视频讲解

第3章

爬虫流程与Requests库的使用

学习目标

1. 知识目标

(1) 掌握 Python 实现网络数据采集的流程。

(2) 理解 Requests 库的原理。

(3) 掌握 Requests 库的使用。

2. 能力目标

(1) 能够领会并学会 Python 网络爬虫编写流程。

(2) 能够应用 Requests 库采集静态网页数据。

3. 素质目标

(1) 能够运用 Requests 库来解决数据采集实际问题。

(2) 培养学生主动查阅资料,积极地互帮、互助学习。

知识导读

本章将系统讲解 Python 网络爬虫的核心流程与 Requests 库的实践应用。通过学习,读者可以掌握静态网页数据采集的基本原理,重点围绕 HTTP 请求发送、数据解析及数据存储展开。首先,梳理爬虫的基本工作流程,为后续实践奠定基础。其次,深入讲解 Requests 库的核心功能,涵盖 GET/POST 请求的发送、请求头设置、状态码识别及 JSON/文本/二进制数据的解析方法。最后,通过电影信息采集和翻译信息采集两个案例,演示了从发起请求到数据存储的全流程操作,帮助读者融会贯通。本章内容的思维导图如图 3-1 所示。

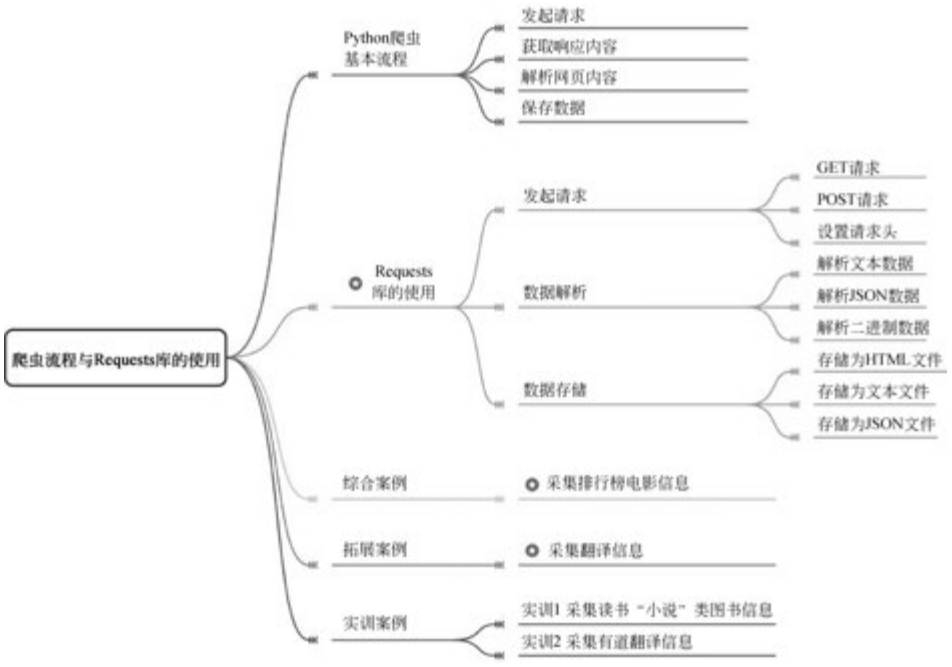

图 3-1　第 3 章思维导图

3.1　Python 爬虫基本流程

静态网页是指其内容在服务器端预先生成,当用户发起请求时,服务器直接将该网页内容发送给用户。此类网页通常包含固定不变的 HTML、CSS 和 JavaScript 代码,不存在动态内容生成或与服务器的交互操作。因此,静态网页的数据采集相对简便,仅需依照 HTTP 向目标站点发送请求,获取整个网页内容后,解析其中的 HTML、CSS 和 JavaScript 代码,即可提取出所需要的数据。

运用 Python 实现网络数据采集的基本流程如图 3-2 所示。

1. 发起请求

依据 HTTP(S)向目标站点发起请求,此过程即发送一个 request 请求,随后等待服务器响应。

在开展数据采集任务前,首先需确定目标网站的种子页面或起始页面。依据目标网站的 robots.txt 文件来明确爬取规则,并结合采集需求编写爬虫程序。通过 Requests 库发送 HTTP 请求,以获取目标网页的内容。在此环节中,可选用 GET 或 POST 方法来发送 HTTP 请求。同时,需要注意目标网站的反爬虫机制,可通过设置 User-Agent、Referer、超时时间等 HTTP 头信息,以此模拟浏览器的行为特征,并且可运用代理 IP 等技术手段来规避因频繁访问导致的 IP 封禁情况。此外,依据实际需求,能够借助循环和递归等编程逻辑实现对多个页面的遍历操作。对于存在分页机制的网站,可依照其分页规则逐页获取数据。

2. 获取响应内容

若服务器能够正常地响应客户端请求,爬虫将接收到一个 Response。Response 的内容即为所需获取的页面内容,其数据类型丰富多样,可能是 HTML 文档、JSON 格式字符串、二进制数据(如图片、视频文件等)。

3. 解析网页内容

从目标网站获取的响应内容,需要对其进行深入的解析与提取操作。可通过 Chrome 浏

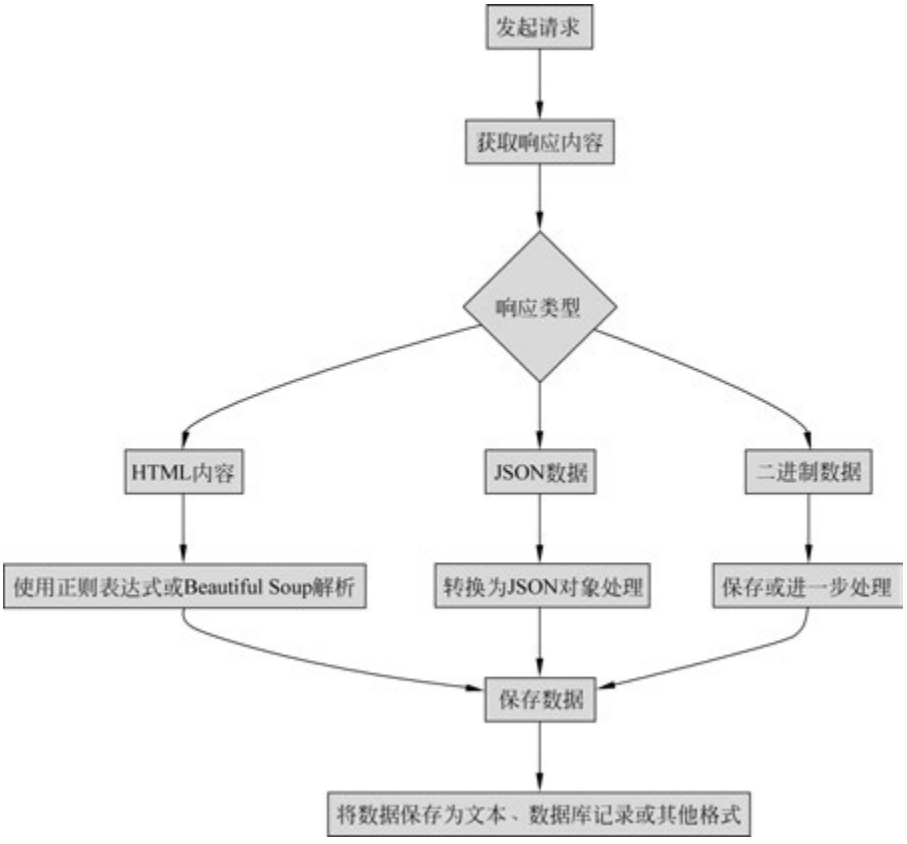

图 3-2　爬虫基本流程

览器的开发者工具来详细查看网页结构，确定所需采集的数据在网页中的具体位置以及所属标签，进而运用相应的解析库对响应内容进行解析。依据网页的结构特点以及所需数据的位置信息，合理选取适宜的数据解析技术来提取数据。

4. 保存数据

在成功提取数据之后，可以对采集到的数据进行清洗与整理操作。然后，将采集到的数据存储至本地文件系统或数据库中，保存形式多样，可以存为文本，也可以保存至数据库，或者保存特定格式的文件。

3.2　Requests 库的使用

Python 标准库内包含一个用于网络数据采集的模块 URLlib，该模块具备一系列处理 URL 的子模块与函数，能够执行发送 HTTP 请求、处理 URL 编码、解析 URL 等操作，可满足较为简单的网络操作需求。不过，在应对更为复杂的网络操作以及高级功能时，URLlib 库的使用过程较为烦琐。

为实现上述操作的便捷性，功能更为强大的 Requests 库应运而生。Requests 库以 Python 语言基于 URLlib 编写而成，遵循 Apache2 Licensed 开源协议，属于 HTTP 库范畴。由于 Requests 库是第三方库而非 Python 内置库，所以在使用前需要安装。

3.2.1　发起请求

Requests 库提供了简洁且人性化的 API，极大地简化了发送 HTTP 请求的流程，使其变

得容易操作。它是 Python 中一款原生的基于网络请求的 HTTP 库,具备强大的功能特性,使用方式简单高效,能够模拟浏览器行为向目标网站发起请求操作,从而为网络数据采集提供了有力的支持与保障,在网络爬虫开发等相关领域得到了广泛应用。

1. Requests 发起请求的类型

Requests 可以发起 GET 和 POST 两种请求类型,其语法格式如下:

```
requests.method(URL, ** kwargs)
```

各参数的含义如表 3-1 所示。

表 3-1 Requests 发起请求各参数含义

参 数	说 明
method	用于接收 string。表示请求的类型,如 GET、HEAD、POST 等。无默认值
URL	用于接收 string。表示字符串形式的网址。无默认值
** kwargs	用于接收 dict 或其他 Python 中的类型的数据。依据具体需要及请求的类型可添加的参数,通常参数赋值为字典类型或为具体数据

【例 3-1】 下面是一个使用 requests.get()方法向目标网站广州商学院官网(网址详见前言二维码)发起 GET 请求的源代码。

```
1.  import requests                          # 导入请求库
2.  URL = "https://www.gcc.edu.cn/"          # 指定目标网站地址
3.  response = requests.get(URL)             # 发起 GET 请求
4.  print(response.text)                     # 输出响应内容
```

在例 3-1 代码中,首先导入了 requests 库。接着指定要访问的目标网站地址。然后,使用 requests.get()方法发起 GET 请求,并将响应内容保存在名为 response 的变量中。最后,将从目标网站获取的响应内容输出到控制台。

说明:当返回的响应内容是文本内容时,则使用 text 属性返回 Unicode 型的数据,而如果返回的响应内容是图片或视频,则使用 content 属性返回 bytes 型(即二进制)数据。

【例 3-2】 下面是一个使用 requests.post()方法向目标网站百度翻译官网(网址详见前言二维码)发起 POST 请求的源代码。

```
1.  import requests                          # 导入请求库
2.  URL = "https://fanyi.baidu.com/"         # 指定目标网站地址
3.  res = requests.post(URL)                 # 发起 POST 请求
4.  print(res.text)                          # 输出响应内容
```

2. 设置请求头与响应时间

在网络数据采集中,目标网站通过检查 HTTP 请求头(Request Headers)来识别并限制自动化程序的访问,这是一种常见的反爬虫策略。因此,在使用 Requests 库发起 HTTP 请求时,通常需要显式配置 headers 参数。通过设置诸如 User-Agent(模拟特定的浏览器或客户端标识)和 Referer(指示请求来源页面)等请求头字段,可以使请求更接近真实用户的浏览器行为。

为防止程序因等待服务器响应时间过长而陷入阻塞状态,为网络请求设定超时限制至关重要。Requests 库提供了 timeout 参数来实现此功能。该参数需传递给请求方法,并指定一个以秒(s)为单位的数值。若服务器在该时限内未能完成响应,请求方法将抛出 Timeout 异常(requests.exceptions.Timeout),从而允许程序进行错误处理,避免无限期等待。

【例 3-3】 下面是一个发起 GET 请求并携带请求头和响应时间的源代码。

```
import requests                                    #导入请求库
URL = "https://www.gcc.edu.cn/"                    #指定目标网站地址
headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
    like Gecko) Chrome/127.0.0.0 Safari/537.36',
    'Accept-Language': 'en-US,en;q=0.9',
    'Connection': 'keep-alive'
}                                                  #设置请求头信息

response = requests.get(URL, headers=headers, timeout=10) #携带请求头和响应时间
print(response.text)
```

在例 3-3 代码中，定义了一个字典 headers，在该字典中设置一些常见的请求头字段信息。然后使用 requests.get()方法发起 GET 请求，并通过 headers 参数传递请求头。其中，User-Agent 字段是模拟浏览器的标识，它告诉目标网站服务器客户端使用 Mozilla 浏览器进行访问；Accept-Language 字段表示客户端接受的语言类型；Connection 字段表示客户端希望保持连接活动。程序响应时间 timeout 设置为 10s。

在网络爬虫采集数据过程中，用户可以根据实际需要自定义请求头，包括其他字段或值。具体的请求头会因不同的网站和应用程序而有所不同。如果想要查看当前浏览器的请求头信息，则可以打开浏览器的开发者工具，在 Network 选项卡中找到请求的详细信息，包含请求头字段与值。

3. 查看状态码与编码

使用 status_code 属性可以查看响应状态，如果返回的是 200，则表示目标服务器成功响应请求。使用 encoding 属性可以查看响应内容的网页编码。

当 Requests 库解析网页编码出错时，需要用户手动设置 encoding 编码，避免返回的网页内容解析出现乱码。手动指定的方法并不灵活，无法自适应对应爬取过程中不同网页的编码，而使用 chardet 库比较简便灵活，chardet 库是一个非常优秀的字符串/文件编码检测模块，首次使用需要安装该库。

使用 chardet 库的 detect()方法检测给定字符串的编码的语法格式及参数说明如下：

```
chardet.detect(byte_str) #byte_str 接收 string，表示需要检测编码的字符串
```

【例 3-4】 使用 Requests 库发起 GET 请求，携带请求头和设置响应时间，并设置响应编码和查看状态码。

```
import requests                                                    #导入请求库
import chardet                                                     #导入编码检测库
URL = "https://www.gcc.edu.cn/"                                    #指定目标网站地址
headers = {
'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/127.0.0.0 Safari/537.36',
'Accept-Language': 'en-US,en;q=0.9',
'Connection': 'keep-alive'
}                                                                  #设置请求头信息
response = requests.get(URL, headers=headers, timeout=10) #携带请求头和响应时间
print('响应状态码: ', response.status_code)                          #输出状态码
```

```
    encode = chardet.detect(response.content) #检测响应内容的编码,参数为字节型的字符串
    response.encoding = encode['encoding']    #设置响应内容的编码
    print('网页编码: ', response.encoding)     #输出响应内容的编码
    print(response.text)                      #输出响应内容
```

3.2.2 数据解析

数据解析是指从获取的网页源代码中提取所需的信息。Requests 库自身提供不同方法来处理所获取的响应数据。当然若要准确地定位和提取信息,可以使用像正则表达式、XPath 和 Beautiful Soup 这些常见的数据解析方法。

在使用 Requests 库向目标网站发起请求并得到成功响应,则可以对返回的响应数据进行解析和输出。Requests 库提供了多种方法来处理不同类型的响应数据,如文本、JSON 格式、二进制等。

【例 3-5】 解析文本数据。

```
import requests                                          #导入请求库
response = requests.get('https://www.gcc.edu.cn/')       #发起请求
response.encoding = 'utf-8'                              #设置编码
text_data = response.text                                #使用 text 属性获取响应内容
print(text_data)                                         #输出响应内容
```

在例 3-5 代码中,使用 response.text 属性获取响应的文本数据。

【例 3-6】 解析 JSON 数据。

```
import requests                                                    #导入请求库
import json                                                        #导入 JSON 库
URL = 'https://jiaotong.baidu.com/trafficindex/city/list'          #设置目标网站
headers = {'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0
Safari/537.36'}                                                    #设置请求头
response = requests.get(URL, headers=headers)                      #发起请求
json_data = json.loads(response.text) #使用 json.loads 加载和解析 JSON 数据
print(json_data['data']['list'])      #提取字典中的'data'里的'list'键值
```

在例 3-6 代码中,首先使用 json.loads(response.text)将 JSON 格式的响应文本解析为 Python 字典对象,并将结果保存在 json_data 变量中。然后可以通过字典的键访问 JSON 数据中各个值。

【例 3-7】 解析二进制数据。

```
import requests                                          #导入库
response = requests.get ('https://www.gcc.edu.cn/images/2024-07/
                         360e70536994437f804ffd39a306beb5.jpg') #发起请求
image_data = response.content                            #使用 content 返回图片二进制响应数据
with open('image.jpg', 'wb') as f:                       #指定保存图片的文件
    f.write(image_data)                                  #写入响应数据
```

在例 3-7 代码中,首先,使用 response.content 属性获取响应的二进制图片数据,并将其保存在 image_data 变量中。接着,将二进制数据写入 image.jpg 文件中。

3.2.3　数据存储

在使用 Requests 库获取到网页响应数据后，可以将数据存储为 HTML 文件、文本文件、JSON 文件，或者存储到 MySQL 数据库。具体存储为哪种媒介取决于实际的采集需求和数据类型。下面举例说明几种常见的数据存储方式。

【例 3-8】 存储为 HTML 文件。

```
import requests                                          #导入库
import chardet                                           #导入编码检测库
response = requests.get('https://www.gcc.edu.cn/xxgk/xxjj/index.htm') #发起请求
encode = chardet.detect(response.content)                #检测响应内容的编码
response.encoding = encode['encoding']                   #设置响应内容的编码
html_data = response.text                                #获取响应数据
with open ('./tmp/gcc.html', 'w', encoding = 'utf-8') as f: #设置存储路径与文件名
    f.write(html_data)                                   #存储数据到 gcc.html 文件
```

在例 3-8 代码中，将获取的响应数据存储到当前工程目录中 tmp 目录下的 gcc.html 的 HTML 文件中。

【例 3-9】 存储为文本文件。

```
import requests                                          #导入库
import chardet                                           #导入编码检测库
response = requests.get('https://www.gcc.edu.cn/xxgk/xxjj/index.htm') #发起请求
encode = chardet.detect(response.content)                #检测响应内容的编码
response.encoding = encode['encoding']                   #设置响应内容的编码
html_data = response.text                                #获取响应数据
with open('./tmp/gcc.txt', 'w', encoding = 'utf-8') as f:#设置存储路径与文件名
    f.write(html_data)                                   #存储数据到 gcc.txt 文件
```

在例 3-9 代码中，将响应的文本内容存储到当前工程目录中 tmp 目录下名称为 gcc.txt 的文本文件中。

【例 3-10】 存储为 JSON 文件。

```
import requests                                          #导入请求库
import json                                              #导入 JSON 库
URL = 'https://jiaotong.baidu.com/trafficindex/city/list' #设置目标网站
headers = {'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) '
                         'Chrome/127.0.0.0 Safari/537.36'} #设置请求头
response = requests.get(URL, headers = headers)          #发起请求
json_data = json.loads(response.text)                    #使用 json.loads 加载 JSON 数据
with open('./tmp/data.json', 'w') as f:                  #设置存储路径与文件名
    json.dump(json_data['data']['list'], f)              #存储数据到 data.json 文件
```

在例 3-10 代码中，将解析后的 Python 对象存储到当前工程目录中 tmp 目录下名称为 data.json 的 JSON 文件中。注意需要使用 json.dump()函数将 JSON 数据写入文件中。

3.3　综合案例——采集排行榜电影信息

1. 数据采集需求

采集电影排行榜中的“喜剧片”电影信息，抓取的字段有电影名称、排名、演员、上映时间、

类型、评分、评价人数、电影详情页链接,如图 3-3 所示。采集第一个页面上的 20 部电影信息,采集的电影信息保存到 douban.json 文件中。

图 3-3　电影排行榜“喜剧片”电影信息

2. 采集思路分析

(1) 打开电影排行榜主页(网址详见前言二维码),单击分类排行榜中的“喜剧”超链接,如图 3-4 所示。

图 3-4　选择分类排行榜中的“喜剧”超链接

(2) 在打开的图 3-3 页面上,当向下滑动鼠标滚轮时,发现网页内容动态加载,如图 3-5 所示。

(3) 打开 Google 开发者工具,选择 Network→Fetch/XHR 选项卡。按 F5 键刷新网页,同时下滑鼠标滚轮,此时在 Name 列表中可以查看动态加载多个网页地址。选择 Name 列表第 1 个网页地址,在右侧 Headers 的 General 信息中可以查看到请求地址(Requests URL)为 https://movie.douban.com/j/chart/top_list?type=24&interval_id=100%3A90&action=&start=

图 3-5　滑动鼠标滚轮动态加载网页内容

0&limit＝20；请求方法(Request Method)为 GET；状态码(Status Code)为 200 OK；内容类型(Content-Type)为 application/json；charset＝utf-8，分析请求网址与头部信息如图 3-6 所示。

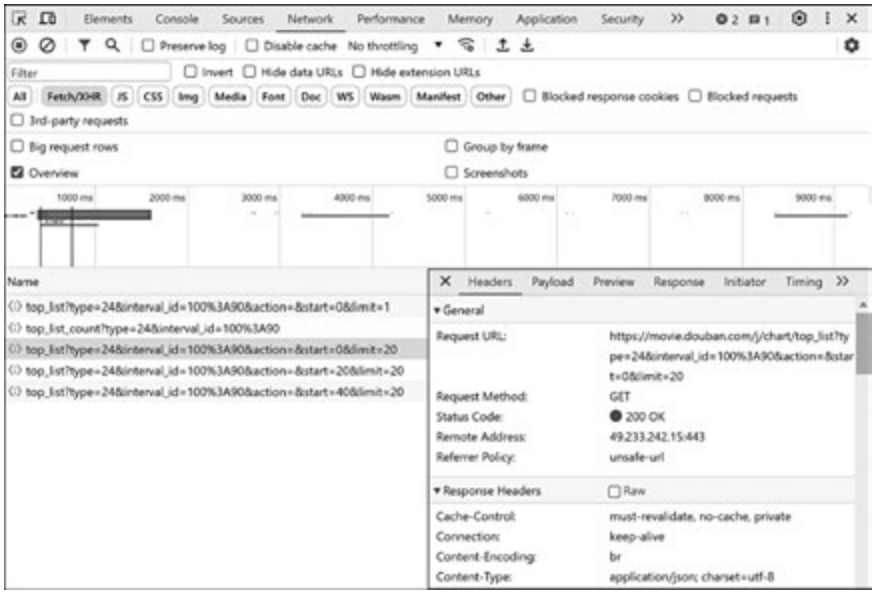

图 3-6　分析请求网址与头部信息

(4) 选择图 3-6 中的 Payload 选项卡，查看请求携带的参数，如图 3-7 所示。

请求携带参数中的“type:24”表示电影类型为喜剧，“start:0”表示电影索引号从 0 开始，“limit:20”表示每个页面有 20 部电影信息。

(5) 选择图 3-6 的 Preview 选项卡，可以看到预览数据中的 20 部电影信息，如图 3-8 所示。

图 3-7　查看请求携带的参数

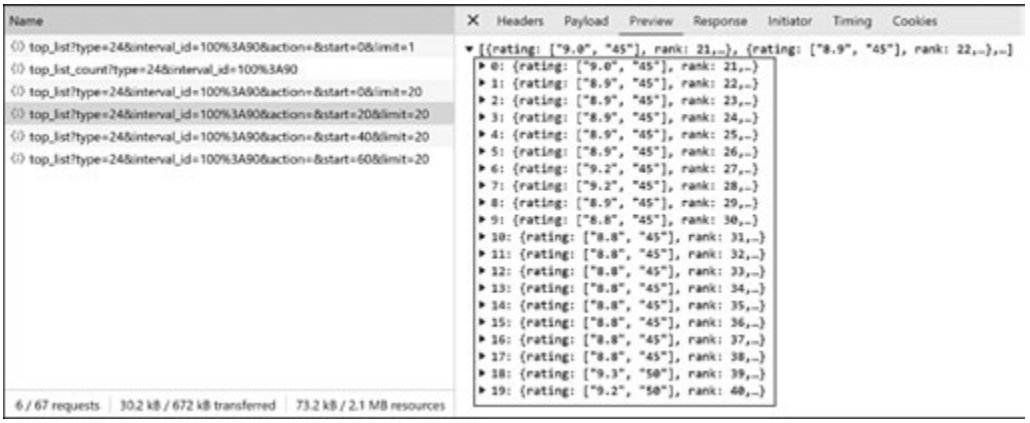

图 3-8　预览数据中的电影信息

3. 采集代码实现

(1) 在工程项目 CrwalProject 中新建一个名称为 Chapter3 Code 的包,并在该包中新建一个名称为“综合案例——采集排行榜电影信息”的 Python 文件。

(2) 接着,在 Python 文件中输入采集排行榜电影信息的爬虫代码,如图 3-9 所示。

```
综合案例—采集排行榜电影信息.py
1   import requests     # 导入请求库
2   import json         # 导入json 库
3
4   if __name__ == '__main__':
5       url = "https://movie.douban.com/j/chart/top_list"   # 网页地址
6       param = {                                           # 设置请求参数
7           'type': 24,
8           'interval_id': '100:90',
9           'action': '',
10          'start': 0,                                     # 设置电影索引号从0开始
11          'limit': 20                                     # 一个页面 20 部电影
12      }
13      headers = {'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 '
14                               '(KHTML, like Gecko) Chrome/127.0.0.0 Safari/537.36'  # 设置请求头
15      }
16      response = requests.get(url=url, params=param, headers=headers)  # 发起请求
17      movieData = response.json()                                      # 加载电影信息
18      fp = open('./tmp/douban.json', 'w', encoding='utf-8')            # 设置存储路径
19      json.dump(movieData, fp=fp, ensure_ascii=False)                  # 存储电影信息
20      print('当前页面电影信息采集完毕!')
21
```

图 3-9　采集排行榜电影信息的爬虫代码

在图 3-9 的爬虫代码中,param 参数用于设置请求携带的电影信息,如 type 为 24 表示电影类型为喜剧。

（3）单击工具栏上的▶按钮，运行采集排行榜电影信息的爬虫代码，此时，可以看到PyCharm窗口底部的Run窗口中显示“当前页面电影信息采集完毕!”的运行结果，如图3-10所示，表明电影信息采集成功。

图3-10　爬虫代码的运行结果

（4）在PyCharm左侧的项目结构区，读者可以看到存储电影信息的文件douban.json。

3.4　拓展案例——采集翻译信息

1. 数据采集需求

用户在百度翻译主页(网址详见前言二维码)的文本框中输入需要翻译的某个英文单词，希望能够抓取该单词的中文翻译内容。例如输入英文单词Computer，采集Computer单词的中文翻译内容，如图3-11所示。采集的数据存储到Computer.json文件中。

图3-11　输入英文单词Computer

2. 采集思路分析

（1）在百度翻译主页上右击，从快捷菜单中选择“检查”选项，打开Google开发者工具。选择Network→Fetch/XHR选项，按F5键刷新页面。接着在文本框中输入Computer，查看Name列表中动态加载的文件，如图3-12所示。

（2）选择最后一个sug文件，查看Headers信息，请求地址(Request URL)为https://fanyi.baidu.com/sug，请求方法(Request Method)为POST，状态码(Status Code)为200 OK，内容类型(Content-Type)为application/json。查看Headers信息如图3-13所示。

（3）切换到Payload选项卡，查看Form Data(表单数据)为kw: Computer，这是POST请求方法需要携带的表单参数。查看表单参数如图3-14所示。

（4）切换到Preview选项卡，查看Computer单词翻译的预览信息，可以看到返回的响应数据是一组JSON格式的数据，如图3-15所示。

3. 采集代码实现

（1）在Chapter3 code包中新建一个名称为“拓展案例——采集翻译信息”的Python文

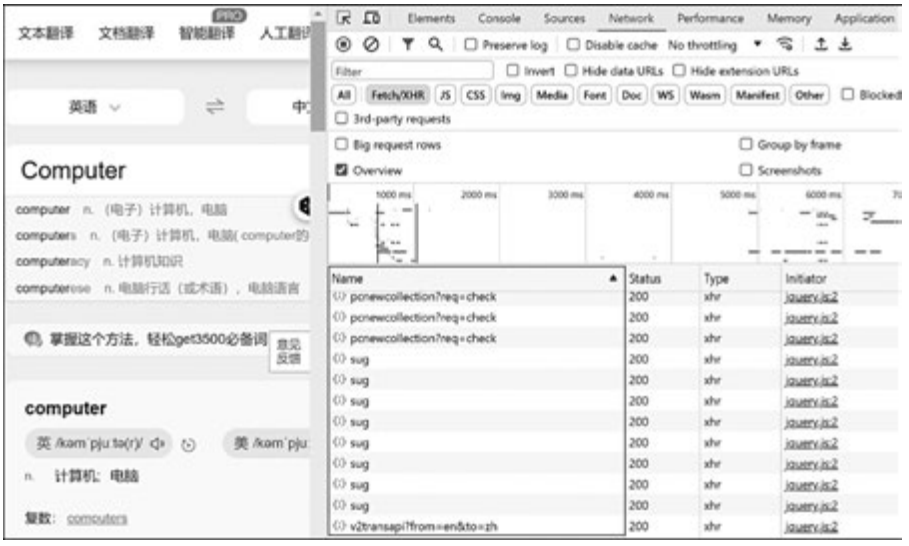

图 3-12　查看 Name 列表中动态加载的文件

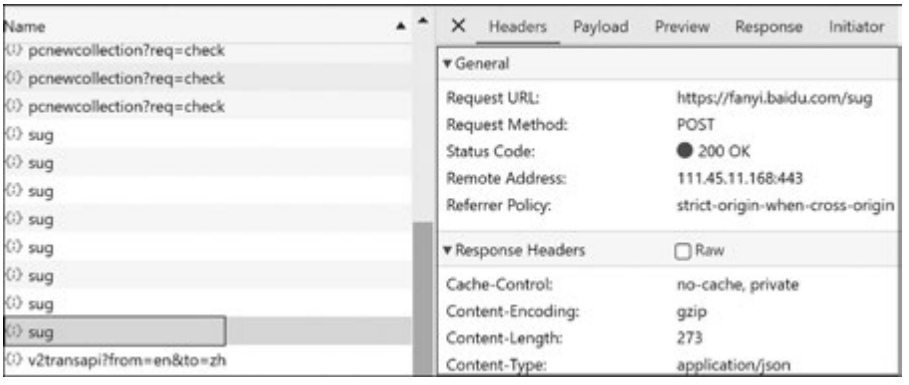

图 3-13　查看 Headers 信息

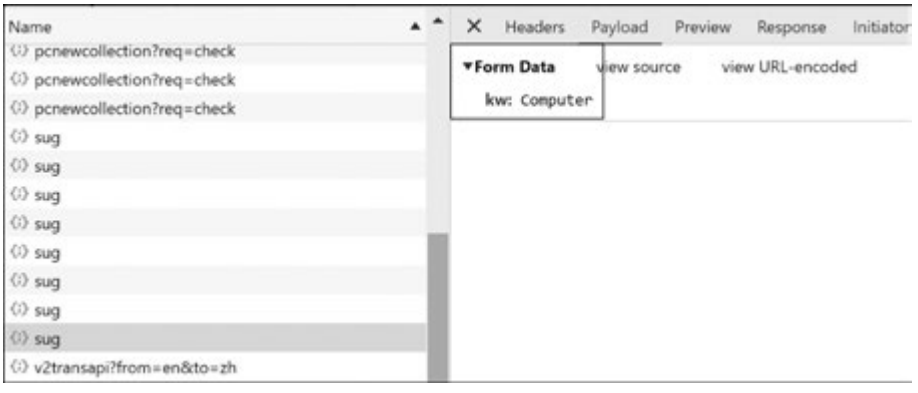

图 3-14　查看表单参数

件，输入采集翻译信息的爬虫代码，如图 3-16 所示。

在图 3-16 的代码中，定义了一个字典 data，用以设置 POST 请求需要携带的数据。当需要向服务器提交数据时，可以使用 POST 请求。与 GET 请求不同，POST 请求将数据作为请求的一部分发送给服务器，而不是作为 URL 的查询参数传递。

需要注意的是，将数据作为参数传递给 post()方法时，数据会被自动编码为表单形式，并

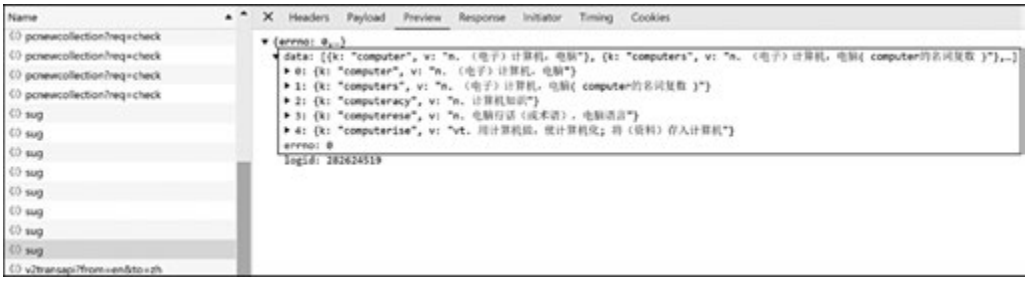

图 3-15　查看预览数据

图 3-16　输入采集翻译信息的爬虫代码

添加到请求体中。另外，赋值给 headers 变量的字典中的值可以直接在图 3-13 中找到并复制。

(2) 单击菜单栏中的 Run 按钮。在底部的 Run 窗口中输入需要翻译的英文单词 Computer，按 Enter 键运行爬虫代码。爬虫代码运行结束后，可以在 Run 窗口查看采集到的 Computer 单词的中文翻译内容，如图 3-17 所示。

图 3-17　查看采集的翻译信息

同时，读者可以在 PyCharm 的项目结构区看到采集到的翻译信息存储在 tmp 目录下的 Computer.json 文件中。

实训案例　　习题 3