

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-关系数据库
姓名：郭夏辉	学号： 10211900416	年级：2022

一.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，已有的代码实现了前60%功能以及对应的测试用例代码，我们需要将其修改为**核心数据使用关系型数据库（PostgreSQL 或 MySQL 数据库），blob 数据（如图片和大段的文字描述）可以分离出来存其它 NoSQL 数据库或文件系统的形式**。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 提取码：hj6q

2.允许向接口中增加或修改参数，允许修改 HTTP 方法，允许增加新的测试接口，请尽量不要修改现有接口的 url 或删除现有接口，请根据设计合理的拓展接口（加分项+2 ~ 5分）。测试程序如果有问题可以提bug（加分项，每提1个 bug +2, 提1个 pull request +5）。

3.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，有较高的覆盖率是加分项 +2~5）。

4.尽量使用正确的软件工程方法及工具，如，版本控制，测试驱动开发（利用版本控制是加分项 +2~5）

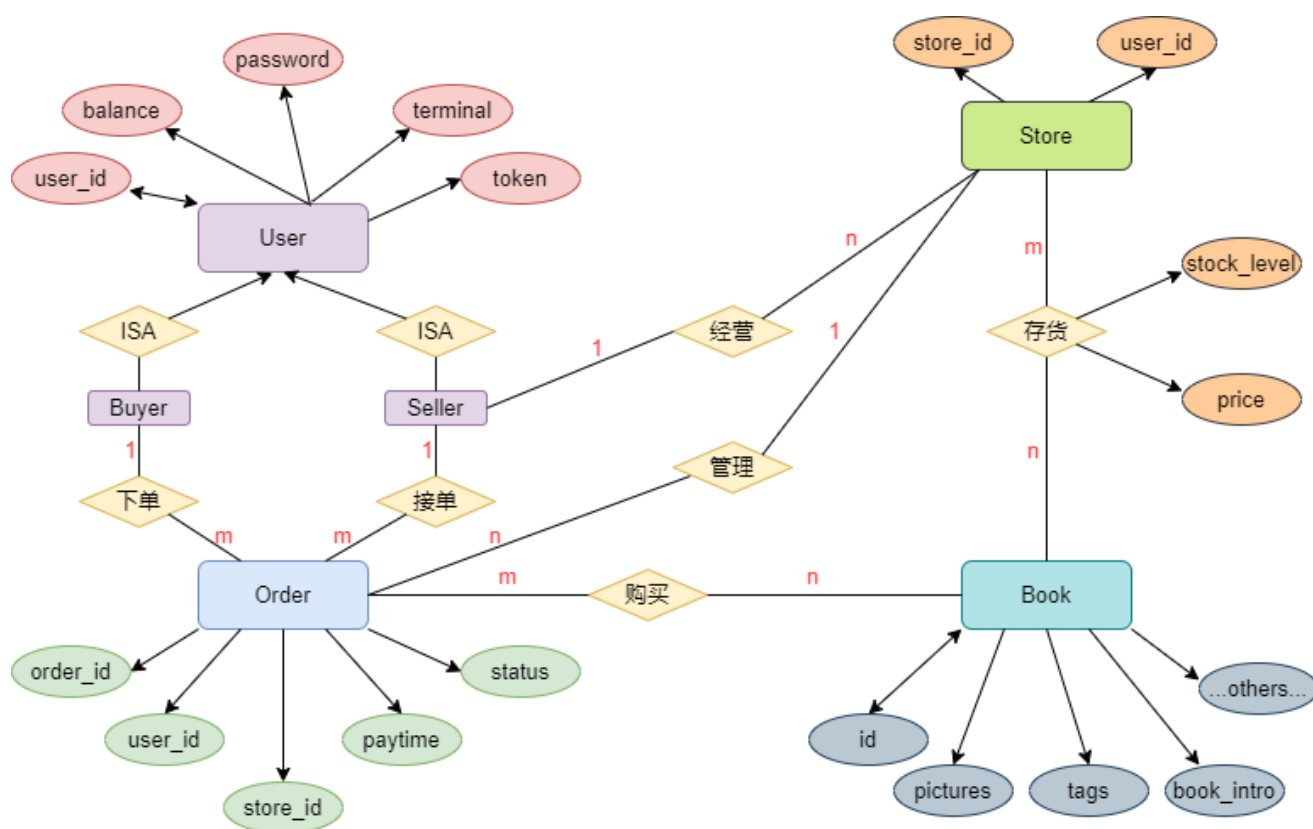
5.后端使用技术，实现语言不限；**不要复制**这个项目上的后端代码（不是正确的实践，减分项 -2~5）

6.不需要实现界面，只需通过代码测试体现功能与正确性

7.实现完整度较高，全部测试通过，效率合理；正确地使用数据库和设计分析工具，ER图，从ER图导出关系模式，规范化，事务处理，索引等

二.数据库设计

ER图



1. 实体类

实体类：用户User(含有子类买家Buyer和卖家Seller)、商铺Store、订单Order和书籍Book。

2. 属性

user: user_id (唯一属性)，单值属性密码、余额、终端和token

登陆的用户token不会重复，未登录的用户token为空。

order: order_id (唯一属性)，单值属性订单应付款金额和订单创建时间、状态

对状态的解释：0为已付款，1为已发货，2为已收货, 4为交易关闭。

store：唯一属性store_id， user_id。

book：id（唯一属性）， 单值属性标题、作者、出版商、isbn号、原价等属性。

3. 关系

订单和书籍：一个订单可以包括购买的多套书籍，一套书也可被多笔订单所购买。

订单和商铺：一家商铺可以拥有多笔订单，但一笔订单只能对应一家商铺。

买家和订单：一个买家可以对应多笔订单，但一笔订单只能对应一个买家。

卖家和订单：一个卖家可以对应多笔订单，但一笔订单只能对应一个卖家。

卖家和商铺：一个卖家可以拥有多个商铺，但一家商铺只能对应一个卖家。

商铺和书籍：一家商铺可以存有多款书籍，一款书也可以被多家商铺存有。

关系模式和索引设计

用户表(user)

属性名称	类型	码
user_id	String	(primary_key)
password	String	
balance	Integer	
token	String	
terminal	String	

user_id是主键，通过user_id进行查找。

例如登录时查看密码是否正确，用户充值时查看还剩多少余额

用户和商铺关系表(store)

属性名称	类型	码
store_id	String	(primary_key)
user_id	String	(Foreign Key)

store_id是主键， user_id作为外键，这样设置可以加快商家查看店铺操作的速度。

商铺详情表(store_detail)

属性名称	类型	码
store_id	String	(primary_key, Foreign Key)
book_id	String	(primary_key, Foreign Key)
stock_level	Integer	
price	Integer	

store_id和book_id组成联合主键，它们两者也都作为外键。这样设置，可以加快搜索时根据书找到对应的店铺和根据店铺找到店铺内的书。

书籍表(book)

属性名称	类型	码
book_id	String	(primary_key)
title	String	
author	String	
publisher	String	
...		
original_price	Integer	
picture	LargeBinary	

book_id是主键。

订单表(order)

属性名称	类型	码
order_id	String	(primary_key)
user_id	String	(Foreign Key)
store_id	String	(Foreign Key)
paytime	DateTime	
status	Integer	

order_id是主键，user_id和store_id作为外键。

status解释：0为已付款，1为已发货，2为已收货, 4为已取消

订单详情表(order_detail)

属性名称	类型	码
order_id	String	(primary_key)
book_id	String	(primary_key, Foreign Key)
count	Integer	

order_id和book_id组成联合主键，其中book_id是外键。

由于订单和书本是多对多关系，因此我们需要建立一张额外的订单详情表。

待付款订单表(order_to_pay)

属性名称	类型	码
order_id	String	(primary_key)
user_id	String	(Foreign Key)
store_id	String	(Foreign Key)
paytime	DateTime	

order_id是主键， user_id和store_id是外键。

由于我引入了自动取消超时订单功能，程序设定为每秒查询一次未付款订单。为了提高检索效率，降低检索复杂度，我将未付款订单的表单独分离出来，这里也不再添加额外设置索引了。

四张倒排索引表 (search_tags,search_author,search_title,search_book_intro)

这四张表各自都有三个属性，以search_author为例，它的三个属性是search_id (Integer) ,author (String[]) 和book_id (String)

这四张表主要用于实现搜索过程中的索引。

- search_id：从1开始逐渐增加 (primary_key)
- author：经过分词等操作后得到的词项数组 (primary_key)
- book_id：词项对应的书籍 (Foreign Key)

为什么我要这样设计？

我从ER图中抽象出了关系模式，接下来从理论的角度来讲一下我为什么要这样设计：

- 设置表尽量满足第三范式，尽量避免插入及删除异常带来的问题，尽力保证了正确性。因为我在基本功能的select操作中几乎只用到了主键查询，所以增加冗余只会增加更新与插入操作的代价，我觉得自己这样设计因此也是有较好合理性的。
- 对于四个实体类而言，每个实体类对应着一张表，其id作为表的主键；由于一个用户既可以是卖家也可以是买家，所以这两类应该用一张用户表来存；两个多对多关系分别对应一张表，用两个id联合形成了主键。
- 其次，考虑到拓展功能，我需要完成自动取消订单，超时未付款的功能将被取消，这样的情况下已下单但是未付款的元组经常会被遍历，为了提高查询性能，我将已下单未付款的订单拿出来单独形成一张表。
- 我只使用了主键的自带索引，没有在主键以外的属性上添加索引。因为我在实现基本功能的select操作时几乎只用到了主键查询，添加索引大概率会增加更新与插入操作的代价。
- 由于一本书可能对应多个词项，一个词项可以对应多本书，同一个词项只能对应同一本书，所以我将search_id和author两个属性作为倒排索引表的联合主键。与此同时，book_id作倒排索引表的外键，加快了book表中查询书籍信息的速度。

文档型数据库到关系型数据库的改动 & 改动的理由

文档数据库阶段我们是怎么设计的

之前的实验中，我的小组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
}

```

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。

对比及分析

文档数据库存储的是一系列文档，更像是一种面向对象的一种表示模式；关系数据库用表格来存储数据，我们将表格的每一列看成是一个属性。虽然文档数据库以其灵活的结构加速了对数据结构的读写，但是面对复杂逻辑的情况下展现了更差的事务处理特点。关系数据库支持原子性操作，要么不做，要么做完。依托ORM我使用关系数据库来进行事务处理，确保了数据的正确性。依托主码，外码等约束，关系数据库保证了文档数据库所不能保证的数据完整性。

在本次实验中，具体的改动我已经在实验报告前面的部分详细阐述了，这里就不再赘述了，我也花了很多篇幅来讲解那样的改动所产生的增益效果，但我还是想来谈一谈表的设计。我记得上次实验过程中，我们经常在一个文档集中找到一个id，再到另外一个文档集中凭借这个id查询到对应的文档。但是在关系数据库中我显然没必要这么麻烦了，多个表连接起来，直接就能在一个查询中把信息串联起来了，这也是我上次实验中还删去user_store但是这次实验中又加了回来（改名为store）的核心原因。当然现在关系数据库不如之前文档数据库那样结构灵活了，所以我根据具体的业务（比如说我要实现自动取消功能，频繁地要去查待付款订单）设计了一些新的表，不能再像之前那样几个文档集就可以实现几乎所有的需求。

总的来说，文档数据库和关系数据库这两种数据模型各有千秋，在设计时要结合实际问题与两种模型的特点，这样才能最大化产品的效果。

三.功能实现

本次实验中我结合之前实验课的内容，使用了ORM来实现对应数据库功能。sqlalchemy保证在session commit之前，如果遇到异常，事务将会回滚，所以我需要在正确的位置commit，以此来保证事务的正确性。但是使用ORM可以极大地方便我们使用SQL语句。

但是在介绍功能实现细节之前，我想说我们上次实验报告中那冗长的接口和流程感觉放在这里并不能很好地起到介绍功能——如果真的要细节，代码已经很清楚地表达了。因此，我在这里只是围绕各个功能来大概梳理一下它们的流程，愿看官海涵。

基本功能（60%）

User

1. 注册

- 根据user_id判断该用户名是否已经存在。
- 插入user_id、password、balance、token、terminal信息至user表。

2. 登录

- 根据user_id获取用户密码，与用户输入密码对比。
- 更新token，terminal。

3. 登出

- 根据user_id查询该user是否处于登陆状态。

- 更新token。

4. 注销

- 根据user_id查询该user是否存在。
- 删除对应user表中条目。

5. 更改密码

- 根据user_id获取用户原有密码，与用户输入的旧密码对比。
- 若相同，更新用户密码。

Seller

1. 创建店铺

- 检查user_id和store_id是否已存在。
- 插入用户id，新建店铺store_id至user_store表。

2. 上架图书

- 检查user_id和store_id是否已存在。
- 根据book_id从book表查询是否存在对应book。
- 若不存在，首先将书本信息插入book表。
- 将store_id, book_id, 出售价格插入store表。

3. 添加库存

- 检查user_id、store_id和book_id是否已存在。
- 根据store_id, book_id寻找对应店家书本库存，并在store表中更新库存。

Buyer

1. 下单

- 根据买家下单信息user_id和store_id在store表中查找商户信息，并进入store_detail表中查看店铺具体信息。
- 根据解析买家传入的book_id和count信息，在store_detail表中查找商户是否存在对应书籍和足够的库存。
- 若库存充足，则减少对应数量的库存，并在order_detail表中插入对应的订单信息：order_id, book_id, count。
- 当买家一个订单中的所有商品都可购买，并且购买记录均已加入order_detail后，在order_to_pay表（已下单未付款表）中插入订单记录，记录相应order_id, user_id, store_id, paytime。

2. 付款

- 查询在order_to_pay表中是否存在属于该买家的待支付订单，若存在，则获取相应的store_id。
- 对买家信息进行校验，检查user_id及password是否相对应或正确。
- 校验通过，检查买家余额是否大于待支付订单总价，若资金足够，则付款成功，并根据卖家user_id在usr表中给卖家增加资金，否则失败。
- 对成功付款的订单，将其记录转移到order表中，更新paytime为订单支付时间，并加入status=0，表示订单已付款未发货。并将其从order_to_pay表中删除。

3. 用户充值

- 根据user_id获取用户密码，并与用户输入密码进行对比。
- 若密码正确，根据用户输入add_value在usr表中更新其余额。

扩展功能（40%）

收发货

1. 卖家发货

- 检查order_id是否存在。
- 查询order的状态是否为已付款。
- 看store的店铺主是不是seller。
- 将order的状态置为1。

2. 买家收货

- 检查order_id是否存在于order表中。
- 检查order的status是否为0或2。
- 查看user_id与buyer_id是否一致。
- 若以上都满足，置order的status为2。

图书搜索

这里我花了很多时间在调试，也是所有功能模块中介绍地最详细的。

搜索方式：搜索标题、搜索作者、搜索标签、搜索内容关键字（都支持模糊搜索）

搜索范围：全局搜索和店铺内搜索

图书搜索主要围绕的是那四张倒排索引表

(search_tags,search_author,search_title,search_book_intro)

搜索方式

- 1.根据标题搜索：在构建标题的倒排索引过程中，我首先只是记录标题内容。后来考虑到实际生活中**读者不一定能完全记住过长的图书标题**，因此我还**提取了标题内部的信息**。考虑到数据集中的图书标题主要是中文的，我特别地采用jieba分词方法，在精确搜索的前提下，对长词进行再次切分，可以对搜索的准确性有进一步提升。在处理英文标题时比较简单，直接按空格提取单词，再加上标题本身即可。特别要注意，如果分词结果不包含标题，需要加上标题本身。
- 2.根据作者搜索：在完成了完整命名搜索功能的基础之上，我还是结合实际情况进行了一些创新。我们在实际生活中完整记住人名其实也有点难度，但是我们记录时有一定倾向性——人名前面部分被记住的概率要比后部分大得多。因此，我基于模糊提示词，构建倒排索引，依次提取人名的前几个字或字母来构建词项。
- 3.根据标签搜索：由于标签已经存在于数据集中，在构建倒排索引时就使用即可。
- 4.根据内容关键词进行搜索：这个主要还是从方便使用者的方向来考虑，就是可能对一本书只知道一点模糊的只言片语，但是忘记了作者是谁，忘记了标题是什么，去按标签查找也太庞大了，这个情况下应该怎么办呢？所幸还有书籍的内容简介，在此之中我提取了一些内容关键词，大大便利了上述提到的情况之搜索。对书的内容简介，考虑到基本也是中文的，我先用jieba分词，再去除停顿词，最后用从中提取TF-IDF最大的20个关键词之后，再构建倒排索引。

搜索范围

1. 全局：输入搜索方式（标题、作者或关键词），在对应的倒排索引表中查找对应的book_id，再在书的表中根据book_id找到这本书的相关信息；
2. 店铺内：输入搜索方式（标题、作者或关键词）和store_id，在对应的倒排索引表中查找对应的book_id，并且在store_detail店铺库存详情表中根据store_id找出所有的book_id，取book_id的交集再在书的表中根据book_id找到这本书的相关信息。

分页显示结果

由于某些查询词对应了超多本书，返回结果太大了，所以我采用分页显示查询结果的方式。

每次查询时，传入一个参数page（大于等于1，一般取1），实现每次返回10行结果作为一页。

查询和取消订单功能

1. 买家取消订单
 - 根据order_id和user_id在order_to_pay及order表中查找订单信息。
 - 若订单记录存在于order_to_pay表中，则该订单只下单未付款，只需获取store_id后为卖家加回库存，再删掉表中记录即可。最后，在order表中加上相应记录，设置status为4，表示订单已关闭。
 - 若订单记录存在于order表中，检测该记录status是否对应0（已付款未发货），因为此处需注意已发货和已收货的订单不能取消。之后，获取对应卖家user_id和store_id，加回库存的同时一同完成资金转移。最后，更改order表中记录，设status为4。
2. 买家查询历史订单

- 根据买家user_id在order及order_to_pay表中进行查询，并根据paytime按时间排列。
- 若查询记录不为空，则依次获取查询所得的每条记录的信息 (order_id, user_id, store_id, order_detail: (book_id, count, single_price), paytime, status)，并计算每格订单的总价total_price，将每条记录以字典的形式插入historys列表中。
- 返回historys (当无订单记录时，返回一个空数组)，并将其包装成json对象。

3. 自动检测超时未付款订单并取消

- 调用Timer类实现自动取消订单。Timer是Thread的子类，通过Timer(t, fun)函数设置每t秒调用一次fun函数。并且，我们通过start()和cancel()函数设置线程开始或取消执行。
- 每次调用函数，将查询在order_to_pay表中是否存在待支付订单。对于所有存在的记录，我们获取其记录中的paytime，并与当前时间做差，若时间差大于15min，则自动取消关闭该订单。
- 对每一个超时应关闭的订单，获取记录对应的order_id, user_id, store_id，并根据store_id将卖家商铺的库存加回。
- 最后，我们将此条要关闭订单的对应记录加入order表，并设置status为4，之后删除order_to_pay表中的记录。

四.测试及性能分析

测试用例介绍

在fe/test文件夹下，除了基础的33个测试点，我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。

test_close_order

1. test_ok:

- **情况：** 测试正常关闭订单的情况。
- **操作：** 调用 `close_order` 方法关闭订单。
- **预期结果：** 返回状态码为200，表示订单关闭成功。

2. test_no_order:

- **情况：** 测试关闭不存在的订单。
- **操作：** 用第二个买家尝试关闭一个不存在的订单。
- **预期结果：** 返回状态码不为200，表示关闭失败。

3. test_close_paid:

- **情况：** 测试已支付订单的关闭。
- **操作：** 为订单支付，然后尝试关闭订单。

- **预期结果：** 返回状态码为200，表示订单关闭成功。

4. **test_authorization_error:**

- **情况：** 测试授权错误的情况。
- **操作：** 修改买家密码或ID，然后尝试关闭订单。
- **预期结果：** 返回状态码不为200，表示关闭失败。

5. **test_repeat_close:**

- **情况：** 测试重复关闭订单的情况。
- **操作：** 尝试两次关闭同一订单。
- **预期结果：** 第一次返回状态码为200，第二次返回状态码不为200。

6. **test_close_send:**

- **情况：** 测试已发货订单的关闭。
- **操作：** 为订单支付，卖家发货，然后尝试关闭订单。
- **预期结果：** 返回状态码不为200，表示关闭失败。

7. **test_close_received:**

- **情况：** 测试已收货订单的关闭。
- **操作：** 为订单支付，卖家发货，买家收货，然后尝试关闭订单。
- **预期结果：** 返回状态码不为200，表示关闭失败。

test_order

1. **test_ok:**

- **情况：** 测试正常搜索订单的情况。
- **操作：** 生成随机数量的订单，包括随机生成订单状态，进行支付、发货、收货等操作。
- **预期结果：** 返回状态码为200，表示订单搜索成功。

2. **test_authorization_error:**

- **情况：** 测试授权错误的情况。
- **操作：** 修改买家密码或ID，然后尝试搜索订单。
- **预期结果：** 返回状态码不为200，表示搜索失败。

3. **test_no_history:**

- **情况：** 测试没有历史订单的情况。
- **操作：** 使用一个没有历史订单的买家账号搜索订单。
- **预期结果：** 返回状态码为200，表示搜索成功，但是没有历史订单信息。

test_receive_books

1. test_ok:

- **情况：** 测试正常接收已发货的订单。
- **操作：** 卖家发货后，买家尝试收货。
- **预期结果：** 返回状态码为200，表示收货成功。

2. test_false_buyer:

- **情况：** 测试使用错误的买家ID尝试接收订单。
- **操作：** 卖家发货后，使用错误的买家ID尝试收货。
- **预期结果：** 返回状态码不为200，表示接收失败。

3. test_non_exist_order:

- **情况：** 测试尝试接收不存在的订单。
- **操作：** 卖家发货后，尝试使用不存在的订单ID进行收货。
- **预期结果：** 返回状态码不为200，表示接收失败。

4. test_repeat_receive_books:

- **情况：** 测试重复接收订单的情况。
- **操作：** 卖家发货后，买家尝试两次收货同一订单。
- **预期结果：** 第一次返回状态码为200，第二次返回状态码不为200。

5. test_can_not_receive:

- **情况：** 测试未发货情况下尝试接收订单。
- **操作：** 买家尝试在未发货状态下接收订单。
- **预期结果：** 返回状态码不为200，表示接收失败。

test_send_books

1. test_ok:

- **情况：** 测试正常发货的情况。
- **操作：** 卖家尝试发货。
- **预期结果：** 返回状态码为200，表示发货成功。

2. test_false_seller:

- **情况：** 测试使用错误的卖家ID尝试发货。
- **操作：** 使用错误的卖家ID尝试发货。
- **预期结果：** 返回状态码不为200，表示发货失败。

3. test_non_exist_order:

- **情况：** 测试尝试发货不存在的订单。

- **操作：** 尝试使用不存在的订单ID进行发货。
- **预期结果：** 返回状态码不为200，表示发货失败。

4. **test_repeat_send_books:**

- **情况：** 测试重复发货订单的情况。
- **操作：** 尝试两次发货同一订单。
- **预期结果：** 第一次返回状态码为200，第二次返回状态码不为200。

5. **test_can_not_send:**

- **情况：** 测试已发货的订单再次发货的情况。
- **操作：** 卖家先发货，买家收货后，再尝试卖家发货。
- **预期结果：** 返回状态码不为200，表示发货失败。

6. **test_none_order:**

- **情况：** 测试尝试发货不存在的订单。
- **操作：** 尝试使用不存在的订单ID进行发货。
- **预期结果：** 返回状态码不为200，表示发货失败。

7. **test_no_seller:**

- **情况：** 测试使用不存在的卖家ID尝试发货。
- **操作：** 使用不存在的卖家ID尝试发货。
- **预期结果：** 返回状态码不为200，表示发货失败。

test_search

1. **test_search_author:**

- **情况：** 测试搜索作者的功能。
- **操作：** 分别使用作者名进行精确和模糊搜索。
- **预期结果：** 返回状态码为200，表示搜索成功。

2. **test_search_book_intro:**

- **情况：** 测试搜索图书简介的功能。
- **操作：** 分别使用图书简介进行精确和模糊搜索。
- **预期结果：** 返回状态码为200，表示搜索成功。

3. **test_search_tags:**

- **情况：** 测试搜索图书标签的功能。
- **操作：** 分别使用图书标签进行精确和模糊搜索。
- **预期结果：** 返回状态码为200，表示搜索成功。

4. **test_search_title:**

- **情况：** 测试搜索图书标题的功能。
- **操作：** 分别使用图书标题进行精确和模糊搜索。
- **预期结果：** 返回状态码为200，表示搜索成功。

5. **test_search_author_in_store:**

- **情况：** 测试在指定店铺中搜索作者的功能。
- **操作：** 分别使用作者名在指定店铺中进行精确和模糊搜索。
- **预期结果：** 返回状态码为200，表示搜索成功。

6. **test_search_book_intro_in_store:**

- **情况：** 测试在指定店铺中搜索图书简介的功能。
- **操作：** 分别使用图书简介在指定店铺中进行精确和模糊搜索。
- **预期结果：** 返回状态码为200，表示搜索成功。

7. **test_search_tags_in_store:**

- **情况：** 测试在指定店铺中搜索图书标签的功能。
- **操作：** 分别使用图书标签在指定店铺中进行精确和模糊搜索。
- **预期结果：** 返回状态码为200，表示搜索成功。

8. **test_search_title_in_store:**

- **情况：** 测试在指定店铺中搜索图书标题的功能。
- **操作：** 分别使用图书标题在指定店铺中进行精确和模糊搜索。
- **预期结果：** 返回状态码为200，表示搜索成功。

9. **test_search_author_vague:**

- **情况：** 测试模糊搜索作者名的功能。
- **操作：** 使用包含作者名部分关键字的搜索。
- **预期结果：** 返回状态码为200，表示搜索成功。

10. **test_search_book_intro_vague:**

- **情况：** 测试模糊搜索图书简介的功能。
- **操作：** 使用包含图书简介部分关键字的搜索。
- **预期结果：** 返回状态码为200，表示搜索成功。

11. **test_search_tags_vague:**

- **情况：** 测试模糊搜索图书标签的功能。
- **操作：** 使用包含图书标签部分关键字的搜索。
- **预期结果：** 返回状态码为200，表示搜索成功。

12. **test_search_title_vague:**

- **情况：** 测试模糊搜索图书标题的功能。
- **操作：** 使用包含图书标题部分关键字的搜索。
- **预期结果：** 返回状态码为200，表示搜索成功。

13. test_search_author_vague_in_store:

- **情况：** 测试在指定店铺中模糊搜索作者名的功能。
- **操作：** 使用包含作者名部分关键字在指定店铺中进行搜索。
- **预期结果：** 返回状态码为200，表示搜索成功。

14. test_search_book_intro_vague_in_store:

- **情况：** 测试在指定店铺中模糊搜索图书简介的功能。
- **操作：** 使用包含图书简介部分关键字在指定店铺中进行搜索。
- **预期结果：** 返回状态码为200，表示搜索成功。

15. test_search_tags_vague_in_store:

- **情况：** 测试在指定店铺中模糊搜索图书标签的功能。
- **操作：** 使用包含图书标签部分关键字在指定店铺中进行搜索。
- **预期结果：** 返回状态码为200，表示搜索成功。

16. test_search_title_vague_in_store:

- **情况：** 测试在指定店铺中模糊搜索图书标题的功能。
- **操作：** 使用包含图书标题部分关键字在指定店铺中进行搜索。
- **预期结果：** 返回状态码为200，表示搜索成功。

接口介绍

后端接口在 be/view/ ,前端接口则在 fe/access

后端接口主要是三部分，分别是auth,buyer和seller；前端接口主要是6部分，分别是auth,buyer,seller,new_buyer,new_seller,search

后端auth

- **登录 (/auth/login):** 用户提交用户名、密码和终端信息，进行登录验证。返回消息和令牌。
- **登出 (/auth/logout):** 用户提交用户ID和令牌，进行登出操作。返回消息。
- **注册 (/auth/register):** 用户提交用户名和密码，进行用户注册。返回消息。
- **注销 (/auth/unregister):** 用户提交用户名和密码，进行用户注销。返回消息。
- **修改密码 (/auth/password):** 用户提交用户ID、旧密码和新密码，进行密码修改。返回消息。

- **搜索作者** (/auth/search_author): 用户提交作者名和页码, 进行作者搜索。返回消息。
- **搜索图书简介** (/auth/search_book_intro): 用户提交图书简介和页码, 进行图书简介搜索。返回消息。
- **搜索标签** (/auth/search_tags): 用户提交标签和页码, 进行标签搜索。返回消息。
- **搜索标题** (/auth/search_title): 用户提交标题和页码, 进行标题搜索。返回消息。
- **在商店中搜索作者** (/auth/search_author_in_store): 用户提交作者名、商店ID和页码, 进行商店中作者搜索。返回消息。
- **在商店中搜索图书简介** (/auth/search_book_intro_in_store): 用户提交图书简介、商店ID和页码, 进行商店中图书简介搜索。返回消息。
- **在商店中搜索标签** (/auth/search_tags_in_store): 用户提交标签、商店ID和页码, 进行商店中标签搜索。返回消息。
- **在商店中搜索标题** (/auth/search_title_in_store): 用户提交标题、商店ID和页码, 进行商店中标题搜索。返回消息。

后端buyer

- **新建订单** (/buyer/new_order): 买家提交用户ID、商店ID和书籍信息, 创建新订单。返回消息和订单ID。
- **支付订单** (/buyer/payment): 买家提交用户ID、订单ID和支付密码, 进行订单支付。返回消息。
- **充值** (/buyer/add_funds): 买家提交用户ID、支付密码和充值金额, 进行账户充值。返回消息。
- **确认收货** (/buyer/receive_books): 买家提交用户ID和订单ID, 确认收到书籍。返回消息。
- **关闭订单** (/buyer/close_order): 买家提交用户ID、支付密码和订单ID, 关闭订单。返回消息。
- **搜索订单** (/buyer/search_order): 买家提交用户ID和支付密码, 查询订单历史记录。返回消息和历史订单信息。

后端seller

- **创建商店** (/seller/create_store): 卖家提交用户ID和商店ID, 创建新商店。返回消息。
- **添加图书** (/seller/add_book): 卖家提交用户ID、商店ID、图书信息和库存水平, 添加新图书到商店。返回消息。
- **增加库存** (/seller/add_stock_level): 卖家提交用户ID、商店ID、图书ID和增加的库存数量, 增加图书库存。返回消息。
- **发货** (/seller/send_books): 卖家提交卖家ID和订单ID, 进行图书发货操作。返回消息。

前端auth

1. 登录 (login):

- 发送登录请求至远程服务, 传递用户ID、密码和终端信息。
- 返回一个包含HTTP状态码和令牌的元组。

2. 注册 (register):

- 发送注册请求至远程服务, 传递用户ID和密码。
- 返回一个包含HTTP状态码的整数。

3. 修改密码 (password):

- 发送修改密码请求至远程服务, 传递用户ID、旧密码和新密码。
- 返回一个包含HTTP状态码的整数。

4. 登出 (logout):

- 发送登出请求至远程服务, 传递用户ID和令牌。
- 返回一个包含HTTP状态码的整数。

5. 注销 (unregister):

- 发送注销请求至远程服务, 传递用户ID和密码。
- 返回一个包含HTTP状态码的整数。

前端buyer

1. 新建订单 (new_order):

- 发送新建订单请求至远程服务, 传递用户ID、商店ID和书籍信息。
- 返回一个包含 HTTP 状态码和订单ID的元组。

2. 支付订单 (payment):

- 发送支付订单请求至远程服务, 传递用户ID、密码和订单ID。
- 返回一个包含 HTTP 状态码的整数。

3. 充值 (add_funds):

- 发送充值请求至远程服务, 传递用户ID、密码和充值金额。
- 返回一个包含 HTTP 状态码的整数。

4. 确认收货 (receive_books):

- 发送确认收货请求至远程服务, 传递买家ID和订单ID。
- 返回一个包含 HTTP 状态码的整数。

5. 关闭订单 (close_order):

- 发送关闭订单请求至远程服务, 传递用户ID、密码和订单ID。

- 返回一个包含 HTTP 状态码的整数。

6. 搜索订单 (`search_order`):

- 发送搜索订单请求至远程服务，传递用户ID和密码。
- 返回一个包含 HTTP 状态码的整数。

前端seller

1. 创建商店 (`create_store`):

- 发送创建商店请求至远程服务，传递卖家ID和商店ID。
- 返回一个包含 HTTP 状态码的整数。

2. 添加图书 (`add_book`):

- 发送添加图书请求至远程服务，传递卖家ID、商店ID、库存水平和图书信息。
- 返回一个包含 HTTP 状态码的整数。

3. 增加库存 (`add_stock_level`):

- 发送增加库存请求至远程服务，传递卖家ID、商店ID、图书ID和增加的库存数量。
- 返回一个包含 HTTP 状态码的整数。

4. 发货 (`send_books`):

- 发送发货请求至远程服务，传递卖家ID和订单ID。
- 返回一个包含 HTTP 状态码的整数。

前端new_buyer

输入:

- `user_id`: 新买家的用户ID
- `password`: 新买家的密码

功能:

- 使用 `auth.Auth` 类进行用户注册，创建一个新买家。
- 注册成功后，使用 `buyer.Buyer` 类初始化买家对象。
- 返回初始化后的买家对象。

前端new_seller

输入:

- `user_id`: 新卖家的用户ID
- `password`: 新卖家的密码

功能:

- 使用 `auth.Auth` 类进行用户注册，创建一个新卖家。
- 注册成功后，使用 `seller.Seller` 类初始化卖家对象。
- 返回初始化后的卖家对象。

前端search

- 提供了多个搜索方法，包括根据作者、图书简介、标签、标题等条件进行搜索，以及在特定商店内进行搜索。由于数量较多我就不在此赘述了。
- 每个搜索方法接受相应的搜索条件和分页信息，通过 POST 请求将条件发送至远程服务进行搜索。

如何启动测试？

首先需要安装postgresql数据库，并建库bookstore；当然还要 `pip install -r requirements.txt` 安装依赖

然后项目根目录下执行 `python init_database.py` 初始化数据库。（这个要根据实际情况修改密码，完成本次实验时我设置的密码是1026）再执行 `bash script/test.sh` 执行测试，执行测试的结果如下所示：

```
tom@DESKTOP-JQ5TSCF MINGW64 /d/database/labs/bookstore2/bookstore
$ python init_database.py
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\tom\AppData\Local\Temp\jieba.cache
Loading model cost 0.701 seconds.
Prefix dict has been built successfully.
```

```

tom@DESKTOP-JQ5TSCF MINGW64 /d/database/labs/bookstore2/bookstore
$ 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:\database\labs\bookstore2\bookstore
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
2023-12-23 11:35:16,178 [Thread-1 (ru)] [INFO ] * Running on http://127.0.0.1:5
000/ (Press CTRL+C to quit)
collected 71 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 [ 2
%]
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 [ 5%
]
fe/test/test_add_funds.py::TestAddFunds::test_ok PASSED [ 7%]
fe/test/test_add_funds.py::TestAddFunds::test_error_user_id PASSED [ 8%]
fe/test/test_add_funds.py::TestAddFunds::test_error_password PASSED [ 9%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_error_user_id PASSED [
11%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_error_store_id PASSED [
12%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_error_book_id PASSED [
14%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_ok PASSED [ 15%]
fe/test/test_bench.py::test_bench PASSED [ 16%]
fe/test/test_close_order.py::TestCloseOrder::test_ok PASSED [ 18%]
fe/test/test_close_order.py::TestCloseOrder::test_no_order PASSED [ 19%]
fe/test/test_close_order.py::TestCloseOrder::test_close_paid PASSED [ 21%]
fe/test/test_close_order.py::TestCloseOrder::test_authorization_error PASSED [ 2
2%]
fe/test/test_close_order.py::TestCloseOrder::test_repeat_close PASSED [ 23%]
fe/test/test_close_order.py::TestCloseOrder::test_close_send PASSED [ 25%]
fe/test/test_close_order.py::TestCloseOrder::test_close_received PASSED [ 26%]
fe/test/test_create_store.py::TestCreateStore::test_ok PASSED [ 28%]
fe/test/test_create_store.py::TestCreateStore::test_error_exist_store_id PASSED
[ 29%]
fe/test/test_login.py::TestLogin::test_ok PASSED [ 30%]
fe/test/test_login.py::TestLogin::test_error_user_id PASSED [ 32%]

```



```

fe/test/test_login.py::TestLogin::test_error_user_id PASSED [ 32%]
fe/test/test_login.py::TestLogin::test_error_password PASSED [ 33%]
fe/test/test_new_order.py::TestNewOrder::test_non_exist_book_id PASSED [ 35%]
fe/test/test_new_order.py::TestNewOrder::test_low_stock_level PASSED [ 36%]
fe/test/test_new_order.py::TestNewOrder::test_ok PASSED [ 38%]
fe/test/test_new_order.py::TestNewOrder::test_non_exist_user_id PASSED [ 39%]
fe/test/test_new_order.py::TestNewOrder::test_non_exist_store_id PASSED [ 40%]
fe/test/test_order.py::TestOrder::test_ok PASSED [ 42%]
fe/test/test_order.py::TestOrder::test_authorization_error PASSED [ 43%]
fe/test/test_order.py::TestOrder::test_no_history PASSED [ 45%]
fe/test/test_password.py::TestPassword::test_ok PASSED [ 46%]
fe/test/test_password.py::TestPassword::test_error_password PASSED [ 47%]
fe/test/test_password.py::TestPassword::test_error_user_id PASSED [ 49%]
fe/test/test_payment.py::TestPayment::test_ok PASSED [ 50%]
fe/test/test_payment.py::TestPayment::test_authorization_error PASSED [ 52%]
fe/test/test_payment.py::TestPayment::test_not_suff_funds PASSED [ 53%]
fe/test/test_payment.py::TestPayment::test_repeat_pay PASSED [ 54%]
fe/test/test_receive_books.py::Test_receive_books::test_ok PASSED [ 56%]
fe/test/test_receive_books.py::Test_receive_books::test_false_buyer PASSED [ 57%]
]
fe/test/test_receive_books.py::Test_receive_books::test_non_exist_order PASSED [ 59%]
fe/test/test_receive_books.py::Test_receive_books::test_repeat_receive_books PASSED [ 60%]
fe/test/test_receive_books.py::Test_receive_books::test_can_not_receive PASSED [ 61%]
fe/test/test_register.py::TestRegister::test_register_ok PASSED [ 63%]
fe/test/test_register.py::TestRegister::test_unregister_ok PASSED [ 64%]
fe/test/test_register.py::TestRegister::test_unregister_error_authorization PASSED [ 66%]
fe/test/test_register.py::TestRegister::test_register_error_exist_user_id PASSED [ 67%]
fe/test/test_search.py::TestSearch::test_search_author PASSED [ 69%]
fe/test/test_search.py::TestSearch::test_search_book_intro PASSED [ 70%]
fe/test/test_search.py::TestSearch::test_search_tags PASSED [ 71%]
fe/test/test_search.py::TestSearch::test_search_title PASSED [ 73%]
fe/test/test_search.py::TestSearch::test_search_author_in_store PASSED [ 74%]
fe/test/test_search.py::TestSearch::test_search_book_intro_in_store PASSED [ 76%]
]
fe/test/test_search.py::TestSearch::test_search_tags_in_store PASSED [ 77%]
fe/test/test_search.py::TestSearch::test_search_title_in_store PASSED [ 78%]
fe/test/test_search.py::TestSearch::test_search_author_vague PASSED [ 80%]
fe/test/test_search.py::TestSearch::test_search_book_intro_vague PASSED [ 81%]
fe/test/test_search.py::TestSearch::test_search_tags_vague PASSED [ 83%]
fe/test/test_search.py::TestSearch::test_search_title_vague PASSED [ 84%]
fe/test/test_search.py::TestSearch::test_search_author_vague_in_store PASSED [ 85%]
fe/test/test_search.py::TestSearch::test_search_book_intro_vague_in_store PASSED [ 87%]

```

```

fe/test/test_search.py::TestSearch::test_search_tags_vague_in_store PASSED [ 88%
]
fe/test/test_search.py::TestSearch::test_search_title_vague_in_store PASSED [ 90
%]
fe/test/test_send_books.py::Test_send_books::test_ok PASSED [ 91%]
fe/test/test_send_books.py::Test_send_books::test_false_seller PASSED [ 92%]
fe/test/test_send_books.py::Test_send_books::test_non_exist_order PASSED [ 94%]
fe/test/test_send_books.py::Test_send_books::test_repeat_send_books PASSED [ 95%
]
fe/test/test_send_books.py::Test_send_books::test_can_not_send PASSED [ 97%]
fe/test/test_send_books.py::Test_send_books::test_none_order PASSED [ 98%]
fe/test/test_send_books.py::Test_send_books::test_no_seller PASSED [100%]D
:\database\labs\bookstore2\bookstore\be\serve.py:18: UserWarning: The 'environ['
werkzeug.server.shutdown']' function is deprecated and will be removed in Werkze
ug 2.1.
  func()
2023-12-23 11:38:02,470 [Thread-4558 ] [INFO ] 127.0.0.1 - - [23/Dec/2023 11:38
:02] "GET /shutdown HTTP/1.1" 200 -

```

===== 71 passed in 169.97s (0:02:49) =====

frontend end test
No data to combine

Name	Stmts	Miss	Branch	BrPart	Cover
be__init__.py	0	0	0	0	100%
be\db_conn.py	16	0	6	0	100%
be\model__init__.py	0	0	0	0	100%
be\model\buyer.py	222	42	100	15	79%
be\model\error.py	31	1	0	0	97%
be\model\seller.py	50	1	22	1	97%
be\model\user.py	211	56	66	6	73%
be\serve.py	34	1	2	1	94%
be\view__init__.py	0	0	0	0	100%
be\view\auth.py	103	0	0	0	100%
be\view\buyer.py	56	0	2	0	100%
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	71	1	12	2	96%
fe\access\buyer.py	54	0	2	0	100%
fe\access\new_buyer.py	8	0	0	0	100%
fe\access\new_seller.py	8	0	0	0	100%
fe\access\search.py	45	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	15	1	8	1	91%
fe\bench\session.py	47	0	12	1	98%

Name	Stmts	Miss	Branch	BrPart	Cover
be__init__.py	0	0	0	0	100%
be\db_conn.py	16	0	6	0	100%
be\model__init__.py	0	0	0	0	100%
be\model\buyer.py	222	42	100	15	79%
be\model\error.py	31	1	0	0	97%
be\model\seller.py	50	1	22	1	97%
be\model\user.py	211	56	66	6	73%
be\serve.py	34	1	2	1	94%
be\view__init__.py	0	0	0	0	100%
be\view\auth.py	103	0	0	0	100%
be\view\buyer.py	56	0	2	0	100%
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	71	1	12	2	96%
fe\access\buyer.py	54	0	2	0	100%
fe\access\new_buyer.py	8	0	0	0	100%
fe\access\new_seller.py	8	0	0	0	100%
fe\access\search.py	45	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	15	1	8	1	91%
fe\bench\session.py	47	0	12	1	98%
fe\bench\workload.py	126	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	48	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	6	2	0	0	67%
fe\test\test_close_order.py	77	0	2	0	100%
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_order.py	60	0	10	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_books.py	58	0	2	0	100%
fe\test\test_register.py	32	0	0	0	100%
fe\test\test_search.py	84	0	2	0	100%
fe\test\test_send_books.py	60	0	2	0	100%
TOTAL	1935	107	312	30	93%
Wrote HTML report to htmlcov\index.html					

可以看到我的基础+拓展功能71个测试点全部通过了；而且最终达到了93%的代码覆盖率，说明我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。

订单吞吐量和延迟

先在项目根目录下启动app.py，再运行fe/bench/run.py，最后打开项目根目录下的fe/bench/logger.log即可看到吞吐量与延迟情况：

```
INFO:root:TPS_C=33376, NO=OK:491010 Thread_num:995 TOTAL:491010 LATENCY:0.015675173144032568, P=OK:489090 Thread_num:994  
TOTAL:490015 LATENCY:0.014150404887541303  
INFO:root:TPS_C=33410, NO=OK:492006 Thread_num:996 TOTAL:492006 LATENCY:0.015675279510541994, P=OK:490062 Thread_num:995  
TOTAL:491010 LATENCY:0.014150306309865319  
INFO:root:TPS_C=33443, NO=OK:493003 Thread_num:997 TOTAL:493003 LATENCY:0.01567539130308114, P=OK:491035 Thread_num:996  
TOTAL:492006 LATENCY:0.014150215116543893  
INFO:root:TPS_C=33477, NO=OK:494001 Thread_num:998 TOTAL:494001 LATENCY:0.01567549663188572, P=OK:492009 Thread_num:997  
TOTAL:493003 LATENCY:0.01415012730410488  
INFO:root:TPS_C=33510, NO=OK:495000 Thread_num:999 TOTAL:495000 LATENCY:0.015675595530596647, P=OK:492984 Thread_num:998  
TOTAL:494001 LATENCY:0.014150036912912094  
INFO:root:TPS_C=33544, NO=OK:496000 Thread_num:1000 TOTAL:496000 LATENCY:0.015675699831497285, P=OK:493959 Thread_num:999  
TOTAL:495000 LATENCY:0.014149938058853149
```

延迟：0.01568秒/笔；吞吐量：33544笔/秒

可以看到bookstore运行的效果还是十分好的。

五.版本控制

这个项目的Github链接为 <https://github.com/tommy7dase8/bookstore>

本人在完成这次项目时走过弯路，对数据库的设计有大改的经历，所以最后放到GitHub仓库里的代码是以最后一次的设计为范本的。通过GitHub进行版本控制，我的效率大大提升了，分模块地完成、优化、测试项目，对项目的快速推进贡献了力量。

六.总结

虽然在不久之前我已经小组协作实现了一个文档数据库版的bookstore,但是在这次实验的过程中依然碰到了很多问题。

从最开始表的设计出发，由于最开始我忘记考虑自动取消订单的功能的实现，所以在之后需要自动取消订单时需要一直对表进行查询，即便也有索引的助力，但是时间开销也很大。发现问题所在之后我痛定思痛从设计到代码都进行了较大程度的修改。

接下来就是程序debug的过程是极度吃力且耗时的，修改往往会导致这里的测试点通过了但是其他测试点却error了。但是一步一步来，最终我也终于完成了bookstore.

理论和实验交相结合让我对关系数据库设计有了更深刻的认识，《当代数据管理系统》这门课程马上就要接近尾声了，但这不是我接触并掌握数据库的结束，反而是开始，来日方长，也感谢老师及助教一路以来的悉心指导~

七.附录

项目目录结构

bookstore	
-- be	后端
-- model	功能实现
-- __init__.py	
-- buyer.py	
-- error.py	
-- seller.py	
-- user.py	
-- view	后端接口
-- __init__.py	
-- auth.py	
-- buyer.py	
-- seller.py	
-- __init__.py	
-- db_conn.py	
-- serve.py	
-- doc	接口测试参考（这次实验我写了那些新加的接口了）
-- fe	前端
-- access	各功能http访问
-- bench	效率(吞吐量)测试
-- data	
-- book.db	sqlite 数据库(book.db, 较少量的测试数据)
-- scraper.py	从豆瓣爬取的图书信息数据
-- test	功能性及覆盖率测试
-- conf.py	测试参数
-- conftest.py	pytest初始化配置
-- ...	
-- init_database.py	初始化数据库的文件
-- app.py	启动flask
-- script	