

课程名称：当代数据管理系统	指导教师：周烜	上机实践名称：Bookstore
姓名：郭夏辉	学号：10211900416	年级：2022
姓名：包亦晟	学号：10215501451	年级：2021
姓名：朱天祥	学号：10225501461	年级：2022

## 1.实验目的及要求

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，买家可以为自己的账户充值，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 用户可以查自己的历史订单，用户也可以取消订单。

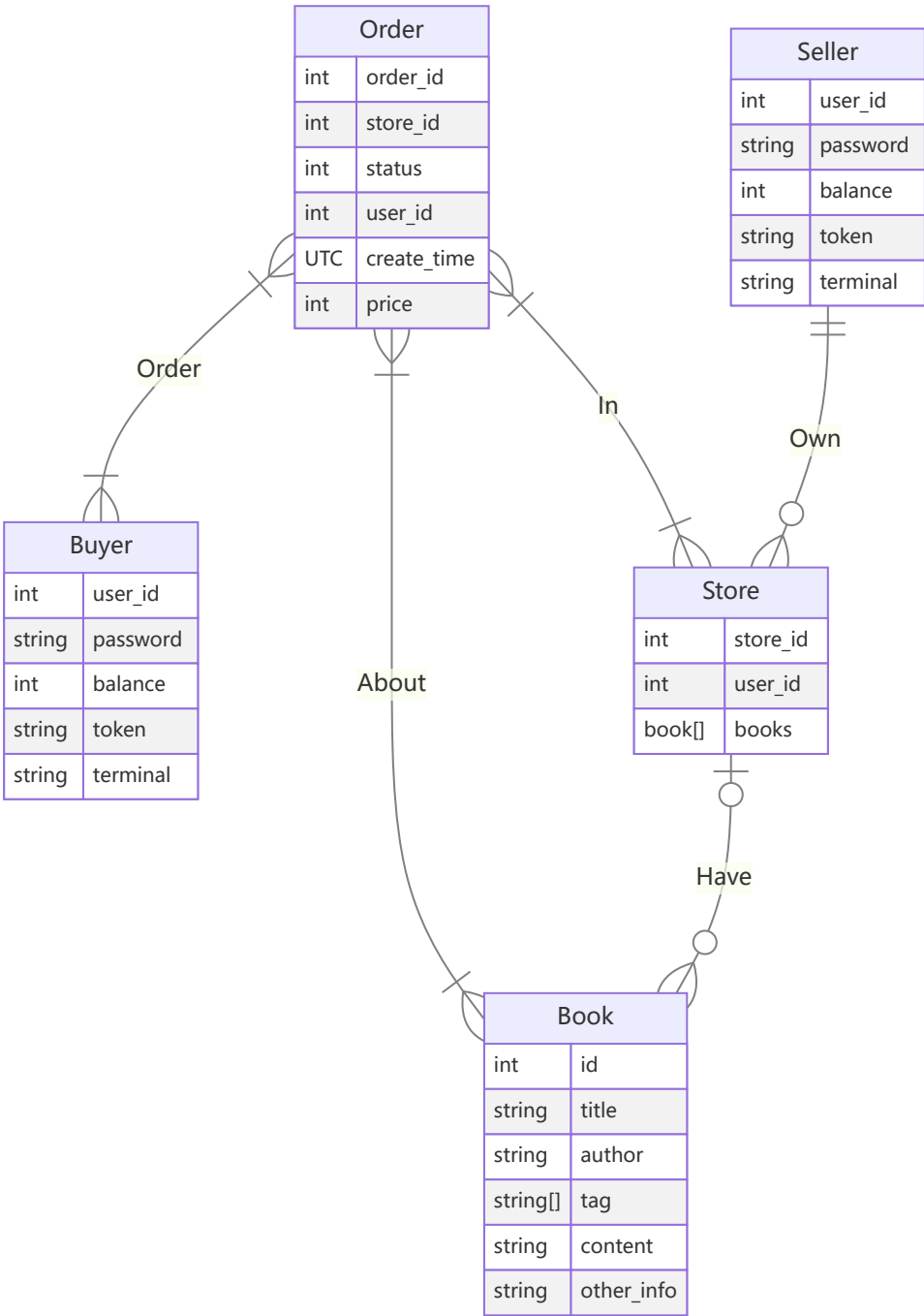
取消订单可由买家主动地取消定单，或者买家下单后，经过一段时间超时仍未付款，订单也会自动取消。

**具体的一些要求：**

- 1.bookstore 文件夹是该项目的 demo，采用 Flask 后端框架与 SQLite 数据库，实现了前60%功能以及对应的测试用例代码。**要求大家创建本地 MongoDB 数据库，将 bookstore/fe/data/book.db 中的内容以合适的形式存入本地数据库，后续所有数据读写都在本地的 MongoDB 数据库中进行。**书本的内容可自行构造一批，也可参从网盘下载，下载地址为：[https://pan.baidu.com/s/1bjCOW8Z5N\\_ClcqU54Pdt8g](https://pan.baidu.com/s/1bjCOW8Z5N_ClcqU54Pdt8g) 提取码：hj6q
- 2.在完成前60%功能的基础上，继续实现后40%功能，要有接口、后端逻辑实现、数据库操作、代码测试。对所有接口都要写 test case，通过测试并计算测试覆盖率（尽量提高测试覆盖率）。
- 3.尽量使用索引，对程序与数据库执行的性能有考量
- 4.尽量使用 git 等版本管理工具
- 5.不需要实现界面，只需通过代码测试体现功能与正确性

## 2.文档数据库的设计

### 2.1数据库的逻辑设计



以上这个便是bookstore整体的业务逻辑图。

### 2.2数据库的结构设计

根据数据库的业务逻辑，为了展现文档数据库相对于关系数据库而言**模型灵活、凸显关系、便于查询**的优势，我们最开始设想的文档集设计是——user,store,book,order.

但是在之后完成项目的过程中，我们发现只有面对**查询历史订单**时，我们才真的需要知道这笔订单具体的内容是什么；然而其他关于订单的场景（比如支付、发书、收书）我们并不需要这么多的信息，只要知道这是哪笔订单就行。因此，我们将order的一些信息拆分了出来，设计了两个文档集——order和order\_detail。

还有一个问题，就是对于实验所给的sqlite代码而言，它设置了一个user\_store表，但是这个在文档数据库中真的适用吗？经过小组成员的讨论和思考，我们觉得这个表的存在没有意义——store,user作为两大类而建集合无可厚非；在此基础上，store文档集中的一个文档完全可以存储user\_id。因为一个store一定只被一个用户所有，但是一个用户可以拥有多个商店，这样做既满足了实际情况，又不增加冗余，不会每次查找一个商店属于哪个用户时多此一举地再用中间的user\_store文档集来查询。

由此，我们的文档数据库具体结构如下所示：

```
user{
  user_id
  password
  balance
  token
  terminal
}
store{
  store_id
  user_id
  books[] {
    book_id
    stock_level
  }
}
book{
  id
  title
  author
  .....
  content
  tags
  picture
}
order{
  order_id
  store_id
  user_id
  status // 0:未付款;1:付款但是未发货;2:付款发货但是还没被接受;3:付款发货接受了;4:取消
  price
}
order_detail{
  order_id
  book_id
  count
  price
}
```

## 2.3索引设计

众所周知，采用索引对于“频繁查找，不常更改”的数据项之查找功能来说起到了十分好的性能增益作用。结合整体的实验设计，考虑各文档集的实际情况，我们最终这样设置的索引：

1. store文档集的store\_id上设置了升序索引，并且保证了其唯一性
2. user文档集的用户\_id上设置了升序索引，并且保证了其唯一性
3. 为了允许对文本字段进行全文搜索，books文档集上设置了多个索引，分别是在字段title,tags,book\_intro,content上（这些字段的内容都是文本，所以类型都设置为了text）

本来我们还期望在order和order\_detail上设置索引，但这两个文档集经常在变化，虽然它们查询的场景还算广泛（下单、查询订单等），但是权衡之后我们最终还是放弃了在此设置索引。

## 3.基本功能的实现

---

### 3.1用户

#### 3.1.1用户注册与注销（register,unregister）

register函数接受以下参数：

- user\_id: 表示要注册的用户ID。
- password: 表示用户的密码。

该函数的返回值是一个元组，包含两个值：

- 一个整数：代表操作的状态码。200 表示成功，528表示某种错误。
- 一个字符串：描述操作的结果，通常是一个消息或错误消息。

函数主要流程：

1. 尝试生成一个唯一的终端标识，格式为 "terminal\_<当前时间戳>"。
2. 利用用户提供的 user\_id 和生成的终端标识，使用 jwt\_encode 函数生成一个 token。
3. 将用户的信息插入到数据库的 user\_col 集合中，包括user\_id、password、balance、token 和 terminal。
4. 如果在上述过程中发生任何异常，函数会捕获异常，并返回状态码 528，以及异常的字符串表示作为错误消息。
5. 如果操作成功，函数返回状态码 200 和 "ok" 作为成功消息。

unregister函数接受以下参数：

- user\_id: 表示要注销的用户ID。
- password: 表示用户的密码。

该函数的返回值是一个元组，包含两个值：

- 一个整数：代表操作的状态码。200 表示成功，530表示异常，以及其他可能的状态码。
- 一个字符串：描述操作的结果，通常是一个消息或错误消息。

函数主要流程：

1. 调用 `check_password` 函数验证用户提供的密码是否正确。
2. 如果密码验证失败，返回相应的错误代码和消息。
3. 如果密码验证成功，尝试从数据库中删除具有指定 `user_id` 和 `password` 的用户信息。
4. 检查删除操作的结果，如果成功删除了一个用户，返回状态码 200 和 "ok" 作为成功消息。
5. 如果删除操作未成功（`deleted_count` 不为 1），返回授权失败的错误消息。
6. 如果在上述过程中发生任何异常，捕获异常并返回状态码 530，以及异常的字符串表示作为错误消息。

### 3.1.2 检验用户的token/password是否正确 (`check_token, check_password`)

`check_token`:

该函数接受以下参数:

- `user_id`: 表示要验证令牌的用户ID。
- `token`: 表示要验证的令牌。

该函数的返回值是一个元组，包含两个值:

- 一个整数: 代表操作的状态码。200 表示成功，528表示某种错误，以及其他可能的状态码。
- 一个字符串: 描述操作的结果，通常是一个消息或错误消息。

函数主要流程:

1. 通过查询数据库获取指定 `user_id` 的用户信息。
2. 如果用户不存在，返回授权失败的错误消息。
3. 确保查询结果只有一个用户。
4. 从查询结果中获取用户的存储的令牌 `token1`。
5. 使用 `__check_token` 函数验证提供的 `token` 是否与存储的 `token1` 一致。
6. 如果令牌验证失败，返回授权失败的错误消息。
7. 如果令牌验证成功，返回状态码 200 和 "ok" 作为成功消息。

`check_password`与`check_token`类似，我主要来说一下它的运行流程吧:

1. 通过查询数据库获取指定 `user_id` 的用户信息。
2. 如果用户不存在，返回授权失败的错误消息。
3. 检查查询结果中存储的密码是否与提供的密码一致。
4. 如果密码验证失败，返回授权失败的错误消息。
5. 如果密码验证成功，返回状态码 200 和 "ok" 作为成功消息。

### 3.1.3 用户登录与登出 (`login, logout`)

`login`函数接受以下参数:

- `user_id`: 表示要登录的用户ID。
- `password`: 表示用户的密码。
- `terminal`: 表示登录的终端标识。

该函数的返回值是一个元组，包含三个值:

- 一个整数：代表操作的状态码。200 表示成功，528表示某种错误，以及其他可能的状态码。
- 一个字符串：描述操作的结果，通常是一个消息或错误消息。
- 一个字符串：如果操作成功，代表生成的令牌 token；如果操作失败，为空字符串。

函数主要流程：

1. 调用 check\_password 函数验证用户提供的密码是否正确。
2. 如果密码验证失败，返回相应的错误代码和消息。
3. 如果密码验证成功，尝试生成一个新的令牌 token。
4. 更新数据库中存储的用户信息，包括更新 token 和 terminal。
5. 检查更新操作是否成功，如果不成功，返回授权失败的错误消息。
6. 如果在上述过程中发生任何异常，捕获异常并返回状态码 528，以及异常的字符串表示作为错误消息。
7. 如果操作成功，返回状态码 200、"ok" 以及生成的令牌 token。

logout函数接受以下参数：

- user\_id: 表示要注销的用户ID。
- token: 表示用户的令牌。

该函数的返回值是一个元组，包含两个值：

- 一个整数：代表操作的状态码。200 表示成功，528表示某种错误，以及其他可能的状态码。
- 一个字符串：描述操作的结果，通常是一个消息或错误消息。

函数主要流程：

1. 调用 check\_token 函数验证用户提供的令牌是否有效。
2. 如果令牌验证失败，返回相应的错误代码和消息。
3. 生成一个新的终端标识和虚拟令牌。
4. 更新数据库中存储的用户信息，将令牌和终端更新为虚拟值。
5. 检查更新操作是否成功，如果不成功，返回授权失败的错误消息。
6. 如果在上述过程中发生任何异常，捕获异常并返回状态码 528，以及异常的字符串表示作为错误消息。
7. 如果操作成功，返回状态码 200 和 "ok" 作为成功消息。

### 3.1.4修改密码 (change\_password)

该函数接受以下参数：

- user\_id: 表示要更改密码的用户ID。
- old\_password: 表示用户的旧密码。
- new\_password: 表示用户的新密码。

该函数的返回值是一个元组，包含两个值：

- 一个整数：代表操作的状态码。200 表示成功，528表示某种错误，以及其他可能的状态码。
- 一个字符串：描述操作的结果，通常是一个消息或错误消息。

函数主要流程：

1. 调用 `check_password` 函数验证用户提供的旧密码是否正确。
2. 如果旧密码验证失败，返回相应的错误代码和消息。
3. 生成一个新的终端标识和令牌。
4. 更新数据库中存储的用户信息，将密码更新为新密码，并更新令牌和终端。
5. 检查更新操作是否成功，如果不成功，返回授权失败的错误消息。
6. 如果在上述过程中发生任何异常，捕获异常并返回状态码 528，以及异常的字符串表示作为错误消息。
7. 如果操作成功，返回状态码 200 和 "ok" 作为成功消息。

## 3.2 卖家

### 3.2.1 添加书籍 (add\_new\_book)

该函数接受以下参数：

- `user_id`: 表示执行此操作的用户的ID。
- `store_id`: 表示要将书籍添加到的商店的ID。
- `book_id`: 表示要添加的书籍的ID。
- `book_json_str`: 表示包含书籍信息的 JSON 字符串。
- `stock_level`: 表示书籍的库存水平。

该函数的返回值是一个元组，包含两个值：

- 一个整数：代表操作的状态码。200 表示成功，528表示某种错误。
- 一个字符串：描述操作的结果，通常是一个消息或错误消息。

函数主要流程：

1. 函数首先检查用户是否存在。如果用户不存在，它返回一个非存在用户ID的错误消息。
2. 然后，函数检查商店是否存在。如果商店不存在，它返回一个非存在商店ID的错误消息。
3. 接下来，函数检查书籍是否已经存在于商店中。如果书籍已经存在，它返回一个书籍已存在的错误消息。
4. 然后，函数将书籍信息插入到商店集合中，包括商店ID、书籍ID和库存水平。
5. 同时，函数还将书籍信息插入到书籍集合中。
6. 如果在上述过程中发生任何异常，函数会捕获异常，并返回状态码 528，以及异常的字符串表示作为错误消息。

### 3.2.2 创建商铺 (create\_store)

该函数接受以下参数：

- `user_id`: 表示执行此操作的用户的ID。
- `store_id`: 表示要创建的商店的ID。

该函数的返回值是一个元组，包含两个值：

- 一个整数：代表操作的状态码。200 表示成功，528表示某种错误。
- 一个字符串：描述操作的结果，通常是一个消息或错误消息。

函数的主要流程：

1. 函数首先检查用户是否存在。如果用户不存在，它返回一个非存在用户ID的错误消息。
2. 然后，函数检查要创建的商店ID是否已存在。如果商店ID已经存在，它返回一个商店已存在的错误消息。
3. 接着，函数使用 `insert_one` 方法，将新商店的信息插入到 `store_col`集合中，包括商店ID和用户ID，以关联用户与商店。
4. 如果在上述过程中发生任何异常，函数会捕获异常，并返回状态码 528，以及异常的字符串表示作为错误消息。

### 3.2.3添加书籍库存 (add\_stock\_level)

该函数接受以下参数：

- `user_id`: 表示执行此操作的用户的ID。
- `store_id`: 表示书籍所属的商店的ID。
- `book_id`: 表示要增加库存的书籍的ID。
- `add_stock_level`: 表示要增加的库存数量。

该函数的返回值是一个元组，包含两个值：

- 一个整数：代表操作的状态码。200 表示成功，528表示某种错误。
- 一个字符串：描述操作的结果，通常是一个消息或错误消息。

函数的主要流程：

1. 函数首先检查用户是否存在。如果用户不存在，它返回一个非存在用户ID的错误消息。
2. 然后，函数检查商店是否存在。如果商店不存在，它返回一个非存在商店ID的错误消息。
3. 接着，函数检查书籍是否已经存在于商店中。如果书籍不存在，它返回一个非存在书籍ID的错误消息。
4. 然后，函数使用 MongoDB 的 `update_one` 方法，根据商店ID和书籍ID，将库存水平增加 `add_stock_level` 个单位。
5. 如果在上述过程中发生任何异常，函数会捕获异常，并返回状态码 528，以及异常的字符串表示作为错误消息。

## 3.3买家

### 3.3.1创建新订单 (new\_order)

这个函数接受四个参数：

- `user_id`: 一个字符串，代表用户的ID。
- `store_id`: 一个字符串，代表商店的ID。
- `id_and_count`: 一个列表，其中包含元组，每个元组包含两个元素：书本的ID（字符串）和数量（整数）。

函数的返回值是一个元组，包含三个值：

- 一个整数：代表操作的状态码。200 表示成功，528 表示某种错误。
- 一个字符串：描述操作的结果，是一个消息或错误消息。
- 一个字符串：订单ID。

函数的主要流程：



1. 首先，函数定义了一个空字符串 `order_id`，稍后将用于存储订单的ID。
2. 然后，函数执行了一系列检查，以确保用户和商店的存在。如果用户或商店不存在，函数会返回相应的错误消息和空的订单ID。
3. 接下来，函数创建了一个唯一的订单ID `uid`，这个ID包括了用户ID、商店ID以及一个基于时间的唯一标识符。
4. 接下来，函数开始遍历 `id_and_count` 列表中的每个书本ID和数量。对于每个书本，它会执行以下步骤：
  - 查询商店库存，检查书本是否存在。如果书本不存在，它将返回相应的错误消息和空的订单ID。
  - 检查库存水平，如果库存不足，它将返回库存不足的错误消息和空的订单ID。
  - 如果库存足够，它将更新库存，减少相应数量的书本库存。
  - 将书本的订单详细信息插入到订单详细信息集合中，并计算总价格。
5. 计算总价格后，函数获取当前时间，并将订单的详细信息插入到订单集合中，包括订单ID、商店ID、用户ID、创建时间、总价格和订单状态。
6. 最后，如果一切顺利，函数返回状态码 200 表示成功、一个 "ok" 消息以描述成功，以及生成的订单ID。
7. 如果任何异常被捕获，函数会记录日志，并返回状态码 528 表示错误，附带异常信息作为错误消息，以及一个空的订单ID。

### 3.3.2支付 (payment)

该函数接受三个参数：

- `user_id`: 表示用户的ID。
- `password`: 表示用户的密码。
- `order_id`: 表示订单的ID。

该函数的返回值是一个元组，包含两个值：

- 一个整数：代表操作的状态码。200 表示成功，528 表示某种错误。
- 一个字符串：描述操作的结果，通常是一个消息或错误消息。

函数的主要流程：

1. 函数首先尝试在订单集合中查找订单信息，使用给定的订单ID和状态为0（表示订单未付款）。如果没有找到相应的订单信息，它会返回一个相应的错误消息，指示无效的订单ID。
2. 如果找到订单信息，它提取了订单的买家ID、商店ID和总价。
3. 接下来，函数检查用户ID是否与订单的买家ID匹配，以确保用户有权限支付这个订单。如果用户ID与买家ID不匹配，它会返回一个授权失败的错误消息。
4. 然后，函数在用户集合中查找买家的信息，验证用户是否存在，同时检查输入的密码是否与用户的密码匹配。如果用户不存在或密码不匹配，它会返回一个授权失败的错误消息。
5. 接下来，函数查找商店信息，以确保商店存在。如果商店不存在，它会返回一个相应的错误消息。
6. 函数提取卖家的ID，并检查卖家是否存在。如果卖家不存在，它会返回一个相应的错误消息。
7. 接下来，函数检查用户的余额是否足够支付订单的总价。如果余额不足，它会返回一个余额不足的错误消息。
8. 如果余额足够，函数首先从买家的账户中扣除订单的总价。
9. 然后，它将订单的总价添加到卖家的账户中。
10. 接下来，函数将订单的状态更新为1（表示订单已支付），并将订单信息插入订单集合中。

11. 最后，函数尝试从订单集合中删除原始状态为0的订单，以确保订单支付成功。如果无法删除，它会返回一个无效的订单ID的错误消息。
12. 如果在上述过程中发生任何异常，函数会捕获异常，并返回状态码 528，以及异常的字符串表示作为错误消息。

### 3.3.3添加资金 (add\_funds)

该函数接受三个参数：

- user\_id: 表示用户的ID。
- password: 表示用户的密码。
- add\_value: 表示要添加到用户账户余额的金额。

该函数的返回值是一个元组，包含两个值：

- 一个整数：代表操作的状态码。200 表示成功，528表示某种错误。
- 一个字符串：描述操作的结果，通常是一个消息或错误消息。

函数的主要流程：

1. 函数首先尝试在用户集合中查找用户信息，使用给定的用户ID。如果没有找到相应的用户信息，它会返回一个授权失败的错误消息。
2. 如果找到用户信息，函数将验证输入的密码是否与用户的密码匹配。如果密码不匹配，它会返回一个授权失败的错误消息。
3. 接下来，函数使用 \$inc 操作符更新用户账户的余额字段。它将给定的 add\_value 添加到用户的余额中。
4. 然后，函数检查是否成功匹配了一个用户，如果没有匹配到任何用户，它会返回一个用户不存在的错误消息。
5. 如果在上述过程中发生任何异常，函数会捕获异常，并返回状态码 528，以及异常的字符串表示作为错误消息。

## 4.拓展功能的实现

---

### 4.1发货与收货

#### 4.1.1卖家发货 (send\_books)

be/model/seller.py

该函数接受以下参数：

- user\_id: 表示执行此操作的用户的ID。
- order\_id: 表示要标记为已发货的订单的ID。

该函数的返回值是一个元组，包含两个值：

- 一个整数：代表操作的状态码。200 表示成功，528表示某种错误。
- 一个字符串：描述操作的结果，通常是一个消息或错误消息。

函数的主要流程：

1. 函数首先执行一个查询，查找具有以下条件的订单：

- 订单ID等于给定的 order\_id。
  - 订单状态为1、2或3，表示订单已支付但尚未发货、已发货但尚未收到、或已收到。
2. 如果找到符合条件的订单，说明订单可以被标记为已发货。如果没有找到符合条件的订单，函数返回一个无效订单ID的错误消息。
  3. 接下来，函数获取订单的商店ID和支付状态。
  4. 然后，函数检查执行此操作的用户是否是商店的所有者。如果不是，它返回一个授权失败的错误消息，表示只有商店的所有者才能标记订单为已发货。
  5. 接着，函数检查订单的支付状态是否为2或3。如果支付状态是2或3，表示订单已经被标记为已发货或已收到，函数返回一个重复发货的错误消息。
  6. 最后，函数使用 update\_one 方法，将订单的状态标记为2，表示已发货。
  7. 如果在上述过程中发生任何异常，函数会捕获异常，并返回状态码 528，以及异常的字符串表示作为错误消息。

### 4.1.2 买家收货 (receive\_books)

be/model/buyer.py

该函数接受两个参数：

- user\_id: 表示用户的ID。
- order\_id: 表示订单的ID。

该函数的返回值是一个元组，包含两个值：

- 一个整数：代表操作的状态码。200 表示成功，528表示某种错误。
- 一个字符串：描述操作的结果，通常是一个消息或错误消息。

函数的主要流程：

1. 函数首先在订单集合中查找订单信息，使用给定的订单ID和状态为1、2或3的订单。这里使用 \$or 操作符来查找匹配的订单。如果没有找到相应的订单信息，它会返回一个无效的订单ID的错误消息。
2. 如果找到订单信息，函数提取了订单的买家ID和订单的支付状态。
3. 接下来，函数检查订单的买家ID是否与给定的用户ID匹配，以确保用户有权限接收这个订单。如果不匹配，它会返回一个授权失败的错误消息。
4. 然后，函数根据支付状态检查订单是否可以接收。如果订单状态是1，表示订单已支付但书籍尚未发出，它会返回书籍未发出的错误消息。如果订单状态是3，表示已经接收过一次，它会返回重复接收书籍的错误消息。
5. 最后，如果一切正常，函数将更新订单的状态为3，表示书籍已经被接收。
6. 如果在上述过程中发生任何异常，函数会捕获异常，并返回状态码 528，以及异常的字符串表示作为错误消息。

## 4.2 搜索图书 (包括优化)

### 4.2.0 简单搜索

我们最开始是在buyer中实现了一个简单的search方法。这个函数接受四个参数：

- keyword: 表示搜索的关键词。
- store\_id: 表示商店的ID，用于限定搜索结果在特定商店中。

- page: 表示页码，用于分页显示搜索结果，默认为1。
- per\_page: 表示每页的书籍数量，默认为10。

该函数的返回值是一个元组，包含两个值：

- 一个整数：代表操作的状态码。200 表示成功，530表示某种错误。
- 一个列表：包含搜索结果的书籍信息。

函数的主要流程：

1. 首先，函数构建了一个基本查询 base\_query，它使用 MongoDB 的 \$text 操作符来执行全文本搜索，搜索的关键字是 keyword。
2. 接下来，函数创建了一个查询 query，初始时等于基本查询 base\_query。
3. 如果提供了 store\_id，函数会执行以下操作：
  - 在商店集合中查找匹配 store\_id 的书籍ID，并将这些书籍ID存储在列表 books\_id 中。
  - 在查询中添加条件，要求书籍的ID必须在 books\_id 列表中。
4. 然后，函数执行查询操作，查找符合查询条件的书籍。它还使用 \$meta 来获取每本书籍的文本分数，以便后续排序。
5. 接着，函数执行分页操作，跳过前面 (page - 1) \* per\_page 个结果，然后限制每页显示 per\_page 条结果。
6. 如果在上述过程中发生任何异常，函数会捕获异常，并返回状态码 530，以及异常的字符串表示作为错误消息。
7. 最后，函数返回状态码 200，以及包含搜索结果的书籍信息的列表。

但是后来发现这样可能有点不太合理——首先，不仅买家，卖家作为用户理论上也应该可以搜索书籍；其次，我们这样设计的比较简陋，对于实验所要求的几大功能实现情况并没有那么好。

因此，我们在be/model目录添加了一套查找的逻辑在book.py

#### 4.2.1搜索指定标题的图书

```
def search_title(self, title: str, page_num: int, page_size: int):
    return self.search_title_in_store(title, "", page_num, page_size)
def search_title_in_store(self, title: str, store_id: str, page_num: int, page_size:
int):
    book = self.conn.book_col
    condition = {
        "title": title
    }
    result = book.find(condition, {"_id": 0}).skip((page_num - 1) *
page_size).limit(page_size)
    result_list = list(result)
    if store_id != "":
        store = self.conn.store_col
        books_in_store = []
        for b in result_list:
            condition1 = {"store_id": store_id, "books.book_id": b.get('id')}
            book_id = list(store.find(condition1, {"books.book_id": 1}))
            if len(book_id) != 0:
                books_in_store.append(b)
        result_list = books_in_store
```

```

if len(result_list) == 0:
    return 501, f"{title} book not exist", []
return 200, "ok", result_list

```

search\_title的运行逻辑:

- 在books集合中查询指定标题的图书，调用find方法指定title字段值为用户通过http请求传过来的参数title
- 由于\_id对于业务逻辑没有任何影响，用户也不需要\_id值，因此指定\_id为0，表示将查询结果中的\_id字段舍弃
- 最后将查询结果进行分页处理，这里的page\_size和page\_num都是前端在http请求中传过来的参数，分别表示页的大小以及页号
- 接着将查询结果转化为list数组，并判断其长度是否为0
- 若查询结果的长度为0，表示没有指定标题的图书，返回执行码501以及错误信息（指定title的图书不存在），并返回一个空列表
- 若查询结果长度不为0，即本次查询能够查到对应的图书，返回执行码200、消息ok以及对应的结果列表

search\_title\_in\_store是search\_title的拓展，在分页处理之后、查询结果转化为list数组之前多出来的运行逻辑如下所示：

- 然后需要判断结果集中的图书是否在id为\${store\_id}的店铺中，将结果集的图书id作为查询条件，查询store集合中的books数组是否有指定的book\_id
- 若有指定的book\_id，表明这本书是指定店铺中的，将其加入到返回列表中
- 若没有，则这本书不是指定店铺中的，舍弃这条记录
- 完成上述业务有两种逻辑，一种是先去store中查询所有的book\_id，然后拿着所有book\_id去books集合里一一对比title是否满足条件。另一种则是先去books中查询所有满足title为指定值的书，然后再去store集合中查询书是否在店铺内。前一种方式查到的所有book\_id都需要在books集合中进行一次搜索，并且book\_id未必连续，为随机io开销很大。而后一种方式先查询所有指定title的书，能够过滤掉大部分的书，结果集较小，再去指定store中对比，效率更高，并且没有随机io，开销小。

## 4.2.2搜索指定标签的图书

```

def search_tag(self, tag: str, page_num: int, page_size: int):
    return self.search_tag_in_store(tag, "", page_num, page_size)
def search_tag_in_store(self, tag: str, store_id: str, page_num: int, page_size:
int):
    book = self.conn.book_col
    condition = {
        "tags": {"$regex": tag}
    }
    result = book.find(condition, {"_id": 0}).skip((page_num - 1) *
page_size).limit(page_size)
    result_list = list(result)
    if store_id != "":
        store = self.conn.store_col
        books_in_store = []
        for b in result_list:
            condition1 = {"store_id": store_id, "books.book_id": b.get('id')}
            book_id = list(store.find(condition1, {"books.book_id": 1}))
            if len(book_id) != 0:
                books_in_store.append(b)

```

```

        result_list = books_in_store
    if len(result_list) == 0:
        return 501, f"{tag} book not exist", []
    return 200, "ok", result_list

```

search\_tag的运行逻辑如下所示：

- 在books集合中查询含有指定tag的图书，调用find方法指定tags数组中需要包含tag字段值，该tag值为用户通过http请求传过来的参数tag
- 由于\_id对于业务逻辑没有任何影响，用户也不需要\_id值，因此指定\_id为0，表示将查询结果中的\_id字段舍弃
- 最后将查询结果进行分页处理，这里的page\_size和page\_num都是前端在http请求中传过来的参数，分别表示页的大小以及页号
- 接着将查询结果转化为list数组，并判断其长度是否为0
- 若查询结果的长度为0，表示没有满足条件的图书，返回执行码501以及错误信息（指定tag的图书不存在），并返回一个空列表
- 若查询结果长度不为0，即本次查询能够查到对应的图书，返回执行码200、消息ok以及对应的结果列表

search\_tag\_in\_store和search\_tag很像，在分页处理之后，查询结果转化为list数组之前，多出来的逻辑如下所示：

- 然后需要判断结果集中的图书是否在id为\${store\_id}的店铺中，将结果集的图书id作为查询条件，查询store集合中的books数组是否有指定的book\_id
- 若有指定的book\_id，表明这本书是指定店铺中的，将其加入到返回列表中
- 若没有，则这本书不是指定店铺中的，舍弃这条记录

### 4.2.3搜索指定内容的图书

```

def search_content_in_store(self, content: str, store_id: str, page_num: int,
page_size: int):
    book = self.conn.book_col
    condition = {
        "$text": {"$search": content}
    }
    result = book.find(condition, {"_id": 0}).skip((page_num - 1) *
page_size).limit(page_size)
    result_list = list(result)
    if store_id != "":
        store = self.conn.store_col
        books_in_store = []
        for b in result_list:
            condition1 = {"store_id": store_id, "books.book_id": b.get('id')}
            book_id = list(store.find(condition1, {"books.book_id": 1}))
            if len(book_id) != 0:
                books_in_store.append(b)
        result_list = books_in_store
    if len(result_list) == 0:
        return 501, f"{content} book not exist", []
    return 200, "ok", result_list

def search_content(self, content: str, page_num: int, page_size: int):

```

```
return self.search_content_in_store(content, "", page_num, page_size)
```

search\_content运行逻辑如下所示：

- 在books集合中查询含有指定content的图书，调用find方法，使用全文索引进行搜索，要求图书的book\_intro或content字段需要包含用户指定的content值
- 由于\_id对于业务逻辑没有任何影响，用户也不需要\_id值，因此指定\_id为0，表示将查询结果中的\_id字段舍弃
- 最后将查询结果进行分页处理，这里的page\_size和page\_num都是前端在http请求中传过来的参数，分别表示页的大小以及页号
- 接着将查询结果转化为list数组，并判断其长度是否为0
- 若查询结果的长度为0，表示没有满足条件的图书，返回执行码501以及错误信息（指定content的图书不存在），并返回一个空列表
- 若查询结果长度不为0，即本次查询能够查到对应的图书，返回执行码200、消息ok以及对应的结果列表

search\_content\_in\_store和search\_content很像，在分页处理之后，查询结果转化为list数组之前，多出来的逻辑如下所示：

- 然后需要判断结果集中的图书是否在id为\${store\_id}的店铺中，将结果集的图书id作为查询条件，查询store集合中的books数组是否有指定的book\_id
- 若有指定的book\_id，表明这本书是指定店铺中的，将其加入到返回列表中
- 若没有，则这本书不是指定店铺中的，舍弃这条记录

#### 4.2.4搜索指定作者的图书

```
def search_author_in_store(self, author: str, store_id: str, page_num: int,
page_size: int):
    book = self.conn.book_col
    condition = {
        "author": author
    }
    result = book.find(condition, {"_id": 0}).skip((page_num - 1) *
page_size).limit(page_size)
    result_list = list(result)
    if store_id != "":
        store = self.conn.store_col
        books_in_store = []
        for b in result_list:
            condition1 = {"store_id": store_id, "books.book_id": b.get('id')}
            book_id = list(store.find(condition1, {"books.book_id": 1}))
            if len(book_id) != 0:
                books_in_store.append(b)
        result_list = books_in_store
    if len(result_list) == 0:
        return 501, f"{author} book not exist", []
    return 200, "ok", result_list

def search_author(self, author: str, page_num: int, page_size: int):
    return self.search_author_in_store(author, "", page_num, page_size)
```

search\_author逻辑如下所示：

- 在books集合中查询含有指定author的图书，调用find方法，指定author字段值为用户通过http请求传过来的参数author
- 由于\_id对于业务逻辑没有任何影响，用户也不需要\_id值，因此指定\_id为0，表示将查询结果中的\_id字段舍弃
- 最后将查询结果进行分页处理，这里的page\_size和page\_num都是前端在http请求中传过来的参数，分别表示页的大小以及页号
- 接着将查询结果转化为list数组，并判断其长度是否为0
- 若查询结果的长度为0，表示没有满足条件的图书，返回执行码501以及错误信息（指定author的图书不存在），并返回一个空列表
- 若查询结果长度不为0，即本次查询能够查到对应的图书，返回执行码200、消息ok以及对应的结果列表

search\_author\_in\_store和search\_author很像，在分页处理之后，查询结果转化为list数组之前，多出来的逻辑如下所示：

- 然后需要判断结果集中的图书是否在id为\${store\_id}的店铺中，将结果集的图书id作为查询条件，查询store集合中的books数组是否有指定的book\_id
- 若有指定的book\_id，表明这本书是指定店铺中的，将其加入到返回列表中
- 若没有，则这本书不是指定店铺中的，舍弃这条记录

可以发现，其实上述的逻辑是类似的，只不过查询的范围存在差异，我们有希望包装好合适的函数进行批处理操作。可惜时间有限，我们的相应的测试用例通过情况也不错，所以这里就没有进一步的包装，可能算是一个小小的遗憾。

## 4.3 查询、取消订单

### 4.3.1 查询订单历史 (check\_hist\_order)

be/model/buyer.py

这个函数接受一个参数：

- user\_id: 表示用户的ID。

该函数没有返回值，而是在内部构建了一个包含历史订单信息的列表 ans，并在最后返回这个列表。

函数的主要流程：

1. 函数首先检查用户是否存在。如果用户不存在，它将返回一个非存在用户ID的错误消息。
2. 接下来，函数初始化一个空列表 ans，该列表将用于存储历史订单的信息。
3. 然后，函数开始检查不同状态的订单：
  - 首先，它查找未付款的订单 (status=0)，并将这些订单的信息添加到 ans 列表中。
  - 然后，它查找已支付但尚未发货、已支付但未收到、已收到的订单 (status=1, 2, 3)，并将这些订单的信息添加到 ans 列表中。
  - 最后，它查找已取消的订单 (status=4)，并将这些订单的信息添加到 ans 列表中。
4. 对于每个订单状态，函数执行以下操作：
  - 查找匹配用户ID和订单状态的订单信息。
  - 对于每个订单，它查找与订单相关的书籍详细信息。



- 为每个订单构建一个字典，包含订单的状态、订单ID、买家ID、商店ID、总价和书籍详细信息。然后将该字典添加到 ans 列表中。
5. 最后，如果没有找到任何历史订单，函数将返回一个成功的消息，指示没有找到订单。否则，它将返回一个成功的消息和包含历史订单信息的 ans 列表。
  6. 如果在上述过程中发生任何异常，函数会捕获异常，并返回状态码 528，以及异常的字符串表示作为错误消息。

### 4.3.2取消订单 (cancel\_order)

be/model/buyer.py

该函数接受两个参数：

- user\_id: 表示用户的ID。
- order\_id: 表示订单的ID。

该函数的返回值是一个元组，包含两个值：

- 一个整数：代表操作的状态码。200 表示成功，528表示某种错误。
- 一个字符串：描述操作的结果，通常是一个消息或错误消息。

函数的主要流程：

1. 首先，函数尝试在订单集合中查找订单信息，使用给定的订单ID和状态为0（表示订单未付款）。如果找到相应的订单信息，它提取了订单的买家ID、商店ID和订单价格，然后从订单集合中删除该订单。
2. 如果在第一步没有找到匹配的订单，函数继续寻找订单状态为1、2或3的订单，使用 \$or 操作符来查找匹配的订单。如果找到相应的订单信息，它提取了订单的买家ID、商店ID和订单价格，并继续执行以下操作：
  - 检查买家ID是否与给定的用户ID匹配，以确保用户有权限取消订单。如果不匹配，它返回一个授权失败的错误消息。
  - 查找卖家ID，然后从卖家账户中扣除订单的价格，同时将相同金额添加到买家账户中。
  - 最后，从订单集合中删除匹配的订单。
3. 如果在上述步骤中没有找到匹配的订单，函数返回一个无效的订单ID的错误消息。
4. 接下来，函数通过查询订单详细信息集合来获取已购书籍的信息。它遍历订单详细信息，对于每本书籍，将库存还原，增加相应数量的库存。
5. 最后，函数将创建一个新订单，状态设置为4，表示订单已取消，包括订单ID、用户ID、商店ID、订单价格和状态，并插入到订单集合中。
6. 如果在上述过程中发生任何异常，函数会捕获异常，并返回状态码 528，以及异常的字符串表示作为错误消息。

### 4.3.3自动取消订单 (auto\_cancel\_order)

be/model/buyer.py

该函数没有传入参数，但是在内部执行一系列操作来自动取消未支付的订单。它的返回值是一个元组，包含两个值：

- 一个整数：代表操作的状态码。200 表示成功，528表示某种错误。
- 一个字符串：描述操作的结果，通常是一个消息或错误消息。

函数的主要流程：

1. 函数首先定义了一个等待时间 `wait_time`，这个时间表示多久未支付的订单将被自动取消，这里设置为20秒。
2. 接着，函数获取当前的UTC时间 `now`，然后计算出一个时间间隔 `interval`，该间隔是当前时间减去 `wait_time` 秒后的时间。
3. 接下来，函数构建了一个查询条件 `cursor`，用于查找满足以下条件的订单：订单状态为0（未支付）且订单创建时间早于 `interval`。
4. 然后，函数执行查询，查找待取消的订单，将这些订单的信息存储在 `orders_to_cancel` 中。
5. 如果找到待取消的订单，函数遍历这些订单，依次执行以下操作：
  - 获取订单的相关信息，包括订单ID、用户ID、商店ID和订单价格。
  - 从订单集合中删除这个订单，取消订单。
  - 查询被取消订单的书籍详细信息，并遍历这些书籍。
  - 对每本书籍，将库存还原，增加相应数量的库存。
  - 如果库存还原失败（`update_result.modified_count == 0`），则返回一个库存不足的错误消息。
6. 最后，对于每个已取消的订单，函数构建一个新的订单文档 `canceled_order`，将其状态设置为4（已取消），然后插入到订单集合中。
7. 如果在上述过程中发生任何异常，函数会捕获异常，并返回状态码 528，以及异常的字符串表示作为错误消息。

这里要给buyer设置一个函数来检查订单是否已经被取消(`is_order_cancelled`):

该函数接受以下参数：

- `self`: 这是一个类方法，通常用于引用类的实例。
- `order_id`: 表示要检查的订单的ID。

该函数的返回值是一个元组，包含两个值：

- 一个整数：代表操作的状态码。200 表示成功，而其他状态码表示不同的错误情况。
- 一个字符串：描述操作的结果，通常是一个消息或错误消息。

函数的主要流程：

1. 函数首先在订单集合中执行一个查询，查找订单ID等于给定的 `order_id`，且状态等于4（表示已取消）的订单。
2. 如果找到符合条件的订单，说明订单已经被取消，函数返回一个成功的状态码 200 和消息 "ok"。
3. 如果没有找到符合条件的订单，说明订单未被取消，函数返回一个指示自动取消失败的错误消息。

## 5.接口与测试

---

### 5.1 接口

后端接口在 `be/view/` ,前端接口则在 `fe/access`

## 5.1.2后端接口

`be/view/auth.py` 中的接口我们并未修改，其中login,logout,register,unregister,password分别对应登录，登出，注册，注销，修改密码，这里就不赘述了。

`be/view/buyer.py` 中提供了几个基本的接口，即new\_order, payment, add\_funds分别对应发起新订单、支付、增加余额；`be/view/seller.py` 中提供了几个基本的接口，即create\_store, add\_book, add\_stock\_level分别对应创建商店、增加书目、增加库存。这些接口我们并没有修改。

### 5.1.2.1 buyer和seller添加的功能的接口

`be/view/buyer.py` 中我们添加了几个接口，分别是receive\_books, cancel\_order, auto\_cancel\_order, is\_order\_cancelled, check\_hist\_order, search 它们分别对应收书，取消订单和自动取消订单，检验订单是否已被取消，查询订单历史，简单查找

`be/view/seller.py` 中我们只添加了一个接口，是send\_books，即发书功能。

### 5.1.2.2 精细化搜索功能的接口

位于`be/view/search.py`

前面的接口其实都有些“照葫芦画瓢”，但是这里的实现有点不同，我们想着重来说一下。

基本的流程大概是这样的：

- 接口路由为"/search/tag\_in\_store"，请求方法为get请求
- 由flask的内置对象request，获取请求参数title、page\_num、page\_size，分别表示标题，请求页的大小
- 判断上述请求参数是否为None，若为None，则为这些变量赋上默认值。
- 调用be.model.book的对应方法
- 将返回值封装到字典中，调用json的jsonify方法序列化并返回给前端，返回结构如下

```
{
    "data": books,
    "message": message,
    "code": code
}
```

我们以此为模板实现了对应的**搜索指定标题的图书接口 (search\_title)**，**搜索指定tag的图书接口 (search\_tag)**，**搜索指定content的图书接口 (search\_content)**，**搜索指定author的图书接口 (search\_author)** 以及**在店铺内搜索指定标题,tag,content,author的图书接口 (search\_title\_in\_store,search\_tag\_in\_store,search\_content\_in\_store,search\_author\_in\_store)**

## 5.1.3前端接口

在修改完后端的接口后，我们在 `fe/access` 实现了与之对应的前端接口,因为实在是过度模式化，没什么亮点，此处就不赘述了，这里主要来谈一下这个模式化吧：

- 先拼接得到相应的接口路由
- 再调用requests库的get方法，对该接口发送一个get请求，指定params为上述请求参数
- 得到response后，调用json.loads方法解析返回的json字符串，返回执行码（即json对象的code）

## 5.2 测试

在fe/test文件夹下，除了基础的33个测试点，我们还对于补充的功能添加了许多额外的测试。这些新添加的测试类都要先进行一下初始化，使用uuid构造一个xx\_id(比如seller\_id),password,并调用相应接口注册，注册之后进行诸如创建店铺这样的操作。

### 5.2.1 test\_cancel\_order

在初始化测试类之后，有几个具体的测试：

#### 1. test\_paid

- 情况：已付款的订单。
- 操作：创建订单，付款，取消订单。
- 预期结果：取消订单成功。

#### 2. test\_unpaid

- 情况：未付款的订单。
- 操作：创建订单，取消订单。
- 预期结果：取消订单成功。

#### 3. test\_invalid\_order\_id\_paid

- 情况：已付款的订单，使用不存在的订单ID取消。
- 操作：创建订单，付款，使用不存在的订单ID取消订单。
- 预期结果：取消订单失败。

#### 4. test\_invalid\_order\_id\_unpaid

- 情况：未付款的订单，使用不存在的订单ID取消。
- 操作：创建订单，使用不存在的订单ID取消订单。
- 预期结果：取消订单失败。

#### 5. test\_authorization\_error\_paid

- 情况：已付款的订单，用户ID不存在。
- 操作：创建订单，付款，使用不存在的用户ID取消订单。
- 预期结果：取消订单失败。

#### 6. test\_authorization\_error\_unpaid

- 情况：未付款的订单，用户ID不存在。
- 操作：创建订单，使用不存在的用户ID取消订单。
- 预期结果：取消订单失败。

#### 7. test\_repeat\_cancel\_paid

- 情况：已付款的订单，尝试重复取消。
- 操作：创建订单，付款，取消订单，再次取消订单。
- 预期结果：第一次取消成功，第二次取消失败。

#### 8. test\_repeat\_cancel\_not\_paid

- 情况：未付款的订单，尝试重复取消。

- 操作：创建订单，取消订单，再次取消订单。
- 预期结果：第一次取消成功，第二次取消失败。

## 5.2.2 test\_cancel\_auto

这个测试文件主要测试了自动取消订单的不同情况，以下是各个测试点的用法解释：

### 1. test\_overtime

- 情况：订单在规定时间内未付款，自动取消。
- 操作：创建订单，等待超过规定时间，检查订单是否被取消。
- 预期结果：订单应该被取消，返回码为200。

### 2. test\_overtime\_paid

- 情况：订单在规定时间内已付款，超时后检查订单状态。
- 操作：创建订单，付款，等待超过规定时间，检查订单是否被取消。
- 预期结果：由于订单已付款，所以不应该被取消，返回码不为200。

### 3. test\_overtime\_canceled\_by\_buyer

- 情况：订单在规定时间内被买家取消，超时后检查订单状态。
- 操作：创建订单，买家取消订单，等待超过规定时间，检查订单是否被取消。
- 预期结果：由于订单在规定时间内被买家取消，所以应该被取消，返回码为200。

## 5.2.3 test\_history\_order

### 1. test\_have\_orders

- 情况：有历史订单的情况。
- 操作：创建10个订单，其中可能包含取消、已付款、发货和收货的情况，然后调用 `check_hist_order` 检查历史订单。
- 预期结果：检查历史订单返回码为200。

### 2. test\_non\_exist\_user\_id

- 情况：使用不存在的用户ID检查历史订单。
- 操作：调用 `check_hist_order` 检查历史订单。
- 预期结果：检查历史订单返回码不为200。

### 3. test\_no\_orders

- 情况：没有历史订单的情况。
- 操作：调用 `check_hist_order` 检查历史订单。
- 预期结果：检查历史订单返回码为200。

## 5.2.4 test\_receive

### 1. test\_ok

- 情况：正常收货。
- 操作：卖家发货，买家收货。
- 预期结果：收货成功，返回码为200。

## 2. test\_order\_error

- 情况：订单ID不存在的情况下尝试收货。
- 操作：卖家发货，使用不存在的订单ID尝试收货。
- 预期结果：收货失败，返回码不为200。

## 3. test\_authorization\_error

- 情况：使用不存在的买家ID尝试收货。
- 操作：卖家发货，使用不存在的买家ID尝试收货。
- 预期结果：收货失败，返回码不为200。

## 4. test\_books\_not\_send

- 情况：未发货的订单尝试收货。
- 操作：未发货的订单尝试收货。
- 预期结果：收货失败，返回码不为200。

## 5. test\_books\_repeat\_receive

- 情况：重复收货。
- 操作：卖家发货，买家收货，再次尝试收货。
- 预期结果：第一次收货成功，第二次收货失败，返回码不为200。

# 5.2.5 test\_send

## 1. test\_ok

- 情况：正常发货。
- 操作：卖家发货。
- 预期结果：发货成功，返回码为200。

## 2. test\_order\_error

- 情况：订单ID不存在的情况下尝试发货。
- 操作：使用不存在的订单ID尝试发货。
- 预期结果：发货失败，返回码不为200。

## 3. test\_authorization\_error

- 情况：使用不存在的卖家ID尝试发货。
- 操作：使用不存在的卖家ID尝试发货。
- 预期结果：发货失败，返回码不为200。

## 4. test\_books\_repeat\_send

- 情况：重复发货。
- 操作：卖家发货，再次尝试发货。
- 预期结果：第一次发货成功，第二次发货失败，返回码不为200。

## 5.2.6 test\_search

在初始化 `TestSearch` 类之后，有几个具体的测试：

### 1. 测试图书全属性搜索 (test\_all\_field\_search)

- 调用buyer的search方法查找指定keyword的图书
- 调用json.loads方法解析返回的json字符串，并打印返回内容
- 由于keyword存在，接口逻辑正确的情况下返回的code应该为200
- 因此断言code为200，若不为200则测试失败，接口错误

### 2. 测试图书分页搜索 (test\_pagination)

- 调用buyer的search方法查找指定keyword的图书
- 由于keyword存在，接口逻辑正确的情况下返回的code应该为200
- 因此断言code为200，若不为200则测试失败，接口错误

### 3. 测试根据指定title搜索图书 (test\_search\_title)

- 使用uuid构造一个数据库中不存在的title的图书，并调用seller的add\_book方法加入此图书，断言code为200，即操作正常
- 调用fe.access.TestRequest类的request\_search\_title方法，传入的title值为之前构造的title，断言code为200
- 接着调用fe.access.TestRequest类的request\_search\_title方法，传入的title值为一个不存在的title，断言code为501（即查询失败，指定图书不存在）

### 4. 测试在店铺内根据指定title搜索图书 (test\_search\_title\_in\_store)

- 使用uuid构造一个数据库中不存在的title的图书，并调用seller的add\_book方法加入此图书，断言code为200，即操作正常
- 调用fe.access.TestRequest类的request\_search\_title\_in\_store方法，传入的title值为之前构造的title，store\_id为seller的store\_id，断言code为200
- 接着调用fe.access.TestRequest类的request\_search\_title\_in\_store方法，传入的title值为一个不存在的title，store\_id为seller的store\_id，断言code为501（即查询失败，指定图书不存在）

### 5. 测试根据指定tag搜索图书 (test\_search\_tag)

- 使用uuid构造一个数据库中不存在的tag的图书，并调用seller的add\_book方法加入此图书，断言code为200，即操作正常
- 调用fe.access.TestRequest类的request\_search\_tag方法，传入的tag值为之前构造的tag，断言code为200
- 接着调用fe.access.TestRequest类的request\_search\_tag方法，传入的tag值为一个不存在的tag，断言code为501（即查询失败，指定图书不存在）

### 6. 测试在店铺内根据指定tag搜索图书 (test\_search\_tag\_in\_store)

- 使用uuid构造一个数据库中不存在的tag的图书，并调用seller的add\_book方法加入此图书，断言code为200，即操作正常
- 调用fe.access.TestRequest类的request\_search\_tag\_in\_store方法，传入的tag值为之前构造的tag，store\_id为seller的store\_id，断言code为200

- 接着调用fe.access.TestRequest类的request\_search\_tag\_in\_store方法，传入的tag值为一个不存在的tag，store\_id为seller的store\_id，断言code为501（即查询失败，指定图书不存在）
7. 测试根据指定content搜索图书（test\_search\_content）
- 使用uuid构造一个数据库中不存在的content的图书，并调用seller的add\_book方法加入此图书，断言code为200，即操作正常
  - 调用fe.access.TestRequest类的request\_search\_content方法，传入的content值为之前构造的content，断言code为200
  - 接着调用fe.access.TestRequest类的request\_search\_content方法，传入的content值为一个不存在的content，断言code为501（即查询失败，指定图书不存在）
8. 测试在店铺内根据指定content搜索图书（test\_search\_content\_in\_store）
- 使用uuid构造一个数据库中不存在的content的图书，并调用seller的add\_book方法加入此图书，断言code为200，即操作正常
  - 调用fe.access.TestRequest类的request\_search\_content\_in\_store方法，传入的content值为之前构造的content，store\_id为seller的store\_id，断言code为200
  - 接着调用fe.access.TestRequest类的request\_search\_content\_in\_store方法，传入的content值为一个不存在的content，store\_id为seller的store\_id，断言code为501（即查询失败，指定图书不存在）
9. 测试根据指定author搜索图书（test\_search\_author）
- 使用uuid构造一个数据库中不存在的author的图书，并调用seller的add\_book方法加入此图书，断言code为200，即操作正常
  - 调用fe.access.TestRequest类的request\_search\_author方法，传入的author值为之前构造的author，断言code为200
  - 接着调用fe.access.TestRequest类的request\_search\_author方法，传入的author值为一个不存在的author，断言code为501（即查询失败，指定图书不存在）
10. 测试在店铺内根据指定author搜索图书（test\_search\_author\_in\_store）
- 使用uuid构造一个数据库中不存在的author的图书，并调用seller的add\_book方法加入此图书，断言code为200，即操作正常
  - 调用fe.access.TestRequest类的request\_search\_author\_in\_store方法，传入的author值为之前构造的author，store\_id为seller的store\_id，断言code为200
  - 接着调用fe.access.TestRequest类的request\_search\_author\_in\_store方法，传入的author值为一个不存在的author，store\_id为seller的store\_id，断言code为501（即查询失败，指定图书不存在）

## 6.遇到的主要问题展示

---

### 6.1store的books插入了太多的信息以致于突破了MongoDB限制

最初我们的想法是一个store对应多本书，在store中存放一个books列表，books中的每一个元素都是一本具体的书。每一个元素都包含book\_id、book\_info和stock\_level。但是这种设计最后的结果会出现如下报错：



```
An error occurred: BSONObj size: 17020874 (0x103B7CA) is invalid. Size must be between 0 and 16793600(16MB) First element: _id: ObjectId('65499f4c6ead219cc4eea59b'), full error: {'index': 0, 'code': 10334, 'errmsg': "BSONObj size: 17020874 (0x103B7CA) is invalid. Size must be between 0 and 16793600(16MB) First element: _id: ObjectId('65499f4c6ead219cc4eea59b')"} }
```

意思就是说，MongoDB的默认大小为16MB，而我们在不断往store中添加书籍的过程中超出了这个大小限制。

在这种设计下，我们最终能通过除了test\_bench之外的所有测试，唯独test\_bench无法通过。

经过思考，我们决定将book\_info字段从store当中移除。因为首先book\_info就是出现上述报错的罪魁祸首，实在是太大了。但是去掉之后，我们就无法直接从store中获得书籍信息了。但这其实无伤大雅，我们想要获得书籍的具体信息只要从book\_col当中寻找即可。

## 6.2 fe/access/book.py 修改的问题

最开始我们采用的是自己修改过后的fe/access/book.py，将本地的book.db导入到了本地MongoDB中。在修改之后，除了test\_bench这个测试点以外都能PASS。后来去看水杉上助教学长的发帖：

文档中说明所有对 SQL 的操作都要替换成对 MongoDB 的操作，这是一个错误，fe/access/book.py 中的 SQL 不需要更改，因为这里是读取 book.db (或者 book\_lx.db) 中的数据，并调用 add\_book 函数来插入到自定义的数据库中用于测试插入数据的性能，所以和自定义数据库无关。但是如果已经改为 MongoDB 操作并且能正常运行 test\_bench, 那就不需要改回 SQL。

然后将fe/access/book.py替换为了最原始的，然后其他代码文件没有修改的情况下，test\_bench也跑通了。

## 7. 实验分工及结果

### 7.1 小组分工

10211900416 郭夏辉（贡献度33.33%）

基本用户的实现、数据库模式设计、查询历史订单、收货与发货、实验报告撰写

10215501451 包亦晟（贡献度33.33%）

基本买家功能实现、git版本控制、手动及自动取消订单、bug调试、实验报告撰写

10225501461 朱天祥（贡献度33.33%）

基本卖家功能实现、搜索功能（包括优化及各种情况）、前端与后端接口测试、数据库性能提升、实验报告撰写

**三人的工作分配完全均匀，协作过程十分丝滑流畅**

### 7.2 测试结果

(基础功能):

```

$ bash script/test.sh
===== test session starts =====
platform win32 -- Python 3.10.9, pytest-7.2.0, pluggy-1.0.0 -- C:\learnAI\anacon
da3\python.exe
cachedir: .pytest_cache
rootdir: D:\CDMS\labs\project\bookstore20231107final
plugins: anyio-3.5.0
collecting ... frontend begin test
* Serving Flask app 'be.serve' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployme
nt.
  Use a production WSGI server instead.
* Debug mode: off
2023-11-10 13:33:46,384 [Thread-1 (ru)] [INFO ] * Running on http://127.0.0.1:5
000/ (Press CTRL+C to quit)
collected 33 items

fe/test/test_add_book.py::TestAddBook::test_ok PASSED [ 3%]
fe/test/test_add_book.py::TestAddBook::test_error_non_exist_store_id PASSED [ 6
%]
fe/test/test_add_book.py::TestAddBook::test_error_exist_book_id PASSED [ 9%]
fe/test/test_add_book.py::TestAddBook::test_error_non_exist_user_id PASSED [ 12%
]
fe/test/test_add_funds.py::TestAddFunds::test_ok PASSED [ 15%]
fe/test/test_add_funds.py::TestAddFunds::test_error_user_id PASSED [ 18%]
fe/test/test_add_funds.py::TestAddFunds::test_error_password PASSED [ 21%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_error_user_id PASSED [
24%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_error_store_id PASSED [
27%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_error_book_id PASSED [
30%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_ok PASSED [ 33%]
fe/test/test_bench.py::test_bench PASSED [ 36%]
fe/test/test_create_store.py::TestCreateStore::test_ok PASSED [ 39%]
fe/test/test_create_store.py::TestCreateStore::test_error_exist_store_id PASSED
[ 42%]
fe/test/test_login.py::TestLogin::test_ok PASSED [ 45%]
fe/test/test_login.py::TestLogin::test_error_user_id PASSED [ 48%]
fe/test/test_login.py::TestLogin::test_error_password PASSED [ 51%]
fe/test/test_new_order.py::TestNewOrder::test_non_exist_book_id PASSED [ 54%]
fe/test/test_new_order.py::TestNewOrder::test_low_stock_level PASSED [ 57%]
fe/test/test_new_order.py::TestNewOrder::test_ok PASSED [ 60%]
fe/test/test_new_order.py::TestNewOrder::test_non_exist_user_id PASSED [ 63%]
fe/test/test_new_order.py::TestNewOrder::test_non_exist_store_id PASSED [ 66%]
fe/test/test_password.py::TestPassword::test_ok PASSED [ 69%]
fe/test/test_password.py::TestPassword::test_error_password PASSED [ 72%]

```

```

fe/test/test_password.py::TestPassword::test_error_password PASSED [ 72%]
fe/test/test_password.py::TestPassword::test_error_user_id PASSED [ 75%]
fe/test/test_payment.py::TestPayment::test_ok PASSED [ 78%]
fe/test/test_payment.py::TestPayment::test_authorization_error PASSED [ 81%]
fe/test/test_payment.py::TestPayment::test_not_suff_funds PASSED [ 84%]
fe/test/test_payment.py::TestPayment::test_repeat_pay PASSED [ 87%]
fe/test/test_register.py::TestRegister::test_register_ok PASSED [ 90%]
fe/test/test_register.py::TestRegister::test_unregister_ok PASSED [ 93%]
fe/test/test_register.py::TestRegister::test_unregister_error_authorization PASSED [ 96%]
fe/test/test_register.py::TestRegister::test_register_error_exist_user_id PASSED [100%]
D:\CDMS\labs\project\bookstore20231107final\be\serve.py:19: UserWarning:
The 'environ['werkzeug.server.shutdown']' function is deprecated and will be removed in Werkzeug 2.1.
  func()
2023-11-10 13:37:14,863 [Thread-3914 ] [INFO ] 127.0.0.1 - - [10/Nov/2023 13:37:14] "GET /shutdown HTTP/1.1" 200 -

```

===== 33 passed in 210.62s (0:03:30) =====

frontend end test

No data to combine

Name	Stmts	Miss	Branch	BrPart	Cover
-----	-----	-----	-----	-----	-----
be\__init__.py	0	0	0	0	100%
be\app.py	3	3	2	0	0%
be\model\__init__.py	0	0	0	0	100%
be\model\book.py	80	69	32	0	10%
be\model\buyer.py	238	136	110	11	39%
be\model\db_conn.py	19	0	6	0	100%
be\model\error.py	33	6	0	0	82%
be\model\seller.py	58	23	22	1	62%
be\model\store.py	22	0	0	0	100%
be\model\user.py	102	15	30	6	84%
be\serve.py	37	1	2	1	95%
be\view\__init__.py	0	0	0	0	100%
be\view\auth.py	42	0	0	0	100%
be\view\buyer.py	76	28	2	0	64%
be\view\search.py	86	64	32	0	19%
be\view\seller.py	38	5	0	0	87%
fe\__init__.py	0	0	0	0	100%
fe\access\__init__.py	0	0	0	0	100%
fe\access\auth.py	31	0	0	0	100%
fe\access\book.py	70	1	12	2	96%
fe\access\buyer.py	73	31	4	0	57%
fe\access\new_buyer.py	8	0	0	0	100%
fe\access\new_seller.py	8	0	0	0	100%
fe\access\search.py	53	53	0	0	0%
fe\access\seller.py	37	5	0	0	86%

Name	Stmts	Miss	Branch	BrPart	Cover
be\__init__.py	0	0	0	0	100%
be\app.py	3	3	2	0	0%
be\model\__init__.py	0	0	0	0	100%
be\model\book.py	80	69	32	0	10%
be\model\buyer.py	238	136	110	11	39%
be\model\db_conn.py	19	0	6	0	100%
be\model\error.py	33	6	0	0	82%
be\model\seller.py	58	23	22	1	62%
be\model\store.py	22	0	0	0	100%
be\model\user.py	102	15	30	6	84%
be\serve.py	37	1	2	1	95%
be\view\__init__.py	0	0	0	0	100%
be\view\auth.py	42	0	0	0	100%
be\view\buyer.py	76	28	2	0	64%
be\view\search.py	86	64	32	0	19%
be\view\seller.py	38	5	0	0	87%
fe\__init__.py	0	0	0	0	100%
fe\access\__init__.py	0	0	0	0	100%
fe\access\auth.py	31	0	0	0	100%
fe\access\book.py	70	1	12	2	96%
fe\access\buyer.py	73	31	4	0	57%
fe\access\new_buyer.py	8	0	0	0	100%
fe\access\new_seller.py	8	0	0	0	100%
fe\access\search.py	53	53	0	0	0%
fe\access\seller.py	37	5	0	0	86%
fe\bench\__init__.py	0	0	0	0	100%
fe\bench\run.py	13	0	6	0	100%
fe\bench\session.py	47	0	12	1	98%
fe\bench\workload.py	125	1	22	2	98%
fe\conf.py	11	0	0	0	100%
fe\conftest.py	17	0	0	0	100%
fe\test\gen_book_data.py	49	0	16	0	100%
fe\test\test_add_book.py	36	0	10	0	100%
fe\test\test_add_funds.py	23	0	0	0	100%
fe\test\test_add_stock_level.py	39	0	10	0	100%
fe\test\test_bench.py	7	3	0	0	57%
fe\test\test_create_store.py	20	0	0	0	100%
fe\test\test_login.py	28	0	0	0	100%
fe\test\test_new_order.py	40	0	0	0	100%
fe\test\test_password.py	33	0	0	0	100%
fe\test\test_payment.py	60	1	4	1	97%
fe\test\test_register.py	31	0	0	0	100%
TOTAL	1693	445	334	25	70%
Wrote HTML report to htmlcov\index.html					

(拓展功能):

```
$ bash script/test.sh
===== test session starts =====
platform win32 -- Python 3.10.9, pytest-7.2.0, pluggy-1.0.0 -- C:\learnAI\anaconda3\python.exe
cachedir: .pytest_cache
rootdir: D:\CDMS\labs\project\bookstore20231107final
plugins: anyio-3.5.0
collecting ... frontend begin test
* Serving Flask app 'be.serve' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
2023-11-10 13:42:46,434 [Thread-1 (ru)] [INFO] * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
collected 66 items

fe/test/test_add_book.py::TestAddBook::test_ok PASSED [ 1%]
fe/test/test_add_book.py::TestAddBook::test_error_non_exist_store_id PASSED [ 3%]
fe/test/test_add_book.py::TestAddBook::test_error_exist_book_id PASSED [ 4%]
fe/test/test_add_book.py::TestAddBook::test_error_non_exist_user_id PASSED [ 6%]
fe/test/test_add_funds.py::TestAddFunds::test_ok PASSED [ 7%]
fe/test/test_add_funds.py::TestAddFunds::test_error_user_id PASSED [ 9%]
fe/test/test_add_funds.py::TestAddFunds::test_error_password PASSED [ 10%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_error_user_id PASSED [ 12%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_error_store_id PASSED [ 13%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_error_book_id PASSED [ 15%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_ok PASSED [ 16%]
fe/test/test_bench.py::test_bench PASSED [ 18%]
fe/test/test_cancel_auto.py::TestCancelauto::test_overtime PASSED [ 19%]
fe/test/test_cancel_auto.py::TestCancelauto::test_overtime_paid PASSED [ 21%]
fe/test/test_cancel_auto.py::TestCancelauto::test_overtime_canceled_by_buyer PASSED [ 22%]
fe/test/test_cancel_order.py::TestCancelOrder::test_paid PASSED [ 24%]
fe/test/test_cancel_order.py::TestCancelOrder::test_unpaid PASSED [ 25%]
fe/test/test_cancel_order.py::TestCancelOrder::test_invalid_order_id_paid PASSED [ 27%]
fe/test/test_cancel_order.py::TestCancelOrder::test_invalid_order_id_unpaid PASSED [ 28%]
fe/test/test_cancel_order.py::TestCancelOrder::test_authorization_error_paid PASSED [ 30%]
fe/test/test_cancel_order.py::TestCancelOrder::test_authorization_error_unpaid PASSED [ 31%]
fe/test/test_cancel_order.py::TestCancelOrder::test_repeat_cancel_paid PASSED [ 33%]
fe/test/test_cancel_order.py::TestCancelOrder::test_repeat_cancel_not_paid PASSED [ 34%]
fe/test/test_create_store.py::TestCreateStore::test_ok PASSED [ 36%]
fe/test/test_create_store.py::TestCreateStore::test_error_exist_store_id PASSED [ 37%]
fe/test/test_history_order.py::Testhistoryorder::test_have_orders PASSED [ 39%]
fe/test/test_history_order.py::Testhistoryorder::test_non_exist_user_id PASSED [ 40%]
fe/test/test_history_order.py::Testhistoryorder::test_no_orders PASSED [ 42%]
fe/test/test_login.py::TestLogin::test_ok PASSED [ 43%]
fe/test/test_login.py::TestLogin::test_error_user_id PASSED [ 45%]
fe/test/test_login.py::TestLogin::test_error_password PASSED [ 46%]
```



```

fe/test/test_new_order.py::TestNewOrder::test_non_exist_book_id PASSED [ 48%]
fe/test/test_new_order.py::TestNewOrder::test_low_stock_level PASSED [ 50%]
fe/test/test_new_order.py::TestNewOrder::test_ok PASSED [ 51%]
fe/test/test_new_order.py::TestNewOrder::test_non_exist_user_id PASSED [ 53%]
fe/test/test_new_order.py::TestNewOrder::test_non_exist_store_id PASSED [ 54%]
fe/test/test_password.py::TestPassword::test_ok PASSED [ 56%]
fe/test/test_password.py::TestPassword::test_error_password PASSED [ 57%]
fe/test/test_password.py::TestPassword::test_error_user_id PASSED [ 59%]
fe/test/test_payment.py::TestPayment::test_ok PASSED [ 60%]
fe/test/test_payment.py::TestPayment::test_authorization_error PASSED [ 62%]
fe/test/test_payment.py::TestPayment::test_not_suff_funds PASSED [ 63%]
fe/test/test_payment.py::TestPayment::test_repeat_pay PASSED [ 65%]
fe/test/test_receive.py::TestReceive::test_ok PASSED [ 66%]
fe/test/test_receive.py::TestReceive::test_order_error PASSED [ 68%]
fe/test/test_receive.py::TestReceive::test_authorization_error PASSED [ 69%]
fe/test/test_receive.py::TestReceive::test_books_not_send PASSED [ 71%]
fe/test/test_receive.py::TestReceive::test_books_repeat_receive PASSED [ 72%]
fe/test/test_register.py::TestRegister::test_register_ok PASSED [ 74%]
fe/test/test_register.py::TestRegister::test_unregister_ok PASSED [ 75%]
fe/test/test_register.py::TestRegister::test_unregister_error_authorization PASSED [ 77%]
fe/test/test_register.py::TestRegister::test_register_error_exist_user_id PASSED [ 78%]
fe/test/test_search.py::TestSearch::test_all_field_search PASSED [ 80%]
fe/test/test_search.py::TestSearch::test_pagination PASSED [ 81%]
fe/test/test_search.py::TestSearch::test_search_title PASSED [ 83%]
fe/test/test_search.py::TestSearch::test_search_title_in_store PASSED [ 84%]
fe/test/test_search.py::TestSearch::test_search_tag PASSED [ 86%]
fe/test/test_search.py::TestSearch::test_search_tag_in_store PASSED [ 87%]
fe/test/test_search.py::TestSearch::test_search_author PASSED [ 89%]
fe/test/test_search.py::TestSearch::test_search_author_in_store PASSED [ 90%]
fe/test/test_search.py::TestSearch::test_search_content PASSED [ 92%]
fe/test/test_search.py::TestSearch::test_search_content_in_store PASSED [ 93%]
fe/test/test_send.py::TestSend::test_ok PASSED [ 95%]
fe/test/test_send.py::TestSend::test_order_error PASSED [ 96%]
fe/test/test_send.py::TestSend::test_authorization_error PASSED [ 98%]
fe/test/test_send.py::TestSend::test_books_repeat_send PASSED [100%]
D:\CDMS\lab
s\project\bookstore20231107final\be\serve.py:19: UserWarning: The 'environ['werkzeug.serve
r.shutdown']' function is deprecated and will be removed in Werkzeug 2.1.
func()
2023-11-10 13:47:29,077 [Thread-4265 ] [INFO ] 127.0.0.1 - - [10/Nov/2023 13:47:29] "GET
/shutdown HTTP/1.1" 200 -

```

===== 66 passed in 284.80s (0:04:44) =====

frontend end test

No data to combine

Name	Stmts	Miss	Branch	BrPart	Cover
be\__init__.py	0	0	0	0	100%
be\app.py	3	3	2	0	0%

be\app.py	3	3	2	0	0%
be\model\__init__.py	0	0	0	0	100%
be\model\book.py	80	4	32	4	93%
be\model\buyer.py	238	47	110	23	78%
be\model\db_conn.py	19	0	6	0	100%
be\model\error.py	33	2	0	0	94%
be\model\seller.py	58	9	22	1	88%
be\model\store.py	22	0	0	0	100%
be\model\user.py	102	15	30	6	84%
be\serve.py	37	1	2	1	95%
be\view\__init__.py	0	0	0	0	100%
be\view\auth.py	42	0	0	0	100%
be\view\buyer.py	76	4	2	0	95%
be\view\search.py	86	4	32	12	86%
be\view\seller.py	38	0	0	0	100%
fe\__init__.py	0	0	0	0	100%
fe\access\__init__.py	0	0	0	0	100%
fe\access\auth.py	31	0	0	0	100%
fe\access\book.py	70	1	12	2	96%
fe\access\buyer.py	73	6	4	1	91%
fe\access\new_buyer.py	8	0	0	0	100%
fe\access\new_seller.py	8	0	0	0	100%
fe\access\search.py	53	0	0	0	100%
fe\access\seller.py	37	0	0	0	100%
fe\bench\__init__.py	0	0	0	0	100%
fe\bench\run.py	13	0	6	0	100%
fe\bench\session.py	47	0	12	1	98%
fe\bench\workload.py	125	1	22	2	98%
fe\conf.py	11	0	0	0	100%
fe\conftest.py	17	0	0	0	100%
fe\test\gen_book_data.py	49	0	16	0	100%
fe\test\test_add_book.py	36	0	10	0	100%
fe\test\test_add_funds.py	23	0	0	0	100%
fe\test\test_add_stock_level.py	39	0	10	0	100%
fe\test\test_bench.py	7	3	0	0	57%
fe\test\test_cancel_auto.py	54	1	4	1	97%
fe\test\test_cancel_order.py	82	1	4	1	98%
fe\test\test_create_store.py	20	0	0	0	100%
fe\test\test_history_order.py	65	10	14	2	82%
fe\test\test_login.py	28	0	0	0	100%
fe\test\test_new_order.py	40	0	0	0	100%
fe\test\test_password.py	33	0	0	0	100%
fe\test\test_payment.py	60	1	4	1	97%
fe\test\test_receive.py	58	1	4	1	97%
fe\test\test_register.py	31	0	0	0	100%
fe\test\test_search.py	98	0	0	0	100%
fe\test\test_send.py	47	1	4	1	96%
-----					
TOTAL	2097	115	364	60	92%

可以看到无论是基础功能还是拓展功能，只要是实验要求中所提到的，我们均实现并通过了。

由于在演示基础功能时用的还是最终完成的代码，所以有很多拓展功能之代码并未覆盖到，所以看起来覆盖率较低。但是随后带上拓展功能运行时，总的覆盖率就达到了92%，说明我们的代码中不仅测试用例覆盖了尽可能多的情况，而且代码冗余较少、模块与接口的利用率较高，算是达到了一个令人满意的水平。

## 7.3版本管理

我们使用git作为版本管理的工具，团队协作的效率大大提升了。分模块地完成、优化、测试项目，分工更加明确的同时也对项目的快速推进贡献了力量。这个项目的Github链接为：<https://github.com/xixohoej/bookstore2>

然而我们在完成实验时是在现实中聚在一起讨论的，有的同学可能完成了某模块主体的功能之后碰到了各种各样的问题，但是在一起的探讨中由其他同学发现并解决了问题，并率先地在github中提交了。我们仓库的commit和贡献情况参考性不大，这个在未来还是需要多多改进。

## 7.4组员感想

郭夏辉:

虽然本次实验确实加深了我对以MongoDB为常用工具的文档数据库的理解，在实践中更加深刻地掌握了相关用法、相应原理，但是我觉得这次实验中大量的努力是重复且枯燥的——围绕着几个id一直找来找去，为了增强程序的健壮性而考虑几乎所有的可能情况，而且这个实验本身还很同质化，感受不到太大的差异性.....由于做实验时还没学多少关系数据库的知识，希望自己在学完了之后再回过头来看它与文档数据库的异同。

包亦晟:

本次项目的经历还是蛮曲折的。从一开始的啥都不懂到后来经过与组员的讨论以及助教的帮助，总算是渐渐上手。记得有一次调bug调到凌晨，当天还有早八，真是痛并快乐着

朱天祥:

搜索功能是本次是实验中的难点，我在理解其他两位组员的基础上，去结合各种字符处理的情况来考虑，对文档数据库应用的理解更深了。数据库系统是一套相对来说独立于前端和后端的系统，在这次实验中我也测试并设计了相关的业务情况使之更符合逻辑，我意识到在实际的应用中也应该把这三者紧密结合，使系统整体自然而高效。