第 9 章

用 Python 自动提取网站数据

9-1 因特网程序设计基础

现代社会大家都在使用网络，几乎所有的数据都可以在网上找到。在平时，我们要搜索某些数据（如天气信息、新闻，甚至是第 8 章中所介绍的地震观测数据等）时，都是以人工的方式通过浏览器去查看，把需要的信息记忆到脑海里或整理到文件中。把这些工作交由程序来做，不只是工作可以自动化，还可以不限时间和空间，帮助我们浏览更多的信息。

只要有合适的网站和可以处理通过这些网址得到的数据的方法，无论是半结构化的数据（HTML 格式）还是结构化的数据（JSON 格式或 XML 格式），都可以轻松通过程序进行处理与使用。

本章将从网址的处理开始，进一步探讨如何通过网址的改变获取更多的信息，然后探讨分析网页和提取网页数据的基本原则与方法。主要的网络程序处理对象为因特网的 HTTP 协议，其他的协议（如 FTP、Socket 程序设计）不在本章的讨论范围。

近年来，通过网络提取数据的方式大多数都是通过程序来完成的，除了本章介绍的基础方法之外，连 Pandas 这一类数据分析模块都有一些简易的指令可以一次性提取网页数据并直接放进结

构化的数据类型中，甚至还有像 Scrapy 这一类的网络爬虫框架，通过配置文件的编辑，就可以自动地大量帮我们下载网页数据并加以保存，这些会在后面的章节中陆续介绍。在本章中，我们从基础的部分开始学起。

9-1-1　因特网与 URL

在所有的数据都放在网上的时代，只要有正确的网址，就可以提取许多想要的数据，而这些数据有些以网页的方式显示（如气象统计数据、百度和谷歌的搜索结果、列车或者航班时刻表等），有些以 DOC、PDF、ODS 或 XLS 的方式存储，有些则以 JSON 的方式提供（如美国的地震观测数据）。无论是什么类型的数据，在下载之前它们都只是一个网址，正确地说，是一个 URL。

以前面介绍的 USGS 地震观测数据网站为例，除了可以从网站（网址为 http://earthquake.usgs.gov/earthquakes/，网站界面如图 9-1 所示）上看到全球地震相关信息之外，还提供了各种不同格式的数据以供程序提取（见图 9-2 左上角的各个链接）。

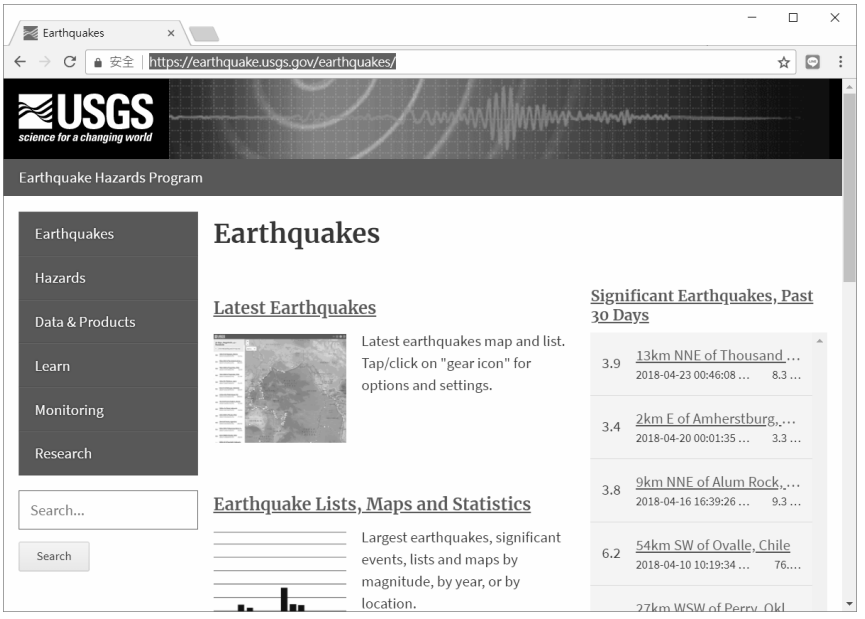


图 9-1　美国 USGS 全球地震观测数据

在图 9-2 箭头所指的地方选择一个想要下载的格式对应的链接，单击之后即可看到所有需要的数据，如图 9-3 所示。这些数据是提供给程序分析用的原始数据，我们在第 8 章中已有分析，且使用程序处理过。

对我们的程序而言，只要拿到上述网址（此网址基本上是不会任意变动的，在此例中为 http://earthquake.usgs.gov/earthquakes/），通过程序提取此网址的数据，等于是随时可以拿到每 5 分钟更新一次的全球大型地震观测数据，通过自动化的设置，你甚至可以比新闻媒体更早得知世界某处发生的大型地震信息。（例如，可以在自己的程序内设置只要震级超过 6.5，就马上寄电子邮件通知你或在你的网站上更新信息，如果你的程序每 5 分钟就提取一次数据，等于是最慢 5 分钟内就可以得知发生了大地震。）

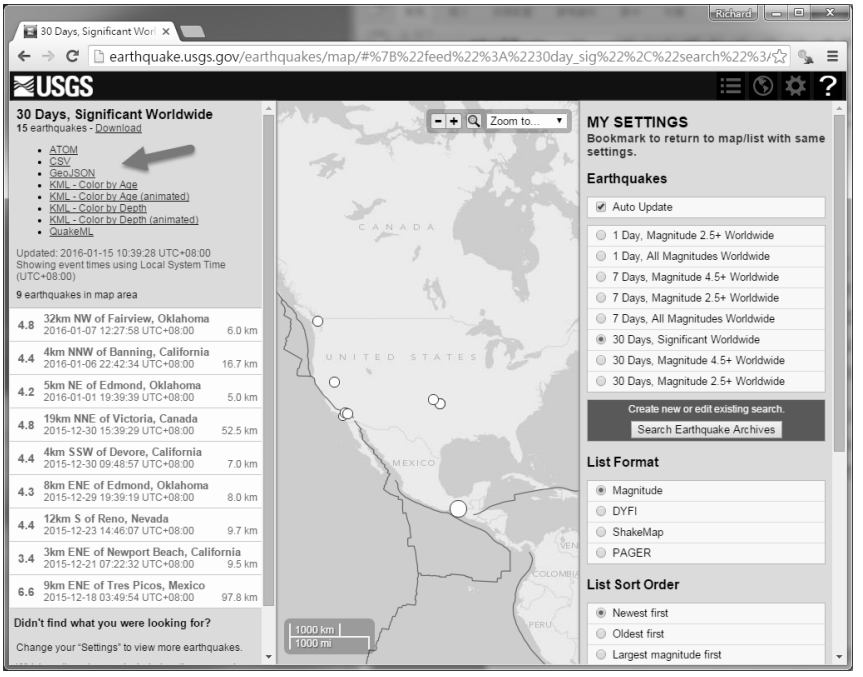


图 9-2　USGS 提供的 30 天重大地震数据下载

图 9-3　USGS 提供的 JSON 格式数据供程序下载之用

类似的情况在各大网站都有，以固定的网址更新一些实时变化的信息。例如，新浪网的股票频道，网址为 http://finance.sina.com.cn/realstock/company/sh000001/nc.shtml，网站页面如图 9-4 所示；中国气象局网站的当天实时天气情况，网址为 http://www.cma.gov.cn/2011qxfw/2011qtqyb/，网站页面如图 9-5 所示。

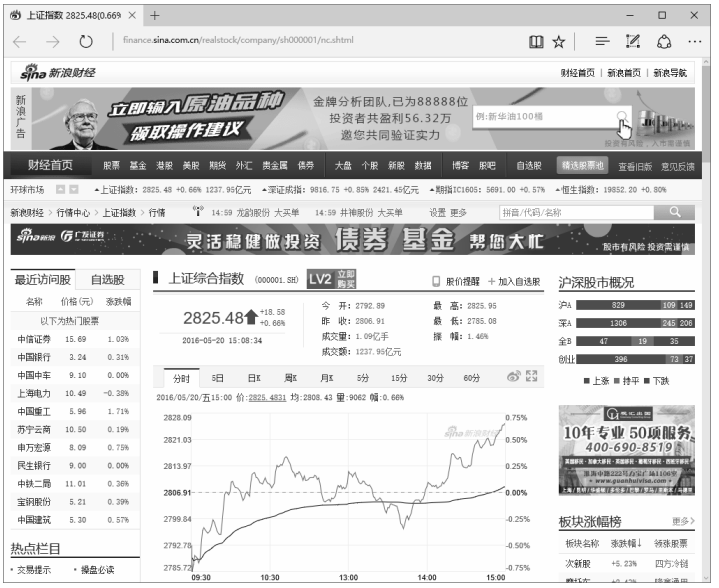


图 9-4　新浪网股票频道实时行情更新情况的网站页面

图 9-5　中国气象局当前实时天气情况的网站页面

由上可知，如果我们想要以程序提取上述数据，只要有网址就行了，只是不同的网页有不同的数据格式，也有不同的界面编排方式，要提取其中特定的内容，还需要使用一些特定的模块和方法，这也是接下来要说明的内容。

其实，大部分网站上的数据都有特定的渠道——政府机构的数据可以通过申请的方式获取，而私人机构则是以付费的方式获取，这样可以得到具有一定结构化的数据，在处理上会比较方便。但是，对于个人用途而言，从公开的网页上提取数据再加以分析是比较低成本且快速的方式，只是网页上的数据不太结构化，尤其是现代的商业网站充满了各种各样的技术和广告单元，在分析上有一定的复杂度，要花比较多的思考时间和程序设计时间，这点可以自行考虑。

9-1-2 解析网址

大部分网页中数据的数量较大，要能够有结构地找到所有我们想要的数据，了解网址组合是第一步。因为可能必须通过搜索或分页的方式才能够提取需要的所有数据。以新浪股市新闻为例，某一天的股市新闻网页如图 9-6 所示。

图 9-6 新浪股市新闻网页

当我们向下滚动屏幕时，可以看到右侧的“个股点评”分栏，如图 9-7 所示。

图 9-7 新浪股市的“个股点评”分栏

单击“选股的目的是选出未来能大涨的票来”标题的链接，结果如图 9-8 所示。

图 9-8　新浪股市个股点评内容

网址看起来是这样的：

```
http://blog.sina.com.cn/s/blog_4e345ca90102wm9v.html?tj=fina
```

看起来，http 是通信协议，而 blog.sina.com.cn 是域名，/s/blog_4e345ca90102wm9v.html 是网页所在的位置和网页文件名，?tj=fina 则是查询用的参数，也是 GET 的参数。通过 Python 的 urllib 模块的 urlparse 分析函数可以把这些参数内容区分开。程序片段如下：

```
    >>>from urllib.parse import urlparse
    >>>uc =
urlparse('http://blog.sina.com.cn/s/blog_4e345ca90102wm9v.html?tj=fina')

    >>>uc
    ParseResult(scheme='http', netloc=' blog.sina.com.cn ',
path='/s/blog_4e345ca90102wm9v.html
    , params='', query='?tj=fina ', fragment='')

    >>>uc.netloc
    ' blog.sina.com.cn'

    >>>uc.path
    '/s/blog_4e345ca90102wm9v.html'

    >>>uc.query
    '?tj=fina'
```

当然，我们在抓取网页数据的时候，除了一些固定会更新的内容外，网址大部分不会变化。

另外，像上述例子，需要读取的信息内容可能超过一页，就需要分析网址的特色，在抓取时再加以组合。在遇到比较复杂的网址时，解析之后就可以了解如何自定义这些网址的参数。我们以中华英才网站为例，招聘信息首页如图 9-9 所示，而选择“北京通信工程师”职位之后，其网页如图 9-10 所示。

图 9-9　中华英才网站招聘信息首页

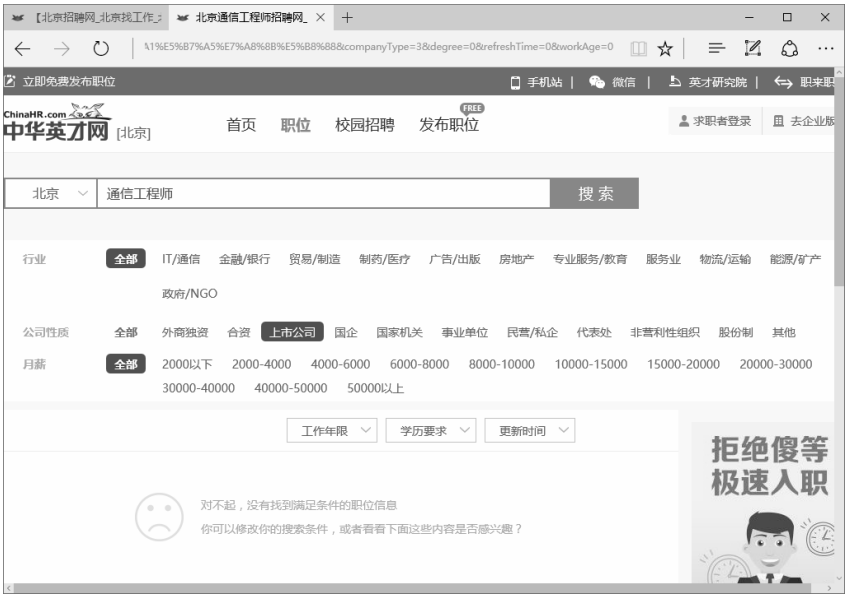


图 9-10　“北京通信工程师”职位信息

可以发现在“北京通信工程师”职位这个网页中，其网址复杂了许多，我们根据此网址编写一个程序分析其网址，如程序 9-1 所示。

程序 9-1

```
    # _*_ coding: utf-8 _*_
    # 程序 9-1  (Python 3 version)
    from urllib.parse import urlparse

    url = 'http://www.chinahr.com/sou/?city=398&keyword=
%E9%80%9A%E4%BF%A1%E5%B7%A5%E7%A8%8B%E5%B8%88&companyType=3&degree=0&refreshTi
me=0&workAge=0 '

    uc = urlparse(url)
    print("NetLoc:", uc.netloc)
    print("Path:", uc.path)

    q_cmds = uc.query.split('&')
    print("Query Commands:")
    for cmd in q_cmds:
        print(cmd)
```

程序 9-1 除了把网址分成网站的域名、网站地址以及查询命令之外，也可以把 query 查询命令以“&”分割开，本例中没有，请读者根据实际情况填写。

```
    In [3]: run 9-1.py
    NetLoc: www.chinahr.com
    Path: /sou/
    Query Commands:
    city=398
    keyword=%E9%80%9A%E4%BF%A1%E5%B7%A5%E7%A8%8B%E5%B8%88
    companyType=3
    degree=0
    refreshTime=0
    workAge=0
```

从上述结果可以清楚地发现，只要通过网址对这些 Query Commands 等号后面的参数进行修改（例如，把 companyType 后面的 3 改为 4，或者把 workAge 后面的 0 改为 2 等），就可以按照我们的查询要求去查询，如果找到匹配的项，就可以把结果呈现在页面上。读者可以自行试试（可以灵活使用字符串的 format()函数组合出不同的查询字符串）。

9-1-3　提取网页数据

有了 9-1-2 节的知识，大部分网站都可以按照要求在网页上呈现出我们需要的信息。接着，如何利用 Python 程序提取这些网页到程序中呢？通过模块 requests 就可以。

这个模块并不是默认的模块，所以在使用之前，可能需要在你的系统中先执行“pip install requests”或“pip3 install requests”才行。确定安装完毕之后，接下来在程序中使用 requests.get 指令读取我们想要处理的网页内容，操作过程如下。

程序 9-2

```
# -*- coding: utf-8 -*-
# 程序 9-2  (Python 3 version)

from pprint import pprint
import requests

url = 'http://www.sohu.com'

html = requests.get(url).text.splitlines()
for i in range(10):
    print(html[i])
```

程序 9-2 以搜狐新闻的网页为目标，使用 requests.get 提取此网页的内容，并以文本文件的形式存放在 html 变量中，同时在存放入 html 之前，先以 splitelines()把内容按换行符分割成一行一行字符串所组成的列表，所以 html 变量就成为一个列表类型的变量，可以通过 for 循环取出任何行数的内容。在此例中，只取出前 10 行进行打印，程序 9-2 的运行结果如下。

```
In [9]: run 9-2.py
<!DOCTYPE html>
<html>

<head>
<title>搜狐</title>
<meta name="Keywords" content="搜狐,门户网站,新媒体,网络媒体,新闻,财经,体育,娱乐,时尚,汽车,房产,科技,图片,论坛,微博,博客,视频,电影,电视剧"/>
<meta name="Description" content="搜狐网为用户提供 24 小时不间断的最新资讯，及搜索、邮件等网络服务。内容包括全球热点事件、突发新闻、时事评论、热播影视剧、体育赛事、行业动态、生活服务信息，以及论坛、博客、微博、我的搜狐等互动空间。" />
<meta name="shenma-site-verification" content="1237e4d02a3d8d73e96cbd97b699e9c3_1504254750">
<meta charset="utf-8"/>
<meta http-equiv="X-UA-Compatible" content="IE=Edge,chrome=1"/>
```

从上面的例子可以看出，虽然提取的是新闻网页的内容，但是前面的一些文字大多还是属于程序、排版、页面属性等相关的细节，这些内容主要是给浏览器和搜索引擎看的，对用户来说没有什么意义，还需要一些技巧才能够把需要的信息从这些程序代码中提取出来，这也是接下来要介绍的内容。

利用程序 9-2 可以修改网址变量 url 的内容，下载任何公开的网站信息。此外，在程序 9-2 中，为了方便展示起见，我们使用换行符号来分割下载的网页数据，但是换行符号对于网页内容是没有意义的，因为网页的组成主要是由 HTML 语言来描述的，这种语言的语句是由一系列标签（tag）所组成的。浏览器就是按照这些标签来决定如何显示网页的数据以及排版的方式。在大多数情况下，排版的样式也是由 CSS 层叠样式表语言，甚至是 JavaScript 语言等的语句来决定的。

9-1-4　提取网页内的电子邮件账号

要分析所得到的网页数据并从中提取所需要的数据，有多种可行的方式，比较复杂的方法是分析网页的组成结构，尤其是以 HTML 语言描述的各个标签和属性，我们将在 9-2 节中说明。在本小节，先把提取的网页信息全部当作一个字符串来看，然后使用正则表达式（Regular Expression）过滤字符串的内容，并取出我们所需格式的内容。

如果只是要找出某一个或某些单词、字符串是否出现在某个网页中，在 Python 中只要使用 in 就可以了。例如，设计一个程序来协助我们查找某人的姓名是否存在于某个网页上(最简单但无效率的查榜服务）。目标网页是某一个大学院系的榜单：http://www.xxx.edu.cn/exam/check_001_NO_0_2015.html（某大学的 2015 学年的计算机 Python 考试榜单）。我们使用程序 9-3 即可查询某个姓名是否在此网页中。

程序 9-3

```
# _*_ coding: utf-8 _*_
# 程序 9-3 (Python 3 version)

import requests

url = 'http://www.xxx.edu.cn/exam/check_001_NO_0_2015.html'
name = input("请输入要查询的姓名:")
html = requests.get(url).text
if name in html:
    print("恭喜名列金榜")
else:
    print("不好意思，榜单中找不到{}".format(name))
```

每次执行程序时，此程序就会去下载该网页，然后把网页转换成一个字符串并存放在 html 变量中，再以 in 运算符来检测输入的姓名（存放在 name 字符串变量中）是否在 html 内。以下是运行结果的范例：

```
In [11]: run 9-3.py
请输入要查询的姓名:林小明
不好意思，榜单中找不到林小明
In [12]: run 9-3.py
请输入要查询的姓名:李家宜
恭喜名列金榜
```

然而，大多数情况下我们并不是要查找某一个特定的文字，而是某一种特定类型的文字，例如电子邮件账号、链接符号或电话号码等。此类信息均有特定的格式，但是其文字内容可能为任何字母或数字符号，使用 in 是找不出来的，这种情况就需要使用正则表达式。

所谓正则表达式，简单地说，就是用一套严谨的语法来表达出我们想要的某种格式的字符串。例如，标准的电话号码格式“(010) 8765-1234”，如果要用这种格式把北京市区的电话号码

填入表中，用中文口语的方式叙述就是“用左小括号开头，接着是 3 个数字的区号，再用右小括号结束区号的部分，接着是电话号码的前 4 个数字，再接一个连字符（“-”），最后加上后 4 个数字”。这么长的文字描述是不是非常低效且很容易被误解？如果使用正则表达式，那么只要这样表示就好了：“\(\d\d\d?\)\d\d\d\d?-\d\d\d\d”。是不是精简多了？（还有更好的表示方法：^\(\d{2}\)\d{3,4}\-\d{4}$，用了更多的正则表达式的记号）。表 9-1 是正则表达式中几个常用的符号及其表示的含义。

表 9-1　正则表达式常用记号及其说明

正则表达式记号	例子与说明
[abc]	代表一个符合 a 或 b 或 c 的任一个字符
[a-z]	代表一个符合 a, b, c, ..., z 的任一个字符
.	代表一个除了\n（换行符号）之外的所有字符符号
*	代表前一项可以出现 0 次或无限多次
+	代表前一项可以出现 1 次或无限多次
?	代表前一项可以出现 0 次或 1 次
\	表示后面接着的字符以一般字符处理
{n}	n 是一个数字，用来指定前一项出现的次数（要一样才符合）
{n,}	n 是一个数字，用来指定前一项出现的次数，至少是 n，最多不限
{n,m}	n、m 均为数字，用来指定前一项出现的次数至少是 n、最多是 m
\d	一个数字字符，等于[0-9]
^	非运算或者反运算，例如[^a]代表不是 a 的所有字符
\D	一个非数字的字符，等于[^0-9]
\w	代表数字、字母或下画线
\W	非\w
\t	制表符号
\n	换行符号
\r	回车（return）符号
\s	所有空白符号（非显示符号）
\S	非\s

我们可以通过网站 http://pythex.org/分析正则表达式的实际作用，分析正确之后再放在自己的程序中使用。

所以，只要想要在网页上找出所有类似的内容，就可以使用正则表达式，然后到网页字符串中找出来即可。以下是电子邮件账号的正则表达式（参考网站 http://emailregex.com/的内容，但是实际应用在网页的时候要把前后的^和$去掉）：

```
[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+
```

程序 9-4 即以 Python 的正则表达式模块中的 findall()函数找出某一个网页中所有的电子邮件账号，并把它们都列出来（基于隐私权，请自行搜索网站上提供了电子邮件账号的测试网页进行测试）。

程序 9-4

```
# _*_ coding: utf-8 *_*
# 程序 9-4 (Python 3 version)

import requests, re

regex = r"([a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+)"
url = 'http://xxxx.xxx.xxx'

html = requests.get(url).text

emails = re.findall(regex,html)
for email in emails:
    print(email)
```

此程序的运行结果会列出所有在网页中找到的电子邮件账号。在程序 9-4 中，url 用来放置要下载的网页网址，而 regex 后放置我们设计的正则表达式。其中，在字符串符号“"”之前的“r”是让 Python 解释器知道在其后的字符串内容要保留原来的样子，不要做任何解释或翻译操作，这些解释或翻译操作交由 re.findall(regex,html)即可。

一个设计好的正则表达式可以找出非常多种类的字符串组合，只要把提取的文件当作一个字符串来看，re.findall()可以找出电子邮件账号、电话号码、外部链接，甚至是某些特定的网址类型。然而，如果是更复杂的形式，例如要找出某一个表格内的某些数据（如实时气温、股票实时报价等），从 HTML 结构下手反而会比较轻松，我们将在 9-2 节介绍此方法。

9-2　网页分析与应用

如同 9-1 节说明的，使用 Python 程序提取网页的内容非常简单，只要短短几行程序代码就可以实现。但是，提取的原始网页内容是给浏览器解析的 HTML 格式，如果我们要提取的是其中比较复杂的文字或数字数据（例如想要提取当前天气预报中的所有温度），如何在这些繁杂的内容中找出所需的数据呢？最好的方式就是解析 HTML 的网页结构，找出所有的标签，然后观察想要提取的数据，其网页原始文件所使用的标签是哪些，这些将是本节的重点。

9-2-1　HTML 网页格式简介

除非特殊情况，现今大部分网站的网页都是使用 HTML（Hyper Text Markup Language）编写的。在发明的时候，HTML 的主要目的在于协助浏览器了解网页文件中每一段内容的编排方式，也就是要显示的外观模样，主要的结构如下。

```
<html>
<head>
<meta 文档属性设置>
```

```
<title>
</title>
<script ...></script>
<link rel=stylesheet type=”text/css” ...>
</head>
<body>
<h1>标题</h1>
<pclass='选择器'id='识别符号' style='css 格式命令'>
内文段落
</p>
<table>
<tr><td>表格内容</td></tr>
</table>
<imgsrc=...>
<a href='...'>外部链接</a>
</body>
</html>
```

由小于号“<”和大于号“>”所括住的字符串叫作标签（tag）。大部分标签都是成对出现的，但是后面出现的标签则多使用了一个除号，例如<body></body>，少部分标签因为要呈现的信息可以通过自身的属性完成，所以只要一个就好，例如<img src='images/pic.png'>，即在当前的位置显示服务器上的 images 文件夹下的 pic.png 图像文件。常用的 HTML 标签及其用途的说明如表 9-2 所示。

表 9-2　常用的 HTML 标签及其用途的说明

标签示例	用途
<html></html>	标示此文件为 HTML 格式，放在文件的第一行和最后一行
<head></head>	标示文件的标头位置，用来放置网页设置用的数据
<title></title>	放置此网页文件的标题，通常会被显示在浏览器的标题栏
<body></body>	标示网页文件显示内容的地方，所有要被显示在浏览器网页页面的内容均被放置在此处
<script></script>	放置描述语言内容的地方，也可以用 src 属性指定外部文件的网址，此描述语言会被浏览器执行，以建立更多的效果或执行与用户互动的功能
	强调标签内文字显示的轻重程度，h1 最重，h6 最轻。通常在格式的设置上，h1 内的文字都会以最大字体和粗体来显示
	用来呈现文件内容分段显示
	排版用的格式标签，通常网页设计者会把同一个 div 标签内的文字以同样的格式进行设置和调整，可以视为网页内文的大段落或显示分块
	同 ，但应用在比较小的范围，大部分都是一些可以用描述语言替换文字内容或显示效果的少量文字
<table></table>	以表格形式显示的内容
<imgsrc='...'>	图像文件的显示设置
<a href='...'>	外部链接的设置
<iframe></iframe>	把另一个文件或网页以窗框的方式无缝地放在当前的网页中一起显示的技巧

简单的标签如<h1> <title>等，大部分情况下只有标签本身，并没有什么属性可以提取（但是

有许多情况是被加上了样式（style）属性，以改变该标签呈现的外观），最多就是完整的标签描述“<h1>内容</h1>”，或者提取其 content（内容）。

但是有些标签本身还有自有的属性需要设置，例如“<img src='images/01.jpg' title='第一张图' alt='北京风景' width='300'>”，此标签名称是 img，而 src、title、alt、width 等都是此标签的属性，可以另外处理。此外，现代越来越复杂的网页内容也让网页设计者替许多标签加上了各种各样的自定义属性名称，这也是现代浏览器允许的，而这些外加的属性往往就是网页分析和提取特定数据的关键。

如前所述，HTML 并不在意文件内每一段文字代表的意义（是摘要、内文、作者还是商品价格等），所呈现的只有对某些内容进行显示或编排格式上的设置。此外，因为读取和解析此文件的是浏览器，所以收到的 HTML 内容有可能是这样的：

```
    <html><head><meta 文档属性设置><title></title><script ...></script><link
rel=stylesheet type="text/css" ...></head><body><h1>标题</h1><pclass='选择器' id=
'识别符号' style='css 格式命令'>内文段落</p><table><tr><td>表格内容</td></tr></table>
<imgsrc=...><a href='...'>外部链接</a></body></html>
```

不只没有按照良好的阅读格式编排，甚至还会遗漏部分成对的标签（并非每一个网站都被良好地维护着），这也是我们把 HTML 文件分类为半结构化文件的原因之一。为了正确解析网页中的内容，取出想要的数据，通常有几种做法，其中之一就是先把一些不需要的信息去掉，只留下想要的内容。例如，描述语言的<script>标签和网页的<head>信息，就是第一批要被去掉的目标。

对于留下来的网页内容，要根据对于网页源文件的观察，找出所需要的信息前后放置的标签是什么。现代大部分网页都会以 搭配 id 或 class 来给主要的文本块分类，有了这些标签，就比较容易通过程序自动找出所需的文字信息或链接信息。

至于如何观察这些信息呢？通过浏览器查看原始文档是最基本的方法，无论是使用什么浏览器，直接在网页上右击，再选择网页的源代码，即可看到原始的 HTML 内容，有经验的用户可以从其中看出想要提取的数据所在的位置，以及它们是被什么标签所指定或括住的，从而找出要提取的数据的类型设置。

Chrome 浏览器的开发者工具提供了更加方便好用的界面。以“中国中央气象台网站”为例，进入网站之后，选择菜单中的“开发者工具”选项，如图 9-11 所示。

图 9-11　中国中央气象台网站的“开发者工具”选项

在选择“开发者工具”选项之后，网页的右侧就会出现经过 Chrome 浏览器解析后的 HTML 源文件内容，如图 9-12 所示。

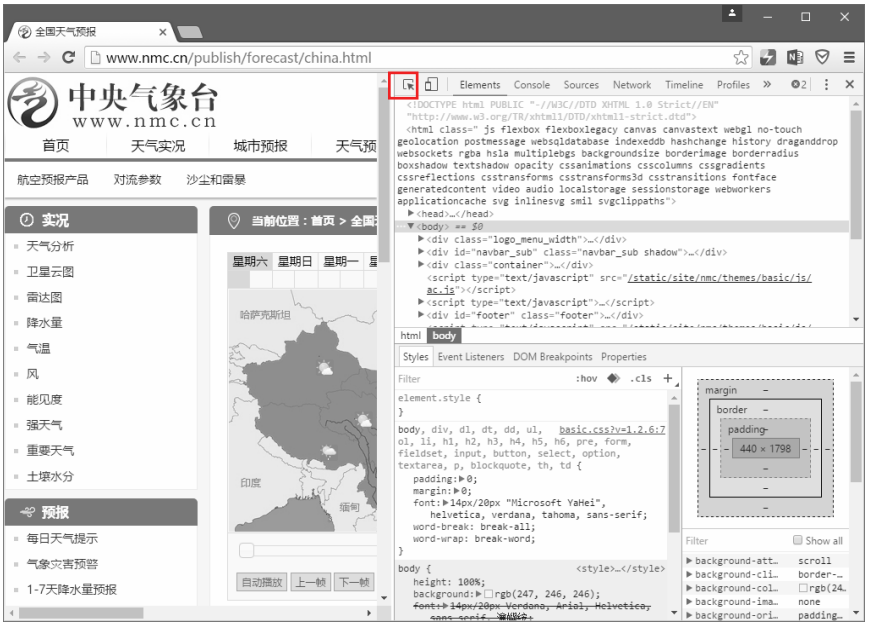


图 9-12　Chrome 开发者工具的界面

在此界面中，图 9-12 中方框的位置是“Inspect”菜单选项，还有 Element 功能，启用它们之后，将鼠标指针移到网页的任何一个地方都会显示当前网页元素所使用的标签，如图 9-13 所示。善用此工具可以让我们更容易分析网页的内容。

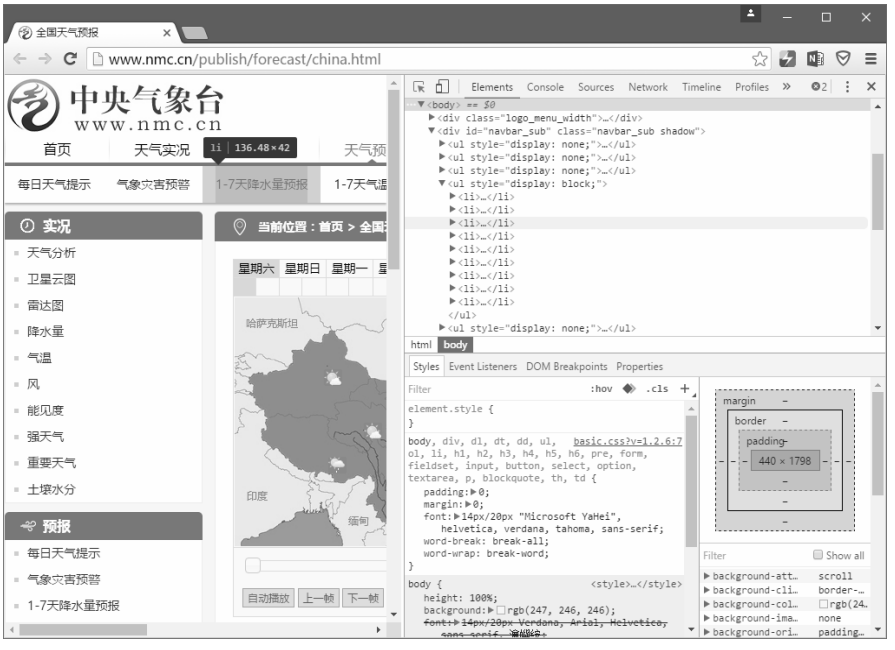


图 9-13　使用 Inspect 和 Element 功能查看网页元素所使用的标签

9-2-2　安装 Beautiful Soup

Beautiful Soup 是一套协助程序设计师解析网页结构的项目，起始于 2004 年，当前最新的版本是 4.6.0，官方网页的网址是 http://www.crummy.com/software/BeautifulSoup/，页面如图 9-14 所示。

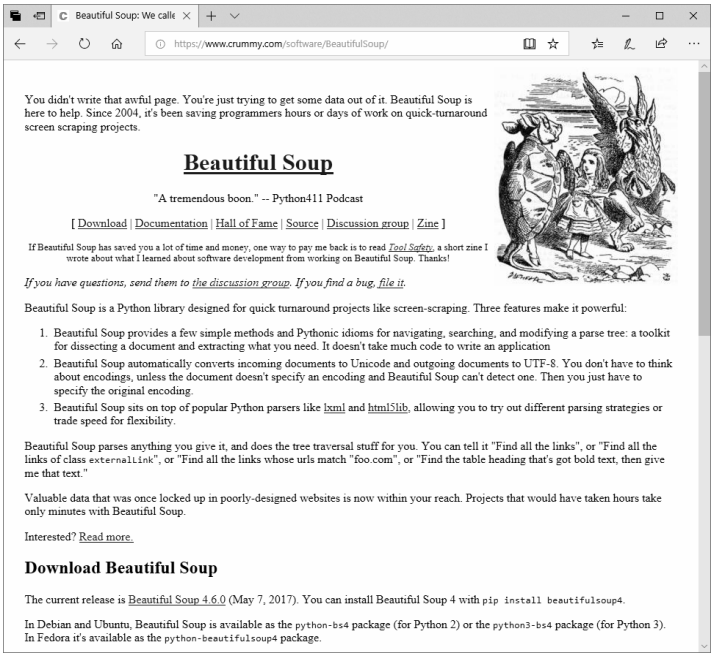


图 9-14　Beautiful Soup 的官方网站页面

如果在程序中使用 from bs4 import BeautifulSoup 发生找不到程序包的情况，那么在使用之前先以“pip install beautifulsoup4”安装此模块，代码如下（其他版本的差异请参考官方网站上的说明）：

```
    C:\ >pip install beautifulsoup4
    Collecting beautifulsoup4
      Downloading
https://files.pythonhosted.org/packages/9e/d4/10f46e5cfac773e22707237bfcd51bbf
feaf0a576b0a847ec7ab15bd7ace/beautifulsoup4-4.6.0-py3-none-any.whl (86kB)
        94% |████████████████████████████████████████████▏  | 81kB 132kB/s
eta 0:00:    100% |████████████████████████████████████████████████| 92kB
146kB/s
    Installing collected packages: beautifulsoup4
    Successfully installed beautifulsoup4-4.6.0
```

以下程序片段是 Beautiful Soup 第 4 版的基本使用方式：

```
    In [6]: from bs4 import BeautifulSoup

    In [7]: import requests
```

```
In [8]: url = "http://www.timeanddate.com/weather/"

In [9]: html = requests.get(url).text

In [10]: sp = BeautifulSoup(html, "html.parser")
```

先使用 from bs4 import BeautifulSoup 导入模块，搭配 requests 模块提取网页的数据，提取的网页数据转换成文本文件之后放在 html 字符串变量中，然后把 html 使用 Beautiful Soup 加以解析，解析之后的结果存在 sp 中，之后可以应用 Beautiful Soup 所提供的函数存取 sp 中解析后的数据，而这些函数主要是以标签为目标来操作的。例如，想要取出网页中所有的链接，可以使用以下指令：

```
In [11]: links = sp.find_all('a')
```

因为在 HTML 中链接的标签是“a”，所以找出所有的 a 标签就等于是找出所有的链接，其结果会以列表变量的方式返回，在上例中我们将其存放在 links 变量中。因此，可以通过 links 列表的操作来列出链接的内容，代码如下：

```
In [12]: links[10]
Out[12]: <a href="/custom/site.html">My Units</a>

In [13]: links[10].contents
Out[13]: ['My Units']

In [14]: links[10].get('href')
Out[14]: '/custom/site.html'
```

在此例中，我们列出第 11 个（索引值为 10）链接的原始字符串、标签内容（contents）以及 href 属性的值。

9-2-3 使用 Beautiful Soup 提取信息

如 9-2-2 小节所述，Beautiful Soup 协助我们整理网页数据，让程序设计人员可以通过指定标签找到网页中想要的数据。表 9-3 是几个 Beautiful Soup 中常用的分析网页格式的方法或属性。

表 9-3 Beautiful Soup 常用的方法和属性

Beautiful Soup 常用的方法和属性	使用说明（假设执行过 sp = BeautifulSoup(html)
title	返回此网页的标题 sp.title
text	除去所有 HTML 标签，把网页变为字符串返回 sp.text
find	返回第一个符合条件的内容 sp.find('img')

（续表）

Beautiful Soup 常用的方法和属性	使用说明（假设执行过 sp = BeautifulSoup(html)
find_all	返回所有符合条件的内容 sp.find_all('a')
select	返回以 CSS 选择器作为运算结果的所有内容，主要操作对象为 id 和 class sp.select('#Showtd')

如同 9-2-2 小节的范例，在表 9-3 中最常用的就是 find_all 函数，因为它可以设置一个搜索的条件以缩小欲锁定的数据范围。例如，可以使用以下函数找到文章中所有的链接（假设网页均已使用 Beautiful Soup 分析并放在 sp 变量中）：

```
all_links = sp.find_all('a')
```

而因为 HTML 的标准链接格式如下：

```
<a href='http://go.to.com'>link text</a>
```

所以可以通过“all_links[0]”提取该链接的全部内容（如上例为全部字符串），“all_link[0].get('href')”提取实际链接的网址（如上例为 http://go.to.com），“all_links.text”提取链接的文字内容（如上例为 link text）。程序 9-5 示范了如何提取某一网页的全部链接，并把这些链接的完整网址列出来（以是否为 http://开头作为判断的根据）。

程序 9-5

```
# _*_ coding: utf-8 _*_
# 程序 9-5 (Python 3 version)

from bs4 import BeautifulSoup
import requests
import sys

if len(sys.argv) < 2:
    print("用法: python 9-5.py <<target url>>")
    exit(1)

url = sys.argv[1]

html = requests.get(url).text
sp = BeautifulSoup(html, 'html.parser')
all_links = sp.find_all('a')

for link in all_links:
    href = link.get('href')
    if href != None and href.startswith('http://'):
        print(href)
```

在程序 9-5 中，我们先使用 find_all('a')提取所有的链接，然后把结果放入 all_links 变量中，再以一个 for 循环取出所有的 link，并以 link.get('href')提取此链接中的实际网址，由于有些链接可能

没有设置 href，因此要检查 href 是否为 None，另外也要检查 href 是否以'http://'起始，两个条件都符合才打印出来。以下是程序执行的过程以及部分结果：

```
    $ python 9-5.py http://www.baidu.com
    http://news.baidu.com
    http://www.hao123.com
    http://map.baidu.com
    http://v.baidu.com
    http://tieba.baidu.com
    http://www.baidu.com/bdorz/login.gif?login&tpl=mn&u=http%3A%2F%2Fwww.baidu
.com%2f%3fbdorz_come%3d1
    http://home.baidu.com
    http://ir.baidu.com
    http://www.baidu.com/duty/
    http://jianyi.baidu.com/
```

因为我们通过命令行参数的方式指定网址，所以读者可以利用此程序试试看你熟悉的网页是否能够提取所有的完整链接。我们也可以用同样的方法提取网页中所有的图像文件链接。程序 9-6 就是一个简单的范例。

程序 9-6

```
    # _*_ coding: utf-8 _*_
    # 程序 9-6 (Python 3 version)

    from bs4 import BeautifulSoup
    import requests
    import sys
    from urllib.parse import urlparse

    if len(sys.argv) < 2:
        print("用法: python 9-6.py <<target url>>")
        exit(1)

    url = sys.argv[1]
    domain = "{}://{}".format(urlparse(url).scheme, urlparse(url).hostname)
    html = requests.get(url).text
    sp = BeautifulSoup(html, 'html.parser')
    all_links = sp.find_all(['a','img'])

    for link in all_links:
        src = link.get('src')
        href = link.get('href')
        targets = [src, href]
        for t in targets:
            if t != None and ('.jpg' in t or '.png' in t):
```

```
            if t.startswith('http'):
                print(t)
            else:
                print(domain+t)
```

程序 9-6 多做了几件事，其中之一就是把搜索的目标扩大，除了<a>之外，也搜索<img>。此外，<a>的标准链接属性是 href，而<img>的标准链接内容是 src，因此我们把 href 和 src 都纳入检索的目标，只要这两个属性不是空的（None），就寻找其内容中有无.jpg 或.png，只要有，就准备显示。但是在显示出来之前，还要检查其是否为完整的网址（看看是否为 http 开头的字符串），如果不是，就为其补上该网站的网址（使用 urlparse 模块找出目标网页的主机网址）。经过此程序的处理，只要输入某一个网页的网址，程序 9-6 就会把此网页中所有放在 a 和 img 中的图像文件链接的网址都显示出来。基于网页隐私权的原因，这里就不列出运行结果了，请读者自行试用。

有了网址，如何把对应的图像文件内容存下来呢？请参考程序 9-7 的内容。

程序 9-7

```
# _*_ coding: utf-8 _*_
# 程序 9-7 (Python 3 version)

from bs4 import BeautifulSoup
import requests
import sys, os
from urllib.parse import urlparse
from urllib.request import urlopen

if len(sys.argv) < 2:
    print("用法: python 9-7.py <<target url>>")
    exit(1)

url = sys.argv[1]
domain = "{}://{}".format(urlparse(url).scheme, urlparse(url).hostname)
html = requests.get(url).text
sp = BeautifulSoup(html, 'html.parser')
all_links = sp.find_all(['a','img'])

for link in all_links:
    src = link.get('src')
    href = link.get('href')
    targets = [src, href]
    for t in targets:
        if t != None and ('.jpg' in t or '.png' in t):
            if t.startswith('http'):
                full_path = t
            else:
                full_path = domain+t
```

```
            print(full_path)
            image_dir = url.split('/')[-1]
            if not os.path.exists(image_dir):
                os.mkdir(image_dir)
            filename = full_path.split('/')[-1]
            ext = filename.split('.')[-1]
            filename = filename.split('.')[-2]
            if 'jpg' in ext:
                filename = filename + '.jpg'
            else:
                filename = filename + '.png'
            image = urlopen(full_path)
            fp = open(os.path.join(image_dir,filename),'wb')
            fp.write(image.read())
            fp.close()
```

程序 9-7 使用 urlopen 打开远端的文件，并通过 fp = open(...)的方式打开本地的文件，最后以 fp.write(image.read())的方式一次性从远端的文件中读取所有的数据之后，直接写入本地计算机成为图像文件。而真正存盘之前，主要的工作是确定要存盘的文件夹名称和文件名。在本范例中会检查要存取的文件夹是否存在，如果不存在，就通过 os.mkdir()创建一个新的。但是在存盘的部分，程序 9-7 并没有检查是否有一样的文件名，因此如果有一样名字的图像文件，后来的文件就会覆盖之前的文件。要避免这种情况发生，这部分设计留在习题中供读者练习之用。

基于网站隐私权的考虑，请读者自行搜索有图像文件的网页来试用本程序。但在使用时，请自行留意相关的法律责任问题，切勿因此造成被测试的目标网站的负担，因此在测试时建议先使用自己的网站测试。

9-2-4 进一步分析网页的内容

在 9-2-3 小节中，我们一视同仁地把所有的<img>和<a>都找出来了，但是在大部分情况下，我们要找的是一些特定的数据信息。例如，中国某地区当地石油公司列在网站上的历年油价信息，如图 9-15 所示。

观察网页，很显然油价是以表格的方式来呈现的，查看其网页源代码，如图 9-16 所示。

图 9-15 历年油价调整表

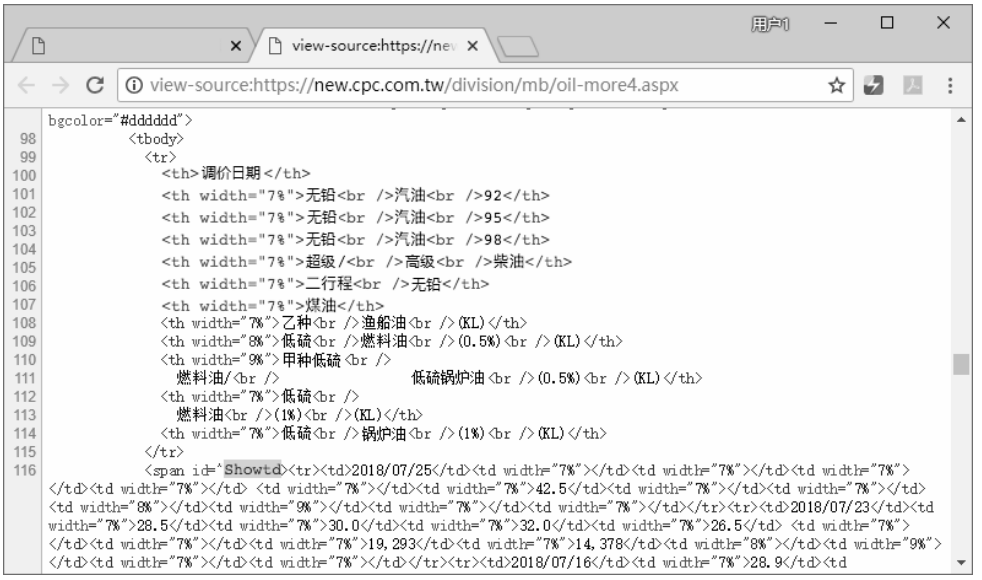


图 9-16　历年油价调整表网页的源代码

果然所有的油价数据都被放在 HTML 的<table>标签中，一大堆<tr><td>等标签都没有特定的属性，要如何锁定呢？所幸在图 9-16 中有一处灰底的标签，其 id 被设置为“Showtd”，这是一个很好的线索。在程序中，可以先通过 CSS Selector 的方式选定属性为 Showtd 的，再根据里面的<tr>和<td>进行分析。

在 HTML 的表格设计中，以<table></table>为最上层，接下来每一行用<tr></tr>括起，然后在里面用<td></td>设置表格中的单元格。观察图 9-15 的表格可以发现，在油价数据中，各行是调价日期，而各列则是油品的种类，只要数值有变化，就会有数字，如果没有改变，就是空的单元格，但是第 2 列一定是 92 无铅汽油，第 3 列一定是 95 无铅汽油，以此类推。

因此，我们可以在提取之后，再按<tr>来提取所有的行数，把每一行以列来分割，取出我们想要的信息。程序 9-8 示范了整个分析的过程。

程序 9-8

```
# _*_ coding: utf-8 _*_
# 程序 9-8 (Python 3 version)

from bs4 import BeautifulSoup
import requests

url = 'https://news.cpc.com.tw/division/mb/oil-more4.aspx'

html = requests.get(url).text
sp = BeautifulSoup(html, 'html.parser')
data = sp.find_all('span', {'id':'Showtd'})
rows = data[0].find_all('tr')

prices = list()
for row in rows:
```

```
        cols = row.find_all('td')
        if len(cols[1].text) > 0:
            item = [cols[0].text, cols[1].text, cols[2].text, cols[3].text]
                    prices.append(item)
    for p in prices:
        print(p)
```

其实看起来很简单，先通过 find_all('span', {'id':'Showtd'})找出记录油价所有单元格的内容，再将其放在 data 中，因为找到的内容是先以行再以列来显示的，所以使用 rows = data[0].find_all('tr')找出行，再使用 cols = row.find_all('td')找出每一列。

最后要使用时，先以字符串的长度判断该单元格内是否有数据，如果第 1 列 col[1].text 是有数据的，就再把它们存进 prices 列表变量中。以下是程序的运行结果（因为篇幅的关系，仅显示部分结果）。

```
In [21]: run 9-8.py
['2018/07/23', '28.5', '30.0', '32.0']
['2018/07/16', '28.9', '30.4', '32.4']
['2018/07/09', '29.1', '30.6', '32.6']
['2018/07/02', '28.9', '30.4', '32.4']
['2018/06/25', '28.5', '30.0', '32.0']
['2018/06/18', '28.7', '30.2', '32.2']
['2018/06/11', '28.6', '30.1', '32.1']
['2018/06/04', '28.8', '30.3', '32.3']
['2018/05/28', '29.2', '30.7', '32.7']
['2018/05/21', '29.0', '30.5', '32.5']
['2018/05/14', '28.7', '30.2', '32.2']
['2018/05/07', '28.1', '29.6', '31.6']
['2018/04/30', '28.2', '29.7', '31.7']
['2018/04/23', '27.8', '29.3', '31.3']
['2018/04/16', '27.3', '28.8', '30.8']
['2018/04/09', '26.7', '28.2', '30.2']
['2018/04/02', '26.8', '28.3', '30.3']
['2018/03/26', '26.6', '28.1', '30.1']
['2018/03/19', '26.0', '27.5', '29.5']
```

不过，因为这不是经常会变动的数据，所以在下载一次之后，理论上要存放在我们自己的数据库中，以避免过度使用别人的网页。这是下一节的教学内容。

9-3 网络应用程序

我们在前几节介绍的程序都属于单打独斗式的，也就是每次要提取数据的时候，都要从别人的网页上提取，其实这是一种相当没有效率的做法。除非要提取的网页信息是实时更新的，每一

次都不一样，这样到网页上提取数据才有意义，否则，像油价的信息已知定期才会更新一次，只要定期去取一次就好了，那么拿到的数据要如何处理呢？有以下两种方法。

其一，存放在数据库中，这是最标准的做法，第 10 章再加以说明；其二，以文件的形式存储，而且通过 HTML 网页的方式进行检索。下面将分成几个小节来说明。

9-3-1　将数据存储为文件

就如同程序 9-6 所做的一样，如果是要下载图像文件，第一个步骤就是把这些图像文件存储在专门的文件夹中，日后要查看的时候，只要以本地计算机操作系统中的图像浏览器浏览即可。同样的方式也适用于.pdf、.txt 以及所有可以浏览的文件格式。

但是，如果像程序 9-8 那样提取的是整理过的文字数据，要如何处理呢？除了存放到数据库的选择之外，我们也可以写成 HTML 格式的网页文件，此类型的文件是一般的文本文件，除了便于 Python 写入之外，写入之后的文件以.html 作为扩展文件名之后，即可用浏览器打开浏览，非常方便。此外，也可以存储到 CSV 格式（以逗号分隔的标准文本文件形式）的文件中，这种格式的文件在各种电子表格和数据库系统中都非常容易导入。

我们把程序 9-8 改为存储成网页形式，如程序 9-9 所示。

程序 9-9

```
# _*_ coding: utf-8 _*_
# 程序 9-9 (Python 3 version)

from bs4 import BeautifulSoup
import requests

pre_html = '''
<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>油价历史数据</title>
</head>
<body>
<h2>油价历史数据（取自网站）</h2>
<table width=600 border=1>
<tr><td>日期</td><td>92 无铅</td><td>95 无铅</td><td>98 无铅</td></tr>
'''

post_html = '''
</table>
</body>
</html>
'''

url = 'https://new.cpc.com.tw/division/mb/oil-more4.aspx'
```

```
html = requests.get(url).text
sp = BeautifulSoup(html, 'html.parser')
data = sp.find_all('span', {'id':'Showtd'})
rows = data[0].find_all('tr')

prices = list()
for row in rows:
    cols = row.find_all('td')
    if len(cols[1].text) > 0:
        item = [cols[0].text, cols[1].text, \
                cols[2].text, cols[3].text]
        prices.append(item)

html_body = ''
for p in prices:
    html_body += "<tr><td>{}</td><td>{}</td><td>{}</td><td>{}</td></tr>".\
         format(p[0],p[1],p[2],p[3])
html_file = pre_html + html_body + post_html

fp = open('oilprice.html','w', encoding='utf-8')
fp.write(html_file)
fp.close()
```

在程序中，我们导入了长字符串的设置方法，就是以 3 个引号（单引号或双引号都可以）开头，直到另一个成对的引号结束，中间的所有字符内容都会被视为字符串的一部分。通过长字符串的设置，我们设置了文件所需的前置标签 pre_html 和后置标签 post_html。另外，把从网页中搜集到的信息放在 html_body 中。最后把这 3 个变量组合成 html_file 字符串，再以 open('oilprice.html','w')的方式存成文本文件 oilprice.html，后面的 encoding 参数是为了让写入的文件变成 uft-8 的编码，以避免在 Windows 系统下的乱码问题。

此程序只要执行一遍，就会在当前的文件夹中产生上述的 html 文件，我们就可以随时通过浏览器打开此文件，看到我们想要的结果。程序 9-9 的运行结果以浏览器打开之后如图 9-17 所示。

日期	92无铅	95无铅	98无铅
2018/07/23	28.5	30.0	32.0
2018/07/16	28.9	30.4	32.4
2018/07/09	29.1	30.6	32.6
2018/07/02	28.9	30.4	32.4
2018/06/25	28.5	30.0	32.0
2018/06/18	28.7	30.2	32.2
2018/06/11	28.6	30.1	32.1
2018/06/04	28.8	30.3	32.3
2018/05/28	29.2	30.7	32.7
2018/05/21	29.0	30.5	32.5
2018/05/14	28.7	30.2	32.2
2018/05/07	28.1	29.6	31.6

图 9-17　程序 9-9 的运行结果

9-3-2　以网页的形式整理数据

除了直接查看数据之外，我们也可以通过网页的方式创建一个索引用的 html 文件，方便自行整理和查找。简单地说，就是通过 HTML 网页格式中的表格功能，再搭配上链接的功能，制成一个 index.html 网站，在该目录之下只要点开 index.html 文件，就可以使用这个网页上面的链接，找到所有存储的文件信息。如果把这个文件放在虚拟主机的目录中，就成了可以被浏览的网页数据。

所有的 HTML 标签和格式都可以在程序中加入，之后可以全部写入 index.html 文件中，能做的变化非常多。以程序 9-7 为例，它是通过网页的搜索把所有目标网页上的图像文件都存放到某一个文件夹中，而在 HTML 中有一个叫作 Bootstrap 的框架（framework），可以使用简单的语句就做到图像幻灯片跑马灯的效果（请参考网页 http://getbootstrap.com/javascript/#carousel）。程序 9-10 示范了如何把这些效果加到我们的 index.html 中。

程序 9-10

```
# _*_ coding: utf-8 _*_
# 程序 9-10 (Python 3 version)

from bs4 import BeautifulSoup
import requests
import sys, os
from urllib.parse import urlparse
from urllib.request import urlopen

post_html = u'''
</body>
</html>
'''

if len(sys.argv) < 2:
    print("用法: python 9-10.py <<target url>>")
    exit(1)

url = sys.argv[1]
domain = "{}://{}".format(urlparse(url).scheme, urlparse(url).hostname)
html = requests.get(url).text
sp = BeautifulSoup(html, 'html.parser')

pre_html = """
<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>网页搜索来的数据</title>
```

```
      <meta name="viewport" content="width=device-width, initial-scale=1">
      <link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/bootstrap/
3.3.6/css/bootstrap.min.css">
      <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/
jquery.min.js"></script>
      <script src="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/
bootstrap.min.js"></script>
      <style>
      .carousel-inner > .item >img,
      .carousel-inner > .item > a >img {
        border: 5px solid white;
        width: 50%;
        box-shadow: 10px 10px 5px #888888;
        margin: auto;
      }
      </style>

    </head>
    <body>
    <center><h3>以下是从网页搜索来的图像跑马灯</h3></center>
    """

    all_links = sp.find_all(['a','img'])

    carousel_part1 = ""
    carousel_part2 = ""
    picno = 0

    for link in all_links:
    src = link.get('src')
    href = link.get('href')
    targets = [src, href]
    for t in targets:
       if t != None and ('.jpg' in t or '.png' in t):
          if t.startswith('http'): full_path = t
          else:                    full_path = domain+t
          print(full_path)
          image_dir = url.split('/')[-1]
          if not os.path.exists(image_dir): os.mkdir(image_dir)
          filename = full_path.split('/')[-1]
          ext = filename.split('.')[-1]
          filename = filename.split('.')[-2]
          if 'jpg' in ext: filename = filename + '.jpg'
          else:            filename = filename + '.png'
          image = urlopen(full_path)
```

```
            fp = open(os.path.join(image_dir,filename),'wb')
            fp.write(image.read())
            fp.close()

            if picno==0:
                carousel_part1 += "<li data-target='#myC' data-slide-to='{}'
class='active'></li>".format(picno)
                carousel_part2 += """
                <div class='item active'>
                <img src='{}' alt='{}'>
                </div>""".format(filename, filename)

            else:
                carousel_part1 += "<li data-target='#myC' data-slide-to='{}'>
</li>".format(picno)
                carousel_part2 += """
                <div class='item'>
                <imgsrc='{}' alt='{}'>
                </div>""".format(filename, filename)
            picno += 1

            html_body = """
            <div id='myC' class='carousel slide' data-ride='carousel'>
                <ol class='carousel-indicators'>
                    {}
                    </ol>
                    <div class='carousel-inner' role='listbox'>
                    {}
                    </div>
                    <a class="left carousel-control" href="#myC" role="button"
data-slide="prev">
                        <span class="glyphiconglyphicon-chevron-left"
aria-hidden="true"></span>
                        <span class="sr-only">前一张</span>
                    </a>
                    <a class="right carousel-control" href="#myC" role="button"
data-slide="next">
                        <span class="glyphiconglyphicon-chevron-right"
aria-hidden="true"></span>
                        <span class="sr-only">后一张</span>
                    </a>
                </div>
                """.format(carousel_part1, carousel_part2)

    fp = open(os.path.join(image_dir,'index.html'), 'w')
```

```
    fp.write(pre_html+html_body+post_html)
    fp.close()
```

程序主要架构的部分和程序 9-9 是差不多的，但是使用 bootstrap 时需要在网页前加上一些 bootstrap 框架所需要的链接和设置，所以 pre_html 的内容多了许多。此外，为了配合幻灯片跑马灯 Carousel 的语句，我们多使用了 carousel_part1 和 carousel_part2 两个变量把搜索到的图像文件的链接加入，最后把 pre_html、carousel_part1、carousel_part2 以及 post_html 全部加在一起写入 index.html。换句话说，程序 9-10 不仅会帮我们把图像文件全部下载到计算机中，还会在同一个文件夹中创建一个 index.html，把这些图像文件用幻灯片跑马灯的方式来展示。出于知识产权的考虑，请读者自行执行程序，观看运行的效果（用浏览器打开 index.html 文件即可）。对于太小的图像文件，可以在下载图像的时候先检查一下图像文件的大小，太小的话就可以忽略它。程序使用方法如下：

```
    c:\>python 9-10.py 你要下载的网页
```

9-3-3 在本地建立网页应用

9-3-2 小节为特定的网站创建了自己的文件夹和 index.html。有没有觉得每次都要打开 index.html 很麻烦？没问题，我们可以在自己的计算机中创建一个网页服务器，日后要查找这些数据，只要打开自己的网页（localhost://localweb）就可以浏览了。

在 Windows 操作系统下，创建网页服务器可以选用 WAMP（Windows + Apache + MySQL + PHP），而在 Mac OS 操作系统下，则使用 MAMP（Mac + Apache + MySQL + PHP）是最方便的选择。WAMP 的网址为 http://www.wampserver.com/en/，网页如图 9-18 所示。MAMP 的网址为 https://www.mamp.info/en/，网页如图 9-19 所示。无论是哪一个操作系统，这些服务器（网页服务器、MySQL 数据库服务器以及 PHP 执行模块）都已经被打包成应用程序，只要下载适当的安装文件，然后执行安装程序完成安装即可。

图 9-18 Windows 用的 WAMP 服务器程序包

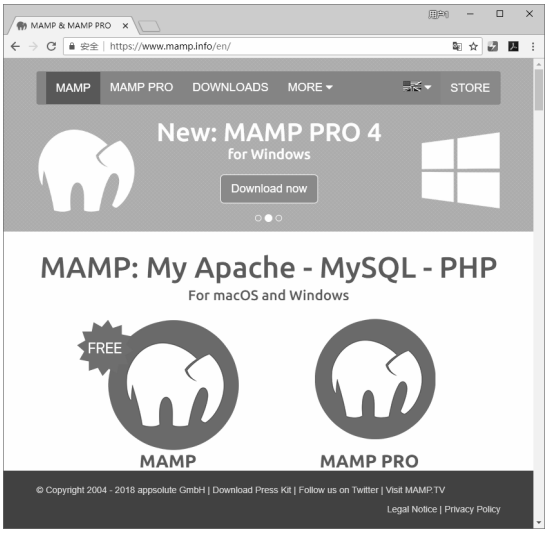


图 9-19　Mac OS 用的 MAMP 服务器程序包

以 WAMPSERVER 为例，在安装完成并执行该程序之后，它会在后端执行一个管理程序，并同时启用 Apache、MySQL 服务器，在默认的情况下，它们会自动侦听本地（localhost）的网页连接，此外在界面右下角的工具栏中也会有一个小图标，用于开启管理界面来设置相关参数。此外，我们只要启动浏览器并连接到 localhost，就可以看到如图 9-20 所示的屏幕显示界面。

图 9-20　WAMPSERVER 的默认网页及管理界面

如图 9-20 所示，在管理界面中找到 www 目录的文件夹位置，只要把程序 9-10 所写入的 index.html 以及相关文件数据都放在此文件夹中，就可以在本地像浏览网页一样浏览下载的脱机内容了，如图 9-21 所示。

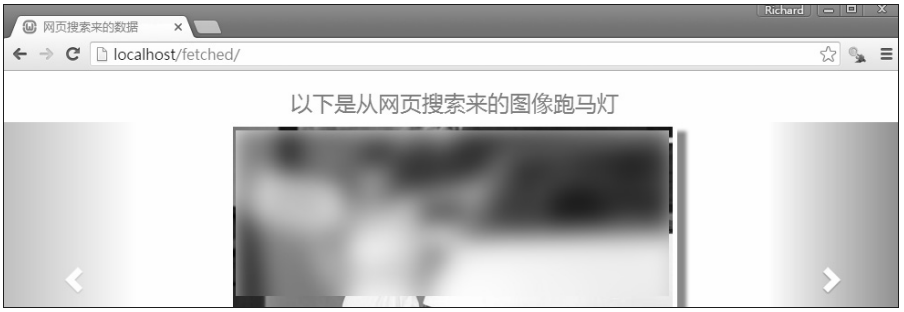


图 9-21　在自己的计算机中浏览下载成果

9-4　习　　题

1. 参考程序 9-5，将其改为可以显示所有 https://的链接。

2. 如果程序 9-7 在存盘的同时有同名的文件，就会直接覆盖掉，请修正这种情况，让所有的文件都能被保留下来。

3. 请参考程序 9-9 的内容，把目标网址改为气象局的当前气温网页，并存储提取后的数据结果。

4. 请利用在第 8 章中学习的 SQLite 知识把下载后的油价信息和气温数据存储到数据库中。

5. 请在你的计算机中安装 WAMP 或 MAMP，并把程序 9-10 的运行结果自动写到本机网页服务器的根目录文件夹中。

第 10 章

Python 网页数据提取实践

10-1　把网页数据存储到数据库中

在第 9 章学习了如何把网页上的数据下载到本地计算机中，并使用文件的方式存储这些数据以供后续使用。然而，有些数据可以通过不同的形式分析和运用，如果能够以数据库的方式存储，在使用上就会更具弹性。

例如，程序 9-8 下载了历年的油价信息，然后以 HTML 的表格格式存储，可是如果我们想要再找出历年最高油价或者平均油价，还是要查询某一时间区间内的平均油价，像以 HTML 格式存储的内容，就不方便进行分析、计算以及查找的操作。若以数据库的数据表方式把油价信息存储下来，需要查询的时候再重新设置查询功能以及查询方式，就可以轻松实现数据重新筛选、计算和统计的功能了。此外，同样的数据只要定期提取一次就好了，不需要每次使用的时候都到网站上去现“抓”，这样应用程序执行的速度也最快，还不会浪费网页主机的流量。

在本章中，我们先从加入数据库功能的数据提取程序应用模式谈起，再说明如何把提取到的数据存储在 SQLite 本地数据库中，以及如何进一步存储到网络数据库中，让它发挥最大的优势。

此外，让计算机可以持续自动地为我们更新数据也是自动化非常重要的一环，这些技巧也会在本章中加以说明。

不过在设计你的程序之前，在此先声明一点，随意从别人的网站提取数据来使用有可能会违反相关的法规，建议读者在使用之前先行了解该网站的规范。最重要的是，千万不要使用别人的网站测试你的程序，以免因为你的程序设计错误而引发不必要的问题。

10-1-1 网页数据的运用模式

到当前为止，我们的程序都是需要数据的时候就上网去提取，但是有些数据其实不是时时更新的，在大部分情况下只要下载一次就够了，不需要每次使用的时候都浪费网络资源把同样的事情再做一遍，不仅执行的时间过长，而且会造成网页服务器不必要的负担。

因此，在下载网页之前，除了要判断此网页在上一次造访之后是否更新过之外，还要有一个地方存储数据提取之后的状态信息。这时，通过数据库存储这些信息是最方便的。

也就是说，要获取某一个网页上的数据，其步骤应该是这样的：

（1）从本地数据库中获取目标网页上次提取数据的时间。
（2）获取目标网页上次的更新时间，如果两者时间一样，直接前往第 4 个步骤。
（3）获取数据并加以分析，然后把结果存储在数据库中。
（4）从数据库中取出数据并输出。

其中，把第 3 步（导入数据）和第 4 步（导出数据）操作分开，各有各的程序，而其中的中介存储库就是我们的数据库，此模式如图 10-1 所示。

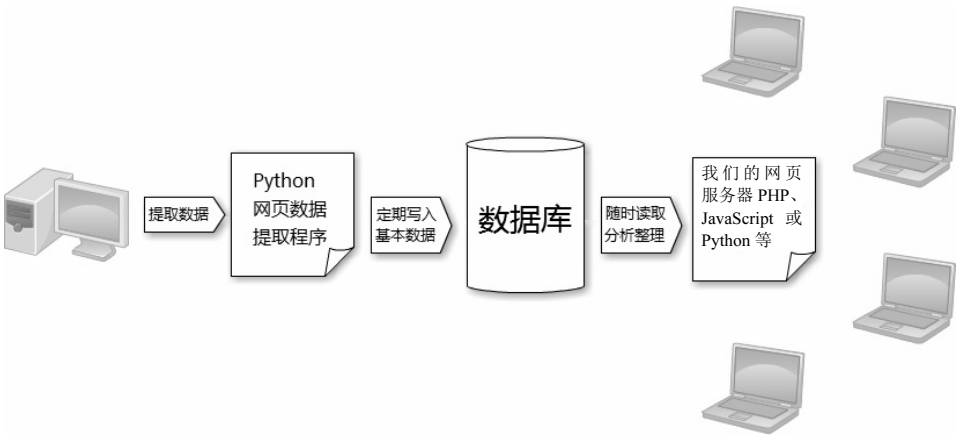


图 10-1 网页数据的应用模式

如图 10-1 所示，我们可以设置网页数据提取程序定期去特定的网页中搜索需要的数据，经过初步的筛选和分析之后，把基本的数据存储在数据库中，然后在这个数据库所在的位置建立一个可以读取此数据库的服务器网站，通过 PHP、JavaScript 或 Python，在网友浏览此网页的时候，按照浏览者设置的数据和需求，分析整理数据之后成为网页显示在浏览器中。

例如，发票兑奖号码、历年油价信息、地震测报数据、气温数据或银行的现行汇率、某些股票的相关信息等，都可以成为你网页中的一部分，提供给自己或网友查询。不过，如果你打算在

网站公开这些数据，千万要留意相关的知识产权问题。此外，有很多信息（如股价或新闻）其实是有正规的数据获取渠道的，只要付费或申请就可以了，而且提供的数据也更简单、更好整理。如果用于商业目的，还是以正规的渠道获取为宜。

10-1-2　把数据存储到 SQLite

为了实现图 10-1 的模式，我们先以在第 8 章介绍过的 SQLite 为数据库，说明如何把提取到的数据存储在数据库中。

要存储数据，需要先分析要存储的数据内容以及类型，以便创建正确的数据表。在本小节的例子中，我们打算存储历年的油价信息。从图 9-15 分析，要存储的内容包括日期、92 无铅汽油、95 无铅汽油、98 无铅汽油 4 项。其中，日期可以文字格式来存储，而汽油的价格则以数字来存储。因此，通过 Firefox 浏览器的附加组件 SQLite Manager 新创建一个数据库 gasoline（会在当前的文件夹中新建一个叫作 gasoline.sqlite 的文件），并新建一个数据表 prices，如图 10-2 所示。

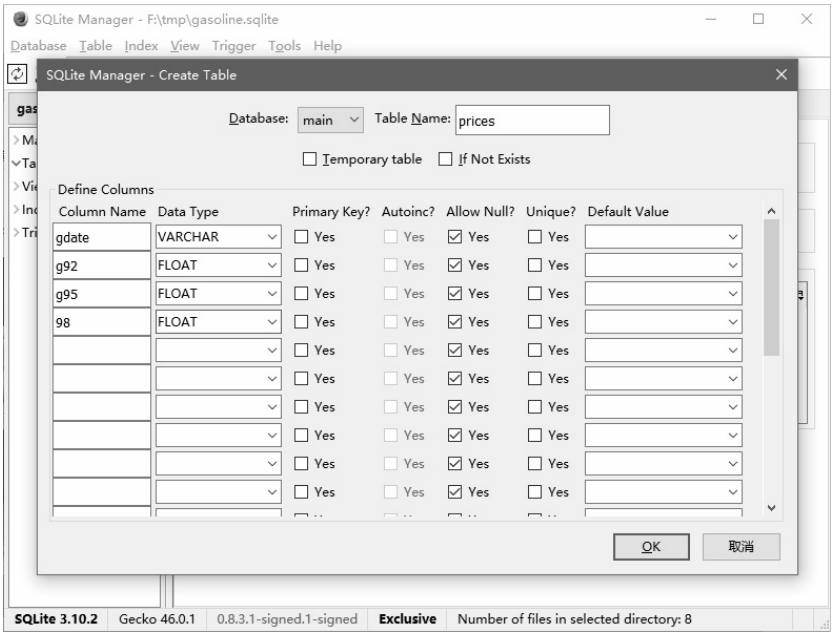


图 10-2　用来存储油价数据的数据库和数据表

第一个字段 gdate 使用可变长度的文字来存储就可以了，因此其类型指定为 VARCHAR；另外 3 个字段则因为有小数点，而且日后可能需要进行计算，所以设置为 FLOAT（浮点数类型），分别命名为 g92、g95 以及 g98。根据之前定义的模式，我们把它们放在同一个程序中，所以开始执行程序的时候，会先在屏幕界面上显示一个菜单：

```
历年油价查询系统
------------
1.从网站载入最新油价
2.显示历年油价信息
3.最近 10 周油价信息
```

```
4.油价走势图
0.结束
------------
请输入你的选择:
```

如菜单所示，程序中不再是一执行就去网页提取数据，相反，我们让到网页提取数据成为其中的一个选项，只有用户选用的时候才去执行，而且在提取数据之后就存储在之前定义的数据表中，之后其他 3 个选项都是从数据库中获取数据而不是从网页，速度就会快很多，而且节省了许多网络数据流量。主程序代码如下。

```
# _*_ coding: utf-8 _*_
# 程序 10-1.py (Python 3 version)

import sqlite3
from bs4 import BeautifulSoup
import requests
import NumPy as np
import matplotlib.pyplot as pt

conn = sqlite3.connect('gasoline.sqlite')

while True:
    disp_menu()
    choice = int(input("请输入你的选择:"))
    if choice == 0 : break
    if choice == 1:
        fetch_data()
    elif choice == 2:
        disp_alldata()
    elif choice == 3:
        disp_10data()
    elif choice == 4:
        chart()
    else: break
    x = input("请按 Enter 键回主菜单")
```

如同在第 8 章说明的，先打开 SQLite 数据库的链接，此链接设置为 conn 全局变量，在所有的函数中均可直接使用。各项功能的程序代码均放到对应的函数中，显示菜单使用 disp_menu()，提取网页数据使用 fetch_data()，显示所有的油价数据使用 disp_alldata()，显示前 10 笔数据则使用 disp_10data()，最后要绘出油价走势图，则是放在 chart()函数中。

在 fetch_data()函数中，我们直接把第 9 章中的油价网页提取程序放在函数中，不同的地方在于，原本提取后的数据是直接输出成文本文件，现在改为以 SQL 的 insert into 指令写入数据库中。这样做的好处是，当要使用其他的函数的时候（包括下次重新执行程序的时候），只要从本地的数据库中取出即可。所以，其他的 3 个函数要使用数据时，使用的都是 SQL 的 Select 指令。以下是 fetch_data()的程序片段：

```
def fetch_data():
    url = 'https://new.cpc.com.tw/division/mb/oil-more4.aspx'

    html = requests.get(url).text
    sp = BeautifulSoup(html, 'html.parser')
    data = sp.find_all('span', {'id':'Showtd'})
    rows = data[0].find_all('tr')

    prices = list()
    for row in rows:
        cols = row.find_all('td')
        if len(cols[1].text) > 0:
            item = [cols[0].text, cols[1].text, \
                    cols[2].text, cols[3].text]
            prices.append(item)
    for p in prices:
        sqlstr = "select * from prices where gdate='{}';".format(p[0])
        cursor = conn.execute(sqlstr)
        if len(cursor.fetchall()) == 0:
            g92 = 0 if p[1]=='' else float(p[1])
            g95 = 0 if p[2]=='' else float(p[2])
            g98 = 0 if p[3]=='' else float(p[3])
            sqlstr = "insert into prices values('{}', {}, {}, {});". \
                format(p[0], g92, g95, g98)
            print(sqlstr)
            conn.execute(sqlstr)
            conn.commit()
```

在提取网页数据的部分（第一个 for row in rows 循环），通过 append 方法把所有关于油价的数据放在 prices 列表中，接下来在写入数据库的部分（第二个 for p in prices 循环），则使用 select 指令先以日期为依据检查此项数据是否已在数据库中，确定不在数据库中才以 insert into 指令添加此项数据，以避免数据重复。在存入数据的同时，也要留意数据的格式，确实把油价的信息字段都调整为 float 类型才加入，如果有缺值的部分，要明确地设置为 0。至于绘图的部分，将在第 13 章进行完整的说明。完整的程序请参考程序 10-1（在执行此程序之前需要安装 NumPy、Matplotlib 等程序包）。

程序 10-1

```
# _*_ coding: utf-8 _*_
# 程序 10-1.py (Python 3 version)

import sqlite3
from bs4 import BeautifulSoup
import requests
import NumPy as np
```

```
import matplotlib.pyplot as pt

def disp_menu():
    print("历年油价查询系统")
    print("------------")
    print("1.从网站载入最新油价")
    print("2.显示历年油价信息")
    print("3.最近 10 周油价信息")
    print("4.油价走势图")
    print("0.结束")
    print("------------")

def fetch_data():
    url = 'https://new.cpc.com.tw/division/mb/oil-more4.aspx'

    html = requests.get(url).text
    sp = BeautifulSoup(html, 'html.parser')
    data = sp.find_all('span', {'id':'Showtd'})
    rows = data[0].find_all('tr')

    prices = list()
    for row in rows:
        cols = row.find_all('td')
        if len(cols[1].text) > 0:
            item = [cols[0].text, cols[1].text, \
                    cols[2].text, cols[3].text]
        prices.append(item)
    for p in prices:
        sqlstr = "select * from prices where gdate='{}';".format(p[0])
        cursor = conn.execute(sqlstr)
        if len(cursor.fetchall()) == 0:
            g92 = 0 if p[1]=='' else float(p[1])
            g95 = 0 if p[2]=='' else float(p[2])
            g98 = 0 if p[3]=='' else float(p[3])
            sqlstr = "insert into prices values('{}', {}, {}, {});". \
                format(p[0], g92, g95, g98)
            print(sqlstr)
            conn.execute(sqlstr)
            conn.commit()


def disp_10data():
    cursor = conn.execute('select * from prices order by gdatedesc;')
    n = 0
    for row in cursor:
```

```
        print("日期：{}，92 无铅：{}，95 无铅：{}，98 无铅：{}". \
            format(row[0],row[1],row[2],row[3]))
        n = n + 1
        if n == 10:
            break

def chart():
    data = []
    cursor = conn.execute('select * from prices order by gdate;')
    for row in cursor:
        data.append(list(row))
    x = np.arange(0,len(data))
    dataset = [list(), list(), list()]
    for i in range(0, len(data)):
        for j in range(0,3):
            dataset[j].append(data[i][j+1])
    w = np.array(dataset[0])
    y = np.array(dataset[1])
    z = np.array(dataset[2])
    pt.ylabel("NTD$")
    pt.xlabel("Weeks ( {} --- {} )".format(data[0][0], data[len(data)-1][0]))
    pt.plot(x, w, color="blue", label="92")
    pt.plot(x, y, color="red", label="95")
    pt.plot(x, y, color="green", label="98")
    pt.xlim(0,len(data))
    pt.ylim(10,40)
    pt.title("Gasoline Prices Trend (Taiwan)")
    pt.legend()
    pt.show()

def disp_alldata():
    cursor = conn.execute('select * from prices order by gdatedesc;')
    n = 0
    for row in cursor:
        print("日期：{}，92 无铅：{}，95 无铅：{}，98 无铅：{}". \
            format(row[0],row[1],row[2],row[3]))
        n = n + 1
        if n == 20:
            x = input("请按 Enter 键继续...(Q:回主菜单)")
            if x == 'Q' or x == 'q': break
            n = 0

conn = sqlite3.connect('gasoline.sqlite')

while True:
```

```
    disp_menu()
    choice = int(input("请输入你的选择:"))
    if choice == 0 : break
    if choice == 1:
        fetch_data()
    elif choice == 2:
        disp_alldata()
    elif choice == 3:
        disp_10data()
    elif choice == 4:
        chart()
    else: break
    x = input("请按 Enter 键回主菜单")
```

执行过程如图 10-3 所示。执行程序 10-1.py 之后，首先会出现菜单，第一次执行此程序需选择第一个选项执行网页数据提取的操作，日后除非网站数据有更新，不然都不需要再执行提取操作，因为所有的数据都已在本地的数据库中了。

当提取了数据之后，选择第二个选项会显示全部的油价数据，每 20 行会暂停，等待用户按 Enter 键后，才继续显示接下来的 20 笔数据。如果用户选择先按 Q 键再按 Enter 键，就会回到主菜单。若选择第三个选项，则只显示最近的 10 笔油价信息。

执行第四个选项，画出的油价走势图如图 10-4 所示。

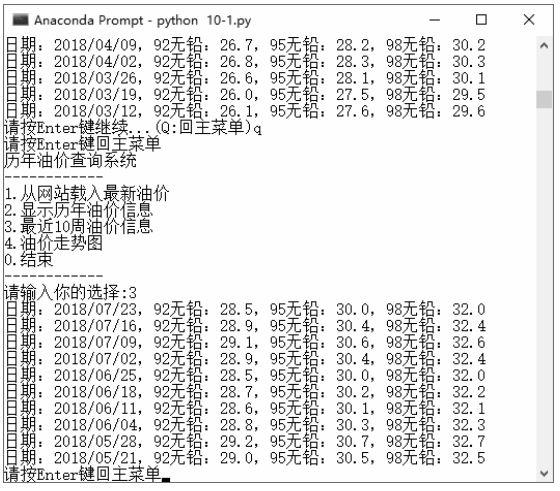


图 10-3　程序 10-1 执行的过程

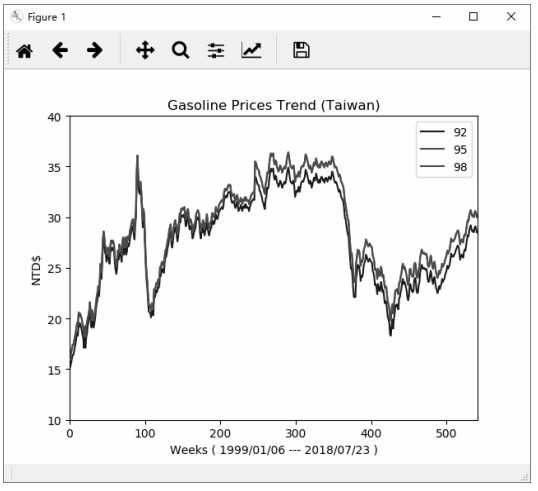


图 10-4　油价走势图

10-1-3　把数据导入网络 MySQL 数据库中

要把数据放在网页上供自己或网友浏览，最重要的就是把数据放在网络的数据库服务器中并建立一个自己的网页（网站）。而大部分网页服务器所支持的数据库服务器都是 MySQL，因此在本小节我们将以 MySQL 服务器为主，先教大家如何把现有的数据导入 MySQL 服务器中，再通过

简单的 PHP 程序设计语言把此数据显示在网页上。

在学习本小节的内容前，读者需要有自己的网页服务器（虚拟主机空间），既有免费的可以申请，也有付费的网站可供选用。虽然网上提供的免费或者付费的虚拟主机均有内建的 MySQL 服务器可供使用，但是为了方便使用 Python 建立数据，我们还是使用免费的 MySQL 服务器 http://db4free.net 来存储我们从网页提取的数据。

在把数据导入 db4free 之前，我们先通过 Firefox 浏览器的 SQLite Manager 导出数据。为了提供兼容性，我们选择导出为 CSV 格式。首先执行 Firefox 浏览器，并启动 SQLite Manager，打开之前的油价数据库文件 gasoline.sqlite，如图 10-5 所示。

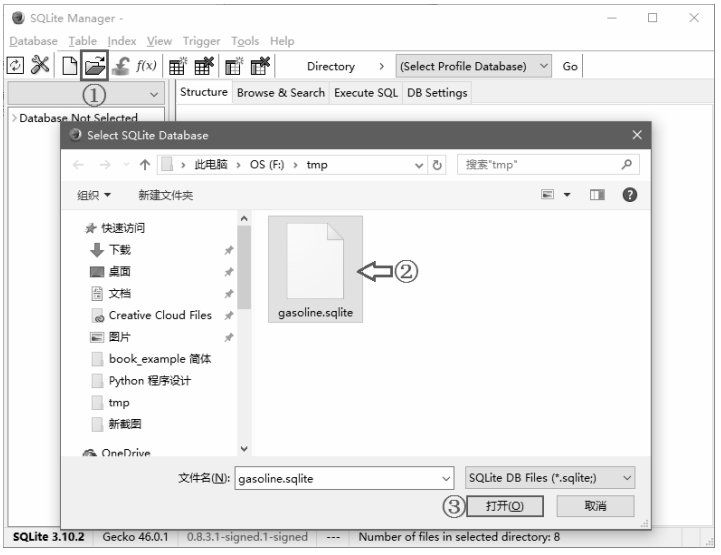


图 10-5　在 SQLite Manager 中打开原有的油价数据库文件

打开之后，找到 prices 数据表，在数据表名称上方右击，选择“Export Table”功能，如图 10-6 所示。

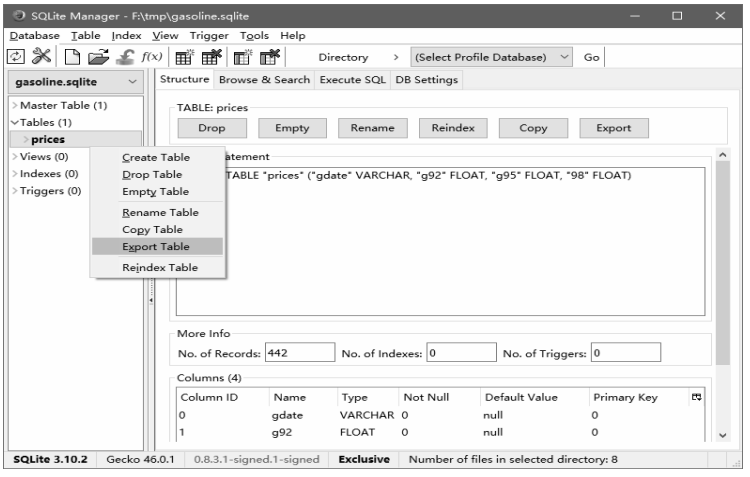


图 10-6　选择导出数据表的菜单选项

接着在图 10-7 中设置所有需要的参数选项。

基于最高兼容性，我们导出为 CSV（以逗号分隔的数据文件）格式，并指定第一行为字段名（勾选 First row contains column names 复选框），单击“OK”按钮之后就会出现存盘的对话框，如图 10-8 所示。

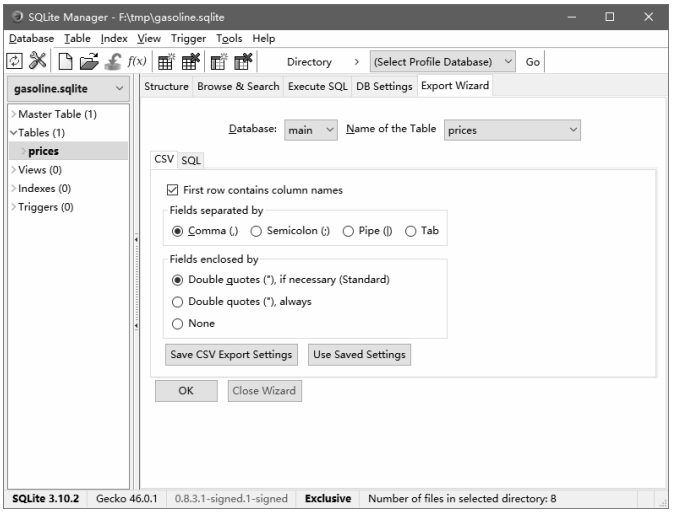


图 10-7　导出数据表为 CSV 格式文件

指定了保存的文件名和文件夹位置之后，即可前往 db4free.net 申请建立一个免费的 MySQL 数据库（如果此网站不稳定或速度过慢，可以参照第 9 章 9-3-3 小节中的说明，安装 WAMP 或 MAMP，在本地计算机建立 MySQL 服务，此时主机的地址只要使用 localhost 即可），如图 10-9 所示。

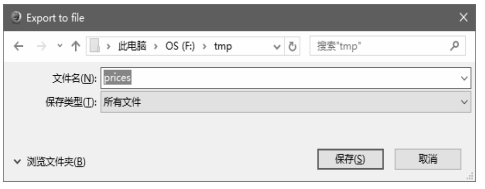


图 10-8　指定保存的 CSV 文件名

图 10-9　在 db4free.net 注册一个新的数据库

由于该网站已经全面中文化，因此申请注册的细节就不在此多做说明了。在此例中，申请了一个叫作 juntest 的数据库，在登录之后，马上就会出现最受欢迎的 MySQL 数据库管理界面 phpMyAdmin，如图 10-10 所示。

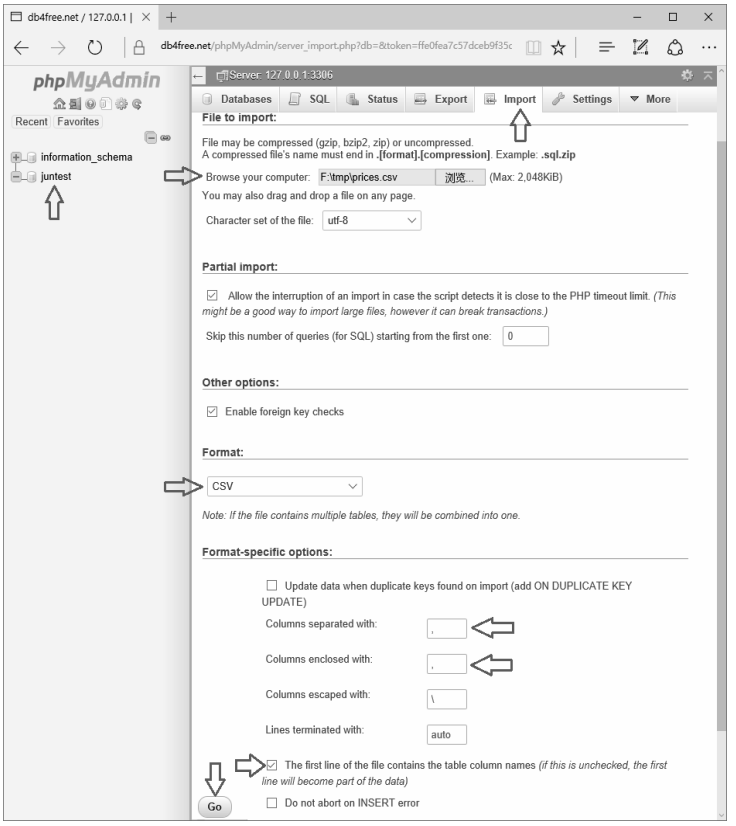


图 10-10　在 db4free 的 phpMyAdmin 界面中导入数据表

在图 10-10 标记箭头处进行必要的设置。注意，字段分隔符和内容分隔符都要设置为半角型的逗号“,”才能够正常导入。单击“Go”按钮，一会就可以顺利地导入数据表了，如图 10-11 所示。

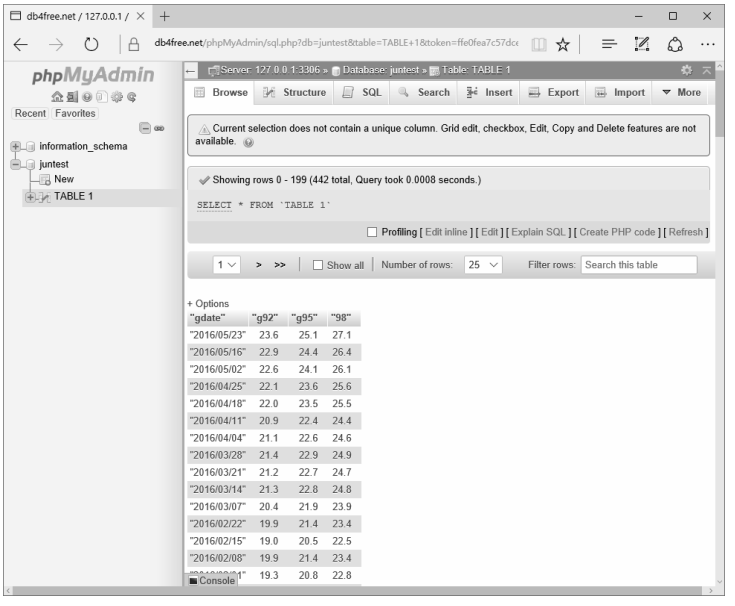


图 10-11　油价信息顺利导入的界面

有了此数据表之后，由于此数据库是因特网上的开放数据库，因此只要启用了确认邮件中的链接网址、账号及密码，在任何地方均可以编写程序读取这些数据，而且不需要再浪费时间以及不必要的网络流量去重新提取网页数据。

10-1-4 编写本地程序读取网络 MySQL 数据库中的数据

在 10-1-3 小节，我们已经把数据库放在 db4free.net 中了，也就是说，只要你的计算机连接了网络，在任何地方编写 Python 程序都可以获取这些数据，而不用再重新到别人的网页中去提取了。

然而，要在个人计算机中使用 Python 存取 MySQL 数据库，需要在计算机中安装可以存取 MySQL 的接口模块，最简单的方式是以如下指令安装（在此使用的是 mysqlclient 程序包，其说明文件的网址为 https://mysqlclient.readthedocs.io/）：

```
pip install mysql-python
```

如果计算机的操作系统是 Mac OS 或 Ubuntu Linux，别忘了在指令的前面加上 sudo。安装完成之后，就可以通过以下程序代码连接到 MySQL 数据库服务器了（要先 import _mysql，不要忘了前面的下画线符号）：

```
db = connector.connect(
    host = '主机链接地址',
    user = '数据库管理员名称',
    passwd = '管理员密码',
    database='数据库名称'
}
```

上述连接数据不加上参数名称也可以，只要按照其顺序来指定就好了。在连接完成之后，得到的是 db 指针，通过这个指针即可使用 SQL 指令执行检索操作，并取得所要的数据。例如以下这段程序，就是取出所有在 PRICES 数据表中的内容，并把它们存放在 rows 列表中。

```
db.query('select * from PRICES;')
r = db.store_result()
rows = r.fetch_row(maxrows=0)
```

运用 SQL 指令，我们也可以只提取前 10 笔数据的日期和 95 无铅汽油油价的字段：

```
db.query('select gdate, g95 from PRICES limit 10;')
```

综合以上说明，我们可以编写一段程序，连接数据库之后，把所有的数据从 db4free.net 下载到 rows 列表变量中，然后全部打印出来。请参考程序 10-2 的内容。

程序 10-2

```
# -*- coding: utf-8 -*-
# 10-2.py (Python 3 version)

import _mysql

db = _mysql.connect(
```

```
    host='db4free.net',
    user='ptest',
    passwd='****',
    db='ptest')
db.query('select * from PRICES;')
res = db.store_result()
rows = list()
while res.has_next:
  row = res.fetch_row()
  rows.append(row)

for i in range(0,10):
  print("日期：{}, 92 无铅：{}, 95 无铅：{}, 98 无铅：{}".\
      format(rows[i][0], rows[i][1], rows[i][2], rows[i][3]))
```

程序的运行结果如下。

```
$ python 10-2.py
日期："2016/01/25", 92 无铅：18.3, 95 无铅：19.8, 98 无铅：21.8
日期："2016/01/18", 92 无铅：18.4, 95 无铅：19.9, 98 无铅：21.9
日期："2016/01/11", 92 无铅：19.5, 95 无铅：21.0, 98 无铅：23.0
日期："2015/12/28", 92 无铅：19.9, 95 无铅：21.4, 98 无铅：23.4
日期："2015/12/21", 92 无铅：20.1, 95 无铅：21.6, 98 无铅：23.6
日期："2015/12/14", 92 无铅：20.8, 95 无铅：22.3, 98 无铅：24.3
日期："2015/12/07", 92 无铅：21.6, 95 无铅：23.1, 98 无铅：25.1
日期："2015/11/30", 92 无铅：21.7, 95 无铅：23.2, 98 无铅：25.2
日期："2015/11/23", 92 无铅：21.5, 95 无铅：23.0, 98 无铅：25.0
日期："2015/11/16", 92 无铅：22.2, 95 无铅：23.7, 98 无铅：25.7
```

可以从 MySQL 中读取数据，当然也能够写入数据。这部分就留给读者作为习题使用。

10-1-5　使用 PHP 建立信息提供网站

在 10-1-4 小节中，我们使用 Python 在个人计算机上创建了一个程序，可以读取在因特网上的 db4free.net 数据库，这样做的好处是，无论你使用哪一台计算机执行范例程序 10-2，都可以存取同一个数据库。然而，如果要把这些数据分享给其他的网友，这种方式就不方便了，最好的方式是建立一个网站来作为显示这些信息的接口。如果你原本就有网站，就可以轻易地通过这项功能在你的网站中提供来源于其他网页但是经过你分析整理的信息（例如实时汇率、股价、天气以及地震消息等）。

Python 也可以作为网页服务器的后端语言，这点我们将在第 14 章中详细介绍。在这一小节中，我们以大家常用的 PHP 作为范例，示范如何在 PHP 中读取 10-1-4 节中的数据，并显示在网页中。同样，如果你没有网站虚拟主机，也可以到网上申请免费的，或者在付费网站购买一个网站虚拟主机。

任一网页服务器中的 PHP 文件要存取远程的 MySQL 数据库，只要使用以下程序片段即可：

```
  $dbuser='juntest';
  $dbname='juntest';
  $dbhost = 'db4free.net';
  $dbpasswd = '******';

  $conn = mysql_connect($dbhost, $dbuser, $dbpasswd) or die ('connect error');
```

主要是使用 mysql_connect 函数进行数据库连接工作，一旦连接完成，PHP 后台会自动为我们处理持续的连接工作，这时只要通过 mysql_query 送出 MySQL 的查询指令即可。当然在此之前，要先使用 mysql_select_db 指定要操作的数据库名称，程序片段如下：

```
mysql_select_db($dbname);
$res = mysql_query('select * from PRICES order by gdatedesc limit 20;');
```

此时所有的数据都已放在$res 变量中了，通过以下循环即可取出所有的数据：

```
while($row = mysql_fetch_array($res)) {
    echo $row[0] . $row[1] . $row[2] . $row[3];
}
```

其中，$row[0]和$row[1]分别代表每一个数据记录中的第 0 个字段和第 1 个字段。如此搭配 HTML 和 PHP 的语法，我们即可轻松编写出可以从 db4free.net 数据库中取出数据的程序。请参考程序 10-3 的内容。

程序 10-3

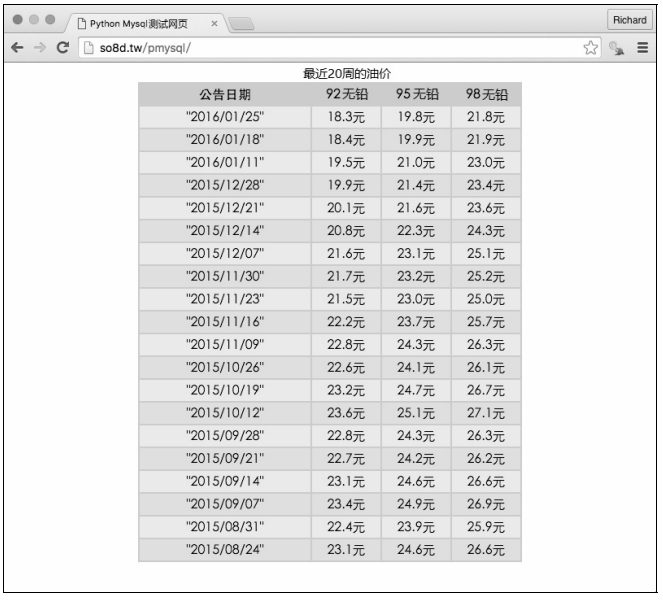
```
<!-- 程序 10-3 (PHP version) -->
<!DOCTYPE html>
<html lang='zh-Hans-CN'>
<head>
<title>Python Mysql 测试网页</title>
</head>
<body>
<table align='center' width='60%' bgcolor='#cccccc'
cellpadding=5 cellspacing=2>
<caption align='center'>最近 20 周的油价</caption>
<tr><th>公告日期</th><th>92 无铅</th><th>95 无铅</th><th>98 无铅</th></tr>
<?php
  $dbuser='juntest';
  $dbname='juntest';
  $dbhost = 'db4free.net';
  $dbpasswd = '******';

  $conn = mysql_connect($dbhost, $dbuser, $dbpasswd) or die ('connect error');
  mysql_select_db($dbname);
  $res = mysql_query('select * from PRICES order by gdatedesc limit 20;');
  $i=0;
```

```
  while($row = mysql_fetch_array($res)) {
    $i++;
    if($i%2)
       echo '<trbgcolor=#ccffcc>';
    else
       echo '<trbgcolor=#ffccff>';
    echo '<td width=200 align=center>' . $row[0] . '</td>' .
         '<td align=center>' . $row[1] . '元</td>' .
         '<td align=center>' . $row[2] . '元</td>' .
         '<td align=center>' . $row[3] . '元</td>';
    echo '</tr>';
  }
mysql_close($conn);
?>
</table>
</body>
</html>
```

此程序是用 PHP 语言写成的，要在能够执行 PHP 的网页服务器中才可以使用，既可以放在虚拟主机中，也可以在 MAMP 或 WAMP 的网页目录中执行（在个人计算机中安装过 MAMP 或 WAMP）。请注意，不是直接执行，而是通过浏览器存取该文件或网址。以程序 10-3 为例，我们将程序 10-3 命名为 index.php，并放在网站 http://so8d.tw 的 pmysql 文件夹之下，通过浏览器前往 http://so8d.tw/pmysql，即可看到执行结果。读者可以把这个范例程序的实现放在自己的虚拟主机网站上测试一下。

程序 10-3 的运行结果如图 10-12 所示。



公告日期	92无铅	95无铅	98无铅
"2016/01/25"	18.3元	19.8元	21.8元
"2016/01/18"	18.4元	19.9元	21.9元
"2016/01/11"	19.5元	21.0元	23.0元
"2015/12/28"	19.9元	21.4元	23.4元
"2015/12/21"	20.1元	21.6元	23.6元
"2015/12/14"	20.8元	22.3元	24.3元
"2015/12/07"	21.6元	23.1元	25.1元
"2015/11/30"	21.7元	23.2元	25.2元
"2015/11/23"	21.5元	23.0元	25.0元
"2015/11/16"	22.2元	23.7元	25.7元
"2015/11/09"	22.8元	24.3元	26.3元
"2015/10/26"	22.6元	24.1元	26.1元
"2015/10/19"	23.2元	24.7元	26.7元
"2015/10/12"	23.6元	25.1元	27.1元
"2015/09/28"	22.8元	24.3元	26.3元
"2015/09/21"	22.7元	24.2元	26.2元
"2015/09/14"	23.1元	24.6元	26.6元
"2015/09/07"	23.4元	24.9元	26.9元
"2015/08/31"	22.4元	23.9元	25.9元
"2015/08/24"	23.1元	24.6元	26.6元

图 10-12　在网站上通过 PHP 读取数据之后的网页

当然，也可以把这些程序片段放在你现有的网页中，丰富你的网站信息。

至于如何让本地的 Python 程序可以直接把数据存储到 db4free.net 数据库中，请参考 10.2 节的内容。

10-2 自动提取数据

在前面几节中，我们学会了如何编写 Python 程序从网页上提取数据，然后放在网站数据库中，并通过本地的 Python 程序或服务器后端程序设计语言 PHP 来获取已存储在数据库中的数据。接着在本节中，我们将学习如何让这些过程自动化。

也就是说，在本地计算机中设置自动执行程序，定期执行我们编写的网页数据提取程序，并存储在数据库中，以供日后使用。

10-2-1 检测网页内容是否曾经更新

为了避免同样的网页被重复分析，在这一节中我们将教大家一个简单的判断技术，即通过 md5 函数获取网页的摘要，如果此次的摘要和上次的一样，就表示网页内容并没有被更新过，不需要再重新分析以及存储数据了。方法如下：

```
import requests
import hashlib
r = requests.get('http://target.web.site.page')
sig = hashlib.md5(r.text.encode('utf-8')).hexdigets()
```

把计算后的 sig 与之前计算过的 sig（记录在数据库或文本文件中）相比较，两个值不一样才会继续往下执行程序。因此，在我们的网站分析程序中要有此记录才行。

为了简化起见，假设程序只针对一个网站进行分析和提取数据，因此每次开始执行的时候都会先查找 eq_sig.txt 是否存在，如果存在，就读取出来作为对比的依据，如果文件更新了，就在分析处理完网页数据之后再把新的摘要（最新计算出来，存放在 sig 变量中的）更新到 eq_sig.txt 中，以备下次使用。

在此，以第 8 章使用过的 USGS 提供的地震信息（网址为 http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/4.5_week.geojson）为例，假设我们想要编写一个程序从该网站下载数据，获取最近一周所有震级超过 4.5 度的地震数据（含震级、日期以及地点），然后把这些数据存放在 MySQL 数据库（在此例中为 db4free.net）中，同时避免对一模一样的数据重复分析及处理（例如后续的数据库操作）。

因为要把数据存储在 MySQL 数据库中，所以为了简化程序，不用再处理创建数据表的相关问题，先前往 db4free.net 创建一个名为 eq 的数据表，如图 10-13 所示。

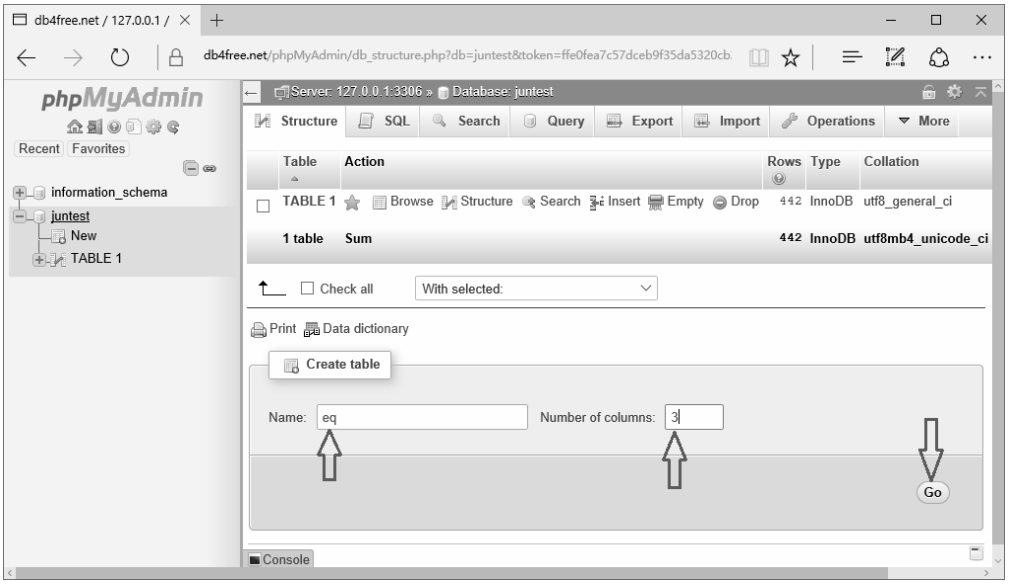


图 10-13　在 db4free.net 中建立数据表

在单击“Go”按钮之后，接下来设置字段的格式，如图 10-14 所示。

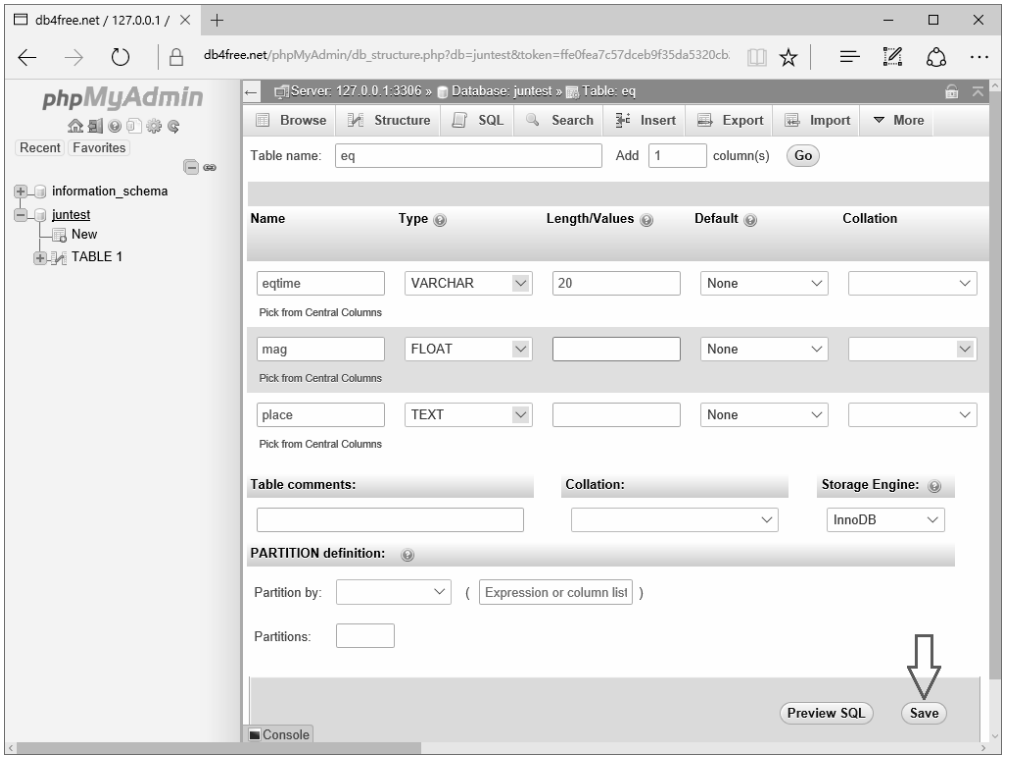


图 10-14　设置 eq 数据表的字段格式

在此我们设置 3 个字段，分别是 eqtime（VARCHAR）、mag（FLOAT）以及 place（TEXT），分别用来记录地震发生的时间、震级以及地点。设置完成的屏幕显示界面如图 10-15 所示。

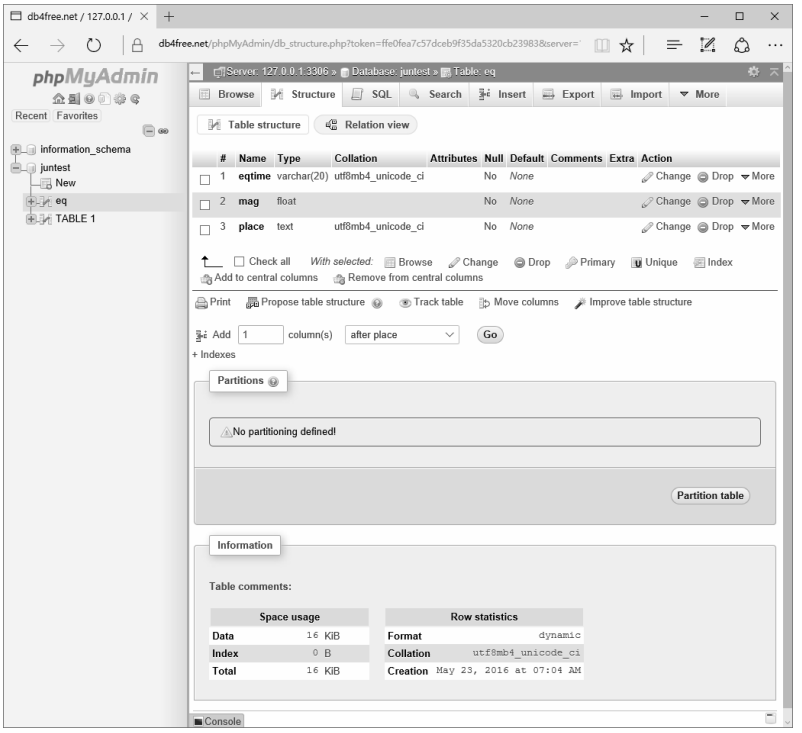


图 10-15　数据表 eq 设置完毕后的摘要页面

主要的程序内容如程序 10-4 所示。

程序 10-4

```
    # _*_ coding: utf-8 *_*
    # 程序 10-4 (Python 3 version)

    import json, requests, hashlib, datetime, os.path
    from mysql import connector

    url = 'http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/
4.5_week.geojson'
    r = requests.get(url)
    sig = hashlib.md5(r.text.encode('utf-8')).hexdigest()
    old_sig=''

    if os.path.exists('eq_sig.txt'):
        with open('eq_sig.txt', 'r') as fp:
            old_sig = fp.read()
        with open('eq_sig.txt', 'w') as fp:
            fp.write(sig)
    else:
        with open('eq_sig.txt', 'w') as fp:
            fp.write(sig)

    if sig == old_sig:
```

```
        print('数据未更新，不需要处理...')
    exit()

    earthquakes = json.loads(r.text)
    dataset = list()
    for eq in earthquakes['features']:
        item = dict()
        eptime = float(eq['properties']['time']) /1000.0
        d = datetime.datetime.fromtimestamp(eptime).strftime('%Y-%m-%d %H:%M:%S')
        item['eqtime'] = d
        item['mag'] = eq['properties']['mag']
        item['place'] =  eq['properties']['place']
        dataset.append(item)

    db = connector.connect(
        host = 'db4free.net',
        user = 'juntest',
        passwd = '******',
        db = 'juntest')

    db.query('delete from eq;')
    for data in dataset:
        sql = 'insert into eq (`eqtime`,`mag`,`place`)
values("{}",{},"{}");'.format( \
             data['eqtime'], data['mag'], data['place'])
        db.query(sql)
        print(sql)
    print('数据更新完成')
    db.commit()
    db.close()
```

程序 10-4 的第一段使用 with 指令来打开文本文件 eq_sig.txt，用来作为识别所提取的网络数据是否和上一次提取数据一样的验证数据，只有不一样才会继续往下执行数据库的导入操作。如果网页的数据经常更新，其实你也可以跳过这一段测试，每一次执行的时候都进行导入数据库的工作。

在导入数据库之前，我们先把所得到的 json 格式数据转换为列表数组，需要的数据放在 dataset 中。有了所有的数据之后，再连接 db4free.net 的数据库，并用 SQL 指令把 dataset 中所有的数据都导入数据库中。由于数据量并不多，因此我们简化了数据库重复数据的检查操作，直接在执行导入之前把表格内所有的数据都先用“delete from eq”全部删除后再导入新的数据。

值得注意的是，如果需要保留原有的数据，但是当前提取的网络数据又有可能会有一些重复数据项，在 insert into 之前，就要先用 select 指令查找，看看当前的数据库中有没有你要新增的数据，没有的话再导入。

最后，要确认数据库内所有的数据都确实被更新了，在退出程序之前，还要有一个 commit 操作，程序结束之后，也别忘了用 close 关闭数据库的连接。运行结果如图 10-16 所示，所有的数据都会被导入 db4free.net 的数据库中。

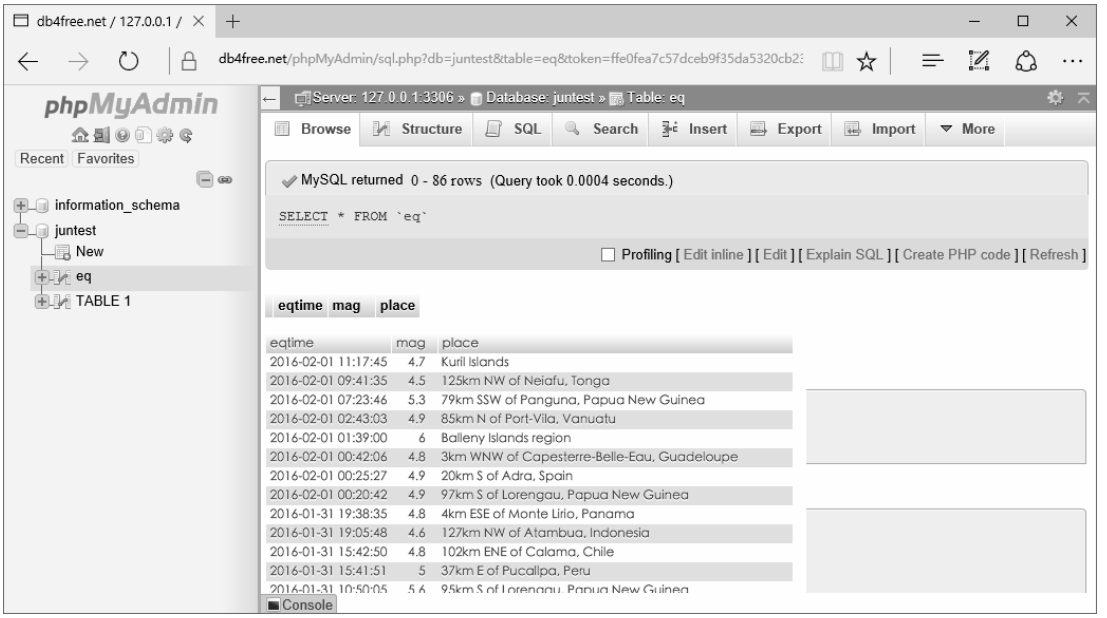


图 10-16　程序 10-4 的运行结果

既然每一次执行都可以自动为我们更新数据库，那么接下来就来看看如何在自己的计算机操作系统中设置自动执行的功能。为了简化说明起见，因为我们所获取的数据量并不多，所以把程序 10-4 简化为程序 10-5，下载之后不做重复检查，直接更新数据库，而且不做任何输出。

程序 10-5

```
    # _*_ coding: utf-8 *_*
    # 程序 10-5 (Python 3 version)

    import json, requests, datetime
    from mysql import connector

    url = 'http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/
4.5_week.geojson'
    r = requests.get(url)

    earthquakes = json.loads(r.text)
    dataset = list()
    for eq in earthquakes['features']:
        item = dict()
        eptime = float(eq['properties']['time']) /1000.0
        d = datetime.datetime.fromtimestamp(eptime). \
            strftime('%Y-%m-%d %H:%M:%S')
        item['eqtime'] = d
        item['mag'] = eq['properties']['mag']
        item['place'] =  eq['properties']['place']
        dataset.append(item)

    db = connector.connect(
```

```
        host = 'db4free.net',
        user = 'juntest',
        passwd = '******'
        database = 'juntest')
    cur = db.cursor()
    cur.execute('delete from eq')
    for data in dataset:
        sql = 'insert into eq ('eqtime', 'mag', 'place') values("{}",{},"{}");
'.format( \
             data['eqtime'], data['mag'], data['place'])
        cur.execute(sql)
    db.commit()
    db.close()
```

10-2-2　Windows 自动化设置

在 10-2-1 小节中，程序 10-4.py 每执行一次，就会把位于 db4free.net 中的数据表更新一次，因为地震数据是每 5 分钟更新一次，所以如果想要在数据库中保持最新的数据，我们的程序最好能够每隔一段时间就执行一次。当然这个操作不需要由人工来做，操作系统本身就提供了定时执行程序的功能，只要设置好，在计算机开机的时候，程序就会被按时执行，达到自动化搜索和收集数据的目的。

首先，为了管理方便，我们在 C:\磁盘驱动器的根目录下创建一个专门用来放置自动执行程序的目录 C:\auto_python，然后把程序 10-5.py 复制到此目录之下。

Windows 操作系统中有一个“任务计划程序”可以负责自动化执行指定程序的工作，若是 Windows 10，则如图 10-17 所示，而 Windows 7 则如图 10-18 所示。

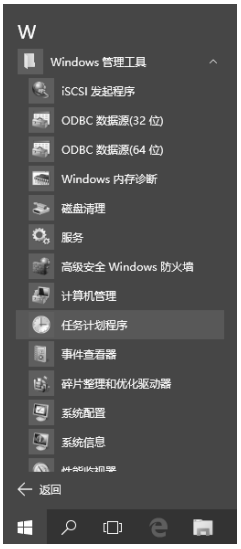


图 10-17　Windows 10 启动“任务计划程序”的地方

图 10-18　Windows 7 启动“任务计划程序”的地方

启动“任务计划程序”之后，打开如图 10-19 所示的屏幕显示界面。

图 10-19　任务计划程序主界面

在右侧选择“创建任务”选项，就会出现如图 10-20 所示的“创建任务”界面。

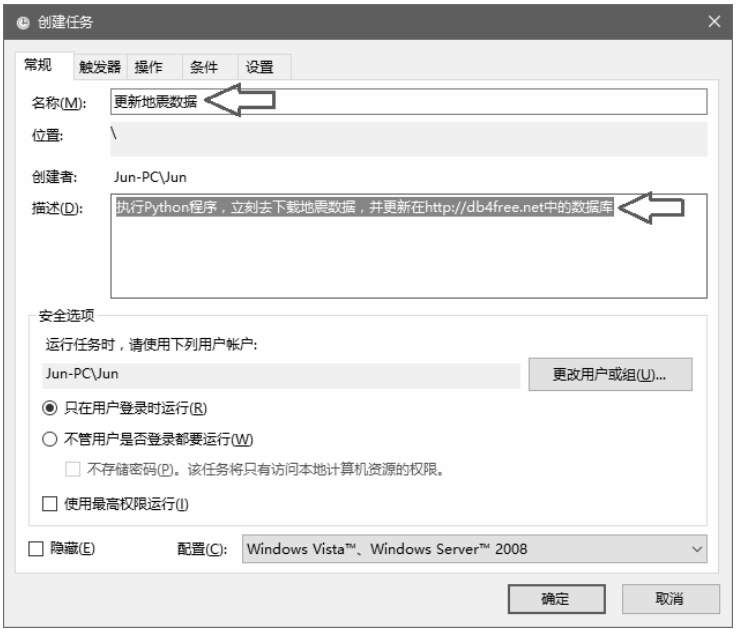


图 10-20　“创建任务”界面

如图 10-20 所示，先指定一个名称，并在“描述”中简单说明一下这个任务的目的，以免日后忘记。接下来单击“触发器”标签，如图 10-21 所示。

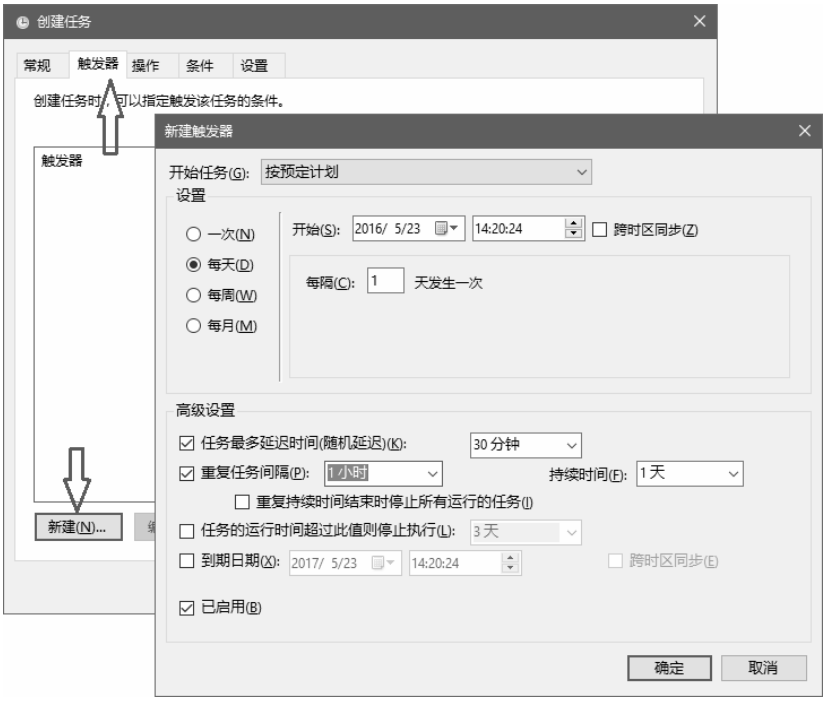


图 10-21　“任务计划程序”的“触发器”选项设置

在图 10-21 中可以设置此程序的运行时间以及重复工作的细节。我们可以在一天中的任一时间开始这项工作，并设置每隔多长时间要重复一次，当然也可以设置停止此工作的日期或条件。在单击“确定”按钮之后，再设置触发之后要执行的程序，如图 10-22 所示。

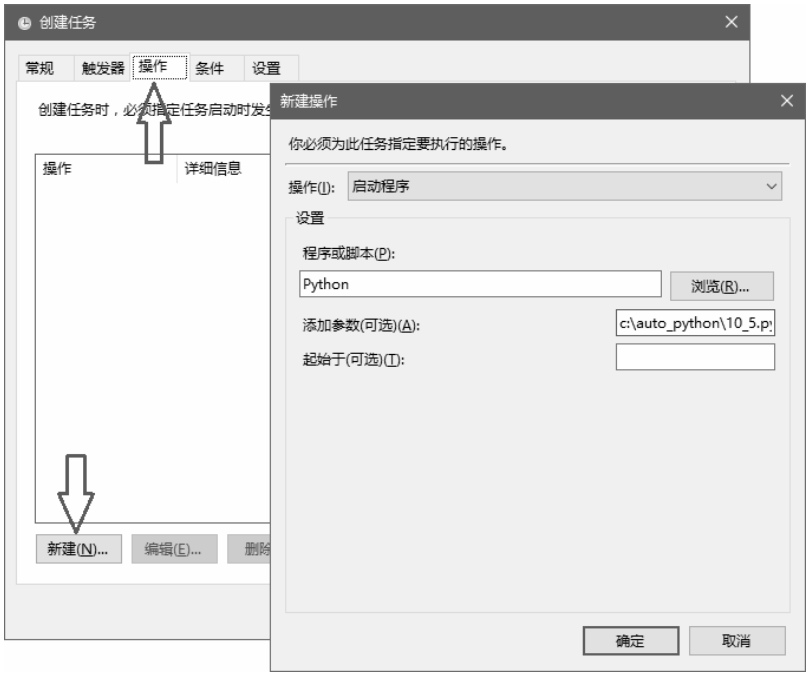


图 10-22　设置触发之后要执行的程序

在这里我们只要设置要执行的程序是 Python，添加参数（可选）设置为 c:\auto_python\10-5.py，再单击“确定”按钮就可以了。设置完毕回到主界面，如图 10-23 所示。

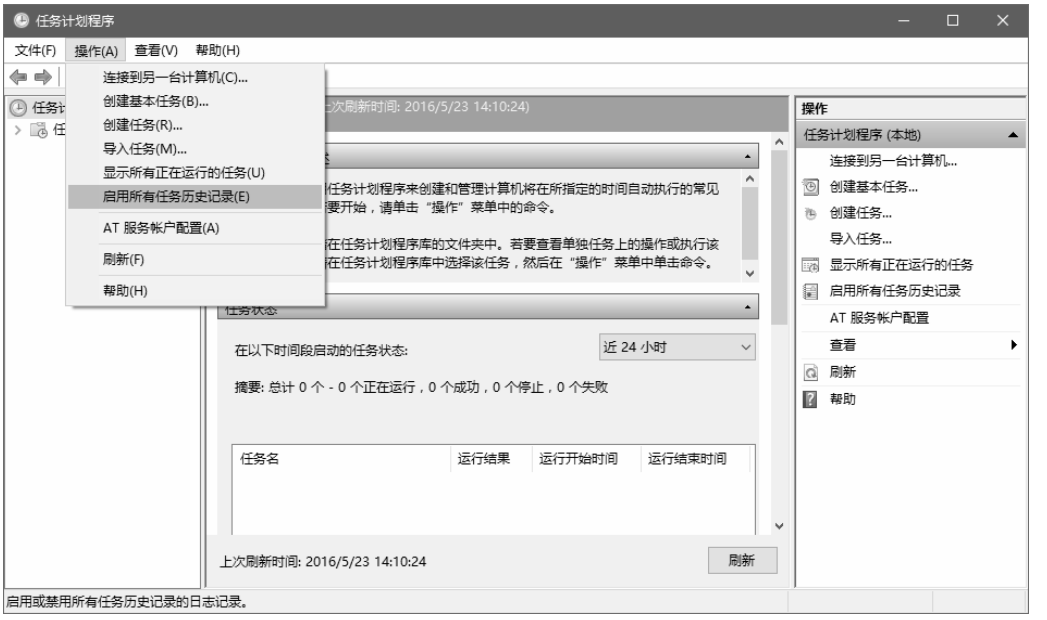


图 10-23　任务计划程序主界面可以执行的操作

如图 10-23 所示，在“操作”菜单中有许多项目可以选择。想要确定所设置的程序是否如期执行，除了观察结果之外，也可以选择“启用所有任务历史记录”，便于日后追踪核查。全部设置完成之后，回到主界面中单击左侧的“任务计划程序库”，就可以看到我们设置的成果了，如图 10-24 所示。

图 10-24　新创建任务的相关信息

10-2-3　Mac OS 自动化设置

在 Mac OS 下负责任务计划的和在 Linux 下一样，都是 crontab，而且因为在 Mac OS 下可以直接在程序 10-5.py 的第一行进行以下设置来执行此文件的程序：

```
#! /usr/bin/python
```

因此，在 Mac OS 操作系统下不需要再执行 python 10-5.py，而是直接执行 10-5.py 就可以。

在终端程序下执行 crontab -e 指令，即可进入设置自动执行的编辑环境，每一行均可设置一个程序，格式如下：

```
0 10 * * 1 ~/auto_python/10-5.py
```

其中，前面 5 个参数以空格隔开，这些数字代表的意义分别是分、时、日、月、周。如果是“*”，就表示该项目不进行设置。如上例，表示在每星期一的 10:00 执行后面的程序。“~”符号表示用户的根目录，所以“~/auto_python/10-5.py”就是要求执行用户根目录 auto_python 文件夹中的 10-5.py 这个程序。

上述格式若要设置成每天每隔 10 分钟执行一次，则改为：

```
*/10 * * * * ~/auto_python/10-5.py
```

若要设置的是每个月 1 日上午 10 点 15 分和 45 分各执行一次，则改为：

```
15,45 10 1 * * ~/auto_python/10-5.py
```

编辑器是使用系统默认的 vi 编辑器，使用方法请参考相关的数据。在设置完毕之后，可以使用以下指令查看：

```
crontab -l
```

经过以上设置后，在网络上搜集信息就不需要再自己动手了，非常方便。不过，因为提取网页数据会造成对方主机的额外负担，请留意相关的法律问题，同时不能太过于频繁和规律，这有可能会让你的网络 IP 被对方网站封锁。

10-3　通过 Python 操作浏览器

在前面几节中，我们都是使用 requests 模块提取网页数据的，但是有些比较复杂或使用 JavaScript 执行的网站有时通过浏览器来读取反而比较方便。在以往，我们直觉地认为浏览器的操作必须通过人工的方式来执行，其实不见得。在这一节中，我们会介绍 Selenium 模块。通过这个模块可以直接在 Python 程序中操作 Firefox 浏览器（经过安装其他相关模块后，也可以操作 Internet Explorer 和 Google Chrome，不过 Firefox 浏览器是默认值），就好像人工操作一样。

10-3-1 安装 Selenium

安装 Selenium 的方法很简单，一般只要使用 pip install 就可以了：

```
pip install selenium
```

如果之前为了使用 Python 绘图功能而安装了 Anaconda，那么有可能在上面的指令执行之后出现如下错误信息：

```
Cannot open e:\Anaconda3\Scripts\pip-script.py
```

这个信息表示在 Anaconda 中并没有安装过 pip 模块，以至于无法在此环境下使用 pip 安装新的模块。要解决这种情况，只要使用如下指令在 Anaconda 环境下安装 pip 即可：

```
conda install pip
```

Selenium 的网址为 http://selenium-python.readthedocs.org/（在官方网站中有详细的安装说明）。图 10-25 所示是在 Windows 10 操作系统下安装成功的屏幕显示界面。

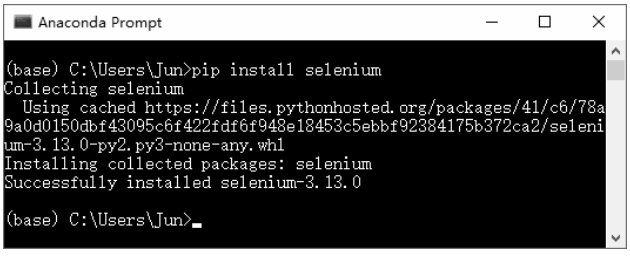


图 10-25 Selenium 在 Windows 10 完成安装的屏幕显示界面

由于 Selenium 默认使用的浏览器是 Firefox，因此如果你的计算机中没有这个浏览器，也要安装才行。Firefox 浏览器的网址为 https://www.mozilla.org/zh-CN/firefox/new/。不过新版的 Selenium 已经支持 Chrome，视你的使用习惯而定。

为了方便分析网页，在 Firefox 浏览器中有一个很好用的附件，即 Firebug，如图 10-26 所示。

图 10-26 Firefox 的 Firebug 附加组件

在安装 Firebug 之后，在网页的右上角会有一个 Firebug 的图标，单击该图标之后即可在下方实时看到网页的源代码以及相关的信息，同时在任一网页元素上右击之后，即可出现“使用 Firebug 查看元素”选项，单击该选项，网页元素所对应的源代码就会立即出现在下方，对于分析网页非常有帮助，如图 10-27 所示。

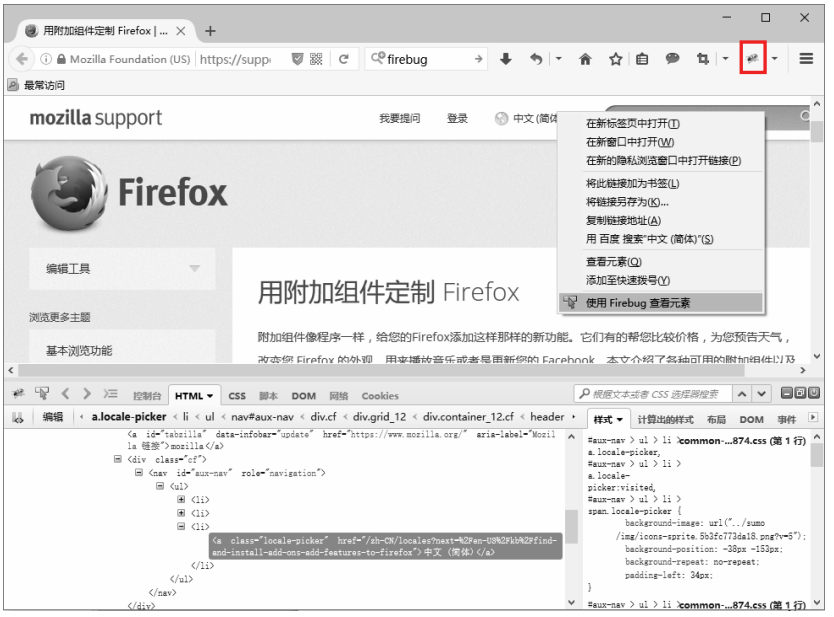


图 10-27　通过 Firebug 分析网页元素

假设我们使用的是 Chrome 浏览器，需要安装一个 ChromeDriver 的 WebDriver，这个程序可以在 https://sites.google.com/a/chromium.org/chromedriver/downloads 下载，选择最新的版本及 32 位版本即可，如图 10-28 所示。

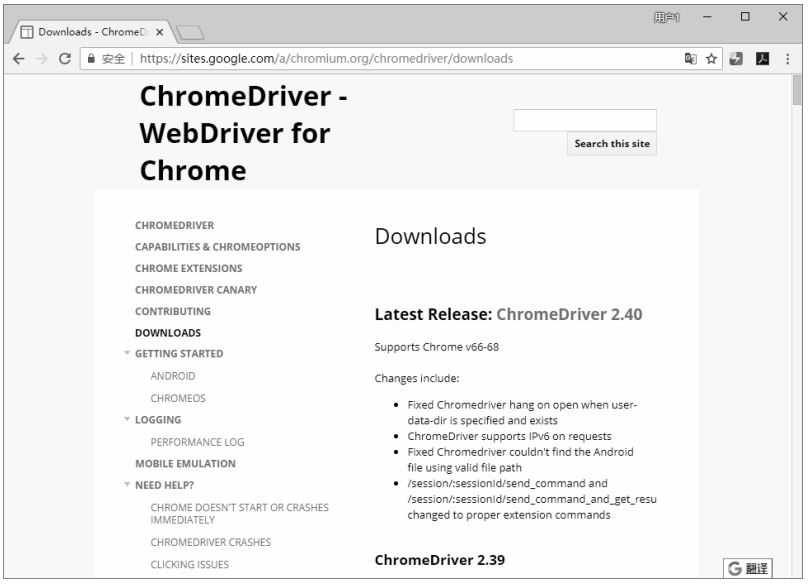


图 10-28　ChromeDriver 的安装页面

下载压缩文件之后，请执行解压缩，并把执行文件 chromedriver.exe 放在固定的文件夹中，在 Python 程序中需要运行这个程序才能够操控 Chrome 浏览器。此处这个文件存放的文件夹如图 10-29 所示。

图 10-29　chromedriver.exe 所存放的文件夹

10-3-2　使用 Selenium 操作 Chrome

要确定 Selenium 是否能够正常工作，最简单的方法是进入 Python 的交互式界面，输入以下程序代码，在 Chrome 函数调用中的网址就是我们存放 chromedriver.exe 的绝对路径。此外，路径名称外的“r”字符，是希望 Python 解释器不要解释或翻译它后面跟着的字符串，直接使用即可：

```
from selenium import webdriver
web = webdriver.Chrome(r"d:\MyPython\chromedriver.exe")
web.get('https://www.sina.com')
web.close()
```

事实上，不用等到整个程序写完，在执行到第 2 行的时候，一个空白、全新的 Chrome 浏览器就会被启动执行，而在第 3 行程序代码输入之后，该浏览器就会打开新浪网站，就好像是我们在网址栏输入该网址一样，如图 10-30 所示。

当然，在上述程序的最后一行（web.close()）输入之后，这个浏览器的窗口就会被关闭。除了close()之外，Selenium 还提供了非常多的方法可以操作浏览器，例如 get_window_position、set_window_position、maximize_window、get_window_size、set_window_size、refresh、back、forward 等。许多人工操作的功能都可以通过这些方法来取代，几个主要的功能如表 10-1 所示。

图 10-30　通过 Python 操作 Chrome 浏览器的屏幕显示页面

表 10-1　几个主要的功能

WebDriver 的方法	主要功能
get_window_position()	获取窗口的位置（左上角）
set_window_position(x, y)	设置窗口的位置（左上角）
maximize_window()	最大化窗口尺寸
get_window_size()	获取窗口的尺寸
set_window_size(x, y)	设置窗口的尺寸
refresh()	刷新页面
back()	回上一页
forward()	到下一页
close()	关闭窗口
quit()	结束浏览器的执行
get(url)	浏览 url 这个网址
save_screenshot(filename)	把当前的屏幕界面存成 PNG 格式，文件名设置为 filename
current_url	当前的网址
page_source	网页的原始文件（源代码文件）
title	当前网页的 title 设置

在表 10-1 中，在名称后面加小括号的是方法(Method)，需要采用调用的方式才可以执行它，其他的则是属性，直接取用即可。除了可以直接获取 page_source（网页源代码）之外，比较有趣的是可以存储网页的截图（截屏）。也就是说，如果我们有 5 个网页要浏览，并想截取这几个网页的屏幕显示界面，就可以编写一个程序自动完成这些工作，如程序 10-6 所示。

程序 10-6

```
# _*_ coding: utf-8 *_*
# 程序 10-6 (Python 3 version)
```

```
from selenium import webdriver
urls = [
'http://www.sina.com.cn',
'http://www.sohu.com',
'http://www.eastmoney.com',
'http://www.newone.com.cn/',
'http://www.baidu.com']

web = webdriver.Chrome(r"d:\MyPython\chromedriver.exe")
web.set_window_position(0,0)
web.set_window_size(800,600)
i = 0
for url in urls:
    web.get(url)
    web.save_screenshot("webpage{}.png".format(i))
    i += 1
web.close()
```

执行程序 10-6 之后，你会发现系统马上会启动 Chrome 浏览器，并被移到左上角，同时把窗口的大小切换成 800×600，并开始自动浏览我们指定的页面，直到 5 个网页都浏览完毕之后关闭窗口。接着到存放此程序的同一个目录下可以找到 5 个图像文件，分别是 webpage1.png、webpage2.png、webpage3.png、webpage4.png 和 webpage5.png，而且特别的是，每一个图像文件存储的都是完整的网页截屏界面，并不会受限于窗口的大小。非常有趣，你一定要试试。

10-3-3 通过 Selenium 读取网页信息

在通过 Selenium 的 WebDriver 打开某个网页之后，其实这个网页的源代码已经在我们的掌握之中了，我们既可以通过 page_source 获取所有的原始网页内容，也可以通过一些函数找出某个或某些特定的网页元素进行操作。不需要 BeautifulSoup，WebDriver 本身就提供网页元素的检索功能，请参考表 10-2。

表 10-2 几个主要的检索功能

WebDriver 的方法	主要功能
find_element(by, value)	使用 by 指定的方法查找第一个符合 value 的元素
find_element_by_class_name(name)	使用类名称查找符合的元素
find_element_by_css_selector(selector)	使用 CSS 选择器查找符合的元素
find_element_by_id(id)	使用 id 名称查找符合的元素
find_element_by_link_text(text)	使用链接文字查找符合的元素
find_element_by_name(name)	使用名称查找符合的元素
find_element_by_tag_name(name)	使用 HTML 标签查找符合的元素
上面的方法在 element 后面加上 s	同上，但是返回的是数组，其中含有所有符合的元素

通过以上函数可以找到当前使用 Chrome 打开的网页中的任一元素，至于要使用哪一个函数，则视网页分析的结果而定。我们在 10-3-1 小节中安装的 Firebug 和 Chrome 自带的“开发者工具”就可以帮上许多忙。例如，网站 http://www.eastmoney.com/就是一个综合的财经证券门户网站，如果我们想要利用程序打开此网站并自动选取其中一个频道，就必须找到该频道所对应的按钮。要找到按钮在网页中的位置，只要在打开网页之后启动“开发者工具”即可，如图 10-31 所示。

图 10-31　在 Chrome 中使用“开发者工具”功能

此时下方就会显示当前对应的网页源代码文件。假设此时我们想知道网页中“查行情”按钮的网页源代码，可以先选择“开发者工具”窗口左上角的“Inspect”功能按钮，然后使用鼠标移到“查行情”按钮上面就行了。此时在页面的下方就会出现此按钮所使用的 HTML 源代码，观察源代码的内容，找出此按钮在网页中独有的地方（一般会先寻找 id 变量，通常是独一无二的），再使用 find_element_by 这类函数来锁定，最后加以处理即可，如图 10-32 所示。

图 10-32　使用“开发者工具”的“Inspect”查看网页中按钮元素对应的 HTML 源代码

也就是说，我们想要在打开网页之后进一步操作网页上的元素，例如输入数据、单击链接或选择某些选项等，可以在找到对象之后再针对该对象操作。可以操作的方法（函数）如表 10-3 所示。

表10-3　可以操作的方法

WebDriver 的方法	主要功能
clear()	清除内容，通常用在文字字段
click()	单击，通常使用于按钮、链接或菜单
is_displayed()	检查此元素在网页中是否为可见的
is_enabled()	检查此元素在网页中是否为可用的
is_selected()	检查此元素是否处于被选中的状态
send_keys(value)	对此元素送出一串字符，也可以是特定的按键

以图 10-32 所示的“查行情”按钮为例，我们可以使用 ts_btn 来操作该按钮。要单击该按钮，只要对 id 为 ts_btn 的元素送出 click()函数即可。程序 10-7 示范打开该网站，按照顺序单击两个按钮后，各停留 10 秒的时间，再关闭浏览器。

程序 10-7

```
# _*_ coding: utf-8 *_*
# 程序 10-7 (Python 3 version)

import time
from selenium import webdriver
url = 'http://www.eastmoney.com'

web = webdriver.Chrome(r"D:\MyPython\chromedriver.exe")
web.get(url)
web.find_element_by_id('ts_btn').click()
time.sleep(10)
web.find_element_by_id('ts_btn1').click()
time.sleep(10)
web.close()
```

程序很简单，每次找到按钮之后就模拟单击（其实就是 ts_btn 和 ts_btn2 两个按钮），同时设置在单击按钮之后让程序停止 10 秒，最后以 web.close()关闭浏览器。

10-3-4　登录会员网站的方法

本小节示范一个可以自动登录会员网站的方法。假设你要登录“京东”商城的网站（http://www.jd.com），想要执行程序帮你自动登录该网站，如何实现这样的操作呢？方法很简单，我们可以直接前往“京东”网页，然后使用 Firebug 进行观察，如图 10-33 所示。

图 10-33　要登录会员的范例网站

如图 10-33 所示，由于要登录账号需要先单击网页中的“你好，请登录”按钮，因此我们可以启用 Firebug，然后在登录按钮上右击。但是由于许多网站都有右键锁，因此遇到在网页上无法右击时，只要安装 RighToClick 附加组件就可以了，如图 10-34 所示。

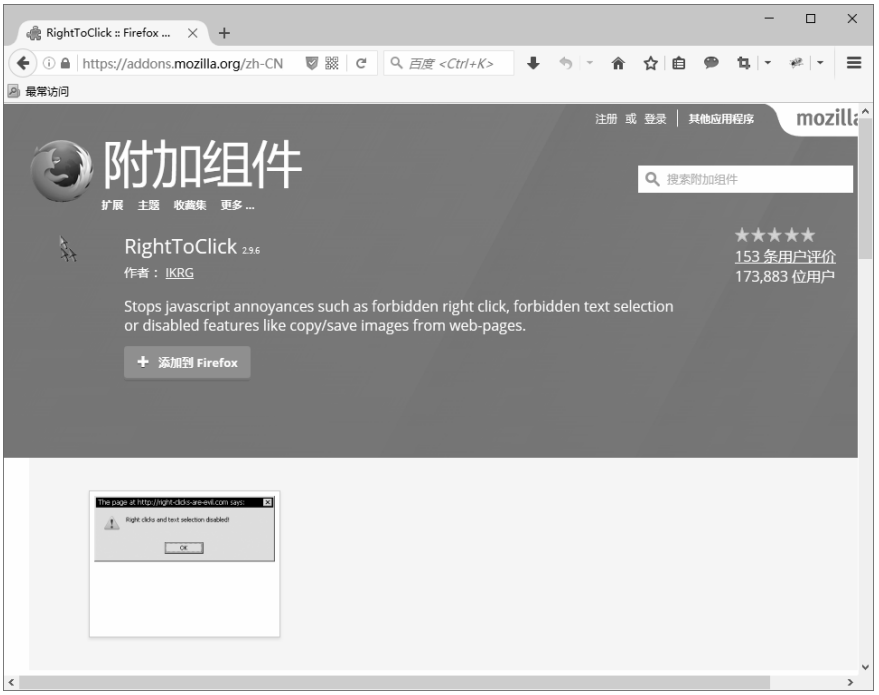


图 10-34　Firefox 解除右键锁的附加组件

然后可以观察到“你好，请登录”按钮的源代码（见图 10-35），以及接下来真正要登录网站时与账号有关的源代码（见图 10-36）。

图 10-35　登录按钮的网页源代码

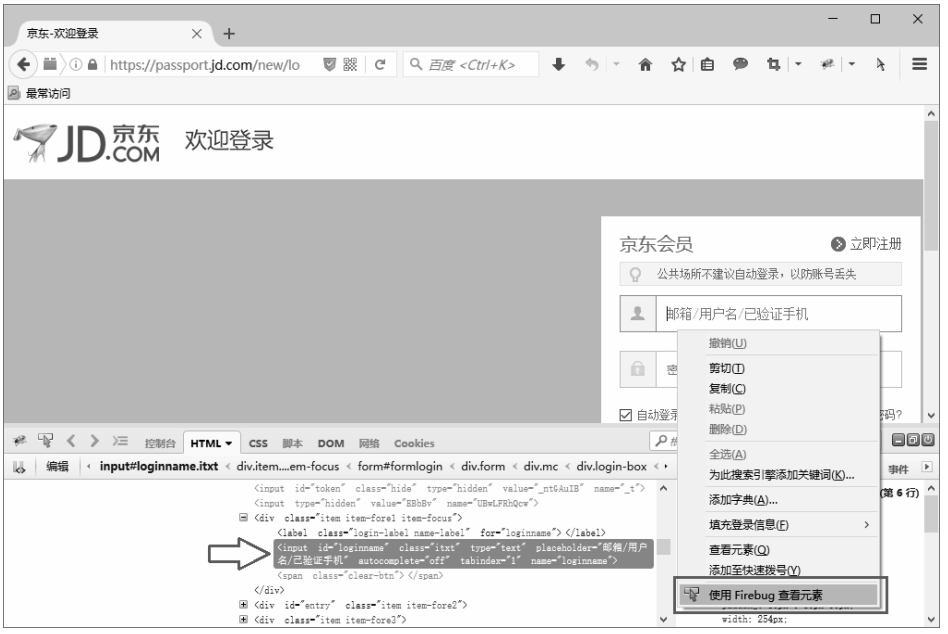


图 10-36　账号字段的网页源代码

按照同样的方法可以找出账号、密码和登录按钮的源代码，然后按照这些信息编写用于自动登录时输入账号和密码的程序，如程序 10-8 所示。

程序 10-8

```
# _*_ coding: utf-8 *_*
# 程序 10-8 (Python 3 version)
```

```
from selenium import webdriver

url = 'http://www.jd.com'

web = webdriver.Firefox()
web.get(url)
web.find_element_by_id('ttbar-login').click()
web.find_element_by_name('loginname').clear()
web.find_element_by_name('loginname').send_keys('your account')
web.find_element_by_name('nloginpwd').clear()
web.find_element_by_name('nloginpwd').send_keys('your password')
web.find_element_by_id('loginsubmit').click()
```

当然，你也可以直接前往“京东”的登录页面（https://passport.jd.com/new/login.aspx?ReturnUrl=http%3A%2F%2Fwww.jd.com%2F），这样可以省去 ttbar-login 的 click()操作，但是这个登录界面的网址太长了，因而还是从首页开始比较简单。当然，我们同样可以用 Chrome 浏览器实现同样的功能，只要用下面的语句替换上面的 web = webdriver.Firefox()语句就行：

```
web = webdriver.Chrome(r"D:\MyPython\chromedriver.exe")
```

10-4　习　　题

1. 请前往 db4free.net（或使用你现有的虚拟主机所提供的 MySQL）创建一个数据库。

2. 请使用 MySQL 服务器的连接功能改写程序 10-1。

3. 请设置一个程序，可以针对网站 http://www.eastmoney.com/打开某一个频道（如单击“查行情”按钮），并在你的系统中设置为每日早上 7 点自动打开。

4. 某些网站一进入就会有分级的按钮，单击同意或已满 18 岁才能够进入浏览，请问此类网站如何利用程序登录？

5. 请练习编写一个程序可以登录你的 Hotmail 账号。