**第二章 python爬虫匹配之re&xpath&bs4**

**本章所讲内容：**

**2.1 re正则匹配**

**2.2 lxml xpath 匹配**

**2.3 beautifulsoup 匹配**

**实战：爬取内涵吧脑筋急转弯**

**实战：爬取猫眼电影排行榜并使用csv保存到本地的csv中。**

**实战：利用xpath 匹配爬取链家买房信息，存储到mysql中**

**实战：爬取百思不得姐小视频，存储到当前界面**

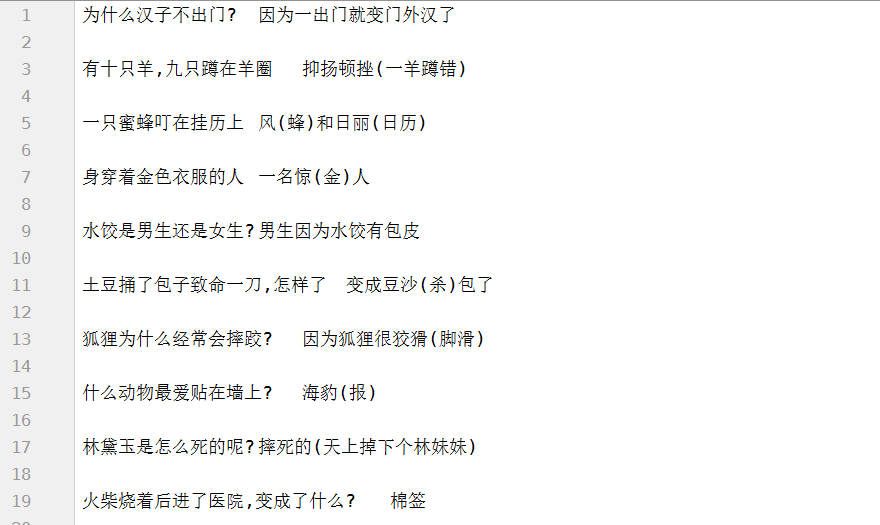
**2.1 re 正则匹配**

**正则表达式是处理字符串的强大工具，它有自己特定的语法结构，有了它，实现字符串的检索、替换、匹配验证都不在话下。当然，对于爬虫来说，有了它，从HTML里提取想要的信息就非常方便了。**

**因为第一阶段中已经交给大家正则的语法了所以我们直接上列子**

**实战：爬取内涵吧脑筋急转弯****import requests  
import re  
from fake\_useragent import UserAgent  
*# https://www.neihan-8.com/njjzw/index.html*base\_url = 'https://www.neihan8.com/njjzw/'  
  
headers = {  
 "User-Agent":UserAgent().random  
}  
def load\_page(url):  
 try:  
 req = requests.get(url,headers=headers)  
 if req.status\_code == 200:  
 *# print(req.text.decode())* return req.content.decode()  
 except:  
 return None  
def parse\_page(html):  
 data\_list = re.findall(r'<div class="text-.\*?title="(.\*?)".\*?<div class="desc">(.\*?)</div>',html,re.S)  
 return data\_list  
def write\_page(data\_list):  
 with open('33333.txt','a',encoding='utf-8')as f:  
 for data in data\_list:  
 f.write(data[0].strip()+'\t'+data[1].strip()+'\n')  
 *# f.write(data[1].strip()+'\n')* f.write('\n')  
if \_\_name\_\_ == '\_\_main\_\_':  
 for i in range(1,10):  
 if i == 1:  
 url = base\_url  
 else:  
 url = base\_url+'index\_'+str(i)+'.html'  
 print(url)  
 html = load\_page(url)  
 data\_list = parse\_page(html)  
 write\_page(data\_list)**

**运行结果如下：**



**实战：爬取猫眼电影排行榜并使用csv保存到本地的csv中。**

**import requests  
from fake\_useragent import UserAgent  
import re,csv  
headers = {  
 'user-agent':UserAgent().random  
}  
def get\_one\_page(url):  
 response = requests.get(url,headers=headers)  
 if response.status\_code == 200:  
 return response.text  
 return None  
def get\_data\_html(html):  
 movie\_data\_list = re.findall('<div class="movie-item-info">.\*?title="(.\*?)".\*?<p class="star">(.\*?)</p>.\*?releasetime">(.\*?)</p>',html,re.S)  
 return movie\_data\_list  
def write\_page(datas):  
 with open('1.csv','a',newline='') as f:  
 writer = csv.writer(f)  
 writer.writerow(["电影名称","主演","上映时间"])  
 for data in datas:  
 *# with open("1.csv","a",newline="") as f:  
 # 创建写入对象* writer = csv.writer(f)  
 *# L = list(r\_tuple)* L = [data[0].strip(),data[1].strip(),data[2].strip()]  
 *# ["霸王别姬","张国荣","1994-01-01"]* writer.writerow(L)  
def main(offset):  
 url = 'https://maoyan.com/board/4?offset={}'.format(offset)  
 html = get\_one\_page(url)  
 data = get\_data\_html(html)  
 write\_page(data)  
if \_\_name\_\_ == '\_\_main\_\_':  
 for i in range(0,10):  
 main(str(i\*10))**

**运行结果如下：**



**这两个例子分别是第一节课的实战，放到这里讲也是很合适的。**

## 2.2 lxml xpath 匹配

**xpath 全称XML Path Language，即XML路径语言是一门。在XML文档中查找信息的语言。（遵循xml，html的树形结构）。xpath 可用来在 XML，html 文档中对元素和属性进行遍历。**

**Xpath安装：pip install lxml**

**XPath的选择功能十分强大，它提供了非常简明的路径选择表达式，另外，它还提供了超过100个内建函数，用于字符串、数值、时间的匹配以及节点、序列的处理等，几乎所有我们想要定位的节点，都可以用XPath来选择 例子：/html/body/div[6]/ul /html/body/div[6]/ul/li[1]/a**

### 2.2.1 xpath语法

**1、选取节点**

**XPath 使用路径表达式在 XML 文档中选取节点。节点是通过沿着路径或者 step 来选取的。**

**列出了最有用的路径表达式.**

| **表达式** | **描述** |
| --- | --- |
| **nodename** | **选取此节点的所有子节点** |
| **/** | **从当前节点选取直接子节点** |
| **//** | **从当前节点选取子孙节点** |
| **.** | **选取当前节点** |
| **..** | **选取当前节点的父节点** |
| **@** | **选取属性** |

**2、在下面的表格中，我们已列出了一些路径表达式以及表达式的结果：**

|  |  |
| --- | --- |
| **路径表达式** | **结果** |
| **bookstore** | **选取 bookstore 元素的所有子节点。** |
| **/bookstore** | **选取根元素 bookstore。**  **注释：假如路径起始于正斜杠( / )，则此路径始终代表到某元素的绝对路径！** |
| **bookstore/book** | **选取属于 bookstore 的子元素的所有 book 元素。** |
| **//book** | **选取所有 book 子元素，而不管它们在文档中的位置。** |
| **bookstore//book** | **选择属于 bookstore 元素的后代的所有 book 元素，而不管它们位于 bookstore 之下的什么位置。** |
| **//@lang** | **选取名为 lang 的所有属性。** |

**3、选取未知节点（re）**

|  |  |
| --- | --- |
| **通配符** | **描述** |
| **\*** | **匹配任何元素节点。** |
| **@\*** | **匹配任何属性节点。** |
| **node()** | **匹配任何类型的节点。** |

**简单实例说明**

|  |  |
| --- | --- |
| **路径表达式** | **结果** |
| **/bookstore/book[1]** | **选取属于 bookstore 子元素的第一个 book 元素。** |
| **/bookstore/book[last()]** | **选取属于 bookstore 子元素的最后一个 book 元素。** |
| **/bookstore/book[last()-1]** | **选取属于 bookstore 子元素的倒数第二个 book 元素。** |
| **/bookstore/book[position()<3]** | **选取最前面的两个属于 bookstore 元素的子元素的 book 元素。** |
| **//title[@lang]** | **选取所有拥有名为 lang 的属性的 title 元素。** |
| **//title[@lang='eng']** | **选取所有 title 元素，且这些元素拥有值为 eng 的 lang 属性。** |
| **/bookstore/book[price>35.00]** | **选取 bookstore 元素的所有 book 元素，且其中的 price 元素的值须大于 35.00。** |
| **/bookstore/book[price>35.00]/title** | **选取 bookstore 元素中的 book 元素的所有 title 元素，且其中的 price 元素的值须大于 35.00。** |

**简单实例说明**

|  |  |
| --- | --- |
| **路径表达式** | **结果** |
| **/bookstore/\*** | **选取 bookstore 元素的所有子元素。** |
| **//\*** | **选取文档中的所有元素。** |
| **//title[@\*]** | **选取所有带有属性的 title 元素。** |

**4、选取若干路径**

|  |  |
| --- | --- |
| **路径表达式** | **结果** |
| **//book/title | //book/price** | **选取 book 元素的所有 title 和 price 元素。** |
| **//title | //price** | **选取文档中的所有 title 和 price 元素。** |
| **/bookstore/book/title | //price** | **选取属于 bookstore 元素的 book 元素的所有 title 元素，以及文档中所有的 price 元素。** |

**示例如下：**

**from lxml import etree  
html = """<div class="wrapper">  
 <i class="iconfont icon-back" id="back">1</i>  
 <a href="/" id="channel">新浪社会</a>  
 <ul id="nav">  
 <li><a href="http://domestic.firefox.sina.com/" title="国内">国内</a></li>  
 <li><a href="http://world.firefox.sina.com/" title="国际">国际</a></li>  
 <li><a href="http://mil.firefox.sina.com/" title="军事">军事</a></li>  
 <li><a href="http://photo.firefox.sina.com/" title="图片">图片</a></li>  
 <li><a href="http://society.firefox.sina.com/" title="社会">社会</a></li>  
 <li><a href="http://ent.firefox.sina.com/" title="娱乐">娱乐</a></li>  
 <li><a href="http://tech.firefox.sina.com/" title="科技">科技</a></li>  
 <li><a href="http://sports.firefox.sina.com/" title="体育">体育</a></li>  
 <li><a href="http://finance.firefox.sina.com/" title="财经">财经</a></li>  
 <li><a href="http://auto.firefox.sina.com/" title="汽车">汽车</a></li>  
 </ul>  
 <i class="iconfont icon-liebiao" id="menu">2</i>  
</div>"""  
*# 构造解析对象*parseHtml = etree.HTML(html)  
*# 利用解析对象调用xpath匹配*r1 = parseHtml.xpath('//a/@href')  
*#print(r1)  
# 获取 /*r2 = parseHtml.xpath('//a[@id="channel"]/@href')  
*#print(r2)  
# 获取非 /*r3 = parseHtml.xpath('//ul[@id="nav"]//a/@href')  
*#print(r3)  
# 获取所有 a 节点的文本内容*r4 = parseHtml.xpath('//a/text()')  
*#print(r4)  
# 获取 图片、军事 ...*r5 = parseHtml.xpath('//ul[@id="nav"]//a')  
for i in r5:  
 print(i.text)  
r6 = parseHtml.xpath('//i[@id="back"]|//i[@id="menu"]')  
for i in r6:  
 print(i)  
 print(i.text)  
 print(i.tag)*#查看标签* print(i.attrib)*#查看文本信息***

**实战：利用xpath爬取链家买房信息，存储到mysql中**

**import requests  
from lxml import etree  
from fake\_useragent import UserAgent  
import pymysql,warnings  
  
*#创建数据库，生成表结构*db = pymysql.connect("localhost","root","123456",charset="utf8")  
cursor = db.cursor()  
create\_db = "create database if not exists Lianjiadb character set utf8"  
use\_db = "use Lianjiadb"  
create\_tab = "create table if not exists housePrice(id int primary key auto\_increment,housename varchar(50),totalprice varchar(50))charset=utf8"  
warnings.filterwarnings("ignore")*#由于已经有数据库了所以会有警告信息在这里让程序性忽略*cursor.execute(create\_db)  
cursor.execute(use\_db)  
cursor.execute(create\_tab)  
  
headers = {  
 'User-Agent':UserAgent().random  
}  
  
base\_url = 'https://bj.lianjia.com/ershoufang/pg{}/'  
*#响应HTML交给parse\_page解析*def load\_page(url):  
 try:  
 res = requests.get(url,headers=headers)  
 if res.status\_code == 200:  
 print('页面请求完毕')  
 return res.text  
 except:  
 print('网络访问错误')  
*#解析HTML拿到想要的数据*def parse\_page(html):  
 xpath\_content = etree.HTML(html)  
 xpath\_datas = xpath\_content.xpath("//\*[@class='info clear']")  
 for data in xpath\_datas:  
 title = data.xpath('./div[1]/a/text()')  
 price = data.xpath('./div[4]//div[@class="totalPrice"]/span/text()')  
 *# print(place)* new\_data = zip(title,price)  
 for tup\_data in new\_data:  
 insert\_mysql(tup\_data)  
 print('数据写入完毕')  
*#插入到指定的数据库中*def insert\_mysql(data):  
 insert\_data = "insert into housePrice(housename,totalprice) values(%s,%s)"  
 if data:  
 cursor.execute(insert\_data,list(data))  
 db.commit()  
def main():  
 for i in range(1,10):  
 url = base\_url.format(str(i))  
 html = load\_page(url)  
 parse\_page(html)  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()**

**运行效果如下：**



## 2.3 Beautiful Soup

**什么是Beautiful Soup？官方解释如下:**

**Beautiful Soup提供一些简单的、Python式的函数来处理导航、搜索、修改分析树等功能。它是一个工具箱，通过解析文档为用户提供需要抓取的数据，因为简单，所以不需要多少代码就可以写出一个完整的应用程序。**

**Beautiful Soup自动将输入文档转换为Unicode编码，输出文档转换为UTF-8编码。你不需要考虑编码方式，除非文档没有指定一个编码方式，这时你仅仅需要说明一下原始编码方式就可以了。**

**Beautiful Soup已成为和lxml、html6lib一样出色的Python解释器，为用户灵活地提供不同的解析策略或强劲的速度。**

### 2.3.1 安装与使用

**1、安装pip install bs4**

**2、解析器的分类**

**Beautiful Soup在解析时实际上依赖解析器，它除了支持Python标准库中的HTML解析器外，还支持一些第三方解析器（比如lxml）。以下列表中列出了Beautiful Soup支持的解析器。**

| **解析器** | **使用方法** | **优势** | **劣势** |
| --- | --- | --- | --- |
| **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** | **Python 2.7.3及Python 3.2.2之前的版本文档容错能力差** |
| **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, "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** |

**Beautiful Soup将复杂HTML文档转换成一个复杂的树形结构,每个节点都是Python对象,所有对**

**Beautiful Soup基本用法  
text = '''  
<html>  
<head>  
 <meta = charset='UTF-8' >  
 <title id =1 href = 'http://example.com/elsie' class = 'title'>Test</title>  
</head>  
<body>  
 <div class = 'ok'>  
 <div class = 'nice'>  
 <p class = 'p'>  
 Hello World  
 </p>  
 </div>  
 </div>  
</body>   
</html>  
'''  
from bs4 import BeautifulSoup  
soup = BeautifulSoup(text, 'lxml')  
*#这个方法可以把要解析的字符串以标准的缩进格式输出，自动更正html*print(soup.prettify())  
*#获取title标签*print(soup.title)  
*#获取标签内容*print(soup.title.string)  
*#获取标签的名字*print(soup.title.name)  
*#获取标签的内容*print(soup.title.text)  
*#获取标签的属性*print(soup.title.attrs)  
print(soup.title.attrs['name'])  
*#指定标签的某个属性*print(soup.title.get('class'))**

### 2.3.2 搜索文档树

**find()和find\_all()**

**find\_all()方法搜索当前tag的所有tag子节点,并判断是否符合过滤器的条件。**

**find()和find\_all()的区别就是，find直接返回元素的一个结果，find\_all返回元素列表**

**find\_all( name , attrs , recursive , text , \*\*kwargs )简介一下参数**

**name 参数可以查找所有名字为name的tag,字符串对象会被自动忽略掉；name参数可以传入字符串、正则表达式、列表、True、自定义的方法等但是各自代表的含义不一样。**

**字符串，在搜索方法中传入一个字符串参数,Beautiful Soup会查找与字符串完整匹配的内容。**

**print(soup.find('body'))  
print(soup.find\_all('body')**

**正则表达式，Beautiful Soup会通过正则表达式的match()来匹配内容列表，Beautiful Soup会将与列表中任一元素匹配的内容返回。**

**#获取所有p标签的内容**

**import re**

**print(soup.find\_all(re.compile("^p")))**

**如果匹配成功将会匹配所有的tag**

**如果一个指定名字的参数不是搜索内置的一些参数名,搜索时会把该参数当作指定名字tag的属性来**

**搜索;例如id=1**

**如果包含一个名字为 id 的参数,Beautiful Soup会搜索每个tag的”id”属性；**

**如果传入 href 参数,Beautiful Soup会搜索每个tag的”href”属性；**

**使用多个指定名字的参数可以同时过滤tag的多个属性；**

**对于class ，可以使用class\_来搜索**

**import re  
print(soup.find\_all(href = re.compile('elsie'),id=1))**

**#返回这个class=‘p’的标签内容。**

**print(soup.find\_all('p',class\_='p'))**

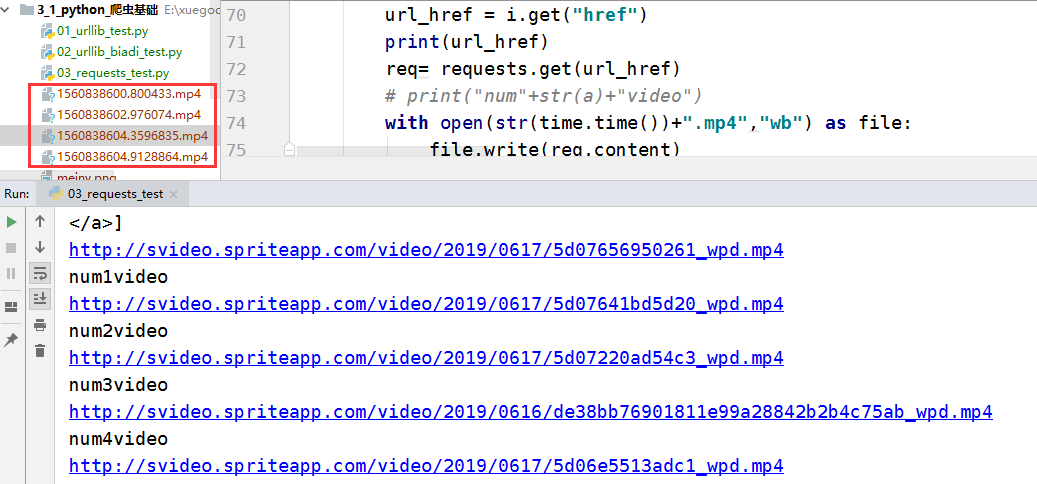
**对于某些tag属性不能通过搜索得到值，可以使用attrs参数得到**

**#返回class为e的标签**

**print(soup.find\_all(attrs={'class':'e'}))**

**实战：爬取百思不得姐小视频，存储到当前界面**  
**from bs4 import BeautifulSoup  
import requests  
import re,time  
url="http://www.budejie.com/video/"  
def get\_page(url,data=None):  
 header={'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:51.0) Gecko/20100101 Firefox/51.0'}  
 html = requests.get(url,headers=header)  
 soup=BeautifulSoup(html.text,"html.parser")  
 lists = soup.findAll('a',href=re.compile('http://svideo.spriteapp.com/video/2019/(.\*?).mp4'))  
 *# print(lists)* a = 0  
 for i in lists:  
 a+=1  
 url\_href = i.get("href")  
 print(url\_href)  
 req= requests.get(url\_href)  
 *# print("num"+str(a)+"video")* with open(str(time.time())+".mp4","wb") as file:  
 file.write(req.content)  
def get\_more\_pages(start,end):  
 for one in range(start,end):  
 get\_page(url+str(one))  
 time.sleep(2)  
if \_\_name\_\_ == '\_\_main\_\_':  
 get\_more\_pages(1,2)**

**运行结果如下：**



**总结：**

**2.1 re正则匹配**

**2.2 lxml xpath 匹配**

**2.3 beautifulsoup 匹配**

**实战：爬取内涵吧脑筋急转弯**

**实战：爬取猫眼电影排行榜并使用csv保存到本地的csv中。**

**实战：利用xpath 匹配爬取链家买房信息，存储到mysql中**

**实战：爬取百思不得姐小视频，存储到当前界面**