第16章

网络爬虫项目——豆瓣图书爬虫和检索

本章内容提要

随着大数据时代的来临，网络信息量也变得更多更大，网络爬虫在互联网中的需求也越来越明显。前面章节中曾讲述过 Python 解析 HTML 文件的方法，其实这也是网络爬虫的一种应用。本章将详细介绍 Python 语言实现网络爬虫的过程和常见的网络爬虫框架，最后通过一个实战项目来学习网络爬虫的方法和技巧。

16.1 什么是网络爬虫


微视频

网络爬虫简称爬虫，可以将其理解为在网络上爬行的一组蜘蛛，如果把互联网比作一张大网，爬虫就是在这张网上爬来爬去的蜘蛛。如果爬虫遇到资源，就会抓取下来。至于抓取什么内容，由用户控制。

例如，爬虫抓取一个网页，在这个网页中发现了一条道路，即指向其他网页的超链接，它就可以爬到另一个网页上获取数据。这样，整个连在一起的大网对这只蜘蛛来说触手可及。

一个网络爬虫的基本工作流程如下：

（1）获取需要爬虫的网页的初始 URL 地址。

（2）爬取页面获取新的 URL 地址。

（3）将获取的新的 URL 地址放入 URL 队列中。

（4）依此读取对立中的 URL 地址，然后下载对应的网页。

（5）设置停止条件，如果没有设置停止条件，爬虫会一直爬取下去，直到无法获取新的 URL 地址位置。

（6）解析网页中的内容，然后抽取需要的数据。

☆**大牛提醒**☆

URL 即统一资源定位符，也就是常说的网址。统一资源定位符是对可以从互联网上得到资源位置和访问方法的一种简洁表示，是互联网上标准资源的地址。互联网上的每个文件都有一个唯一的 URL，其包含的信息指出文件的位置及浏览器应该怎么处理。由于爬虫爬取数据时必须有一个目标的 URL 才可以获取数据，因此它是爬虫获取数据的基本依据，准确理解它的含义对爬虫学习有很大帮助。

在用户浏览网页的过程中，可能会发现许多好看的图片。例如，输入百度的图片网址：

http://image.baidu.com/，会看到几张图片及百度搜索框，其实这个过程就是用户输入网址之后，经过 DNS 服务器找到服务器主机，并向服务器发出一个请求，服务器经过解析发送给用户的浏览器 HTML、JS、CSS 等文件，将浏览器解析出来，用户便可以看到形形色色的图片。因此，用户看到的网页实质是由 HTML 代码构成的，爬虫爬下来的便是这些内容，通过分析和过滤这些 HTML 代码，实现对图片、文字等资源的获取。


微视频

16.2 网络爬虫的常用技术

本节将学习网络爬虫中常用的技术。

16.2.1 网络请求技术

获取 URL 地址和下载网页是网络爬虫中的两个最重要的功能。实现这两个功能有两种常见的方式，包括 urllib 和 requests。

1. urllib 模块

urllib 模块内置在 Python 中，该模块可以处理客户端的请求和服务器端的响应，还可以解析 URL 地址。该模块中最常用的子模块包括 request 模块和 parse 模块。

（1）request 模块

request 模块是使用 socket 读取网络数据的接口，支持 HTTP、FTP 及 gopher 等连接。

要读取一个网页文件，可以使用 urlopen()方法。其语法如下：

```
urllib.request.urlopen(url [, data])
```

其中，参数 url 是一个 URL 字符串；参数 data 用来指定一个 GET 请求。

urlopen()方法返回一个 stream 对象，可以使用 file 对象的方法来操作此 stream 对象。

例如，要爬取 http://image.baidu.com/中的数据，代码如下：

```
from urllib import request
htmlpage = urllib.request.urlopen("http://image.baidu.com/")
content=htmlpage.read().decode('utf-8')
print(content)
```

urlopen()方法返回的 stream 对象有两个属性，即 url 与 headers。url 属性是设置的 URL 字符串值；headers 属性是一个字典集，包含网页的表头。

下面的案例是将读取 http://www.python.org 主页的内容。

```
import urllib.request
response = urllib.request.urlopen("http://www.python.org")
html = response.read()
```

也可以使用以下代码实现上述功能：

```
import urllib.request
req = urllib.request.Request("http://www.python.org")
response = urllib.request.urlopen(req)
the_page = response.read()
```

下面的案例是将 http://www.python.org 网页存储到本机的 16.1.html 文件中。

【例 16.1】使用 urlopen()方法抓取网页文件（源代码\ch16\16.1.py）。

```
import urllib.request
#打开网页文件
htmlhandler = urllib.request.urlopen("http://www.python.org")

#在本机上创建一个新文件
file = open("D:\\python\\ch16\\16.1.html", "wb")

#将网页文件存储到本机文件上，每次读取 512 个字节
while 1:
    data = htmlhandler.read(512)
    if not data:
        break
    file.write(data)

#关闭本机文件
file.close()
#关闭网页文件
htmlhandler.close()
```

保存并运行程序，即可将 http://www.python.org 网页存储到本机的 16.1.html 文件中。

（2）parse 模块

parse 模块解析 URL 字符串并返回一个元组：(addressing scheme, netword location, path, parameters, query, fragment identifier)。parse 模块可以将 URL 分解成数个部分，并能组合回来，还可以将相对地址转换为绝对地址。

2. requests 模块

requests 模块也可以爬取网络数据。requests 模块是 Python 语言基于 urllib 模块编写的，采用的是 Apache2 Licensed 开源协议的 HTTP 模块。使用 requests 会比 urllib 模块更加方便，可以节约大量的工作。requests 模块属于第三方模块，该模块需要下载并安装后才能使用。下面讲述该模块的安装方法。

安装 requests 模块的命令如下：

```
pip install requests
```

安装过程如图 16-1 所示。如出现 Successfully installed ……信息，表示 requests 模块已经安装成功。

☆**大牛提醒**☆

在安装的过程中，可能会提示连接服务器失败，这是由于网络不稳定造成的，可以多运行几次即可解决。另外如果提示 pip 的版本比较低，可以使用下面的命令升级 pip:

```
python -m pip install --upgrade pip
```

requests 模块安装完成后，可以使用以下命令进行检测是否安装成功：

```
import requests
```

如果没有提示错误，那说明已经安装成功了，如图 16-2 所示。

以 get 请求方式，发送 HTTP 网络请求，代码如下：

```
import requests                          #导入 requests 模块
response = requests.get('http://image.baidu.com/')
print(response.status_code)              #打印状态码
print(response.url)                      #打印请求 url
print(response.headers)                  #打印头部信息
print(response.cookies)                  #打印 cookies 信息
```

```
print(response.text)                          #以文本形式打印网页源码
print(response.content)                       #以字节流形式打印网页源码
```

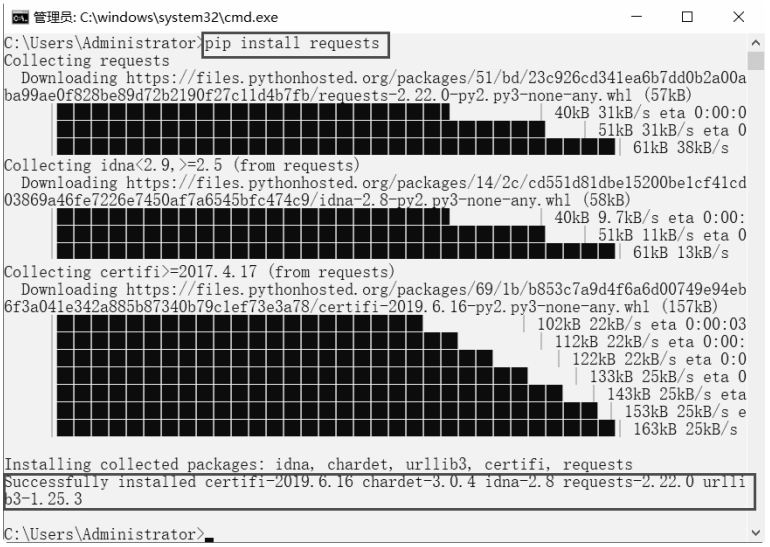

图 16-1　安装 requests 模块

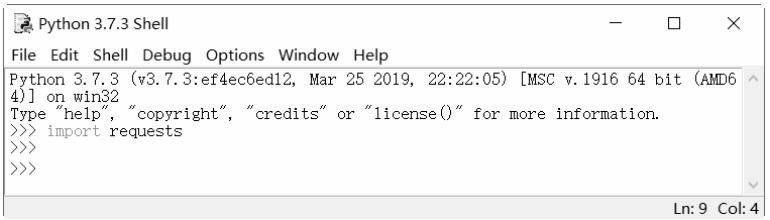

图 16-2　加载 requests 模块

requests 模块请求数据最常用的方式有两种，包括 get 方式和 post 方式，下面通过案例来学习。

以 post 请求方式，发送 HTTP 网络请求，代码如下：

```
import requests                               #导入 requests 模块
dt = {'pic': 'dog'}                           #表单参数
response = requests.post('http://image.baidu.com/post', data=dt)
print(response.content)                       #以字节流形式打印网页源码
```

如果请求的 URL 地址中参数是跟在“？”后面，例如“http://image.baidu.com/get？key=val”，可以使用关键字 params 参数来解决，以一个字符串字典来提供这些参数。例如，传递“key1=dog”和“key2=cat”到“http://image.baidu.com/get”，代码如下：

```
import requests                               #导入 requests 模块
dt = {'pic': 'dog'}                           #表单参数
response = requests.get('http://image.baidu.com/post', data=dt)
print(response.content)                       #以字节流形式打印网页源码
```

16.2.2　请求 headers 处理

如果请求网页内容时，提示 403 错误，说明该网页设置反爬虫，从而拒绝了用户的访问。

解决这个问题的方法，就是模拟浏览器的头部信息进行访问，网站服务器就会认为是浏览器的正常访问，从而跳过反爬虫设置的障碍。

请求头部 headers 的处理方法如下：

通过浏览器的网络监视器查看头部信息。通过火狐浏览器打开网址 https://image.baidu.com/，然后按下 Ctrl+Shift+E 组合键，即可打开网络监控器。按 F5 键刷新当前页面，网络监控器将显示请求的数据信息，如图 16-3 所示。

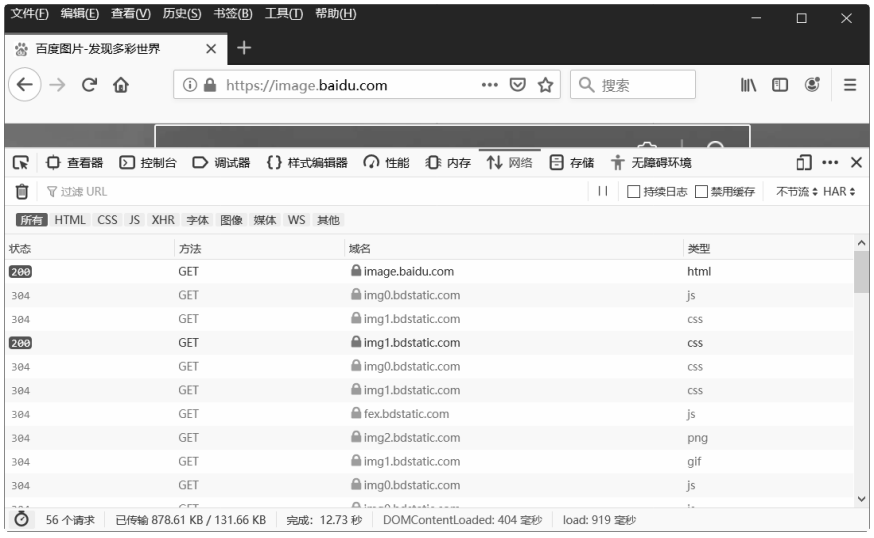

图 16-3　网络监控器

选中第一条 get 信息，即可查看对应的信息头。这里需要复制该头部信息，如图 16-4 所示。

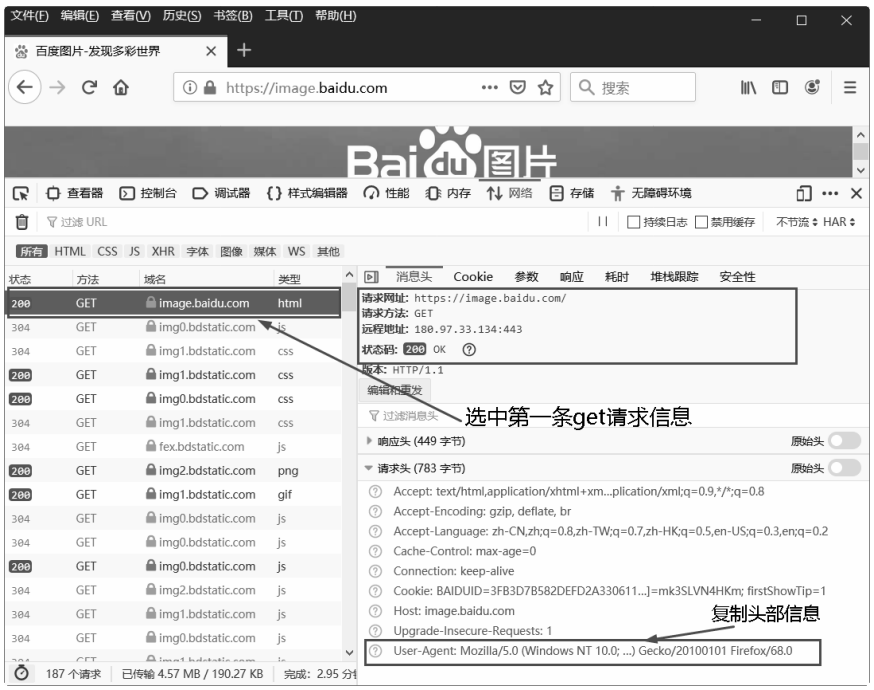

图 16-4　查看并复制头部信息

要爬取 https://image.baidu.com/地址，就可以通过创建 headers 头部信息，然后再发送等待相应，最后打印网页，代码如下：

```
    import requests                                          #导入 requests 模块
    url= 'http://image.baidu.com/'                           #创建需要爬取的网址
    #创建头部信息
    headers = {' User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:68.0)
Gecko/20100101 Firefox/68.0}
    response = requests.get(url,headers=headers)             #发送网络请求
    print(response.content)                                  #以字节流形式打印网页源码
```

16.2.3 网络超时问题

如果访问一个网页长时间没有登录进去，系统就会判断该网页超时，此时无法打开该网页。下面通过代码来模拟一个网络超时的现象，代码如下：

```
    import requests                                          #导入 requests 模块
    #循环发送请求 5 次
    for n in range(0,5):
       try:
          #设置超时为 1 秒
          response = requests.get()
          response = requests.post('http://image.baidu.com/post',timeout=1)
          print(response.url)                                #打印请求 url
       except Exception as e:                                #捕获异常
          print('异常是: '+str(e))                           #打印异常信息
```

执行结果如下：

```
    异常是: get() missing 1 required positional argument: 'url'
    异常是: get() missing 1 required positional argument: 'url'
    异常是: get() missing 1 required positional argument: 'url'
    异常是: get() missing 1 required positional argument: 'url'
    异常是: get() missing 1 required positional argument: 'url'
```

从结果可以看出，当 1s 内服务器没有做出相应命令则视为超时。根据以上模拟测试结果，可以确认在不同的情况下设置不同的 timeout 值。

16.2.4 代理服务

在爬取网页的过程中，经常会出现正常爬取的网页不能爬取了，这是因为用户的 IP 被爬取网站的服务器屏蔽了。解决上述问题的一个很好的方法，就是设置代理服务器。例如，代理服务器的地址为 144.164.177:806。使用该代理服务器爬取网页的代码如下：

```
    import requests                                          #导入 requests 模块
    py = {'http': '144.164.177: 806',
    'https': '144.164.177: 8080'}                            #设置代理 ip 与对应的端口号
    #对需要爬取的网页发送请求
    response = requests.get('http://image.baidu.com/post', proxies=py)
    print(response.content)                                  #以字节流形式打印网页源码
```

☆**大牛提醒**☆

网上有很多免费的代理服务器。但是其 IP 地址是不固定的，所以需要经常更新。

16.3　豆瓣读书爬虫项目分析


微视频

随着网络的迅速发展，万维网成为大量信息的载体，毫无疑问，目前已经进入大数据时代。而如果要进行数据分析，首先得有数据源，而且是有针对的数据源。通过爬虫技术，可以获取到更多的准确数据源，并且这些数据源可以一定的规则进行采集，去掉很多无关数据。

用 Python 写一个爬虫是很容易的，一只简单的爬虫只需要几分钟就能实现。因为 Python 有关爬虫的第三方模块非常完善，尤其可以把爬虫和机器学习、自然语言分析、数据可视化无缝衔接起来，非常方便。

本案例通过写一个爬虫抓取豆瓣读书的网页并解析图书信息，然后给定检索词，返回包含该词的图书。

通过该案例的学习，强化读者对 Python 基础知识的理解，通过亲身体验一个爬虫的实现过程，可以加深对搜索引擎爬虫的工作原理的理解，从而更好地进行搜索引擎优化，也为学习其他爬虫方法打下基础。

该项目运行过程如图 16-5 所示。

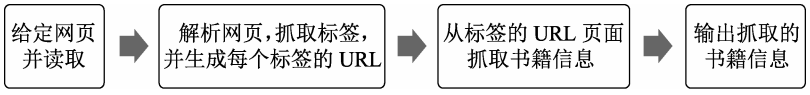

图 16-5　项目运行过程

DoubanCrawler.py 为本案例的核心文件，其功能是实现爬取数据，主要思路如下：

（1）读取指定的网页，采用 requests 模块的 get()方法。

（2）解析网页，获得所需要的内容，采用 beautifulsoup 模块的方法。

（3）查看网页源代码，找到要爬取数据所在的标签，抓取该标签，并生成该标签的 url。

（4）根据（3）得到的 url，找到需要爬取的数据所在的标签，抓取其内容，组成一个列表。

（5）输出（4）的结果，也就是最终需要爬取的内容。

16.4　环境配置


微视频

本书在第 1 章已经讲述了 Python 环境的配置方法，这里就不再重述。唯一不同的是，豆瓣读书爬虫项目的运行需要 2 个模块，包括 beautifulsoup 模块和 requests 模块。

16.4.1　下载并安装模块文件

1. beautifulsoup 模块

beautifulsoup 模块是一个可以从 HTML 或 XML 文件中提取数据的 Python 模块，简单来说，它能将 HTML 的标签文件解析成树形结构，方便获取到指定标签的对应属性。beautifulsoup 模块不需要编写正则表达式就可以很方便地实现网页信息的提取，既灵活又高效，是非常受欢迎的网页解析模块。

beautifulsoup 模块的下载地址是 https://pypi.org/project/beautifulsoup4/，目前，beautifulsoup 模块的最新版本是 4.8.0，如图 16-6 所示。

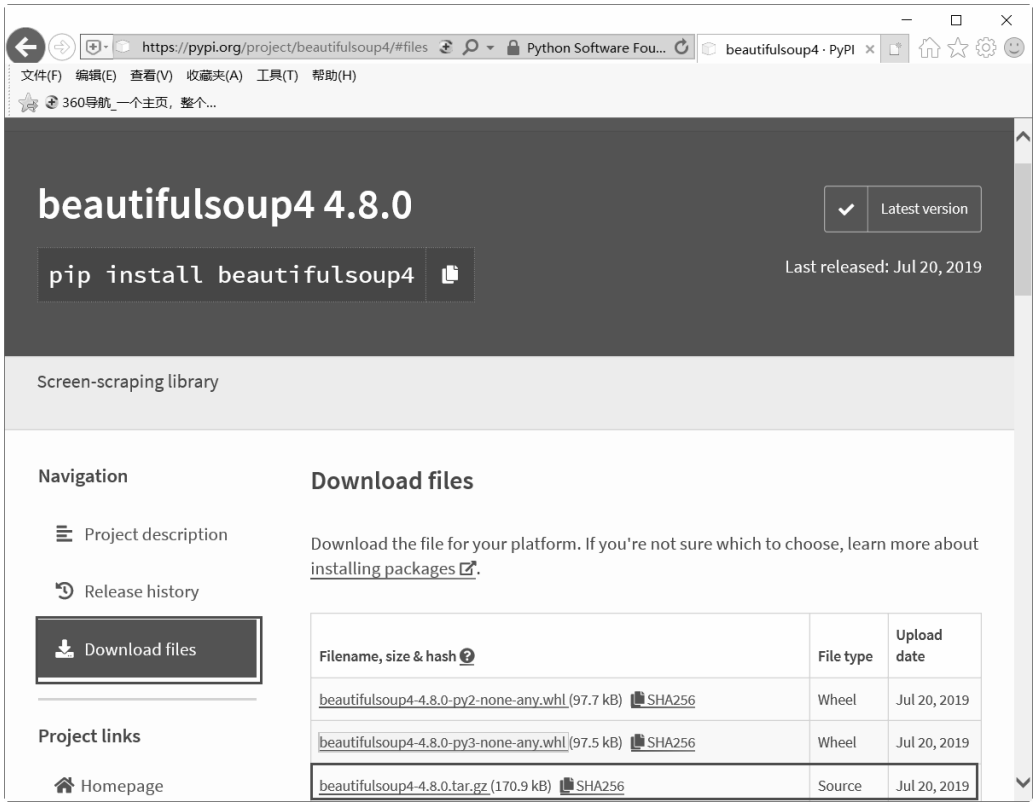

图 16-6　beautifulsoup 模块的下载界面

将下载的 beautifulsoup4−4.8.0.tar.gz 压缩文件解压，即可发现有一个 setup.py 安装文件，如图 16-7 所示。

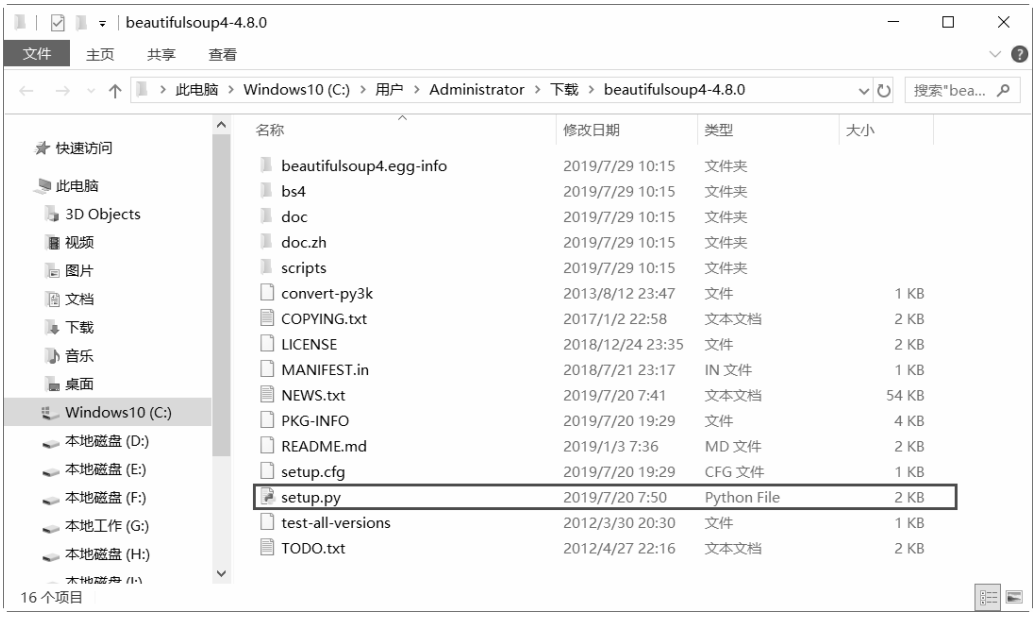

图 16-7　解压文件

安装时，以管理员的身份运行“命令提示符”，进入文件解压的目录，然后执行下面的命令即可自动安装 beautifulsoup 模块：

```
python setup.py install
```

安装过程如图 16-8 所示。

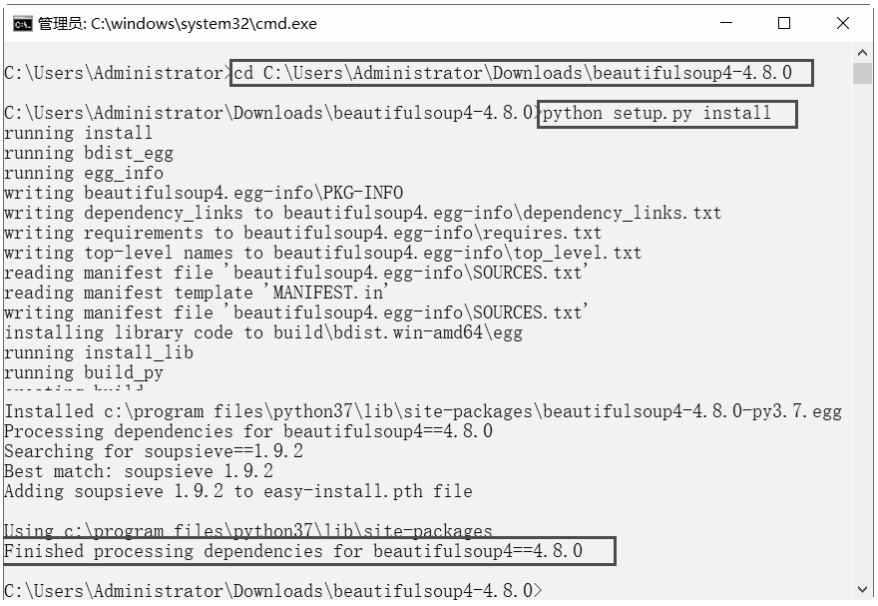

图 16-8　安装 beautifulsoup 模块

除了以上安装模块文件的方法外，用户还可以在线安装 beautifulsoup 模块，方法比较简单。同样也以管理员的身份运行“命令提示符”，执行在线安装 beautifulsoup 模块的命令：

```
python -m pip install beautifulsoup4
```

开始自动下载并安装 beautifulsoup 模块，执行过程如图 16-9 所示。

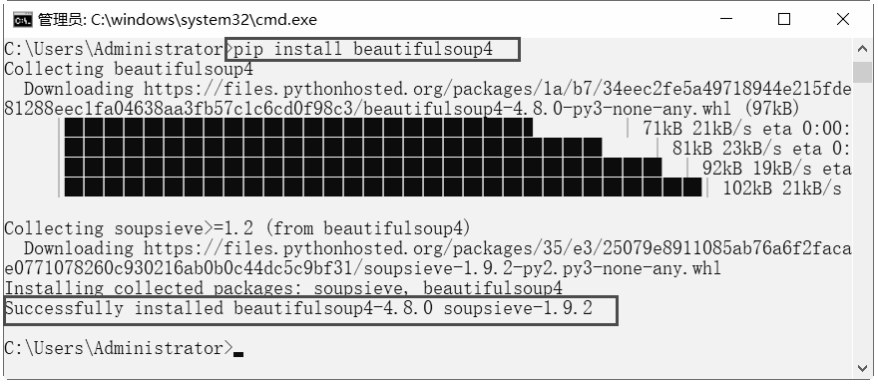

图 16-9　在线安装 beautifulsoup 模块

2. requests 模块

requests 模块是简单易用的 HTTP 模块，使用起来要比 urllib 模块简洁许多。用户可以使用 pip 命令安装 requests 模块，方法比较简单。以管理员的身份运行“命令提示符”，执行 pip 安装命令：

```
pip install requests
```

16.4.2 检查模块文件是否安装成功

16.4.1 节两个模块文件安装完成后，用户需要检查一下是否安装成功。

以管理员的身份运行“命令提示符”，检查当前安装了哪些模块，命令如下：

```
python -m pip list
```

检查结果如图 16-10 所示。

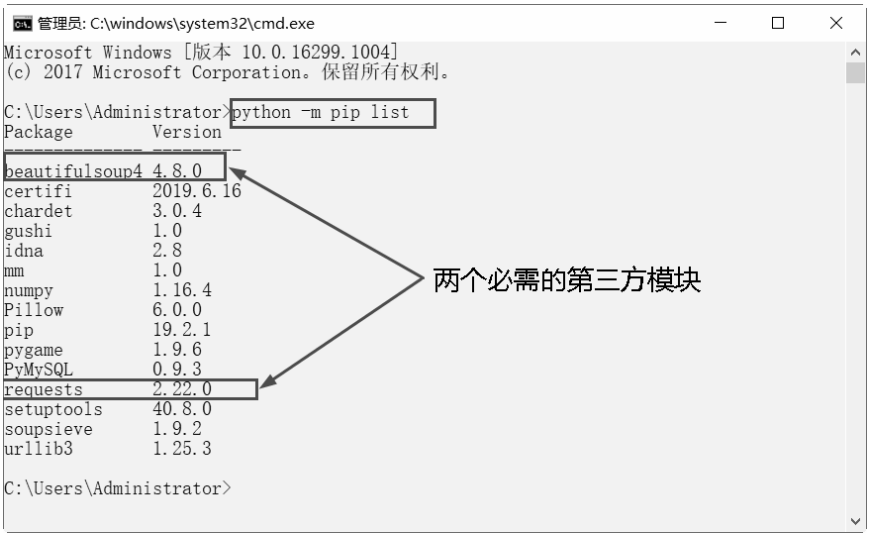

图 16-10 检查当前安装的模块

从检查结果可以看出，beautifulsoup 模块和 requests 模块已全部安装成功。

微视频

16.5 具体功能实现

豆瓣读书爬虫项目主要实现两个功能：爬取图书数据和检索图书信息。

16.5.1 爬取图书数据

DoubanCrawler.py 文件定义了一个抓取页面的爬虫的类（类 DoubanBookCrawler）和一个执行函数——main()函数。main()函数定义了从给定网页爬取书籍信息的运行流程和逻辑。

类 DoubanBookCrawler 定义了 3 个函数，它们的作用如下。

（1）TagCrawl()函数：抓取给定网页内的标签及内容，并生成该标签的 url。

（2）BookInfoParser()函数：根据给定的网页来抽取每本书的数据，并将这些数据组成一个列表。

（3）Crawling()函数：根据不同标签生成的 url，对每个标签下面的图书列表进行抓取。

DoubanCrawler.py 文件的具体代码如下：

```
import requests
import time
```

```
    import re
    import random
    from bs4 import BeautifulSoup

    class DoubanBookCrawler:
        """抓取豆瓣读书页面的爬虫"""
        def __init__(self, url):
            self.rooturl = url
            self.tagurls = []
            self.bookinfo_list = []

        def TagCrawl(self):
            html = requests.get(self.rooturl)

            soup = BeautifulSoup(html.text)  #利用 BeautifulSoup 解析网页信息
            tags = soup.select("#content > div > div.article > div > div > table > tbody
> tr > td > a")
            #利用 BeautifulSoup 查找网页源代码中包含 tag 的内容
            for tag in tags:
                tag = tag.get_text()  #将列表中的每一个标签信息提取出来
                preurl = "https://www.douban.com/tag/"
                                    #基于豆瓣网址规则，利用检测到的 tag 生成要抓取的页面
                url = preurl + str(tag) + "/book"
                self.tagurls.append(url)
            #return tagurls

        def BookInfoParser(self, url):
            info_list = []
            html = requests.get(url) #根据给的 url 来抽取每本书中的信息
            bsoup = BeautifulSoup(html.text.encode("utf-8"))#, "lxml")
            list_soup = bsoup.find('div', {'class': 'mod book-list'})

            for book_info in list_soup.findAll('dd'):
                title = book_info.find('a', {'class':'title'}).string.strip()
                desc = book_info.find('div', {'class':'desc'}).string.strip()
                desc_list = desc.split('/')
                book_url = book_info.find('a', {'class':'title'}).get('href')

                try:
                    authors = '作者/译者:  ' + '/'.join(desc_list[0:-3])
                except:
                    authors ='作者/译者:  暂无'
                try:
                    publication = '出版信息:  ' + '/'.join(desc_list[-3:])
                except:
                    publication = '出版信息:  暂无'
                try:
                    rating = book_info.find('span', {'class':'rating_nums'}).string.
strip()
                except:
                    rating='0.0'

                info_list.append((title,rating,authors,publication,book_url))
```

```
            return info_list

        def Crawling(self, n):
            #根据不同 tag 生成的 url，对每个 tag 下面的图书列表进行抓取
            #值得注意的是，对于每个 tag，我们这里只抓取第一页的图书信息
            #如果需要抓取剩下页的信息，只需要根据豆瓣 url 的格式生成新的 url 即可
            for url in self.tagurls[:n]:
                infolist = self.BookInfoParser(url)
                self.bookinfo_list += infolist
                time.sleep(2)

    def main():
    #豆瓣读书保存 tag 的入口 url
        url="https://book.douban.com/tag/?icn=index-nav"
        crawler = DoubanBookCrawler(url)
        crawler.TagCrawl()
        crawler.Crawling(2)
        for u in crawler.bookinfo_list:
            print(u)
```

16.5.2 检索图书信息

BookSearch.py 文件实现了图书检索功能。该文件定义了一个检索函数——SearchBookByTitle()函数。该函数首先调用 DoubanCrawler.py 文件中的 DoubanBookCrawler(url)函数，实现豆瓣读书网页的图书信息的抓取，然后是一个循环，将符合条件的标题组成一个不重复的无序数组，最后程序执行时将检索结果输出。

DoubanCrawler.py 文件的具体代码如下：

```
#!/usr/bin/python
#-*- coding: utf-8 -*-

import DoubanCrawler

def SearchBookByTitle(query):
    """抓取豆瓣读书的图书信息
    并且基于给定的检索词
    返回包含该检索词的图书"""
    url="https://book.douban.com/tag/?icn=index-nav"
    crawler = DoubanCrawler.DoubanBookCrawler(url)
    crawler.TagCrawl()
    crawler.Crawling(4)
    books = set(crawler.bookinfo_list)

    results = set()
    for book in books:
        (title,rating,authors,publication,book_url) = book
        #print(title)
        if title.find(query) >= 0:
            results.add(title)
    return results

if __name__ == '__main__':
```

```
query = input("请输入查询词: ")
results = SearchBookByTitle(query)
print("检索结果为: ")
for item in results:
    print(item.encode('utf-8').decode('utf-8'))
```

16.6　项目测试


微视频

在编辑器中写好以上模块内容后保存。下面将测试豆瓣读书爬虫项目。运行本示例的 BookSearch.py 文件，提示输入查询的关键词，结果如图 16-11 所示。

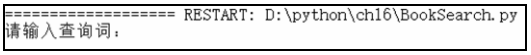

图 16-11　运行项目

输入查询的关键词“百年”，按 Enter 键确认，结果如图 16-12 所示。

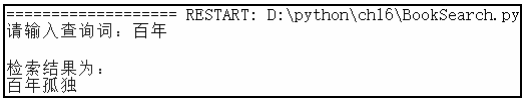

图 16-12　查询结果