第 11 章

项目实战：爬虫程序

大数据时代，数据采集是进行数据分析的一项重要前提工作，单靠人工获取数据效率低下、成本极高。为了提高数据采集的效率，爬虫应运而生。爬虫又称网络机器人，可以代替人工自动从互联网中采集和整理数据。本章通过一个简单的爬虫案例来介绍爬虫程序的相关知识。另外，读者可通过学习 Scrapy 框架掌握借助第三方库实现爬虫程序的设计流程，并通过上机实践巩固爬虫框架 Scrapy 的典型应用。

11.1 爬虫概述

爬虫又称网络蜘蛛、网络蚂蚁、网络机器人等，它可以不受人工干涉，自动按照既定规则浏览互联网中的信息。通常把这些既定规则称为爬虫算法。使用 Python 语言可以方便地实现这些算法，编写爬虫程序，完成信息的自动获取。

提示

常见的网络爬虫主要有百度公司的 Baiduspider、360 公司的 360Spider、搜狗公司的 Sogouspider、微软公司的 Bingbot 等。

大数据时代，面对海量的互联网数据，如何才能高效地获取有用的数据呢？爬虫便是最好的解决方法，人们可以根据自己的需求制定相应的规则，实现对应的爬虫程序，进而从互联网获取需要的信息。未来，随着数据量的爆炸性增长，爬虫将越来越重要。因此，本章通过爬虫程序实战案例让读者对这一网络数据获取利器有一定的认识，并能够熟练地根据需求定制规则，完成爬虫程序的编程实现。

11.1.1　准备工作

在爬取一个站点之前，通常需要对该站点的规模和结构进行大致的了解。站点自身的 robots 和 sitemap 文件都能够为了解其构成提供有效的帮助。

1. robots 文件

一般情况下，大部分站点都会定义自己的 robots 文件，以便引导爬虫按照自己的意图爬取相关数据。因此，在爬取站点之前应检查 robots 文件，了解该站点的限制条件，做到有的放矢，提升爬虫成功获取数据的概率。同时，还可以通过此文件了解站点结构的相关信息，从而有针对性地设计爬虫程序，完成数据的获取工作。

2. sitemap 文件

站点提供的 sitemap 文件呈现了整个站点的组成结构。它能够帮助爬虫根据用户的需求定位需要的内容，而无须爬取每一个页面。然而，站点地图有可能存在更新不及时或不完整的情况，因而在使用 sitemap 时需要格外谨慎。

3. 估算站点规模

目标站点的大小会影响爬取的效率。通常可以通过百度搜索引擎的 site 关键字过滤域名结果，从而获取百度爬虫对目标站点的统计信息。

提示

在百度首页搜索框中输入“site：目标站点域名”，即可获取相关统计信息。

11.1.2　爬虫类型

网络爬虫按照实现的技术和结构可以分为通用网络爬虫、聚焦网络爬虫、增量式网络爬虫和深层网络爬虫。实际的网络爬虫系统通常由这几种爬虫组合而成。下面对常见的这 4 种爬虫类型进行简单介绍。

（1）通用网络爬虫：别名全网爬虫，主要由初始 URL 集合、URL 队列、页面爬行模块、页面分析模块、页面数据库以及链接过滤模块构成。该类型的爬虫获取的目标资源在整个互联网中，其目标数据量庞大，爬行的范围广泛。正是由于这种特性，此类爬虫对性能的要求非常高。它主要用于大型的搜索引擎中；例如，百度搜索，具有比较高的应用价值。

（2）聚焦网络爬虫：别名主题网络爬虫，主要由初始 URL 集合、URL 队列、页面爬行模块、页面分析模块、页面数据库、链接过滤模块、内容评价模块以及链接评价模块构成，是一种按照预先定义好的主题有选择地进行网页爬取的网络爬虫。与通用网络爬虫相比，它的目标数据只和预定义的主题相关，爬行的范围相对固定，对网络带宽资源及服务器资源要求较低，主要用于特定信息的获取。

（3）增量式网络爬虫：主要由本地页面 URL 集、待爬行 URL 集、本地页面集、爬行模块、排序模块以及更新模块构成，是指对已下载网页采取增量式更新和只爬行新产生的或者已经发生变化的网页的爬虫，它能够在一定程度上保证所爬行的是尽可能新的页面。与周期性爬行和刷新页面的网络爬虫相比，增量式爬虫只会在需要的时候爬行新产生或发

生更新的页面，并不重新下载没有发生变化的页面，可有效减少数据下载量，及时更新已爬行的网页，减少时间和空间上的耗费，但是增加了爬行算法的复杂度和实现难度。

（4）深层网络爬虫：主要由 URL 列表、LVS 列表、爬行控制器、解析器、LVS 控制器、表单分析器、表单处理器以及响应分析器构成。其中 LVS 是指标签/数值集合，用来表示填充表单的数据源。深层网络爬虫是一种用于爬取互联网中深层页面的爬虫程序。与通用网络爬虫相比，深层页面的爬取需要想办法自动填写对应的表单；因此，深层网络爬虫的核心在于表单的填写。

提示

互联网中页面按存取方式可分为两类：表层页面和深层页面。表层页面是指不需要提交表单，使用静态链接能够到达的静态页面；深层页面则是指需要提交表单，使用动态链接才能够到达的动态生成页面；例如，网络中的信息查询页面。

11.1.3 爬虫原理

不同的爬虫程序，其实现原理也不尽相同，但也有许多相似之处，即共性。基于此，本节用一个通用的网络爬虫结构来说明爬虫的基本工作流程，如图 11.1 所示。其基本工作流程如下。

（1）按照预定主题，选取一部分精心挑选的种子 URL。

（2）将种子 URL 放入待抓取 URL 队列中。

（3）从待抓取 URL 队列中依次读取种子 URL，解析其对应的 DNS，并得到对应的主机 IP，将 URL 对应的网页下载下来，并存入已下载网页的数据库中，随后将已访问的种子 URL 出队，放入已抓取 URL 队列中。

（4）分析已抓取队列中的 URL，从已下载网页数据中分析出其他的 URL，并和已抓取的 URL 进行重复性比较。最后，将去重过的 URL 放入待抓取 URL 队列中，重复步骤（3）和（4）的操作，直至待抓取 URL 队列为空。

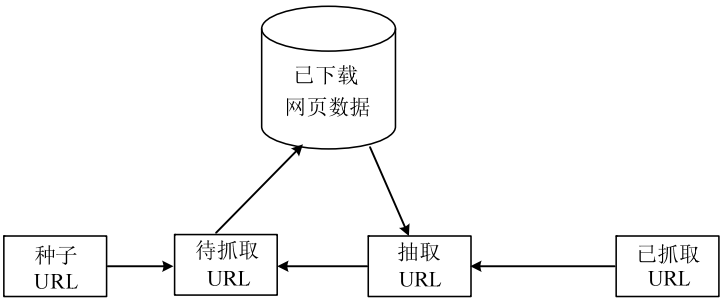

图 11.1 爬虫工作流程

11.2 爬虫三大库

众所周知，Python 语言属于胶水语言，可扩展性强。用 Python 编写爬虫程序的最大

好处就是其本身有很多实用的第三方库，免去了开发人员自己实现相应功能的环节。Python 爬虫有 3 个比较实用的库：Requests、BeautifulSoup 和 lxml，为编写爬虫程序提供了必不可少的技术支持，下面逐一介绍。

11.2.1　Requests 库

用 Requests 实现 HTTP 请求非常简单，操作也相当人性化。因此，Python 中常用 Requests 来实现 HTTP 请求过程，这也是在 Python 爬虫开发中最常用的方式。

1. Requests 库的安装

Requests 库是第三方模块，需要额外进行安装。使用 Requests 库需要先进行安装，常用的安装方式有两种。

- 第 1 种方式：使用 pip 进行安装。命令为 pip install requests。
- 第 2 种方式：直接到官网下载最新发行版，然后解压缩文件夹，运行 setup.py 即可。

这里采用第 2 种安装方式，具体操作步骤如下。

（1）打开 Requests 官方网站，如图 11.2 所示。

图 11.2　Requsets 库官方网站

（2）单击 Python 标签，打开下载界面，选择 Navigation 下的 Download files 菜单，选择 requests.tar.gz 压缩包，单击下载，如图 11.3 所示。

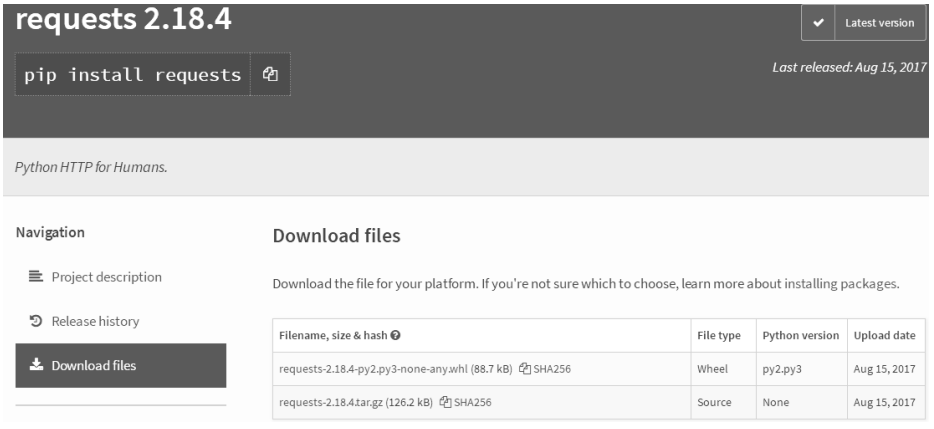

图 11.3　Requests 库下载页面

（3）解压 requests.tar.gz 安装包，通过命令运行安装包中的 setup.py 文件。

提示

Requests 库安装成功与否检查方法：在 Python 的 Shell 中输入 import requests，如果不报错，则安装成功，如图 11.4 所示。

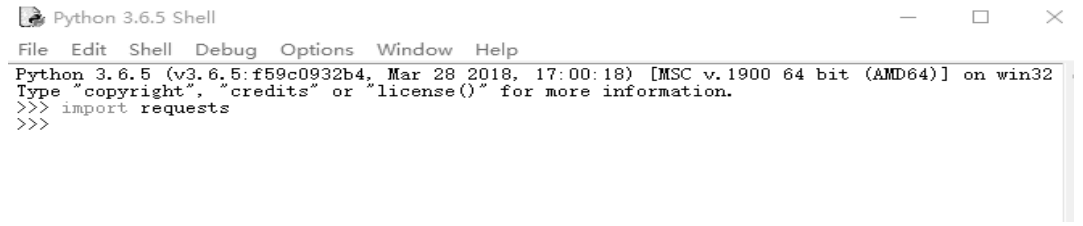

图 11.4　检查 Requests 库安装成功与否

2. Requests 库的使用

Requests 库的主要方法有 7 种，现简单介绍如下。

（1）Requests 库的 get()方法主要用于获取 HTML 网页，相当于 HTTP 的 GET。其返回对象 response 的常用属性如表 11.1 所示，可通过这些属性获取要访问域名的基本信息。

表 11.1　response 的常用属性

属　　性	说　　明
r.status_code	HTTP 请求的返回状态，200 表示链接成功，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 中猜测的响应内容的编码方式
r.apparent_encoding	从内容中分析出的响应内容的编码方式
r.content	HTTP 响应内容的二进制形式

下面演示如何通过 get()方法返回对象 response 来获取 www.baidu.com 域名的基本信息。示例代码如下。

```
>>> import requests
>>> r = requests.get("http://www.baidu.com")
>>> print(r.status_code)
200
>>> print(r.text)
<!DOCTYPE html>
<!..STATUSOK..><html>
【鉴于代码篇幅较长，此处省略 www.baidu.com 页面代码】
  </html>
>>> print(r.encoding)
ISO.8859.1
>>> print(r.apparent_encoding)
utf.8
>>> print(r.content)
```

```
b'<!DOCTYPE html>\r\n<!..STATUS OK..><html>
【鉴于代码篇幅较长，此处省略 www.baidu.com 页面代码】
</html>\r\n'
```

如上代码所示，requests 对象的 get()方法返回 response 对象 r，通过 print()函数打印 r 的属性值，便可获取网站域名的相关信息。

（2）Requests 库的 head()方法主要用于获取 HTML 网页头信息，相当于 HTTP 的 HEAD。例如，抓取百度首页的头部信息。示例代码如下。

```
>>> import requests
>>> r = requests.head('http://www.baidu.com')
>>> r.headers
{'Cache.Control': 'private, no.cache, no.store, proxy.revalidate, no.transform', 'Connection': 'Keep.Alive', 'Content.Encoding': 'gzip', 'Content.Type': 'text/html', 'Date': 'Tue, 19 Jun 2018 05:35:22 GMT', 'Last.Modified': 'Mon, 13 Jun 2016 02:50:40 GMT', 'Pragma': 'no.cache', 'Server': 'bfe/1.0.8.18'}
>>> r.text
''
>>>
```

如上代码所示，首先导入 requests 包；然后调用 head()方法，返回 response 对象 r，最后通过引用 r 的相关属性，显示对应网址的相关信息。

（3）Requests 库的 post()方法主要用于向 HTTP 网页提交 POST 请求，相当于 HTTP 的 POST。

例如，给指定的 URL 地址 http://httpbin.org 用 post()方法添加 sendinfo 信息。示例代码如下。

```
>>> import requests
>>> sendinfo = {'name':'lily','sex':'female'}
>>> r = requests.post('http://httpbin.org/post',data = sendinfo)
>>> print(r.text)
{"args":{},"data":"","files":{},"form":{"name":"lily","sex":"female"},"headers":{"Accept":"*/*","Accept.Encoding":"gzip, deflate","Connection":"close","Content.Length":"20","Content.Type":"application/x.www.form.urlencoded","Host":"httpbin.org","User.Agent":"python.requests/2.18.4"},"json":null,"origin":"113.123.80.176","url":"http://httpbin.org/post"}

>>>
```

由以上代码的交互式输出结果不难发现，字典 sendinfo 以 form 表单的形式被发送给 response 对象。也可以直接向 url 地址发送字符串，示例代码如下。

```
>>> r = requests.post('http://httpbin.org/post',data = 'i am string')
>>> print(r.text)
{"args":{},"data":"i  am  string","files":{},"form":{},"headers":{"Accept":"*/*","Accept.Encoding":"gzip, deflate","Connection":"close","Content.Length":"11","Host":"httpbin.org","User.Agent":"python.requests/2.18.4"},"json":null,"origin":"113.123.80.176","url":"http://httpbin.org/post"}

>>>
```

从代码的交互式输出结果不难发现，字符串以 data:iam string 键-值对被保存下来。

（4）Requests 库的 put()方法主要用于向 HTML 网页提交 PUT 请求，相当于 HTTP 的 PUT。

例如，给指定的 url 地址用 put()方法添加字典 sendinfo 信息。示例代码如下。

```
>>> sendinfo = {'name':'lily','sex':'female'}
>>> r = requests.put('http://httpbin.org/put',data = sendinfo)
>>> print(r.text)
{"args":{},"data":"","files":{},"form":{"name":"lily","sex":"female"},"headers":{"Accept":"*/*","Accept.
Encoding":"gzip,  deflate","Connection":"close","Content.Length":"20","Content.Type":"application/
x.www.form.urlencoded","Host":"httpbin.org","User.Agent":"python.requests/2.18.4"},"json":null,"or
igin":"113.123.80.176","url":"http://httpbin.org/put"}
```

同样，由上述交互式输出结果可以看出，put()也是用 form 表单的方式存储自定义的字典，并返回 response 对象。

（5）Requests 库的 patch()方法主要用于向 HTML 网页提交局部修改请求，相当于 HTTP 的 PATCH。

例如，用 patch()方法修改刚才用 put()方法添加的字典 sendinfo。示例代码如下。

```
>>> sendinfo = {'name':'berry','sex':'female'}
>>> r = requests.patch('http://httpbin.org/patch',data = sendinfo)
>>> print(r.text)
{"args":{},"data":"","files":{},"form":{"name":"berry","sex":"female"},"headers":{"Accept":"*/*","Accept
.Encoding":"gzip,  deflate","Connection":"close","Content.Length":"21","Content.Type":"application/
x.www.form.urlencoded","Host":"httpbin.org","User.Agent":"python.requests/2.18.4"},"json":null,"
origin":"113.123.80.176","url":"http://httpbin.org/patch"}
```

由以上代码可知，通过 patch()成功修改字典中的 name 值。

（6）Requests 库的 delete()方法主要用于向 HTML 页面提交删除请求，相当于 HTTP 的 DELETE。

例如，删除刚才用 patch()方法修改的 sendinfo 字典。示例代码如下。

```
>>> r=requests.delete("https://httpbin.org/delete")
>>> print(r.text)
{"args":{},"data":"","files":{},"form":{},"headers":{"Accept":"*/*","Accept.Encoding":"gzip, deflate",
"Connection":"close","Content.Length":"0","Host":"httpbin.org","User.Agent":"python.requests/
2.18.4"},"json":null,"origin":"113.123.80.176","url":"https://httpbin.org/delete"}
```

由以上代码执行结果可以看出，form 表单的内容为空，说明删除成功。

（7）Requests 库的 request()方法主要用来构造一个请求，支撑以上各个基础方法。

通常使用下面的格式来完成该方法的调用。

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

其中，method 是指请求方式，对应上面所讲的 get()、put()、post()等方法；url 代表目标页面的 URL 链接地址；**kwargs 代表控制访问参数，共 13 个；例如，params 参数，代表字典或字节序列，可作为参数增加到 url 中。示例代码如下。

```
>>> sendinfo = {'name':'lily','sex':'female'}
>>> r = requests.request('PUT','http://httpbin.org/put',data = sendinfo)
>>> print(r.text)
{"args":{},"data":"","files":{},"form":{"name":"lily","sex":"female"},"headers":{"Accept":"*/*","Accept.
Encoding":"gzip,  deflate","Connection":"close","Content.Length":"20","Content.Type":"application/
x.www.form.urlencoded","Host":"httpbin.org","User.Agent":"python.requests/2.18.4"},"json":null,
"origin":"113.123.80.176", "url":"http://httpbin.org/put"}
```

由以上代码可以看出，使用 request()方法可以包装出通用的接口，来模拟 requests 对象常用方法的功能。另外，request()方法的**kwargs 参数属于可选参数。其他参数具体的使用方法可以参照 requests 文档进行学习。鉴于篇幅关系，这里不再赘述。

3. 爬取定向网页的通用代码框架

基于 Requests 定向网页爬虫程序的模板框架如下。

```
import requests
def getHTMLText(url):
try :
r = requests.get( url, timeout =30)
r.raise_for_status()                    # 如果状态码不是 200，引发 HTTPError 异常
r.encoding= r.apparent_encoding
return r.text
except:
return”产生异常”
if__name__==”__main__”:                 # 限定 getHTMLText()只在所定义的文件中执行
url =”http://www.baidu.com”
print(getHTMLText(url))
```

建议读者按照统一的编程风格编写程序，以提高通用代码的可读性。

11.2.2　BeautifulSoup 库

BeautifulSoup 库是用 Python 语言编写的一个 HTML/XML 的解释器，可以很好地处理不规范的标记并生成剖析树（parse tree），其本身提供了简单又常用的导航、搜索以及修改剖析树的操作，利用它可以大大缩短编程时间。本节主要介绍如何使用该库处理不规范的标记，按照指定格式输出对应的文档。

1. BeautifulSoup 的安装

安装 BeautifulSoup 库与安装 Requests 库的方式类似，有两种方式：第 1 种是使用 pip install 命令进行安装。第 2 种是到官网下载，运行 Setup.py 文件。这里采用第 1 种安装方式，在 DOS 窗口中运行安装命令，代码如下。

```
pip install beautifulsoup4
```

安装过程如图 11.5 所示。

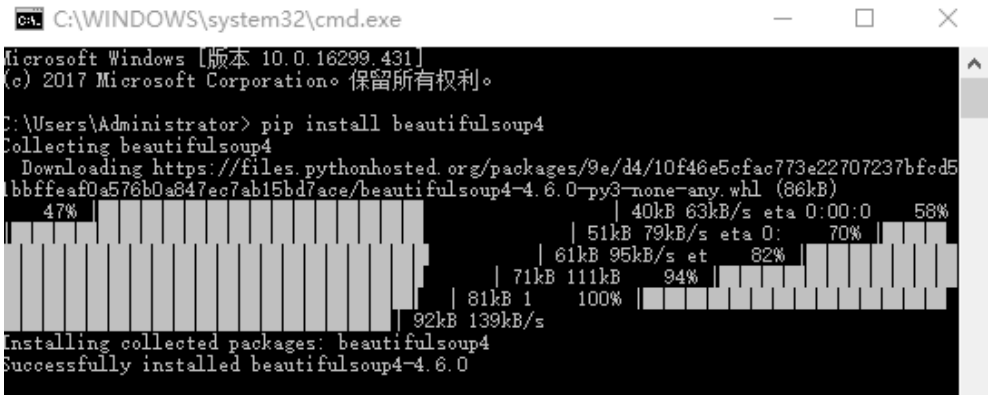

图 11.5　安装 BeautifulSoup

这里安装的是 beautifulsoup 4.4.6.0 版本。

2. Beautiful Soup 的基本操作

1）创建 BeautifulSoup 对象

创建 BeautifulSoup 对象时，首先要导入其对应的 bs4 库，格式如下：

```
from bs4 import BeautifulSoup
```

下面通过一个简单的例子来演示该库的使用。

首先，创建一个用来完成演示的 HTML 字符串，其定义了一个标准的 HTML 文档，也就是 BeautfiulSoup 对象的数据源，代码如下。

```
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>
"""
```

接下来，创建 BeautifulSoup 对象，代码如下。

```
soup = BeautifulSoup(html,"lxml")
```

另外，也可以用本地存在的 HTML 文件（例如，名为 index.html 的文件）来创建 soup 对象，其格式如下。

```
soup = BeautifulSoup(open('index.html'))
```

这里，需要注意路径问题。

下面通过 soup 对象的格式化函数格式化输出 soup 对象中的内容，代码如下。

```
print(soup.prettify())
```

输出结果如下。

```
<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>
```

以上便是 soup 对象格式化输出的方式，prettify()函数是通过 soup 对象分析 HTML 文档的第一步，一定要熟练掌握该函数的用法。

2）BeautifulSoup 库的对象

通常，BeautifulSoup 库用于将一个复杂的 HTML 文档转换成一个复杂的树形结构，每个节点都是一个 Python 对象。根据功能，将 BeautifulSoup 库的对象划分为 4 类，具体介绍如下。

（1）Tag。Tag 相当于 HTML 中的一个标签，代码如下。

```
# 提取 Tag 标签
>>> print(soup.title)
<title>The Dormouse's story</title>
```

```
>>> print (type(soup.title))
<class 'bs4.element.Tag'>
```

关于 Tag 标签，有两个重要的属性：name 和 attrs，使用方法分别如下：

① name：每个 Tag 对象的 name 就是标签本身的名称；例如，超链接标签 a 的 name，代码如下。

```
>>> print(soup.a.name)
A
>>>
```

② attrs：每个 Tag 对象的 attrs 就是一个字典，包含标签的全部属性；例如，超链接 a 的 attrs 属性如下。

```
>>> print(soup.a.attrs)
{'href': 'http://example.com/elsie', 'class': ['sister'], 'id': 'link1'}
>>>
```

（2）NavigableString。使用 Tag 对象得到标签的内容。那么，要想获取标签内部的文字该怎么办呢？很简单，用 .string 即可，其类型为 NavigableString，示例代码如下。

```
>>> print (soup.p.string)
The Dormouse's story
>>> print(type(soup.p.string))
<class 'bs4.element.NavigableString'>
>>>
```

（3）BeautifulSoup。BeautifulSoup 对象表示的是一个文档的全部内容。大部分时候，可以把它当作 Tag 对象，它是一个特殊的 Tag，可以分别获取它的名称、类型以及属性。接本节例子输出如下。

```
>>> print(soup.name)
[document]
>>> print(type(soup.name))
<class 'str'>
>>> print (soup.attrs)
{}
>>>
```

（4）Comment。Comment 对象是一个特殊类型的 NavigableString 对象，其实际输出的内容仍然不包括注释符号，但是如果不好好处理，可能会给文本处理带来意想不到的麻烦。从本节开始定义的数据源 HTML 字符串中，找一个带有注释的 a 标签，代码如下。

```
<a href="http://example.com/elsie" class="sister" id="link1"><!.. Elsie ..></a>,
```

进行相关操作，代码如下。

```
>>> print(soup.a)
<a class="sister" href="http://example.com/elsie" id="link1"><!.. Elsie ..></a>
>>> print(soup.a.string)
 Elsie
```

```
>>> print(type(soup.a.string))
<class 'bs4.element.Comment'>
>>>
```

由以上代码运行结果可知，其注释输出只显示其中的内容。

3. 遍历文档

下面重点学习搜索文档树的 find_all()方法。参照 BeautifulSoup 库的帮助文档，其 find_all()方法的标准格式如下。

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

现通过示例简单介绍其带有指定参数的使用方法。

1）name 参数

name 参数可以查找所有名字为 name 的 Tag，自动忽略字符串对象；例如，搜索文档中 name 为超链接 a 的 Tag，代码如下。

```
>>> print (soup.find_all('a'))
[<a class="sister" href="http://example.com/elsie" id="link1"><!.. Elsie ..></a>, <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>, <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
>>>
```

另外，name 参数也可以是列表、正则表达式或方法。

（1）name 参数为列表，例如，下例中的['a','b']，代码如下。

```
>>> print(soup.find_all(['a','b']))
[<b>The Dormouse's story</b>, <a class="sister" href="http://example.com/elsie" id="link1"><!.. Elsie ..></a>, <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>, <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
>>>
```

（2）name 参数为正则表达式；例如，下例中的'^b'，以 b 开头的标签都能够找到，代码如下。

```
>>> import re
>>> print(soup.find_all(re.compile('^b')))
[<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>, <b>The Dormouse's story</b>]
>>>
```

（3）传递函数。如果没有合适的过滤器，那么还可以定义一个方法，方法只接受一个元素参数，如果这个方法返回 True，表示当前元素匹配并且被找到，如果不是，则返回 False。

下面方法校验了当前元素，如果包含 class 属性却不包含 id 属性，那么将返回 True，代码如下。

```
>>> def has_class_but_no_id(tag):
    return tag.has_attr('class') and not tag.has_attr('id')
>>> soup.find_all(has_class_but_no_id)

[<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>]
>>>
```

2）keyword 参数

如果一个指定名字的参数不是搜索内置的参数名，搜索时会把该参数当作指定名字 Tag 的属性来搜索，如果包含一个名字为 id 的参数，BeautifulSoup 会搜索每个 Tag 的 id 属性。示例代码如下。

```
>>> soup.find_all(id='link2')
[<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>]
```

Soup 对象会把 link2 当作标签搜索所有 Tag 的 id 属性。

如果传入 href 参数，BeautifulSoup 会搜索每个 Tag 的 href 属性。示例代码如下。

```
>>> soup.find_all(href=re.compile("elsie"))
[<a class="sister" href="http://example.com/elsie" id="link1"><!.. Elsie ..></a>]
```

如果使用多个指定名字的参数，可以同时过滤 Tag 的多个属性。示例代码如下。

```
>>> soup.find_all(href=re.compile("elsie"), id='link1')
[<a class="sister" href="http://example.com/elsie" id="link1"><!.. Elsie ..></a>]
>>>
```

Soup 对象只保留同时具有指定限定符的标签。

3）text 参数

通过 text 参数可以搜索文档中的字符串内容。与 name 参数的可选值一样，text 参数接受字符串、正则表达式、列表等参数。

示例代码如下。

```
>>> soup.find_all(text="Elsie")
[]
>>> soup.find_all(text=["Tillie", "Elsie", "Lacie"])
['Lacie', 'Tillie']
>>> soup.find_all(text=re.compile("Dormouse"))
["The Dormouse's story", "The Dormouse's story"]
>>> soup.find_all(text=True)
["The Dormouse's story", '\n', '\n', "The Dormouse's story", '\n', 'Once upon a time there were three little sisters; and their names were\n', ' Elsie ', ',\n', 'Lacie', ' and\n', 'Tillie', ';\nand they lived at the bottom of a well.', '\n', '...', '\n']
>>>
```

4）limit 参数

find_all() 方法返回全部的搜索结果，如果文档树很大，那么搜索会很慢。不需要全部结果，可以使用 limit 参数限制返回结果的数量，效果与 SQL 中的 limit 关键字类似，当搜索到的结果数量达到 limit 的限制时，就停止搜索，返回结果；例如，本节开始定义的 HTML 字符串文档，文档树中有 3 个 Tag 符合搜索条件，但结果只返回了两个，这是由于限制了返回数量。示例代码如下。

```
>>> soup.find_all("a", limit=2)
[<a class="sister" href="http://example.com/elsie" id="link1"><!.. Elsie ..></a>, <a class="sister"
href="http://example.com/lacie" id="link2">Lacie</a>]
>>>
```

5）recursive 参数

通常，调用 Tag 的 find_all()方法时，BeautifulSoup 会检索当前 Tag 的所有子孙节点，如果只想搜索 Tag 的直接子节点，可以使用 recursive=False。示例代码如下。

```
>>> soup.html.find_all("title")
[<title>The Dormouse's story</title>]
>>> soup.html.find_all("title", recursive=False)
[]
>>>
```

上述代码显示了是否使用 recursive 参数的区别。另外，find 还有许多衍生的方法，这里就不再一一叙述，如有需要，可参阅 BeautifulSoup 的帮助文档。

11.2.3　lxml 库

前面已经学习了 Requests 和 BeautifulSoup 库的相关操作，下面学习另一种高效的网页解析的库——lxml。lxml 是 Python 语言中和 XML 以及 HTML 工作的功能中最丰富和最容易使用的库。它是 Python 爬虫的常用库，是基于 libxml2 这一 XML 解析库的 Python 封装，速度比 BeautifulSoup 快。

1. lxml 库的安装

lxml 库的安装方法也有两种。这里使用 pip install 命令进行安装，代码如下。

```
pip  install lxml
```

安装成功界面如图 11.6 所示。

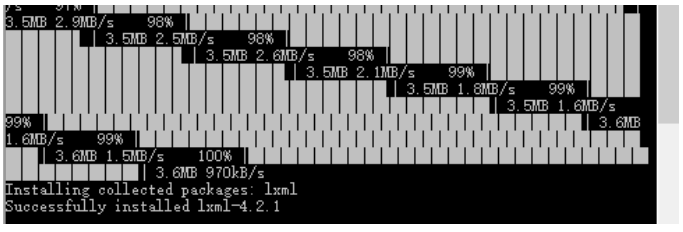

图 11.6　Lxml 安装成功界面

这里安装的是 lxml 4.2.1 版本。

2. lxml 的基本操作

1）解析 html 文档

下面通过一个简单的例子了解一下 lxml 库是如何解析如下 text 字符串中的 HTML 文档的，代码如下。

```
text= """
<title>The Dormouse's story</title>
<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>
"""
```

示例代码如下。

```
>>> from lxml import etree
>>> text = """
<title>The Dormouse's story</title>
<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>
"""
>>> html = etree.HTML(text)
>>> result = etree.tostring(html)
>>> print(result)
b'<html><head><title>The Dormouse\'s story</title>\n</head><body><p class="title" name="dromouse"><b>The Dormouse\'s story</b></p>\n<p class="story">Once upon a time there were three little sisters; and their names were\n<a href="http://example.com/elsie" class="sister" id="link1"><!.. Elsie ..></a>,\n<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and\n<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;\nand they lived at the bottom of a well.</p>\n<p class="story">... </p>\n</body></html>'
>>>
```

如上程序运行结果显示，lxml 并没有格式化输出 HTML 文档，不过 lxml 自动补齐了 text 中缺少的 HTML 标签。那该如何进行格式化呢？这里使用 decode()函数来完成格式化输出。示例代码如下。

```
>>> print(result.decode("utf.8"))
<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>
```

2）读取 html 文件

现在学习如何通过 parse()方法读取 HTML 文件。首先,定义一个名为 text.html 的 HTML 文件，内容如下。

```
<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>
</body>
```

然后，编写程序代码，代码如下。

```
from lxml import etree
html = etree.parse('text.html')
result = etree.tostring(html, pretty_print=True)
print(result.decode("utf.8"))
```

运行结果如图 11.7 所示。

注意：凡是涉及调用路径问题，在 IDE 中编辑比较方便。

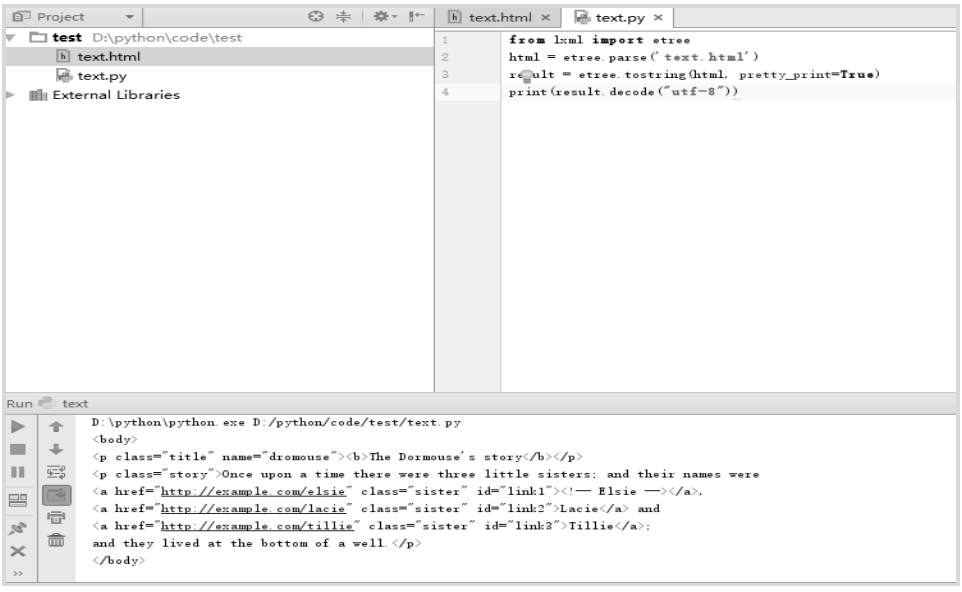

图 11.7　lxml 库文件读取功能演示

3）lxml 库中的标签搜索方法

在 lxml 库中可用 find()、findall()和 xpath()3 种方式来搜索 Element 包含的标签对象。其区别如下：

（1）find()：返回第一个匹配对象，其参数使用的 xpath 语法只能用相对路径（以“.//”开头）。

（2）findall()：返回一个标签对象的列表，其参数使用的 xpath 语法只能用相对路径。

（3）xpath()：返回一个标签对象的列表，其参数使用的 xpath 语法既可以是相对路径，也可以是绝对路径，示例代码如下。

```
>>> from lxml import etree
>>> text = """
<title>The Dormouse's story</title>
<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>
"""
>>> html = etree.HTML(text)
>>> result = etree.tostring(html)
>>> print(result.decode("utf.8"))
<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>
>>> print(type(html))
<class 'lxml.etree._Element'>
>>> result    = html.xpath('//li')
>>> print(result)
[]
>>> result = html.find('.//a')
>>> print(result)
<Element a at 0x1bcf979cec8>
>>> result = html.findall('.//a')
>>> print(result)
[<Element a at 0x1bcf979cec8>, <Element a at 0x1bcf979ce48>, <Element a at 0x1bcf979ce88>]
>>> result = html.xpath('//a')
>>> print(result)
[<Element a at 0x1bcf979cec8>, <Element a at 0x1bcf979ce88>, <Element a at 0x1bcf979ce48>]
>>> result = html.find('//a')
Traceback (most recent call last):
```

```
  File "<pyshell#16>", line 1, in <module>
result = html.find('//a')
SyntaxError: cannot use absolute path on element
```

上述代码以检索目标文档 text 中的超链接标签 a 为例，分别使用 3 种标签搜索方式查找。通过实战，不难发现：

（1）find()返回目标标签第一个位置，而且必须使用相对路径，否则提示语法错误，代码如下。

```
 # 返回第一个元素位置
>>> result = html.find('.//a')
>>> print(result)
<Element a at 0x1bcf979cec8>
# find()  使用绝对路径报错
>>> result = html.find('//a')
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
result = html.find('//a')
SyntaxError: cannot use absolute path on element
```

（2）findall()与 xpath() 的检索结果一致，只是 findall()中必须使用相对路径，而 xpath()使用两种路 径格式均可。

```
#find all()  函数
>>> result = html.findall('.//a')
>>> print(result)
[<Element a at 0x1bcf979cec8>, <Element a at 0x1bcf979ce48>, <Element a at 0x1bcf979ce88>]
# xpath  绝对路径检索
>>> result = html.xpath('//a')
>>> print(result)
[<Element a at 0x1bcf979cec8>, <Element a at 0x1bcf979ce88>, <Element a at 0x1bcf979ce48>]
# xpath  相对路径检索
>>> result = html.xpath('.//a')
>>> print(result)
[<Element a at 0x1bcf979cec8>, <Element a at 0x1bcf979ce48>, <Element a at 0x1bcf979ce88>]
>>>
```

提示

何为 XPath 语法?

XPath 是一门在 XML 文档中查找信息的语言，可用来在 XML 文档中对元素和属性进行遍历。XPath 是 W3C XSLT 标准的主要元素，并且 XQuery 和 XPointer 都构建于 XPath 表达之上。因此，对 XPath 的理解是很多高级 XML 应用的基础。在 XPath 中，有 7 种类型的节点：元素、属性、文本、命名空间、处理指令、注释以及文档（根）节点。XML 文档是被作为节点树来对待的。树的根被称为文档节点或者根节点。

关于 Xpath 语法的具体内容，感兴趣的读者可自行学习，鉴于篇幅的关系，这里不再详细描述。读者只需知道 Lxml 标签搜索方法参数的固定格式即可。

11.3 案例剖析：酷狗 TOP500 数据爬取

本节以获取“酷狗 TOP500”排行榜数据为例，实战练习基于 Python 的简单爬虫程序设计思路及实现过程。

11.3.1 思路简析

1. 任务要求

以酷狗音乐“酷狗 TOP500”榜单网页为定向爬取对象，通过爬虫程序获取其页面中排名前 500 的歌曲的 rank、singer、titles 和 times 字段值，并输出结果。

2. 环境要求

确保已经正确安装 Python 安装包、Requests 模块、BeautifulSoup 模块、lxml 模块。

11.3.2 代码实现

示例代码如下。

```
import requests
from bs4 import BeautifulSoup
import time
headers = {
    'User.Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36'
}
def get_info(url):
    wb_data = requests.get(url, headers=headers)
    soup = BeautifulSoup(wb_data.text, 'lxml')
    ranks = soup.select('span.pc_temp_num')
    titles = soup.select('div.pc_temp_songlist > ul > li > a')
    times = soup.select('span.pc_temp_tips_r > span')
    for rank, title, time in zip(ranks, titles, times):
      str1 = title.get_text().split('.')
      data = {
          'rank': rank.get_text().strip(),
          'singer': str1[0],
          'song': str1[-1],
          'time': time.get_text().strip()
              }
      print(data)
if __name__ == '__main__':
    urls = [
        'http://www.kugou.com/yy/rank/home/{}.8888.html'.format(str(i)) for i in range(1, 30)
    ]
    for url in urls:
        get_info(url)
        time.sleep(2)
```