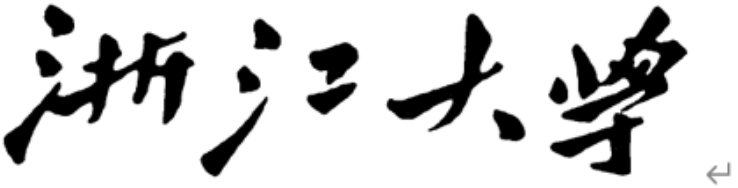


课程名称:	**数据库系统**
姓 名:	**杨吉祥**
学 院:	**计算机科学与技术学院**
系:	**竺可桢学院图灵班**
专 业:	**计算机科学与技术**
学 号:	**3230106222**

一、实验目的

- 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。
- 提供一个基于MySQL的精简图书管理程序，该图书管理程序应具备较好的可扩展性、鲁棒性和安全性，并且在高并发场景下仍能正确运行。

二、实验需求

- 在`LibraryManagementSystemImpl`中实现图书管理系统中所有应具备的功能模块
- 完成图书管理系统的前后端页面，使其成为一个用户能真正使用的图书管理系统。

三、实验环境

- **编程语言**：Java。
- **数据库**：MySQL。
- **框架/工具**：
  - 前端：Vue.js + Axios（参考提供的前端框架文档）。
  - 构建工具：Maven（`pom.xml`依赖管理）。

四、系统各模块的设计思路和实现

- `ApiResult storeBook(Book book)`：图书入库模块。先判断数据库中是否有书与要入库的书<类别, 书名, 出版社, 年份, 作者>均相同,如果有就认为两本书相同,入库失败。如果没有就认为这本书不同,往书库中插入这本书,`Statement.RETURN_GENERATED_KEYS`设置了自增ID,根据数据库生成的book_id值去更新book对象里的book_id。

```
public ApiResult storeBook(Book book) {
        Connection conn = connector.getConn();
        try {
            String checkSql = "SELECT book_id FROM book WHERE category = ? AND
title = ? AND press = ? AND publish_year = ? AND author = ?";
            PreparedStatement checkStmt = conn.prepareStatement(checkSql);
            checkStmt.setString(1, book.getCategory());
            checkStmt.setString(2, book.getTitle());
            checkStmt.setString(3, book.getPress());
            checkStmt.setInt(4, book.getPublishYear());
            checkStmt.setString(5, book.getAuthor());
            System.out.println(checkStmt);
            ResultSet rs = checkStmt.executeQuery();
            if (rs.next()) {
                return new ApiResult(false, "书籍已存在");
            }

            String insertSql = "INSERT INTO book (category, title, press,
publish_year, author, price, stock) VALUES (?, ?, ?, ?, ?, ?, ?)";
            PreparedStatement insertStmt = conn.prepareStatement(insertSql,
Statement.RETURN_GENERATED_KEYS);
            insertStmt.setString(1, book.getCategory());
```

```
            insertStmt.setString(2, book.getTitle());
            insertStmt.setString(3, book.getPress());
            insertStmt.setInt(4, book.getPublishYear());
            insertStmt.setString(5, book.getAuthor());
            insertStmt.setDouble(6, book.getPrice());
            insertStmt.setInt(7, book.getStock());
            insertStmt.executeUpdate();
            System.out.println(insertStmt);
            ResultSet generatedKeys = insertStmt.getGeneratedKeys();
            if (generatedKeys.next()) {
                book.setBookId(generatedKeys.getInt(1));
            }
            commit(conn);
            System.out.println("storeBook success");
            return new ApiResult(true, null);
        } catch (Exception e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
    }
```

- `ApiResult incBookStock(int bookId, int deltaStock)`：图书增加库存模块。先根据book_id去数据库查询书籍是否存在,如果图书不存在,则增加库存失败。由于增加的库存可正可负,所以再判断增加库存后的新库存是否为负数,如果是负数,则增加库存失败,如果不是,就根据book_id去更新该图书的库存。

```
public ApiResult incBookStock(int bookId, int deltaStock) {
        Connection conn = connector.getConn();
        try {
            String checkSql = "SELECT stock FROM book WHERE book_id = ?";
            PreparedStatement checkStmt = conn.prepareStatement(checkSql);
            checkStmt.setInt(1, bookId);
            System.out.println(checkStmt);
            ResultSet rs = checkStmt.executeQuery();
            if (!rs.next()) {
                return new ApiResult(false, "书籍不存在");
            }
            int currentStock = rs.getInt("stock");
            if (currentStock + deltaStock < 0) {
                return new ApiResult(false, "库存不足，最终库存不能为负");
            }

            String updateSql = "UPDATE book SET stock = stock + ? WHERE book_id =
?";
            PreparedStatement updateStmt = conn.prepareStatement(updateSql);
            updateStmt.setInt(1, deltaStock);
            updateStmt.setInt(2, bookId);
            System.out.println(updateStmt);
            updateStmt.executeUpdate();
            commit(conn);
            System.out.println("incBookStock success");
            return new ApiResult(true, null);
        } catch (Exception e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
```

```
        }
    }
```

- `ApiResult storeBook(List<Book> books)`：图书批量入库模块。因为有一本图书入库失败就会导致所有图书入库失败,所以在入库之前先将列表中的每一本书与数据库中的书进行比较,如果有两本书相同,就代表入库失败。如果所有书都没在数据库中,就将每一本书进行入库操作。需要注意的是,每次check和insert完成后都要将checkStmt和insertStmt中的参数清空。

```
public ApiResult storeBook(List<Book> books) {
        Connection conn = connector.getConn();
        try {
            String checkSql = "SELECT book_id FROM book WHERE category = ? AND
title = ? AND press = ? AND publish_year = ? AND author = ?";
            PreparedStatement checkStmt = conn.prepareStatement(checkSql);
            for (Book book : books) {
                checkStmt.setString(1, book.getCategory());
                checkStmt.setString(2, book.getTitle());
                checkStmt.setString(3, book.getPress());
                checkStmt.setInt(4, book.getPublishYear());
                checkStmt.setString(5, book.getAuthor());
                System.out.println(checkStmt);
                ResultSet rs = checkStmt.executeQuery();
                if (rs.next()) {
                    return new ApiResult(false, "书籍已存在（冲突书籍：" +
book.getTitle() + " by " + book.getAuthor() + ")");
                }
                checkStmt.clearParameters();
            }
            String insertSql = "INSERT INTO book (category, title, press,
publish_year, author, price, stock) VALUES (?, ?, ?, ?, ?, ?, ?)";
            PreparedStatement insertStmt = conn.prepareStatement(insertSql,
Statement.RETURN_GENERATED_KEYS);
            for (Book book : books) {
                insertStmt.setString(1, book.getCategory());
                insertStmt.setString(2, book.getTitle());
                insertStmt.setString(3, book.getPress());
                insertStmt.setInt(4, book.getPublishYear());
                insertStmt.setString(5, book.getAuthor());
                insertStmt.setDouble(6, book.getPrice());
                insertStmt.setInt(7, book.getStock());
                System.out.println(insertStmt);
                insertStmt.executeUpdate();
                ResultSet generatedKeys = insertStmt.getGeneratedKeys();
                if (generatedKeys.next()) {
                    book.setBookId(generatedKeys.getInt(1));
                }
                insertStmt.clearParameters();
            }
            commit(conn);
            System.out.println("storeBook success");
            return new ApiResult(true, "批量入库成功", books);
        } catch (Exception e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
```

```
    }
```

- `ApiResult removeBook(int bookId)`：图书删除模块。先根据book_id和return_tiime=0查询是否还有人未归还这本书,如果还有人尚未归还这本书,则删除失败。接着根据book_id查询数据库中这本书是否存在,如果不存在,删除操作失败。最后就可以根据book_id将数据库中的这本书删除。

```
public ApiResult removeBook(int bookId) {
        Connection conn = connector.getConn();
        try {
            String checkborrowSql = "SELECT * FROM borrow WHERE book_Id = ? AND
return_time = 0";
            PreparedStatement checkStmt = conn.prepareStatement(checkborrowSql);
            checkStmt.setInt(1, bookId);
            System.out.println(checkStmt);
            ResultSet rs = checkStmt.executeQuery();
            if (rs.next()) {
                return new ApiResult(false, "书籍尚未归还");
            }
            String checkbookSql = "SELECT * FROM book WHERE book_Id = ?";
            PreparedStatement checkStmt1 = conn.prepareStatement(checkbookSql);
            checkStmt1.setInt(1, bookId);
            ResultSet rs1 = checkStmt1.executeQuery();
            if (!rs1.next()) {
                return new ApiResult(false, "书籍不存在");
            }
            String deleteSql = "DELETE FROM book WHERE book_id = ?";
            PreparedStatement deleteStmt = conn.prepareStatement(deleteSql);
            deleteStmt.setInt(1, bookId);
            System.out.println(deleteStmt);
            deleteStmt.executeUpdate();
            commit(conn);
            System.out.println("removeBook success");
            return new ApiResult(true, "删除成功");
        } catch (Exception e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
    }
```

- `ApiResult modifyBookInfo(Book book)`：图书修改模块。先根据book_id查询数据库中这本书是否存在,如果不存在,则无法修改图书。如果存在,就根据给的新的book信息去更新数据库中这本书的信息,但不能修改书号和库存。

```
public ApiResult modifyBookInfo(Book book) {
        Connection conn = connector.getConn();
        try {
            String checkSql = "SELECT * FROM book WHERE book_id = ?";
            PreparedStatement checkStmt = conn.prepareStatement(checkSql);
            checkStmt.setInt(1, book.getBookId());
            System.out.println(checkStmt);
            ResultSet rs = checkStmt.executeQuery();
            if (!rs.next()) {
                return new ApiResult(false, "书籍不存在");
            }
```

```
            String updateSql = "UPDATE book SET category = ?, title = ?, press =
?, publish_year = ?, author = ?, price = ?WHERE book_id = ?";
            PreparedStatement updateStmt = conn.prepareStatement(updateSql);
            updateStmt.setString(1, book.getCategory());
            updateStmt.setString(2, book.getTitle());
            updateStmt.setString(3, book.getPress());
            updateStmt.setInt(4, book.getPublishYear());
            updateStmt.setString(5, book.getAuthor());
            updateStmt.setDouble(6, book.getPrice());
            updateStmt.setInt(7, book.getBookId());
            System.out.println(updateStmt);
            updateStmt.executeUpdate();
            commit(conn);
            System.out.println("modifyBookInfo success");
            return new ApiResult(true, "修改成功");
        } catch (Exception e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
    }
```

- `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。先判断查询条件是否为空,如果非空,则将该查询条件添加到查询语句中,并且将该查询条件的值提取到`params`,将所有查询条件判断完之后,再将`params`中的值依次插入到查询语句中,最后将查询到的书返回

```
public ApiResult queryBook(BookQueryConditions conditions) {
        Connection conn = connector.getConn();
        try {
            StringBuilder sqlBuilder = new StringBuilder(
                    "SELECT * FROM book WHERE 1=1");
            List<Object> params = new ArrayList<>();
            if (conditions.getCategory() != null) {
                sqlBuilder.append(" AND category = ?");
                params.add(conditions.getCategory());
            }
            if (conditions.getTitle() != null) {
                sqlBuilder.append(" AND title LIKE ?");
                params.add("%" + conditions.getTitle() + "%");
            }
            if (conditions.getPress() != null) {
                sqlBuilder.append(" AND press LIKE ?");
                params.add("%" + conditions.getPress() + "%");
            }
            if (conditions.getAuthor() != null) {
                sqlBuilder.append(" AND author LIKE ?");
                params.add("%" + conditions.getAuthor() + "%");
            }
            if (conditions.getMinPublishYear() != null) {
                sqlBuilder.append(" AND publish_year >= ?");
                params.add(conditions.getMinPublishYear());
            }
```

```
            if (conditions.getMaxPublishYear() != null) {
                sqlBuilder.append(" AND publish_year <= ?");
                params.add(conditions.getMaxPublishYear());
            }
            if (conditions.getMinPrice() != null) {
                sqlBuilder.append(" AND price >= ?");
                params.add(conditions.getMinPrice());
            }
            if (conditions.getMaxPrice() != null) {
                sqlBuilder.append(" AND price <= ?");
                params.add(conditions.getMaxPrice());
            }
            sqlBuilder.append(" ORDER BY ")
                    .append(conditions.getSortBy().getValue())
                    .append(" ")
                    .append(conditions.getSortOrder().getValue())
                    .append(", book_id ASC");
            PreparedStatement stmt =
conn.prepareStatement(sqlBuilder.toString());
            for (int i = 0; i < params.size(); i++) {
                stmt.setObject(i + 1, params.get(i));
            }
            System.out.println(stmt);
            ResultSet rs = stmt.executeQuery();
            List<Book> books = new ArrayList<>();
            while (rs.next()) {
                Book book = new Book(
                        rs.getString("category"),
                        rs.getString("title"),
                        rs.getString("press"),
                        rs.getInt("publish_year"),
                        rs.getString("author"),
                        rs.getDouble("price"),
                        rs.getInt("stock"));
                book.setBookId(rs.getInt("book_id"));
                books.add(book);
            }
            commit(conn);
            System.out.println("queryBook success");
            return new ApiResult(true, new BookQueryResults(books));
        } catch (Exception e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
    }
```

- `ApiResult borrowBook(Borrow borrow)`：借书模块。先根据book_id查询数据库中是否存在这本书,如果这本书不存在或者这本书库存小于等于0,则借书失败。接着根据card_id查询数据库中是否存在这个借书证,如果不存在,则借书失败。然后根据book_id,card_id和return_time=0判断该用户是否已借过该图书但未归还,如果有,则借书失败。最后向borrow数据库中插入这条借书记录,并且将这本书的库存减1。需要注意的是,要在查询语句后面加上`FOR UPDATE`,这样就能够避免查询发生了但数据还未更新导致查询到错误结果。

```
public ApiResult borrowBook(Borrow borrow) {
```

```
        Connection conn = connector.getConn();
        try {
            String stockSql = "SELECT * FROM book WHERE book_id=? FOR UPDATE";
            PreparedStatement stockStmt = conn.prepareStatement(stockSql);
            stockStmt.setInt(1, borrow.getBookId());
            System.out.println(stockStmt);
            ResultSet rs = stockStmt.executeQuery();
            if (!rs.next()) {
                return new ApiResult(false, "书籍不存在");
            }
            if (rs.getInt("stock") <= 0) {
                return new ApiResult(false, "书籍无库存");
            }
            String cardSql = "SELECT * FROM card WHERE card_id=? FOR UPDATE";
            PreparedStatement cardStmt = conn.prepareStatement(cardSql);
            cardStmt.setInt(1, borrow.getCardId());
            System.out.println(cardStmt);
            if (!cardStmt.executeQuery().next()) {
                return new ApiResult(false, "借书证不存在");
            }
            String borrowSql = "SELECT * FROM borrow WHERE book_id=? AND
card_id=? AND return_time=0 FOR UPDATE";
            PreparedStatement borrowStmt = conn.prepareStatement(borrowSql);
            borrowStmt.setInt(1, borrow.getBookId());
            borrowStmt.setInt(2, borrow.getCardId());
            System.out.println(borrowStmt);
            if (borrowStmt.executeQuery().next()) {
                return new ApiResult(false, "该用户已借此书未还");
            }
            String insertSql = "INSERT INTO borrow VALUES (?,?,?,0)";
            PreparedStatement insertStmt = conn.prepareStatement(insertSql);
            insertStmt.setInt(1, borrow.getCardId());
            insertStmt.setInt(2, borrow.getBookId());
            insertStmt.setLong(3, borrow.getBorrowTime());
            System.out.println(insertStmt);
            insertStmt.executeUpdate();
            String updateSql = "UPDATE book SET stock=stock-1 WHERE book_id=?";
            PreparedStatement updateStmt = conn.prepareStatement(updateSql);
            updateStmt.setInt(1, borrow.getBookId());
            System.out.println(updateStmt);
            updateStmt.executeUpdate();
            commit(conn);
            System.out.println("borrowBook success");
            return new ApiResult(true, "借书成功");
        } catch (Exception e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
    }
```

- `ApiResult returnBook(Borrow borrow)`：还书模块。先根据book_id查询要归还的书是否存在,如果不存在,则还书失败。再根据card_id查询借书证是否存在,如果不存在,则还书失败。然后根据book_id,card_id和return_time=0查询借书记录是否存在,如果不存在,则还书失败。再比较借书时间和还书时间,如果还书时间早于借书时间,逻辑上不成立,还书失败。最后将借书记录中的还书时间更

新,根据book_id将这本书的库存加1。同样也要在查询语句后加上`FOR UPDATE`,确保查询的结果正确。

```
public ApiResult returnBook(Borrow borrow) {
        Connection conn = connector.getConn();
        try {
            String stockSql = "SELECT * FROM book WHERE book_id=? FOR UPDATE";
            PreparedStatement stockStmt = conn.prepareStatement(stockSql);
            stockStmt.setInt(1, borrow.getBookId());
            System.out.println(stockStmt);
            ResultSet rs = stockStmt.executeQuery();
            if (!rs.next()) {
                return new ApiResult(false, "书籍不存在");
            }
            String cardSql = "SELECT * FROM card WHERE card_id=? FOR UPDATE";
            PreparedStatement cardStmt = conn.prepareStatement(cardSql);
            cardStmt.setInt(1, borrow.getCardId());
            System.out.println(cardStmt);
            if (!cardStmt.executeQuery().next()) {
                return new ApiResult(false, "借书证不存在");
            }
            String checkSql = "SELECT * FROM borrow WHERE book_id=? AND card_id=?
AND return_time=0 FOR UPDATE";
            PreparedStatement checkStmt = conn.prepareStatement(checkSql);
            checkStmt.setInt(1, borrow.getBookId());
            checkStmt.setInt(2, borrow.getCardId());
            System.out.println(checkStmt);
            rs = checkStmt.executeQuery();
            if (!rs.next()) {
                return new ApiResult(false, "未找到借书记录");
            }
            long borrowTime = rs.getLong("borrow_time");
            if (borrowTime >= borrow.getReturnTime()) {
                return new ApiResult(false, "还书时间早于借书时间");
            }
            String updateSql = "UPDATE borrow SET return_time=? WHERE book_id=?
AND card_id=? AND return_time=0";
            PreparedStatement updateStmt = conn.prepareStatement(updateSql);
            updateStmt.setLong(1, borrow.getReturnTime());
            updateStmt.setInt(2, borrow.getBookId());
            updateStmt.setInt(3, borrow.getCardId());
            System.out.println(updateStmt);
            updateStmt.executeUpdate();
            String updateStockSql = "UPDATE book SET stock=stock+1 WHERE
book_id=?";
            PreparedStatement updateStockStmt =
conn.prepareStatement(updateStockSql);
            updateStockStmt.setInt(1, borrow.getBookId());
            System.out.println(updateStockStmt);
            updateStockStmt.executeUpdate();
            commit(conn);
            System.out.println("returnBook success");
            return new ApiResult(true, "还书成功");
        } catch (Exception e) {
            rollback(conn);
```

```
            return new ApiResult(false, e.getMessage());
        }
    }
```

- `ApiResult showBorrowHistory(int cardId)`：借书记录查询模块。先根据card_id查询借书证是否存在,如果不存在,则借书记录查询失败。然后根据card_id对查询到的借书记录按借书时间递减,书号递增的方式进行排序,将得到的card_id,book,borrow构造为`BorrowHistories`中的`item`对象,最后利用`items`构造`BorrowHistories`对象返回

```
public ApiResult showBorrowHistory(int cardId) {
        Connection conn = connector.getConn();
        try {
            String checkSql = "SELECT * FROM card WHERE card_id=?";
            PreparedStatement checkStmt = conn.prepareStatement(checkSql);
            checkStmt.setInt(1, cardId);
            System.out.println(checkStmt);
            if (!checkStmt.executeQuery().next()) {
                return new ApiResult(false, "借书证不存在");
            }
            String sql = "SELECT * FROM book NATURAL JOIN borrow WHERE card_id=?
ORDER BY borrow_time DESC, book_id ASC";
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setInt(1, cardId);
            System.out.println(stmt);
            ResultSet rs = stmt.executeQuery();

            List<BorrowHistories.Item> items = new ArrayList<>();
            while (rs.next()) {
                Book book = new Book(
                        rs.getString("category"),
                        rs.getString("title"),
                        rs.getString("press"),
                        rs.getInt("publish_year"),
                        rs.getString("author"),
                        rs.getDouble("price"),
                        rs.getInt("stock"));
                book.setBookId(rs.getInt("book_id"));
                Borrow borrow = new Borrow(
                        rs.getInt("book_id"),
                        rs.getInt("card_id"));
                borrow.setBorrowTime(rs.getLong("borrow_time"));
                borrow.setReturnTime(rs.getLong("return_time"));
                BorrowHistories.Item item = new BorrowHistories.Item(
                        rs.getInt("card_id"),
                        book,
                        borrow);
                items.add(item);
            }
            commit(conn);
            System.out.println("showBorrowHistory success");
            return new ApiResult(true, new BorrowHistories(items));
        } catch (Exception e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
```

```
    }
```

- `ApiResult registerCard(Card card)`：借书证注册模块。先根据name,department和type在数据库中查询该借书证是否存在,如果已经存在,则注册失败。然后向数据库中添加新借书证,`Statement.RETURN_GENERATED_KEYS`设置了自增id,根据数据库生成的card_id值去更新card对象里的card_id。

```
public ApiResult registerCard(Card card) {
        Connection conn = connector.getConn();
        try {
            String checkSql = "SELECT * FROM card WHERE name=? AND department=?
AND type=?";
            PreparedStatement checkStmt = conn.prepareStatement(checkSql);
            checkStmt.setString(1, card.getName());
            checkStmt.setString(2, card.getDepartment());
            checkStmt.setString(3, card.getType().getStr());
            System.out.println(checkStmt);
            if (checkStmt.executeQuery().next()) {
                return new ApiResult(false, "借书证已存在");
            }
            String insertSql = "INSERT INTO card (name,department,type) VALUES
(?,?,?)";
            PreparedStatement insertStmt = conn.prepareStatement(insertSql,
Statement.RETURN_GENERATED_KEYS);
            insertStmt.setString(1, card.getName());
            insertStmt.setString(2, card.getDepartment());
            insertStmt.setString(3, card.getType().getStr());
            System.out.println(insertStmt);
            insertStmt.executeUpdate();
            ResultSet rs = insertStmt.getGeneratedKeys();
            if (rs.next()) {
                card.setCardId(rs.getInt(1));
            }
            commit(conn);
            System.out.println("registerCard success");
            return new ApiResult(true, "注册成功", card);
        } catch (SQLException e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
    }
```

- `ApiResult removeCard(int cardId)`：删除借书证模块。先根据card_id查询该借书证是否存在,如果不存在,则删除借书证失败。然后再根据card_id和return_tme=0查询该借书证是否还有书未归还,如果有,则删除借书证失败。最后根据card_id将数据库中的借书证删除。

```
public ApiResult removeCard(int cardId) {
        Connection conn = connector.getConn();
        try {
            String checkSql = "SELECT * FROM card WHERE card_id=?";
            PreparedStatement checkStmt = conn.prepareStatement(checkSql);
            checkStmt.setInt(1, cardId);
            System.out.println(checkStmt);
            ResultSet rs = checkStmt.executeQuery();
```

```
            if (!rs.next()) {
                return new ApiResult(false, "借书证不存在");
            }
            String checkBorrowSql = "SELECT * FROM borrow WHERE card_id=? AND
return_time=0";
            PreparedStatement checkBorrowStmt =
conn.prepareStatement(checkBorrowSql);
            checkBorrowStmt.setInt(1, cardId);
            System.out.println(checkBorrowStmt);
            if (checkBorrowStmt.executeQuery().next()) {
                return new ApiResult(false, "借书证尚有未还书籍");
            }
            String deleteSql = "DELETE FROM card WHERE card_id=?";
            PreparedStatement deleteStmt = conn.prepareStatement(deleteSql);
            deleteStmt.setInt(1, cardId);
            System.out.println(deleteStmt);
            deleteStmt.executeUpdate();
            commit(conn);
            System.out.println("removeCard success");
            return new ApiResult(true, "删除成功");
        } catch (SQLException e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
    }
```

- `ApiResult showCards()`：借书证查询模块。从数据库中按card_id将查询到的所有借书证进行排序,然后将查询结果添加到类型为card的列表中,最后利用该列表构造CardList对象返回。

```
public ApiResult showCards() {
        Connection conn = connector.getConn();
        try {
            String sql = "SELECT * FROM card ORDER BY card_id";
            PreparedStatement stmt = conn.prepareStatement(sql);
            System.out.println(stmt);
            ResultSet rs = stmt.executeQuery(sql);
            List<Card> cards = new ArrayList<>();
            while (rs.next()) {
                Card card = new Card(
                        rs.getInt("card_id"),
                        rs.getString("name"),
                        rs.getString("department"),
                        Card.CardType.values(rs.getString("type")));
                cards.add(card);
            }
            commit(conn);
            System.out.println("showCards success");
            return new ApiResult(true, new CardList(cards));
        } catch (Exception e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
    }
```

- `ApiResult modifyCard(Card card)`：图书证修改模块。先根据card_id查询借书证是否存在,如果不存在,则修改失败。然后根据card_id更新该借书证的信息。

```
public ApiResult modifyCard(Card card) {
        Connection conn = connector.getConn();
        try {
            String checkSql = "SELECT * FROM card WHERE card_id=?";
            PreparedStatement checkStmt = conn.prepareStatement(checkSql);
            checkStmt.setInt(1, card.getCardId());
            System.out.println(checkStmt);
            ResultSet rs = checkStmt.executeQuery();
            if (!rs.next()) {
                return new ApiResult(false, "借书证不存在");
            }
            String updateSql = "UPDATE card SET name=?, department=?, type=?
WHERE card_id=?";
            PreparedStatement updateStmt = conn.prepareStatement(updateSql);
            updateStmt.setString(1, card.getName());
            updateStmt.setString(2, card.getDepartment());
            updateStmt.setString(3, card.getType().getStr());
            updateStmt.setInt(4, card.getCardId());
            System.out.println(updateStmt);
            updateStmt.executeUpdate();
            commit(conn);
            System.out.println("modifyCard success");
            return new ApiResult(true, "修改成功");
        } catch (SQLException e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
    }
```

- `ApiResult showBooks()`：图书展示模块。将数据库查询到的图书添加到类型为book的列表中,然后利用该列表构造`BookQueryResults`对象返回。

```
public ApiResult showBooks() {
        Connection conn = connector.getConn();
        try {
            String sql = "SELECT * FROM book";
            PreparedStatement stmt = conn.prepareStatement(sql);
            System.out.println(stmt);
            ResultSet rs = stmt.executeQuery();
            List<Book> books = new ArrayList<>();
            while (rs.next()) {
                Book book = new Book(
                        rs.getString("category"),
                        rs.getString("title"),
                        rs.getString("press"),
                        rs.getInt("publish_year"),
                        rs.getString("author"),
                        rs.getDouble("price"),
                        rs.getInt("stock"));
                book.setBookId(rs.getInt("book_id"));
                books.add(book);
```

```
                System.out.println(book);
            }
            commit(conn);
            System.out.println("showBooks success");
            return new ApiResult(true, new BookQueryResults(books));
        } catch (Exception e) {
            rollback(conn);
            return new ApiResult(false, e.getMessage());
        }
    }
```

- 前端
  - `Card.vue`：借书证模块。只展示了我们自己要实现的代码部分。`ConfirmNewCard()`为新建借书证,采用POST请求,将要新建的借书证信息传给后端。`ConfirmModifyCard()`为修改借书证,采用POST请求,将要修改的借书证信息传给后端。`ConfirmRemoveCard()`为删除借书证,采用POST请求,将要删除的借书证id传给后端。`QueryCards()`为展示借书证,采用GET请求,接收后端传回来的借书证信息。当页面被渲染或者新建、删除、修改借书证后,都要采用`QueryCards()`来获得新的页面。

```
<script>
import { Delete, Edit, Search } from '@element-plus/icons-vue'
import { ElMessage } from 'element-plus'
import axios from 'axios'

export default {
methods: {
        ConfirmNewCard() {
            // 发出POST请求
            axios.post('/card/ConfirmNewCard',
                { // 请求体
                    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
                })
        },
        ConfirmModifyCard() {
            // TODO: YOUR CODE HERE
            axios.post('/card/ConfirmModifyCard',
                { // 请求体
                    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
                })
```

```
        },
        ConfirmRemoveCard() {
            // TODO: YOUR CODE HERE
            axios.post('/card/ConfirmRemoveCard',
                { // 请求体
                    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
                })
        },
        QueryCards() {
            this.cards = [] // 清空列表
            let response = axios.get('/card/QueryCards') // 发出GET请求
                .then(response => {
                    this.cards = response.data.cards // 接收响应负载
                })
        }
    },
    mounted() { // 当页面被渲染时
        this.QueryCards() // 查询借书证
    }
}
</script>
```

- `Borrow.vue`：借书记录模块。该模块只需要实现根据借书证查询借书记录即可。`QueryBorrows()`为查询借书记录,采用GET请求,将要查询的借书证id传给后端,同时要接收后端传回的借书记录。

```
<script>
import axios from 'axios';
import { Search } from '@element-plus/icons-vue'

export default {
    data() {
        return {
            isShow: false, // 结果表格展示状态
            tableData: [{ // 列表项
                cardId: 1,
                bookId: 1,
                borrowTime: "2024.03.04 21:48",
                returnTime: "2024.03.04 21:49"
            }],
            toQuery: '', // 待查询内容(对某一借书证号进行查询)
            toSearch: '', // 待搜索内容(对查询到的结果进行搜索)
            Search
        }
    },
    computed: {
        fitlerTableData() { // 搜索规则
            return this.tableData.filter(
                (tuple) =>
                    (this.toSearch == '') || // 搜索框为空，即不搜索
```

```
                    tuple.bookId == this.toSearch || // 图书号与搜索要求一致
                    tuple.borrowTime.toString().includes(this.toSearch) || //
借出时间包含搜索要求
                    tuple.returnTime.toString().includes(this.toSearch) // 归
还时间包含搜索要求
                )
            }
        },
        methods: {
            async QueryBorrows() {
                this.tableData = [] // 清空列表
                let response = await axios.get('/borrow', { params: { cardID:
this.toQuery } }) // 向/borrow发出GET请求，参数为cardID=this.toQuery
                this.tableData = response.data.items // 获取响应负载
                this.isShow = true // 显示结果列表
            }
        }
    }
    </script>
```

- `Book.vue`：图书模块。`handleQuery()`为查询书籍,采用POST请求,将要查询的条件传给给后端。`ConfirmNewBook()`为入库图书,采用POST请求,将新入库的图书信息传给后端。`ConfirmModifyBook()`为修改图书信息,采用POST请求,将要修改的图书信息传给后端。`ConfirmAddStock()`为增加图书库存,采用POST请求,将要增加库存的图书id和增加的库存量传给后端。`ConfirmRemoveBook()`为删除图书,采用POST请求,将要删除的图书书号传给后端。`ConfirmBorrowBook()`为借书,采用POST请求,将借的书的书号,借书证号和借书时间传给后端。`ConfirmReturnBook()`为还书,采用POST请求,将换的书的书号,借书证号和还书时间传给后端。`QueryBooks()`为获取所有图书,采用GET请求,接收后端传回的图书信息,并按照页面所容纳的信息条数截取出当前页面所要展示的图书。`handleFileChange(file)`为批量入库图书,先判断要上传的文件类型是否为Excel表格,如果不是,则上传失败,然后读取Excel表格中的数据,将其转换为JSON类型数据,采用POST请求,将转换后的JSON类型数据传给后端。由于我提供了分页大小的选项,所以需要判断调整分页大小或页码后当前页面应该显示全部书籍还是查询后的书籍,`initialize`一开始为true,表示未查询,所以要显示全部书籍,执行查询功能后,将`initialize`设为false,表示要显示查询后的书籍,按下重置按钮后将`initialize`调为true,重新显示全部书籍。每次进行入库,删除,修改图书等等操作后,都要执行`QueryBooks()`来获得最新的图书信息。

```
<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; ">图书管理
        </div>

        <el-form :model="queryParams" label-width="80px" class="query-form"
style="margin: 20px 40px;">
            <el-row :gutter="20">
                <el-col :span="8">
                    <el-form-item label="类别">
                        <el-input v-model="queryParams.category"
placeholder="输入类别关键词" clearable />
                    </el-form-item>
                </el-col>
```

```
                <el-col :span="8">
                    <el-form-item label="书名">
                        <el-input v-model="queryParams.title" placeholder="输
入书名关键词" clearable />
                    </el-form-item>
                </el-col>

                <el-col :span="8">
                    <el-form-item label="出版社">
                        <el-input v-model="queryParams.publisher"
placeholder="输入出版社关键词" clearable />
                    </el-form-item>
                </el-col>
            </el-row>

            <el-row :gutter="20">
                <el-col :span="8">
                    <el-form-item label="年份范围" style="margin-bottom: 0">
                        <div style="display: flex; align-items: center">
                        <el-date-picker
                            v-model="queryParams.startYear"
                            type="year"
                            value-format="YYYY"
                            placeholder="开始年份"
                            :picker-options="startYearOptions"
                            style="width: 110px"
                        />
                        <span style="margin: 0 8px; color: #888">至</span>
                        <el-date-picker
                            v-model="queryParams.endYear"
                            type="year"
                            value-format="YYYY"
                            placeholder="结束年份"
                            :picker-options="endYearOptions"
                            style="width: 110px"
                        />
                        </div>
                    </el-form-item>
                    </el-col>

                <el-col :span="8">
                    <el-form-item label="价格范围">
                        <el-input-number v-model="queryParams.minPrice"
:min="0" :precision="2" placeholder="最低价" />
                        <span style="margin: 0 10px">-</span>
                        <el-input-number v-model="queryParams.maxPrice"
:min="0" :precision="2" placeholder="最高价" />
                    </el-form-item>
                </el-col>
            </el-row>

            <el-row :gutter="20">
                <el-col :span="8">
                    <el-form-item label="排序方式">
```

```
                        <el-select v-model="queryParams.sortField"
placeholder="选择排序字段">
                            <el-option label="书号" value="BOOK_ID" />
                            <el-option label="书名" value="TITLE" />
                            <el-option label="类别" value="CATEGORY" />
                            <el-option label="作者" value="AUTHOR" />
                            <el-option label="出版年份" value="PUBLISH_YEAR"
/>
                            <el-option label="价格" value="PRICE" />
                            <el-option label="库存" value="STOCK" />
                            <el-option label="出版社" value="PRESS" />
                        </el-select>
                    </el-form-item>
                </el-col>

                <el-col :span="8">
                    <el-form-item label="排序顺序">
                        <el-radio-group v-model="queryParams.sortOrder">
                            <el-radio-button label="ASC">升序</el-radio-
button>
                            <el-radio-button label="DESC">降序</el-radio-
button>
                        </el-radio-group>
                    </el-form-item>
                </el-col>

                <el-col :span="8" style="text-align: right;">
                    <el-button type="primary" @click="handleQuery">查询</el-
button>
                    <el-button @click="resetQuery">重置</el-button>
                </el-col>
            </el-row>
        </el-form>

        <!-- 图书卡片显示区 -->
        <div style="display: flex;flex-wrap: wrap; justify-content: start;">

            <!-- 图书卡片 -->
            <div class="cardBox" v-for="book in books" :key="book.id">
                <div>
                    <!-- 卡片标题 -->
                    <div style="font-size: 25px; font-weight: bold;">{{
book.title }}</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;">书号：</span>{{ book.bookId }}</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;">类别：</span>{{ book.category }}</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;">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;">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: 10px;">
                        <el-button type="primary" :icon="Edit" @click="
                            this.toModifyInfo.title = book.title,
                            this.toModifyInfo.category = book.category,
                            this.toModifyInfo.author = book.author,
                            this.toModifyInfo.publisher = book.press,
                            this.toModifyInfo.price = book.price,
                            this.toModifyInfo.year = book.publishYear,
                            this.toModifyInfo.id = book.bookId,
                            this.modifyBookVisible = true" circle />
                        <el-button type="success" :icon="Plus" circle
                            @click="this.toAddStock = book.bookId,
this.addStockVisible = true, this.addStockAmount = 1" />
                        <el-button type="danger" :icon="Delete" circle
                            @click="this.toRemove = book.bookId,
this.removeBookVisible = true"
                            style="margin-left: 30px;" />
                        <el-button type="warning" :icon="Reading" circle
                            @click="this.toBorrow = book.bookId,
this.borrowBookVisible = true"
                            style="margin-left: 30px;" />
                        <el-button type="info" :icon="Back" circle
                            @click="this.toReturn = book.bookId,
this.returnBookVisible = true, this.returnCardId = ''"
                            style="margin-left: 30px;" />
                    </div>
                </div>
            </div>

            <!-- 新建图书卡片 -->
            <el-button class="newCardBox"
                @click="newBookInfo.title = '', newBookInfo.category = '',
newBookInfo.author = '',
                        newBookInfo.publisher = '', newBookInfo.year = new
Date().getFullYear(), newBookInfo.price = 0.00,
                        newBookInfo.stock = 1, newBookVisible = true">
                <el-icon style="height: 50px; width: 50px;">
                    <Plus style="height: 100%; width: 100%;" />
                </el-icon>
            </el-button>
```

```
            <!-- 批量入库按钮 -->
            <el-button class="newCardBox" @click="batchImportVisible = true">
                <el-icon style="height: 50px; width: 50px;">
                    <Upload style="height: 100%; width: 100%;" />
                </el-icon>
                <span style="margin-top: 10px; display: block;">批量入库
</span>
            </el-button>
        </div>


         <!-- 分页 -->
         <el-pagination
            v-model:current-page="queryParams.pageNum"
            v-model:page-size="queryParams.pageSize"
            :total="this.total"
            :page-sizes="[10, 20, 50, 100]"
            layout="total, sizes, prev, pager, next, jumper"
            @size-change="handleSizeChange"
            @current-change="handleCurrentChange"
            style="margin: 20px 40px;"
        />


        <!-- 新建图书对话框 -->
        <el-dialog v-model="newBookVisible" title="图书入库" width="30%"
align-center>
            <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem;
margin-top: 20px; ">
                书名：
                <el-input v-model="newBookInfo.title" style="width: 12.5vw;"
clearable />
            </div>
            <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem;
margin-top: 20px; ">
                类别：
                <el-input v-model="newBookInfo.category" 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="newBookInfo.author" 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="newBookInfo.publisher" 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-number v-model="newBookInfo.year" :min="1"
:max="new Date().getFullYear()" style="width: 12.5vw;" />
```

```
            </div>
            <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem;
margin-top: 20px; ">
                价格：
                <el-input-number v-model="newBookInfo.price" :min="0"
:step="0.01" style="width: 12.5vw;" />
            </div>
            <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem;
margin-top: 20px; ">
                库存数量：
                <el-input-number v-model="newBookInfo.stock" :min="1"
style="width: 12.5vw;" />
            </div>

            <template #footer>
                <span>
                    <el-button @click="newBookVisible = false">取消</el-
button>
                    <el-button type="primary" @click="ConfirmNewBook"
                        :disabled="newBookInfo.title.length === 0 ||
newBookInfo.author.length === 0 || newBookInfo.category.length === 0 ||
newBookInfo.year === null
                        || newBookInfo.price === null || newBookInfo.stock
=== null || newBookInfo.publisher.length === 0  ">确定</el-button>
                </span>
            </template>
        </el-dialog>


        <!-- 修改图书信息对话框 -->
        <el-dialog v-model="modifyBookVisible" :title="'修改图书信息(书号：' +
this.toModifyInfo.id + ')'" width="30%"
            align-center>
            <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem;
margin-top: 20px; ">
                书名：
                <el-input v-model="toModifyInfo.title" style="width: 12.5vw;"
clearable />
            </div>
            <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem;
margin-top: 20px; ">
                类别：
                <el-input v-model="toModifyInfo.category" 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="toModifyInfo.author" 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="toModifyInfo.publisher" 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-number v-model="toModifyInfo.year" :min="1"
:max="new Date().getFullYear()" style="width: 12.5vw;" />
            </div>
            <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem;
margin-top: 20px; ">
                价格：
                <el-input-number v-model="toModifyInfo.price" :min="0"
:step="0.01" style="width: 12.5vw;" />
            </div>

            <template #footer>
                <span class="dialog-footer">
                    <el-button @click="modifyBookVisible = false">取消</el-
button>
                    <el-button type="primary" @click="ConfirmModifyBook"
                        :disabled="toModifyInfo.title === null ||
toModifyInfo.author === null || toModifyInfo.category === null ||
toModifyInfo.year === null
                        || toModifyInfo.price === null ||
toModifyInfo.publisher === null">确定</el-button>
                </span>
            </template>
        </el-dialog>

        <!-- 增加库存对话框 -->
        <el-dialog v-model="addStockVisible" :title="'增加库存(书号: ' +
this.toAddStock + ')'" width="30%" align-center>
            <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem;
margin-top: 20px; ">
                增加数量：
                <el-input-number v-model="addStockAmount" :min="-999999"
style="width: 12.5vw;" />
            </div>

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

        <!-- 删除图书对话框 -->
        <el-dialog v-model="removeBookVisible" title="删除图书" width="30%">
            <span>确定删除<span style="font-weight: bold;">{{ toRemove }}号图书
</span>吗？</span>

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

```
                    <el-button @click="removeBookVisible = false">取消</el-
button>
                    <el-button type="danger" @click="ConfirmRemoveBook">
                        删除
                    </el-button>
                </span>
            </template>
        </el-dialog>

        <!-- 借书对话框 -->
        <el-dialog v-model="borrowBookVisible" title="借阅图书" width="30%">
            <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem;
margin-top: 20px; ">
                借书证号:
                <el-input v-model="borrowCardId" style="width: 12.5vw;"
clearable />
            </div>

            <template #footer>
                <span class="dialog-footer">
                    <el-button @click="borrowBookVisible = false">取消</el-
button>
                    <el-button type="primary" @click="ConfirmBorrowBook"
                        :disabled="borrowCardId.length === 0">确定</el-
button>
                </span>
            </template>
        </el-dialog>

         <!-- 还书对话框 -->
        <el-dialog v-model="returnBookVisible" title="归还图书" width="30%">
            <div style="margin-left: 2vw; font-weight: bold; font-size: 1rem;
margin-top: 20px; ">
                借书证号:
                <el-input v-model="returnCardId" style="width: 12.5vw;"
clearable />
            </div>

            <template #footer>
                <span class="dialog-footer">
                    <el-button @click="returnBookVisible = false">取消</el-
button>
                    <el-button type="primary" @click="ConfirmReturnBook"
                        :disabled="returnCardId.length === 0">确定归还</el-
button>
                </span>
            </template>
        </el-dialog>

        <!-- 批量入库对话框 -->
        <el-dialog v-model="batchImportVisible" title="批量入库" width="50%">
            <el-upload
                ref="uploadRef"
                class="upload-demo"
                drag
                multiple
```

```
                :auto-upload="false"
                :on-change="handleFileChange">
                <el-icon class="el-icon--upload"><upload-filled /></el-icon>
                <div class="el-upload__text">
                    拖拽文件到此处或<em>点击上传</em>
                </div>
                <template #tip>
                    <div class="el-upload__tip">
                        请上传Excel文件,包含书名、类别、作者、出版社、年份、库存等列
                    </div>
                </template>
            </el-upload>
        </el-dialog>

    </el-scrollbar>
</template>

<script>
import { Delete, Edit, Search, Plus, Upload, Reading, Back } from '@element-
plus/icons-vue'
import { ElMessage } from 'element-plus'
import axios from 'axios'
import * as XLSX from 'xlsx';



export default {
    data() {
        return {
            books: [], // 图书列表
            Delete,
            Edit,
            Search,
            Plus,
            Upload,
            Reading,
            Back,
            newBookVisible: false, // 新建图书对话框可见性
            removeBookVisible: false, // 删除图书对话框可见性
            modifyBookVisible: false, // 修改图书对话框可见性
            addStockVisible: false, // 增加库存对话框可见性
            borrowBookVisible: false, // 借书对话框可见性
            returnBookVisible: false, // 还书对话框可见性
            batchImportVisible: false, // 批量入库对话框可见性
            toRemove: '', // 待删除图书ID
            toAddStock: '', // 待增加库存图书ID
            toBorrow: '', // 待借阅图书ID
            toReturn: '', // 待归还图书ID
            addStockAmount: 1, // 增加库存数量
            borrowCardId: '', // 借书证号
            returnCardId: '', // 还书证号
            initialize: true, // 是否初始化
            total: 0, // 图书总数

            newBookInfo: { // 待新建图书信息
```

```
                title: '',
                category: '',
                author: '',
                publisher: '',
                year: '',
                price: 0.00,
                stock: 1
            },
            toModifyInfo: { // 待修改图书信息
                id: 0,
                title: null,
                category: null,
                author: null,
                publisher: null,
                year: null,
                price: 0.00
            },
            queryParams: { // 查询参数
                category: null,
                title: null,
                publisher: null,
                author: null,
                startYear: null,
                endYear: null,
                minPrice: null,
                maxPrice: null,
                sortField: 'BOOK_ID',
                sortOrder: 'ASC',
                pageNum: 1,
                pageSize: 10
            },
        }
    },
    methods: {

        handleQuery() {
            this.books = [] // 清空图书列表
            this.initialize = false // 设置初始化为false
            axios.post('/book/handleQuery', this.queryParams) // 发送查询请求
                .then(response => {
                    const start = (this.queryParams.pageNum - 1) *
this.queryParams.pageSize
                    const end = start + this.queryParams.pageSize
                    this.books = response.data.results.slice(start, end) //
获取当前页的图书数据
                    this.total = response.data.count // 获取图书总数
                    if (this.books.length === 0) {
                        ElMessage.warning("没有符合条件的图书")
                    }
                })
                .catch(error => {
                    ElMessage.error("查询图书失败: " + error.message)
                })
        },

        resetQuery() {
```

```
            this.queryParams = {
                category: null,
                title: null,
                publisher: null,
                author: null,
                startYear: null,
                endYear: null,
                minPrice: null,
                maxPrice: null,
                sortField: 'BOOK_ID',
                sortOrder: 'ASC',
                pageNum: 1,
                pageSize: 10
            }
            this.initialize = true // 设置初始化为true
            this.QueryBooks()
        },

        // 分页大小变化
        handleSizeChange(val) {
            this.queryParams.pageSize = val
            this.queryParams.pageNum = 1 // 重置当前页为1
            if(this.initialize) {
                this.QueryBooks()
            } else {
                this.handleQuery()
            }
        },

        // 当前页变化
        handleCurrentChange(val) {
            this.queryParams.pageNum = val
            if(this.initialize) {
                this.QueryBooks()
            } else {
                this.handleQuery()
            }
        },


        ConfirmNewBook() {
            axios.post('/book/add', this.newBookInfo)
                .then(response => {
                    ElMessage.success("图书入库成功")
                    this.newBookVisible = false
                    this.QueryBooks()
                })
                .catch(error => {
                    ElMessage.error("入库失败: " + error.message)
                })
        },
        ConfirmModifyBook() {
            axios.post('/book/update', this.toModifyInfo)
                .then(response => {
                    ElMessage.success("图书信息修改成功")
                    this.modifyBookVisible = false
```

```
                this.QueryBooks()
            })
            .catch(error => {
                ElMessage.error("修改失败: " + error.message)
            })
    },
    ConfirmAddStock() {
        axios.post('/book/addstock', {
            id: this.toAddStock,
            amount: this.addStockAmount
        })
        .then(response => {
            ElMessage.success(`成功增加 ${this.addStockAmount} 本库存`)
            this.addStockVisible = false
            this.QueryBooks()
        })
        .catch(error => {
            ElMessage.error("增加库存失败: " + error.message)
        })
    },
    ConfirmRemoveBook() {
        axios.post('/book/delete', { id: this.toRemove })
            .then(response => {
                ElMessage.success("图书删除成功")
                this.removeBookVisible = false
                this.QueryBooks()
            })
            .catch(error => {
                ElMessage.error("删除失败: " + error.message)
            })
    },
    ConfirmBorrowBook() {
        axios.post('/book/borrow', {
            bookId: this.toBorrow,
            cardId: this.borrowCardId,
            borrowTime: new Date().toISOString()
        })
        .then(response => {
            ElMessage.success("借书成功")
            this.borrowBookVisible = false
            this.QueryBooks()
        })
        .catch(error => {
            ElMessage.error("借书失败: " + error.message)
        })
    },
    ConfirmReturnBook() {
        axios.post('/book/return', {
            bookId: this.toReturn,
            cardId: this.returnCardId,
            returnTime: new Date().toISOString()
        })
        .then(response => {
            ElMessage.success("还书成功")
            this.returnBookVisible = false
            this.QueryBooks()
```

```
            })
            .catch(error => {
                ElMessage.error("还书失败: " + error.message)
            })
        },
        QueryBooks() {
            this.books = []
            axios.get('/book/QueryBooks')
                .then(response => {
                    const start = (this.queryParams.pageNum - 1) *
this.queryParams.pageSize
                    const end = start + this.queryParams.pageSize
                    this.books = response.data.results.slice(start, end) //
获取当前页的图书数据
                    this.total = response.data.count // 获取图书总数
                    if (this.books.length === 0) {
                        ElMessage.warning("没有图书可供显示")
                    }
                })
                .catch(error => {
                    ElMessage.error("获取图书列表失败: " + error.message)
                })
        },
        handleFileChange(file) {
            // 首先检查文件类型
        const isExcel = file.raw.type === 'application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet' ||
                        file.raw.type === 'application/vnd.ms-excel';

        if (!isExcel) {
            ElMessage.error('只能上传Excel文件!');
            this.$refs.uploadRef.clearFiles(); // 清除非Excel文件
            return false; // 停止处理
        }
            const reader = new FileReader();
            reader.onload = (e) => {
            const data = new Uint8Array(e.target.result);
            const workbook = XLSX.read(data, { type: 'array' });
            const firstSheet = workbook.Sheets[workbook.SheetNames[0]];
            const jsonData = XLSX.utils.sheet_to_json(firstSheet);
            console.log(jsonData); // 打印解析后的数据
            axios.post('/book/batch-import', jsonData)
                .then(response => {
                    ElMessage.success("批量入库成功")
                    this.batchImportVisible = false
                    this.QueryBooks()
                })
                .catch(error => {
                    this.batchImportVisible = false
                    ElMessage.error("批量入库失败: " + error.message)
                })
            }
            reader.readAsArrayBuffer(file.raw); // 读取文件为ArrayBuffer
        },
    },
    mounted() {
```

```
            this.QueryBooks()
        }
    }
    </script>

    <style scoped>
    .cardBox {
        height: 400px;
        width: 250px;
        box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px 20px 0 rgba(0, 0, 0,
    0.19);
        text-align: center;
        margin-top: 40px;
        margin-left: 27.5px;
        margin-right: 10px;
        padding: 7.5px;
        padding-right: 10px;
        padding-top: 15px;
    }

    .newCardBox {
        height: 400px;
        width: 250px;
        margin-top: 40px;
        margin-left: 27.5px;
        margin-right: 10px;
        padding: 7.5px;
        padding-right: 10px;
        padding-top: 15px;
        box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px 20px 0 rgba(0, 0, 0,
    0.19);
        text-align: center;
        display: flex;
        flex-direction: column;
        align-items: center;
        justify-content: center;
    }

    .query-form {
        background: #f5f7fa;
        padding: 20px;
        border-radius: 4px;
        margin-bottom: 20px;
    }

    .upload-demo {
        width: 100%;
    }
    </style>
```

- 后端

先在本地8000端口创建一个服务器,然后在`/card`,`/borrow`,`/book`分别绑定各自的handler,并将与数据库连接的library传进handler中,之后启动服务器。每个前端传送过来的请求都是以`/card`,`/borrow`,`/book`开头,根据这个将前端发送的请求交给各自的handler,handler之中再根据请求的方法,将不同请求进行区分,最后根据发送的请求的URL后面的东西来判断前端要进行的操作,然后去调用

之前基础部分实现的各种函数来完成对数据库的操作,并将前端所需要的数据传回去。

```
import utils.ConnectConfig;
import utils.DatabaseConnector;
import java.io.IOException;
import java.io.OutputStream;
import java.io.InputStream;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.InetSocketAddress;
import com.sun.net.httpserver.HttpServer;
import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpHandler;
import com.sun.net.httpserver.Headers;

import java.util.ArrayList;
import java.util.List;
import java.util.logging.Logger;
import java.time.Instant;
import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;

import org.json.JSONObject;

import com.google.gson.Gson;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import com.google.gson.JsonArray;

import entities.Card;
import entities.Book;
import entities.Borrow;

import queries.*;

public class Main {

    private static final Logger log = Logger.getLogger(Main.class.getName());
    private DatabaseConnector connector;
    private LibraryManagementSystem library;

    public void connect() {
        try {
            // parse connection config from "resources/application.yaml"
            ConnectConfig conf = new ConnectConfig();
            log.info("Success to parse connect config. " + conf.toString());
            // connect to database
            connector = new DatabaseConnector(conf);
            boolean connStatus = connector.connect();
            if (!connStatus) {
                log.severe("Failed to connect database.");
                System.exit(1);
            }
            library = new LibraryManagementSystemImpl(connector);
```

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

    public void release() {
        boolean releaseStatus = connector.release();
        if (releaseStatus) {
            System.out.println("Successfully release database connection.");
        } else {
            System.out.println("Failed to release database connection.");
        }
    }

    static class BookHandler implements HttpHandler {
        private final LibraryManagementSystem library;

        public BookHandler(LibraryManagementSystem library) {
            this.library = library;
        }

        @Override
        public void handle(HttpExchange exchange) throws IOException {
            Headers headers = exchange.getResponseHeaders();
            headers.add("Access-Control-Allow-Origin", "*");
            headers.add("Access-Control-Allow-Methods", "GET, POST, OPTIONS");
            headers.add("Access-Control-Allow-Headers", "Content-Type");
            String requestMethod = exchange.getRequestMethod();
            if (requestMethod.equals("GET")) {
                handleGetRequest(exchange);
            } else if (requestMethod.equals("POST")) {
                handlePostRequest(exchange);
            } else if (requestMethod.equals("OPTIONS")) {
                handleOptionsRequest(exchange);
            } else {
                exchange.sendResponseHeaders(405, -1);
            }
        }

        private void handleGetRequest(HttpExchange exchange) throws IOException {
            // 响应头，因为是JSON通信
            exchange.getResponseHeaders().set("Content-Type",
"application/json");
            // 获取输出流，java用流对象来进行io操作
            OutputStream outputStream = exchange.getResponseBody();
            String path = exchange.getRequestURI().getPath();
            System.out.println("Received GET request");
            try {
                if (path.equals("/book/QueryBooks")) {
                    ApiResult result = library.showBooks();
                    String response = new Gson().toJson(result.payload);
                    exchange.sendResponseHeaders(200,
response.getBytes().length);
                    outputStream.write(response.getBytes());
                    outputStream.close();
                } else {
```

```
                    exchange.sendResponseHeaders(404, 0);
                }
            } catch (Exception e) {
                e.printStackTrace();
                exchange.sendResponseHeaders(500, 0);
            }
        }

        private void handlePostRequest(HttpExchange exchange) throws IOException
{
            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);
            }
            exchange.getResponseHeaders().set("Content-Type", "text/plain");
            OutputStream outputStream = exchange.getResponseBody();
            String path = exchange.getRequestURI().getPath();
            System.out.println("Received POST request to create book with data: "
+ requestBodyBuilder.toString());
            try {
                if (path.equals("/book/handleQuery")) {
                    JsonObject json =
JsonParser.parseString(requestBodyBuilder.toString()).getAsJsonObject();
                    String category = !json.get("category").isJsonNull() ?
json.get("category").getAsString() : null;
                    String title = !json.get("title").isJsonNull() ?
json.get("title").getAsString() : null;
                    String press = !json.get("publisher").isJsonNull() ?
json.get("publisher").getAsString() : null;
                    Integer minPublishYear = !json.get("startYear").isJsonNull()
? json.get("startYear").getAsInt()
                            : null;
                    Integer maxPublishYear = !json.get("endYear").isJsonNull() ?
json.get("endYear").getAsInt() : null;
                    String author = !json.get("author").isJsonNull() ?
json.get("author").getAsString() : null;
                    Double minPrice = !json.get("minPrice").isJsonNull() ?
json.get("minPrice").getAsDouble() : null;
                    Double maxPrice = !json.get("maxPrice").isJsonNull() ?
json.get("maxPrice").getAsDouble() : null;
                    Book.SortColumn sortBy =
Book.SortColumn.valueOf(json.get("sortField").getAsString()) == null
                            ? Book.SortColumn.BOOK_ID
                            :
Book.SortColumn.valueOf(json.get("sortField").getAsString());
                    String sortOrder = json.get("sortOrder").getAsString();
                    BookQueryConditions conditions = new BookQueryConditions();
                    conditions.setCategory(category);
                    conditions.setTitle(title);
                    conditions.setPress(press);
                    conditions.setMinPublishYear(minPublishYear);
                    conditions.setMaxPublishYear(maxPublishYear);
```

```
                    conditions.setAuthor(author);
                    conditions.setMinPrice(minPrice);
                    conditions.setMaxPrice(maxPrice);
                    conditions.setSortBy(sortBy);
                    conditions.setSortOrder(SortOrder.valueOf(sortOrder));
                    ApiResult result = library.queryBook(conditions);
                    String response = new Gson().toJson(result.payload);
                    exchange.sendResponseHeaders(200,
response.getBytes().length);
                    outputStream.write(response.getBytes());
                    outputStream.close();
                } else if (path.equals("/book/add")) {
                    JsonObject json =
JsonParser.parseString(requestBodyBuilder.toString()).getAsJsonObject();
                    String category = json.get("category").getAsString();
                    String title = json.get("title").getAsString();
                    String press = json.get("publisher").getAsString();
                    int publishYear = json.get("year").getAsInt();
                    String author = json.get("author").getAsString();
                    double price = json.get("price").getAsDouble();
                    int stock = json.get("stock").getAsInt();
                    Book book = new Book(category, title, press, publishYear,
author, price, stock);
                    ApiResult result = library.storeBook(book);
                    if (result.ok) {
                        exchange.sendResponseHeaders(200, 0);
                    } else {
                        exchange.sendResponseHeaders(404, 0);
                    }
                    outputStream.close();
                } else if (path.equals("/book/update")) {
                    JsonObject json =
JsonParser.parseString(requestBodyBuilder.toString()).getAsJsonObject();
                    int bookId = json.get("id").getAsInt();
                    String category = json.get("category").getAsString();
                    String title = json.get("title").getAsString();
                    String press = json.get("publisher").getAsString();
                    int publishYear = json.get("year").getAsInt();
                    String author = json.get("author").getAsString();
                    double price = json.get("price").getAsDouble();
                    Book book = new Book(category, title, press, publishYear,
author, price, 0);
                    book.setBookId(bookId);
                    ApiResult result = library.modifyBookInfo(book);
                    if (result.ok) {
                        exchange.sendResponseHeaders(200, 0);
                    } else {
                        exchange.sendResponseHeaders(404, 0);
                    }
                    outputStream.close();
                } else if (path.equals("/book/addstock")) {
                    JsonObject json =
JsonParser.parseString(requestBodyBuilder.toString()).getAsJsonObject();
                    int bookId = json.get("id").getAsInt();
                    int stock = json.get("amount").getAsInt();
                    ApiResult result = library.incBookStock(bookId, stock);
```

```
                    if (result.ok) {
                        exchange.sendResponseHeaders(200, 0);
                    } else {
                        exchange.sendResponseHeaders(404, 0);
                    }
                    outputStream.close();
                } else if (path.equals("/book/delete")) {
                    JsonObject json =
JsonParser.parseString(requestBodyBuilder.toString()).getAsJsonObject();
                    int bookId = json.get("id").getAsInt();
                    ApiResult result = library.removeBook(bookId);
                    if (result.ok) {
                        exchange.sendResponseHeaders(200, 0);
                    } else {
                        exchange.sendResponseHeaders(404, 0);
                    }
                    outputStream.close();
                } else if (path.equals("/book/borrow")) {
                    JsonObject json =
JsonParser.parseString(requestBodyBuilder.toString()).getAsJsonObject();
                    int bookId = json.get("bookId").getAsInt();
                    int cardId = json.get("cardId").getAsInt();
                    String borrowTimeStr = json.get("borrowTime").getAsString();
                    ZonedDateTime borrowDateTime =
ZonedDateTime.parse(borrowTimeStr,
                            DateTimeFormatter.ISO_ZONED_DATE_TIME);
                    long borrowTime = borrowDateTime.toInstant().toEpochMilli();
                    Borrow borrow = new Borrow(bookId, cardId);
                    borrow.setBorrowTime(borrowTime);
                    borrow.setReturnTime(0);
                    ApiResult result = library.borrowBook(borrow);
                    if (result.ok) {
                        exchange.sendResponseHeaders(200, 0);
                    } else {
                        exchange.sendResponseHeaders(404, 0);
                    }
                    outputStream.close();
                } else if (path.equals("/book/return")) {
                    JsonObject json =
JsonParser.parseString(requestBodyBuilder.toString()).getAsJsonObject();
                    int bookId = json.get("bookId").getAsInt();
                    int cardId = json.get("cardId").getAsInt();
                    String returnTimeStr = json.get("returnTime").getAsString();
                    ZonedDateTime returnDateTime =
ZonedDateTime.parse(returnTimeStr,
                            DateTimeFormatter.ISO_ZONED_DATE_TIME);
                    long returnTime = returnDateTime.toInstant().toEpochMilli();
                    Borrow borrow = new Borrow(bookId, cardId);
                    borrow.setReturnTime(returnTime);
                    ApiResult result = library.returnBook(borrow);
                    if (result.ok) {
                        exchange.sendResponseHeaders(200, 0);
                    } else {
                        exchange.sendResponseHeaders(404, 0);
                    }
                    outputStream.close();
```

```
                } else if (path.equals("/book/batch-import")) {
                    JsonArray jsonArray =
JsonParser.parseString(requestBodyBuilder.toString()).getAsJsonArray();
                    List<Book> books = new ArrayList<>();
                    for (int i = 0; i < jsonArray.size(); i++) {
                        JsonObject json = jsonArray.get(i).getAsJsonObject();
                        String category = json.get("类别").getAsString();
                        String title = json.get("书名").getAsString();
                        String press = json.get("出版社").getAsString();
                        int publishYear = json.get("年份").getAsInt();
                        String author = json.get("作者").getAsString();
                        double price = json.get("价格").getAsDouble();
                        int stock = json.get("初始库存").getAsInt();
                        Book book = new Book(category, title, press, publishYear,
author, price, stock);
                        books.add(book);
                    }
                    ApiResult result = library.storeBook(books);
                    if (result.ok) {
                        exchange.sendResponseHeaders(200, 0);
                    } else {
                        exchange.sendResponseHeaders(404, 0);
                    }
                    outputStream.close();
                } else {
                    exchange.sendResponseHeaders(404, 0);
                }
            } catch (Exception e) {
                e.printStackTrace();
                exchange.sendResponseHeaders(500, 0);
            }
        }

        private void handleOptionsRequest(HttpExchange exchange) throws
IOException {
            exchange.sendResponseHeaders(204, -1);
        }
    }

    static class CardHandler implements HttpHandler {
        private final LibraryManagementSystem library;

        public CardHandler(LibraryManagementSystem library) {
            this.library = library;
        }

        @Override
        public void handle(HttpExchange exchange) throws IOException {
            Headers headers = exchange.getResponseHeaders();
            headers.add("Access-Control-Allow-Origin", "*");
            headers.add("Access-Control-Allow-Methods", "GET, POST, OPTIONS");
            headers.add("Access-Control-Allow-Headers", "Content-Type");
            String requestMethod = exchange.getRequestMethod();
            if (requestMethod.equals("GET")) {
                System.out.println("Received GET request");
```

```
                handleGetRequest(exchange);
            } else if (requestMethod.equals("POST")) {
                System.out.println("Received POST request");
                handlePostRequest(exchange);
            } else if (requestMethod.equals("OPTIONS")) {
                System.out.println("Received OPTIONS request");
                handleOptionsRequest(exchange);
            } else {
                System.out.println("Received invalid request");
                exchange.sendResponseHeaders(405, -1);
            }
        }

        private void handleGetRequest(HttpExchange exchange) throws IOException {
            // 响应头，因为是JSON通信
            exchange.getResponseHeaders().set("Content-Type",
"application/json");
            // 获取输出流，java用流对象来进行io操作
            OutputStream outputStream = exchange.getResponseBody();
            String path = exchange.getRequestURI().getPath();
            try {
                if (path.equals("/card/QueryCards")) {
                    ApiResult result = library.showCards();
                    String response = new Gson().toJson(result.payload);
                    exchange.sendResponseHeaders(200,
response.getBytes().length);
                    outputStream.write(response.getBytes());
                    outputStream.close();
                } else {
                    exchange.sendResponseHeaders(404, 0);
                }
            } catch (Exception e) {
                e.printStackTrace();
                exchange.sendResponseHeaders(500, 0);
            }
        }

        private void handlePostRequest(HttpExchange exchange) throws IOException
{
            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);
            }
            exchange.getResponseHeaders().set("Content-Type", "text/plain");
            OutputStream outputStream = exchange.getResponseBody();
            String path = exchange.getRequestURI().getPath();
            System.out.println("Received POST request to create card with data: "
+ requestBodyBuilder.toString());
            try {
                if (path.equals("/card/ConfirmNewCard")) {
                    JsonObject json =
JsonParser.parseString(requestBodyBuilder.toString()).getAsJsonObject();
```

```
                    String name = json.get("name").getAsString();
                    String department = json.get("department").getAsString();
                    String type = json.get("type").getAsString();
                    Card card = new Card();
                    card.setName(name);
                    card.setDepartment(department);
                    card.setType(Card.CardType.valueOf(type));
                    ApiResult result = library.registerCard(card);
                    exchange.sendResponseHeaders(200, 0);
                    outputStream.close();
                } else if (path.equals("/card/ConfirmRemoveCard")) {
                    JsonObject json =
JsonParser.parseString(requestBodyBuilder.toString()).getAsJsonObject();
                    int cardId = json.get("id").getAsInt();
                    ApiResult result = library.removeCard(cardId);
                    exchange.sendResponseHeaders(200, 0);
                    outputStream.close();
                } else if (path.equals("/card/ConfirmModifyCard")) {
                    JsonObject json =
JsonParser.parseString(requestBodyBuilder.toString()).getAsJsonObject();
                    int cardId = json.get("id").getAsInt();
                    String name = json.get("name").getAsString();
                    String department = json.get("department").getAsString();
                    String type = json.get("type").getAsString();
                    Card card = new Card(cardId, name, department,
Card.CardType.valueOf(type));
                    ApiResult result = library.modifyCard(card);
                    exchange.sendResponseHeaders(200, 0);
                    outputStream.close();
                } else {
                    exchange.sendResponseHeaders(404, 0);
                }
            } catch (Exception e) {
                e.printStackTrace();
                exchange.sendResponseHeaders(500, 0);
            }
        }

        private void handleOptionsRequest(HttpExchange exchange) throws
IOException {
            exchange.sendResponseHeaders(204, -1);
        }
    }

    static class BorrowHandler implements HttpHandler {
        private final LibraryManagementSystem library;

        public BorrowHandler(LibraryManagementSystem library) {
            this.library = library;
        }

        @Override
        public void handle(HttpExchange exchange) throws IOException {
            Headers headers = exchange.getResponseHeaders();
            headers.add("Access-Control-Allow-Origin", "*");
            headers.add("Access-Control-Allow-Methods", "GET, POST, OPTIONS");
```

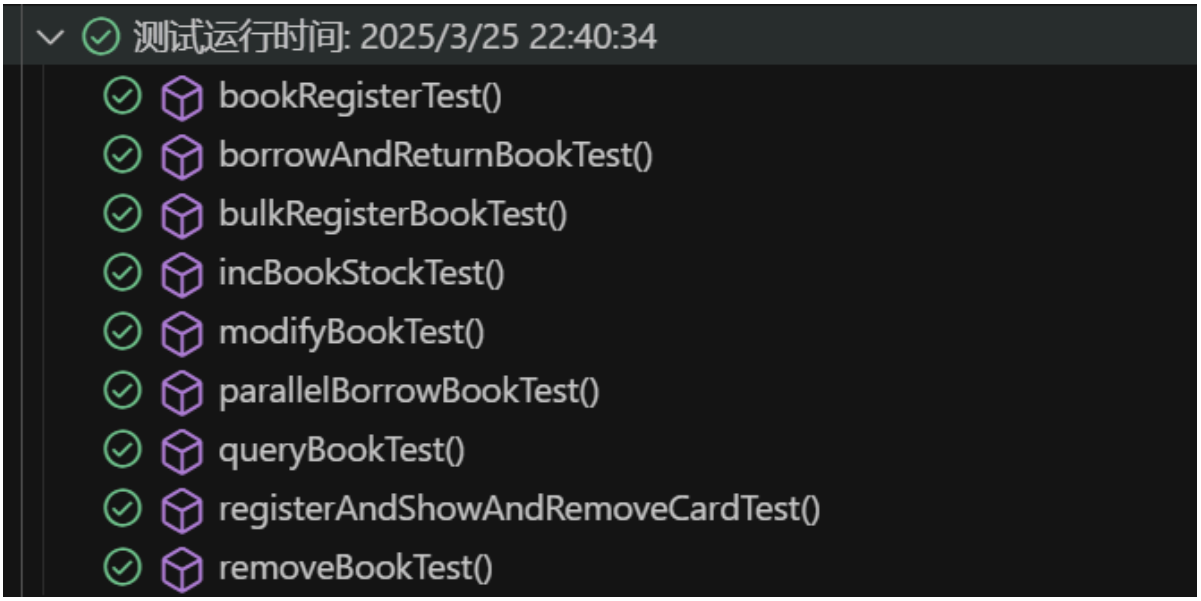
```
            headers.add("Access-Control-Allow-Headers", "Content-Type");
            String requestMethod = exchange.getRequestMethod();
            if (requestMethod.equals("GET")) {
                System.out.println("Received GET request");
                handleGetRequest(exchange);
            } else if (requestMethod.equals("POST")) {
                System.out.println("Received POST request");
                handlePostRequest(exchange);
            } else if (requestMethod.equals("OPTIONS")) {
                System.out.println("Received OPTIONS request");
                handleOptionsRequest(exchange);
            } else {
                System.out.println("Received invalid request");
                exchange.sendResponseHeaders(405, -1);
            }
        }

        private void handleGetRequest(HttpExchange exchange) throws IOException {
            // 响应头，因为是JSON通信
            exchange.getResponseHeaders().set("Content-Type",
"application/json");
            // 获取输出流，java用流对象来进行io操作
            OutputStream outputStream = exchange.getResponseBody();
            String query = exchange.getRequestURI().getQuery();
            String querys = query.split("=")[1];
            try {
                if (querys.matches("\\d+")) {
                    int cardId = Integer.parseInt(querys);
                    ApiResult result = library.showBorrowHistory(cardId);
                    String response = new Gson().toJson(result.payload);
                    exchange.sendResponseHeaders(200,
response.getBytes().length);
                    outputStream.write(response.getBytes());
                    outputStream.close();
                } else {
                    exchange.sendResponseHeaders(404, 0);
                }
            } catch (Exception e) {
                e.printStackTrace();
                exchange.sendResponseHeaders(500, 0);
            }
        }

        private void handlePostRequest(HttpExchange exchange) throws IOException
{
            exchange.sendResponseHeaders(200, 0);
        }

        private void handleOptionsRequest(HttpExchange exchange) throws
IOException {
            exchange.sendResponseHeaders(204, -1);
        }
    }

    public static void main(String[] args) throws IOException {
```

```
        Main mainInstance = new Main();
        mainInstance.connect();

        HttpServer server = HttpServer.create(new InetSocketAddress(8000), 0);

        server.createContext("/card", new CardHandler(mainInstance.library));
        server.createContext("/book", new BookHandler(mainInstance.library));
        server.createContext("/borrow", new BorrowHandler(mainInstance.library));
        // 启动服务器
        server.start();

        // 标识一下，这样才知道我的后端启动了（确信
        System.out.println("Server is listening on port 8000");

        while (System.in.read() != 'q') {
            continue;
        }

        mainInstance.release();
    }

}
```

五、系统验证测试

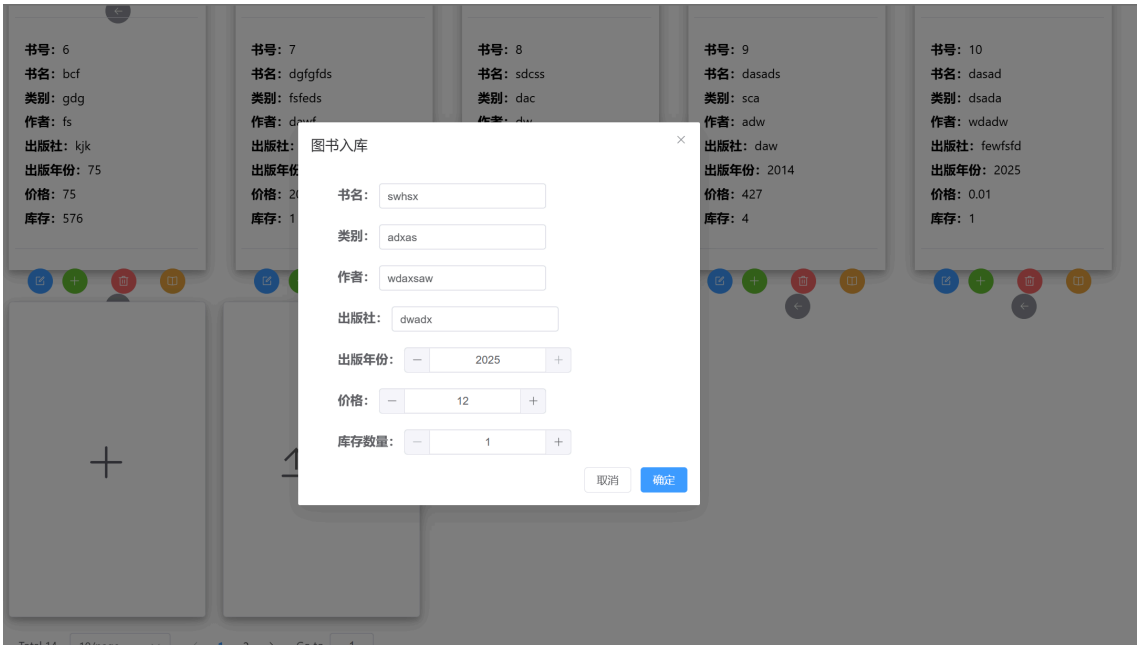
- 正确性测试

```
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 72.23 s -- in LibraryTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:16 min
[INFO] Finished at: 2025-04-02T17:40:39+08:00
[INFO] ------------------------------------------------------------------------
PS D:\code\c\database\librarymanagementsystem-master>
```



- 功能性测试

- 图书入库


图书入库
书名：swhsx
类别：adxas
作者：wdaxsaw
出版社：dwadx
出版年份：2025
价格：12
库存数量：1
取消
确定


bcf
书号：6
书名：bcf
类别：gdg
作者：fs
出版社：kjk
出版年份：75
价格：75
库存：576
dgfgfds
书号：7
书名：dgfgfds
类别：fsfeds
作者：dawf
出版社：sgsdf
出版年份：2025
价格：20
库存：1
sdcss
书号：8
书名：sdcss
类别：dac
作者：dw
出版社：daw
出版年份：47
价格：42
库存：75
dasads
书号：9
书名：dasads
类别：sca
作者：adw
出版社：daw
出版年份：2014
价格：427
库存：4
dasad
书号：10
书名：dasad
类别：dsada
作者：wdadw
出版社：fewfsfd
出版年份：2025
价格：0.01
库存：1
fsd
书号：11
书名：fsd
类别：gdd
作者：海明威
出版社：as
出版年份：1258
价格：12
库存：25
fsdvx
书号：12
书名：fsdvx
类别：ghj
作者：gjh
出版社：assa
出版年份：14
价格：14
库存：64
vxdgv
书号：13
书名：vxdgv
类别：hk
作者：gfd
出版社：qssq
出版年份：75
价格：867
库存：57
bcf
书号：14
书名：bcf
类别：gdg
作者：fs
出版社：sqxa
出版年份：75
价格：75
库存：576
swhsx
书号：15
书名：swhsx
类别：adxas
作者：wdaxsaw
出版社：dwadx
出版年份：2025
价格：12
库存：1

- 增加库存


Database System Concepts
书号：1
书名：Database System Concepts
类别：Nature
作者：Yuuku
出版社：Press-B
出版年份：2009
价格：78.85
库存：0
fdsfds
出版年份：2025
价格：20
库存：0
fsd
出版年份：1258
价格：12
库存：25
fsdvx
书号：4
书名：fsdvx
类别：ghj
作者：gjh
出版社：;k;
出版年份：14
价格：14
库存：64
vxdgv
书号：5
书名：vxdgv
类别：hk
作者：gfd
出版社：dfsv
出版年份：75
价格：867
库存：57
增加库存(书号: 1)
增加数量：12
取消
确定

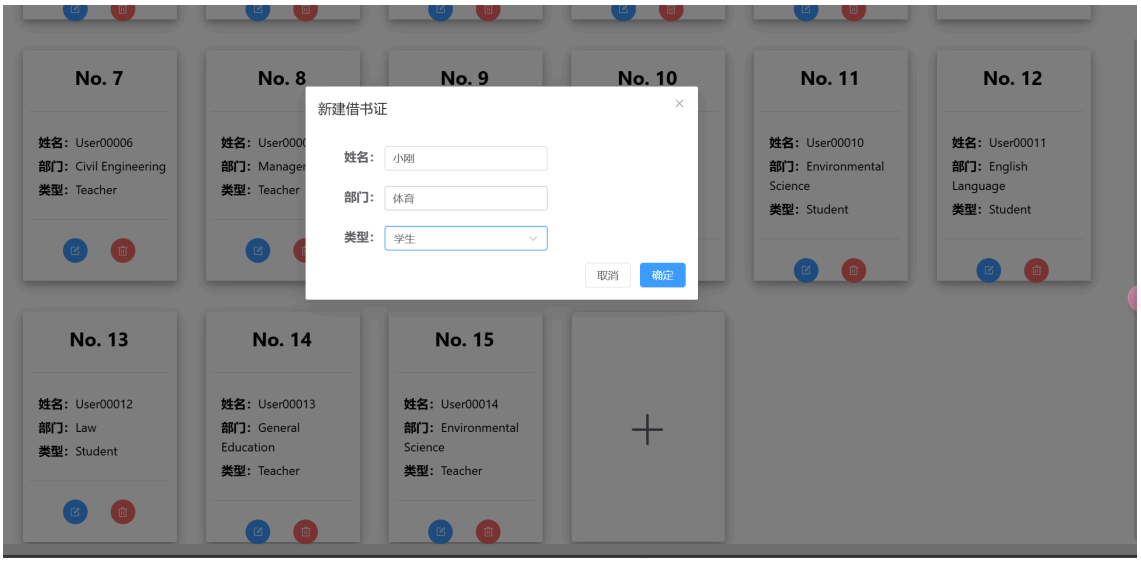
- 修改图书信息

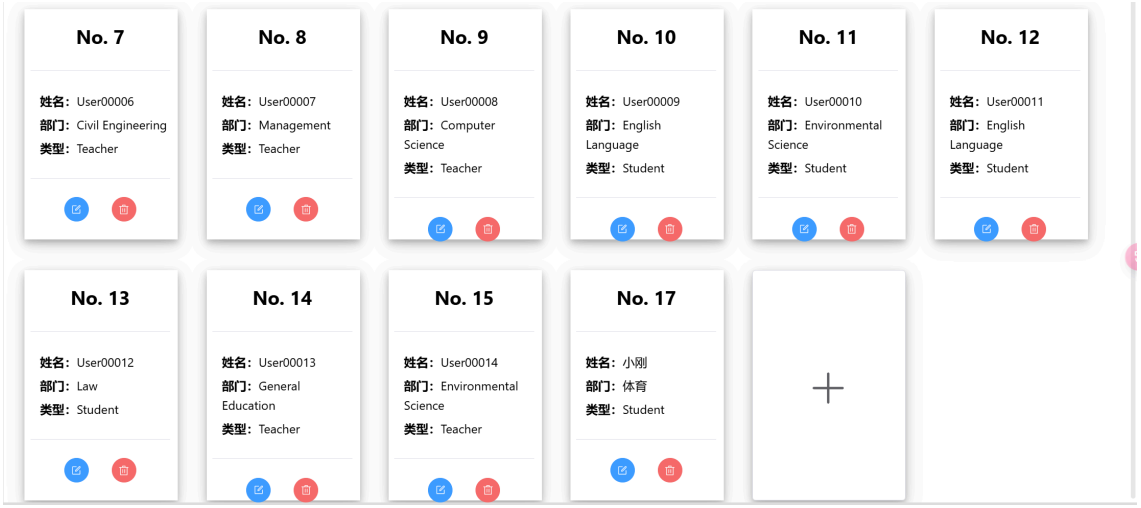
- 批量入库

书名	作者	出版社	年份	价格	类别	初始库存
sadsaas	das	daw	2018	42	dsf	58
dsac	cd	daw	2014	28	sda	14
dcsc	sdc	fdsd	2016	25	dbf	41
jgjhg	dgdf	grdg	2017	35	dfb	23


书号：11
书名：fsd
类别：gdd
作者：海明威
出版社：as
出版年份：1258
价格：12
库存：25
书号：12
书名：fsdvx
类别：ghj
作者：gjh
出版社：assa
出版年份：14
价格：14
库存：64
书号：13
书名：vxdgv
类别：hk
作者：gfd
出版社：qssq
出版年份：75
价格：867
库存：57
书号：14
书名：bcf
类别：gdg
作者：fs
出版社：sqxa
出版年份：75
价格：75
库存：576
书号：15
书名：swhsx
类别：adxas
作者：wdaxsaw
出版社：dwadx
出版年份：2025
价格：12
库存：1
sadsaas
书号：16
书名：sadsaas
类别：dsf
作者：das
出版社：daw
出版年份：2018
价格：42
库存：58
dsac
书号：17
书名：dsac
类别：sda
作者：cd
出版社：daw
出版年份：2014
价格：28
库存：14
dcsc
书号：18
书名：dcsc
类别：dbf
作者：sdc
出版社：fdsd
出版年份：2016
价格：25
库存：41
jgjhg
书号：19
书名：jgjhg
类别：dfb
作者：dgdf
出版社：grdg
出版年份：2017
价格：35
库存：23

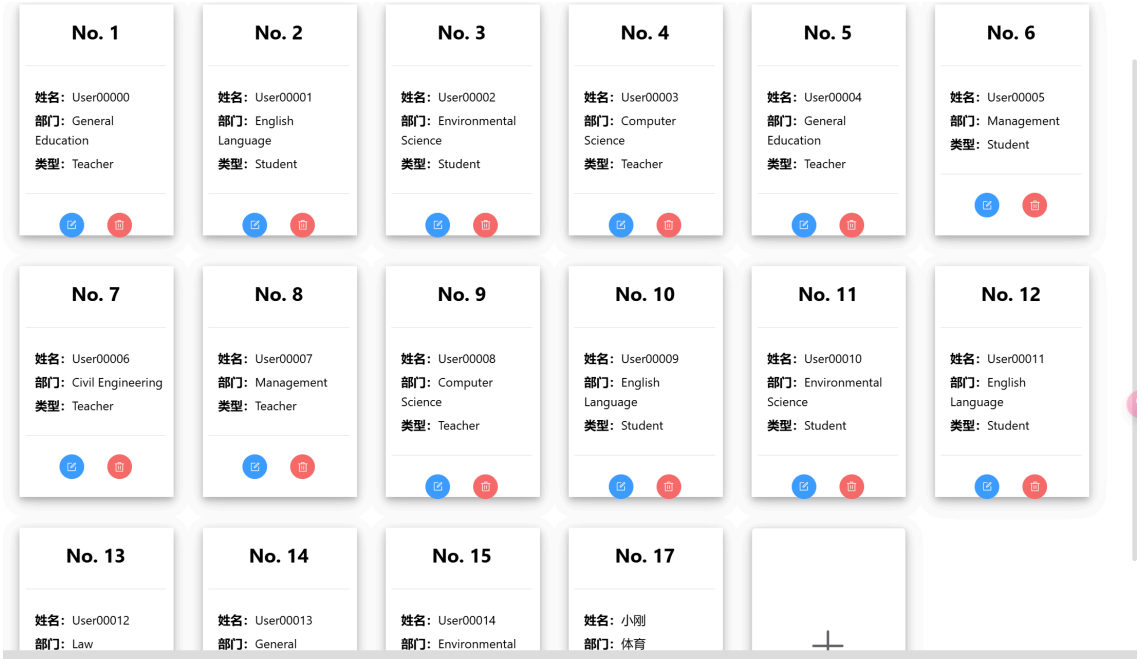
- 添加借书证


No. 7
姓名：User00006
部门：Civil Engineering
类型：Teacher
No. 8
No. 9
No. 10
No. 11
姓名：User00010
部门：Environmental Science
类型：Student
No. 12
姓名：User00011
部门：English Language
类型：Student
新建借书证
姓名：小刚
部门：体育
类型：学生
取消
确定
No. 13
姓名：User00012
部门：Law
类型：Student
No. 14
姓名：User00013
部门：General Education
类型：Teacher
No. 15
姓名：User00014
部门：Environmental Science
类型：Teacher

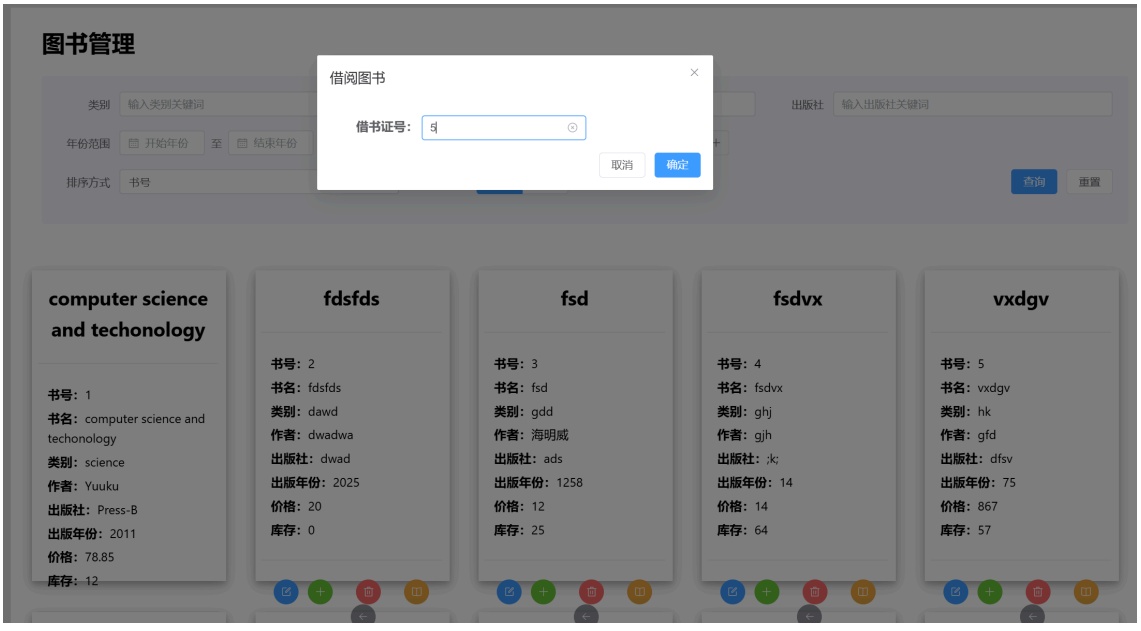

No. 7
姓名：User00006
部门：Civil Engineering
类型：Teacher
No. 8
姓名：User00007
部门：Management
类型：Teacher
No. 9
姓名：User00008
部门：Computer Science
类型：Teacher
No. 10
姓名：User00009
部门：English Language
类型：Student
No. 11
姓名：User00010
部门：Environmental Science
类型：Student
No. 12
姓名：User00011
部门：English Language
类型：Student
No. 13
姓名：User00012
部门：Law
类型：Student
No. 14
姓名：User00013
部门：General Education
类型：Teacher
No. 15
姓名：User00014
部门：Environmental Science
类型：Teacher
No. 17
姓名：小刚
部门：体育
类型：Student

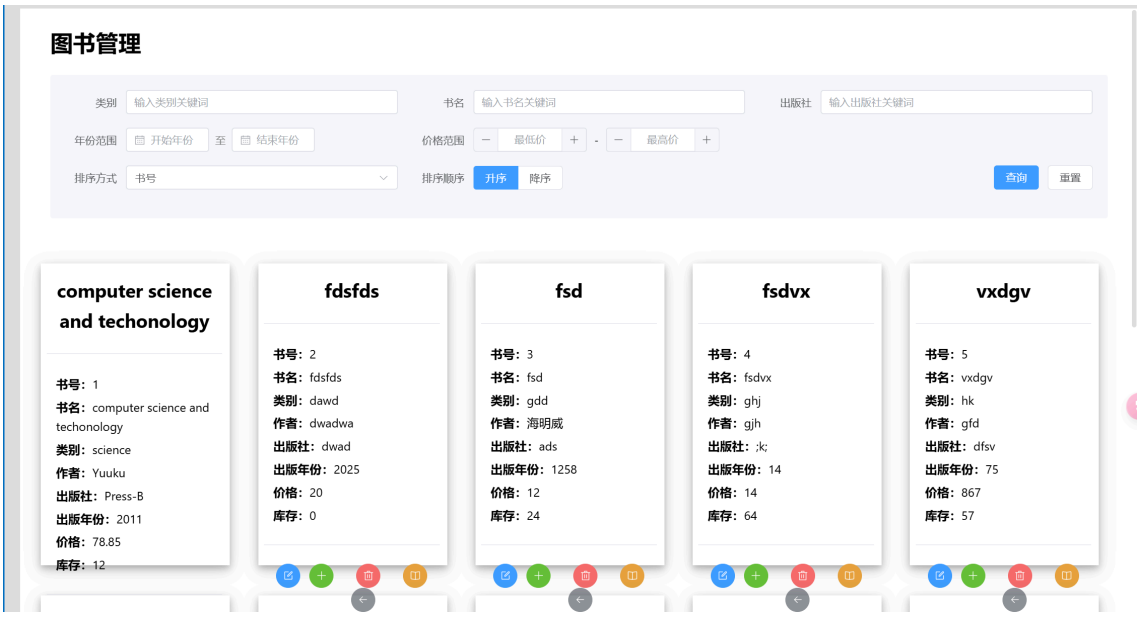
- 查询借书证

```
mysql> select * from Card;
+---------+-----------+-----------------------+------+
| card_id | name      | department            | type |
+---------+-----------+-----------------------+------+
|       7 | User00006 | Civil Engineering     | T    |
|       4 | User00003 | Computer Science      | T    |
|       9 | User00008 | Computer Science      | T    |
|       2 | User00001 | English Language      | S    |
|      10 | User00009 | English Language      | S    |
|      12 | User00011 | English Language      | S    |
|       3 | User00002 | Environmental Science | S    |
|      11 | User00010 | Environmental Science | S    |
|      15 | User00014 | Environmental Science | T    |
|       1 | User00000 | General Education     | T    |
|       5 | User00004 | General Education     | T    |
|      14 | User00013 | General Education     | T    |
|      13 | User00012 | Law                   | S    |
|       6 | User00005 | Management            | S    |
|       8 | User00007 | Management            | T    |
|      17 | 小刚      | 体育                  | S    |
+---------+-----------+-----------------------+------+
16 rows in set (0.00 sec)
```



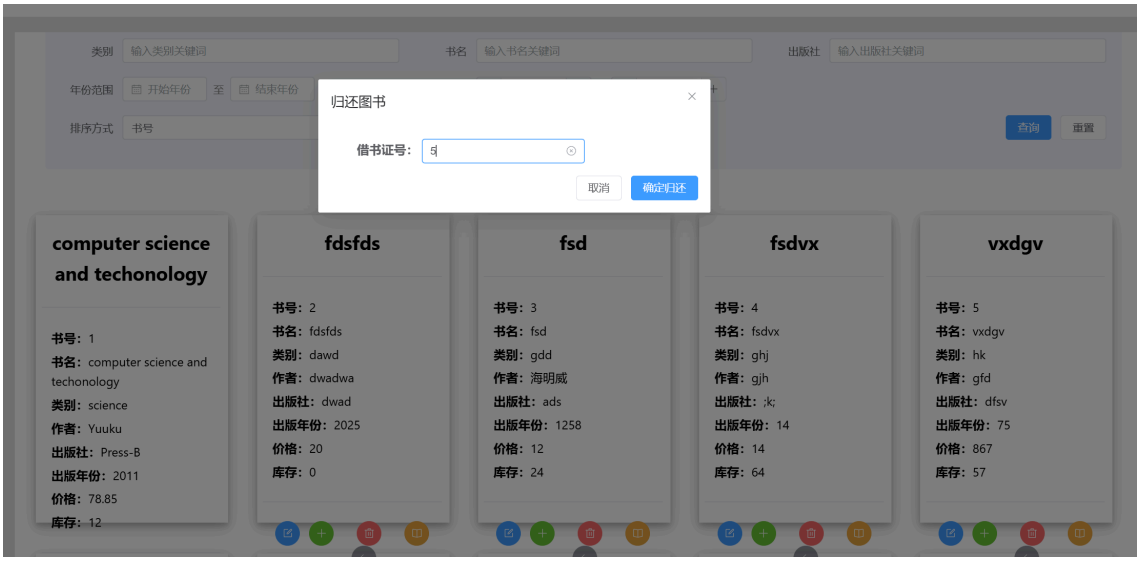
- 借书

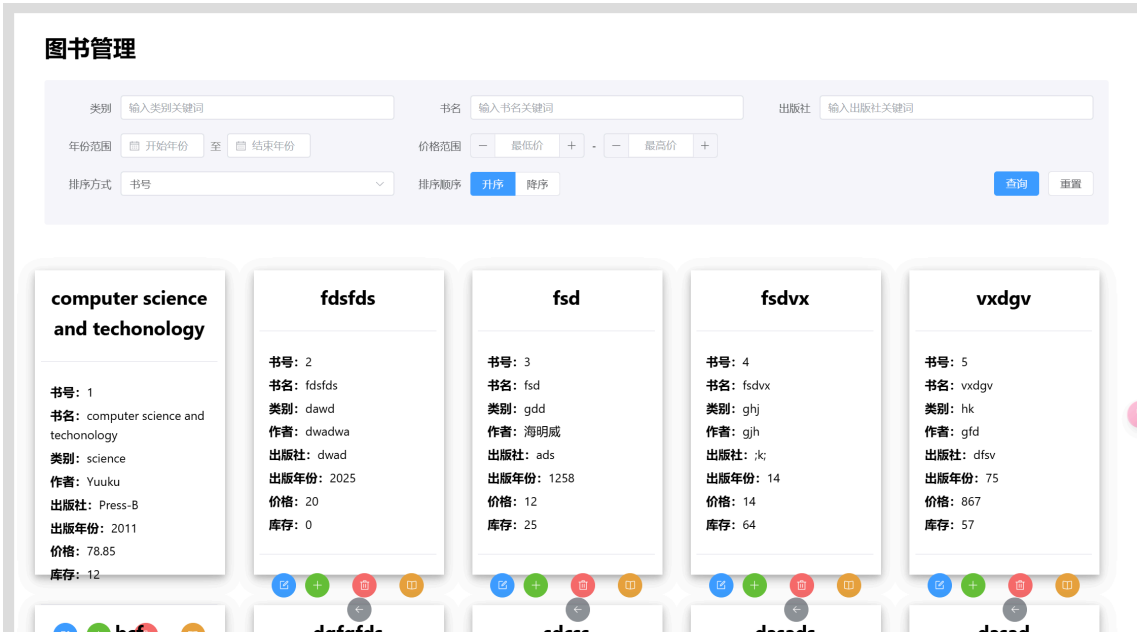



图书管理
类别
输入类别关键词
书名
输入书名关键词
出版社
输入出版社关键词
年份范围
开始年份
至
结束年份
价格范围
最低价
最高价
排序方式
书号
排序顺序
升序
降序
查询
重置
computer science and techonology
书号：1
书名：computer science and techonology
类别：science
作者：Yuuku
出版社：Press-B
出版年份：2011
价格：78.85
库存：12
fdsfds
书号：2
书名：fdsfds
类别：dawd
作者：dwadwa
出版社：dwad
出版年份：2025
价格：20
库存：0
fsd
书号：3
书名：fsd
类别：gdd
作者：海明威
出版社：ads
出版年份：1258
价格：12
库存：24
fsdvx
书号：4
书名：fsdvx
类别：ghj
作者：gjh
出版社：;k;
出版年份：14
价格：14
库存：64
vxdgv
书号：5
书名：vxdgv
类别：hk
作者：gfd
出版社：dfsv
出版年份：75
价格：867
库存：57

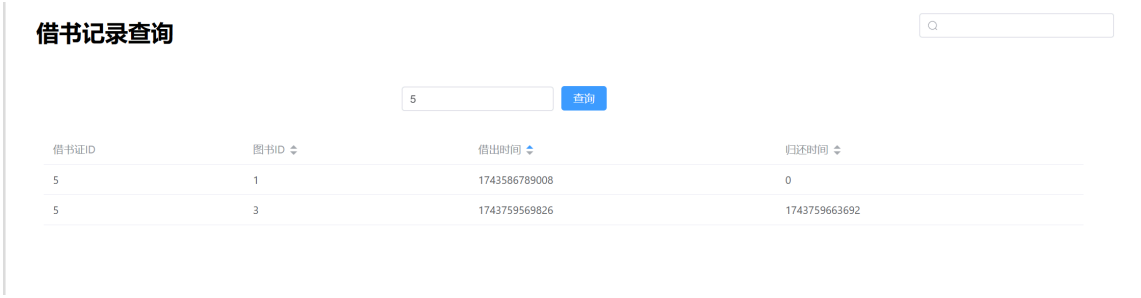

借书记录查询
5
查询
借书证ID
图书ID
借出时间
归还时间
5
1
1743586789008
0
5
3
1743759569826
0

- 还书

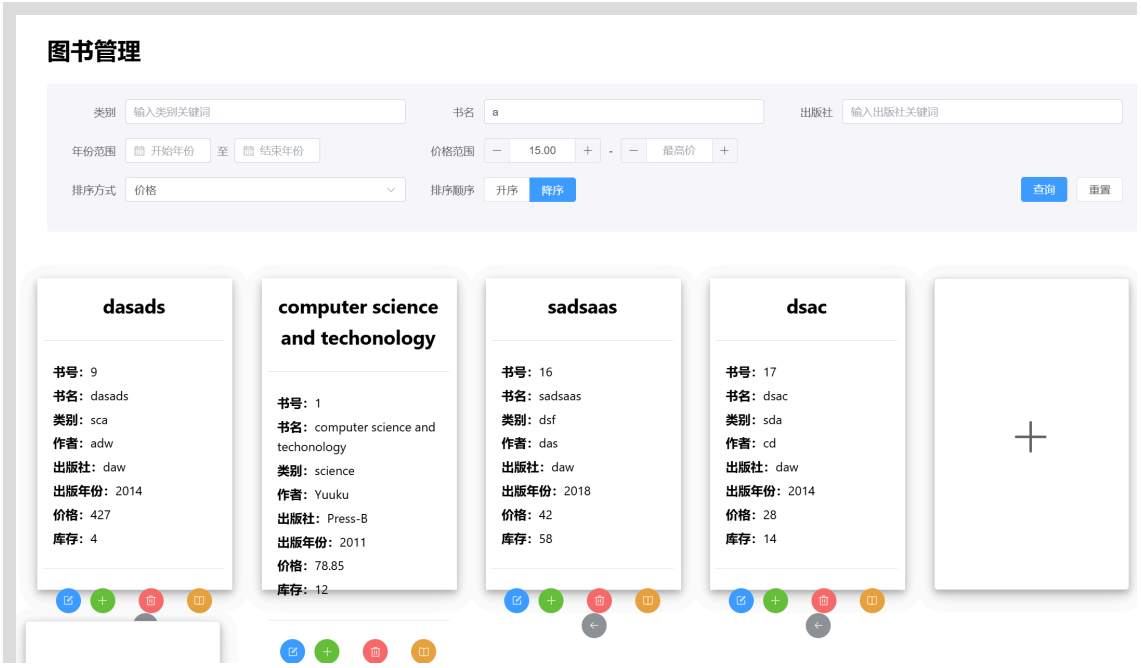

归还图书
借书证号：
5
取消
确定归还
computer science and techonology
书号：1
书名：computer science and techonology
类别：science
作者：Yuuku
出版社：Press-B
出版年份：2011
价格：78.85
库存：12
fdsfds
书号：2
书名：fdsfds
类别：dawd
作者：dwadwa
出版社：dwad
出版年份：2025
价格：20
库存：0
fsd
书号：3
书名：fsd
类别：gdd
作者：海明威
出版社：ads
出版年份：1258
价格：12
库存：24
fsdvx
书号：4
书名：fsdvx
类别：ghj
作者：gjh
出版社：;k;
出版年份：14
价格：14
库存：64
vxdgv
书号：5
书名：vxdgv
类别：hk
作者：gfd
出版社：dfsv
出版年份：75
价格：867
库存：57



- 借书记录查询



- 图书查询

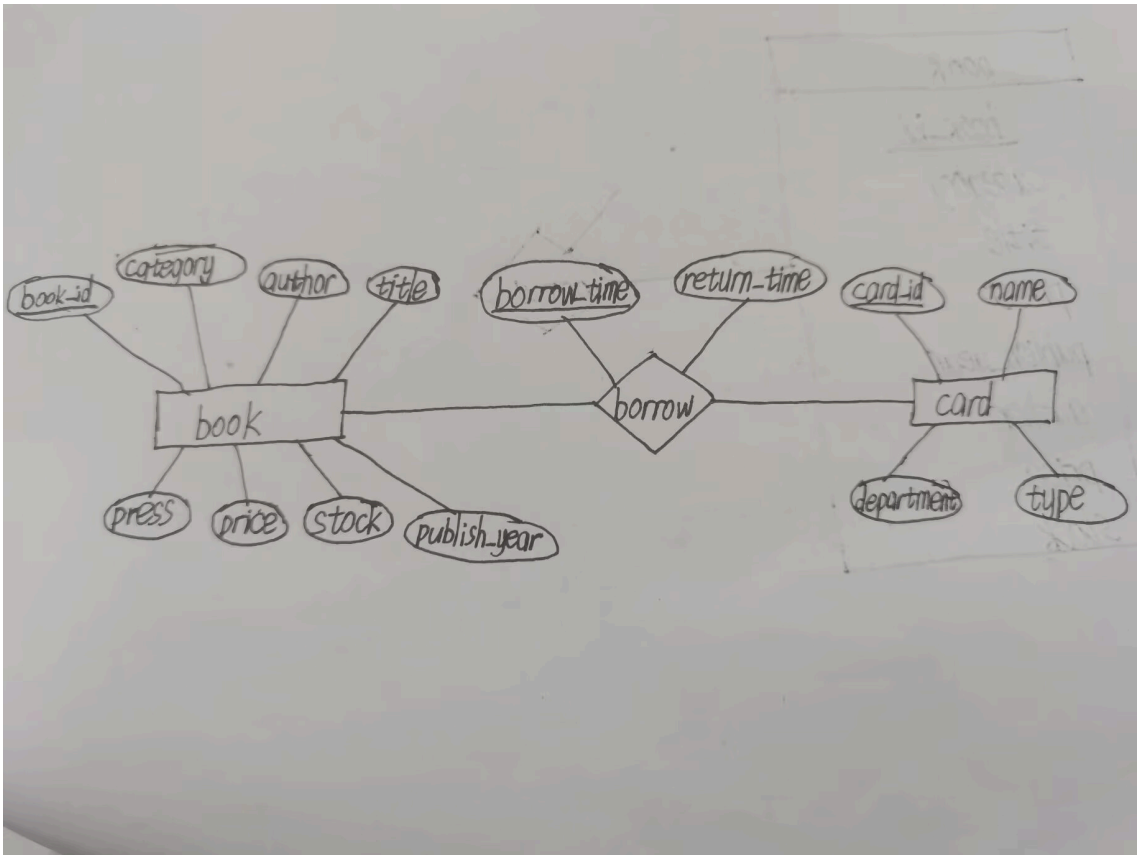


六、遇到的问题及解决方法

- 前端实现批量入库的时候,成功上传一个文件之后,再上传新的文件后前端没有向后端发送任何请求
  - 发现是没有读取文件,加上`reader.readAsArrayBuffer(file.raw);`后就可以了
- 基础部分实现批量入库的时候,在入库图书前要判断所有图书是否存在数据库中,无法正确执行查询语句

    - 发现是没有将前一个查询语句中的参数清空,加上`checkStmt.clearParameters();`将查询语句中的参数清空就可以了
- 基础部分并行借书测试无法通过,查看输出发现有时候查询语句出现在更新语句之前,导致查询出错
    - 在查询语句后面加上`FOR UPDATE`,这样就能保证查询发生在更新之后,查询结果正确
- 基础部分借书还书测试无法通过,查看报错的地方发现测试中有还书时间早于借书时间的情况
    - 在还书部分添加如果还书时间早于借书时间,则还书失败,便通过测试
- 实现后端的时候,前端往后端传空值会导致查询语句出错,比如传给后端`id:''`
    - 发现上面这样写不会被认为是空值,将其改为`null`便可以了
- 实现后端的时候,直接判断读取到的值是否为`null`出现报错,发现是有的函数不能接受`null`
    - 将其改为类似`!json.get("title").isJsonNull()`进行判断就可以了

七、思考题

1. 

2. SQL 注入是一种攻击方式,攻击者通过构造恶意 SQL 语句,在未正确处理用户输入的情况下执行非预期的数据库操作,例如:
    - **绕过身份验证**
    - **修改或删除数据库数据**
    - **获取敏感数据**

    ### 举例

    假设有一个用户登录的 SQL 语句:

    ```
    SELECT * FROM users WHERE username = '" + inputUsername + "' AND password = '" + inputPassword + "';
    ```

如果攻击者输入：

- `username = ' OR '1'='1'; --`

那么 SQL 变成：

```
SELECT * FROM users WHERE username = '' OR '1'='1' -- ' AND password = 'xxx';
```

这里 `--` 注释掉了 `AND password = 'xxx'`,这样最终 SQL 变成：

```
SELECT * FROM users WHERE username = '' OR '1'='1';
```

只要表中有任何用户,就会返回结果,绕过验证!

图书管理系统的易受攻击模块

在图书管理系统中，以下功能可能会遭受 SQL 注入攻击：

1. **用户登录模块**（如 `SELECT * FROM users WHERE username='...' AND password='...'`）
2. **书籍搜索功能**（如 `SELECT * FROM books WHERE title LIKE '%输入%'`）
3. **借书、还书记录查询**（如 `SELECT * FROM borrow_record WHERE user_id='...'`）

如何防御?

- **使用预处理语句（Prepared Statements）：**

```
String sql = "SELECT * FROM users WHERE username = ? AND password = ?";
PreparedStatement pstmt = connection.prepareStatement(sql);
pstmt.setString(1, inputUsername);
pstmt.setString(2, inputPassword);
ResultSet rs = pstmt.executeQuery();
```

这样 `?` 作为占位符,用户输入不会直接拼接到 SQL 里,彻底防止SQL注入。

- **最小权限原则** 数据库用户权限应最小化,避免 `DROP`、`DELETE` 等权限,降低风险。
- **输入校验** 限制特殊字符,比如 `--`、`'`、`;` 等,防止恶意 SQL 注入。

3. InnoDB 的默认隔离级别是 RR（Repeatable Read）,它可以防止脏读和不可重复读,但不会防止幻读。在高并发场景下,可能出现超卖库存的问题。

解决方案

方法 1：使用 “当前读” + `FOR UPDATE`

让 `SELECT` 语句加锁,避免并发读取相同数据

```
SELECT stock FROM books WHERE book_id = 1 FOR UPDATE;
```

- 事务 A 先执行,B必须等待A事务完成后才能继续执行,避免并发问题。

方法 2：基于行级锁（悲观锁）

在业务代码中加锁,例如：

```
Connection conn = dataSource.getConnection();
conn.setAutoCommit(false); // 关闭自动提交
PreparedStatement pstmt = conn.prepareStatement(
    "SELECT stock FROM books WHERE book_id = ? FOR UPDATE"
);
pstmt.setInt(1, bookId);
ResultSet rs = pstmt.executeQuery();
```

这样,查询语句会锁定该行数据,避免并发读取造成超卖。

方法 3：使用原子操作（CAS 乐观锁）

在更新库存时增加版本号或校验库存

```
UPDATE books SET stock = stock - 1 WHERE book_id = 1 AND stock > 0;
```

这样,如果库存已经减少为 0,更新操作不会生效,可以防止超卖。