

Day05回顾

■ Ajax动态加载数据抓取流程

```
1  【1】 F12打开控制台，执行页面动作抓取网络数据包
2
3  【2】 抓取json文件URL地址
4      2.1) 控制台中 XHR ：找到异步加载的数据包
5      2.2) GET请求: Network -> XHR -> URL 和 Query String Parameters(查询参数)
6      2.3) POST请求:Network -> XHR -> URL 和 Form Data
```

■ json模块

```
1  【1】 抓取的json数据转为python数据类型
2      1.1) html = json.loads('[{},{},{}]')
3      1.2) html = requests.get(url=url,headers=headers).json()
4      1.3) html = requests.post(url=url,data=data,headers=headers).json()
5
6  【2】 抓取数据保存到json文件
7      import json
8      with open('xxx.json','w') as f:
9          json.dump([[],[],{}],f,ensure_ascii=False)
```

■ 数据抓取最终梳理

```
1  【1】 响应内容中存在
2      1.1) 确认抓取数据在响应内容中是否存在
3
4      1.2) 分析页面结构，观察URL地址规律
5          a) 大体查看响应内容结构，查看是否有更改 --（百度视频案例）
6          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）
7
8      1.3) 开始写代码进行数据抓取
9
10 【2】 响应内容中不存在
11     2.1) 确认抓取数据在响应内容中是否存在
12
13     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
14         a) GET请求: Request URL、Request Headers、Query String Paramters
15         b) POST请求:Request URL、Request Headers、FormData
16
17     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
18         a) 比如有道翻译的 salt+sign，抓取并分析JS做进一步处理
19         b) 此处注意请求头中的cookie和referer以及User-Agent
20
```

■ 多线程爬虫思路梳理

```

1  【1】所用到的模块
2      1.1) from threading import Thread
3      1.2) from threading import Lock
4      1.3) from queue import Queue
5
6  【2】整体思路
7      2.1) 创建URL队列: q = Queue()
8      2.2) 产生URL地址,放入队列: q.put(url)
9      2.3) 线程事件函数: 从队列中获取地址,开始抓取: url = q.get()
10     2.4) 创建多线程,并运行
11
12 【3】代码结构
13     def __init__(self):
14         """创建URL队列"""
15         self.q = Queue()
16
17     def url_in(self):
18         """生成待爬取的URL地址,入队列"""
19         pass
20
21     def parse_html(self):
22         """线程事件函数,获取地址,进行数据抓取"""
23         while True:
24             self.lock.acquire()
25             if not self.q.empty():
26                 url = self.q.get()
27                 self.lock.release()
28             else:
29                 break
30
31     def run(self):
32         self.url_in()
33         t_list = []
34         for i in range(3):
35             t = Thread(target=self.parse_html)
36             t_list.append(t)
37             t.start()
38
39         for th in t_list:
40             th.join()
41
42 【4】队列要点: q.get()防止阻塞方式
43     4.1) 方法1: q.get(block=False)
44     4.2) 方法2: q.get(block=True,timeout=3)
45     4.3) 方法3:
46         if not q.empty():
47             q.get()

```

Day06笔记

小米应用商店抓取(多线程)

■ 目标

- ```
1 【1】 网址：百度搜 - 小米应用商店，进入官网 http://app.mi.com/
2
3 【2】 目标：抓取聊天社交分类下的
4 2.1) 应用名称
5 2.2) 应用链接
```

### ■ 实现步骤

- ```
1 【1】 确认是否为动态加载
2     1.1) 页面局部刷新
3     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
4
5 【2】 F12抓取网络数据包
6     2.1) 抓取返回json数据的URL地址 (Headers中的Request URL)
7         http://app.mi.com/categoryAllListApi?page={}&categoryId=2&pageSize=30
8
9     2.2) 查看并分析查询参数 (headers中的Query String Parameters)
10        page: 1        只有page在变, 0 1 2 3 ... ..
11        categoryId: 2
12        pageSize: 30
13
14 【3】 将抓取数据保存到csv文件 - 注意线程锁问题
15     from threading import Lock
16     lock = Lock()
17     # 加锁 + 释放锁
18     lock.acquire()
19     lock.release()
```

■ 代码实现

```
1 import requests
2 from fake_useragent import UserAgent
3 import json
4 from threading import Thread, Lock
5 from queue import Queue
6 import time
7 import random
8
9 class XiaomiSpider:
10     def __init__(self):
11         self.url = 'http://app.mi.com/categoryAllListApi?page={}&categoryId=2&pageSize=30'
12         self.headers = {'User-Agent': UserAgent().random}
13         self.q = Queue()
14         self.lock = Lock()
15         # 计数
```

```

16         self.i = 0
17         # 存放所有字典的大列表
18         self.all_app_list = []
19
20     # URL入队列
21     def url_in(self):
22         for page in range(67):
23             url = self.url.format(page)
24             self.q.put(url)
25
26     # 线程事件函数
27     def parse_html(self):
28         while True:
29             # 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)
30             self.lock.acquire()
31             if not self.q.empty():
32                 url = self.q.get()
33                 # 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
34                 self.lock.release()
35                 # 请求 + 解析  html: {'count':2000,'data':[{}},{},{}]}
36                 try:
37                     res = requests.get(url=url, headers=self.headers)
38                     html = json.loads(res.text)
39                     for one_app in html['data']:
40                         item = {}
41                         item['app_name'] = one_app['displayName']
42                         item['app_type'] = one_app['level1CategoryName']
43                         item['app_link'] = one_app['packageName']
44                         print(item)
45
46                     # 加锁+释放锁
47                     self.lock.acquire()
48                     self.all_app_list.append(item)
49                     self.i += 1
50                     self.lock.release()
51                     # 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
52                     time.sleep(random.uniform(0,1))
53                 except Exception as e:
54                     print(e)
55             else:
56                 # 如果队列为空了,上面已经上锁,所以此处释放锁
57                 self.lock.release()
58                 break
59
60     # 入口函数
61     def run(self):
62         # 1.先让URL地址入队列
63         self.url_in()
64         # 2.多线程,开始执行
65         t_list = []
66         for i in range(2):
67             t = Thread(target=self.parse_html)
68             t_list.append(t)
69             t.start()
70
71         for j in t_list:
72             j.join()

```

```

73
74         print('数量:', self.i)
75         with open('xiaomi.json','w',encoding='utf-8') as f:
76             json.dump(self.all_app_list,f,ensure_ascii=False)
77
78     if __name__ == '__main__':
79         start_time = time.time()
80         spider = XiaomiSpider()
81         spider.run()
82         end_time = time.time()
83         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  【1】URL：百度搜索腾讯招聘 - 查看工作岗位
2  【2】目标:抓取职位的如下信息
3      a> 职位名称
4      b> 职位地址
5      c> 职位类别（技术类、销售类...）
6      d> 发布时间
7      e> 工作职责
8      f> 工作要求

```

■ 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: postid
4  【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

```

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

```

■ 二级页面地址

```

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

```

■ 多线程编写思路提示

```

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

```

■ 代码实现

```
1 import requests
2 import json
3 import time
4 from fake_useragent import UserAgent
5 from queue import Queue
6 from threading import Thread, Lock
7 from urllib import parse
8
9 class TencentSpider(object):
10     def __init__(self):
11         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'
12         self.two_url = 'https://careers.tencent.com/tencentcareer/api/post/ByPostId?
timestamp=1563912374645&postId={}&language=zh-cn'
13         self.one_q = Queue()
14         self.two_q = Queue()
15         self.lock1 = Lock()
16         self.lock2 = Lock()
17         self.i = 0
18         # 存放所有数据的大列表
19         self.item_list = []
20
21
22     def get_html(self, url):
23         headers = { 'User-Agent': UserAgent().random }
24         html = requests.get(url=url, headers=headers).text
25         return html
26
27     def url_in(self):
28         keyword = input('请输入职位类别:')
29         keyword = parse.quote(keyword)
30         total = self.get_total(keyword)
31         for page in range(1, total+1):
32             one_url = self.one_url.format(keyword, page)
33             self.one_q.put(one_url)
34
35     # 获取总页数
36     def get_total(self, keyword):
37         url = self.one_url.format(keyword, 1)
38         html = requests.get(url=url, headers={'User-Agent': UserAgent().random}).json()
39         n = int(html['Data']['Count'])
40         total = n//10 if n%10==0 else n//10+1
41
42         return total
43
44     # 线程1事件函数
45     def parse_one_page(self):
46         while True:
47             self.lock1.acquire()
48             if not self.one_q.empty():
49                 one_url = self.one_q.get()
50                 self.lock1.release()
51                 html = json.loads(self.get_html(one_url))
```

```

52         for job in html['Data']['Posts']:
53             post_id = job['PostId']
54             two_url = self.two_url.format(post_id)
55             self.lock1.acquire()
56             self.two_q.put(two_url)
57             self.lock1.release()
58         else:
59             self.lock1.release()
60         break
61
62 # 线程2事件函数
63 def parse_two_page(self):
64     while True:
65         try:
66             self.lock2.acquire()
67             two_url = self.two_q.get(block=True, timeout=3)
68             self.lock2.release()
69             html = json.loads(self.get_html(two_url))
70             # 名称+地址+类别+时间+职责+要求
71             item = {}
72             item['name'] = html['Data']['RecruitPostName']
73             item['address'] = html['Data']['LocationName']
74             item['type'] = html['Data']['CategoryName']
75             item['time'] = html['Data']['LastUpdateTime']
76             item['duty'] = html['Data']['Responsibility']
77             item['require'] = html['Data']['Requirement']
78
79             self.item_list.append(item)
80
81             print(item)
82             self.lock2.acquire()
83             self.i += 1
84             self.lock2.release()
85         except Exception as e:
86             self.lock2.release()
87             print(e, end="")
88             break
89
90 def run(self):
91     self.url_in()
92     t1_list = []
93     t2_list = []
94     for i in range(5):
95         t = Thread(target=self.parse_one_page)
96         t1_list.append(t)
97         t.start()
98
99     for i in range(5):
100         t = Thread(target=self.parse_two_page)
101         t2_list.append(t)
102         t.start()
103
104     for t in t1_list:
105         t.join()
106
107     for t in t2_list:
108         t.join()

```

```

109
110     print('数量:',self.i)
111     # 将数据写入到json文件
112     with open('tencent.json', 'w', encoding='utf-8') as f:
113         json.dump(self.item_list, f, ensure_ascii=False)
114
115
116 if __name__ == '__main__':
117     start_time = time.time()
118     spider = TencentSpider()
119     spider.run()
120     end_time = time.time()
121     print('执行时间:%.2f' % (end_time-start_time))

```

cookie模拟登录

1 | 【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 | 【1】 先登录成功1次,获取到携带登录信息的Cookie
2 |     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)
3 |
4 | 【2】 headers中携带着Cookie发请求
5 |     headers = {
6 |         'Cookie': '',
7 |         'User-Agent': ''
8 |     }

```

```

1 | """方法一代码实现"""
2 |
3 | # 1、将url改为 个人主页的URL地址
4 | # 2、将Cookie的值改为 登录成功的Cookie值
5 | import requests
6 |
7 | def login():
8 |     url = '个人主页的URL地址'
9 |     headers = {
10 |         'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like
11 |         Gecko) Chrome/80.0.3987.116 Safari/537.36',
12 |         'Cookie': '自己抓到的Cookie值',
13 |     }
14 |     html = requests.get(url=url,headers=headers).text
15 |     # 查看html中是否包含个人主页的信息 - 比如搜索 "个人主页"
16 |     print(html)
17 | login()

```


■ 方法二

```
1 【1】 原理
2     1.1) 把抓取到的cookie处理为字典
3     1.2) 使用requests.get()中的参数:cookies - 格式为字典
4
5 【2】 处理cookie为字典
6     cookies = {}
7     cookies_str = 'xxxx'
8     for kv in cookies_str.split('; '):
9         key = kv.split('=')[0]
10        value = kv.split('=')[1]
11        cookies[key] = value
```

```
1 """方法二代码实现"""
2
3 import requests
4
5 def login():
6     url = '自己账号的个人主页'
7     headers = {
8         'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like
        Gecko) Chrome/80.0.3987.116 Safari/537.36',
9     }
10    # 处理cookie为字典
11    cookies_str = '自己抓到的Cookie'
12    cookies = {}
13    for kv in cookies_str.split('; '):
14        key = kv.split('=')[0]
15        value = kv.split('=')[1]
16        cookies[key] = value
17
18    # 确认html
19    html = requests.get(url=url,headers=headers,cookies=cookies).text
20    print(html)
21
22 login()
```

■ 方法三 - requests模块处理Cookie

```
1 【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
3 【2】 原理
4     2.1) 实例化session对象 : s = requests.session()
5     2.2) 让session对象发送get或者post请求
6         res = s.post(url=url,data=data,headers=headers)
7         res = s.get(url=url,headers=headers)
8
9 【3】 思路梳理
10    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
11    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
12    3.3) 具体步骤
13        a> 实例化session对象
```

```
14         b> 登录网站：由session对象发送请求,登录对应网站
15         c> 访问页面：由session对象请求需要登录才能访问的页面
16
17     【4】如何把用户名和密码信息提交给服务器
18         4.1) 输入用户名和错误密码,登录1次进行抓包
19         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
20         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
21
22     【5】所抓数据包信息
23         5.1) POST_URL: https://accounts.douban.com/j/mobile/login/basic
24         5.2) Form Data:
25             ck:
26             name: 自己的账号
27             password: 自己的密码
28             remember: false
29             ticket:
```

```
1
2     """方法三代码实现"""
3     import requests
4
5     session = requests.session()
6
7     def login():
8         post_url = 'https://accounts.douban.com/j/mobile/login/basic'
9         post_data = {
10             'ck': '',
11             'name': '自己的用户名',
12             'password': '自己的密码',
13             'remember': 'false',
14             'ticket': '',
15         }
16         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'}
17         session.post(url=post_url,data=post_data,headers=headers)
18         url = '自己个人主页的URL地址'
19         html = session.get(url=url,headers=headers).text
20         print(html)
21
22     login()
```

■ 模拟登录小结

```

1  【1】方法1
2    先登录成功1次, 抓取到Cookie, 定义到 headers={'Cookie':''}
3
4  【2】方法2
5    先登录成功1次, 抓取到Cookie, 并处理成字典, 作为requests.get()中的cookies参数值
6    html = requests.get(url=url, headers=headers, cookies=cookies).text
7
8  【3】方法3
9    s = requests.session()
10   3.1) 登录失败1次抓包, 找到 POST 的地址和Form表单数据
11   3.2) 先POST登录: s.post(url=post_url, data=data, headers=headers)
12   3.3) 再GET抓取: html = s.get(url=get_url, headers=headers).text

```

selenium+phantomjs/Chrome/Firefox

■ selenium

```

1  【1】定义
2    1.1) 开源的Web自动化测试工具
3
4  【2】用途
5    2.1) 对Web系统进行功能性测试, 版本迭代时避免重复劳动
6    2.2) 兼容性测试(测试web程序在不同操作系统和不同浏览器中是否运行正常)
7    2.3) 对web系统进行大数量测试
8
9  【3】特点
10   3.1) 可根据指令操控浏览器
11   1.2) 只是工具, 必须与第三方浏览器结合使用
12
13  【4】安装
14   4.1) Linux: sudo pip3 install selenium
15   4.2) Windows: python -m pip install selenium

```

■ phantomjs浏览器

```

1  【1】定义
2    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
3
4  【2】下载地址
5    2.1) chromedriver : 下载对应版本
6        http://chromedriver.storage.googleapis.com/index.html
7
8    2.2) geckodriver
9        https://github.com/mozilla/geckodriver/releases
10
11   2.3) phantomjs
12        https://phantomjs.org/download.html
13
14  【3】Ubuntu安装
15   3.1) 下载后解压 : tar -zxvf geckodriver.tar.gz
16
17   3.2) 拷贝解压后文件到 /usr/bin/ (添加环境变量)

```

```

18         sudo cp geckodriver /usr/bin/
19
20     3.3) 添加可执行权限
21         sudo chmod 777 /usr/bin/geckodriver
22
23     【4】Windows安装
24         4.1) 下载对应版本的phantomjs、chromedriver、geckodriver
25         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)
26             # 查看python安装路径: where python
27         4.3) 验证
28             cmd命令行: chromedriver

```

■ 示例代码

```

1  """示例代码一: 使用 selenium+浏览器 打开百度"""
2
3  # 导入selenium的webdriver接口
4  from selenium import webdriver
5  import time
6
7  # 创建浏览器对象
8  browser = webdriver.Chrome()
9  browser.get('http://www.baidu.com/')
10 # 5秒钟后关闭浏览器
11 time.sleep(5)
12 browser.quit()

```

```

1  """示例代码二: 打开百度, 搜索赵丽颖, 点击搜索, 查看"""
2
3  from selenium import webdriver
4  import time
5
6  # 1.创建浏览器对象 - 已经打开了浏览器
7  browser = webdriver.Chrome()
8  # 2.输入: http://www.baidu.com/
9  browser.get('http://www.baidu.com/')
10 # 3.找到搜索框,向这个节点发送文字: 赵丽颖
11 browser.find_element_by_xpath('//*[@id="kw"]').send_keys('赵丽颖')
12 # 4.找到 百度一下 按钮,点击一下
13 browser.find_element_by_xpath('//*[@id="su"]').click()

```

■ 浏览器对象(browser)方法

```

1  【1】 browser.get(url=url)    - 地址栏输入url地址并确认
2  【2】 browser.quit()         - 关闭浏览器
3  【3】 browser.close()        - 关闭当前页
4  【4】 browser.page_source    - HTML结构源码
5  【5】 browser.page_source.find('字符串')
6      从html源码中搜索指定字符串,没有找到返回: -1,经常用于判断是否为最后一页
7  【6】 browser.maximize_window() - 浏览器窗口最大化

```

■ 定位节点八种方法

```

1  【1】单元素查找('结果为1个节点对象')
2      1.1) browser.find_element_by_id('id属性值')
3      1.2) browser.find_element_by_name('name属性值')
4      1.3) browser.find_element_by_class_name('class属性值')
5      1.4) browser.find_element_by_xpath('xpath表达式')
6      1.5) browser.find_element_by_link_text('链接文本')
7      1.6) browser.find_element_by_partical_link_text('部分链接文本')
8      1.7) browser.find_element_by_tag_name('标记名称')
9      1.8) browser.find_element_by_css_selector('css表达式')
10
11  【2】多元素查找('结果为[节点对象列表]')
12      2.1) browser.find_elements_by_id('id属性值')
13      2.2) browser.find_elements_by_name('name属性值')
14      2.3) browser.find_elements_by_class_name('class属性值')
15      2.4) browser.find_elements_by_xpath('xpath表达式')
16      2.5) browser.find_elements_by_link_text('链接文本')
17      2.6) browser.find_elements_by_partical_link_text('部分链接文本')
18      2.7) browser.find_elements_by_tag_name('标记名称')
19      2.8) browser.find_elements_by_css_selector('css表达式')

```

■ 猫眼电影示例

```

1  from selenium import webdriver
2  import time
3
4  url = 'https://maoyan.com/board/4'
5  browser = webdriver.Chrome()
6  browser.get(url)
7
8  def get_data():
9      # 基准xpath: [<selenium xxx li at xxx>,<selenium xxx li at>]
10     li_list = browser.find_elements_by_xpath('//*[@id="app"]/div/div/div[1]/dl/dd')
11     for li in li_list:
12         item = {}
13         # info_list: ['1', '霸王别姬', '主演: 张国荣', '上映时间: 1993-01-01', '9.5']
14         info_list = li.text.split('\n')
15         item['number'] = info_list[0]
16         item['name'] = info_list[1]
17         item['star'] = info_list[2]
18         item['time'] = info_list[3]
19         item['score'] = info_list[4]
20
21         print(item)
22
23  while True:
24     get_data()
25     try:
26         browser.find_element_by_link_text('下一页').click()
27         time.sleep(2)
28     except Exception as e:
29         print('恭喜你!抓取结束')
30         browser.quit()
31         break

```

■ 节点对象操作

```
1 【1】 文本框操作
2     1.1) node.send_keys('') - 向文本框发送内容
3     1.2) node.clear() - 清空文本
4     1.3) node.get_attribute('value') - 获取文本内容
5
6 【2】 按钮操作
7     1.1) node.click() - 点击
8     1.2) node.is_enabled() - 判断按钮是否可用
9     1.3) node.get_attribute('value') - 获取按钮文本
```

作业1 - 京东爬虫

■ 目标

```
1 【1】 目标网址 : https://www.jd.com/
2 【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 【1】 打开京东，到商品搜索页
2 【2】 匹配所有商品节点对象列表
3 【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 【4】 提取完1页后，判断如果不是最后1页，则点击下一页
5     # 如何判断是否为最后1页？？？
```

■ 实现步骤-参考与提示

```
1 # 1. 找节点
2 1、首页搜索框 : //*[@id="key"]
3 2、首页搜索按钮 ://*[@id="search"]/div/div[2]/button
4 3、商品页的 商品信息节点对象列表 ://*[@id="J_goodsList"]/ul/li
5 4、for循环遍历后
6 名称: .//div[@class="p-name"]/a/em
7 价格: .//div[@class="p-price"]
8 评论: .//div[@class="p-commit"]/strong
9 商家: .//div[@class="p-shopnum"]
10
11 # 2. 执行JS脚本，获取动态加载数据
12 browser.execute_script(
13     'window.scrollTo(0,document.body.scrollHeight)'
14 )
```

■ 代码实现

```
1 browser.excute_script(
2     'window.scrollTo(0,document.body.scrollHeight)'
3 )
4 time.sleep(2)
5
6 【1】 搜索内容: 爬虫书
```

```
7     li_list = [<li1>,<li2>,...<lin>]
8     for li in li_list:
9         方法1: print(li.text)
10        方法2: item['name']=li.find_element_by_xpath('')
11
12    【2】一定要注意给页面元素加载预留时间
13
14    【3】执行JS脚本
```

作业2

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