第 5 章

CHAPTER 5

解 析 网 页

在前面的章节已经能够使用 requests 库从网页把整个源代码爬取下来了，接着需要从每个网页中提取一些数据。会用到类库，常用的类库有 3 种，分别为 lxm、BeautifulSoup 及 re(正则)。

5.1 获取豆瓣电影

下面以获取豆瓣电影正在热映的电影名为例，网址为：url＝'https://movie.douban.com/cinema/nowplaying/beijing/'，利用这 3 种方法实现解析网页，然后再分别对这 3 种类库进行介绍。

1. 网页分析

部分网页源码为：

```
<ul class="lists">
<li
                        id="3878007"
                        class="list-item"
                        data-title="海王"
                        data-score="8.2"
                        data-star="40"
                        data-release="2018"
                        data-duration="143 分钟"
                        data-region="美国澳大利亚"
                        data-director="温子仁"
                        data-actors="杰森·莫玛 / 艾梅柏·希尔德 / 威廉·达福"
                        data-category="nowplaying"
                        data-enough="True"
                        data-showed="True"
                        data-votecount="105013"
                        data-subject="3878007"
>
```

由分析可知，我们要的电影名称信息在 li 标签的 data-title 属性中。

2. 编写代码

爬虫完整的源码展示如下：

```
import requests
from lxml import etree                          #导入库
from bs4 import BeautifulSoup
import re
import time
#定义爬虫类
class Spider():
    def __init__(self):
        self.url = 'https://movie.douban.com/cinema/nowplaying/beijing/'
        self.headers = {
            'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/70.0.3538.77 Safari/537.36'
        }
        r = requests.get(self.url,headers=self.headers)
        r.encoding = r.apparent_encoding
        self.html = r.text
    def lxml_find(self):
        '''用 lxml 解析'''
        start = time.time()                     #3 种方式速度对比
        selector = etree.HTML(self.html)    #转换为 lxml 解析的对象
        titles = selector.xpath('//li[@class="list-item"]/@data-title') #这里返回的
                                                                        #是一个列表
        for each in titles:
            title = each.strip()                #去掉字符左右的空格
            print(title)
        end = time.time()
        print('lxml 耗时', end-start)

    def BeautifulSoup_find(self):
        '''用 BeautifulSoup 解析'''
        start = time.time()
        soup = BeautifulSoup(self.html, 'lxml') #转换为 BeautifulSoup 的解析对象,其中'lxml'为
                                                #解析方式
        titles = soup.find_all('li', class_='list-item')
        for each in titles:
            title = each['data-title']
            print(title)
        end = time.time()
        print('BeautifulSoup 耗时', end-start)
    def re_find(self):
        '''用 re 解析'''
        start = time.time()
        titles = re.findall('data-title="(.+)"',self.html)
        for each in titles:
            print(each)
        end = time.time()
        print('re 耗时', end-start)
if __name__ == '__main__':
    spider = Spider()
    spider.lxml_find()
    spider.BeautifulSoup_find()
    spider.re_find()
```

3. 爬取结果

爬取的结果输出如下：

```
哥斯拉 2: 怪兽之王
哆啦 A 梦: 大雄的月球探险记
阿拉丁
潜艇总动员: 外星宝贝计划
托马斯大电影之世界探险记
巧虎大飞船历险记
大侦探皮卡丘
…
妈阁是座城
lxml 耗时 0.019963502883911133
哥斯拉 2: 怪兽之王
哆啦 A 梦: 大雄的月球探险记
阿拉丁
潜艇总动员: 外星宝贝计划
托马斯大电影之世界探险记
巧虎大飞船历险记
大侦探皮卡丘
…
妈阁是座城
BeautifulSoup 耗时 0.12004613876342773
哥斯拉 2: 怪兽之王
哆啦 A 梦: 大雄的月球探险记
阿拉丁
潜艇总动员: 外星宝贝计划
托马斯大电影之世界探险记
巧虎大飞船历险记
大侦探皮卡丘
…
妈阁是座城
re 耗时 0.039952754974365234
```

下面分别对这 3 个类库进行分析。

5.2 正则表达式解析网页

正则表达式并不是 Python 的一部分。正则表达式是用于处理字符串的强大工具，拥有自己独特的语法以及一个独立的处理引擎，效率上可能不如 str 自带的方法，但功能十分强大。得益于这一点，在提供了正则表达式的语言中，正则表达式的语法都是一样的，区别只在于不同的编程语言实现支持的语法数量不同；但不用担心，不被支持的语法通常是不常用的部分。

图 5-1 展示了使用正则表达式进行匹配的流程。

正则表达式的大致匹配过程是：依次拿出表达式和文本中的字符比较，如果每一个字符都能匹配，则匹配成功；一旦有匹配不成功的字符则匹配失败。如果表达式中有量词或边界，那么这个过程会稍微有一些不同。

在提取网页中的数据时，可以先把源代码变成字符串，然后用正则表达式匹配想要的数

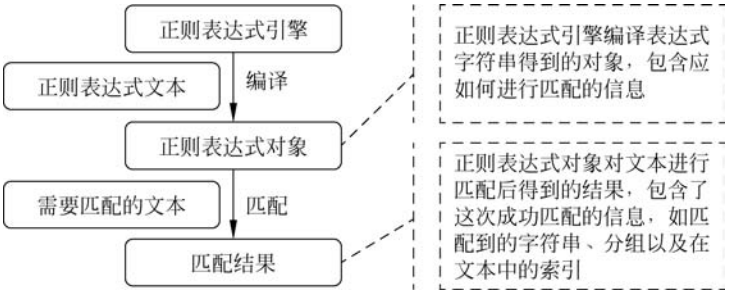

图 5-1　正则表达式进行匹配

据。使用正则表达式可以迅速地用极简单的方式实现字符串的复杂控制。

表 5-1 是常见的正则字符和含义。

表 5-1　常见的正则字符和含义

模式	描　　述	模式	描　　述
.	匹配任意字符，除了换行符	\s	匹配空白字符
*	匹配前一个字符 0 次或多次	\S	匹配任何非空白字符
+	匹配前一个字符 1 次或多次	\d	匹配数字，等价于[0-9]
?	匹配前一个字符 0 次或 1 次	\D	匹配任何非数字，等价于[^0-9]
^	匹配字符串开头	\w	匹配字母数字，等价于[A-ZA-Z0-9]
$	匹配字符串末尾	\W	匹配非字母数字，等价于[^A-ZA-Z0-9]
()	匹配括号内的表达式，也表示一个组	[]	用来表示一组字符

下面介绍 Python 正则表达式的 3 种方法，分别是 match、search 和 findall。

5.2.1　字符串匹配

本节利用 Python 中的 re.match 实现字符串匹配并找到匹配的位置。而 re.match 的意思是从字符串起始位置匹配一个模式，如果从起始位置匹配不了，match()就返回 none。

re.match 的语法格式为：

```
re.match(string[, pos[, endpos]]) | re.match(pattern, string[, flags])
```

match 只找到一次可匹配的结果即返回。

这个方法将从 string 的 pos 下标处开始尝试匹配 pattern；如果 pattern 结束时仍可匹配，则返回一个 match 对象；如果匹配过程中 pattern 无法匹配，或者匹配未结束就已到达 endpos，则返回 none。

pos 和 endpos 的默认值分别为 0 和 len(string)；re.match()无法指定这两个参数，参数 flags 用于编译 pattern 时指定匹配模式。

注意：这个方法并不是完全匹配。当 pattern 结束时，若 string 还有剩余字符，则仍然视为成功。想要完全匹配，可以在表达式末尾加上边界匹配符'$'。

【例 5-1】 使用两个字符串匹配并找到匹配的位置。

```
# encoding: UTF-8
import re
m = re.match('www','www.taobao.com')
```

```
print('匹配的结果: ',m)
print('匹配的起始与终点: ',m.span())
print('匹配的起始位置: ',m.start())
print('匹配的终点位置: ',m.end())
```

运行程序,输出如下:

```
匹配的结果: <_sre.SRE_Match object; span = (0, 3), match = 'www'>
匹配的起始与终点: (0, 3)
匹配的起始位置: 0
匹配的终点位置: 3
```

上面例子中的 pattern 只是一个字符串,也可以把 pattern 改成正则表达式,从而匹配具有一定模式的字符串,例如:

```
# encoding: UTF - 8
import re
line = 'Fat apples are smarter than bananas,is it right?'
m = re.match(r'(\w + ) (\w + )(?P < sign >. * )',line)
print('匹配的整句话',m.group(0))
print('匹配的第一个结果',m.group(1))
print('匹配的第二个结果',m.group(2))
print('匹配的结果列表',m.group())
```

运行程序,输出如下:

```
匹配的整句话 Fat apples are smarter than bananas,is it right?
匹配的第一个结果 Fat
匹配的第二个结果 apples
匹配的结果列表 Fat apples are smarter than bananas,is it right?
```

为什么要在 match 的模式前加上 r 呢?

r'(\w+) (\w+)(? P < sign >. *)'前面的 r 的意思是 raw string,代表纯粹的字符串,使用它就不会对引号中的反斜杠'\'进行特殊处理。因为在正则表达式中有一些类似'\d'(匹配任何数字)的模式,所以模式中的单个反斜杠'\'符号都要进行转义。

假如需要匹配文本中的字符"\",使用编程语言表示的正则表达式里就需要 4 个反斜杠“\\\\”,前两个反斜杠“\\”和后两个反斜杠“\\”各自在编程语言中转义成一个反斜杠“\”,所以 4 个反斜杠“\\\\”就转义成了两个反斜杠“\\”,这两个反斜杠“\\”最终在正则表达式中转义成一个反斜杠“\”。

5.2.2 起始位置匹配字符串

re.match 只能从字符串的起始位置进行匹配,而 re.search 扫描整个字符串并返回。re.search()方法扫描整个字符串,并返回第一个成功的匹配,如果匹配失败,则返回 None。

与 re.match()方法不同,re.match()方法要求必须从字符串的开头进行匹配,如果字符串的开头不匹配,那么整个匹配就失败了; re.search()并不要求必须从字符串的开头进行匹配,也就是说,正则表达式可以是字符串的一部分。

re.search()的语法格式为:

```
re.search(pattern, string, flags = 0)
```

其中,pattern: 正则中的模式字符串。string: 要被查找替换的原始字符串。flags: 标志位,

用于控制正则表达式的匹配方式，如：是否区分大小写、多行匹配等。

【例 5-2】 从起始位置匹配字符串演示实例。

```
import re
content = 'Hello 123456789 Word_This is just a test 666 Test'
result = re.search('(\d+).*?(\d+).*', content)
print(result)
print(result.group())  # print(result.group(0)) 同样效果的字符串
print(result.groups())
print(result.group(1))
print(result.group(2))
```

运行程序，输出如下：

```
<_sre.SRE_Match object; span=(6, 49), match='123456789 Word_This is just a test 666 Test'>
123456789 Word_This is just a test 666 Test
('123456789', '666')
123456789
666
```

适当调用以下代码，可实现数字匹配。例如：

```
import re
content = 'Hello 123456789 Word_This is just a test 666 Test'
result = re.search('(\d+)', content)
print(result)
print(result.group())  # print(result.group(0)) 同样效果的字符串
print(result.groups())
print(result.group(1))
```

运行程序，输出如下：

```
<_sre.SRE_Match object; span=(6, 15), match='123456789'>
123456789
('123456789',)
123456789
```

5.2.3 所有子串匹配

re.findall()在字符串中找到正则表达式所匹配的所有子串，并返回一个列表；如果没有找到匹配的，则返回空列表。返回结果是列表类型，需要遍历一下才能依次获取每组内容。

re.findall()的语法格式为：

```
findall(patern, string, flags=0)
```

其中，pattern：正则中的模式字符串。string：要被查找替换的原始字符串。flags：标志位，用于控制正则表达式的匹配方式，如：是否区分大小写、多行匹配等。

【例 5-3】 匹配所有子串演示。

```
import re
content = 'Hello 123456789 Word_This is just a test 666 Test'
results = re.findall('\d+', content)
print(results)
for result in results:
    print(result)
```

运行程序，输出如下：

```
['123456789', '666']
```

```
123456789
666
```

findall 与 match、search 不同的是，findall 能够找到所有匹配的结果，并且以列表的形式返回。

5.2.4 Requests 爬取猫眼电影排行

本节利用 Requests 库和正则表达式来爬取猫眼电影 TOP100 的相关内容。Requests 比 urllib 使用更加方便，在此选用正则表达式来作为解析工具。

【例 5-4】 利用 Requests 和正则表达式爬取猫眼电影排行信息。

```
# - * - coding: utf - 8 - * -
import re
import os
import json
import requests
from multiprocessing import Pool
from requests.exceptions import RequestException
def get_one_page(url):
    '''
获取网页 html 内容并返回
    '''
    try:
        # 获取网页 html 内容
        response = requests.get(url)
        # 通过状态码判断是否获取成功
        if response.status_code == 200:
            return response.text
        return None
    except RequestException:
        return None
def parse_one_page(html):
    '''
解析 HTML 代码，提取有用信息并返回
    '''
    # 用正则表达式进行解析
    pattern = re.compile('<dd>.*?board - index.*?>(\d + )</i>.*?data - src = "(.*?)".
*?name">'
        + '<a.*?>(.*?)</a>.*?"star">(.*?)</p>.*?releasetime">(.*?)</p>'
        + '.*?integer">(.*?)</i>.*?fraction">(.*?)</i>.*?</dd>', re.S)
    # 匹配所有符合条件的内容
    items = re.findall(pattern, html)
    for item in items:
        yield {
            'index': item[0],
            'image': item[1],
            'title': item[2],
            'actor': item[3].strip()[3:],
            'time': item[4].strip()[5:],
            'score': item[5] + item[6]
        }
def write_to_file(content):
    '''
将文本信息写入文件
```

```
        '''
        with open('result.txt', 'a', encoding = 'utf - 8') as f:
            f.write(json.dumps(content, ensure_ascii = False) + '\n')
            f.close()
    def save_image_file(url, path):
        '''
    保存电影封面
        '''
        ir = requests.get(url)
        if ir.status_code == 200:
            with open(path, 'wb') as f:
                f.write(ir.content)
                f.close()
    def main(offset):
        url = 'http://maoyan.com/board/4?offset = ' + str(offset)
        html = get_one_page(url)
        # 封面文件夹不存在则创建
        if not os.path.exists('covers'):
            os.mkdir('covers')
        for item in parse_one_page(html):
            print(item)
            write_to_file(item)
            save_image_file(item['image'], 'covers/' + '%03d' % int(item['index']) + item['title'] +
    '.jpg')
    if __name__ == '__main__':
        # 使用多进程提高效率
        pool = Pool()
        pool.map(main, [i * 10 for i in range(10)])
```

运行程序,输出如下:

```
{'index': '1', 'image': 'https://p1.meituan.net/movie/20803f59291c47e1e116c11963ce019e68711.
jpg@160w_220h_1e_1c', 'title': '霸王别姬', 'actor': '张国荣,张丰毅,巩俐', 'time': '1993 - 01 -
01', 'score': '9.5'}
...
{'index': '100', 'image': 'https://p0.meituan.net/movie/c304c687e287c7c2f9e22cf78257872d277201.
jpg@160w_220h_1e_1c', 'title': '龙猫', 'actor': '秦岚,糸井重里,岛本须美', 'time': '2018 - 12 - 14',
'score': '9.1'}
```

5.3 BeautifulSoup 解析网页

BeautifulSoup 是 Python 的一个 HTML 解析框架,利用它可以方便地处理 HTML 和 XML 文档。BeautifulSoup 有 3 和 4 两个版本,目前 3 已经停止开发。所以这里学习最新的 BeautifulSoup4。

首先是利用 pip 安装 BeautifulSoup。使用下面的命令。

```
pip install beautifulsoup4
```

安装 BeautifulSoup 后,就可以开始使用它了。

BeautifulSoup 只是一个 HTML 解析库,所以如果想解析网上的内容,第一件事情就是把它下载下来。对于不同的网站,可能对请求进行过滤。糗事百科的网站就会直接拒绝没有 UA 的请求。所以如果要爬这样的网站,首先需要把请求伪装成浏览器的样子。具体网站具体分析,经过测试,糗事百科只要设置了 UA 就可以爬取到内容,对于其他网站,你需要测试一下才能确定什么设置可用。

有了 Request 对象还不够，还需要实际发起请求才行。下面代码的最后一句就使用了 Python3 的 urllib 库发起了一个请求。urlopen(req)方法返回的是 Reponse 对象，调用它的 read()函数获取整个结果字符串。最后调用 decode('utf-8')方法将它解码为最终结果，如果不调用这一步，那么汉字等非 ASCII 字符就会变成\xXXX 这样的转义字符。

```
import urllib.request as request
user_agent = 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/56.0.2924.87 Safari/537.36'
headers = {'User-Agent': user_agent}
req = request.Request('http://www.qiushibaike.com/', headers=headers)
page = request.urlopen(req).read().decode('utf-8')
```

有了文档字符串，就可以开始解析文档了。第一步是建立 BeautifulSoup 对象，这个对象在 bs4 模块中。注意，在建立对象的时候可以额外指定一个参数，作为实际的 HTML 解析器。解析器的值可以指定 html.parser，这是内置的 HTML 解析器。更好的选择是使用下面的 lxml 解析器，不过它需要额外安装一下，使用 pip install lxml 就可以安装。

```
mport bs4
soup = bs4.BeautifulSoup(page, "lxml")
```

有了 BeautifulSoup 对象，就可以开始解析了。首先介绍 BeautifulSoup 的对象种类，常用的有标签(bs4.element.Tag)以及文本(bs4.element.NavigableString)等，其中，注解等对象不常用，在此不展开介绍。在标签对象上，可以调用一些查找方法例如 find_all 等，还有一些属性返回标签的父节点、兄弟节点、直接子节点、所有子节点等。在文本对象上，可以调用.string 属性获取具体文本。

基本所有 BeautifulSoup 的遍历方法操作都需要通过 BeautifulSoup 对象来使用。使用方式主要有两种：一是直接引用属性，例如 soup.title，会返回第一个符合条件的节点；二是通过查找方法，例如 find_all，传入查询条件来查找结果。

接下来了解查询条件。查询条件可以是：字符串，会返回对应名称的节点；正则表达式，按照正则表达式匹配；列表，会返回所有匹配列表元素的节点；真值 True，会返回所有标签节点，不会返回字符节点；方法，可以编写一个方法，按照自己的规则过滤，然后将该方法作为查询条件。

BeautifulSoup 支持 Python 标准库中的 HTML 解析器，还支持一些第三方的解析器。表 5-2 列出了主要的解析器及其优缺点。

表 5-2 解析器及其优缺点

解 析 器	使 用 方 法	优 点	缺 点
Python 标准库	BeautifulSoup(markup, "html.parser")	Python 的内置标准库，执行速度适中，文档容错能力强	在 Python3.2.2 前的版本中文档容错能力差
lxml HTML 解析器	BeautifulSoup(markup, "lxml")	速度快，文档容错能力强	需要安装 C 语言库
lxml XML 解析器	BeautifulSoup(markup, ["lxml","xml"])	速度快，唯一支持 XML 的解析器	需要安装 C 语言库
html5lib	BeautifulSoup(markup, "html5lib")	最好的容错性，以浏览器的方式解析文档，生成 HTML5 格式的文档	速度慢 不依赖外部扩展

使用 lxml 的解析器将会解析得更快,建议大家使用。

【例 5-5】 利用 requests 和 BeautifulSoup 爬取猫眼电影排行信息。

```
import requests
from bs4 import BeautifulSoup
import os
import time
start = time.clock()                                    #添加程序运行计时功能
file_path = 'D:\python3.6\scrapy\猫眼'                  #定义文件夹,方便后续 check 文件夹是否存在
file_name = 'maoyan.txt'                                #自定义命名文件名称
file = file_path + '\\' + file_name                     #创建文件全地址,方便后续引用
url = "http://maoyan.com/board/4"                       #获取 url 的开始页
headers = {"User - Agent": "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/65.0.3325.181 Safari/537.36"}
def Create_file(file_path,file):                        #定义检查和创建目标文件夹和文件的函数
    if os.path.exists(file_path) == False:              #check 文件夹不存在
        os.makedirs(file_path)                          #创建新的自定义文件夹
        fp = open(file,'w')                             #创建新的自定义文件
    """ "w" 以写方式打开,只能写文件,如果文件不存在,创建该文件; 如果文件已存在,先清空,再
打开文件 """
    elif os.path.exists(file_path) == True: # check 文件夹存在
        with open(file, 'w', encoding = 'utf - 8') as f: #打开目标文件夹中的文件
            f.seek(0)
    """f.seek(offset[,where])把文件指针移动到相对于 where 的 offset 位置.where 为 0 表示文
件开始处,这是默认值; 1 表示当前位置; 2 表示文件结尾"""
            f.truncate()
    """清空文件内容,注意: 仅当以 "r + " "rb + " "w" "wb" "wb + "等以可写模式打开的文件才可以
执行该功能"""
    def get_all_pages(start):
    #定义获取所有 pages 页的目标内容的函数
    pages = []
    for n in range(0,100,10):
    #获取 offset 的步进值,注意把 int 的 n 转换为 str
    #遍历所有的 url,并获取每一页 page 的目标内容
        if n == 0:
            url = start
        else:
            url = start + '?offset = ' + str(n)
        r = requests.get(url, headers = headers)
        soup = BeautifulSoup(r.content, 'lxml')
        page = soup.find_all(name = 'dd')
        #获取该 page 的所有 dd 节点的内容
        pages.extend(page)
        #将获取的所有 page list 扩展成 pages,方便下面遍历每个 dd 节点内容
        return pages
    #返回所有 pages 的 dd 节点的内容,每个 dd 节点内容都以 list 方式存储其中
    Create_file(file_path,file)
    text = get_all_pages(url)
    for film in text:
    #遍历列表 text 中的所有元素,也就是每个 dd 节点内容
    #这个 for 循环应该优化成自定义函数形式;
    dict = {}
    #创建空 dict
```

```
        #print(type(film))          #确认 film 属性为 tag,故可以使用 tag 相关的方法处理 film
        #print('*'*50)              #可以分隔检查输出的内容,方便对照
        dict['Index'] = film.i.string  #选取 film 的第一个子节点 i 的 string 属性值
        #获取第三重直接子孙节点,例如下面注释中的<div class="movie-item-info">节点全部元素
        comment1 = film.div.div.div
        name = comment1.find_all(name='p')[0].string
        star = comment1.find_all(name='p')[1].string
        releasetime = comment1.find_all(name='p')[2].string
        dict['name'] = name
        dict['star'] = str.strip(star)
        dict['releasetime'] = releasetime
        comment2 = comment1.find_next_sibling()
        """获取第三重直接子孙节点的 next 节点,例如下面注释中的<div class="movie-item-
    info">节点全部元素"""
        #print(comment2)             #检查 comment2 是否为目标文本
        sco1 = comment2.i.string
        sco2 = comment2.i.find_next_sibling().string
        #print(type(sco1))           #判断 sco1 为 tag 类型
        #print(sco1)                 #检查 sco1 是否为目标输出内容
        score = (sco1.string + str.strip(sco2))   #获取合并后的 score 字符串
        dict['score'] = score
        print(dict)                 #检查 dict 是否为目标输出内容
        with open(file, 'a', encoding='utf-8') as f: #以打开目标 file 文件
            f.write(str(dict) + '\n') #注意添加换行符 '\n',实现每个 dict 自动换行写入 txt 中
        end = time.clock()          #添加程序运行计时功能
        print('爬取完成','\n','耗时: ',end-start)   #添加程序运行计时功能
```

运行程序,效果如图 5-2 所示。

图 5-2　爬取猫眼电影排行信息

5.4 PyQuery 解析库

前面介绍了 BeautifulSoup 的用法，它是一个非常强大的网页解析库，你是否觉得它的一些方法用起来有点不适用？有没有觉得它的 CSS 选择器的功能没有那么强大？

下面来介绍一个更适合的解析库——PyQuery。PyQuery 库是 jQuery 的 Python 实现，能够以 jQuery 的语法来操作解析 HTML 文档，易用性和解析速度都很好，使用起来还是可以的，有些地方用起来很方便简洁。

5.4.1 使用 PyQuery

如果之前没有安装 PyQuery，可在命令窗口中直接使用 pip install PyQuery 进行安装。

1. 初始化

像 BeautifulSoup 一样，初始化 PyQuery 的时候，也需要传入 HTML 文本来初始化一个 PyQuery 对象，它的初始化方式有多种，比如直接传入字符串、传入 URL、传入文件名，等等。下面详细介绍。

1）字符串初始化

首先，通过一个实例来感受一下：

```
html = """
<html lang="en">
<head>
简单好用的
<title>PyQuery</title>
</head>
<body>
<ul id="container">
<li class="object-1">Python</li>
<li class="object-2">爬虫</li>
<li class="object-3">好</li>
</ul>
</body>
</html>
"""
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
print(type(doc))
print(doc)
```

运行程序，输出如下：

```
<class 'pyquery.pyquery.PyQuery'>
<html lang="en">
<head>
简单好用的
<title>PyQuery</title>
</head>
```

```
<body>
<ul id="container">
<li class="object-1">Python</li>
<li class="object-2">爬虫</li>
<li class="object-3">好</li>
</ul>
</body>
</html>
```

这里首先引入 PyQuery 这个对象，取别名为 pq，然后声明了一个长 HTML 字符串，并将其当作参数传递给 PyQuery 类，这样就成功完成了初始化。接下来，将初始化的对象传入 CSS 选择器。在这个实例中，传入 li 节点，这样就可以选择所有的 li 节点。

2）对网址响应进行初始化

初始化的参数不仅可以以字符串的形式传递，还可以传入网页的 URL，此时只需要指定参数为 url 即可：

```
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
response = pq(url = 'https://www.baidu.com')
print(type(response))
print(response)
```

运行程序，输出如下：

```
<class 'pyquery.pyquery.PyQuery'>
<html><head><meta http-equiv="content-type" content="text/html;charset=utf-8"/>
<meta http-equiv="X-UA-Compatible" content="IE=Edge"/><meta content="always" name="refe
...
```

3）HTML 文件初始化

除了传递 URL，还可以传递本地的文件名，此时将参数指定为 filename 即可：

```
#filename 参数为 html 文件路径
test_html = pq(filename = 'test.html')
print(type(test_html))
print(test_html)
```

运行程序，输出如下：

```
<class 'pyquery.pyquery.PyQuery'><html lang="en">
<head>
<title>PyQuery 学习</title>
</head>
<body>
<ul id="container">
<li class="object-1"/>
<li class="object-2"/>
<li class="object-3"/>
</ul>
</body>
</html>
```

这里需要有一个本地 HTML 文件 test.html，其内容是待解析的 HTML 字符串。这样它会首先读取本地的文件内容，然后用文件内容以字符串的形式传递给 PyQuery 类来初始化。

以上 3 种初始化方式均可，最常用的初始化方式是以字符串形式传递。

2. CSS 选择器

首先，用一个实例来感受 PyQuery 的 CSS 选择器的用法：

```
html = """
<html lang="en">
<head>
简单好用的
<title>PyQuery</title>
</head>
<body>
<ul id="container">
<li class="object-1">Python</li>
<li class="object-2">爬虫</li>
<li class="object-3">好</li>
</ul>
</body>
</html>
"""
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
print(doc('#container'))
print(type(doc('#container')))
```

运行程序，输出如下：

```
<ul id="container">
<li class="object-1">Python</li>
<li class="object-2">爬虫</li>
<li class="object-3">好</li>
</ul>
<class 'pyquery.pyquery.PyQuery'>
```

这里初始化 PyQuery 对象之后，传入了一个 CSS 选择器 #container.list li，它的意思为选取 id 为 container 的节点，然后再选取其内部的 class 为 list 的节点内部的所有 li 节点。然后，打印输出，可以看到，成功获取了符合条件的节点。最后，将它的类型打印输出，可以看到，它的类型依然是 PyQuery 类型。

再例如，打印 class 为 object-1 的标签：

```
print(doc('.object-1'))
```

输出如下：

```
<li class="object-1">Python</li>
```

打印标签名为 body 的标签：

```
print(doc('body'))
```

输出如下：

```
<body>
<ul id="container">
<li class="object-1">Python</li>
<li class="object-2">爬虫</li>
```

```
<li class="object-3">好</li>
</ul>
</body>
```

3. 查找节点

下面介绍一些常用的查询函数。

1）子节点

查找子节点时，需要用到 find()方法，此时传入的参数是 CSS 选择器。

```
html = """
<div>
<ul>
<li class="item-0">first item</li>
<li class="item-1"><a href="link2.html">second item</a></li>
<li class="item-0 active"><a href="link3.html"><span class="bold">third item</span>
</a></li>
<li class="item-1 active"><a href="link4.html">fourth item</a></li>
<li class="item-0"><a href="link5.html">fifth item</a></li>
</ul>
</div>
"""
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
items = doc('.list')
print(type(items))
print(items)
lis = items.find('li')
print(type(lis))
print(lis)
```

运行程序，输出如下：

```
<class 'pyquery.pyquery.PyQuery'>
<ul class="list">
<li class="item-0">first item</li>
<li class="item-1"><a href="link2.html">second item</a></li>
<li class="item-0 active"><a href="link3.html"><span class="bold">third item</span>
</a></li>
<li class="item-1 active"><a href="link4.html">fourth item</a></li>
<li class="item-0"><a href="link5.html">fifth item</a></li>
</ul>
<class 'pyquery.pyquery.PyQuery'>
<li class="item-0">first item</li>
<li class="item-1"><a href="link2.html">second item</a></li>
<li class="item-0 active"><a href="link3.html"><span class="bold">third item</span>
</a></li>
<li class="item-1 active"><a href="link4.html">fourth item</a></li>
<li class="item-0"><a href="link5.html">fifth item</a></li>
```

首先，选取 class 为 list 的节点，然后调用了 find()方法，传入 CSS 选择器，选取其内部的 li 节点，最后打印输出。可以发现，find()方法会将符合条件的所有节点选择出来，结果的类型是 PyQuery 类型。

其实 find()的查找范围是节点的所有子孙节点，而如果只想查找子节点，那么可以用 children()方法：

```
lis = items.children()
print(type(lis))
print(lis)
```

运行程序，输出如下：

```
<class 'pyquery.pyquery.PyQuery'>
<li class="item-0">first item</li>
<li class="item-1"><a href="link2.html">second item</a></li>
<li class="item-0 active"><a href="link3.html"><span class="bold">third item</span>
</a></li>
<li class="item-1 active"><a href="link4.html">fourth item</a></li>
<li class="item-0"><a href="link5.html">fifth item</a></li>
```

如果要筛选所有子节点中符合条件的节点，比如想筛选出子节点中 class 为 active 的节点，可以向 children()方法传入 CSS 选择器.active：

```
lis = items.children('.active')
print(lis)
```

运行程序，输出如下：

```
<li class="item-0 active"><a href="link3.html"><span class="bold">third item</span>
</a></li>
<li class="item-1 active"><a href="link4.html">fourth item</a></li>
```

可以看到，输出结果已经做了筛选，留下了 class 为 active 的节点。

2）父节点

在 Python 中，可以用 parent()方法来获取某个节点的父节点，例如：

```
html = """
<div class="wrap">
<div id="container">
<ul class="list">
<li class="item-0">first item</li>
<li class="item-1"><a href="link2.html">second item</a></li>
<li class="item-0 active"><a href="link3.html"><span class="bold">third item</span>
</a></li>
<li class="item-1 active"><a href="link4.html">fourth item</a></li>
<li class="item-0"><a href="link5.html">fifth item</a></li>
</ul>
</div>
</div>
"""
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
items = doc('.list')
container = items.parent()
print(type(container))
print(container)
```

运行程序，输出如下：

```
<class 'pyquery.pyquery.PyQuery'>
```

```
<div id="container">
<ul class="list">
<li class="item-0">first item</li>
<li class="item-1"><a href="link2.html">second item</a></li>
<li class="item-0 active"><a href="link3.html"><span class="bold">third item</span>
</a></li>
<li class="item-1 active"><a href="link4.html">fourth item</a></li>
<li class="item-0"><a href="link5.html">fifth item</a></li>
</ul>
</div>
```

代码中，首先用.list 选取 class 为 list 的节点，然后调用 parent()方法得到其父节点，其类型依然是 PyQuery。

此处的父节点是该节点的直接父节点，也就是说，它不会再去查找父节点的父节点，即祖先节点。但是如果想获取某个祖先节点，该怎么办呢？可以用 parents()方法：

```
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
items = doc('.list')
parents = items.parents()
print(type(parents))
print(parents)
```

运行程序，输出如下：

```
<class 'pyquery.pyquery.PyQuery'>
<div class="wrap">
<div id="container">
<ul class="list">
<li class="item-0">first item</li>
<li class="item-1"><a href="link2.html">second item</a></li>
<li class="item-0 active"><a href="link3.html"><span class="bold">third item</span>
</a></li>
<li class="item-1 active"><a href="link4.html">fourth item</a></li>
<li class="item-0"><a href="link5.html">fifth item</a></li>
</ul>
</div>
</div><div id="container">
<ul class="list">
<li class="item-0">first item</li>
<li class="item-1"><a href="link2.html">second item</a></li>
<li class="item-0 active"><a href="link3.html"><span class="bold">third item</span>
</a></li>
<li class="item-1 active"><a href="link4.html">fourth item</a></li>
<li class="item-0"><a href="link5.html">fifth item</a></li>
</ul>
</div>
```

可以看到，输出结果有两个：一个是 class 为 wrap 的节点；另一个是 id 为 container 的节点。也就是说，parents()方法会返回所有的祖先节点。

要想筛选某个祖先节点，可以向 parents()方法传入 CSS 选择器，这样就会返回祖先节点中符合 CSS 选择器条件的节点：

```
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
items = doc('.list')
parent = items.parents('.wrap')
print(parent)
```

运行程序，输出如下：

```
<div class = "wrap">
<div id = "container">
<ul class = "list">
<li class = "item - 0"> first item </li>
<li class = "item - 1"><a href = "link2.html"> second item </a></li>
<li class = "item - 0 active"><a href = "link3.html"><span class = "bold"> third item </span>
</a></li>
<li class = "item - 1 active"><a href = "link4.html"> fourth item </a></li>
<li class = "item - 0"><a href = "link5.html"> fifth item </a></li>
</ul>
</div>
</div>
```

由输出结果可以看到，输出结果少了一个节点，只保留了 class 为 wrap 的节点。

3）兄弟节点

除了前面介绍的子节点、父节点外，还有一种节点，那就是兄弟节点。如果要获取兄弟节点，可以使用 siblings()方法。例如：

```
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
li = doc('.list .item - 0.active')
print(li.siblings())
```

运行程序，输出如下：

```
<li class = "item - 1"><a href = "link2.html"> second item </a></li>
<li class = "item - 0"> first item </li>
<li class = "item - 1 active"><a href = "link4.html"> fourth item </a></li>
<li class = "item - 0"><a href = "link5.html"> fifth item </a></li>
```

从结果可以看到，这正是刚才所说的 4 个兄弟节点。

如果要筛选某个兄弟节点，依然可以向 siblings 方法传入 CSS 选择器，这样就会从所有兄弟节点中挑选出符合条件的节点了：

```
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
li = doc('.list .item - 0.active')
print(li.siblings('.active'))
```

运行程序，输出如下：

```
<li class = "item - 1 active"><a href = "link4.html"> fourth item </a></li>
```

这里筛选了 class 为 active 的节点，通过刚才的结果可以观察到，class 为 active 的兄弟节点只有第四个 li 节点，所以结果应该是一个。

4. 遍历

由刚才可观察到，PyQuery 的选择结果可能是多少节点，也可能是单个节点，类型都是 PyQuery，并没有返回像 BeautifulSoup 那样的列表。

对于单个节点来说，可以直接打印输出，也可以直接转成字符串：

```
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
li = doc('.item-0.active')
print(li)
print(str(li))
```

运行程序，输出如下：

```
<li class="item-0 active"><a href="link3.html"><span class="bold">third item</span>
</a></li>
<li class="item-0 active"><a href="link3.html"><span class="bold">third item</span>
</a></li>
```

对于多个节点的结果，就需要遍历来获取了。例如，这里把每个 li 节点进行遍历，需要调用 items()方法：

```
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
lis = doc('li').items()
print(type(lis))
for li in lis:
 print(li,type(li))
```

运行程序，输出如下：

```
<class 'generator'>
<li class="item-0">first item</li>
<class 'pyquery.pyquery.PyQuery'>
<li class="item-1"><a href="link2.html">second item</a></li>
<class 'pyquery.pyquery.PyQuery'>
<li class="item-0 active"><a href="link3.html"><span class="bold">third item</span>
</a></li>
<class 'pyquery.pyquery.PyQuery'>
<li class="item-1 active"><a href="link4.html">fourth item</a></li>
<class 'pyquery.pyquery.PyQuery'>
<li class="item-0"><a href="link5.html">fifth item</a></li>
<class 'pyquery.pyquery.PyQuery'>
```

由结果可发现，调用 items()方法后，会得到一个生成器，遍历一下，就可以逐个得到 li 节点对象了，它的类型也是 PyQuery 类型。每个 li 节点还可以调用前面所说的方法进行选择，比如继续查询子节点，寻找某个祖先节点等，非常灵活。

5. 获取信息

提取到节点之后，最终目的是提取节点所包含的信息。比较重要的信息有两类：一是获取属性，二是获取文本。下面具体说明。

1) 获取属性

提取到某个 PyQuery 类型的节点后，就可以调用 attr()方法来获取属性：

```
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
a = doc('.item-0.active a')
print(a,type(a))
print(a.attr('href'))
```

运行程序,输出如下:

```
<a href="link3.html"><span class="bold">third item</span></a><class 'pyquery.pyquery.
PyQuery'>
link3.html
```

在代码中,首先选中 class 为 item-0 和 active 的 li 节点内的 a 节点,它的类型是 PyQuery。然后调用 attr()方法。在这个方法中传入属性的名称,就可以得到这个属性值了。

此外,也可以通过调用 attr 属性来获取属性,例如:

```
print(a.attr.href)
```

运行程序,输出如下:

```
link3.html
```

这两种方法的结果完全一样。如果选中的是多个元素,然后调用 attr()方法,会出现怎样的结果呢?用实例来测试一下:

```
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
a = doc('a')
print(a,type(a))
print(a.attr('href'))
print(a.attr.href)
```

运行程序,输出如下:

```
<a href="link2.html">second item</a><a href="link3.html"><span class="bold">third
item</span></a><a href="link4.html">fourth item</a><a href="link5.html">fifth item
</a><class 'pyquery.pyquery.PyQuery'>
link2.html
link2.html
```

照理来说,选中的 a 节点有 4 个,打印结果也应该是 4 个,但是当调用 attr()方法时,返回结果却只是第一个。这是因为,当返回结果包含多个节点时,调用 attr()方法,只会得到第一个节点的属性。那么,遇到这种情况,如果想获取所有的 a 节点的属性,就要用到前面所说的遍历了:

```
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
a = doc('a')
for item in a.items():
 print(item.attr('href'))
```

运行程序,输出如下:

```
link2.html
```

```
link3.html
link4.html
link5.html
```

因此，在进行属性获取时，可以观察返回节点是一个还是多个。如果是多个，则需要遍历才能依次获取每个节点的属性。

2）获取文本

获取节点之后的另一个主要操作就是获取其内部的文本了，此时可以调用 text()方法来实现：

```
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
a = doc('.item-0.active a')
print(a)
print(a.text())
```

运行程序，输出如下：

```
<a href="link3.html"><span class="bold">third item</span></a>
third item
```

此处首先选中一个 a 节点，然后调用 text()方法，就可以获取其内部的文本信息了。此时它会忽略掉节点内部包含的所有 HTML，只返回纯文字内容。

但如果想要获取这个节点内部的 HTML 文本，就要用 html()方法了：

```
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
li = doc('.item-0.active')
print(li)
print(li.html())
```

运行程序，输出如下：

```
<li class="item-0 active"><a href="link3.html"><span class="bold">third item</span>
</a></li>
<a href="link3.html"><span class="bold">third item</span></a>
```

在程序中选中了第三个 li 节点，然后调用了 html()方法，它返回的结果应该是 li 节点内的所有 HTML 文本。

这里同样有一个问题，如果选中的结果是多个节点，text()或 html()会返回什么内容？用实例来测试下：

```
html = """
<div class="wrap">
<div id="container">
<ul class="list">
<li class="item-1"><a href="link2.html">second item</a></li>
<li class="item-0 active"><a href="link3.html"><span class="bold">third item</span>
</a></li>
<li class="item-1 active"><a href="link4.html">fourth item</a></li>
<li class="item-0"><a href="link5.html">fifth item</a></li>
</ul>
</div>
```

```
</div>
"""
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
li = doc('li')
print(li.html())
print(li.text())
print(type(li.text))
```

运行程序,输出如下:

```
<a href = "link2.html"> second item </a>
second item third item fourth item fifth item
<class 'method'>
```

结果可能比较出人意料,html()方法返回的是第一个 li 节点的内部 HTML 文本,而 text()返回了所有的 li 节点内部的纯文本,中间用一个空格分隔开,即返回结果是一个字符串。

值得注意的是,如果得到的结果是多个节点,并且想要获取每个节点的内部 HTML 文本,则需要遍历每个节点;而使用 text()方法不需要遍历就可以获取,它对所有节点取文本之后合并成一个字符串。

6. 节点操作

PyQuery 提供了一系列方法来对节点进行动态修改,比如为某个节点添加一个 class,移除某个节点等,这些操作有时会为提取信息带来极大的便利。

由于节点操作的方法太多,下面举几个典型的例子来说明它的用法。

1) addClass 和 removeClass

下面先体会实例演示:

```
html = """
<div class = "wrap">
<div id = "container">
<ul class = "list">
<li class = "item - 0"> first item </li>
<li class = "item - 1"><a href = "link2.html"> second item </a></li>
<li class = "item - 0 active"><a href = "link3.html"><span class = "bold"> third item </span>
</a></li>
<li class = "item - 1 active"><a href = "link4.html"> fourth item </a></li>
<li class = "item - 0"><a href = "link5.html"> fifth item </a></li>
</ul>
</div>
</div>
"""
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
li = doc('.item - 0.active')
print(li)
li.removeClass('active')
print(li)
```

```
li.addClass('active')
print(li)
```

在代码中，首先选中了第三个 li 节点，然后调用 removeClass()方法，将 li 节点的 active 这个 class 移除；接着又调用 addClass()方法，将 class 添加回来。每执行一次操作，就打印输出当前 li 节点的内容。

运行程序，输出如下：

```
<li class="item-0 active"><a href="link3.html"><span class="bold">third item</span>
</a></li>
<li class="item-0"><a href="link3.html"><span class="bold">third item</span></a></li>
<li class="item-0 active"><a href="link3.html"><span class="bold">third item</span>
</a></li>
```

从结果可看到，一共输出了 3 次。第二次输出时，li 节点的 active 这个 class 被移除了，第三次 class 又添加回来了。

所以，addClass()和 removeClass()这些方法可以动态改变节点的 class 属性。

2) attr、text 和 html

当然，除了操作 class 这个属性外，也可以用 attr()方法对属性进行操作。此外，还可以用 text()和 html()方法来改变节点内部的内容。例如：

```
html = """
<ul class="list">
<li class="item-0 active"><a href="link3.html"><span class="bold">third item</span>
</a></li>
</ul>
"""
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
li = doc('.item-0.active')
print(li)
li.attr('name','link')
print(li)
li.text('changed item')
print(li)
li.html('<span>changed item</span>')
print(li)
```

运行程序，输出如下：

```
<li class="item-0 active"><a href="link3.html"><span class="bold">third item</span>
</a></li>
<li class="item-0 active" name="link"><a href="link3.html"><span class="bold">third
item</span></a></li>
<li class="item-0 active" name="link">changed item</li>
<li class="item-0 active" name="link"><span>changed item</span></li>
```

这里首先选中 li 节点，然后调用 attr()方法来修改属性，其中该方法的第一个参数为属性名，第二个参数为属性值。接着，调用 text()和 html()方法来改变节点内部的内容。两次操作后，分别打印输出当前的 li 节点。

由结果可发现，调用 attr()方法后，li 节点多了一个原本不存在的属性 name，其值为

link。接着调用 text()方法,传入文本之后,li 节点内部的文本全部被改为传入的字符串文本了。最后,调用 html()方法传入 HTML 文本后,li 节点内部又变为传入的 HTML 文本了。

所以,如果 attr()方法只传入第一个参数的属性名,则是获取这个属性值;如果传入第二个参数,则可以用来修改属性值。text()和 html()方法如果不传参数,则是获取节点内纯文本和 HTML 文本;如果传入参数,则进行赋值。

3) remove()

remove()的方法即为移除,它有时会为信息的提取带来非常大的便利。下面有一段 HTML 文本:

```
html = """
<div class="wrap">
Hello, World
<p>This is a paragraph.</p>
</div>
"""
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
wrap = doc('.wrap')
print(wrap.text())
```

运行程序,输出如下:

```
Hello, World
This is a paragraph.
```

我们想提取的是“Hello, World”这个字符串,而这个结果还包含了内部的 p 节点的内容,也就是说,text()把所有的纯文本全提取出来了。如果想去掉 p 节点内部的文本,可以选择再把 p 节点内的文本提取一遍,然后从整个结果中移除这个子串,但这个做法明显比较烦琐。而 remove()方法就可以实现该功能,例如:

```
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
wrap = doc('.wrap')
wrap.find('p').remove()
print(wrap.text())
```

运行程序,输出如下:

```
Hello, World
```

以上代码的思路是:首先选中 p 节点,然后调用 remove()方法将其移除,这时 wrap 内部就只剩下“Hello,World”了,再利用 text()方法提取即可。

7. 伪类选择器

CSS 选择器之所以强大,有一个重要的原因,那就是它支持多种多样的伪类选择器,例如,选择第一个节点、最后一个节点、奇偶数节点、包含某一文本的节点等等。实例如下:

```
html = """
<div class="wrap">
<div id="container">
```

```
<ul class="list">
<li class="item-0">first item</li>
<li class="item-1"><a href="link2.html">second item</a></li>
<li class="item-0 active"><a href="link3.html"><span class="bold">third item</span>
</a></li>
<li class="item-1 active"><a href="link4.html">fourth item</a></li>
<li class="item-0"><a href="link5.html">fifth item</a></li>
</ul>
</div>
</div>
"""
from pyquery import PyQuery as pq
#初始化为 PyQuery 对象
doc = pq(html)
li = doc('li:first-child')
print(li)
li = doc('li:last-child')
print(li)
li = doc('li:nth-child(2)')
print(li)
li = doc('li:gt(2)')
print(li)
li = doc('li:nth-child(2n)')
print(li)
li = doc('li:contains(second)')
print(li)
```

运行程序,输出如下:

```
<li class="item-0">first item</li>
<li class="item-0"><a href="link5.html">fifth item</a></li>
<li class="item-1"><a href="link2.html">second item</a></li>
<li class="item-1 active"><a href="link4.html">fourth item</a></li>
<li class="item-0"><a href="link5.html">fifth item</a></li>
<li class="item-1"><a href="link2.html">second item</a></li>
<li class="item-1 active"><a href="link4.html">fourth item</a></li>
<li class="item-1"><a href="link2.html">second item</a></li>
```

在代码中,使用了 CSS 的伪类选择器,依次选择了第一个 li 节点、最后一个 li 节点、第二个 li 节点、第三个 li 之后的 li 节点、偶数位置的 li 节点、包含 second 文本的 li 节点。

5.4.2 PyQuery 爬取煎蛋网商品图片

图片一般都是以链接的形式出现在 HTML 文本中,因此只需要找到图片连接即可(一般是在 img src 中),这时再把图片 url 打开,利用 content 保存成具体的文件。这里使用的 hashlib 是一个编码库,为了使得每一个图片的名字不一样,就用 md5 这个方法把图片的内容进行了编码。爬取煎蛋网加入了一定的反爬取措施,即并不是直接将图片的 url 列出来,而是利用一个.js 文件,在每一次加载图片的时候都要加载这个.js 文件,进而把图片的 url 解析出来。实现代码为:

```
import requests
from pyquery import PyQuery as pq
import hashlib
```

```
import base64
from hashlib import md5
def ty(body):
    print(type(body))
# 处理 md5 编码问题
def handle_md5(hd_object):
    return hashlib.md5(hd_object.encode('utf-8')).hexdigest()
# 处理 base64 编码问题
def handle_base64(hd_object):
    return str(base64.b64decode(hd_object))[2:-1]
# 解密图片链接
def parse(ig_hs, ct):
    count = 4
    contains = handle_md5(ct)
    ig_hs_copy = ig_hs
    p = handle_md5(contains[0:16])
    m = ig_hs[0:count]
    c = p + handle_md5(p + m)
    n = ig_hs[count:]
    l = handle_base64(n)
    k = []
    for h in range(256):
        k.append(h)
    b = []
    for h in range(256):
        b.append(ord(c[h % len(c)]))
    g = 0
    for h in range(256):
        g = (g + k[h] + b[h]) % 256
        tmp = k[h]
        k[h] = k[g]
        k[g] = tmp
    u = ''
    q = 0
    z = 0
    for h in range(len(l)):
        q = (q + 1) % 256
        z = (z + k[q]) % 256
        tmp = k[q]
        k[q] = k[z]
        k[z] = tmp
        u += chr(ord(l[h]) ^(k[(k[q] + k[g]) % 256]))
    u = u[26:]
    u = handle_base64(ig_hs_copy)
    return u
for i in range(1,10):
    url = 'http://jandan.net/ooxx/page-' + str(i) + '#comments'
    response = requests.get(url)
    doc = pq(response.text)
    links = doc('#wrapper #body #content #comments .commentlist .row .text .img-hash')
    print(links)
    arg = '5HTs9vFpTZjaGnG2M473PomLAGtI37M8'
    for link in links:
```

```
        l = link.text
        print(type(l))
        u = parse(l,arg)
        #print(u)
        u1 = 'http:' + u
        print(u1)
        r = requests.get(u1)
        with open('D:/image/' + md5(r.content).hexdigest() + '.jpg','wb') as f:
            f.write(r.content)
            f.close()
```

5.5 lxml 解析网页

lxml 是一个 HTML/XML 的解析器，主要的功能是如何解析和提取 HTML/XML 数据。lxml 和正则表达式一样，也是用 C 实现的，是一款高性能的 Python HTML/XML 解析器，可以利用之前学习的 XPath 语法，来快速定位特定元素以及节点信息。

安装 lxml 也非常简单，直接使用 pip 安装，代码为：

```
pip install lxml
```

5.5.1 使用 lxml

使用 lxml 爬取网页源代码数据也有 3 种方法，即 XPath 选择器、CSS 选择器和 Beautiful Soup 的 find()方法。与利用 Beautiful Soup 相比，lxml 还多了一种 XPath 选择器方法。

下面利用 lxml 来解析 HTML 代码：

```
from lxml import etree
html = '''
<html>
<head>
<meta name="content-type" content="text/html; charset=utf-8" />
    <title>友情链接查询 - 站长工具</title>
    <!-- uRjOAk8VLEPhjWhg3m9z4EjXJwc -->
<meta name="Keywords" content="友情链接查询" />
<meta name="Description" content="友情链接查询" />
</head>
<body>
<h1 class="heading">Top News</h1>
<p style="font-size: 200%">World News only on this page</p>
Ah, and here's some more text, by the way.
<p>...    and this is a parsed fragment ...</p>
<a href="http://www.cydf.org.cn/" rel="nofollow" target="_blank">青少年发展基金会</a>
<a href="http://www.4399.com/flash/32979.htm" target="_blank">洛克王国</a>
<a href="http://www.4399.com/flash/35538.htm" target="_blank">奥拉星</a>
<a href="http://game.3533.com/game/" target="_blank">手机游戏</a>
<a href="http://game.3533.com/tupian/" target="_blank">手机壁纸</a>
<a href="http://www.4399.com/" target="_blank">4399 小游戏</a>
```

```
<a href = "http://www.91wan.com/" target = "_blank"> 91wan 游戏</a>
</body>
</html>
'''
page = etree.HTML(html.lower().encode('utf-8'))
hrefs = page.xpath(u"//a")
for href in hrefs:
    print(href.attrib)
```

运行程序,输出如下:

```
{'href': 'http://www.cydf.org.cn/', 'rel': 'nofollow', 'target': '_blank'}
{'href': 'http://www.4399.com/flash/32979.htm', 'target': '_blank'}
{'href': 'http://www.4399.com/flash/35538.htm', 'target': '_blank'}
{'href': 'http://game.3533.com/game/', 'target': '_blank'}
{'href': 'http://game.3533.com/tupian/', 'target': '_blank'}
{'href': 'http://www.4399.com/', 'target': '_blank'}
{'href': 'http://www.91wan.com/', 'target': '_blank'}
```

提示:lxml 可以自动修正 HTML 代码。

5.5.2 文件读取

除了直接读取字符串,lxml 还支持从文件中读取内容。

新建一个 hello.HTML 文件:

```
<!-- hello.html -->
<div>
<ul>
<li class = "item-0"><a href = "link1.html"> first item </a></li>
<li class = "item-1"><a href = "link2.html"> second item </a></li>
<li class = "item-inactive"><a href = "link3.html"><span class = "bold"> third item </span>
</a></li>
<li class = "item-1"><a href = "link4.html"> fourth item </a></li>
<li class = "item-0"><a href = "link5.html"> fifth item </a></li>
</ul>
</div>
```

读取 HTML 文件中的代码为:

```
from lxml import etree
# 读取外部文件 hello.html
html = etree.parse('./hello.html')
result = etree.tostring(html, pretty_print = True)
print(result)
```

运行程序,输出如下:

```
<html><body>
<div>
<ul>
<li class = "item-0"><a href = "link1.html"> first item </a></li>
<li class = "item-1"><a href = "link2.html"> second item </a></li>
<li class = "item-inactive"><a href = "link3.html"> third item </a></li>
<li class = "item-1"><a href = "link4.html"> fourth item </a></li>
<li class = "item-0"><a href = "link5.html"> fifth item </a></li>
```

```
</ul>
</div>
</body></html>
```

5.5.3 XPath 使用

XPath 是一门在 XML 文档中查找信息的语言。XPath 使用路径表达式来选取 XML 文档中的节点或节点数，也可以用在 HTML 获取数据中。

XPath 使用路径表达式可以在网页源代码中选取节点，它是沿着路径来选取的，如表 5-3 所示。

表 5-3 XPath 路径表达式及其描述

表 达 式	描 述
nodename	选取此节点的所有子节点
/	从根节点选取
//	从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置
.	选取当前节点
..	选取当前节点的父节点
@	选取属性

表 5-4 列出了 XPath 的一些路径表达式及结果。

表 5-4 路径表达式及结果

路径表达式	结 果
bookstore	选取 bookstore 元素的所有子节点
/bookstore	选择根元素 bookstore 解释：假如路径起始于正斜杠(/)，此路径始终代表到某元素的绝对路径
bokstore/book	选取属于 bookstore 子元素的所有 book 元素
//book	选取所有 book 子元素，无论它们在文档中什么位置
bookstore//book	选择属于 bookstore 元素后代的所有 book 元素，无论它们位于 bookstore 下的什么内容
//@lang	选取名为 lang 的所有属性

下面代码为 XPath 实例测试：

```
#使用 lxml 的 etree 库
from lxml import etree
text = '''
<div>
<ul class = 'page'>
<li class = "item-0"><a href = "link1.html"> first item </a></li>
<li class = "item-1"><a href = "link2.html"> second item </a></li>
<li class = "item-inactive"><a href = "link3.html"> third item </a></li>
<li class = "item-1"><a href = "link4.html"> fourth item </a></li>
<li class = "item-0"><a href = "link5.html"> fifth item </a> # 注意,此处缺少一个</li>闭合
                                                          #标签
</ul>
</div>
```

```
'''
#利用 etree.HTML,将字符串解析为 HTML 文档
html = etree.HTML(text)
#获取所有的<li>标签
result = html.xpath('//li')
# 获取<li>标签的所有 class 属性
result = html.xpath('//li/@class')
#获取<li>标签下 hre 为 link1.html 的<a>标签,/用于获取直接子节点,//用于获取子孙节点
result = html.xpath('//li/a[@href="link1.html"]')
#获取<li>的父节点使用..:获取 li 标签的父节点,然后获取父节点的 class 属性
result = html.xpath('//li/../@class')
# 获取<li>标签下的<a>标签里的所有 class
result = html.xpath('//li/a//@class')
#获取最后一个<li>的<a>的属性 href
result = html.xpath('//li[last()]/a/@href')
# 获取倒数第二个元素的内容
result = html.xpath('//li[last()-1]/a/text()')
# 获取 class 值为 item-inactive 的标签名
result = html.xpath('//*[@class="item-inactive"]')
print(result)
```

运行程序,输出如下:

```
[<Element li at 0x2d47db61308>]
```

表 5-5 总结了各种 HTML 解析器的优缺点。

表 5-5 HTML 解析器的优缺点

HTML 解析器	运 行 速 度	易 用 性	提取数据方式
正则表达式	快	较难	正则表达式
BeautifulSoup	快(使用 lxml 解析)	简单	Find 方法 CSS 选择器
lxml	快	简单	XPath CSS 选择器

如果你面对的是复杂的网页代码,那么正则表达式的书写可能花费较长时间,这时选择 BeautifulSoup 和 lxml 比较简单。由于 BeautifulSoup 已经支持 lxml 解析,因此速度和 lxml 差不多,使用者可以根据熟悉程度进行选择。因为学习新的方法也需要时间,所以熟悉 XPath 的读者可以选择 lxml。假如是初学者,就需要快速掌握提取网页中的数据,推荐使用 BeautifulSoup 的 find()方法。

5.5.4 爬取 LOL 百度贴吧图片

下面一个实例演示 lxml 解析网页:爬取 LOL 百度贴吧的图片,实现代码为:

```
from lxml import etree
from urllib import request,error,parse
class Spider:
  def __init__(self):
   self.tiebaName = 'lol'
   self.beginPage = int(input('请输入开始页:'))
```

```
        self.endPage = int(input('请输入结束页:'))
        self.url = 'http://tieba.baidu.com/f'
        self.header = {"User-Agent": "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1 Trident/
    5.0;"}
        self.userName = 1 ## 图片编号
      def tiebaSpider(self):
        for page in range(self.beginPage, self.endPage + 1):
            pn = (page - 1) * 50
            data = {'pn':pn, 'kw':self.tiebaName}
            myUrl = self.url + '?' + parse.urlencode(data)
            # 调用页面处理函数 load_Page,获取页面所有帖子链接
            links = self.loadPage(myUrl)
      # 读取页面内容
      def loadPage(self,url):
        req = request.Request(url, headers = self.header)
        resp = request.urlopen(req).read()
        # 将 resp 解析为 html 文档
        html = etree.HTML(resp)
        # 抓取当前页面的所有帖子的 url 的后半部分,也就是帖子编号
        url_list = html.xpath('//div[@class = "threadlist_lz clearfix"]/div/a/@href')
        # url_list 类型为 etreeElementString 列表
        # 遍历列表,并且合并成一个帖子地址,调用图片处理函数 loadImage
        for row in url_list:
            row = "http://tieba.baidu.com" + row
            self.loadImages(row)
      # 下载图片
      def loadImages(self, url):
        req = request.Request(url, headers = self.header)
        resp = request.urlopen(req).read()
        # 将 resp 解析为 html 文档
        html = etree.HTML(resp)
        # 获取这个帖子里所有图片的 src 路径
        image_urls = html.xpath('//img[@class = "BDE_Image"]/@src')
        # 依次取出图片路径,下载保存
        for image in image_urls :
            self.writeImages(image)
            # 保存页面内容
      def writeImages(self, url):
        '''
    将 images 里的二进制内容存入 userName 文件中
        '''
        print(url)
        print("正在存储文件 %d ..." % self.userName)
        # 1. 打开文件,返回一个文件对象
        path = r'D:\example' + '\\' + str(self.userName) + '.png' #example 为自己新建存放获取图
                                                                  #像的文件夹
        file = open(path, 'wb')
        # 获取图片里的内容
        images = request.urlopen(url).read()
        # 调用文件对象 write() 方法,将 page_html 的内容写入文件中
        file.write(images)
        file.close()
        # 计数器自增 1
        self.userName += 1
```

```
#模拟 main 函数
if __name__ == "__main__":
 # 首先创建爬虫对象
 mySpider = Spider()
 # 调用爬虫对象的方法,开始工作
 mySpider.tiebaSpider()
```

运行程序,即可将 LOL 网页中的图片下载到新建的 example 文件夹中,共下载 261 张图片,如图 5-3 所示,输出如下:

```
请输入开始页:1
请输入结束页:1
https://imgsa. baidu. com/forum/w% 3D580/sign = 06f1b0a37ecf3bc7e800cde4e102babd/
cb9eb2fd5266d01668dcd286992bd40734fa3518.jpg
…
正在存储文件 260 ...
https://fc-feed.cdn.bcebos.com/0/pic/b509deeac5ba2fc8b259533dcb718c75.jpg
正在存储文件 261 ...
```

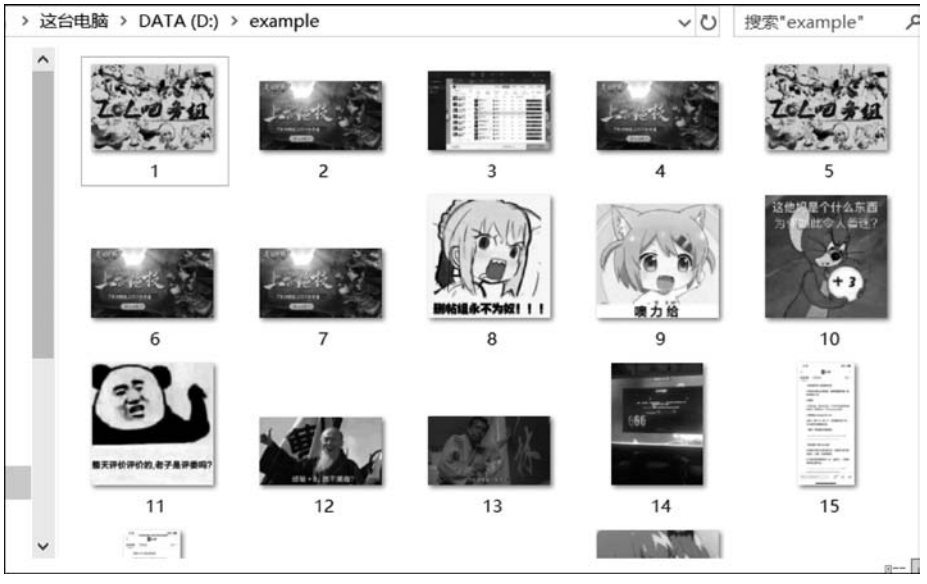

图 5-3 下载的图片

5.6 爬取二手房网站数据

本节的实践中仅获取了搜索结果的房源数据。

首先,需要分析一下要爬取郑州的二手房信息的网络结构,如图 5-4 所示。

由上可以看到网页一条条的房源信息,单击进去后就会发现房源的详细信息,如图 5-5 所示。

查看页面的源代码的效果如图 5-6 所示。

图 5-4　郑州二手房数据

图 5-5　房子详细信息

下面采用 Python3 中的 Requests、Beautiful Soup 模块来进行爬取页面，先由 Requests 模块进行请求：

```
# 网页的请求头
header = {
'user-agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/60.0.3112.113 Safari/537.36'
}
# url 链接
url = 'https://zhengzhou.anjuke.com/sale/'
response = requests.get(url, headers = header)
print(response.text)
```

```
<!-- <script type="text/javascript" src="//include.anjukestatic.com/ujs/chat.opener/ajbb6j0b5d/ab4bb780/00cb4b42957a.js
crossorigin></script> -->
<script type="text/javascript">J.site.init()</script>


<!--    <script type="text/javascript" src="https://s.anjuke.com/bb.js"></script>
 -->
<link rel="shortcut icon" href="https://pages.anjukestatic.com/usersite/site/img/global/1/favicon.ico" />
<link rel="icon" href="https://pages.anjukestatic.com/usersite/site/img/global/1/favicon.ico" type="image/ico" />

</head>
<body data-page="Ershou_Web_Property_List_SearchPage">
<div id="container">
    <div id="header">
        <!-- START: 顶通广告 -->
<div class="top-poster">
        </div>
<!-- END: 顶通广告 -->

<div class="top-nav  glbNavigation ">
    <div class="glbListNavigation1180 nav-content">
        <ul class="L_tabsnew">

    <li class="li_single first-child li_unselected">
        <a class="a_navnew" hidefocus="true" href="https://zhengzhou.anjuke.com/" _soj="navigation">首 页</a>
    </li>

    <li class="li_single li_itemsnew li_unselected">
        <a class="a_navnew" hidefocus="true" href="https://zz.fang.anjuke.com/" _soj="navigation">新 房</a>
    </li>

    <li class="li_single li_itemsnew li_selected">
        <a class="a_navnew a_curr" hidefocus="true" href="https://zhengzhou.anjuke.com/sale/" _soj="navigation">二手房</a>
```

图 5-6　房源的源代码

运行程序，得到这个网站的 HTML 代码如图 5-7 所示。

```
<div class="items"><span class="item-title">区域：</span><span class="elems-l"><span class="selected-item">全部</span><a
 href='https://zhengzhou.anjuke.com/sale/jinshui/' class='' title='金水二手房' >金水</a><a href='https://zhengzhou.anjuk
e.com/sale/zhengdongxinqu/' class='' title='郑东新区二手房' >郑东新区</a><a href='https://zhengzhou.anjuke.com/sale/zhon
gyuanb/' class='' title='中原二手房' >中原</a><a href='https://zhengzhou.anjuke.com/sale/guanchenga/' class='' title='管
城二手房' >管城</a><a href='https://zhengzhou.anjuke.com/sale/erqic/' class='' title='二七二手房' >二七</a><a href='http
s://zhengzhou.anjuke.com/sale/xinzhengshi/' class='' title='新郑二手房' >新郑</a><a href='https://zhengzhou.anjuke.com/s
ale/zhongmou/' class='' title='中牟二手房' >中牟</a><a href='https://zhengzhou.anjuke.com/sale/huiji/' class='' title='
惠济二手房' >惠济</a><a href='https://zhengzhou.anjuke.com/sale/hangkongganga/' class='' title='航空港二手房' >航空港</a
><a href='https://zhengzhou.anjuke.com/sale/gaoxinqub/' class='' title='高新区二手房' >高新区</a><a href='https://zhengz
hou.anjuke.com/sale/jingkaiz/' class='' title='经开二手房' >经开</a><a href='https://zhengzhou.anjuke.com/sale/zzxingyan
gshi/' class='' title='荥阳二手房' >荥阳</a><a href='https://zhengzhou.anjuke.com/sale/zzshangjiequ/' class='' title='上
街二手房' >上街</a><a href='https://zhengzhou.anjuke.com/sale/xinmishi/' class='' title='新密二手房' >新密</a><a href='h
ttps://zhengzhou.anjuke.com/sale/gongyishi/' class='' title='巩义二手房' >巩义</a><a href='https://zhengzhou.anjuke.com/
sale/dengfengshi/' class='' title='登封二手房' >登封</a><a href='https://zhengzhou.anjuke.com/sale/zhengzhouzhoubian/' c
lass='' title='郑州周边二手房' >郑州周边</a></span></div><!— 区域 end—>

<!—  户型 begin—>
<!— 售价 begin—>
<div class="items "><span class="item-title">售价：</span><span class="elems-l"><span class="selected-item">全部</span><
a href='https://zhengzhou.anjuke.com/sale/m5107/' class=' ' rel="nofollow">50万以下</a><a href='https://zhengzhou.anjuke
.com/sale/m186/' class=' ' rel="nofollow">50-60万</a><a href='https://zhengzhou.anjuke.com/sale/m187/' class=' ' rel="no
follow">60-80万</a><a href='https://zhengzhou.anjuke.com/sale/m188/' class=' ' rel="nofollow">80-100万</a><a href='https
://zhengzhou.anjuke.com/sale/m189/' class=' ' rel="nofollow">100-120万</a><a href='https://zhengzhou.anjuke.com/sale/m19
0/' class=' ' rel="nofollow">120-150万</a><a href='https://zhengzhou.anjuke.com/sale/m191/' class=' ' rel="nofollow">150
-200万</a><a href='https://zhengzhou.anjuke.com/sale/m5108/' class=' ' rel="nofollow">200-300万</a><a href='https://zhen
gzhou.anjuke.com/sale/m5109/' class=' ' rel="nofollow">300-500万</a><a href='https://zhengzhou.anjuke.com/sale/m5110/' c
lass=' ' rel="nofollow">500万以上</a>
        <div class="pricecond">
        <form action="https://zhengzhou.anjuke.com/sale/" id="pr_form_apf_id_11" onsubmit="return priceCondition.submit(
0)">
```

图 5-7　HTML 代码

通过分析可以得到每个房源都在 class＝"list-item"的 li 标签中，那么就可以根据 BeautifulSoup 包进行提取：

```
from bs4 import BeautifulSoup
# 通过 BeautifulSoup 解析出每个房源详细列表并进行打印
soup = BeautifulSoup(response.text, 'html.parser')
result_li = soup.find_all('li', {'class': 'list-item'})
for i in result_li:
    print(i)
```

运行程序，效果如图 5-8 所示。

```
<i class="house-icon house-icon-anxuan" style="font-weight: normal;">安选</i>
<i class="house-icon house-icon-default border-line">
                        真实在售                        </i>
</div>
<div class="details-item">
<span>3室2厅</span><em class="spe-lines">|</em><span>89m²</span><em class="spe-lines">|</em><spa
n>高层(共30层)</span><em class="spe-lines">|</em><span>2015年建造</span>
</div>
<div class="details-item">
<span class="comm-address" title="绿都紫荆华庭    管城-紫荆山路-南三环">
                    绿都紫荆华庭
                    管城-紫荆山路-南三环                </span>
</div>
<div class="broker-item">
<span class="broker-img-wrap"><img alt="" src="http://pic1.ajkimg.com/display/anjuke/2225f0bdfc0
18640d88e18c30eb5747b/254x337x0x20/100x133.jpg"/></span>
<span class="broker-name broker-text">杨龙</span>
<span class="broker-text"></span>
                    //是否安选经纪人 显示icon
                    <span><img class="broker-ax-img" src="//pages.anjukestatic.com/usersite/touc
h/img/broker/esf_list_icon_wyjy.png"/></span> </div>
<div class="tags-bottom">
</div>
</div>
<div class="pro-price">
<span class="price-det"><strong>146</strong>万</span><span class="unit-price">16289元/m²</span>
</div>
</li>
```

图 5-8　每个房源详细列表

进一步减少代码量，继续提取：

```
# 通过 BeautifulSoup 解析出每个房源详细列表并进行打印
soup = BeautifulSoup(response.text, 'html.parser')
result_li = soup.find_all('li', {'class': 'list-item'})
# 进行循环遍历其中的房源详细列表
for i in result_li:
    # 由于 BeautifulSoup 传入的必须为字符串，所以进行转换
    page_url = str(i)
    soup = BeautifulSoup(page_url, 'html.parser')
    # 由于通过 class 解析的为一个列表，所以只需要第一个参数
    result_href = soup.find_all('a', {'class': 'houseListTitle'})[0]
    print(result_href.attrs['href'])
```

运行程序，效果如图 5-9 所示。

```
https://zhengzhou.anjuke.com/prop/view/A1629122142?from=filter-saleMetro-salesxq&spread=commsear
ch_p&position=8&kwtype=filter&now_time=1559311246
https://zhengzhou.anjuke.com/prop/view/A1689199934?from=filter&spread=commsearch_p&position=9&kw
type=filter&now_time=1559311246
https://zhengzhou.anjuke.com/prop/view/A1712263372?from=filter&spread=commsearch_p&position=10&k
wtype=filter&now_time=1559311246
https://zhengzhou.anjuke.com/prop/view/A1705651458?from=filter&spread=commsearch_p&position=11&k
wtype=filter&now_time=1559311246
https://zhengzhou.anjuke.com/prop/view/A1688583833?from=filter&spread=commsearch_p&position=12&k
wtype=filter&now_time=1559311246
https://zhengzhou.anjuke.com/prop/view/A1712256449?from=filter&spread=commsearch_p&position=13&k
wtype=filter&now_time=1559311246
https://zhengzhou.anjuke.com/prop/view/A1659057218?from=filter&spread=commsearch_p&position=14&k
wtype=filter&now_time=1559311246
https://zhengzhou.anjuke.com/prop/view/A1694914990?from=filter&spread=commsearch_p&position=15&k
wtype=filter&now_time=1559311246
https://zhengzhou.anjuke.com/prop/view/A1705035175?from=filter-saleMetro-salesxq&spread=commsear
ch_p&position=16&kwtype=filter&now_time=1559311246
https://zhengzhou.anjuke.com/prop/view/A1706723205?from=filter&spread=commsearch_p&position=17&k
wtype=filter&now_time=1559311246
https://zhengzhou.anjuke.com/prop/view/A1693890097?from=filter&spread=commsearch_p&position=18&k
wtype=filter&now_time=1559311246
```

图 5-9　打印所有 url

下一步即进入页面开始分析详细页面了，所以，就需要先分析该页面是否有下一页，在页面右击选择“审查元素”命令，效果如图 5-10 所示。

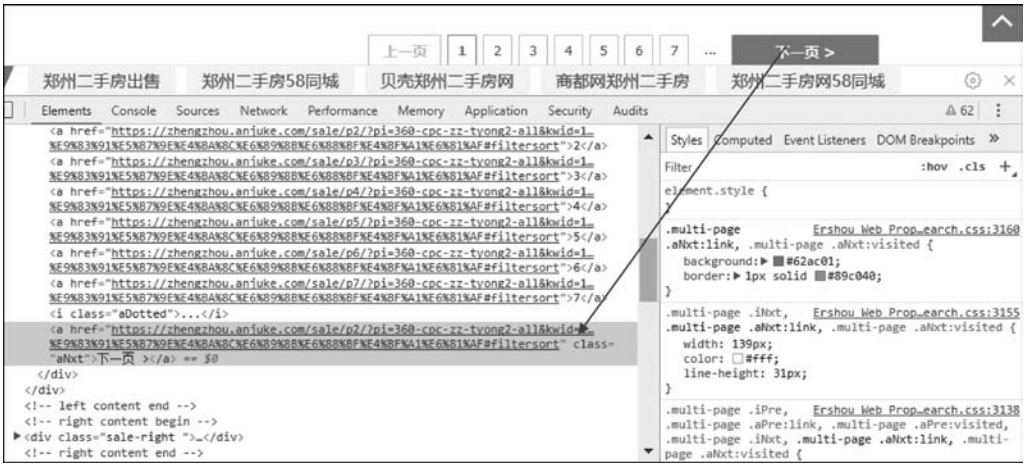

图 5-10　分析网页

其利用 Python 代码爬取的方法为：

```
# 进行下一页的爬取
result_next_page = soup.find_all('a', {'class': 'aNxt'})
if len(result_next_page) != 0:
    print(result_next_page[0].attrs['href'])
else:
    print('没有下一页了')
```

运行程序，效果如图 5-11 所示。

```
https://zhengzhou.anjuke.com/prop/view/A1696141061?from=filter&spread=commsearch_p&pos
ition=51&kwtype=filter&now_time=1559311676
https://zhengzhou.anjuke.com/prop/view/A1690660644?from=filter&spread=commsearch_p&pos
ition=52&kwtype=filter&now_time=1559311676
https://zhengzhou.anjuke.com/prop/view/A1669304375?from=filter&spread=commsearch_p&pos
ition=53&kwtype=filter&now_time=1559311676
https://zhengzhou.anjuke.com/prop/view/A1713389422?from=filter&spread=commsearch_p&pos
ition=54&kwtype=filter&now_time=1559311676
https://zhengzhou.anjuke.com/prop/view/A1711541704?from=filter&spread=commsearch_p&pos
ition=55&kwtype=filter&now_time=1559311676
https://zhengzhou.anjuke.com/prop/view/A1703937211?from=filter&spread=commsearch_p&pos
ition=56&kwtype=filter&now_time=1559311676
https://zhengzhou.anjuke.com/prop/view/A1715153195?from=filter&spread=commsearch_p&pos
ition=57&kwtype=filter&now_time=1559311676
https://zhengzhou.anjuke.com/prop/view/A1622373486?from=filter&spread=commsearch_p&pos
ition=58&kwtype=filter&now_time=1559311676
https://zhengzhou.anjuke.com/prop/view/A1709883432?from=filter&spread=commsearch_p&pos
ition=59&kwtype=filter&now_time=1559311676
https://zhengzhou.anjuke.com/prop/view/A1687459298?from=filter&spread=commsearch_p&pos
ition=60&kwtype=filter&now_time=1559311676
没有下一页了
```

图 5-11　爬取下一页信息

如果存在下一页的时候，网页中就有一个 a 标签，如果没有，就会成为 i 标签了，因此，就能完善一下，将以上这些封装为一个函数：

```
import requests
from bs4 import BeautifulSoup
# 网页的请求头
header = {
    'user-agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/60.0.3112.113 Safari/537.36'
}
def get_page(url):
    response = requests.get(url, headers=header)
    # 通过 BeautifulSoup 解析出每个房源详细列表并进行打印
    soup = BeautifulSoup(response.text, 'html.parser')
    result_li = soup.find_all('li', {'class': 'list-item'})
    # 进行下一页的爬取
    result_next_page = soup.find_all('a', {'class': 'aNxt'})
    if len(result_next_page) != 0:
        # 函数进行递归
        get_page(result_next_page[0].attrs['href'])
    else:
        print('没有下一页了')
    # 进行循环遍历其中的房源详细列表
    for i in result_li:
        # 由于 BeautifulSoup 传入的必须为字符串,所以进行转换
        page_url = str(i)
        soup = BeautifulSoup(page_url, 'html.parser')
        # 由于通过 class 解析的为一个列表,所以只需要第一个参数
        result_href = soup.find_all('a', {'class': 'houseListTitle'})[0]
        # 先不做分析,等一会进行详细页面函数完成后进行调用
        print(result_href.attrs['href'])
if __name__ == '__main__':
    # url 链接
    url = 'https://zhengzhou.anjuke.com/sale/'
    # 页面爬取函数调用
    get_page(url)
```

具体实现详细页面的爬取的完整代码为：

```
import requests
from bs4 import BeautifulSoup
# 网页的请求头
header = {
    'user-agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/60.0.3112.113 Safari/537.36'
}
def get_page(url):
    response = requests.get(url, headers=header)
    # 通过 BeautifulSoup 解析出每个房源详细列表并进行打印
    soup_idex = BeautifulSoup(response.text, 'html.parser')
    result_li = soup_idex.find_all('li', {'class': 'list-item'})
    # 进行循环遍历其中的房源详细列表
    for i in result_li:
        # 由于 BeautifulSoup 传入的必须为字符串,所以进行转换
        page_url = str(i)
        soup = BeautifulSoup(page_url, 'html.parser')
        # 由于通过 class 解析的为一个列表,所以只需要第一个参数
```

```
            result_href = soup.find_all('a', {'class': 'houseListTitle'})[0]
            # 详细页面的函数调用
            get_page_detail(result_href.attrs['href'])
        # 进行下一页的爬取
        result_next_page = soup_idex.find_all('a', {'class': 'aNxt'})
        if len(result_next_page) != 0:
            # 函数进行递归
            get_page(result_next_page[0].attrs['href'])
        else:
            print('没有下一页了')
    # 进行字符串中空格,换行,Tab键的替换及字符串两边的空格删除
    def my_strip(s):
        return str(s).replace(" ", "").replace("\n", "").replace("\t", "").strip()
    # 由于频繁进行BeautifulSoup的使用,因此要封装一下
    def my_Beautifulsoup(response):
        return BeautifulSoup(str(response), 'html.parser')
    # 详细页面的爬取
    def get_page_detail(url):
        response = requests.get(url, headers = header)
        if response.status_code == 200:
            soup = BeautifulSoup(response.text, 'html.parser')
            # 标题
            result_title = soup.find_all('h3', {'class': 'long-title'})[0]
            result_price = soup.find_all('span', {'class': 'light info-tag'})[0]
            result_house_1 = soup.find_all('div', {'class': 'first-col detail-col'})
            result_house_2 = soup.find_all('div', {'class': 'second-col detail-col'})
            result_house_3 = soup.find_all('div', {'class': 'third-col detail-col'})
            soup_1 = my_Beautifulsoup(result_house_1)
            soup_2 = my_Beautifulsoup(result_house_2)
            soup_3 = my_Beautifulsoup(result_house_3)
            result_house_tar_1 = soup_1.find_all('dd')
            result_house_tar_2 = soup_2.find_all('dd')
            result_house_tar_3 = soup_3.find_all('dd')
            '''
    文博公寓,省实验中学,首付只需70万,大三房,诚心卖,价可谈270万
    宇泰文博公寓金水-花园路-文博东路4号2010年普通住宅
            3室2厅2卫140平方米南北中层(共32层)
    精装修19285元/m² 81.00万
            '''
            print(my_strip(result_title.text), my_strip(result_price.text))
            print(my_strip(result_house_tar_1[0].text),
                  my_strip(my_Beautifulsoup(result_house_tar_1[1]).find_all('p')[0].text),
                  my_strip(result_house_tar_1[2].text), my_strip(result_house_tar_1[3].text))
            print(my_strip(result_house_tar_2[0].text), my_strip(result_house_tar_2[1].text),
                  my_strip(result_house_tar_2[2].text), my_strip(result_house_tar_2[3].text))
            print(my_strip(result_house_tar_3[0].text), my_strip(result_house_tar_3[1].text),
                  my_strip(result_house_tar_3[2].text))
    if __name__ == '__main__':
        # url链接
        url = 'https://zhengzhou.anjuke.com/sale/'
        # 页面爬取函数调用
        get_page(url)
```

5.7 习题

1. lxml 是一个 HTML/XML 的解析器，主要的功能是_____和_____数据。
2. 简述正则表达式的定义。
3. 正则表达式的大致匹配过程是怎样的？
4. Beautiful Soup 的优势有哪些？
5. 利用 PyQuery 爬取头条部分指定网页内容。