扫一扫

视频讲解

第3章 Python爬虫实战

大家在学会使用 requests 库和 BeautifulSoup 库以后，基本上可以编写爬虫对网页页面进行爬取并解析，从而获得所需数据。但在实际操作时，不同网站的模板结构几乎不同，网页中的数据也存在结构化、半结构化和非结构化的差异，无法采用统一的采集方法。本章进行爬虫实战，3.1 节通过对中国 A 股上市公司的相关数据进行获取，帮助大家理解如何爬取和解析结构化的数据；3.2 节介绍解析出来的数据的文件存储形式，主要包括适用于非结构化数据的文本文件、适用于结构化数据的 CSV 文件和适用于半结构化数据的 JSON 文件；3.3 节以豆瓣读书排行榜 Top250 的数据为例，进行半结构化数据的获取和解析；3.4 节主要讲解正则表达式的使用，以提高文本的解析效率；在 3.5 节以人民网科技类新闻为例，进行非结构化数据的获取和解析。

3.1 实战：中国 A 股上市公司相关数据的获取

本节编写爬虫对结构化数据——中国 A 股上市公司的相关数据进行爬取和解析，数据来源于中商情报网(https://s.askci.com/stock/a/)，从页面展示来看，这些数据以结构化的表格样式呈现出来，如图 3.1 所示。

序号	股票代码	股票简称	公司名称	省份	城市	主营业务收入(202109)	净利润(202109)	员工人数
1	000001	平安银行	平安银行股份有限公司	广东	深圳市	--	291.35亿	36676
2	000002	万科A	万科企业股份有限公司	广东	深圳市	2714.86亿	246.04亿	140565
3	000004	国华网安	深圳国华网安科技股份有限公司	广东	深圳市	1.66亿	3395.90万	264
4	000005	ST星源	深圳世纪星源股份有限公司	广东	深圳市	2.92亿	1.92亿	629
5	000006	深振业A	深圳市振业(集团)股份有限公司	广东	深圳市	23.02亿	5.88亿	397
6	000007	*ST全新	深圳市全新好股份有限公司	广东	深圳市	1.28亿	-308.50万	76
7	000008	神州高铁	神州高铁技术股份有限公司	北京	北京市	10.43亿	-9568.30万	2394
8	000009	中国宝安	中国宝安集团股份有限公司	广东	深圳市	122.07亿	15.18亿	13345
9	000010	美丽生态	深圳美丽生态股份有限公司	广东	深圳市	10.57亿	4003.73万	278

图 3.1 中商情报网页面

3.1.1 目标网站分析

对目标网站“中商情报网”进行预分析，有助于爬虫代码程序的顺利编写。

(1) 查看目标网站的 robots 协议，了解爬取规范。

(2) 使用 Chrome 工具查看数据所在网页页面的特征。

1. 查看 robots 协议

在浏览器的地址栏中输入“https://www.askci.com/robots.txt”，查看目标网站的 robots 协议。可以看出中商情报网对爬虫比较友好，除了 TongJiNews、TongJiReport、404、customreport 目录以外，网站上的其他资源都允许被爬取，如图 3.2 所示。

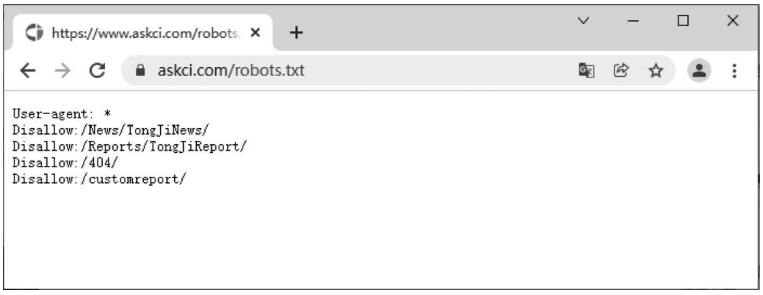

图 3.2 中商情报网的 robots 协议

2. 使用 Chrome 工具进行分析

使用 Chrome 工具查看数据所在的网页页面，主要查看页面请求的 URL 和特点，以及请求类型、请求头的相关信息、页面中的数据所在的位置特征等。

1) 查看 Network 面板

通过 Chrome 工具的 Network 面板可以查看请求 URL 和特点，以及请求类型、请求头的相关信息，如图 3.3 所示。

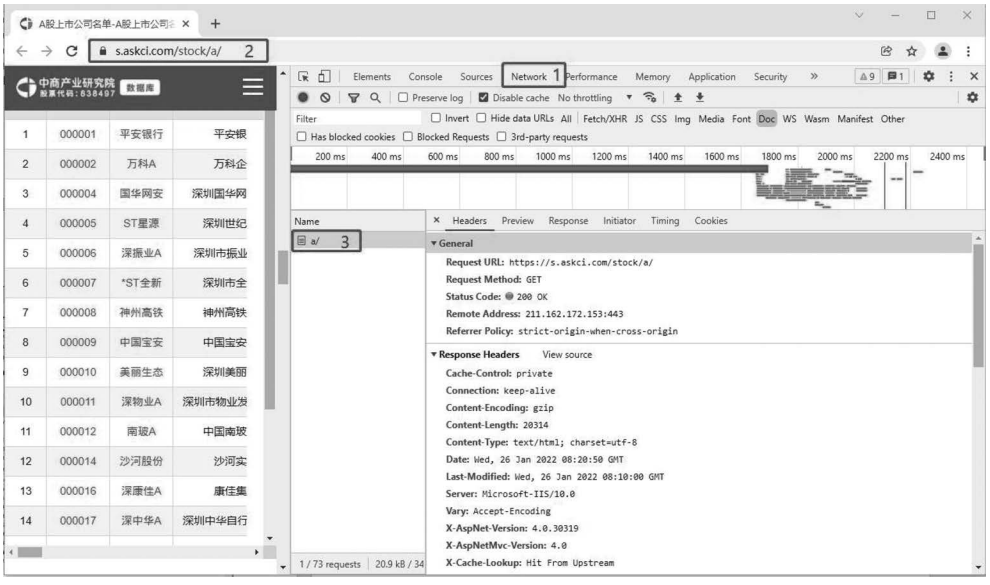

图 3.3 使用 Chrome 工具的 Network 面板查看相关内容

其具体操作步骤如下：

(1) 在 Chrome 工具中单击 Network 选项卡。

(2) 在地址栏中输入网页地址“https://s.askci.com/stock/a/”，或者将鼠标指针放置在已经存在的网页地址后按回车键进行刷新。

(3) 在 Name 栏下出现了资源路径/a，单击该资源路径后将出现该资源的头部(Headers)、预览(Preview)等信息，通过查看这些信息可以获得请求 URL、请求类型等相关内容。

2) 查看 Elements 面板

通过 Elements 面板可以查看页面中的数据所在的位置特征，如图 3.4 所示。

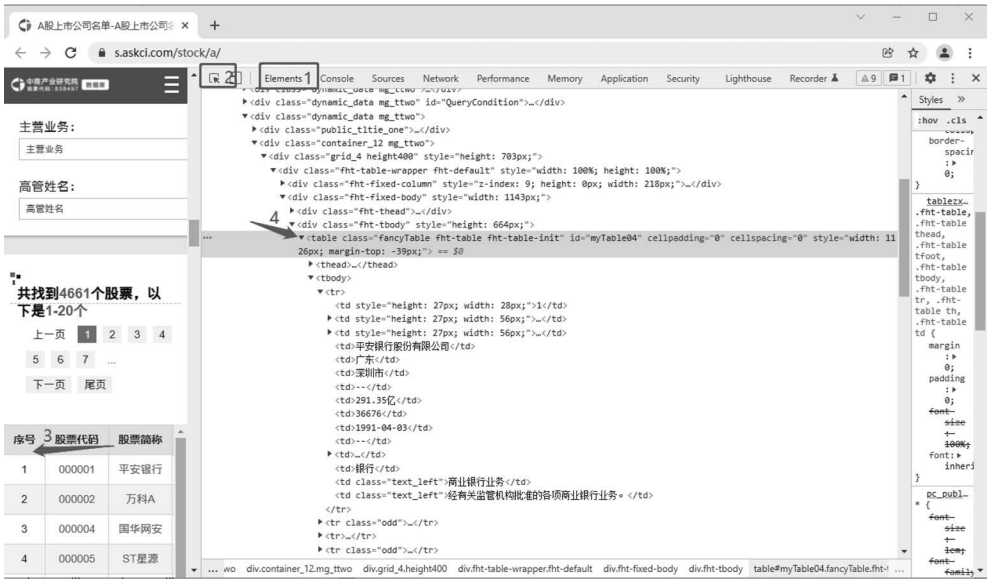

图 3.4 使用 Chrome 工具的 Elements 面板查看相关内容

其具体操作步骤如下：

(1) 在 Chrome 工具中单击 Elements 选项卡。

(2) 单击工具左上角的检查按钮。

(3) 在网页中单击要爬取的数据。

(4) 在 Elements 主页面中定位到该数据资源所在的位置。

通过 Chrome 工具对目标网站进行预分析，可以得到如表 3.1 所示的信息。

表 3.1 通过预分析获得的信息

类 型	内 容
请求 URL 基础地址	https://s.askci.com/stock/a/
请求类型	GET 请求
分页 URL 特点	https://s.askci.com/stock/a/0-0?reportTime=2021-09-30&pageNum=1 https://s.askci.com/stock/a/0-0?reportTime=2021-09-30&pageNum=2
请求头中的 User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.71 Safari/537.36
数据所在的页面特征	<table>标签，其中 id="myTable04"

扫一扫

视频讲解

3.1.2 表格数据的爬取和解析

在对目标网站进行预分析以后，就可以编写代码对表格数据进行爬取和解析。

(1) 使用 requests 库模拟用户请求爬取网页数据。

(2) 使用 BeautifulSoup 库提取网页中的表格数据并解析。

1. 模拟发送请求、爬取数据

发送请求，爬取数据，具体操作如下：

(1) 确定 URL 和相关参数。

确定爬取的 URL，因为分页时 URL 地址中带有 reportTime 和 pageNum 两个参数，所以在请求方法的 get 方法中设置 param；同时为了伪装浏览器，在 header 参数中设置浏览器信息。

```
import requests
url = "https://s.askci.com/stock/a/0-0"
param = {"reportTime": "2021-09-30","pageNum": 1}
header = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)  AppleWebKit/537.36 (KHTML,
like Gecko)  Chrome/92.0.4515.159 Safari/537.36"}
```

(2) 调用 requests 库的 get 方法。

通过 get 方法发送请求，这样就得到了响应，通过响应对象 r 的 text 属性查看响应的 HTML 文档信息。

```
r = requests.get(url,params = param,headers = header)
html = r.text
print(html)
```

如果响应文档无法正常显示中文字符，还需要设置页面响应的 encoding 编码。

2. 解析表格数据

通过分析 Elements 元素，可以看出中国 A 股上市公司相关数据所在的 table 标签内容包括两部分，如图 3.5 所示。

第一部分是标题所在的< thead >标签，包括一对< tr >标签，具体表格标题的内容在< th >标签中。

第二部分是表格数据所在的< tbody >标签，具体每个上市公司的数据在某个< tr >标签中，在每个< tr >标签内又包含了若干< td >标签，存放的是具体的数据内容。

其具体解析步骤如下：

(1) 使用 BeautifulSoup 类将 HTML 文档封装成文档树，这里采用了 lxml 解析器。

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(r.text,"lxml")
```

(2) 使用 soup 对象的 find 方法找到数据所在的标签，通过上面的分析可知要查找 table 标签的 id 是“myTable04”。

```
table = soup.find(id = "myTable04")
```

图 3.5 使用 Chrome 工具的 Elements 面板查看相关内容

(3) 解析提取表格标题数据。在 thead 标签下只有一对 tr 标签,放置的是标题的标签 th,所以针对标题数据的查找只需直接找到 table 标签下所有的 th 标签。

```
ths = table.find_all("th")              #查找 table 中所有的 th 标签
title = [th.text for th in ths]         #使用列表推导式提取 th 标签中的文本标题信息
```

(4) 解析提取表格内容数据。在 tbody 标签下有多对 tr 标签,可在查找到 tbody 标签后查找其下面所有的 tr 标签,再针对每一行的数据查找该行中所有的 td 标签,并提取其中的文本信息。此处用循环依次遍历表格中的每一行,用列表推导式遍历并提取每个单元格的数据,并最终将提取的数据存储在 data 列表中。

```
tbody = table.find("tbody")             #查找 table 中表格内容所在的 tbody 标签
trs = tbody.find_all("tr")              #查找每行所在的标签
data = []
for tr in trs:                          #遍历每一个 tr 标签
    tds = tr.find_all("td")             #查找每行中所有的 td 标签
    tdsv = [td.text for td in tds]      #使用列表推导式提取 td 标签中的文本标题信息
    data.append(tdsv)
```

扫一扫

视频讲解

3.1.3 模块化程序的编写

前面解决了一个页面中的中国 A 股上市公司的数据的爬取和解析,但是所有的 A 股上市公司的数据存放在 234 个页面中,这也就意味着页面的爬取和解析需要重复 234 次,为了实现起来方便,按照功能对代码进行模块化处理,具体步骤如下:

(1) 定义获取 URL 请求的方法。因为每个页面的 URL 差异仅在于请求参数 pageNum 的不同,这里将页数作为方法的参数,定义方法头为 getHtml(page)。

(2) 定义解析表格标题和表格数据的方法。每页都含有表格标题,标题只需要解析一次,将标题和表格的解析设计成两个方法,均以标签树(也就是 BeautifulSoup 对象)作为参数,定

义的方法头分别为 parseTitle(soup)和 parseData(soup)。

(3) 实现上述方法的循环调用。循环调用上述方法,提取中国 A 股上市公司的全部数据,将结果保存在列表 tableData 中。

【实战案例代码 3.1】 中国 A 股上市公司的数据的获取。

```
import requests
from bs4 import BeautifulSoup
#发送请求,获得数据
def getHtml(page) :
    url = "https://s.askci.com/stock/a/0-0"
    header = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)  \
    AppleWebKit/537.36 (KHTML, like Gecko)  Chrome/92.0.4515.159 Safari/537.36"}
    r = requests.get(url,params = {"reportTime": "2021-09-30","pageNum": page},headers =
header)
    return r.text
#解析表格标题
def parseTitle(soup) :
    table = soup.find(id = "myTable04")
    ths = table.find_all("th")
    title = [th.text for th in ths]
    return title
#解析表格数据
def parseData(soup) :
    tbody = soup.find(id = "myTable04") .find("tbody")
    trs = tbody.find_all("tr")
    data = []
    for tr in trs:
        tds = tr.find_all("td")
        tdsv = [td.text for td in tds]
        data.append(tdsv)
    return data
#爬取和解析全部数据
tableData = []
for page in range(1,224) :
    html = getHtml(page)
    soup = BeautifulSoup(html,"lxml")
    if page == 1:
        title = parseTitle(soup)                    #解析标题
        tableData.append(title)
    pageData = parseData(soup)                      #解析每页数据
tableData.extend(pageData)
tableData[:5]
```

tableData 中存放的是爬取下来的全部数据,打印输出 tableData 的前 5 个元素,显示结果如下:

```
[['序号', '股票代码', '股票简称', '公司名称', '省份', '城市', '主营业务收入(202106)  ', 净利润
(202106) ', '员工人数', '上市日期', '招股书', '公司财报', '行业分类', '产品类型', '主营业务'],
['1', '000001', '平安银行', '平安银行股份有限公司', '广东', '深圳市', '846.80 亿', '175.83 亿',
'36676', '1991-04-03', '--', '', '银行', '商业银行业务', '经有关监管机构批准的各项商业银行业
```

```
务。'], ['2', '000002', '万科 A', '万科企业股份有限公司', '广东', '深圳市', '1671.11 亿', '161.74 亿',
'140565', '1991 - 01 - 29', '--', '', '房地产开发', '房地产、物业管理、投资咨询', '房地产开发和物业服
务。'], ['3', '000004', '国华网安', '深圳国华网安科技股份有限公司', '广东', '深圳市', '9301.23 万',
'613.16 万', '264', '1991 - 01 - 14', '--', '', '生物医药', '移动应用安全服务、移动互联网游戏', '移
动应用安全服务业务。'], ['4', '000005', 'ST 星源', '深圳世纪星源股份有限公司', '广东', '深圳市',
'1.64 亿', '1.92 亿', '629', '1990 - 12 - 10', '--', '', '环保工程、物业管理', '酒店经营、物业管理、
环保业务', '绿色低碳城市社区建设相关的服务业务。']]
```

3.2 解析数据的存取

3.1 节的实战爬取并解析了数据,为了方便后续对数据进行分析和处理,可以对数据进行保存。数据的保存形式多种多样,可以保存到文件中,也可以保存到数据库中,本节学习文件类型数据的存取,包括文本文件、CSV 文件和 JSON 文件。

扫一扫

视频讲解

3.2.1 文本文件的存取

文本文件几乎兼容任何平台,将数据保存到文本文件的操作简单,但它的缺点是不利于检索。如果追求方便,对检索性能和数据的结构要求不高,可以采用文本文件。使用 Python 内置的文件处理方法可以方便地对文本文件进行存取。

1. 存储文本文件

使用 Python 存储文本文件的步骤如下:

(1) 使用 open 函数以写入模式打开文本文件,获得文件对象。

(2) 调用文件对象的 write 或 writelines 方法写入解析出来的数据内容。

(3) 调用文件对象的 close 方法关闭文件。

下面先看两个示例。

第一个示例是将一个字符串写入 files 目录下的 data.txt 文件中,f 是文件对象,通过 open 方法获得,然后调用 write 方法写入字符串,最后调用 close 方法关闭文件对象。

【例 3.1】 将字符串写入文本文件。

```
data = " Python 数据分析实践课程理论和实践相结合,助力在业务领域获得数据,分析和处理数据。"
f = open('files/data.txt','w')
f.write(data)
f.close()
```

第二个示例是将一个字符串列表写入文件中,调用文件对象的 writelines 方法写入列表。

【例 3.2】 将列表写入文本文件。

```
urlList = ['https://www.buu.edu.cn', 'https://www.baidu.com']
f = open('files/urls1.txt','w')
f.writelines(urlList)
f.close()
```

在上面的两个例子中用到了 open 函数,其作用是创建可以操作的文件对象,open 函数的核心语法为:

```
open(file, mode = 'r', encoding = None)
```

常用参数的说明如下。

(1) file：接收 string，用字符串表示的文件路径。

(2) mode：接收 string，用字符表示文件的使用模式，默认为只读模式。

(3) encoding：接收 string，文件的编码。

文件的使用模式用于控制以何种方式打开文件，open 函数提供了 7 种基本的使用模式，如表 3.2 所示。

表 3.2 文件的使用模式

模　式	作　用
'r'	只读模式，文件不存在则返回异常，默认值
'w'	覆盖写模式，文件不存在则创建，存在则完全覆盖
'a'	追加写模式，文件不存在则创建，存在则在文件的最后追加内容
'x'	创建写模式，文件不存在则创建，存在则返回异常
'b'	二进制文本模式，适用于非文本文件，例如图片、音频文件等
't'	文本文件模式，默认值，适用于文本文件
'+'	与 r/w/x/a 一起使用，在原有功能上同时增加读/写功能

文件默认使用的模式是'r'，表明以只读形式打开已经存在的文件。文件的使用模式还有'w'(覆盖写)、'a'(追加写)、'x'(创建写)，此外还有 3 个可以和这些模式结合使用的符号'b'、't'、'+'。

(1) 'b'是二进制文本模式，例如'rb'就是读取文件的二进制信息，适合读取图片、音频文件等。

(2) 't'是文本文件模式，适用于文本文件，open 函数默认是'rt'模式，即文本只读模式。

(3) '+'和表示读/写模式的'r'、'w'、'a'、'x'一起使用，表示扩展原有功能，增加读/写。

- 'r+'：既能从文件中读取数据，又能向文件写入数据。
- 'w+'：既能向文件中写入数据，又能从文件读取数据。

它们的区别在于，当打开一个不存在的文件时，'r+'会报错，但是'w+'会创建这个文件，如果打开一个已经存在的文件，'w+'会把原文件的内容清空。

在 Python 中与文件内容写入有关的两个常用方法如表 3.3 所示。在进行文件内容的写入时应保证 open 函数中文件的打开方式是非只读的，例如 r+、w、w+、a 或 a+等。

表 3.3 文件内容的写入方法

方　法	作　用
<file>.write(str)	将字符串 str 写入文件
<file>.writelines(strList)	将字符串列表 strList 写入文件

注意：将字符串列表写入文件的 writelines 方法相当于一次往文件中写入多行数据，但该方法不会自动给各行添加换行符。示例 3.2 的输出结果如图 3.6 所示。

图 3.6 示例 3.2 的文件写入效果

如果要实现换行效果,可以在字符串列表的每个元素后添加换行符'\n'。

【例 3.3】 将列表中的各个元素换行写入文本文件。

```
urlList = ['https://www.buu.edu.cn' + '\n', 'https://www.baidu.com' + '\n']
f = open('files/urls2.txt','w')
f.writelines(urlList)
f.close()
```

输出效果如图 3.7 所示。

图 3.7 示例 3.3 的文件写入效果

2. 读取文本文件

Python 读取文本文件的步骤如下:

(1) 使用 open 函数以读取模式打开文本文件,获得文件对象。

(2) 调用文件对象的 read、readline 或 readlines 方法读取文件中的内容。

(3) 调用文件对象的 close 方法关闭文件。

读取和存储文本文件的步骤相似,主要区别在于第二步是调用文件对象的读取方法。Python 中常见的 3 个读取文件的方法如表 3.4 所示。

表 3.4 文件内容的读取方法

方 法	作 用
<file>.read()	读取文件中的所有内容,返回一个字符串或字节流
<file>.readline()	读取文件中的一行内容,返回一个字符串或字节流
<file>.readlines()	读取文件中的所有行内容,返回以每行为元素的列表

下面 3 个示例分别展示了这 3 个读取方法的效果。

【例 3.4】 读取文本文件的所有内容。

```
f = open('files/data.txt')
data = f.read()
print(data)
f.close()
```

结果显示为:

```
Python 数据分析实践课程理论和实践相结合,助力在业务领域获得数据,分析和处理数据。
```

【例 3.5】 读取文本文件的一行内容。

```
f = open('files/urls2.txt')
line = f.readline()
```

```
print(line)
f.close()
```

结果显示为：

```
https://www.buu.edu.cn
```

【例 3.6】 读取文本文件的所有行内容。

```
f = open('files/urls2.txt')
lines = f.readlines()
print(lines)
f.close()
```

结果显示为：

```
['https://www.buu.edu.cn\n', 'https://www.baidu.com\n']
```

3. 存取文本文件的简便方法

文件对象的 close 方法表示关闭文件对象。每次对文件操作完毕后都要执行 close 方法，以便释放文件资源。为了避免遗忘该操作，在实际使用中一般采用 with as 语句来操作上下文管理器，帮助系统自动分配和释放资源，因此有了文件存取的简便写法：

```
with open() as f:
    程序语句
```

通过 with open() as f 语句创建了文件句柄，所有和文件相关的操作都在该语句块下执行。下面的代码和示例 3.4 是等价的。

```
with open('files/data.txt') as f:
    data = f.read()
    print(data)
```

扫一扫

视频讲解

3.2.2 CSV 文件的存取

CSV 是一种通用的、相对简单的文件格式，以纯文本形式存取表格数据，是电子表格、数据库最常见的导入和导出格式，被用户、商业和科学广泛应用。CSV 文件本质上是一个字符序列，可以由任意数目的记录组成，记录间以某种分隔符分隔成字段。每条记录由若干字段组成，字段间的分隔符最常见的是逗号或制表符。所有记录都有完全相同的字段序列，Python 使用 csv 库实现对 CSV 文件的存取。

使用 csv 库中的 reader 和 writer 方法生成对象可以读/写字符序列，也可以用 DictReader 和 DictWriter 方法生成对象读/写字典类型的数据。

1. 存储 CSV 文件

Python 使用 csv 库存储 CSV 文件的步骤如下：

(1) 使用 open 函数以写入模式获得要写入的文件对象。

(2) 调用 csv 库的 writer 方法初始化写入对象,生成 writer 对象。

(3) 调用 writer 对象的 writerow 或 writerows 方法传入每行或所有行数据。

writer 方法返回一个 writer 对象,该对象将用户的数据在给定的文件类对象上转换为带分隔符的字符串,其语法如下:

```
csv.writer(csvfile , dialect = 'excel', ** fmtparams)
```

常见参数的说明如下。

(1) csvfile: 必须是支持迭代(Iterator) 的对象,可以是文件对象或列表对象,如果是文件对象,需要在生成该文件对象的 open 函数中使用参数 newline=''。

(2) dialect: 用于指定 CSV 的格式模式,不同程序输出的 CSV 格式有细微差别,默认是 Excel 程序风格。

(3) fmtparams: 格式化参数,用来覆盖之前 dialect 对象指定的程序风格。例如 delimiter 参数用于分隔字段的单字符字符串,默认为','。

表 3.5 列出了 writer 对象的两个写入方法。

表 3.5 writer 对象的写入方法

方法或属性	作 用
<writer>.writerow(row)	写入一行数据
<writer>.writerows(rows)	写入多行数据

下面通过几个示例详细学习如何进行 CSV 文件的存储。

【例 3.7】 每次写入一行数据到 CSV 文件中。

```
with open('files/data1.csv','w',newline="") as f:
    writer = csv.writer(f)
    writer.writerow(['产品 ID','产品名称','生产企业','价格'])
    writer.writerow(['0001','小米','小米',1999])
    writer.writerow(['0002','OPPO Reno','OPPO',2188])
    writer.writerow(['0003','荣耀手机','华为',3456])
```

保存在 data1.csv 文件中的数据如图 3.8 所示。

例 3.7 使用 with open as 获得写入文件对象 f,然后生成 writer 对象,接着调用了 4 次 writerow 方法写入了 4 行数据,其中第一行是标题。

	A	B	C	D
1	产品ID	产品名称	生产企业	价格
2	1	小米	小米	1999
3	2	OPPO Reno	OPPO	2188
4	3	荣耀手机	华为	3456
5				

图 3.8 CSV 文件写入效果

注意: 在默认情况下,writerow 方法会在每写入一行后加一个空行,为避免这种情况发生,需要在 open 中设置参数 newline='',另外如果写入的中文显示为乱码,还需要在 open 函数中设置 encoding 参数。

【例 3.8】 每次写入多行数据到 CSV 文件中。

```
with open('files/data2.csv','w',newline="")  as f:
    writer = csv.writer(f,delimiter=';')
    writer.writerow(['产品 ID','产品名称','生产企业','价格'])
    writer.writerows([['0001','iPhone9','Apple',9999],
                      ['0002', 'OPPO Reno','OPPO',2188],
                      ['0003', '荣耀手机', '华为', 3456]])
```

与例 3.7 不同，例 3.8 在生成 writer 对象后分别调用了一次 writerow 方法写入第一行标题，调用一次 writerows 方法写入 3 行数据。另外，在 writer 方法中设置 delimiter=';'，表明分隔字段的字符是“;”。

csv 库除了写入列表类型的数据以外，还可以写入字典类型数据，此时需要调用 csv 库的 DictWriter 方法，其语法格式如下：

```
csv.DictWriter(csvfile, fieldnames)
```

常见参数的说明如下。

(1) csvfile：必须是支持迭代(Iterator) 的对象，可以是文件对象，也可以是列表对象，如果是文件对象，需要在生成该文件对象的 open 函数中使用参数 newline=''。

(2) fieldnames：一个字典 keys 的序列，用于标识 writerow 方法传递字典中的值的顺序。

【例 3.9】 写入字典类型的数据到 CSV 文件中。

```
with open('files/data4.csv','w',newline="") as f:
    fieldnames = ['产品 ID','产品名称','生产企业','价格']
    writer = csv.DictWriter(f,fieldnames=fieldnames)
    writer.writeheader()                    #写入标题字段名
    #写入一行数据
    writer.writerow({'产品 ID': '0001', '产品名称': '小米', '生产企业': '小米', '价格': 1999})
    #写入多行数据
writer.writerows([{'产品 ID': '0002', '产品名称': 'OPPO Reno', '生产企业': 'OPPO', '价格': 2188},
{'产品 ID': '0003', '产品名称': '荣耀手机', '生产企业': '华为', '价格': 3456}])
```

从例 3.9 可以看出使用 DictWriter 方法生成的 writer 对象，在写入标题时要调用 writeheader 方法，在写入数据时可以用 writerow 方法一次写入一行数据，也可以用 writerows 方法一次写入多行数据。

2. 读取 CSV 文件

Python 使用 csv 库读取 CSV 文件的步骤如下：

(1) 使用 open 函数以读取模式获得要读取的 CSV 文件对象。

(2) 调用 csv 库的 reader 方法读取文件句柄，得到读取文件对象。

(3) 对读取文件对象进行遍历，读取每一行数据。

reader 方法用于文件的读取，返回一个 reader 对象，其语法格式如下：

```
csv.reader(csvfile, dialect='excel', **fmtparams)
```

常见参数的说明如下。

(1) csvfile：文件对象或者 list 对象。

(2) dialect：用于指定 CSV 的格式模式，不同程序输出的 CSV 格式有细微差别。

(3) fmtparams：一系列参数列表，主要用于设置特定的格式，以覆盖 dialect 中的格式。

【例 3.10】 读取 CSV 文件中的数据。

```
import csv
with open('files/data1.csv','r') as f:
```

```
    reader = csv.reader(f)                          #生成 reader 对象
    for row in reader:
        print(row)                                  #读取的数据为列表形式
```

读取的结果为列表,如下所示:

```
['产品 ID', '产品名称', '生产企业', '价格']
['0001', '小米', '小米', '1999']
['0002', 'OPPO Reno', 'OPPO', '2188']
['0003', '荣耀手机', '华为', '3456']
```

3. 存储中国 A 股上市公司数据的实战

在 3.1 节中爬取并解析出了结构化的中国 A 股上市公司的相关数据,在掌握了 csv 库中对 CSV 文件的存储和读取方法后,就可以将解析出来的数据存储在 CSV 文件中。

【实战案例代码 3.2】 中国 A 股上市公司的数据的存储。

```
import csv
def saveCSV(data) :                                 #定义保存 CSV 文件的方法
    with open("files/stockData.csv","w",newline = "") as f:
        writer = csv.writer(f)                      #创建 writer 对象
        writer.writerows(data)                      #写入列表数据
saveCSV(tableData)                                  #调用方法保存解析出来的数据
```

这里采用模块化的思想,定义了一个方法 saveCSV 用于将数据保存到 CSV 文件,将二维列表 data 作为方法的参数。在前面的实战中最终解析的数据存储在二维列表 tableData 中,因此调用 saveCSV 方法将 tableData 作为实参传入,数据存储的部分结果如图 3.9 所示。

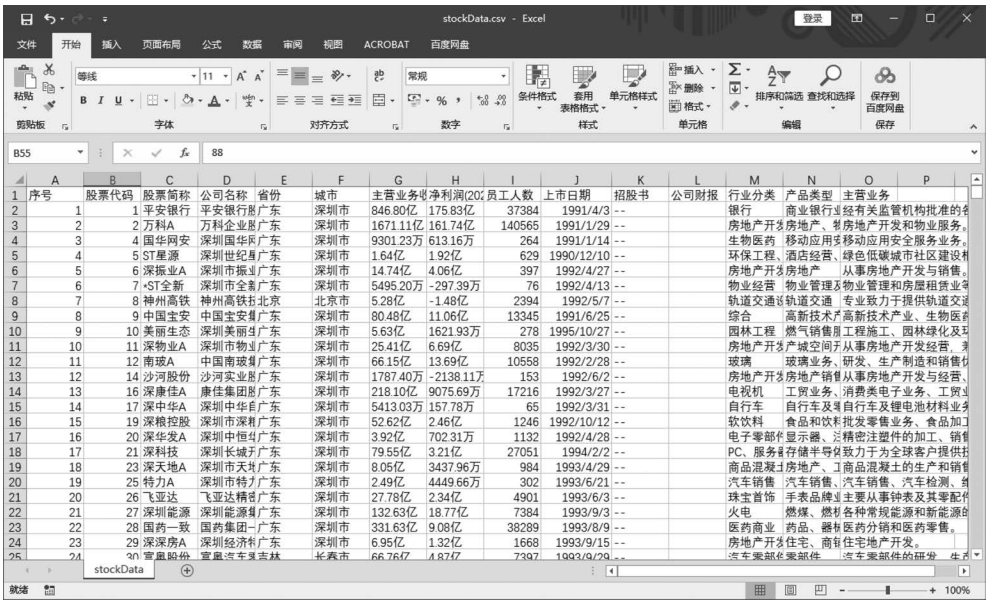

图 3.9 中国 A 股上市公司相关数据的存储结果

扫一扫

视频讲解

3.2.3 JSON 文件的存取

JSON 的全称为 JavaScript Object Notation，它是 JavaScript 对象标记，通过对象和数组的组合来表示数据，构造简洁，但是结构化程度非常高，是一种轻量级的数据交换格式。

使用 json 库，Python 可以很方便地对 JSON 文件进行存取。

1. 对象和数组

JSON 对象在 JavaScript 中是使用大括号“{ }”括起来的内容，数据结构为{key1：value1，key2：value2，…}的键值对结构。

- key 必须是字符串，value 可以是合法的 JSON 数据类型，包括字符串、数字、对象、数组、布尔值或 null。
- key 和 value 使用冒号“：”分隔，每个键值对使用逗号分隔。

JSON 对象的用法类似于 Python 中的字典类型数据。

JSON 数组在 JavaScript 中是使用中括号“[ ]”括起来的内容，数据结构为类似["java"，"javascript"，"Python"，…]的索引结构。使用中括号括起来的值可以是任意类型。

JSON 数组的用法类似于 Python 中的列表类型数据。

JSON 可以由以上两种形式自由组合而成，可以无限次嵌套，结构清晰，是数据交换的极佳方式。

```
[{ "name":"小米","price":1999, "count":3000},
{"name":"华为","price":2999, "count":122}]
```

2. 存储 JSON 文件

Python 使用 json 库存储 JSON 文件的步骤如下：

(1) 使用 json 库的 dumps 方法将 JSON 对象转换为字符串。

(2) 使用 open 函数以写入模式获得要写入的文件句柄。

(3) 调用文件句柄的 write 方法将①中转换后的字符串写入文件。

dumps 方法用于将对象编码成 JSON 字符串格式。其语法格式如下：

```
dumps(obj, ensure_ascii = True, indent = None,sort_keys = False)
```

常见参数的说明如下。

(1) obj：JSON 的对象。

(2) ensure_ascii：默认值为 True，如果 obj 内含有非 ASCII 字符，则会以 UTF-8 编码值的形式显示数据，类似\uXXXX，设置成 False 后，可以正常显示字符。

(3) indent：一个非负的整数值，如果是 0 或者为空，则显示的数据没有缩进格式，且不换行；如果设为大于 0 的整数值，则会换行且缩进 indent 指定的数值，便于 JSON 数据进行格式化显示。

(4) sort_keys：将数据根据 key 值进行排序。

存储 JSON 文件需要先使用 dumps 方法将 JSON 对象转换成字符串，然后使用常规的文件写入操作把转换好的字符串写入 JSON 文件中。

【例 3.11】 存储数据到 JSON 文件中。

```
import json
data = [{"name":"小米","price":"1999","count":"3000"},
        {"name":"华为","price":"2999","count":"122"}]   #要存储的对象
#将 JSON 对象编码为 JSON 字符串
jsonData = json.dumps(data,indent = 2,ensure_ascii = False)
with open("files/data.json","w") as f:                    #打开 JSON 文件,将 JSON 字符串写入文件
    f.write(jsonData)
```

例 3.11 使用 dumps 方法设置了 indent=2,表明在实现数据存储时可以自动换行,且每行缩进两个字符,如果不做该设置,存储在文件中的数据将在一行显示。另外,因为数据中有中文字符,为了能正常显示出中文,需要设置 ensure_ascii=False,否则将显示中文字符对应的 UTF-8 编码。数据存储到 JSON 文件的结果如图 3.10 所示。

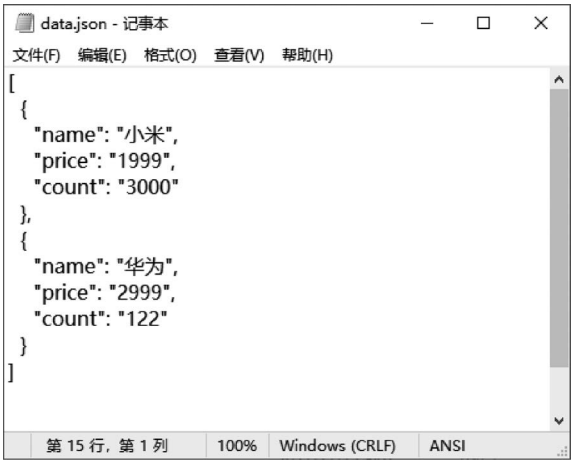

图 3.10 JSON 文件的存储结果

3. 读取 JSON 文件

Python 使用 json 库读取 JSON 文件的步骤如下:

(1) 使用 open 函数以读取模式获得要读取的 JSON 文件句柄。

(2) 使用文件句柄的 read 方法读取文件得到字符串。

(3) 调用 json 库的 loads 方法将字符串转化为 JSON 对象。

loads 方法用于将已编码的 JSON 字符串解码为 JSON 对象。其语法格式如下:

```
loads(str)
```

其中,str 是已编码的 JSON 字符串,例如'{"a":1,"b":2,"c":3,"d":4,"e":5}'。

读取 JSON 文件,先用常规的读取文件操作得到字符串,然后用 json 库中的 loads 方法将字符串转换为 JSON 对象。

【例 3.12】 读取 JSON 文件中的数据。

```
import json
with open("files/data.json","r") as f:
    str = f.read()
```

```
data = json.loads(str)                    #将字符串解码为 JSON 对象
print(data)
```

程序的输出结果为列表类型数据，列表中的每个元素为字典类型数据，如下所示：

```
[{'name': '小米', 'price': '1999', 'count': '3000'}, {'name': '华为', 'price': '2999', 'count': '122'}]
```

3.3 实战：豆瓣读书 Top250 的数据的获取

本节进行半结构化数据的获取——编写爬虫获取豆瓣读书 Top250 的相关数据，数据来源于豆瓣读书 Top250(https://book.douban.com/top250)，如图 3.11 所示。本实战的任务是爬取排行榜中每本图书的具体信息，存储在 JSON 文件中。

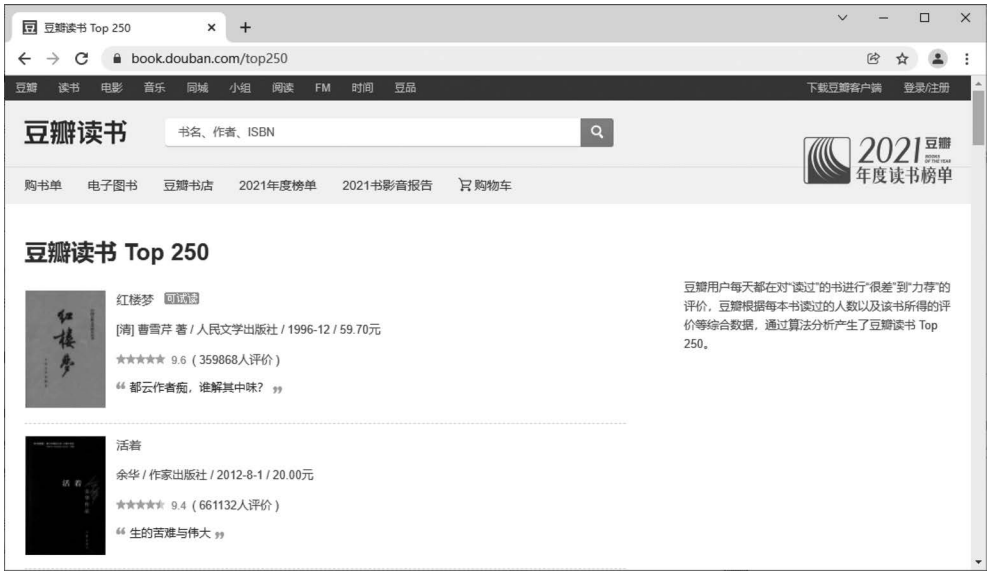

图 3.11 豆瓣读书 Top250 的主页

扫一扫

视频讲解

3.3.1 目标网站分析

1. 查看 robots 协议

查看豆瓣读书的 robots 协议，了解网站是否允许爬虫爬取豆瓣读书 Top250 的数据。在浏览器的地址栏中输入网址“https://book.douban.com/robots.txt”，协议的具体内容如图 3.12 所示，豆瓣读书网站没有禁止对 Top250 目录下资源的爬取。

2. 使用 Chrome 工具进行网站分析

使用 Chrome 工具的 Network 面板查看发送请求的相关内容，包括 URL、请求类型、分页 URL 的特点、请求头中的 User-Agent 信息等，如图 3.13 所示。

查看到的具体信息如表 3.6 所示。

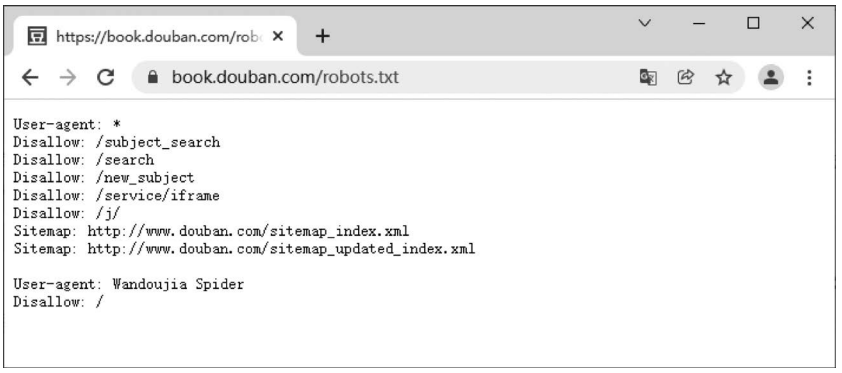

图 3.12　豆瓣读书网站的 robots 协议

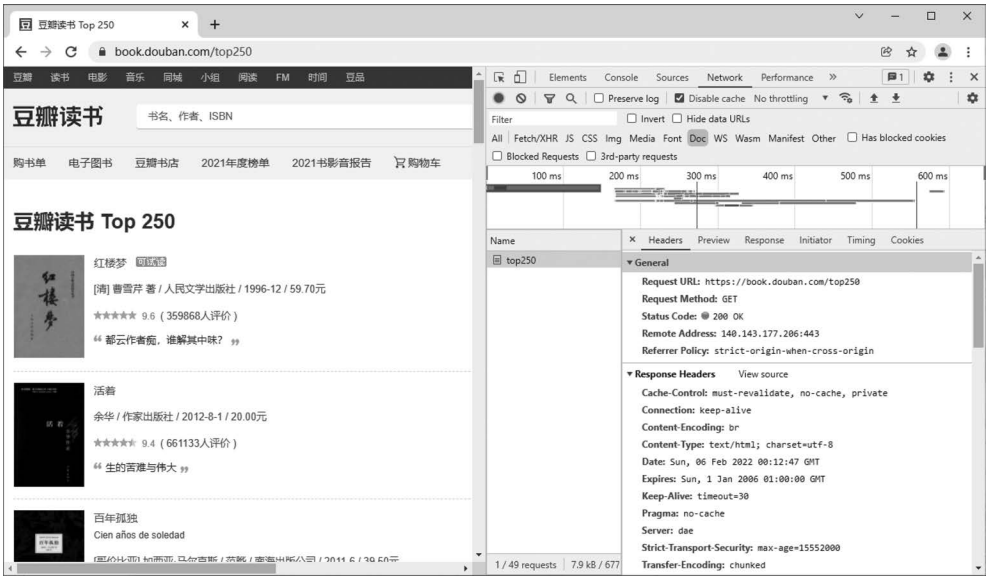

图 3.13　Chrome 工具中豆瓣读书 Top250 的 Network 面板内容

表 3.6　预分析获得的信息

类　　型	内　　容
请求 URL 基础地址	https://book.douban.com/top250
请求类型	GET 请求
分页 URL 的特点	https://book.douban.com/top250?start=25 https://book.douban.com/top250?start=50
请求头中的 User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.99 Safari/537.36

使用 Chrome 工具的 Elements 面板对数据所在网页的特征进行分析，如图 3.14 所示。可以看出在网页页面中每本书的信息都在一对 table 标签中，具体来看，在 table 标签下仅有一对 tr 标签，tr 标签中包括两对 td 标签，其中，第一个 td 标签包含书籍详情的链接 URL 地址和书的封面图片 URL 地址；第二个 td 标签包含书名、作者、出版社、出版时间、定价、豆瓣评分、参与评价人数、一句话书评等相关信息。

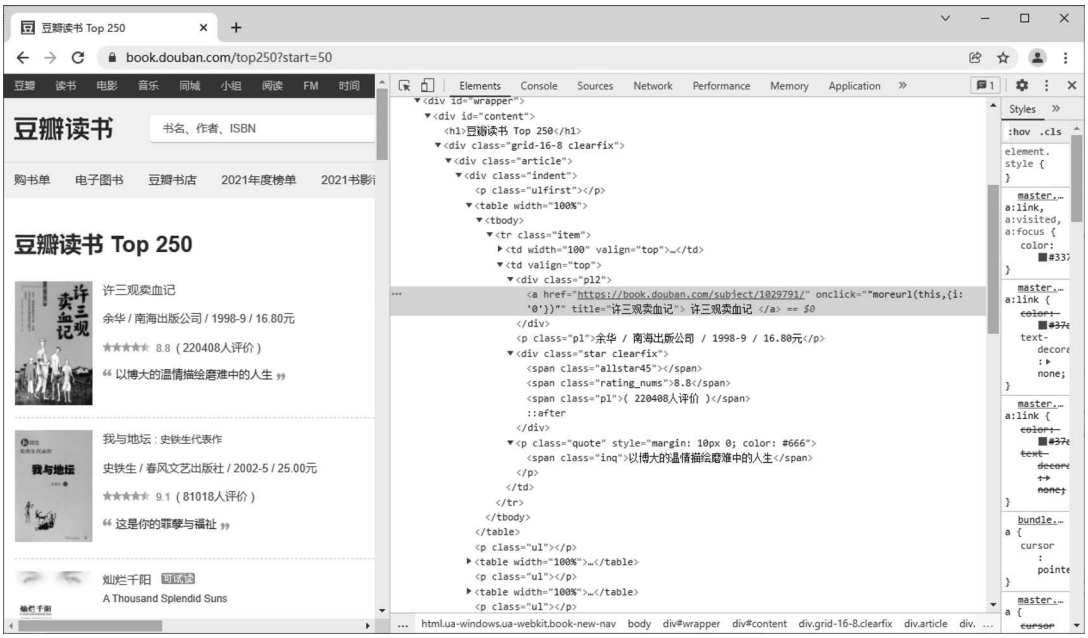

图 3.14 Chrome 工具中豆瓣读书 Top250 的 Elements 面板内容

3.3.2 半结构化数据的爬取、解析和存储

在对目标网站进行预分析后，就可以编写代码对半结构化数据进行爬取、解析和存储，具体如下。

(1) 用 requests 库模拟用户请求对网页数据进行爬取。

(2) 用 BeautifulSoup 库对网页中的书籍信息进行解析。

(3) 用 json 库将解析出来的数据保存为 JSON 文件。

1. 模拟发送请求、爬取数据

豆瓣读书 Top250 上的数据是分页显示的，URL 的请求参数为 start，页码从 0 开始，start 参数对应的值是页码的 25 倍，设计请求页面方法 getHTML(num)，其中 num 实际取值是 25 的倍数。

```
import requests
def getHTML(num) :                    #定义发送请求、爬取数据的方法
    url = 'https://book.douban.com/top250'
    header = {
    'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64)  '\
    'AppleWebKit/537.36 (KHTML, like Gecko)  Chrome/80.0.3987.163 Safari/537.36'
    }
    r = requests.get(url,headers = header,params = {"start":num})
    return r.text
```

调用该方法传入参数，可得到对应页面的数据，下面的代码获取了豆瓣读书第 2 页的数据。

```
html = getHTML(25)
print(html[:1000])
```

输出前 1000 个字符的信息,结果如图 3.15 所示。

```
<!DOCTYPE html>
<html lang="zh-cmn-Hans" class="ua-windows ua-webkit book-new-nav">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>豆瓣读书 Top 250</title>

<script>!function(e){var o=function(o,n,t){var c,i,r=new Date;n=n||30,t=t||"/",r.setTim
e(r.getTime()+24*n*60*60*1e3),c="; expires="+r.toGMTString();for(i in o)e.cookie=i+"="+
o[i]+c+"; path="+t},n=function(o){var n,t,c,i=o+"=",r=e.cookie.split(";");for(t=0,c=r.l
ength;t<c;t++)if(n=r[t].replace(/^\s+|\s+$/g,""),0==n.indexOf(i))return n.substring(i.l
ength,n.length).replace(/\"/g,"");return null},t=e.write,c={"douban.com":1,"douban.fm":
1,"google.com":1,"google.cn":1,"googleapis.com":1,"gmaptiles.co.kr":1,"gstatic.com":
1,"gstatic.cn":1,"google-analytics.com":1,"googleadservices.com":1},i=function(e,o){var
n=new Image;n.onload=function(){},n.src="https://www.douban.com/j/except_report?kind=ra
022&reason="+encodeURIComponent(e)+"&environment="+encodeURIComponent(o)},r=function(o)
{try{t.call(e,o)}catch(
```

图 3.15 爬取网页中前 1000 个字符的信息

2. 解析数据

在实现数据解析时,首先定义 getPrintData 方法完成网页数据的解析及打印输出,其中参数 html 是调用 getHTML 方法得到的以字符串表示的网页信息。其具体的代码如下:

```
from bs4 import BeautifulSoup
def getPrintData(html):
    soup = BeautifulSoup(html,"lxml")                    #将 HTML 页面封装成文档树
    books = soup.select("tr")                            #提取页面中所有的 tr 标签
    #对每个 tr 标签进行遍历
    for book in books:
        tds = book.select("td")                          #提取当前 tr 标签下的所有 td 标签
        print("书名:",tds[1].div.a.text.strip() .split("\n") [0])
        print("书籍详情:",tds[0].a.get("href") )
        print("封面:",tds[0].img.get("src") )
        print("出版信息:",tds[1].p.text)
        #提取第二个 td 标签下所有带有 class 属性的 span 标签
        spans = tds[1].select("span[class]")
        print("评分:",spans[1].text)
        print("评论人数:",spans[2].text.replace("(","") .replace(") ","") .strip() )
        if len(spans) == 4:
            print("备注:",spans[3].text)
        print("-----------------------------------")
```

在定义的方法内,使用 BeautifulSoup 函数得到文档树之后,调用 soup 对象的 select 方法查找相关标签内容,具体操作如下:

(1) 通过 soup.select("tr")方法查找到页面中所有的 tr 标签,每个 tr 标签中的内容就是一本书的详细信息。

(2) 将查找结果保存为列表 books。

(3) 遍历 books 列表,即遍历每一对 tr 标签。

(4) 在遍历时,先用 book.select("td")提取 tr 中的 td 标签,将其结果存储在列表 tds 中。由前面的分析可知,列表 tds 中只包括以下两个元素。

- tds[0]:包括书籍详情和书籍封面的 URL 地址信息。
- tds[1]:包括书名、出版信息(作者、出版社、出版时间、定价)、评分、评价人数、备注信息。

然后根据各项数据所在标签的特征进行数据的提取。

(5) 评分、评价人数以及备注信息在 td 标签下带有 class 属性的 span 标签中,调用 tds[1].select("span[class]")提取第二个 td 标签下所有带有 class 属性的 span 标签,将其存放在列表 spans 中。

(6) 通过页面分析发现,第二个 span 标签显示的是评分信息,第三个 span 标签显示的是评价人数,但是有些书籍没有备注信息,也就意味着有些书籍只显示 3 个 span 标签,有的书籍显示 4 个 span 标签,如果有备注信息,则在第四个 span 标签中显示,因此显示备注信息是增加一个条件判断 if len(spans) ==4。

执行调用:

```
getPrintData(html)
```

部分结果如图 3.16 所示,其中 html 是爬取的第 2 页的网页文档信息。

```
书名:  基督山伯爵
书籍详情: https://book.douban.com/subject/1085860/
封面:  https://img9.doubanio.com/view/subject/s/public/s3248016.jpg
出版信息: 大仲马 / 周克希 / 上海译文出版社 / 1991-12-1 / 43.90元
评分:  9.0
评论人数:  117470人评价
备注: 一个报恩复仇的故事,以法国波旁王朝和七月王朝为背景
--------------------------------------------------
书名:  肖申克的救赎
书籍详情: https://book.douban.com/subject/1829226/
封面:  https://img9.doubanio.com/view/subject/s/public/s4007145.jpg
出版信息: [美] 斯蒂芬·金 / 施寄青 / 人民文学出版社 / 2006-7 / 29.90元
评分:  9.1
评论人数:  105032人评价
备注: 豆瓣电影Top1原著
--------------------------------------------------
书名:  嫌疑人X的献身
书籍详情: https://book.douban.com/subject/3211779/
封面:  https://img9.doubanio.com/view/subject/s/public/s3254244.jpg
```

图 3.16 豆瓣读书 Top250 页面的解析和打印效果

为了后续将数据存储为 JSON 文件,在 getPrintData 方法的基础上创建 getListData(html)方法,将解析出来的数据先保存为 JSON 数据对象。具体为在方法中增加一个列表 booklist 保存所有书籍信息,每本书籍的信息用字典 bookdic 来保存,代码如下:

```
def getListData(html):
    booklist = []                                    #定义列表保存所有书籍信息
    soup = BeautifulSoup(html,"lxml")
    books = soup.select("tr")
    for book in books:
        bookdic = {}                                 #定义字典保存每本书籍的信息
        tds = book.select("td")
        bookdic["书名"] = tds[1].div.a.text.strip() .split("\n") [0]
```

```
            bookdic["书籍详情"] = tds[0].a.get("href")
            bookdic["封面"] = tds[0].img.get("src")
            bookdic["出版信息"] = tds[1].p.text
            spans = tds[1].select("span[class]")
            bookdic["评分"] = spans[1].text
            bookdic["评论人数"] = spans[2].text.replace("(","") .replace(") ","") .strip()
            if len(spans) == 4:
                bookdic["备注"] = spans[3].text
            booklist.append(bookdic)      #将字典元素添加到 booklist 列表中
        return booklist   #返回 booklist
```

3. 保存数据

定义 saveJson 方法保存解析的数据为 JSON 文件,在该方法中有以下 3 个参数。

- data:解析出来的数据。
- path:用户指定的文件存储路径。
- filename:用户指定的文件名。

为了帮助创建用户指定的系统中不存在的文件路径和文件名,此处引入 os 库,具体代码如下:

```
import json
import os
def saveJson(data,path,filename) :
    jData = json.dumps(data,indent = 2,ensure_ascii = False)
    if not os.path.exists(path) :                    #判断文件路径不存在
        os.makedirs(path)                            #如果不存在指定的文件路径,则新建
with open(path + filename,"w",encoding = "utf - 8") as f:
       f.write(jData)
```

3.3.3 模块化程序的编写

在豆瓣读书 Top250 排行榜中共有 250 本书籍信息,分 10 个页面显示。这里设计一个列表 allbooks,对每个页面的书籍信息进行爬取、解析并存储在一个页面的列表后,将每个页面的书籍列表扩展到 allbooks 中。下面是获取豆瓣读书 Top250 排行榜的相关数据的实战案例代码。

【实战案例代码 3.3】 获取豆瓣读书 Top250 排行榜的相关数据。

```
import requests
from bs4 import BeautifulSoup
import json
import os
allbooks = [ ]                          #存储所有页面的书籍信息
for i in range(10) :                    #10 个页面
    #调用 getHTML 方法爬取当前页面,返回 HTML 字符串
    html = getHTML(i * 25)
    #调用 getListData 方法解析当前页面,返回存储当前页面所有书籍信息的列表
    page = getListData(html)
```

```
            allbooks.extend(page)        #将列表 page 扩展到 allbooks 中
    #保存所有数据到 JSON 文件
    saveJson(allbooks,"mdata/","douban250.json")
    #定义发送请求爬取数据的方法
    def getHTML(num):
        url = 'https://book.douban.com/top250'
        header = {
        'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) '\
        'AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163 Safari/537.36'
        }
        r = requests.get(url,headers = header,params = {"start":num})
        return r.text
    #定义解析数据,保存在列表中的方法
    def getListData(html):
        booklist = []
        soup = BeautifulSoup(html,"lxml")
        books = soup.select("tr")
        for book in books:
            bookdic = {}
            tds = book.select("td")
            bookdic["书名"] = tds[1].div.a.text.strip().split("\n") [0]
            bookdic["书籍详情"] = tds[0].a.get("href")
            bookdic["封面"] = tds[0].img.get("src")
            bookdic["出版信息"] = tds[1].p.text
            spans = tds[1].select("span[class]")
            bookdic["评分"] = spans[1].text
            bookdic["评论人数"] = spans[2].text.replace("(","") .replace(") ","") .strip()
            if len(spans) == 4:
                bookdic["备注"] = spans[3].text
            booklist.append(bookdic)
        return booklist
    #定义保存数据到 JSON 文件的方法
    def saveJson(data,path,filename) :
        jData = json.dumps(data,indent = 2,ensure_ascii = False)
        if not os.path.exists(path) :
            os.makedirs(path)
        with open(path + filename,"w",encoding = "utf-8") as f:
            f.write(jData)
```

以上代码中的一些注意事项如下：

(1) 首页 URL 中的 start 参数值为 0,因此在做多个页面循环时,只需要使用 range(10)即可。

(2) 使用 extend 方法将 page 列表添加到 allbooks 列表,即在原有列表的尾部追加列表,实现原列表 allbooks 的扩展。

(3) 在调用 saveJson 方法中,第二个参数表示路径,应在给出的表示路径的字符串后增加“/”字符。

最终文件 douban250.json 的存储结果如图 3.17 所示。

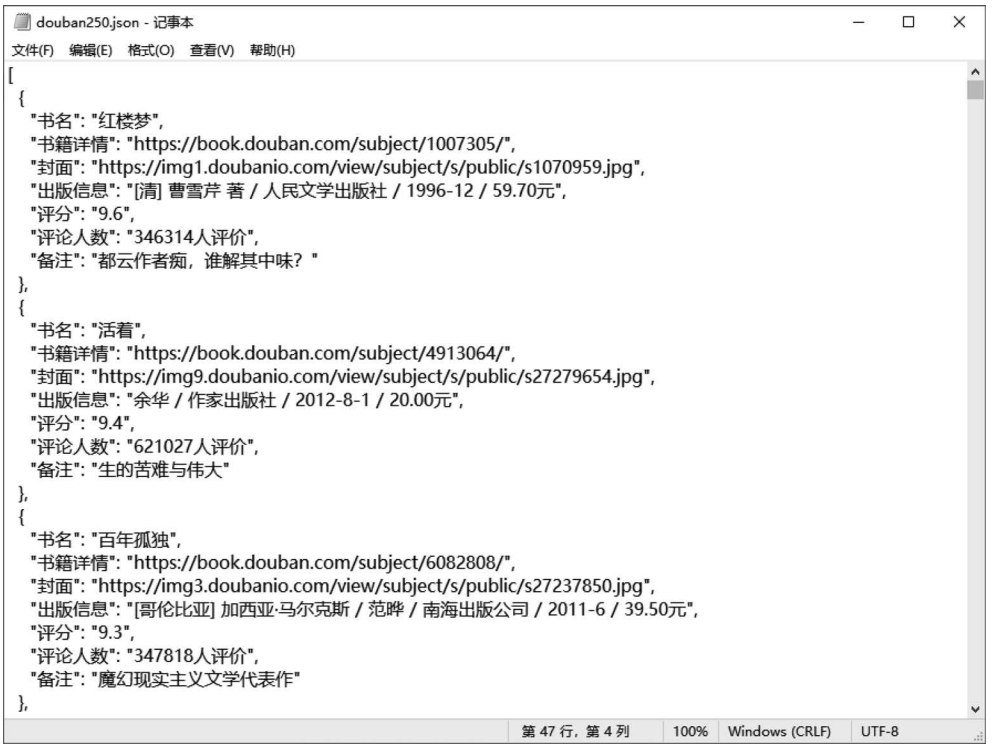

图 3.17 douban250.json 文件的存储结果

3.4 正则表达式

正则表达式是一个非常强大的字符串处理工具，几乎任何关于字符串的操作都可以使用正则表达式来完成。它是对字符串操作的一种逻辑公式，用事先定义好的一些特定字符及其组合组成一个“规则字符串”，表达对字符串的一种过滤逻辑。编写爬虫解析数据，掌握正则表达式是不可或缺的技能，它可以让数据的解析变得高效、简便。正则表达式不是 Python 独有的，Python 通过自带的 re 库提供了对正则表达式的支持。

扫一扫

视频讲解

3.4.1 正则表达式基础

正则表达式描述了一种字符串匹配的模式(pattern)，可以用来做以下操作：

- 检查一个字符串是否含有某种子串。
- 替换匹配的子串。
- 从某个字符串中取出符合某种条件的子串等。

例如，使用正则表达式\d{11}可以从下列文本中匹配出 11 位手机号码。

```
张至中,手机 15912378901,QQ 66531
刘小云,手机 15662378988,QQ 67319178
王均,手机 13452378118,QQ 3191178
```

1. 正则表达式的构成

创建正则表达式的方法和创建数学表达式的方法一样，都是用多种元字符与运算符将小的表达式结合在一起创建更大的表达式。正则表达式的组成可以是单个字符、字符集、字符范围、字符间的选择或者所有组件的任意组合。正则表达式的基本构成可以是字符、预定义字符集、数量词、边界匹配、逻辑分组等。

1）字符

掌握正则表达式需要熟悉它的特定符号的作用，表 3.7 列出了正则表达式中字符的表示。

表 3.7 字符

字　符	含　义
一般字符	匹配自身
.	匹配除“\n”以外的任何单个字符，在 DOTALL 模式中可以匹配“\n”
\	转义字符，使后一个字符改变原来的意思
[...]	字符集，对应的位置可以是字符集的任意字符，所有的特殊字符都将失去其原有的特殊含义

其中：

（1）一般字符是匹配自身的，例如 abc 的匹配结果就是 abc。

（2）.字符为匹配除换行符“\n”以外的任意字符。例如 a.c 匹配的结果可以是 abc、a&c、arc 等。

（3）\是转义字符，让它后面出现的字符失去原来的特殊作用，匹配的是字符本身。例如 a\.c 匹配的结果是 a.c。

（4）[...]是字符集，字符集中的字符可以有以下几种形式。

- 可以逐个列出，例如 a[bc]e 可以匹配 abe 或 ace。
- 可以给出范围，例如 a[b-d]e 可以匹配 abe、ace、ade。
- 字符集中的第一个字符如果是^表示取反，例如[^abc]表示不是 a、b、c 的其他字符。

2）预定义字符集

在正则表达式中有 3 对常用的预定义字符集，每一对预定义字符集的区别在于大小写不同，匹配的字符互为补集，如表 3.8 所示。

表 3.8 预定义字符集

字　符	含　义
\d	匹配数字，即[0-9]
\D	匹配非数字，即[^0-9]
\s	匹配空白字符，即[<空格>\t\r\n\f\v]
\S	匹配非空白字符，即[^\s]
\w	匹配单词字符，即[A-Za-z0-9_]
\W	匹配非单词字符，即[^\w]

其中：

（1）\d 表示匹配 0～9 共 10 个数字；\D 则匹配非数字。例如，a\dc 可以匹配 a1c；a\Dc 则可以匹配 abc。

（2）\s 匹配所有的空白字符，空格、\t、\r、\n 都能被匹配出来；\S 匹配所有的非空白字符。例如，a\sc 可以匹配 a c；a\Sc 则可以匹配 abc。

（3）\w 匹配所有的单词字符；\W 匹配所有的非单词字符。例如，a\wc 可以匹配 abc；a\Wc

则可以匹配 a c。

3) 数量词

在正则表达式中数量词是出现在字符之后的,用于专门控制匹配字符的次数,表 3.9 列出了表示数量词的符号。

表 3.9 数量词

字 符	含 义
*	匹配前一个字符 0 次或多次
+	匹配前一个字符 1 次或多次
?	匹配前一个字符 0 次或 1 次
{m}	匹配前一个字符 m 次
{m,n}	匹配前一个字符 m 到 n 次,m 和 n 可以省略:若省略 m,匹配 0 到 n 次;若省略 n,匹配 m 到无限次

其中:

(1) * 表示匹配 0 次或多次。例如,abc * 可以匹配 ab,也可以匹配 abccc。

(2) +表示匹配 1 次或多次。例如,abc+可以匹配 abc,也可以匹配 abccc。

(3) ? 表示匹配 0 次或 1 次。例如,abc? 可以匹配 ab,也可以匹配 abc。

(4) {m}表示匹配前一个字符 m 次。例如,ab{3}c 可以匹配 abbbc。

(5) {m,n}则表示匹配前一个字符 m 到 n 次。例如,ab{1,3}c 可以匹配 abc,也可以匹配 abbc 和 abbbc。

4) 边界匹配

在正则表达式中有一些符号用于边界匹配,如表 3.10 所示。

表 3.10 边界匹配

字 符	含 义
^	匹配字符串的开头,在多行模式中匹配每一行的开头
$	匹配字符串的末尾,在多行模式中匹配每一行的末尾
\A	仅匹配整个字符串的开头
\Z	仅匹配整个字符串的末尾
\b	单词边界
\B	非单词边界,即[^\b]

其中:

(1) ^表示匹配字符串的开头。例如,^ab 可以匹配字符串 abc123,但不能匹配字符串 123abc。

(2) $ 表示匹配字符串的末尾。例如,ab $ 可以匹配字符串 123cab,但不能匹配字符串 123abc。

(3) \A 仅匹配整个字符串的开头。在进行单行文本的匹配时,作用和^相同,但它不能匹配多行文本的开头。

(4) \Z 仅匹配整个字符串的末尾。在进行单行文本的匹配时,作用和 $ 相同,但它同样不能匹配多行文本的末尾。

(5) \b 匹配单词边界,即单词和符号之间的边界,这里的单词是指\w 代表的字符,包括中/英文字符和数字,符号是指中/英文符号、空格、制表符、换行符等。例如,正则表达式'\bfoo\b'匹配'foo'、'foo.'、'(foo) '、'bar foo baz',但不匹配 'foobar'或者 'foo3'。

(6) \B是\b的取非,代表的是非单词边界,即非单词和符号之间的边界,也就是\B代表的是单词与单词之间、符号和符号之间的边界。例如,正则表达式r'py\B'匹配'python'、'py3'、'py2',但不匹配'py'、'py.'或者'py!'。

5) 逻辑分组

在正则表达式中也可以实现逻辑运算和分组,表3.11列出了正则表达式中表示逻辑和分组的两个符号。

表3.11 边界匹配

字 符	含 义
|	表示|左边和右边的表达式任意匹配一个
(...)	被()括起来的表达式将作为分组

其中:

(1) |表示或者,它总是先尝试匹配左边的表达式,一旦匹配成功,则跳过匹配右边的表达式。例如,abc|def既可以匹配abc也可以匹配def。

(2) ()表示分组,从表达式左边开始每遇到一个分组的左括号"(",分组编号+1。分组表达式是一个整体,后面也可以接数量词,例如(abc){2}可以匹配abcabc。"()"中的表达式也可以加"|",表明"|"仅在该组中有效。例如,a(123|456)c可以匹配a123c,也可以匹配a456c。

正则表达式就是上述这些符号的组合。对于更多正则表达式的符号规则,可以参看re库的官方网站①。

2. 正则表达式的验证网站

在开始学习正则表达式时,在程序中直接调试验证表达式会比较麻烦,一般可以使用在线的正则表达式工具,其中比较知名的有regex101网站(https://regex101.com),如图3.18所示,它提供了正则表达式的匹配调试功能。

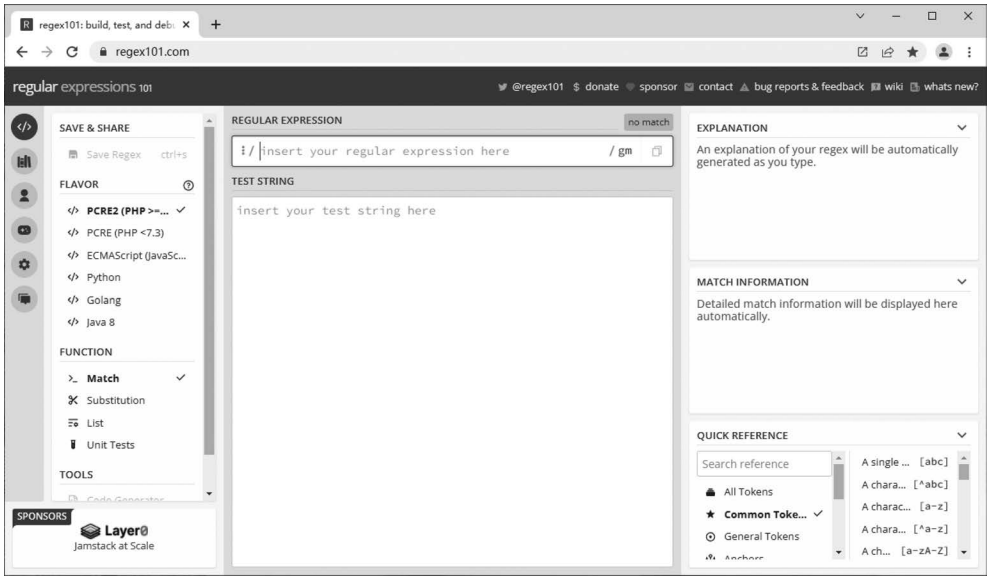

图3.18 regex101在线工具

① re库文档,https://docs.python.org/3/library/re.html#

另一个是国内 OSCHINA 网站提供的在线工具,如图 3.19 所示,它提供了一些常用的正则表达式。

图 3.19 OSCHINA 在线工具

扫一扫

视频讲解

3.4.2 正则表达式的用法

1. re 库常用函数

正则表达式指定了字符串的匹配模式,re 库是 Python 的标准库,它提供若干功能函数检测某个字符串是否与给定的正则表达式匹配,如表 3.12 所示。

表 3.12 re 库的主要功能函数

功 能	函 数	说 明
查找一个匹配项	re.search(pattern, string, flags=0)	在一个字符串中寻找第一个匹配的位置,返回匹配对象
	re.match(pattern, string, flags=0)	从一个字符串的开始位置匹配,返回匹配对象
查找多个匹配项	re.findall(pattern, string, flags=0)	搜索字符串,以列表类型返回所有匹配对象
	re.finditer(pattern, string, flags=0)	搜索字符串,返回一个匹配结果的迭代类型,每个迭代元素是匹配对象
字符串分割	re.split(pattern, string, maxsplit=0, flags=0)	将一个字符串按照正则表达式匹配结果进行分割,返回列表类型
字符串替换	re.sub(pattern, repl, string, count=0, flags=0)	在一个字符串中用 repl 替换所有匹配正则表达式的子串,返回替换后的字符串

在上述 6 个常用的功能函数中都包括 pattern、string 和 flags 参数,具体说明如下。

(1) pattern 参数:正则表达式的字符串或原生字符串表示。

(2) string 参数:待匹配的字符串。

(3) flags 参数:表示正则表达式的匹配模式。

2. 正则表达式的匹配模式

正则表达式的匹配模式用于控制正则表达式的匹配方式，如表 3.13 所示。例如将匹配模式设置为 re.S，那么在这种模式下，符号“.”就可以匹配换行符“\n”。

表 3.13 正则表达式的匹配模式

flags 常用的可选值	说 明
re.I 或 re.IGNORECASE	忽略大小写匹配，表达式[A-Z]也会匹配小写字符
re.M 或 re.MULTILINE	字符'^'匹配字符串的开始和每一行的开始；字符'$'匹配字符串的结尾和每一行的结尾
re.S 或 re.DOTALL	字符'.'匹配任何字符，包括换行符
re.A 或 re.ASCII	让转义字符集的字符(例如\w、\W 等)只匹配 ASCII，而不是 Unicode
re.X 或 re.VERBOSE	详细模式，在该模式下正则表达式可以为多行，忽略空白字符，并可以加入注释

3. re 库功能函数的使用方法

下面以 re.search 函数为例，说明 re 库中常用功能函数在进行正则表达式匹配时返回的结果，以及匹配对象的常用方法和匹配模式的设置。

1) 匹配结果

在调用 re 库的 search 函数匹配正则表达式时，如果没有匹配成功，返回 None；如果匹配成功，则返回匹配对象。

【例 3.13】 正则表达式匹配不成功的情形。

```
text = "张至中,手机 15912378901,QQ66531"
r = re.search("手 机",text)
print("匹配结果",r)
```

运行结果为：

```
匹配结果 None
```

【例 3.14】 正则表达式匹配成功的情形。

```
text = "张至中,手机 15912378901,QQ66531"
r = re.search("手机",text)
print("匹配结果",r)
```

运行结果为：

```
匹配结果 <re.Match object; span = (4, 6) , match = '手机'>
```

通过匹配对象可以查看到匹配上的第一个字符串的起始和终止位置，例如示例 3.14 中显示(4,6)，匹配上的字符串文本为'手机'。

2) 匹配对象的常用方法

匹配对象支持属性和方法,这里介绍常见的方法,对于更多方法和属性可以参考 re 库的官方文档。

(1) Match.group([group1,…]):返回一个或者多个匹配的子组。

- 默认参数值为 0,返回整个匹配结果。
- 如果设置一个参数,当值的范围是[1..99]且小于或等于正则表达式中定义的组数时,返回对应组的字符串。
- 如果参数设置为负数或大于正则表达式中定义的组数,则引发 IndexError 异常。
- 如果有多个参数,返回一个元组。

例如在示例 3.14 后执行:

```
print("匹配内容",r.group(0))
```

或执行:

```
print("匹配内容",r.group())
```

结果均为:

```
匹配内容 手机
```

(2) Match.start([group]):返回 group 匹配到的字符串的开始标号。例如在示例 3.14 后执行:

```
print("匹配结果所在起始位置",r.start() )
```

结果显示为:

```
匹配结果所在起始位置 4
```

(3) Match.end([group]):返回 group 匹配到的字符串的结束标号。例如在示例 3.14 后执行:

```
print("匹配结果所在结束位置",r.end() )
```

结果显示为:

```
匹配结果所在结束位置 6
```

(4) Match.span([group]):以二元组形式返回 group 匹配到的字符串的开始和结束标号。例如在示例 3.14 后执行:

```
print("匹配结果所在索引位置",r.span())
```

结果显示为:

```
匹配结果所在索引位置 (4, 6)
```

3）匹配模式

设置匹配模式可以改变原有特殊字符的行为，示例 3.15 设置匹配模式为 re.I，匹配字符串时可以忽略大小写。

【例 3.15】 匹配正则表达式忽略大小写。

```
text = "张至中,手机 15912000000,QQ66000"
r = re.search("Qq",text,re.I)
print("匹配结果",r)
```

结果显示为：

```
匹配结果 <re.Match object; span = (18, 20) , match = 'QQ'>
```

4. 正则表达式常见应用示例

下面使用 re 库的 search 函数，结合 3.4.1 节中介绍的正则表达式的语法基础，通过示例来了解常见的正则表达式的用法。

1）字符匹配

在实际使用中，常见的需要匹配的字符有任意字符、数字、单词字符、汉字等。

【例 3.16】 匹配任意字符。

```
text = "张至中,手机 15912000000,QQ66000"
r = re.search("张.",text)
print("匹配结果",r)
```

结果显示为：

```
匹配结果 <re.Match object; span = (0, 2) , match = '张至'>
```

【例 3.17】 匹配任意数字。

```
text = "张至中,手机 15912000000,QQ66000"
r = re.search("\d",text)
print("匹配结果",r)
```

结果显示为：

```
匹配结果 <re.Match object; span = (6, 7) , match = '1'>
```

【例 3.18】 匹配非单词字符。

```
text = "张至中,手机 15912000000,QQ66000"
r = re.search("\W",text)
print("匹配结果",r)
```

结果显示为：

```
匹配结果 <re.Match object; span = (3, 4) , match = ','>
```

【例 3.19】 匹配汉字。

```
text = "张至中,手机 15912000000,QQ66000"
r = re.search("[\u4e00 - \u9fa5]",text)
print("匹配结果",r)
```

结果显示为:

```
匹配结果 < re.Match object; span = (0, 1) , match = '张'>
```

2) 重复匹配

正则表达式中使用数量词限定符表示重复匹配时,默认是贪婪匹配,即在整个表达式得到匹配的前提下匹配尽可能多的字符。

【例 3.20】 匹配前一个字符最多次。

```
text = "张至中,手机 15912000000,QQ66000"
r1 = re.search("张. + ",text)                    # 匹配前一个字符最多次
r2 = re.search("张. * ",text)                    # 匹配前一个字符最多次
print(". + 匹配结果",r1)
print(". * 匹配结果",r2)
```

结果显示为:

```
. + 匹配结果 < re.Match object; span = (0, 25), match = '张至中,手机 15912000000,QQ66000'>
. * 匹配结果 < re.Match object; span = (0, 25) , match = '张至中,手机 15912000000,QQ66000'>
```

如果需要懒惰匹配,即在整个表达式得到匹配的前提下匹配尽可能少的字符,可以在数量词限定符(例如"+"或"*")后加上一个问号"?"。

【例 3.21】 匹配前一个字符最少次。

```
text = "张至中,手机 15912000000,QQ66000"
r1 = re.search("张. + ?",text)                   # 匹配前一个字符 1 次
r2 = re.search("张. * ?",text)                   # 匹配前一个字符 0 次
print(". + ?匹配结果",r1)
print(". * ?匹配结果",r2)
```

结果显示为:

```
. + ?匹配结果 < re.Match object; span = (0, 2) , match = '张至'>
. * ?匹配结果 < re.Match object; span = (0, 1) , match = '张'>
```

如果想实现指定次数的重复匹配,可以使用{n}或{m,n}限定符。

【例 3.22】 匹配字符串中指定位数的数字。

```
text = "张至中,手机 15912000000,QQ66000"
r = re.search("\d{11}",text)
print("匹配手机号码",r)                          # 匹配 11 位手机号码
r = re.search("QQ\d{5,8}",text)
print("匹配 QQ 号码",r)                          # 匹配 5 到 8 位 QQ 号码
```

结果显示为:

```
匹配手机号码 <re.Match object; span=(6, 17), match='15912000000'>
匹配 QQ 号码 <re.Match object; span=(18, 25), match='QQ66000'>
```

3）分组匹配

在正则表达式中，用“()”括起来表示一个分组，执行分组后向引用，即对前面出现过的分组会再一次引用，如果引用已经匹配过的分组内容，可以在 re.group 方法中通过具体的数字来引用对应的分组，例如 re.group(1)引用第一个分组，re.group(2)引用第二个分组，而 re.group(0)引用整个被匹配的字符串本身。

【例 3.23】 分组提取字符串中的数字。

```
text="张至中,手机 15912000000,QQ66000"
r=re.search("(\d+).*(\d{5,8})",text)
print("匹配结果",r)
print("分组信息",r.groups())
print("匹配全部内容",r.group(0))
print("匹配第一组内容",r.group(1))
print("匹配第二组内容",r.group(2))
```

结果显示为：

```
匹配结果 <re.Match object; span=(6, 25), match='15912000000,QQ66000'>
分组信息 ('15912000000', '66000')
匹配全部内容 15912000000,QQ66000
匹配第一组内容 15912000000
匹配第二组内容 66000
```

5. re 库功能函数应用示例

re 库对正则表达式的支持除了 search 函数以外，还有其他函数。

1）re.match 函数

re.match 函数尝试从字符串的起始位置匹配一个模式，如果不是起始位置匹配成功，返回 None。

【例 3.24】 使用 match 方法匹配字符串"手机"。

```
text="张至中,手机 15912000000,QQ66000"
r=re.match("手机",text)
print("匹配结果",r)
```

结果显示为：

```
匹配结果 None
```

2）re.findall 函数

re.findall 函数是从字符串的任意位置查找正则表达式所匹配的所有子串，返回一个所有匹配结果的列表，如果没有找到匹配的，则返回空列表。

【例 3.25】 匹配字符串中所有的 11 位手机号。

```
text='''张至中,手机 15912000000,QQ66000
```

```
刘小云,手机 15662000000,QQ67000000
王均,手机 13452000000,QQ3000000'''
r = re.findall("\d{11}",text)
print("匹配结果",r)
```

结果显示为：

```
匹配结果 ['15912000000', '15662000000', '13452000000']
```

3）re.split 函数

re.split 函数是用正则表达式匹配字符串以实现字符串的分隔,并返回一个列表。

【例 3.26】 拆分出字符串中的单词。

```
text = "text;word,key,teacher.worker"
r = re.split("[;,.]",text)
print("匹配结果",r)
```

结果显示为：

```
匹配结果 ['text', 'word', 'key', 'teacher', 'worker']
```

4）re.sub 函数

re.sub 函数将字符串中匹配正则表达式模式的内容进行替换。

【例 3.27】 提取 HTML 代码中的内容信息。

提取 HTML 代码中的内容信息就是将 HTML 中所有的便签信息替换为空。

```
text = '''<div class = "star clearfix">
                        <span class = "allstar50"></span>
                        <span class = "rating_nums">9.6</span>
                      <span class = "pl">(
                          347480 人评价
                      ) </span>
                    </div>
                    <p class = "quote" style = "margin: 10px 0; color: #666">
                        <span class = "inq">都云作者痴,谁解其中味?</span>
                    </p>'''
r = re.sub("<.*?>","",text)              #将所有的标签替换为空
print("匹配结果",r)
```

结果如图 3.20 所示。

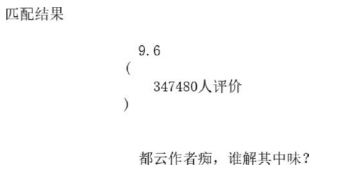

图 3.20 网页内容信息的提取结果

在使用 re.sub 函数去掉 HTML 中的所有标签以后,可以对匹配结果进行字符串的进一步处理,以便于得到具体的内容。该方式可以用于解析网页内容。

6. 正则表达式对象

使用 re 库中的 compile 函数可以将正则表达式的字符串编译转化为正则表达式对象 pattern,在编译时还可以设置 flag 匹配模式。其具体语法格式如下：

```
re.compile(string,flag = 0)
```

使用编译后的 pattern 对象进行字符串处理，不仅可以提高处理字符串的速度，还可以提供更强大的字符串处理功能。

正则表达式对象具有和 re 库同名的 search、match、findall 方法，通过正则表达式对象调用这些方法进行字符串处理，不需要每次重复写匹配模式，可以实现复用。这里以 search 函数为例：

```
re.search(regexString,string)
```

等价于：

```
pattern = re.compile(regexString)
pattern.search(string)
```

扫一扫

视频讲解

3.4.3 用正则表达式提取豆瓣读书排行榜网页数据的实战案例

在 3.3.2 节中用 BeautifulSoup 库对豆瓣读书排行榜网页的数据进行了解析，这里用正则表达式解析豆瓣读书排行榜中书籍的信息。

正则表达式提取网页中的书籍信息，需要关注要提取的书籍信息所在的字符串上下文，如图 3.21 所示，找出其中的模式，然后书写恰当的正则表达式。比如，要提取排行榜中的图书名称、出版信息、评分、评价人数以及点评信息，就需要关注这些内容所在的字符串上下文。考虑到提取信息的多样性，在正则表达式中使用分组符号“()”来提取对应的各个元素信息。具体待提取信息的特征分析如下：

```
▼<div class="pl2">
   <a href="https://book.douban.com/subject/1007305/" onclick=""moreurl(this,{i:'0'})"" title="红楼梦"> 红楼梦 </a>
   "   "
   <img src="/pics/read.gif" alt="可试读" title="可试读">
  </div>
  <p class="pl">[清] 曹雪芹 著 / 人民文学出版社 / 1996-12 / 59.70元</p>
▼<div class="star clearfix">
   <span class="allstar50"></span>
   <span class="rating_nums">9.6</span>
   <span class="pl">( 384509人评价 )</span>
   ::after
  </div>
▼<p class="quote" style="margin: 10px 0; color: #666">
   <span class="inq">都云作者痴，谁解其中味？</span>
  </p>
```

图 3.21 提取书籍信息所在的字符串上下文

- 图书名称：图书名称在该字符串中出现两次，其中在 title 属性中的信息特征明显，在字符串中具有唯一性，容易抽取出模式，这里标记出书名所在的前后字符或字符串，具体表示为 title="(.*?) "，其中.*?表示懒惰模式的任意匹配字符。
- 出版信息：根据出版信息所在前后字符串的特征，这部分正则表达式表示为 pl">(.*?)</p>。
- 评分：根据评分所在前后字符串的特征以及要提取的数字内容特征，这部分正则表达式表示为 rating_nums">(\d.\d) ，其中\d.\d 表示提取中间带有小数点的两个数字。
- 点评信息：根据评价人数所在前后字符串的特征，正则表达式表示为(\d+)人评价，其中\d+表示按照贪婪模式提取多个数字。
- 点评信息：根据点评信息所在前后字符串的特征，正则表达式表示为 inq">(.*?)。

注意：在实际操作时往往希望通过一个统一的正则表达式就能提取到上述全部内容，因此要提取的各元素特征的正则表达式之间用.*?连接，表示各元素之间有任意字符。

下面以爬取到的豆瓣读书排行榜首页的内容为例来看具体正则表达式的用法和抽取结果。

【实战案例代码 3.4】 爬取并提取豆瓣读书排行榜首页的图书信息。

```
import requests
import re
url = 'https://book.douban.com/top250'
header = {
    'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64)  '\
    'AppleWebKit/537.36 (KHTML, like Gecko)  Chrome/80.0.3987.163 Safari/537.36'
    }
def getHTML(num) :
    r = requests.get(url,headers = header,params = {"start":num})
    return r.text
html = getHTML(0)                                          #获得第1页网页内容
pattern = re.compile('\title = "(.*?) ".*?pl">(.*?) </p>.*?rating_nums">(\d.\d) </span>.
*?(\d+) 人评价.*?inq">(.*?) </span>',re.S)                 #编译正则表达式对象
items = re.findall(pattern,html)                           #查找所有匹配模式的信息
print(items)
```

结果如图 3.22 所示，在该案例中使用了 re.compile 函数预先将正则表达式编译成正则表达式对象 pattern，提高了正则表达式的匹配效率。有了正则表达式对象 pattern 以后，只需要在 re 库的各个功能函数中将原来字符串表示的正则表达式替换为 pattern 对象即可，例如这里 re.findall 函数中使用的就是 pattern 对象。

[('红楼梦', '[清] 曹雪芹 著 / 人民文学出版社 / 1996-12 / 59.70元', '9.6', '34748
9', '都云作者痴，谁解其中味？'), ('活着', '余华 / 作家出版社 / 2012-8-1 / 20.00
元', '9.4', '624013', '生的苦难与伟大'), ('百年孤独', '[哥伦比亚] 加西亚·马尔克
斯 / 范晔 / 南海出版公司 / 2011-6 / 39.50元', '9.3', '349177', '魔幻现实主义文学
代表作'), ('1984', '[英] 乔治·奥威尔 / 刘绍铭 / 北京十月文艺出版社 / 2010-4-1 /
28.00', '9.4', '192176', '栗树荫下，我出卖你，你出卖我'), ('飘', '[美国] 玛格丽特
·米切尔 / 李美华 / 译林出版社 / 2000-9 / 40.00元', '9.3', '183191', '革命时期的
爱情，随风而逝'), ('三体全集', '刘慈欣 / 重庆出版社 / 2012-1-1 / 168.00元', '9.
4', '106583', '地球往事三部曲'), ('三国演义（全二册）', '[明] 罗贯中 / 人民文学出
版社 / 1998-05 / 39.50元', '9.3', '141099', '是非成败转头空'), ('白夜行', '[日]
东野圭吾 / 刘姿君 / 南海出版公司 / 2008-9 / 29.80元', '9.1', '473266', '暗夜独行
的残破灵魂，爱与恶本就难分难舍'), ('小王子', '[法] 圣埃克苏佩里 / 马振聘 / 人民文
学出版社 / 2003-8 / 22.00元', '9.0', '653737', '献给长成了大人的孩子们'), ('福尔
摩斯探案全集（上中下）', '[英] 阿·柯南道尔 / 丁钟华 等 / 群众出版社 / 1981-8 / 5
3.00元/68.00元', '9.3', '109705', '名侦探的代名词'), ('房思琪的初恋乐园', '林奕含
/ 北京联合出版公司 / 2018-2 / 45.00元', '9.2', '269092', '向死而生的文学绝唱'),
('动物农场', '[英] 乔治·奥威尔 / 荣如德 / 上海译文出版社 / 2007-3 / 10.00元',
'9.3', '118561', '太阳底下并无新事'), ('撒哈拉的故事', '三毛 / 哈尔滨出版社 / 200
3-8 / 15.80元', '9.2', '121073', '游荡的自由灵魂'), ('天龙八部', '金庸 / 生活·读
书·新知三联书店 / 1994-5 / 96.00元', '9.1', '115649', '有情皆孽，无人不冤'),
('安徒生童话故事集', '（丹麦）安徒生 / 叶君健 / 人民文学出版社 / 1997-08 / 25.00
元', '9.2', '106456', '为了争取未来的一代'), ('平凡的世界（全三部）', '路遥 / 人
民文学出版社 / 2005-1 / 64.00元', '9.0', '282651', '中国当代城乡生活全景'), ('围
城', '钱锺书 / 人民文学出版社 / 1991-2 / 19.00', '8.9', '406756', '幽默的语言和对
生活深刻的观察'), ('霍乱时期的爱情', '[哥伦比亚] 加西亚·马尔克斯 / 杨玲 / 南海出
版公司 / 2012-9-1 / 39.50元', '9.0', '224046', '义无反顾地直达爱情的核心'), ('局
外人', '[法] 阿尔贝·加缪 / 柳鸣九 / 上海译文出版社 / 2010-8 / 22.00元', '9.0',
'174314', '人生在世，永远也不该演戏作假'), ('明朝那些事儿（1-9）', '当年明月 / 中
国海关出版社 / 2009-4 / 358.20元', '9.1', '122221', '不拘一格的历史书写'), ('沉默
的大多数', '王小波 / 中国青年出版社 / 1997-10 / 27.00元', '9.1', '123108', '沉默
是沉默者的通行证'), ('哈利·波特', 'J.K.罗琳 (J.K.Rowling) / 苏农 / 人民文学出版
社 / 2008-12-1 / 498.00元', '9.7', '55601', '从9¾站台开始的旅程'), ('追风筝的人',
'[美] 卡勒德·胡赛尼 / 李继宏 / 上海人民出版社 / 2006-5 / 29.00元', '8.9', '71134
2', '为你，千千万万遍'), ('人类简史', '[以色列] 尤瓦尔·赫拉利 / 林俊宏 / 中信出
版社 / 2014-11 / 68.00元', '9.1', '160471', '跟着人类一同走过十万年'), ('月亮和六
便士', '[英] 毛姆 / 傅惟慈 / 上海译文出版社 / 2006-8 / 15.00元', '9.0', '168964',
'有多少人会经历顿悟，就有更少的人甘愿自我放逐')]

图 3.22 正则表达式提取图书信息的结果

3.5 实战：人民网科技类新闻的获取

本节进行非结构化数据的获取——编写爬虫获取人民网的科技类新闻，数据来源于人民网的科技类新闻板块（http://scitech.people.com.cn/GB/1057/index.html），如图 3.23 所示。该实战的任务是爬取人民网科技栏目下的所有新闻文档，用 JSON 文件来保存新闻目录，用文本文件来保存新闻内容，文本文件要实现按日期归档。

图 3.23 人民网的科技栏目页面

扫一扫

视频讲解

3.5.1 目标网站分析

1. 查看 robots 协议

在浏览器的地址栏中输入网址“http://www.people.com.cn/robots.txt”，查看到如图 3.24 所示的人民网的 robots 协议，可以看出该网站支持爬虫对所有目录资源进行爬取。

图 3.24 人民网的 robots 协议内容

2. 使用 Chrome 工具进行网站分析

经过网站浏览分析，可以看出要完成目标任务，爬虫的编写其实可以分解为两个子任务，首先是获取科技新闻的列表，然后是根据新闻列表中提供的 URL 去获取对应的新闻文本。

1）查看 Network 面板

使用 Chrome 工具的 Network 面板查看访问科技新闻列表网页时发送请求的相关内容，包括 URL、请求类型、分页 URL 的特点、请求头中的 User-Agent 信息等，如图 3.25 所示。

查看到的具体信息如表 3.14 所示。

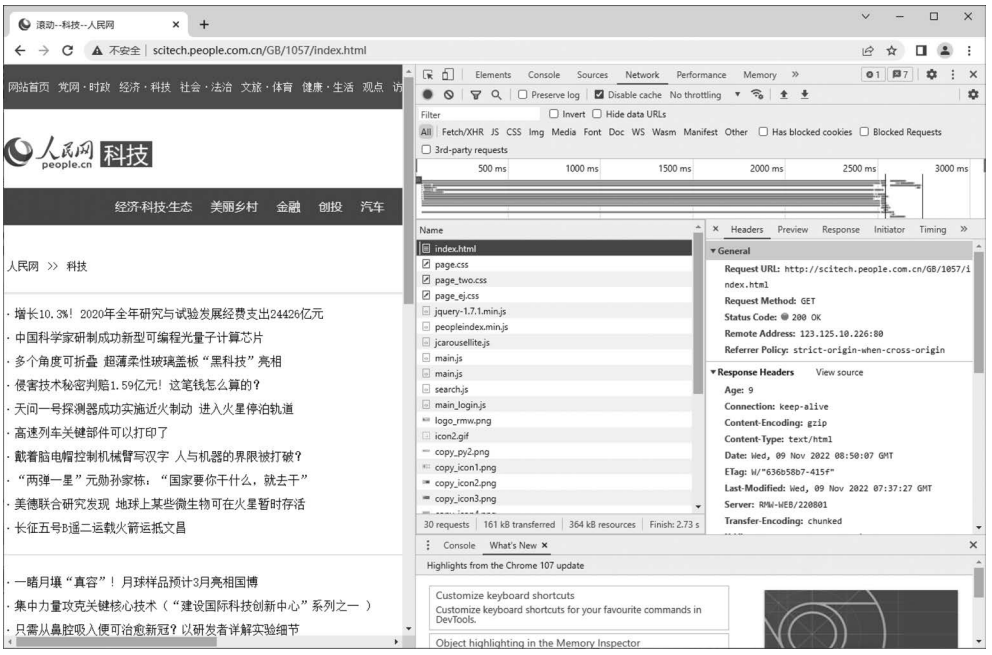

图 3.25 Chrome 工具中人民网科技新闻列表的 Network 面板内容

表 3.14 预分析获得的信息

类 型	内 容
请求 URL 基础地址	http://scitech.people.com.cn/GB/1057/index.html
请求类型	GET 请求
分页 URL 的特点	http://scitech.people.com.cn/GB/1057/index2.html http://scitech.people.com.cn/GB/1057/index3.html
请求头中的 User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.99 Safari/537.36

2) 查看 Elements 面板

使用 Chrome 工具的 Elements 面板对新闻列表所在的网页进行分析，如图 3.26 所示，可以看出在网页页面中每篇新闻的 URL 地址、标题和发布时间都在< li >标签中。

从 URL 分析可以看出，有些新闻来自金融目录 finance，有些来自文化目录 culture，因此分别查看这两类目录新闻页面的特点，如图 3.27 和图 3.28 所示，可以看出 finance 目录下新闻的主体内容，包括标题、发布时间、来源、新闻内容等在< div class="col col-1 f1">标签中，culture 目录下新闻的内容在< div class="clearfix w1000_320 text_con_left">标签中。

扫一扫

视频讲解

3.5.2 科技新闻列表的获取与存储

在对目标网站进行预分析后，可以编写代码对科技新闻列表数据进行爬取、解析及存储。采用的爬取和解析库仍然为 requests 库和 Beautifulsoup 库，在进行数据存储时，考虑后续还需要使用新闻列表中的超链接进行具体新闻内容的爬取，因此用键值对的形式保存新闻列表的信息便于检索，这里采用 json 库将解析出来的新闻列表数据保存为 JSON 文件。

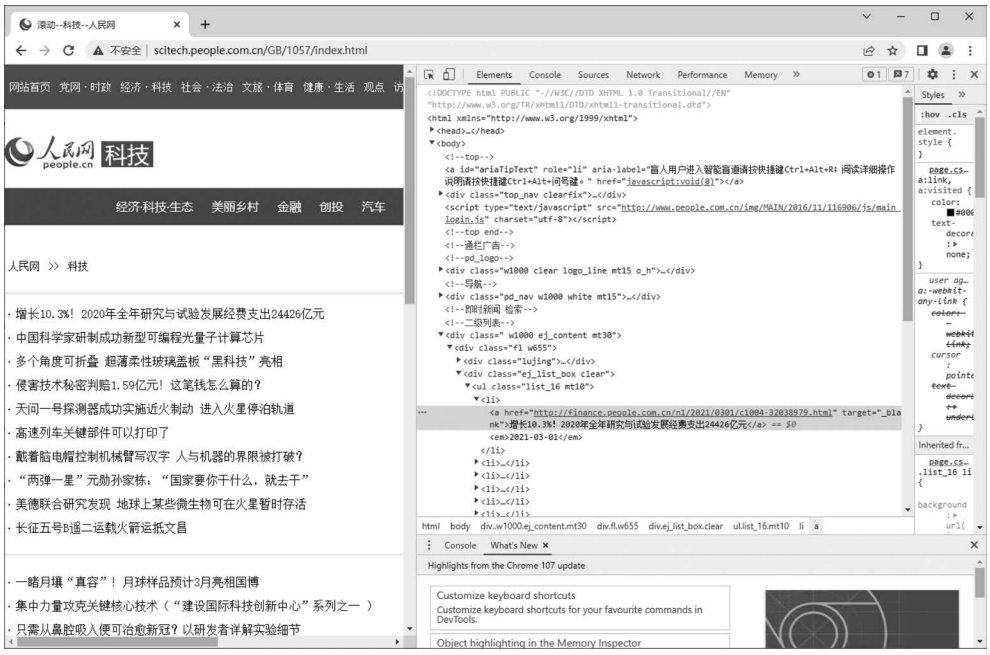

图 3.26 Chrome 工具中新闻列表网页的 Elements 面板内容

图 3.27 finance 目录下新闻页面的特点

1. 发送请求获取网页数据

人民网科技新闻列表信息呈分页显示，分页 URL 地址的特点是 index 后跟变化的数字，考虑使用字符串的 format 函数来设置具体的分页数字。设计请求新闻列表页面方法 getNewsHtml(page)，其中 page 表示具体页码。

图 3.28 culture 目录下新闻页面的特点

```
import requests
base_url = "http://scitech.people.com.cn/GB/1057/index{}.html"          #设置基础 URL
def getNewsHtml(page) :
    url = base_url.format(page)
    r = requests.get(url)
    r.encoding = r.apparent_encoding
    return r.text
```

2. 解析新闻列表数据

定义 parseNewsList(html)方法实现对新闻列表页面数据的解析，其中 html 是调用 getNewsHTML 方法得到以字符串表示的网页信息，具体代码如下：

```
from bs4 import BeautifulSoup
def parseNewsList(html) :
    soup = BeautifulSoup(html)
    pagelist = []                                    #存放本页新闻列表信息
    for  item in soup.select("li") :                 #搜索页面中所有的 li 标签
        itemdic = {}                                 #保存新闻信息
        itemdic["标题"] = item.a.text                 #提取标题
        urlitem = item.a.get("href")                 #提取超链接
        itemdic["url"] = urlitem                     #提取 RUL 地址
        itemdic["time"] = item.em.text               #提取时间
        pagelist.append(itemdic)
    return pagelist
```

定义列表类型的变量 pagelist 保存当前页面的所有新闻信息，每个新闻的信息保存在字典类型的变量 itemdic 中，包括 3 个 key，分别为标题、超链接 URL 地址和时间。因所有的新

闻列表信息都在 li 标签中，这里调用 BeautifulSoup 对象的 select 方法找到页面中所有的 li 标签，对其进行遍历，分别提取 3 个 key 所对应的 value 值。

因为科技板块中的新闻列表页共有 18 页，所以定义一个保存全部新闻列表信息的列表类型的变量 urllist，通过 for 循环执行多个页面的获取和解析，具体代码如下：

```
urllist = []
for page in range(1,18) :
    html = getNewsHtml(page)
    page = parseNewsList(html)
urllist.extend(page)
```

注意：因为此处的 page 是列表，这里调用 urllist 的 extend 方法将 page 列表中的各个元素追加到 urllist 列表的末尾。

3. 保存数据

将获取的新闻列表保存为 JSON 文件，具体的实现思路和方法同 3.3.2 节，代码如下：

```
import os
def saveJson(dic,path,filename) :
    jData = json.dumps(dic,indent = 2,ensure_ascii = False)
    if not os.path.exists(path) :
        os.makedirs(path)
    with open(path + filename,"w",encoding = "utf - 8") as f:
        f.write(jData)
#将信息保存在 files 文件夹下的 newslist.json 文件中
saveJson(urllist,"files/","newslist.json")
```

下面是全部新闻列表的获取和存储的实战案例代码。

【实战案例代码 3.5】 爬取并存储人民网科技新闻列表。

```
import json
import os
import requests
from bs4 import BeautifulSoup
#爬取新闻列表页面
def getNewsHtml(page) :
    url = base_url.format(page)
    r = requests.get(url)
    r.encoding = r.apparent_encoding
    return r.text
#解析新闻列表页面
def parseNewsList(html) :
    soup = BeautifulSoup(html)
    pagelist = []                          #存放本页新闻列表信息
    for  item in soup.select("li") :       #搜索页面中所有的 li 标签
        itemdic = {}                       #保存新闻信息
        itemdic["标题"] = item.a.text      #提取标题
        urlitem = item.a.get("href")       #提取超链接
        itemdic["url"] = urlitem           #提取 URL 地址
        itemdic["time"] = item.em.text     #提取时间
```

```
            pagelist.append(itemdic)
    return pagelist
#存储新闻列表信息为 JSON 文件
def saveJson(dic,path,filename) :
    jData = json.dumps(dic,indent = 2,ensure_ascii = False)
    if not os.path.exists(path) :
        os.makedirs(path)
    with open(path + filename,"w",encoding = "utf - 8") as f:
        f.write(jData)
##方法调用##
#设置基础 URL
base_url = "http://scitech.people.com.cn/GB/1057/index{}.html"
urllist = []
for page in range(1,18) :
    html = getNewsHtml(page)
    page = parseNewsList(html)
urllist.extend(page)
#将信息保存在 files 文件夹下的 newslist.json 文件中
saveJson(urllist,"files/","newslist.json")
```

3.5.3 新闻的获取与存储

具体新闻内容的获取要用到之前爬取到的新闻列表的相关信息。

(1) 采用模块化的处理方式将具体的获取和存储定义成 4 个方法。

(2) 依次遍历每篇新闻的 URL,提取并保存新闻信息。

1. 定义模块方法

定义的 4 个模块方法如下:

- 读取新闻列表 JSON 文件的 readJson(filename)方法。
- 发送请求获取数据的 getHtml(url)方法。
- 解析新闻文本数据的 parseNews(html)方法。
- 保存新闻文本的 saveFile(text,path,filename)方法。

其具体实现代码如下:

```
import json
import requests
from bs4 import BeautifulSoup
import os
#读取新闻列表文件
def readJson(filename) :
    with open(filename,"r",encoding = "utf - 8") as f:
        newStr = f.read()
        JData = json.loads(newStr)
    return JData
#发送请求,获取数据
def getHtml(url) :
    r = requests.get(url)
```

```
    r.encoding = r.apparent_encoding
    return r.text
#解析新闻文本数据
def parseNews(html) :
    soup = BeautifulSoup(html)
    text = ""
    #正则表达式匹配,找到 class 属性中包括'_con'字符串的 div 标签
    for p in soup.select("div[class * = '_con'] p") :
        text += p.text
    return text
#保存数据
def saveFile(text,path,filename) :
    if not os.path.exists(path) :
        os.makedirs(path)
    with open(path + filename,"w",encoding = "utf - 8") as f:
        f.write(text)
```

由于新闻分别来源于 finance 和 culture 两个不同的目录,使用的网页页面模板不同,要提取的新闻文本数据所在的 div 标签属性不同,为了简化处理,定义解析新闻文本内容方法,使用 BeautifulSoup 的 select 方法查找目标 div 标签时使用正则表达式,提取 class 属性中含有"_con"的字符串。

2. 调用方法获取和保存新闻内容

调用上面定义的 4 个方法获取和保存新闻内容的具体步骤如下:

(1) 调用 readJson 方法获取新闻列表的 JSON 数据。

(2) 进行新闻列表遍历。

- 提取新闻所在的 URL 地址。
- 提取时间 time 作为新闻文件所在的文件夹路径 path,以便将同一天的新闻归档存储在同一个文件目录中。
- 提取标题 title,并去掉非文件名字符,作为文件名 filename。
- 调用 getHtml 和 parseNews 方法获得新闻内容文本 txt。
- 调用 saveFile 方法保存新闻文件。

其具体的代码如下:

```
import re
JData = readJson("files/newslist.json")
for item in JData:
    url = item["url"]                                    #提取 URL 链接
    time = item["time"]
    title = item["标题"]
    title = re.sub('[\\/:*?"<>|]',"",title)              #去掉标题中的非文件名字符
    html = getHtml(url)
    page = parseNews(html)
    saveFile(page,"files/" + time + "/",title + ".txt")
```

有的新闻标题中含有类似'?'、':'等的文件名禁用字符,调用 re.sub 方法对这些字符进行正则替换后作为文件名。保存后的结果如图 3.29～图 3.31 所示。

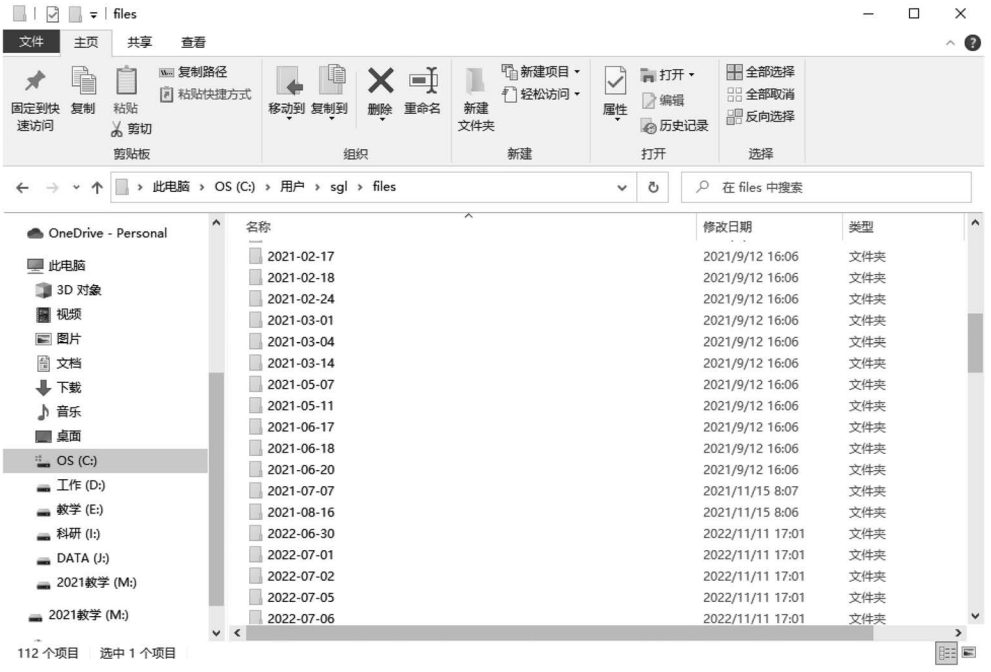

图 3.29 存储的以时间命名的文件目录

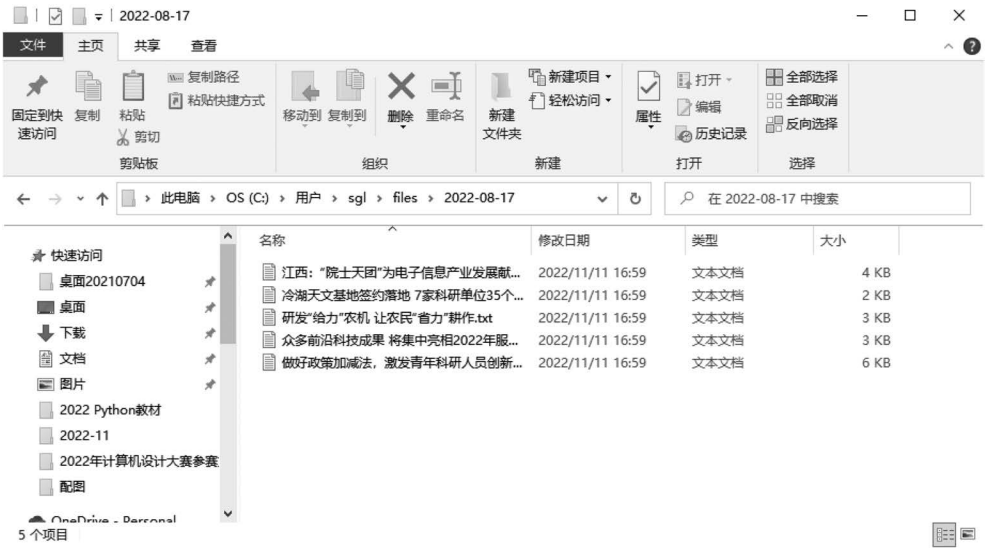

图 3.30 文件夹下保存的新闻文本文件

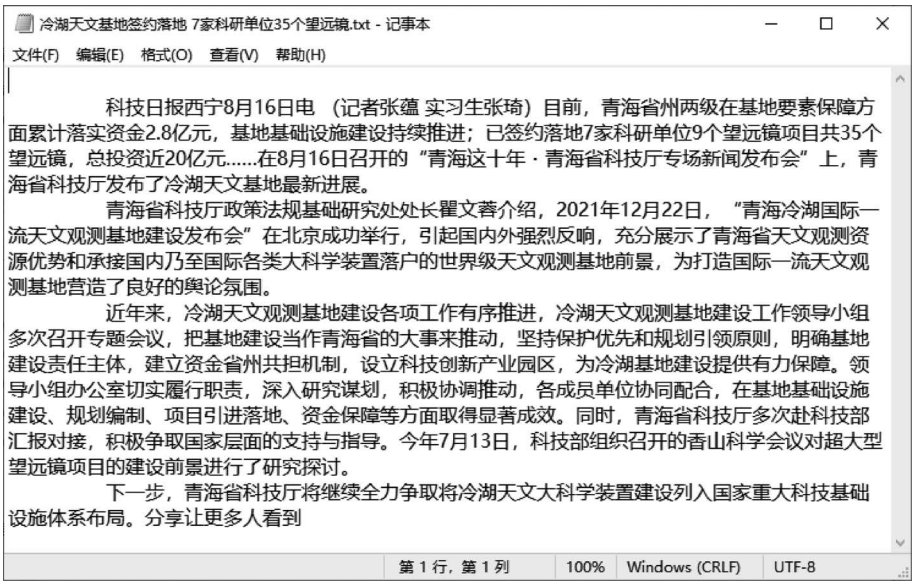

图 3.31 文件中的新闻内容

下面是全部新闻内容的获取和存储的实战案例代码。

【实战案例代码 3.6】 爬取与存储人民网的科技新闻内容。

```
import json
import requests
from bs4 import BeautifulSoup
import os
import re
#读取新闻列表文件
def readJson(filename) :
    with open(filename,"r",encoding = "utf - 8") as f:
        newStr = f.read()
        JData = json.loads(newStr)
    return JData
#发送请求,获取数据
def getHtml(url) :
    r = requests.get(url)
    r.encoding = r.apparent_encoding
    return r.text
#解析新闻文本数据
def parseNews(html) :
    soup = BeautifulSoup(html)
    text = ""
    #正则表达式匹配,找到 class 属性中包括'_con'字符串的 div 标签
    for p in soup.select("div[class * = '_con'] p") :
        text += p.text
    return text
#保存数据
def saveFile(text,path,filename) :
    if not os.path.exists(path) :
```

```
            os.makedirs(path)
        with open(path + filename,"w",encoding = "utf - 8") as f:
            f.write(text)
    # 获取新闻列表数据
    JData = readJson("files/newslist.json")
    for item in JData:
        url = item["url"]                                    # 提取 RUL 链接
        time = item["time"]
        title = item["标题"]
        title = re.sub('[\\/:*?"<>|]',"",title)     # 去掉标题中的非文件名字符
        html = getHtml(url)
        page = parseNews(html)
        saveFile(page,"files/" + time + "/",title + ".txt")
```

本章小结

本章介绍了3个Python爬虫实战项目,涉及结构化、半结构化和非结构化网站数据。每个实战项目均涉及目标网站分析,数据的爬取、解析和存储以及模块程序的编写等相关内容。本章首先介绍了结构化数据——中国A股上市公司相关数据的获取;然后介绍了如何存取、解析数据,主要介绍文件的存取方法,包括文本文件、CSV文件和JSON文件;接下来介绍了半结构化数据——豆瓣读书Top250数据的获取;为了更便捷地解析数据,引入了正则表达式,包括正则表达式基础、用法以及用其提取豆瓣排行榜网页数据的实战案例;最后介绍了非结构化数据——人民网科技类新闻的获取。

习题3

扫一扫

习题

扫一扫

自测题