

第 3 章



ChatGPT 与 Python 数据采集

数据采集与办公自动化有着密切的关系,两者的关系如下:

(1) 数据源获取:办公自动化一般需要从不同的数据源中获取数据,如数据库、API、文件等。数据采集负责从这些数据源中收集数据,以供后续的自动化处理和分析使用。

(2) 实时数据更新:办公自动化可能需要基于实时数据进行操作和决策。数据采集可以定期或实时地从数据源中获取最新的数据,确保自动化过程始终基于最新的数据进行操作。

(3) 数据预处理:数据采集包括对采集到的数据进行预处理和清洗的过程。办公自动化可能需要处理缺失值、去除异常值和标准化数据等。数据采集可以在采集数据的同时进行必要的的数据预处理,以提供高质量的数据供办公自动化使用。

(4) 数据整合和转换:在办公自动化过程中,可能需要整合来自不同数据源的数据,或将数据转换为所需的格式。数据采集负责从不同数据源中收集数据,并进行整合和转换,以满足办公自动化的需求。

(5) 决策支持和报告生成:办公自动化一般需要基于数据进行决策支持和报告生成。数据采集可以通过收集和整理相关数据,为自动化过程提供必要的的数据基础,从而帮助生成决策支持材料和报告。

综上所述,数据采集在办公自动化中起着重要的作用,它为自动化过程提供了数据的基础和支持。通过数据采集,办公自动化可以更准确、高效地处理和分析数据,帮助提高工作效率和决策质量。

3.1 ChatGPT 在数据采集过程中的作用

ChatGPT 在数据采集过程中的作用如下:

(1) 解析和处理数据采集的结果:ChatGPT 可以用于解析和处理数据采集的结果。用户可以与 ChatGPT 交互,让它帮助解释和处理数据采集得到的结果,例如提取特定字段、过滤数据、转换格式等。

(2) 提供数据采集过程中的帮助和指导:ChatGPT 可以作为一个智能助手,在数据采

集过程中提供帮助、建议和指导。用户可以向 ChatGPT 提出问题，寻求关于数据采集的最佳实践、技巧或解决方案。

然而，实际上，ChatGPT 作为一个语言模型，它无法直接进行网络请求、数据抓取或处理。为了实现自动化的数据采集和处理，需要结合 ChatGPT 与其他适当的 Python 库和工具，例如 Web Scraping 库、API(应用程序编程接口)调用库、数据库访问库等。

这种结合可以帮助实现自动化的数据采集过程，通过编写 Python 代码和与 ChatGPT 的集成，实现数据的获取、处理和存储。

总结来说，ChatGPT 在数据采集过程中可以作为辅助工具和智能助手，但其本身并不直接处理和执行数据采集任务。它可以提供对话交互和文本生成的能力，帮助用户在数据采集过程中做出决策、提供反馈和解决方案。具体的自动化数据采集任务的实现还需要结合其他的数据采集工具和技术。

3.2 数据采集概述

数据采集是指通过各种技术手段和方法，从不同的数据源中获取数据的过程。数据采集在许多领域中都非常重要，包括科学研究、市场调查和业务决策等。随着互联网的普及和大数据时代的到来，数据采集变得更加广泛和重要。

3.2.1 数据采集的重要性和挑战

数据采集的重要性在于它提供了信息和洞察力，可以用于支持决策、分析和创新。通过采集各种类型的数据，我们可以发现模式、趋势和关联，从而得出有用的结论。数据采集还可以帮助建立和改进模型、算法和预测，从而提高工作效率和结果准确性。

然而，数据采集也面临以下挑战。

(1) 数据来源的多样性：数据可以来自各种不同的来源，如网页、数据库、传感器等。不同的数据源可能使用不同的格式和协议，需要采集和整合这些数据变得更加复杂。

(2) 数据量的庞大：随着数据产生速度的加快和存储成本的降低，我们能够处理的数据量越来越大。采集和处理大规模数据集需要具备高效的算法和技术。

(3) 数据质量的保证：采集到的数据可能存在噪声、缺失或错误。确保数据的质量对于后续的分析 and 应用非常重要，需要进行数据清洗和验证的工作。

3.2.2 数据采集的基本步骤

数据采集的基本步骤可以概括为以下几个阶段：

(1) 定义采集目标：明确你想要采集的数据类型、范围和目标。这有助于确定采集的重点和方向。

(2) 确定数据源：确定你需要采集的数据来源，如网页、API、数据库等。了解数据源的结构、格式和访问方式。

(3) 设计采集方案：根据数据源的特点和要求，设计采集方案。这包括选择合适的采集工具、编写采集代码和设置采集参数等。

(4) 执行数据采集：根据采集方案，执行数据采集操作。这可能涉及网络爬虫、API 调用、数据库查询等技术手段。

(5) 数据清洗和整合：采集到的数据可能需要进行清洗和整合，以确保数据的质量和一致性。这包括处理缺失值、去除噪声、统一格式等操作。

(6) 存储和管理数据：将采集到的数据进行存储和管理，以便后续的分析应用。可以使用数据库、数据湖、云存储等工具和技术来管理数据。

3.2.3 数据采集技术和工具

在数据采集过程中，可以使用一些常见的技术和工具。

(1) 网络爬虫：通过自动化程序从网页中提取数据。

(2) API 调用：使用 API 与远程服务器进行通信，获取数据。

(3) 数据库查询：通过执行 SQL 查询语句从数据库中检索数据。

(4) 传感器和设备：使用传感器和设备采集实时数据，如温度、湿度、位置等。

(5) 数据处理框架和库：使用 Python 中的数据处理框架和库（如 Pandas、NumPy、BeautifulSoup 等）处理和分析采集到的数据。

这些技术和工具可以根据具体的需求和情况进行选择和应用。

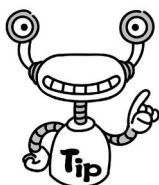
3.3 网页数据爬取

从网页中爬取数据是数据采集的一种常见形式。在网页爬取数据时，可以使用 Python 编程语言结合相应的库和工具来自动化地从网页中提取数据。下面介绍一些常用的网页爬取数据技术和方法。

3.3.1 网页中的静态和动态数据

裹挟在 HTML 代码中的数据并非唾手可得。大多数情况下，Web 前端与后台服务器进行通信时采用同步请求，即一次请求返回整个页面所有的 HTML 代码，这些裹挟在 HTML 中的数据就是所谓的“静态数据”。为了改善用户体验，Web 前端与后台服务器通信也可以采用异步请求技术 AJAX^①，异步请求返回的数据就是所谓的“动态数据”，异步请求返回的数据一般是 JSON 或 XML 等结构化数据，Web 前端获得这些数据后，再通过 JavaScript 脚本程序动态地添加到 HTML 标签中。

^① AJAX(Asynchronous JavaScript and XML)可以异步发送请求获取数据，请求过程中不用刷新页面，用户体验好，而且异步请求过程中，不是返回整个页面的 HTML 代码，只是返回少量的数据，这可以减少网络资源的占用，提高通信效率。



同步请求也可以有动态数据。就是一次请求返回所有 HTML 代码和数据,数据并不是 HTML 放到标签中的,而是被隐藏起来,例如放到 `hide` 等隐藏字段中,或放到 JavaScript 脚本程序的变量中。然后通过 JavaScript 脚本程序动态地添加到 HTML 标签中。

图 3-1 所示的搜狐证券网页显示了某只股票的历史数据,其中图 3-1(a)所示的 HTML 内容都是静态数据,而动态数据则由 JavaScript 脚本程序动态地添加到 HTML 标签中,如图 3-1(b)所示。



图 3-1 网页中的历史数据

3.3.2 使用 urllib 爬取静态数据

urllib 是 Python 标准库中的一个模块,提供了用于进行 HTTP 请求的基本功能。它包含了多个子模块,用于不同的请求任务和操作。

下面是 urllib 库中主要的子模块及其功能:

(1) urllib.request: 用于发送 HTTP 请求和获取响应。它提供了一些函数,如 `urlopen()` 用于打开 URL 并返回响应对象, `urlretrieve()` 用于下载文件等。

(2) urllib.parse: 用于解析 URL、拼接 URL 和处理 URL 编码。它包含了一些函数,如 `urlparse()` 用于解析 URL 字符串, `urljoin()` 用于拼接 URL, `urlencode()` 用于将参数编码为 URL 查询字符串等。

(3) urllib.error: 定义了与 URL 请求相关的异常类。当在请求过程中发生错误时,可以捕获这些异常来进行适当的处理。

(4) urllib.robotparser: 用于解析和分析 robots.txt 文件,该文件用于指示爬虫哪些页

面可以访问。

使用 `urllib` 库,你可以发送 HTTP 请求(GET、POST 等)、设置请求头、处理响应数据和错误等。它是 Python 标准库的一部分,因此不需要安装额外的依赖库。

然而,需要注意的是,`urllib` 库相对较低级,对于一些高级的功能,如处理异步请求、处理复杂的请求参数和响应处理,可能需要借助其他库,如 `requests` 或 `aiohttp`。

下面是一个简单的示例,演示了使用 `urllib.request` 发送 GET 请求并获取响应的过程:

```
import urllib.request

url = 'https://example.com'

# 发送 GET 请求并获取响应
response = urllib.request.urlopen(url)

# 读取响应内容
data = response.read()

# 关闭响应
response.close()

# 处理数据
# ...
```

3.3.3 示例 1: 爬取纳斯达克苹果股票数据

下面通过一个案例介绍如何使用 `urllib` 爬取静态网页数据,如图 3-2 所示是纳斯达克苹果公司股票历史数据网页。

示例实现代码如下:

```
# 代码文件:chapter3/3.2.2.py
import os
import urllib.request

# url = 'https://www.nasdaq.com/symbol/aapl/historical#.UWdnJBDMhHk' ①
# 本地文件访问
url = "file:/// " + os.path.abspath("./nasdaq-Apple1.html").replace("\\", "/") ②

req = urllib.request.Request(url)
with urllib.request.urlopen(req) as response:
    data = response.read()
    html_data = data.decode()
    print(html_data)
```

示例运行后,输出结果如下:

```
<!doctype html >
<html lang = "en">
< head>
```

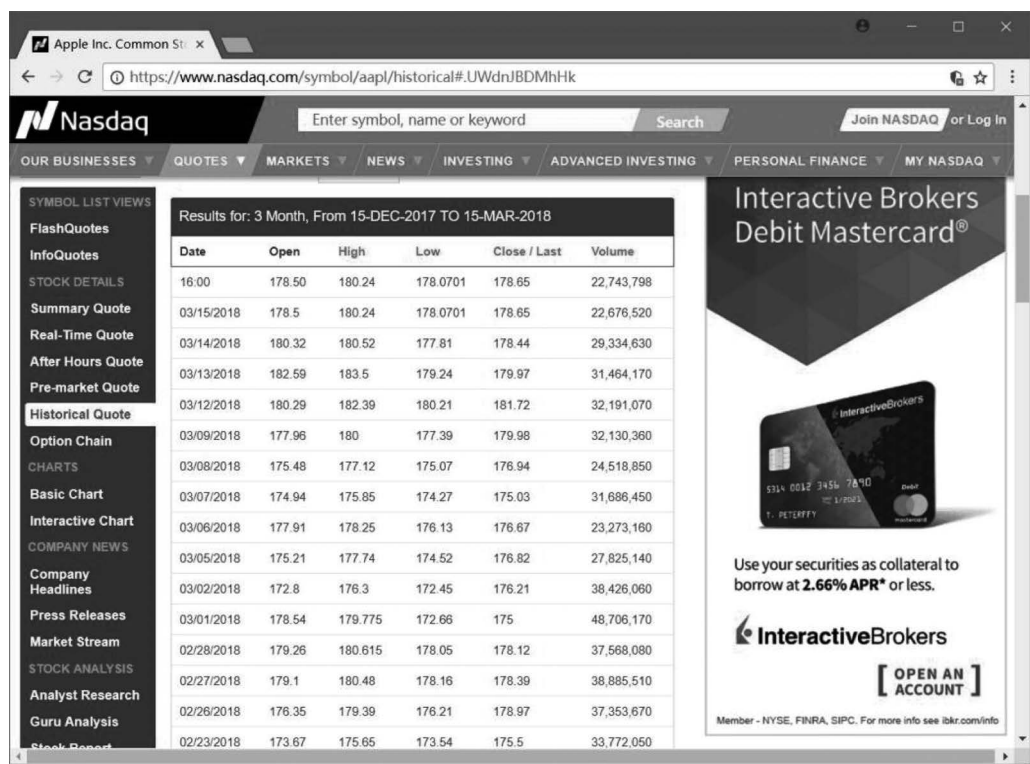


图 3-2 纳斯达克苹果公司股票历史数据网页

```
<meta charset = "UTF - 8">
<meta name = "Generator" content = "EditPlus">
<meta name = "Author" content = "">
<meta name = "Keywords" content = "">
<meta name = "Description" content = "">
<title> Document </title>
</head>
<body>
<div id = "quotes_content_left_pnlAJAX">
  <table class = "historical - data__table">
    <thead class = "historical - data__table - headings">
      <tr class = "historical - data__row historical - data__row -- headings">
        <th class = "historical - data__table - heading" scope = "col"> Date </th>
        <th class = "historical - data__table - heading" scope = "col"> Open </th>
        <th class = "historical - data__table - heading" scope = "col"> High </th>
        <th class = "historical - data__table - heading" scope = "col"> Low </th>
        <th class = "historical - data__table - heading" scope = "col"> Close/Last </th>
        <th class = "historical - data__table - heading" scope = "col"> Volume </th>
      </tr>
    </thead>
    <tbody class = "historical - data__table - body">
```

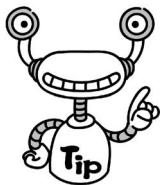
```

<tr class = "historical - data__row">
    <th> 10/04/2019 </th>
    <td> 225.64 </td>
    <td> 227.49 </td>
    <td> 223.89 </td>
    <td> 227.01 </td>
    <td> 34,755,550 </td>
</tr>
<tr class = "historical - data__row">
    <th> 10/03/2019 </th>
    <td> 218.43 </td>
    <td> 220.96 </td>
    <td> 215.132 </td>
    <td> 220.82 </td>
    <td> 30,352,690 </td>
</tr>
...
</tbody>
</table>
</div>
</body>
</html>

```

代码解释如下：

- 代码第①行指定 URL 网址。
- 代码第②行中指定本地文件地址 nasdaq-Apple1.html。



为什么要采用本地文件呢？这是因为我们爬取的网址经常容易改版，出于学习方便，笔者提供了本地文件，读者需要注意根据自己的实际情况将代码第②行改成自己的文件地址。

3.4 解析数据

数据爬取回来后，需要从 HTML 代码中分析出需要的数据，这个过程可以使用适当的数据解析技术实现，例如使用正则表达式、BeautifulSoup 和 XPath 等进行 HTML 或 XML 解析，或使用 JSON 解析库处理 JSON 数据。笔者推荐使用 BeautifulSoup 库。当然也可以利用 ChatGPT 辅助解析数据。

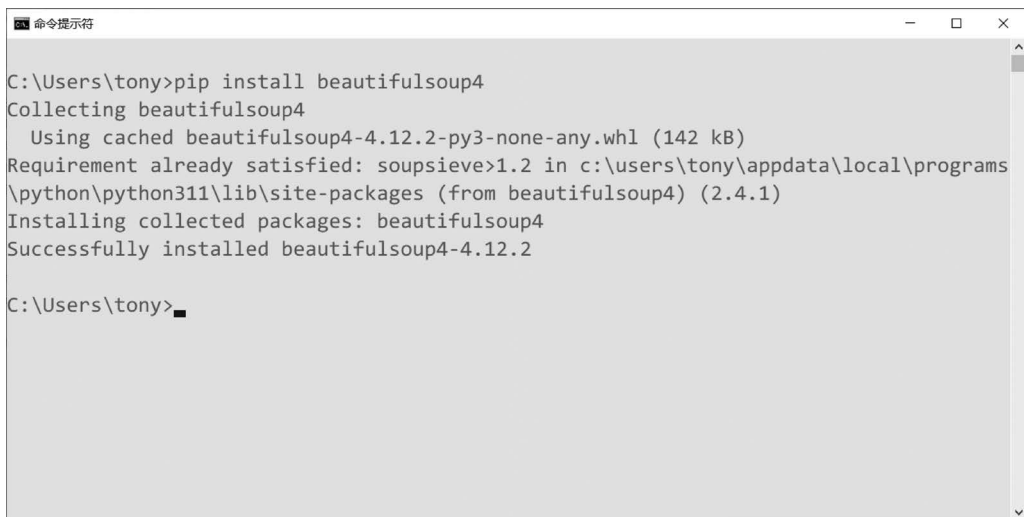
3.4.1 使用 BeautifulSoup 库

BeautifulSoup 是一套帮助程序设计师解析网页结构项目,BeautifulSoup 官网是 <https://www.crummy.com/software/BeautifulSoup/>。

要使用 BeautifulSoup 库首先需要安装 BeautifulSoup,可以通过如下 pip 指令进行安装:

```
pip install beautifulsoup4
```

安装过程如图 3-3 所示。



```
C:\Users\tony>pip install beautifulsoup4
Collecting beautifulsoup4
  Using cached beautifulsoup4-4.12.2-py3-none-any.whl (142 kB)
Requirement already satisfied: soupsieve>1.2 in c:\users\tony\appdata\local\programs\python\python311\lib\site-packages (from beautifulsoup4) (2.4.1)
Installing collected packages: beautifulsoup4
Successfully installed beautifulsoup4-4.12.2

C:\Users\tony>
```

图 3-3 BeautifulSoup 安装过程

下面我们介绍 BeautifulSoup 常用 API。

BeautifulSoup 中主要使用的对象是 BeautifulSoup 实例,BeautifulSoup 常用函数如下:

- find_all(tagname): 根据标签名返回所有符合条件的元素列表。
- find(tagname): 根据标签名返回符合条件的第一个元素。
- select(selector): 通过 CSS 中选择器查找符合条件所有元素。
- get(key, default=None): 获取标签属性值,key 是标签属性名。

BeautifulSoup 常用属性如下:

- title: 获得当前 HTML 页面的 title 属性值。
- text: 返回标签中的文本内容。

在网络爬虫抓取 HTML 代码时,开发人员需要知道数据裹挟在哪些 HTML 标签中,要想找到这些数据,可以使用一些浏览器中的 Web 开发工具。笔者推荐使用 Chrome 或 Firefox 浏览器,因为它们都自带了 Web 开发工具箱。Chrome 浏览器可以通过菜单“更多工具”→“开发者工具”打开,如图 3-4 所示。Firefox 浏览器可以通过菜单“Web 开发者”→

“切换工具箱”打开,如图 3-5 所示。还可通过快捷键打开它们,在 Windows 平台下两个浏览器打开 Web 工具箱的快捷键都是 F12。

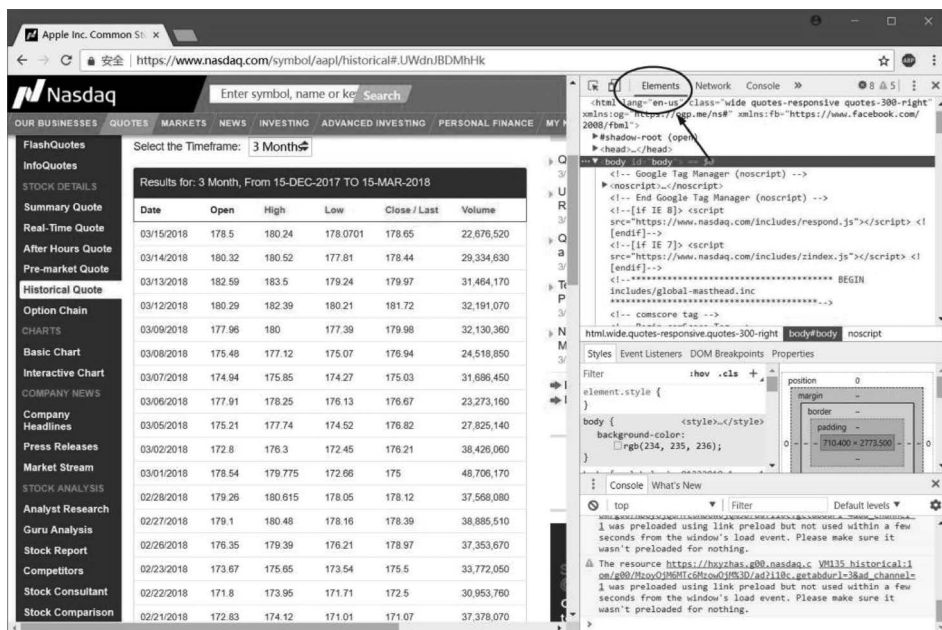


图 3-4 Chrome 浏览器 Web 开发工具箱

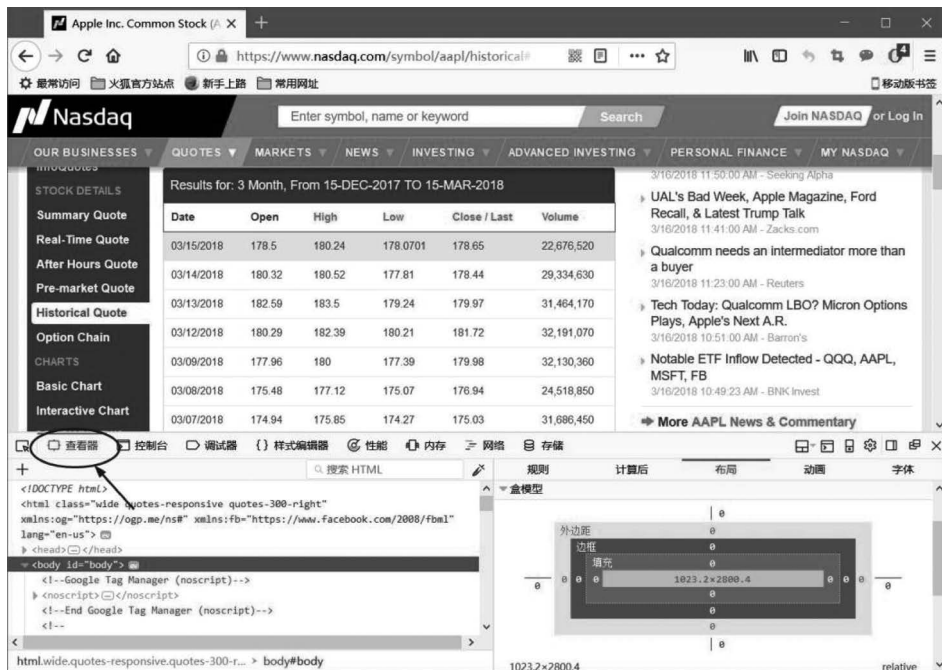


图 3-5 Firefox 浏览器 Web 开发工具箱

3.4.2 示例 2：解析纳斯达克苹果股票数据

下面通过解析纳斯达克苹果公司股票数据熟悉一下如何使用 BeautifulSoup 库解析 HTML 数据。

在编写代码之前先分析一下纳斯达克股票网页数据，首先需要在浏览器中打开网页，单击 F12 键打开 Web 工具箱，如图 3-6 所示。在打开的 Web 工具箱中单击“查看器”标签，查看 HTML 代码，从中可见所需的数据是包裹在<table>元素的<tbody>中的，且每行数据都放到一个<tr>元素中。

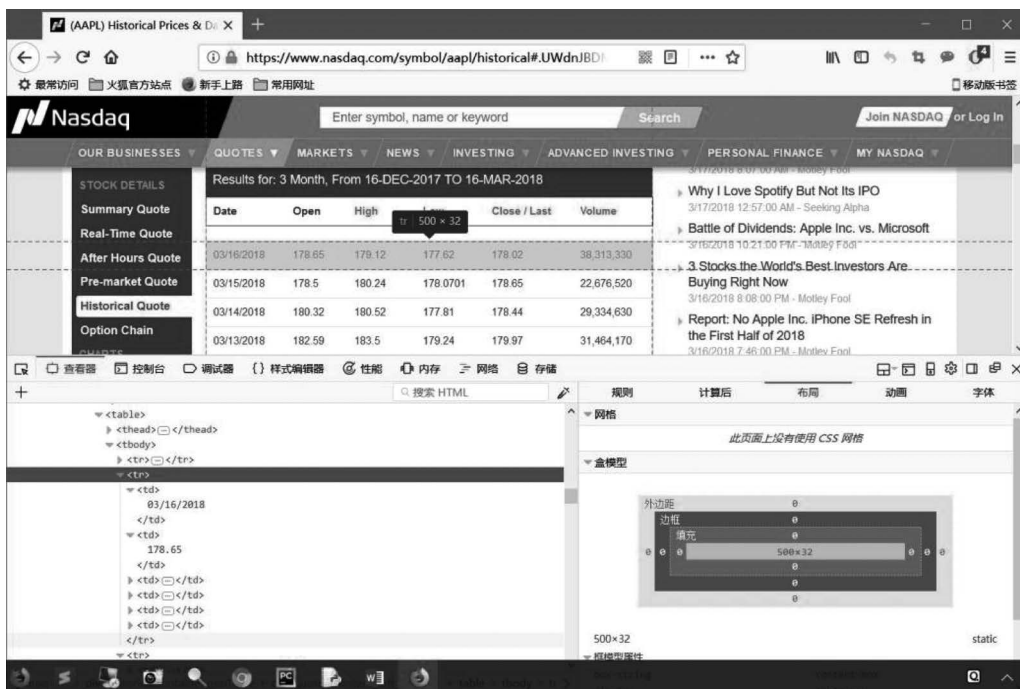


图 3-6 浏览器 Web 工具箱

示例实现代码如下：

```
import urllib.request
from bs4 import BeautifulSoup
# url = 'https://www.nasdaq.com/symbol/aapl/historical#.UWdnJBDmHhK'
# 本地文件访问
url = "file:/// " + os.path.abspath("./nasdaq - Apple1.html").replace("\\", "/")

req = urllib.request.Request(url)

with urllib.request.urlopen(req) as response:
    data = response.read()
    html_data = data.decode()
```

```

sp = BeautifulSoup(html_data, 'html.parser') ①

# 返回<tbody>标签元素
tbody = sp.find('tbody') ②
# 返回<tbody>标签下所有的<tr>元素
trlist = tbody.select('tr') ③
# 保存股票数据列表
data = []
for tr in trlist: ④
    fields = {} # 保存一行数据
    # 获得交易日期<th>元素
    th = tr.find('th')
    fields['Date'] = th.text # 日期
    # 获得tr下的所有td元素
    tds = tr.select('td')
    fields['Open'] = tds[0].text # 开盘
    fields['High'] = tds[1].text # 最高
    fields['Low'] = tds[2].text # 最低
    fields['Close'] = tds[3].text # 收盘
    fields['Volume'] = tds[4].text # 成交量
    data.append(fields) ⑤

print("解析完成。", data)

```

示例运行后,输出结果如下:

```

解析完成。[{'Date': '10/04/2022', 'Open': '225.64', 'High': '227.49', 'Low': '223.89', 'Close':
'227.01', 'Volume': '34,755,550'}, {'Date': '10/03/2022', 'Open': '218.43', 'High': '220.96', 'Low':
'215.132', 'Close': '220.82', 'Volume': '30,352,690'}, {'Date': '10/02/2022', 'Open': '223.06',
'High': '223.58', 'Low': '217.93', 'Close': '218.96', 'Volume': '35,767,260'}, {'Date': '10/01/2022',
'Open': '225.07', 'High': '228.22', 'Low': '224.2', 'Close': '224.59', 'Volume': '36,187,160'},
{'Date': '09/30/2022', 'Open': '220.9', 'High': '224.58', 'Low': '220.79', 'Close': '223.97', 'Volume':
'26,318,580'}, {'Date': '09/27/2022', 'Open': '220.54', 'High': '220.96', 'Low': '217.2814', 'Close':
'218.82', 'Volume': '25,361,290'}, {'Date': '09/26/2022', 'Open': '220', 'High': '220.94', 'Low':
'218.83', 'Close': '219.89', 'Volume': '19,088,310'}, {'Date': '09/25/2022', 'Open': '218.55',
'High': '221.5', 'Low': '217.1402', 'Close': '221.03', 'Volume': '22,481,010'}, {'Date': '09/24/
2022', 'Open': '221.03', 'High': '222.49', 'Low': '217.19', 'Close': '217.68', 'Volume': '31,434,
370'}, {'Date': '09/23/2022', 'Open': '218.95', 'High': '219.84', 'Low': '217.65', 'Close':
'218.72', 'Volume': '19,419,650'}, {'Date': '09/20/2022', 'Open': '221.38', 'High': '222.56', 'Low':
'217.473', 'Close': '217.73', 'Volume': '57,977,090'}, {'Date': '09/19/2022', 'Open': '222.01',
'High': '223.76', 'Low': '220.37', 'Close': '220.96', 'Volume': '22,187,880'}, {'Date': '09/18/2022', 'Open':
'221.06', 'High': '222.85', 'Low': '219.44', 'Close': '222.77', 'Volume': '25,643,090'}, {'Date':
'09/17/2022', 'Open': '219.96', 'High': '220.82', 'Low': '219.12', 'Close': '220.7', 'Volume':
'18,386,470'}, {'Date': '09/16/2022', 'Open': '217.73', 'High': '220.13', 'Low': '217.56', 'Close':
'219.9', 'Volume': '21,158,140'}, {'Date': '09/13/2022', 'Open': '220', 'High': '220.79', 'Low':
'217.02', 'Close': '218.75', 'Volume': '39,763,300'}, {'Date': '09/12/2022', 'Open': '224.8', 'High':
'226.42', 'Low': '222.86', 'Close': '223.085', 'Volume': '32,226,670'}, {'Date': '09/11/2022',
'Open': '218.07', 'High': '223.71', 'Low': '217.73', 'Close': '223.59', 'Volume': '44,289,650'}]

```

代码解释如下:

- 代码第①行使用 BeautifulSoup 构造一个解析器对象 sp, 将 HTML 数据作为输入, 并指定解析器为 'html.parser'。
 - 代码第②行使用 sp.find('tbody') 查找 HTML 页面中的第一个<tbody>标签, 并将结果保存在变量 tbody 中。<tbody>标签通常包含表格数据。
 - 代码第③行使用 tbody.select('tr') 查找 tbody 标签下的所有<tr>标签, 并将结果保存在列表 trlist 中。每个<tr>标签表示表格中的一行数据。
 - 代码第④行遍历 trlist 列表, 对于每个<tr>标签, 执行以下操作:
 - 使用 tr.find('th') 查找当前行中的第一个<th>标签, 并将结果保存在变量 th 中。<th>标签通常用于表示表格中的表头或日期等特殊信息。
 - 使用 tr.select('td') 查找当前行中的所有<td>标签, 并将结果保存在列表 tds 中。每个<td>标签表示表格中的一个单元格。
 - 代码第⑤行将包含每行数据的字典 fields 添加到列表 data 中。
- 最后, 代码输出解析完成后的股票数据 data。

3.4.3 使用 Selenium 爬取动态网页数据

使用 urllib 爬取数据时经常被服务器反爬技术拦截。服务器有一些办法识别请求是否来自浏览器。另外, 有的数据需要登录系统后才能获得, 例如邮箱数据, 而且在登录时会有验证码识别, 验证码能够识别出是人工登录系统, 还是计算机程序登录系统。试图破解验证码不是一个好主意, 现在的验证码也不是简单的图像, 有的会有声音等识别方式。

如果是一个真正的浏览器, 那么服务器设置重重“障碍”就不是问题了。Selenium 可以启动本机浏览器, 然后通过程序代码操控它。Selenium 直接操控浏览器, 可以返回任何形式的动态数据。使用 Selenium 操控浏览器的过程中也可以人为干预, 例如在登录时, 如果需要输入验证码, 则由人工输入, 登录成功之后, 再由 Selenium 操控浏览器爬取数据。

1. 安装 Selenium

要使用 Selenium 库首先需要安装 Selenium, 可通过如下 pip 指令进行安装:

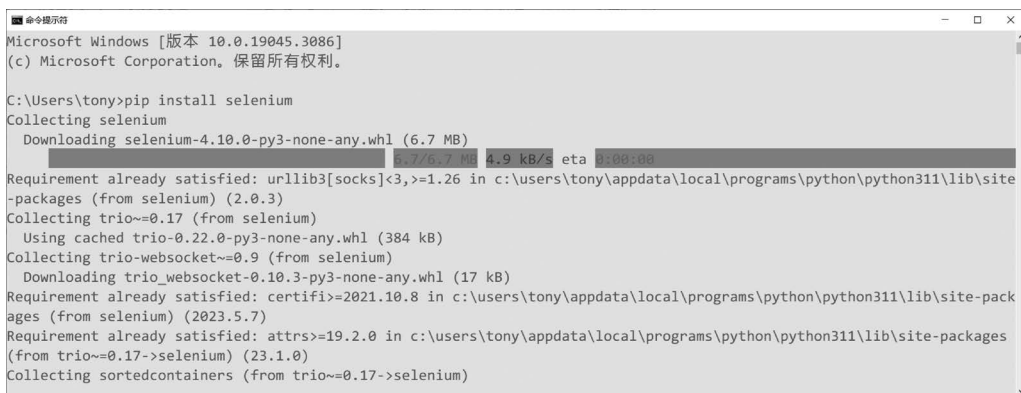
```
pip install selenium
```

安装过程如图 3-7 所示。

2. 配置 Selenium

运行 Selenium 需要操作本地浏览器, 默认是 Firefox, 因此比较推荐安装 Firefox 浏览器, 要求 Firefox 浏览器是 55.0 以上版本。由于版本兼容的问题还需要下载浏览器引擎 GeckoDriver, GeckoDriver 可以在 <https://github.com/mozilla/geckodriver/releases> 下载, 根据自己的平台选择对应的版本, 不需要安装 GeckoDriver, 只需将下载包解压处理就可以了。

然后需要配置环境变量, 将 Firefox 浏览器的安装目录和 GeckoDriver 解压目录添加到系统的 PATH 中, 如图 3-8 所示是在 Windows 10 下添加 PATH。



```

Microsoft Windows [版本 10.0.19045.3086]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\tony>pip install selenium
Collecting selenium
  Downloading selenium-4.10.0-py3-none-any.whl (6.7 MB)
    5.7/6.7 MB 4.9 kB/s eta 0:00:00
Requirement already satisfied: urllib3[socks]<3,>=1.26 in c:\users\tony\appdata\local\programs\python\python311\lib\site-packages (from selenium) (2.0.3)
Collecting trio~=0.17 (from selenium)
  Using cached trio-0.22.0-py3-none-any.whl (384 kB)
Collecting trio-websocket~=0.9 (from selenium)
  Downloading trio_websocket-0.10.3-py3-none-any.whl (17 kB)
Requirement already satisfied: certifi>=2021.10.8 in c:\users\tony\appdata\local\programs\python\python311\lib\site-packages (from selenium) (2023.5.7)
Requirement already satisfied: attrs>=19.2.0 in c:\users\tony\appdata\local\programs\python\python311\lib\site-packages (from trio~=0.17->selenium) (23.1.0)
Collecting sortedcontainers (from trio~=0.17->selenium)

```

图 3-7 Selenium 安装过程

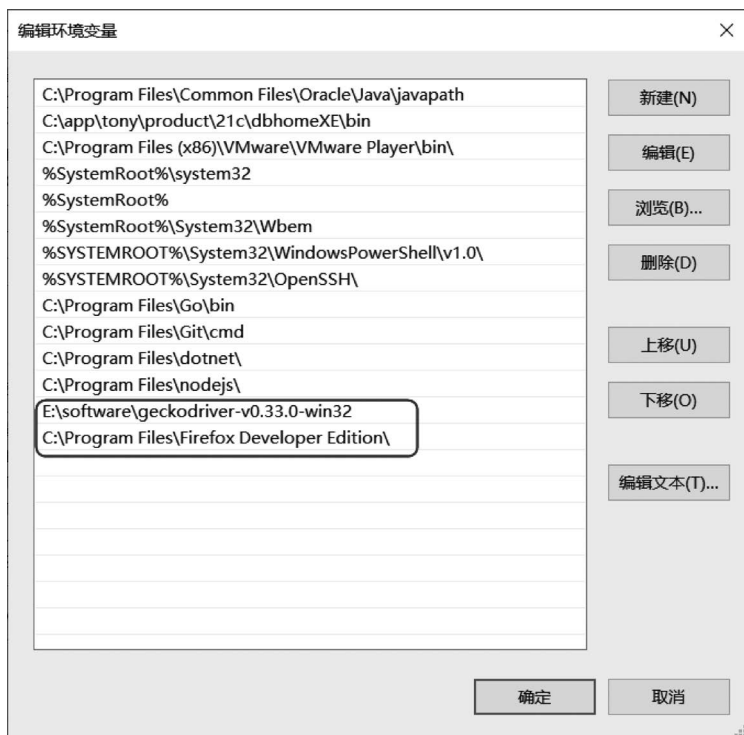


图 3-8 添加 PATH

3. Selenium 常用 API

Selenium 操作浏览器主要通过 WebDriver 对象实现, WebDriver 对象提供了操作浏览器和访问 HTML 代码中数据的函数。

操作浏览器的函数如下:

- refresh(): 刷新网页。

- `back()`：回到上一个页面。
- `forward()`：进入下一个页面。
- `close()`：关闭窗口。
- `quit()`：结束浏览器执行。
- `get(url)`：浏览 URL 所指的网页。

访问 HTML 代码中数据的函数如下：

- `find_element(By.ID, id)`：通过元素的 id 查找符合条件的第一个元素。
- `find_elements(By.ID, id)`：通过元素的 id 查找符合条件的所有元素。
- `find_element(By.NAME, name)`：通过元素名字查找符合条件的第一个元素。
- `find_elements(By.NAME, name)`：通过元素名字查找符合条件的所有元素。
- `find_element(By.LINK_TEXT, link_text)`：通过链接文本查找符合条件的第一个元素。
- `find_elements(By.LINK_TEXT, link_text)`：通过链接文本查找符合条件的所有元素。
- `find_element(By.TAG_NAME, name)`：通过标签名查找符合条件的第一个元素。
- `find_elements(By.TAG_NAME, name)`：通过标签名查找符合条件的所有元素。
- `find_element(By.XPATH, xpath)`：通过 XPath 查找符合条件的第一个元素。
- `find_elements(By.XPATH, xpath)`：通过 XPath 查找符合条件的所有元素。
- `find_element(By.CLASS_NAME, name)`：通过 CSS 中 class 属性查找符合条件的第一个元素。
- `find_elements(By.CLASS_NAME, name)`：通过 CSS 中 class 属性查找符合条件的所有元素。
- `find_element(By.CSS_SELECTOR, css_selector)`：通过 CSS 中选择器查找符合条件的第一个元素。
- `find_elements(By.CSS_SELECTOR, css_selector)`：通过 CSS 中选择器查找符合条件的所有元素。

3.4.4 示例 3：爬取搜狐证券贵州茅台股票数据

下面通过爬取搜狐证券贵州茅台股票数据案例，熟悉一下如何使用 Selenium 库爬取和解析 HTML 数据。

读者如果直接使用 `urllib` 库是无法直接获取 HTML 数据的，原因是这些数据是同步动态数据。而使用 Selenium 返回这些数据是非常简单的。

在爬取数据之前，还是先分析一下搜狐证券贵州茅台股票的 HTML 数据，主流的浏览器都提供了 Web 工具箱，找到显示这些数据的 HTML 标签，如图 3-9 所示，在 Web 工具箱的查看器中，找到显示页面表格对应的 HTML 标签，注意在查看器中选中对应的标签，页面会将该部分灰色显示。经过查找分析最终找到一个 `table` 标签，复制它的 id 或 class 属性

值,以备在代码中进行查询。



图 3-9 Web 工具箱

案例实现代码如下:

```
from selenium import webdriver ①
from selenium.webdriver.common.by import By

driver = webdriver.Firefox() ②
driver.get('http://q.stock.sohu.com/cn/600519/lshq.shtml') ③
table_element = driver.find_element(By.ID, 'BIZ_hq_historySearch') ④
print(table_element.text) ⑤
driver.quit() ⑥
```

代码解释如下:

- 代码第①行导入 Selenium 库中的 webdriver 模块,该模块提供了用于控制不同浏览器的驱动程序。
- 代码第②行创建一个 Firefox 浏览器的 WebDriver 实例,将其赋值给变量 driver。这将启动一个 Firefox 浏览器窗口。
- 代码第③行使用 WebDriver 加载指定的 URL,这里是“http://q.stock.sohu.com/cn/600519/lshq.shtml”,即搜狐股票网站中贵州茅台(股票代码 600519)的历史行情页面。
- 代码第④行使用 WebDriver 的 find_element() 函数通过元素的 ID 查找页面上的一

个特定元素。这里通过 By.ID 参数指定按照元素的 ID 进行查找, ID 值为 'BIZ_hq_historySearch'。

- 代码第⑤行打印找到的元素的文本内容。text 属性返回元素的可见文本。
- 代码第⑥行 driver.quit() 关闭浏览器窗口并终止 WebDriver 的会话。

3.4.5 示例 4：使用 Selenium 解析 HTML 数据

Selenium 库不仅可以模拟人工操作 Web 页面, 我们也可以利用它的一系列 find_element() 函数进行解析 HTML 数据, 使用过程类似于 BeautifulSoup 库, 本节来介绍如何使用 Selenium 库解析搜狐证券贵州茅台股票 HTML 数据。

案例实现代码如下：

```
from selenium import webdriver
from selenium.webdriver.common.by import By

driver = webdriver.Firefox()
driver.get('http://q.stock.sohu.com/cn/600519/lshq.shtml')
table_element = driver.find_element(By.ID, 'BIZ_hq_historySearch') ①
tbody = table_element.find_element(By.TAG_NAME, "tbody") ②
trlist = tbody.find_elements(By.TAG_NAME, 'tr') ③
# 股票数据列表
data = []

for idx, tr in enumerate(trlist): ④
    if idx == 0:
        # 跳过 table 第一行
        continue ⑤

    td_list = tr.find_elements(By.TAG_NAME, "td") ⑥
    fields = {}
    fields['Date'] = td_list[0].text      # 日期
    fields['Open'] = td_list[1].text     # 开盘
    fields['Close'] = td_list[2].text    # 收盘
    fields['Low'] = td_list[5].text      # 最低
    fields['High'] = td_list[6].text     # 最高
    fields['Volume'] = td_list[7].text   # 成交量
    data.append(fields)

print(data)
driver.quit()
```

示例运行后, 输出结果如下：

```
[{'Date': '2023-06-21', 'Open': '1740.00', 'Close': '1735.83', 'Low': '1735.00', 'High': '1756.60', 'Volume': '17721'}, {'Date': '2023-06-20', 'Open': '1740.00', 'Close': '1743.46', 'Low': '1735.00', 'High': '1765.00', 'Volume': '20947'}, {'Date': '2023-06-19', 'Open': '1790.00', 'Close': '1744.00', 'Low': '1738.00', 'High': '1797.95', 'Volume': '31700'}, {'Date': '2023-06-16', 'Open': '1757.00', 'Close': '1797.69', 'Low': '1750.10', 'High': '1800.00', 'Volume': '37918'}, {'Date': '2023-06-15', 'Open': '1730.34', 'Close': '1755.00', 'Low': '1723.00', 'High': '1755.65', 'Volume': '25223'}, {'Date': '2023-06-14', 'Open': '1719.00', 'Close': '1719.00', 'Low': '1719.00', 'High': '1719.00', 'Volume': '1719.00'}]
```

```
...
'1813.74', 'Low': '1783.30', 'High': '1822.01', 'Volume': '23952'}, {'Date': '2023-02-27', 'Open':
'1778.50', 'Close': '1810.41', 'Low': '1775.02', 'High': '1815.00', 'Volume': '22065'}, {'Date':
'2023-02-24', 'Open': '1810.11', 'Close': '1788.00', 'Low': '1782.18', 'High': '1810.19', 'Volume':
'24635'}]]
```

代码解释如下：

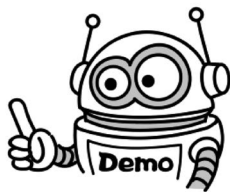
- 代码第①行使用 WebDriver 在页面中查找具有 ID 为 'BIZ_hq_historySearch' 的元素,并将其赋值给变量 table_element。这个元素应该是包含历史行情数据的表格。
- 代码第②行在 table_element 元素中查找名为 "tbody" 的子元素,并将其赋值给变量 tbody。这个操作是为了定位表格中的 tbody 部分,其中包含了行情数据的行。
- 代码第③行在 tbody 元素中查找所有名为 "tr" 的子元素,返回一个包含这些元素的列表。这个操作是为了获取每一行行情数据的 tr 元素。
- 代码第④行使用 enumerate() 函数遍历 trlist 列表中的每个元素,并为每个元素分配一个索引 idx 和一个变量 tr,用于迭代行情数据的每一行。
- 代码第⑤行 if idx == 0: continue; 如果索引 idx 等于 0,也就是第一行表头行,就跳过此次循环,不处理表头行的数据。
- 代码第⑥行在当前行的 tr 元素中查找所有名为 "td" 的子元素,返回一个包含这些元素的列表。这个操作是为了获取当前行中每个列的数据。

在循环中,每行的各列数据被提取出来,以字典的形式存储在 fields 变量中,并添加到 data 列表中(data 列表包含了每行行情数据的字典)。

最后,代码打印出 data 列表,即历史行情数据,关闭浏览器并终止 WebDriver 的会话。

3.4.6 示例 5: 利用 ChatGPT 辅助解析数据

如果读者编程能力不是很强,也可以利用 ChatGPT 解析 HTML,下面具体介绍。



读者向 ChatGPT 发送如下提问：

根据如下的 HTML 代码,帮我从其中 historical-data__table 表格中解析出数据,注意不要返回程序代码,你解析返回 CSV 表格数据。

```
<!doctype html >
<html lang = "en">
<head>
  <meta charset = "UTF - 8">
  <meta name = "Generator" content = "EditPlus®">
  <meta name = "Author" content = "">
```

```

    < meta name = "Keywords" content = "">
    < meta name = "Description" content = "">
    < title> Document </title>
</head>
< body>
< div id = "quotes_content_left_pnlAJAX">
    < table class = "historical - data__table">
        < thead class = "historical - data__table - headings">
            < tr class = "historical - data__row historical - data__row -- headings">
                < th class = "historical - data__table - heading" scope = "col"> Date </th>
                < th class = "historical - data__table - heading" scope = "col"> Open </th>
                < th class = "historical - data__table - heading" scope = "col"> High </th>
                < th class = "historical - data__table - heading" scope = "col"> Low </th>
                < th class = "historical - data__table - heading" scope = "col"> Close </th>
                < th class = "historical - data__table - heading" scope = "col"> Volume </th>
            </tr>
        </thead>
        < tbody class = "historical - data__table - body">
            < tr class = "historical - data__row">
                < th> 10/04/2022 </th>
                < td> 225.64 </td>
                < td> 227.49 </td>
                < td> 223.89 </td>
                < td> 227.01 </td>
                < td> 34,755,550 </td>
            </tr>
            ...
            < tr class = "historical - data__row">
                < th> 09/11/2022 </th>
                < td> 218.07 </td>
                < td> 223.71 </td>
                < td> 217.73 </td>
                < td> 223.59 </td>
                < td> 44,289,650 </td>
            </tr>
        </tbody>
    </table>
</div>
</body>
</html>

```

执行上述的提问 ChatGPT 回答如下。

以下是从给定的 HTML 中解析出的数据,并以 CSV 格式返回的结果:

```

Date,Open,High,Low,Close,Volume
10/04/2022,225.64,227.49,223.89,227.01,34,755,550
10/03/2022,218.43,220.96,215.132,220.82,30,352,690
10/02/2022,223.06,223.58,217.93,218.96,35,767,260
10/01/2022,225.07,228.22,224.2,224.59,36,187,160
09/30/2022,220.9,224.58,220.79,223.97,26,318,580
09/27/2022,220.54,220.96,217.2814,218.82,25,361,290
09/26/2022,220,220.94,218.83,219.89,19,088,310
09/25/2022,218.55,221.5,217.1402,221.03,22,481,010

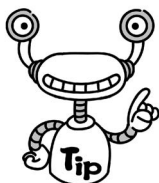
```

```

09/24/2022,221.03,222.49,217.19,217.68,31,434,370
09/23/2022,218.95,219.84,217.65,218.72,19,419,650
09/20/2022,221.38,222.56,217.473,217.73,57,977,090
09/19/2022,222.01,223.76,220.37,220.96,22,187,880
09/18/2022,221.06,222.85,219.44,222.77,25,643,090
09/17/2022,219.96,220.82,219.12,220.7,18,386,470
09/16/2022,217.73,220.13,217.56,219.9,21,158,140
09/13/2022,220,220.79,217.02,218.75,39,763,300
09/12/2022,224.8,226.42,222.86,223.085,32,226,670
09/11/2022,218.07,223.71,217.73,223.59,44,289,650

```

这是从表格中提取的日期、开盘价、最高价、最低价、收盘价和成交量的数据。



CSV(Comma Separated Values)是用逗号分隔数据项(也称为字段)的数据交换格式,CSV 主要应用于电子表格和数据库之间的数据交换。CSV 是文本文件,可以使用记事本等文本编辑工具打开。

3.5 本章总结

本章主要介绍了 ChatGPT 在数据采集中的应用,以及 Python 进行网页爬虫的数据采集方法。

首先,讨论了 ChatGPT 可以在数据采集的多个方面提供帮助,比如提出采集思路,说明采集场景,甚至直接提供采集代码。合理利用 ChatGPT 可以极大地提高数据采集的效率。

随后,介绍了数据采集的基本概念,包括其重要性、步骤、技术和工具等。数据采集是后续数据分析的基础。

接下来,重点介绍了 Python 进行网页爬虫的数据采集。先介绍了网页的静态和动态数据,然后使用 urllib 库爬取了纳斯达克苹果股票数据。为了解析爬取的数据,学习使用了 BeautifulSoup 库的方法。

对于动态网页,介绍了 Selenium 的安装配置和用法,并爬取了搜狐证券的贵州茅台股票数据。还介绍了如何使用 Selenium 解析 HTML 数据。

最后,以利用 ChatGPT 辅助解析数据为例,展示了 ChatGPT 在数据采集中的强大作用。

通过本章的学习,我们掌握了 Python 进行网页爬虫的数据采集的基本方法,为后续数据分析项目奠定了基础。