# **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 ：找到异步加载的数据包  
   2.2) GET请求: Network -> XHR -> URL 和 Query String Parameters(查询参数)  
   2.3) POST请求:Network -> XHR -> URL 和 Form Data
* **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,data=data,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 URL、Request Headers、Query String Paramters  
   b) POST请求:Request URL、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
* **execjs模块使用**
* import execjs  
    
  with open('xxx.js', 'r') as f:  
   js\_code = f.read()  
     
  loader = execjs.compile(js\_code)  
  loader.call('js中函数名', 'js中参数1', 'js中参数2', ...)
* **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()  
   self.lock = Lock()  
     
   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:  
   self.lock.acquire()  
   if not self.q.empty():  
   url = self.q.get()  
   self.lock.release()  
   else:  
   self.lock.release()  
   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()  
     
  【4】队列要点: q.get()防止阻塞方式  
   4.1) 方法1: q.get(block=False)  
   4.2) 方法2: q.get(block=True,timeout=3)  
   4.3) 方法3:  
   if not q.empty():  
   q.get()

# **Day06笔记**

## **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()
* **代码实现**
* import requests  
  from fake\_useragent import UserAgent  
  import json  
  from threading import Thread,Lock  
  from queue import Queue  
  import time  
  import random  
    
  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  
   # 存放所有字典的大列表  
   self.all\_app\_list = []  
    
   # 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()  
   # 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()  
   # 请求 + 解析 html: {'count':2000,'data':[{},{},{}]}  
   try:  
   res = requests.get(url=url, headers=self.headers)  
   html = json.loads(res.text)  
   for one\_app in html['data']:  
   item = {}  
   item['app\_name'] = one\_app['displayName']  
   item['app\_type'] = one\_app['level1CategoryName']  
   item['app\_link'] = one\_app['packageName']  
   print(item)  
    
   # 加锁+释放锁  
   self.lock.acquire()  
   self.all\_app\_list.append(item)  
   self.i += 1  
   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(0,1))  
   except Exception as e:  
   print(e)  
   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)  
   with open('xiaomi.json','w',encoding='utf-8') as f:  
   json.dump(self.all\_app\_list,f,ensure\_ascii=False)  
    
  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**
* import requests  
  import json  
  import time  
  from fake\_useragent import UserAgent  
  from queue import Queue  
  from threading import Thread,Lock  
  from urllib import parse  
    
  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  
   # 存放所有数据的大列表  
   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)  
   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']  
    
   self.item\_list.append(item)  
    
   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)  
   # 将数据写入到json文件  
   with open('tencent.json', 'w', encoding='utf-8') as f:  
   json.dump(self.item\_list, f, ensure\_ascii=False)  
    
    
  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,自动提交Cookie  
    
  【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 = s.post(url=url,data=data,headers=headers)  
   res = s.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  
     
  【5】所抓数据包信息  
   5.1) POST\_URL: https://accounts.douban.com/j/mobile/login/basic  
   5.2) Form Data:  
   ck:   
   name: 自己的账号  
   password: 自己的密码  
   remember: false  
   ticket:
* """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()
* **模拟登录小结**
* 【1】方法1  
   先登录成功1次,抓取到Cookie,定义到 headers={'Cookie':''}  
     
  【2】方法2  
   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  
   html = requests.get(url=url,headers=headers,cookies=cookies).text  
     
  【3】方法3  
   s = requests.session()  
   3.1) 登录失败1次抓包,找到 POST 的地址和Form表单数据  
   3.2) 先POST登录: s.post(url=post\_url,data=data,headers=headers)  
   3.3) 再GET抓取: html = s.get(url=get\_url,headers=headers).text

## **selenium+phantomjs/Chrome/Firefox**

* **selenium**
* 【1】定义  
   1.1) 开源的Web自动化测试工具  
     
  【2】用途  
   2.1) 对Web系统进行功能性测试,版本迭代时避免重复劳动  
   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)  
   2.3) 对web系统进行大数量测试  
     
  【3】特点  
   3.1) 可根据指令操控浏览器  
   3.2) 只是工具，必须与第三方浏览器结合使用  
     
  【4】安装  
   4.1) Linux: sudo pip3 install selenium  
   4.2) 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  
   http://npm.taobao.org/mirrors/chromedriver/  
     
   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.get(url=url) - 地址栏输入url地址并确认   
  【2】browser.quit() - 关闭浏览器  
  【3】browser.close() - 关闭当前页  
  【4】browser.page\_source - HTML结构源码  
  【5】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,经常用于判断是否为最后一页  
  【6】browser.maximize\_window() - 浏览器窗口最大化
* **定位节点八种方法**
* 【1】单元素查找('结果为1个节点对象')  
   1.1) browser.find\_element\_by\_id('id属性值')  
   1.2) browser.find\_element\_by\_name('name属性值')  
   1.3) browser.find\_element\_by\_class\_name('class属性值')  
   1.4) browser.find\_element\_by\_xpath('xpath表达式')  
   1.5) browser.find\_element\_by\_link\_text('链接文本')  
   1.6) browser.find\_element\_by\_partical\_link\_text('部分链接文本')  
   1.7) browser.find\_element\_by\_tag\_name('标记名称')  
   1.8) browser.find\_element\_by\_css\_selector('css表达式')  
    
  【2】多元素查找('结果为[节点对象列表]')  
   2.1) browser.find\_elements\_by\_id('id属性值')  
   2.2) browser.find\_elements\_by\_name('name属性值')  
   2.3) browser.find\_elements\_by\_class\_name('class属性值')  
   2.4) browser.find\_elements\_by\_xpath('xpath表达式')  
   2.5) browser.find\_elements\_by\_link\_text('链接文本')  
   2.6) browser.find\_elements\_by\_partical\_link\_text('部分链接文本')  
   2.7) browser.find\_elements\_by\_tag\_name('标记名称')  
   2.8) browser.find\_elements\_by\_css\_selector('css表达式')
* **猫眼电影示例**
* 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】文本框操作  
   1.1) node.send\_keys('') - 向文本框发送内容  
   1.2) node.clear() - 清空文本  
   1.3) node.get\_attribute('value') - 获取文本内容  
     
  【2】按钮操作  
   1.1) node.click() - 点击  
   1.2) node.is\_enabled() - 判断按钮是否可用  
   1.3) node.get\_attribute('value') - 获取按钮文本

### **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)

### **selenium - 键盘操作**

from selenium.webdriver.common.keys import Keys  
  
browser = webdriver.Chrome()  
browser.get('http://www.baidu.com/')  
# 1、在搜索框中输入"selenium"  
browser.find\_element\_by\_id('kw').send\_keys('赵丽颖')  
# 2、输入空格  
browser.find\_element\_by\_id('kw').send\_keys(Keys.SPACE)  
# 3、Ctrl+a 模拟全选  
browser.find\_element\_by\_id('kw').send\_keys(Keys.CONTROL, 'a')  
# 4、Ctrl+c 模拟复制  
browser.find\_element\_by\_id('kw').send\_keys(Keys.CONTROL, 'c')  
# 5、Ctrl+v 模拟粘贴  
browser.find\_element\_by\_id('kw').send\_keys(Keys.CONTROL, 'v')  
# 6、输入回车,代替 搜索 按钮  
browser.find\_element\_by\_id('kw').send\_keys(Keys.ENTER)

### **==selenium - 鼠标操作==**

from selenium import webdriver  
# 导入鼠标事件类  
from selenium.webdriver import ActionChains  
  
driver = webdriver.Chrome()  
driver.get('http://www.baidu.com/')  
  
# 移动到 设置，perform()是真正执行操作，必须有  
element = driver.find\_element\_by\_xpath('//\*[@id="u1"]/a[8]')  
ActionChains(driver).move\_to\_element(element).perform()  
  
# 单击，弹出的Ajax元素，根据链接节点的文本内容查找  
driver.find\_element\_by\_link\_text('高级搜索').click()

## **作业 - 京东商品爬虫**

* **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