Cookies

讲师：潘sir

Cookie原理

产生原因：

- http每次请求都是独立的，但某些操作需要保证用户是登录状态，所以使用cookie进行关联

真实场景：

- 输入账号/密码进行登录，登录后需要访问登录之后的接口，服务端需要校验你是否已登录，则通过cookies识别你是否已经登录，即：第一个请求不需要cookies，第二个请求需要cookies，则需要进行关联cookies

实现过程：

- 登录之后，服务端返回一个cookies，保存到电脑本地，下一次发请求的时候，浏览器通过缓存地址从缓存中取出cookies，使用cookies去提交请求，服务端通过cookies识别到你是否已经登录

实例演示：

1. 开启charles抓包，过滤127
2. 使用账号密码，登录http://127.0.0.1:5001/login
3. 跳转到user页面
4. 重发/user/panwj 请求带cookie，返回200
5. 重发/user/panwj 请求把cookie删掉，返回302，重定向到/login

解释说明

- 并不是所有的网站都使用cookies，有些使用token，实现的功能都一样
- cookie特点：
    1. 保存在客户端，一般由浏览器负责存储到电脑本地
    2. 通常是加密存储，但由于存储到本地，很难保证数据不被非法访问，并不很安全，所以cookies中不宜保存敏感信息，如密码等
    3. cookie有有效期，保存多长时间，由服务器(开发)决定
    4. cookie保存在http协议的头部，有一个Set-Cooke字段来指示浏览器或其他客户端在本地保存cookie信息
    5. cookie保存在客户端本地是为了下次访问网站的时候可以直接取出来，上送服务器，所以http协议中通过客户端发送给服务器的请求报文头中，有一个cookies域专门用于存放这个信息，以便把信息发给服务器
- 一般浏览器都会缓存cookie，为了模拟第一次请求，需要使用无痕浏览器
    - 浏览器有缓存功能，但代码不会自动缓存，所以会出现浏览器能正常访问，但代码不行

Cookie使用

实例演示：

1. 启动抓包工具：charles
2. 启动无痕浏览器，打开https://www.chsi.com.cn/ ，抓包发现get请求不带cookie，返回内容中有Set-Cookie字段
3. 再次请求：https://www.chsi.com.cn/ ，发现请求头携带了cookie

现象解释：

- 第一次请求的时候，服务端返回的cookie存到了本地，然后第二次请求，浏览器就会自动带上该cookie
- 发现cookie中有携带_ga / _gid / _gat 等字段，这些都是用作行为统计的，不用关注

代码模拟：

获取cookies的两种方式：jar格式，string格式

```
import requests

url = 'https://www.chsi.com.cn/'

h = {
    # 伪装成为浏览器
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_2)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.67 Safari/537.36",
    "Upgrade-Insecure-Requests": "1"  # 告诉服务端支持https
}
# 第一次请求，verify=False取消https请求的ssl认证
res = requests.get(url, headers=h, verify=False)
# print(res.status_code, res.text)

# 1. 获取jar格式的cookie
ck1 = res.cookies
print(ck1, type(ck1))

# # 2. 获取str格式的cookie
res_h = res.headers
print(res_h["Set-Cookie"])
```

传递cookies的三种方式：

1. 把cookie放入heders

```
import requests
```

```
url = 'https://www.chsi.com.cn/'

h = {
    # 伪装成为浏览器
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_2)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.67 Safari/537.36",
    "Upgrade-Insecure-Requests": "1"  # 不确定的参数
}
# 第一次请求
res = requests.get(url, headers=h, verify=False)
# print(res.status_code, res.text)

# 获取str格式的cookie
res_h = res.headers
print(res_h["Set-Cookie"])

# Cookie参数关联
h2 = {
    # 伪装成为浏览器
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_2)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.67 Safari/537.36",
    "Upgrade-Insecure-Requests": "1",  # 不确定的参数
    "Cookie": res_h["Set-Cookie"]
}

# 第二次带cookie请求
res2 = requests.get(url, headers=h2, verify=False)
```

2. 通过cookies关键字参数传递

```
import requests

url = 'https://www.chsi.com.cn/'

h = {
    # 伪装成为浏览器
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_2)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.67 Safari/537.36",
    "Upgrade-Insecure-Requests": "1"  # 不确定的参数
}
# 第一次请求
res = requests.get(url, headers=h, verify=False)
# print(res.status_code, res.text)

# 获取jar格式的cookie
cks = res.cookies
print(cks)
```

```
# jar格式可以转换成字典
# cks_dic = dict(cks)
# print(cks_dic)


# 第二次带cookie请求
res2 = requests.get(url, headers=h, cookies=cks, verify=False)
```

3. 构造cookies字典，传给cookies关键字参数

```
import requests

url = 'https://www.chsi.com.cn/'

h = {
    # 伪装成为浏览器
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_2)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.67 Safari/537.36",
    "Upgrade-Insecure-Requests": "1"  # 不确定的参数
}
# 第一次请求
res = requests.get(url, headers=h, verify=False)
# print(res.status_code, res.text)

# 获取str格式的cookie
res_h = res.headers
print(res_h["Set-Cookie"])

# 构造字典格式的cookie
cks = {
    "Cookie": res_h["Set-Cookie"]
}

# 第二次带cookie请求
res2 = requests.get(url, headers=h, cookies=cks, verify=False)
```

小结

- 头部信息说明：User-Agent需要模拟浏览器，否则会被封掉ip；未知参数最好填上
- 流程说明：
    1. 启动charles抓包工具
    2. 执行脚本
    3. 观察发现，第一次请求不带cookie，第二次请求带上了cookie
- 三种传输cookie的方式
    1. 获取str格式，将cks放到headers["Cookie"]中传过去

2. 获取jar格式，通过 cookies=传过去，jar格式传输，不会产生多余的值
3. 获取str格式，通过cookies={"Cookie": cks}

- 获取方式都是通过响应头的Set-Cookie获取
- Requests将verify设置为False后取消认证警告的方式，针对https请求，必须设置这个参数！

重定向

示例演示

1. 打开抓包工具charles
2. 打开：http://127.0.0.1:5001/login ，点击用户，即可触发重定向，定向到登录页面

现象解释

- 登录才能访问某些页面，若直接使用无痕浏览器访问该url，则会触发重定向302，重定向到登录页面
- 由开发决定重定向到哪里，判断未登录时候，则重定向哪里，跟本地的cookie无关

代码模拟

- 禁止重定向

```
import requests

url = 'http://127.0.0.1:5001/user/panwj'
# 直接访问，触发重定向，res.status_code 就会是最终的结果，为：200
r = requests.get(url)
print(r.status_code)

# 禁止重定向，则返回302
r = requests.get(url, allow_redirects=False)
print(r.status_code)
```

```
200
302
```

- 获取重定向地址

```python
import requests

url = 'http://127.0.0.1:5001/user/'

# 禁止重定向，则返回302
r = requests.get(url, allow_redirects=False)
print(r.status_code)

# 获取重定向后的地址:http://127.0.0.1:5001/login
loc = r.headers["Location"]
print(loc)
```

```
302
http://127.0.0.1:5001/login
```

- 获取重定向历史

```python
import requests

url = 'http://127.0.0.1:5001/user'

# 禁止重定向，则返回302
r = requests.get(url)
print(r.status_code)

# 获取重定向的历史
print(r.history)

# 获取第一个重定向的详情，Location
r1 = r.history[0]
print(r1.headers)
print(r1.cookies)

r2 = r.history[1]
print(r2.headers)
print(r2.cookies)
```

```
200
[<Response [308]>, <Response [302]>]
{'Server': 'gunicorn/19.9.0', 'Date': 'Sat, 05 Oct 2019 10:07:10 GMT',
'Content-Type': 'text/html; charset=utf-8', 'Content-Length': '261',
'Location': 'http://127.0.0.1:5001/user/', 'Connection': 'keep-alive'}
<RequestsCookieJar[]>
{'Server': 'gunicorn/19.9.0', 'Date': 'Sat, 05 Oct 2019 10:07:10 GMT',
'Content-Type': 'text/html; charset=utf-8', 'Content-Length': '219',
'Location': 'http://127.0.0.1:5001/login', 'Connection': 'keep-alive'}
<RequestsCookieJar[]>
```

登录案例

- 访问需要登录的地址：http://127.0.0.1:5001/user/panwj
- 登录需要传csrf_token，来获取有效cookie

```python
import requests, time, re

url = 'http://127.0.0.1:5001/login'

data = {
    "username": "dcs",
    "password": "123",
}
# get请求login页获取csrf_token
r = requests.get(url)
# 获取csrf_token
csrf_token = re.findall(r'csrf_token.*?value="(.*?)">-', r.text)
data["csrf_token"] = csrf_token

# 登录
r2 = requests.post(url, data=data, cookies=r.cookies)
# 获取重定向的cookie
# print(r2.history)
cks = r2.history[0].cookies

# # 访问登录后才能访问的页面
url2 = 'http://127.0.0.1:5001/user/dcs'
r3 = requests.get(url2, cookies=cks)
# 直接使用r2的cookies无法登录，因为r2是被重定向后的结果
# r3 = requests.get(url2, cookies=r2.cookies)
print(r3.status_code)
# print(r3.text)

assert '资料编辑' in r3.text, "访问失败"
```

```
200
```

- 直接手动获取

```
import requests
# 使用302返回的cookies，或者重定向后的请求使用的cookies 都可
headers = {
    "Cookie": "session=.eJwlj0mKAzEMAP_icw6WLMtWPtNYGxMCM9CdnEL-ng5zr4KqV9lyj-
OnXB_7My5lu3m5FvLGCL0uS8lYQtijQhCu5Cp95eAmIK5tzq6WKtMGcgT1SkZT3XOQq0B072JDTkir
E2g4sYqss4389J5H7P8TQOX9AdT4PrQ.EHolTA.XyoaRdRkA2_9jRQIR0fn_FFBCrQ; HttpOnly;
Path=/"
}
url = 'http://127.0.0.1:5001/user/dcs'
res = requests.get(url, headers=headers)

assert '资料编辑' in res.text, "访问失败"
```

- 练习题：通过抓包分析，找到cookies，通过cookies免登录访问：csdn -> 个人中心 -> 个人主页

```
import requests
url = 'https://www.csdn.net/nav/watchers'

# 增加cookies
headers = {
    "Cookie": "uuid_tt_dd=10_19477468940-1556098880235-322353;
dc_session_id=10_1556098880235.729866; UN=qq_36016692;
UM_distinctid=16b25b06ad36a8-016ea862239739-1f3b6652-13c680-16b25b06ad45ab;
TY_SESSION_ID=84bc1787-dfba-4acc-a17f-4ecc59e8b547;
Hm_ct_6bcd52f51e9b3dce32bec4a3997715ac=6525*1*10_19477468940-1556098880235-
322353!5744*1*qq_36016692!1788*1*PC_VC;
smidV2=2019062318045671a4330663634af34085d6367f7bf8420005d0eba4c954530;
_ga=GA1.2.1914274812.1561976739; aliyun_webUmi33d3a81942683d;
UserToken=ba3a278440944c38a933d3a81942683d;
UserNick=%E5%B0%8F%E6%BD%98dd%E5%85%84; AU=349; BT=1567579115766;
p_uid=U000000;
acw_tc=2760824b15700970758126825e9681512ba4c7dee1eabf1cc1c2ae9c3529d3;
Hm_lvt_6bcd52f51e9b3dce32bec4a3997715ac=1570265000,1570266310,1570268386,15702
80803; hasSub=true; Hm_lpvt_6bcd52f51e9b3dce32bec4a3997715ac=1570281734;
dc_tos=pywl51"
}
r = requests.get(url, headers=headers,verify=False)

# print(r.text)
assert "评论" in r.text, '访问失败'
```

```
/Users/panwj/.local/share/virtualenvs/mypython3-Xz6Q90qj/lib/python3.7/site-packages/urllib3/connectionpool.py:852: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  InsecureRequestWarning)
```

```
```