Day05回顾

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
```

动态加载网站数据抓取

```
1、F12打开控制台，页面动作抓取网络数据包
2、抓取json文件URL地址
# 控制台中 XHR : 异步加载的数据包
# XHR -> Query String(查询参数)
```

day06笔记

作业1 - 小米应用商店

将抓取数据保存到csv文件

```
注意多线程写入的线程锁问题
from threading import Lock
lock = Lock()
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
2、每个线程抓取数据后将数据进行文件写入，写入文件时需要加锁
3、所有数据抓取完成关闭文件
```

代码实现

```python
import requests
from threading import Thread
from queue import Queue
import time
from fake_useragent import UserAgent
from lxml import etree
from threading import Lock
import csv
import random

class XiaomiSpider(object):
  def __init__(self):
    self.url = 'http://app.mi.com/categotyAllListApi?page={}&categoryId={}&pageSize=30'
    # 存放所有URL地址的队列
    self.q = Queue()
    self.ua = UserAgent()
    self.i = 0
    # 存放所有类型id的空列表
    self.id_list = []
    self.lock = Lock()
    self.f = open('小米.csv','a',encoding='gb18030',newline='')
    self.writer = csv.writer(self.f)

  def get_cateid(self):
    # 请求
    url = 'http://app.mi.com/'
    headers = { 'User-Agent':self.ua.random }
    html = requests.get(url=url,headers=headers).text
    # 解析
    parse_html = etree.HTML(html)
    xpath_bds = '//ul[@class="category-list"]/li'
    li_list = parse_html.xpath(xpath_bds)
    for li in li_list:
      typ_name = li.xpath('./a/text()')[0]
      typ_id = li.xpath('./a/@href')[0].split('/')[-1]
      # 计算每个类型的页数
      pages = self.get_pages(typ_id)
      self.id_list.append( (typ_id,pages) )

    # 入队列
    self.url_in()

  # 获取counts的值并计算页数
  def get_pages(self,typ_id):
    # 每页返回的json数据中,都有count这个key
    url = self.url.format(0,typ_id)
    html = requests.get(url=url,headers={'User-Agent':self.ua.random}).json()
    count = html['count']
    pages = int(count) // 30 + 1

    return pages

  # url入队列
  def url_in(self):
```

```
    for id in self.id_list:
      # id为元组,('2',pages)
      for page in range(id[1]):
        url = self.url.format(page,id[0])
        print(url)
        # 把URL地址入队列
        self.q.put(url)

  # 线程事件函数: get() - 请求 - 解析 - 处理数据
  def get_data(self):
    while True:
      if not self.q.empty():
        url = self.q.get()
        headers = {'User-Agent':self.ua.random}
        html = requests.get(url=url,headers=headers).json()
        self.parse_html(html)
        # 每爬取一页随机休眠0.5秒钟
        time.sleep(random.uniform(0,1))
      else:
        break

  # 解析函数
  def parse_html(self,html):
    app_info = []
    for app in html['data']:
      # 应用名称
      name = app['displayName']
      link = 'http://app.mi.com/details?id=' + app['packageName']
      # 应用类别
      type_name = app['level1CategoryName']
      app_info.append([name,type_name,link])
      print(name,type_name,link)
      self.i += 1

    # 在文件中写入数据中进行加锁处理
    self.lock.acquire()
    self.writer.writerows(app_info)
    self.lock.release()



  # 主函数
  def main(self):
    # URL入队列
    self.get_cateid()
    t_list = []
    # 创建多个线程
    for i in range(1):
      t = Thread(target=self.get_data)
      t_list.append(t)
      t.start()

    # 回收线程
    for t in t_list:
      t.join()

    print('数量:',self.i)
```

```
112
113     # 所有数据写入完成后释放锁
114     self.f.close()
115
116 if __name__ == '__main__':
117   start = time.time()
118   spider = XiaomiSpider()
119   spider.main()
120   end = time.time()
121   print('执行时间:%.2f' % (end-start))
```

作业2 - 腾讯招聘数据抓取

确定URL地址及目标

```
1 1、URL：百度搜索腾讯招聘 - 查看工作岗位
2 2、目标：职位名称、工作职责、岗位要求
```

要求与分析

```
1 1、通过查看网页源码,得知所需数据均为 Ajax 动态加载
2 2、通过F12抓取网络数据包,进行分析
3 3、一级页面抓取数据：职位名称
4 4、二级页面抓取数据：工作职责、岗位要求
```

一级页面json地址(index在变,timestamp未检查)

```
1 https://careers.tencent.com/tencentcareer/api/post/Query?
  timestamp=1563912271089&countryId=&cityId=&bgIds=&productId=&categoryId=&parentCategoryId=&attr
  Id=&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)

```
1 https://careers.tencent.com/tencentcareer/api/post/ByPostId?timestamp=1563912374645&postId=
  {}&language=zh-cn
```

代码实现

```
1 import requests
2 import json
3 import time
4 import random
5 from fake_useragent import UserAgent
6
7 class TencentSpider(object):
8   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'

  # 获取User-Agent
  def get_headers(self):
    ua = UserAgent()
    headers = { 'User-Agent': ua.random }
    return headers

  # 获取响应内容函数
  def get_page(self,url):
    html = requests.get(url=url,headers=self.get_headers()).content.decode('utf-8','ignore')
    # json.loads()把json格式的字符串转为python数据类型
    html = json.loads(html)
    return html

  # 主线函数：获取所有数据
  def parse_page(self,one_url):
    html = self.get_page(one_url)
    item = {}
    for job in html['Data']['Posts']:
      item['name'] = job['RecruitPostName']
      item['address'] = job['LocationName']
      # 拿postid为了拼接二级页面地址
      post_id = job['PostId']
      # 职责和要求(二级页面)
      two_url = self.two_url.format(post_id)
      item['duty'],item['requirement'] = self.parse_two_page(two_url)
      print(item)

  def parse_two_page(self,two_url):
    html = self.get_page(two_url)
    # 职责 + 要求
    duty = html['Data']['Responsibility']
    requirement = html['Data']['Requirement']

    return duty,requirement

  # 获取总页数
  def get_pages(self):
      url = self.one_url.format(1)
      html = self.get_page(url)
      pages = int(html['Data']['Count']) // 10 + 1

      return pages

  def main(self):
    # 总页数
    pages = self.get_pages()
    for index in range(1,pages):
      one_url = self.one_url.format(index)
      self.parse_page(one_url)
      time.sleep(random.uniform(0.5,1.5))

```

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

多线程有什么思路?

```
把所有一级页面链接提交到队列,进行多线程数据抓取
```

代码实现

```
import requests
import json
import time
import random
from fake_useragent import UserAgent
from threading import Thread
from queue import Queue

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.q = Queue()
    # 计数
    self.i = 0


  # 获取User-Agent
  def get_headers(self):
    ua = UserAgent()
    headers = { 'User-Agent': ua.random }
    return headers

  # 获取响应内容函数
  def get_page(self,url):
    html = requests.get(url=url,headers=self.get_headers()).content.decode('utf-8','ignore')
    # json.loads()把json格式的字符串转为python数据类型
    html = json.loads(html)
    return html

  # 主线函数: 获取所有数据
  def parse_page(self):
    while True:
        if not self.q.empty():
            one_url = self.q.get()
            html = self.get_page(one_url)
            item = {}
            for job in html['Data']['Posts']:
              item['name'] = job['RecruitPostName']
```

```
                item['address'] = job['LocationName']
                # 拿postid为了拼接二级页面地址
                post_id = job['PostId']
                # 职责和要求(二级页面)
                two_url = self.two_url.format(post_id)
                item['duty'],item['requirement'] = self.parse_two_page(two_url)
                print(item)
                self.i += 1
            # 每爬取按完成1页随机休眠
            time.sleep(random.uniform(0,1))
        else:
            break

  def parse_two_page(self,two_url):
    html = self.get_page(two_url)
    # 职责 + 要求
    duty = html['Data']['Responsibility']
    requirement = html['Data']['Requirement']

    return duty,requirement

  # 获取总页数
  def get_pages(self):
      url = self.one_url.format(1)
      html = self.get_page(url)
      pages = int(html['Data']['Count']) // 10 + 1

      return pages

  def main(self):
    # one_url入队列
    pages = self.get_pages()
    for index in range(1,pages):
      one_url = self.one_url.format(index)
      self.q.put(one_url)

    t_list = []
    for i in range(5):
      t = Thread(target=self.parse_page)
      t_list.append(t)
      t.start()

    for t in t_list:
        t.join()

    print('数量:',self.i)

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

cookie模拟登录

适用网站及场景

```
抓取需要登录才能访问的页面
```

cookie和session机制

```
# http协议为无连接协议
cookie：存放在客户端浏览器
session：存放在Web服务器
```

人人网登录案例

- **方法一 - 登录网站手动抓取Cookie**

```
1、先登录成功1次,获取到携带登陆信息的Cookie
   登录成功 - 个人主页 - F12抓包 - 刷新个人主页 - 找到主页的包(profile)
2、携带着cookie发请求
   ** Cookie
   ** User-Agent
```

```
import requests

class RenRenLogin(object):
    def __init__(self):
        # url为需要登录才能正常访问的地址
        self.url = 'http://www.renren.com/967469305/profile'
        # headers中的cookie为登录成功后抓取到的cookie
        self.headers = {
            # 此处注意cookie，要自己抓取
            "Cookie": "xxx",
            "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36",
        }

    # 获取个人主页响应
    def get_html(self):
        html = requests.get(url=self.url,headers=self.headers,verify=False).text
        print(html)
        self.parse_html(html)

    # 可获取并解析整个人人网内需要登录才能访问的地址
    def parse_html(self,html):
        pass

if __name__ == '__main__':
    spider = RenRenLogin()
    spider.get_html()
```

- **方法二 - requests模块处理Cookie**

原理思路及实现

```
# 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
1、实例化session对象
   session = requests.session()
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
浏览器原理: 访问需要登录的页面会带着之前登录过的cookie
程序原理: 同样带着之前登录的cookie去访问 - 由session对象完成
1、实例化session对象
2、登录网站: session对象发送请求,登录对应网站,把cookie保存在session对象中
3、访问页面: session对象请求需要登录才能访问的页面,session能够自动携带之前的这个cookie,进行请求
```

具体步骤

```
1、寻找Form表单提交地址 - 寻找登录时POST的地址
   查看网页源码,查看form表单,找action对应的地址: http://www.renren.com/PLogin.do

2、发送用户名和密码信息到POST的地址
   * 用户名和密码信息以什么方式发送? -- 字典
     键 : <input>标签中name的值(email,password)
     值 : 真实的用户名和密码
     post_data = {'email':'','password':''}
```

程序实现

```
整体思路
1、先POST: 把用户名和密码信息POST到某个地址中
2、再GET: 正常请求去获取页面信息
```

```
import requests

class RenrenLogin(object):
  def __init__(self):
    # 定义常用变量
    self.post_url = 'http://www.renren.com/PLogin.do'
    self.get_url = 'http://www.renren.com/967469305/profile'
    self.post_data = {
      'email' : 'xxx',
      'password' : 'xxx'
    }
    self.headers = {
      'User-Agent' : 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36',
      'Referer' : 'http://www.renren.com/SysHome.do'
    }
```

```
16
17      # 实例化session会话保持对象
18      self.session = requests.session()
19
20    # 先post 再get
21    def post_get_data(self):
22      # 先POST,把用户名和密码信息POST到一个地址
23      self.session.post(url=self.post_url,data=self.post_data,headers=self.headers)
24
25      # 再get个人主页
26      html = self.session.get(url=self.get_url,headers=self.headers).text
27      print(html)
28
29  if __name__ == '__main__':
30      spider = RenrenLogin()
31      spider.post_get_data()
```

- **方法三**

原理

```
1  1、把抓取到的cookie处理为字典
2  2、使用requests.get()中的参数:cookies
```

代码实现

```
1   import requests
2
3   class RenRenLogin(object):
4       def __init__(self):
5           # url为需要登录才能正常访问的地址
6           self.url = 'http://www.renren.com/967469305/profile'
7           self.headers = {
8               "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
    like Gecko) Chrome/74.0.3729.169 Safari/537.36",
9           }
10
11      # 将字符串cookie转为字典格式
12      def cookie_to_dict(self,cookie_str):
13          cookie_dic = {}
14          for i in cookie_str.split('; '):
15              cookie_dic[i.split('=')[0]] = i.split('=')[1]
16          return cookie_dic
17
18      # 获取个人主页响应
19      def get_html(self):
20          cookie_str = 'xxx'
21          cookie_dict = self.cookie_to_dict(cookie_str)
22          html =
    requests.get(url=self.url,headers=self.headers,verify=False,cookies=cookie_dict).text
23          print(html)
24          self.parse_html(html)
25
26      # 可获取并解析整个人人网内需要登录才能访问的地址
27      def parse_html(self,html):
```

```
        pass

if __name__ == '__main__':
    spider = RenRenLogin()
    spider.get_html()
```

json解析模块

json.loads(json)

- 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.dumps(python)

- **作用**

```
把 python 类型 转为 json 类型
```

- **示例**

```
import json

# json.dumps()之前
item = {'name':'QQ','app_id':1}
print('before dumps',type(item))
# json.dumps之后
item = json.dumps(item)
print('after dumps',type(item))
```

json.load(f)

作用

```
将json文件读取,并转为python类型
```

示例

```python
import json

with open('D:\\spider_test\\xiaomi.json','r') as f:
    data = json.load(f)

print(data)
```

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1**

```python
import json

item = {'name':'QQ','app_id':1}
with open('小米.json','a') as f:
  json.dump(item,f,ensure_ascii=False)
```

- **示例2**

```python
import json

item_list = []
for i in range(3):
  item = {'name':'QQ','id':i}
  item_list.append(item)

with open('xiaomi.json','a') as f:
    json.dump(item_list,f,ensure_ascii=False)
```

练习: 将腾讯招聘数据存入到json文件

```
# 1. __init__()
    self.f = open('tencent.json','a')
    self.item_list = []
# 2. parse_page()
    self.item_list.append(item)
# 3. main()
    json.dump(self.item_list,self.f,ensure_ascii=False)
    self.f.close()
```

json模块总结

```
# 爬虫最常用
1、数据抓取 - json.loads(html)
   将响应内容由: json 转为 python
2、数据保存 - json.dump(item_list,f,ensure_ascii=False)
   将抓取的数据保存到本地 json文件

# 抓取数据一般处理方式
1、txt文件
2、csv文件
3、json文件
4、MySQL数据库
5、MongoDB数据库
6、Redis数据库
```

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
1、chromedriver : 下载对应版本
http://chromedriver.storage.googleapis.com/index.html
2、geckodriver
https://github.com/mozilla/geckodriver/releases
3、phantomjs
https://phantomjs.org/download.html
```

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

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')
```

作业

```
1、使用selenium+浏览器抓取 民政部 数据
2、尝试去破解一下百度翻译
```