Lab5 图书管理系统

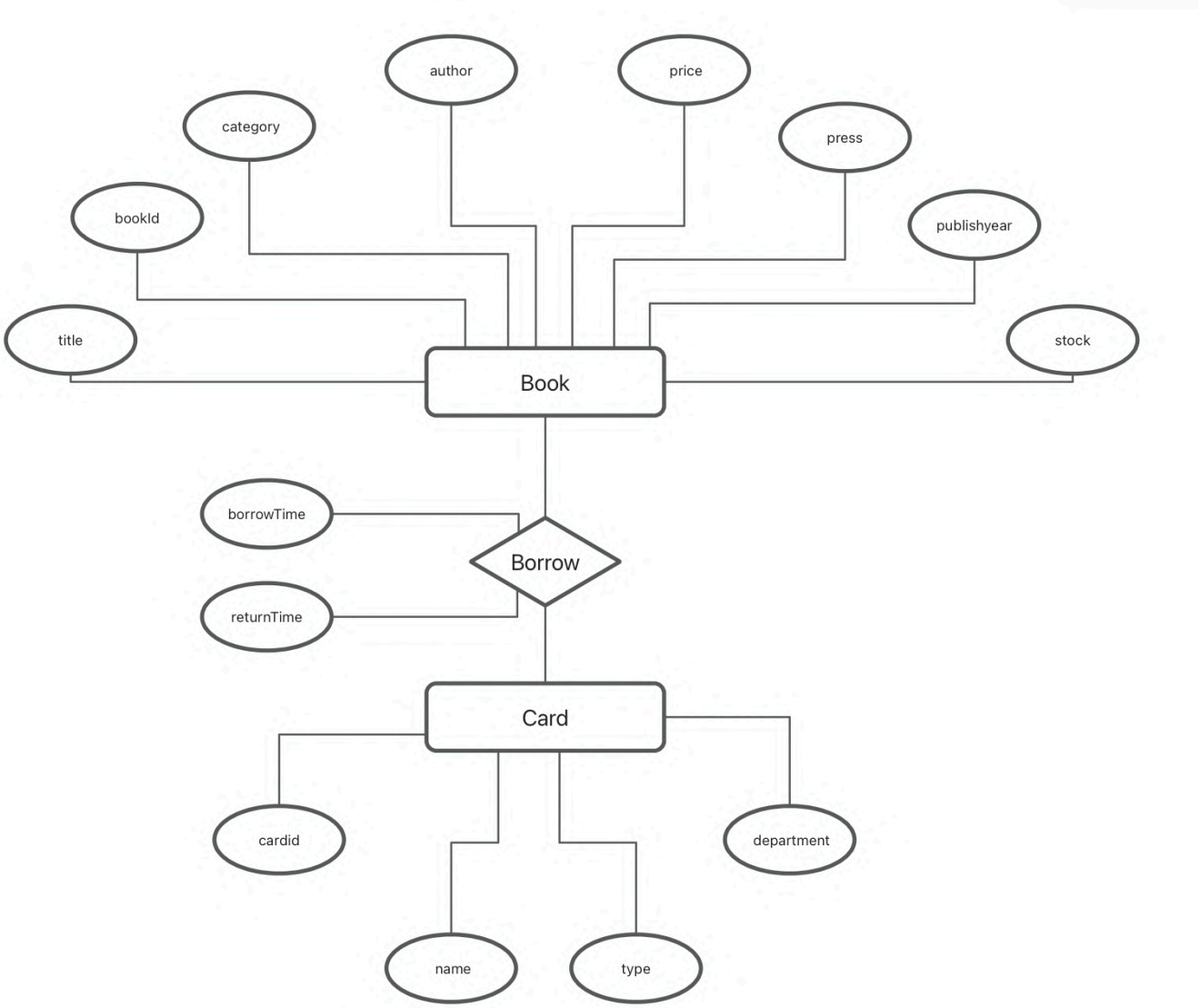
索引

简介

本项目旨在设计一个可用的图书管理系统。

本项目使用JDBC，mySql实现了后端数据库操作类，使用vue3实现了前端交互页面，使用axios和SpringBoot实现了前后端的交互。需要强调的是，本实验的主要目的在后端数据库操作的实现。

E-R图

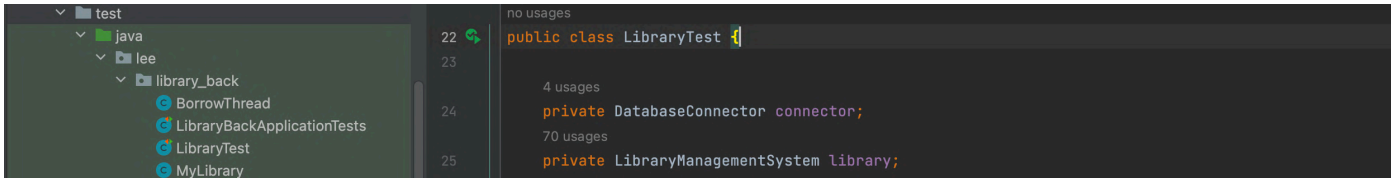


DDL

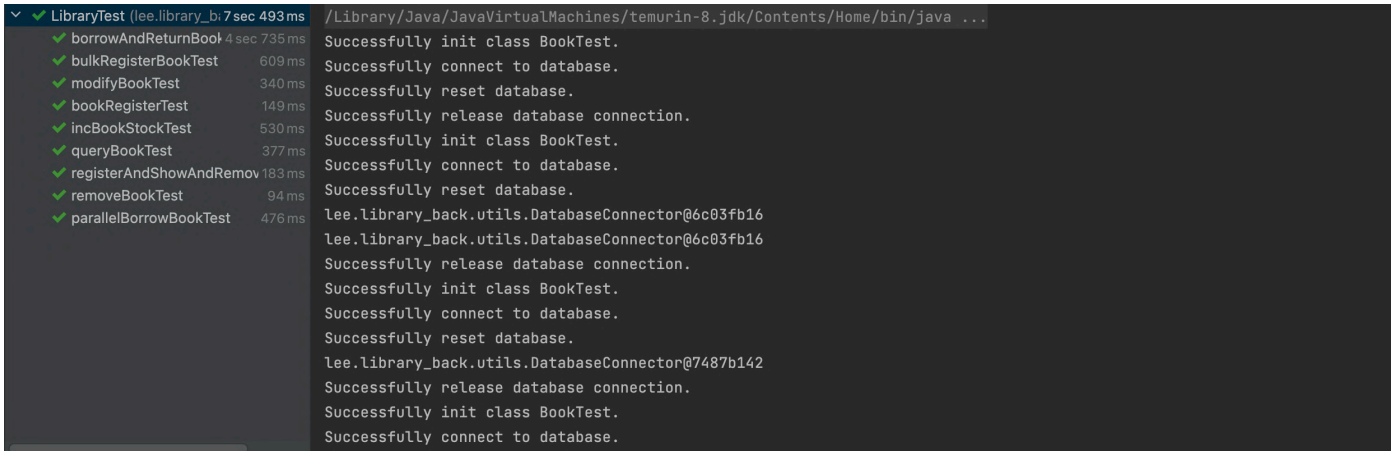
```
create table `book` (
    `book_id` int not null auto_increment,
    `category` varchar(63) not null,
    `title` varchar(63) not null,
    `press` varchar(63) not null,
    `publish_year` int not null,
    `author` varchar(63) not null,
    `price` decimal(7, 2) not null default 0.00,
    `stock` int not null default 0,
    primary key (`book_id`),
    unique (`category`, `press`, `author`, `title`, `publish_year`)
) engine=innodb charset=utf8mb4;

create table `card` (
    `card_id` int not null auto_increment,
    `name` varchar(63) not null,
    `department` varchar(63) not null,
    `type` char(1) not null,
    primary key (`card_id`),
    unique (`department`, `type`, `name`),
    check ( `type` in ('T', 'S') )
) engine=innodb charset=utf8mb4;

create table `borrow` (
  `card_id` int not null,
  `book_id` int not null,
  `borrow_time` bigint not null,
  `return_time` bigint not null default 0,
  primary key (`card_id`, `book_id`, `borrow_time`),
  foreign key (`card_id`) references `card`(`card_id`) on delete cascade on update
cascade,
  foreign key (`book_id`) references `book`(`book_id`) on delete cascade on update
cascade
) engine=innodb charset=utf8mb4;
```

效果演示

正确性检验



点击绿色图案运行所有测试（**注意**：由于项目结构已经变化，请**不要**使用`*mvn -Dtest=LibraryTest clean test*`运行测试）



正确通过所有测试阳历

功能性检验

功能简介

本项目的前端默认设置在http://localhost:5173，运行前后端项目后进入引导页。

首先需要注册卡，点击注册，输入身份，姓名，部门即可注册卡。点击注册后显示分配的卡号。

点击师生将会提示输入卡号，此时会进行卡号正确性检验，如果输入的卡号已注册会进入师生操作页。

此处点击各个按钮可以进行查询书籍，借书，还书，查看借书记录，注销卡的操作。

点击管理员按钮会进入管理员操作页，在此页点击各个按钮可以进行查询书籍，添加图书，删除图书，修改图书信息，修改库存，查询借书记录，查询所有卡，批量导入书的操作。

具体操作演示

1. **图书入库**

点击添加图书按钮，输入图书信息，点击添加。如果添加成功将会提示添加成功，并给出分配给书籍的id

查询此书，确定成功入库书籍

查询结果

关闭

书籍名	作者	bookId	类别	出版社	价格	出版年份	库存
数据库的艺术	jack	2	Computer Science	PressA	188.88	2000	50

2. **增加库存**

点击修改库存按钮，输入需要修改的图书编号和库存变化，点击修改。如果修改成功将会提示“修改成功”

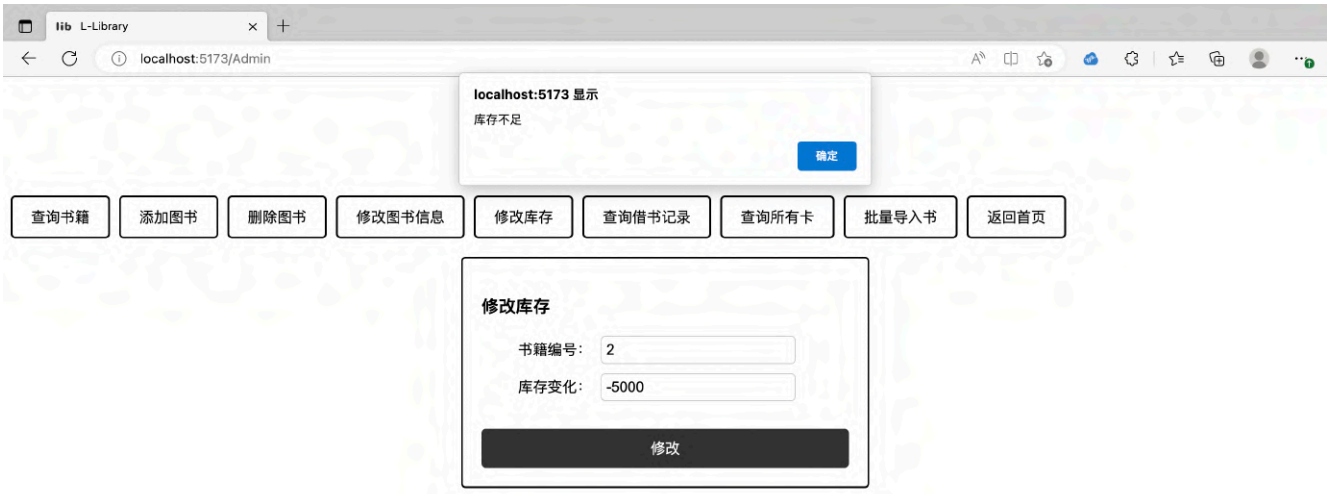
查询此书，确定修改库存成功：

查询结果

关闭

书籍名	作者	bookId	类别	出版社	价格	出版年份	库存
数据库的艺术	jack	2	Computer Science	PressA	188.88	2000	1050

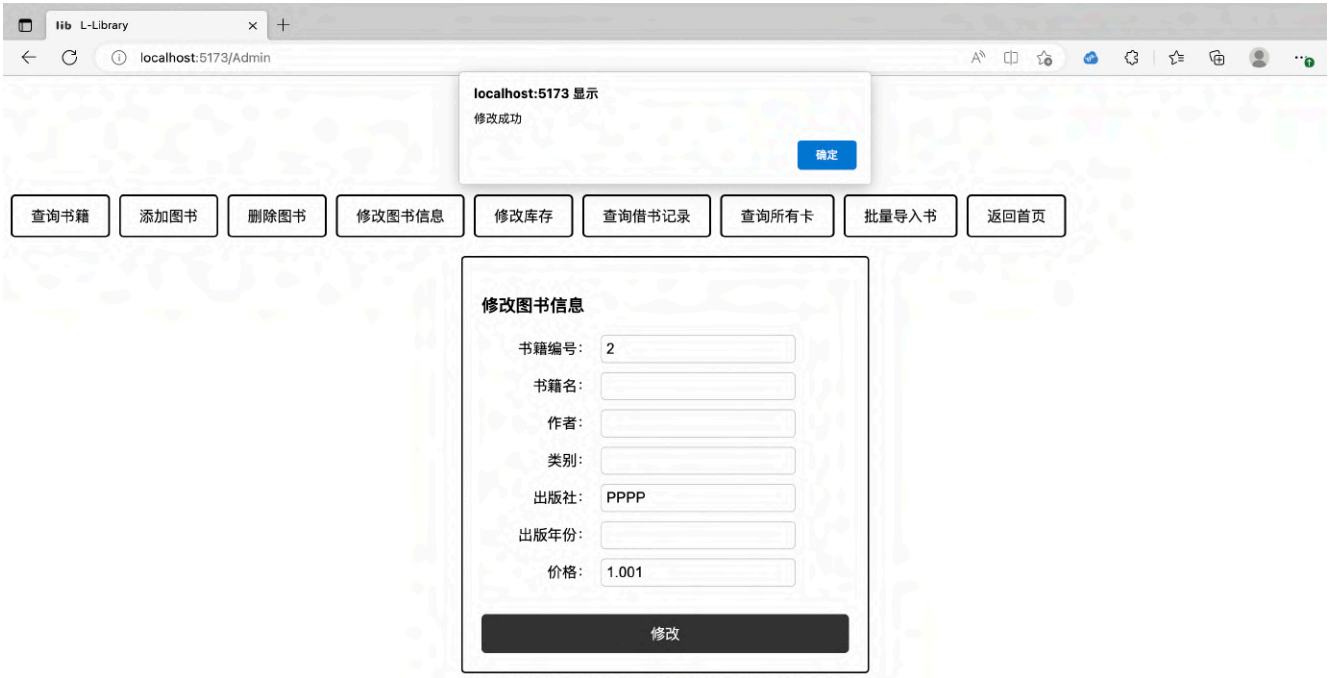
尝试异常操作，修改超过当前库存的变化量：



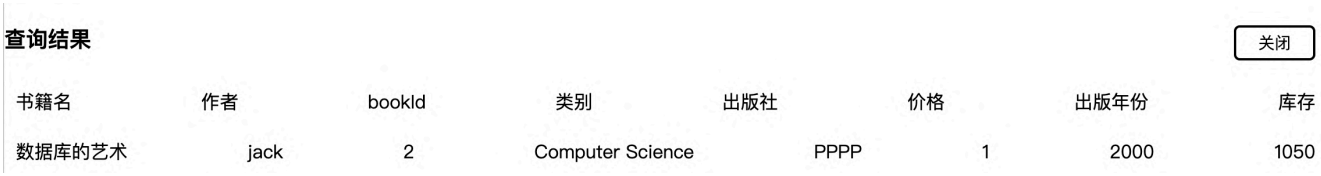
提示库存不足。

3. **修改图书信息**

   点击修改图书信息，输入需要修改的书籍编号，填入需要修改的属性。点击修改显示修改成功



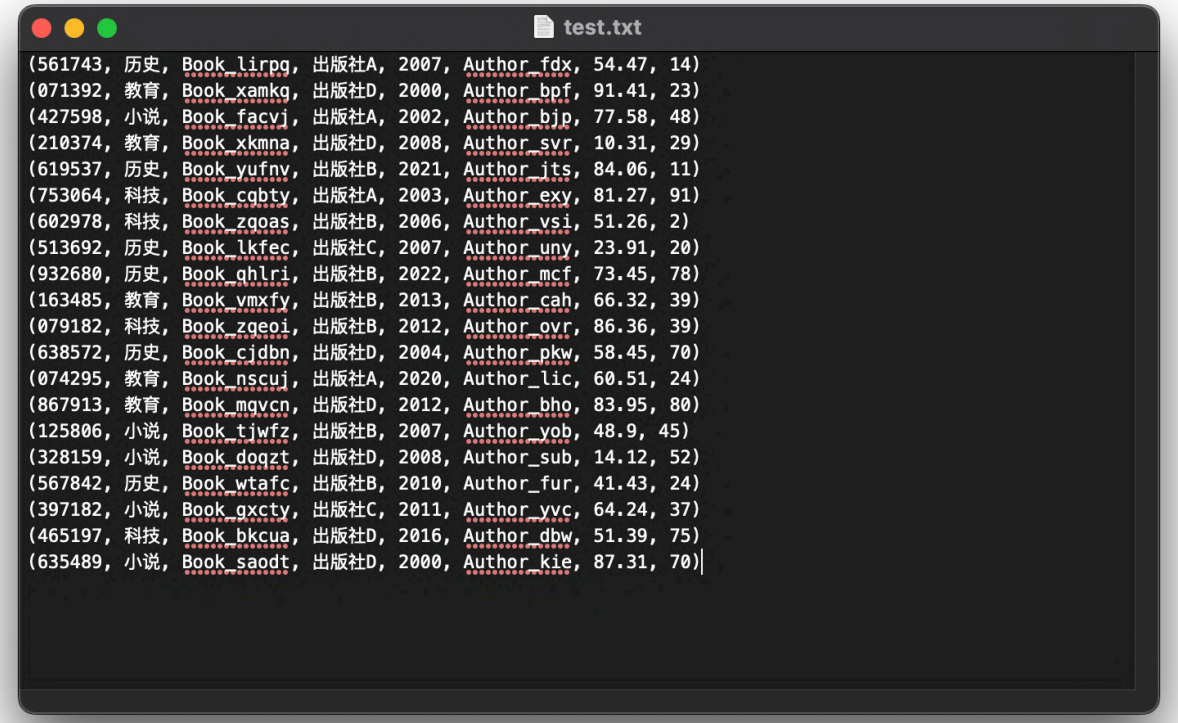
   查询此书，确定图书信息修改成功



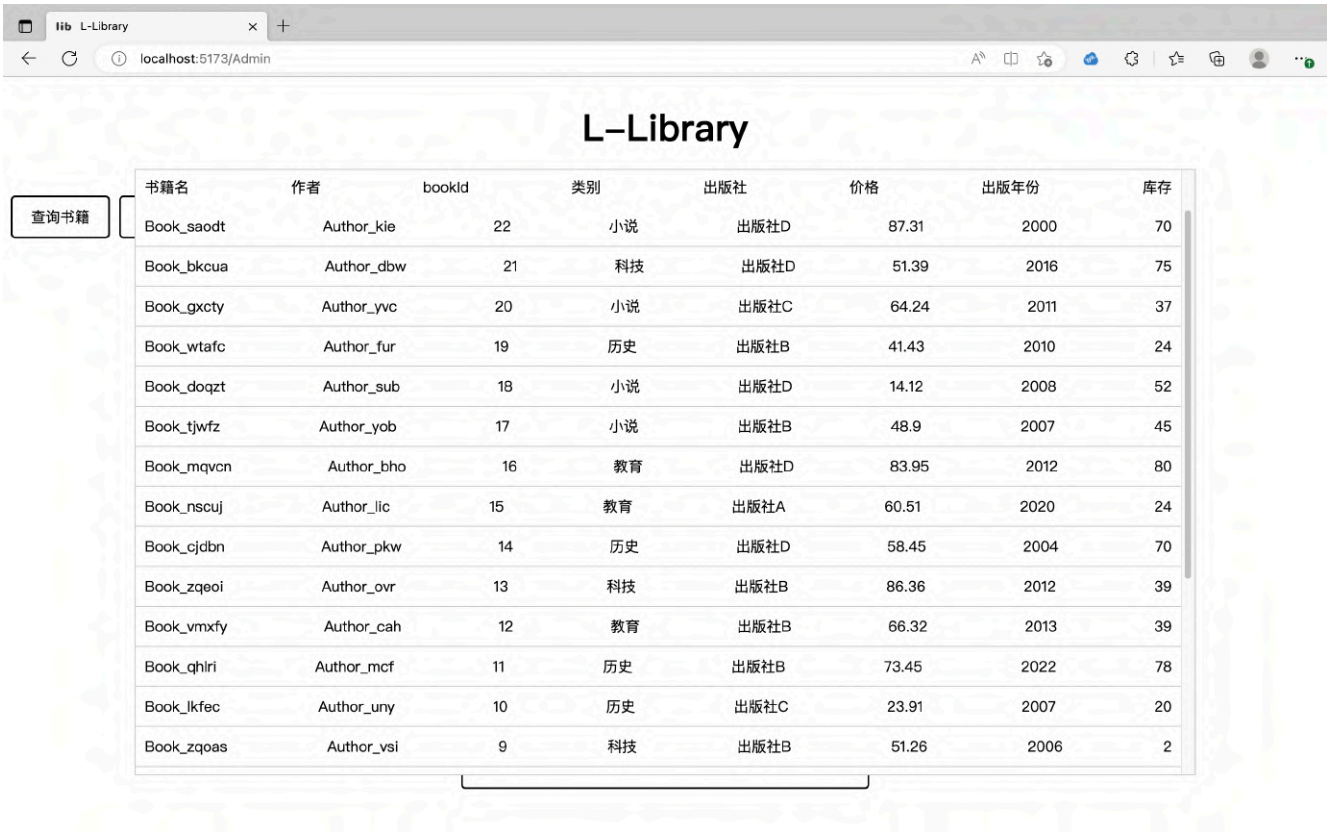
   同样的，异常操作如错误的编号或提交空白的修改会返回对应的错误提示，不一一列举。

4. **批量入库**

   按照格式（书号，类别，书名，出版社，年份，作者，价格，数量）随机生成一个测试数据文件。



点击批量导入书，选中需要导入的文件，点击批量导入。注意文件中的book_id不会真的被使用，书本的编号由数据库进行分配。



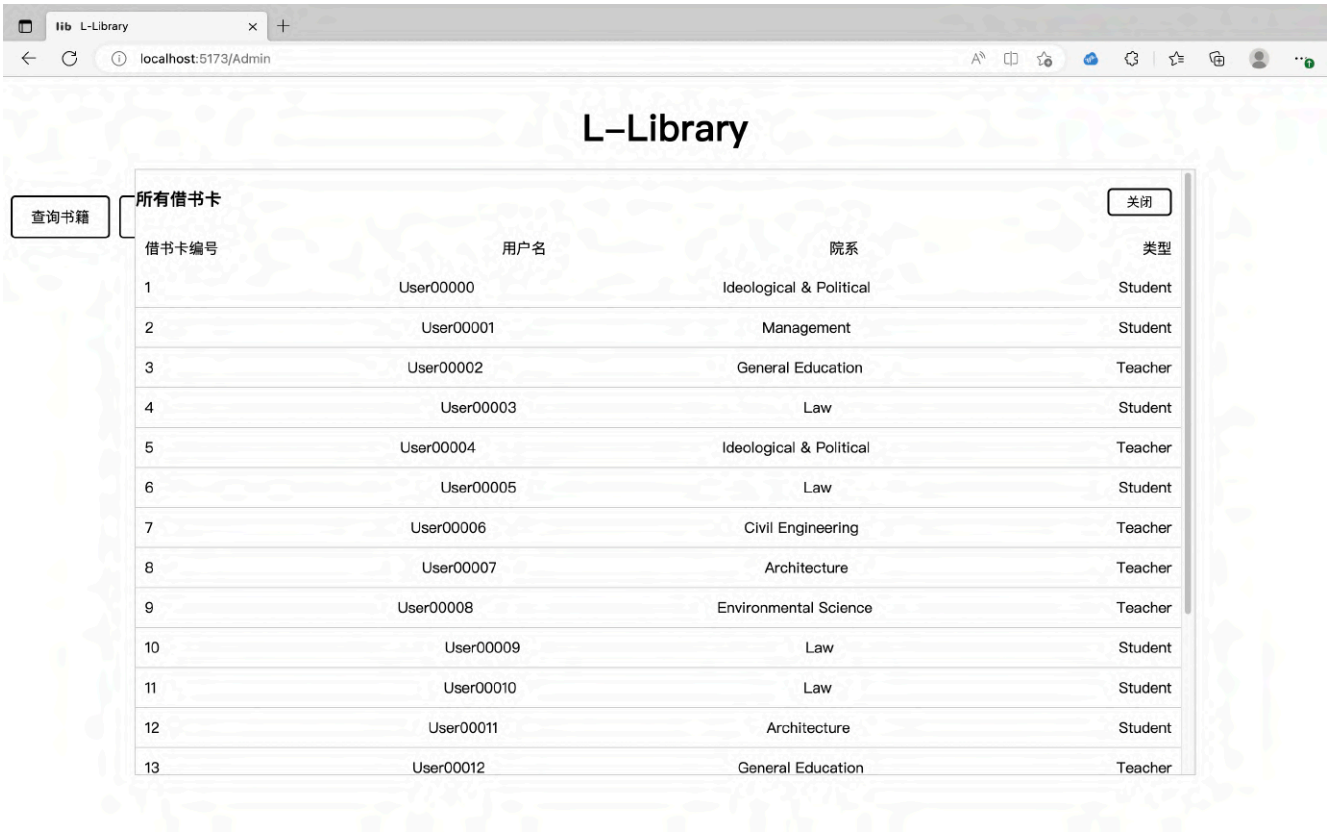
点击查询，确认书本都被正确入库

5. **添加借书证**

进入注册页，输入身份，姓名，部门。点击注册后显示分配的卡号。

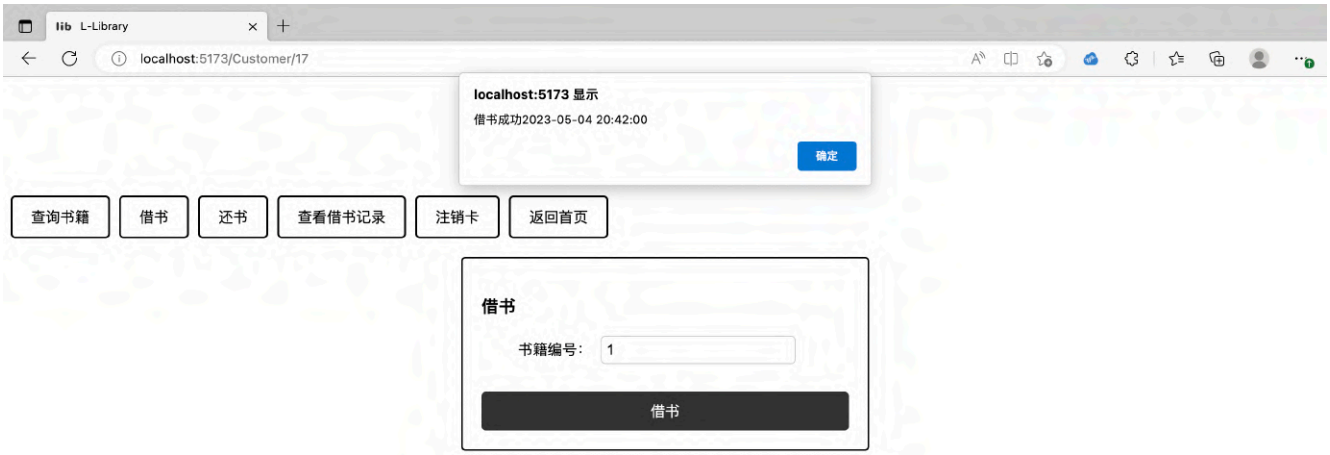
6. **查询借书证**

点击查询所有卡，展示所有借书卡信息

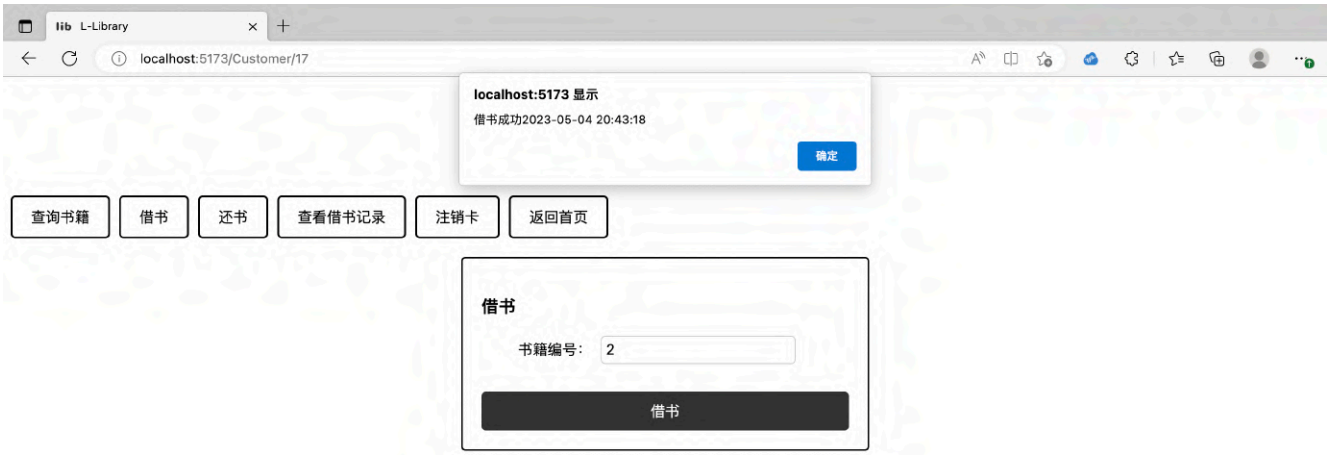


7. **借书**

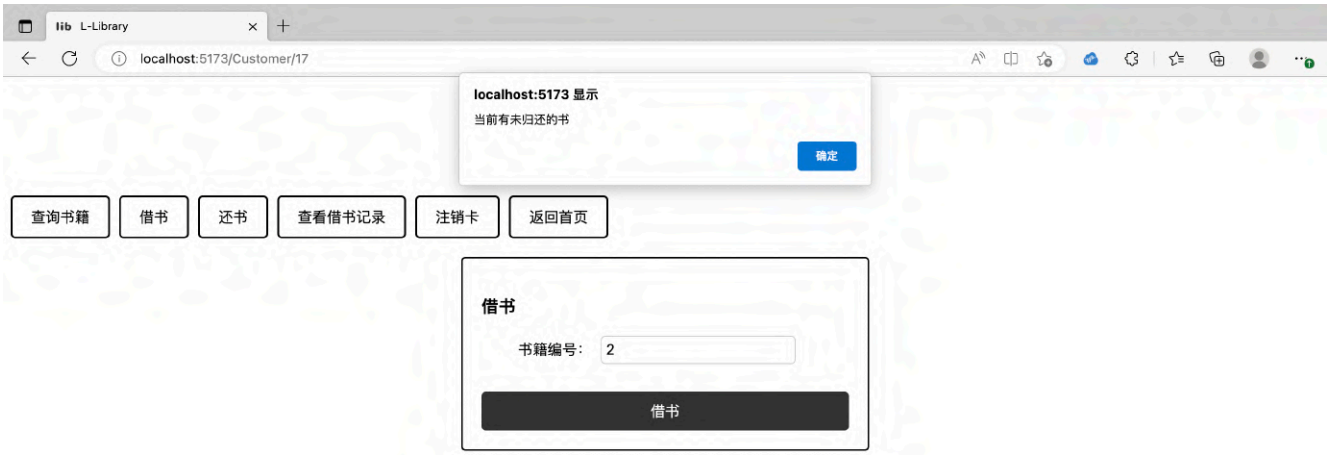
使用编号为17的借书卡借书，输入书籍编号1，点击借书，结束成功



使用编号为17的借书卡借书，输入书籍编号1，点击借书，结束成功

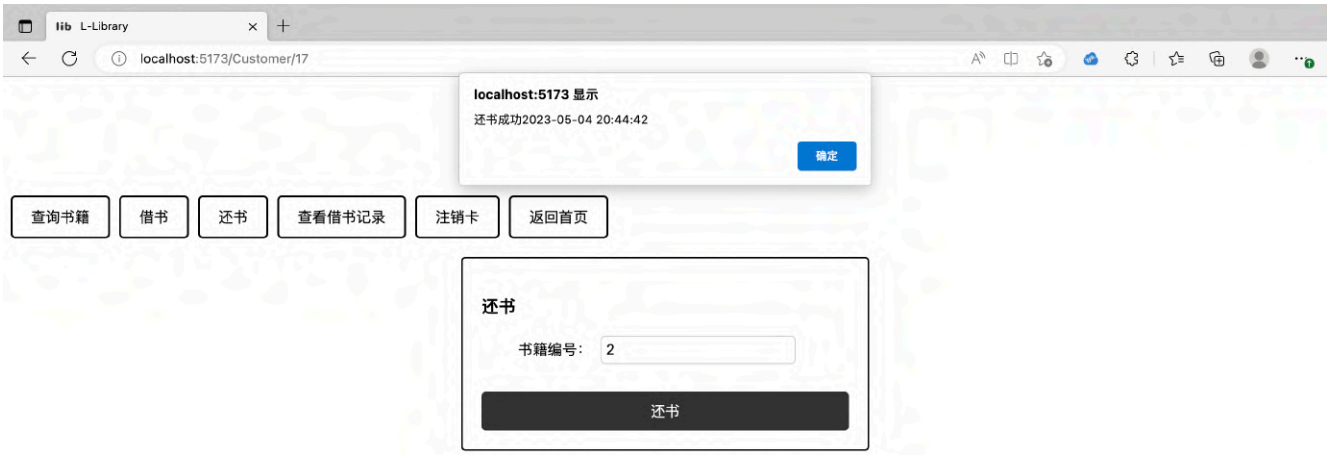


使用编号为17的借书卡借书，输入书籍编号2，点击借书，在同一本书没有归还的情况下重复借书，借书失败



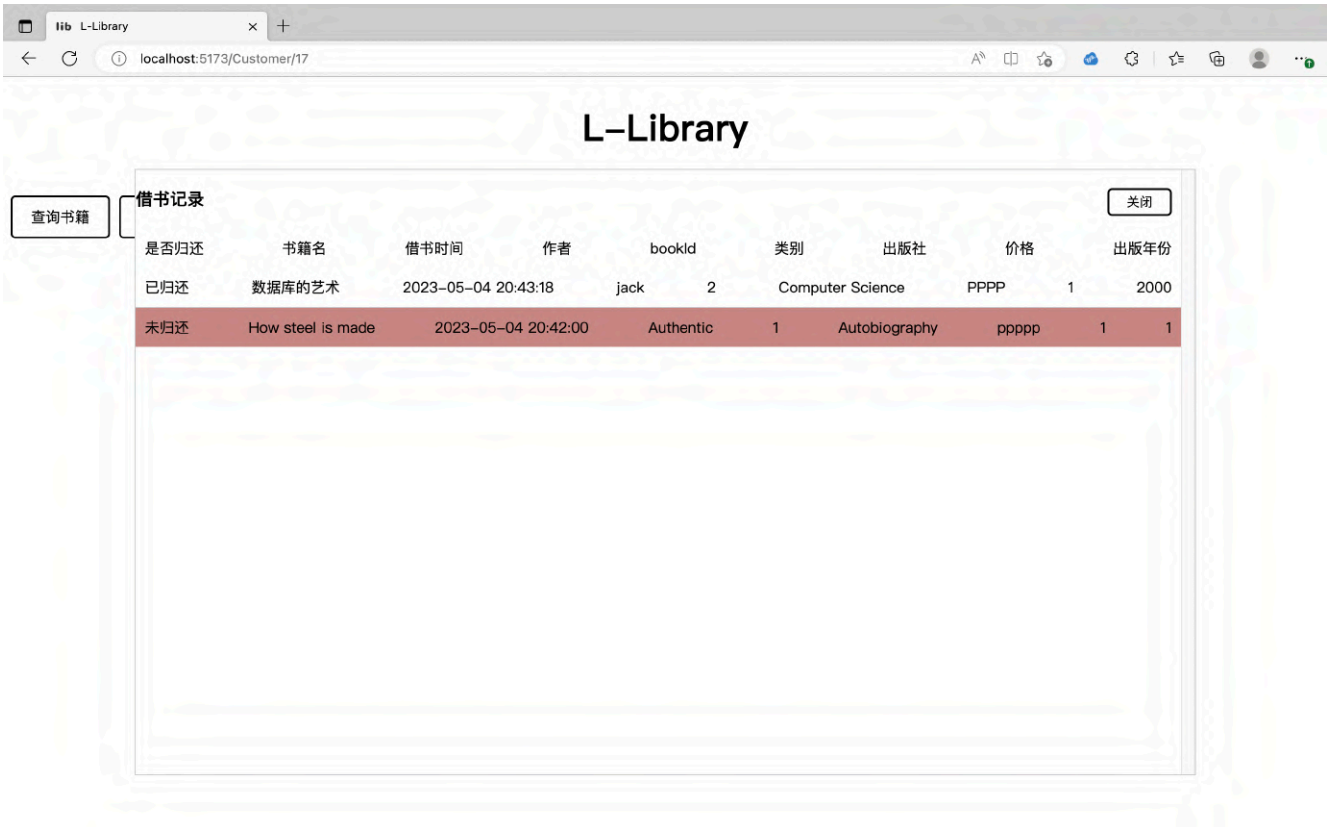
8. **还书**

   用编号17的借书证归还刚才接的编号为2的书



9. **借书记录查询**

   查询编号17的借书证的借书记录



   正确显示了前面对第一本书，第二本书的借阅。未归还的书标红

10. **图书查询** 从查询条件<类别点查(精确查询)，书名点查(模糊查询)，出版社点查(模糊查询)，年份范围查，作者点查(模糊查询)，价格范围差>中随机选取N个条件，并随机选取一个排序列和顺序

    查询的情况比较多，仅举例展示。

首先尝试模糊查询“数据库的艺术”，输入书名“数据库”进行查询：

正确查出结果：

查询结果

关闭

书籍名	作者	bookId	类别	出版社	价格	出版年份	库存
数据库的艺术	jack	2	Computer Science	PPPP	1	2000	1050

然后尝试设定价格和出版年范围，按照价格降序排列：

正确查询到了对应结果

书籍名	作者	bookId	类别	出版社	价格	出版年份	库存
数据库的艺术	jack	2	Computer Science	PPPP	1	2000	1050
Book_xkmna	Author_svr	6	教育	出版社D	10.31	2008	29
Book_lkfec	Author_uny	10	历史	出版社C	23.91	2007	20
Book_doqzt	Author_sub	18	小说	出版社D	14.12	2008	52

数据库操作实现

主要对后端的每个功能实现逻辑进行解释，并指出注意点。

以下所有函数都基于JDBC，都进行了防SQL注入，都用手动commit来保证原子性。

1. `ApiResult storeBook(Book book);`

   此方法存入一本书。首先要检查数据库中是否有各种属性都完全一致的书，如果有要返回对应的错误。然后用JDBC进行插入即可。

2. `ApiResult incBookStock(int bookId, int deltaStock);`

   此方法改变书籍的库存。首先要检查bookId是否有效。然后要检查deltaStock是否有效，即如果原来的库存加上deltaStock如果是负数，需要返回对应的错误。然后用JDBC设置相应的属性即可。

3. `ApiResult storeBook(List<Book> books);`

   此方法插入一系列书。遍历每一本书，设置statement，加入到批处理中，`statement.addBatch();`最后统一commit，一旦有任何错误，全部回滚。

4. `ApiResult removeBook(int bookId);`

   此方法删除本书，首先检查bookId是否有效。然后检查此书是否有未归还的，如果有，返回对应的错误。然后用JDBC进行删除即可。

5. `ApiResult modifyBookInfo(Book book);`

   此方法按照输入的book修改书本的信息。验证bookId之后进行修改即可。

6. `ApiResult queryBook(BookQueryConditions conditions);`

   此方法按照给定的查询条件进行查询，返回查询结果。这里要注意如果condition某个属性为null，则不应该设置对应的查询条件。

7. `ApiResult borrowBook(Borrow borrow);`

   此方法借阅一本书。首先检验bookId是否有效。然后查询此卡是否有此书为归还的书，如果有，返回对应的错误。然后查询此书是否有足够的库存，如果没有，返回对应的错误。接着插入一条借书记录，更新对应的书籍库存即可。

8. `ApiResult returnBook(Borrow borrow);`

   此方法归还一本书。首先检验bookId是否有效。查询是否有未归还的记录，如果没有，返回对应的错误。然后更新借书记录即可。

9. `ApiResult showBorrowHistory(int cardId);`

   此方法对应借书卡的所有借书记录。首先检验bookId是否有效。然后查询此卡所有记录，返回即可。

10. `ApiResult registerCard(Card card);`

    此方法注册一个新借书卡。首先查询是否有出了id各个属性都一样的借书卡。如果有重复的卡，返回对应的错误。

11. `ApiResult removeCard(int cardId);`

    此方法删除一个借书卡。首先检验cardId是否有效。然后查询此卡是否有未归还的书。如果有，返回对应的错误。然后删除对应的卡即可。

12. `ApiResult showCards();`

    此方法返回所有借书卡信息。查询出所有借书卡信息，用数组返回即可。

前端实现

本项目前端使用vue3实现交互页面。这一部分不是实验的重点，只稍微提一下注意点：

1. vueRouter可以很方便的实现页面的导航，注意router导航的时候是可以带参数的。
2. 注意前后端不要写在同一个端口
3. 对所有展示界面都要提供返回或关闭按钮

前端实现

前后端交互

前端

前端使用post请求来向后端提交数据，使用get向后端获得数据。

post请求的例子：下面的axios请求向后端请求修改库存

```
changeStock() {
      // 修改库存的逻辑
      if(!this.stockBookId||!this.deltaStock){
        alert("请将信息填写齐全")
        return
      }
      let data = {
        bookId:this.stockBookId,
        delta:this.deltaStock
      };

      //post要写成data
      axios.post("http://localhost:8050/stock", data)
          .then(response => {
                if(response.data.ok==true){
                  alert("修改成功")
                }else if(response.data.message=="no so many book stock"){
                  alert("库存不足")
                }else if(response.data.message=="book_id not exist"){
                  alert("不存在的编号")
                }else{
                  alert("接口错误");
                }
          })
          .catch(error => {
            console.error(error);
          });
    },
```

get请求的例子：下面的函数向后端请求借书记录

```
async getBorrowRecords() {
      // 发送查看借书记录请求，并将结果保存到borrowHistory数组中
      let params = new URLSearchParams();
      params.append('cardId', this.cardId);
      axios.get("http://localhost:8050/history",{ params: params })
          .then(response=>{
            if(response.data.count==0){
              alert("没有借书记录")
              return
            }
            this.borrowHistory = response.data.items
          })
```

```
            .catch(error=>{
              console.log(error)
            });
    },
```

后端

后端使用SpringBoot框架来进行端口监听和相应。首先使用Bean实例化一个`LibraryManagementSystemImpl`类，然后将这个类自动绑定到每一个控制器类上，这样可以使用同一个实例操作数据库。

然后对每一个api借口监听，下面举一个例子说明：此类监听/addbook的post请求，接收到请求的时候调用`LibraryManagementSystemImpl`实例进行书籍的添加。

```
@RestController
public class AddBookController {
    private final LibraryManagementSystemImpl libraryManagementSystem;
    @Autowired
    public AddBookController(LibraryManagementSystemImpl libraryManagementSystem) {
        this.libraryManagementSystem = libraryManagementSystem;
    }

    @PostMapping("/addbook")
    @ResponseBody
    @CrossOrigin
    public ApiResult borrow(@RequestBody Map<String, Object> requestBody) {
        String title = (String)requestBody.get("title");
        String author = (String) requestBody.get("author");
        String category = (String)requestBody.get("category");
        String press = (String)requestBody.get("press");
        String priceStr = (String)requestBody.get("price");
        double price = Double.parseDouble(priceStr);
        int stock = (int)requestBody.get("stock");
        int publishyear = (int)requestBody.get("publishyear");
        Book newbook = new Book();
        newbook.setAuthor(author);
        newbook.setCategory(category);
        newbook.setPress(press);
        newbook.setPrice(price);
        newbook.setTitle(title);
        newbook.setStock(stock);
        newbook.setPublishYear(publishyear);
        return libraryManagementSystem.storeBook(newbook);
    }
}
```

额外功能和小结

在交互上，将角色分成了师生和管理员，更加符合实际，提高可用性。

在后端实现上，对各种错误进行了大量检测（如额外实现的检查卡是否存在的函数），可以让前端精确反馈错误，提高使用体验。

本项目是我第一次编写后端程序，也算是当了一次全栈工程师。自己创建一个系统工程的体验非常棒。当然中间也遇到了很多问题，常常在深夜独自崩溃。最大的收获是学会了用JDBC操作数据库，让数据库成为了我可以使用的工具。