

## 第二十三章 网络爬虫基础

在数据量爆发式增长的互联网时代，网站与用户的沟通本质上是数据的交换：搜索引擎从数据库中提取搜索结果，将其展现在用户面前；电商将产品的描述、价格展现在网站上，以供买家选择心仪的产品；社交媒体在用户生态圈的自我交互下产生大量文本、图片和视频数据等。这些数据如果得以分析利用，不仅能够帮助第一方企业（拥有这些数据的企业）做出更好的决策，对于第三方企业也是有益的。而网络爬虫技术，则是大数据分析领域的第一个环节。

### 23.1 初识网络爬虫

#### 23.1.1 什么是网络爬虫

如果我们把互联网比作一张大的蜘蛛网，那一台计算机上的数据便是蜘蛛网上的一个猎物，而爬虫程序就是一只小蜘蛛，沿着蜘蛛网抓取自己想要的猎物/数据。

通俗理解：爬虫是一个模拟人类请求网站行为的程序。可以自动请求网页、并把数据抓取下来，然后使用一定的规则提取有价值的信息。

比如百度搜索引擎的爬虫叫做百度蜘蛛（Baiduspider）。百度蜘蛛每天会在海量的互联网信息中爬取，爬取优质信息并搜寻，当用户在百度搜索引擎上检索对应关键词时，百度将对关键词进行分析处理，从收录的网页中找出相关网页，按照一定的排名规则进行排序并将结果展现给用户。

除了百度搜索引擎离不开爬虫以外，其他搜索引擎也离不开爬虫，它们也拥有自己的爬虫。比如 360 的爬虫叫 360Spider，搜狗的爬虫叫 Sogospider，必应的爬虫叫 Bingbot。

大数据时代也离不开爬虫，比如在进行大数据分析或数据挖掘时，可以去一些比较大型的官方站点下载数据源。但这些数据源比较有限，那么如何才能获取更多更高质量的数据源呢？此时，我们可以编写自己的爬虫程序，从互联网中进行数据信息的获取。

#### 23.1.2 能从网络上爬取什么数据

简单来说，平时在浏览网站时，所有能见到的数据都可以通过爬虫程序保存下来。从社交媒体的每一条发帖到团购网站的价格及点评，再到招聘网站的招聘信息，这些数据都可以存储下来。

#### 23.1.3 应不应该学爬虫

应不应该学爬虫？这也是我之前问自己的一个问题，我认为对于任何一个与互联网有关的从业人员，无论是非技术的产品、运营或营销人员，还是前断、后端的程序员，都应该学习网络爬虫技术。

一方面，网络爬虫简单易学、门槛很低。没有任何编程基础的人在认真看完本章的爬虫基础内容后，都能够自己完成简单的网络爬虫任务，从网站上自动获取需要的数据。

另一方面，网络爬虫不仅仅能使你学会一项新的技术，还能让你在工作的时候节省大量

的时间。如果你对网络爬虫的世界有兴趣，就算你不懂编程也不要担心，本章将深入浅出讲解网络爬虫。

### 23.1.4 网络爬虫是否合法

网络爬虫领域目前还属于早期的拓荒阶段，虽然互联网世界已经通过自身的协议建立起一定的道德规范（Robots 协议），但法律部分还在建立和完善中。从目前情况来看，如果抓取的数据属于个人使用或科研范畴，基本不存在问题；而如果数据属于商业盈利范畴，就要就事论事，有可能属于违法行为，也有可能不违法。

除了上述 Robots 协议之外，我们使用网络爬虫的时候还要对自己进行约束：过于快速或者频密的网络爬虫都会对服务器产生巨大的压力，网站可能封锁你的 IP，甚至采取进一步的法律行动。因此，需要约束自己的网络爬虫行为，将请求的速度限定在一个合理的范围之内。

## 23.2 爬虫的基本流程

网络爬虫的基本工作流程如下：

- 1) 浏览器通过 DNS 服务器查找域名对应的 IP 地址。
- 2) 向 IP 地址对应的 Web 服务器发送请求。
- 3) Web 服务器响应请求，发回 HTML 页面。
- 4) 浏览器解析 HTML 内容，并显示出来。



图 23-1 网络爬虫的基本工作流程

## 23.3 HTTP 协议

HTTP 协议（HyperText Transfer Protocol，超文本传输协议）是用于 WWW 服务器传输超文本到本地浏览器的传送协议。它可以使浏览器更加高效，减少网络传输。它不仅保证计算机正确快速传输超文本文档，还确定传输文档中的哪一部分，以及哪部分内容首先显示（如文本先于图形）等。之后的 Python 爬虫开发，主要就是和 HTTP 协议打交道。

### 23.3.1 HTTP 请求过程

HTTP 协议采取的是请求响应模型，HTTP 协议永远都是客户端发起请求，服务器回送响应。模型如图 23-2 所示。



图 23-2 请求响应模型

HTTP 协议是一个无状态的协议，同一个客户端的这次请求和上次请求没有对应关系。一次 HTTP 操作称为一个事物，其执行过程可分为四步：

- 首先客户端与服务器建立连接，例如单击某个超链接。
- 建立连接后，客户端发送一个请求给服务器，请求方式的格式为：统一资源标识符（URL）、协议版本号，后边是 MIME 信息，包括请求修饰符、客户机信息和可能的内容。
- 服务器接到请求后，给予相应的响应信息，其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码，后边是 MIME 信息，包括服务器信息、实体信息和可能的内容。
- 客户端接收服务器所返回的信息，通过浏览器将信息显示在用户的显示屏上，然后客户端与服务器端口断开连接。

### 23.3.2 HTTP 状态码

当浏览者访问一个网页时，浏览者的浏览器会向网页所在的服务器发出请求。在浏览器接收并显示网页前，此网页所在的服务器会返回一个包含 HTTP 状态码的信息头（server header）用以响应浏览器的请求。HTTP 状态码主要是为了标识此次 HTTP 请求的运行状态。下面是常见的 HTTP 状态码：

- 200——请求成功。
- 301——资源被永久转移到其他 URL。
- 404——请求的资源不存在。
- 500——内部服务器错误。

HTTP 状态码由三个十进制数字组成，第一个十进制数字定义了状态码的类型。HTTP 状态码分为 5 中类型，如表 23-1 所示。

表 23-1 HTTP 状态码

分类	分类描述
1**	信息，服务器收到请求，需要请求者继续执行操作
2**	成功，操作被成功接收并处理
3**	重定向，需要进一步的操作以完成请求
4**	客户端错误，请求包含语法错误或无法完成请求

5\*\*

服务器错误，服务器在处理请求的过程中发生了错误

### 23.3.3 HTTP 头部信息

HTTP 头部信息由众多的头域组成，每个头域由一个域名、冒号（:）和域值三部分组成。域名是大小写无关的，域值前可以添加任何数量的空格符，头域可以被扩展为多行，在每行开始处，使用至少一个空格或制表符。

通过浏览器访问百度首页时，使用 F12 打开开发者工具，里面可以监控整个 HTTP 访问的过程。下面就以访问百度的 HTTP 请求进行分析，首先是浏览器发出请求，请求头的数据如下：

```
GET / HTTP/1.1
Host: www.baidu.com
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.113 Mobile 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
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
```

在请求头中包含以下内容：

- GET 代表的是请求方式，HTTP/1.1 表示使用 HTTP1.1 协议标准。
- Host 头域，用于指定请求资源的 Internet 主机和端口号，必须表示请求 URL 的原始服务器或网关的位置。HTTP/1.1 请求必须包含主机头域，否则系统会以 400 状态码返回。
- User-Agent 头域，里面包含发出请求的用户信息，其中有使用的浏览器型号、版本和操作系统的信息。这个头域经常用来作为反爬虫的措施。
- Accept 请求报头域，用于指定客户端接受哪些类型的信息。例如：Accept: image/gif，表明客户端希望接受 GIF 图像格式的资源；Accept: text/html，表明客户端希望接受 html 文本。
- Accept-Language 请求报头域，类似于 Accept，但是它用于指定一种自然语言。例如：Accept-Language: zh-ch。如果请求消息中没有设置这个报头域，服务器假定客户端对各种语言都可以接受。
- Accept-Encoding 请求报头域，类似于 Accept，但是它用于指定可接受的内容编码。

- Connection 报头域允许发送用于指定连接的选项。例如指定连接的状态是连续，或者指定“close”选项，通知服务器，在响应完成后，关闭连接。

请求发送成功后，服务器进行响应，响应的信息，数据如下：

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Coremonitorno: 0
Date: Mon, 18 May 2020 09:00:11 GMT
Vary: Accept-Encoding
Transfer-Encoding: chunked
```

- HTTP/1.1 表示使用 HTTP1.1 协议标准，200 OK 说明请求成功。
- Date 表示消息产生的日期和时间。
- Content-Type 实体报头域用于指明发送给接收者的实体正文的媒体类型。text/html; charset=utf-8 代表 HTML 文本文档，UTF-8 编码。
- Transfer-Encoding: chunked 表示输出的内容长度不能确定。
- Connection 报头域允许发送用于指定连接的选项。例如指定连接的状态是连续，或者指定“close”选项，通知服务器，在响应完成后，关闭连接。
- Vary 头域指定了一些请求头域，这些请求头域用来决定当缓存中存在一个响应，并且该缓存没有过期失效时，是否被允许利用此响应去回复后续请求而不需要重复验证。
- Cache-Control 用于指定缓存指令，缓存指令是单向的，且是独立的。
- Content-Encoding 实体报头域被用作媒体类型的修饰符，它的值表示了已经被应用到实体正文的附加内容的编码，因而要获得 Content-Type 报头域中所引用的媒体类型，必须采用响应的解码机制。

### 23.3.4 HTTP 请求方式

HTTP 协议请求主要分为 6 中类型，各类型的主要作用如下：

- 1) GET 请求：GET 请求会通过 URL 网址传递信息，可以直接在 URL 中写上要传递的信息，也可以由表单进行传递。如果使用表单进行传递，这表单中的信息会自动转为 URL 地址中的数据，通过 URL 地址传递。
- 2) POST 请求：可以向服务器提交数据，是一种比较主流也比较安全的数据传递方式，比如在登录时，经常使用 POST 请求发送数据。
- 3) PUT 请求：请求服务器存储一个资源，通常要指定存储的位置。



- 4) DELETE 请求：请求服务器删除一个资源。
- 5) HEAD 请求：请求获取对应的 HTTP 报头信息。
- 6) OPTIONS 请求：可以获得当前 URL 所支持的请求类型。

除此之外，还有 TRACE 请求与 CONNECT 请求等，TRACE 请求主要用于测试或诊断。由于用得非常少，所以这里不再提及。

接下来，我们将通过实例讲解 HTTP 协议请求中的 GET 请求和 POST 请求，这两种是最常用的请求方式。

#### a) GET 请求实例分析

有时候在百度上查询一个关键词，我们会打开百度首页，并输入该关键词进行查询，比如输入“Python”，然后按回车键，此时会出现对应的查询结果，观察一下 URL 的变化，如下所示。

```
https://www.baidu.com/s?wd=Python&rsv_spt=1&rsv_iqid=0x8a7b64fb0001c521&issp=1&f=8&rsv_bp=1&rsv_idx=2&ie=utf-8&rqlang=cn&tn=baiduhome_pg&rsv_enter=1&rsv_dl=tb&oq=%25E7%25BD%2591%25E7%25BB%259C%25E7%2588%25AC%25E8%2599%25AB%25E7%259A%2584%25E5%25A5%25BD%25E5%25A4%2584&rsv_t=87f3Kz6nUiM7HXHQZtBpxWSGowc4u4IuzpZOSjJIWWAjkruEp0ZptPDzt43dqEfbewEw&rsv_btype=t&inputT=3469&rsv_pq=b71284a20002123b&rsv_sug3=39&rsv_sug1=31&rsv_sug7=100&rsv_sug2=0&rsv_sug4=69751414&rsv_sug=2
```

可以发现，对应的查询信息是通过 URL 传递的，这里采用的是 HTTP 请求中的 GET 方法。其中字段 ie 的值为 utf-8，代表的是编码信息，而字段 wd 的值为 Python，代表的是用户检索的关键词。

#### b) POST 请求实例分析

POST 请求则是常用在登录、注册等操作的时候，提交的数据放置在实体区内提交。下面展示一个完整的 POST 请求。图 23-3 是为大家提供的一个 POST 表单测试网页，输入用户名和密码后点击登录。使用 F12 打开开发者工具，捕获的请求数据如图 23-4 所示：

用户名：

密 码：

表 23-3 POST 表单测试网页

▼ Form Data    view source    view URL encoded

user_name: admin	请求数据
password: admin	

表 23-4 POST 请求数据

### c) POST 请求与 GET 请求的区别

1、GET 请求，请求的数据会附加在 URL 之后，以?分割 URL 和传输数据，多个参数用&连接。POST 请求：POST 请求会把请求的数据放置在 HTTP 请求包的包体中。因此，GET 请求的数据会暴露在地址栏中，而 POST 请求则不会。

### 2、传输数据的大小

对于 GET 请求方式提交的数据最多只能有 1024 字节，而 POST 请求没有限制。

### 3、安全性问题

使用 GET 请求的时候，参数会显示在地址栏上，而 POST 请求不会。所以，如果这些数据是非敏感数据，那么使用 GET；如果用户输入的数据包含敏感数据，那么还是使用 POST 为好。

## 23.3.5 请求

Requests 是用 python 语言基于 urllib 编写的，采用的是 Apache2 Licensed 开源协议的 HTTP 库。

### 1) Requests 库安装

以“管理员身份运行 cmd”，直接使用 pip 进行安装，语法格式如下：

```
pip install requests
```

### 2) Requests 库的 7 个主要方法

表 23-2 Requests 库的主要方法

方法	说明
requests.request()	构造一个请求，支撑一下个方法的基础方法。
requests.get()	获取 HTML 网页的主要方法，对应 HTTP 的 GET
requests.head()	获取 HTML 网页投信息的方法，对应 HTTP 的 HEAD
requests.post()	向 HTML 网页提交 POST 请求的方法，对应 HTTP 的 POST
requests.put()	向 HTML 网页提交 PUT 请求的方法，对应 HTTP 的 PUT
requests.patch()	向 HTML 网页提交局部修改请求，对应 HTTP 的 PATCH
requests.delete()	向 HTML 网页提交删除请求，对应 HTTP 的 DELETE

### 3) requests.request()

```
requests.request(method,url,**kwargs)
```

其中参数 method 表示请求方式，对应 get/put/post 等 7 种；url 表示获取页面的 url 链接；\*\*kwargs 表示控制访问的参数，均为可选项。

### 4) Requests 库的 get()方法

```
requests.get(url,params=None,**kwargs)
```

其中参数 url 表示获取页面的 url 链接；params 表示 url 中额外参数，字典或字节流格式，

是可选参数；\*\*kwargs 表示 12 个控制访问的参数。

### 23.3.6 响应

Response 对象包含服务器返回的所有信息，分为三部分：响应状态码、响应头、响应内容。

表 23-3 Response 对象的属性

属性	说明
r.status_code	HTTP 请求的返回状态，200 表示连接成功，404 表示失败
r.text	HTTP 响应内容的字符串形式，即，url 对应的页面内容
r.encoding	从 HTTP header 中猜测的相应内容编码方式
r.apparent_encoding	从内容中分析出的相应内容编码方式（备选编码方式）
r.content	HTTP 响应内容的二进制形式
r.encoding	如果 header 中不存在 charset，则认为编码为 ISO-8859-1
r.apparent_encoding	根据网页内容分析出的编码方式可以看作是 r.encoding 的备选

Response 的编码：

- 1) r.encoding：从 HTTP header 中猜测的响应内容的编码方式；如果 header 中不存在 charset，则认为编码为 ISO-8859-1，r.text 根据 r.encoding 显示网页内容。
- 2) r.apparent\_encoding：根据网页内容分析出的编码方式，可以看作 r.encoding 的备选。

#### 【示例 23-1】第一个简单的爬虫

```
import requests
r = requests.get('http://www.baidu.com')
print('content:',r.content)
print('text:',r.text)
```

执行结果如下所示：

```
<!DOCTYPE html>
<!--STATUS      OK--><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=referrer><link      rel=stylesheet      type=text/css
href=http://s1.bdstatic.com/r/www/cache/bdorz/baidu.min.css><title>ç™³/4ã°|ä,€ä,ç¼4Œä½  â±çŸ
¥é “</title></head> <body link=#0000cc> <div id=wrapper> <div id=head> <div
class=head_wrapper> <div class=s_form> <div class=s_form_wrapper> <div id=lg> <img
hidefocus=true src=//www.baidu.com/img/bd_logo1.png width=270 height=129> </div> <form
id=form name=f action=//www.baidu.com/s class=fm> <input type=hidden name=bdorz_come
value=1> <input type=hidden name=ie value=utf-8> <input type=hidden name=f value=8> <input
```



```

type=hidden name=rsv_bp value=1> <input type=hidden name=rsv_idx value=1> <input
type=hidden name=tn value=baidu><span class="bg s_ipt_wr"><input id=kw name=wd
class=s_ipt value maxlength=255 autocomplete=off autofocus></span><span class="bg
s_btn_wr"><input type=submit id=su value=ç™³¼äº|ä,€ä,< class="bg s_btn"></span> </form>
</div> </div> <div id=u1> <a href=http://news.baidu.com name=tj_trnews
class=mnav>æ-°é—»</a> <a href=http://www.hao123.com name=tj_trhao123
class=mnav>hao123</a> <a href=http://map.baidu.com name=tj_trmap class=mnav>âœ°ã¾¼</a>
<a href=http://v.baidu.com name=tj_trvideo class=mnav>è§†é¢‘</a> <a
href=http://tieba.baidu.com name=tj_trtieba class=mnav>è’â§</a> <noscript> <a
href=http://www.baidu.com/bdorz/login.gif?login&amp;tpl=mn&amp;u=http%3A%2F%2Fwww.
baidu.com%2f%3fbdorz_come%3d1 name=tj_login class=lb>ç™»½•</a> </noscript>
<script>document.write('<a href="http://www.baidu.com/bdorz/login.gif?login&tpl=mn&u='+
encodeURIComponent(window.location.href+ (window.location.search == "" ? "?" : "&")+
"bdorz_come=1")+ "" name="tj_login" class="lb">ç™»½•</a>');</script> <a
href=//www.baidu.com/more/ name=tj_briicon class=bri style="display:
block;">æ’âðšä°§â“ ¤ </a> </div> </div> </div> <div id=ftCon> <div id=ftConw> <p id=lh> <a
href=http://home.baidu.com>â...³ä°žç™³¼äº|</a> <a href=http://ir.baidu.com>About Baidu</a>
</p> <p id=cp>id=cp>&copy;2017&nbsp;Baidu&nbsp;<a href=http://www.baidu.com/duty/>ä½²ç”ç™³¼äº|â‰‘ â½…ë–»</a>&nbsp;<a
href=http://jianyi.baidu.com/
class=cp-feedback>æ,, è§ ¤ å¤ ¤ é¦‘</a>&nbsp;&nbsp;ICPè¬ 030173å¤ ·&nbsp; <img
src=//www.baidu.com/img/g.s.gif> </p> </div> </div> </div> </body> </html>

```

上述代码是获取百度首页的 HTML 代码，首先导入包 requests，之后获取网页内容。r 是 requests 的 Response 回复对象，调用该对象的 text 属性可以获取网页内容代码。但发现获取的内容乱码，需要修改编码，示例如下。

### 【示例 23-2】Response 设置编码

```
import requests

r = requests.get('http://www.baidu.com')

print('content:',r.content)

r.encoding='utf-8'

print('new text: ',r.text)
```

执行结果如下所示:

new text: <!DOCTYPE html>

```

<!--STATUS      OK--><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=referrer><link      rel=stylesheet      type=text/css
href=http://s1.bdstatic.com/r/www/cache/bdorz/baidu.min.css><title> 百度一下，你就知道
</title></head>  <body link=#0000cc>  <div id=wrapper>  <div id=head>  <div
class=head_wrapper>  <div class=s_form>  <div class=s_form_wrapper>  <div id=lg>  <img
hidefocus=true src=//www.baidu.com/img/bd_logo1.png width=270 height=129>  </div>  <form
id=form name=f action=//www.baidu.com/s class=fm>  <input type=hidden name=bdorz_come
value=1>  <input type=hidden name=ie value=utf-8>  <input type=hidden name=f value=8>  <input
type=hidden name=rsv_bp value=1>  <input type=hidden name=rsv_idx value=1>  <input
type=hidden name=tn value=baidu><span class="bg_s ipt_wr"><input id=kwd name=wd
class=s ipt value maxlength=255 autocomplete=off autofocus></span><span class="bg
s_btn_wr"><input type=submit id=submit value= 百度一下 class="bg_s_btn"></span>  </form>
</div>  </div>  <div id=u1>  <a href=http://news.baidu.com name=tj_trnews class=mnav>新闻</a>
<a href=http://www.hao123.com name=tj_trhao123 class=mnav>hao123</a>  <a
href=http://map.baidu.com name=tj_trmap class=mnav>地图</a>  <a href=http://v.baidu.com
name=tj_trvideo class=mnav>视频</a>  <a href=http://tieba.baidu.com name=tj_trtieba
class=mnav>
      贴      吧      </a>      <noscript>      <a
href=http://www.baidu.com/bdorz/login.gif?login&tpl=mn&u=http%3A%2F%2Fwww.
baidu.com%2F%3Fbdorz_come%3D1 name=tj_login class=lb>登 录 </a>  </noscript>
<script>document.write('<a href="http://www.baidu.com/bdorz/login.gif?login&tpl=mn&u='+
encodeURIComponent(window.location.href+ (window.location.search == "" ? "?" : "&")+
"bdorz_come=1")+ "" name="tj_login" class="lb">登 录 </a>');</script>  <a
href=//www.baidu.com/more/ name=tj_briicon class=bri style="display: block;">更多产品</a>
</div>  </div>  </div>  <div id=ftCon>  <div id=ftConw>  <p id=lh>  <a
href=http://home.baidu.com>关于百度</a>  <a href=http://ir.baidu.com>About Baidu</a>  </p>
<p id=cp>&copy;2017&nbsp;Baidu&nbsp;<a href=http://www.baidu.com/duty/>使用百度前必
读</a>&nbsp;<a href=http://jianyi.baidu.com/ class=cp-feedback>意见反馈</a>&nbsp;京 ICP
证 030173 号&nbsp;<img src=//www.baidu.com/img/gs.gif>  </p>  </div>  </div>  </div>  </body>
</html>

```

## 23.4 Beautiful Soup 库

BeautifulSoup4 是一个 HTML/XML 的解析器，主要的功能是解析和提取 HTML/XML 的数据。和 lxml 库一样。

lxml 只会局部遍历，而 BeautifulSoup4 是基于 HTML DOM 的，会加载整个文档，解析整个 DOM 树，因此内存开销比较大，性能比较低。

BeautifulSoup4 用来解析 HTML 比较简单，API 使用非常人性化，支持 CSS 选择器，是 Python 标准库中的 HTML 解析器，也支持 lxml 解析器。

### 23.4.1 BeautifulSoup 简介

### 23.4.2 BeautifulSoup 安装

目前，Beautiful Soup 的最新版本是 4.x 版本，之前的版本已经停止开发，这里推荐使用 pip 来安装，安装命令如下：

```
pip install beautifulsoup4
```

#### 【示例 23-3】查看 BeautifulSoup 安装是否成功

```
from bs4 import BeautifulSoup
soup = BeautifulSoup('<p>Hello</p>','html.parser')
print(soup.p.string)
```

执行结果如图 23-5 所示：



图 23-5 示例 23-3 运行效果图

#### 注意：

- 这里虽然安装的是 beautifulsoup4 这个包，但是引入的时候却是 bs4，因为这个包源代码本身的库文件名称就是 bs4，所以安装完成后，这个库文件就被移入到本机 Python3 的 lib 库里，识别到的库文件就叫作 bs4。
- 因此，包本身的名称和我们使用时导入包名称并不一定是一致的。

### 23.4.3 BeautifulSoup 库解析器

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）。表 23-4 列出了 BeautifulSoup 支持的解析器。

表 23-4 BeautifulSoup 库解析器

解析器	使用方法	条件	优势
bs4 的 HTML 解析器	BeautifulSoup(mk,'html.parser')	安装 bs4 库	Python 的内置标准库、执行速度适中、文档容错能力强
lxml 的 HTML 解析	BeautifulSoup(mk,'lxml')	pip install lxml	速度快、文档容错能力强

器			
lxml 的 XML 解析器	BeautifulSoup(mk,'xml')	pip install lxml	速度快、唯一支持 XML 的解析器
html5lib 的解析器	BeautifulSoup(mk,'html5lib')	pip install 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

【示例 23-4】初始化 BeautifulSoup 使用 lxml，把第二个参数改为 lxml

```
from bs4 import BeautifulSoup
bs = BeautifulSoup('<p>Python</p>','lxml')
print(bs.p.string)
```

执行结果如图 23-6 所示：

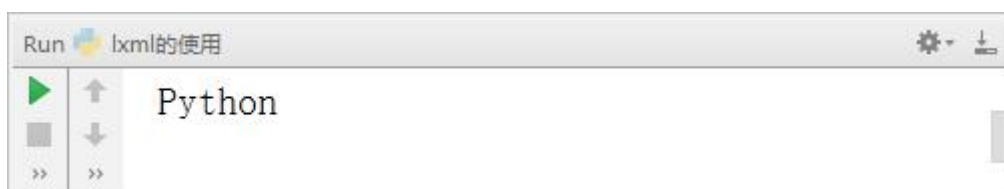


图 23-6 示例 23-4 运行效果图

## 23.4.4 BeautifulSoup

### 1) 基本用法

表 23-5 BeautifulSoup 类的基本元素

基本元素	说明
Tag	标签，基本信息组织单元，分别用<和</>标明开头和结尾
Name	标签的名字，<p></p>的名字是 'p'，格式：<tag>.name
Attributes	标签的属性，字典形式组织，格式：<tag>.attrs
NavigableString	标签内非属性字符串，<...>中字符串，格式：<tag>.string
Comment	标签内字符串的注释部分，一种特殊的 Comment 类型

【示例 23-5】获取 title 节点，查看它的类型

```
html = """
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title" name="dromouse"><b>The Dormouse's story</b></p>
<p class="story">Once upon a time there were three little sisters; and their names were
```

```
<a href="http://example.com/elsie" class="sister" id="link1"><!-- Elsie --></a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>
<p class="story">...</p>
</body>
</html>
"""

from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.prettify())
print(soup.title.string)
```

执行结果如下所示：

```
<html>
<head>
  <title>
    The Dormouse's story
  </title>
</head>
<body>
  <p class="title" name="dromouse">
    <b>
      The Dormouse's story
    </b>
  </p>
  <p class="story">
    Once upon a time there were three little sisters; and their names were
    <a class="sister" href="http://example.com/elsie" id="link1">
      <!-- Elsie -->
    </a>
    ,
    <a class="sister" href="http://example.com/lacie" id="link2">
      Lacie
    </a>
```



```

and
<a class="sister" href="http://example.com/tillie" id="link3">
    Tillie
</a>
;
and they lived at the bottom of a well.
</p>
<p class="story">
    ...
</p>
</body>
</html>

```

The Dormouse's story

上述示例首先声明变量 `html`，它是一个 HTML 字符串。接着将它当作第一个参数传给 `BeautifulSoup` 对象，该对象的第二个参数为解析器的类型（这里使用 `lxml`），此时就完成了 `BeautifulSoup` 对象的初始化。

接着调用 `soup` 的各个方法和属性解析这串 HTML 代码了。

调用 `prettify()` 方法。可以把要解析的字符串以标准的缩进格式输出。这里需要注意的是，输出结果里面包含 `body` 和 `html` 节点，也就是说对于不标准的 HTML 字符串 `BeautifulSoup`，可以自动更正格式。

调用 `soup.title.string`，输出 HTML 中 `title` 节点的文本内容。所以，`soup.title` 可以选出 HTML 中的 `title` 节点，再调用 `string` 属性就可以得到里面的文本了。

### 【示例 23-6】选择元素

```

from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.head)
print(soup.p)

```

执行结果如图 23-7 所示：

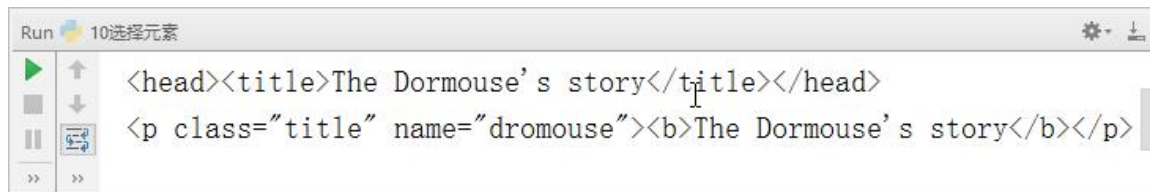


图 23-7 示例 23-6 运行效果图

从上述示例运行结果可以看到，获取 head 节点的结果是节点加其内部的所有内容。最后，选择了 p 节点。不过这次情况比较特殊，我们发现结果是第一个 p 节点的内容，后面的几个 p 节点并没有选到。也就是说，当有多个节点时，这种选择方式只会选择到第一个匹配的节点，其他的后面节点都会忽略。

#### 【示例 23-7】调用 name 属性获取节点的名称

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.title.name)
```

执行结果如图 23-8 所示：

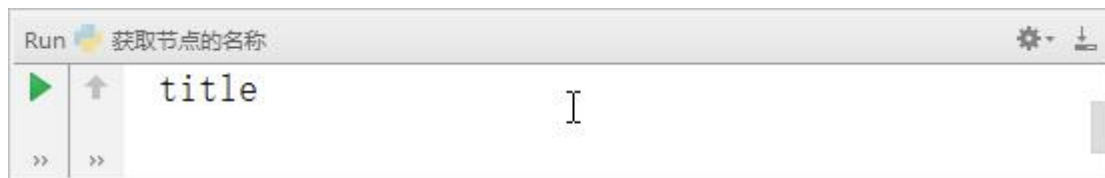


图 23-8 示例 23-7 运行效果图

#### 【示例 23-8】调用 attrs 获取所有属性

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.p.attrs)
print(soup.p.attrs['name'])
```

执行结果如图 23-9 所示：

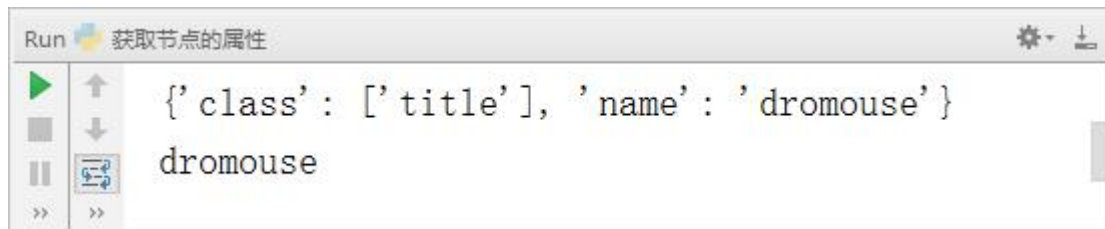


图 23-9 示例 23-8 运行效果图

从上述运行结果可以看到，attrs 的返回结果是字典形式，它把选择节点的所有属性和属性值组合成一个字典。如果要获取 name 属性，就相当于从字典中获取某个键值，只需要用中括号加属性名就可以了。例如，要获取 name 属性，就可以通过 attrs['name']来得到。

#### 【示例 23-9】简单获取属性的方式

```
from bs4 import BeautifulSoup
```

```
soup = BeautifulSoup(html, 'lxml')
print(soup.p.attrs)
print(soup.p.attrs['name'])
print('简单方式获取属性: ')
print(soup.p['name'])
print(soup.p['class'])
```

执行结果如图 23-10 所示:

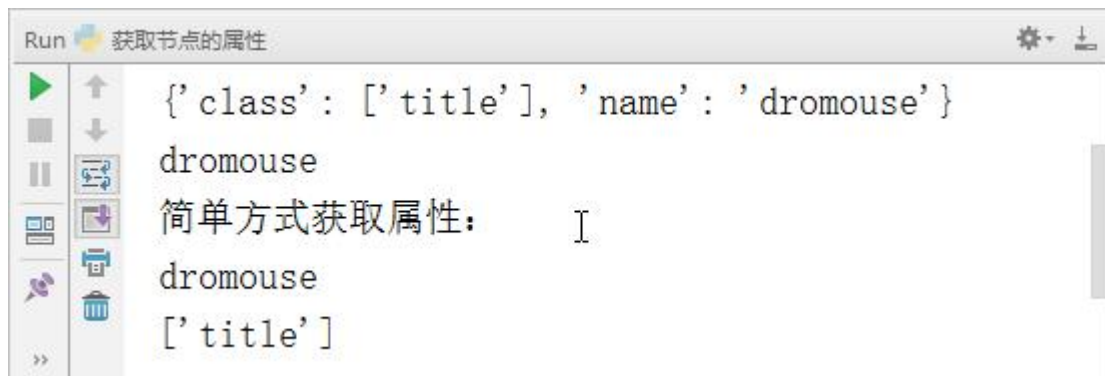


图 23-10 示例 23-9 运行效果图

这里需要注意的是, 获取属性有的返回结果是字符串, 有的返回结果是字符串组成的列表。比如, name 属性的值是唯一的, 返回的结果就是单个字符串。而对于 class, 一个节点元素可能有多个 class, 所以返回的是列表。

#### 【示例 23-10】调用 string 属性获取节点元素包含的文本内容

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.p.string)
```

执行结果如图 23-11 所示:

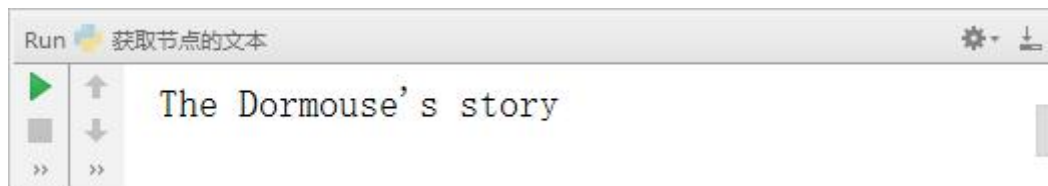


图 23-11 示例 23-10 运行效果图

#### 【示例 23-11】嵌套选择

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.head.title)
```

```
print(type(soup.head.title))
print(soup.head.title.string)
```

执行结果如图 23-12 所示：

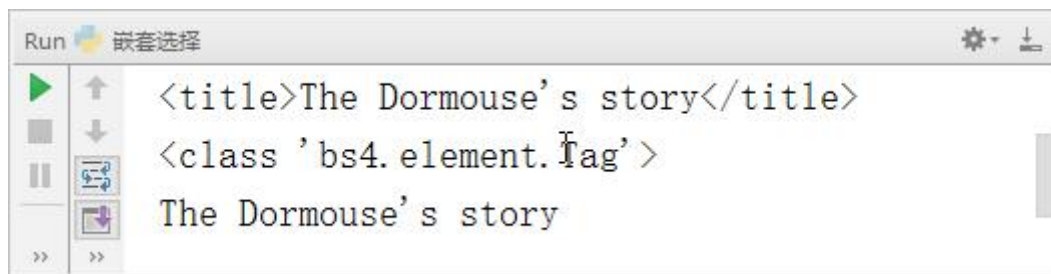


图 23-12 示例 23-11 运行效果图

从上述示例运行结果可以看到，调用 head 之后再次调用 title 可以选择 title 节点元素。输出了它的类型可以看到，它仍然是 bs4.element.Tag 类型。也就是说，我们在 Tag 类型的基础上再次选择得到的依然还是 Tag 类型，每次返回的结果都相同。

#### 【示例 23-12】调用 children 属性，获取它的直接子节点

```
html = """
<html>
  <head>
    <title>The Dormouse's story</title>
  </head>
  <body>
    <p class="story">
      Once upon a time there were three little sisters; and their names were
      <a href="http://example.com/elsie" class="sister" id="link1">
        <span>Elsie</span>
      </a>
      <a href="http://example.com/lacie" class="sister" id="link2">Lacie</a>
      and
      <a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>
      and they lived at the bottom of a well.
    </p>
    <p class="story">...</p>
  </body>
</html>
"""

from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
```

```
print(soup.p.children)
for i, child in enumerate(soup.p.children):
    print(i, child)
```

执行结果如下所示：

```
<list_iterator object at 0x00000246ACC46400>
0
    Once upon a time there were three little sisters; and their names were

1 <a class="sister" href="http://example.com/elsie" id="link1">
  <span>Elsie</span>
</a>
2

3 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>
4
    and

5 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
6
    and they lived at the bottom of a well.
```

从上述示例运行结果可以看到，调用 `children` 属性，返回结果是生成器类型。用 `for` 循环输出相应内容。

### 【示例 23-13】调用 `parent` 属性，获取某个节点元素的父节点

```
html = """
<html>
  <head>
    <title>The Dormouse's story</title>
  </head>
  <body>
    <p class="story">
      Once upon a time there were three little sisters; and their names were
      <a href="http://example.com/elsie" class="sister" id="link1">
```



```

        <span>Elsie</span>
    </a>
</p>
<p class="story">...</p>
"""
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.a.parent)

```

执行结果如下所示：

```

<p class="story">
    Once upon a time there were three little sisters; and their names were
    <a class="sister" href="http://example.com/elsie" id="link1">
        <span>Elsie</span>
    </a>
</p>

```

从上述示例运行结果可以看到，我们选择的是第一个 a 节点的父节点元素，它的父节点是 p 节点，输出结果便是 p 节点及其内部的内容。

需要注意的是，这里输出的仅仅是 a 节点的直接父节点，而没有再向外寻找父节点的祖先节点。如果想获取所有的祖先节点，可以调用 parents 属性。

#### 【示例 23-14】调用 parents 属性，获取某个节点元素的祖先节点

```

html = """
<html>
    <body>
        <p class="story">
            <a href="http://example.com/elsie" class="sister" id="link1">
                <span>Elsie</span>
            </a>
        </p>
    </body>
</html>
"""
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(type(soup.a.parents))
print(list(enumerate(soup.a.parents)))

```

执行结果如下所示：

```
<class 'generator'>
[(0, <p class="story">
  <a class="sister" href="http://example.com/elsie" id="link1">
    <span>Elsie</span>
  </a>
</p>), (1, <body>
  <p class="story">
    <a class="sister" href="http://example.com/elsie" id="link1">
      <span>Elsie</span>
    </a>
  </p>
</body>), (2, <html>
  <body>
    <p class="story">
      <a class="sister" href="http://example.com/elsie" id="link1">
        <span>Elsie</span>
      </a>
    </p>
  </body></html>), (3, <html>
  <body>
    <p class="story">
      <a class="sister" href="http://example.com/elsie" id="link1">
        <span>Elsie</span>
      </a>
    </p>
  </body></html>)]
```

**【示例 23-15】**调用 `next_sibling` 和 `previous_sibling` 分别获取节点的下一个和上一个兄弟元素

```
html = ""
<html>
  <body>
```

```
<p class="story">
    Once upon a time there were three little sisters; and their names were
    <a href="http://example.com/elsie" class="sister" id="link1">
        <span>Elsie</span>
    </a>
    Hello
    <a href="http://example.com/lacie" class="sister" id="link2">Lacie</a>
    and
    <a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>
    and they lived at the bottom of a well.
</p>

"""
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print('Next Sibling', soup.a.next_sibling)
print('Prev Sibling', soup.a.previous_sibling)
```

执行结果如下所示：

```
Next Sibling
    Hello

Prev Sibling
    Once upon a time there were three little sisters; and their names were
```

## 2) 方法选择器

上面所讲的选择方法都是通过属性来选择的，这种方法非常快，但是如果进行比较复杂的选择的话，它就比较烦琐，不够灵活了。Beautiful Soup 还提供了一些查询方法，例如 `find_all()` 和 `find()` 等。

`find_all` 是查询所有符合条件的元素。给它传入一些属性或文本，就可以得到符合条件的元素，它的功能十分强大，语法格式如下：

```
find_all(name , attrs , recursive , text , **kwargs)
```

**【示例 23-16】** `find_all` 方法传入 `name` 参数，根据节点名来查询元素

```
html="""
```

```
<div class="panel">
  <div class="panel-heading">
    <h4>Hello</h4>
  </div>
  <div class="panel-body">
    <ul class="list" id="list-1">
      <li class="element">Foo</li>
      <li class="element">Bar</li>
      <li class="element">Jay</li>
    </ul>
    <ul class="list list-small" id="list-2">
      <li class="element">Foo</li>
      <li class="element">Bar</li>
    </ul>
  </div>
</div>
'''
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
for ul in soup.find_all(name='ul'):
    print(ul.find_all(name='li'))
    for li in ul.find_all(name='li'):
        print(li.string)
```

执行结果如图 23-13 所示：



图 23-13 示例 23-16 运行效果图

从上述示例可以看到，调用 `find_all()` 方法，`name` 参数值为 `ul`。返回结果是查询到的所有 `ul` 节点列表类型，长度为 2，每个元素依然都是 `bs4.element.Tag` 类型。因为都是 `Tag` 类型，所以依然可以进行嵌套查询。再继续查询其内部的 `li` 节点，返回结果是 `li` 节点列表类型，遍历列表中的每个 `li`，获取它的文本。

**【示例 23-17】**`find_all` 方法传入 `attrs` 参数，根据属性来查询

```
html=""
<div class="panel">
  <div class="panel-heading">
    <h4>Hello</h4>
  </div>
  <div class="panel-body">
    <ul class="list" id="list-1" name="elements">
      <li class="element">Foo</li>
      <li class="element">Bar</li>
      <li class="element">Jay</li>
    </ul>
    <ul class="list list-small" id="list-2">
      <li class="element">Foo</li>
      <li class="element">Bar</li>
    </ul>
  </div>
</div>
```



```

        </ul>
    </div>
</div>
'''

from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.find_all(attrs={'id': 'list-1'}))
print(soup.find_all(attrs={'name': 'elements'}))

```

执行结果如图 23-14 所示：

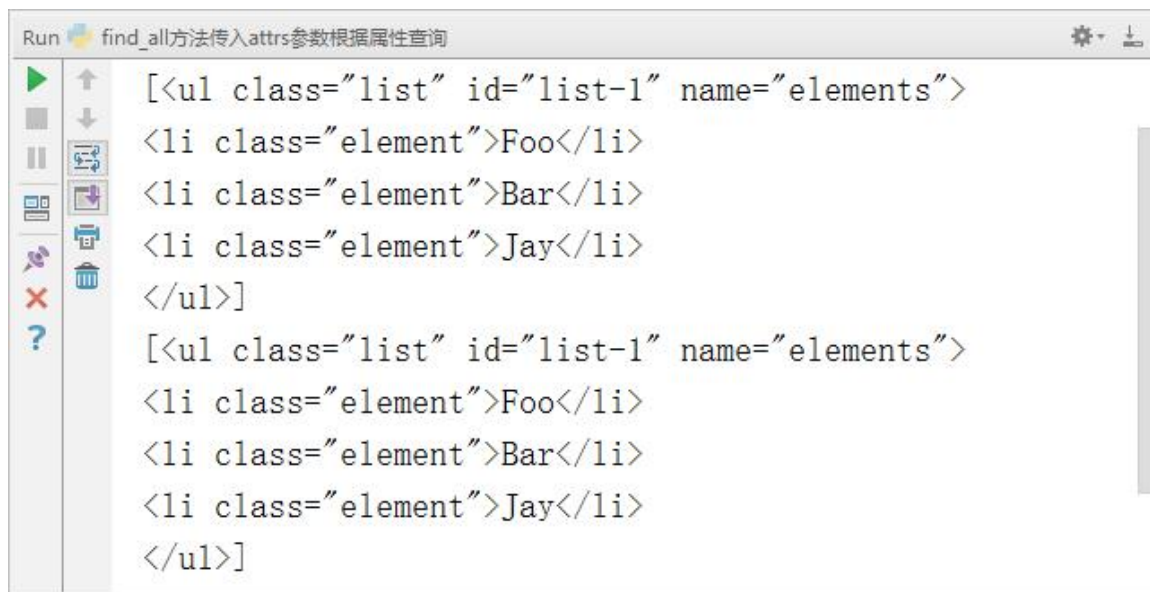


图 23-14 示例 23-17 运行效果图

从上述示例可以看到，传入 `attrs` 参数，参数的类型是字典类型。比如，要查询 `id` 为 `list-1` 的节点，可以传入 `attrs={'id': 'list-1'}` 的查询条件，得到的结果是列表形式，包含的内容就是符合 `id` 为 `list-1` 的所有节点。符合条件的元素个数是 1，长度为 1 的列表。

对于一些常用的属性，比如 `id` 和 `class` 等，可以不用 `attrs` 来传递。比如，要查询 `id` 为 `list-1` 的节点，可以直接传入 `id` 这个参数。示例如下：

#### 【示例 23-18】find\_all 方法根据属性来查询

```

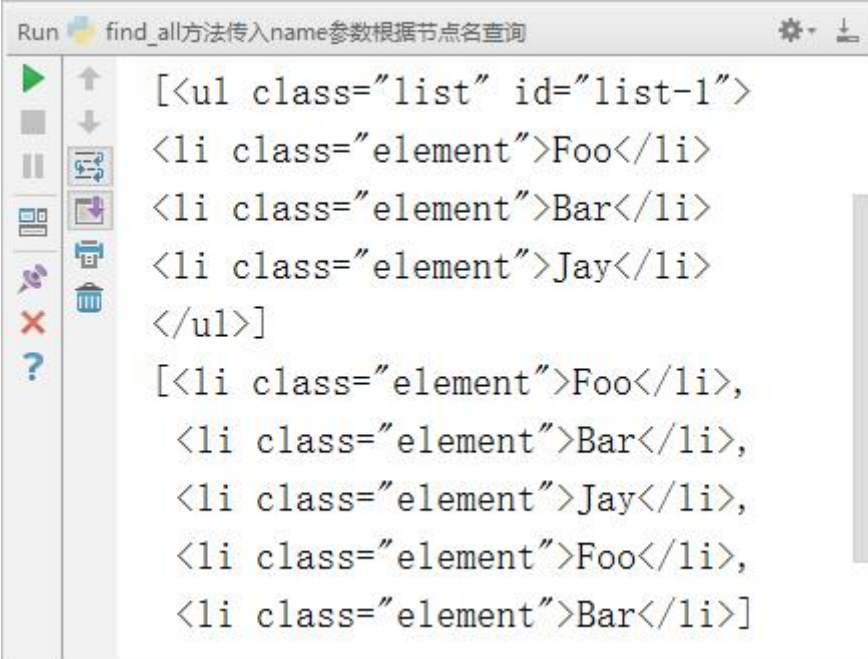
html=""
<div class="panel">
  <div class="panel-heading">
    <h4>Hello</h4>
  </div>

```

```
<div class="panel-body">
    <ul class="list" id="list-1">
        <li class="element">Foo</li>
        <li class="element">Bar</li>
        <li class="element">Jay</li>
    </ul>
    <ul class="list list-small" id="list-2">
        <li class="element">Foo</li>
        <li class="element">Bar</li>
    </ul>
</div>
</div>
'''

from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.find_all(id='list-1'))
print(soup.find_all(class_='element'))
```

执行结果如图 23-15 所示：



```
Run find_all方法传入name参数根据节点名查询

[<ul class="list" id="list-1">
<li class="element">Foo</li>
<li class="element">Bar</li>
<li class="element">Jay</li>
</ul>]
[<li class="element">Foo</li>,
<li class="element">Bar</li>,
<li class="element">Jay</li>,
<li class="element">Foo</li>,
<li class="element">Bar</li>]
```

图 23-15 示例 23-18 运行效果图

上述示例直接传入 `id='list-1'`，就可以查询 `id` 为 `list-1` 的节点元素了。而对于 `class` 来说，由于 `class` 在 Python 里是一个关键字，所以后面需要加一个下划线，即 `class_='element'`，返回的结果依然还是 Tag 组成的列表。

`find_all` 方法传入 `text` 参数可用来匹配节点的文本，传入的形式可以是字符串，可以是正则表达式对象。

**【示例 23-19】** `find_all` 方法根据文本来查询

```
import re
html=""
<div class="panel">
    <div class="panel-body">
        <a>Hello, this is a link</a>
        <a>Hello, this is a link, too</a>
    </div>
</div>
"""

from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.find_all(text=re.compile('link')))
```

执行结果如图 23-16 所示：

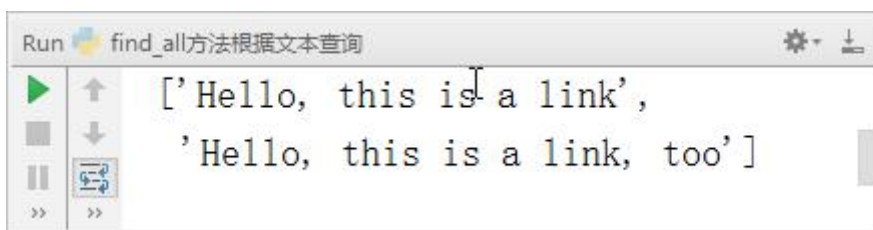


图 23-16 示例 23-19 运行效果图

上述示例有两个 `a` 节点，其内部包含文本信息。这里在 `find_all()` 方法中传入 `text` 参数，该参数为正则表达式对象，结果返回所有匹配正则表达式的节点文本组成的列表。

除了 `find_all()` 方法，还有 `find()` 方法，不过后者返回的是单个元素，也就是第一个匹配的元素，而前者返回的是所有匹配的元素组成的列表。

**【示例 23-20】** `find` 方法查询第一个匹配的元素

```
html=""
<div class="panel">
    <div class="panel-heading">
        <h4>Hello</h4>
    </div>
```

```
<div class="panel-body">
    <ul class="list" id="list-1">
        <li class="element">Foo</li>
        <li class="element">Bar</li>
        <li class="element">Jay</li>
    </ul>
    <ul class="list list-small" id="list-2">
        <li class="element">Foo</li>
        <li class="element">Bar</li>
    </ul>
</div>
</div>
'''
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.find(name='ul'))
print(type(soup.find(name='ul')))
print(soup.find(class_='list'))
```

执行结果如图 23-17 所示：

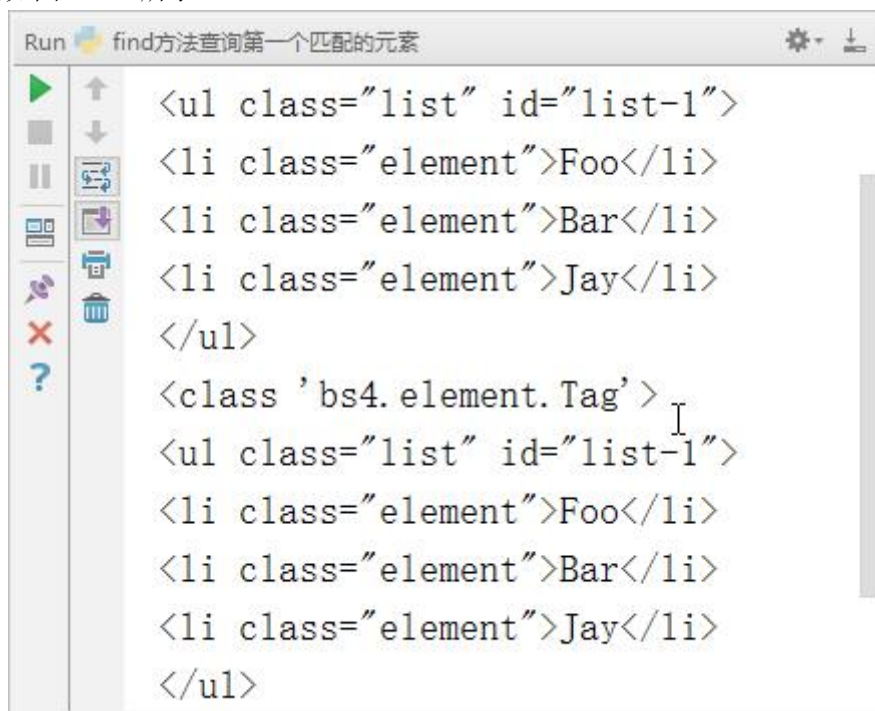


图 23-17 示例 23-20 运行效果图

上述示例使用 find 方法返回结果不再是列表形式，而是第一个匹配的节点元素，类型

依然是 Tag 类型。

### 3) CSS 选择器

Beautiful Soup 还提供了另外一种选择器，那就是 CSS 选择器。使用 CSS 选择器时，只需要调用 `select()` 方法，传入相应的 CSS 选择器即可。

#### 【示例 23-21】CSS 选择器的使用

```
html=""
<div class="panel">
  <div class="panel-heading">
    <h4>Hello</h4>
  </div>
  <div class="panel-body">
    <ul class="list" id="list-1">
      <li class="element">Foo</li>
      <li class="element">Bar</li>
      <li class="element">Jay</li>
    </ul>
    <ul class="list list-small" id="list-2">
      <li class="element">Foo</li>
      <li class="element">Bar</li>
    </ul>
  </div>
</div>
"""

from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.select('.panel .panel-heading'))
print(soup.select('ul li'))
lis = soup.select('#list-2 .element')
for l in lis:
    print('Get Text:', l.get_text())
    print('String:', l.string)
```

执行结果如图 23-18 所示：



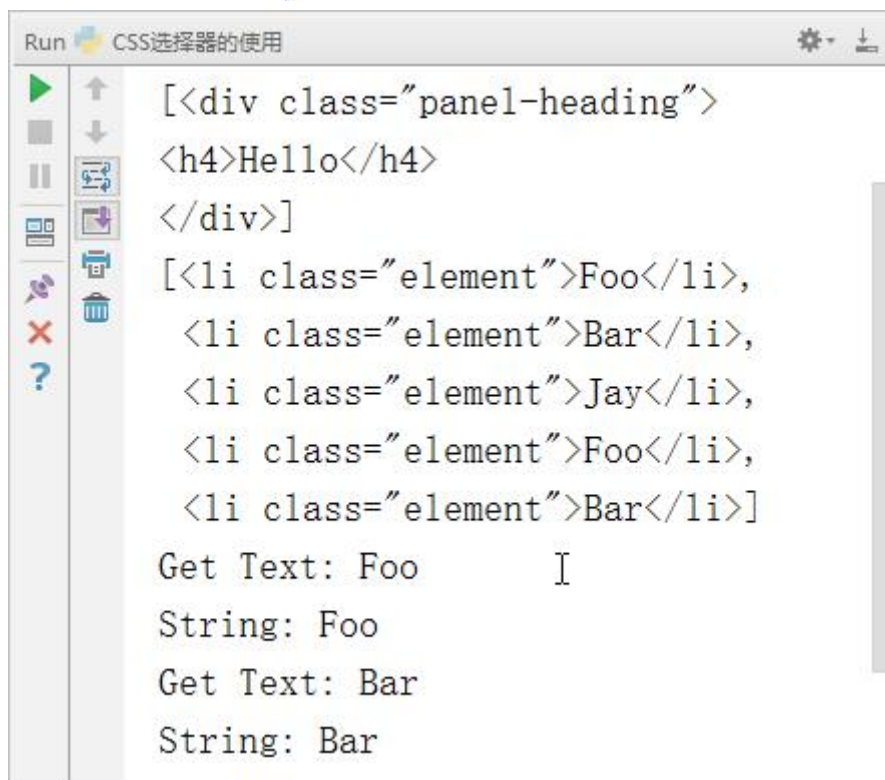


图 23-18 示例 23-21 运行效果图

上述示例，用了 3 次 CSS 选择器，返回的结果均是符合 CSS 选择器的节点组成的列表。例如，`select('ul li')` 则是选择所有 `ul` 节点下面的所有 `li` 节点，结果便是所有的 `li` 节点组成的列表。要获取文本，当然也可以用前面所讲的 `string` 属性。此外，还有一个方法，那就是 `get_text()`。

## 23.5 XPath

### 23.5.1 XPath 简介

XPath，全称 XML Path Language，即 XML 路径语言，它是一门在 XML 文档中查找信息的语言。最初是用来搜寻 XML 文档的，但同样适用于 HTML 文档的搜索。所以在做爬虫时完全可以使用 XPath 做相应的信息抽取。

XPath 的选择功能十分强大，它提供了非常简洁明了的路径选择表达式。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，几乎所有想要定位的节点都可以用 XPath 来选择。官方文档：<https://www.w3.org/TR/xpath/>。

### 23.5.2 安装

```
pip install lxml
```

### 23.5.3 XPath 语法

XPath 使用路径表达式来选取 XML 文档中的节点或节点集。节点是沿着路径（path）或者步（steps）来选取的。下面列举了一些常用的路径表达式进行节点的选取，如表 23-6

所示。

表 23-6 路径表达式

表达式	描述
nodename	选取此节点的所有子节点
/	从当前节点选区直接子节点
//	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

XPath 的常用匹配规则如下所示：

`//title[@lang='eng']`

上述 XPath 匹配规则代表的是选择所有名称为 title，同时属性 lang 的值为 eng 的节点，后面会通过 Python 的 lxml 库，利用 XPath 进行 HTML 的解析。

上面路径都是选取了所有符合条件的节点，是否能选取某个特定的节点或者包含某一个指定值得节点，这就需要用到谓语句。谓语句被嵌在方括号中，用来查找某个特定的节点或者包含某个指定的值的节点。

表 23-7 谓语句示例

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

表 23-8 选取未知节点

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。

表 23-9 通配符选取示例

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。

通过在路径表达式中使用 “|” 运算符，您可以选取若干个路径。

表 23-10 选取若干路径示例

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。

## 23.5.4 XPath 使用

首先给出 XPath\_test.html，实例分析就按照这个文档来进行，文档内容如下：

```
<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>
```

**【示例 23-22】** 用以 // 开头的 XPath 规则来选取所有符合要求的节点

```
from lxml import etree
html = etree.parse('./XPath_test.html', etree.HTMLParser())
result = html.xpath('//*[*)
print(result)
```

执行结果如图 23-19 所示：

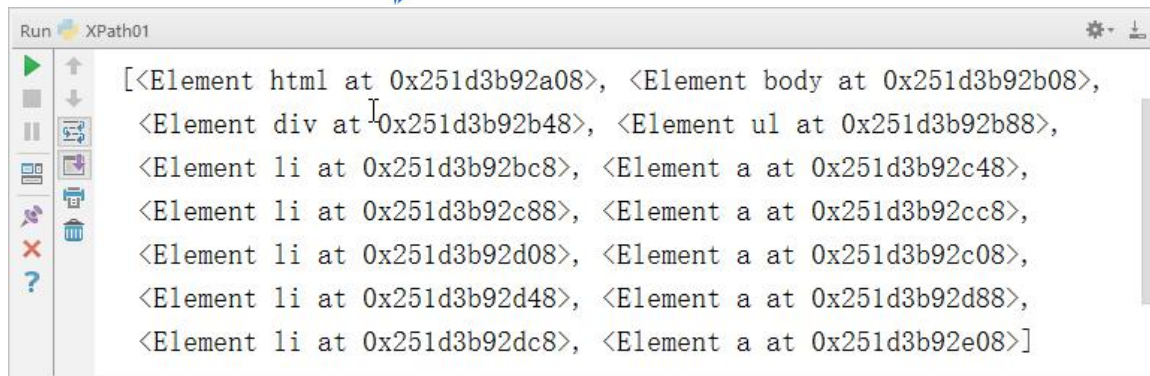


图 23-19 示例 23-22 运行效果图

上述示例中 \* 代表匹配所有节点，返回的结果是一个列表，每个元素都是一个 Element 类型，后跟节点名称。也可以指定匹配的节点名称，示例如下。

### 【示例 23-23】 指定匹配的节点名称

```
from lxml import etree
html = etree.parse('./XPath_test.html', etree.HTMLParser())
result = html.xpath('//li')
print(result)
```

执行结果如图 23-20 所示：

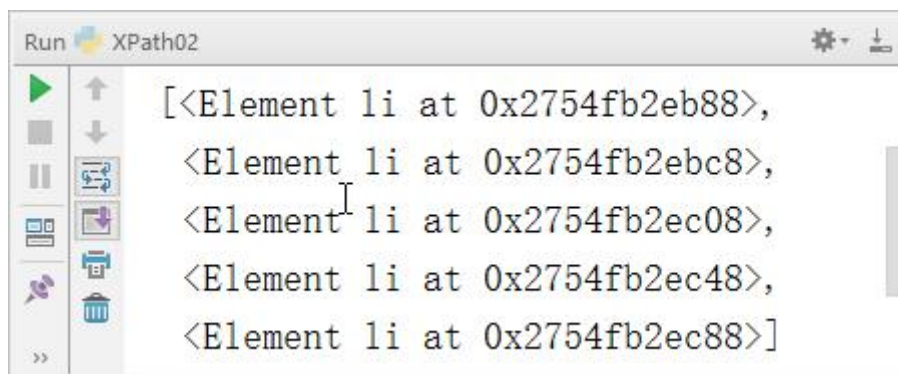


图 23-20 示例 23-23 运行效果图

通过 / 或 // 即可查找元素的子节点或子孙节点。选择 li 节点的所有直接 a 子节点。

### 【示例 23-24】 选择 li 节点的所有直接 a 子节点

```
from lxml import etree
html = etree.parse('./XPath_test.html', etree.HTMLParser())
result = html.xpath('//li/a')
print(result)
```

执行结果如图 23-21 所示：

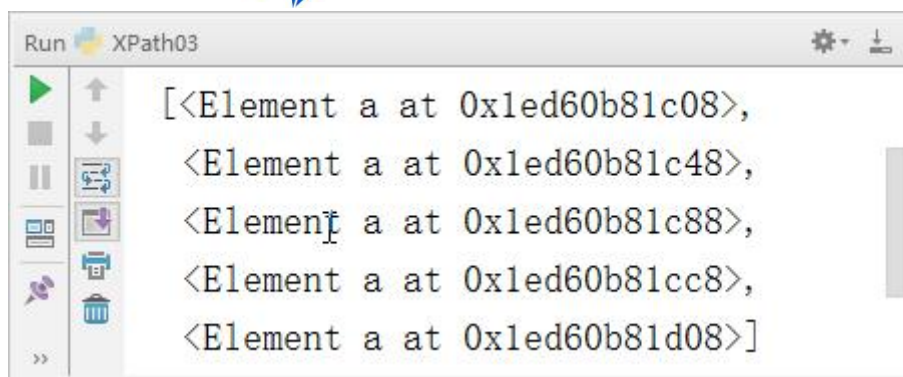


图 23-21 示例 23-24 运行效果图

【示例 23-25】 用@符号进行属性过滤

```
from lxml import etree
html = etree.parse('./XPath_test.html', etree.HTMLParser())
result = html.xpath('//li[@class="item-inactive"]')
print(result)
```

执行结果如图 23-22 所示：

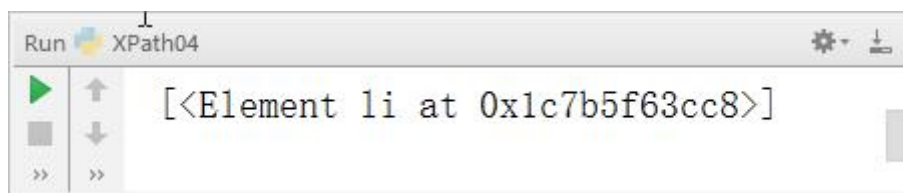


图 23-22 示例 23-25 运行效果图

【示例 23-26】 调用 text()方法获取文本

```
from lxml import etree
html = etree.parse('./XPath_test.html', etree.HTMLParser())
result = html.xpath('//li[@class="item-0"]/a/text()')
print(result)
```

执行结果如图 23-23 所示：

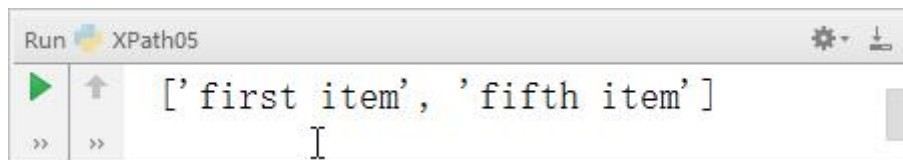


图 23-23 示例 23-26 运行效果图

【示例 23-27】 @符号跟属性名直接获取节点的属性值

```
from lxml import etree
```

```
html = etree.parse('./XPath_test.html', etree.HTMLParser())
result = html.xpath('//li/a/@href')
print(result)
```

执行结果如图 23-24 所示：

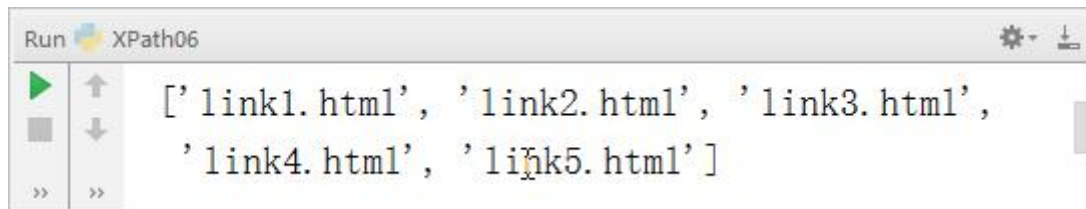


图 23-24 示例 23-27 运行效果图

### 【示例 23-28】多属性匹配

```
from lxml import etree
text = """
<li class="li li-first" name="item"><a href="link.html">first item</a></li>
"""
html = etree.HTML(text)
result = html.xpath('//li[contains(@class, "li") and @name="item"]/a/text()')
print(result)
```

执行结果如图 23-25 所示：

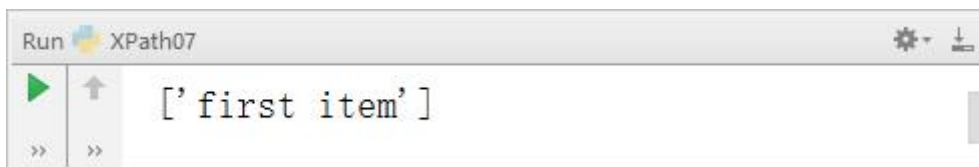


图 23-25 示例 23-28 运行效果图



## 习题

### 一、简答题

1. 什么是爬虫。
2. 爬虫的基本流程。
3. HTTP 请求方式有哪些。
4. GET 请求和 POST 请求有什么区别。

### 二、编码题

1. 爬取一张图片并保存。
2. 使用 BeautifulSoup4 解析器，将招聘网页上的职位名称、职位类别、招聘人数、工作地点、时间、以及每个职位详情的点击链接存储出来。
3. 从网址 “<https://bj.zu.ke.com/zufang>”，获取到房源的朝向，大小，租金，名字，租赁方式及厅室情况等相关信息。
4. 用 XPath 来做一个简单的爬虫，爬取某个贴吧里的所有帖子，并且将该这个帖子里每个楼层发布的图片下载到本地。
5. 用 XPath 爬取一下豆瓣音乐专区数据。