《数据库系统》
实验报告

姓名: 刘韬

学院: 竺可桢学院

专业: 人工智能

邮箱: 3220103422@zju.edu.cn

日期: 2024 年 4 月 25 日

图书管理系统

图书管理系统

1 实验目的

基于 mysql 设计并实现一个精简的图书管理程序，要求具有图书入库、查询、借书、还书、借书证管理等功能。

使用提供的前端框架，正确完成图书管理系统的前端页面，使其成为一个用户能真正使用的图书管理系统。

2 系统需求

2.1 基本数据对象

对象名称	类名	包含属性
书	Book	书号, 类别, 书名, 出版社, 年份, 作者, 价格, 剩余库存
借书证	Card	卡号, 姓名, 单位, 身份(教师或学生)
借书记录	Borrow	卡号, 书号, 借书日期, 还书日期

数据库表设计如下:

```
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`)
);

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') )
);

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
);
```

2.2 基本功能模块

- `ApiResult storeBook(Book book)`：图书入库模块。向图书库中注册(添加)一本新书，并返回新书的书号。如果该书已经存在于图书库中，那么入库操作将失败。当且仅当书的`<类别，书名，出版社，年份，作者>`均相同时，才认为两本书相同。请注意，book_id 作为自增列，应该插入时由数据库生成。插入完成后，需要根据数据库生成的 book_id 值去更新 book 对象里的 book_id。
- `ApiResult incBookStock(int bookId, int deltaStock)`：图书增加库存模块。为图书库中的某一本书增加库存。其中库存增量 deltaStock 可正可负，若为负数，则需要保证最终库存是一个非负数。
- `ApiResult storeBook(List<Book> books)`：图书批量入库模块。批量入库图书，如果有一本书入库失败，那么就需要回滚整个事务(即所有的书都不能被入库)。
- `ApiResult removeBook(int bookId)`：图书删除模块。从图书库中删除一本书。如果还有人尚未归还这本书，那么删除操作将失败。
- `ApiResult modifyBookInfo(Book book)`：图书修改模块。修改已入库图书的基本信息，该接口不能修改图书的书号和存量。
- `ApiResult queryBook(BookQueryConditions conditions)`：图书查询模块。根据提供的查询条件查询符合条件的图书，并按照指定排序方式排序。查询条件包括：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。如果两条记录排序条件的值相等，则按 book_id 升序排序。
- `ApiResult borrowBook(Borrow borrow)`：借书模块。根据给定的书号、卡号和借书时间添加一条借书记录，然后更新库存。若用户此前已经借过这本书但尚未归还，那么借书操作将失败。
- `ApiResult returnBook(Borrow borrow)`：还书模块。根据给定的书号、卡号和还书时间，查询对应的借书记录，并补充归还时间，然后更新库存。
- `ApiResult showBorrowHistory(int cardId)`：借书记录查询模块。查询某个用户的借书记录，按照借书时间递减、书号递增的方式排序。
- `ApiResult registerCard(Card card)`：借书证注册模块。注册一个借书证，若借书证已经存在，则该操作将失败。当且仅当`<姓名，单位，身份>`均相同时，才认为两张借书证相同。
- `ApiResult removeCard(int cardId)`：删除借书证模块。如果该借书证还有未归还的图书，那么删除操作将失败。
- `ApiResult showCards()`：借书证查询模块。列出所有的借书证。

除了以上框架内给出的声明接口外，根据前端实现的需要还添加了以下接口:

- `ApiResult modifyCardInfo(Card card);`：修改借书证信息模块。修改已注册的借书证的基本信息，该接口不能修改借书证的卡号。

3 实验环境

数据库平台: MySQL 8.3.0

JAVA 版本: openjdk version "17.0.10" 2024-01-16

4 系统设计及实现

4.1 基本功能模块实现

首先是图书管理系统接口函数的思路和实现，以下分接口进行说明:

4.1.1 `storeBook(Book book)`

`storeBook`函数用于图书入库，首先需要查询数据库中是否已经存在该书，如果存在则返回失败，否则插入新书。使用 PreparedStatement 来避免注入攻击，首先查询数据库中是否已经存在该书，如果存在则返回失败，否则插入新书。插入完成后，需要根据数据库生成的 book_id 值去更新 book 对象里的 book_id。注意由于 unique 属性`<类别，书名，出版社，年份，作者>`，所以查询只需设置这几个属性即可。

```
    public ApiResult storeBook(Book book) {
        Connection conn = connector.getConn();//connect to database
        try{
            //use preparedstatement to avoid injection attack
            String query_sql = "SELECT book_id from book where category = ? and title = ? and press = ? and publish_year = ? and author = ?;";
            PreparedStatement pstmtquery = conn.prepareStatement(query_sql);
            String insert_sql = "Insert into book(category, title, press, publish_year, author, price, stock) values(?, ?, ?, ?, ?, ?, ?);";
            PreparedStatement pstmtinsert = conn.prepareStatement(insert_sql);
            //set query condition,
            pstmtquery.setString(1, book.getCategory());
            pstmtquery.setString(2, book.getTitle());
            pstmtquery.setString(3, book.getPress());
            pstmtquery.setInt(4, book.getPublishYear());
            pstmtquery.setString(5, book.getAuthor());

            ResultSet rs = pstmtquery.executeQuery();

            if(rs.next()){//the book already exists
                return new ApiResult(false, "Book already exists");
            }
            //perform insert operation
            pstmtinsert.setString(1, book.getCategory());
            pstmtinsert.setString(2, book.getTitle());
            pstmtinsert.setString(3, book.getPress());
            pstmtinsert.setInt(4, book.getPublishYear());
```

```
            pstmtinsert.setString(5, book.getAuthor());
            pstmtinsert.setDouble(6, book.getPrice());
            pstmtinsert.setInt(7, book.getStock());
            pstmtinsert.executeUpdate();

            rs = pstmtquery.executeQuery();

            if(rs.next()){
                book.setBookId(rs.getInt("book_id"));
            }
            pstmtinsert.close();
            pstmtquery.close();
            commit(conn);
        }catch(Exception e){
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
        return new ApiResult(true, "Store successfully");
    }
```

4.1.2 `incBookStock(int bookId, int deltaStock)`

`incBookStock`函数用于图书增加库存，首先根据`book_id`查询数据库中是否存在该书，如果存在则更新库存，否则返回失败。注意最终的库存量应该是一个非负数，如果更新后库存小于 0，则需要回滚操作，并且返回失败。

```
    public ApiResult incBookStock(int bookId, int deltaStock) {
        Connection conn = connector.getConn();
        try{
            String sql = "SELECT stock from book where book_id = ? ;";
            PreparedStatement pstmt = conn.prepareStatement(sql);
            pstmt.setInt(1, bookId);
            ResultSet rs = pstmt.executeQuery();
            if(rs.next()){
                int stock = rs.getInt("stock");
                stock += deltaStock;
                if(stock < 0){
                    rollback(conn);
                    return new ApiResult(false, "Stock can't be negative!");
                }
                sql = "UPDATE book SET stock = ? where book_id = ? ;";
                pstmt = conn.prepareStatement(sql);
                pstmt.setInt(1, stock);
                pstmt.setInt(2, bookId);
                pstmt.executeUpdate();
                pstmt.close();
                commit(conn);
                return new ApiResult(true, "Stock updated successfully");
            }else{
                rollback(conn);
                return new ApiResult(false, "Book not found");
            }
        }catch(Exception e){
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
    }
```

4.1.3 `storeBook(List<Book> books)`

`storeBook`函数用于图书批量入库，对列表`Books`，我们遍历其中每一个`book`，每次先查询是否有这本书的存在，如果有的话就要回滚报错，如果没有就是合法情况，使用`addBatch()`来记录插入操作，直到所有的书籍都完成检查确认可以入库后一起执行`Bacth`指令。随后查询自动生成的`book_id`，并添加到`book`属性中，一切操作完成后`commit`然后返回`true`。

```
    public ApiResult storeBook(List<Book> books) {
        Connection conn = connector.getConn();
        try{
            String query_sql = "SELECT book_id from book where category = ? and title = ? and press = ? and publish_year = ? and author = ?;";
            PreparedStatement pstmtquery = conn.prepareStatement(query_sql);
            String insert_sql = "Insert into book(category, title, press, publish_year, author, price, stock) values(?, ?, ?, ?, ?, ?, ?);";
            PreparedStatement pstmtinsert = conn.prepareStatement(insert_sql);
            for(Book book : books){
                pstmtquery.setString(1, book.getCategory());
                pstmtquery.setString(2, book.getTitle());
                pstmtquery.setString(3, book.getPress());
                pstmtquery.setInt(4, book.getPublishYear());
                pstmtquery.setString(5, book.getAuthor());
                ResultSet rs = pstmtquery.executeQuery();
                if(rs.next()){
                    rollback(conn);
                    return new ApiResult(false, "Some books have already exits");
                }
                pstmtinsert.setString(1, book.getCategory());
                pstmtinsert.setString(2, book.getTitle());
                pstmtinsert.setString(3, book.getPress());
                pstmtinsert.setInt(4, book.getPublishYear());
                pstmtinsert.setString(5, book.getAuthor());
                pstmtinsert.setDouble(6, book.getPrice());
                pstmtinsert.setInt(7, book.getStock());
                pstmtinsert.addBatch();
            }
            pstmtinsert.executeBatch();
            for(Book book: books){
                pstmtquery.setString(1, book.getCategory());
                pstmtquery.setString(2, book.getTitle());
                pstmtquery.setString(3, book.getPress());
                pstmtquery.setInt(4, book.getPublishYear());
                pstmtquery.setString(5, book.getAuthor());
                ResultSet rs = pstmtquery.executeQuery();
                if(rs.next()){
                    book.setBookId(rs.getInt("book_id"));
                }
            }
            pstmtinsert.close();
            pstmtquery.close();
            commit(conn);
        }
        catch(Exception e){
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
        return new ApiResult(true, "Store successfully");
```

```
49      }
50
```

4.1.4 `removeBook(int bookId)`

`removeBook`函数用于删除库中书籍，参数为`bookId`。在删除之前要做检查，从表`borrow`中检查这本书是否还有未归还的情况，如果有那么回滚返回错误。如果没有未还的书籍就执行删除。注意有可能这本书是不存在的，因此利用执行删除语句的返回值，查看删除是否成功。如果返回值为 0 说明失败，就回滚操作并且报告错误。如果成功就`commit`，返回`true`。

```
1     public ApiResult removeBook(int bookId) {
2         Connection conn = connector.getConn();
3         try{
4             String sql = "select * from Borrow where book_id = ? and
  return_time = 0";
5             PreparedStatement pstmt = conn.prepareStatement(sql);
6             pstmt.setInt(1, bookId);
7             ResultSet rs = pstmt.executeQuery();
8             if(rs.next()){
9                 rollback(conn);
10                return new ApiResult(false, "Book is borrowed and not
  returned");
11            }
12            sql = "DELETE from book where book_id = ? ;";
13            pstmt = conn.prepareStatement(sql);
14            pstmt.setInt(1, bookId);
15            int res = pstmt.executeUpdate();
16            if(res == 0){
17                rollback(conn);
18                return new ApiResult(false, "Book not found");
19            }
20            commit(conn);
21        }catch(Exception e){
22            rollback(conn);
23            return new ApiResult(false, e.getMessage());
24        }
25        return new ApiResult(true, "Book removed successfully");
26    }
27
```

4.1.5 `modifyBookInfo(Book book)`

`modifyBookInfo`函数用于修改图书信息，但是不能修改`book_id`和`stock`。`stock`的修改由函数`incBookStock`实现。首先查询数据库中是否存在这本书，如果不存在就返回错误。如果存在就执行更新操作，注意更新的时候不能修改`book_id`和`stock`，其他属性可以修改。最后`commit`返回`true`。

```
1     public ApiResult modifyBookInfo(Book book) {
2         //return new ApiResult(false, "Unimplemented Function");
3         Connection conn = connector.getConn();
4         try{
```

```
            String sql = "update book set category = ?, title = ?, press = ?,
publish_year = ?, author = ?, price = ? where book_id = ?;";
            PreparedStatement pstmt = conn.prepareStatement(sql);

            pstmt.setString(1, book.getCategory());
            pstmt.setString(2, book.getTitle());
            pstmt.setString(3, book.getPress());
            pstmt.setInt(4, book.getPublishYear());
            pstmt.setString(5, book.getAuthor());
            pstmt.setDouble(6, book.getPrice());
            pstmt.setInt(7, book.getBookId());

            pstmt.executeUpdate();
            commit(conn);
        }catch(Exception e){
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
        return new ApiResult(true, "modify information successfully!");
    }
```

4.1.6 `queryBook(BookQueryConditions conditions)`

`queryBook`函数用于查询图书，参数为`BookQueryConditions`，根据条件查询图书。首先构造查询语句，然后使用`PreparedStatement`来避免注入攻击。查询参数有类别、书名、出版社、年份、作者、价格，其中类别、书名、出版社、作者是模糊查询，年份和价格是范围查询。查询完成后根据排序方式排序，最后返回结果。查询条件字符串条件都是包含查询，即`like`，所以在设置参数时要在前后加上`%`。对于空查询条件就设置为`%`，即任意。排序方式有升序和降序，根据`SortOrder`来判断。最后返回`ApiResult`的`payload`是包含所有查询结果的`List<Book>`。

```
    public ApiResult queryBook(BookQueryConditions conditions) {
        Connection conn = connector.getConn();
        List<Book> results = new ArrayList<>();
        try{
            String sql = "select * from book where category like ? and title
like ? and press like ? and publish_year >= ? and publish_year <= ? and author
like ? and price >= ? and price <= ?;";
            PreparedStatement pstmt = conn.prepareStatement(sql);

            pstmt.setString(1, conditions.getCategory() == null ? "%" : "%" +
conditions.getCategory() +"%");
            pstmt.setString(2, conditions.getTitle() == null ? "%" : "%" +
conditions.getTitle() + "%");
            pstmt.setString(3, conditions.getPress() == null ? "%" : "%" +
conditions.getPress() + "%");
            pstmt.setInt(4, conditions.getMinPublishYear() == null ?
Integer.MIN_VALUE : conditions.getMinPublishYear());
            pstmt.setInt(5, conditions.getMaxPublishYear() == null ?
Integer.MAX_VALUE : conditions.getMaxPublishYear());
            pstmt.setString(6, conditions.getAuthor() == null ? "%" : "%" +
conditions.getAuthor() + "%");
            pstmt.setDouble(7, conditions.getMinPrice() == null ?
Double.MIN_VALUE : conditions.getMinPrice());
```

```
            pstmt.setDouble(8, conditions.getMaxPrice() == null ?
Double.MAX_VALUE : conditions.getMaxPrice());

            ResultSet rs = pstmt.executeQuery();
            while(rs.next()){
                Book book = new Book();
                book.setBookId(rs.getInt("book_id"));
                book.setCategory(rs.getString("category"));
                book.setTitle(rs.getString("title"));
                book.setPress(rs.getString("press"));
                book.setPublishYear(rs.getInt("publish_year"));
                book.setAuthor(rs.getString("author"));
                book.setPrice(rs.getDouble("price"));
                book.setStock(rs.getInt("stock"));
                results.add(book);
            }

            if(conditions.getSortOrder() ==  SortOrder.ASC){
                results.sort(conditions.getSortBy().getComparator());
            }
            else{

results.sort(conditions.getSortBy().getComparator().reversed());
            }
            pstmt.close();
            commit(conn);
        }catch(Exception e){
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
        BookQueryResults bookQueryResults = new BookQueryResults(results);
        return new ApiResult(true, bookQueryResults);
    }
```

4.1.7 `borrowBook(Borrow borrow)`

`borrowBook`函数用于借书，参数为`Borrow`，根据给定的书号、卡号和借书时间添加一条借书记录，然后更新库存。首先查询数据库中是否有这本书，如果没有就返回错误。然后检查这本书的库存，如果库存小于 1 就返回错误。接着查询是否这个人已经借过这本书，如果借过就返回错误。最后执行借书操作，更新库存，插入借书记录，最后`commit`返回`true`。

```
    public ApiResult borrowBook(Borrow borrow) {
        Connection conn = connector.getConn();
        try {
            String query_sql = "select * from book where book_id = ? for
update;";
            PreparedStatement querlstmt = conn.prepareStatement(query_sql);
            querlstmt.setInt(1, borrow.getBookId());
            ResultSet rs = querlstmt.executeQuery();
            // check the stock
            if(rs.next()){
                if(rs.getInt("stock") < 1){
                    rollback(conn);
                    return new ApiResult(false, "The book has no stock!");
                }
            }else{
                rollback(conn);
                return new ApiResult(false, "The book doesn't exist!");
```

```
            }
            // check the borrower
            query_sql = "select * from borrow where book_id = ? and card_id = ?
and return_time = 0;";
            querlstmt = conn.prepareStatement(query_sql);
            querlstmt.setInt(1, borrow.getBookId());
            querlstmt.setInt(2, borrow.getCardId());
            rs = querlstmt.executeQuery();
            if(rs.next()){
                rollback(conn);
                return new ApiResult(false, "The borrower hasn't return the
book!");
            }
            // perform borrow
            String sql = "Update book set stock = stock - 1 where book_id
= ?;";
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setInt(1, borrow.getBookId());
            stmt.executeUpdate();
            sql = "Insert into borrow (card_id, book_id, borrow_time)
values(?, ?, ?);";
            stmt = conn.prepareStatement(sql);
            stmt.setInt(1, borrow.getCardId());
            stmt.setInt(2, borrow.getBookId());
            stmt.setLong(3,borrow.getBorrowTime());
            //stmt.setLong(3,borrow.getReturnTime());
            stmt.executeUpdate();
            stmt.close();
            conn.commit();
        }catch(Exception e){
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
        return new ApiResult(true, "Borrow successfully!");
    }

```

4.1.8 `returnBook(Borrow borrow)`

`returnBook`函数用于还书，参数为`Borrow`，根据给定的书号、卡号和还书时间，查询对应的借书记录，并补充归还时间，然后更新库存。首先查询数据库中是否有这本书，如果没有就返回错误。然后检查输入的还书时间和库内的借书时间，如果还书时间早于借书时间，返回错误。如果一切都合法，就更新借书记录，更新还书时间和库存，最后`commit`返回`true`。

```
public ApiResult returnBook(Borrow borrow) {
    //return new ApiResult(false, "Unimplemented Function");
    Connection conn = connector.getConn();
    try{
        String sql = "select * from borrow where card_id = ? and book_id = ?
and return_time = 0;";
        PreparedStatement qstmt = conn.prepareStatement(sql);
        qstmt.setInt(1, borrow.getCardId());
        qstmt.setInt(2, borrow.getBookId());
        ResultSet rs = qstmt.executeQuery();
        if(!rs.next()){
            rollback(conn);
            return new ApiResult(false, "The borrow doesn't exist!");
        }
```

```
        Long borrowTime = rs.getLong("borrow_time");
        if(borrowTime >= borrow.getReturnTime()){
            rollback(conn);
            return new ApiResult(false, "The return time is earlier than the
borrow time!");
        }
        sql = "Update borrow set return_time = ? where card_id = ? and book_id
= ? and borrow_time = ?;";
        PreparedStatement stmt = conn.prepareStatement(sql);
        stmt.setLong(1, borrow.getReturnTime());
        stmt.setInt(2, borrow.getCardId());
        stmt.setInt(3, borrow.getBookId());
        stmt.setLong(4, borrowTime);
        int re = stmt.executeUpdate();
        if( re == 0 ){
            rollback(conn);
            return new ApiResult(false, "There is not the borrow!");
        }
        sql = "Update book set stock = stock + 1 where book_id = ?;";
        stmt = conn.prepareStatement(sql);
        stmt.setInt(1, borrow.getBookId());
        stmt.executeUpdate();
        stmt.close();
        commit(conn);
    }catch(Exception e){
        rollback(conn);
        return new ApiResult(false, e.getMessage());
    }
    return new ApiResult(true, "return successfully!");
}
```

4.1.9 `showBorrowHistory(int cardId)`

`showBorrowHistory`函数用于查询借书记录，参数为`cardId`，查询某个用户的借书记录，按照借书时间递减、书号递增的方式排序。首先查询数据库中是否有这个用户，如果没有就返回错误。然后查询这个用户的借书记录，按照要求排序，最后返回`ApiResult`的`payload`是包含所有查询结果的`List<BorrowHistories.Item>`。

```
    public ApiResult showBorrowHistory(int cardId) {
        Connection conn = connector.getConn();
        List<BorrowHistories.Item> items = new ArrayList<>();
        try{
            String sql = "select * from borrow natural join book where card_id
= ? order by borrow_time DESC, book_id ASC;";
            PreparedStatement pstmt = conn.prepareStatement(sql);
            pstmt.setInt(1, cardId);
            ResultSet rs = pstmt.executeQuery();
            while(rs.next())
            {
                Book book = new Book();
                book.setBookId(rs.getInt("book_id"));
                book.setCategory(rs.getString("category"));
                book.setTitle(rs.getString("title"));
                book.setPress(rs.getString("press"));
                book.setPublishYear(rs.getInt("publish_year"));
                book.setAuthor(rs.getString("author"));
                book.setPrice(rs.getDouble("price"));
                book.setStock(rs.getInt("stock"));
```

```
                Borrow borrow = new Borrow();
                borrow.setBookId(rs.getInt("book_id"));
                borrow.setCardId(rs.getInt("card_id"));
                borrow.setBorrowTime(rs.getLong("borrow_time"));
                borrow.setReturnTime(rs.getLong("return_time"));
                Item item = new Item(cardId, book, borrow);
                items.add(item);
            }
            pstmt.close();
            commit(conn);
        }catch(Exception e){
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
        return new ApiResult(true, new BorrowHistories(items));
    }
```

4.1.10 `registerCard(Card card)`

`registerCard`函数用于注册借书证，参数为`Card`，注册一个借书证，若借书证已经存在，即所有信息都相同，则该操作将失败。当且仅当`<姓名，单位，身份>`均相同时，才认为两张借书证相同。首先查询数据库中是否有这个借书证，如果有就返回错误。然后插入新的借书证，插入借书证后会得到一个自动生成的`card_id`，将`card_id`写回`card`变量中，最后`commit`返回`true`。

```
    public ApiResult registerCard(Card card) {
        Connection conn = connector.getConn();
        try{
            String sql = "select card_id from card where name = ? and
 department = ? and type = ?;";
            PreparedStatement pstmtquery = conn.prepareStatement(sql);
            pstmtquery.setString(1, card.getName());
            pstmtquery.setString(2, card.getDepartment());
            pstmtquery.setString(3, card.getType().getStr());
            ResultSet rs = pstmtquery.executeQuery();
            if(rs.next()){
                rollback(conn);
                return new ApiResult(false, "There exists a card!");
            }
            sql = "insert into card (name, department, type) values
 (?, ?, ?);";
            PreparedStatement pstmtinsert = conn.prepareStatement(sql);
            pstmtinsert = conn.prepareStatement(sql);
            pstmtinsert.setString(1, card.getName());
            pstmtinsert.setString(2, card.getDepartment());
            pstmtinsert.setString(3, card.getType().getStr());
            pstmtinsert.executeUpdate();
            rs = pstmtquery.executeQuery();
            if(rs.next())
            {
                card.setCardId(rs.getInt("card_id"));
            }
            pstmtinsert.close();
            pstmtquery.close();
            commit(conn);
        }catch(Exception e){
            rollback(conn);
            return new ApiResult(false, e.getMessage());
```

```
        }
        return new ApiResult(true, "register successfully!");
    }
```

4.1.11 `removeCard(int cardId)`

`removeCard` 函数用于删除借书证，参数为 `cardId`，如果该借书证还有未归还的图书，那么删除操作将失败。首先查询数据库中是否有这个借书证，如果没有就返回错误。然后检查这个借书证是否有未归还的书籍，如果有就返回错误。最后执行删除操作，如果删除失败就返回错误，如果成功就 `commit` 返回 `true`。在实现上，我选择先查询是否有未归还书籍，然后直接进行删除操作，如果删除操作的返回值是 0，说明没有这个借书证，返回错误。这样可以省去一次查询的操作，提高了效率。

```
    public ApiResult removeCard(int cardId) {
        //return new ApiResult(false, "Unimplemented Function");
        Connection conn = connector.getConn();

        try{
            // check if exists unreturned book
            String sql = "Select * from borrow where card_id = ? and
 return_time = 0;";
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setInt(1, cardId);
            ResultSet rs = stmt.executeQuery();
            if(rs.next()){
                rollback(conn);
                return new ApiResult(false, "There are unreturned books!");
            }
            // delete the card
            sql = "delete from card where card_id = ?;";
            stmt = conn.prepareStatement(sql);
            stmt.setInt(1, cardId);
            int re = stmt.executeUpdate();
            if(re == 0){
                rollback(conn);
                return new ApiResult(false, "The card doesn't exist!");
            }
            stmt.close();
            commit(conn);
        }catch(Exception e){
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
        return new ApiResult(true, "Remove successfully!");

    }
```

4.1.12 `showCards()`

`showCards`函数用于查询借书证，列出所有的借书证。首先以借书证升序查询数据库中所有的借书证，然后返回所有的借书证。最后返回`ApiResult`的`payload`是包含所有查询结果的`List<Card>`。

```
    public ApiResult showCards() {
        Connection conn = connector.getConn();
        List<Card> cards = new ArrayList<>();
        try{
            String sql = "select * from card order by card_id ASC;";
            PreparedStatement statement = conn.prepareStatement(sql);
            ResultSet rs = statement.executeQuery();
            while(rs.next())
            {
                Card temp = new Card();
                temp.setCardId(rs.getInt("card_id"));
                temp.setName(rs.getString("name"));
                temp.setDepartment(rs.getString("department"));
                temp.setType(Card.CardType.values(rs.getString("type")));
                cards.add(temp);
            }
            statement.close();
            commit(conn);
        }catch(Exception e){
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
        return new ApiResult(true, new CardList(cards));
    }
```

4.1.13 测试情况

在完成以上代码后，使用框架提供的测试程序进行测试，结果如下:

图 1: 测试结果

基本功能模块测试全部通过，完成基本的设计要求。

4.2 前端交互及功能模块设计

根据给定的框架，我们分三部分实现图书管理系统的交互和功能模块设计。

使用 Vue3 和 Element-Plus 实现了图书管理系统的前端页面，实现了图书入库、查询、借书、还书、借书证管理等功能。

使用 java 实现了图书管理系统的后端接口，接收来自前端的请求，并给出相应的响应。以下给出后端的框架:

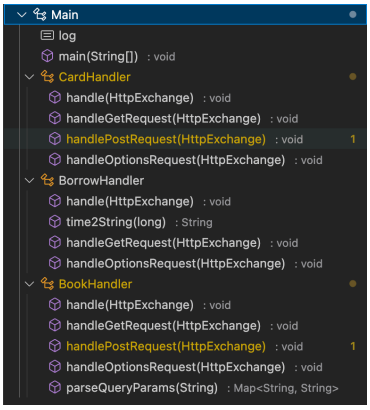

图 2: 后端处理框架

我们在`Main`类中实现了数据库的连接，监听端口的绑定，并为"/card"、"/borrow" 和 “/book” 路径创建了对应的处理器，这些处理器将在实现处理前端请求，并且调用基本功能模块中实现的函数与数据库进行交互，然后返回相应信息到前端。上面给出后端`main`框架，我们创建了三个处理器`CardHandler`、`BorrowHandler`和`BookHandler`，分别处理借书证、借书和图书的请求。每个处理器都处理`GET`和`POST`请求，根据具体请求的方法调用相应的函数，然后返回相应的信息。注意`OPTION`请求是预检请求，用于检查是否可以发送真正的请求，我们在处理器中也处理了这个请求。

```
    public static void main(String[] args) {
        try {
            // parse connection config from "resources/application.yaml"
            ConnectConfig conf = new ConnectConfig();
            log.info("Success to parse connect config. " + conf.toString());
            // connect to database
            DatabaseConnector connector = new DatabaseConnector(conf);
            boolean connStatus = connector.connect();
            if (!connStatus) {
                log.severe("Failed to connect database.");
                System.exit(1);
            }
            /* do somethings */
            //监听 8000 端口
            HttpServer server = HttpServer.create(new InetSocketAddress(8000),
0);
            //创建处理器
            server.createContext("/card", new CardHandler());
            server.createContext("/borrow", new BorrowHandler());
            server.createContext("/book", new BookHandler());
            //server.createContext("/upload", new UploadHandler());
            server.start();
            System.out.println("Server is listening on port 8000");
            // release database connection handler
            if (connector.release()) {
                log.info("Success to release connection.");
```

```
            } else {
                log.warning("Failed to release connection.");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

每个处理器的结构都是类似的，我们以`BookHandler`为例，给出其实现:

```
    static class BookHandler implements HttpHandler {
        @Override
        public void handle(HttpExchange exchange) throws IOException {
            // 允许所有域的请求，cors 处理
            Headers headers = exchange.getResponseHeaders();
            headers.add("Access-Control-Allow-Origin", "*");
            headers.add("Access-Control-Allow-Methods", "GET, POST");
            headers.add("Access-Control-Allow-Headers", "Content-Type");
            // 解析请求的方法，看 GET 还是 POST
            String requestMethod = exchange.getRequestMethod();
            if (requestMethod.equals("GET")) {
                // 处理 GET
                handleGetRequest(exchange);
            } else if (requestMethod.equals("POST")) {
                // 处理 POST
                handlePostRequest(exchange);
            } else if (requestMethod.equals("OPTIONS")) {
                // 处理 OPTIONS
                handleOptionsRequest(exchange);
            } else {
                // 其他请求返回 405 Method Not Allowed
                exchange.sendResponseHeaders(405, -1);
            }
        }
    }
```

下面启动前后端，展示相应功能。以下是启动命令:

```
$ mvn exec:java -Dexec.mainClass="Main" -Dexec.cleanupDaemonThreads=false
$ cd librarymanagementsystem-frontend
$ npm run dev
```

4.2.1 Book

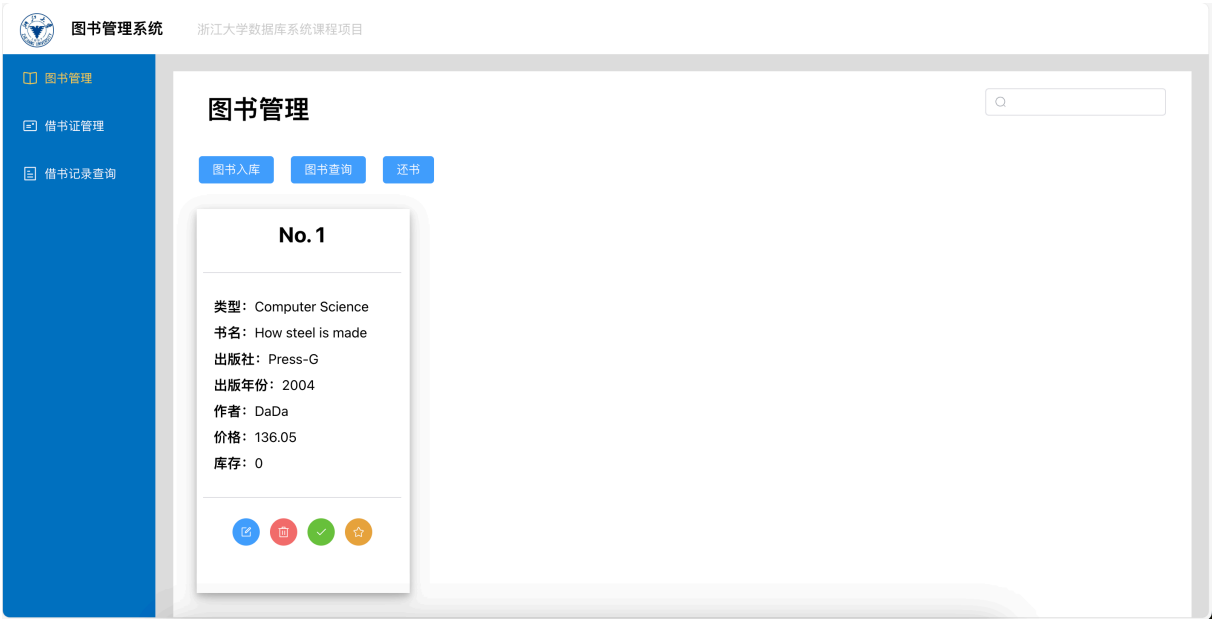

图 3: Book 页面概览

以上是实现的前端页面 `Book`，包括图书入库、查询、还书按钮。显示的卡片是图书信息，包含图书 id，类别，书名，出版社，年份，作者，价格，库存，卡片上的四个按钮分别是对图书信息修改，删除图书，修改库存，借书。这里的卡片信息来源于变量 `books`，我们后续的增删改查的结果都存储在这之中。这是 `Book` 页面的全部静态元素，实现代码如下:

所有元素都被包含在 `<template>` 标签中，下面代码用到了 `el-element` 的组件 `el-button`，设计了所见的三个按钮，分别将其需要用到的信息关联到相应变量中，按下按钮对应的 `@click` 绑定了函数，并且触发对话框弹出。

```
    <!-- 标题和搜索框 -->
        <div style="margin-top: 20px; margin-left: 40px; font-size: 2em; font-
weight: bold; ">图书管理
            <el-input v-model="toSearch" :prefix-icon="Search"
                style=" width: 15vw;min-width: 150px; margin-left: 30px;
margin-right: 30px; float: right;" clearable />
        </div>
    <!-- 按钮设计 -->
        <div style="width:50%;margin:10px ; padding-top:3vh;">
        <el-button style="margin-left: 20px;" type="primary"
        @click="newBookInfo.Category='',
        newBookInfo.Title='',newBookInfo.Press='',newBookInfo.Publishyear='',
        newBookInfo.Author='',newBookInfo.Price='',newBookInfo.Number=''
        ;newBookVisible = true">图书入库
        </el-button>
        <el-button style="margin-left: 20px;" type="primary"
        @click="queryBookInfo.Category='',
        queryBookInfo.Title='',queryBookInfo.Press='',queryBookInfo.
        MinPublishyear='',
        queryBookInfo.MaxPublishyear='',queryBookInfo.Author='',queryBookInfo.
        MinPrice='',
        queryBookInfo.MaxPrice='',queryBookVisible=true">图书查询
        </el-button>
        <el-button style="margin-left: 20px;" type="primary"
        @click="returnBookVisible =true" >还书
```

```
        </el-button>
    </div>

```

以下是所有用到的变量，在后面的实现描述中就不再赘述。

```
    data() {
        return {
            Delete,
            Edit,
            Search,
            Check,
            Star,
            toSearch: '', // 搜索内容
            books: [{
                id : 1,
                category: '计算机',
                title: '计算机网络',
                press: '清华大学出版社',
                publishyear: '2021',
                author: '谢希仁',
                price: '50',
                stock: '100'
            }], // 图书列表
            newBookVisible: false, // 新建图书对话框可见性
            newCardVisible: false, // 新建借书证对话框可见性
            removeCardVisible: false, // 删除借书证对话框可见性
            toRemove: 0, // 待删除借书证号
            newBookInfo: { // 待新建图书信息
                Category: '',
                Title: '',
                Press: '',
                Publishyear: '',
                Author: '',
                Price: '',
                Number: ''
            },
            queryBookVisible: false, // 查询图书对话框可见性
            queryBookInfo: { // 待查询图书信息
                Category: '',
                Title: '',
                Press: '',
                MinPublishyear: '',
                MaxPublishyear: '',
                Author: '',
                MinPrice: '',
                MaxPrice: ''
            },
            modifyBookVisible: false, // 修改信息对话框可见性
            modifyBookInfo: { // 待新建图书信息
                id : '',
                Category: '',
                Title: '',
                Press: '',
                Publishyear: '',
                Author: '',
                Price: '',
            },
            incstockVisible: false, // 增加库存对话框可见性
            selectedBook: {
                id: '',
```

```
                deltastock: 0
            },// 选中的图书
            borrow :{
                id: '',
                time: Date.now(),
                cardid:0
            },
            borrowVisible: false, // 借书对话框可见性
            currentTimestamp: Date.now(),
            removeBookVisible: false ,// 删除图书对话框可见性
            toRemove: 0, // 待删除图书号
            returnBookVisible: false, // 还书对话框可见性
            returnBookInfo: { // 待还书信息
                id: '',
                cardid: ''
            },
            uploadVisible: false,
            fileList: []
        }
    },
```

下面将对各个功能进行演示，并且说明相应的实现。对于每个功能我们都有前端元素设计 `<template>`，前端方法实现 `<script>`，后端处理实现 `<java>` 三个部分组成，每个功能实现也都围绕这三个方面来展开。

4.2.1.1 页面刷新

首先实现页面的刷新，即在页面加载时，获取所有图书信息（即空查询条件）。这个请求我们使用 `GET` 来实现，代码如下

```
Refresh(){
    this.books = []
    axios.get("/book/")
        .then(response => {
            let books = response.data
            books.forEach(book => {
                this.books.push(book)
            })
        })
        .catch(error => {
            ElMessage.error("查询失败") // 显示消息提醒
        })
},
```

在 `Refresh` 函数中，我们首先将 `books` 清空，然后发送 `GET` 请求，获取所有图书信息，将其添加到 `books` 中。这个函数在页面加载时调用，以及在一些操作后调用，用于刷新页面。接着来看相应的后端处理。`/book` 的 `GET` 请求的处理如下:

```
    private void handleGetRequest(HttpExchange exchange) throws IOException {
        // 响应头，因为是 JSON 通信
        // System.out.println("Receive GET!");
        exchange.getResponseHeaders().set("Content-Type", "application/json");
        // 状态码为 200，也就是 status ok
        exchange.sendResponseHeaders(200, 0);
        // 获取输出流，java 用流对象来进行 io 操作
        OutputStream outputStream = exchange.getResponseBody();
```

```
        // 连接数据库
        try{
            ConnectConfig conf = new ConnectConfig();
            DatabaseConnector connector = new DatabaseConnector(conf);
            boolean connStatus = connector.connect();
            if (!connStatus) {
                log.severe("Failed to connect database.");
                System.exit(1);
            }
            LibraryManagementSystem library = new
LibraryManagementSystemImpl(connector);
            BookQueryConditions conditions = new BookQueryConditions();
            ApiResult bookApiResult = library.queryBook(conditions);
            BookQueryResults results =
((BookQueryResults)bookApiResult.payload);
            List<Book> books = results.getResults();
            String response = "[";
            for(Book book: books){//传回去的时候字典都是小写的
                response += "{\"id\": " + book.getBookId() + ", \"category\":
\"" + book.getCategory() + "\", \"title\": \"" + book.getTitle() + "\",
\"press\": \"" + book.getPress() + "\", \"publishyear\": " +
book.getPublishYear() + ", \"author\": \"" + book.getAuthor() + "\", \"price\":
" + book.getPrice() + ", \"stock\": " + book.getStock() + "},";
            }
            if(books.size() > 0){
                response = response.substring(0, response.length() - 1);
            }
            response += "]";
            // System.out.println(response);
            outputStream.write(response.getBytes());
            outputStream.close();
            //System.out.println("Send response!");
        }catch (Exception e) {
            e.printStackTrace();
        }
        }
```

在 `handleGetRequest` 函数中，我们首先设置响应头，然后连接数据库，调用 `queryBook` 函数，获取所有图书信息，将其转化为 JSON 格式，最后返回给前端。这样就实现了页面的刷新功能。接着来看其他功能的实现。后面功能实现均采用 `POST` 请求。

4.2.1.2 图书入库

图 4: 图书入库对话框

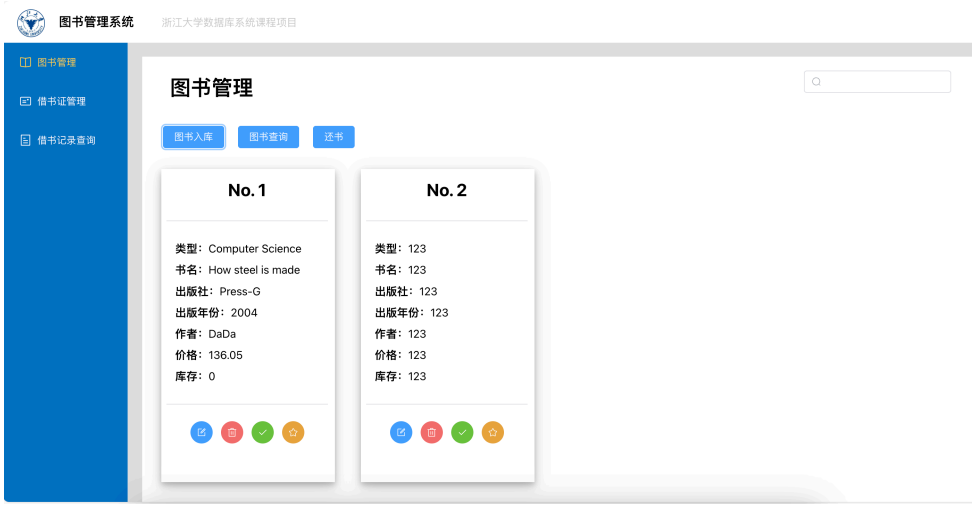

图 5: 成功入库

点击图书入库按钮，弹出对话框，填写图书信息，点击确定按钮，成功入库。实现代码如下:

首先是对话框的设计，使用`el-dialog`组件，设置对话框的标题，宽度，是否可见，以及关闭按钮的事件。

```
!-- 图书入库对话框 -->
    <el-dialog v-model="newBookVisible" title="图书入库" width="30%">
        <el-form :model="newBookInfo" label-width="80px">
            <el-form-item label="类别">
                <el-input v-model="newBookInfo.Category"></el-input>
            </el-form-item>
            <el-form-item label="书名">
                <el-input v-model="newBookInfo.Title"></el-input>
            </el-form-item>
            <el-form-item label="出版社">
                <el-input v-model="newBookInfo.Press"></el-input>
            </el-form-item>
            <el-form-item label="出版年份">
                <el-input v-model="newBookInfo.Publishyear"></el-input>
            </el-form-item>
            <el-form-item label="作者">
                <el-input v-model="newBookInfo.Author"></el-input>
            </el-form-item>
            <el-form-item label="价格">
                <el-input v-model="newBookInfo.Price"></el-input>
            </el-form-item>
            <el-form-item label="数量">
                <el-input v-model="newBookInfo.Number"></el-input>
            </el-form-item>
        </el-form>

        <template #footer>
            <span class="dialog-footer">
                <el-button @click="newBookVisible = false">取消</el-button>
                <el-button type="primary" @click="ConfirmNewBook">确定</el-
button>
            </span>
        </template>

    </el-dialog>
```

上面的代码涉及到了`el-form`组件，用于表单的设计，`el-form-item`组件用于表单项的设计，`el-input`组件用于输入框的设计，也包含了一些变量以及函数的绑定。以下是函数的实现:

```
ConfirmNewBook() {
    // 发出 POST 请求
    axios.post("/book/",
        { // 请求体
            action: "newbook",
            Category: this.newBookInfo.Category,
            Title: this.newBookInfo.Title,
            Press: this.newBookInfo.Press,
            Publishyear: this.newBookInfo.Publishyear,
            Author: this.newBookInfo.Author,
            Price : this.newBookInfo.Price,
            Number : this.newBookInfo.Number
        })
        .then(response => {
            ElMessage.success("图书入库成功") // 显示消息提醒
            this.Refresh()
            this.newBookVisible = false // 将对话框设置为不可见
        })
        .catch(error => {
            ElMessage.error(error.response.data) // 显示消息提醒
        })
}
```

`ConfirmNewBook`函数通过发送`post`请求，传递前端输入的`Category`，`Title`，`Press`，`Publishyear`，`Author`，`Price`，`Number`等信息，并且传递标记信息`action`，得到后端返回后，显示入库成功信息，然后调用`Refresh`函数刷新页面，最后将对话框设置为不可见。接着来看相应的后端处理。

由于我们是采用`POST`请求，并且在参数中明确操作的类型，所以我们先读取并拼接请求体的内容，随后使用`map`方法对请求体进行解析，根据获得的`action`的不同，调用不同的函数，最后返回相应的信息。其余两个处理器`CardHandler`和`BorrowHandler`的`POST`请求的思路也是类似的，后面对于结构上的分析也不再赘述。下面是`BookHandler`的`POST`请求的处理前端传入信息的实现:

```
    private void handlePostRequest(HttpExchange exchange) throws IOException{
        // 读取请求体，并存储到 requestBody 中
        InputStream requestBody = exchange.getRequestBody();
        BufferedReader reader = new BufferedReader(new
InputStreamReader(requestBody));
        StringBuilder requestBodyBuilder = new StringBuilder();
        String line;
        while ((line = reader.readLine()) != null) {
            requestBodyBuilder.append(line);
        }
        // 解析请求体
        String requestBodyString = requestBodyBuilder.toString();
        Gson gson = new Gson();
        Map<String, Object> requestBodyMap = gson.fromJson(requestBodyString,
Map.class);
        String action = (String) requestBodyMap.get("action");
        // 根据 action 的不同，做出不同的处理
```

```
        try{
                ConnectConfig conf = new ConnectConfig();
                DatabaseConnector connector = new DatabaseConnector(conf);
                boolean connStatus = connector.connect();
                if (!connStatus) {
                    log.severe("Failed to connect database.");
                    System.exit(1);
                }
                LibraryManagementSystem library = new
LibraryManagementSystemImpl(connector);
                /* 这里略去了具体的功能模块，在后面功能演示时讲解*/
        }catch (Exception e) {
                e.printStackTrace();
        }
    }
```

在图书入库的功能模块中，传入的`action`是`newbook`，我们根据这个`action`作出相应的处理，以下是具体实现:

```
    if("newbook".equals(action)){ // 匹配到 action 为 newbook
        Book book = new Book();
        book.setCategory((String) requestBodyMap.get("Category"));
        book.setTitle((String) requestBodyMap.get("Title"));
        book.setPress((String) requestBodyMap.get("Press"));
        book.setPublishYear(Integer.parseInt((String)
requestBodyMap.get("Publishyear")));
        book.setAuthor((String) requestBodyMap.get("Author"));
        book.setPrice(Double.parseDouble((String)
requestBodyMap.get("Price")));
        book.setStock(Integer.parseInt((String) requestBodyMap.get("Number")));

        ApiResult result = library.storeBook(book);

        exchange.getResponseHeaders().set("Content-Type", "text/plain");
        if(result.ok == false){
            exchange.sendResponseHeaders(400, 0);
        }else{
            exchange.sendResponseHeaders(200, 0);
        }
        OutputStream outputStream = exchange.getResponseBody();
        outputStream.write(result.message.getBytes());
        outputStream.close();
    }
```

对于图书入库的处理，我们首先根据传入的信息，构造一个`Book`对象，然后调用`storeBook`函数，将这个`Book`对象传入，得到返回的`ApiResult`，根据`ApiResult`的`ok`字段，设置返回的状态码，然后将返回的信息写入到输出流中，最后关闭输出流。这样就完成了图书入库的功能。我们利用状态码来说明是否成功，利用返回信息来提示用户。

4.2.1.3 图书查询

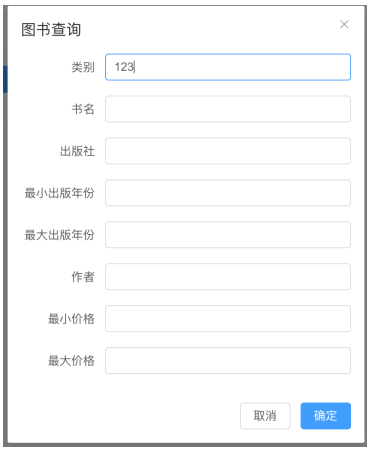

图 6: 图书查询对话框

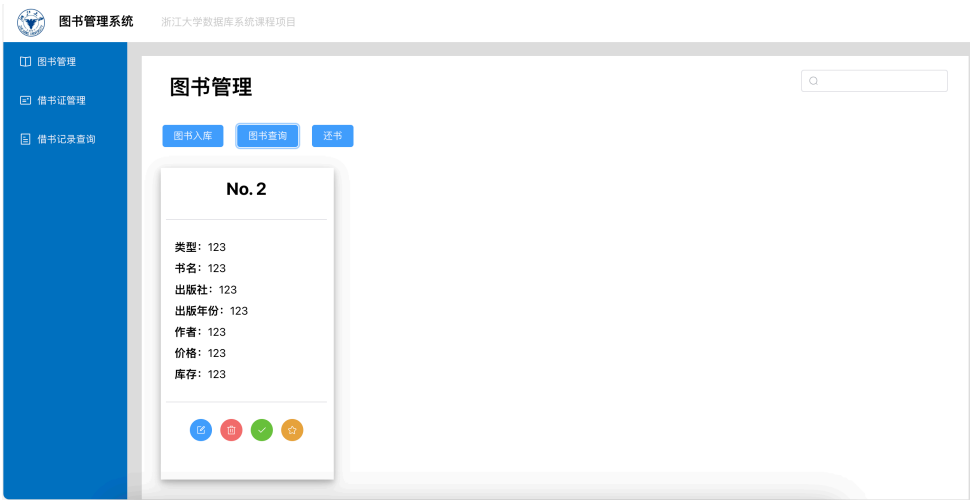

图 7: 图书查询结果

点击图书查询按钮，弹出对话框，填写查询信息，点击确定按钮，查询图书。在上图中可以看到成功查询了类型为'123'的书籍。实现代码如下:

同样使用对话框进行信息收集，以下是对话框的设计:

```
<!-- 图书查询对话框 -->
    <el-dialog v-model="queryBookVisible" title="图书查询" width="30%">
        <el-form :model="queryBookInfo" label-width="100px">
            <el-form-item label="类别">
                <el-input v-model="queryBookInfo.Category"></el-input>
            </el-form-item>
            <el-form-item label="书名">
                <el-input v-model="queryBookInfo.Title"></el-input>
            </el-form-item>
            <el-form-item label="出版社">
                <el-input v-model="queryBookInfo.Press"></el-input>
            </el-form-item>
            <el-form-item label="最小出版年份">
                <el-input v-model="queryBookInfo.MinPublishyear"></el-input>
            </el-form-item>
            <el-form-item label="最大出版年份">
                <el-input v-model="queryBookInfo.MaxPublishyear"></el-input>
            </el-form-item>
            <el-form-item label="作者">
                <el-input v-model="queryBookInfo.Author"></el-input>
```

```
            </el-form-item>
            <el-form-item label="最小价格">
                <el-input v-model="queryBookInfo.MinPrice"></el-input>
            </el-form-item>
            <el-form-item label="最大价格">
                <el-input v-model="queryBookInfo.MaxPrice"></el-input>
            </el-form-item>
        </el-form>

        <template #footer>
            <span class="dialog-footer">
                <el-button @click="queryBookVisible = false">取消</el-button>
                <el-button type="primary" @click="QueryBook">确定</el-button>
            </span>
        </template>
    </el-dialog>
```

这个对话框的结构与图书入库的对话框类似，不再赘述。以下是函数的实现:

```
    QueryBook() {
        // 发出 POST 请求
        this.books = []
        axios.post("/book/",
            { // 请求体
                action: "querybook",
                Category: this.queryBookInfo.Category,
                Title: this.queryBookInfo.Title,
                Press: this.queryBookInfo.Press,
                MinPublishyear: this.queryBookInfo.MinPublishyear,
                MaxPublishyear: this.queryBookInfo.MaxPublishyear,
                Author: this.queryBookInfo.Author,
                MinPrice: this.queryBookInfo.MinPrice,
                MaxPrice: this.queryBookInfo.MaxPrice
            })
            .then(response => {
                let books = response.data
                books.forEach(book => {
                    this.books.push(book)
                })
                ElMessage.success("查询成功")
                this.queryBookVisible = false // 将对话框设置为不可见
            })
            .catch(error => {
                ElMessage.error("查询失败") // 显示消息提醒
            })
    },
```

`QueryBook`函数通过发送`post`请求，传递前端输入的`Category`，`Title`，`Press`，`MinPublishyear`，`MaxPublishyear`，`Author`，`MinPrice`，`MaxPrice`等信息，并且传递标记信息`action:"querybook"`，得到后端返回后，将返回的数据存入`books`中，然后显示查询成功信息，此时前端的显示已经更新，最后将对话框设置为不可见。接着来看相应的后端处理。

```
    else if("querybook".equals(action)){
        BookQueryConditions conditions = new BookQueryConditions();
        System.out.println("Query book!");
        System.out.println((String) requestBodyMap.get("Category"));
        conditions.setCategory((String) requestBodyMap.get("Category"));
        conditions.setTitle((String) requestBodyMap.get("Title"));
```

```
        conditions.setPress((String) requestBodyMap.get("Press"));
        conditions.setAuthor((String) requestBodyMap.get("Author"));
        //conditions.setPublishYear(() requestBodyMap.get("Publishyear"));

        ApiResult bookApiResult = library.queryBook(conditions);
        BookQueryResults results = ((BookQueryResults)bookApiResult.payload);
        List<Book> books = results.getResults();
        String response = "[";
        for(Book book: books){
            response += "{\"id\": " + book.getBookId() + ", \"category\": \"" +
book.getCategory() + "\", \"title\": \"" + book.getTitle() + "\", \"press\":
\"" + book.getPress() + "\", \"publishyear\": " + book.getPublishYear() + ",
\"author\": \"" + book.getAuthor() + "\", \"price\": " + book.getPrice() + ",
\"stock\": " + book.getStock() + "},";
        }
        if(books.size() > 0){
            response = response.substring(0, response.length() - 1);
        }
        response += "]";
        exchange.getResponseHeaders().set("Content-Type", "application/json");
        exchange.sendResponseHeaders(200, 0);
        System.out.println(response);
        OutputStream outputStream = exchange.getResponseBody();
        outputStream.write(response.getBytes());
        outputStream.close();
        }
```

对于图书查询的处理，我们首先根据传入的信息，构造一个`BookQueryConditions`对象，然后调用`queryBook`函数，将这个`BookQueryConditions`对象传入，得到返回的`ApiResult`，根据`ApiResult`的`ok`字段，设置返回的状态码，然后将返回的信息写入到输出流中，最后关闭输出流。这样就完成了图书查询的功能。唯一注意到就是在拼字符串的时候，我们需要将`Book`对象的信息拼接成`json`格式的字符串，然后返回给前端。如果书籍数量为 0，我们不需要去除尾巴上的`,`，只需要返回`[]`即可。这个方法与之前刷新页面的查询类似，只是在查询条件上有所不同。

4.2.1.4 图书卡片

后面的几个功能模块是基于图书卡片的，因此先将图书卡片的实现放在这里，然后再进行后续功能的实现。

```
<!-- 图书卡片显示区 -->
<div style="display: flex;flex-wrap: wrap; justify-content: start;">
    <!-- 图书卡片 -->
    <div class="bookBox" v-for="book in books" v-
show="book.category.includes(toSearch)" :key="book.id">
        <div>
            <!-- 卡片标题 -->
            <div style="font-size: 25px; font-weight: bold;">No. {{ book.id }}
</div>
            <el-divider />

            <!-- 卡片内容 -->
            <div style="margin-left: 10px; text-align: start; font-size:
16px;">
```

```
                <p style="padding: 2.5px;"><span style="font-weight: bold;">类型: </span>{{ book.category }}</p>
                <p style="padding: 2.5px;"><span style="font-weight: bold;">书名: </span>{{ book.title }}</p>
                <p style="padding: 2.5px;"><span style="font-weight: bold;">出版社 : </span>{{ book.press }}</p>
                <p style="padding: 2.5px;"><span style="font-weight: bold;">出版年份 : </span>{{ book.publishyear }}</p>
                <p style="padding: 2.5px;"><span style="font-weight: bold;">作者 : </span>{{ book.author }}</p>
                <p style="padding: 2.5px;"><span style="font-weight: bold;">价格 : </span>{{ book.price }}</p>
                <p style="padding: 2.5px;"><span style="font-weight: bold;">库存: </span>{{ book.stock }}</p>
            </div>

            <el-divider />

            <!-- 卡片操作 -->
            <div style="margin-top: 5px;">
                <el-button type="primary" :icon="Edit" @click="
                modifyBookInfo.id=book.id,
                modifyBookInfo.Category=book.category,
                modifyBookInfo.Title=book.title,
                modifyBookInfo.Press=book.press,
                modifyBookInfo.Publishyear=book.publishyear,
                modifyBookInfo.Author=book.author,
                modifyBookInfo.Price=book.price,
                modifyBookInfo.stock=book.stock,
                modifyBookVisible = true" circle />
                <el-button type="danger" :icon="Delete" circle
                    @click="this.toRemove = book.id, this.removeBookVisible = true"
                        />
                <el-button type="success" :icon="Check" circle
                    @click="this.selectedBook.id = book.id, this.selectedBook.deltastock=0,this.incstockVisible = true"
                />
                <el-button type="warning" :icon="Star" circle
                    @click="this.borrow.id = book.id, this.borrowVisible = true"
                />
            </div>

        </div>
    </div>
</div>
```

首先我们需要一个显示区域，这里使用了`flex`布局，然后使用`v-for`指令遍历`books`，并且使用`v-show`指令根据搜索内容`toSearch`来过滤显示的图书卡片。在每个图书卡片中，我们设置了标题，内容，操作，分别显示了图书的信息，以及四个操作按钮，分别是修改图书信息，删除图书，修改库存，借书。这里的操作按钮都绑定了相应的函数，点击按钮会弹出对话框，填写信息，点击确定按钮，执行相应的操作。接着来看相应的功能的实现。

4.2.1.5 修改图书信息

图 8: 修改图书信息对话框

图 9: 修改信息完成

点击图书卡片下面的修改图书信息按钮，弹出对话框，填写修改信息，点击确定按钮，成功修改图书信息。我们实现了在弹出对话框中显示原本图书的信息，方便修改。实现代码如下:

```
    <el-button type="primary" :icon="Edit" @click="
    modifyBookInfo.id=book.id,
    modifyBookInfo.Category=book.category,
    modifyBookInfo.Title=book.title,
    modifyBookInfo.Press=book.press,
    modifyBookInfo.Publishyear=book.publishyear,
    modifyBookInfo.Author=book.author,
    modifyBookInfo.Price=book.price,
    modifyBookInfo.stock=book.stock,
    modifyBookVisible = true" circle />
```

这是修改图书信息按钮的实现，为了方便修改，我们将原本的图书信息显示在对话框中，、这里的`modifyBookInfo`是一个对象，包含了`id`，`Category`，`Title`，`Press`，`Publishyear`，

`Author`， `Price`， `stock`等信息，在点击按钮后，将这些信息赋值给`modifyBookInfo`，然后弹出对话框。以下是对话框的设计:

```
<el-dialog v-model="modifyBookVisible" title="修改信息" width="30%">
    <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem; margin-top: 20px; ">
        编号
        <el-input v-model="modifyBookInfo.id" style="width: 12.5vw;" disabled />
    </div>
    <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem; margin-top: 20px; ">
        类型
        <el-input v-model="modifyBookInfo.Category" style="width: 12.5vw;" clearable />
    </div>
    <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem; margin-top: 20px; ">
        书名
        <el-input v-model="modifyBookInfo.Title" style="width: 12.5vw;" clearable />
    </div>
    <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem; margin-top: 20px; ">
        出版社
        <el-input v-model="modifyBookInfo.Press" style="width: 12.5vw;" clearable />
    </div>
    <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem; margin-top: 20px; ">
        出版年份
        <el-input v-model.number="modifyBookInfo.Publishyear" style="width: 12.5vw;" clearable />
    </div>
    <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem; margin-top: 20px; ">
        作者
        <el-input v-model="modifyBookInfo.Author" style="width: 12.5vw;" clearable />
    </div>
    <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem; margin-top: 20px; ">
        价格
        <el-input v-model.number="modifyBookInfo.Price" style="width: 12.5vw;" clearable />
    </div>

    <template #footer>
        <span class="dialog-footer">
            <el-button @click="modifyBookVisible = false">取消</el-button>
            <el-button type="primary" @click="modifyBook">确定</el-button>
        </span>
    </template>
</el-dialog>
```

这里对话框的设计思路与之前的对话框类似，唯一新增的功能是使用了`disabled`属性，将`id`输入框设置为不可编辑，这样用户就无法修改`id`，保证了`id`的唯一性。以下是函数的实现:

```
modifyBook(){
    axios.post("/book/",
    { // 请求体
        action: "modifybook",
        id: this.modifyBookInfo.id,
        Category: this.modifyBookInfo.Category,
        Title: this.modifyBookInfo.Title,
        Press: this.modifyBookInfo.Press,
        Publishyear: this.modifyBookInfo.Publishyear,
        Author: this.modifyBookInfo.Author,
        Price: this.modifyBookInfo.Price
    })
    .then(response => {
        ElMessage.success("修改成功") // 显示消息提醒
        this.Refresh()
        this.modifyBookVisible = false // 将对话框设置为不可见
    })
    .catch(error => {
        ElMessage.error(error.response.data) // 显示消息提醒
    })
},
```

`modifyBook`函数通过发送`post`请求，传递前端输入的`id`，`Category`，`Title`，`Press`，`Publishyear`，`Author`，`Price`等信息，并且传递标记信息`action:"modifybook"`，得到后端返回后，显示修改成功信息，然后调用`Refresh`函数刷新页面，最后将对话框设置为不可见。接着来看相应的后端处理。

```
    else if("modify".equals(action)){
        Book book = new Book();
        book.setBookId(Integer.parseInt((String) requestBodyMap.get("id")));
        book.setCategory((String) requestBodyMap.get("Category"));
        book.setTitle((String) requestBodyMap.get("Title"));
        book.setPress((String) requestBodyMap.get("Press"));
        book.setPublishYear(Integer.parseInt((String)
requestBodyMap.get("Publishyear")));
        book.setAuthor((String) requestBodyMap.get("Author"));
        book.setPrice(Double.parseDouble((String)
requestBodyMap.get("Price")));
        book.setStock(Integer.parseInt((String) requestBodyMap.get("Number")));
        ApiResult result = library.modifyBookInfo(book);
        System.out.println(result.message);
        exchange.getResponseHeaders().set("Content-Type", "text/plain");
        if(result.ok == false){
            exchange.sendResponseHeaders(400, 0);
        }else{
            exchange.sendResponseHeaders(200, 0);
        }
        OutputStream outputStream = exchange.getResponseBody();
        outputStream.write(result.message.getBytes());
        outputStream.close();
    }
```

对于修改图书信息的处理，我们首先根据传入的信息，构造一个`Book`对象，然后调用`modifyBookInfo`函数，将这个`Book`对象传入，得到返回的`ApiResult`，根据`ApiResult`的`ok`字段，设置返回的状态码，然后将返回的信息写入到输出流中，最后关闭输出流。这样就完成了修改图书信息的功能。这里的处理与之前的处理类似，只是在传入的信息上有所不同。

4.2.1.6 删除图书

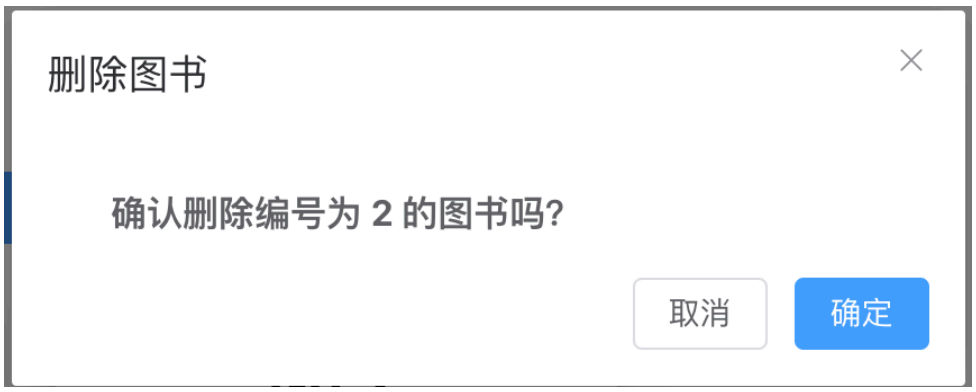

图 10: 删除图书对话框

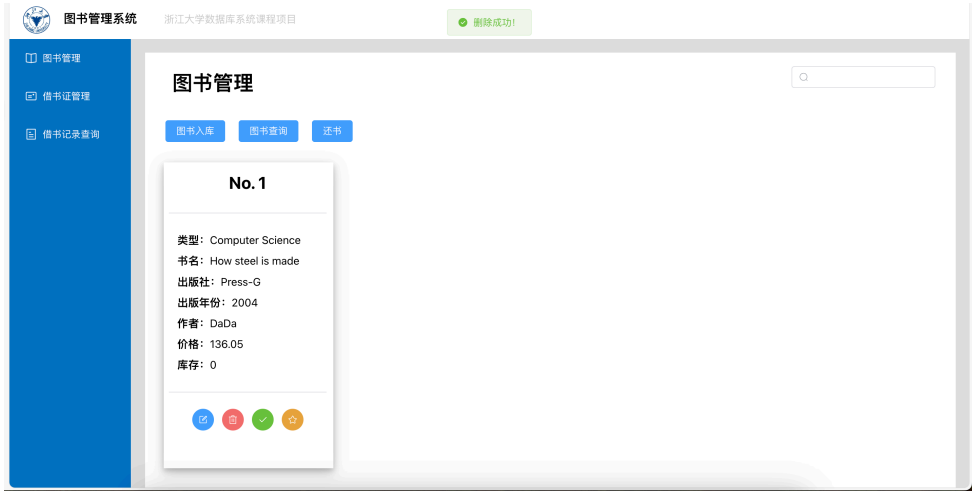

图 11: 删除图书成功

点击图书卡片下面的删除图书按钮，弹出对话框，点击确定按钮，成功删除图书。实现代码如下:

```
<el-button type="danger" :icon="Delete" circle
    @click="this.toRemove = book.id, this.removeBookVisible = true"
/>
```

首先是按钮的绑定，绑定了`book_id`的值用于后面对话框显示。

```
<!-- 删除图书对话框 -->
    <el-dialog v-model="removeBookVisible" title="删除图书" width="30%">
        <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem;
margin-top: 20px; ">
            确认删除编号为 {{ this.toRemove }} 的图书吗？
        </div>

        <template #footer>
            <span class="dialog-footer">
                <el-button @click="this.removeBookVisible = false">取消</el-
button>
                <el-button type="primary" @click="this.removeBook">确定</el-
button>
            </span>
```

```
        </template>
    </el-dialog>
```

这里对话框的实现与之前完全类似，不再赘述。以下是函数实现:

```
removeBook(){
    axios.post("/book/",
    {
        action: "removebook",
        id: this.toRemove
    })
    .then(response => {
        ElMessage.success("删除成功！")
        this.Refresh()
        this.removeBookVisible = false
    })
    .catch(error =>{
        ElMessage.error(error.response.data)
    })
},
```

`removeBook`函数通过发送`post`请求，传递前端输入的`id`，并且传递标记信息`action:"removebook"`，得到后端返回后，显示删除成功信息，然后调用`Refresh`函数刷新页面，最后将对话框设置为不可见。接着来看相应的后端处理。

```
    else if("removebook".equals(action)){
        int id = ((Double) requestBodyMap.get("id")).intValue();
        ApiResult result = library.removeBook(id);
        exchange.getResponseHeaders().set("Content-Type", "text/plain");
        if(result.ok == false){
            exchange.sendResponseHeaders(400, 0);
        }else{
            exchange.sendResponseHeaders(200, 0);
        }
        OutputStream outputStream = exchange.getResponseBody();
        outputStream.write(result.message.getBytes());
        outputStream.close();
    }
```

对于删除图书的处理，我们首先根据传入的信息，构造一个`id`，然后调用`removeBook`函数，将这个`id`传入，得到返回的`ApiResult`，根据`ApiResult`的`ok`字段，设置返回的状态码，然后将返回的信息写入到输出流中，最后关闭输出流。这样就完成了删除图书的功能。这里的处理与之前的处理类似，只是在传入的信息上有所不同。唯一需要注意的点就是传入参数的类型转换。

4.2.1.7 增加库存

图 12: 增加库存

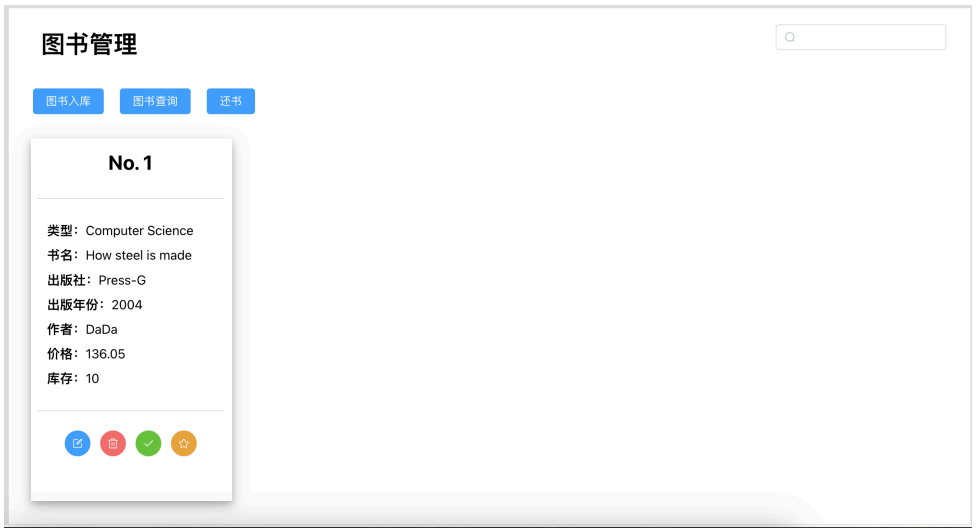

图 13: 增加库存结果

点击图书卡片下面的增加库存按钮，弹出对话框，填写增加数量，点击确定按钮，成功增加库存。实现代码如下:

```
<!-- 增加库存对话框 -->
    <el-dialog v-model="this.incstockVisible" title="增加库存" width="30%">
        <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem;
margin-top: 20px; ">
            编号
            <el-input v-model="this.selectedBook.id" style="width: 12.5vw;"
disabled />
        </div>
        <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem;
margin-top: 20px; ">
            增加库存
            <el-input v-model.number="this.selectedBook.stock" style="width:
12.5vw;" clearable />
        </div>

        <template #footer>
            <span class="dialog-footer">
```

```
                <el-button @click="this.incstockVisible = false">取消</el-
  button>
                <el-button type="primary" @click="incstock">确定</el-button>
            </span>
        </template>

    </el-dialog>
```

这里的按钮同样绑定了`book_id`的值用于后面对话框显示，其余对话框的设计与之前类似，不再赘述。以下是函数的实现:

```
incstock(){
    axios.post("/book/",
    { // 请求体
        action: "incstock",
        id: this.selectedBook.id,
        deltastock: this.selectedBook.stock
    })
    .then(response => {
        ElMessage.success("增加库存成功") // 显示消息提醒
        this.Refresh()
        this.incstockVisible = false // 将对话框设置为不可见
    })
    .catch(error =>{
        ElMessage.error(error.response.data)
    })
},
```

`incstock`函数通过发送`post`请求，传递前端输入的`id`，`deltastock`，并且传递标记信息`action:"incstock"`，得到后端返回后，显示增加库存成功信息，然后调用`Refresh`函数刷新页面，最后将对话框设置为不可见。接着来看相应的后端处理。

```
    else if("incstock".equals(action)){
        int id = ((Double) requestBodyMap.get("id")).intValue();
        int num = ((Double) requestBodyMap.get("deltastock")).intValue();
        ApiResult result = library.incBookStock(id, num);
        exchange.getResponseHeaders().set("Content-Type", "text/plain");
        if(result.ok == false){
            exchange.sendResponseHeaders(400, 0);
        }else{
            exchange.sendResponseHeaders(200, 0);
        }
        OutputStream outputStream = exchange.getResponseBody();
        outputStream.write(result.message.getBytes());
        outputStream.close();
}
```

对于增加库存的处理，我们首先根据传入的信息，构造一个`id`，`num`，然后调用`incBookStock`函数，将这个`id`，`num`传入，得到返回的`ApiResult`，根据`ApiResult`的`ok`字段，设置返回的状态码，然后将返回的信息写入到输出流中，最后关闭输出流。这样就完成了增加库存的功能。这里的处理与之前的处理类似，只是在传入的信息上有所不同。唯一需要注意的点就是传入参数的类型转换。

4.2.1.8 借书

图 14: 借书

这是图书卡片的最后一个功能——借书功能。点击图书卡片下面的借书按钮，弹出对话框，填写借书证 id，点击确定按钮，成功借书。实现代码如下:

```
<!-- 借书对话框 -->
    <el-dialog v-model="this.borrowVisible" title="借书" width="30%">
        <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem;
margin-top: 20px; ">
            编号
            <el-input v-model="this.borrow.id" style="width: 12.5vw;"
disabled />
        </div>
        <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem;
margin-top: 20px; ">
        时间
        <el-input v-model="formattedTimestamp" style="width: 12.5vw;"
disabled />
    </div>
    <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem; margin-
top: 20px; ">
        借书证编号
        <el-input v-model.number="this.borrow.cardid" style="width: 12.5vw;"
clearable />
    </div>

        <template #footer>
            <span class="dialog-footer">
                <el-button @click="this.borrowVisible = false">取消</el-button>
                <el-button type="primary" @click="borrowbook">确定</el-button>
            </span>
        </template>

    </el-dialog>
```

这里的按钮同样绑定了`book_id`的值用于后面对话框显示，同时还获取了实时的时间戳来显示时间，具体实现方式如下:

```
computed: {
    formattedTimestamp() {
      const date = new Date(this.currentTimestamp);
      const year = date.getFullYear();
      const month = date.getMonth() + 1;
      const day = date.getDate();
      const hours = date.getHours();
      const minutes = date.getMinutes();
      const seconds = date.getSeconds();
      return `${year}-${month.toString().padStart(2, '0')}-
${day.toString().padStart(2, '0')} ${hours.toString().padStart(2, '0')}:
${minutes.toString().padStart(2, '0')}:${seconds.toString().padStart(2, '0')}`;
       }
}
```

通过这个函数将时间戳转化为更具有可读性的年月日时分秒形式。下面是请求发送函数的实现

```
borrowbook(){
    axios.post("/book/",
    {
        action: "borrowbook",
        id: this.borrow.id,
        cardid: this.borrow.cardid,
        time: this.currentTimestamp
    })
    .then(response => {
        ElMessage.success("借书成功！")
        this.Refresh()
        this.borrowVisible = false
    })
    .catch(error =>{
        ElMessage.error(error.response.data)
        })
},
```

`borrowbook`函数通过发送`post`请求，传递前端输入的`id`，`cardid`，`time`，并且传递标记信息`action:"borrowbook"`，得到后端返回后，显示借书成功信息，然后调用`Refresh`函数刷新页面，最后将对话框设置为不可见。接着来看相应的后端处理。

```
    else if("borrowbook".equals(action)){
        Borrow borrow = new Borrow();
        borrow.setBookId(((Double) requestBodyMap.get("id")).intValue());
        borrow.setCardId(((Double) requestBodyMap.get("cardid")).intValue());
        borrow.setBorrowTime(((Double)
requestBodyMap.get("time")).longValue()/1000);
        ApiResult result = library.borrowBook(borrow);
        exchange.getResponseHeaders().set("Content-Type", "text/plain");
        if(result.ok == false){
            exchange.sendResponseHeaders(400, 0);
        }else{
            exchange.sendResponseHeaders(200, 0);
        }
        OutputStream outputStream = exchange.getResponseBody();
        outputStream.write(result.message.getBytes());
        outputStream.close();
    }
```

对于借书的处理，我们首先根据传入的信息，构造一个`Borrow`对象，然后调用`borrowBook`函数，将这个`Borrow`对象传入，得到返回的`ApiResult`，根据`ApiResult`的`ok`字段，设置返回的状态码，然后将返回的信息写入到输出流中，最后关闭输出流。这样就完成了借书的功能。这里的处理与之前的处理类似，只是在传入的信息上有所不同。唯一需要注意的点就是传入参数的类型转换和时间戳单位的换算。

4.2.1.9 还书

图 15: 还书

点击还书按钮，弹出对话框，输入要还的书籍编号和借书证编号，点击确定按钮，成功还书。实现代码如下:

```
<!-- 还书对话框 -->
<el-dialog v-model="returnBookVisible" title="还书" width="30%">
    <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem; margin-top: 20px; ">
        编号
        <el-input v-model.number="returnBookInfo.id" style="width: 12.5vw;" clearable />
    </div>
    <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem; margin-top: 20px; ">
        借书证编号
        <el-input v-model.number="returnBookInfo.cardid" style="width: 12.5vw;" clearable />
    </div>
    <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem; margin-top: 20px; ">
        还书时间
        <el-input v-model="formattedTimestamp" style="width: 12.5vw;" disabled />
    </div>

    <template #footer>
```

```
        <span class="dialog-footer">
            <el-button @click="returnBookVisible = false">取消</el-button>
            <el-button type="primary" @click="returnBook">确定</el-button>
        </span>
    </template>
</el-dialog>
```

这里的按钮绑定了`book_id`的值用于后面对话框显示，同时还获取了实时的时间戳来显示时间，再来看具体的函数实现:

```
returnBook(){
    axios.post("/book/",
    {
        action: "returnbook",
        id: this.returnBookInfo.id,
        cardid: this.returnBookInfo.cardid,
        time: this.currentTimestamp
    })
    .then(response => {
        ElMessage.success("还书成功！")
        this.Refresh()
        this.returnBookVisible = false
    })
    .catch(error =>{
        ElMessage.error(error.response.data)
    })
    },
```

`returnBook`函数通过发送`post`请求，传递前端输入的`id`，`cardid`，`time`，并且传递标记信息`action:"returnbook"`，得到后端返回后，显示还书成功信息，然后调用`Refresh`函数刷新页面，最后将对话框设置为不可见。接着来看相应的后端处理。

```
    else if("returnbook".equals(action)){
        Borrow borrow = new Borrow();
        borrow.setBookId(((Double) requestBodyMap.get("id")).intValue());
        borrow.setCardId(((Double) requestBodyMap.get("cardid")).intValue());
        borrow.setReturnTime(((Double)
requestBodyMap.get("time")).longValue()/1000);
        ApiResult result = library.returnBook(borrow);
        exchange.getResponseHeaders().set("Content-Type", "text/plain");
        if(result.ok == false){
            exchange.sendResponseHeaders(400, 0);
        }else{
            exchange.sendResponseHeaders(200, 0);
        }
        OutputStream outputStream = exchange.getResponseBody();
        outputStream.write(result.message.getBytes());
        outputStream.close();
    }
```

对于还书的处理，我们首先根据传入的信息，构造一个`Borrow`对象，然后调用`returnBook`函数，将这个`Borrow`对象传入，得到返回的`ApiResult`，根据`ApiResult`的`ok`字段，设置返回的状态码，然后将返回的信息写入到输出流中，最后关闭输出流。这样就完成了还书的功能。

这里的处理与之前的处理类似，只是在传入的信息上有所不同。唯一需要注意的点就是传入参数的类型转换和时间戳单位的换算。

4.2.2 `Borrow`

4.2.2.1 借书记录查询

图 16: 借书后

图 17: 还书后

这是我们在前两个功能模块中演示的借书还书的记录。可以看到成功的查询到刚才借书的记录，时间也是对的上的。一个是在还书前查询的，可以看到显示的是未归还，当还书后，就更新为具体的还书时间。这里我们实现了借书记录查询功能，点击借书记录查询按钮，弹出对话框，填写查询信息，点击确定按钮，成功查询借书记录。实现代码如下:

```
<template>
    <el-scrollbar height="100%" style="width: 100%;">

        <!-- 标题和搜索框 -->
        <div style="margin-top: 20px; margin-left: 40px; font-size: 2em; font-weight: bold;">
            借书记录查询
            <el-input v-model="toSearch" :prefix-icon="Search"
                style=" width: 15vw;min-width: 150px; margin-left: 30px; margin-right: 30px; float: right; ;"
                clearable />
        </div>

        <!-- 查询框 -->
        <div style="width:30%;margin:0 auto; padding-top:5vh;">

            <el-input v-model="this.toQuery" style="display:inline; " placeholder="输入借书证 ID"></el-input>
            <el-button style="margin-left: 10px;" type="primary" @click="QueryBorrows">查询</el-button>

        </div>

        <!-- 结果表格 -->
```

```
        <el-table v-if="isShow" :data="fitlerTableData" height="600"
            :default-sort="{ prop: 'borrowTime', order: 'ascending' }" :table-layout="'auto'"
            style="width: 100%; margin-left: 50px; margin-top: 30px; margin-right: 50px; max-width: 80vw;">
            <el-table-column prop="cardID" label="借书证 ID" />
            <el-table-column prop="bookID" label="图书 ID" sortable />
            <el-table-column prop="borrowTime" label="借出时间" sortable />
            <el-table-column prop="returnTime" label="归还时间" sortable />
        </el-table>

    </el-scrollbar>
</template>
```

这里出现了之前没有出现的`el-table`组件，这是一个表格组件，用于显示查询的结果。这里我们设置了四列，分别是借书证 ID，图书 ID，借出时间，归还时间，并且能够实现顺序的选择。接着来看函数的实现:

```
methods: {
    async QueryBorrows() {
        //ElMessage.success("开始查询")
        this.tableData = [] // 清空列表
        //{ params: { cardID: this.toQuery } }
        let response = await axios.get('/borrow', { params: { cardID: this.toQuery } }) // 向/borrow 发出 GET 请求，参数为 cardID=this.toQuery
        //ElMessage.success("请求发送成功")
        let borrows = response.data // 获取响应负载
        ElMessage.success("查询成功")
        borrows.forEach(borrow => { // 对于每一个借书记录
            //ElMessage.success("查询成功") // 提示查询成功
            this.tableData.push(borrow) // 将它加入到列表项中
            //this.tableData.push()
        });
            // 提示查询成功
        this.isShow = true // 显示结果列表
    }
}
```

`QueryBorrows`函数通过发送`get`请求，传递前端输入的`cardID`，得到后端返回后，显示查询成功信息，然后将返回的信息写入到`tableData`中，最后将结果列表设置为可见。这里使用了`async`和`await`关键字，使得函数变成异步函数，可以等待`axios`的返回。接着来看相应的后端处理。

```
private void handleGetRequest(HttpExchange exchange) throws IOException {
    // 响应头，因为是 JSON 通信
    exchange.getResponseHeaders().set("Content-Type", "application/json");
    // 状态码为 200，也就是 status ok
    exchange.sendResponseHeaders(200, 0);
    // 获取输出流，java 用流对象来进行 io 操作
    OutputStream outputStream = exchange.getResponseBody();
    URI requeryUri = exchange.getRequestURI();
    String query = requeryUri.getQuery();
    int id;
    id = Integer.parseInt(query.substring(7));//这是由于前端传递的参数是 cardID=xxx，所以需要截取后面的数字
```

```
12
13      try{
14          //连接数据库
15          ConnectConfig conf = new ConnectConfig();
16          DatabaseConnector connector = new DatabaseConnector(conf);
17          boolean connStatus = connector.connect();
18          if (!connStatus) {
19              log.severe("Failed to connect database.");
20              System.exit(1);
21          }
22          LibraryManagementSystem library = new
    LibraryManagementSystemImpl(connector);
23          //查询借阅历史
24          ApiResult borrowApiResult = library.showBorrowHistory(id);
25          //System.out.println(borrowApiResult.message);
26          BorrowHistories borrowHistories =
    ((BorrowHistories)borrowApiResult.payload);
27          //写入输出流
28          String response = "[";
29          for(BorrowHistories.Item item: borrowHistories.getItems()){
30              String formattedBorrowTime = time2String(item.getBorrowTime());
31              String formattedReturnTime = time2String(item.getReturnTime());
32              response += "{\"cardID\": " + item.getCardId() + ", \"bookID\": " +
    item.getBookId() + ", \"borrowTime\":\"" + formattedBorrowTime + "\",
    \"returnTime\":\"" + formattedReturnTime + "\"},";
33          }
34          response = response.substring(0, response.length() - 1);
35          response += "]";
36          outputStream.write(response.getBytes());
37          outputStream.close();
38      }catch (Exception e) {
39          e.printStackTrace();
40      }
41  }
```

对于查询借书记录的处理，我们首先根据传入的信息，构造一个`id`，然后调用`showBorrowHistory`函数，将这个`id`传入，得到返回的`ApiResult`，根据`ApiResult`的`ok`字段，设置返回的状态码，然后将返回的信息写入到输出流中，最后关闭输出流。这样就完成了查询借书记录的功能。这里的处理与之前的处理类似，只是在传入的信息上有所不同。需要注意的是传入参数的类型转换，和读取参数的方式的不同

4.2.3 `Card`

4.2.3.1 借书证显示

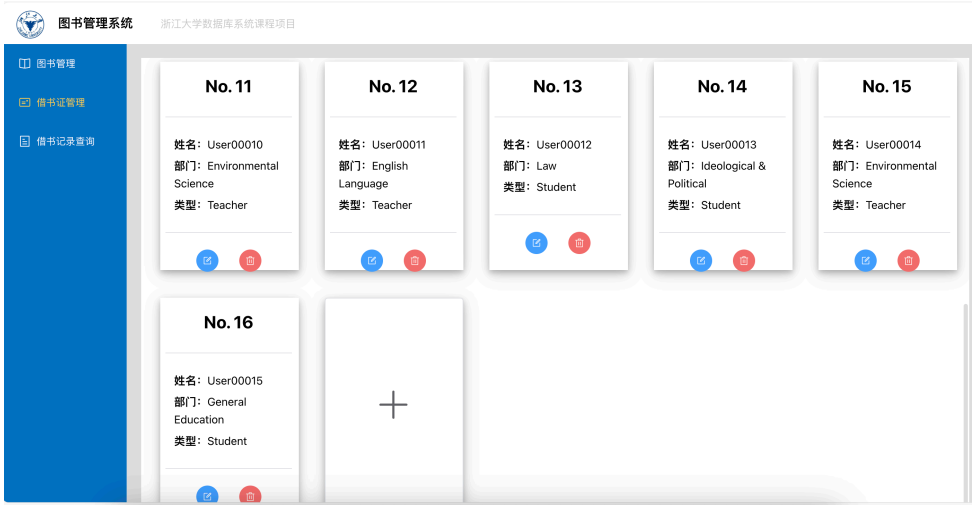

图 18: 借书证显示

这是我们在借书证管理模块中演示的借书证显示。可以看到成功的查询到了借书证的信息，包括借书证 ID，姓名，部门，类型。这里我们实现了借书证显示功能，点击借书证显示按钮，弹出对话框，填写查询信息，点击确定按钮，成功查询借书证信息。实现代码如下:

```
<!-- 借书证卡片 -->
<div class="cardBox" v-for="card in cards" v-show="card.name.includes(toSearch)" :key="card.id">
    <div>
        <!-- 卡片标题 -->
        <div style="font-size: 25px; font-weight: bold;">No. {{ card.id }}</div>

        <el-divider />

        <!-- 卡片内容 -->
        <div style="margin-left: 10px; text-align: start; font-size: 16px;">
            <p style="padding: 2.5px;"><span style="font-weight: bold;">姓名：</span>{{ card.name }}</p>
            <p style="padding: 2.5px;"><span style="font-weight: bold;">部门：</span>{{ card.department }}</p>
            <p style="padding: 2.5px;"><span style="font-weight: bold;">类型: </span>{{ card.type }}</p>
        </div>

        <el-divider />

        <!-- 卡片操作 -->
        <div style="margin-top: 10px;">
            <el-button type="primary" :icon="Edit" @click="this.toModifyInfo.id = card.id, this.toModifyInfo.name = card.name,
    this.toModifyInfo.department = card.department, this.toModifyInfo.type = card.type,
    this.modifyCardVisible = true" circle />
            <el-button type="danger" :icon="Delete" circle
                @click="this.toRemove = card.id, this.removeCardVisible = true"
                style="margin-left: 30px;" />
        </div>

    </div>
</div>
```

这个卡片与之前的卡片，这里显示了借书证的 ID，姓名，部门，类型，并且显示了修改和删除按钮。接着来看函数的实现:

```
QueryCards() {
    this.cards = [] // 清空列表
    let response = axios.get('/card') // 向/card 发出 GET 请求
        .then(response => {
            let cards = response.data // 接收响应负载
            cards.forEach(card => { // 对于每个借书证
                this.cards.push(card) // 将其加入到列表中
            })
        })
}
```

`QueryCards` 函数通过发送 `get` 请求，得到后端返回后，将返回的信息写入到 `cards` 中。这里使用了 `async` 和 `await` 关键字，使得函数变成异步函数，可以等待 `axios` 的返回。接着来看相应的后端处理。

```
private void handleGetRequest(HttpExchange exchange) throws IOException {
    // 响应头，因为是 JSON 通信
    exchange.getResponseHeaders().set("Content-Type", "application/json");
    // 状态码为 200，也就是 status ok
    exchange.sendResponseHeaders(200, 0);
    // 获取输出流，java 用流对象来进行 io 操作
    OutputStream outputStream = exchange.getResponseBody();
    // 连接数据库
    try{
        ConnectConfig conf = new ConnectConfig();
        //log.info("Success to parse connect config. " + conf.toString());
        // connect to database
        DatabaseConnector connector = new DatabaseConnector(conf);
        boolean connStatus = connector.connect();
        if (!connStatus) {
            log.severe("Failed to connect database.");
            System.exit(1);
        }
        LibraryManagementSystem library = new
LibraryManagementSystemImpl(connector);

        ApiResult cardApiResult = library.showCards();
        CardList cardList = ((CardList)cardApiResult.payload);
        String response = "[";
        for(Card card: cardList.getCards()){
            response += "{\"id\": " + card.getCardId() + ", \"name\": \"" +
card.getName() + "\", \"department\": \"" + card.getDepartment() + "\",
\"type\": \"" + card.getType() + "\"},";
        }
        response = response.substring(0, response.length() - 1);
        response += "]";
        outputStream.write(response.getBytes());
        outputStream.close();

    }catch (Exception e) {
        e.printStackTrace();
    }
}
```

对于查询借书证的处理，我们首先调用`showCards`函数，得到返回的`ApiResult`，根据`ApiResult`的`ok`字段，设置返回的状态码，然后将返回的信息写入到输出流中，最后关闭输出流。这样就完成了查询借书证的功能。

4.2.3.2 新建借书证

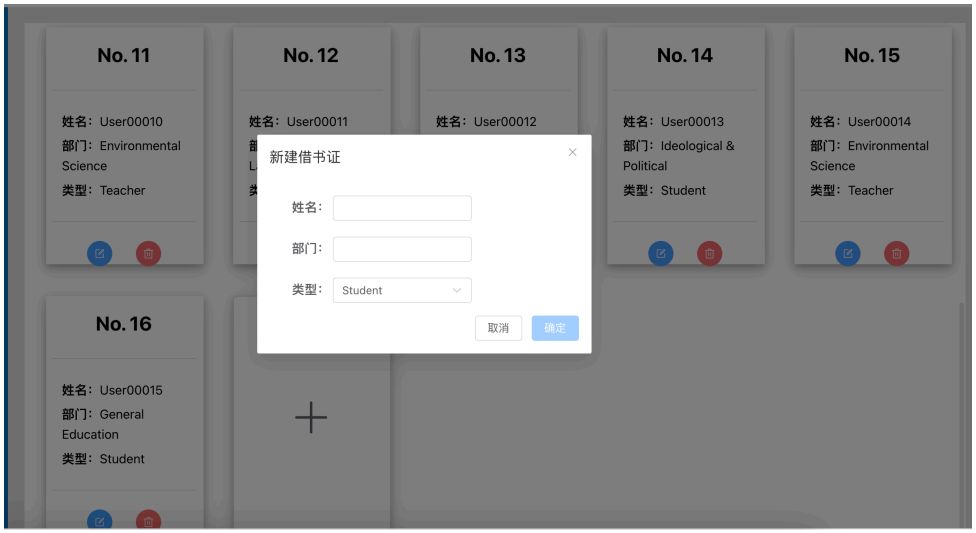

图 19: 新建借书证

图 20: 新建借书证完毕

点击新建借书证按钮，弹出对话框，填写新建借书证信息，点击确定按钮，成功新建借书证。实现代码如下:

```
<!-- 新建借书证对话框 -->
<el-dialog v-model="newCardVisible" title="新建借书证" width="30%" align-center>
    <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem; margin-top: 20px; ">
        姓名：
        <el-input v-model="newCardInfo.name" style="width: 12.5vw;" clearable />
    </div>
    <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem; margin-top: 20px; ">
```

```
        部门：
        <el-input v-model="newCardInfo.department" style="width: 12.5vw;" clearable />
    </div>
    <div style="margin-left: 2vw;   font-weight: bold; font-size: 1rem; margin-top: 20px; ">
        类型:
        <el-select v-model="newCardInfo.type" size="middle" style="width: 12.5vw;">
            <el-option v-for="type in types" :key="type.value" :label="type.label" :value="type.value" />
        </el-select>
    </div>

    <template #footer>
        <span>
            <el-button @click="newCardVisible = false">取消</el-button>
            <el-button type="primary" @click="ConfirmNewCard"
                :disabled="newCardInfo.name.length === 0 || newCardInfo.department.length === 0">确定</el-button>
        </span>
    </template>
</el-dialog>
```

这个对话框用到了`el-select`组件，用于选择借书证的类型。接着来看函数的实现:

```
ConfirmNewCard() {
    // 发出 POST 请求
    axios.post("/card/",
        { // 请求体
            action: "newcard",
            name: this.newCardInfo.name,
            department: this.newCardInfo.department,
            type: this.newCardInfo.type
        })
        .then(response => {
            ElMessage.success("借书证新建成功") // 显示消息提醒
            this.newCardVisible = false // 将对话框设置为不可见
            this.QueryCards() // 重新查询借书证以刷新页面
        })
        .catch(error => {
            ElMessage.error("借书证新建失败") // 显示消息提醒
        })
},
```

`ConfirmNewCard`函数通过发送`post`请求，传递前端输入的`name`，`department`，`type`，并且传递标记信息`action:"newcard"`，得到后端返回后，显示借书证新建成功信息，然后调用`QueryCards`函数刷新页面，最后将对话框设置为不可见。接着来看相应的后端处理。

```
if("newcard".equals(action)){
    Card card = new Card();
    card.setName((String) requestBodyMap.get("name"));
    card.setDepartment((String) requestBodyMap.get("department"));
    //System.out.println((String) requestBodyMap.get("type"));
    String typeString = "Student".equals((String) requestBodyMap.get("type"))?"S":"T";
    card.setType(Card.CardType.values(typeString));
```

```
8       ApiResult result = library.registerCard(card);
9
10      exchange.getResponseHeaders().set("Content-Type", "text/plain");
11      if(result.ok == false){
12          exchange.sendResponseHeaders(400, 0);
13      }else{
14          exchange.sendResponseHeaders(200, 0);
15      }
16      OutputStream outputStream = exchange.getResponseBody();
17      outputStream.write(result.message.getBytes());
18      outputStream.close();
19  }
```

对于新建借书证的处理，我们首先根据传入的信息，构造一个`Card`对象，然后调用`registerCard`函数，将这个`Card`对象传入，得到返回的`ApiResult`，根据`ApiResult`的`ok`字段，设置返回的状态码，然后将返回的信息写入到输出流中，最后关闭输出流。这样就完成了新建借书证的功能。需要提到的是，这里的`type`是一个枚举类型，需要根据前端传入的字符串来转化为枚举类型，处理的时候需要注意。

4.2.3.3 修改借书证

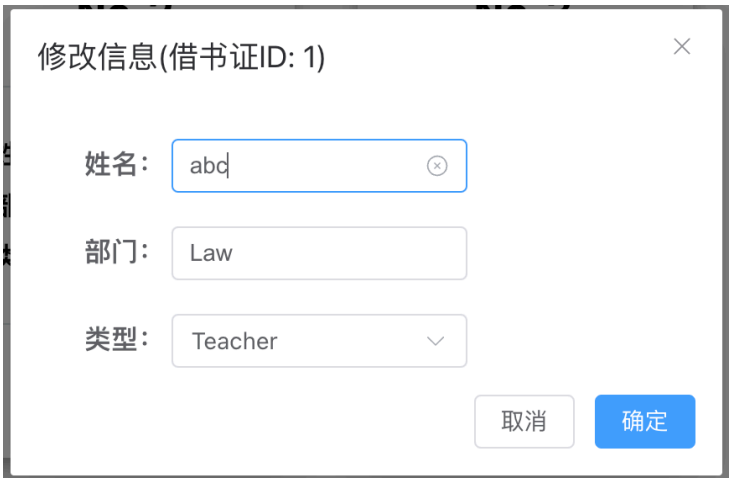

图 21: 修改借书证信息

图 22: 修改结果

按下借书证卡片下的修改按钮，弹出对话框，这个对话框显示了借书证的 ID，姓名，部门，类型，并且显示了确定和取消按钮。接着来看函数的实现:

```
<!-- 修改信息对话框 -->
    <el-dialog v-model="modifyCardVisible" :title="'修改信息(借书证 ID: ' + this.toModifyInfo.id + ')'" width="30%"
        align-center>
        <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem; margin-top: 20px; ">
            姓名：
            <el-input v-model="toModifyInfo.name" style="width: 12.5vw;" clearable />
        </div>
        <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem; margin-top: 20px; ">
            部门：
            <el-input v-model="toModifyInfo.department" style="width: 12.5vw;" clearable />
        </div>
        <div style="margin-left: 2vw;   font-weight: bold; font-size: 1rem; margin-top: 20px; ">
            类型:
            <el-select v-model="toModifyInfo.type" size="middle" style="width: 12.5vw;">
                <el-option v-for="type in types" :key="type.value" :label="type.label" :value="type.value" />
            </el-select>
        </div>

        <template #footer>
            <span class="dialog-footer">
                <el-button @click="modifyCardVisible = false">取消</el-button>
                <el-button type="primary" @click="ConfirmModifyCard"
                    :disabled="toModifyInfo.name.length === 0 || toModifyInfo.department.length === 0">确定</el-button>
            </span>
        </template>
    </el-dialog>
```

这个对话框用到了`el-select`组件，用于选择借书证的类型。而且这里的`disabled`条件增加了修改信息是否为空的限制，如果有一个为空那么就无法进行修改。接着来看函数的实现:

```
ConfirmModifyCard() {
    // 发出 POST 请求
    axios.post("/card/",
        { // 请求体
            action: "modifycard",
            id: this.toModifyInfo.id,
            name: this.toModifyInfo.name,
            department: this.toModifyInfo.department,
            type: this.toModifyInfo.type
        })
        .then(response => {
            ElMessage.success("借书证信息修改成功") // 显示消息提醒
            this.modifyCardVisible = false // 将对话框设置为不可见
            this.QueryCards() // 重新查询借书证以刷新页面
        })
```

```
        .catch(error => {
            ElMessage.error("借书证信息修改失败") // 显示消息提醒
        })
},
```

`ConfirmModifyCard`函数通过发送`post`请求，传递前端输入的`id`，`name`，`department`，`type`，并且传递标记信息`action:"modifycard"`，得到后端返回后，显示借书证信息修改成功信息，然后调用`QueryCards`函数刷新页面，最后将对话框设置为不可见。接着来看相应的后端处理。

```
    else if("modifycard".equals(action)){
        Card card = new Card();
        Double id = (double) requestBodyMap.get("id");
        card.setCardId(id.intValue());
        card.setName((String) requestBodyMap.get("name"));
        card.setDepartment((String) requestBodyMap.get("department"));
        String typeString = "Student".equals((String)
requestBodyMap.get("type"))?"S":"T";
        card.setType(Card.CardType.values(typeString));
        ApiResult result = library.modifyCardInfo(card);
        //System.out.println(result.message);
        exchange.getResponseHeaders().set("Content-Type", "text/plain");
        if(result.ok == false){
            exchange.sendResponseHeaders(400, 0);
        }else{
            exchange.sendResponseHeaders(200, 0);
        }
        OutputStream outputStream = exchange.getResponseBody();
        outputStream.write(result.message.getBytes());
        outputStream.close();
    }
```

对于修改借书证的处理，我们首先根据传入的信息，构造一个`Card`对象，然后调用`modifyCardInfo`函数，将这个`Card`对象传入，得到返回的`ApiResult`，根据`ApiResult`的`ok`字段，设置返回的状态码，然后将返回的信息写入到输出流中，最后关闭输出流。这样就完成了修改借书证的功能。需要提到的是，这里的`type`是一个枚举类型，需要根据前端传入的字符串来转化为枚举类型，处理的时候需要注意。

4.2.3.4 删除借书证

图 23: 删除借书证

删除借书证功能与前面删除图书的的实现完全类似。

```
<!-- 删除借书证对话框 -->
```

```
<el-dialog v-model="removeCardVisible" title="删除借书证" width="30%">
    <span>确定删除<span style="font-weight: bold;">{{ toRemove }}号借书证</span>吗? </span>

    <template #footer>
        <span class="dialog-footer">
            <el-button @click="removeCardVisible = false">取消</el-button>
            <el-button type="danger" @click="ConfirmRemoveCard">
                删除
            </el-button>
        </span>
    </template>
</el-dialog>
```

这个对话框有取消和确定删除的按钮，点击确认删除后会调用`ConfirmRemoveCard`函数，接着来看函数的实现:

```
ConfirmRemoveCard() {
    // 发出 DELETE 请求
    axios.post("/card/",
        {
            action: "deletecard",
            id:this.toRemove
        })
        .then(response => {
            ElMessage.success("借书证删除成功") // 显示消息提醒
            this.removeCardVisible = false // 将对话框设置为不可见
            this.QueryCards() // 重新查询借书证以刷新页面
        })
        .catch(error => {
            ElMessage.error("借书证删除失败，存在未还书籍！") // 显示消息提醒
        })
},
```

`ConfirmRemoveCard`函数通过发送`post`请求，传递前端输入的`id`，并且传递标记信息`action:"deletecard"`，得到后端返回后，显示借书证删除成功信息，然后调用`QueryCards`函数刷新页面，最后将对话框设置为不可见。接着来看相应的后端处理。

```
    else if("deletecard".equals(action)){
        Double id = (double) requestBodyMap.get("id");
        ApiResult result = library.removeCard(id.intValue());

        exchange.getResponseHeaders().set("Content-Type", "text/plain");
        if(result.ok == false){
            exchange.sendResponseHeaders(400, 0);
        }else{
            exchange.sendResponseHeaders(200, 0);
        }
        // exchange.sendResponseHeaders(200, 0);
        OutputStream outputStream = exchange.getResponseBody();
        outputStream.write(result.message.getBytes());
        outputStream.close();
    }
```

对于删除借书证的处理，我们首先根据传入的信息，调用`removeCard`函数，将这个`id`传入，得到返回的`ApiResult`，根据`ApiResult`的`ok`字段，设置返回的状态码，然后将返回的信息写入到输出流中，最后关闭输出流。这样就完成了删除借书证的功能。

4.2.4 说明与补充

至此，我们已经展示了包括图书入库，增加库存，修改图书信息，删除图书，添加借书证，修改借书证，删除借书证，借书，还书，借书记录查询等要求的全部功能性模块，均获得成功。除了以上提到的功能，对一些边界情况的测试（如修改库存为负数等）均已在验收时展示，在这里就不再演示。

5 思考题

5.1 图书管理系统的 E-R 图

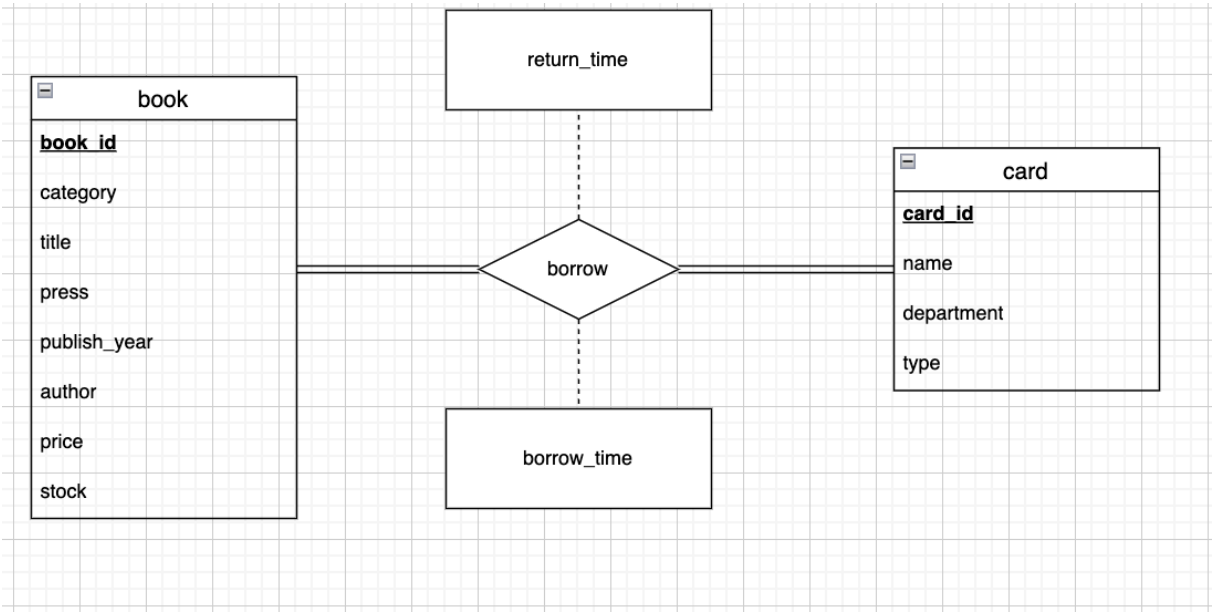

图 24: E-R 图

5.2 SQL 注入攻击

SQL 注入攻击是指通过把 SQL 命令插入到 Web 表单递交或输入域名的查询字符串，最终欺骗服务器执行恶意的 SQL 命令。这种攻击通常通过网络提交表单，或者在查询字符串中注入 SQL 代码，达到欺骗服务器执行恶意 SQL 命令的目的。会产生这种攻击的主要原因是程序没有细致地过滤用户输入的数据，致使非法数据侵入系统。

在我们的图书管理系统中，我们使用了`PreparedStatement`来防止 SQL 注入攻击。`PreparedStatement`是预编译的 SQL 语句，可以防止 SQL 注入攻击。在`PreparedStatement`中，SQL 语句的参数使用`?`来代替，然后使用`setXXX`方法来设置参数，这样可以防止 SQL 注入攻击。

下面举一个 SQL 注入攻击的例子： 对于一个信息查询系统，输入编号 id 就可以查询相应的信息，假设实现的函数如下:

```
public String queryInfo(String id){
    String sql = "SELECT * FROM info WHERE id = " + id;
    // 执行 sql 语句
    return return jdbcTemplate.query(sql,new BeanPropertyRowMapper(info.class));
}
```

那么如果输入的是`2 or 1 = 1`这样的条件，查询条件为真，那么就会返回所有的信息；或者说输入的是`2;show table;`这样的指令就是利用漏洞不光执行原程序设计者允许执行的一条指令，还可以额外的获得一些权限，有极大的安全隐患。

解决方法: 使用预编译语句，使用存储过程，检查数据类型，使用安全函数，多层验证，权限控制等。

5.3 并发访问

由于在重复读隔离级别下，事务在查询时会看到一致性的数据视图，但并不意味着其对应的数据在整个事务过程中都保持不变。通过以下方式来解决这个并发访问的问题:

1. 使用行级锁: 在读取库存量时，对相应的行进行加锁，直到事务完成。这样可以防止其他事务同时修改相同的数据，避免并发问题。
2. 增加库存检查: 在更新库存前，再次检查库存量是否足够。即使在查询到余量为 1 后，事务在更新库存前再次检查，如果此时余量已经不足，则不执行更新操作。

6 遇到的问题和解决办法

在测试过程中，在测试`borrowAndReturnBookTest`中，有以下这段测试:

```
// corner case, for return_time > borrow_time
Borrow nt = new Borrow(b0, c0);
nt.setReturnTime(returnTime:666);
Assert.assertFalse(library.returnBook(nt).ok);
nt.setReturnTime(r0.getBorrowTime());
Assert.assertFalse(library.returnBook(nt).ok);
```

图 25: 测试代码

其本意应该是检测当`return_time < borrow_time`的时候不能够进行还书操作。这符合逻辑，但是实际测试的时候，如果并未对还书时间进行检查，也能够通过这条测试。具体的原理由于时间原因我并没有深究。也许可以通过增加一次样例测试来避免这个问题。

而在我添加检测传入的`borrow`中的借书还书时间检测，而非从库中查询到的记录的`borrow_time`，这时候测试失败了，说明上面提到的测试点是有效的检验了这个问题。

具体的区别如下:

```java
public ApiResult returnBook(Borrow borrow) {
    Connection conn = connector.getConn();
    try{
        String sql = "select * from borrow where card_id = ? and book_id = ? and return_time = 0;";
        PreparedStatement qstmt = conn.prepareStatement(sql);
        qstmt.setInt(1, borrow.getCardId());
        qstmt.setInt(2, borrow.getBookId());
        ResultSet rs = qstmt.executeQuery();
        if(!rs.next()){
            rollback(conn);
            return new ApiResult(false, "The borrow doesn't exist!");
        }
        Long borrowTime = rs.getLong("borrow_time");
        if(borrowTime >= borrow.getReturnTime()){ //这里如果是 borrow.getReturnTime() >= borrowTime 则测试不通过，这起到了检验的效果
            rollback(conn);
            return new ApiResult(false, "The return time is earlier than the borrow time!");
        }
        sql = "Update borrow set return_time = ? where card_id = ? and book_id = ? and borrow_time = ?;";
        PreparedStatement stmt = conn.prepareStatement(sql);
        stmt.setLong(1, borrow.getReturnTime());
        stmt.setInt(2, borrow.getCardId());
        stmt.setInt(3, borrow.getBookId());
        stmt.setLong(4, borrowTime);
        int re = stmt.executeUpdate();
        if( re == 0 ){
            rollback(conn);
            return new ApiResult(false, "There is not the borrow!");
        }
        sql = "Update book set stock = stock + 1 where book_id = ?;";
        stmt = conn.prepareStatement(sql);
        stmt.setInt(1, borrow.getBookId());
        stmt.executeUpdate();
        stmt.close();
        commit(conn);
    }catch(Exception e){
        rollback(conn);
        return new ApiResult(false, e.getMessage());
    }
    return new ApiResult(true, "return successfully!");
}
```

以上的版本是可以通过测试而且也满足检验目标的，而下面的版本也可以通过测试，但是从逻辑上来说缺少了一些检验的部分。

```java
public ApiResult returnBook(Borrow borrow) {
    Connection conn = connector.getConn();
    try{
        String sql = "select * from borrow where card_id = ? and book_id = ? and borrow_time = ?;";//区别之一，这里直接查询 borrow_time
        PreparedStatement qstmt = conn.prepareStatement(sql);
        qstmt.setInt(1, borrow.getCardId());
        qstmt.setInt(2, borrow.getBookId());
        qstmt.setLong(3, borrow.getBorrowTime());
        ResultSet rs = qstmt.executeQuery();
        if(!rs.next()){
            rollback(conn);
```

```
            return new ApiResult(false, "The borrow doesn't exist!");
        }
        //区别 2，这里直接从传入的 borrow 中获取时间，而且没有检验，这似乎绕过了测试程序中的检验
        // Long borrowTime = rs.getLong("borrow_time");
        // if(borrowTime >= borrow.getReturnTime()){
        //     rollback(conn);
        //     return new ApiResult(false, "The return time is earlier than the borrow time!");
        // }
        sql = "Update borrow set return_time = ? where card_id = ? and book_id = ? and borrow_time = ?;";
        PreparedStatement stmt = conn.prepareStatement(sql);
        stmt.setLong(1, borrow.getReturnTime());
        stmt.setInt(2, borrow.getCardId());
        stmt.setInt(3, borrow.getBookId());
        stmt.setLong(4, borrow.getBorrowTime());
        int re = stmt.executeUpdate();
        if( re == 0 ){
            rollback(conn);
            return new ApiResult(false, "There is not the borrow!");
        }
        sql = "Update book set stock = stock + 1 where book_id = ?;";
        stmt = conn.prepareStatement(sql);
        stmt.setInt(1, borrow.getBookId());
        stmt.executeUpdate();
        stmt.close();
        commit(conn);
    }catch(Exception e){
        rollback(conn);
        return new ApiResult(false, e.getMessage());
    }
    return new ApiResult(true, "return successfully!");
}
```

以上两种均可以通过测试程序的检验，但第二种情况是不符合逻辑的，因为没有对借书还书时间进行检验，这说明测试程序中有漏洞。也许可以直接设置一组返回时间 < 借书时间的样例来检验这个问题。

7 总结

本实验中，我们实现了一个图书管理系统，掌握了数据库应用程序的设计方法，熟悉了 JDBC 的语法；除此之外，在写前端上耗费的时间三倍于基本功能模块的实现，尽管最后的成果也比较粗糙，但也是一个完整的前后端程序，能够正常运行和交互，也算是一件小有成就感的事情。对看到这里的助教，我表示万分感激，毕竟前面很多都是重复的讲解内容，辛苦助教了。