数据管理系统第一次大作业-bookstore

孙钊琳	庄欣熠	刘江涛
10225501459	10225501451	10225501445

2024.10

目录

1 实验摘要

实验仓库地址：https://github.com/ecnu-Sun/dbProject1.git

本次实验主要目的是开发一个提供网上购书功能的网站后端，实验中我们微调了数据库的 schema，实现了买家和卖家的注册、登录、登出和注销功能，支持卖家创建店铺、添加书籍信息、管理库存，以及买家在商店中下单、付款和充值等操作。网站后端还完整支持了购书流程中的下单、付款、发货和收货环节。

为了提升用户体验，我们添加了额外的功能，包括订单状态查询、订单取消和超时自动取消机制。用户可以通过关键字在全站或指定店铺内搜索书籍，支持按题目、标签、目录和内容等多维度搜索，并实现了结果分页展示。此外，**我们还优化了数据库操作的性能，为相关字段创建了索引**。

在技术实现上，我们选择了 MongoDB 作为文档型数据库，设计了合理的数据库结构，确保数据的高效存储和访问。**项目的版本管理采用了 GitHub 和 Git，以便于团队协作和代码版本和分支维护**。通过编写全面的测试用例，我们验证了各项功能的正确性和稳定性。

2 文档数据库设计

对原来 sqlite 数据库 schema 稍作修改后，我们使用三个文档集合: user, store, new-order：

Listing 1: 用户集合示例

```
{
  "_id": ObjectId("..."),  // MongoDB 自动生成的唯一标识符
  "user_id": "unique_user_id",  // 用户唯一 ID，索引字段
  "password": "hashed_password",
  "balance": 1000,  // 用户余额
  "token": "jwt_token",
  "terminal": "user_terminal_info"
}
```

Listing 2: 书店集合示例

```
{
  "_id": ObjectId("..."),
  "store_id": "unique_store_id",  // 书店唯一 ID，索引字段
  "user_id": "unique_user_id",  // 店主的用户 ID
  "books": [
    {
      "book_id": "unique_book_id",  // 每本书的唯一 ID，嵌套索
          引字段
      "book_info": {
        "title": "Book Title",
        "author": "Author Name",
        "price": 50,
        // 其他书籍相关信息
      },
      "stock_level": 100  // 库存量
    }
    // 更多书籍...
  ]
}
```

Listing 3: 订单集合示例

```
{
  "_id": ObjectId("..."),
  "order_id": "unique_order_id",  // 订单唯一 ID，索引字段
  "user_id": "unique_user_id",  // 下单的用户 ID
  "store_id": "unique_store_id",  // 书店的 ID
  "items": [
    {
      "book_id": "unique_book_id",
      "count": 2,
      "price": 50  // 每本书的单价
    }
    // 更多书籍条目...
  ],
  "total_price": 100,  // 订单总价
  "status": "pending"  // 订单状态：pending、completed、
      cancelled
}
```

3 用户权限接口

本模块负责处理用户的注册、登录、注销、密码修改等操作，实现了用户权限的管理。接口在 **be\view\auth.py** 中实现，用于接收来自测试线程或前端的 HTTP 请求，解析参数后，将其传递给后端的支持函数，这些函数存放在 **be\model\user.py** 中。

3.1 接口概述

be\view\auth.py 中定义了以下接口：

- `/register`：用户注册
- `/login`：用户登录
- `/logout`：用户注销
- `/unregister`：用户注销账户
- `/password`：用户修改密码

这些接口接收相应的 POST 请求，提取请求中的参数（如 user_id、password、terminal 等），然后调用 **be\model\user.py** 中的对应方法处理对应逻辑，实现具体功能。

3.2 后端逻辑

在 **be\model\user.py** 中，实现了用户权限管理的核心功能。主要包括以下类和方法：

3.2.1 类 User

该类负责用户权限相关的所有操作，主要方法有：

- register()：注册新用户
- login()：用户登录
- logout()：用户注销
- unregister()：注销用户账户
- change_password()：修改用户密码
- check_password()：验证用户密码
- check_token()：验证用户令牌

3.2.2 数据库适配

在初始化时，使用 PyMongo 连接到 MongoDB 数据库：

```
self.client = pymongo.MongoClient('mongodb://localhost:27017/')
self.db = self.client['bookstore']
self.user_col = self.db['user']
```

并确保 user_id 的唯一性：

```
self.user_col.create_index('user_id', unique=True)
```

3.3 功能实现流程

为了报告的可读性，代码逻辑尽可能使用流程图呈现，以下展示用户注册和登录功能实现的流程图，完具体代码可以在压缩包中查看。

3.3.1 用户注册

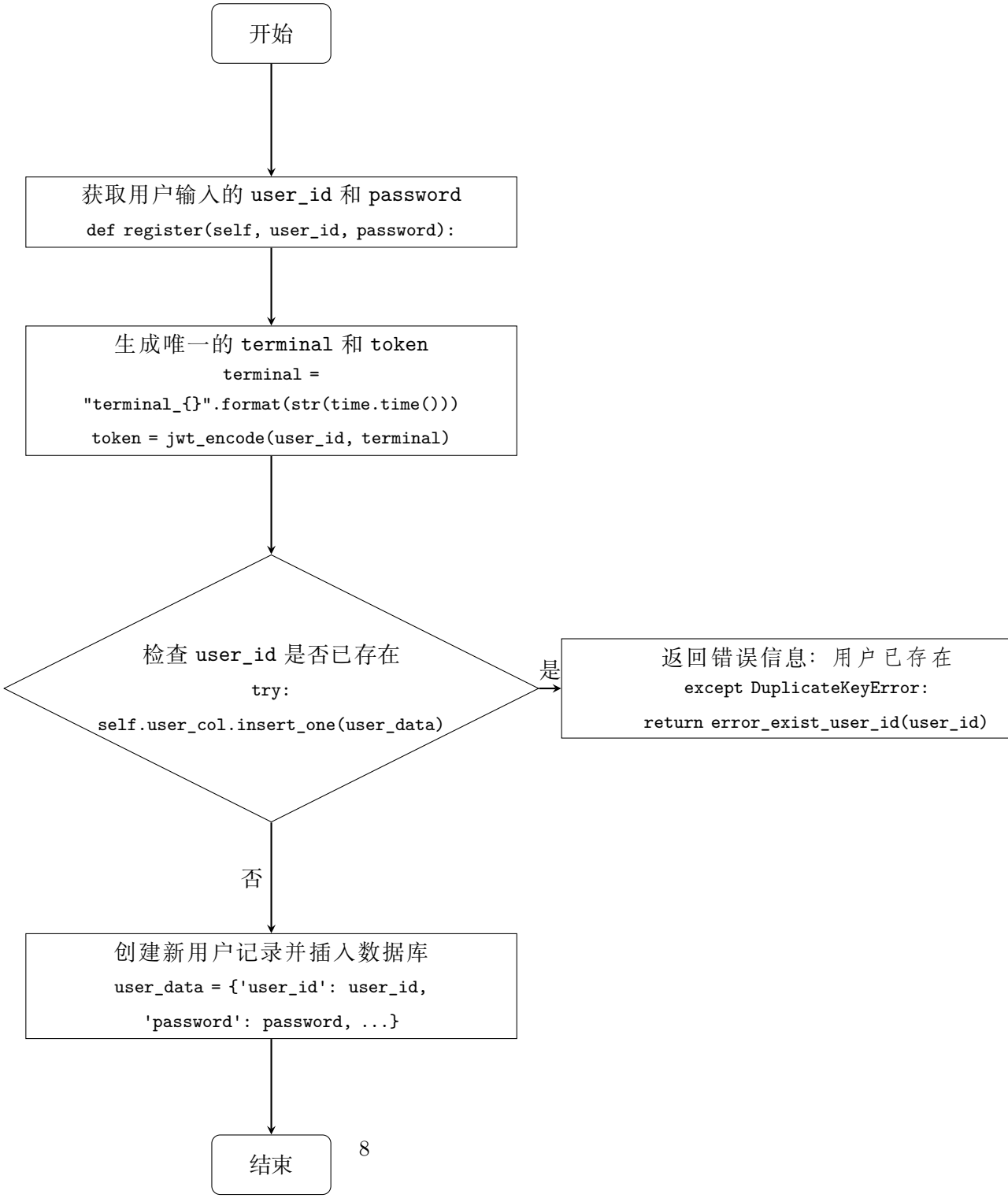

图 1: 用户注册流程图

3.3.2 用户登录

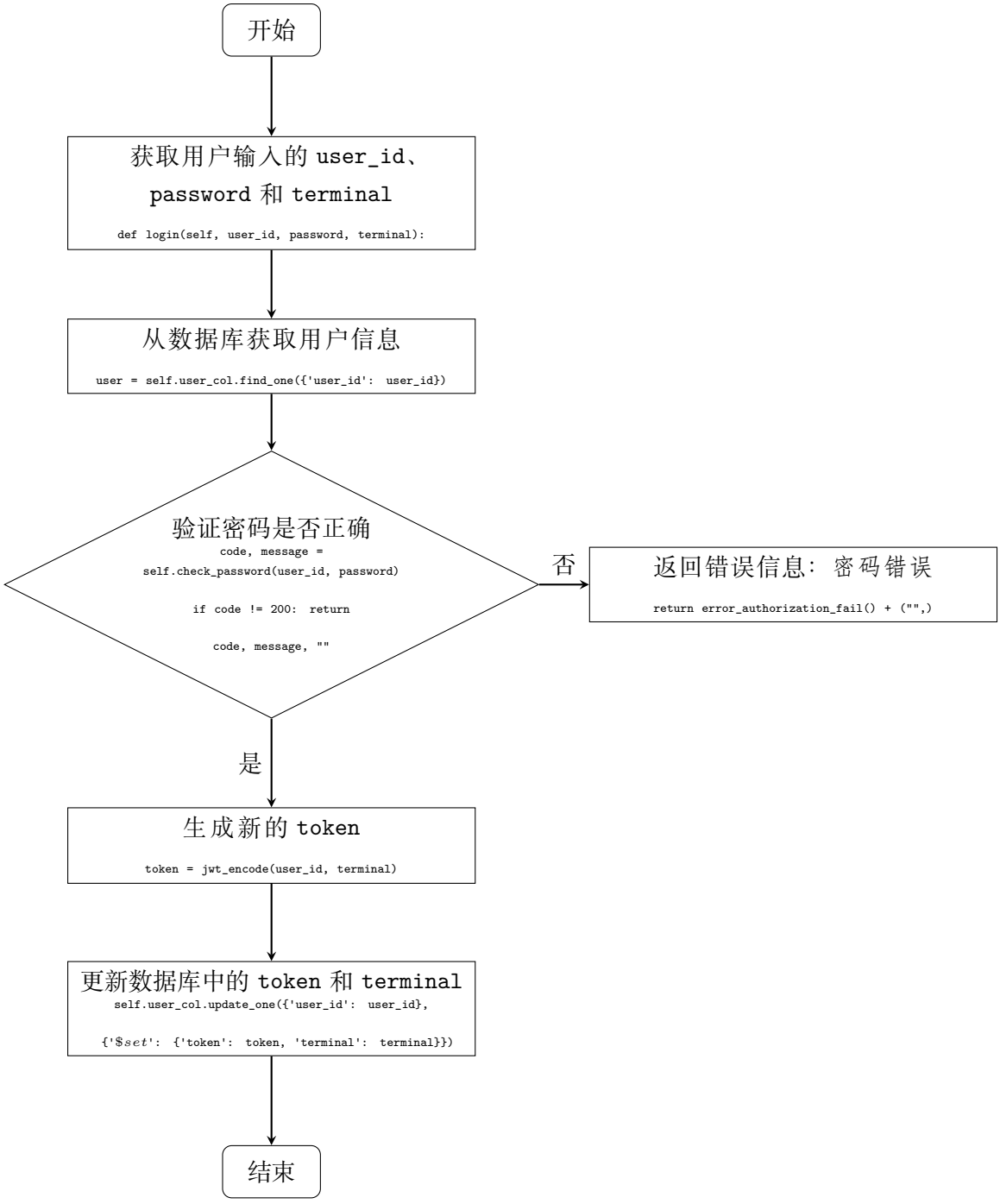

图 2: 用户登录流程图

3.3.3 用户注销

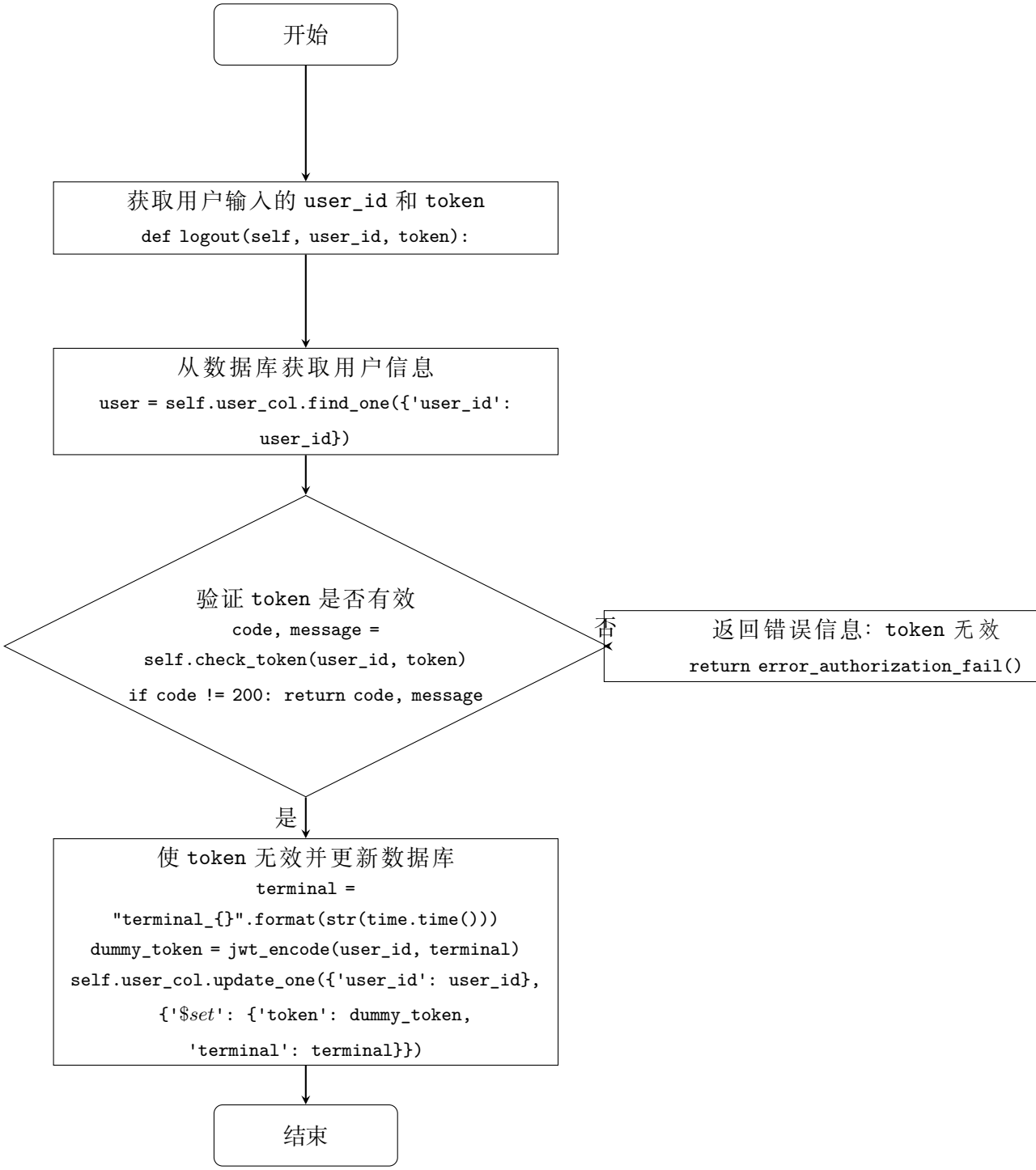

图 3: 用户注销流程图

4 买家接口

接口在 be/view/buyer.py 中实现,用于接收测试线程发来的 HTTP 请求，并解析参数，然后传递给后端的支持函数（存放在 be/model/buyer.py 中）。

后端逻辑在 be/model/buyer.py 中实现。修改了 new_order()、payment()、add_funds() 函数，以适配 MongoDB 数据库，主要逻辑与原来大致相同，详见代码注释。

沿用原本的测试函数。

4.1 接口概述

be/view/buyer.py 中定义了以下接口：

- /new_order：创建新订单
- /payment：支付订单
- /add_funds：充值资金
- /order_history：查询历史订单
- /cancel_order：取消订单
- /receive_order：收货

这些接口接收相应的 POST.GET 请求,提取请求中的参数(如 user_id、store_id、order_id、password、add_value 等),然后调用 be/model/buyer.py 中的对应方法，实现具体功能

4.2 后端逻辑

在 be/model/buyer.py 中，实现了买家功能的核心逻辑。主要包括以下类和方法：

4.2.1 类 Buyer

该类负责买家相关的所有操作，主要方法有：

- new_order()：创建新订单

- `payment()`：支付订单
- `add_funds()`：充值资金
- `get_user_orders()`：查询用户历史订单
- `cancel_order()`：取消订单
- `receive_order()`：确认收货

4.2.2 数据库适配

在初始化时，使用 PyMongo 连接到 MongoDB 数据库：

```
self.client = pymongo.MongoClient('mongodb://localhost:27017/')
self.db = self.client['bookstore']
self.buyer_col = self.db['buyer']
```

并确保相关字段的唯一性和索引：

```
self.buyer_col.create_index('user_id', unique=True)
self.buyer_col.create_index('order_id', unique=True)
```

4.3 买家接口流程

以下展示相关操作的流程图，具体代码可以在压缩包查看：

4.3.1 创建新订单

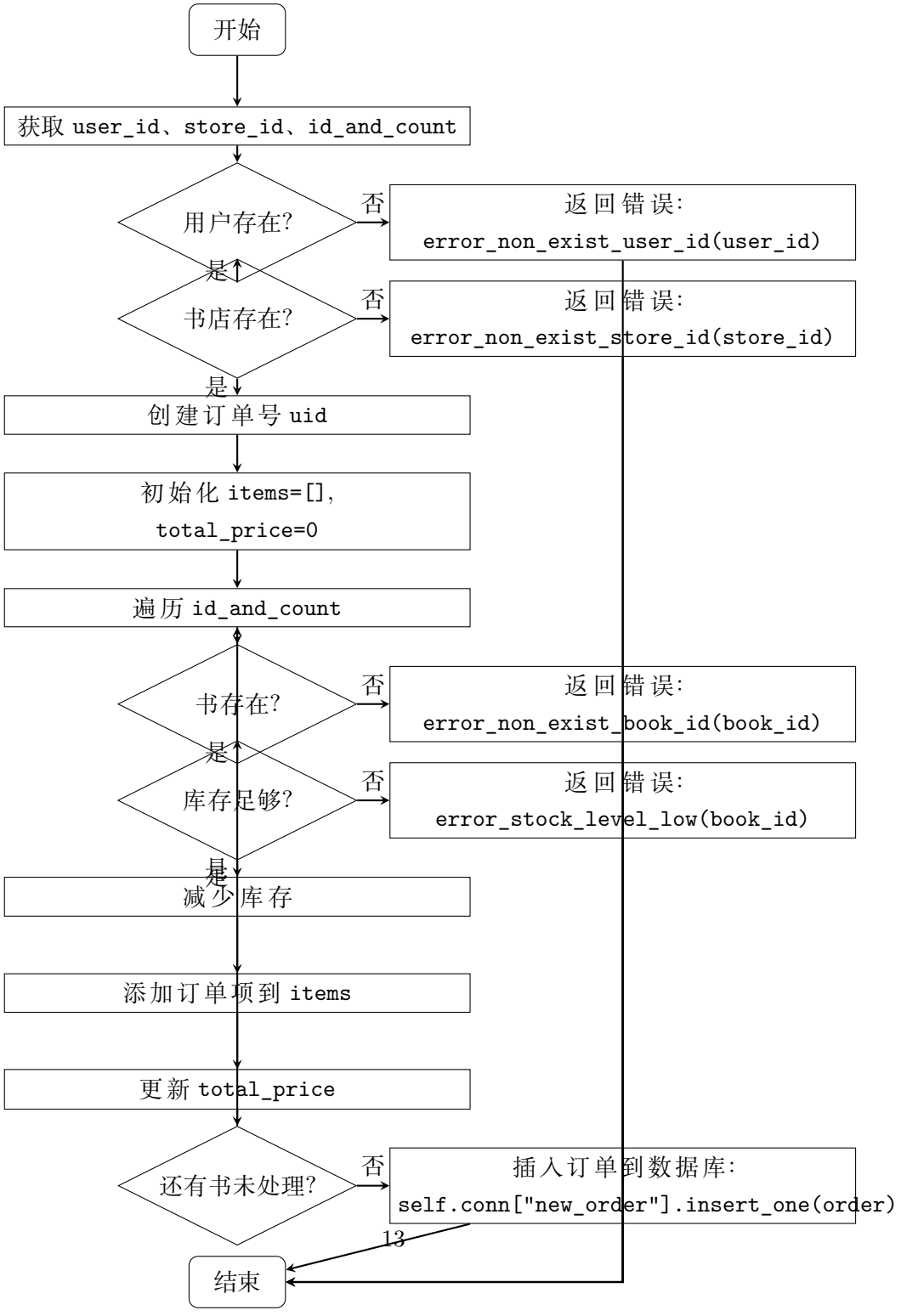

图 4: 创建新订单流程图

4.3.2 支付订单

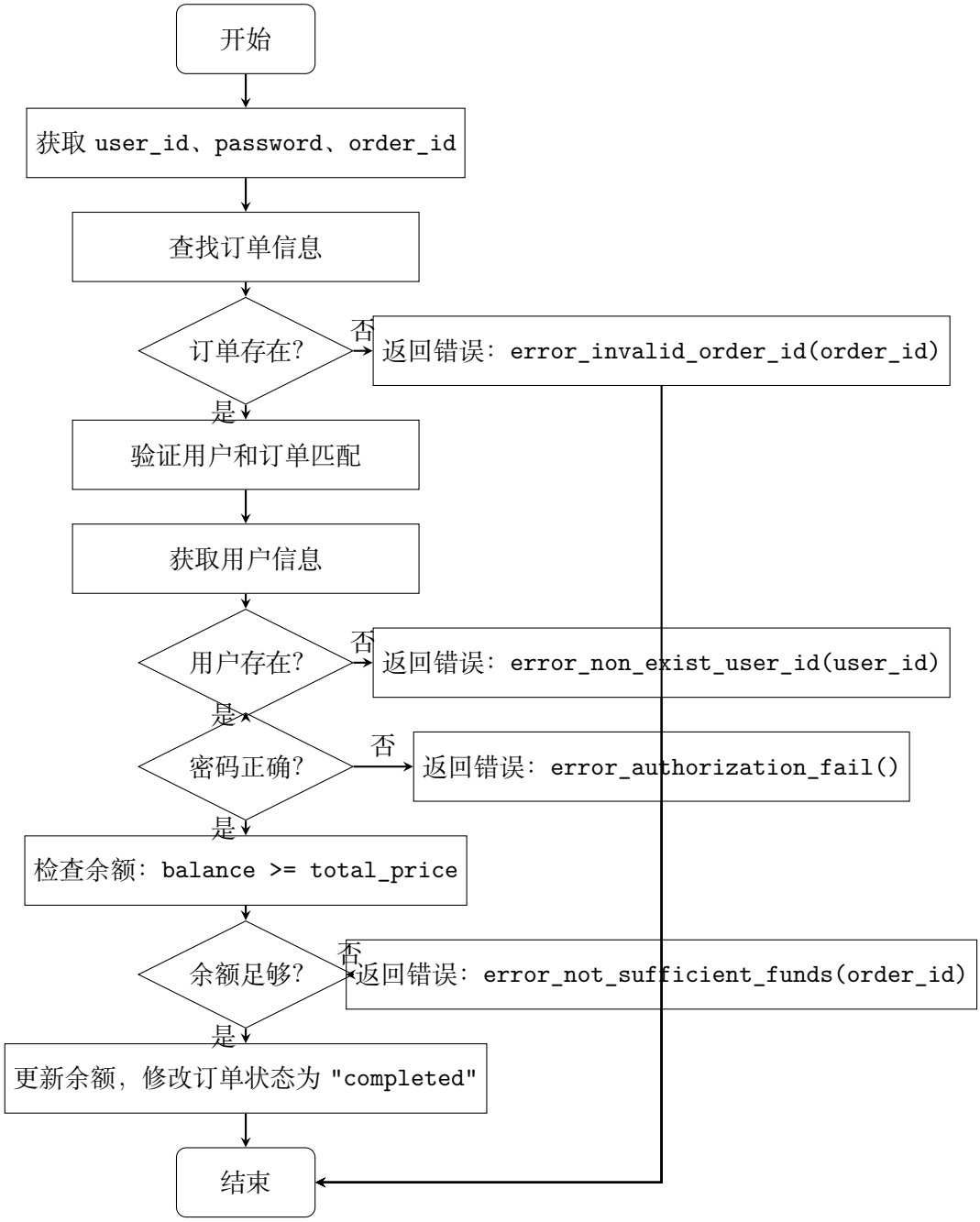

图 5: 支付订单流程图

4.3.3 充值资金

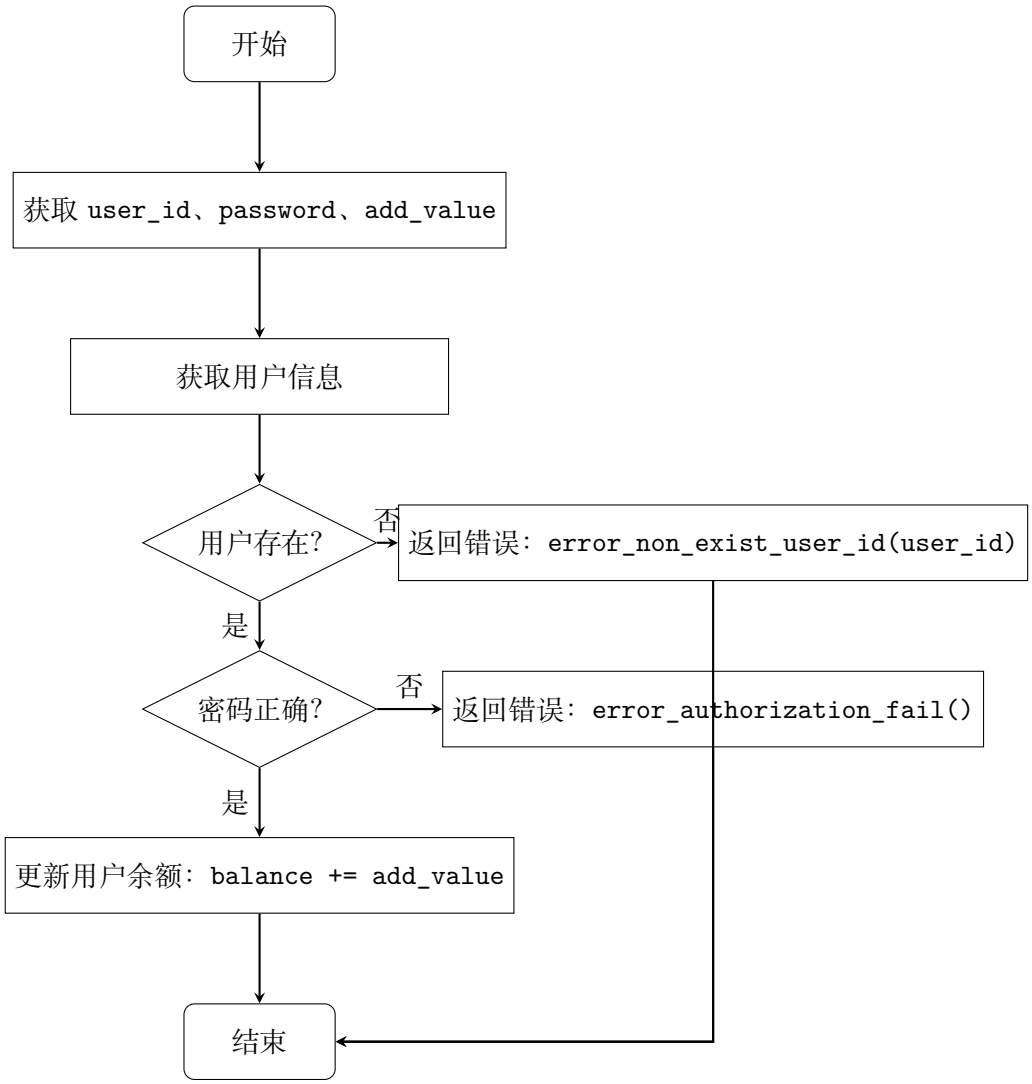

图 6: 充值资金流程图

5 卖家接口

接口在 be/view/seller.py 中实现，用于接收测试线程发来的 HTTP 请求,并解析参数,然后传递给后端的支持函数(存放在 be/model/seller.py

中)。

后端逻辑在 be/model/seller.py 中实现。实现了 create_store()、add_book()、add_stock_level() 等函数，以适配 MongoDB 数据库，主要逻辑与原来大致相同，详见代码注释。

沿用原本的测试函数。

5.1 接口概述

be/view/seller.py 中定义了以下接口：

- /create_store：创建店铺
- /add_book：添加书籍
- /add_stock_level：增加库存
- /ship_order：发货

这些接口接收相应的 POST 请求，提取请求中的参数（如 user_id、store_id、book_id、book_info、stock_level 等),然后调用 be/model/seller.py 中的对应方法，实现具体功能。

5.2 后端逻辑

在 be/model/seller.py 中，实现了卖家功能的核心逻辑。主要包括以下类和方法：

5.2.1 类 Seller

该类负责卖家相关的所有操作，主要方法有：

- create_store()：创建店铺
- add_book()：添加书籍
- add_stock_level()：增加库存
- ship_order()：发货

本部分介绍前三个接口的实现，发货接口在报告的下一部分中

5.2.2 数据库适配

在初始化时，使用 PyMongo 连接到 MongoDB 数据库：

```
self.client = pymongo.MongoClient('mongodb://localhost:27017/')
self.db = self.client['bookstore']
self.seller_col = self.db['seller']
```

并确保相关字段的唯一性和索引：

```
self.seller_col.create_index('user_id')
self.seller_col.create_index('store_id', unique=True)
```

5.3 卖家接口流程

以下展示相关操作的流程图，具体代码可以在压缩包查看：

5.3.1 创建店铺

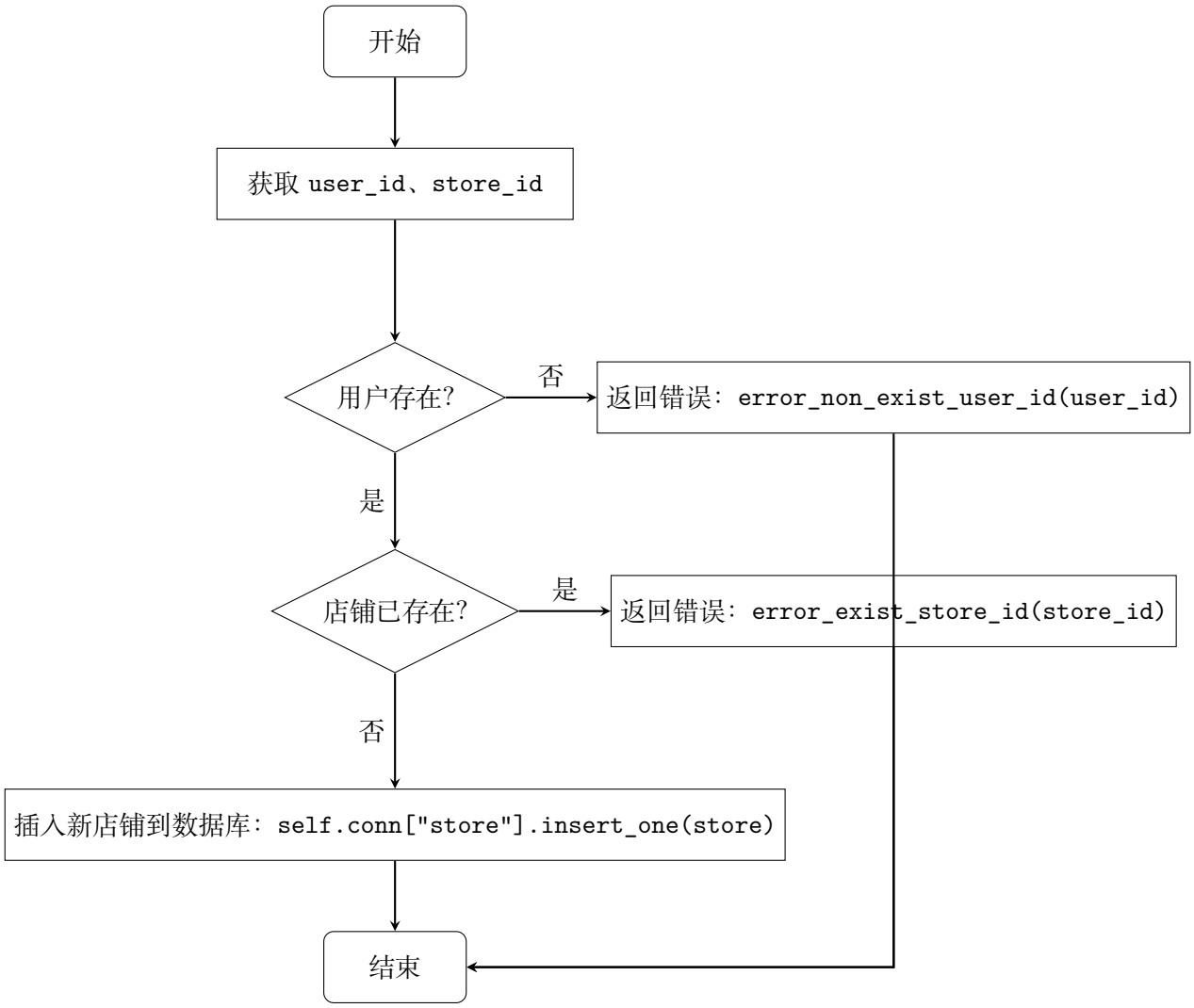

图 7: 创建店铺流程图

5.3.2 添加书籍

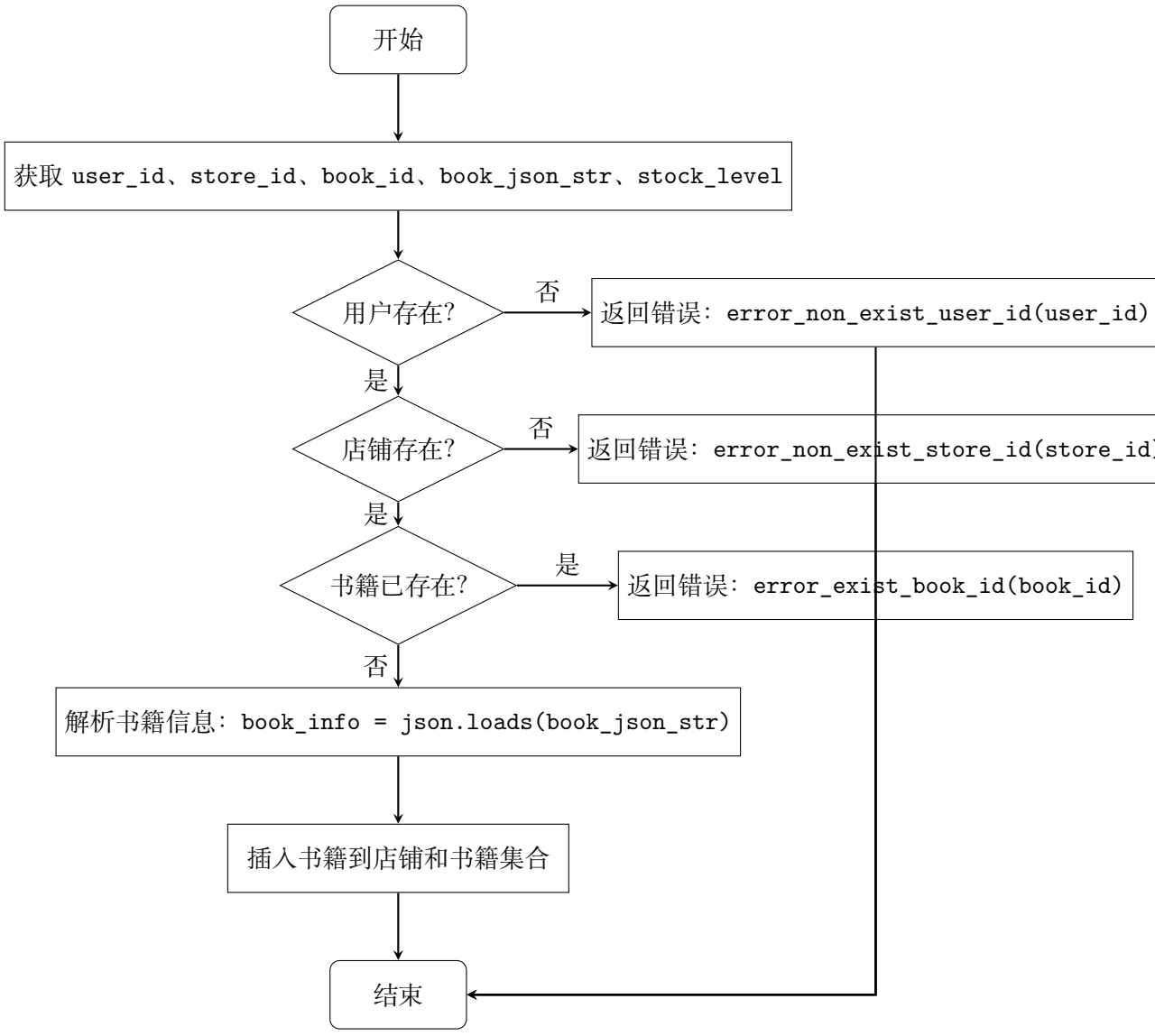

图 8: 添加书籍流程图

5.3.3 增加库存

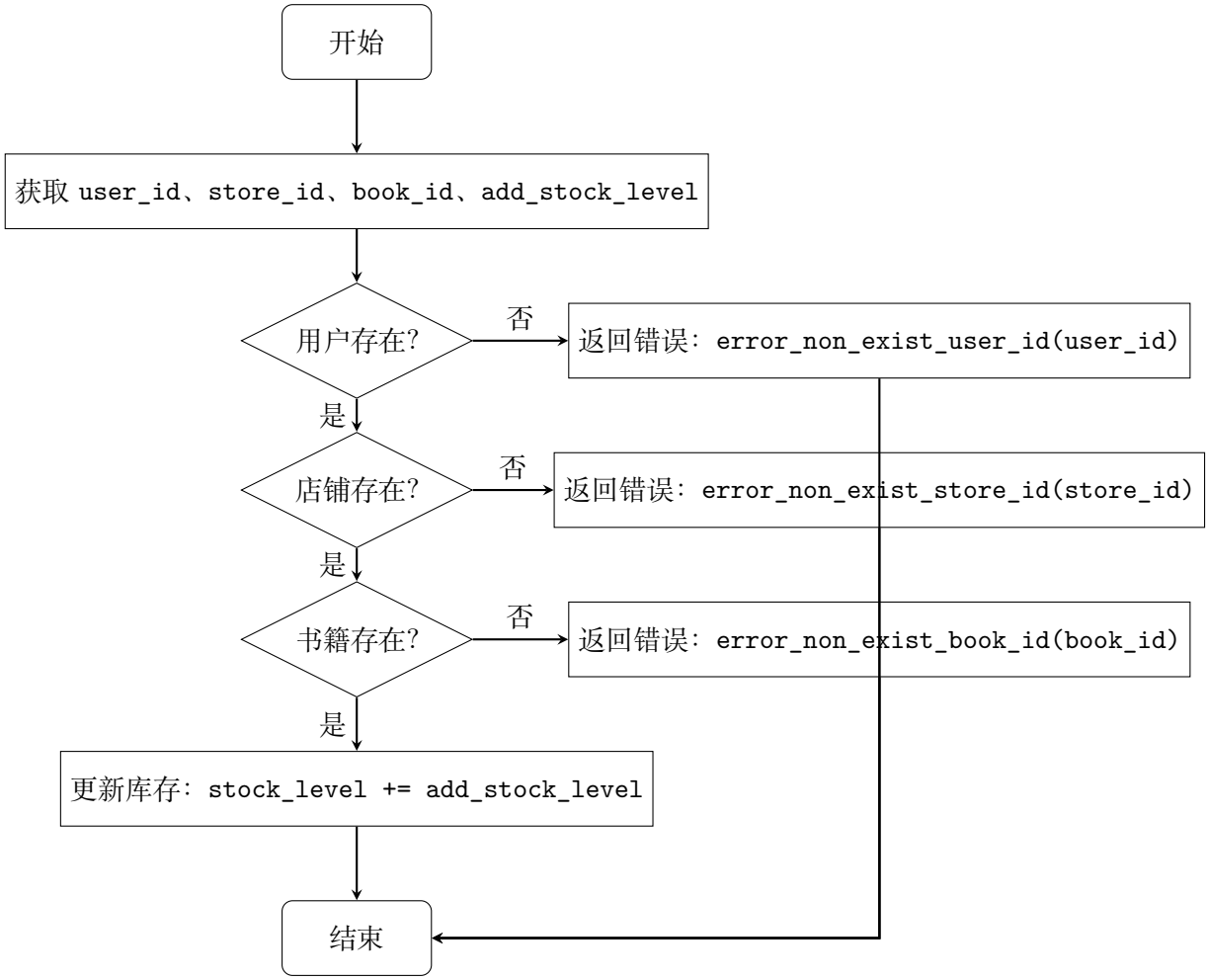

图 9: 增加库存流程图

6 额外功能一：发货收货

6.1 发货功能

功能描述:

`ship_order` 方法用于将指定订单的状态更新为 “已发货”。首先检查用户、书店和订单是否存在，然后在数据库中查找符合条件的订单，将其状态

更新为“已发货”。如果一切正常，返回成功消息；如果发生错误（如订单不存在或数据库问题），则返回相应的错误信息。

核心代码:

```
def ship_order(self, user_id: str, store_id: str,
    order_id: str) -> (int, str):
    try:
        # 检查用户、书店和订单是否存在
        if not self.user_id_exist(user_id):
            return error.error_non_exist_user_id(
                user_id)
        if not self.store_id_exist(store_id):
            return error.error_non_exist_store_id(
                store_id)
        if not self.order_id_exist(order_id):
            return error.error_invalid_order_id(
                order_id)

        # 更新订单状态为“已发货”
        update_result = self.conn["new_order"].
            update_one(
            {"order_id": order_id, "store_id":
                store_id, "status": "completed"},
            {"$set": {"status": "shipped"}}
        )
        if update_result.matched_count == 0:
            return error.error_invalid_order_id(
                order_id)
    except PyMongoError as e:
        logging.error(f"Database error in ship_order:
            {e}", exc_info=True)
        return 528, f"{str(e)}"
    except Exception as e:
        logging.error(f"Unexpected error in ship_order
```

```
            :␣{e}")
        return 530, f"{str(e)}"
    return 200, "Order␣shipped␣successfully"
```

流程图:

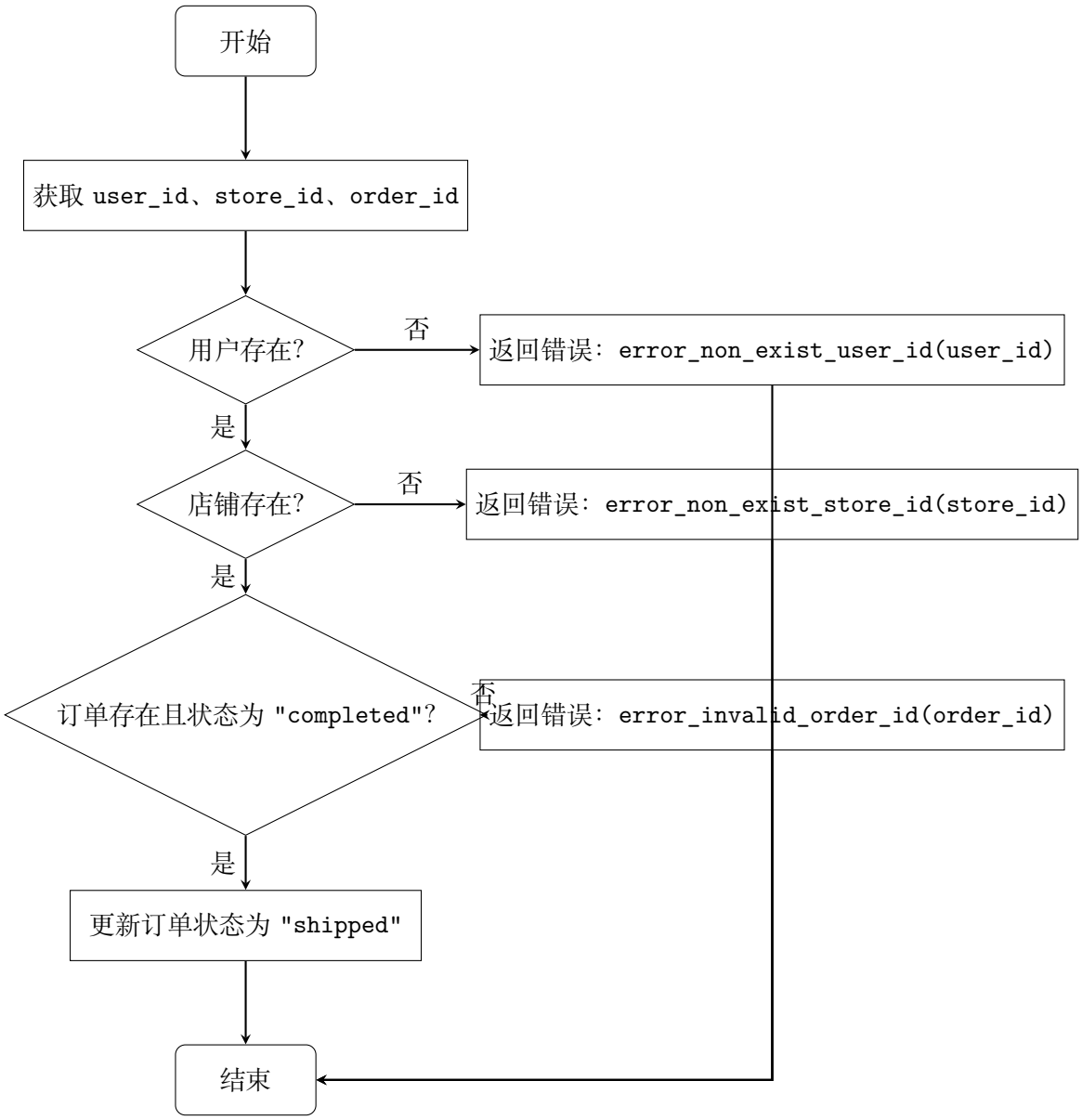

图 10: 发货功能流程图

接口实现:

使用 POST 请求:

```
@bp_seller.route("/ship_order", methods=["POST"])
```

```
def ship_order():
    user_id: str = request.json.get("user_id")
    store_id: str = request.json.get("store_id")
    order_id: str = request.json.get("order_id")

    s = seller.Seller()
    code, message = s.ship_order(user_id, store_id,
        order_id)

    return jsonify({"message": message}), code
```

6.2 收货功能

功能描述:

`receive_order` 方法根据订单 ID 和用户 ID 查找订单。如果订单状态不是“已发货”，则返回错误，提示订单未处于可收货状态。若订单状态为“已发货”，则更新订单状态为“已收货”。

核心代码:

```
def receive_order(self, user_id: str, order_id: str)
    -> (int, str):
    try:
        # 查找订单
        order = self.conn["new_order"].find_one({"
            order_id": order_id, "user_id": user_id})
        if order is None:
            return error.error_invalid_order_id(
                order_id)

        # 检查订单状态是否为“已发货”
        if order.get("status") != "shipped":
            return 400, "Order not in a receivable
                state"
```

```
            # 更新订单状态为 “已收货”
            self.conn["new_order"].update_one(
                {"order_id": order_id},
                {"$set": {"status": "received"}}
            )
        except PyMongoError as e:
            logging.error(f"Database error: {e}")
            return 528, f"{str(e)}"
        except Exception as e:
            logging.error(f"Unexpected error: {e}")
            return 530, f"{str(e)}"

        return 200, "Order received successfully"
```

流程图:

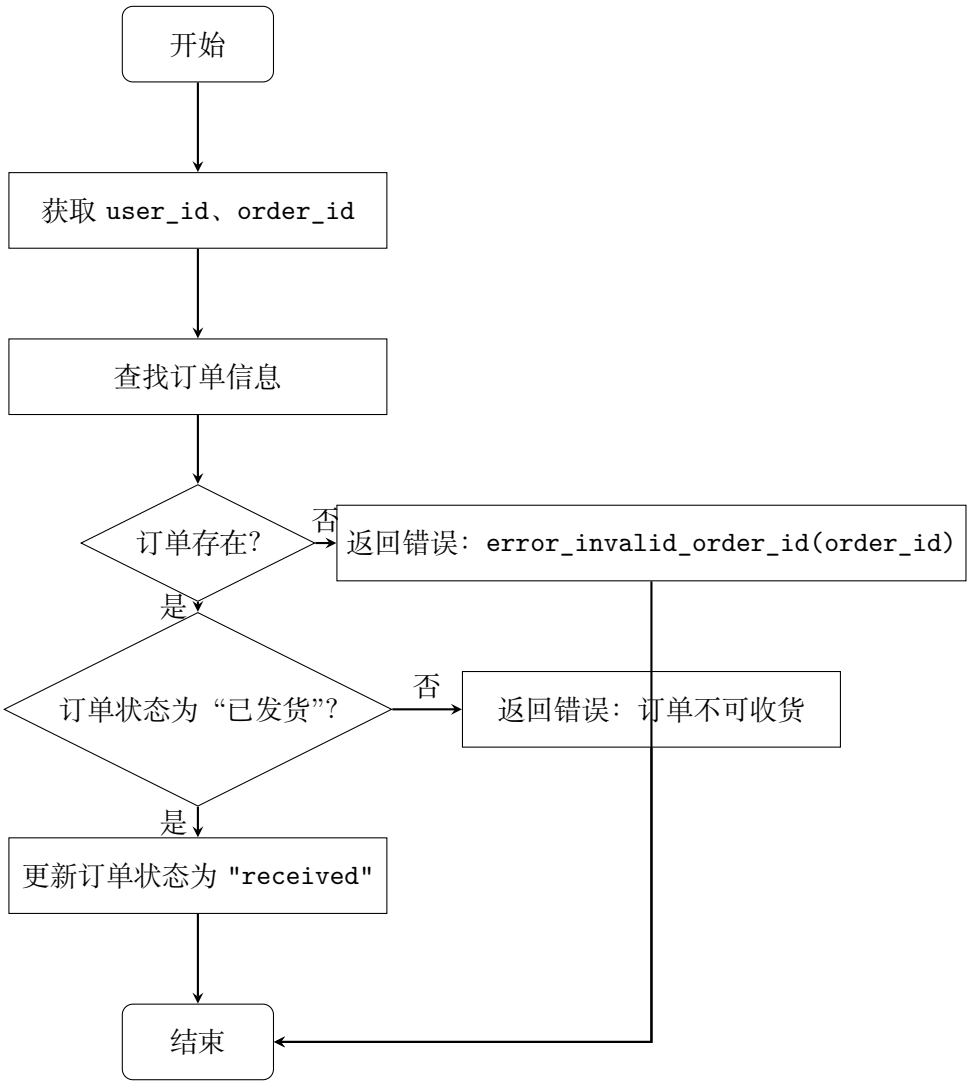

图 11: 收货功能流程图

接口实现:

使用 POST 请求:

```
@bp_buyer.route("/receive_order", methods=["POST"])
def receive_order():
    user_id: str = request.json.get("user_id")
    order_id: str = request.json.get("order_id")
```

```
    b = Buyer()
    code, message = b.receive_order(user_id, order_id)
    return jsonify({"message": message}), code
```

测试用例:

- **发货功能测试**:

```
class TestShipOrder:
    @pytest.fixture(autouse=True)
    def setup(self):
        # 初始化用户、书店和订单
        self.user_id = f"test_user_{str(uuid.uuid1
            ())}"
        self.store_id = f"test_store_{str(uuid.
            uuid1())}"
        self.order_id = f"test_order_{str(uuid.
            uuid1())}"
        self.password = "test_password"

        # 注册卖家并创建书店
        self.seller = register_new_seller(self.
            user_id, self.password)
        assert self.seller.create_store(self.
            store_id) == 200

        # 注册买家并下单
        self.buyer = register_new_buyer(f"buyer_{
            uuid.uuid1()}", "buyer_password")
        # 假设下单成功并返回订单ID
        self.order_id = self.buyer.place_order(
            self.store_id, book_id="book1",
            quantity=1)

    def test_ship_order_success(self):
```

```
        code, message = self.seller.ship_order(
            self.user_id, self.store_id, self.
            order_id)
        assert code == 200
        assert message == "Order shipped
            successfully"

    def test_ship_order_non_exist_order(self):
        non_exist_order_id = "non_exist_order"
        code, message = self.seller.ship_order(
            self.user_id, self.store_id,
            non_exist_order_id)
        assert code != 200
        assert "invalid_order_id" in message
```

- **收货功能测试**:

```
class TestReceiveOrder:
    @pytest.fixture(autouse=True)
    def setup(self):
        # 初始化用户、书店、订单
        self.user_id = f"test_user_{str(uuid.uuid1
            ())}"
        self.store_id = f"test_store_{str(uuid.
            uuid1())}"
        self.password = "test_password"
        self.order_id = "fake_order_id"

        # 注册卖家并创建书店
        self.seller = register_new_seller(self.
            user_id, self.password)
        assert self.seller.create_store(self.
            store_id) == 200
```

```
        # 注册买家并创建一个订单
        self.buyer = register_new_buyer(self.
            user_id, self.password)
        self.buyer.add_funds(1000)
        self.order_id = self.buyer.new_order(self.
            store_id, [{"id": "book1", "count":
            1}])

    def test_receive_order_success(self):
        # 模拟订单发货状态
        self.seller.ship_order(self.user_id, self.
            store_id, self.order_id)

        # 买家收货
        code, message = self.buyer.receive_order(
            self.user_id, self.order_id)
        assert code == 200
        assert message == "Order␣received␣
            successfully"

    def test_receive_order_not_shipped(self):
        # 买家在订单未发货时尝试收货
        code, message = self.buyer.receive_order(
            self.user_id, self.order_id)
        assert code == 400
        assert message == "Order␣not␣in␣a␣
            receivable␣state"

    def test_receive_order_invalid_order_id(self):
        # 使用不存在的订单 ID
        invalid_order_id = "non_exist_order"
        code, message = self.buyer.receive_order(
            self.user_id, invalid_order_id)
```

```
        assert code != 200
        assert "invalid_order_id" in message
```

7 额外功能二：搜索图书

7.1 功能描述

search_books 方法允许用户通过关键字搜索书籍，支持参数化搜索，包括标题、标签、目录和内容。可以指定搜索范围为全站或当前店铺，并支持分页显示结果。

7.2 核心代码

```
from be.model import db_conn
import logging
from pymongo.errors import PyMongoError

class Search(db_conn.DBConn):
    def __init__(self):
        super().__init__()

    def search_books(self, keywords: str, store_id:
        str = None, page: int = 1, page_size: int = 10)
        :
        try:
            # 构建查询条件
            query = {'$text': {'$search': keywords}}

            if store_id:
                # 如果指定了 store_id, 需要在该店铺的
                    库存中查找
                store = self.conn['store'].find_one({'
                    store_id': store_id})
                if not store:
```

```
                return 513, f"non␣exist␣store␣id␣{
                    store_id}", []

            # 获取该店铺的所有书籍ID
            book_ids_in_store = [book['book_id']
                for book in store.get('books', [])]
            if not book_ids_in_store:
                return 200, "ok", []  # 店铺中没有
                    书籍

            # 在查询条件中增加 book_id 过滤
            query['book_id'] = {'$in':
                book_ids_in_store}

        # 分页
        skip = (page - 1) * page_size

        # 查询
        cursor = self.conn['book'].find(query).
            skip(skip).limit(page_size)

        # 收集结果
        results = list(cursor)

        return 200, "ok", results

    except PyMongoError as e:
        logging.error(f"Database␣error␣in␣
            search_books:␣{e}")
        return 528, f"{str(e)}", []
    except Exception as e:
        logging.error(f"Unexpected␣error␣in␣
            search_books:␣{e}")
```

```
        return 530, f"{str(e)}", []
```

7.3 流程图

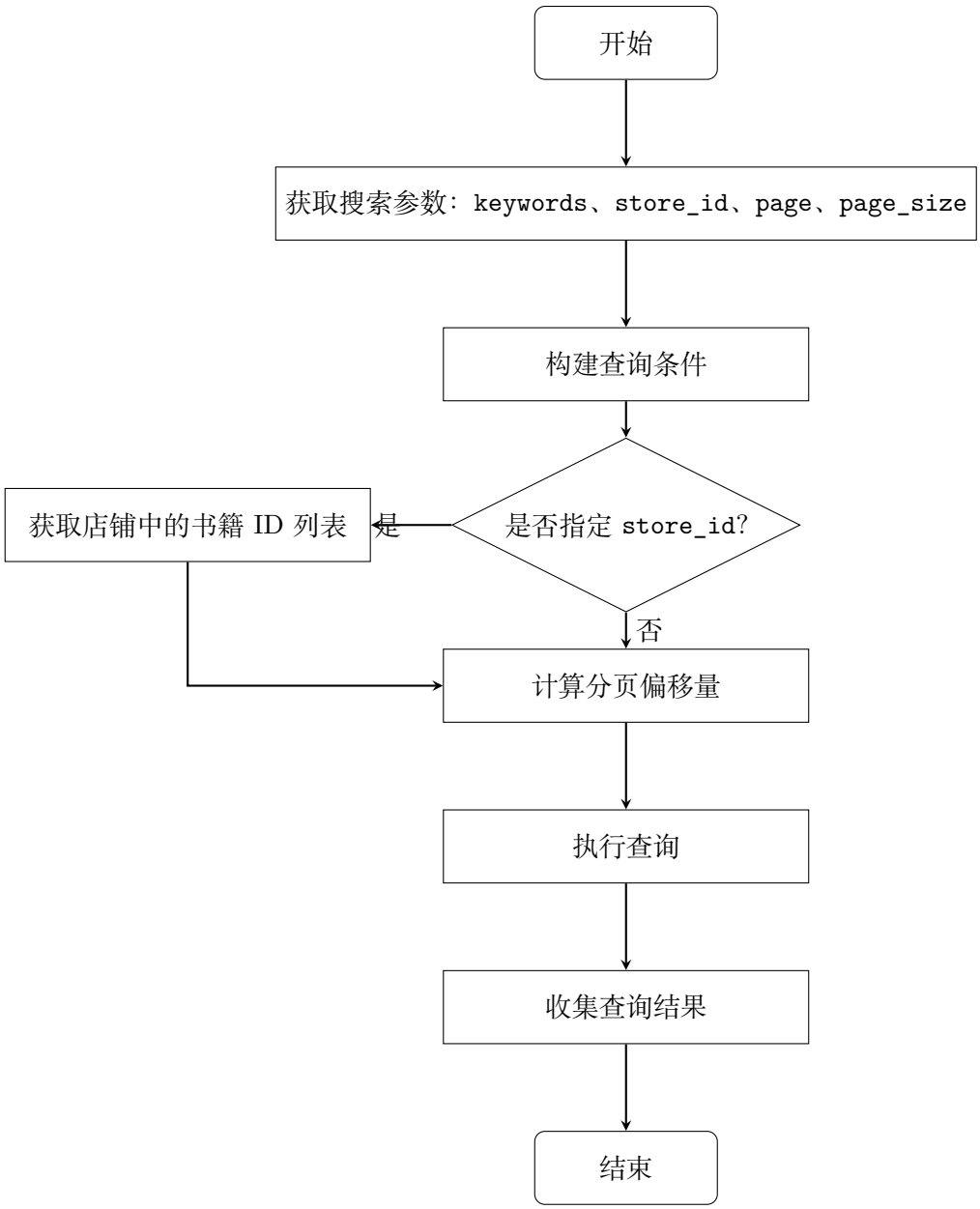

图 12: 搜索功能流程图

7.4 接口实现

使用 GET 请求：

```
from flask import Blueprint, request, jsonify
from be.model.search import Search

bp_search = Blueprint("search", __name__, url_prefix="
    /search")

@bp_search.route("/books", methods=["GET"])
def search_books():
    keywords = request.args.get("keywords", "")
    store_id = request.args.get("store_id")
    page = int(request.args.get("page", 1))
    page_size = int(request.args.get("page_size", 10))

    s = Search()
    code, message, results = s.search_books(keywords,
        store_id, page, page_size)
    return jsonify({"message": message, "results":
        results}), code
```

7.5 数据库索引优化

在 `model/store.py` 中，为 `book` 集合创建全文索引，以优化搜索性能：

```
def init_collections(self):
    try:
        # ... 省略部分代码
        # 为 book 集合创建全文索引
        indexes = self.db['book'].index_information()
        if 'book_full_text_index' not in indexes:
            self.db['book'].create_index(
                [
                    ('title', 'text'),
```

```
                    ('tags', 'text'),
                    ('catalog', 'text'),
                    ('content', 'text')
                ],
                name='book_full_text_index'
            )
        logging.info("MongoDB collections and indexes
            are initialized.")
    except errors.PyMongoError as e:
        logging.error(f"Error initializing MongoDB
            collections: {e}")
```

7.6 测试用例

针对搜索功能编写测试用例:

```
import pytest
import requests
import uuid

BASE_URL = "http://localhost:5000"

def test_search_books():
    # 使用唯一的ID
    unique_id = str(uuid.uuid4())
    user_id = f"test_user_{unique_id}"
    store_id = f"test_store_{unique_id}"
    book_id = f"test_book_{unique_id}"
    password = "test_password"

    # 注册用户
    response = requests.post(f"{BASE_URL}/auth/
        register", json={
         "user_id": user_id,
```

```
    "password": password
})
assert response.status_code == 200

# 登录用户
response = requests.post(f"{BASE_URL}/auth/login",
    json={
    "user_id": user_id,
    "password": password,
    "terminal": f"terminal_{unique_id}"
})
assert response.status_code == 200
token = response.json().get("token")

headers = {"Authorization": f"Bearer␣{token}"}

# 创建店铺
response = requests.post(f"{BASE_URL}/seller/
   create_store", json={
    "user_id": user_id,
    "store_id": store_id
}, headers=headers)
assert response.status_code == 200

# 添加书籍
book_info = {
    "id": book_id,
    "title": "Python␣Programming",
    "tags": ["Programming", "Python"],
    "catalog": "Chapter␣1:␣Introduction",
    "content": "This␣is␣a␣book␣about␣Python␣
       programming."
}
```

```
response = requests.post(f"{BASE_URL}/seller/
    add_book", json={
     "user_id": user_id,
     "store_id": store_id,
     "book_info": book_info,
     "stock_level": 10
}, headers=headers)
assert response.status_code == 200

# 测试店铺内搜索功能
params = {
     "keywords": "Python",
     "store_id": store_id,
     "page": 1,
     "page_size": 10
}
response = requests.get(f"{BASE_URL}/search/books"
    , params=params, headers=headers)
assert response.status_code == 200
results = response.json().get("results", [])
assert any(book['id'] == book_id for book in
    results)

# 测试全站搜索
params = {
     "keywords": "Programming",
     "page": 1,
     "page_size": 10
}
response = requests.get(f"{BASE_URL}/search/books"
    , params=params, headers=headers)
assert response.status_code == 200
```

```
    results = response.json().get("results", [])
    assert any(book['id'] == book_id for book in
        results)

    # 注销用户
    response = requests.post(f"{BASE_URL}/auth/
        unregister", json={
         "user_id": user_id,
         "password": password
    }, headers=headers)
    assert response.status_code == 200
```

8 额外功能三：订单状态

8.1 功能描述

本功能实现了订单的查询和取消，以及订单状态的管理。用户可以查看自己的历史订单，取消未付款的订单。订单有三种状态：`pending`（待付款）、`completed`（已完成）和 `cancelled`（已取消）。当订单支付成功后，订单状态更新为 `completed`；如果用户主动取消订单或订单超时未付款，订单状态更新为 `cancelled`。

8.2 核心代码

8.2.1 订单创建

在 `buyer.py` 的 `new_order` 函数中，当创建一个新订单时，订单状态被设定为 `pending`：

```
order = {
    "order_id": uid,
    "user_id": user_id,
    "store_id": store_id,
    "items": items,
    "total_price": total_price,
```

```
    "status": "pending",  # 订单状态为 'pending'
}
```

8.2.2 订单支付

在 buyer.py 的 payment 函数中，支付订单前首先检查订单状态是否为 pending:

```
if order.get("status") != "pending":
    return error.error_invalid_order_status(order_id)
```

支付成功后，更新订单状态为 completed:

```
self.conn["new_order"].update_one(
    {"order_id": order_id},
    {"$set": {"status": "completed"}}
)
```

8.2.3 订单查询

在 buyer.py 中实现了 get_user_orders 函数，用户可以查询自己的历史订单:

```
def get_user_orders(self, user_id: str):
    try:
        # 检查用户是否存在
        if not self.user_id_exist(user_id):
            return error.error_non_exist_user_id(
                user_id) + ([],)

        orders = self.conn["new_order"].find({"user_id
            ": user_id}).sort("created_at", -1)

        orders_list = []
        for order in orders:
            order["_id"] = str(order["_id"])  # 将
                ObjectId 转换为字符串
```

```
            orders_list.append(order)

        return 200, "ok", orders_list
    except PyMongoError as e:
        logging.error(f"Database error in
            get_user_orders: {e}")
        return 528, f"{str(e)}", []
```

8.2.4 取消订单

在 buyer.py 中实现了 cancel_order 函数,用户可以取消状态为 pending 的订单，同时库存会恢复:

```
def cancel_order(self, user_id: str, order_id: str):
    try:
        # 查询订单信息
        order = self.conn["new_order"].find_one({"
            order_id": order_id, "user_id": user_id})
        if order is None:
            return error.error_invalid_order_id(
                order_id)

        # 检查订单状态
        if order["status"] != "pending":
            return 400, "Order cannot be cancelled"

        # 更新订单状态为 `cancelled`
        self.conn["new_order"].update_one({"order_id":
            order_id}, {"$set": {"status": "cancelled"
            }})

        # 恢复库存
        for item in order["items"]:
            self.conn["store"].update_one(
```

```
                {"store_id": order["store_id"], "books
                    .book_id": item["book_id"]},
                {"$inc": {"books.$.stock_level": item[
                    "count"]}}
            )
    except PyMongoError as e:
        logging.error(f"Database error in cancel_order
            : {e}")
        return 528, f"{str(e)}"
    except Exception as e:
        logging.error(f"Unexpected error in
            cancel_order: {e}")
        return 530, f"{str(e)}"

    return 200, "ok"
```

8.3 流程图

8.3.1 订单查询流程图

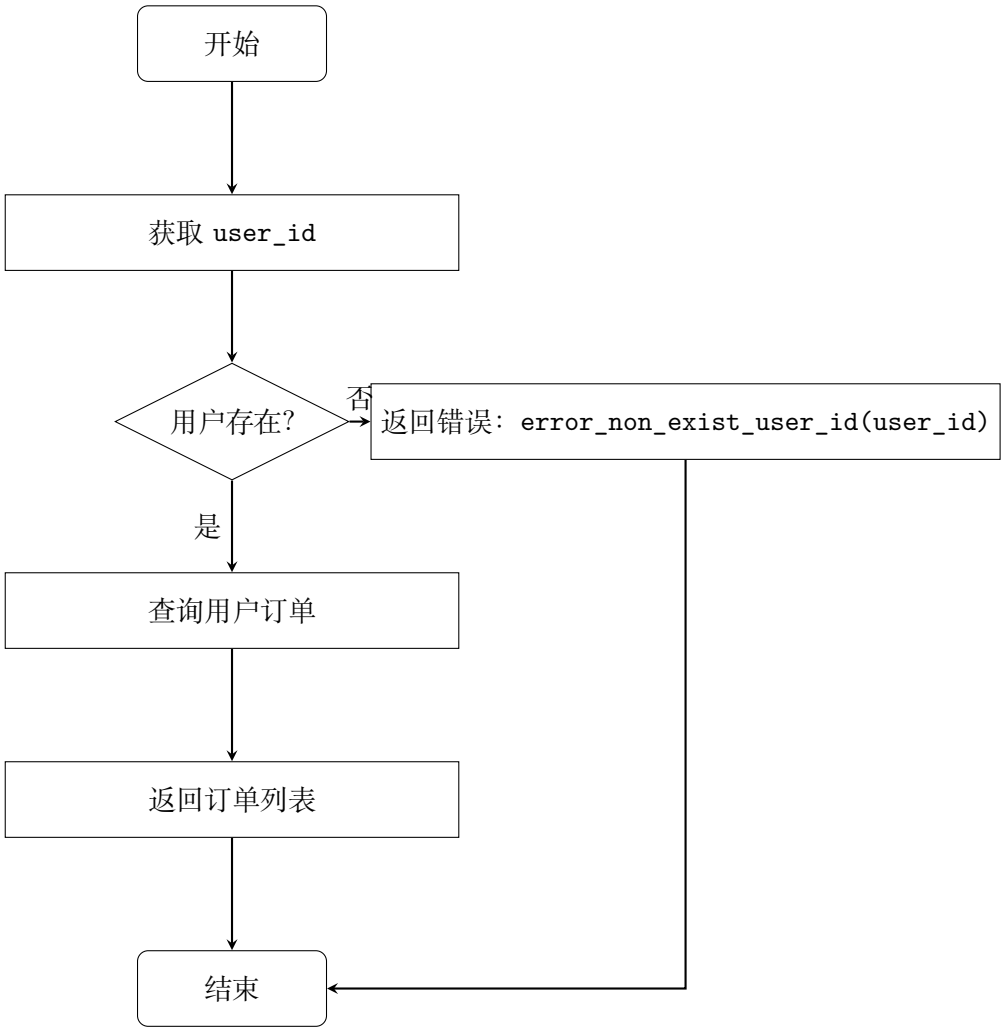

图 13: 订单查询流程图

8.3.2 取消订单流程图

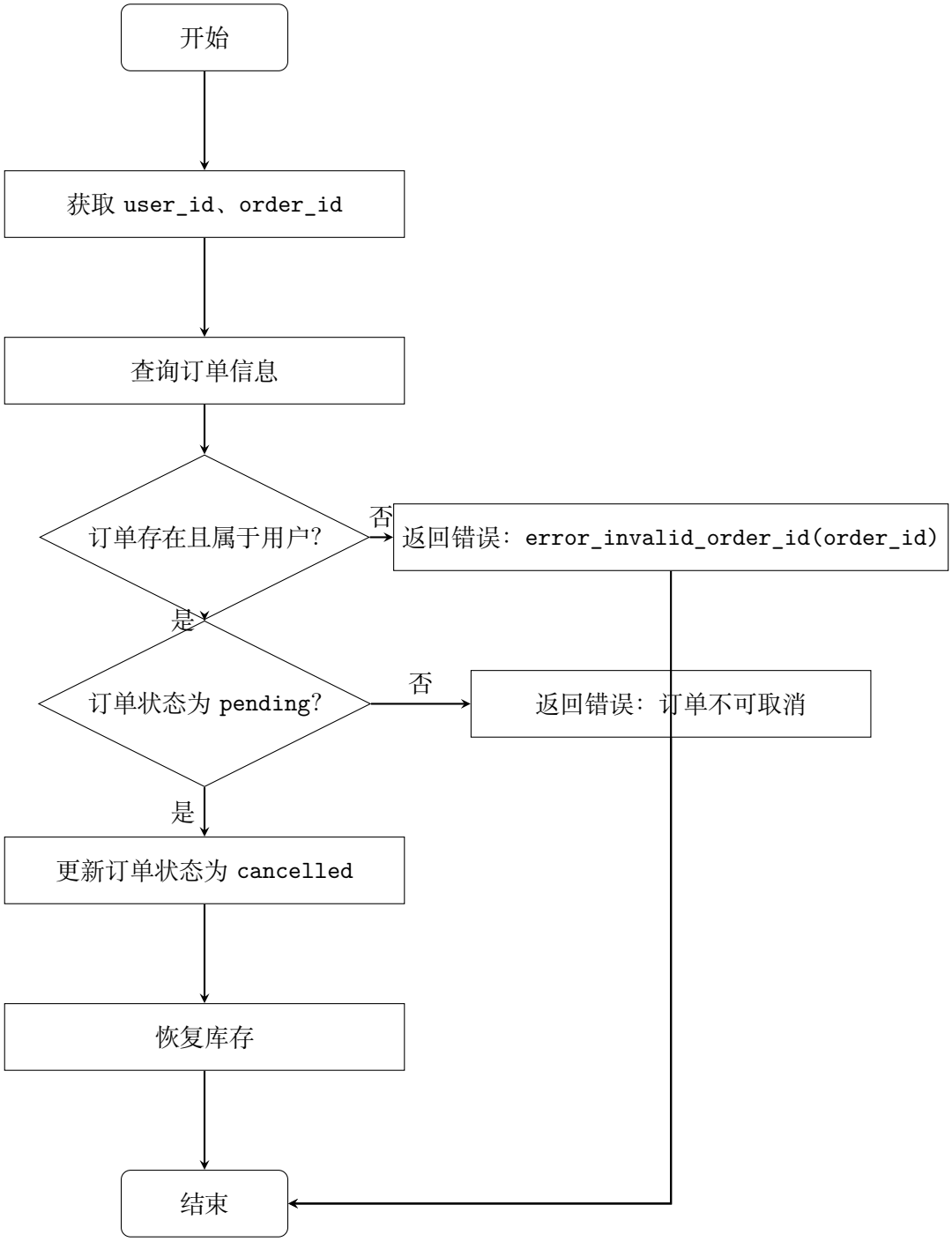

图 14: 取消订单流程图

8.4 接口实现

在 be/view/buyer.py 中，添加对应的接口实现。

8.4.1 订单查询接口

使用 GET 请求：

```
@bp_buyer.route("/order_history", methods=["GET"])
def order_history():
    user_id = request.args.get("user_id")
    b = Buyer()
    code, message, orders = b.get_user_orders(user_id)
    return jsonify({"message": message, "orders":
        orders}), code
```

8.4.2 取消订单接口

使用 POST 请求：

```
@bp_buyer.route("/cancel_order", methods=["POST"])
def cancel_order():
    user_id = request.json.get("user_id")
    order_id = request.json.get("order_id")
    b = Buyer()
    code, message = b.cancel_order(user_id, order_id)
    return jsonify({"message": message}), code
```

8.5 测试用例

8.5.1 订单查询测试

在 fe/test/test_order_history.py 中，测试了以下三种情况：

- 没有历史订单时，返回空列表。
- 有历史订单时，返回订单列表并包含新创建的订单。
- 用户不存在时，返回错误信息。

测试代码如下：

```
import pytest
from fe.access.new_buyer import register_new_buyer
from fe.access.new_seller import register_new_seller
from fe.test.gen_book_data import GenBook
import uuid

class TestOrderHistory:
    @pytest.fixture(autouse=True)
    def pre_run_initialization(self):
        self.seller_id = "
            test_order_history_seller_id_{}".format(str
            (uuid.uuid1()))
        self.store_id = "test_order_history_store_id_
            {}".format(str(uuid.uuid1()))
        self.buyer_id = "test_order_history_buyer_id_
            {}".format(str(uuid.uuid1()))
        self.password = self.seller_id
        self.buyer = register_new_buyer(self.buyer_id,
             self.password)
        self.gen_book = GenBook(self.seller_id, self.
            store_id)
        yield

    def test_empty_order_history(self):
        # 查询没有订单的历史
        code, message, orders = self.buyer.
            order_history()
        assert code == 200
        assert len(orders) == 0

    def test_order_history_with_orders(self):
        # 生成订单并检查订单历史
```

```
        ok, buy_book_id_list = self.gen_book.gen(
            non_exist_book_id=False, low_stock_level=
            False, max_book_count=5)
        assert ok
        code, order_id = self.buyer.new_order(self.
            store_id, buy_book_id_list)
        assert code == 200

        # 检查订单是否出现在历史中
        code, message, orders = self.buyer.
            order_history()
        assert code == 200
        assert any(order["order_id"] == order_id for
            order in orders)

    def test_order_history_invalid_user(self):
        # 用一个不存在的用户 ID 查询订单历史
        self.buyer.user_id = self.buyer.user_id + "
            _invalid"
        code, message, orders = self.buyer.
            order_history()
        assert code != 200
```

8.5.2 取消订单测试

在 `fe/test/test_cancel_order.py` 中，测试了以下情况：

- 成功取消订单。
- 取消不存在的订单。
- 非法用户尝试取消订单。

测试代码如下：

```
import pytest
```

```
from fe.access.new_buyer import register_new_buyer
from fe.test.gen_book_data import GenBook
import uuid

class TestCancelOrder:
    @pytest.fixture(autouse=True)
    def pre_run_initialization(self):
        self.seller_id = "test_cancel_order_seller_id_
            {}".format(str(uuid.uuid1()))
        self.store_id = "test_cancel_order_store_id_{}
            ".format(str(uuid.uuid1()))
        self.buyer_id = "test_cancel_order_buyer_id_{}
            ".format(str(uuid.uuid1()))
        self.password = self.seller_id
        self.buyer = register_new_buyer(self.buyer_id,
             self.password)
        self.gen_book = GenBook(self.seller_id, self.
            store_id)
        yield

    def test_cancel_order_success(self):
        # 生成一个订单
        ok, buy_book_id_list = self.gen_book.gen(
            non_exist_book_id=False, low_stock_level=
            False, max_book_count=5)
        assert ok
        code, order_id = self.buyer.new_order(self.
            store_id, buy_book_id_list)
        assert code == 200

        # 成功取消订单
        code, message = self.buyer.cancel_order(
            order_id)
```

```
        assert code == 200
        assert message == "ok"

    def test_cancel_order_non_existent(self):
        # 尝试取消一个不存在的订单
        code, message = self.buyer.cancel_order("
            non_existent_order_id")
        assert code != 200

    def test_cancel_order_invalid_user(self):
        # 生成一个订单
        ok, buy_book_id_list = self.gen_book.gen(
            non_exist_book_id=False, low_stock_level=
            False)
        assert ok
        code, order_id = self.buyer.new_order(self.
            store_id, buy_book_id_list)
        assert code == 200

        # 修改用户 ID 来模拟无效用户
        self.buyer.user_id = self.buyer.user_id + "
            _invalid"
        code, message = self.buyer.cancel_order(
            order_id)
        assert code != 200
```

8.6 其他改动

为了实现订单状态的管理，我们还对数据库进行了以下修改：

- 在 `new_order` 集合中增加了 `status` 字段，用于存储订单状态。
- 订单状态可能的取值为：`pending`、`completed`、`cancelled`。

同时，在 `fe/access/buyer.py` 中，添加了对应的请求函数：

```
# 新增: 查询订单历史
def order_history(self):
    url = urljoin(self.url_prefix, "order_history")
    headers = {"token": self.token}
    params = {"user_id": self.user_id}
    r = requests.get(url, headers=headers, params=
        params)

    # 检查请求是否成功
    if r.status_code == 200:
        response_json = r.json()
        return r.status_code, response_json.get("
            message"), response_json.get("orders")
    else:
        # 返回错误信息
        return r.status_code, r.text, None

# 新增: 取消订单
def cancel_order(self, order_id: str):
    json = {
        "user_id": self.user_id,
        "order_id": order_id,
    }
    url = urljoin(self.url_prefix, "cancel_order")
    headers = {"token": self.token}
    r = requests.post(url, headers=headers, json=json)
    response_json = r.json()
    return r.status_code, response_json.get("message")
```

9 测试情况

9.1 通过情况

```
fe/test/test_add_book.py::TestAddBook::test_ok PASSED                    [  2%]
fe/test/test_add_book.py::TestAddBook::test_error_non_exist_store_id PASSED [  4%
fe/test/test_add_book.py::TestAddBook::test_error_exist_book_id PASSED   [  6%]
fe/test/test_add_book.py::TestAddBook::test_error_non_exist_user_id PASSED [  8%]
fe/test/test_add_funds.py::TestAddFunds::test_ok PASSED                  [ 11%]
fe/test/test_add_funds.py::TestAddFunds::test_error_user_id PASSED       [ 13%]
fe/test/test_add_funds.py::TestAddFunds::test_error_password PASSED      [ 15%]
fe/test/test_add_stock_level.py::TestAddStockLevel::test_error_user_id PASSED [ 1
fe/test/test_add_stock_level.py::TestAddStockLevel::test_error_store_id PASSED [
fe/test/test_add_stock_level.py::TestAddStockLevel::test_error_book_id PASSED [ 2
fe/test/test_add_stock_level.py::TestAddStockLevel::test_ok PASSED       [ 24%]
fe/test/test_bench.py::test_bench PASSED                                 [ 26%]
fe/test/test_cancel_order.py::TestCancelOrder::test_cancel_order_success PASSED [
fe/test/test_cancel_order.py::TestCancelOrder::test_cancel_order_non_existent PAS
fe/test/test_cancel_order.py::TestCancelOrder::test_cancel_order_invalid_user PAS
fe/test/test_create_store.py::TestCreateStore::test_ok PASSED            [ 35%]
fe/test/test_create_store.py::TestCreateStore::test_error_exist_store_id PASSED [
fe/test/test_login.py::TestLogin::test_ok PASSED                         [ 40%]
fe/test/test_login.py::TestLogin::test_error_user_id PASSED              [ 42%]
fe/test/test_login.py::TestLogin::test_error_password PASSED             [ 44%]
fe/test/test_new_order.py::TestNewOrder::test_non_exist_book_id PASSED   [ 46%]
fe/test/test_new_order.py::TestNewOrder::test_low_stock_level PASSED     [ 48%]
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  [ 55%]
fe/test/test_order_history.py::TestOrderHistory::test_empty_order_history PASSED
fe/test/test_order_history.py::TestOrderHistory::test_order_history_with_orders P
fe/test/test_order_history.py::TestOrderHistory::test_order_history_invalid_user
fe/test/test_password.py::TestPassword::test_ok PASSED                   [ 64%]
fe/test/test_password.py::TestPassword::test_error_password PASSED       [ 66%]
fe/test/test_password.py::TestPassword::test_error_user_id PASSED        [ 68%]
fe/test/test_payment.py::TestPayment::test_ok PASSED                     [ 71%]
fe/test/test_payment.py::TestPayment::test_authorization_error PASSED    [ 73%]
fe/test/test_payment.py::TestPayment::test_not_suff_funds PASSED         [ 75%]
fe/test/test_payment.py::TestPayment::test_repeat_pay PASSED             [ 77%]
fe/test/test_receive_order.py::TestReceiveOrder::test_receive_order_success PASSE
fe/test/test_receive_order.py::TestReceiveOrder::test_receive_order_not_shipped P
fe/test/test_receive_order.py::TestReceiveOrder::test_receive_order_invalid_order
fe/test/test_register.py::TestRegister::test_register_ok PASSED          [ 86%]
fe/test/test_register.py::TestRegister::test_unregister_ok PASSED        [ 88%]
fe/test/test_register.py::TestRegister::test_unregister_error_authorization PASSE
fe/test/test_register.py::TestRegister::test_register_error_exist_user_id PASSED
fe/test/test_search.py::test_search_books PASSED                         [ 95%]
fe/test/test_ship_order.py::TestShipOrder::test_ship_order_success PASSED [ 97%]
fe/test/test_ship_order.py::TestShipOrder::test_ship_order_non_exist_order PASSED
```

总体覆盖率为 90%

9.2 测试覆盖率

```
======================== 45 passed in 66.11s (0:01:06) ========================
frontend end test
No data to combine
Name                                  Stmts   Miss Branch BrPart  Cover
-------------------------------------------------------------------------
be\app.py                                 3      3      2      0     0%
be\model\buyer.py                       143     47     50     11    69%
be\model\db_conn.py                      13      0      0      0   100%
be\model\error.py                        23      1      0      0    96%
be\model\search.py                       27      8      6      2    70%
be\model\seller.py                       72     25     26      6    66%
be\model\store.py                        34      2      6      1    92%
be\model\user.py                        119     29     30      6    77%
be\serve.py                              38      1      2      1    95%
be\view\auth.py                          42      0      0      0   100%
be\view\buyer.py                         54      5      2      0    91%
be\view\search.py                        12      0      0      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                        62      0     10      1    99%
fe\access\buyer.py                       52      0      4      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                      43      5      0      0    88%
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     20      2    97%
fe\conf.py                               11      0      0      0   100%
fe\conftest.py                           19      0      0      0   100%
fe\test\gen_book_data.py                 49      1     16      2    95%
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_cancel_order.py             33      0      0      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_history.py            31      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_order.py            47      0      0      0   100%
fe\test\test_register.py                 31      0      0      0   100%
fe\test\test_search.py                   23      0      0      0   100%
fe\test\test_ship_order.py               23      0      0      0   100%
-------------------------------------------------------------------------
TOTAL                                  1562    132    216     35    90%
Wrote HTML report to htmlcov\index.html
```

测试程序全部通过

10 实验总结

在本次实验中，我们成功地基于 MongoDB 实现了一个功能完整的网上购书系统后端。我们通过重新微调设计数据库的 schema，优化了数据的存储结构，使其更适合文档型数据库的特点。在实现用户权限管理、买家和卖家功能的过程中，我们深入理解了 flask 框架的运行过程和组成部分，特别是对于 MongoDB 的操作方式和指令有了比较可观的熟练度，并利用其强大的查询和索引功能完成了功能的实现。

我们还增加了发货收货、订单状态管理和多维度书籍搜索等额外功能，丰富了系统的功能性和用户体验。在搜索功能中，我们利用全文索引和分页技术，实现了高效的书籍检索。再此之外，我们通过为相关字段创建索引，显著提高了数据库的查询效率。

在实验过程中，我们采用了 GitHub 进行版本控制，使用 overleaf 进行实验报告的团队协作书写，确保了团队协作的顺畅和代码的可维护性。通过编写全面的测试用例，我们验证了各模块的功能和稳定性，测试覆盖率也达到了 90% 以上。

本次实验使我们对文档型数据库有了更深入的理解，掌握了基于 MongoDB 进行后端开发的实用技能并且令我们深刻体会到了团队合作和良好代码规范的重要性。

11 小组分工

孙钊琳：实现用户权限接口和搜索图书功能，书写对应实验报告内容，代码仓库管理，实验报告整合排版，在项目管理和协调中起首要作用

刘江涛：实现买家用户接口和订单状态查询取消等功能，书写了对应实验报告内容，在项目的 bebug 环节，以及代码的实现中起首要作用

庄欣熠：实现了卖家用户接口以及发货和收货等功能，书写了对应的实验报告内容，并在项目的测试环节，以及测试用例编写中起首要作用