2023_ECNU_PJ1_报告

1. 小组分工

我们小组共由两位成员组成。针对本次项目，我们进行了合理的分工，同时保持了良好的沟通，互相帮助解决各自在项目实践过程中遇到的问题。这里先大致列出我们两人各自负责的主要工作，在下面的功能介绍部分中，会在代码文件标题旁边标注清楚负责编写的成员姓名（部分代码文件为合作编写）。

姓名	学号	主要工作
钱凯恒	10215501406	User接口， 部分Buyer接口， 40%附加功能的测试， 部分性能测试
龙羿霏	10215501415	Seller接口， 部分Buyer接口， 60%基础功能的测试， 部分性能测试

2. 目录结构

本次项目采用了MVC（Model–View–Controller）这种常见的软件架构模式。在MVC模式中，应用程序被分为三个主要组件：模型（Model）、视图（View）和控制器（Controller），这三个组件各自承担不同的责任，便于实现分层和松耦合的设计，以促进代码的可维护性和可扩展性。

我们将Model层代码放在/be/model文件夹下，将View层代码放在/be/view文件夹下，将Controller层代码放在/fe/access文件夹下。此外，我们将功能测试的代码放在/fe/test文件夹下。

Python

```
.
├─be
│   │  app.py
│   │  serve.py
│   │
│   ├─model
│   │   │  buyer.py
│   │   │  db_conn.py
│   │   │  error.py
│   │   │  seller.py
│   │   │  store.py
│   │   │  user.py
│   │
│   ├─view
│   │   │  auth.py
│   │   │  buyer.py
│   │   │  seller.py
│
├─fe
│   ├─access
│   │       auth.py
│   │       book.py
│   │       buyer.py
│   │       new_buyer.py
│   │       new_seller.py
│   │       seller.py
│   │
│   ├─data
│   │       data_transfer.py
│   │       gen_book_db.py
│   │       scraper.py
│   │
│   └─test
│           gen_book_data.py
│           test_add_book.py
│           test_add_funds.py
│           test_add_stock_level.py
│           test_bench.py
│           test_buyer_cancel_order.py
│           test_create_store.py
│           test_deliver_book.py
│           test_login.py
│           test_new_order.py
│           test_overtime_cancel_order.py
│           test_password.py
```

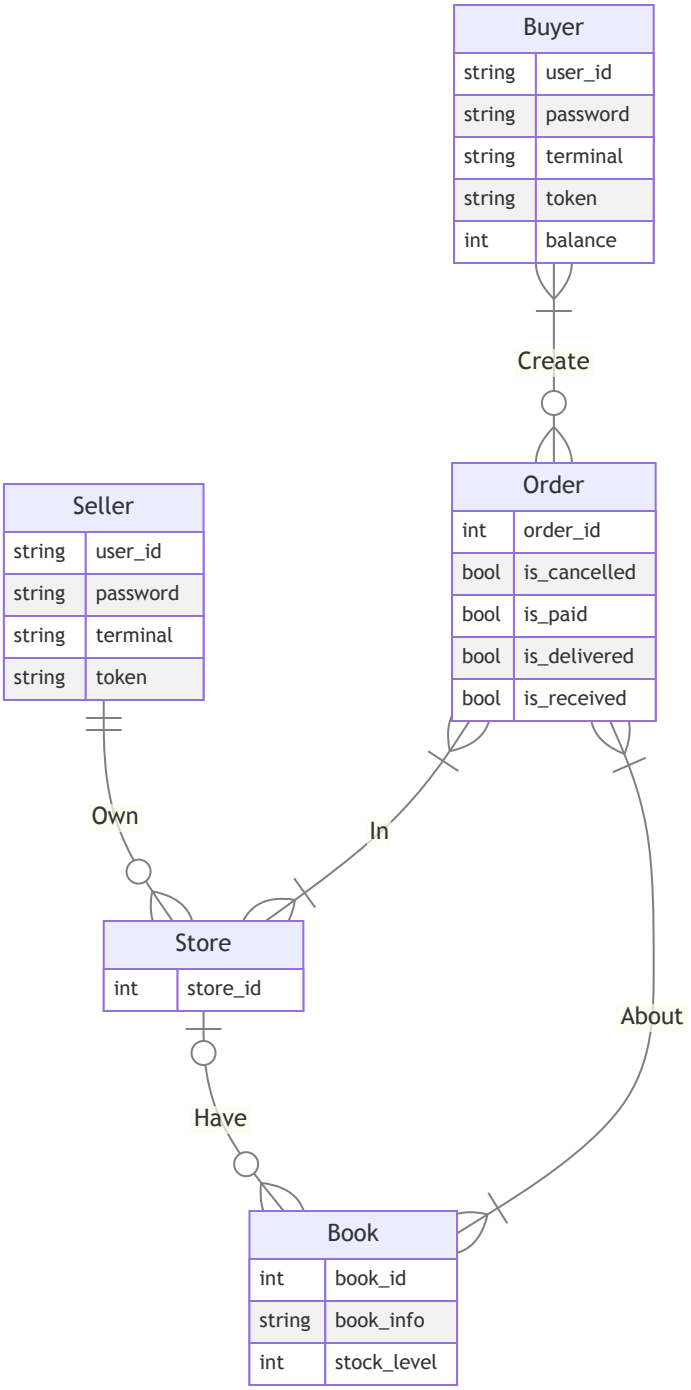
```
│          test_payment.py
│          test_receive_book.py
│          test_register.py
│          test_search_book.py
│          test_search_history_order.py
│
└─script
        test.sh
```

3. 文档数据库设计

3.1. ER图

- 对于用户，他能够注册、注销、登录、登出、更改密码
- 对于买家，他还能够充值、搜索书籍、创建订单、取消订单、付款、收货、搜索历史订单
- 对于商家，他还能够创建店铺、添加书籍信息、添加书籍库存、发货

基于上述逻辑，我们先绘制ER图，再利用ER图指导我们进行文档模式设计。



3.2. 从ER图到文档模式设计

3.2.1. 文档结构

根据ER图，我们共设计了7个文档集合，具体文档结构如下：

- user

JSON

```
{
  "user_id": "$user name$",          // 用户名
  "password": "$user password$",     // 用户密码
  "balance": $balance$,              // 用户余额
  "token": "$access token$",         // 登录token
  "terminal": "$terminal code$"      // 登录terminal
}
```

- user_store

JSON

```
{
  "store_id": "$store name$",    // 商店名
  "user_id": "$user name$"       // 商家名
}
```

- store

JSON

```
{
  "store_id": "$store name$",     // 商店名
  "book_id": $book id$,           // 书籍id
  "book_info": {
    "title": "$book title$",
    "author": "$book author$",
    "book_intro": "$book intro$",
    "content": "$book content$",
    "tags": ["$tag1$", "$tag2$", "$tag3$", "..."]
  },                              // 书籍信息（标题、作者、简介、内容、标签）
  "stock_level": $stock level$    // 书籍库存
}
```

- new_order

JSON

```
{
  "order_id": "$order id$",      // 订单id
  "store_id": "$store name$",    // 商店名
  "user_id": "$buyer name$"      // 买家名
}
```

- new_order_detail

JSON

```
{
  "order_id": "$order id$",     // 订单id
  "book_id": $book id$,         // 书籍id
  "count": $book count$,        // 书籍数量
  "price": $book price$         // 书籍单价
}
```

- history_order

JSON

```
{
  "order_id": "$order id$",           // 订单id
  "store_id": "$store name$",         // 商店名
  "user_id": "$buyer name$",          // 买家名
  "book_info": [
    {
      "book_id": $book_id1$,
      "count": $count1$,
      "price": $price1$
    },
    {
      "book_id": $book_id2$,
      "count": $count2$,
      "price": $price2$
    },
    {
      ...
    }
  ],                                  // 购买书籍的信息，包括书籍id、数量、单价
  "is_cancelled": bool(cancelled),    // 订单是否取消
  "is_paid": bool(paid),              // 订单是否支付
  "is_delivered": bool(delivered),    // 订单是否发货
  "is_received": bool(received)       // 订单是否收货
}
```

- book_detail

```JSON
{
  "book_id": $book id$,                                    // 书籍id
  "title": "$book title$",                                 // 书籍标题
  "author": "$book author$",                               // 书籍作者
  "book_intro": "$book intro$",                            // 书籍简介
  "content": "$book content$",                             // 书籍内容
  "tags": ["$tag1$", "$tag2$", "$tag3$", "..."],           // 书籍标签
  "description": "$all text$"                              // 文本信息，包括书籍标题、简介和内容的分词结果
}
```

3.2.2. 索引

这一部分将在4.1节中详细介绍。

4. 功能介绍：Model层接口

Model层接口提供对数据库的原子操作，后续由View层和Controller层调用这些接口来相应前端的请求。

4.1. `store.py`（@龙羿霏）

该文件主要用来**初始化书店网站的后端及其数据库**，主要是 `Store` **类**的建立与初始化。

该类首先通过 `__init__` 函数建立与本地MongoDB数据库的连接，并默认使用 `be` 数据库；接下来在 `init_collections` 中建立后续代码中会使用到的**集合**及其**索引**：

- `user`：存放已经注册的用户信息
  - 在 `user_id` 上建立升序索引并设置为unique，防止重复的用户数据插入，同时加快后续通过 `user_id` 查找用户的速度。
- `user_store`：存放用户（商家）id及其拥有的店铺id信息
  - 为 `user_id` 和 `store_id` 建立复合索引并设置为unique，每个商家与自己的店铺是绑定在一起的关系（所以查询时经常是两个信息组合出现，复合索引可以加快查询速度），允许用户与店铺存在一对多或多对一的关系。
- `store`：存放店铺中在售的书籍id及其库存量
  - 为 `store_id` 和 `book_id` 建立复合索引，因为添加书籍或修改书籍库存时通常将这两个信息同时输入，复合索引相比于普通索引更有速度上的优势。

- `new_order`：存放用户（买家）新创建订单信息
  - 在 `order_id` 上建立升序索引并设置为unique，保证同一笔订单只插入一次，同时加快根据订单号对用户及店家信息的查找。
- `new_order_detail`：存放某笔订单中某本书的详细订单信息
  - 在 `order_id` 上建立升序索引，加快根据订单号查找的速度。另外，由于单笔订单中可能存在下单多种书籍，因此不对 `order_id` 做unique限制，允许在同一个订单情况下重复插入不同书籍信息。
- `history_order`：控制订单后续状态
  - 在 `user_id` 和 `order_id` 上建立复合索引并设置为unique，这样既可以控制用户与订单一一对应的信息不重复插入，也可以加快后续历史订单组合条件查询的速度。
- `book_detail`：存放所有店铺在售书籍的详细信息，这里建立4个索引：
  - `book_id`，`author` 和 `tags`：分别建立升序索引，同时设置 `book_id` 为unique，防止多家店铺同时售卖同一种书籍造成的信息重复插入，同时加快用户搜索书籍速度。
  - `description`：对书籍信息中的文本描述进行组合后的信息建立全文索引，加快后续买家关键词模糊查找速度。

接下来通过实例化一个 `Store` 类，返回其数据库接口，为后续文件使用。

4.2. `db_conn.py`（@钱凯恒）

该文件主要初始化数据库连接，然后定义了一些到数据库中**验证存在性**的基本操作。

- `user_id_exist(user_id)`：**用户是否存在**
  根据传入 `user_id`，到 `user` 集合中找到 `user_id` 对应文档，如果成功，则用户存在。
- `book_id_exist(store_id, book_id)`：**书籍是否在售**
  根据传入 `store_id` 和 `user_id`，到 `store` 集合中寻找对应文档，如果成功，则该家店铺存在该在售书籍。
- `store_id_exist(store_id)`：**店铺是否存在**
  根据传入 `store_id`，到 `user_store` 集合中寻找对应文档，如果成功，则该家店铺存在。

4.3. `user.py`（@钱凯恒）

该文件主要在 `User` 类中定义了一些**基本的用户操作**，后续操作中买家和商家都会使用到。

- `register(user_id, password)`：**用户注册**
  用户传入注册所需基本信息（用户名、密码），系统为其自动生成terminal值和token，并默认账户

余额为0。接下来尝试将该条数据存入 `user` 集合，如果用户名已经存在，则用户不能成功注册，需要更改用户名重新尝试。

- `login(user_id, password, terminal)`：**用户登录**
  将 `user_id` 和密码传入 `check_password` 函数，到 `user` 集合中验证二者是否匹配：
  - 如果不匹配或者该用户不存在，登录失败。
  - 如果匹配，登录成功，自动为用户生成登录token并存入数据库。
- `logout(user_id, token)`：**用户登出**
  用户传入 `user_id` 和登录时产生的token，调用 `check_token` 函数到 `user` 集合中验证二者是否匹配：
  - 如果不匹配或者该用户不存在，登出失败。
  - 如果匹配，登出成功，更新对应token。
- `unregister(user_id, password)`：**用户注销**
  将 `user_id` 和密码传入 `check_password` 函数，到 `user` 集合中验证二者是否匹配：
  - 验证失败则直接返回。
  - 验证成功，则在 `user` 集合中找到 `user_id` 对应文档进行删除。
- `change_password(user_id, old_password, new_password)`：**修改密码**
  将 `user_id` 和密码传入 `check_password` 函数，到 `user` 集合中验证二者是否匹配：
  - 验证失败则直接返回。
  - 验证成功，则在 `user` 集合中找到 `user_id` 对应文档，将其密码进行更新。
- `check_password(user_id, password)`：**检查密码**
  到 `user` 集合中找到 `user_id` 对应文档：
  - 如果没找到或密码不相同，验证失败。
  - 密码相同，验证成功。
- `check_token(user_id, token)`：**检查token**
  到 `user` 集合中找到 `user_id` 对应文档：
  - 如果没找到 / token不相同 / token生成时间超过3600秒，验证失败。
  - 不是以上情况，验证成功。

4.4. `buyer.py`（@钱凯恒，龙羿霏）

该文件主要定义了**买家**的各种操作。

- `new_order(user_id, store_id, id_and_count)`：**买家下单**
  首先验证买家和店铺是否存在，若均存在：

- 使用 `uid` 生成该订单的 `order_id` 。
- 从 `id_and_count` 中下单书籍的 `book_id` 和数量，到 `store` 集合中验证是否有足够库存，若有，则更新商店库存，然后分别向 `new_order` 、 `new_order_detail` 、 `history_order` 集合中插入一条新记录。

- `payment(user_id, password, order_id)` ： **买家付款**
  先使用 `order_id` 到 `new_order` 集合中验证订单是否存在，若存在：
  - 验证用户、密码是否均存在且对应。
  - 若通过验证，在 `user` 集合中验证用户余额是否足够支付，足够就更新买家账户余额。
  - 支付成功后，删除 `new_order` 、 `new_order_detail` 中对应 `order_id` 的记录，同时在 `history_order` 集合中设置该订单状态为已支付。
- `add_funds(user_id, password, add_value)` ： **用户充值**
  验证该用户与密码是否存在且匹配，若验证成功，则在 `user` 集合中更新该用户账户余额。
- `receive_book(user_id, order_id)` ： **买家收货**
  首先验证该用户是否存在，若存在，就从 `history_order` 取出该 `order_id` 对应记录，如果订单存在且没有被取消 / 已经收货，就将该条记录状态更新为已收货。
- `buyer_cancel_order(user_id, order_id)` ： **买家主动取消订单**
  首先验证该用户是否存在，若存在，就从 `history_order` 取出该 `order_id` 对应记录，如果订单存在且没有被取消 / 已经付款，就将该条记录状态更新为已取消。
  （这里预设买家支付后不能再主动取消订单）
- `overtime_cancel_order()` ： **超时未付款自动取消订单**
  默认设置未支付订单最长保留时间为15分钟，并且默认为系统行为。
  检查 `history_order` 中所有订单：
  - 若订单状态为已支付 / 已取消，则跳过该条订单。
  - 否则对 `order_id` 进行解码，获取下单时间并与现在时间进行比较，若已超出最长保留时间，则调用 `cancel_order` 函数取消该订单。
- `cancel_order(history_order_col, order, order_id)` ： **取消订单信息更新**
  - 根据 `order_id` 到 `history_order` 中将订单状态修改为已取消。
  - 根据 `order_id` 找到对应 `store_id` 、 `book_id` 和 下单数量 `count` ，然后到 `store` 集合中对应店铺将对应书籍库存进行还原。
  - 将 `new_order` 、 `new_order_detail` 中对应 `order_id` 的记录删除。
- `search_history_order(user_id, order_id, page, per_page)` ： **搜索历史订单**
  首先验证用户是否存在，若存在：
  - 如果用户传入 `order_id` ，则从 `history_order` 中找到对应订单信息，按照分页信息返回。

    - 如果用户没有传入 `order_id`，则从 `history_order` 中找到该用户对应所有订单信息，按照分页信息返回。
- `search_book(store_id, title, author, intro, content, tags, page, per_page)`：**用户参数化搜索在售书籍**
  参数化搜索主要分为**三个阶段**：
    - 首先根据用户是否传入 `store_id`，判断是否为店铺内搜索 / 全站搜索，如果是店铺内搜索，就先使用 `store_id` 从 `store` 集合中将该店铺中所有对应 `book_id` 取出，作为下一步的筛选条件。
    - 接下来将用户传入的 `author` 和 `tags` 信息进行查询条件的组合（因为这两个信息需要进行精确查找，所以提前执行筛选，缩小后续模糊查找的范围），然后与上一步得到的 `book_id` 信息（如果有的话）组合在一起，将其传入 `store` 集合中再次进行查询，返回这次符合条件的 `book_id` 信息。
    - 最后将 `title`、`content` 和 `intro` 三部分文本信息字符串进行组合、分词，得到一个关键词列表，然后将列表中的词汇与上一步的 `book_id` 信息经过组合，依次传入 `book_detail` 集合进行查询，遍历完所有词汇后，对得到的 `book_id` 进行去重，即为所有查询书籍结果，从 `book_detail` 中取出对应信息返回即可。

这三个阶段的查询中，每一步都为下一步的查询缩小了范围，并且每一步的单次查询代价都比上一步的高（尤其是文本查询）。通过不断缩小查询范围，我们可以降低高代价查询的次数，从而更好地降低总查询代价，加快搜索的速度。

4.5. `seller.py`（@龙羿霏）

该文件主要定义了**商家**的各种操作。

- `add_book(user_id, store_id, book_id, book_json_str, stock_level)`：**向店铺添加在售书籍信息**
  首先判断用户和店铺是否存在，如果存在，判断这本书是否已经在当前店铺添加过，如果不存在，更新两部分内容：
    - 向 `store` 集合中新加入一条数据，包括用户传入的所有参数信息。
    - 向 `book_detail` 集合中新加入一条数据，主要是对 `book_json_str` 中书籍的详细信息进行存储（方便买家搜索书籍），这里最大的变化是将其中的 `title`、`content` 和 `book_info` 三个字符串连接后分词（以空格形式分开）作为新字符串存入 `description`。
- `add_stock_level(user_id, store_id, book_id, add_stock_level)`：**添加书籍库存**
  判断用户、店铺和待修改书籍是否都存在，如果都存在，就更新 `store` 集合中对应的书籍库存数。

- `create_store(user_id, store_id)`：**商家创建店铺**
  首先判断用户是否存在，若存在，继续判断该店铺是否已经存在，若店铺不存在，则向 `user_store` 集合中添加一条数据，绑定用户与店铺的关系。
- `deliver_book(user_id, store_id, order_id)`：**商家发货**
  首先判断用户、店铺和订单是否均存在，若存在，则判断订单是否已经被支付且未被取消，若均满足条件，则将 `history_order` 集合中该订单对应状态修改为已发货。

5. 功能介绍：View层接口

View层主要定义了网站需要用到的各种路由。

5.1 `auth.py`（@钱凯恒）

这个文件下所定义的请求url前缀均为 `/auth/...`，发送请求方法均为POST，都是关于用户的操作。

- `login()`：`/login`，**用户登录**
  解析用户传入的 `user_id`，`password` 和 `terminal` 信息，调用后端逻辑，生成此次登录token并存入 `user` 集合中并返回到前端。
- `logout()`：`/logout`，**用户登出**
  解析用户传入的 `user_id` 和 `token`，调用后端逻辑，修改用户在 `user` 集合中的token值。
- `register()`：`/register`，**用户注册**
  解析用户传入的 `user_id` 和 `password`，调用后端逻辑，在 `user` 集合中新加入一条用户记录。
- `unregister()`：`/unregister`，**用户注销**
  解析用户传入的 `user_id` 和 `password`，调用后端逻辑，在 `user` 集合中删除对应用户记录。
- `change_password()`：`password`，**修改密码**
  解析用户传入的 `user_id`，`oldPassword` 和 `newPassword`，调用后端逻辑，在 `user` 集合中修改用户密码值。

5.2 `buyer.py`（@钱凯恒，龙羿霏）

这个文件下所定义的请求url前缀均为 `/buyer/...`，发送请求方法均为POST，都是关于买家的操作。

- `new_order()`：`/new_order`，**买家下单**
  解析用户传入的 `user_id`， `store_id` 和 `books`，调用后端Buyer接口，向数据库中添加新订单对应信息。
- `payment()`：`/payment`，**买家付款**
  解析用户传入的 `user_id`， `order_id` 和 `password`，调用后端Buyer接口，更新数据库中的订单状态。
- `add_funds()`：`/add_funds`，**买家充值**
  解析用户传入的 `user_id`， `password` 和 `add_value`，调用后端Buyer接口，修改 `user` 集合中用户的账户余额。
- `receive_book()`：`/receive_book`，**买家收货**
  解析用户传入的 `user_id` 和 `order_id`，调用后端Buyer接口，将 `history_order` 中的订单状态修改为已收货。
- `overtime_cancel_order()`：`/overtime_cancel_order`，**超时取消订单**
  调用后端Buyer接口中自动检查逻辑，将 `history_order` 中的超时订单状态修改为取消。
- `cancel_order()`：`/buyer_cancel_order`，**买家取消订单**
  解析用户传入的 `user_id` 和 `order_id`，调用后端Buyer接口，将 `history_order` 中的未付款订单状态改为已取消。
- `search_book()`：`/search_book`，**买家搜索书籍信息**
  解析用户传入的 `store_id`、`title`、`author`、`book_intro`、`content` 和 `tags`，同时解析请求参数中的分页参数 `page` 和 `per_page`（若未传入，则默认为1和3），将这些信息一起传入后端，返回分页后的书籍信息。
- `search_history_order()`：`/search_history_order`，**买家搜索历史订单**
  解析用户传入的 `user_id` 和 `order_id`，同时解析请求参数中的分页参数 `page` 和 `per_page`（若未传入，则默认为1和3），将这些信息一起传入后端，返回分页后的历史订单信息。

5.3 `seller.py`（@龙羿霏）

这个文件下所定义的请求url前缀均为 `/seller/...`，发送请求方法均为POST，都是关于卖家的操作。

- `seller_create_store()`：`/create_store`，**卖家创建店铺**
  解析用户传入的 `user_id` 和 `store_id`，调用后端Seller接口，向 `user_store` 集合中增加一条店铺信息。
- `seller_add_book()`：`/add_book`，**卖家添加在售书籍**
  解析用户传入的 `user_id`、`store_id`、`book_info` 和 `stock_level`，调用后端Seller接口，向 `store` 和 `book_detail` 集合中添加该条书籍信息。

- `add_stock_level()`： `/add_stock_level`，**卖家增加库存**
  解析用户传入的 `user_id`、`store_id`、`book_id` 和 `add_stock_level`，调用后端Seller接口，更新 `store` 中对应书籍库存数量。
- `deliver_book()`： `/deliver_book`，**卖家发货**
  解析用户传入的 `user_id`、`store_id` 和 `book_id`，调用后端Seller接口，更新 `history_order` 中的订单状态为已发货。

6. 功能介绍：Controller层接口

Controller层的功能是使用POST方法发送HTTP请求到服务器，以及接收服务器的状态码等内容。

6.1 `auth.py`（@钱凯恒）

该文件是在 `Auth` 类中定义了关于用户认证的请求。

- `login`：**用户登录请求**
  将 `user_id`，`password` 和 `terminal` 放入请求中发送，请求用户登录。
- `register`：**用户注册请求**
  将 `user_id` 和 `password` 放入请求中发送，请求新用户注册。
- `password`：**用户修改密码请求**
  将 `user_id`，`oldPassword` 和 `newPassword` 放入请求中发送，请求修改密码。
- `logout`：**用户登出请求**
  将 `user_id` 和 `token` 放入请求中发送，请求用户登出。
- `unregister`：**用户注销请求**
  将 `user_id` 和 `password` 放入请求中发送，请求用户注销。

6.2 `book.py`

该文件中 `Book` 类定义了书的详细信息的基本样式，然后在 `BookDB` 类中定义了从SQLite数据库中读取数据的路径以及读取信息的操作。

- `__init__(large: bool = False)`：**设置读取数据库**
  默认 `large = False`，即从 `book.db` 中读取书籍信息；若 `large = True`，则从更大的数据库 `book_lx.db` 中读取书籍信息用于后续操作。
- `get_book_count`：**获取数据库书籍数量**

获取 `__init__` 中选择数据库中数据条数。

- `get_book_info`：**获取书籍信息**
  从已选择的数据库中按照指定的起始与终止位置，按行读取书籍信息，并将每行数据转换为一个 `Book` 对象，添加到 `books` 列表中并返回。

6.3 `new_buyer.py`

该文件使用 `register_new_buyer` 函数，使用传入的用户名和密码参数，发送一个注册用户的请求，接着创建一个Buyer对象并返回，生成一个新的买家。

6.4 `buyer.py`（@钱凯恒，龙羿霏）

该文件在 `Buyer` 类中注册并登录一个买家对象，然后定义了买家的后续操作请求。

- `new_order`：**买家下单请求**
  通过将 `store_id` 和对应的下单书籍与数量数组 `book_id_and_count` 放入请求中发送，创建一个新订单。
- `payment`：**买家付款请求**
  通过将 `order_id` 和新建买家对象的用户名、密码放入请求中发送，买家完成该笔订单支付。
- `add_funds`：**买家充值请求**
  通过将 `add_value` 和新建买家对象的用户名、密码放入请求中发送，买家对自己的账户进行充值。
- `receive_book`：**买家收货请求**
  通过将 `order_id` 和新建买家对象的用户名、密码放入请求中发送，买家完成该笔订单收货。
- `buyer_cancel_order`：**买家取消订单请求**
  通过将 `order_id` 和新建买家对象的用户名、密码放入请求中发送，买家主动取消未支付的订单。
- `overtime_cancel_order`：**超时订单取消请求**
  发送该请求，系统自动取消当前所有超时未付款订单。
- `search_history_order`：**买家搜索历史订单请求**
  通过将 `order_id` 和新建买家对象的用户名、密码，以及分页参数放入请求中发送，买家搜索指定历史订单，按分页返回结果。
- `search_book`：**买家搜索书籍信息请求**
  通过将 `store_id`、`title`、`author`、`intro`、`content`、`tags` 书籍信息以及 `page`、`per_page` 的分页信息放入请求发送，买家按条件搜索图书，按分页返回结果。

6.5 `new_seller.py`

该文件使用 `register_new_seller` 函数，使用传入的用户名和密码参数，发送一个注册用户的请求，接着创建一个Seller对象并返回，生成一个新的卖家。

6.6 `seller.py` （@龙羿霏）

该文件在 `Seller` 类中注册并登录一个卖家对象，然后定义了卖家的后续操作请求。

- `create_store`：**卖家创建店铺请求**
  通过将 `store_id` 和新建卖家对象的用户名、密码放入请求中发送，卖家完成该店铺创建。
- `add_book`：**卖家添加在售书籍请求**
  通过将 `store_id` 、`book_info` 、`stock_level` 和新建卖家的用户名放入请求中发送，卖家完成在售书籍的添加。
- `add_stock_level`：**卖家增加库存请求**
  通过将 `store_id` 、`book_info` 、`add_stock_level` 和新建卖家的用户名放入请求中发送，卖家完成该书库存的增加。
- `deliver_book`：**卖家发货请求**
  通过将 `store_id` 和新建卖家对象的用户名、密码放入请求中发送，卖家完成该订单发货。

7. 功能测试

7.1. 60%基础功能

7.1.1. 测试用例

- `test_register`
  - 测试用户注册与注销
  - 若用户名不存在，则注销报错
  - 若用户名已存在，则注册报错
- `test_login`
  - 测试用户登录
  - 若用户名不存在，则登录报错
  - 若密码不正确，则登录报错

- `test_password`
  - 测试用户更改密码
  - 若原密码不正确，则更改密码报错
  - 若用户名不存在，则更改密码报错
- `test_create_store`
  - 测试商家创建店铺
  - 若商店名已存在，则创建店铺报错
- `test_add_book`
  - 测试商家添加书籍信息
  - 若商店名不存在，则添加书籍信息报错
  - 若书籍id已存在，则添加书籍信息报错
  - 若商家名不存在，则添加书籍信息报错
- `test_add_stock_level`
  - 测试商家添加书籍库存
  - 若商店名不存在，则添加书籍库存报错
  - 若书籍id已存在，则添加书籍库存报错
  - 若商家名不存在，则添加书籍库存报错
- `test_add_funds`
  - 测试买家充值
  - 若用户名不存在，则充值报错
  - 若密码不正确，则充值报错
- `test_new_order`
  - 测试买家创建订单
  - 若用户名不存在，则创建订单报错
  - 若商店名不存在，则创建订单报错
  - 若书籍id不存在，则创建订单报错
  - 若书籍库存不足，则创建订单报错
- `test_payment`
  - 测试买家付款
  - 若密码不正确，则付款报错
  - 若余额不足，则付款报错
  - 若重复付款，则付款报错

- `test_bench`
  - 测试后端吞吐量
  - 首先把 book.db 中的内容通过调用插入书本的后端插入到 MongoDB 的数据库中，然后通过大量线程同时调用下订单和付款的后端接口，来测试读写性能

7.1.2. 测试结果

经过测试，33个测试用例全部通过，测试覆盖率为95%，说明取得了一个良好的测试结果。

```
========================================= 33 passed in 505.38s (0:08:25) =========================================
E:\database\bookstore\be\serve.py:18: UserWarning: The 'environ['werkzeug.server.shutdown']' function is deprecated and
  func()
2023-11-04 17:44:44,235 [Thread-11465] [INFO ]  127.0.0.1 - - [04/Nov/2023 17:44:44] "GET /shutdown HTTP/1.1" 200 -
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\buyer.py                      96     15     34      6    84%
be\model\db_conn.py                    22      0      6      0   100%
be\model\error.py                      23      1      0      0    96%
be\model\seller.py                     45      7     16      1    87%
be\model\store.py                      37      7      0      0    81%
be\model\user.py                      111     12     24      3    89%
be\serve.py                            35      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                       34      0      2      0   100%
be\view\seller.py                      31      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                     36      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\seller.py                    31      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      2    97%
fe\bench\workload.py                  125      2     22      2    97%
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      1    98%
fe\test\test_add_book.py               37      0     10      0   100%
fe\test\test_add_funds.py              23      0      0      0   100%
fe\test\test_add_stock_level.py        40      0     10      0   100%
fe\test\test_bench.py                   6      2      0      0    67%
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                                1243     52    180     19    95%
Wrote HTML report to ❐8;;file://E:\database\bookstore\htmlcov\index.htmlhtmlcov\index.html❐8;;
```

7.2. 40%附加功能

7.2.1. 测试用例

- `test_buyer_cancel_order`

    - 测试买家取消订单
    - 若订单id不存在，则取消订单报错
    - 若重复取消订单，则取消订单报错
    - 若订单已取消（可能因为超时自动取消），则取消订单报错
- `test_overtime_cancel_order`
    - 测试超时自动取消订单
- `test_deliver_book`
    - 测试商家发货
    - 若用户名不存在，则发货报错
    - 若商店名不存在，则发货报错
    - 若订单id不存在，则发货报错
    - 若订单已取消，则发货报错
    - 若订单未支付（还未取消），则发货报错
- `test_receive_book`
    - 测试买家收货
    - 若用户名不存在，则收货报错
    - 若订单id不存在，则收货报错
    - 若订单已取消，则收货报错
    - 若订单未发货，则收货报错
- `test_search_book`
    - 测试买家搜索书籍
    - 若没有搜索结果，则搜索书籍报错
    - 若想在当前店铺搜索，而商店名不存在，则搜索书籍报错
- `test_search_history_order`
    - 测试买家搜索历史订单
    - 若没有搜索结果，则搜索历史订单报错
    - 若用户名不存在，则搜索历史订单报错

7.2.2. 测试结果

经过测试，55个测试用例全部通过，测试覆盖率为86%（在实现附加功能时，添加了较多 `try ... except ...` 语句用于错误捕获，所以部分文件的覆盖率可能有所下降），说明取得了一个良好的测试结果。

```
Prefix dict has been built successfully.
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 [  5%]
fe/test/test_add_book.py::TestAddBook::test_error_non_exist_user_id PASSED [  7%]
fe/test/test_add_funds.py::TestAddFunds::test_ok PASSED [  9%]
fe/test/test_add_funds.py::TestAddFunds::test_error_user_id PASSED [ 10%]
fe/test/test_add_funds.py::TestAddFunds::test_error_password PASSED [ 12%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_error_user_id PASSED [ 14%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_error_store_id PASSED [ 16%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_error_book_id PASSED [ 18%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_ok PASSED [ 20%]
fe/test/test_bench.py::test_bench PASSED [ 21%]
fe/test/test_buyer_cancel_order.py::TestCancelOrder::test_ok PASSED [ 23%]
fe/test/test_buyer_cancel_order.py::TestCancelOrder::test_invalid_order_id PASSED [ 25%]
fe/test/test_buyer_cancel_order.py::TestCancelOrder::test_repeat_cancel PASSED [ 27%]
fe/test/test_buyer_cancel_order.py::TestCancelOrder::test_cancel_paid_order PASSED [ 29%]
fe/test/test_create_store.py::TestCreateStore::test_ok PASSED [ 30%]
fe/test/test_create_store.py::TestCreateStore::test_error_exist_store_id PASSED [ 32%]
fe/test/test_deliver_book.py::TestDeliverBook::test_ok PASSED [ 34%]
fe/test/test_deliver_book.py::TestDeliverBook::test_error_non_exist_user_id PASSED [ 36%]
fe/test/test_deliver_book.py::TestDeliverBook::test_error_non_exist_store_id PASSED [ 38%]
fe/test/test_deliver_book.py::TestDeliverBook::test_invalid_order_id PASSED [ 40%]
fe/test/test_deliver_book.py::TestDeliverBook::test_cancelled_order PASSED [ 41%]
fe/test/test_deliver_book.py::TestDeliverBook::test_not_paid_order PASSED [ 43%]
fe/test/test_login.py::TestLogin::test_ok PASSED [ 45%]
fe/test/test_login.py::TestLogin::test_error_user_id PASSED [ 47%]
fe/test/test_login.py::TestLogin::test_error_password PASSED [ 49%]
fe/test/test_new_order.py::TestNewOrder::test_non_exist_book_id PASSED [ 50%]
fe/test/test_new_order.py::TestNewOrder::test_low_stock_level PASSED [ 52%]
fe/test/test_new_order.py::TestNewOrder::test_ok PASSED [ 54%]
fe/test/test_new_order.py::TestNewOrder::test_non_exist_user_id PASSED [ 56%]
fe/test/test_new_order.py::TestNewOrder::test_non_exist_store_id PASSED [ 58%]
fe/test/test_overtime_cancel_order.py::TestOvertimeCancelOrder::test_ok PASSED [ 60%]
fe/test/test_password.py::TestPassword::test_ok PASSED [ 61%]
fe/test/test_password.py::TestPassword::test_error_password PASSED [ 63%]
fe/test/test_password.py::TestPassword::test_error_user_id PASSED [ 65%]
fe/test/test_payment.py::TestPayment::test_ok PASSED [ 67%]
fe/test/test_payment.py::TestPayment::test_authorization_error PASSED [ 69%]
fe/test/test_payment.py::TestPayment::test_not_suff_funds PASSED [ 70%]
fe/test/test_payment.py::TestPayment::test_repeat_pay PASSED [ 72%]
fe/test/test_receive_book.py::TestReceiveBook::test_ok PASSED [ 74%]
fe/test/test_receive_book.py::TestReceiveBook::test_invalid_order_id PASSED [ 76%]
fe/test/test_receive_book.py::TestReceiveBook::test_cancelled_order PASSED [ 78%]
fe/test/test_receive_book.py::TestReceiveBook::test_not_delivered_order PASSED [ 80%]
fe/test/test_receive_book.py::TestReceiveBook::test_error_non_exist_user_id PASSED [ 81%]
fe/test/test_register.py::TestRegister::test_register_ok PASSED [ 83%]
fe/test/test_register.py::TestRegister::test_unregister_ok PASSED [ 85%]
fe/test/test_register.py::TestRegister::test_unregister_error_authorization PASSED [ 87%]
fe/test/test_register.py::TestRegister::test_register_error_exist_user_id PASSED [ 89%]
fe/test/test_search_book.py::TestSearchBook::test_ok PASSED [ 90%]
fe/test/test_search_book.py::TestSearchBook::test_non_search_result PASSED [ 92%]
fe/test/test_search_book.py::TestSearchBook::test_non_exist_store_id PASSED [ 94%]
fe/test/test_search_history_order.py::TestSearchHistoryOrder::test_non_history_order PASSED [ 96%]
fe/test/test_search_history_order.py::TestSearchHistoryOrder::test_ok PASSED [ 98%]
fe/test/test_search_history_order.py::TestSearchHistoryOrder::test_non_exist_user_id PASSED [100%]

==================== 55 passed in 83.69s (0:01:23) ====================
```

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\buyer.py	369	142	180	33	58%
be\model\db_conn.py	22	0	6	0	100%
be\model\error.py	35	1	0	0	97%
be\model\seller.py	98	25	44	1	70%
be\model\store.py	45	2	0	0	96%
be\model\user.py	117	18	30	3	82%
be\serve.py	35	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	75	0	2	0	100%
be\view\seller.py	39	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	66	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\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	2	97%
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	1	16	1	97%
fe\test\test_add_book.py	37	0	10	0	100%
fe\test\test_add_funds.py	23	0	0	0	100%
fe\test\test_add_stock_level.py	40	0	10	0	100%
fe\test\test_bench.py	6	2	0	0	67%
fe\test\test_buyer_cancel_order.py	55	1	4	1	97%
fe\test\test_create_store.py	20	0	0	0	100%
fe\test\test_deliver_book.py	79	1	4	1	98%
fe\test\test_login.py	28	0	0	0	100%
fe\test\test_new_order.py	40	0	0	0	100%
fe\test\test_overtime_cancel_order.py	40	1	4	1	95%
fe\test\test_password.py	33	0	0	0	100%
fe\test\test_payment.py	60	1	4	1	97%
fe\test\test_receive_book.py	83	1	4	1	98%
fe\test\test_register.py	31	0	0	0	100%
fe\test\test_search_book.py	28	0	0	0	100%
fe\test\test_search_history_order.py	39	0	4	0	100%
TOTAL	2004	202	380	50	86%

8. 性能测试

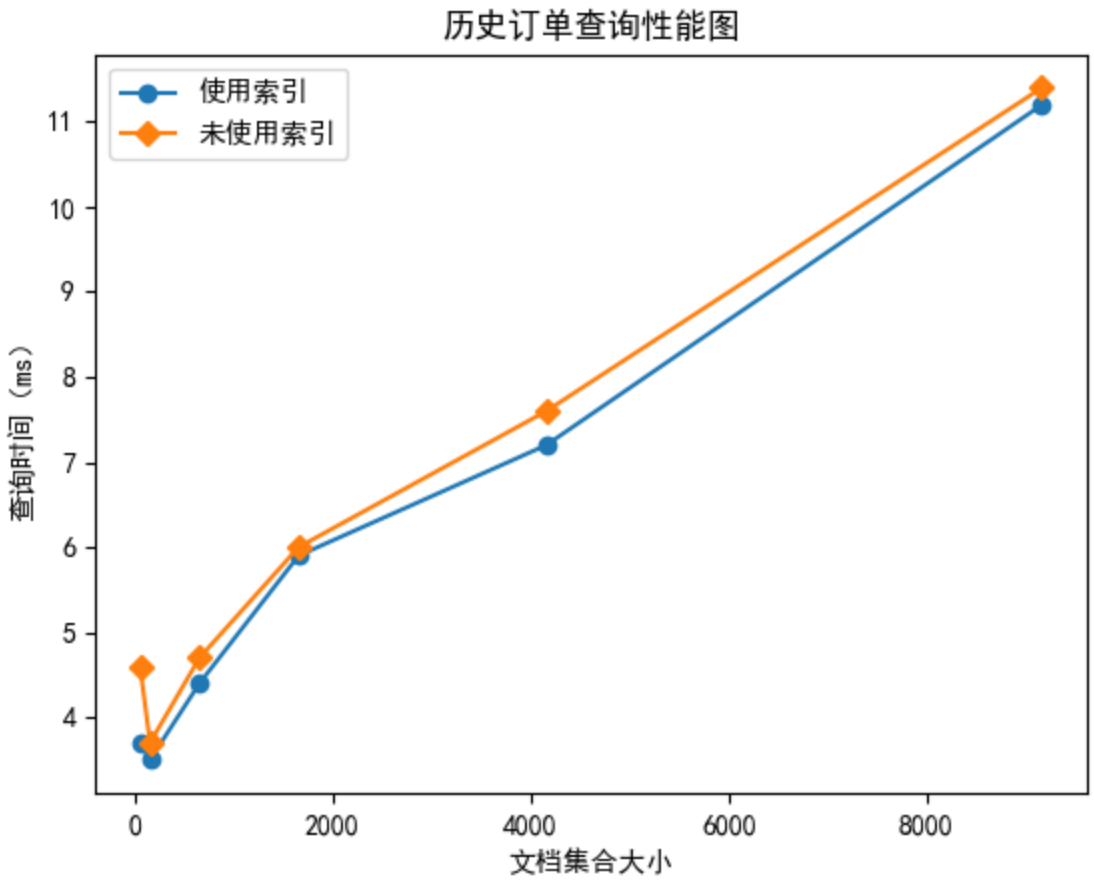
8.1. 历史订单查询性能

8.1.1. 测试用例

`test_search_history_order_performance`

- 先生成一个原始买家，并创建一个属于他的订单
- 再设置重复轮数，每次循环中，生成新的买家、商家、商店，并创建一个属于新买家的订单。循环结束后查询原始买家的历史订单，计算查询时间，并获得此时history_order文档集合的大小
- 重复轮数取值为[50, 100, 500, 1000, 2500, 5000]

8.1.2. 测试结果



观察上图可得，使用索引有助于提升程序与数据库执行的性能。

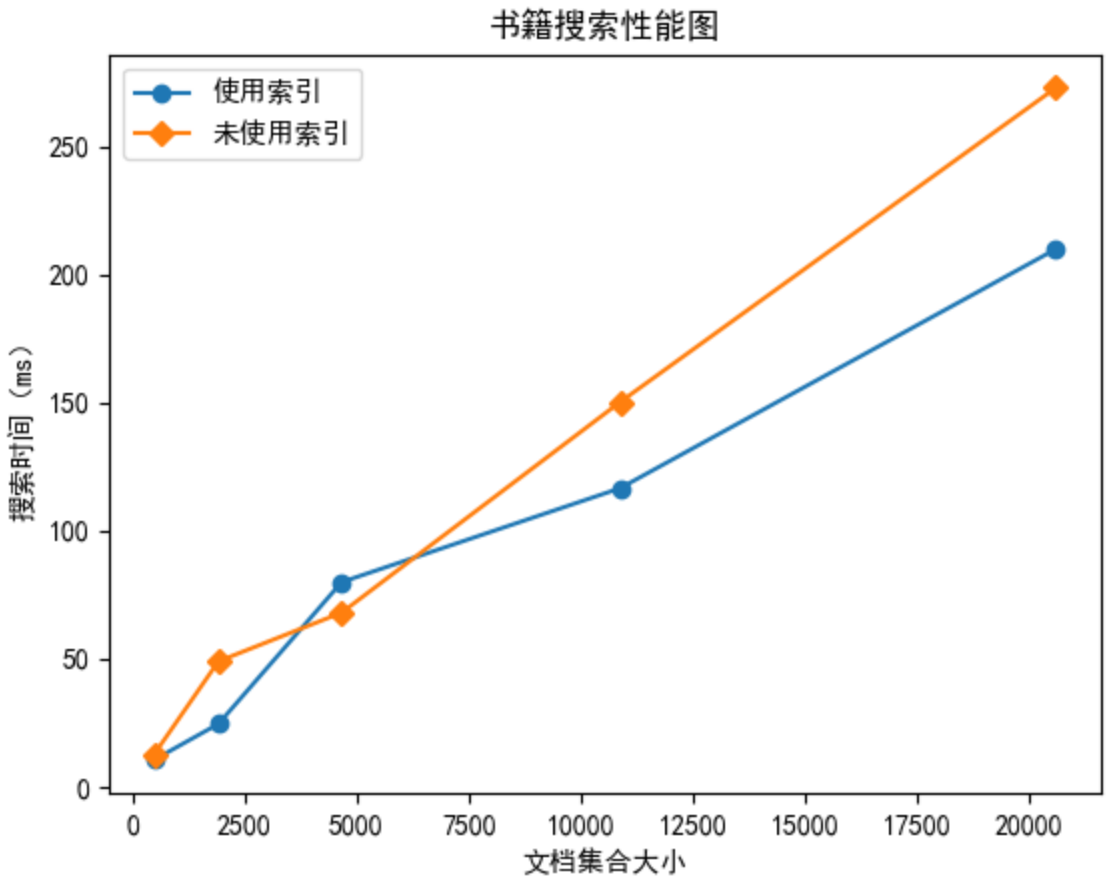
8.2. 书籍搜索性能

8.2.1. 测试用例

`test_search_book_performance`

- 先生成一个买家
- 再设置重复轮数，每次循环中，生成新的商家、商店，并且商家会向商店添加书籍，同时会向全站书籍名录添加书籍。循环结束后买家全站搜索书籍，计算搜索时间，并获得此时book_detail文档集合的大小
- 重复轮数取值为[100, 500, 1000, 2500, 5000]

8.2.2. 测试结果



观察上图同样可得，使用索引有助于提升程序与数据库执行的性能。

9. 版本管理 & 成员总结

9.1. 版本管理

- 我们使用git作为版本管理工具，使得团队协作更加容易和高效。使用git后能够方便回溯到项目的任何历史版本，同时可以清晰地看到每位成员对于项目的代码贡献
- 仓库链接：https://github.com/KaihengQian/CDMS/tree/main/Project1

9.2. 成员总结

钱凯恒：

- 对文档数据库的文档设计思路有了更好的理解，广泛了解了MongoDB事务处理的相关内容
- 掌握了功能测试以及性能测试的方法，提升了前后端项目开发中的Debug能力

龙羿霏：

- 熟悉了MongoDB数据库的各种操作方式，对一些常见出错方式有了更多的了解
- 加深了对搜索操作的理解，尤其在分词和建立索引方面