Day06回顾

cookie模拟登陆

```
1、适用网站类型：爬取网站页面时需要登录后才能访问，否则获取不到页面的实际响应数据
2、方法1（利用cookie）
    1、先登录成功1次,获取到携带登陆信息的Cookie（处理headers）
    2、利用处理的headers向URL地址发请求
3、方法2（利用session会话保持）
    1、实例化session对象
        session = requests.session()
    2、先post : session.post(post_url,data=post_data,headers=headers)
        1、登陆，找到POST地址：form -> action对应地址
        2、定义字典，创建session实例发送请求
            # 字典key ： <input>标签中name的值(email,password)
            # post_data = {'email':'','password':''}
    3、再get : session.get(url,headers=headers)
```

设置断点调试

```
1、抓取到js文件的相关代码
2、单击前面的行号,刷新页面,会执行JS到断点处
3、鼠标移动到相关位置,可进行函数的跳转
```

在程序中执行js文件

```
import execjs

with open('文件名.js','r') as f:
    js_data = f.read()

js_obj = execjs.compile(js_data)
result = js_obj.eval('js函数名("参数")')
```

增量爬取思路

```
1、将爬取过的地址存放到数据库中
2、程序爬取时先到数据库中查询比对，如果已经爬过则不会继续爬取
```

Day07笔记

多线程爬虫

应用场景

```
1、多进程 ： CPU密集程序
2、多线程 ： 爬虫(网络I/O)、本地磁盘I/O
```

知识点回顾

- 队列

```python
# 导入模块
from queue import Queue
# 使用
q = Queue()
q.put(url)
q.get() # 当队列为空时，阻塞
q.empty() # 判断队列是否为空，True/False
```

- 线程模块

```python
# 导入模块
from threading import Thread

# 使用流程
t = Thread(target=函数名) # 创建线程对象
t.start() # 创建并启动线程
t.join()  # 阻塞等待回收线程

# 创建多线程
t_list = []
for i in range(5):
    t = Thread(target=函数名)
```

```
    t_list.append(t)
    t.start()

for i in t_list:
    i.join()
```

小米应用商店抓取(多线程)

- **目标**

```
1、网址 ：百度搜 - 小米应用商店，进入官网
2、目标 ：应用分类 - 聊天社交
    应用名称
    应用链接
```

- **实现步骤**

1、确认是否为动态加载

```
1、页面局部刷新
2、右键查看网页源代码，搜索关键字未搜到
# 此网站为动态加载网站，需要抓取网络数据包分析
```

2、F12抓取网络数据包

```
1、抓取返回json数据的URL地址（Headers中的Request URL）
    http://app.mi.com/categotyAllListApi?page={}&categoryId=2&pageSize=30

2、查看并分析查询参数（headers中的Query String Parameters）
    page: 1
    categoryId: 2
    pageSize: 30
    # 只有page再变，0 1 2 3 ... ... ，这样我们就可以通过控制page的直拼接多个返回json数据的URL地址
```

- 代码实现

```
import requests
from threading import Thread
from queue import Queue
import json
import time

class XiaomiSpider(object):
    def __init__(self):
        self.headers = {'User-Agent':'Mozilla/5.0'}
        self.url = 'http://app.mi.com/categotyAllListApi?page={}&categoryId=2&pageSize=30'
        # 定义队列，用来存放URL地址
        self.url_queue = Queue()
```

```python

    # URL入队列
    def url_in(self):
        # 拼接多个URL地址,然后put()到队列中
        for i in range(67):
            self.url.format((str(i)))
            self.url_queue.put(self.url)

    # 线程事件函数(请求,解析提取数据)
    def get_page(self):
        # 先get()URL地址,发请求
        # json模块做解析
        while True:
            # 当队列不为空时,获取url地址
            if not self.url_queue.empty():
                url = self.url_queue.get()
                html = requests.get(url,headers=self.headers).text
                self.parse_page(html)
            else:
                break
    # 解析函数
    def parse_page(self,html):
        app_json = json.loads(html)
        for app in app_json['data']:
            # 应用名称
            name = app['displayName']
            # 应用链接
            link = 'http://app.mi.com/details?id={}'.format(app['packageName'])
            d = { '名称' : name,'链接' : link }
            print(d)

    # 主函数
    def main(self):
        self.url_in()
        # 存放所有线程的列表
        t_list = []

        for i in range(10):
            t = Thread(target=self.get_page)
            t.start()
            t_list.append(t)

        # 统一回收线程
        for p in t_list:
            p.join()

if __name__ == '__main__':
    start = time.time()
    spider = XiaomiSpider()
    spider.main()
    end = time.time()
    print('执行时间:%.2f' % (end-start))
```

json解析模块

json.loads(json格式字符串)

- 作用

```
把json格式的字符串转为Python数据类型
```

- 示例

```
html_json = json.loads(res.text)
```

json.dump(python,f,ensure_ascii=False)

- 作用

```
把python数据类型 转为 json格式的字符串
# 一般让你把抓取的数据保存为json文件时使用
```

- 参数说明

```
第1个参数：python类型的数据(字典，列表等)
第2个参数：文件对象
第3个参数：ensure_ascii=False # 序列化时编码
```

- 示例

```
import json

app_dict = {
    '应用名称' : 'QQ',
    '应用链接' : 'http://qq.com'
}
with open('小米.json','a') as f:
    json.dump(app_dict,f,ensure_ascii=False)
```

练习: 将链家二手房代码改写为多线程方式

selenium+phantomjs/Chrome/Firefox

selenium

- **定义**

```
1、Web自动化测试工具，可运行在浏览器,根据指令操作浏览器
2、只是工具，必须与第三方浏览器结合使用
```

- **安装**

```
Linux: sudo pip3 install selenium
Windows: python -m pip install selenium
```

phantomjs浏览器

- **定义**

```
无界面浏览器(又称无头浏览器)，在内存中进行页面加载,高效
```

- **安装(phantomjs、chromedriver、geckodriver)**

Windows

```
1、下载对应版本的phantomjs、chromedriver、geckodriver
2、把chromedriver.exe拷贝到python安装目录的Scripts目录下(添加到系统环境变量)
   # 查看python安装路径: where python
3、验证
   cmd命令行: chromedriver

# 下载地址
chromedriver ： 下载对应版本
http://chromedriver.storage.googleapis.com/index.html

geckodriver
https://github.com/mozilla/geckodriver/releases
```

Linux

```
1、下载后解压
   tar -zxvf geckodriver.tar.gz
2、拷贝解压后文件到 /usr/bin/  (添加环境变量)
   sudo cp geckodriver /usr/bin/
3、更改权限
   sudo -i
   cd /usr/bin/
   chmod 777 geckodriver
```

- **使用**

示例代码一：使用 selenium+浏览器 打开百度

```
from selenium import webdriver
import time

browser = webdriver.Chrome()
browser.get('http://www.baidu.com/')
browser.save_screenshot('baidu.png')
browser.quit()
```

示例代码二：打开百度，搜索赵丽颖，查看

```
from selenium import webdriver
import time

browser = webdriver.Chrome()
browser.get('http://www.baidu.com/')

# 向搜索框(id kw)输入 赵丽颖
ele = browser.find_element_by_xpath('//*[@id="kw"]')
ele.send_keys('赵丽颖')

time.sleep(1)
# 点击 百度一下 按钮(id su)
su = browser.find_element_by_xpath('//*[@id="su"]')
su.click()

# 截图
browser.save_screenshot('赵丽颖.png')
# 关闭浏览器
browser.quit()
```

- 浏览器对象(browser)方法

```
1、browser = webdriver.Chrome(executable_path='path')
2、browser.get(url)
3、browser.page_source # 查看响应内容
4、browser.page_source.find('字符串')
   # 从html源码中搜索指定字符串,没有找到返回: -1
5、browser.quit() # 关闭浏览器
```

- 定位节点

单元素查找(1个节点对象)

```
1、browser.find_element_by_id('')
2、browser.find_element_by_name('')
3、browser.find_element_by_class_name('')
4、browser.find_element_by_xpath('')
... ...
```

多元素查找([节点对象列表])

```
1、browser.find_elements_by_id('')
2、browser.find_elements_by_name('')
3、browser.find_elements_by_class_name('')
4、browser.find_elements_by_xpath('')
... ...
```

- 节点对象操作

```
1、ele.send_keys('') # 搜索框发送内容
2、ele.click()
3、ele.text          # 获取文本内容
4、ele.get_attribute('src') # 获取属性值
```

京东爬虫案例

- 目标

```
1、目标网址 ： https://www.jd.com/
2、抓取目标 ： 商品名称、商品价格、评价数量、商品商家
```

- 思路提醒

```
1、打开京东，到商品搜索页
2、匹配所有商品节点对象列表
3、把节点对象的文本内容取出来，查看规律，是否有更好的处理办法?
4、提取完1页后，判断如果不是最后1页，则点击下一页
   # 如何判断是否为最后1页？？？
```

- 实现步骤

1. **找节点**

```
1、首页搜索框 ： //*[@id="key"]
2、首页搜索按钮   ://*[@id="search"]/div/div[2]/button
3、商品页的 商品信息节点对象列表 ://*[@id="J_goodsList"]/ul/li
```

2. **执行JS脚本，获取动态加载数据**

```
browser.execute_script(
    'window.scrollTo(0,document.body.scrollHeight)'
)
```

3. **代码实现**

```
from selenium import webdriver
import time
```

```python

class JdSpider(object):
    def __init__(self):
        self.browser = webdriver.Chrome()
        self.url = 'https://www.jd.com/'
        self.i = 0

    # 获取商品页面
    def get_page(self):
        self.browser.get(self.url)
        # 找2个节点
        self.browser.find_element_by_xpath('//*[@id="key"]').send_keys('爬虫书籍')
        self.browser.find_element_by_xpath('//*[@id="search"]/div/div[2]/button').click()
        time.sleep(2)

    # 解析页面
    def parse_page(self):
        # 把下拉菜单拉到底部,执行JS脚本
        self.browser.execute_script(
            'window.scrollTo(0,document.body.scrollHeight)'
        )
        time.sleep(2)
        # 匹配所有商品节点对象列表
        li_list = self.browser.find_elements_by_xpath('//*[@id="J_goodsList"]/ul/li')
        for li in li_list:
            li_info = li.text.split('\n')
            if li_info[0][0:2] == '每满':
                price = li_info[1]
                name =li_info[2]
                commit = li_info[3]
                market = li_info[4]
            else:
                price = li_info[0]
                name = li_info[1]
                commit = li_info[2]
                market = li_info[3]
            print('\033[31m**********************************\033[0m')
            print(price)
            print(commit)
            print(market)
            print(name)
            self.i += 1

    # 主函数
    def main(self):
        self.get_page()
        while True:
            self.parse_page()
            # 判断是否该点击下一页,没有找到说明不是最后一页
            if self.browser.page_source.find('pn-next disabled') == -1:
                self.browser.find_element_by_class_name('pn-next').click()
                time.sleep(2)
            else:
                break
        print(self.i)

if __name__ == '__main__':
```

```
    spider = JdSpider()
    spider.main()
```

chromedriver设置无界面模式

```
from selenium import webdriver

options = webdriver.ChromeOptions()
# 添加无界面参数
options.add_argument('--headless')
browser = webdriver.Chrome(options=options)
browser.get('http://www.baidu.com/')
browser.save_screenshot('baidu.png')
```

scrapy框架

- 定义

```
异步处理框架,可配置和可扩展程度非常高,Python中使用最广泛的爬虫框架
```

- 安装

```
# Ubuntu安装
1、安装依赖包
    1、sudo apt-get install libffi-dev
    2、sudo apt-get install libssl-dev
    3、sudo apt-get install libxml2-dev
    4、sudo apt-get install python3-dev
    5、sudo apt-get install libxslt1-dev
    6、sudo apt-get install zlib1g-dev
    7、sudo pip3 install -I -U service_identity
2、安装scrapy框架
    1、sudo pip3 install Scrapy
```

```
# Windows安装
cmd命令行(管理员): python -m pip install Scrapy
```

- Scrapy框架五大组件

```
1、引擎(Engine)       : 整个框架核心
2、调度器(Scheduler) : 维护请求队列
3、下载器(Downloader): 获取响应对象
4、爬虫文件(Spider)  : 数据解析提取
5、项目管道(Pipeline): 数据入库处理
**********
# 下载器中间件(Downloader Middlewares) : 引擎->下载器,包装请求(随机代理等)
# 蜘蛛中间件(Spider Middlewares) : 引擎->爬虫文件,可修改响应对象属性
```

- scrapy爬虫工作流程

```
# 爬虫项目启动
1、由引擎向爬虫程序索要第一个要爬取的URL,交给调度器去入队列
2、调度器处理请求后出队列,通过下载器中间件交给下载器去下载
3、下载器得到响应对象后,通过蜘蛛中间件交给爬虫程序
4、爬虫程序进行数据提取:
    1、数据交给管道文件去入库处理
    2、对于需要继续跟进的URL,再次交给调度器入队列, 依次循环
```

- scrapy常用命令

```
# 1、创建爬虫项目
scrapy startproject 项目名
# 2、创建爬虫文件
scrapy genspider 爬虫名 域名
# 3、运行爬虫
scrapy crawl 爬虫名
```

- scrapy项目目录结构

```
Baidu                   # 项目文件夹
├── Baidu               # 项目目录
│   ├── items.py        # 定义数据结构
│   ├── middlewares.py  # 中间件
│   ├── pipelines.py    # 数据处理
│   ├── settings.py     # 全局配置
│   └── spiders
│       ├── baidu.py    # 爬虫文件
└── scrapy.cfg          # 项目基本配置文件
```

- 全局配置文件settings.py详解

```
# 1、定义User-Agent
USER_AGENT = 'Mozilla/5.0'
# 2、是否遵循robots协议, 一般设置为False
ROBOTSTXT_OBEY = False
# 3、最大并发量, 默认为16
CONCURRENT_REQUESTS = 32
# 4、下载延迟时间
DOWNLOAD_DELAY = 1
# 5、请求头, 此处也可以添加User-Agent
DEFAULT_REQUEST_HEADERS={}
# 6、项目管道
```

```
ITEM_PIPELINES={
    '项目目录名.pipelines.类名':300
}
```

- 创建爬虫项目步骤

```
1、新建项目 ： scrapy startproject 项目名
2、cd 项目文件夹
3、新建爬虫文件 ： scrapy genspider 文件名 域名
4、明确目标(items.py)
5、写爬虫程序(文件名.py)
6、管道文件(pipelines.py)
7、全局配置(settings.py)
8、运行爬虫 ： scrapy crawl 爬虫名

```

- pycharm运行爬虫项目

```
1、创建begin.py(和scrapy.cfg文件同目录)
2、begin.py中内容:
    from scrapy import cmdline
    cmdline.execute('scrapy crawl maoyan'.split())
```

今日作业

1、熟记多线程爬虫原理及会写，更改腾讯招聘项目为多线程爬虫

2、熟记如下问题

```
1、scrapy框架有哪几大组件?
2、各个组件之间是如何工作的?
```

3、Windows安装scrapy

```
Windows ： python -m pip install Scrapy
# Error: Microsoft Visual C++ 14.0 is required
# 解决 ： 下载安装 Microsoft Visual C++ 14.0
```