Python数据获取实例

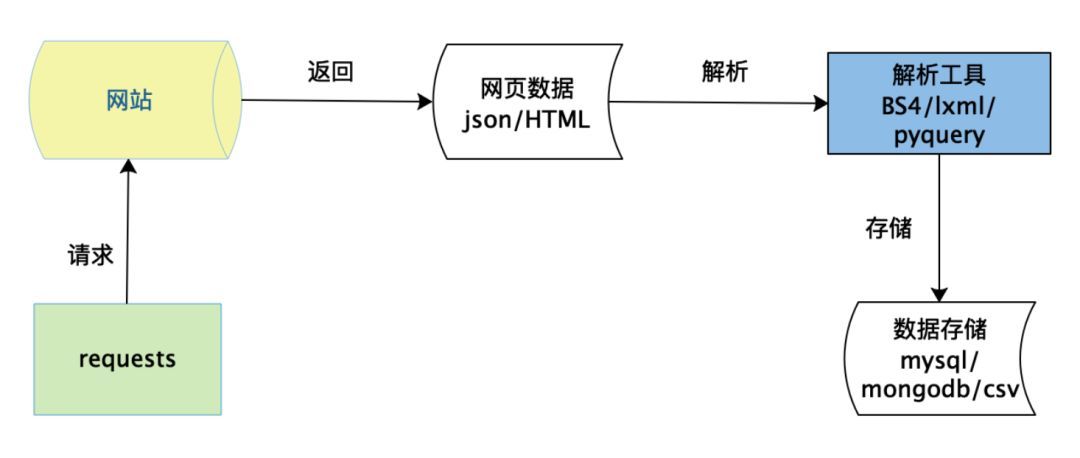
链接: <https://cloud.tencent.com/developer/article/1555150>

# **引言**

案例从北京公交路线数据的获取和预处理入手，记录使用python中requests库获取数据。文章在保证按照一定处理逻辑的前提下，以自问自答的方式，对其中每一个环节进行详细阐述。本次代码均在jupyter notebook中测试通过，希望对大家有所启示。

本次我们从公交网获取北京公交的数据。

(http://beijing.gongjiao.com/lines\_all.html)



如上图所示，数据获取分为**请求**，**解析**，**存储**三个最主要的步骤。

# **1.如何用python模拟网络请求？**

使用request库可以模拟不同的请求，例如requests.get()模拟get请求，requests.post()模拟post请求。必要的时候可以添加请求头header，header通常包括user-agent，cookie，refer等信息，还可以增加请求参数data和代理信息。主要代码形式为：response = requests.request("GET", url, headers=headers, params=querystring)response是网站返回的响应信息，可以调用其text方法获取网站的HTML源码。本次我们的目标网站比较简单，获取网页源码的代码如下：

url = 'http://beijing.gongjiao.com/lines\_all.html'

text = requests.get(url).text

# **2.如何对网页进行解析？**

python中提供了多种库用于网页解析，例如lxml，BeautifulSoup，pyquery等。每一个工具都有相应的解析规则，但都是把HTML文档当做一个DOM树，通过选择器进行节点和属性的定位。本次我们使用lxml对网页进行解析，主要用到了xpath的语法。lxml的执行效率通常也比BeautifulSoup更高一些。

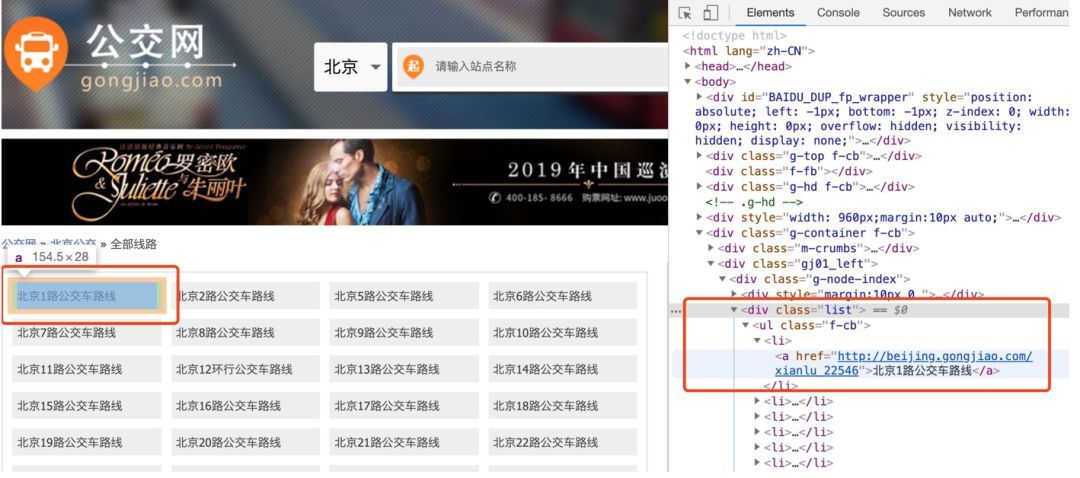
doc = etree.HTML(text)

all\_lines = doc.xpath("//div[@class='list']/ul/li")

for line in all\_lines:

line\_name = line.xpath("./a/text()")[0].strip()

line\_url = line.xpath("./a/@href")[0]



我们将图和代码结合起来看。第一行代码将上一步返回的HTML文本转换为xpath可以解析的对象。第二行代码定位到class=list的div下面所有的li标签，即右图中的红色框的部分，得到的是一个列表。从第三行开始对其进行遍历，处理每一个li下面的a标签。第4行取出a标签下的文本，用到了xpath的text()方法，对应到第一个li就是“北京1路公交车路线”，第5行取出a标签下对应的链接，用到了xpath的@href取出a标签下的href属性值。直接取都是列表的形式，所以需要用索引取出具体的值。

这样我们就可以得到整个公交线路列表中的线路名称和线路url。然后从线路url出发，就可以获取每条线路的具体信息。如下面代码和图片所示，虽然数据略多，但主要的逻辑和上面类似，可以查看代码中的注释。



url = 'http://beijing.gongjiao.com/xianlu\_38753'#先以一个url为例，进行页面的分析

text = requests.get(url).text

print(len(text))

doc = etree.HTML(text)

infos = doc.xpath("//div[@class='gj01\_line\_header clearfix']")#定位到相应的div块

for info in infos:

start\_stop = info.xpath("./dl/dt/a/text()")#获取起点站和终点站的文本，xpath的逻辑为：div->dl->dt->a

op\_times = info.xpath("./dl/dd[1]/b/text()")#获取运营时间的文本，xpath的逻辑为：div->dl->第一个dd->b

interval = info.xpath("./dl/dd[2]/text()")#获取发车间隔的文本，xpath的逻辑为：div->dl->第二个dd

price = info.xpath("./dl/dd[3]/text()")#获取票价信息的文本，xpath的逻辑为：div->dl->第三个dd

company = info.xpath("./dl/dd[4]/text()")#获取汽车公司的文本，xpath的逻辑为：div->dl->第四个dd

up\_times = info.xpath("./dl/dd[5]/text()")#获取更新时间的文本，xpath的逻辑为：div->dl->第五个dd

all\_stations\_up = doc.xpath('//ul[@class="gj01\_line\_img JS-up clearfix"]')#定位到相应的div块

for station in all\_stations\_up:

station\_name = station.xpath('./li/a/text()')#遍历取出该条线路上的站点名称

all\_stations\_down = doc.xpath('//ul[@class="gj01\_line\_img JS-down clearfix"]')#定位到返程线路相应的div块

for station in all\_stations\_down:

station\_name = station.xpath('./li/a/text()')#遍历取出该条线路上返程的站点名称

如果将获取的文本都输出(请自行添加相应的print语句)运行结果如下：

['老山公交场站(1)', '四惠枢纽站(27)']

['5:00-23:00']

['5:00-23:00']

['发车间隔：未知']

['票价信息：10公里以内票价2元，每增加5公里以内加价1元，最高票价6元']

['汽车公司：北京公交集团第六客运分公司']

['更新时间：2015-04-05 03:32:16']

['老山公交场站(1)', '老山南路东口(2)', '地铁八宝山站(3)', '玉泉路口西(4)', '五棵松桥西(6)', '翠微路口(8)', '公主坟(9)', '军事博物馆(10)', '木樨地西(11)', '工会大楼(12)', '南礼士路(13)', '复兴门内(13)', '西单路口东(15)', '天安门西(16)', '天安门东(17)', '东单路口西(18)', '北京站口东(19)', '日坛路(20)', '永安里路口西(21)', '大北窑西(22)', '大北窑东(23)', '郎家园(23)', '四惠枢纽站(27)']

['四惠枢纽站(27)', '八王坟西(24)', '郎家园(23)', '大北窑东(23)', '大北窑西(22)', '永安里路口西(21)', '日坛路(20)', '北京站口东(19)', '东单路口西(18)', '天安门东(17)', '天安门西(16)', '西单路口东(15)', '复兴门内(13)', '南礼士路(13)', '工会大楼(12)', '木樨地西(11)', '军事博物馆(10)', '公主坟(9)', '翠微路口(8)', '五棵松桥东(6)', '玉泉路口西(4)', '地铁八宝山站(3)', '老山南路东口(2)', '老山公交场站(1)']

# **3.如何存储获取的数据？**

[数据存储](https://cloud.tencent.com/product/cdcs?from=10680)的载体通常有文件(例如csv，excel)和[数据库](https://cloud.tencent.com/solution/database?from=10680" \t "_blank)(例如mysql，[MongoDB](https://cloud.tencent.com/product/mongodb?from=10680" \t "_blank))。我们这里选择了csv文件的形式，一方面是数据量不是太大，另一方面也不需要进行数据库安装，只需将数据整理成dataframe的格式，直接调用pandas的to\_csv方法就可以将dataframe写入csv文件中。主要代码如下：

#准备一个存储数据的字典

df\_dict = {

'line\_name': [], 'line\_url': [], 'line\_start': [], 'line\_stop': [],

'line\_op\_time': [], 'line\_interval': [], 'line\_price': [], 'line\_company': [],

'line\_up\_times': [], 'line\_station\_up': [], 'line\_station\_up\_len': [],

'line\_station\_down': [], 'line\_station\_down\_len': []

}

#将上面获取的数据写入到字典中，注意这里只是示例，实际运行时候要将下面的代码放到循环中，每解析一条线路就需要append一次。

df\_dict['line\_name'].append(line\_name)

df\_dict['line\_url'].append(line\_url)

df\_dict['line\_start'].append(start\_stop[0])

df\_dict['line\_stop'].append(start\_stop[1])

df\_dict['line\_op\_time'].append(op\_times[0])

df\_dict['line\_interval'].append(interval[0][5:])#为了把前面的文字“发车间隔”截掉，其余的类似

df\_dict['line\_company'].append(company[0][5:])

df\_dict['line\_price'].append(price[0][5:])

df\_dict['line\_up\_times'].append(up\_times[0][5:])

df\_dict['line\_station\_up'].append(station\_up\_name)

df\_dict['line\_station\_up\_len'].append(len(station\_up\_name))

df\_dict['line\_station\_down'].append(station\_down\_name)

df\_dict['line\_station\_down\_len'].append(len(station\_down\_name))

#将数据保存成csv文件

df = pd.DataFrame(df\_dict)

df.to\_csv('bjgj\_lines\_utf8.csv', encoding='utf-8', index=None)

# **4.看一看完整代码？**

以上我们分模拟**请求**，网页**解析**，数据**存储**3个步骤，学习了数据获取的流程。实际运行过程中，还需要增加一些保证代码“**健壮性**”的逻辑。例如，控制爬取的频率，处理请求失败的情况，处理不同的线路网页结构可能有差异的情况等等。本次的数据源没有做很多反扒限制，因此前两种情况我们可以不处理。至于第三种，有的路线会出现线路运营时间是空值的情况，需要进行判断。另外还可以增加一些爬虫运行过程的提示信息，让我们知道爬取进度，当然你也可以增加多线程，代理，ua切换等代码，此处我们还用不上这些。完整的代码可以在后台回复“北京公交”进行获取。