

**Bookstore——网上购书功能网站后端**

|  |  |  |  |
| --- | --- | --- | --- |
| **课程** | **当代数据管理系统** | | |
| **姓名** | **唐小卉** | **刘钊瑄** | **仲韦萱** |
| **学号** | **10215501437** |  |  |

**2023年 11 月**

目录

[Bookstore——网上购书功能网站后端 1](#_Toc28973)

[1、 实验要求 3](#_Toc28728)

[2、 项目运行 4](#_Toc18724)

[3、 数据库设计 4](#_Toc3685)

[4、 功能实现 4](#_Toc6151)

[4.1 用户功能 4](#_Toc21152)

[4.1.1 token与jwt 4](#_Toc12612)

[4.1.2 register登录 5](#_Toc27452)

[4.1.3 check password检查密码 5](#_Toc30743)

[4.1.4 logout登出 6](#_Toc11246)

[4.1.5 unregister注销 6](#_Toc8482)

[4.1.6 change password更改密码 7](#_Toc28552)

[4.2 卖家功能 7](#_Toc6597)

[4.3 买家功能 7](#_Toc15480)

[4.4 发货收货 7](#_Toc2833)

[4.5 搜索图书 7](#_Toc27232)

[4.6 订单状态 17](#_Toc23957)

[5、 测试 17](#_Toc14655)

[5.1 test\_sh 17](#_Toc3158)

[5.2 关于搜索功能的pytest检验 17](#_Toc3838)

[5.3 测试结果与覆盖率 18](#_Toc18603)

[6、 小组分工 18](#_Toc30313)

[7、 项目说明 19](#_Toc15828)

# 实验要求

实现一个提供网上购书功能的网站后端。

网站支持书商在上面开商店，购买者可以通过网站购买。

买家和卖家都可以注册自己的账号。

一个卖家可以开一个或多个网上商店，

买家可以为自已的账户充值，在任意商店购买图书。

支持 下单->付款->发货->收货 流程。

1.实现对应接口的功能，见项目的doc文件夹下面的.md文件描述 （60%）

其中包括：

1)用户权限接口，如注册、登录、登出、注销

2)买家用户接口，如充值、下单、付款

3)卖家用户接口，如创建店铺、填加书籍信息及描述、增加库存

通过对应的功能测试，所有test case都pass

2.为项目添加其它功能 ：（40%）

1)实现后续的流程

发货 -> 收货

2)搜索图书

用户可以通过关键字搜索，参数化的搜索方式；

如搜索范围包括，题目，标签，目录，内容；全站搜索或是当前店铺搜索。

如果显示结果较大，需要分页

(使用全文索引优化查找)

3)订单状态，订单查询和取消定单

用户可以查自已的历史订单，用户也可以取消订单。

取消定单可由买家主动地取消定单，或者买家下单后，经过一段时间超时仍未付款，定单也会自动取消。

# 项目运行

# 数据库设计

# 功能实现

## 4.1 用户功能

注册功能是在原有代码上进行修改的。一共有11个内置函数。

### 4.1.1 token与jwt

可以看到在用户功能板块，利用了token和jwt来确保用户的账户安全。

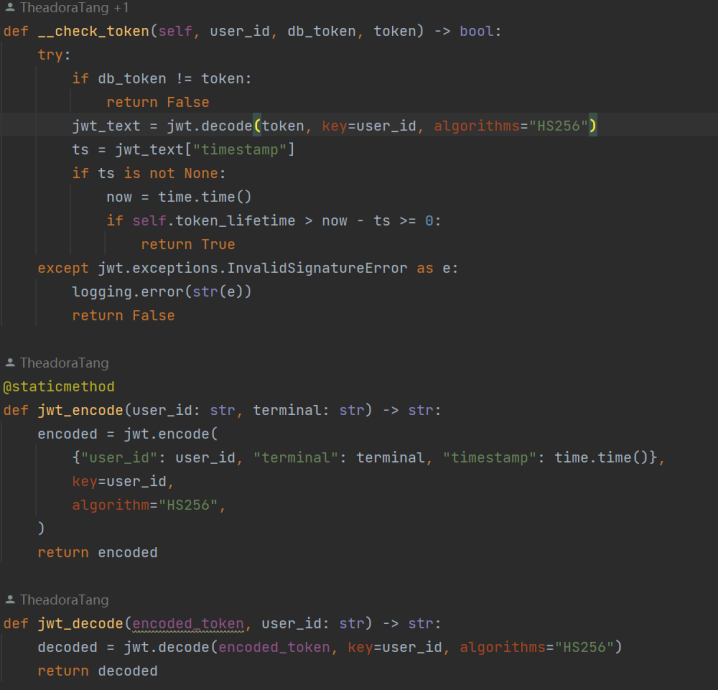
Token，即令牌，是一种用于验证用户身份和授权的技术。 token可以理解为一个字符串，里面包含了足够的信息，以便于服务器识别和验证用户。 token的基本原理是，服务器会为每个已验证的用户生成一个唯一的、有时效性的令牌。 当用户进行一些需要认证的操作时，他们需要携带这个token。而jwt也是经常作为一种安全的token使用。

下面的函数分别用于：

Check\_token:用来检测用户的token，识别和验证用户信息。

Jwt\_encode:用来进行jwt加密。

Jwt\_decode:用来进行jwt解密。

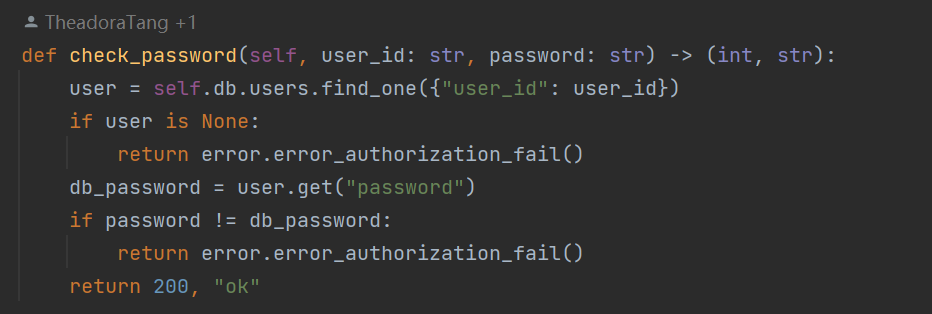


### 4.1.2 register登录



首先检测用户是否存在，如果已经存在，根据error中的定义返回错误的code和message，如果用户不存在，我们将用户的信息存储到bookstore数据库的users collection中，同时返回code和message。

### 4.1.3 check password检查密码



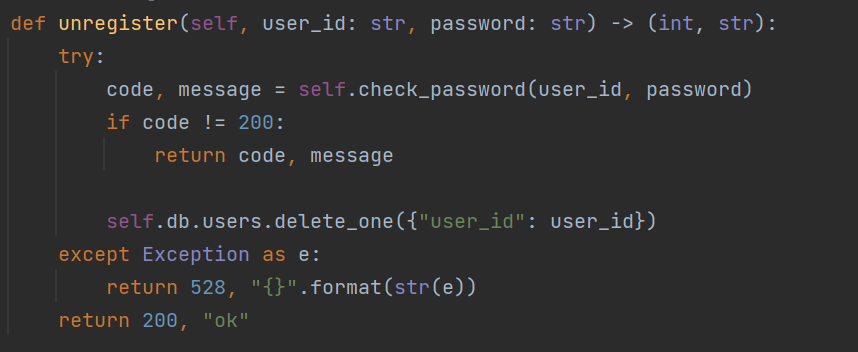
该函数是用来检测用户的密码是否正确，如果不正确返回对应error的code和message，如果正确的话，返回200。

### 4.1.4 logout登出



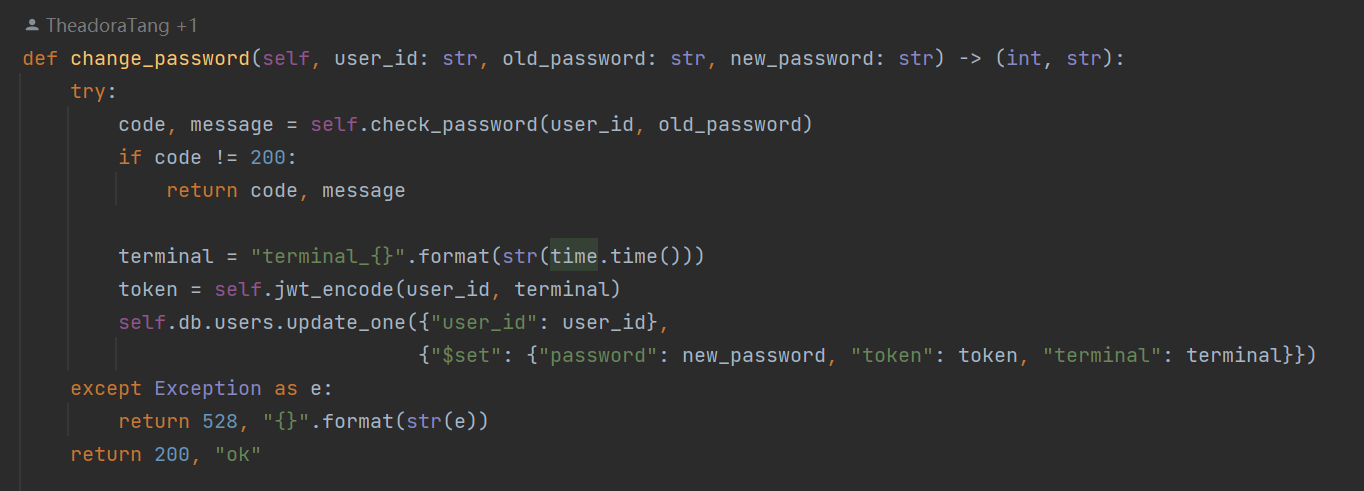
首先利用chech token进行服务器识别和验证用户，然后更新terminal和token，填入到数据库中。

### 4.1.5 unregister注销



首先检测用户输入的密码是否正确，如果验证用户身份成功，则从users中删除有关该用户的所有信息。

### 4.1.6 change password更改密码



首先检测用户输入的密码是否正确，如果验证用户身份成功，则在users的数据原有基础上对password进行更新（terminal和token也同时更新）。

## 4.2 卖家功能

## 4.3 买家功能

## 4.4 发货收货

## 4.5 搜索图书

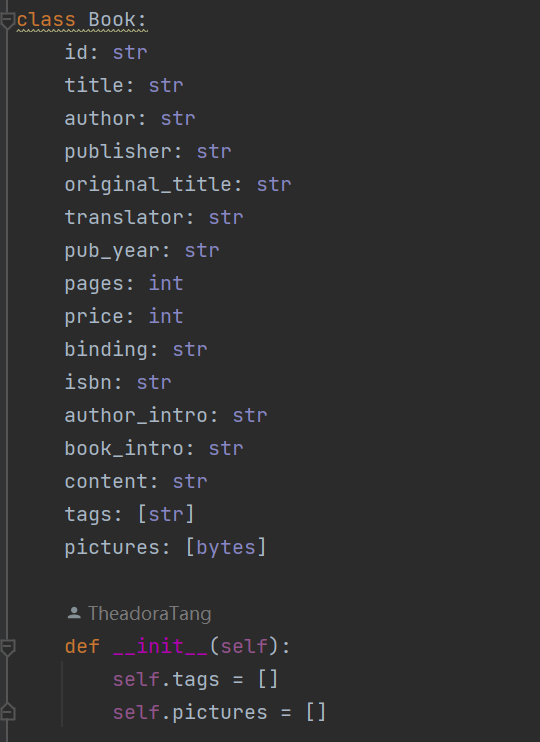
首先，将book\_lx.db中的信息存储进数据库中，我们在init.py中进行存储的操作。

为了将数据从SQLite数据库中提取，并插入到MongoDB数据库中。我们进行了如下的操作：

1. Book类：

用于表示书籍信息。它包含了书籍的各种属性，例如id、title、author、publisher等等。这些属性对应了书籍的各种信息。

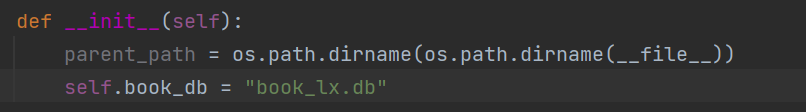
\_\_init\_\_ 方法用于初始化一个Book对象。



2.BookDB类：

用于与SQLite数据库交互以获取书籍信息。

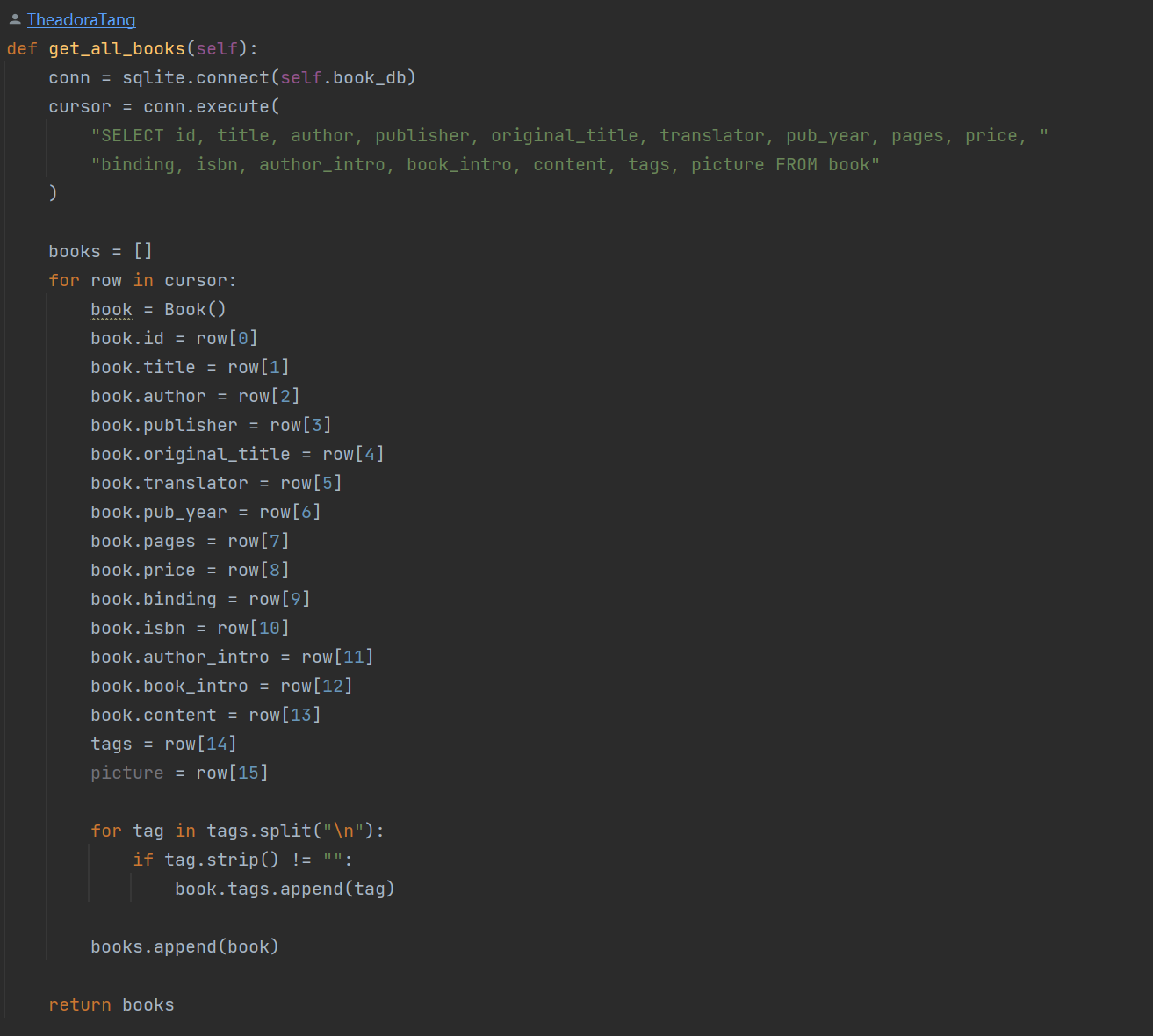
\_\_init\_\_方法初始化了SQLite数据库的路径。



3.\*get\_all\_books方法：

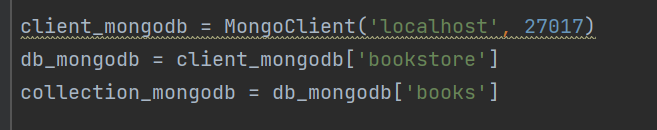
`get\_all\_books` 方法用于从SQLite数据库中提取所有书籍信息。

连接到SQLite数据库，执行一个SQL查询，然后将查询结果转化为Book对象的列表。



4.连接到MongoDB：

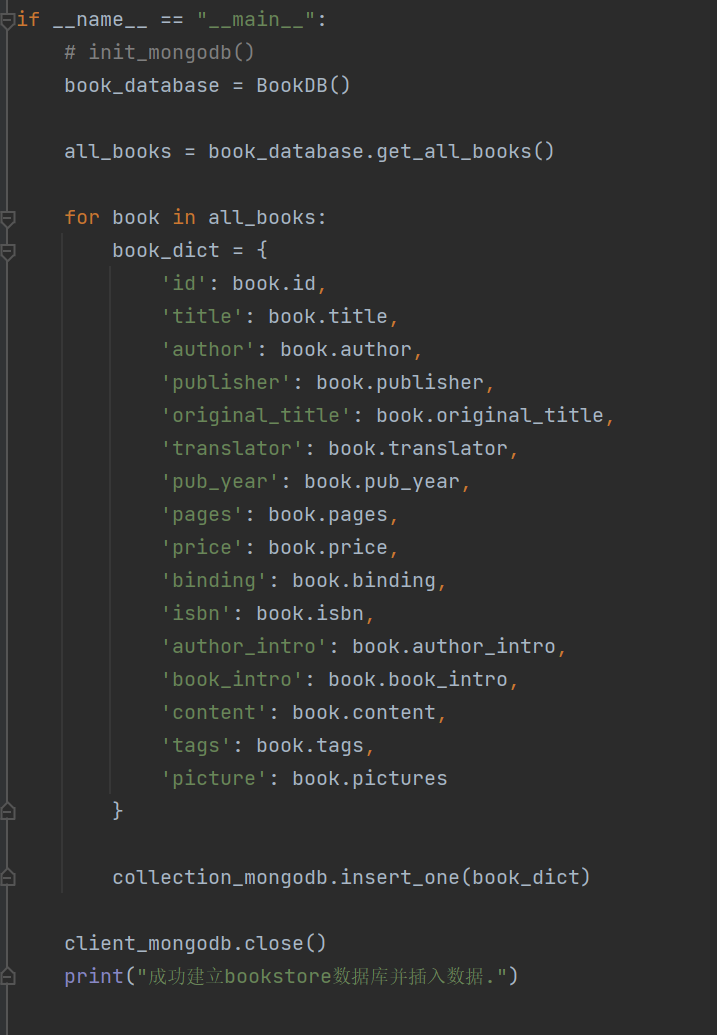
创建了一个MongoDB的客户端，连接到'bookstore'的数据库，并获取名为books的collection。



5. 主程序：

`book\_database = BookDB()` 创建一个BookDB对象，用于与SQLite数据库交互。

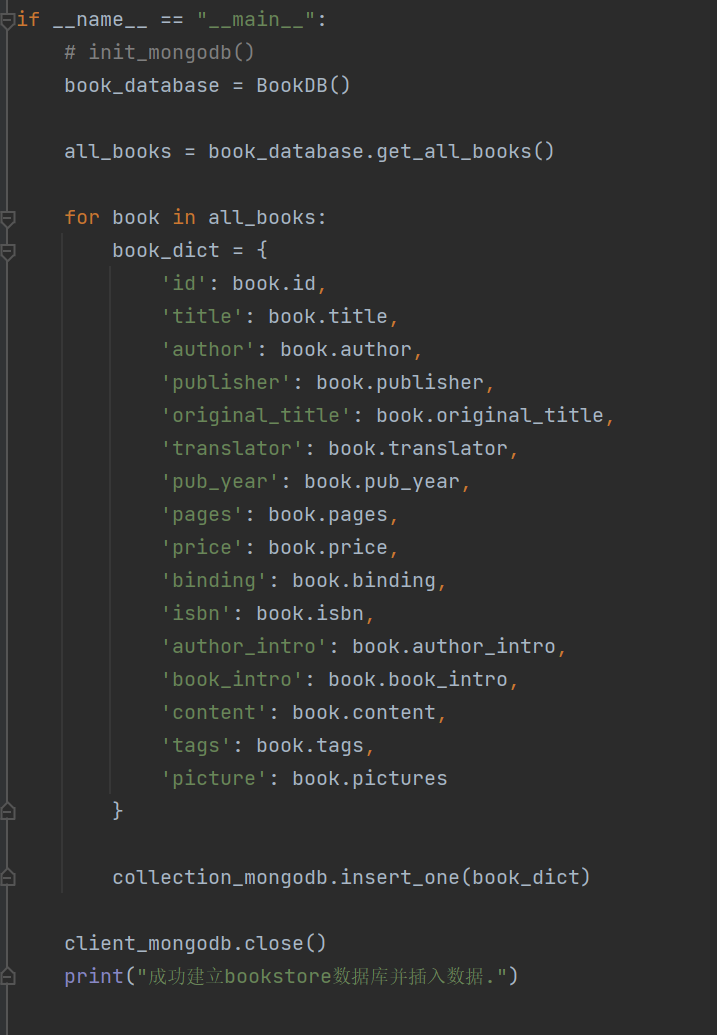
`all\_books = book\_database.get\_all\_books()` 调用`get\_all\_books` 方法获取所有书籍信息，并将其存储在`all\_books`列表中。



6.将数据插入MongoDB：

使用一个循环遍历`all\_books`列表，将每本书的信息存储在一个字典`book\_dict`中。

最后，使用`collection\_mongodb.insert\_one(book\_dict)` 将每本书的字典插入到MongoDB的'books'集合中。



7.关闭MongoDB连接：使用`client\_mongodb.close()`关闭了MongoDB的连接。

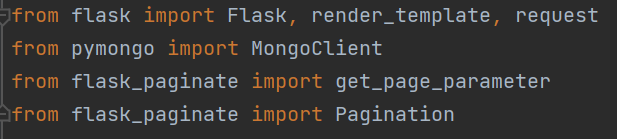
接下来，实现搜索功能。为了更好的显示搜索功能，本次项目中还特意制作了一个简易

的前端，用来可视化搜索的结果。

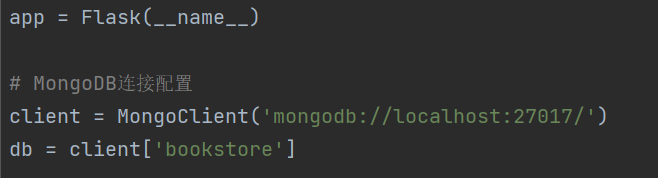
本次项目中实现的书店搜索引擎允许用户在书店数据库中进行搜索，并提供分页功能，以便在多个搜索结果中进行浏览。

下面是代码的主要逻辑和功能介绍：

1. 首先，应用程序导入了必要的库，包括Flask用于Web应用程序的构建，pymongo用于与MongoDB数据库进行交互，以及Flask-Paginate用于分页。



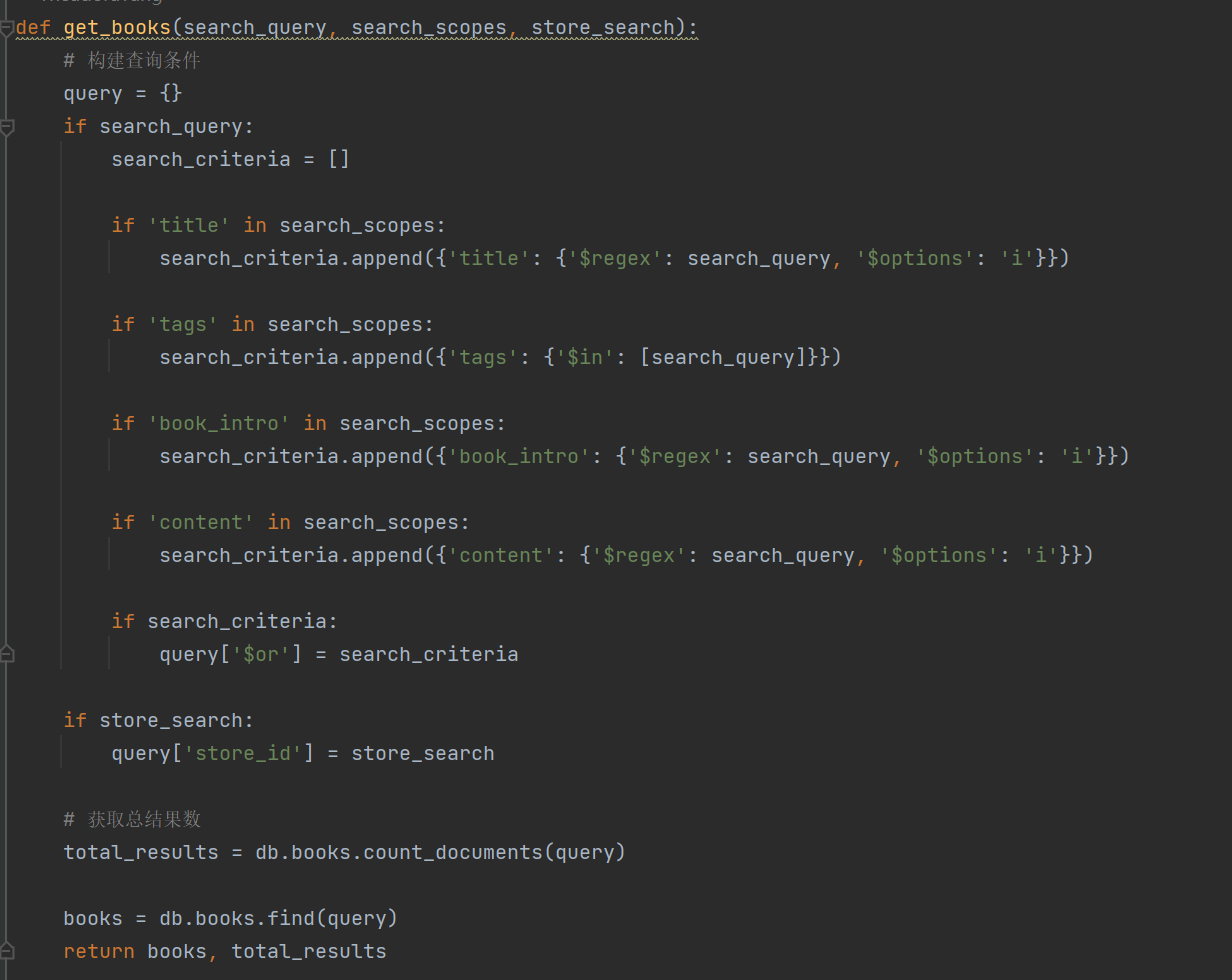
1. 创建一个Flask实例，并配置MongoDB的连接，指向本地的MongoDB服务器，并选择了"bookstore"的数据库。



3. RESULTS\_PER\_PAGE变量定义了每页显示的结果数量（10个），用于分页功能。

4. get\_books函数用于执行实际的书籍搜索操作。它接受搜索查询、搜索范围和是否进行店铺搜索作为参数，并构建MongoDB查询条件来获取匹配的书籍。

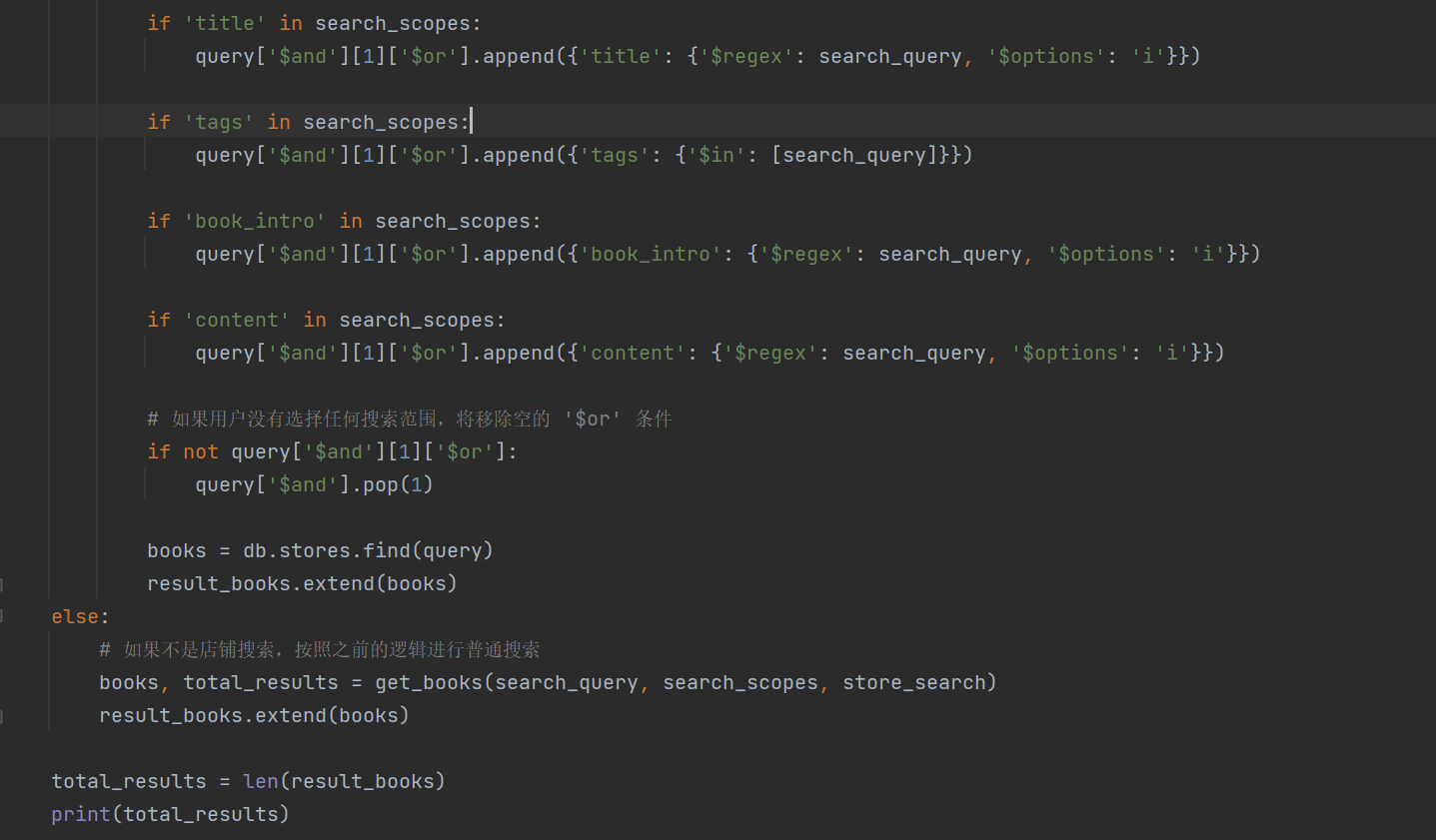
在不选择在店铺里搜索的情况下，根据用户勾选的选项，在范围内进行关键词搜索，函数返回结果为查询出的所有书籍以及结果的数量（用于分页）。



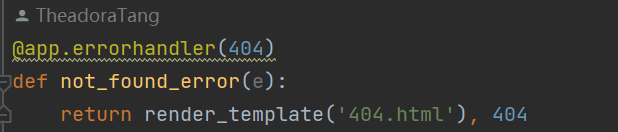
5. `/`路由是应用程序的主页，用于接受用户输入的搜索查询、搜索范围和店铺搜索选项。它执行以下操作：

* 从请求中获取搜索查询、搜索范围和店铺搜索选项。
* 初始化一个结果列表result\_books。
* 如果用户选择了店铺搜索并且选择了 "Store" 选项，应用程序将在店铺数据中查找符合搜索关键词的书本。主要逻辑为：先筛选需要搜索的店铺，之后在筛选的query中继续添加用户输入的搜索范围和查询的条件，利用AND连接店铺搜索与其他条件，利用OR并列其他条件。
* 否则，应用程序按照通常的搜索逻辑进行普通搜索。
* 计算总结果数。
* 根据当前页数，计算本页要显示的书籍的起始和结束索引。
* 使用Flask-Paginate扩展创建分页对象，并将结果渲染到模板中。



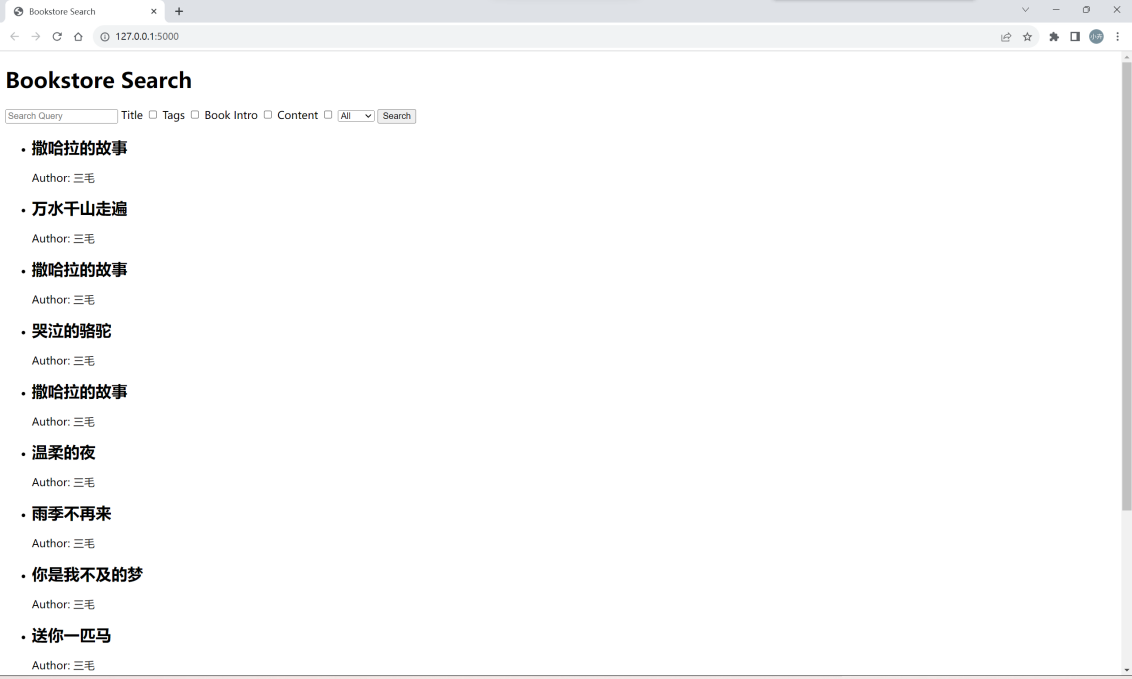


1. `/`路由还包含了一个错误处理程序，用于处理404错误，返回一个404页面。假如没能找到符合条件的搜索结果，则返回404（主要用于pytest检测）。



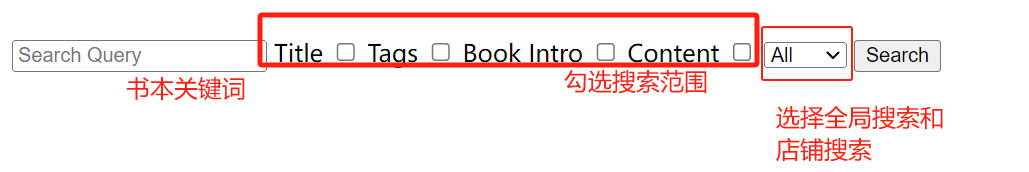
关于搜索引擎的前端代码（index.html），在报告中就不再详细解释，我们直接观察结果即可。

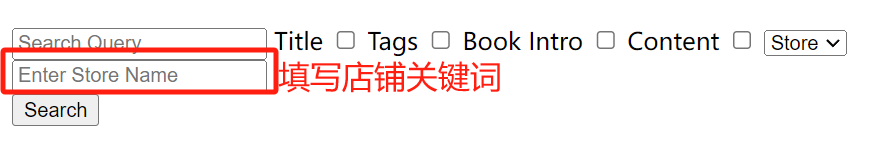
1. 运行app.py，打开网页的主页面。主页面会显示所有书籍的名字以及作者。最上方的是搜索栏，用于获取用户的搜索条件。最下方是分页的链接，可以根据自己的需求跳选到需要的页面。

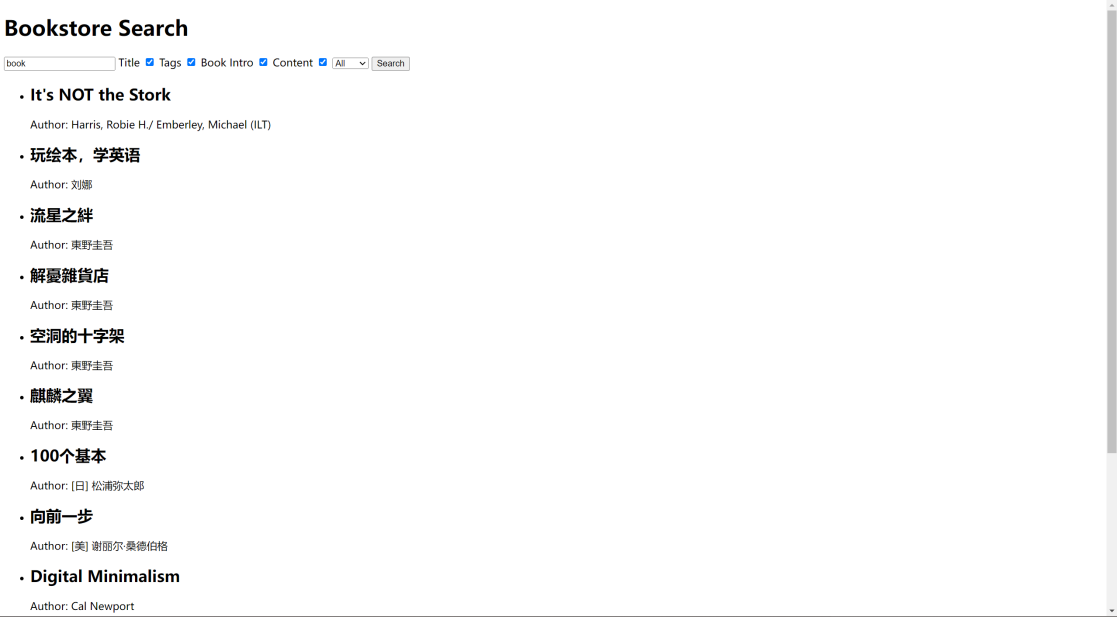




1. 我们尝试在输入框中勾选自己需要的选项。当勾选了进行“店铺搜索”时，会弹出填写店铺关键词的位置。（注意，如果已经勾选了Store但没有填写店铺关键词，默认进行全局搜索；当已经在第二个输入框中输入了内容但是仍然想进行全局搜索，需要将输入框里的内容清空。）

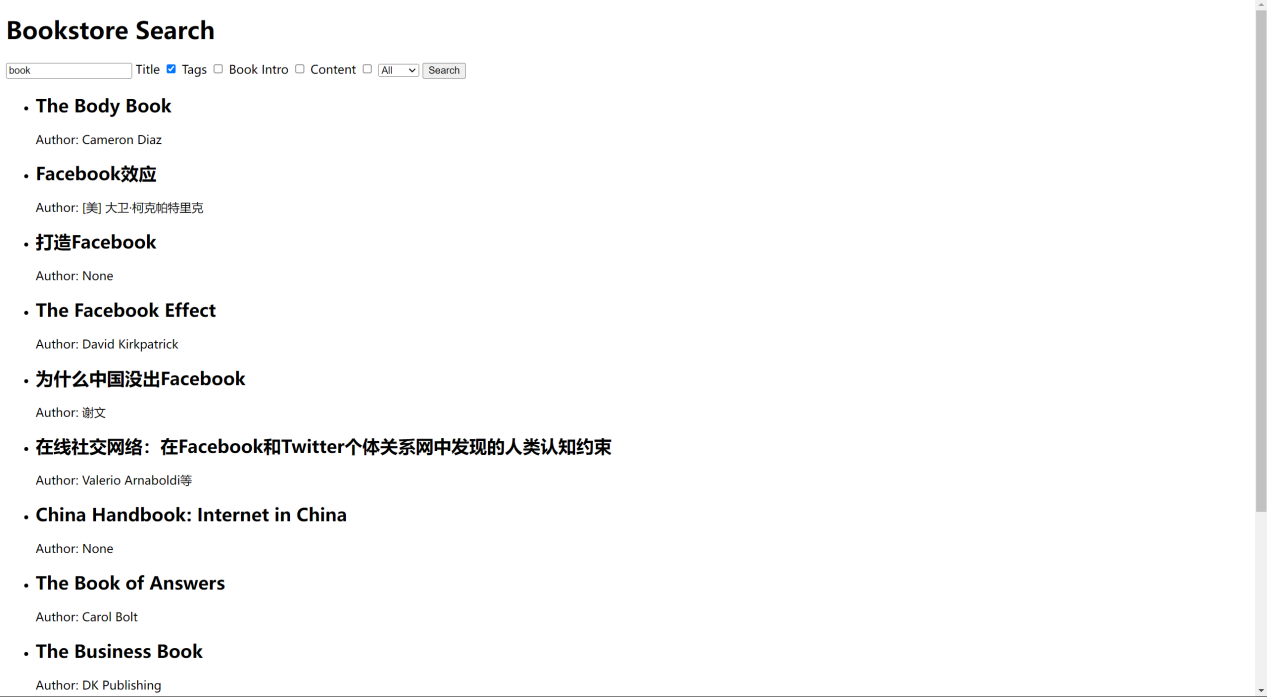




以全局搜索为例，我们搜索在题目、标签、简介、内容（至少一个选项中）包含关键词为book的数据。结果如下：  




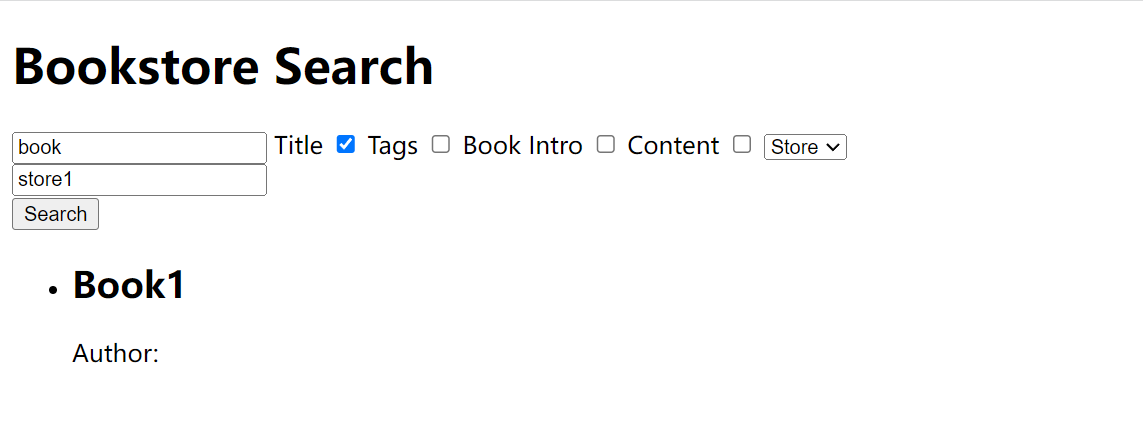
接下来只勾选‘title’，看一下结果的区别。





我们从翻页的链接可以明显看出搜索结果的数量上有差距，而且第二次搜索中，明显能看出搜索结果的题目中都含有book（不分大小写）。

接下来，再检验一下店铺搜索。我们搜索store1店铺中题目中含有book的书目。



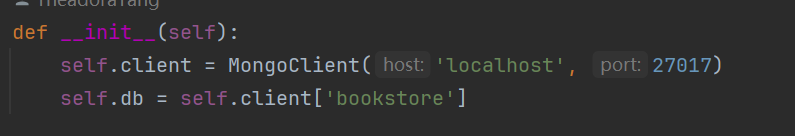
同时也可以看出，当搜索结果不足10条时，不会出现翻页链接，很好的实现了分页功能。

为了通过pytest，小组又另外撰写了符合仓库结构的search代码，整体逻辑与上述代码基本相同：

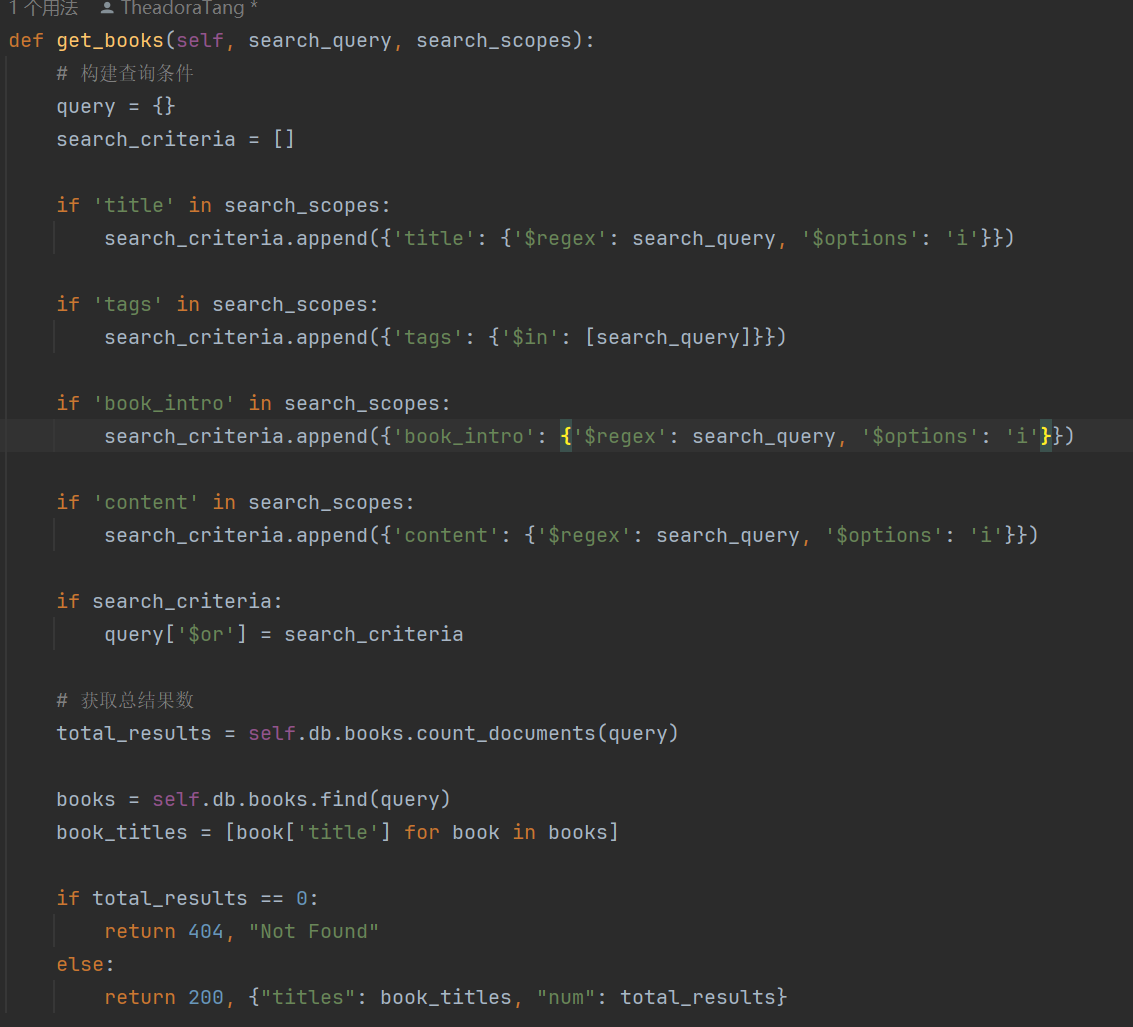
这段代码是一个 Python 类 `SearchBooks`，旨在实现书店搜索功能。以下是该代码的逻辑和功能介绍：

`SearchBooks`类：

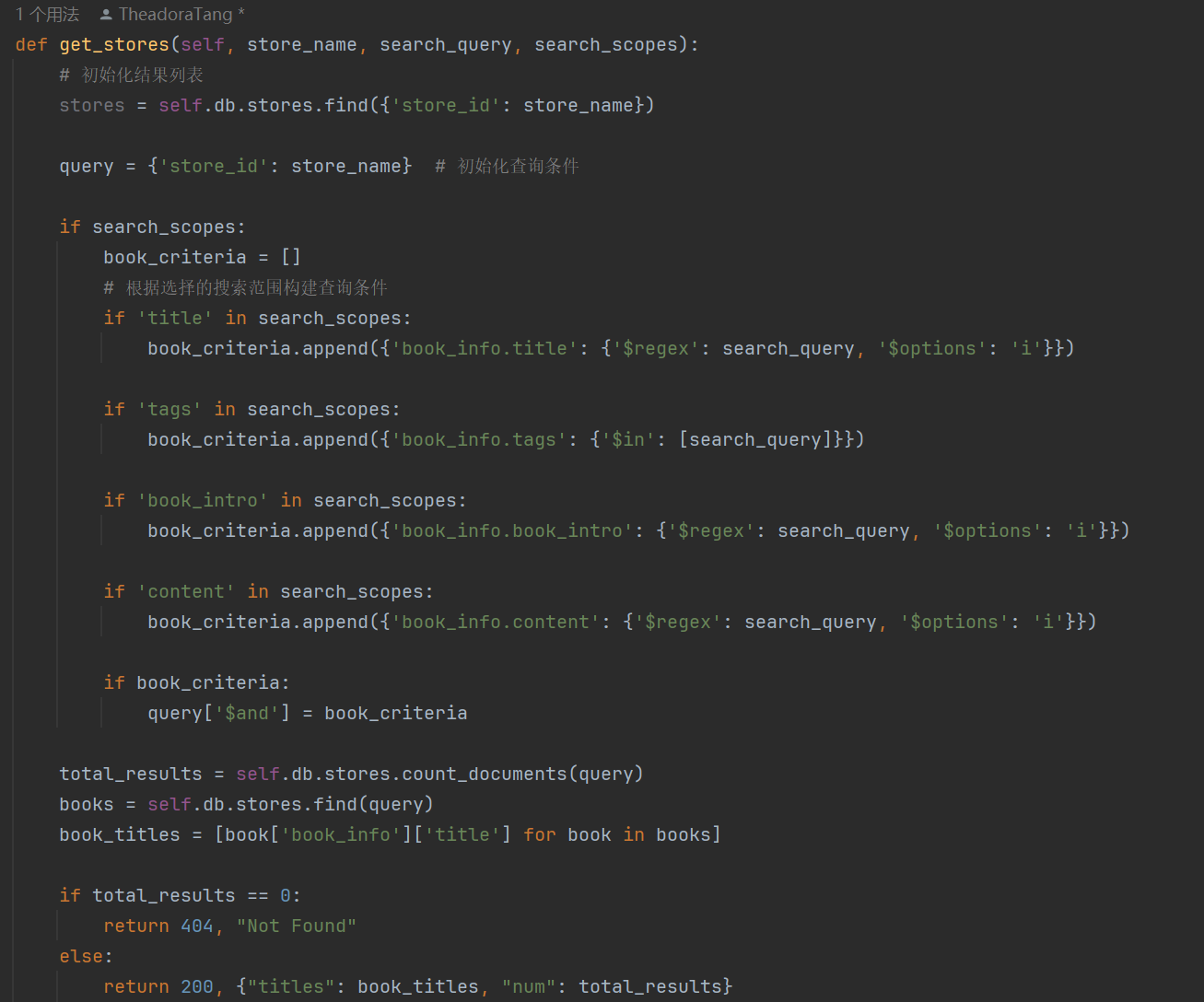
- `\_\_init\_\_` 方法：在初始化对象时，它创建了一个 MongoDB 客户端连接到本地的 MongoDB 数据库，并选择名为 'bookstore' 的数据库作为工作数据库。



- `get\_books` 方法：用于搜索书籍。根据传入的搜索关键词 (`search\_query`) 和搜索范围 (`search\_scopes`) 构建 MongoDB 查询条件，然后查询数据库中的书籍信息。查询条件会根据搜索范围的不同包括书名、标签、书籍简介和内容，使用正则表达式进行模糊匹配。如果有匹配的书籍，它返回 HTTP 状态码 200 和包含书籍标题列表和总结果数的响应。如果没有匹配的书籍，返回 HTTP 状态码 404 和 "Not Found" 的响应。



- `get\_stores` 方法：用于搜索店铺内的书籍。根据传入的店铺名称 (`store\_name`)、搜索关键词 (`search\_query`) 和搜索范围 (`search\_scopes`) 构建 MongoDB 查询条件，然后查询数据库中的店铺信息。查询条件会根据搜索范围的不同包括书名、标签、书籍简介和内容，使用正则表达式进行模糊匹配。如果有匹配的书籍，它返回 HTTP 状态码 200 和包含书籍标题列表和总结果数的响应。如果没有匹配的书籍，返回 HTTP 状态码 404 和 "Not Found" 的响应。



## 4.6 订单状态

# 测试

## 5.1 test\_sh

原有的pytest代码没有进行修改，直接使用。

## 5.2 关于搜索功能的pytest检验

为了检验图书搜索功能是否正确，小组后期自行编写了test\_research.py代码。以下是这段代码的逻辑和功能介绍：

1. 定义测试类 `TestSearch`：

这个类包含了一系列测试用例，用于测试书店搜索功能。

2. `@pytest.fixture(autouse=True)` 装饰器：

-`pre\_run\_initialization` 方法被标记为自动使用的 pytest fixture，意味着它会在每个测试方法之前自动执行。

在这个 fixture 方法中，进行了一些初始化工作，包括创建用于检测的书本和店铺名称，以及创建了一个 `search\_books.Search` 实例、设置了搜索关键词（`search\_query`）、搜索范围（`search\_scopes`）和一个店铺名称（`store\_name`）。

3. 测试用例：

`test\_search\_books` 方法测试书店搜索功能。它使用预定义的搜索关键词和搜索范围，向书店发送搜索请求，并检查返回的状态码是否为 200（成功）。

`test\_search\_books\_wrong` 方法测试错误的搜索情况。它使用一个无效的搜索关键词（'txh'）发送搜索请求，然后检查返回的状态码是否为 404（未找到）。

`test\_search\_stores` 方法测试店铺搜索功能。它使用预定义的店铺名称、搜索关键词和搜索范围，向书店发送店铺搜索请求，并检查返回的状态码是否为 200（成功）。

`test\_search\_stores\_wrong` 方法测试错误的店铺搜索情况。它使用一个不存在的店铺名称（'store111'）发送店铺搜索请求，然后检查返回的状态码是否为 404（未找到）。

`test\_search\_books\_no\_scope`方法测试在没有指定搜索范围的情况下执行图书搜索的行为。并检查返回的状态码是否为 200（成功）。（在没有指定搜索范围的情况下默认进行全局搜索）

`test\_search\_books\_no\_query\_and\_scope`方法测试了在没有指定搜索查询和搜索范

围的情况下执行图书搜索的行为。并检查返回的状态码是否为 200（成功）。

`test\_search\_books\_single\_scope\_no\_match`方法测试了在指定了搜索范围但搜索不

到匹配项的情况下执行图书搜索的行为。然后检查返回的状态码是否为 404（未找到）。

`test\_search\_books\_multiple\_scopes\_no\_match`方法测试了在指定了多个搜索范围

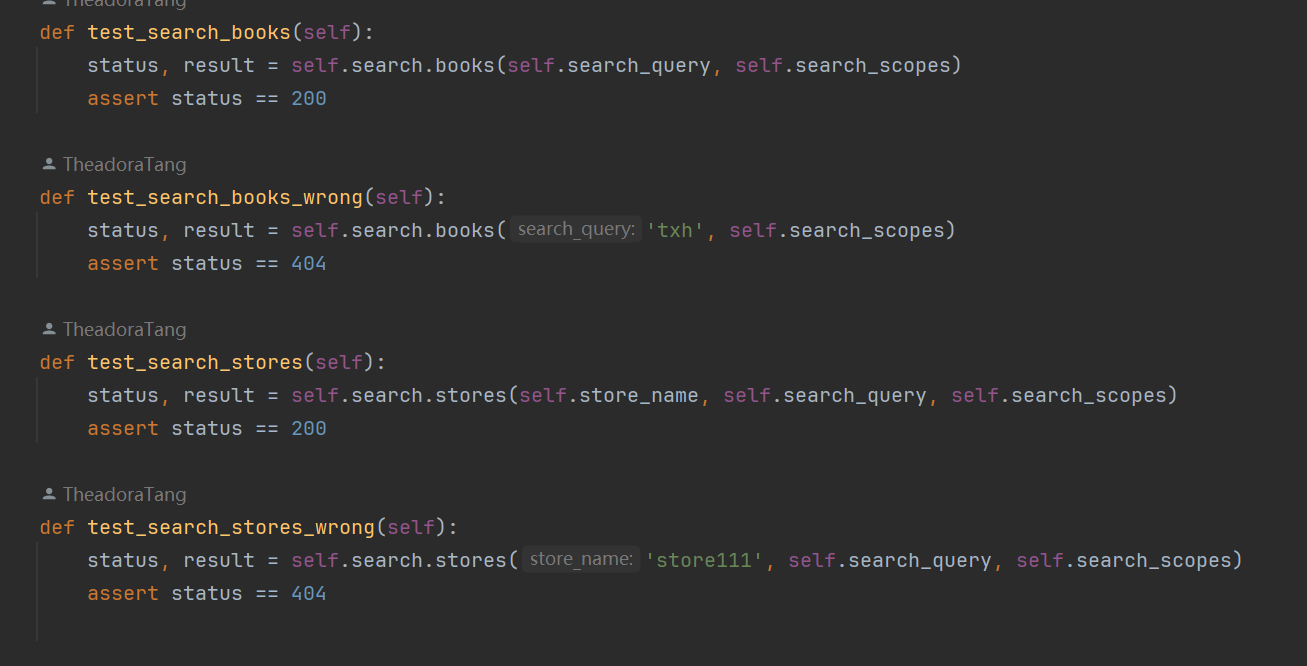
但搜索不到匹配项的情况下执行图书搜索的行为。然后检查返回的状态码是否为 404（未找到）。

`test\_search\_stores\_single\_store\_no\_match`:方法测试了在指定了商店名称和搜索范

围但搜索不到匹配项的情况下执行商店搜索的行为。然后检查返回的状态码是否为 404（未找到）。

`test\_search\_stores\_multiple\_stores\_no\_match`:这个测试用例测试了在指定了多个

商店名称和搜索范围但搜索不到匹配项的情况下执行商店搜索的行为。然后检查返回的状态码是否为 404（未找到）。





## 5.3 测试结果与覆盖率

经测试，所有pytest均已通过，覆盖率

# 小组分工

本项目需要实现的功能有：  
 1)用户权限接口，如注册、登录、登出、注销

2)买家用户接口，如充值、下单、付款

3)卖家用户接口，如创建店铺、填加书籍信息及描述、增加库存

4)实现后续的流程 :发货 -> 收货

5)搜索图书

6)订单状态，订单查询和取消定单

* 唐小卉：负责1、5功能开发，撰写相关的实验报告，负责5功能的前端实现以及pytest检测，数据库的初始化和SQLite与MongoDB数据库之间的信息传递。
* 刘钊瑄：
* 仲韦萱：

# 项目说明

本次项目中小组成员利用Github仓库进行项目开发，详情可见：https://github.com/TheadoraTang/Bookstore。