# 景和千城—爬虫

一．工作准备

爬虫是为公司以后的用户行为分析，商铺针对性建议等提供一定的决策支持的重要工作，所以爬虫的好坏关系非常大。

工欲善其事，必先利其器，相关技术也得准备齐：

Python下载，版本：python3.5（2.7也可以）。

Selenium下载， 版本：Selenium 2 （Selenium webdriver） 可以通过pip install selenium 命令来安装

浏览器下载 ， Chrome 或者 Firefox 都可以 以及相应浏览器的driver（放在相关浏览器安装目录下），网上下载即可

其它相关包等，通过pip insatall方法来安装

二．技术框架

主要爬取对象是大众，美团等具有大量商家信息的网站，同时这些网站采用了很多有效的反爬策略来防止爬虫，所以爬取这些网站有一定的难度和风险。

进过多方的考虑和对比，最终采用的框架是 selenium + python自动化框架（如果有更好的后期会换）。

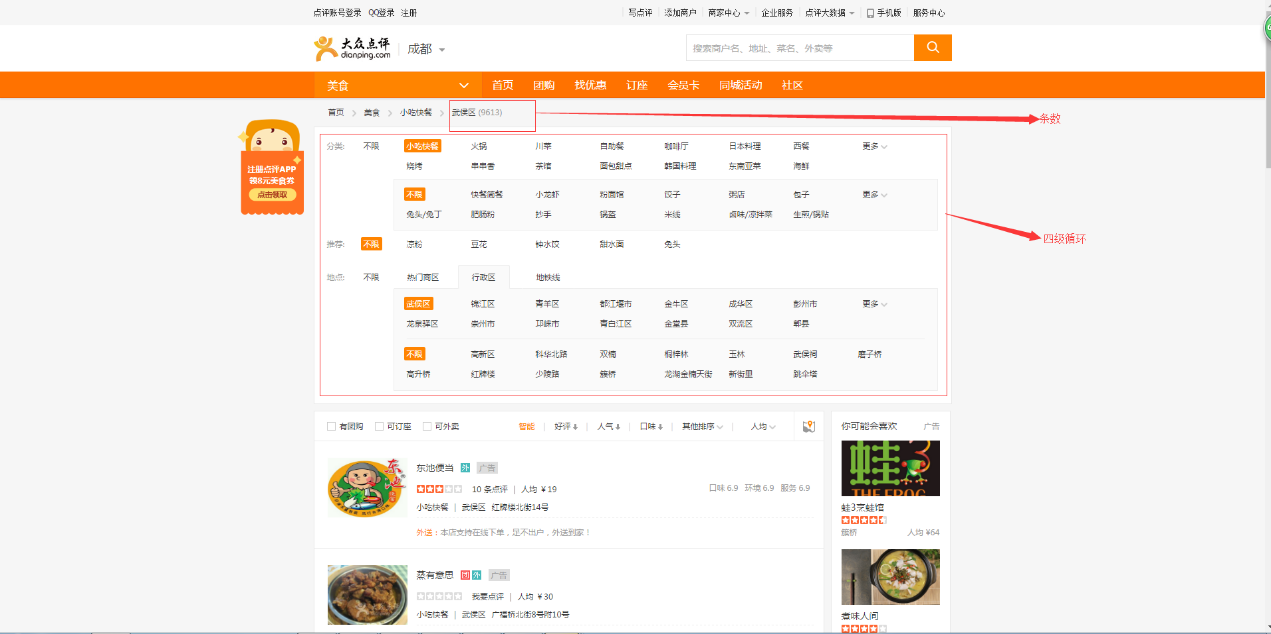
采用该框架的最主要的一点是因为selenium 是可以很方便的实现网页的模拟浏览器点击操作，从而可以有效的规避很多针对主流的 urllib request 而采取的针对策略，并且更符合我们爬取项目的实际情况。

该框架部署简单实用，易开发。

框架流程如图：

三．爬取过程

**1、四级循环**：

 如下2图（截图上部分和下部分），这是我们爬取网站的入口，中间部分商铺的信息是我们主要抓取的对象。



如图上2图所示，整个美食部分(点击不限)它显示大概有14W多（实际上比这个少），但是它每页只显示15条数据，一共只显示50页，所以它首页提供的数据只有750条数据（应该是推荐数据）！因此我们如想尽可能的多的抓取数据，就只有去上面的条件中一个一个点来细化数据，这样才能拿到最多的数据。如果已经细化到最小的条件还是超过了750条数据，剩下的也没有办法拿到了（可能是大众的假数据）。

 如下图：最细化的结果少于750条，这样就可以拿到最多的数据。

点击的条件主要是分类和地区，分类点击后可能会出现分类小类，地区点击后也又可能出现地区小类。所以最多时候会有4层循环，点击任何一个按钮的过程中页面布局，甚至数据都会变化，所以会抛出很多异常，可能会需要根据每次的情况做单独的异常处理，过程非常麻烦。

Selenium做模拟点击非常方便，这里最好先选用分类做成列表做第一层循环（它是在其它按钮点击中变化最少的），因为它是第一层所有列表数据是固定的19个，出问题也是最少的，展示获取分类并循环代码如下。

soup = GET\_SOUP(driver)

type\_list = soup.find(**'div'**, attrs={**'id'**: **"classfy"**}).find\_all(**'a'**) *# 拿取分类*startstep=MARKING(**'big\_type'**, **''**, 2)*#读取上次程序中断后停留的地方***for** k **in** range(startstep, len(type\_list)):  
 *#每次循环一次重置一下分类和地区，因为可能与最开始看到的顺序不一致* time.sleep(5)  
 driver.find\_element\_by\_xpath(**'//div[@class="nav-category"]/a[1]/span'**).click() *# 点击分类不限* time.sleep(5)  
 driver.find\_element\_by\_xpath(**'//div[@class="nav-category nav-tabs"]/a[1]/span'**).click() *# 点击地区不限* time.sleep(5)  
 **if** k==15:  
 driver.find\_element\_by\_xpath(**'//\*[@id="classfy"]/a[last()]'**).click() *# 点击更多* soup=GET\_SOUP(driver)  
 big\_type = soup.find(**'div'**,attrs={**'id'**:**'classfy'**}).find\_all(**'a'**)[k-1].text *# 大类名称* button = **'//\*[@id="classfy"]/a['** + str(k) + **']/span'  
 try**:  
 driver.find\_element\_by\_xpath(button).click() *# 点击分类*  
 **except**:  
 driver.refresh()  
 time.sleep(2)  
 **if** k == 15:  
 driver.find\_element\_by\_xpath(**'//\*[@id="classfy"]/a[last()]'**).click() *# 点击更多* driver.find\_element\_by\_xpath(button).click() *# 点击分类* print(**'######################################################################分类大类：%s'** % big\_type)

如上述代码，用python中的bs4可以轻松拿到分类中的所有值，并形成一个列表，在循环点击的时候，再通过xpath精确定位到每个需要点击的按钮（值）并点击。

Selenium在点击的时候有时候会莫名其妙的情况，这个时候可能是网络不好，页面加载没有完全完成，导致它找不到该元素的情况，所以可以加个异常，当找不到的时候刷新，等待几秒钟再点击。

当点击完分类的的时候，会出现分类小类（可能会没有，没有就跳过），嵌套在一级循环中形成二级循环，跟上述代码一样，获取、循环、点击，地区与地区小类也是如此，嵌套形成三级和四级循环，如此四级的点击循环就完成了。

在点击该四级循环的时候，每级列表都有可能因为本级或者其它级的点击而发生变化而导致异常，如下图：

这是正常的点击，当点击都江堰的时候，本级列表发生变化。

此时，如果都江堰的地区小类循环完毕，去找地区的下一个值点击的时候就会报错，因为列表已经发生了变动，这个时候我一般去点击不限才解决该问题

**2、获取商铺基础信息：**

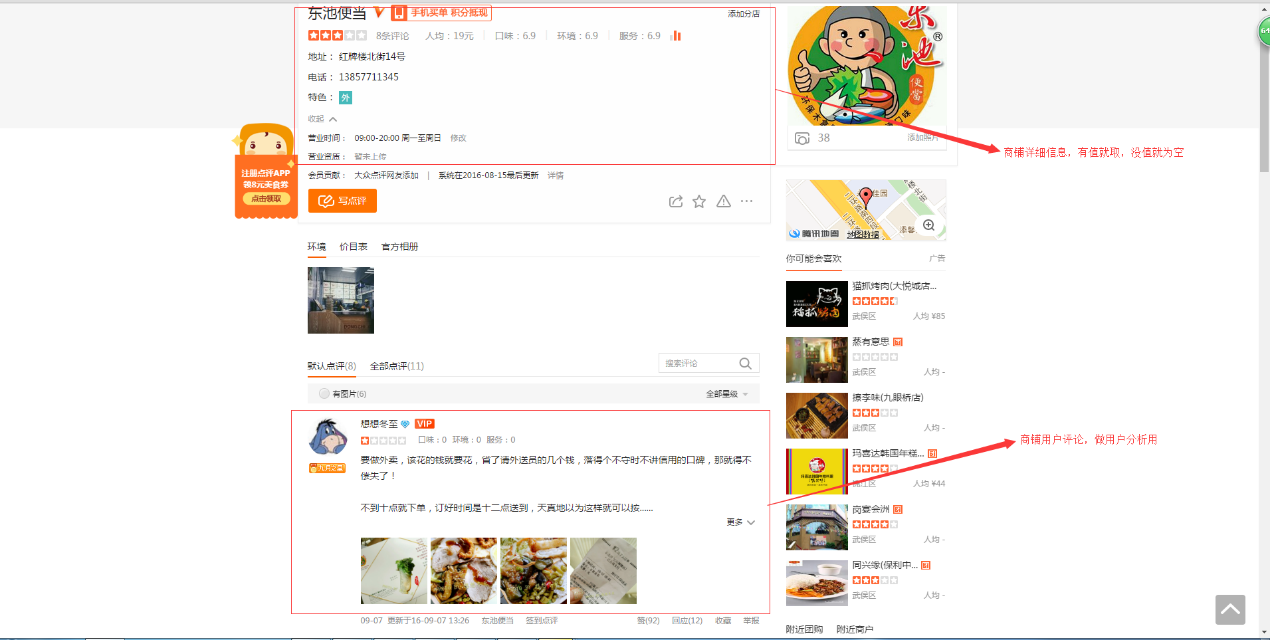
如此循环可以正确的往下执行，每最底层循环一次我们该去列表中间拿我们的数据一次，拿去中间部分代码如下：

body\_data = soup.find(**'div'**, attrs={**'class'**: **'shop-list J\_shop-list shop-all-list'**})  
res\_list = body\_data.find\_all(**'li'**)  
**for** res **in** res\_list:  
 shop\_web\_address = res.find(**'a'**, attrs={**'target'**: **'\_blank'**}).get(**'href'**) *# 商店网址* shop\_id = shop\_web\_address.split(**'/'**)[-1] *# 商铺ID* img\_address = res.find(**'img'**).get(**'src'**) *# 图片地址* shop\_name = res.find(**'h4'**).next *# 商铺名称* star\_level = res.find(**'span'**, attrs={**'class'**: **'sml-rank-stars'**}).get(**'class'**)[1] *# 商铺星级* **try**:  
 mean\_price = res.find(**'a'**, attrs={**'class'**: **'mean-price'**}).find(**'b'**).next *# 人均* **except**:  
 mean\_price = **'-'** *# 人均* shop\_type = res.find(**'div'**, attrs={**'class'**: **'tag-addr'**}).find\_all(**'span'**)[0].next *# 商铺类型* shop\_address = **'%s %s'** % (res.find(**'div'**, attrs={**'class'**: **'tag-addr'**}).find\_all(**'span'**)[1].next,  
 res.find(**'div'**, attrs={**'class'**: **'tag-addr'**}).find\_all(**'span'**)[2].next) *# 商铺地址* **try**:  
 shop\_socre = **'质量：%s 环境：%s 服务：%s '** % (  
 res.find(**'span'**, attrs={**'class'**: **'comment-list'**}).find\_all(**'span'**)[0].find(**'b'**).next, res.find(**'span'**, attrs={ **'class'**: **'comment-list'**}).find\_all(**'span'**)[1].find(**'b'**).next,res.find(**'span'**, attrs={**'class'**: **'comment-list'**}).find\_all(**'span'**)[2].find(**'b'**).next) *# 店铺评分* **except**:  
 shop\_socre = **' '** shop\_info = **'%s[}%s[}%s[}%s[}%s[}%s[}人均：%s[}%s[}%s[}%s[}%d[}%s[}%s[}%s[}%s'** % (  
 shop\_id, shop\_name, shop\_web\_address, img\_address, star\_level, **' '**, mean\_price, shop\_type, shop\_socre,  
 shop\_address, channel\_type, **' '**,small\_type,big\_region,small\_region)  
 get\_detail\_address = **'%s[}%s'** % (shop\_web\_address, shop\_type)  
 GET\_FILE(name, filename, shop\_info, type,5000)  
 GET\_FILE(name, detail\_address, get\_detail\_address, type,5000)

这段代码是用BS4拿去中间部分的商铺基础信息代码，并格式化存入文本中。

**3、获取详细商铺信息：**

去拿去基础信息中，会得到每个商铺的网址信息，通过该信息，我们一个个循环进入去拿去该对应商铺的各种详细信息和用户评论（如下图），并存入文本中，如果该商铺已经成功抓取了数据，将把该网址存入成功抓取文本中，当程序意外停止或者其它原因中断时，已成功访问的商铺将不再重复抓取，当成功数达到要访问的所有商铺数时循环结束。

代码如下：

**for** i **in** range(0,34):  
 file=**'C:/Users/Administrator/Desktop/DZDP\_BASIC/%s\_%d.txt'** %(filename,i)  
 **with** open(file, **'r'**,encoding=**'utf-8'**) **as** f:  
 **for** line **in** f:  
 web\_address.append(line.split(**'[}'**)[0])  
 web\_address=list(set(web\_address))  
 print(len(web\_address))  
sum=0  
**while** sum<len(web\_address):  
 **for** i **in** web\_address:  
 url=**'http://www.dianping.com'** + str(i)  
 web\_success = []  
 **with** open(**'C:/Users/Administrator/Desktop/DZDP\_BASIC/dzdp\_success\_list.txt'**, **'r'**) **as** f:  
 **for** line **in** f:  
 web\_success.append(line.replace(**'\n'**,**''**))  
 **if** url **not in** web\_success:  
 **try**:  
 dz\_spiders.SHOP\_DETAILS\_INFO\_1(**'dzdp'**, **'food\_detail'**,url)#获取商铺详细信息  
 print(**'成功'**)  
 **with** open(**'C:/Users/Administrator/Desktop/DZDP\_BASIC/dzdp\_success\_list.txt'**, **'a'**) **as** f:  
 f.write(url)  
 f.write(**'\n'**)  
 dz\_spiders.GET\_LOG(**'dzdp\_deatail'**, **'%s:%s'** % (url, **'成功'**))  
 **except** Exception **as** e:  
 print(url)  
 print(e)  
 dz\_spiders.GET\_LOG(**'dzdp\_deatail'**,**'[ERROR:::::::::]%s:%s'** %(url,str(e)))  
 sum=len(open(**'C:/Users/Administrator/Desktop/DZDP\_BASIC/dzdp\_success\_list.txt'**).readlines())  
 print(**'成功数：%s'** %sum)

四．爬虫待解决问题。

1.效率瓶颈（最大的问题）：

拿取基础信息效率还好，一次拿15条数据，主要耗时在点击、停顿和页面加载（如果点击异常会额外耗时）上，在现有公司网络环境不稳定的情况下，还可以勉强接受，基本上是一天爬1W的数据下来。但是爬取详单信息的情况就不容乐观了，因为大众的反爬机制比较严格（刚爬了几天就差点把我的IP给封了），如果频率太高，就很容易封IP，没办法就只能加时间间隔，而且我是单IP就只能一条一条的爬，效率之慢，一天大概只能爬一千多条。

解决方法：只能用多IP多线程的方法来解决效率问题。