第 3 章

网络爬虫 App

网络爬虫能够实现数据的自动化、批量化采集操作，应用广泛。根据爬虫的功能定位，爬虫主要分为通用式与主题式两类。前者常见于各大搜索引擎，后者常见于各类主题定制应用。从软件结构上看，爬虫一般包括控制器、解析器、资源库三部分。控制器负责抓取页面，解析器负责页面内容的分类提取，资源库用于数据存储。

本章以苹果树病虫害数据的爬取为例完成一个主题定向的爬虫设计，遵循爬虫控制器、解析器和资源库的设计逻辑，采用 Python 自身集成的 HTTP 编程库 urllib 完成页面采集，采用第三方网页数据解析库 BeautifulSoup 完成页面解析，采用 SQLite 数据库完成数据存储。

视频讲解

3.1 主模块概要设计

爬虫程序的运行逻辑如图 3.1 所示，包括读取 URL 列表、抓取页面、页面解析、写入数据库、创建数据库、下载图片六个二级子模块，输入文件 input.txt 中包含待爬取页面的 URL 地址列表，输出文件 apple.db 是一个 SQLite 数据库，存放爬取的数据。虚线箭头表示数据获取和流动的方向。

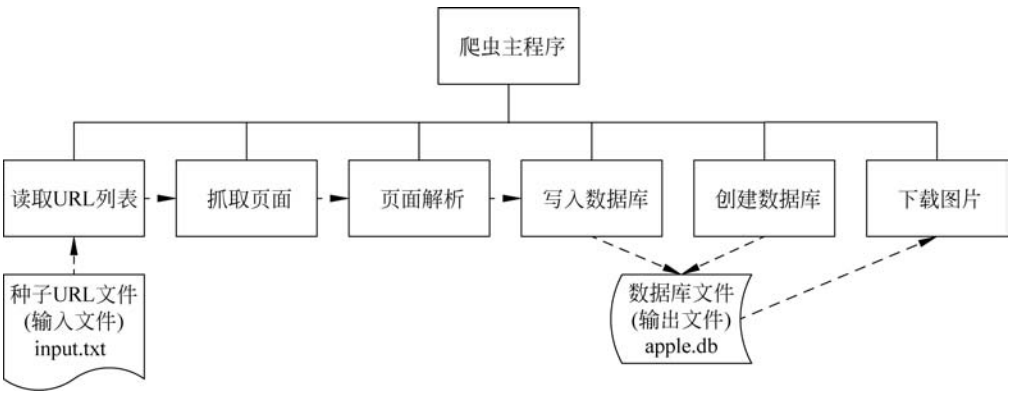

图 3.1 爬虫程序的运行逻辑

在 NetworkProgram 项目下新建子目录 chapter3，在 chapter3 中新建主程序 spider_page.py，完成其初始编码逻辑，如程序段 P3.1 所示，主函数 main()中只包含两个打印语句，创建数据库、读取 URL 列表、抓取页面、页面解析、写入数据库这五个二级子模块编码

完成后，再对主程序段 P3.1 做迭代设计。

P3.1：爬虫主程序

```
import os
import argparse
# 主逻辑函数
def main(database:str, input_urls:str):
    print(f'存储数据的数据库是:{database}')
    print(f'网页地址列表文件是:{input_urls}')
if __name__ == '__main__':
    # 定义命令参数行
    parse = argparse.ArgumentParser()
    parse.add_argument('-db', '--database', help='SQLite 数据库名称')
    parse.add_argument('-i', '--input', help='包含 url 的文件名称')
    # 读取命令参数
    args = parse.parse_args()
    database_file = args.database
    input_file = args.input
    # 调用主函数
    main(database=database_file, input_urls=input_file)
```

为了能够在 PyCharm 的 IDE 环境中对包含命令参数的爬虫主程序做初步测试，需要配置程序 spider_page.py 的运行参数，打开配置运行参数对话框，如图 3.2 所示，对程序 spider_page.py 配置命令参数-i input.txt -db apple.db，指定输入文件和输出文件两个参数，单击 OK 按钮。

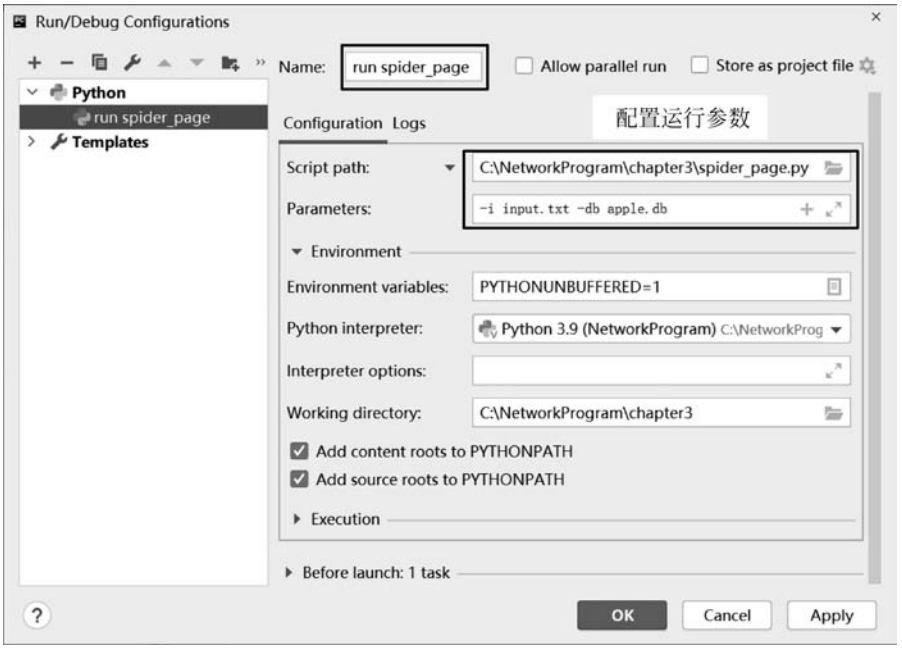

图 3.2　配置运行参数

运行程序 spider_page.py，观察输出结果，厘清主程序逻辑。


视频讲解

3.2 子模块概要设计

在 chapter3 目录下新建 Python 包 utility,在 utility 中新建用于处理 URL 的 Python 程序 url_handle.py,在 url_handle.py 中新建三个函数:read_url()读取 URL 列表;get_page()抓取页面;extract_page()解析页面。读取 URL 列表、抓取页面和解析页面如程序段 P3.2 所示。

P3.2:读取 URL 列表、抓取页面和解析页面

```
# 读取 URL 列表,从文本文件中读取 URL
def read_url(file_path: str):
    pass
# 抓取页面,从 Web 下载页面
def get_page(url: str):
    pass
# 解析页面,提取需要的内容
def extract_page(page_contents: str):
    pass
```

在 utility 包下面新建用于数据库操作的 Python 程序 db_handle.py,在 db_handle.py 中新建三个函数:create_database()创建数据库和数据表;save_to_database()将页面数据写入数据库;download_image()读取图片 URL,完成图片下载。数据库操作如程序段 P3.3 所示。

P3.3:数据库操作

```
# 创建数据库
def create_database(db_path: str):
    pass
# 写入数据库
def save_to_database(db_path: str, records: list):
    pass
# 根据数据库中的图片 URL 去下载图片
def download_image(db_path: str, img_path: str):
    pass
```

各模块详细设计与实现将在随后各节一一给出。


视频讲解

3.3 抓取页面

在 PyCharm 中选择 chapter3 文件夹,右击,在弹出的快捷菜单中选择 New 命令新建文本文件 input.txt,将苹果树病虫害防治页面的 URL 存放其中,input.txt 作为 URL 的种子文件存放于 chapter3 的根目录下。

修改、完善 url_handle.py 中的 read_url()函数和 get_page()函数设计,读取 URL 列表并抓取页面如程序段 P3.4 所示。

P3.4:读取 URL 列表并抓取页面

```
from urllib.request import urlopen
```

```
from bs4 import BeautifulSoup
# 读取 URL 列表,从文本文件中读取 URL
def read_url(file_path: str):
    try:
        with open(file_path) as f:
            url_list = f.readlines()
            return url_list
    except FileNotFoundError:
        print(f'找不到文件{file_path}')
        exit(2)
# 抓取页面,从 Web 下载页面
def get_page(url: str):
    response = urlopen(url)
    html = response.read().decode('utf-8')
    return html
```

程序段 P3.4 首先导入 urllib 和 BeautifulSoup 库。其中,BeautifulSoup 库需要在项目的虚拟环境中单独安装配置。可以通过两种方法安装 BeautifulSoup:一是在项目虚拟环境的命令窗口中执行 pip install beautifulsoup4 命令;二是在 PyCharm 的项目解释器中配置。

read_url()函数读取指定的文本文件后,返回以行为单位的字符串列表,每一行代表一个页面的 URL。

get_page()函数根据 URL 发送 HTTP 请求,获得页面响应后,读取并返回页面的 HTML 格式的数据内容。

3.4　页面解析


视频讲解

页面解析之前,需要首先查看和理解页面的 HTML 结构信息,明确提取数据的结构特点,根据 HTML 标签的组织与排列,灵活运用 BeautifulSoup 库提供的页面解析函数,完成数据的提取。采用 html5lib 作为页面解析器,所以完成本节内容之前,需要在项目的虚拟环境中安装 html5lib 模块,安装命令为 pip install html5lib。

苹果树病虫害页面中包含 27 种病虫害的名称、图片、症状、发病规律和防治方法,图 3.3 所示为其中一个数据片段,以第一条数据为例,数字标注部分是需要提取的内容,提取数据的逻辑通过 extract_page()函数实现。

不难看出,病虫害名称在<p>标签中,症状、发病规律和防治方法在<p>标签中,图片在<p><img>中。<p>节点是所有数据的父节点,是并列关系。

修改完善 url_handle.py 中的 extract_page()函数设计,如程序段 P3.5 所示。

P3.5:页面解析

```
# 解析页面,提取需要的内容
def extract_page(page_contents: str):
    article = BeautifulSoup(page_contents, 'html5lib').article   # 正文
    print('********** 显示所有的数据条目 ********** ')
    print(article.prettify())
    p_tags = article.find_all('p')                               # 所有<p>节点
    p_tags = BeautifulSoup(str(p_tags), 'html5lib').find_all('p')# 重组<p>列表
```

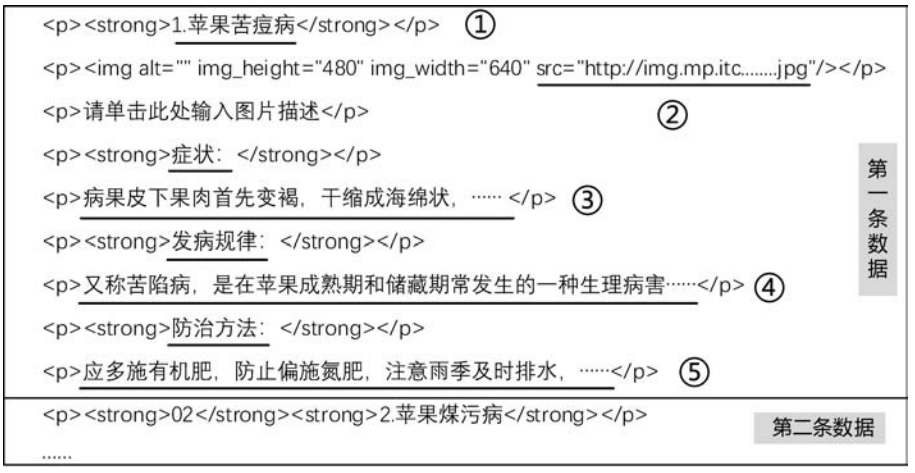

图 3.3 页面数据结构解析

```
    i = 1                                                    # 数据条目序号
    records = []                                             # 存放所有数据条目
    for tag in p_tags:                                       # 遍历
        if tag.children is not None:                         # 有子节点
            for child in tag.children:                       # 遍历子节点
                if f'{i}.' in str(child.string):             # 是病虫害名称
                    disease_name = str(child.string)
                    next_tag = tag.find_next_sibling()       # 指向下一个兄弟节点
                    img_url = next_tag.img.get('src')        # 提取图片的 URL
                    # 向下移动标签,跳过无用的数据,遇到<p><strong>节点为止
                    while True:
                        next_tag = next_tag.find_next_sibling()
                        if next_tag.strong:
                            break
                    # 提取症状
                    disease_feature = ''
                    if '症状' in str(next_tag.strong.string):
                        while True:
                            next_tag = next_tag.find_next_sibling()
                            if '发病规律:' in str(next_tag.string):
                                break
                            disease_feature += next_tag.string
                    # 提取发病规律
                    disease_regular = ''
                    if '发病规律:' in str(next_tag.string):
                        while True:
                            next_tag = next_tag.find_next_sibling()
                            if '防治方法:' in str(next_tag.string):
                                break
                            disease_regular += next_tag.string
                    # 提取防治方法
                    disease_cure = ''
                    if '防治方法:' in str(next_tag.string):
```

```
                        while True:
                            next_tag = next_tag.find_next_sibling()
                            if next_tag.children is not None:
                                if next_tag.strong:
                                    break
                            disease_cure += str(next_tag.string)
                    # 构建数据字典,存储提取的数据
                    record = {'name': disease_name, 'feature': disease_feature,
                    'regular': disease_regular, 'cure': disease_cure, 'img_url': img_url}
                    records.append(record)                          # 字典数据加入列表中
                    i += 1                                          # 条目数量加 1
    return records                                                  # 返回数据列表
```

3.5　创建数据库


视频讲解

创建 apple.db 数据库,存放于 chapter3 的根目录下。如果安装了 PyCharm 专业版,则以 PyCharm 的数据库面板作为辅助工具,完成数据库和数据表的创建,复制数据库和数据表的 SQL 脚本,完成数据库的程序设计。

打开 PyCharm 的数据库面板,新建 SQLite 数据库文件 apple.db。

PyCharm 是用 Java 语言编写的,首次创建数据库时,需要安装 SQLite 数据库的 JDBC 驱动程序。SQLite 是文件型数据库,其连接字符串即为文件路径,如图 3.4 所示。可以即时测试数据库的连通情况。

在 apple.db 数据库中创建数据表 apples,其结构如图 3.5 所示,包含 id、name、feature、regular、cure、img_url 六个字段,其中 id 为主键,类型为 integer,其他字段类型为 text。

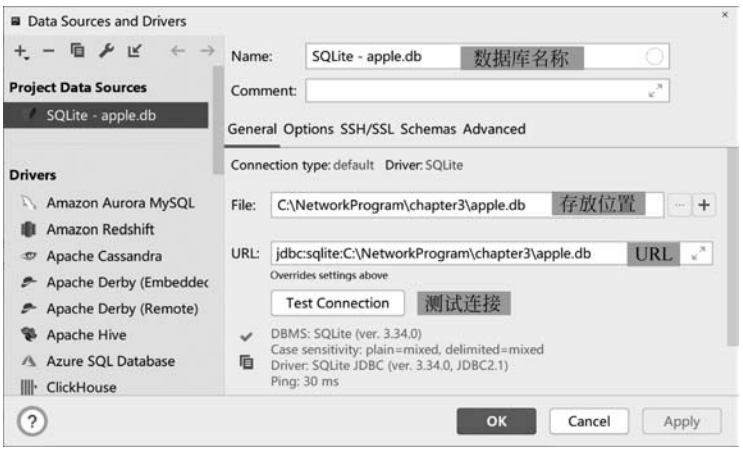

图 3.4　创建数据库并配置驱动程序

图 3.5　apples 数据表结构

为了能够根据需要在程序中动态创建数据库和数据表,借助数据库面板的生成 DDL 语句功能,分别生成创建数据库和数据表的 SQL 语句,完成 db_handle.py 模块中的 create_database()函数设计,如程序段 P3.6 所示。

P3.6:创建数据库和数据表

```
import sqlite3 as lite
# 创建数据库
def create_database(db_path: str):
    conn = lite.connect(db_path)                    # 创建或打开数据库
    with conn:
        cur = conn.cursor()                         # 数据库游标
        cur.execute('drop table if exists apples')  # 删除已存在的数据表
        # 创建新的数据表 apples
        ddl = 'create table apples (id integer not null constraint apples_pk primary key
             autoincrement, name text not null,feature text,regular text,cure text
             ,img_url text);'
        cur.execute(ddl)
        # 创建索引
        ddl = 'create unique index apples_id_uindexon apples (id);'
        cur.execute(ddl)
```

如果没有安装 PyCharm 专业版,则可以采用 DB Browser for SQLite 完成数据库的可视化设计和 DDL 脚本的定义。如果对 SQL 非常熟悉,也可以在 PyCharm 社区版中直接采用程序段 P3.6 中给出的数据库的 DDL 定义。


视频讲解

3.6 写入数据库

db_handle.py 模块中的 save_to_database()函数负责将解析页面返回的数据列表写入数据库中,如程序段 P3.7 所示。

P3.7:页面数据写入数据库

```
# 写入数据库
def save_to_database(db_path: str, records: list):
    conn = lite.connect(db_path)                # 打开数据库
    with conn:
        cur = conn.cursor()
        for record in records:                  # 遍历数据条目表
            print(record)                       # 显示当前条目
            # 根据条目名称查询数据表
            name = record['name']               # 名称
            sql = f"select count(name) from apples where name = '{name}'"
            cur.execute(sql)
            count = cur.fetchone()[0]
            if count <= 0:                      # 数据条目不存在
                feature = record['feature']     # 症状
                regular = record['regular']     # 发病规律
                cure = record['cure']           # 防治方法
                img_url = record['img_url']     # 图像 URL
                # 数据条目插入数据表中
                sql = f"insert into apples(name, feature, regular, cure, img_url)
                      values ('{name}', '{feature}', '{regular}', '{cure}', '{img_url}')"
                cur.execute(sql)
    print('数据存储工作已完成!')
```

3.7　下载图片


视频讲解

数据库 apple.db 中已经存储了所有图片的 URL，据此可以逐个下载对应的图片，为了让图片与数据条目一一对应，图片文件用 id.jpg 命名，例如 1.jpg、2.jpg 等。

图像数据的获取有多种方法，例如 urllib 模块或者 requests 模块。urllib 模块内置在 Python 标准库中，前面抓取页面时采用的即是 urllib 模块。而 requests 是第三方库，需要用 pip install requests 命令单独安装。

模块 urllib 与 requests 均面向 HTTP 通信，requests 在用法上较 urllib 更为简洁。图片下载逻辑由 db_handle.py 模块的 download_image()函数实现，如程序段 P3.8 所示。

P3.8：下载图片

```
# 根据数据库中的图片 URL 去下载图片
def download_image(db_path: str, img_path: str):
    records = []
    conn = lite.connect(db_path)                              # 打开数据库
    with conn:
        cur = conn.cursor()
        sql = "select id,img_url from apples"                 # 所有记录
        cur.execute(sql)
        records = cur.fetchall()                              # 列表包含所有数据
    print('\n 开始图片下载……')
    for record in records:                                    # 遍历所有图片
        file = img_path + '\\' + str(record[0]) + '.jpg'      # 用 ID 命名文件
        img_data = requests.get(record[1])                    # 获取图像数据
        with open(file, 'wb') as f:
            f.write(img_data.content)                         # 保存
    print('\n 已完成图片下载!')
```

3.8　集成测试


视频讲解

创建数据库、读取 URL 列表、抓取页面、页面解析、写入数据库、下载图片六个二级子模块已经全部完成，重新回到主程序模块 spider_page.py，完成主函数和主逻辑设计。

主函数中需要调用各个二级子模块，所以程序头部需要添加模块引用：

```
from utility import url_handle, db_handle
```

主程序逻辑如程序段 P3.9 所示。

P3.9：主程序逻辑

```
import os
import argparse
from utility import url_handle, db_handle
# 主逻辑函数
def main(database: str, input_urls: str):
    print(f'存储数据的数据库是:{database}')
    print(f'网页地址列表文件是:{input_urls}')
```

```
    all_records = []                                                    # 所有数据
    urls = url_handle.read_url(input_urls)                              # 读取 URL 列表
    for url in urls:                                                    # 遍历 URL
        print('读取 url:' + url)
        html = url_handle.get_page(url)                                 # 抓取页面
        records = url_handle.extract_page(html)                         # 解析页面
        all_records.extend(records)                                     # 扩展数据集
    db_path = os.path.join(os.getcwd(), database)                       # 数据库路径
    db_handle.create_database(db_path = db_path)                        # 创建数据库
    db_handle.save_to_database(db_path = db_path, records = all_records) # 写入数据库
    # 下载图片
    directory = 'images'
    if not os.path.exists(directory):
        os.makedirs(directory)
    img_path = os.path.join(os.getcwd(), directory)                     # 图片存放路径
    download_image = threading.Thread(target = db_handle.download_image,
                                      args = (db_path, img_path))
    download_image.start()
if __name__ == '__main__':
    # 定义命令参数行
    parse = argparse.ArgumentParser()
    parse.add_argument('-db', '--database', help = 'SQLite 数据库名称')
    parse.add_argument('-i', '--input', help = '包含 url 的文件名称')
    # 读取命令参数
    args = parse.parse_args()
    database_file = args.database                                       # 数据库名称
    input_file = args.input                                             # 地址列表文件
    # 调用主函数
    main(database = database_file, input_urls = input_file)
```

项目结构如图 3.6 所示。

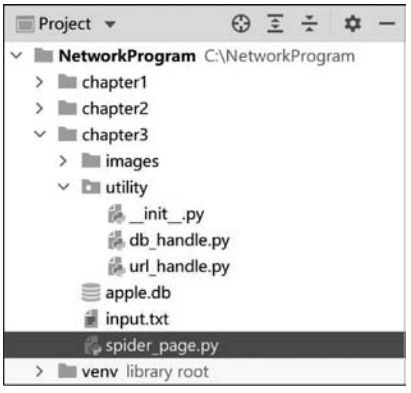

图 3.6 项目结构

运行主程序 spider_page.py,分别在数据库、图片文件夹和控制台观察运行结果。

数据库中包含 27 条苹果树病虫害数据信息,图片文件夹 images 中包含 27 幅图片,图片以数据条目的 ID 命名,控制台上会输出数据采集过程的一些提示信息。


视频讲解

3.9　小结

本章基于 urllib、requests 模块的 HTTP 数据获取能力，基于 BeautifulSoup、html5lib 的页面解析能力，基于 SQLite 的数据管理能力，完成了以苹果病虫害为主题的爬虫设计，展示了爬虫采集数据的过程与逻辑设计，为基于 HTTP 的网络应用与编程提供了很好的案例引导。本章爬虫生成的数据库 apple.db，将直接部署到后面第 5 章和第 6 章开发的 App 中。

3.10　习题

一、简答题

1. 通用式爬虫与主题式爬虫有何不同？
2. 搜索引擎一般采用什么样的爬虫采集数据？
3. urllib 库的作用是什么？有哪些常用函数？
4. BeautifulSoup 库的作用是什么？有哪些常用函数？
5. 采用 html5lib 作为页面解析器的优点是什么？
6. SQLite 是一种什么类型的数据库？有何特点？
7. 第三方 requests 库模块与 Python 自带的库模块 urllib 均面向 HTTP 通信，有何不同？
8. HTML 页面结构有何特点？用什么方法可以根据 URL 获取页面内容？
9. 爬虫为什么需要对图片数据单独下载？

二、编程题

参照本章苹果树病虫害主题爬虫的设计逻辑，自由选择一个主题，完成新主题爬虫的程序设计。爬取的文本数据存入 SQLite 数据库。如果有图片，下载的图片存入单独的文件夹中。