

}

r=requests.get("http://127.0.0.1:5000/testcss/handleAjax", proxies=proxies)

print(r.text)