Day05回顾

- **Ajax动态加载数据抓取流程**

```
【1】F12打开控制台，执行页面动作抓取网络数据包

【2】抓取json文件URL地址
  2.1) 控制台中 XHR ： 异步加载的数据包
  2.2) XHR -> Query String Parameters(查询参数)
```

- **json模块**

```
【1】抓取的json数据转为python数据类型
   1.1) html = json.loads('[{},{},{}]')
   1.2) html = requests.get(url=url,headers=headers).json()
   1.3) html = requests.post(url=url,headers=headers).json()

【2】抓取数据保存到json文件
   import json
   with open('xxx.json','w') as f:
       json.dump([{},{},{}],f,ensure_ascii=False)
```

- **数据抓取最终梳理**

```
【1】响应内容中存在
  1.1) 确认抓取数据在响应内容中是否存在

  1.2) 分析页面结构，观察URL地址规律
       a) 大体查看响应内容结构,查看是否有更改 --（百度视频案例）
       b) 查看页面跳转时URL地址变化,查看是否新跳转 --（民政部案例）

  1.3) 开始码代码进行数据抓取

【2】响应内容中不存在
  2.1) 确认抓取数据在响应内容中是否存在

  2.2) F12抓包,开始刷新页面或执行某些行为,主要查看XHR异步加载数据包
       a) GET请求: Request Headers、Query String Paramters
       b) POST请求:Request Headers、FormData

  2.3) 观察查询参数或者Form表单数据规律,如果需要进行进一步抓包分析处理
       a) 比如有道翻译的 salt+sign,抓取并分析JS做进一步处理
       b) 此处注意请求头中的cookie和referer以及User-Agent

  2.4) 使用res.json()获取数据,利用列表或者字典的方法获取所需数据
```

- **多线程爬虫思路梳理**

```
【1】所用到的模块
    1.1) from threading import Thread
    1.2) from threading import Lock
    1.3) from queue import Queue

【2】整体思路
    2.1) 创建URL队列: q = Queue()
    2.2) 产生URL地址,放入队列: q.put(url)
    2.3) 线程事件函数: 从队列中获取地址,开始抓取: url = q.get()
    2.4) 创建多线程,并运行

【3】代码结构
    def __init__(self):
        """创建URL队列"""
        self.q = Queue()

    def url_in(self):
        """生成待爬取的URL地址,入队列"""
        pass

    def parse_html(self):
        """线程事件函数,获取地址,进行数据抓取"""
        while True:
            if not self.q.empty():
                url = self.q.get()
            else:
                break

    def run(self):
        self.url_in()
        t_list = []
        for i in range(3):
            t = Thread(target=self.parse_html)
            t_list.append(t)
            t.start()

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

Day06笔记

多线程爬虫

- **多线程回顾**

```
【1】应用场景
    1.1) 多进程 : CPU密集程序
```

```
    1.2) 多线程 ： 爬虫(网络I/O)、本地磁盘I/O

【2】队列要点：q.get()防止阻塞方式
    2.1) 方法1：q.get(block=False)
    2.2) 方法2：q.get(block=True,timeout=3)
    2.3) 方法3：
        if not q.empty():
            q.get()

【3】线程锁的使用：
    from threading import Lock
    lock = Lock()
    lock.acquire()
    lock.release()
    【注意】：上锁后再次上锁会阻塞,多线程操作共享资源时需要加锁和释放锁
```

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 http://app.mi.com/

【2】目标 ： 抓取聊天社交分类下的
    2.1) 应用名称
    2.2) 应用链接
```

- **实现步骤**

```
【1】确认是否为动态加载
    1.1) 页面局部刷新
    1.2) 右键查看网页源代码,搜索关键字未搜到,为动态加载，需要抓取网络数据包分析

【2】 F12抓取网络数据包
    2.1) 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

    2.2) 查看并分析查询参数（headers中的Query String Parameters)
         page: 1        只有page在变，0 1 2 3 ... ...
         categoryId: 2
         pageSize: 30

【3】将抓取数据保存到csv文件 - 注意线程锁问题
    from threading import Lock
    lock = Lock()
    # 加锁 + 释放锁
    lock.acquire()
    lock.release()
```

- **整体思路**

```
【1】在 __init__(self) 中创建文件对象，多线程操作此对象进行文件写入
    self.f = open('xiaomi.csv','a')
```

```
    self.writer = csv.writer(self.f)
    self.lock = Lock()

【2】每个线程抓取1页数据后将数据进行文件写入，写入文件时需要加锁
    def parse_html(self):
        app_list = []
        for xxx in xxx:
            app_list.append([name,link,typ])
        self.lock.acquire()
        self.wirter.writerows(app_list)
        self.lock.release()

【3】所有数据抓取完成关闭文件
    def run(self):
        self.f.close()
```

- **代码实现**

```
class XiaomiSpider:
    def __init__(self):
        self.url = 'http://app.mi.com/categotyAllListApi?page={}&categoryId=2&pageSize=30'
        self.headers = { 'User-Agent':UserAgent().random }
        self.q = Queue()
        self.lock = Lock()
        # 计数
        self.i = 0
        # 存入csv文件
        self.f = open('xiaomi.csv','w',encoding='utf-8')
        self.writer = csv.writer(self.f)

    # 获取html
    def get_html(self,url):
        html = requests.get(url=url,headers=self.headers).text

        return html

    # URL入队列
    def url_in(self):
        for page in range(67):
            url = self.url.format(page)
            self.q.put(url)

    # 线程事件函数
    def parse_html(self):
        while True:
            # 加锁 - 防止出现死锁(self.q中剩余1个地址,但是被多个线程判断的情况)
            self.lock.acquire()
            if not self.q.empty():
                url = self.q.get()
                print(url)
                # 获取地址成功后马上释放锁,给其他线程机会,安全前提下提升效率
                self.lock.release()
                app_list = []
                # 请求 + 解析  html: {'count':2000,'data':[{},{},{}]}
                html = self.get_html(url=url)
                print(html)
```

```
                html = json.loads(html)
                item = {}
                for one_app in html['data']:
                    item['app_name'] = one_app['displayName']
                    item['app_type'] = one_app['level1CategoryName']
                    item['app_link'] = one_app['packageName']
                    print(item)
                    # 一页的app信息
                    app_list.append( (item['app_name'],item['app_type'],item['app_link']) )
                    # 加锁+释放锁
                    self.lock.acquire()
                    self.i += 1
                    self.lock.release()
                # 把1页的数据写入到csv文件
                self.lock.acquire()
                self.writer.writerows(app_list)
                self.lock.release()
                # 简单控制一下数据抓取频率,因为我们没有代理IP,容易被封掉IP
                time.sleep(random.uniform(2,4))
            else:
                # 如果队列为空了,上面已经上锁,所以此处释放锁
                self.lock.release()
                break

    # 入口函数
    def run(self):
        # 1.先让URL地址入队列
        self.url_in()
        # 2.多线程,开始执行
        t_list = []
        for i in range(2):
            t = Thread(target=self.parse_html)
            t_list.append(t)
            t.start()

        for j in t_list:
            j.join()

        print('数量:',self.i)
        # 最终关闭文件
        self.f.close()

if __name__ == '__main__':
    start_time = time.time()
    spider = XiaomiSpider()
    spider.run()
    end_time = time.time()
    print('执行时间:%.2f' % (end_time-start_time))
```

腾讯招聘数据抓取(多线程)

- **确定URL地址及目标**

```
【1】URL: 百度搜索腾讯招聘 - 查看工作岗位
【2】目标:抓取职位的如下信息
      a> 职位名称
      b> 职位地址
      c> 职位类别（技术类、销售类...）
      d> 发布时间
      e> 工作职责
      f> 工作要求
```

- **要求与分析**

```
【1】通过查看网页源码,得知所需数据均为动态加载
【2】通过F12抓取网络数据包,进行分析
【3】一级页面抓取数据: postid
【4】二级页面抓取数据: 名称+地址+类别+时间+职责+要求
```

- **一级页面json地址**

```
"""index在变,timestamp未检查"""
https://careers.tencent.com/tencentcareer/api/post/Query?timestamp=1563912271089&countryId=&cityId=&bgIds=&productId=&categoryId=&parentCategoryId=&attrId=&keyword={}&pageIndex={}&pageSize=10&language=zh-cn&area=cn
```

- **二级页面地址**

```
"""postId在变,在一级页面中可拿到"""
https://careers.tencent.com/tencentcareer/api/post/ByPostId?timestamp=1563912374645&postId={}&language=zh-cn
```

- **多线程编写思路提示**

```
【思考】 两级页面是否需要指定两个队列分别存放?
        提示1: 建立2个队列,分别存放不同级的URL地址
        提示2: 从对列中get地址,最好使用timeout参数
```

- **代码实现**

```python
import requests
import json
import time
from fake_useragent import UserAgent
from queue import Queue
from threading import Thread,Lock
from urllib import parse
import csv

class TencentSpider(object):
    def __init__(self):
        self.one_url = 'https://careers.tencent.com/tencentcareer/api/post/Query?timestamp=1563912271089&countryId=&cityId=&bgIds=&productId=&categoryId=&parentCategoryId=&attrId=&keyword={}&pageIndex={}&pageSize=10&language=zh-cn&area=cn'
```

```
        self.two_url = 'https://careers.tencent.com/tencentcareer/api/post/ByPostId?timestamp=1563912374645&postId={}&language=zh-cn'
        self.one_q = Queue()
        self.two_q = Queue()
        self.lock1 = Lock()
        self.lock2 = Lock()
        self.i = 0
        self.f = open('tencent.csv','w',encoding='utf-8')
        self.writer = csv.writer(self.f)
        # 存放所有数据的大列表,用于writerows()方法
        self.item_list = []


    def get_html(self,url):
        headers = { 'User-Agent':UserAgent().random }
        html = requests.get(url=url,headers=headers).text
        return html

    def url_in(self):
        keyword = input('请输入职位类别:')
        keyword = parse.quote(keyword)
        total = self.get_total(keyword)
        for page in range(1, total+1):
            one_url = self.one_url.format(keyword,page)
            print(one_url)
            self.one_q.put(one_url)

    # 获取总页数
    def get_total(self,keyword):
        url = self.one_url.format(keyword,1)
        html = requests.get(url=url,headers={'User-Agent':UserAgent().random}).json()
        n = int(html['Data']['Count'])
        total = n//10 if n%10==0 else n//10+1

        return total

    # 线程1事件函数
    def parse_one_page(self):
        while True:
            self.lock1.acquire()
            if not self.one_q.empty():
                one_url = self.one_q.get()
                self.lock1.release()
                html = json.loads(self.get_html(one_url))
                for job in html['Data']['Posts']:
                    post_id = job['PostId']
                    two_url = self.two_url.format(post_id)
                    self.lock1.acquire()
                    self.two_q.put(two_url)
                    self.lock1.release()
            else:
                self.lock1.release()
                break

    # 线程2事件函数
    def parse_two_page(self):
        while True:
```

```
            try:
                self.lock2.acquire()
                two_url = self.two_q.get(block=True,timeout=3)
                self.lock2.release()
                html = json.loads(self.get_html(two_url))
                # 名称+地址+类别+时间+职责+要求
                item = {}
                item['name'] = html['Data']['RecruitPostName']
                item['address'] = html['Data']['LocationName']
                item['type'] = html['Data']['CategoryName']
                item['time'] = html['Data']['LastUpdateTime']
                item['duty'] = html['Data']['Responsibility']
                item['require'] = html['Data']['Requirement']
                # 写入csv文件数据类型: [(),(),(),...,()]
                item_tuple = (item['name'],item['duty'],item['require'])
                self.item_list.append(item_tuple)

                print(item)
                self.lock2.acquire()
                self.i += 1
                self.lock2.release()
            except Exception as e:
                self.lock2.release()
                print(e,end="")
                break

    def run(self):
        self.url_in()
        t1_list = []
        t2_list = []
        for i in range(5):
            t = Thread(target=self.parse_one_page)
            t1_list.append(t)
            t.start()

        for i in range(5):
            t = Thread(target=self.parse_two_page)
            t2_list.append(t)
            t.start()

        for t in t1_list:
            t.join()

        for t in t2_list:
            t.join()

        print('数量:',self.i)
        # 将所有数据一次性写入文件
        self.writer.writerows(self.item_list)
        self.f.close()

if __name__ == '__main__':
    start_time = time.time()
    spider = TencentSpider()
    spider.run()
    end_time = time.time()
    print('执行时间:%.2f' % (end_time-start_time))
```

cookie模拟登录

```
【1】适用网站及场景 ： 抓取需要登录才能访问的页面
```

豆瓣网登录案例

- **方法一 - 登录网站手动抓取Cookie**

```
【1】先登录成功1次,获取到携带登录信息的Cookie
    登录成功 - 我的豆瓣 - F12抓包 - 刷新主页 - 找到主页的包(一般为第1个网络数据包)

【2】headers中携带着Cookie发请求
    headers = {
        'Cookie':'',
        'User-Agent':''
    }
```

```
"""方法一代码实现"""

# 1、将url改为 个人主页的URL地址
# 2、将Cookie的值改为 登录成功的Cookie值
import requests

def login():
    url = '个人主页的URL地址'
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.116 Safari/537.36',
        'Cookie':'自己抓到的Cookie值',
    }
    html = requests.get(url=url,headers=headers).text
    # 查看html中是否包含个人主页的信息 - 比如搜索 "个人主页"
    print(html)

login()
```

- **方法二**

```
【1】原理
    1.1) 把抓取到的cookie处理为字典
    1.2) 使用requests.get()中的参数:cookies - 格式为字典

【2】处理cookie为字典
    cookies = {}
    cookies_str = 'xxxx'
    for kv in cookies_str.split('; ')
        key = kv.split('=')[0]
        value = kv.split('=')[1]
        cookies[key] = value
```

```
"""方法二代码实现"""

import requests

def login():
    url = '自己账号的个人主页'
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.116 Safari/537.36',
    }
    # 处理cookie为字典
    cookies_str = '自己抓到的Cookie'
    cookies = {}
    for kv in cookies_str.split('; '):
        key = kv.split('=')[0]
        value = kv.split('=')[1]
        cookies[key] = value

    # 确认html
    html = requests.get(url=url,headers=headers,cookies=cookies).text
    print(html)

login()
```

- **方法三 - requests模块处理Cookie**

```
【1】 思路 : requests模块提供了session类,来实现客户端和服务端的会话保持

【2】原理
    2.1) 实例化session对象 : s = requests.session()
    2.2)  让session对象发送get或者post请求
       res = session.post(url=url,data=data,headers=headers)
       res = session.get(url=url,headers=headers)

【3】思路梳理
    3.1) 浏览器原理: 访问需要登录的页面会带着之前登录过的cookie
    3.2) 程序原理: 同样带着之前登录的cookie去访问 - 由session对象完成
    3.3) 具体步骤
        a> 实例化session对象
        b> 登录网站: 由session对象发送请求,登录对应网站
        c> 访问页面: 由session对象请求需要登录才能访问的页面
```

```
【4】如何把用户名和密码信息提交给服务器
    4.1) 输入用户名和错误密码,登录1次进行抓包
    4.2) 在网络数据包中找到具体提交用户名和密码信息的地址,一般为POST请求
    4.3) 将正确的用户名和密码信息POST到网络数据包的URL地址 - Request URL
```

```python
"""方法三代码实现"""
import requests

session = requests.session()

def login():
    post_url = 'https://accounts.douban.com/j/mobile/login/basic'
    post_data = {
        'ck':'',
        'name': '自己的用户名',
        'password': '自己的密码',
        'remember': 'false',
        'ticket': '',
    }
    headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.116 Safari/537.36'}
    session.post(url=post_url,data=post_data,headers=headers)
    url = '自己个人主页的URL地址'
    html = session.get(url=url,headers=headers).text
    print(html)

login()
```

selenium+phantomjs/Chrome/Firefox

- **selenium**

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

【2】安装
    2.1) Linux: sudo pip3 install selenium
    2.2) Windows: python -m pip install selenium
```

- **phantomjs浏览器**

```
【1】定义
    phantomjs为无界面浏览器(又称无头浏览器)，在内存中进行页面加载,高效

【2】下载地址
    2.1) chromedriver ： 下载对应版本
         http://chromedriver.storage.googleapis.com/index.html
    2.2) geckodriver
         https://github.com/mozilla/geckodriver/releases
    2.3) phantomjs
```

```
        https://phantomjs.org/download.html

【3】Ubuntu安装
    3.1) 下载后解压 : tar -zxvf geckodriver.tar.gz

    3.2) 拷贝解压后文件到 /usr/bin/  (添加环境变量)
         sudo cp geckodriver /usr/bin/

    3.3) 添加可执行权限
         sudo chmod 777 /usr/bin/geckodriver

【4】Windows安装
    4.1) 下载对应版本的phantomjs、chromedriver、geckodriver
    4.2) 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
    4.3) 验证
         cmd命令行: chromedriver
```

- **示例代码**

```
"""示例代码一: 使用 selenium+浏览器 打开百度"""

# 导入seleinum的webdriver接口
from selenium import webdriver
import time

# 创建浏览器对象
browser = webdriver.Chrome()
browser.get('http://www.baidu.com/')
# 5秒钟后关闭浏览器
time.sleep(5)
browser.quit()
```

```
"""示例代码二: 打开百度, 搜索赵丽颖, 点击搜索, 查看"""

from selenium import webdriver
import time

# 1.创建浏览器对象 - 已经打开了浏览器
browser = webdriver.Chrome()
# 2.输入: http://www.baidu.com/
browser.get('http://www.baidu.com/')
# 3.找到搜索框,向这个节点发送文字: 赵丽颖
browser.find_element_by_xpath('//*[@id="kw"]').send_keys('赵丽颖')
# 4.找到 百度一下 按钮,点击一下
browser.find_element_by_xpath('//*[@id="su"]').click()
```

- **浏览器对象(browser)方法**

```
【1】browser = webdriver.Chrome(executable_path='path')
【2】browser.get(url)
【3】browser.page_source # HTML结构源码
【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.close() # 关闭当前页
【6】browser.quit() # 关闭浏览器
```

- **定位节点**

```
【1】单元素查找('结果为1个节点对象')
    1.1) browser.find_element_by_id('')
    1.2) browser.find_element_by_name('')
    1.3) browser.find_element_by_class_name('')
    1.4) browser.find_element_by_xpath('')
    1.5) browser.find_element_by_link_text('')
    ... ...

【2】多元素查找('结果为[节点对象列表]')
    2.1) browser.find_elements_by_id('')
    2.2) browser.find_elements_by_name('')
    2.3) browser.find_elements_by_class_name('')
    2.4) browser.find_elements_by_xpath('')
    ... ...
```

- **猫眼电影示例**

```python
from selenium import webdriver
import time

url = 'https://maoyan.com/board/4'
browser = webdriver.Chrome()
browser.get(url)

def get_data():
    # 基准xpath: [<selenium xxx li at xxx>,<selenium xxx li at>]
    li_list = browser.find_elements_by_xpath('//*[@id="app"]/div/div/div[1]/dl/dd')
    for li in li_list:
        item = {}
        # info_list: ['1', '霸王别姬', '主演: 张国荣', '上映时间: 1993-01-01', '9.5']
        info_list = li.text.split('\n')
        item['number'] = info_list[0]
        item['name'] = info_list[1]
        item['star'] = info_list[2]
        item['time'] = info_list[3]
        item['score'] = info_list[4]

        print(item)

while True:
    get_data()
    try:
        browser.find_element_by_link_text('下一页').click()
        time.sleep(2)
    except Exception as e:
```

```
        print('恭喜你!抓取结束')
        browser.quit()
        break
```

- **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，包含子节点和后代节点的文本内容
【4】ele.get_attribute('src') # 获取属性值
```

作业1 - 京东爬虫

- **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-参考与提示**

```
# 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
4、for循环遍历后
  名称: .//div[@class="p-name"]/a/em
  价格: .//div[@class="p-price"]
  评论: .//div[@class="p-commit"]/strong
  商家: .//div[@class="p-shopnum"]

# 2. 执行JS脚本，获取动态加载数据
browser.execute_script(
  'window.scrollTo(0,document.body.scrollHeight)'
)
```

- **代码实现**

```
browser.excute_script(
'window.scrollTo(0,document.body.scrollHeight)'
)
time.sleep(2)

```

```
【1】 搜索内容：爬虫书
   li_list = [<li1>,<li2>,...<lin>]
   for li in li_list:
       方法1：print(li.text)
       方法2：item['name']=li.find_element_by_xpath('')

【2】一定要注意给页面元素加载预留时间

【3】执行JS脚本
```

作业2

```
【1】多线程改写 - 链家二手房案例
【2】多线程改写 - 汽车之家案例
```