设计报告

系统需求分析

编程实现图书的借阅功能，主要提供以下功能：

1. 图书的录入
2. 人员信息的录入
3. 图书的查询
4. 借阅图书的录入
5. 人员借阅信息的录入
6. 退出

上述六个功能可归结为：

(1) 图书信息的查询
依据图书名称、种类等信息进行图书检索，检索出的结果应逐条展示在界面上；同时为适应较大数据量的展示，应设置有分页功能。

(2) 图书信息的新增
用户在填写图书信息后，进行添加。

(3) 图书信息的修改
用户选择准备修改的图书后，对图书的名称、种类等信息进行修改。

(4) 图书信息的删除
用户选择界面展示出的某本图书进行删除操作。

(5) 用户信息的管理
为简化开发，用户信息的管理在图书的管理基础上进行小范围更改，功能大致相同。

(6) 图书借阅信息的查询
应为用户管理界面的子界面，先选择用户，再对其借阅信息进行查询

(7) 图书借阅及归还
应为用户管理界面的子界面，先选择用户，再选择其要借阅或归还的图书。

同时还应添加

(8) 登录
用户在界面上输入登录用户名和密码后，经判断跳转到信息管理界面

(9) 退出
信息管理界面应有退出功能，用户点击后退回到登录界面。

系统总体设计

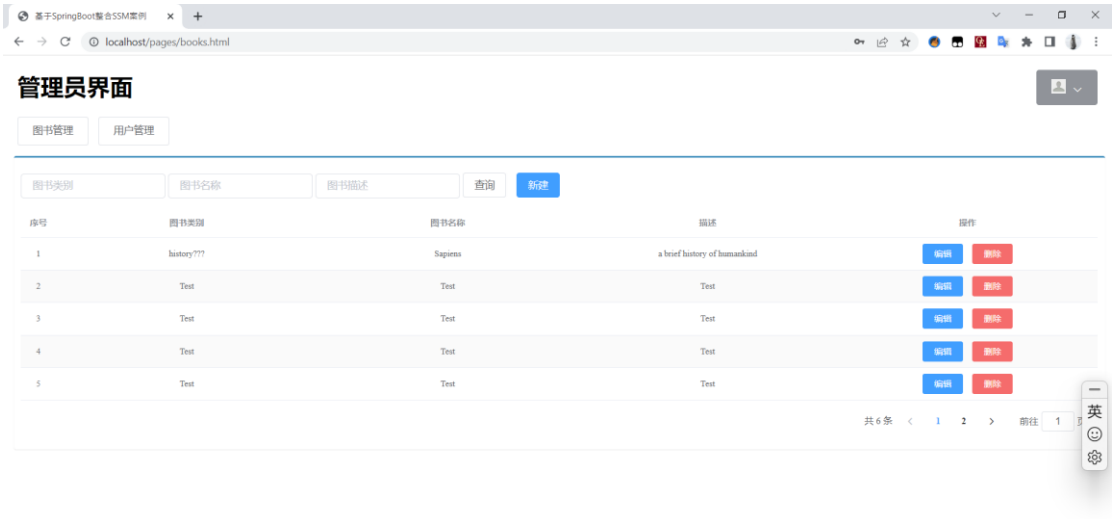
为实现以上功能，同时更加接近企业开发，进行如下总体设计：
本系统采用前后端分离架构模式，前端采用 Vue 框架简化开发， Element-ui

美化界面，axios 来发送请求；后端采用 Springboot 简化配置，Mybatis-plus 简化数据库操作；数据库采用 MySQL 来更好地与框架配合；使用 maven 来对项目进行管理

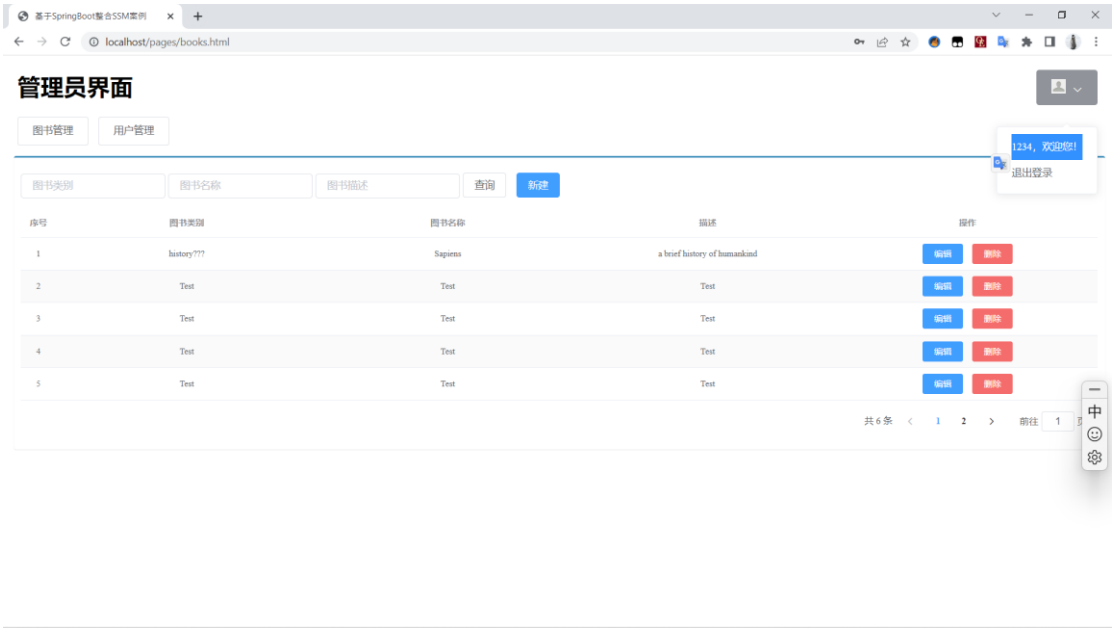
模块结构图如下

系统功能模块介绍

跳转到管理界面



右上角显示登录用户名，下拉菜单有退出登录的功能



一、图书管理模块

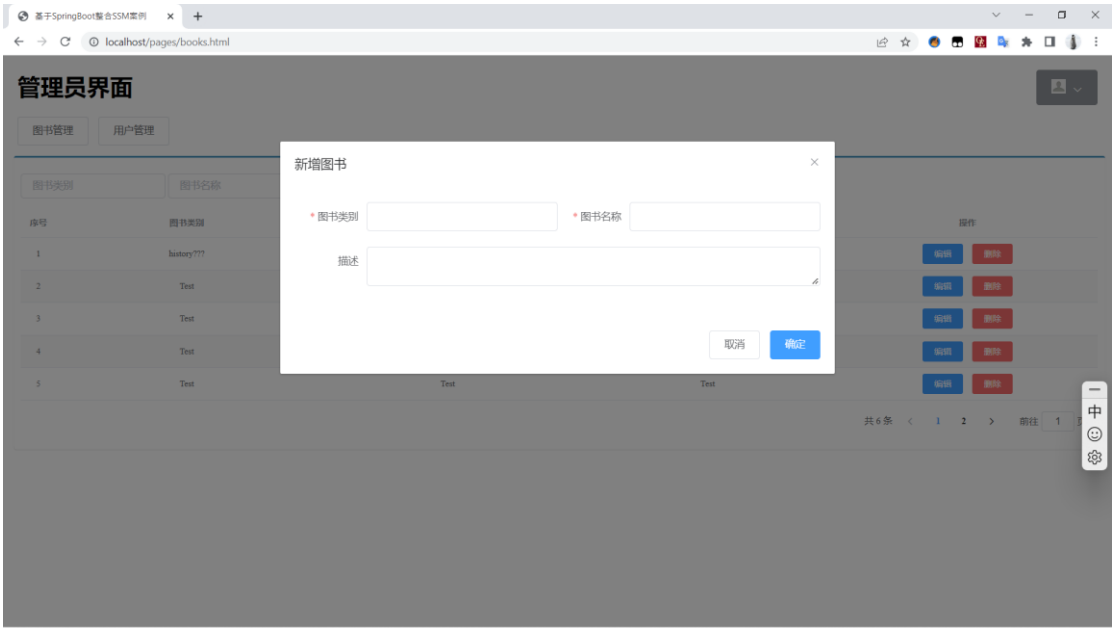
图书管理模块主要完成对图书的增删改查

1. 图书录入

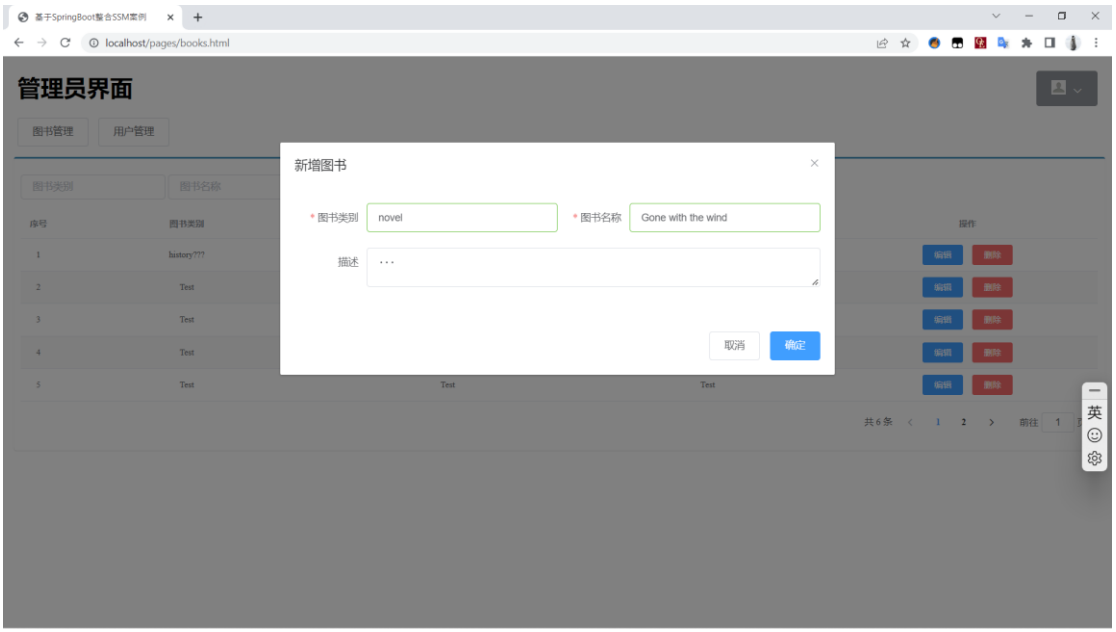
主要实现方式是

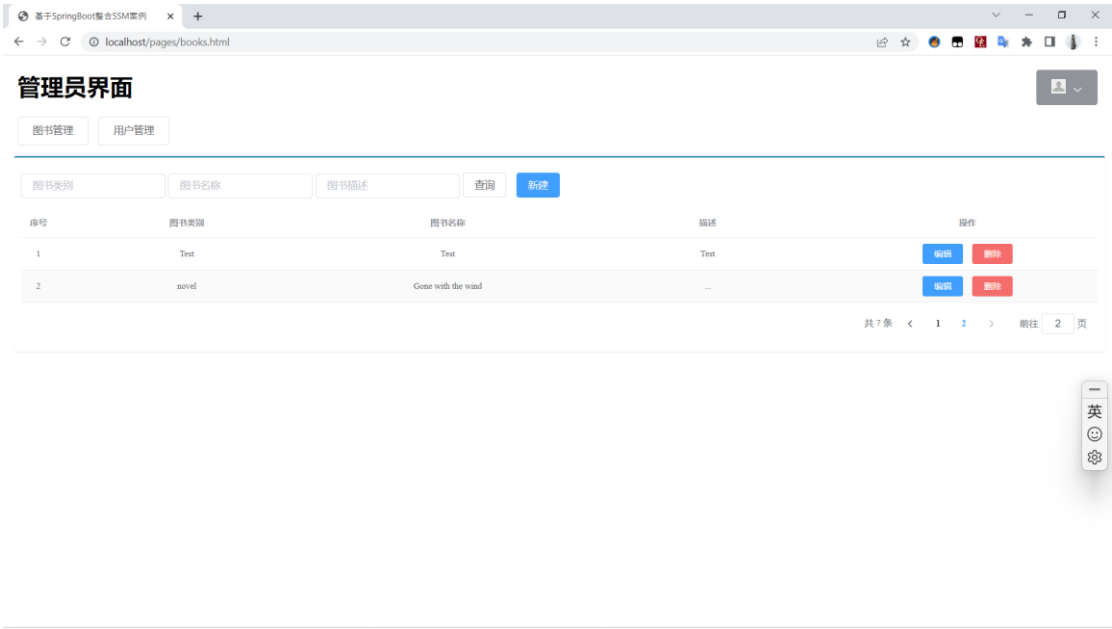
- 前端利用 axios 将 表单中的图书信息数据 传到后端的 Controller 层
- Controller 利用 Service 层中的 save() 方法，添加图书信息
- 将返回结果打包成 Result 数据回传到前端，并以此做出变化
  - 成功则刷新数据
  - 失败则弹出相应的提示信息

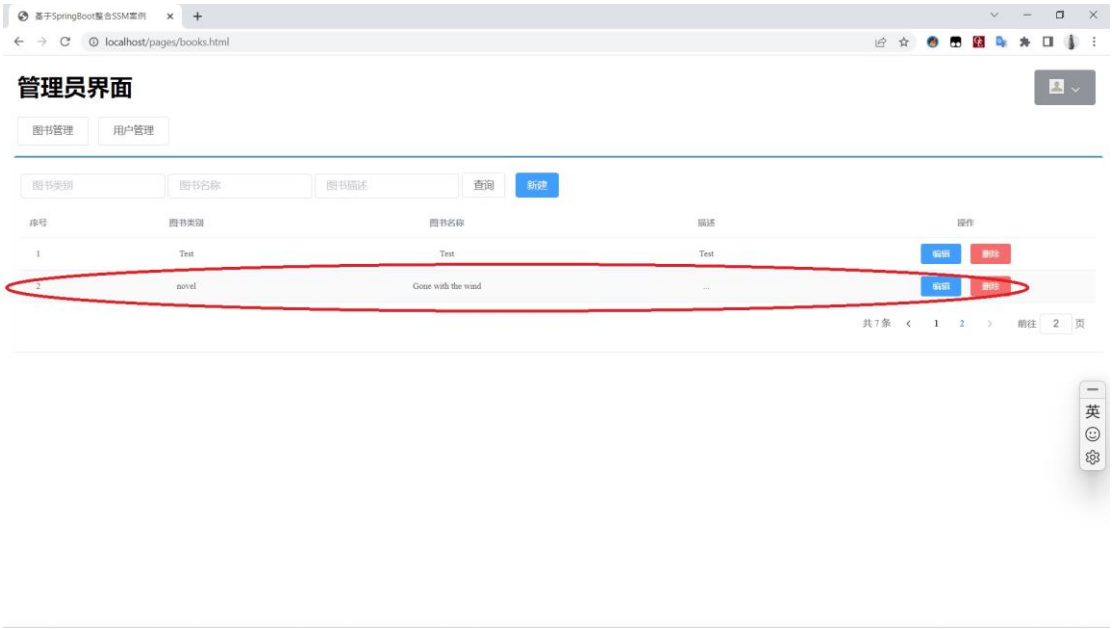
点击新建按钮后，出现新建表单弹窗



填入新增的图书信息，点确定按钮后，完成图书载入操作





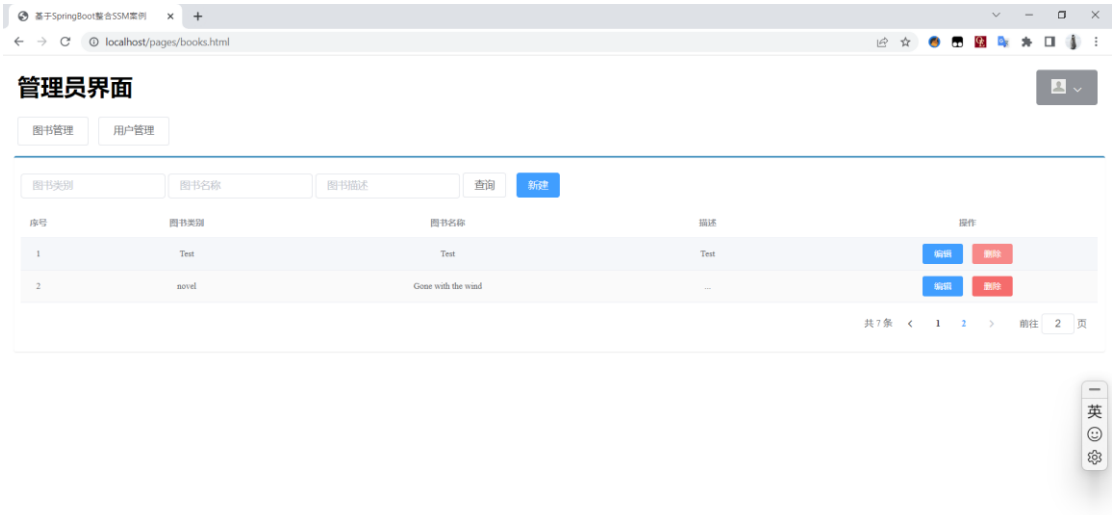


2. 图书删除

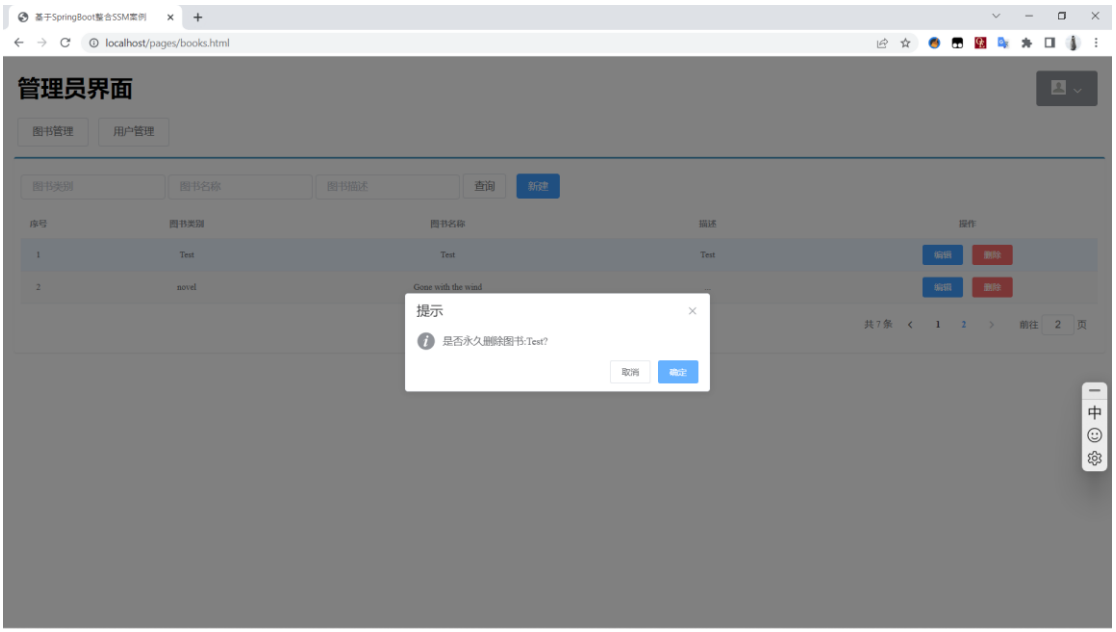
主要实现方式是

- 前端利用 axios 将 该行的图书信息数据 传到后端的 Controller 层
- Controller 利用 Service 层中的 save() 方法，添加图书信息
- 将返回结果打包成 Result 数据回传到前端，并以此做出变化
  - 成功则刷新数据
  - 失败则弹出相应的提示信息

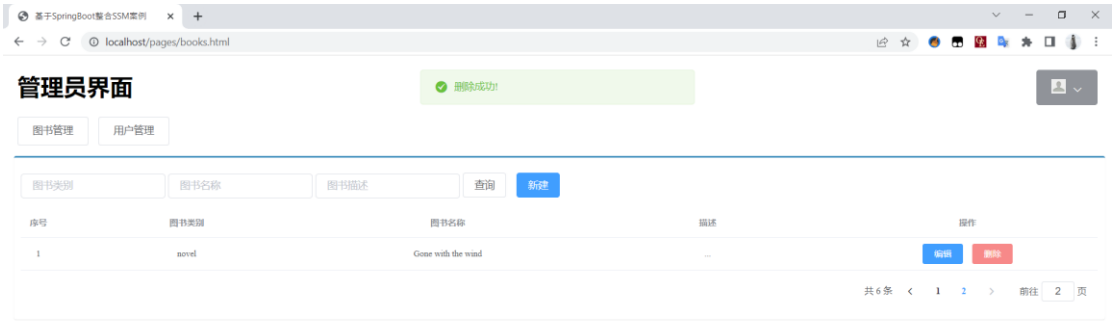
点击图书信息后面的删除按钮，将图书信息删除



弹出是否删除的提示



点击确定后，成功删除

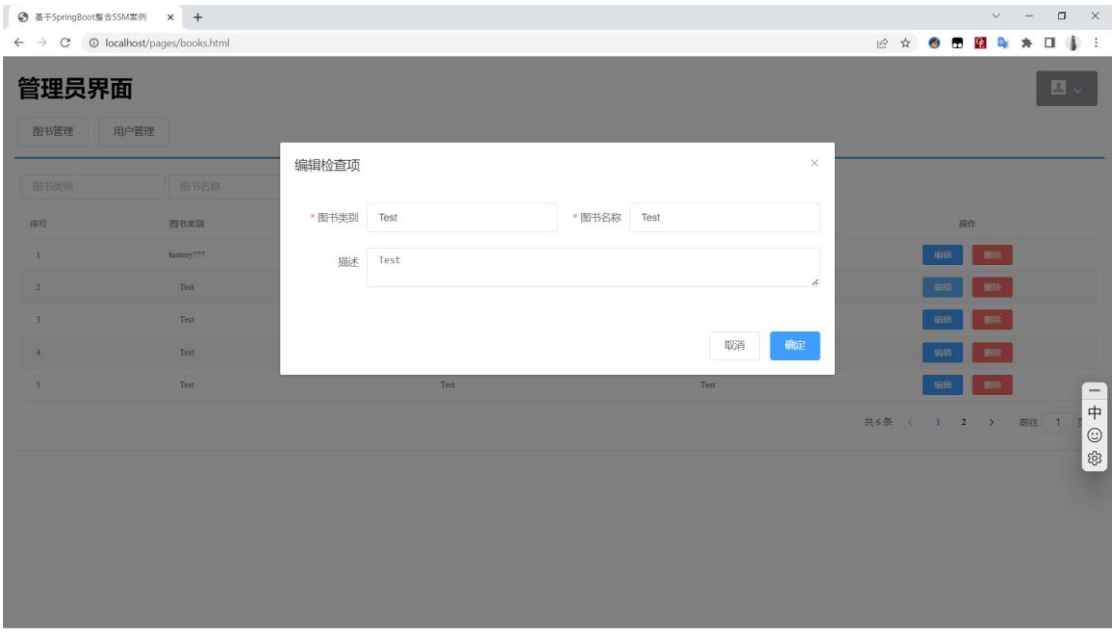


3.图书编辑

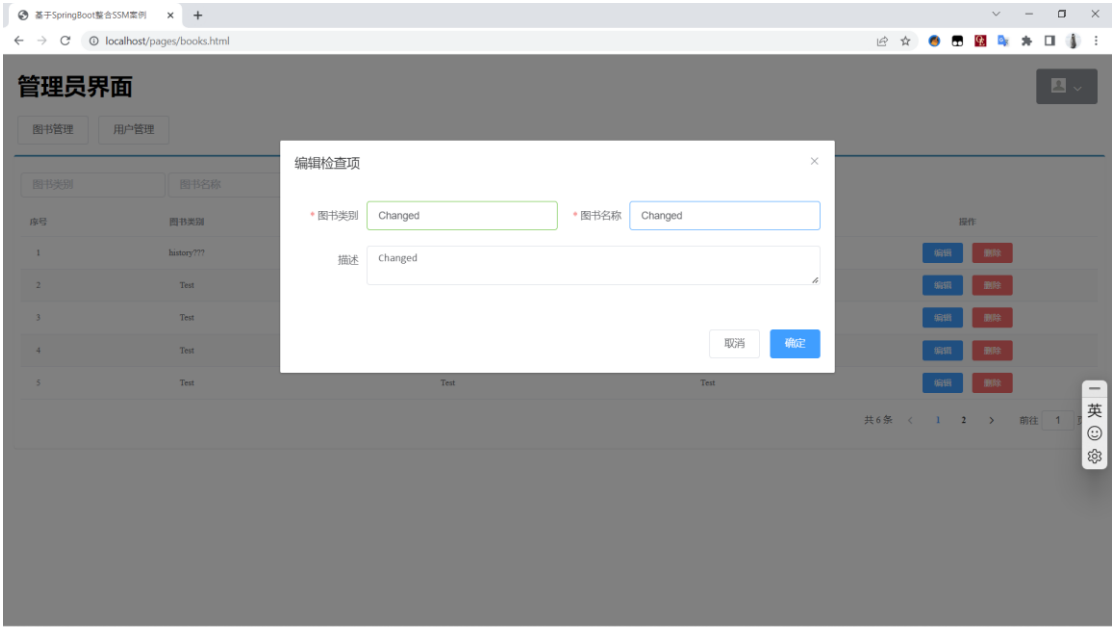
主要实现方式是

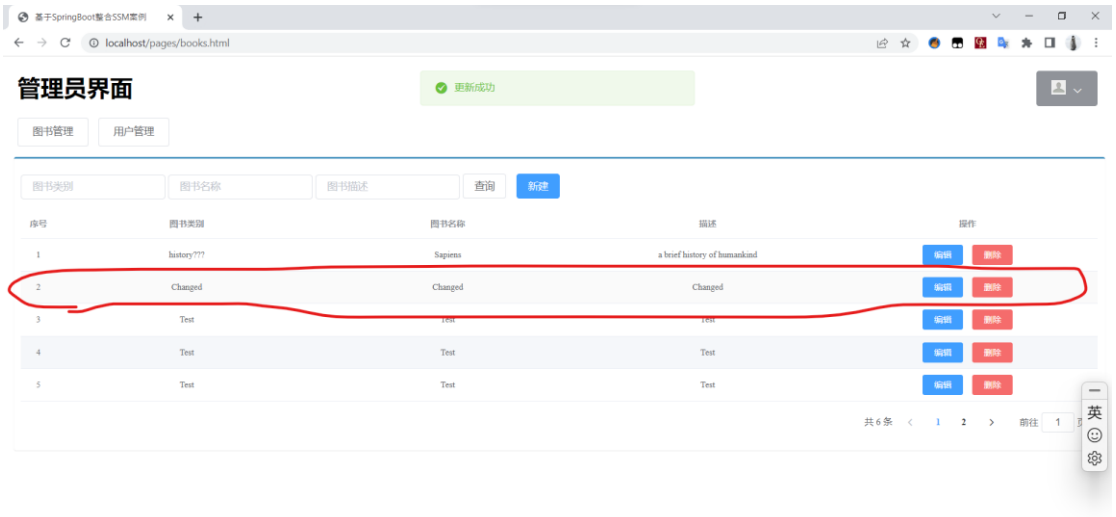
- 前端利用 axios 获取 该行原有图书信息数据，并初始化到表单里
- 再利用 axios 将更改后的数据 传到后端的 Controller 层
- Controller 利用 Service 层中的 update() 方法，更改图书信息
- 将返回结果打包成 Result 数据回传到前端，并以此做出变化
  - 成功则刷新数据
  - 失败则弹出相应的提示信息

点击编辑按钮，弹出编辑弹窗



在指定位置填写对应数据进行编辑



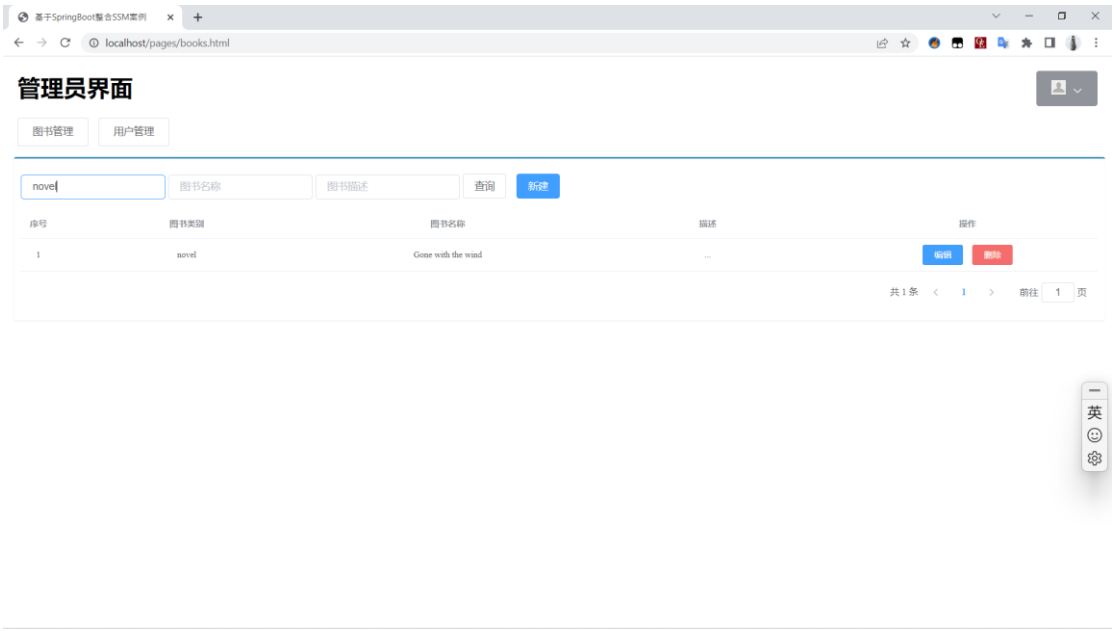


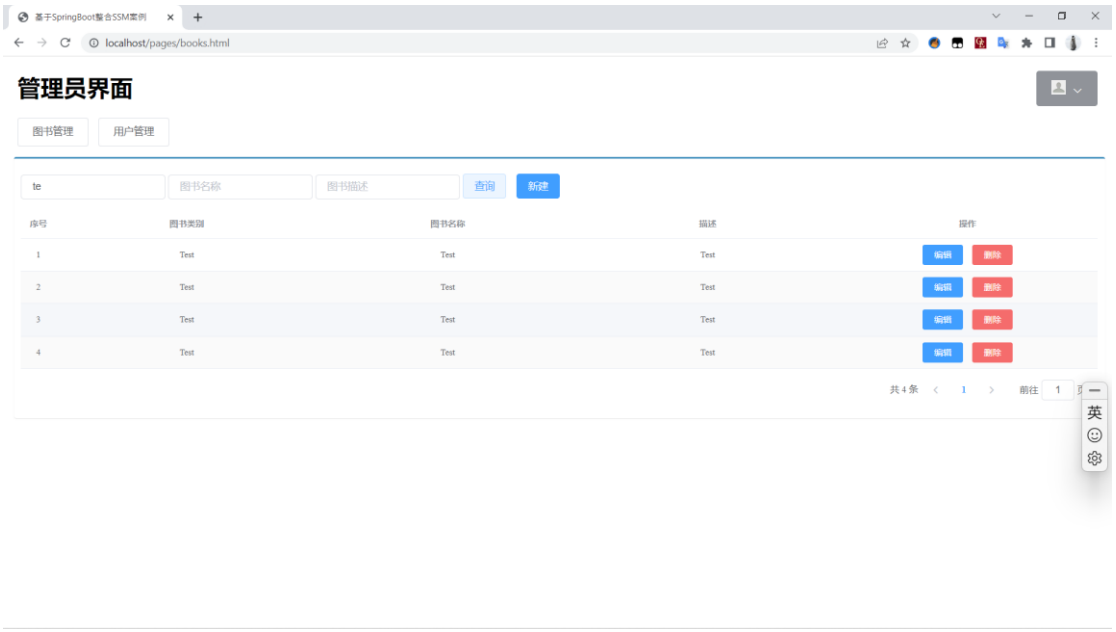
4.图书查询

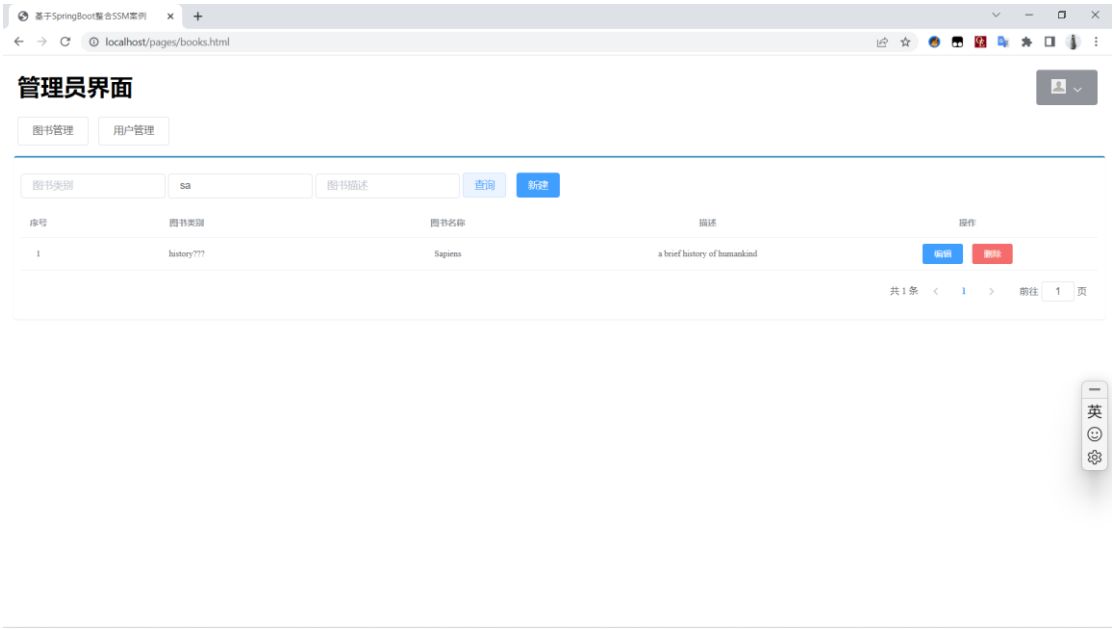
主要实现方式是

- 前端利用 axios 将查询内容 传到后端的 Controller 层
- Controller 利用 Service 层中的 getPage () 方法，返回查询结果
- 将返回结果打包成 Result 数据回传到前端，并以此做出变化
  - 成功则刷新数据
  - 失败则弹出相应的提示信息

在输入框内指定位置输入想要查询的图书信息，点击查询按钮进行查询







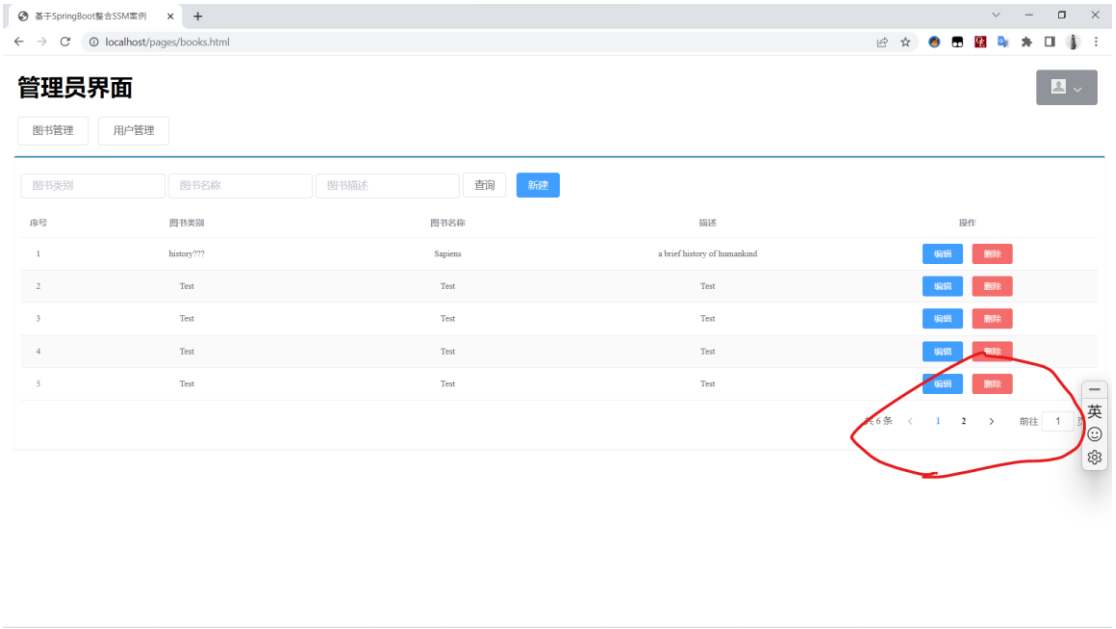
5.分页功能

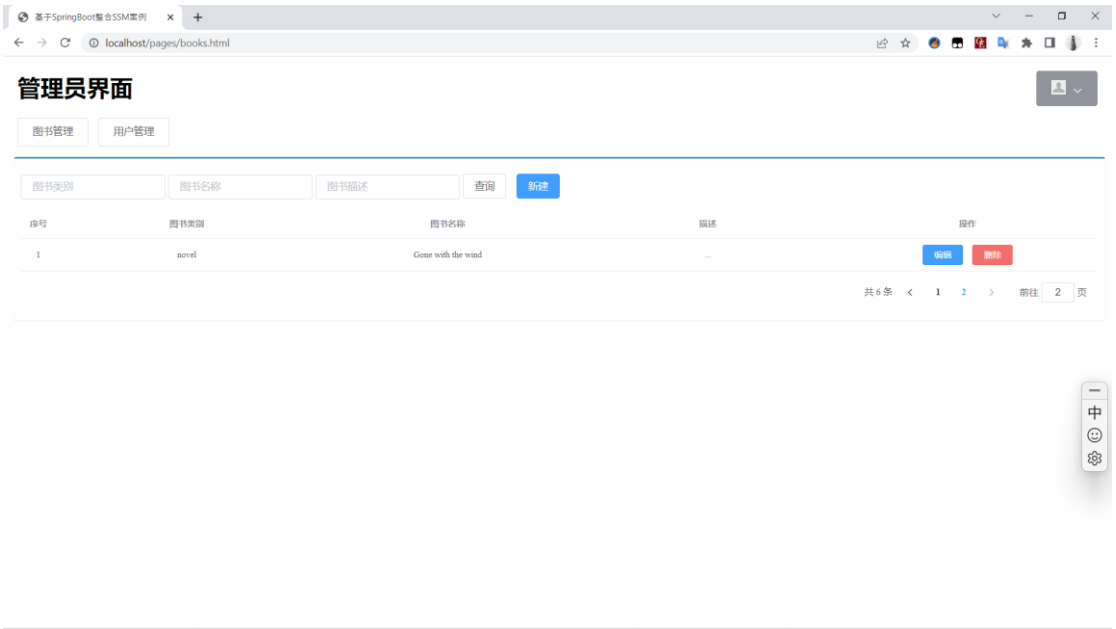
主要实现方式是

- 前端利用 axios 将查询内容 传到后端的 Controller 层
- Controller 利用 Service 层中的 getPage () 方法，返回查询结果
- 将返回结果打包成 Result 数据回传到前端，并以此做出变化
  - 成功则刷新数据
  - 失败则弹出相应的提示信息

底层使用的是 Mybatis-plus 的 selectPage() 方法，再添加 PaginationInnerInterceptor() 拦截器，实现在 sql 语句后添加 limit;

在页面右下角可以进行分页查询





二、用户管理模块

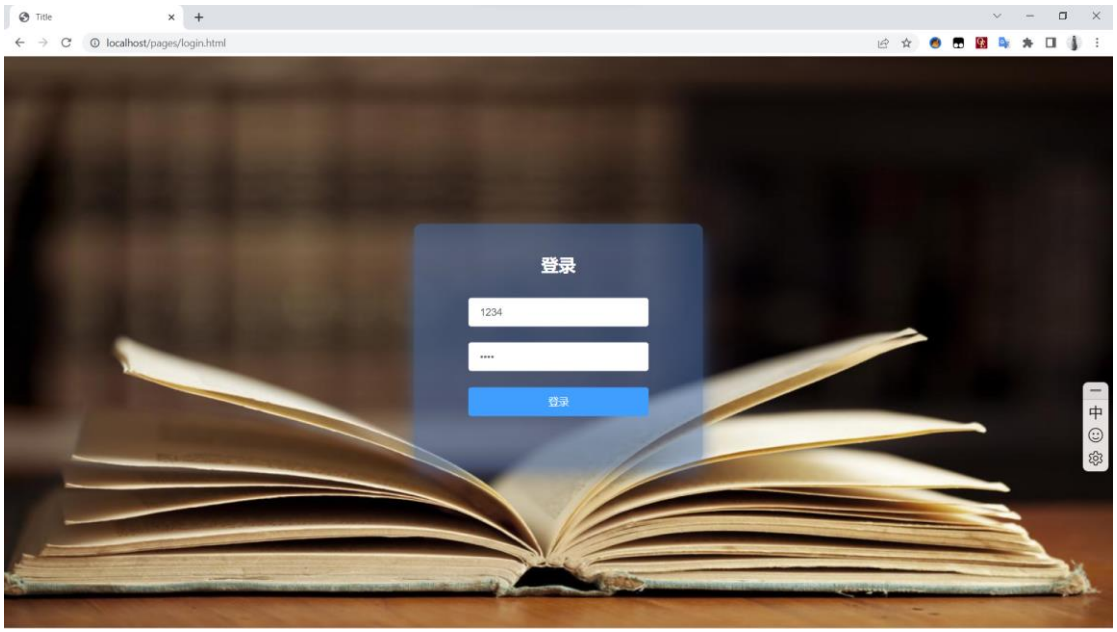
1. 用户登录

主要实现的是用户登录的功能，登陆失败弹出用户不存在以及密码错误的提示，登陆成功则将页面定向到管理界面。

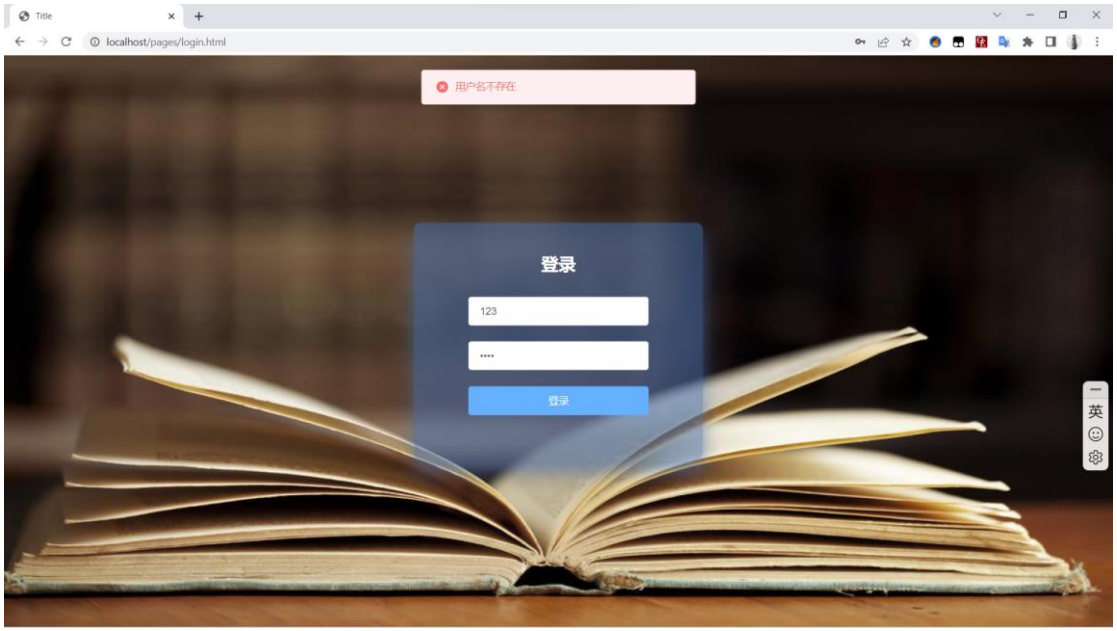
主要实现方式是

- 前端利用 axios 将 表单中的图书信息数据 传到后端的 Controller 层

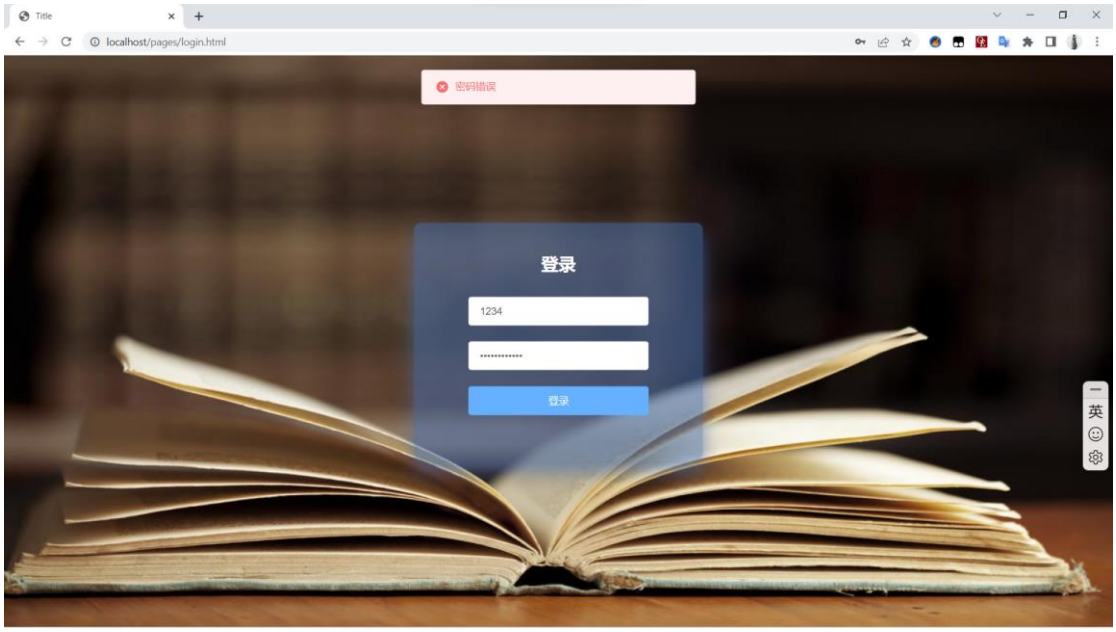
- Controller 利用 Service 层中的 getOne() 方法，查询数据库中是否有相应的用户信息，并以此判断登录用户是否成功
- 将返回结果打包成 Result 数据回传到前端，并以此做出变化
  - 成功则跳转到管理界面
  - 失败则弹出相应的提示信息



用户名不存在：



密码错误：



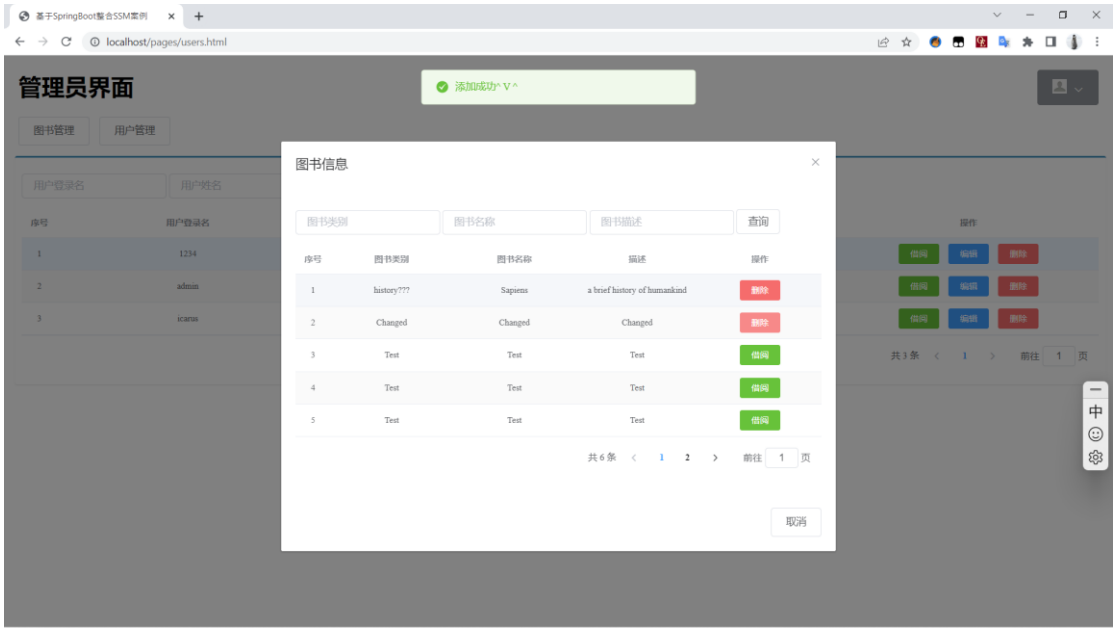
2. 借书还书

(1)图书借阅

主要实现方式是

- 前端获取借书用户的 id，以及图书 id，并用 axios 传到后端的 Controller 层
- Controller 利用 Service 层中的 saveBookById () 方法，保存图书信息
- 将返回结果打包成 Result 数据回传到前端，并以此做出变化
  - 成功则重新查询以刷新页面
  - 失败则弹出相应的提示信息

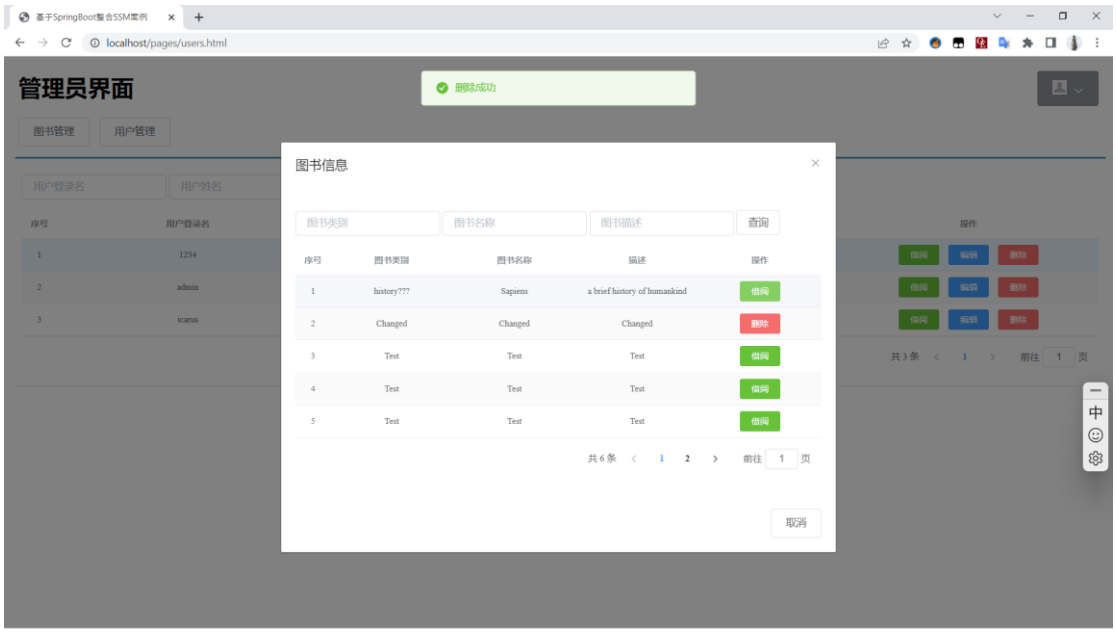
在想借阅的图书右侧点击借阅按钮，借阅按钮将变成删除按钮，代表图书借阅操作成功，完成借阅操作



(2)图书归还

实现方式同理

在想归还的图书右侧点击删除按钮，借阅按钮将重新启用，完成对图书的归还操作

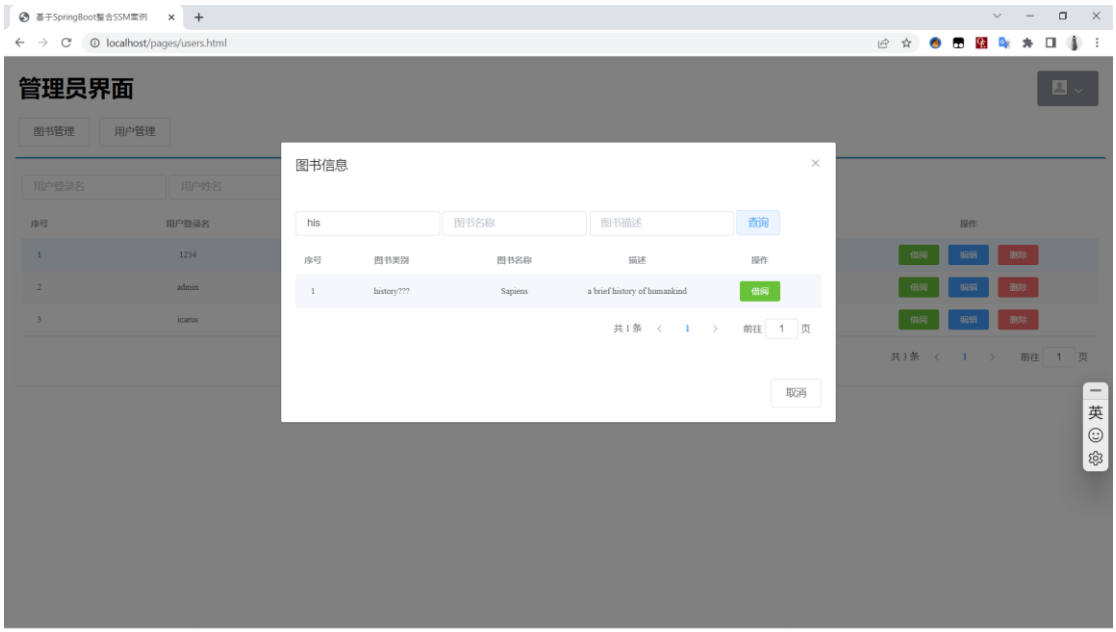


(3)图书查询

- 前端获取借书用户的 id，并用 axios 传到后端的 Controller 层
- Controller 利用 Service 层中的 getPagesByUserId () 方法，返回查询结果
- 将返回结果打包成 Result 数据回传到前端，并以此做出变化

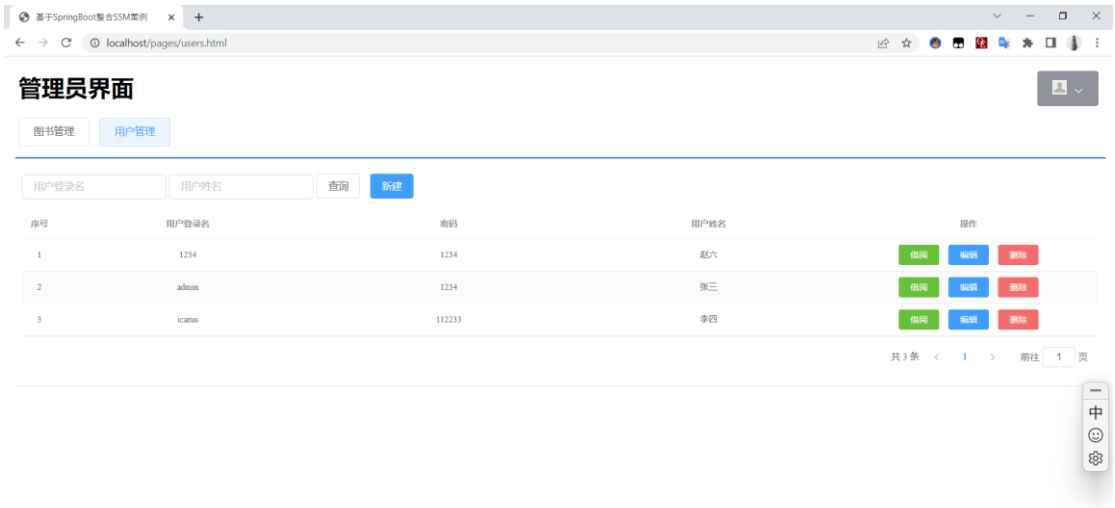
- 成功则展示查询得到的 page
- 失败则弹出相应的提示信息

在图书查询框中输入想要查询的内容，点击查询按钮对图书进行查询



用户管理模块主要完成对用户信息的增删改查操作，操作同图书管理模块类似

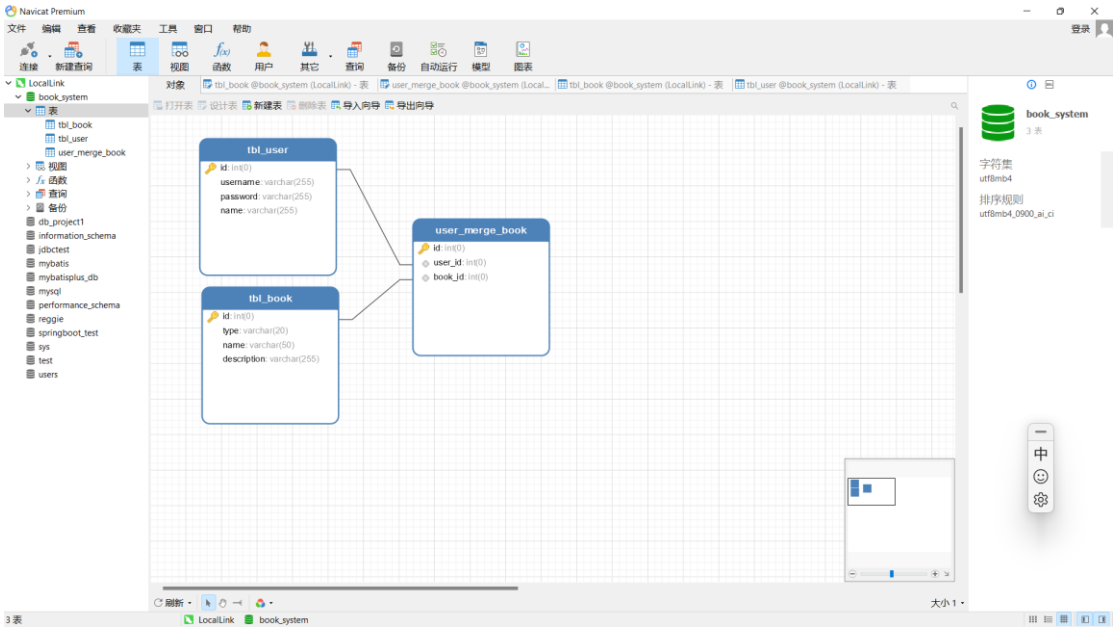
点击用户管理按钮跳转到用户管理界面



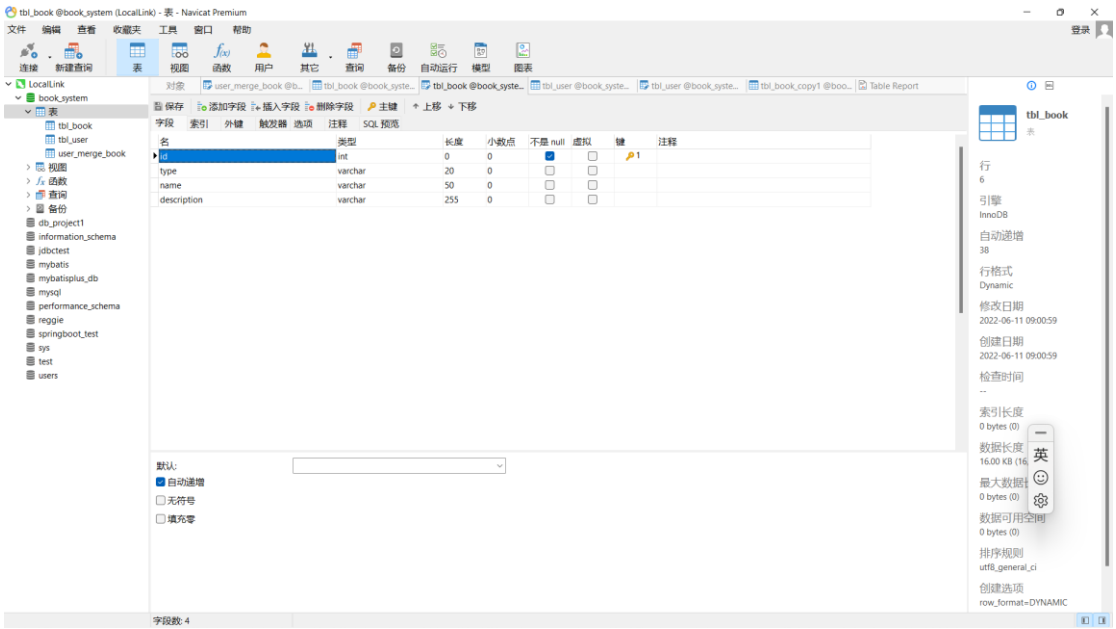
借阅功能将在借阅模块详细说明

系统数据结构描述

本系统采用 Mysql 数据库作为数据源，设计有三张表结构，一张是图书表，一张是用户表，一张是图书用户关系表



1.用户表结构

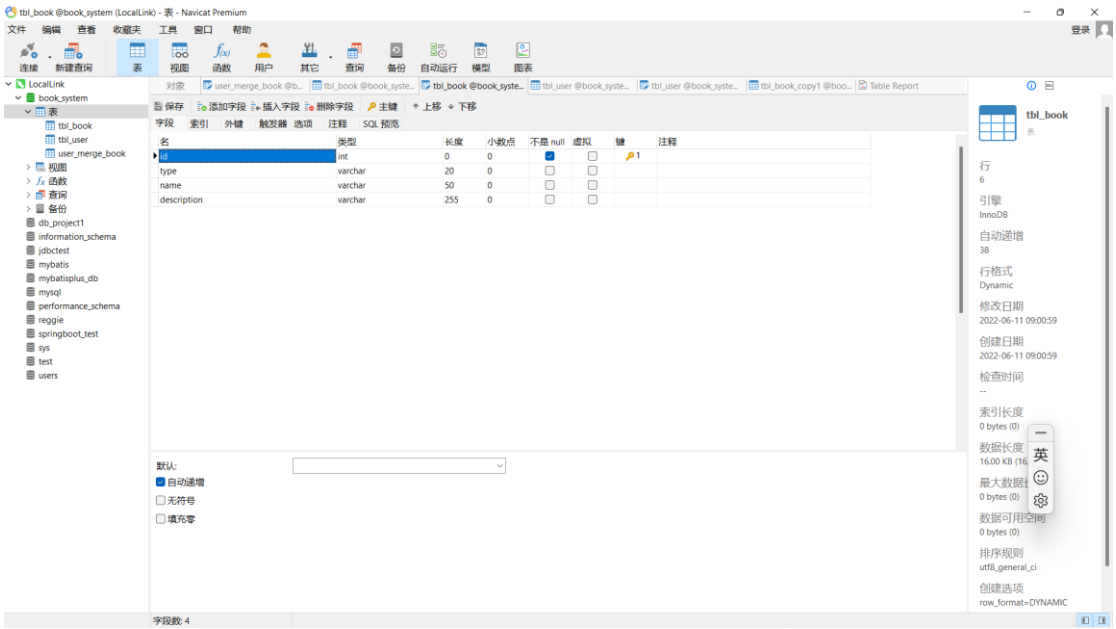


表结构代码

```
-- ----------------------------
-- Table structure for tbl_book
-- ----------------------------
DROP TABLE IF EXISTS `tbl_book`;
CREATE TABLE `tbl_book`  (
  `id` int(0) NOT NULL AUTO_INCREMENT,
  `type` varchar(20) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL,
  `name` varchar(50) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL,
  `description` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 38 CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Dynamic;

SET FOREIGN_KEY_CHECKS = 1;
```

2.图书表结构

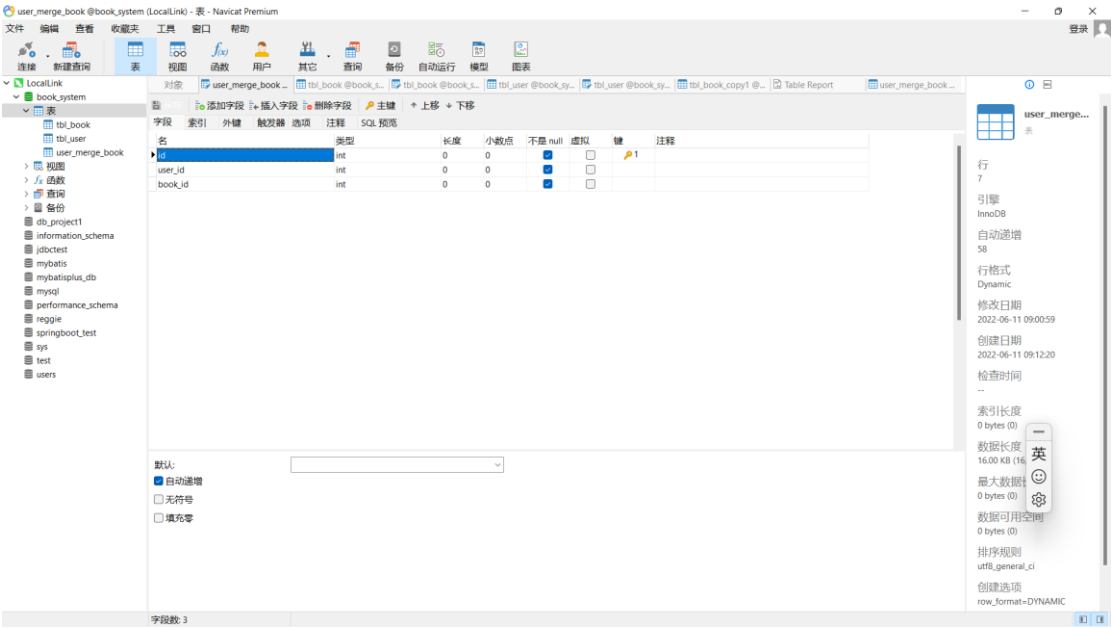


表结构代码：

```
-- ----------------------------
-- Table structure for tbl_user
-- ----------------------------
DROP TABLE IF EXISTS `tbl_user`;
CREATE TABLE `tbl_user`  (
  `id` int(0) NOT NULL AUTO_INCREMENT,
```

```
  `username` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL,
  `password` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL,
  `name` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 5 CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Dynamic;
```

3.关系表结构



表结构代码：

```
-- ----------------------------
-- Table structure for user_merge_book
-- ----------------------------
DROP TABLE IF EXISTS `user_merge_book`;
CREATE TABLE `user_merge_book`  (
  `id` int(0) NOT NULL AUTO_INCREMENT,
  `user_id` int(0) NOT NULL,
  `book_id` int(0) NOT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 58 CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Dynamic;
```

附：

重要实现代码：

后端

Controller 层

BookController

```
@RestController
@RequestMapping("/books")
public class BookController {
    @Resource
    private BookService bookService;

    //新增图书
    @PostMapping
    public Result save(@RequestBody Book book){
        if (bookService.save(book)){
            return Result.success(null,"添加成功^ V ^");
        }else{
            return Result.error("添加失败!");
        }
    }

    //修改图书
    @PutMapping
    public Result update(@RequestBody Book book){
        if (bookService.update(book)){
            return Result.success(null,"更新成功");
        }else{
            return Result.error("数据同步失败");
        }
    }

    //删除图书
    @DeleteMapping("/{id}")
    public Result delete(@PathVariable Integer id){
        if (bookService.delete(id)){
```

```
            return Result.success(null,"删除成功!");
        }else{
            return Result.error("数据同步失败");
        }
    }

    //初始化原图书数据
    @GetMapping("/{id}")
    public Result getById(@PathVariable Integer id){
        Book resBook = bookService.getById(id);
        if (resBook != null){
            return Result.success(resBook,"获取成功");
        }else {
            return Result.error("数据同步失败");
        }
    }

    //分页查询
    @GetMapping("/{currentPage}/{pageSize}")
    public Result getPage(@PathVariable Integer
currentPage,@PathVariable Integer pageSize,Book book){
        System.out.println(book);
        IPage<Book> page = bookService.getPage(currentPage,
pageSize,book);
        if(currentPage > page.getPages())
            page = bookService.getPage((int)
page.getPages(),pageSize,book);
        return Result.success(page,"获取成功");
    }
}
```

UserController

```
@RestController
@RequestMapping("/users")
public class UserController {
    @Resource
    UserService userService;

    //用户登录
    @PostMapping("/login")
    public Result login(@RequestBody User user){
        System.out.println("111");
```

```
        LambdaQueryWrapper<User> lambdaQueryWrapper = new
LambdaQueryWrapper();
        lambdaQueryWrapper.eq(User::getUsername,user.getUsername());
        System.out.println("222");
        User user1 = userService.getOne(lambdaQueryWrapper);//依据用户名
得到用户
        System.out.println("333");
        if (user1 == null){
            return Result.error("用户名不存在");
        }
        if (!user1.getPassword().equals(user.getPassword())){
            return Result.error("密码错误");
        }
        return Result.success(user,"登录成功");
    }

    //新建用户
    @PostMapping
    public Result save(@RequestBody User user){
        if (userService.save(user)){
            return Result.success(null,"添加成功^ V ^");
        }else{
            return Result.error("添加失败!");
        }
    }

    //修改用户
    @PutMapping
    public Result update(@RequestBody User user){
        if (userService.update(user)){
            return Result.success(null,"更新成功");
        }else{
            return Result.error("数据同步失败");
        }
    }

    //删除用户
    @DeleteMapping("/{id}")
    public Result delete(@PathVariable Integer id){
        if (userService.delete(id)){
            return Result.success(null,"删除成功!");
        }else{
            return Result.error("数据同步失败");
        }
```

```
    }

    //初始化原用户信息
    @GetMapping("/{id}")
    public Result getById(@PathVariable Integer id){
        User resUser = userService.getById(id);
        if (resUser != null){
            return Result.success(resUser,"获取成功");
        }else {
            return Result.error("数据同步失败");
        }
    }

    //分页查询
    @GetMapping("/{currentPage}/{pageSize}")//这里这么写就行了
    public Result getPage(@PathVariable Integer
currentPage,@PathVariable Integer pageSize,String name,User user){
        System.out.println(name);
        System.out.println(user);
        IPage<User> page = userService.getPage(currentPage,
pageSize,user);
        if(currentPage > page.getPages())
            page = userService.getPage((int)
page.getPages(),pageSize,user);
        return Result.success(page,"获取成功");
    }

    //图书借阅
    @PostMapping("/books")
    public Result saveBook(@RequestBody UserBookRelation ubr){
        if (userService.saveBookById(ubr.getUserId(),ubr.getBookId())){
            return Result.success(null,"添加成功^ V ^");
        }else{
            return Result.error("添加失败!");
        }
    }

    //图书归还
    @DeleteMapping("/books/{userId}/{bookId}")
    public Result deleteBook(@PathVariable Integer userId,@PathVariable
Integer bookId){
        if(userService.deleteBookById(userId,bookId)){
            return Result.success(null,"删除成功");
        }else{
```

```
            return Result.error("数据同步失败");
        }
    }

    //借阅图书分页查询
    @GetMapping("/books/{currentPage}/{pageSize}/{userId}")//这里这么写
就行了
    public Result getPage(@PathVariable Integer
currentPage,@PathVariable Integer pageSize,@PathVariable Integer
userId,Book book){
        IPage<Book> page =
userService.getPagesByUserId(userId,currentPage,pageSize,book,"all");
        System.out.println(page.getRecords());
        if(currentPage > page.getPages())
            page = userService.getPagesByUserId(userId,(int)
page.getPages(),pageSize,book,"all");
        return Result.success(page,"获取成功");
    }
}
```

Service 层

BookService

```
public interface BookService {
    boolean save(Book book);//保存图书信息
    boolean update(Book book);//更新图书信息
    boolean delete(Integer id);//删除图书信息
    Book getById(Integer id);//以 id 获取图书
    List<Book> getAll();//获取所有图书信息
    LambdaQueryWrapper<Book> searchBooks(Book book);//搜索目标图书
    IPage<Book> getPage(Integer currentPage,Integer pageSize,Book
book);//获取图书分页
}
```

UserService

```
@Service
public interface UserService {
    User getOne(Wrapper<User> queryWrapper);//依条件获取用户信息
```

```
    boolean save(User user);//新增用户信息
    boolean delete(Integer id);//删除用户信息
    User getById(Integer id);//以 id 获取用户
    boolean update(User user);//更新用户信息
    List<User> getAll();//获取所有用户
    LambdaQueryWrapper<User> searchUsers(User user);//搜索用户
    IPage<User> getPage(Integer currentPage, Integer pageSize, User
user);//获取用户分页

    LambdaQueryWrapper<Book> searchBooks(Book book);//搜索借阅图书
    IPage<Book> getPagesByUserId(Integer userId,Integer currentPage,
Integer pageSize, Book book , String filter);//获得借阅图书分页
    boolean deleteBookById(Integer userId,Integer bookId);//删除借阅信息
    boolean saveBookById(Integer userId,Integer bookId);//添加借阅信息
}
```

Dao 层

```
@Repository
@Mapper
public interface BookDao extends BaseMapper<Book> {
}
```

```
@Repository
@Mapper
public interface UserDao extends BaseMapper<User> {
}
```

```
@Repository
@Mapper
public interface UbrDao extends BaseMapper<UserBookRelation> {
    @Select("select b.* from tbl_book b,user_merge_book ub where
ub.user_id = #{id} and b.id = ub.book_id;")
    List<Book> selectBooksByUserId(Integer id);
}
```

实体类

Book

```
@Data
public class Book {
    private Integer id;
    private String name;
    private String type;
    private String description;
    @TableField(exist = false)
    private boolean isBorrowed = false;
}
```

User

```
@Data
public class User {
    private Integer id;
    private String username;
    private String password;
    private String name;
}
```

UserBookRelation

```
@Data
@TableName("user_merge_book")
public class UserBookRelation {
    private Integer id;
    @TableField("user_id")
    private Integer userId;
    @TableField("book_id")
    private Integer bookId;
}
```