requests模块处理cookie相关的请求

> 网站经常利用请求头中的Cookie字段来做用户访问状态的保持，那么我们可以在headers参数中添加Cookie，模拟普通用户的请求

学习目标：

- 掌握 headers中携带cookie
- 掌握 cookies参数的使用
- 掌握 cookieJar的转换方法

后端请求过程

请求过程（原理）

第一次请求过程

1.我们的浏览器第一次请求服务器的时候，不会携带任何cookie信息

2.服务器接受到请求之后，发现 请求中没有任何cookie信息

3.服务器设置一个cookie信息，这个cookie设置在响应中

4.我们的浏览器接受到这个响应之后，发现响应中有cookie信息，浏览器会将这个cookie信息保存起来

第二次及其之后的请求过程

5.当我们的浏览器第二次及其之后的请求都会携带cookie信息，cookie信息是携带在请求头中的

6.我们的服务器接收到请求之后，发现请求中的cookie信息，就知道是谁的请求了

cookie记录了用户访问的状态信息

1. 反爬
2. 模拟登录（cookie记录的信息是登录相关记录）登录的cookie拿过来，发请求获取响应

1.在headers参数中携带cookie

1. 打开浏览器，右键-检查，点击Net work，勾选Preserve log
2. 访问github登陆的url地址 https://movie.douban.com/
3. 输入账号密码点击登陆后，刷新首页
4. 确定url之后，再确定发送该请求所需要的请求头信息中的User-Agent和Cookie

```
import requests

url = 'https://movie.douban.com/'
```

```
# 构造请求头字典
headers = {
    # 从浏览器中复制过来的User-Agent
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/67.0.3396.87 Safari/537.36',
    # 从浏览器中复制过来的Cookie
    'Cookie': 'xxx这里是复制过来的cookie字符串'
}

# 请求头参数字典中携带cookie字符串
resp = requests.get(url, headers=headers)

print(resp.text)    # 在输出结果中，看到有用户信息，说明是获取登录之后的信息
```

总结:

带上cookie的好处:

能够访问登录后的页面

正常的浏览器在请求服务器的时候会带上cookie（第一次请求除外），所以对方服务器有可能会通过是否携带cookie来判断我们是否是一个爬虫，对应的能起到一定的反爬效果

带上cookie的坏处:

一套cookie往往对应的是一个用户的信息，请求太频繁有更大可能性被对方识别为爬虫

那么，面对这种情况如何解决---->使用多个账号

2.cookies参数的使用

1. cookies参数的形式：字典

```
cookies = {"cookie的name":"cookie的value"}
```

   - 该字典对应请求头中Cookie字符串，以分号、空格分割每一对字典键值对
   - 等号左边的是一个cookie的name，对应cookies字典的key
   - 等号右边对应cookies字典的value

2. cookies参数的使用方法

```
response = requests.get(url, cookies)
```

3. 将cookie字符串转换为cookies参数所需的字典：

```
cookies_dict = {cookie.split('=')[0]:cookie.split('=')[-1] for cookie in cookies_str.split('; ')}
```

4. 注意：**cookie一般是有过期时间的，一旦过期需要重新获取**

```
import requests

url = 'https://movie.douban.com/'

# 构造请求头字典
headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/67.0.3396.87 Safari/537.36'
}
# 构造cookies字典
cookies_str = '从浏览器中copy过来的cookies字符串'

cookies_dict = {cookie.split('=')[0]:cookie.split('=')[-1] for cookie in cookies_str.split('; ')}

# 请求头参数字典中携带cookie字符串
resp = requests.get(url, headers=headers, cookies=cookies_dict)

print(resp.text)
```

3.cookieJar对象转换为cookies字典的方法

> 使用requests获取的resposne对象，具有cookies属性。该属性值是一个cookieJar类型，包含了对方服务器设置在本地的cookie。我们如何将其转换为cookies字典呢?

requests.utils.dict_from_cookiejar 把cookiejar对象转化为字典

> respose获取cookies （cookie是不是我们需要的cookie)

```
import requests

url = "https://www.baidu.com"
response = requests.get(url)
print(type(response.cookies)  # cookiejar类型

cookies = requests.utils.dict_from_cookiejar(response.cookies)
print(cookies)  # 字典类型
```

在前面的requests的session类中，我们不需要处理cookie的任何细节，如果有需要，我们可以使用上述方法来解决

4. 超时参数的使用

在平时冲浪的过程中，我们经常会遇到网络波动，这个时候，一个请求等了很久可能任然没有结果。

对应的，在爬虫中，一个请求很久没有获取结果，就会让整个项目的效率变得非常低，这个时候，我们就需要对请求进行强制要求，让他必须在特定时间内返回结果，否则就报错

使用方法如下：

```
response = requests.get(url, timeout=3)  # 设定是否三秒内有响应内容，如果没有，直接报错
```

通过添加timeout参数，能够保证在3秒钟之类返回响应，否则就报错

这个方法还能用来检测IP代理的质量，如果一个代理IP在很长时间没有响应，那么添加超时参数，通过报错，达到筛选IP的目的

5. retrying模块的使用

21:02上课

上述方法能够加快我们整体的请求速度，但是在正常的网页浏览浏览过程中，如果发生速度很慢的情况，我们会点击刷新页面，那么，在代码中，我们是否也能刷新请求呢?

retrying模块的地址：https://pypi.org/project/retrying/

retrying模块的使用

使用retrying模块提供的retry方法

通过装饰器的方式使用，让被装饰的函数反复执行

retry中可以传入参数，stop_max_attempt_number让函数报错后继续重新执行，达到最大执行次数的上限，如果每次都报错，整个函数报错，如果中间有一个成功，程序继续往后执行

> retrying第三方模块
>
> pip install retrying -i 换源地址

代码参考：

```
import requests
from retrying import retry


@retry(stop_max_attempt_number=3)
def _parse_url(url):

```

```
    # 前面加_代表私有，这里代表私有方法，其他文件调用此包，此属性不能被调用
    print('*' * 20)
    headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl\
            eWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.116 Safari/537.36"}
    response = requests.get(url, headers=headers, timeout=3)
    # 断言：状态码为200，否则报错
    assert response.status_code == 200
    return response.content.decode()


def parse_url(url):
    try:
        html_str = _parse_url(url)
    except Exception as e:
        print(e)
        html_str = None
    return html_str


if __name__ == '__main__':
    # url = 'http://www.baidu.com'
    url = 'www.baidu.com'
    # print(parse_url(url)[:20])  # 字符串切片，取前二十个字符
    print(parse_url(url))
```