基于 Python 的网络爬虫原理与实训

210810xxx 周 xx 大数据分析实验三

实验报告

摘要

本文是基于 python 的**网络爬虫原理**与实训。文章分为三个部分，分别介绍了爬虫的基本原理、高阶技术和实战案例。

在第一部分，文章介绍了**网页编程**、网络访问、爬虫的基本原理，为基础爬虫技术打下理论基础。文章通过简单的示例代码，解释了如何使用 python 的 requests 库和 Beautiful Soup 库进行网页请求和解析，以及如何使用正则表达式进行数据提取。

在第二部分，文章介绍了高阶爬虫技术，包括多进程**多线程技术**、**数据库交互**与**反爬虫**，以此应对更复杂的爬虫环境。文章详细地讲解了如何使用 python 的 multiprocessing 库和 threading 库进行并发爬取，如何使用 pymysql 库进行数据存储和管理，以及如何使用代理、cookies、headers 等方法应对网站的反爬机制。

在第三部分，文章以虚拟书店和豆瓣影评为例子，对基于 python 的爬虫技术进行实训讲解。文章从需求分析、数据源选择、代码编写等方面，展示了两个完整的爬虫项目的开发过程和结果。文章还进行了串联和多进程爬虫的效率对比。

在本文的末尾，文章给出了一些爬虫过程中遇到的困难及调试过程，并在附录中粘贴了全流程代码。代码封装良好、简洁清晰，具有较高的错误处理能力，考虑了多种异常情况。

关键词：网络爬虫原理 网页编程 多进程多线程技术 数据库交互 反爬虫技术

目录

一、介绍网页编程的基本原理

1.1 WEB 的基本原理

WEB 是 WWW(World Wide Web)的简称，也称为万维网，它是一种基于超文本和 HTTP 的、全球性的、动态交互的、跨平台的分布式图形信息系统。是建立在 Internet 上的一种网络服务，为浏览者在 Internet 上查找和浏览信息提供了图形化的、易于访问的直观界面，其中的文档及超级链接将 Internet 上的信息节点组织成一个互为关联的网状结构。

Web 是一种典型的分布式应用架构，使用 HTML(Hypertext Markup Language)实现信息文档的表示，使用 URL(Uniform Resource Locator)实现全球信息的精确定位，使用 HTTP(HyperText Transfer Protocol)实现分布式的信息传输。

常用的 WEB 服务器有很多。Windows 平台下最常用的服务器是微软的 IIS(Internet Information Server)。Unix 和 LInux 平台下常用的有 Apache(最流行)、Tomcat、IBMWebSphere、Nginx、Lighttpd。

WEB 主要有静态网站和动态网站。静态网站主要是 HTML 文件，动态网站主要是可以根据请求内容动态地生成 HTML、XML 或其他格式文档的 Web 网页。

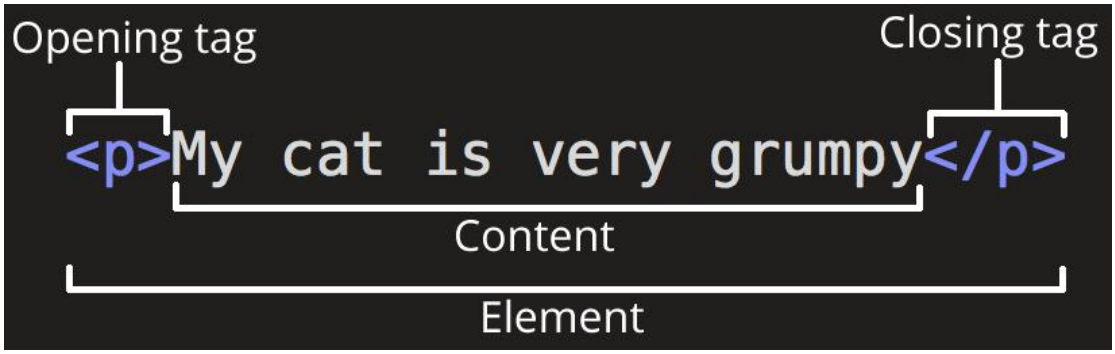
1.2 网页的组成

网页可以分为三大部分——HTML、CSS 和 JavaScript。

1.2.1 HTML

HTML 是用来描述网页的一种语言，其全称叫作 Hyper Text Markup Language，即超文本标记语言。网页包括文字、按钮、图片和视频等各种复杂的元素，其基础架构就是 HTML。不同类型的文字通过不同类型的标签来表示，如图片用 img 标签表示，视频用 video 标签表示，段落用 p 标签表示，它们之间的布局又常通过布局标签 div 嵌套组合而成，各种标签通过不同的排列和嵌套才形成了网页的框架。这些标签定义的节点元素相互嵌套和组合形成了复杂的层次关系，就形成了网页的架构。

比如，在下面这个段落中：



这个元素的主要部分有：

1） 开始标签（Opening tag）：包含元素的名称（本例为 p），被大于号、小于号所包围。表示元素从这里开始或者开始起作用 —— 在本例中即段落由此开始。

2） 结束标签（Closing tag）：与开始标签相似，只是其在元素名之前包含了

一个斜杠。这表示着元素的结尾 —— 在本例中即段落在此结束。初学者常常会犯忘记包含结束标签的错误，这可能会产生一些奇怪的结果。
3） 内容（Content）：元素的内容，本例中就是所输入的文本本身。
4） 元素（Element）：开始标签、结束标签与内容相结合，便是一个完整的元素。
元素也可以有属性（Attribute）：

属性包含了关于元素的一些额外信息，这些信息本身不应显现在内容中。本例中，class 是属性名称，editor-note 是属性的值。class 属性可为元素提供一个标识名称，以便进一步为元素指定样式或进行其他操作时使用。也可以将一个元素置于其他元素之中，称作嵌套。

以上介绍了一些基本的 HTML 元素，但孤木不成林。元素和元素可以彼此协同构成一个完整的 HTML 页面，但同时一个页面的 HTML 元素也会非常复杂。本实验的爬虫主要是识别网页 HTML 中的特定元素进行提取，因此熟悉 HTML 的结构非常重要。本文将在爬取时详细介绍具体提取的 HTML 元素。

1.2.2 CSS

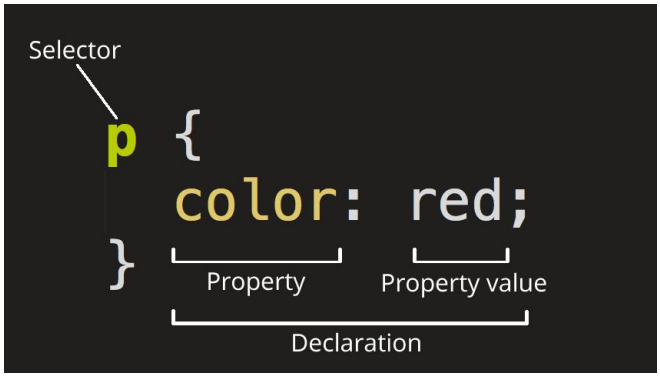
HTML 定义了网页的结构，但是只有 HTML 页面的布局并不美观，可能只是简单的节点元素的排列，为了让网页看起来更好看一些，这里借助了 CSS。CSS，全称叫作 Cascading Style Sheets，即层叠样式表。“层叠”是指当在 HTML 中引用了数个样式文件，并且样式发生冲突时，浏览器能依据层叠顺序处理。“样式”指网页中文字大小、颜色、元素间距、排列等格式。CSS 是目前唯一的网页页面排版样式标准，有了它的帮助，页面才会变得更为美观。在网页中，一般会统一定义整个网页的样式规则，并写入 CSS 文件中（其后缀为.css）。在 HTML 中，只需要用 link 标签即可引入写好的 CSS 文件，比如：

```
1.    <link href="styles/style.css" rel="stylesheet" />
```

CSS 是一门样式表语言，这也就是说人们可以用它来选择性地为 HTML 元素添加样式。举例来说，以下 CSS 代码选择了所有的段落文字，并将它们设置为红色。

```
1.    p {
2.      color: red;
3.    }
```

让我们来剖析以下这个代码的组成部分：



整个结构称为规则集（规则集通常简称规则），注意各个部分的名称：

1） 选择器（Selector）

HTML 元素的名称位于规则集开始。它选择了一个或多个需要添加样式的元素（在这个例子中就是 <p> 元素）。要给不同元素添加样式，只需要更改选择器。

2） 声明（Declaration）

一个单独的规则，如 color: red; 用来指定添加样式元素的属性。

3） 属性（Properties）

改变 HTML 元素样式的途径（本例中 color 就是 <p> 元素的属性）。CSS 中，由编写人员决定修改哪个属性以改变规则。

4） 属性的值（Property value）

在属性的右边，冒号后面即属性的值，它从指定属性的众多外观中选择一个值（我们除了 red 之外还有很多属性值可以用于 color ）。

注意其他重要的语法：

1） 除了选择器部分，每个规则集都应该包含在成对的大括号里（{}）。

2） 在每个声明里要用冒号（:）将属性与属性值分隔开。

3） 在每个规则集里要用分号（;）将各个声明分隔开。

4） 如果要同时修改多个属性，只需要将它们用分号隔开，就像这样：

```
p {
  color: red;
  width: 500px;
  border: 1px solid black;
}
```

也可以选择多种类型的元素并为它们添加一组相同的样式。将不同的选择器用逗号分开。例如：

```
p,
li,
h1 {
  color: red;
}
```

选择器有许多不同的类型。上面只介绍了元素选择器，用来选择 HTML 文档中给定的元素。但是选择操作可以更加具体。下面是一些常用的选择器类型：

选择器名称	选择的内容	示例
元素选择器（也称作标签或类型选择器）	所有指定类型的 HTML 元素	`p` 选择 `<p>`
ID 选择器	具有特定 ID 的元素。单一 HTML 页面中，每个 ID 只对应一个元素，一个元素只对应一个 ID	`#my-id` 选择 `<p id="my-id">` 或 `<a id="my-id">`
类选择器	具有特定类的元素。单一页面中，一个类可以有多个实例	`.my-class` 选择 `<p class="my-class">` 和 `<a class="my-class">`
属性选择器	拥有特定属性的元素	`img[src]` 选择`<img src="myimage.png">` 但不是 `<img>`
伪类选择器	特定状态下的特定元素（比如鼠标指针悬停于链接之上）	`a:hover` 选择仅在鼠标指针悬停在链接上的 `<a>` 元素

等等。

1.2.3 JavaScript

JavaScript，简称 JS，是一种脚本语言。HTML 和 CSS 配合使用，提供给用户的只是一种静态信息，缺乏交互性。我们在网页里可能会看到一些交互和动画效果，如下载进度条、提示框、轮播图等，这通常就是 JavaScript 的功劳。它的出现使得用户与信息之间不只是一种浏览与显示的关系，而是实现了一种实时、动态、交互的页面功能。JavaScript 相当简洁，却非常灵活。开发者们基于 JavaScript 核心编写了大量实用工具，可以使 开发工作事半功倍。

JavaScript 通常也是以单独的文件形式加载的，后缀为 js，在 HTML 中通过 script 标签即可引入：

```
1.    <script src="scripts/main.js" defer></script>
```

JavaScript 比 HTML 和 CSS 都要复杂一些，更类似于 C 语言等编译语言，有变量、运算符、条件语句等语法，由于本实验几乎不涉及 JavaScript 的识别和编写，因此不在此过多赘述。

综上所述，HTML 定义了网页的内容和结构，CSS 描述了网页的布局，JavaScript 定义了网页的行为。

二、介绍网络访问的过程

1.2.1 网络访问的步骤

网络访问的步骤主要是：

1. 用户输入或触发：用户在浏览器中输入网址或点击链接，触发对特定资源的请求。
2. DNS 解析：浏览器通过域名系统（DNS）将用户输入的域名解析为相应的 IP 地址，以确定要访问的服务器位置。

3. 建立 TCP 连接：使用传输控制协议（TCP）建立与目标服务器的连接。TCP 提供可靠的、面向连接的通信。
4. 发起 HTTP 请求：浏览器通过 HTTP（或 HTTPS）协议向目标服务器发送请求，请求特定的资源（例如网页、图像或文件）。
5. 服务器处理请求：服务器接收到请求后，开始处理。这可能涉及到从数据库检索数据、执行业务逻辑等。
6. 服务器发送响应：服务器将请求的资源封装在 HTTP 响应中，并通过 TCP 连接发送回用户的浏览器。
7. 浏览器接收响应：浏览器接收到服务器的响应后，开始解析响应内容。
8. 渲染页面：如果响应是 HTML 页面，浏览器开始解析 HTML、CSS 和 JavaScript，并构建 DOM（文档对象模型）树，然后进行页面渲染。
9. 显示页面：渲染完成后，浏览器显示最终的页面内容，用户可以看到请求的资源。
10. 连接关闭：一旦资源传输完成，浏览器和服务器之间的 TCP 连接被关闭。

这些步骤中，其实最重要的是 4、5、6、7 步。总的来说，可以分为请求和响应两大步骤。

1.2.2 HTTP 请求

HTTP（Hypertext Transfer Protocol）是一种用于传输超文本的协议，常用于在客户端和服务器之间传递信息。浏览器根据用户的输入或触发（例如在地址栏中输入 URL 或点击链接）构建一个 HTTP 请求。这个请求通常由一个请求行、请求头和请求体组成。

- 请求行包含请求的方法（GET、POST 等）、请求的 URL 和协议版本。
- 请求头包含关于请求的附加信息，如用户代理、可接受的响应内容类型、Cookie 等。
- 如果使用 POST 等方法，数据可能会包含在请求体中，例如表单提交的数据。

发送 HTTP 请求需要先建立 TCP 连接。TCP 即传输控制协议，用以建立到目标服务器的连接。这是通过三次握手来实现的，确保客户端和服务器之间的可靠通信。

所谓三次握手(Three-way Handshake)，是指建立一个 TCP 连接时，需要客户端和服务器总共发送 3 个包。三次握手的目的是连接服务器指定端口，建立 TCP 连接,并同步连接双方的序列号和确认号并交换 TCP 窗口大小信息。

一旦 TCP 连接建立，浏览器通过该连接将构建的 HTTP 请求发送到目标服务

器。目标服务器接收到 HTTP 请求后，开始解析请求行、请求头和请求体。

1.2.3 HTTP 响应

服务器接受到 HTTP 请求之后，会根据请求的方法和 URL 执行相应的处理，并生成 HTTP 响应，包括一个响应行、响应头和响应体。其中：

- 响应行包含响应的状态码和描述。
- 响应头包含关于响应的附加信息。
- 响应体包含服务器返回的实际数据。

服务器通过 TCP 连接将构建的 HTTP 响应发送回浏览器。

三、介绍爬虫的基本原理

爬虫是模拟用户在浏览器或者某个应用上的操作，把操作的过程实现自动化的程序。爬虫的基本步骤包括：

1. 发送 HTTP 请求：使用编程语言中的 HTTP 库或框架，向目标 URL 发送 HTTP 请求。请求中包含了必要的信息，如请求方法（通常是 GET）、请求头等。
2. 获取页面内容：接收到服务器的响应后，爬虫获取页面的原始 HTML 或其他标记语言的内容。这可以通过解析 HTTP 响应体实现。
3. 解析页面内容：利用解析库或模块，例如 Beautiful Soup、lxml 等，解析 HTML 文档以提取有用的信息。这可能包括文本、链接、图像等。
4. 提取数据：从解析后的页面内容中提取目标数据。这可能涉及使用正则表达式等方法，以匹配和提取所需的信息。
5. 存储数据：将提取的数据存储在适当的地方，如数据库等数据存储介质。

Python 有许多强大且广泛使用的爬虫库，这些库提供了各种功能，包括发起 HTTP 请求、解析 HTML、处理 Cookies 和 Sessions、处理数据等。以下是一些常用的 Python 爬虫库：

Requests:

requests 模块是一个用于网络请求的模块，主要用来模拟浏览器发请求。requests 模块简单强大高效，使得其在众多网络请求模块中脱颖而出。

Selenium：

Selenium 是一个自动化测试工具，利用它可以驱动浏览器执行特定的动作，如点击、下拉等操作，同时还可以获取浏览器当前呈现的页面的源代码，做到“可见即可爬”。对于一些 JavaScript 动态渲染的页面来说，此种抓取方式非常有效。

Beautiful Soup：

Beautiful Soup 是一个用于解析 HTML 和 XML 文档的 Python 库。它提供了简单又实用的方法来遍历、搜索以及修改 HTML 和 XML 文档的内容。Beautiful Soup 将复杂的 HTML 文档转换为一个复杂的树形结构，每个节点都是一个 Python 对象，通过这些对象，你可以轻松地浏览整个文档树，查找特定的标签或内容。

四、高阶爬虫技术

4.1 多进程多线程技术

多进程和多线程是用于提高爬虫效率的关键技术。它们分别是并行处理和并发处理的手段，可以同时处理多个任务，提高爬虫的整体性能。

多进程和多线程概念：

1) 多进程：多进程是指在一个程序中同时运行多个独立的进程，每个进程有自己的内存空间和系统资源。在 Python 中，可以使用 multiprocessing 模块实现多进程。
2) 多线程：多线程是指在一个进程中同时运行多个线程，线程共享相同的内存空间，但每个线程有自己的执行流。在 Python 中，可以使用 threading 模块实现多线程。

多进程和多线程有以下优势：

1) 提高并发度：多进程和多线程可以同时处理多个任务，提高爬虫的并发度，从而更快地获取数据。
2) 充分利用资源：多进程和多线程能够充分利用多核处理器的性能，加速爬虫的运行速度。
3) 解决阻塞问题：在网络爬虫中，等待网络请求的响应是一种阻塞操作，通过多线程或多进程可以在等待响应的同时执行其他任务，提高效率。
4) 任务分发：多进程和多线程可以用于将任务分配给不同的进程或线程，实现任务的分工协作。

由于 python 的线程锁，本文采用多进程的方法提高效率。代码如下：

```
1.    num_threads = 6
2.    with ThreadPoolExecutor(max_workers=num_threads) as executor:
3.          executor.map(scrape_movie_info, urls)
```

4.2 数据库交互

数据库交互是指爬虫与数据库之间的数据存储、检索、更新和删除操作。这种交互在复杂的爬虫系统中变得尤为重要，因为它允许爬虫有效地管理和组织大量的数据，支持数据的持久性存储和后续的分析处理。

常见的数据库包括：

1) 关系型数据库（RDBMS）： 如 MySQL、PostgreSQL、SQLite，适用于需要事务支持和复杂查询的场景。
2) NoSQL 数据库： 如 MongoDB、Cassandra，适用于大数据量、高并发的场景，具有灵活的数据模型。
3) 键值存储： 如 Redis，用于高速缓存和快速检索。

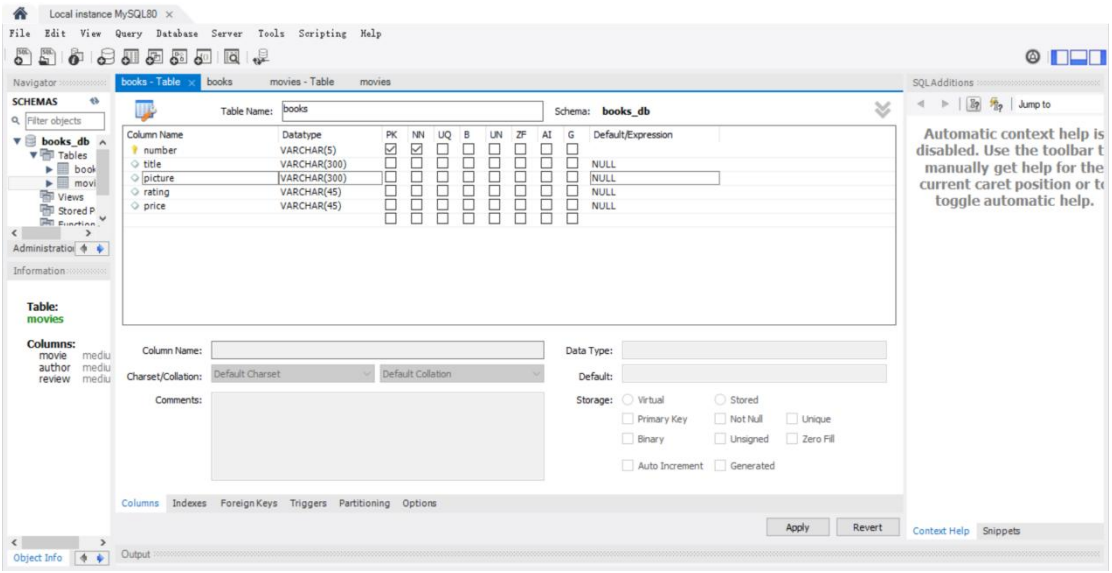
在高阶爬虫中，数据库交互技术的选择取决于具体的需求和系统架构。合理使用数据库可以提高爬虫系统的性能、可维护性和可扩展性。在本次实验中，选

取 MYSQL 作为数据库。使用 python 控制 MYSQL 的代码如下：

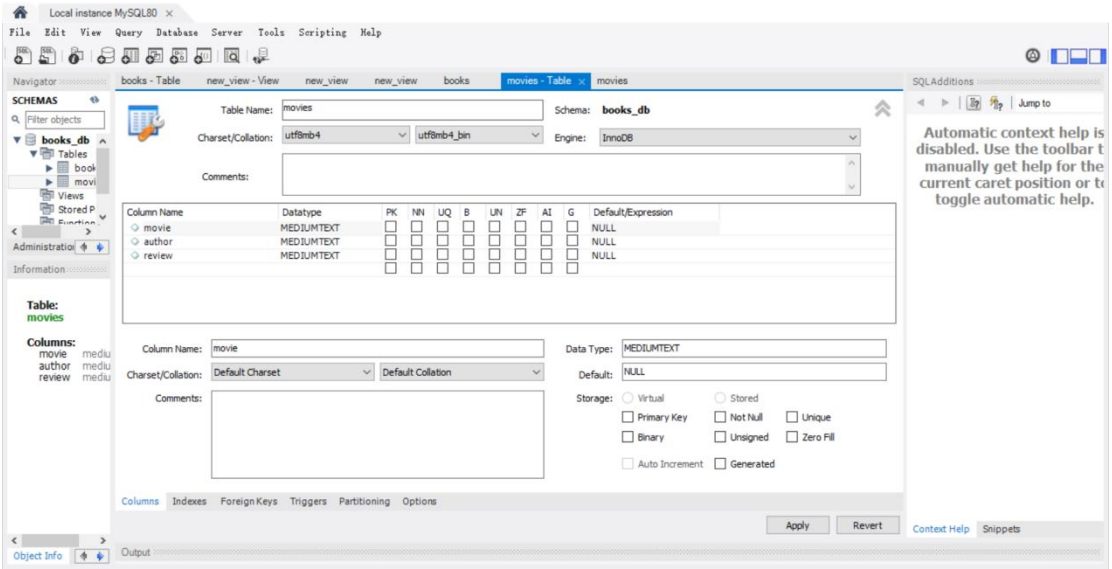
```
    # 初始化数据库连接
    mysql_conn = pymysql.connect(host='localhost', user='root', password='mysqlpassword', database='books_db')
    mysql_cursor = mysql_conn.cursor()

    # 关闭数据库连接
    mysql_cursor.close()
    mysql_conn.close()
```

不同的数据可能需要不同的表格，以本次实验为例，实验一（虚拟书店）的 SQL 信息如图：



实验二（豆瓣影评）的 SQL 信息如图：



4.3 反爬虫

反爬虫是指网站采取一系列措施，以防止爬虫程序对其网站进行非法或过度的访问。这些措施的目的是保护网站的数据、提高服务器性能、防止滥用和维护合理的服务质量。在爬取一些大型网站，比如豆瓣的时候，我们经常会被反爬虫机制困扰。

一些常见的反反爬虫手段包括：

1) 改变 User_agent。这是在模拟不同的浏览器进行提取。
2) 使用 cookie。模仿真人登录。
3) 适当休眠。这是避免频繁提取导致网站察觉。
4) 改变 proxies，即 ip 地址。

在文章*单机 30 分钟抓取豆瓣电影 7 万+数据：论爬虫策略的重要性（附全部数据下载地址）*中，作者分享了许多反爬虫的试验。无论是改变 user_agent 还是 cookie，最终提取过多 ip 都会被封，而每次请求后都休眠又会导致效率过低。因此，最好的办法仍然是使用代理 ip。本文使用快代理的免费 ip 进行提取，如下是提取 ip 地址的代码：

```
    def get_ip():
        url = "http://api.tianqiip.com/getip?secret=cx1xlfx52o8s1rdq&num=5&type=txt&lb=%5Cn®ion=440000&port=1&time=5&mr=1&sign=15fb650d0f872b891e37db13014c89d7"
        res = requests.get(url)
        print(res.text)

        if res.status_code==200:
            prox = res.text
        prox = res.text.split("\n")[:-1]
        proxy = [{'http':'http://'+i} for i in prox]
        # 使用前可判断一下是否可用，也可能返回其他 state code，具体就依据具体情况解决
        print(proxy)
        ts = "http://www.baidu.com/"
        for j in proxy:
            try:
                r = requests.get(ts,proxies=j)
            except:
                print("ip 不可用")
        return prox
```

五、虚拟网站爬取结果

我们的目标是爬取虚拟书店的书籍信息，包括图片、价格、评价，书名等书籍所有内容。网址为 http://books.toscrape.com。分析网站的源代码，确定如下提取方案：

对于源代码：

```
    <article class="product_pod">
            <div class="image_container">
                    <a href="catalogue/tipping-the-velvet_999/index.html
"><img src="media/cache/26/0c/260c6ae16bce31c8f8c95daddd9f4a1c.jpg" alt
="Tipping the Velvet" class="thumbnail"></a>
            </div>
                <p class="star-rating One">
                    <i class="icon-star"></i>
                    <i class="icon-star"></i>
                    <i class="icon-star"></i>
                    <i class="icon-star"></i>
                    <i class="icon-star"></i>
                </p>
             <h3><a href="catalogue/tipping-the-velvet_999/index.html" ti
tle="Tipping the Velvet">Tipping the Velvet</a></h3>
            <div class="product_price">
```

可以通过：

```
    soup = BeautifulSoup(response.text, 'html.parser')
    links = [urljoin(base_url, a['href']) for h3 in soup.find_all('h3')
for a in h3.find_all('a', href=True)]
```

提取出每个书本的链接。需要注意的是，由于 base_url = f"http://books.toscrape.com/catalogue/"并未覆盖所有的书本界面，所以还需要观察页面特征，遍历所有目录页：

```
    all_links = []
        for i in range(1,51):
            url = base_url + f'page-'+str(i)+f'.html'
```

进入到书本的单独界面之后，仍需要提取图片、价格、评价，书名等书籍信息。截取部分源代码：

```
    # 标题
    <h1>A Light in the Attic</h1>

    # 价格
    <p class="price_color">£51.77</p>

    # 评价
    <p class="star-rating Three">
            <i class="icon-star"></i>
            <i class="icon-star"></i>
            <i class="icon-star"></i>
            <i class="icon-star"></i>
            <i class="icon-star"></i>

```

```
            <!-- <small><a href="/catalogue/a-light-in-the-attic_1000/reviews/">


                        0 customer reviews

            </a></small>
             -->

    <!--
        <a id="write_review" href="/catalogue/a-light-in-the-attic_1000/reviews/add/#addreview" class="btn btn-success btn-sm">
            Write a review
        </a>

     --></p>

    #图片
    <div class="item active">
                                    <img src="../../media/cache/fe/72/fe72f0532301ec28892ae79a629a293c.jpg" alt="A Light in the Attic" />

    </div>
```

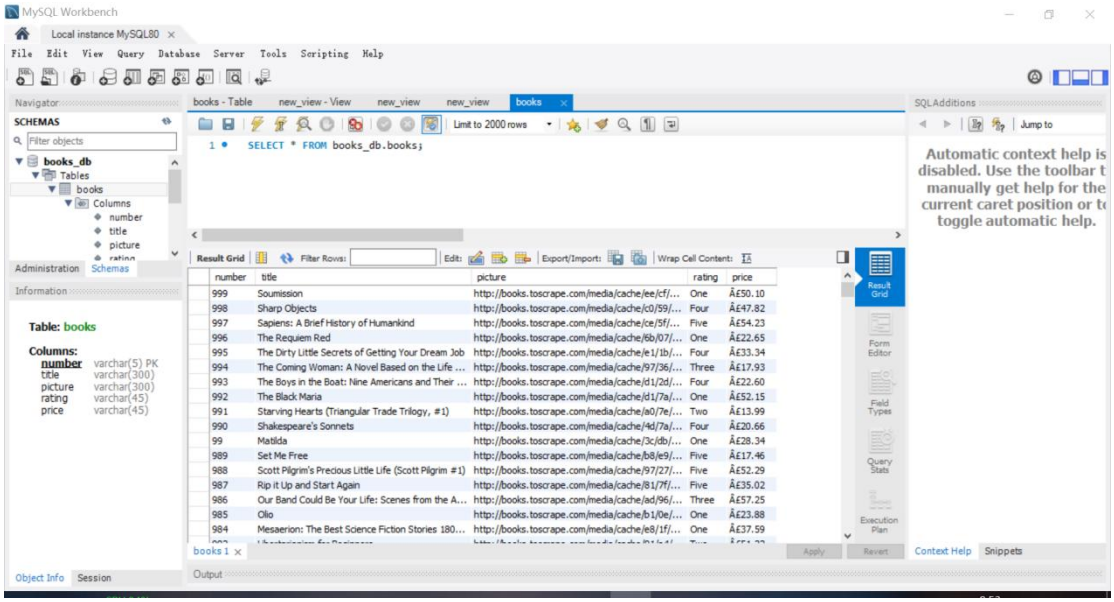
可以确定如下的提取策略：

```
    title = soup.find('h1').text.strip()
    image_url = urljoin(url, soup.find('div', class_='item active').find('img')['src'])
    price = soup.find('p', class_='price_color').text.strip()
    rating = soup.find('p', class_='star-rating')['class'][1]
    number = soup.find('h3').find('a')['href'].split('_')[-1].replace("/index.html", "")
```

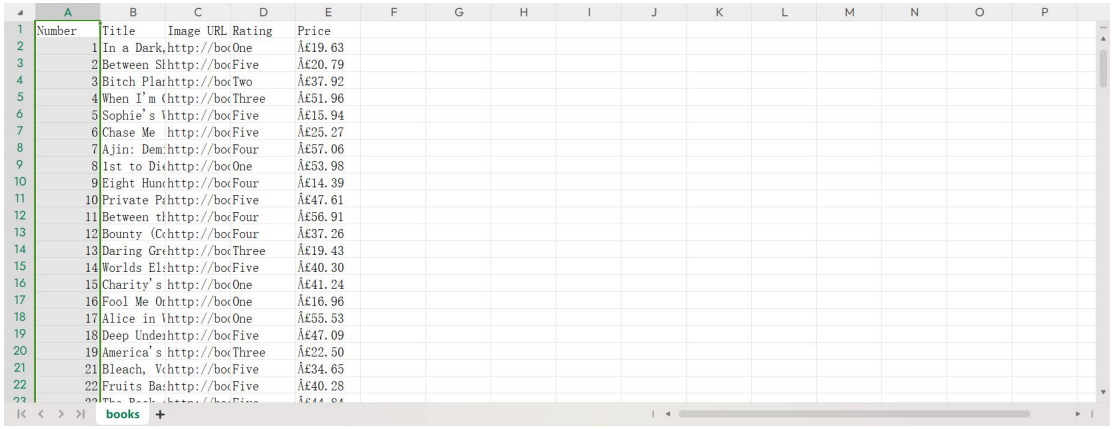
5.1 串联爬取结果与性能分析

5.1.1 串联爬取结果如下：

SQL 界面：



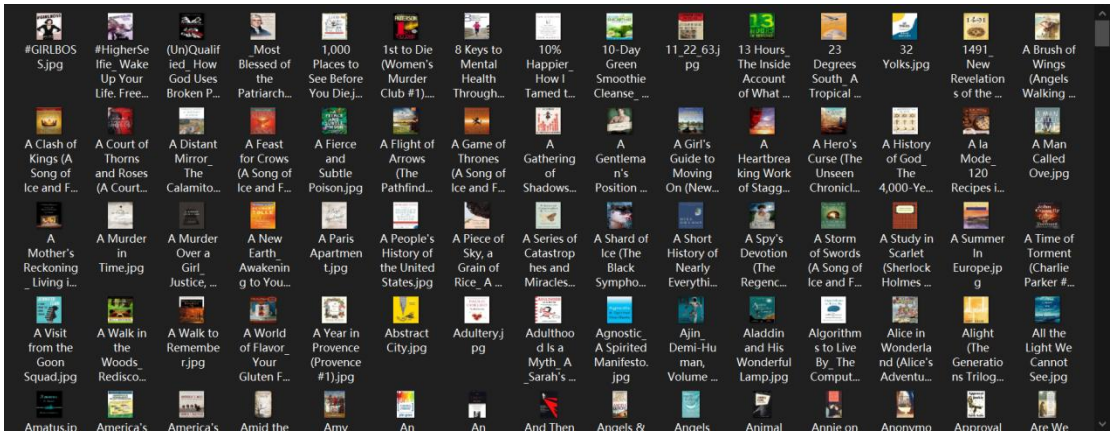
CSV 界面：



	A	B	C	D	E
1	Number	Title	Image URL	Rating	Price
2	1	In a Dark,	http://boo	One	Â£19.63
3	2	Between Sl	http://boo	Five	Â£20.79
4	3	Bitch Pla	http://boo	Two	Â£37.92
5	4	When I'm (	http://boo	Three	Â£51.96
6	5	Sophie's V	http://boo	Five	Â£15.94
7	6	Chase Me	http://boo	Five	Â£25.27
8	7	Ajin: Dem	http://boo	Four	Â£57.06
9	8	1st to Di	http://boo	One	Â£53.98
10	9	Eight Hun	http://boo	Four	Â£14.39
11	10	Private P	http://boo	Five	Â£47.61
12	11	Between t	http://boo	Four	Â£56.91
13	12	Bounty (C	http://boo	Four	Â£37.26
14	13	Daring Gr	http://boo	Three	Â£19.43
15	14	Worlds El	http://boo	Five	Â£40.30
16	15	Charity's	http://boo	One	Â£41.24
17	16	Fool Me O	http://boo	One	Â£16.96
18	17	Alice in W	http://boo	One	Â£55.53
19	18	Deep Unde	http://boo	Five	Â£47.09
20	19	America's	http://boo	Three	Â£22.50
21	21	Bleach, V	http://boo	Five	Â£34.65
22	22	Fruits Ba	http://boo	Five	Â£40.28

books

保存的图片界面：



5.2 高阶爬虫结果与性能分析

改变进程数目，多次爬取，得到下表的数据：

线程数目	1	2	3	4	5
爬取图片					
爬取时间	35m11s	18m24s	12m4s	9m28s	7m48s

线程数目	6	7	8	10	12
爬取图片					
爬取时间	6m40s	5m31s	5m12s	3m51s	3m47s

**根据观察，爬取的时长也受网速的影响。里面的实验并不在同一网络下跑完，因此可能与复现结果有一定差距。*

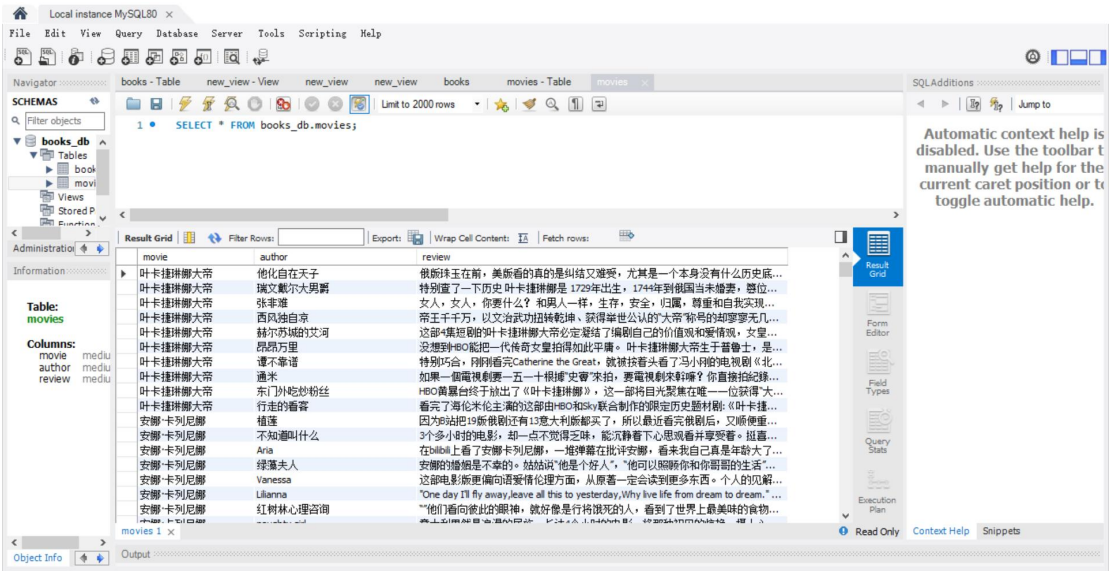
观察得知，在进程数目较少的时候，爬取时间有较明显的提升。在进程数目变大的时候，爬取时间的提升并不明显，但仍然是有提升的。

六、豆瓣影评爬取结果

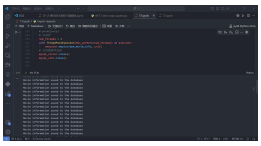
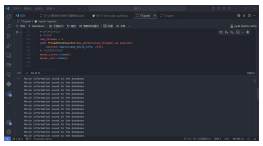
我们的目标是爬取至少一类电影的影评、至少 10 部电影的影评。我们选取“世界古典名著名篇改篇剧集”这一类电影，选取前十部电影的所有影评进行爬取，网址为 https://www.douban.com/doulist/5257286/。分析网站的源代码，确定提取方案。

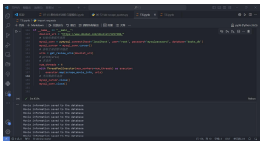
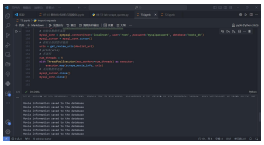
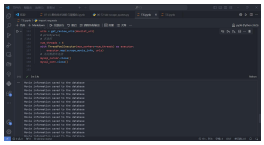
6.1 串联爬取结果与性能分析

SQL 界面：



6.2 高阶爬虫结果与性能分析

线程数目	1	2	3
爬取图片			
爬取时间	8m9s	4m23s	3m22s

线程数目	4	5	6
爬取图片			
爬取时间	2m44s	2m24s	2m5.4a

观察得知，整体趋势与实验一相似。在进程数目较少的时候，爬取时间有较明显的提升。在进程数目变大的时候，爬取时间的提升并不明显，但仍然有提升。

七、结论(困难或调试)

7.1 有关 SQL

问题：Error saving to the database: (0, '')
这个错误表明 SQL 设置了主键，而即将保存的数据的主键与数据库原先的数据冲突了。经过排查发现是因为多次提取没有清除表格内容。为此，我专门写了一个函数，使得每次重新提取的时候首先清除表格。

问题：Error saving to the database: (1406, "Data too long for column 'title' at row 1")
这是因为在 sql 中有设置 str 的最大长度，将 sql 的最大长度调整即可。当然，在豆瓣爬取的时候，我们发现 VARCHAR 的最大长度还不够，因此后面我们使用了 TEXT 格式。

问题：Error saving to the database: (1366, "Incorrect string value: '\\xF0\\x9F\\x8D\\xAC' for column 'author' at row 1")
这个错误表明数据库中的 'author' 列的字符集不兼容 Emoji 表情符号。将数据库表的字符集设置为支持它们的字符集，如 utf8mb4 即可。

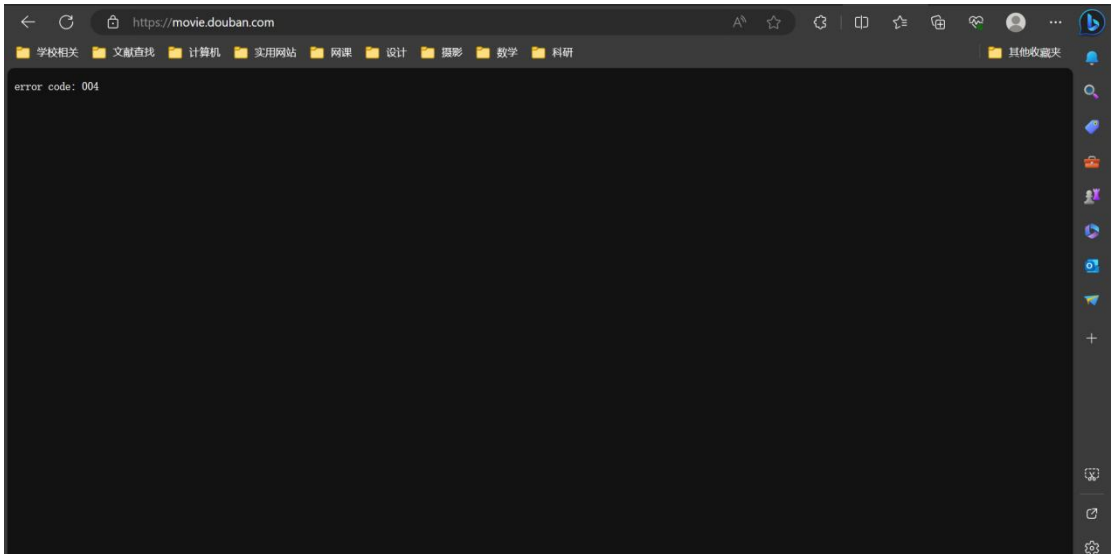
问题：Error saving to the database: (2013, 'Lost connection to MySQL server during query')
这个错误表明在执行 MySQL 查询的过程中，与 MySQL 服务器的连接丢失了。这可能是由于多种原因引起的，比如网络问题、MySQL 服务器崩溃、连接超时等。

问题：Error saving to the database: Packet sequence number wrong - got 1 expected 2/Error saving to the database: (2013, 'Lost connection to MySQL server during query')
感觉是网络不太稳定。

7.2 有关网络

在整个任务过程，遇到了许许多多的网络相关的问题。比如：

1） HTTPSConnectionPool(host='www.douban.com', port=443) 非常经典的忘记关梯子
2） ip 被封了



这个 ip 被封也让我记忆尤深。因为我一开始以为是电脑的 ip 被封，特地跑到了机房配了一通环境，想用机房电脑的 ip，结果发现还是不行。后来才发现被封的是网络 ip，把手机热点开了就可以了。。。
3）port=408 或者 413。Port=408 是因为没有弄 user-agent，413 是因为请求太多次了。诸如此类的错误已经有点记不上来，总之 ip 池才是最终反爬的诀窍。

7.3 其他

还有一些其他的问题，比如在模拟登录时会遇到：
SessionNotCreatedException: Message: session not created: This version of ChromeDriver only supports Chrome version 104 Current browser version is 117.0.5938.149 with binary path C:\Program Files\Google\Chrome\Application\chrome.exe

这是因为 Chrome 的版本和 ChromeDriver 的版本不匹配。这个问题实际上解决起来并不简单，因为 Chrome 会强制自动更新，而最新版的 ChromeDriver 非常难找。因此需要接触 Chrome 的自动更新再安装两者匹配的版本，还需要配好 ChromeDriver 的环境。

根据多次尝试，火狐的浏览器操作要方便一些，因为火狐可以屏蔽自动更新。

八、参考文献

[1]https://blog.csdn.net/hy592070616/article/details/89493672
[2]https://developer.mozilla.org/zh-CN/docs/Learn/Getting_started_with_the_web/HTML_basics
[3]https://zhuanlan.zhihu.com/p/24035574

附录：关键函数的功能简介与使用示例

A. 实验二：低阶爬虫

（1） 完整理解基础代码的功能和结果（即对每一行基础代码添加注释）

```
    # 创建一个不验证 SSL 证书的上下文
    context = ssl._create_unverified_context()

    # 初始化一个空列表，用于存储所有名言的信息
    all_quotes = []

    # 指定要抓取的网址
    url = f'https://quotes.toscrape.com/page/1/' # 目标网址

```

```
    # 打开指定的网页并获取其内容，同时使用上面创建的 SSL 上下文
    page = urlopen(url, context=context) # 请求网页信息

    # 使用 BeautifulSoup 库将网页内容解析为 HTML 结构
    soup = BeautifulSoup(page, 'html.parser')

    # 查找网页中所有具有 class 属性为'quote'的 div 元素，这些元素包含了名言的信息
    quotes = soup.find_all('div', class_='quote')

    # 遍历所有找到的名言元素，并从中提取文本、作者和标签信息
    for quote in quotes:
        # 提取名言文本
        text = quote.find('span', class_='text').text
        # 提取作者名
        author = quote.find('small', class_='author').text
        # 提取标签元素
        tags = quote.find('div', class_='tags').find_all('a')

        # 创建一个空列表 tags_list，用于存储名言的标签
        tags_list = []

        # 遍历名言的标签元素，提取每个标签的文本，并将其添加到 tags_list 列表中
        for tag in tags:
            tags_list.append(tag.text)

        # 创建一个包含名言文本、作者和标签列表的列表，并将其添加到 all_quotes 列表中
        single_quote = [text, author, tags_list]
        all_quotes.append(single_quote)

        # 打印当前已抓取的所有名言信息
        print(all_quotes)
```

（2） 爬取相同网站的 10 个页面内容，并将爬取内容存储在同一个 CSV 格式文件

与高阶爬虫类似，高阶爬虫中全部做了保存到 csv 的处理，因此任务二的代码不再粘贴。

（3） 模拟会员登录过程

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

# 创建一个 WebDriver 对象，这里使用 Chrome
driver = webdriver.Chrome(".\chromedriver.exe")  # 请替换为你的 Chrome WebDriver 的路径
```

```
6.
7.      # 打开目标网页
8.      driver.get("http://quotes.toscrape.com/login")
9.
10.     # 找到用户名和密码输入框，并输入你的用户名和密码
11.     username = "username"
12.     password = "password"
13.
14.     username_field = driver.find_element_by_name("username")
15.     password_field = driver.find_element_by_name("password")
16.
17.     username_field.send_keys(username)
18.     password_field.send_keys(password)
19.
20.     # 提交登录表单
21.     password_field.send_keys(Keys.RETURN)
22.
23.     # 最后，关闭浏览器
24.     driver.quit()
```

（4） 爬取虚拟书店内容和图片，并存储为 CSV 格式，图片单独命名存储。
（5） 爬豆瓣影评信息

B. 实验三：高阶爬虫

（1） 虚拟网站爬取
• 用于清除数据库表里的内容。由于多次爬取相同内容，为了控制变量并且防止主键重复出现错误，在每次爬取前先清一遍。

```
1.      def clear_table():
2.          try:
3.              # Update the table name accordingly
4.              sql = "DELETE FROM books"
5.              mysql_cursor.execute(sql)
6.              mysql_conn.commit()
7.              print("Table cleared successfully")
8.          except Exception as e:
9.              print(f"Error clearing table: {e}")
```

• 获取书本链接。

```
1.      def get_book_url():
2.          base_url = f"http://books.toscrape.com/catalogue/"
3.          # 遍历所有页面
4.          all_links = []
```

```
        for i in range(1,51):
            url = base_url + f'page-'+str(i)+f'.html'
            response = requests.get(url)
            soup = BeautifulSoup(response.text, 'html.parser')
             links = [urljoin(base_url, a['href']) for h3 in soup.find_all('h3') for a in h3.find_all('a', href=True)]
            all_links.extend(links)
        return all_links
```

• 爬取书本信息。

```
    def scrape_book_info(url):
        try:
            response = requests.get(url)
            if response.status_code == 200:
                soup = BeautifulSoup(response.text, 'html.parser')
                title = soup.find('h1').text.strip()
                 image_url = urljoin(url, soup.find('div', class_='item active').find('img')['src'])
                price = soup.find('p', class_='price_color').text.strip()

                rating = soup.find('p', class_='star-rating')['class'][1]

                number = soup.find('h3').find('a')['href'].split('_')[-1].replace("/index.html", "")
                print([number, title, image_url, rating, price])
                download_image(image_url, folder, title)
                save_to_mysql(number, title, image_url, rating, price)
                save_to_csv([number, title, image_url, rating, price])

        except Exception as e:
            print(f"Error scraping {url}: {e}")
```

• 将信息写入数据库。

```
    def save_to_mysql(book_number, title, image_url, rating, price):
        try:
             sql = "INSERT INTO books (number, title, picture, rating, price) VALUES (%s, %s, %s, %s, %s)"
            values = (book_number, title, image_url, rating, price)
            mysql_conn.ping(reconnect=True)
            mysql_cursor.execute(sql, values)
            mysql_conn.commit()
            print("Book information saved to the database")
        except Exception as e:
            print(f"Error saving to the database: {e}")
```

• 将信息写入 csv。

```
    def save_to_csv(data):
        header = ["Number", "Title", "Image URL", "Rating", "Price"]
        csv_filename = 'books.csv'
        if not os.path.exists(csv_filename):
            with open(csv_filename, 'w', newline='', encoding='utf-8') as csvfile:
                csv_writer = csv.writer(csvfile)
                csv_writer.writerow(header)

        with open(csv_filename, 'a', newline='', encoding='utf-8') as csvfile:
            csv_writer = csv.writer(csvfile)
            csv_writer.writerow(data)
```

• 保存图片到本地文件夹。

```
    def download_image(image_url, folder, filename):
        # 使用正则表达式去除文件名中的非法字符并截断为较短的长度
        image_name = re.sub(r'[\/:*?"<>|]', '_', filename)[:100] + ".jpg"
        os.makedirs(folder, exist_ok=True)
        response = requests.get(image_url)
        with open(os.path.join(folder, f"{image_name}"), 'wb') as f:
            f.write(response.content)
```

• 使用多进程爬取。

```
    if __name__ == "__main__":
        # 初始化数据库连接
        mysql_conn = pymysql.connect(host='localhost', user='root', password='mysqlpassword', database='books_db')
        mysql_cursor = mysql_conn.cursor()

        folder = "./task3_picture"
        # 由于过程重复爬取，故每次爬取时先清除表的内容
        clear_table()
        urls = get_book_url()
        num_threads = 1
        with ThreadPoolExecutor(max_workers=num_threads) as executor:
            executor.map(scrape_book_info, urls)
        # 关闭数据库连接
        mysql_cursor.close()
        mysql_conn.close()
```

（2） 豆瓣影评爬取

• 获取 ip 地址

```
    def get_ip():
        url = "http://api.tianqiip.com/getip?secret=cx1xlfx52o8s1rdq&num=5&type=txt&lb=%5Cn®ion=440000&port=1&time=5&mr=1&sign=15fb650d0f872b891e37db13014c89d7"
        res = requests.get(url)
        print(res.text)

        if res.status_code==200:
            prox = res.text
        prox = res.text.split("\n")[:-1]
        proxy = [{'http':'http://'+i} for i in prox]
        # 使用前可判断一下是否可用，也可能返回其他 state code，具体就依据具体情况解决
        print(proxy)
        ts = "http://www.baidu.com/"
        for j in proxy:
            try:
                r = requests.get(ts,proxies=j)
            except:
                print("ip 不可用")
        return prox
```

• 用 ip 地址对页面发起请求。

```
    def download_pages(url):
        print(url)
        ip_list = get_ip()
        headers = {
                'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117.0.0.0 Safari/537.36',

                'cookie': 'XXX',
                'Connection':'close'}
        ip = random.choice(ip_list)
        ip = '125.89.44.123:40018'
        proxies = {"http":f"http://{ip}"}
        try:
            response = requests.get(url, headers=headers, stream=True, verify=False, proxies=proxies, timeout=2)
            response.raise_for_status()  # Raise an HTTPError for bad responses
            return response.content.decode('utf-8')  # Decode the bytes to a string
        except requests.RequestException as e:
```

```
            print(f"Error downloading page {url}: {e}")
            return None
```

• 获取全部的影评链接

```
    def get_review_urls(doulist_url):
        urls = []
         movie_links = get_doulist_movie_links(doulist_url, num_links=10)

        for movie_link in movie_links:
            reviews_links = get_reviews_links(movie_link)
            urls.extend(reviews_links)
        return urls

    def get_reviews_links(movie_url):
        reviews_url = f"{movie_url}reviews?start="
        reviews_links = []
        start = 0
        while True:
            response = download_pages(f"{reviews_url}{start}")
            # print(f"{reviews_url}{start}")
            soup = BeautifulSoup(response, 'html.parser')

            # Extract href from review links
            review_elements = soup.find_all('h2')
             current_review_links = [element.find('a')['href'] for elemen
t in review_elements]
            # If no more reviews, break the loop
            if not current_review_links:
                break
            reviews_links.extend(current_review_links)
             start += 20  # Assuming 20 reviews per page, adjust if neede
d
        return reviews_links

    def get_doulist_movie_links(doulist_url, num_links=10):
        response = download_pages(doulist_url)
        # print(response)
        soup = BeautifulSoup(response, 'html.parser')

        # Extract href from class="post" elements
        post_elements = soup.find_all('div', class_='post')
         movie_links = [element.find('a')['href'] for element in post_ele
ments]

```

```
        return movie_links[:num_links]
```

• 爬取某影评链接的影评信息并储存到 SQL 和 csv 里。

```
    def scrape_movie_info(url):
        try:
            response = download_pages(url)
            soup = BeautifulSoup(response, 'html.parser')
             review = soup.find('div', class_='review-content').get_text(separator='\n', strip=True)
             author = soup.find('header').find('a').find('span').get_text(strip=True)
             movie = soup.find('header').find_all('a')[1].get_text(strip=True)
            save_to_mysql(movie, author, review)
            save_to_csv([movie, author, review])
            # print(author,movie,review)

        except Exception as e:
            print(f"Error scraping {url}: {e}")
```

（储存到 SQL 和 CSV 的代码与上面类似）

• 主函数与多进程。

```
    if __name__ == "__main__":
        doulist_url = "https://www.douban.com/doulist/5257286/"
        # 初始化数据库连接
         mysql_conn = pymysql.connect(host='localhost', user='root', password='mysqlpassword', database='books_db')
        mysql_cursor = mysql_conn.cursor()
        # 获取全部的影评链接
        urls = get_review_urls(doulist_url)
        # print(urls)
        # 多进程
        num_threads = 4
        with ThreadPoolExecutor(max_workers=num_threads) as executor:
            executor.map(scrape_movie_info, urls)
        # 关闭数据库连接
        mysql_cursor.close()
        mysql_conn.close()
```