

第九章 Web数据源

本章导读

自Web诞生以来,大量的信息以网站形式呈现,网站也因此成为最重要的数据源之一。如电子商务网站含有大量商品信息,如果能定期获得商品单价并分析呈现,对很多领域有重要价值。此时,电子商务网站就是Web数据源。世界上有各种各样的网站,绝大多数网站都能成为数据源,将其分析应用,对很多领域都具有重要意义。

学习目标:

1. 理解网页结构;
2. 熟练使用正则表达式抽取网页中的数据;
3. 掌握Web文件的下载;
4. 了解网络爬虫工作机制;

本章目录

第一节 快速了解

第二节 过程认知

第三节 页面解析

1、初识BeautifulSoup

2、BeautifulSoup与正则表达式

3、BeautifulSoup与CSS

4、遍历元素树

第四节 小结

第一节 快速了解

以Web为数据源,常常需要从Web网站下载数据,例程9-1几行简单代码就可以完成网站内容抓取,实质性代码仅第2、5、6、7共4行。在金山词霸网站, <http://www.iciba.com/republic>显示republic单词的内容,如果修改republic换成如great等单词,则可以获得其他单词的信息。

例程9-1

```
第1行  #网页抓取简单示例
第2行  from urllib import request
第3行
第4行  #打开金山词霸网站的一个单词republic链接
第5行  dataFromWeb=request.urlopen("http://www.iciba.com/republic")
第6行  strData=dataFromWeb.read()
第7行  print(strData.decode("utf-8"))
第8行
第9行  #eof
```

从网站抓取内容,可以用Python标准功能库urllib中的request,如第2行代码所示。当现实其代码时,要确保解码方式与网站设置一致。在浏览器查看网页源码(一般单击右键的快捷菜单有查看网页源码的选项),观察到<meta charset='utf-8'>或类似即为utf-8编码,有些是gbk或其他。将第7行decode()函数的编码字符串设置为与网站一致即可。

金山词霸网站的单词大多支持美式发音和英式发音。观察其代码会发现,其发音部分的形式如下。通过正则表达式,可以获得其链接,并从网站下载该链接对应的文件且保存到外存储器之中。例程9-2是其实现代码。

```
sound('http://res-tts.iciba.com/tts_dj/d/c/8/dc8f79ae3b60d452dfb4e7700d19e8be.mp3')
```

```

第1行 #网页抓取简单示例
第2行 from urllib import request
第3行
第4行 #打开金山词霸网站的一个单词republic链接
第5行 dataFromWeb=request.urlopen("http://www.iciba.com/republic")
第6行 strData=dataFromWeb.read().decode("utf-8")
第7行
第8行 import re #装入正则表达式功能库
第9行
第10行 matchAll=re.findall("sound\\(\\'.*?mp3\\'",strData)
第11行 print(matchAll) #输出效果如下图所示
第12行
第13行 print()# 输出一空行
第14行
第15行 for dataItem in matchAll:
第16行     urlLink=dataItem[7:-2] #从matchAll的每一个条目dataItem获得链接
第17行     print(urlLink)
第18行
第19行     theLastSlashPosition=urlLink.rfind(r"/") #最后一个斜杠的位置
第20行     fileName=urlLink[theLastSlashPosition+1:] #从urlLink获得mp3的文件名
第21行
第22行     mp3FileOpen=open(fileName,"wb") #打开文件准备以二进制方式写入
第23行
第24行     mp3FromWeb=request.urlopen(urlLink) #根据urlLink从网站获取数据
第25行     mp3FileOpen.write(mp3FromWeb.read()) #输出每个链接对应的文件
第26行
第27行 #eof

```



```

命令提示符
["sound('http://res.iciba.com/resource/amp3/oxford/0/70/a1/70a182aa19a8e99febd443bd1c008488.mp3')", "sound('http://res.iciba.com/resource/amp3/1/0/95/15/9515f2c0b5f874cac6b04162c126dbfa.mp3')", "sound('http://res-tts.iciba.com/tts_dj/d/c/8/dc8f79ae3b60d452dfb4e7700d19e8be.mp3')", "sound('http://res-tts.iciba.com/tts_dj/6/b/4/6b49283935fe07fab86de39a3f1827e8.mp3')"]
http://res.iciba.com/resource/amp3/oxford/0/70/a1/70a182aa19a8e99febd443bd1c008488.mp3
http://res.iciba.com/resource/amp3/1/0/95/15/9515f2c0b5f874cac6b04162c126dbfa.mp3
http://res-tts.iciba.com/tts_dj/d/c/8/dc8f79ae3b60d452dfb4e7700d19e8be.mp3
http://res-tts.iciba.com/tts_dj/6/b/4/6b49283935fe07fab86de39a3f1827e8.mp3

```

图9-1 例程9-2执行效果

Python第三方功能库requests(<http://cn.python-requests.org>)功能更加强大,使用也颇为简单,但使用前需要用“pip install requests”或“pip3 install requests”进行安装。例程9-3是其简单示例。

```

第1行 import requests
第2行
第3行 r = requests.get('http://www.pku.edu.cn') #取得http://www.pku.edu.cn网址的内容
第4行 r.encoding="utf-8" #设置网页编码,此处一般为utf-8

```

第5行	
第6行	<code>print(r.text) #输出网页文本，r.text结果为字符串</code>
第7行	
第8行	<code>webFile=open("pku.html","w",encoding="utf-8")</code>
第9行	<code>webFile.write(r.text)</code>
第10行	<code>webFile.close()</code>
第11行	
第12行	<code>#eof</code>

除了可以获取网页代码，也可以获取Web相关文件，如例程9-4所示。注意第8行代码是res.content而不是res.text。当取得二进制非文本文件如图片、音乐等时，要采用res.content字节模式。经验证，下载的MP3能正常播放效果一致。

例程9-4

第1行	<code>import requests</code>
第2行	
第3行	<code>#res是requests.get()产生的对象，此处一般称之为response对象</code>
第4行	<code>res = requests.get('http://res.iciba.com/resource/amp3/oxford/0/70/a1/70a182aa19a8e99febd443bd1c008488.mp3')</code>
第5行	<code>#金山词霸发音文件所在</code>
第6行	
第7行	<code>mp3File=open("republic.mp3","wb") #以二进制方式存储</code>
第8行	<code>mp3File.write(res.content) #注意：二进制文件需要用res.content，如果是文本，可以用res.text</code>
第9行	<code>mp3File.close()</code>
第10行	
第11行	<code>#eof</code>

URL带参数或者通过页面的form传递请求request是很常见，requests能很好地支持，如例程9-5所示。

例程9-5

第1行	<code>import requests</code>
第2行	<code>dictKey={"wd":"中国"}</code>
第3行	
第4行	<code>rsp=requests.get("https://www.baidu.com/s",params=dictKey,headers=Headers)</code>
第5行	<code>print(rsp.text)</code>

另外，requests.get()中的get可以更换为post、put等，分别对应method的post和put等，其用法与requests.get()相似。

第二节 过程认知

网络爬虫是一种抓取万维网信息的程序。用户端以各种形式向Web服务器发出请求（request），服务器根据请求反馈（response）相应内容。例程9-6是一个Python+Flask构成的Web服务器程序，例程9-7是一个简单爬虫程序，通过该程序有助于认知其基本原理。

从例程9-6第10-13行代码可以看出，当执行路由“/crawler”将输出请求request的headers信息。当例程启动执行后，在浏览器地址栏输入：http://127.0.0.1/crawler，将在服务器端输出调试信息如图9-2所示。当正确启动执行例程9-7后，服务器端输出的调试信息如图9-3所示。比较图9-2和图9-3，可以很容易判断是浏览器正常访问还是网络爬虫。

例程9-6

第1行	<code>#用于测试Web爬虫</code>
第2行	<code>from flask import Flask,request</code>
第3行	
第4行	<code>spiderWeb=Flask(__name__)</code>

```

第5行
第6行 @spiderWeb.route("/")
第7行 def firstPage():
第8行     return "firstPage"
第9行
第10行 @spiderWeb.route("/crawler")
第11行 def getMe():
第12行     print(request.headers)
第13行     return f"{request.headers}"
第14行
第15行 if __name__ == "__main__":
第16行     spiderWeb.run(host="0.0.0.0",port=80,debug=True)

```

例程9-7

```

第1行 import requests
第2行
第3行 rsp=requests.get("http://127.0.0.1/crawler")
第4行 rsp.encoding="utf-8"
第5行 print(rsp.text)
第6行 print(rsp.status_code)

```

```

Host: 127.0.0.1
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.92 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9

```

图9-2 在浏览器输入http://127.0.0.1/crawler后输出的调试信息

```

Host: 127.0.0.1
User-Agent: python-requests/2.23.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive

```

图9-3 正确启动例程9-7后服务器端输出的调试信息

通过headers参数，requests能更好地模拟浏览器，如例程9-8所示，从例程第3、第4行可以看出，headers是一个Python字典，可以设置各种参数。从图9-4可以看出，已经很像浏览器访问Web服务器了。

例程9-8

```

第1行 import requests
第2行
第3行 headers={'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.92 Safari/537.36'}
第4行 headers["Accept"] = "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q="

```

```
0.8,application/signed-exchange;v=b3;q=0.9"
```

第5行

第6行

```
rsp=requests.get("http://127.0.0.1/crawler",headers=headers)
```

第7行

```
rsp.encoding="utf-8"
```

第8行

```
print(rsp.text)
```

第9行

```
print(rsp.status_code)
```

```
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.92 Safari/537.36
Accept-Encoding: gzip, deflate
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Connection: keep-alive
```

图9-4 例程9-8正确执行后服务器端输出的调试信息

对于要爬取的Web网站，构造Web服务器通常不现实，但通过浏览器的开发者工具(按键盘上的F12可以进入)能很好地观察到浏览器与服务器的交互过程。图9-5是在例程9-6正常运行的前提下，在浏览器地址栏输入“tp://127.0.0.1/crawler?wd=10”所看到的界面。虽然例程不处理问号?后的参数，但并不影响输入。从图中可以看出，request的模式是get（如前面所学，还有post等模式）；执行状态(status code)200（执行正常）；远程地址（remote address）是127.0.0.0:80。

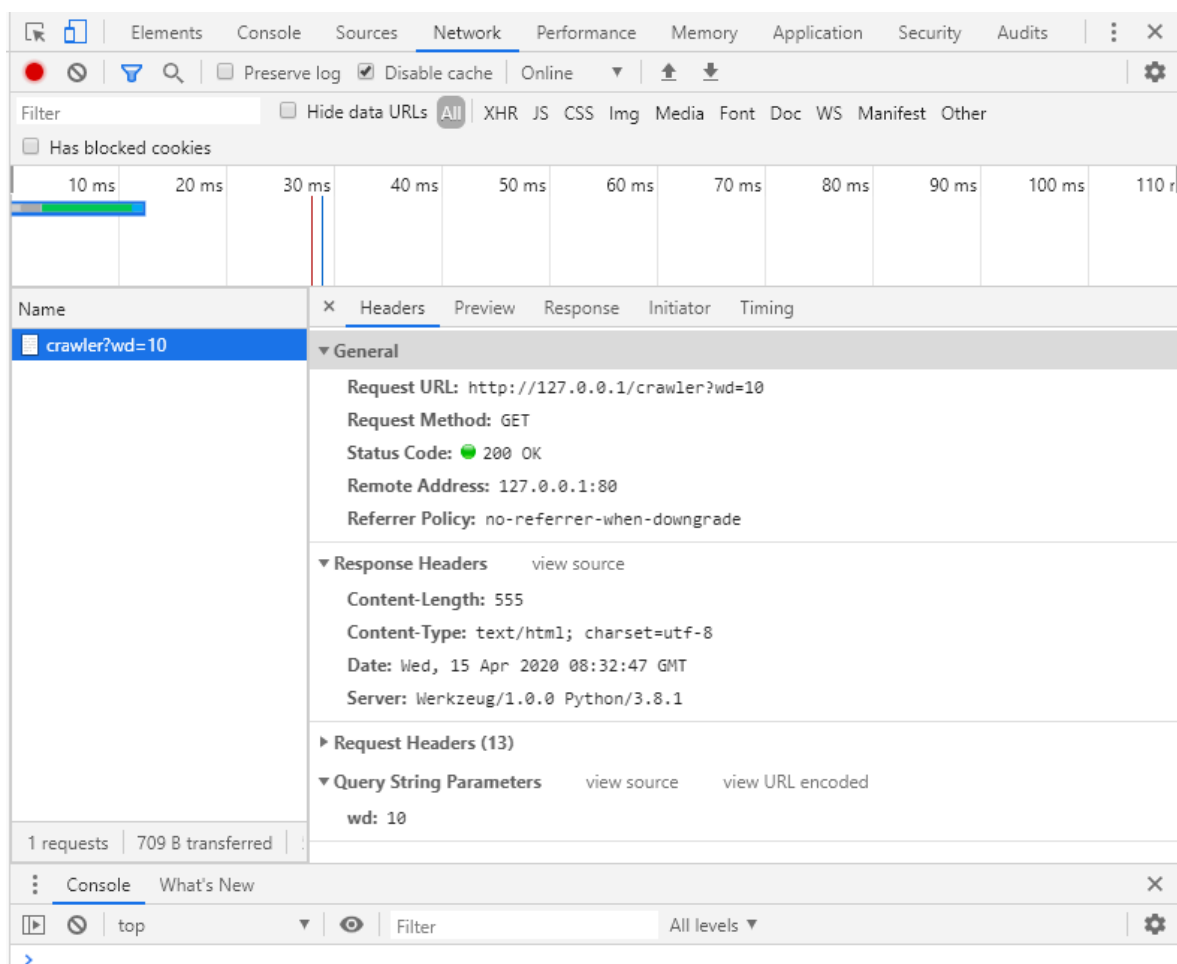


图9-5 浏览器的开发者工具

单击图中“Request Headers”，能得到如图9-6所示界面，从中可以看出各种headers参数。如果用多种浏览器或多种操作系统平台，可以更好观察对比。

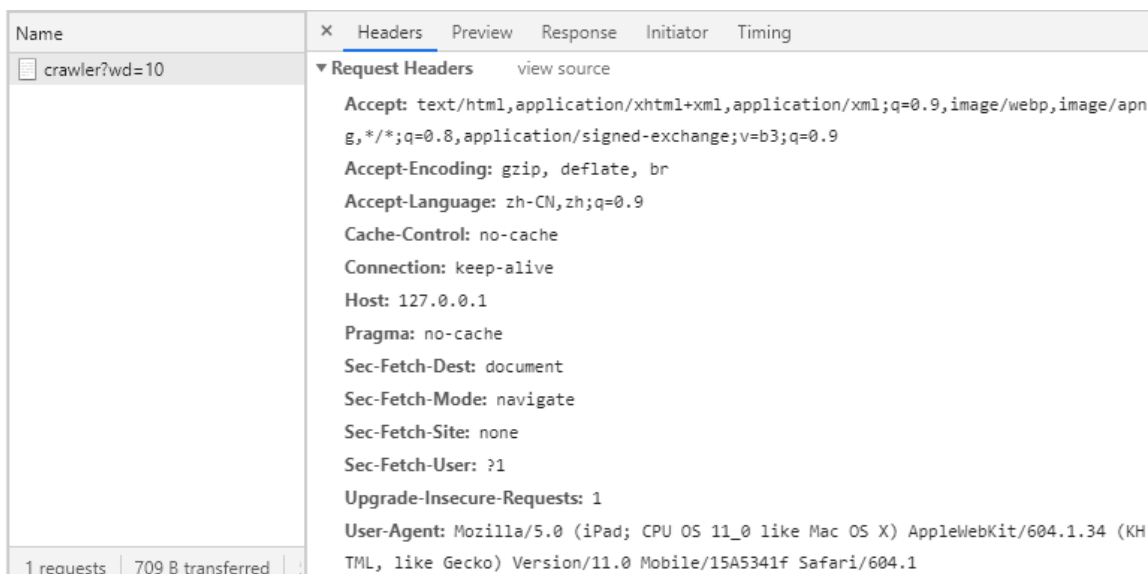


图9-6 在浏览器开发者工具中查看Request Headers

图9-7是在访问www.baidu.com页面时，浏览器开发者工具呈现的界面，从中可以看出该页面访问各种资源如：页面、JavaScript文件、CSS、图片、XHR（通过AJAX时访问时启动）等。当在页面进行操作时，该界面也随之变化，能很好地观察网站的运行过程。

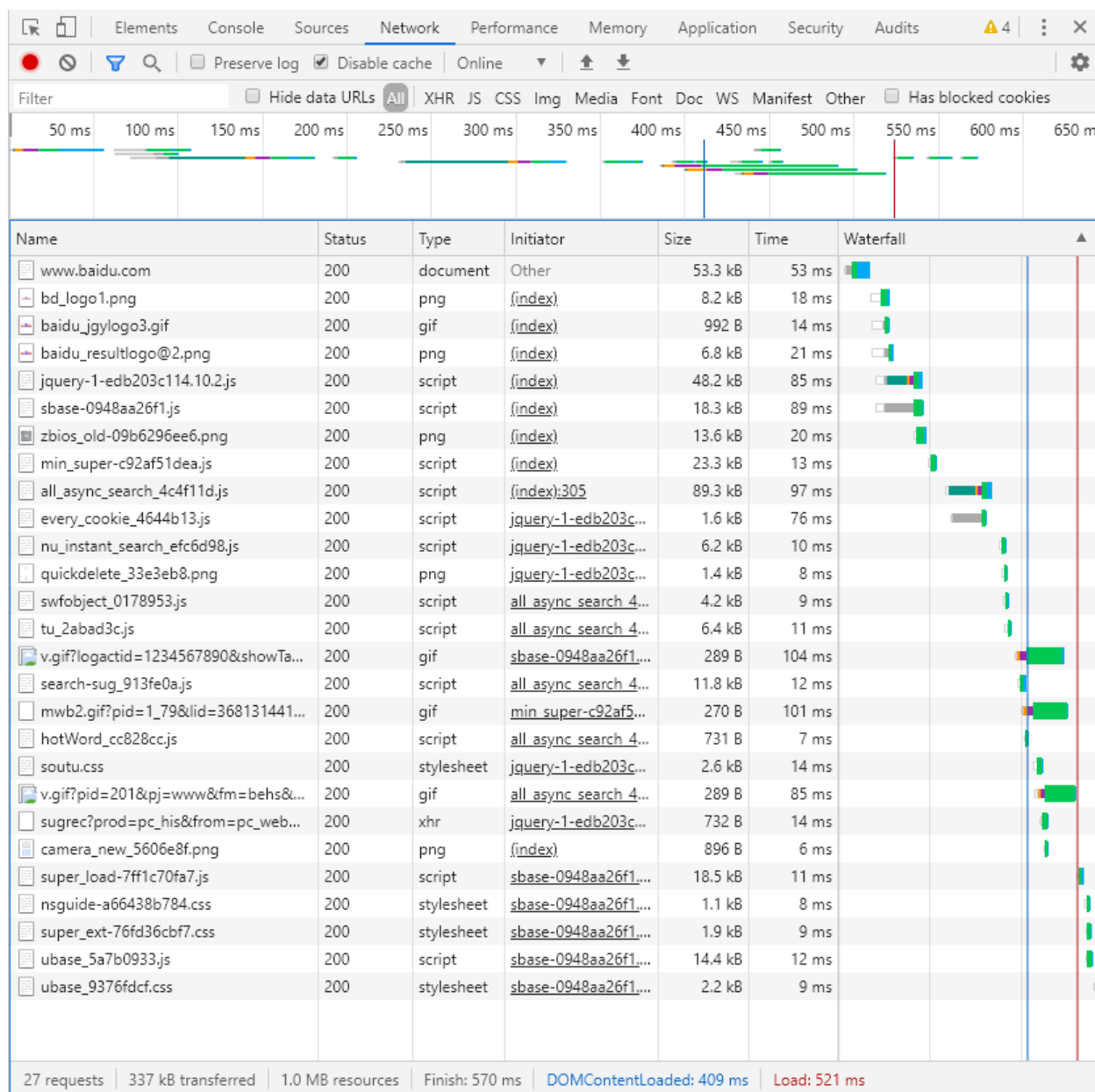


图9-7 访问www.baidu.com时浏览器开发者工具界面

第三节 页面解析

页面解析的核心是根据页面要素查找满足条件网页元素。Python提供多种页面解析方式，可以是正则表达式，也可以用标准库如HTMLParser，也有功能更加强大的第三方库如BeautifulSoup(靓汤?)。

1、初识BeautifulSoup

例程9-9是BeautifulSoup的应用示例。BeautifulSoup是第三方库，需要安装后才能使用，安装命令是“pip install beautifulsoup4”或“pip3 install beautifulsoup4”。例程9-9第8行代码oBS=BeautifulSoup(htmlCode,"html.parser")生成一个BeautifulSoup对象，生成该对象时有两个参数，分别是HTML文档和解析器。HTML文档可以来自网站由urllib.request或requests采集而来，也可以是HTML代码文件以及字符串。解析器负责解析HTML文档，可以设定Python内置模块html.parser如本例也可以设置其他解析器如xml(第三方库需要安装)。

BeautifulSoup对象有一系列方法和属性。第9行代码prettify()方法用于生成层级缩进易读性很好的网页代码，如图9-8。BeautifulSoup提供了非常简便的网页元素访问方式，如第10-14行所示，可以直接访问元素如oBS.h1或oBS.a分别访问h1或a元素的内容，也可以层级访问元素如oBS.body.h1或oBS.html.body.h1。如果层级范围内或页面范围内仅有一个元素，这种方法效果不过，如果有多个元素，仅访问第1个元素，如例程第14行所示，多个a元素仅显示第一个a元素。另外，在BeautifulSoup中，元素名称必须是小写，否则返回值为None如例程第16行所示，虽然在网页中元素名称可以不是小写。

第18行代码中的find_all()在BeautifulSoup的网页解析中大量使用，其功能是查找符合参数的所有元素，其结果为元素的list，其顺序按在网页中出现的先后为序，本例oBS.find_all("a")是查找网页中所有的a元素，如图9-9所示。

例程9-9

```
第1行
第2行
第3行  htmlCode="""<!doctype html><html lang="en">
第4行  <head><meta charset="UTF-8"><title>苏东坡</title></head>
      <body><h1>苏东坡简介</h1><p id="firstP">苏轼（1037年1月8日—1101年8月24日），字子瞻，又字和仲，号
第5行  铁冠道人、东坡居士，世称苏东坡、苏仙。汉族，眉山<span class="placeName">眉山</span>（今属四川省眉山
      市）人，祖籍河北<span class="placeName">栾城</span>，北宋著名文学家、书法家、画家。</p>
      <p id="secondP"><span class="reignTitle">嘉祐</span>二年（1057年），苏轼进士及第。<a href="http://ba
第6行  ike.baidu.com/item/宋神宗">宋神宗</a>时曾在<span class="placeName">凤翔</span>、<span class="place
      Name">杭州</span>、<span class="placeName">密州</span>、<span class="placeName">徐州</span
      >、<span class="placeName">湖州</span>等地任职。<span class="reignTitle">元丰</span>三年（1080
      年），因“乌台诗案”受诬陷被贬<span class="placeName">黄州</span>任团练副使。<a href="http://baike.b
第7行  aidu.com/item/宋哲宗">宋哲宗</a>即位后，曾任<span class="Officer">翰林学士</span>、<span class="Offi
      cer">侍读学士</span>、<span class="Officer">礼部尚书</span>等职，并出知<span class="placeName">杭
      州</span>、<span class="placeName">颍州</span>、<span class="placeName">扬州</span>、<span cla
第8行  ss="placeName">定州</span>等地，晚年因新党执政被贬<span class="placeName">惠州</span>、<span cla
      ss="placeName">儋州</span>。<a href="http://baike.baidu.com/item/宋徽宗">宋徽宗</a>时获大赦北还，
第9行  途中于<span class="placeName">常州</span>病逝。<A href="http://baike.baidu.com/item/宋高宗">宋高宗
      </a>时追赠太师，谥号“文忠”。</p></body></html>"""
第10行
第11行  from bs4 import BeautifulSoup
第12行
第13行  oBS=BeautifulSoup(htmlCode,"html.parser")
第14行  print(oBS.prettify()) #标准层级缩进格式输出HTML代码
第15行  print(oBS.h1) #输出：<h1>苏东坡简介</h1>
第16行  print(oBS.html.body.h1) #输出：<h1>苏东坡简介</h1>
第17行  print(oBS.body.h1) #输出：<h1>苏东坡简介</h1>
第18行
第19行  print(oBS.a)
```

```

第17行 #多个只输出一个: <a href="http://baike.baidu.com/item/宋神宗">宋神宗</a>
第18行 print(oBS.A) #输出: None
第19行
第20行 hrefs=oBS.find_all("a")
第21行 for hrefItem in hrefs:
第22行     print(hrefItem) #先后输出4个链接
第23行 #形如: <a href="http://baike.baidu.com/item/宋神宗">宋神宗</a>
第24行
第25行 #eof

```

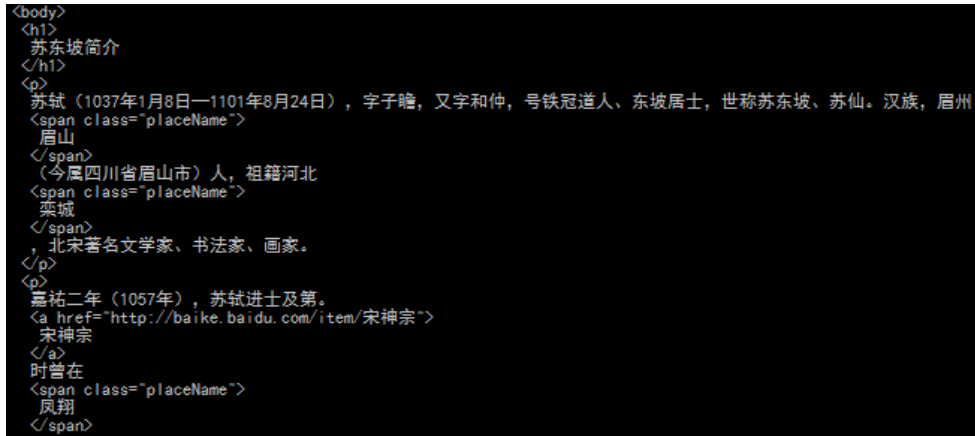


图9-8 BeautifulSoup之prettify()执行效果局部



图9-9 例程9-9第10行起执行效果

find_all()在BeautifulSoup被大量使用，其格式如下，例程9-10是其应用示例。格式中的tag表示元素；attributes表示属性；recursive表示是否递归查找，如果其值为False则仅查找一级元素，否则将包含查找所有子元素；text表示查找元素的文本内容，limit表示结果集的数量；keywords则表示查找具有指定属性的元素。

find_all(tag,attributes,recursive,string,limit,keywords)

例程9-10

```

第1行 htmlCode=""同上例""
第2行
第3行 from bs4 import BeautifulSoup
第4行
第5行 oBS=BeautifulSoup(htmlCode,"html.parser")
第6行 print(oBS.find_all("span")) #查找所有span元素
第7行 print(oBS.find_all({"span","a"})) #查找所有span和a元素
第8行 print(oBS.find_all("",{"class":"placeName"})) #查找所有class名为placeName的元素
第9行 print(oBS.find_all(string="杭")) #None
第10行 print(oBS.find_all(string="杭州")) #输出: ['杭州', '杭州']
第11行 print(oBS.find_all("span",limit=5)) #仅输出前5个
第12行 print(oBS.find_all(id="firstP")) #输出keyword之id为firstP

```



```

第13行 print(oBS.find_all({"id":{"firstP","secondP"}})) #属性查找方法
第14行 print(oBS.find_all(class_=["placeName"])) #class是Python关键字, class_其替身
第15行 print(oBS.find_all({"span","a"},class_=["placeName","reignTitle"]))
第16行
第17行 #eof

```

bs.find_all()可以直接查找元素，可以一次找一种元素如例程第6行也可以一次找多个元素如例程第7行。当查找多个元素时，被查找元素放在序列数据类型之中，可以是list，也可以是set或者tuple。查找是可以按属性查找，如例程第8行单属性值以及第13行多属性值。注意当查找多属性值时，其关系为“或”。

当设定string值时，要求是全匹配，如例程9-10行所示。第12、14、15行是按keyword查找。注意其与属性查找的区别，第13行是属性查找，而第12行为keyword查找。当使用keyword是class时，必须改为class_，其原因是class是Python的关键字，或者改为用属性查找(如第8行)。第15行代码oBS.find_all({"span","a"},class_=["placeName","reignTitle"])的含义是查找span或a元素，且其class属性值为placeName或reignTitle知一。

找到元素有一些属性和方法，如tag.name是元素的名称，tag.attrs是元素的属性，tag.get_text()获得元素的文本等等，例程9-11其应用示例。例程第8行代码tagP.get_text()获得tagP所代表元素的全部文本，注意仅仅是文本，不包括html元素，如图9-10所示。第9行tagP.name是获得元素的名称此处输出p正是tagP所代表的元素；tagP.attrs获得元素的属性，以字典方式列示；tagP.has_attr()用于判断元素是否具有某个属性，返回值为布尔型。注意tagP.string的输出值为None，与tagP.get_text()的输出有很大不同，二者之间的区别如例程9-12所示。

例程9-11

```

第1行 #同上例
第2行 htmlCode=""
第3行 from bs4 import BeautifulSoup
第4行
第5行 oBS=BeautifulSoup(htmlCode,"html.parser")
第6行 tagP=oBS.p
第7行 print(tagP.get_text()) #显示第1个p元素的所有文本
第8行 print(tagP.string,tagP.name,tagP.attrs,tagP.has_attr("class"))
第9行 #上行输出: None p {'id': 'firstP'} False
第10行
第11行 allA=oBS.find_all({"class":"reignTitle"}) #选中所有class名称为reignTitle的元素
第12行 for ele in allA:
第13行     print(ele.string,ele.name,ele.attrs)
第14行
第15行 """#输出:
第16行 嘉祐 span {'class': ['reignTitle']}
第17行 元丰 span {'class': ['reignTitle']}
第18行 """
第19行 #eof

```

```

苏轼（1037年1月8日—1101年8月24日），字子瞻，又字和仲，号铁冠道人、东坡居士，世称苏东坡、苏仙。汉族，眉州眉山（今属四川省眉山市）人，祖籍河北栾城，北宋著名文学家、书法家、画家。
None p {'id': 'firstP'} False
嘉祐 span {'class': ['reignTitle']}
元丰 span {'class': ['reignTitle']}

```

图9-10 例程9-11执行效果

如例程9-12所示，当元素内没有其他元素是bs.get_text()与bs.string没有区别，正如例程第10、11行所示。当元素内有其他元素时，则bs.get_text()将所有元素的文本组合在一起，而bs.string则返回值是None。不过可以通过bs.strings逐一访问元素内每一个元素的文本内容。

例程9-12

```

第1行 from bs4 import BeautifulSoup
第2行
第3行 htmlCode=""
第4行 <p>床前明月光，疑是地上霜。</p>
第5行 <div><span>举头望明月，</span><span>低头思故乡。</span>---唐·李白</div>
第6行 ""
第7行
第8行 oBS=BeautifulSoup(htmlCode,"html.parser")
第9行 tagP=oBS.p
第10行 print(tagP.get_text()) #输出：床前明月光，疑是地上霜。
第11行 print(tagP.string) #输出：床前明月光，疑是地上霜。
第12行
第13行 tagDiv=oBS.div
第14行 print(tagDiv.get_text()) #输出：举头望明月，低头思故乡。---唐·李白
第15行 print(tagDiv.string) #输出：None
第16行
第17行 for strItem in tagDiv.strings:
第18行     print(strItem) #输出：div中每个元素的string属性
第19行
第20行 for eleItem in oBS(): #oBS()等价于oBS(True)，包含全部元素
第21行     print(eleItem.contents) #输出每个元素的内容，包含html代码
第22行
第23行 #eof

```

```

床前明月光，疑是地上霜。
床前明月光，疑是地上霜。
举头望明月，低头思故乡。——唐·李白
None
举头望明月，
低头思故乡。
——唐·李白
['床前明月光，疑是地上霜。']
[<span>举头望明月，</span> <span>低头思故乡。</span>, '——唐·李白']
['举头望明月，']
['低头思故乡。']

```

图9-11 例程9-12执行效果

另外，bs.contents也比较常用，其功能是返回元素的内容，如果内容含有元素也一并返回，这时与bs.get_text()最大的不同，如例程9-12第21行所示。

2、BeautifulSoup与正则表达式

例程9-13

```

第1行 #htmlCode同上例
第2行 htmlCode=""
第3行
第4行 from bs4 import BeautifulSoup
第5行 import re
第6行
第7行 oBS=BeautifulSoup(htmlCode,"html.parser")
第8行 print(oBS.find_all(re.compile("an|a"))) #输出：所有元素名称中含有an或a的元素
第9行 print(oBS.find_all(text=re.compile("州"))) #输出：所有文本含“州”字的元素
第10行 print(oBS.find_all("span",text=re.compile("州"))) #输出：所有span元素含“州”的元素
第11行 print(oBS.find_all("a",href=re.compile("^http"))) #所有a元素属性以http开始

```

```

第12行 print(oBS.find_all("span",class_=re.compile("ce")))#所有span元素其class名称中含有ce的元素
第13行
第14行 #eof

```

3、BeautifulSoup与CSS

例程9-14

```

第1行 #同上例
第2行 htmlCode=""
第3行 from bs4 import BeautifulSoup
第4行 import re
第5行
第6行 oBS=BeautifulSoup(htmlCode,"html.parser")
第7行 print(oBS.select("span")) #选中所有span元素
第8行 print(oBS.select("#firstP")) #选中id为firstP的元素
第9行 print(oBS.select(".placeName")) #选中class名称为placeName的元素
第10行 print(oBS.select(".reignTitle,.placeName")) #选中class名称为placeName或reignTitle的元素
第11行 print(oBS.select("span.reignTitle")) #选中span元素中class名称为reignTitle的元素
第12行 print(oBS.select("#firstP span")) #选中id为firstP元素之下的晚辈span元素
第13行 print(oBS.select("#firstP > span")) #选中id为firstP元素之子辈span元素，大于符号两侧必须有空格
第14行 print(oBS.select("span[class]")) #含有class属性的span元素
第15行 print(oBS.select("span[class='placeName']")) #含有class属性且其值为placeName的span元素
第16行 print(oBS.select("span[class^='place']")) #含有class属性且其值以place开始的span元素
第17行 print(oBS.select("span[class$='Title']")) #含有class属性且其值以Title结尾的span元素
第18行 print(oBS.select("span[class*='ce']")) #含有class属性且其值含ce的span元素
第19行 print(oBS.select("span:nth-of-type(4)")) #选中文档中的第4个span元素,第1个编号为1，不是0
第20行 print(oBS.select("span:nth-of-type(4) ~ span")) #文档中第4个span元素之后的所有同辈span元素
第21行 print(oBS.select("span:nth-of-type(4) + span")) #文档中第4个span元素之后的第1个同辈span元素
第22行
第23行 #eof

```

bs.select()执行结果为list，如果仅需要选中一个，可以将bs.select()修改为bs.select_one()。

4、遍历元素树

网页文档是一棵元素树，以某个元素或者节点为基础，可以找到前辈同辈后辈元素，例程9-15是网页元素遍历的部分示例，图9-12是其执行效果，表9-1列出BeautifulSoup支持的遍历函数和属性。bs.find_parents()与bs.find_parent()的参数与bs.find_all()相似，但没有limit参数，其他都一样。bs.find_parent()返回值为元素对象，而bs.find_parents()则是list对象，包含一个或多个元素对象。注意比较第14、15行输出的异同。

例程9-15

```

第1行 from bs4 import BeautifulSoup
第2行
第3行 htmlCode="""<html><head><title>古诗词</title></head><body><div id="GuShiCi">
第4行     <ul id="TangShi"><li>李白</li><li>杜甫</li><li>王维</li></ul>
第5行     <ul id="SongCi"><li>苏东坡</li><li>陆游</li><li>李清照</li></ul>
第6行 </div></body>
第7行 """
第8行
第9行 oBS=BeautifulSoup(htmlCode,"html.parser")

```

```

第10行 tagSelected=oBS.select_one("#TangShi > li:nth-of-type(2)")
第11行 print(tagSelected.get_text()) #输出: 杜甫
第12行 print(tagSelected.parent.contents) #输出: [<li>李白</li>, <li>杜甫</li>, <li>王维</li>]
第13行 print([strTag.name for strTag in tagSelected.parents]) #输出: ['ul', 'div', 'body', 'html', '[document]']
第14行 print(tagSelected.find_parents(id="GuShiCi"))
第15行 print(tagSelected.find_parent(id="GuShiCi"))
第16行
第17行 print(tagSelected.next_sibling) #输出: <li>王维</li>
第18行 print([tagItem.get_text() for tagItem in tagSelected.next_siblings]) #输出: ['王维']
第19行 print([tagItem.get_text() for tagItem in tagSelected.find_all_next()])
第20行 #输出: ['王维', '苏东坡陆游李清照', '苏东坡', '陆游', '李清照']
第21行
第22行 #eof

```

```

杜甫
[<li>李白</li>, <li>杜甫</li>, <li>王维</li>]
['ul', 'div', 'body', 'html', '[document]']
[<div id="GuShiCi">
<ul id="TangShi"><li>李白</li><li>杜甫</li><li>王维</li></ul>
<ul id="SongQi"><li>苏东坡</li><li>陆游</li><li>李清照</li></ul>
</div>
<div id="GuShiCi">
<ul id="TangShi"><li>李白</li><li>杜甫</li><li>王维</li></ul>
<ul id="SongQi"><li>苏东坡</li><li>陆游</li><li>李清照</li></ul>
</div>
<li>王维</li>
['王维']

```

图9-12 例程9-15执行效果

表9-1: BeautifulSoup的遍历函数和属性

名称	描述
bs.parent/bs.parents	bs.parent用于找到指定元素的父辈元素。bs.parents则是找到包括父辈元素在内的所有长辈元素。父辈元素仅有一个，长辈元素则多是多个。
bs.find_parent()/bs.find_parents()	bs.find_parent()用于找到指定的元素的父辈元素，而bs.find_parents()则是找到所有长辈元素。
bs.next_sibling/bs.next_siblings	bs.next_sibling用于查找其后的第1个同辈元素；bs.next_siblings用于查找其后的所有同辈元素。
bs.find_next_siblings()/bs.find_next_sibling()	bs.find_next_siblings()用于查找其后的所有同辈元素；bs.find_next_sibling()用于查找其后的第1个同辈元素。
bs.previous_sibling/bs.previous_siblings	bs.previous_sibling用于查找其前的第1个同辈元素；bs.previous_siblings用于查找其前的所有同辈元素。
bs.find_previous_siblings()/bs.find_previous_sibling()	与bs.find_next_siblings()/bs.find_next_sibling()相似，仅方向不同而已。
bs.descendants/bs.children	bs.descendants用于查找后辈元素；bs.children用于查找子元素；
bs.next_element/bs.next_elements	bs.next_element用于查找其后的第1个元素；bs.next_elements用于查找其后的所有元素。
bs.find_next()/bs.find_all_next()	bs.find_next()用于查找下一个元素；bs.find_all_next()用于其后的所有元素。
bs.previous_element/bs.previous_elements	与bs.find_next()/bs.find_all_next()相似，但方向相反。
bs.find_previous()/bs.find_all_previous()	bs.find_previous()用于查找前面的元素；bs.find_all_previous()用于查找前面的所有元素。
所有find系列函数的参数都与bs.find_all()相似，但没有limit参数项。	

第四节 小结

