

面向对象分析与设计项目文档

选题及成员

选题：图书管理系统

学号	姓名	分工
11310057	张宇	前端代码实现, 后端 Controller 实现
11310073	张凌祺	需求分析
11310082	欧泽彬	实体类、完整类图
11310116	孙佳明	设计模式
11310182	方致远	可扩展性分析
11310388	邓收港	后端代码实现, 部分顺序图实现, 单元测试及其活动图
11410328	皮志成	用例图, 用例分析文档, 顺序图

目录

[面向对象分析与设计项目文档](#)

[选题及成员](#)

[目录](#)

[需求分析](#)

[图书搜索](#)

[登录](#)

[借书](#)

[还书](#)

[个人信息查询](#)

[用户管理](#)

[图书管理](#)

[用例分析](#)

[用例图](#)

[用例分析文档](#)

[实体类类图](#)

[名词分析法](#)

[仅带有属性的实体类类图](#)

[设计模式](#)

[顺序图](#)

[登录](#)

[查看个人信息](#)

[查看已借阅图书](#)

[检查是否为 Admin](#)

[所有用户列表](#)
[所有图书列表](#)
[增加图书](#)
[删除图书](#)
[增加用户](#)
[删除用户](#)
[改变用户的类型](#)
[feelLucky](#)
[查找图书](#)
[借阅图书](#)
[归还图书](#)
[完整类图](#)
 [user 包类图](#)
 [book 包类图](#)
 [web 包类图](#)
[项目实现](#)
[测试](#)
 [借书测试及活动图](#)
 [用户登录测试及活动图](#)
[可扩展性分析](#)
 [加入修改图书信息的功能](#)
[解决方案](#)

需求分析

图书搜索

可以直接在搜索框搜索。用户可以输入书名进行模糊搜索，也可按图书类型搜索。使用搜索功能不需要登录。

登录

用户用 id 和密码登录。登录之后可以查看账户信息，借阅信息，借还书。另外，管理员登录之后还可以进行用户管理及图书管理

借书

用户登录之后，可以进行借书。若图书在馆，则可以借阅成功。若图书已被借出，则借阅失败。

还书

用户登录之后，可以访问个人页面，对已借阅的图书可以进行还书操作。

个人信息查询

用户登录之后，可以访问个人页面查询个人信息。

用户管理

管理员登录之后，可以访问个人页面，进行用户管理。可以

- 改变现有用户的等级（普通，高级，管理员）。
- 增加新用户
- 删 除用户

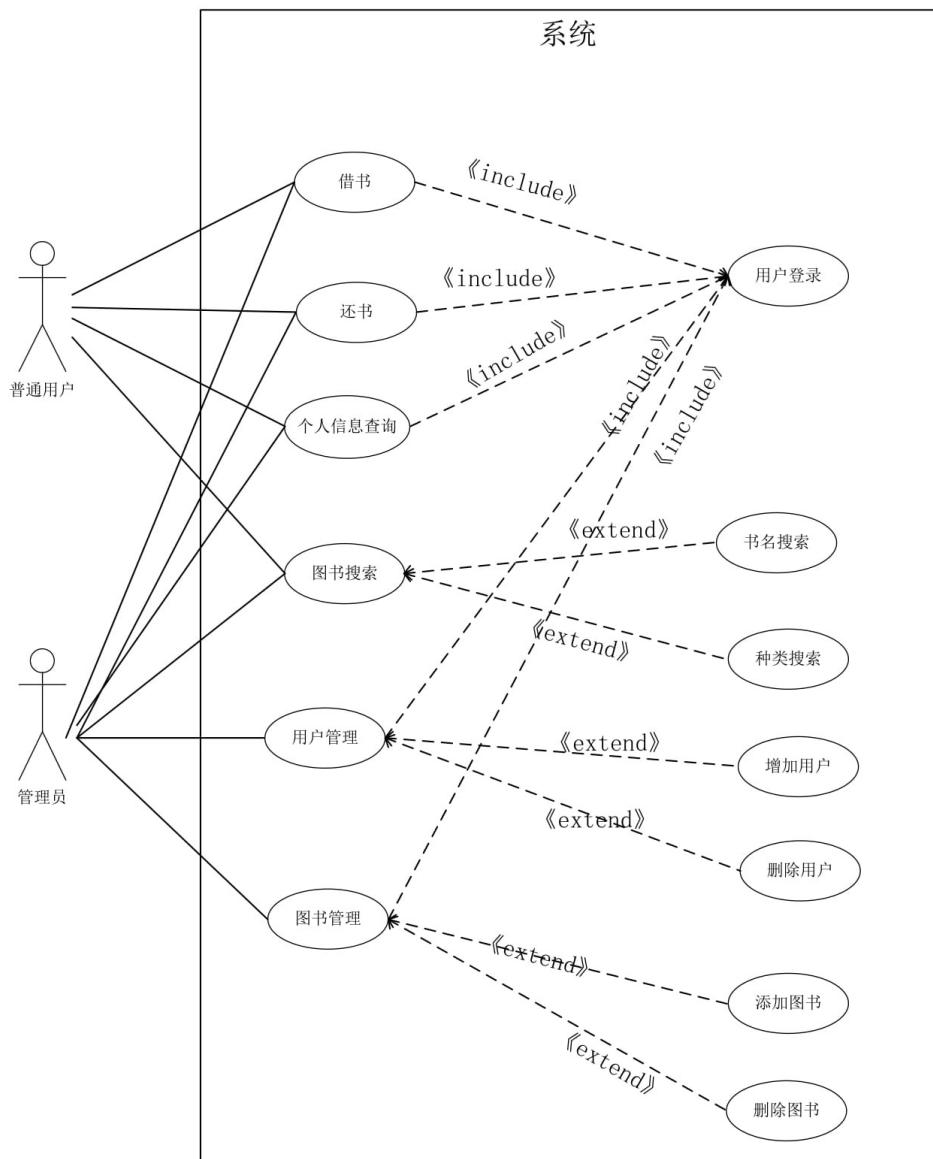
图书管理

管理员登录之后，可以访问个人页面，进行图书管理。可以

- 增加新图书
- 删 除图书（还未归还的图书不可删除）

用例分析

用例图



用例分析文档

用例名称	借书	
参与者	用户	
用例概述	此用例展现用户借书的整个过程	
基本事件流	参与者事件 1. 参与者发起借书请求 6. 带着图书跑路	系统事件 2. 查看书籍数量是否足够 3. 查看是否允许用户借书 4. 修改用户借书信息 5. 修改图书状态
扩展事件流	步骤 2: 如果书籍数量不足则结束用例 步骤 3: 用户最大借书量小于或等于已借书数目则结束用例	
前置条件	用户已登录	

用例名称	登录	
参与者	用户	
用例概述	用户用 id 和密码登录	
基本事件流	参与者事件 1. 参与者输入 ID 和密码发起登录请求	系统事件 2. 通过 ID 找到用户 3. 比较输入密码和账号密码 4. 跳转到相应页面
扩展事件流	步骤 2: 如果未找到用户则提示错误，结束用例 步骤 3: 密码不匹配则提示错误，结束用例	

用例名称	还书	
参与者	用户	
用例概述	用户还书	
基本事件流	参与者事件 1. 用户发起还书请求	系统事件 2. 系统识别书籍信息 3. 修改图书状态 4. 修改用户信息
扩展事件流	步骤 2: 若找不到图书信息则提示错误，结束用例 步骤 3: 若图书状态为未借出则提示错误，结束用例	

用例名称	个人信息查询	
参与者名称	用户	
用例概述	查询信息	
基本事件流	参与者事件 1. 用户发起查询个人信息请求	系统事件 2. 返回用户信息
前置条件	用户已登录	

用例名称	图书搜索	
参与者名称	用户	
用例概述	搜索图书	
基本事件流	参与者事件 1.输入要搜索的内容，发起请求	系统事件 2.判断输入内容是否合法 3.模糊搜索 4.返回结果
扩展事件流	步骤 3：若输入内容为图书类型，按类型搜索，若输入为书名，按书名搜索	

用例名称	增加用户	
参与者名称	管理员	
用例概述	管理员添加用户	
基本事件流	参与者事件 1.管理员发起添加请求 3.填写表单，提交表单	系统事件 2.返回一张填写表单 4.判断信息完整正确与否 5.添加用户
扩展事件流	信息不完整或有问题则结束用例	
前置条件	管理员登录	

用例名称	删除用户	
参与者名称	管理员	
用例概述	管理员删除用户	
基本事件流	参与者事件 1.发起删除请求	系统事件 2.删除用户
前置条件	管理员登录	

用例名称	增加图书	
参与者名称	管理员	
用例概述	管理员添加图书	
基本事件流	参与者事件 1 管理员发起添加请求 3.填写表单，提交表单	系统事件 2 给管理返回一张表单 4 判断信息完整正确与否 5 添加图书
扩展事件流	信息不完整或有问题则结束用例	
前置条件	管理员登录	

用例名称	删除图书	
参与者名称	管理员	
用例概述	管理员删除图书	
基本事件流	参与者事件 1.发起删除请求	系统事件 2.删除图书
前置条件	管理员登录	

实体类类图

名词分析法

对于上述每个用例

潜在类列表	是/否	类名/原因
用户	是	User
书籍	是	Book
借书信息	否	可作为 User 类的属性
图书状态	是	BookStatus
余额	否	可作为 User 类的属性

潜在类列表	是/否	类名/原因
用户	是	User

潜在类列表	是/否	类名/原因
用户	是	User
书籍	是	Book
用户信息	否	可作为 User 类的属性
图书状态	是	BookStatus

潜在类列表	是/否	类名/原因
用户	是	User
用户信息	否	可作为 User 类的属性

潜在类列表	是/否	类名/原因
图书	是	Book
图书类型	是	BookType

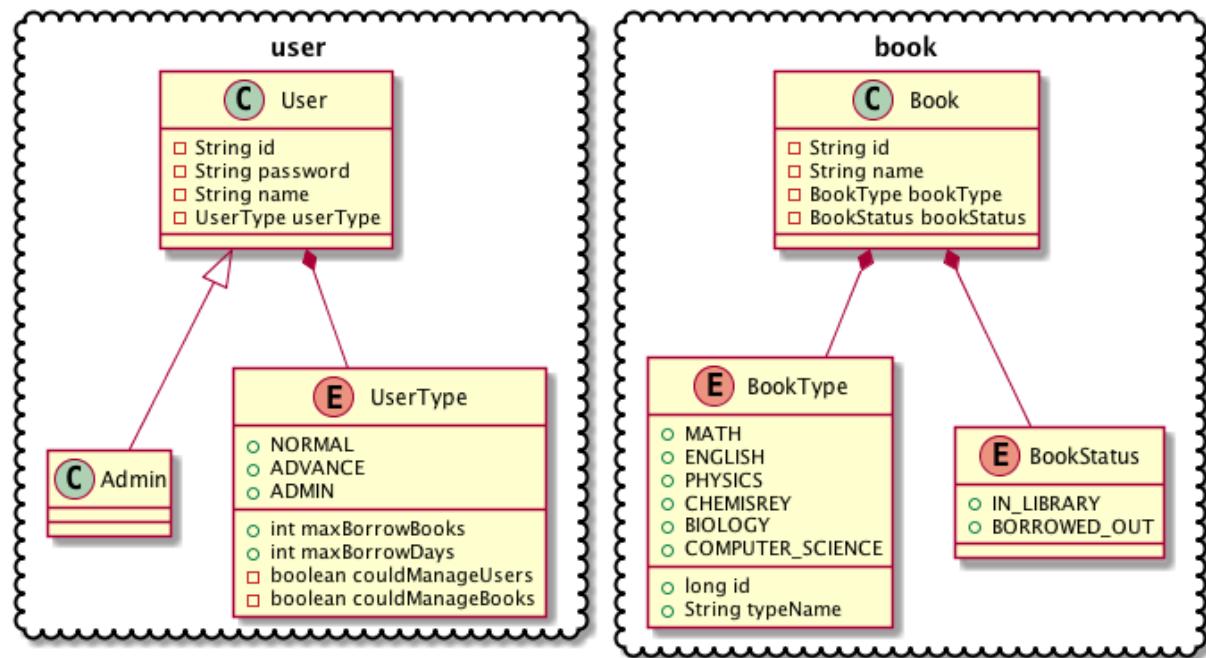
潜在类列表	是/否	类名/原因
管理员	是	Admin
表单	否	由控制器自动生成

潜在类列表	是/否	类名/原因
管理员	是	Admin
用户	是	User

潜在类列表	是/否	类名/原因
管理员	是	Admin
图书	是	Book
表单	否	无具体实例

潜在类列表	是/否	类名/原因
管理员	是	Admin
图书	是	Book

仅带有属性的实体类类图



设计模式

在 `User` 类及其子类 `Admin` 中, 使用了抽象工厂的设计模式

```
@Data
public class User{

    protected String id;

    protected String name;

    protected transient String password;

    protected UserType userType;

    protected final static UserFactory factory = new UserFactory();

    public static UserBuilder createNormalUserBuilder(){
        return factory.createNormalUser();
    }

    public static UserBuilder createAdvanceUserBuilder(){
        return factory.createAdvanceUser();
    }

    ...
}

@Data
public class Admin extends User{

    protected UserType userType = UserType.ADMIN;

    public static UserBuilder createAdminUser(){
        return factory.createAdminUser();
    }

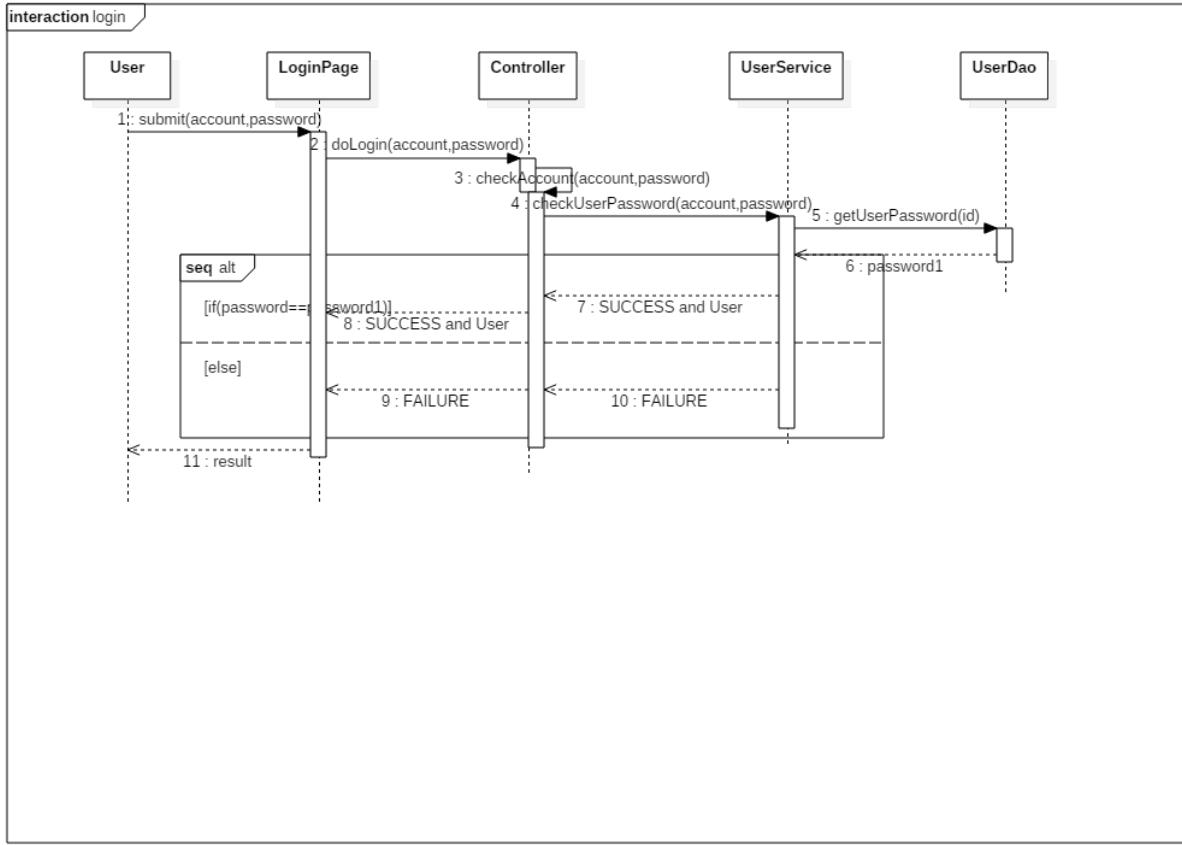
}
```

在这里使用抽象工厂方法主要是为了屏蔽 `UserBuilder` 类的 `userType(UserType type)` 方法。这样做可以屏蔽掉普通用户想要创建 `Admin` 用户的风险。

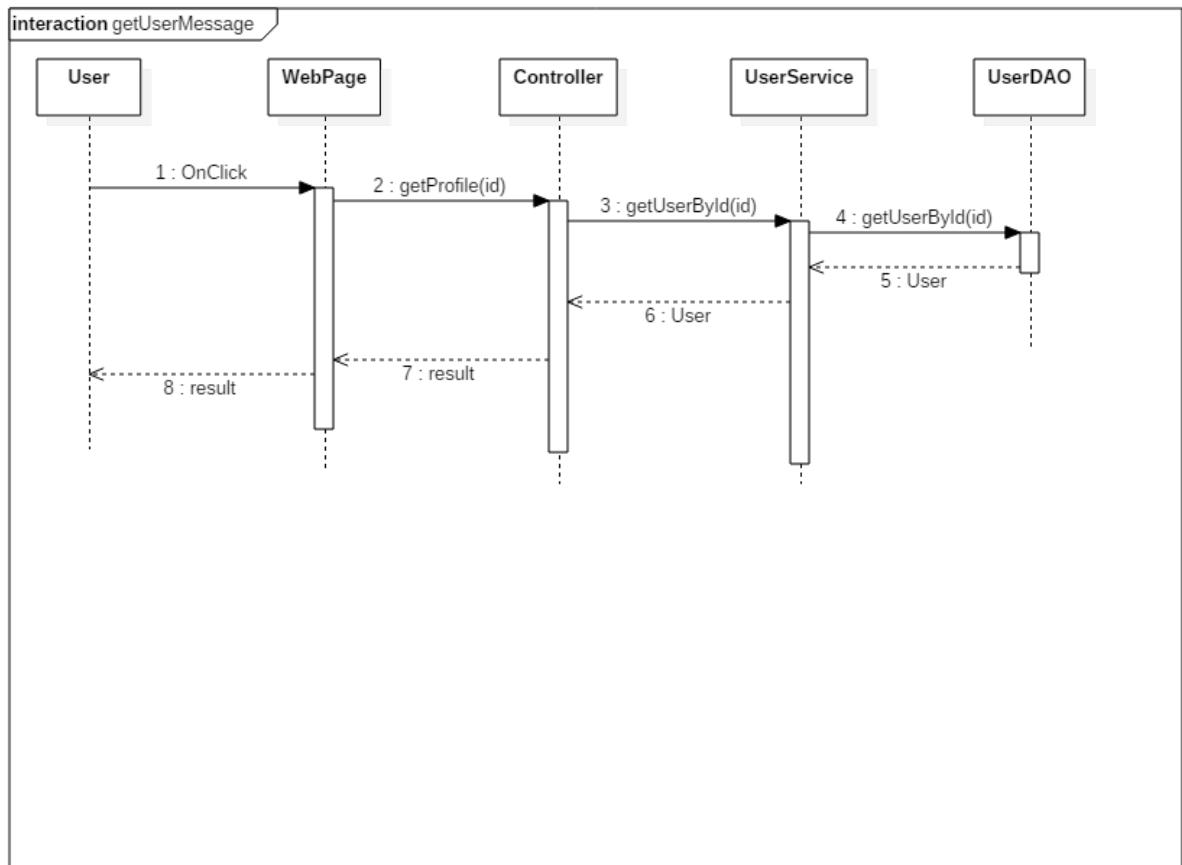
```
public class UserBuilder{  
  
    User user;  
  
    UserBuilder(){  
        user = new User();  
    }  
  
    public UserBuilder id(String o){  
        user.setId(o);  
        return this;  
    }  
  
    public UserBuilder name(String name){  
        user.setName(name);  
        return this;  
    }  
  
    public UserBuilder password(String password){  
        user.setPassword(password);  
        return this;  
    }  
  
    UserBuilder type(UserType userType){  
        user.setUserType(userType);  
        return this;  
    }  
  
    public User build(){  
        return this.user;  
    }  
}
```

顺序图

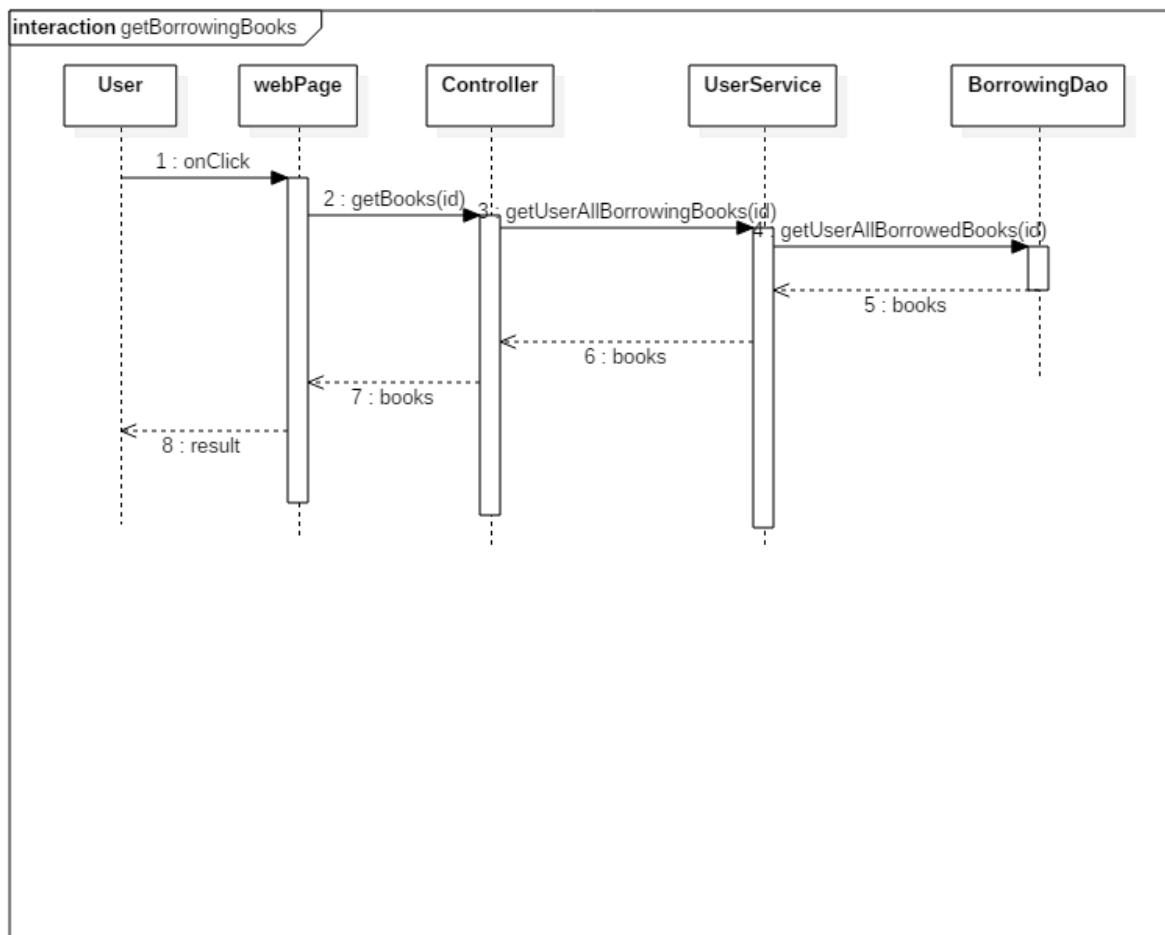
登录



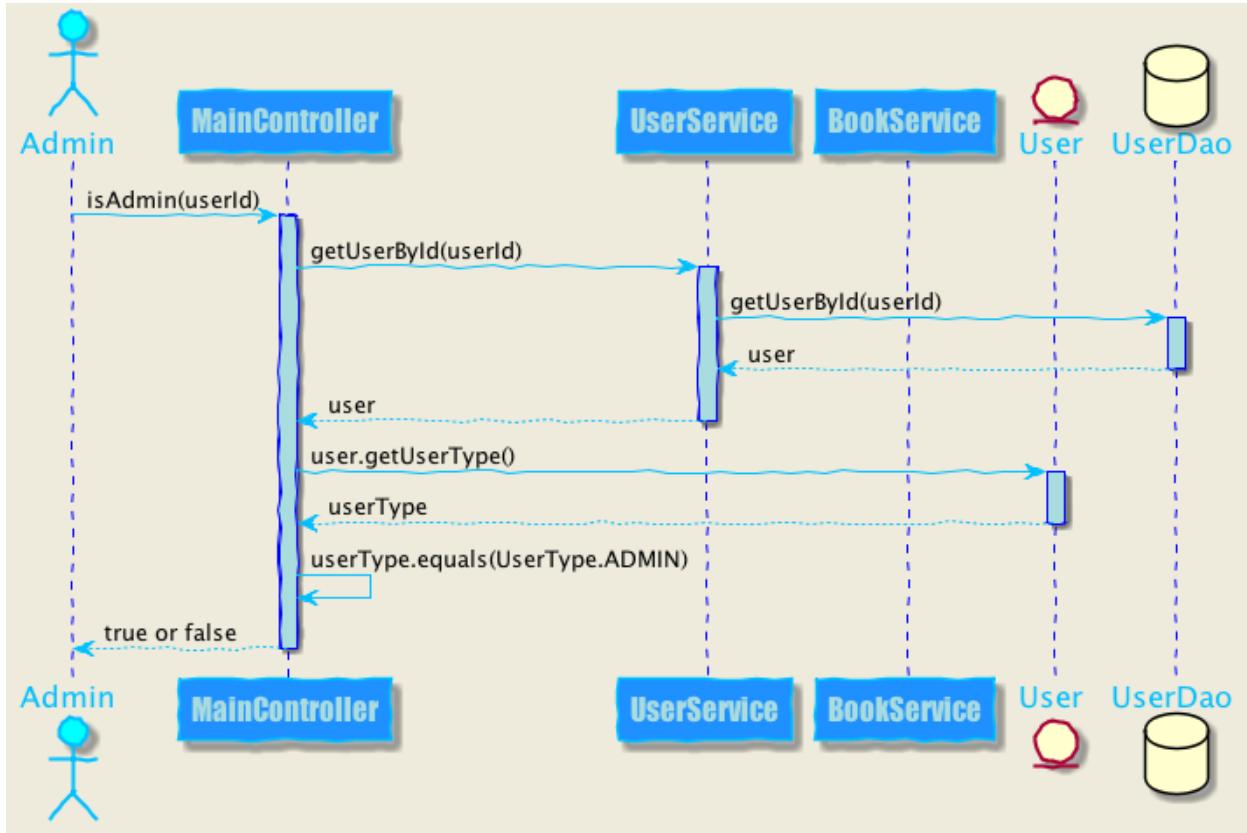
查看个人信息



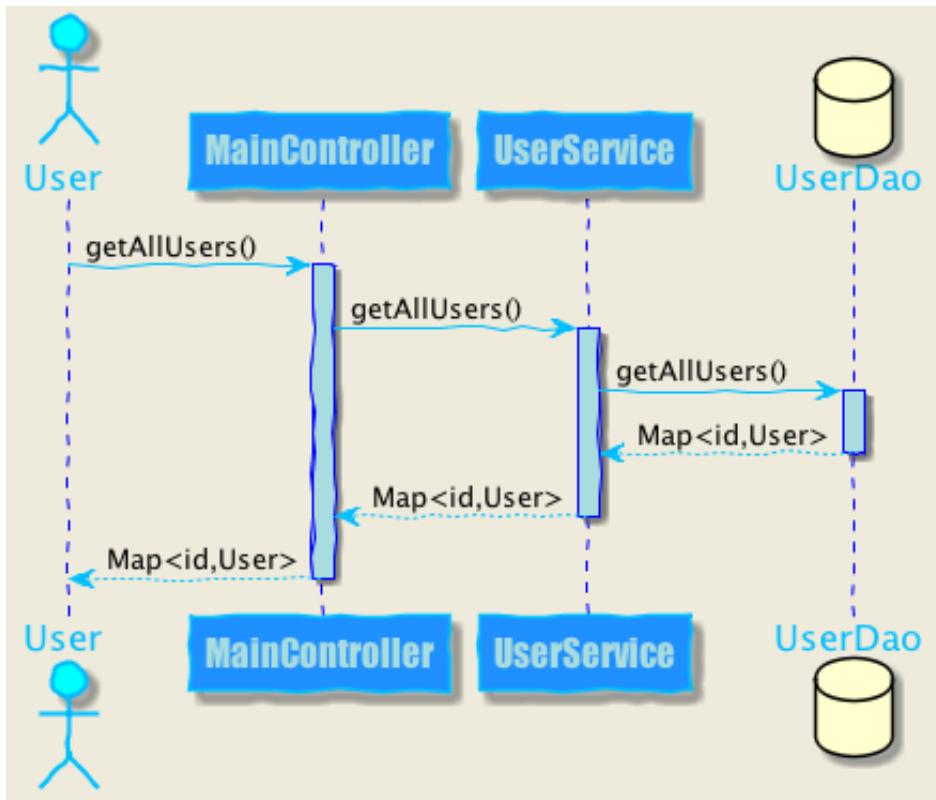
查看已借阅图书



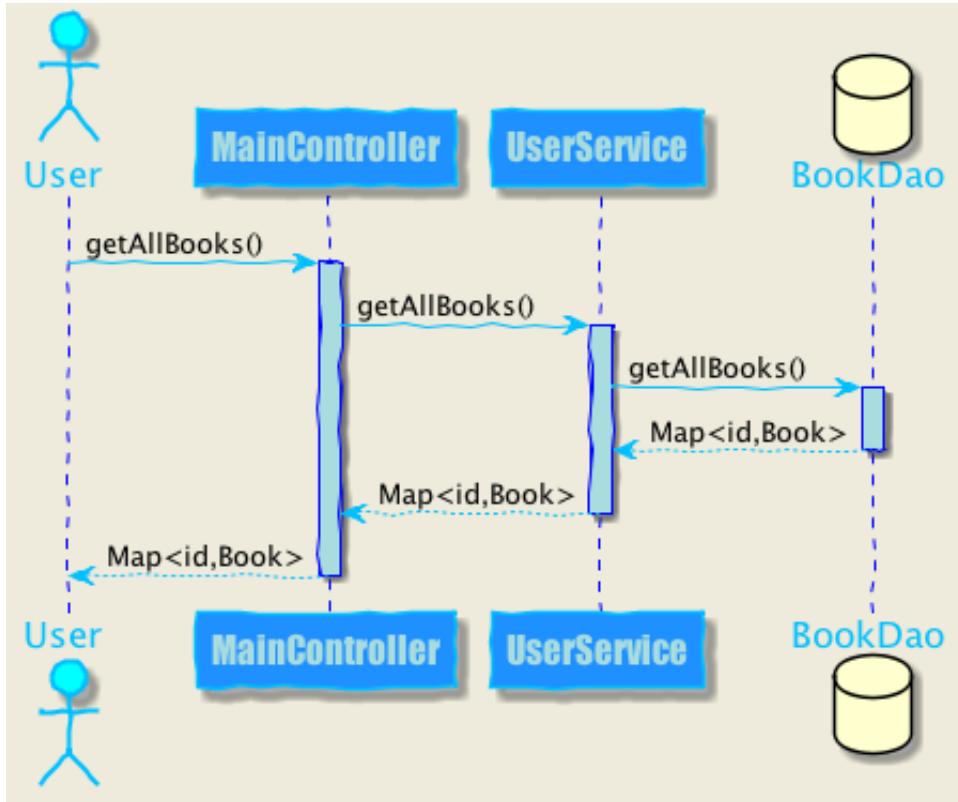
检查是否为 Admin



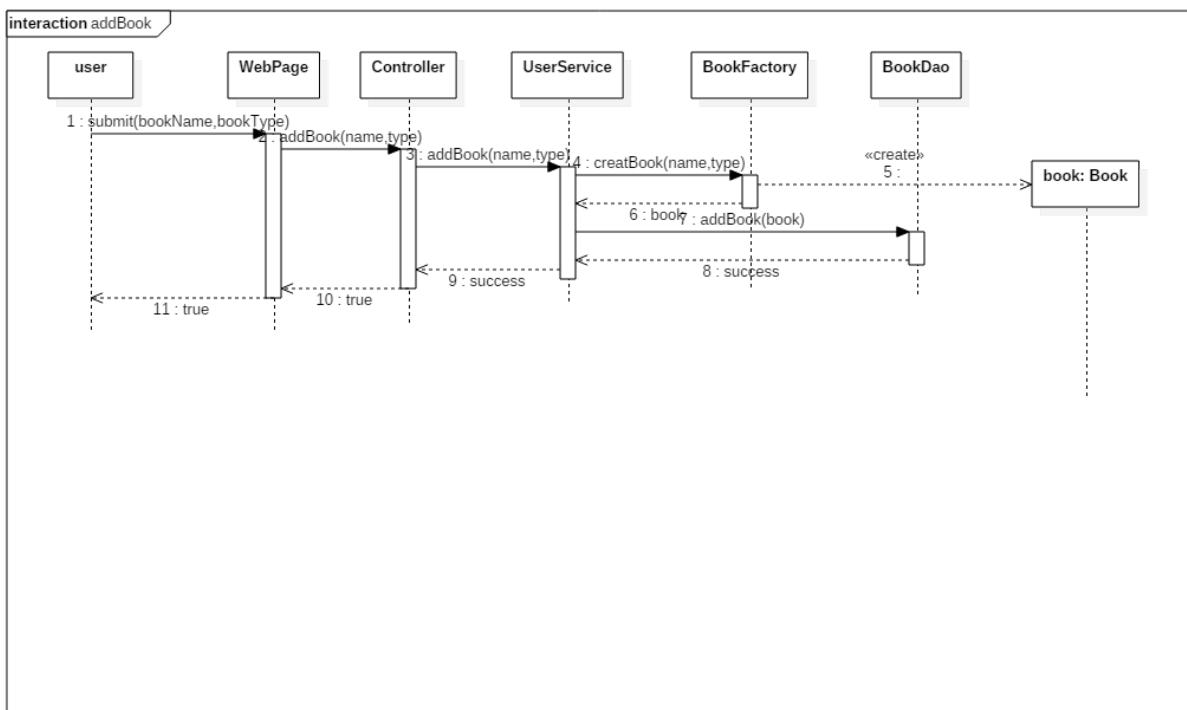
所有用户列表



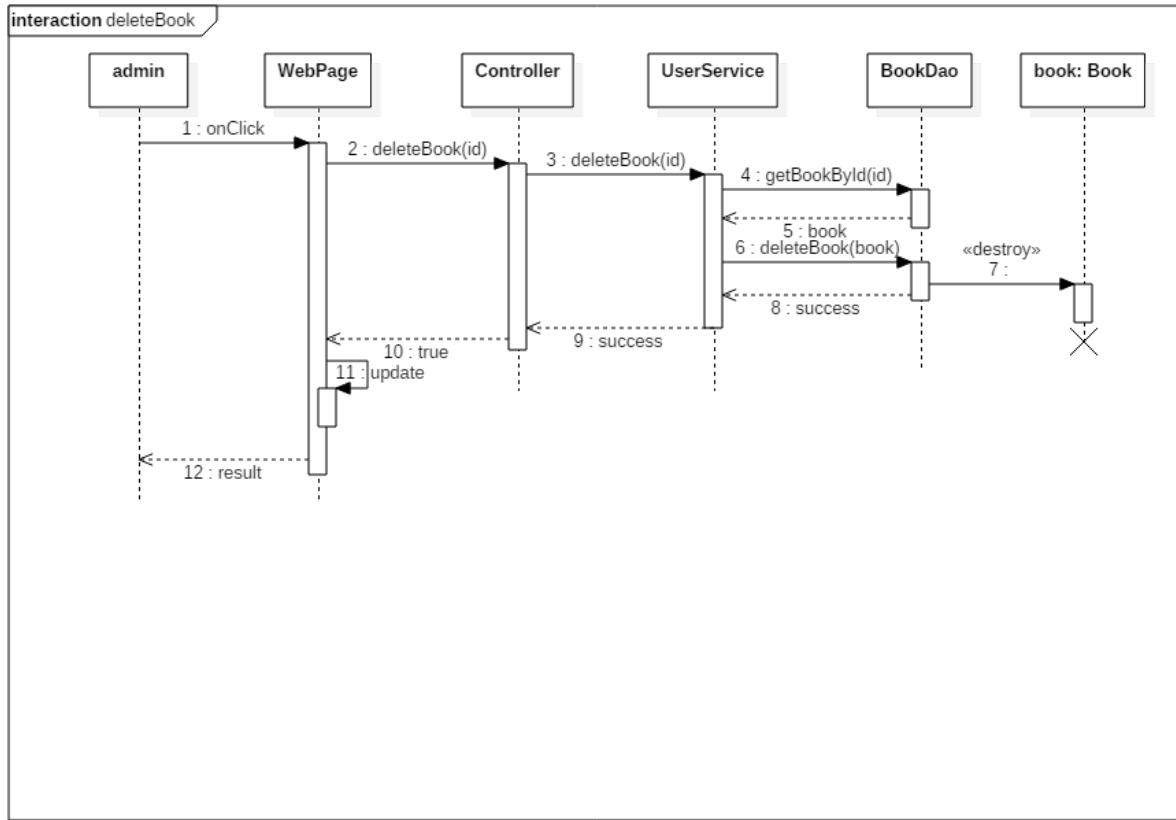
所有图书列表



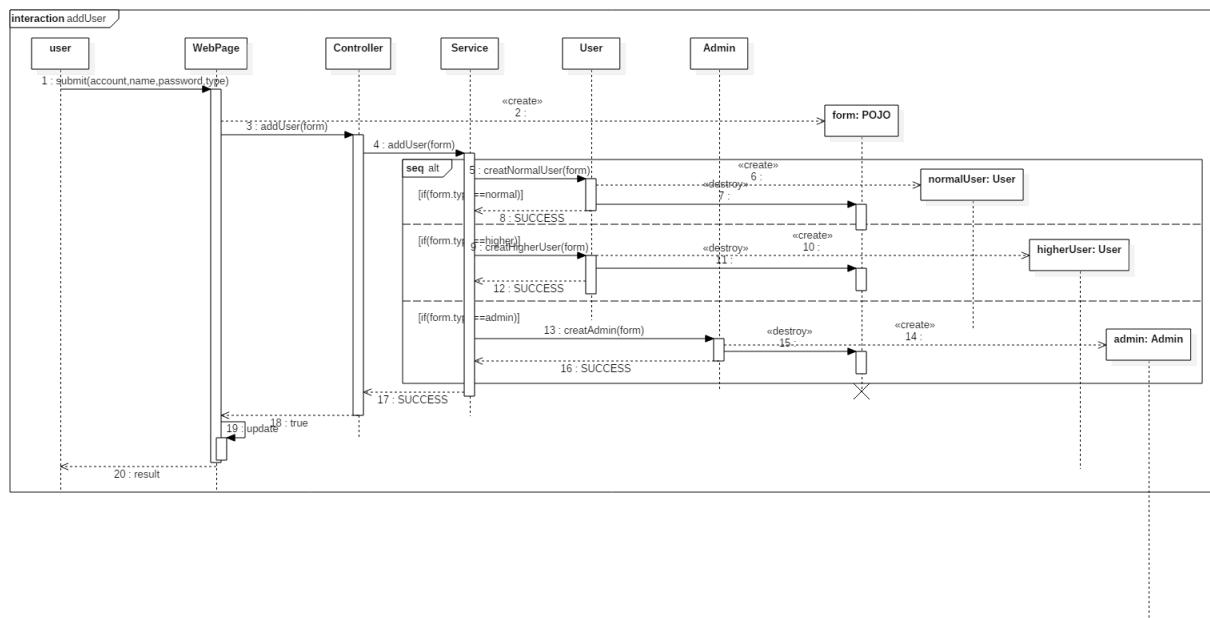
增加图书



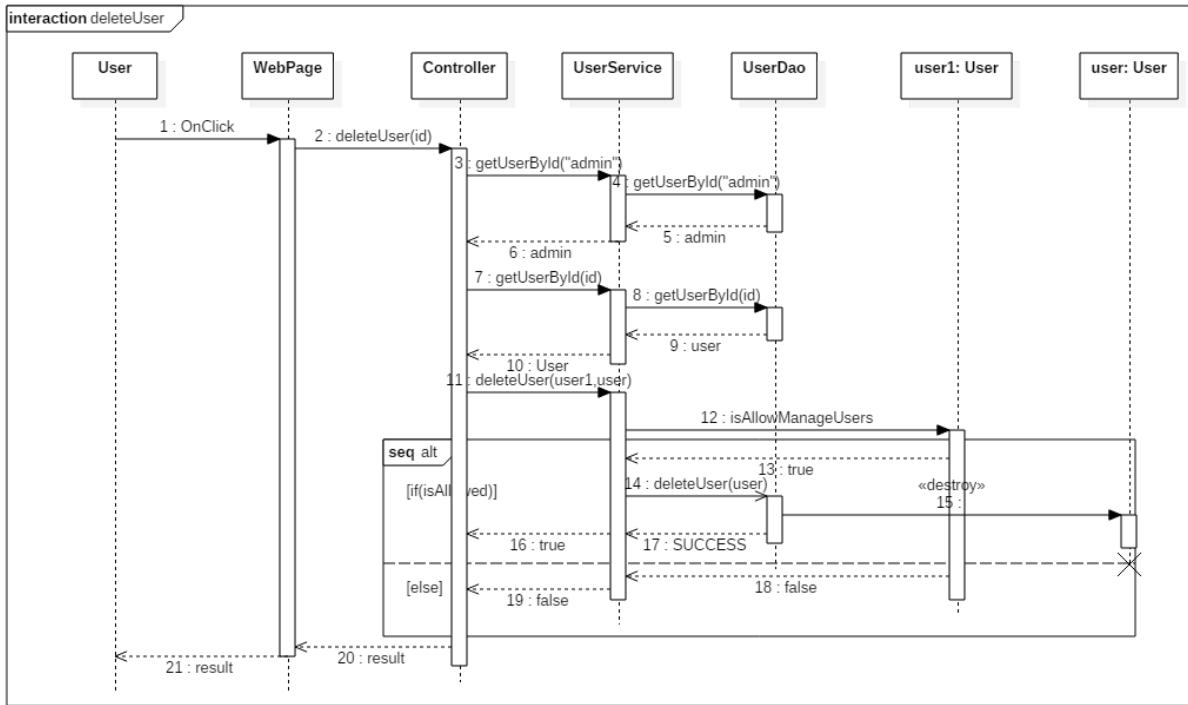
删除图书



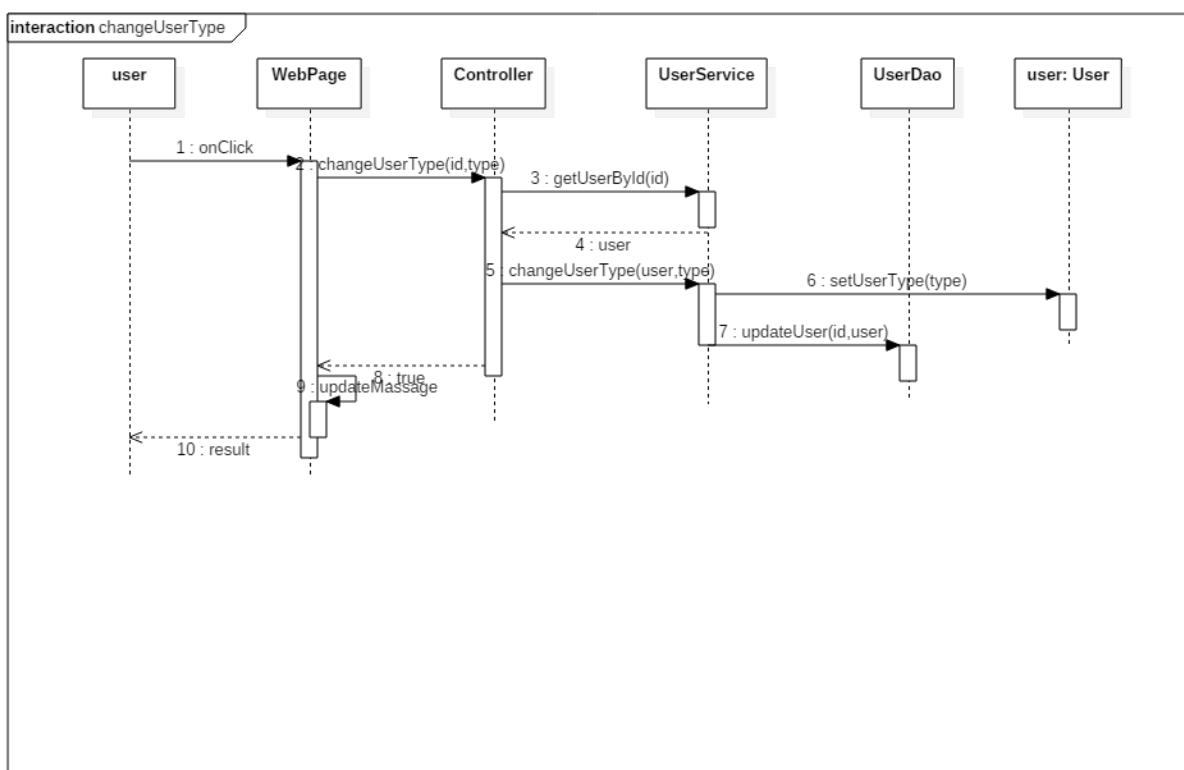
增加用户



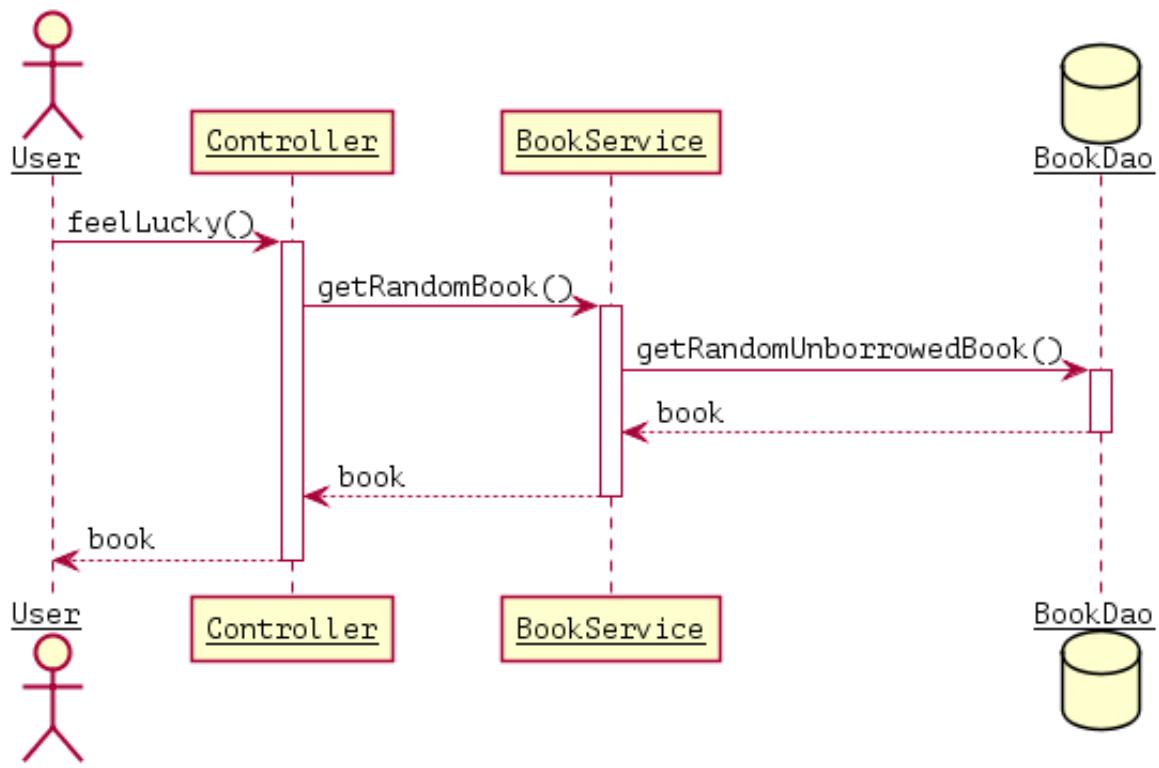
删除用户



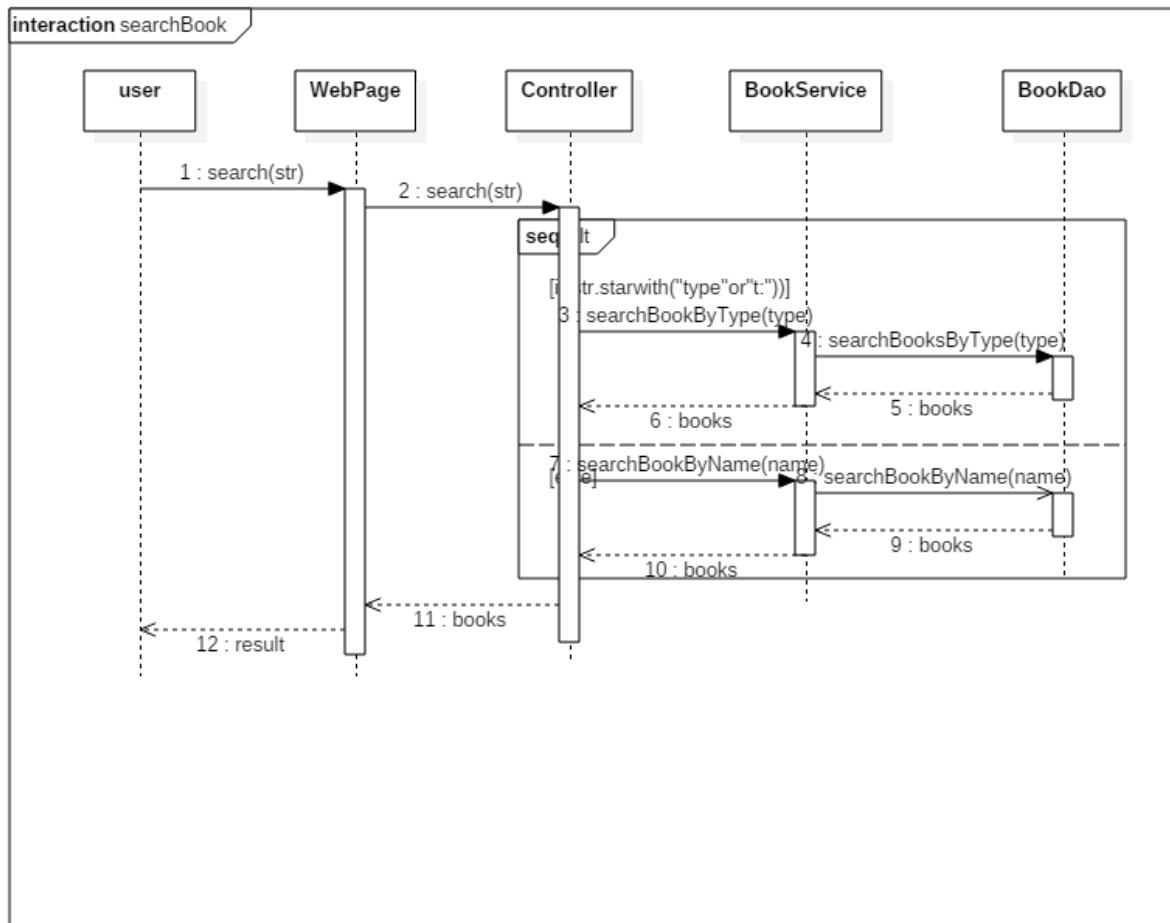
改变用户的类型



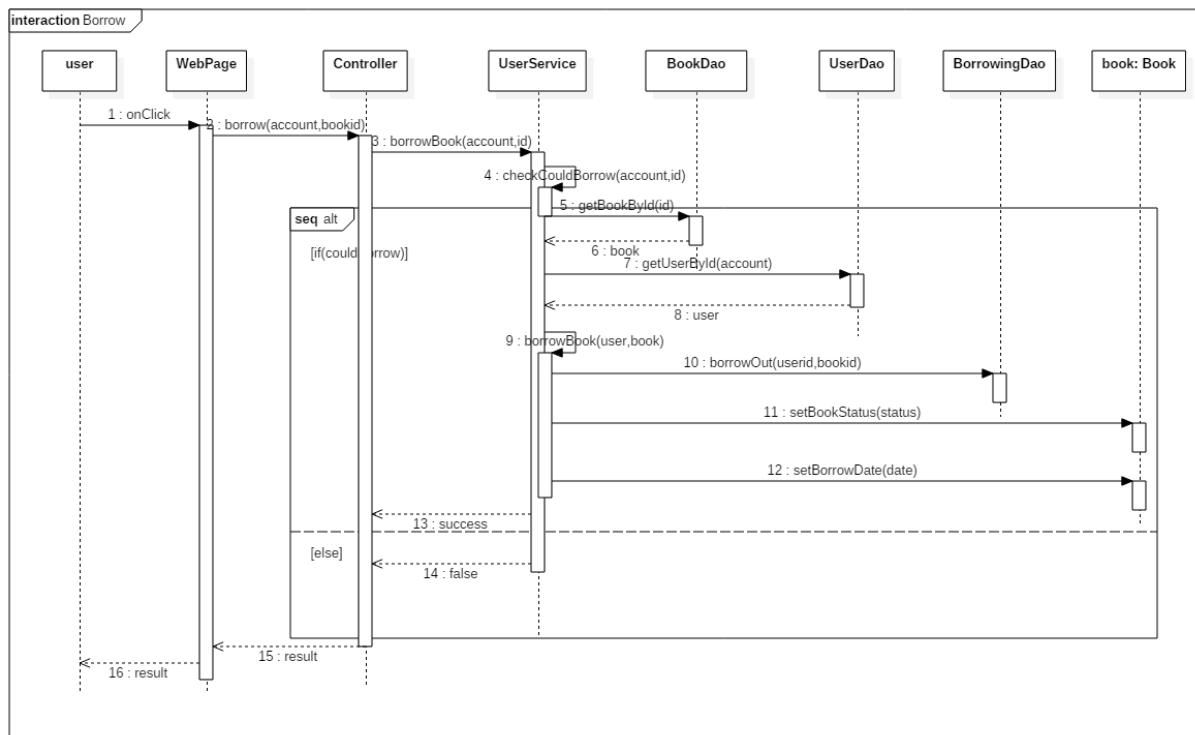
feelLucky



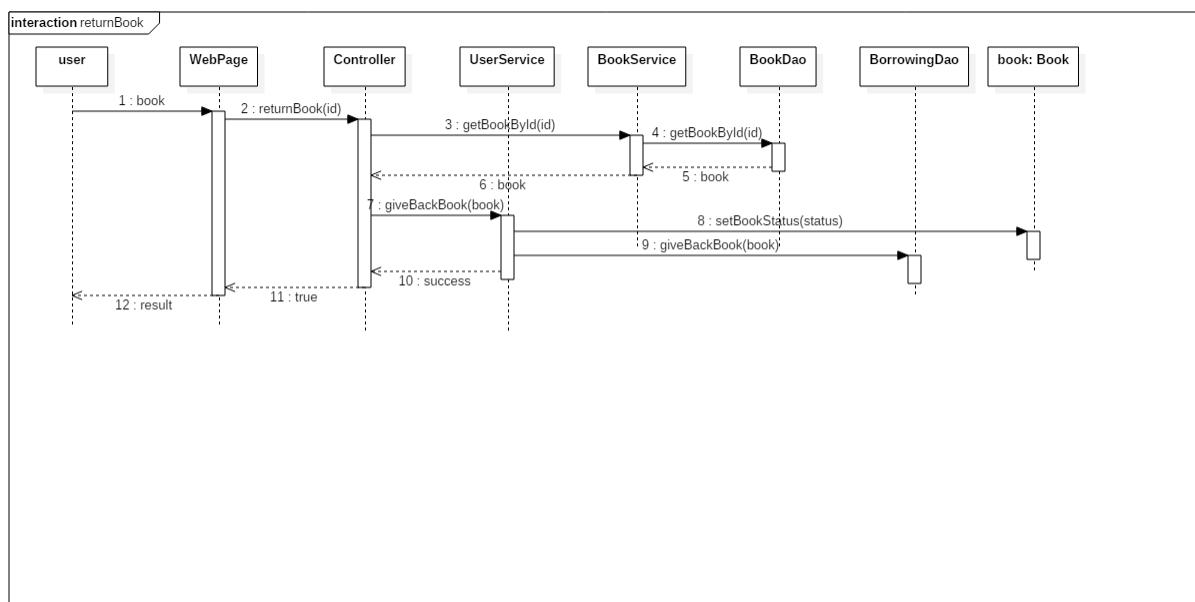
查找图书



借阅图书

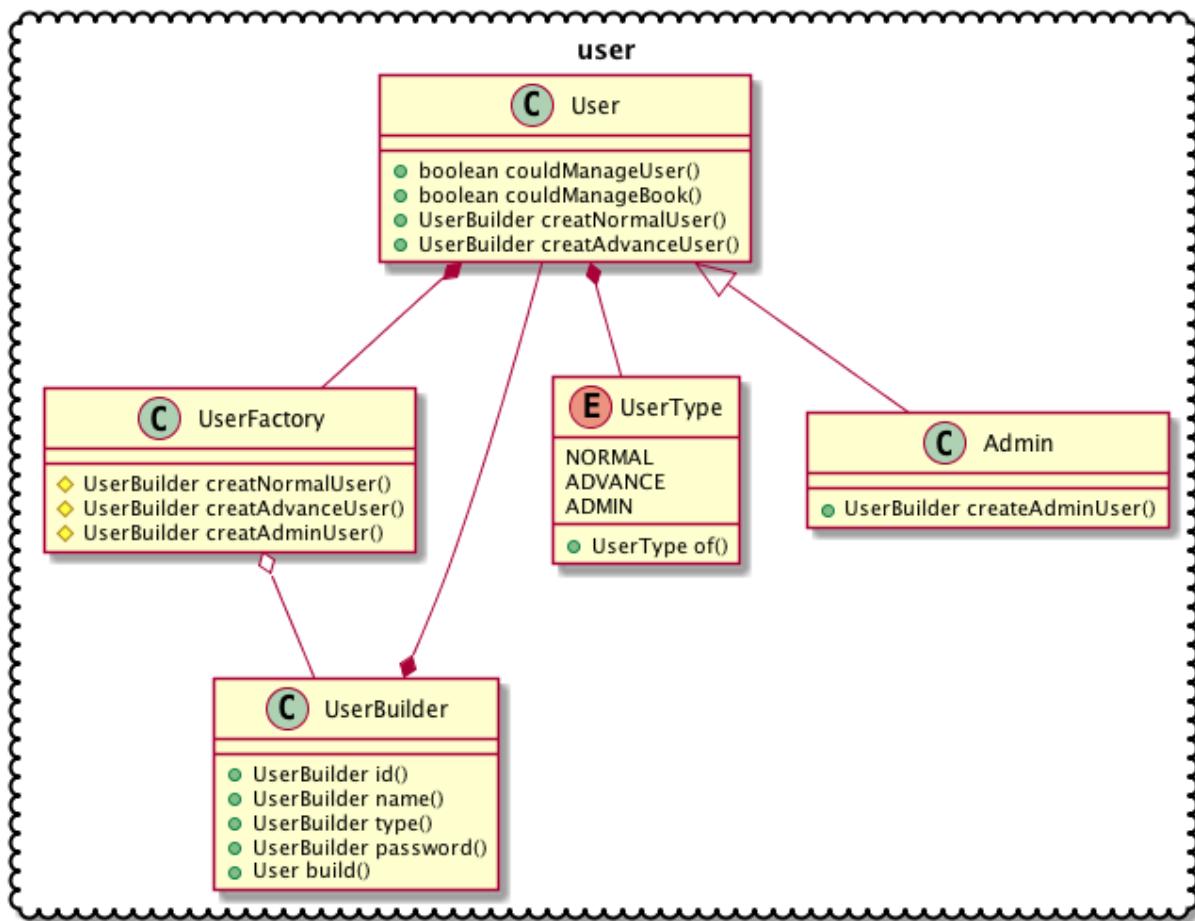


归还图书

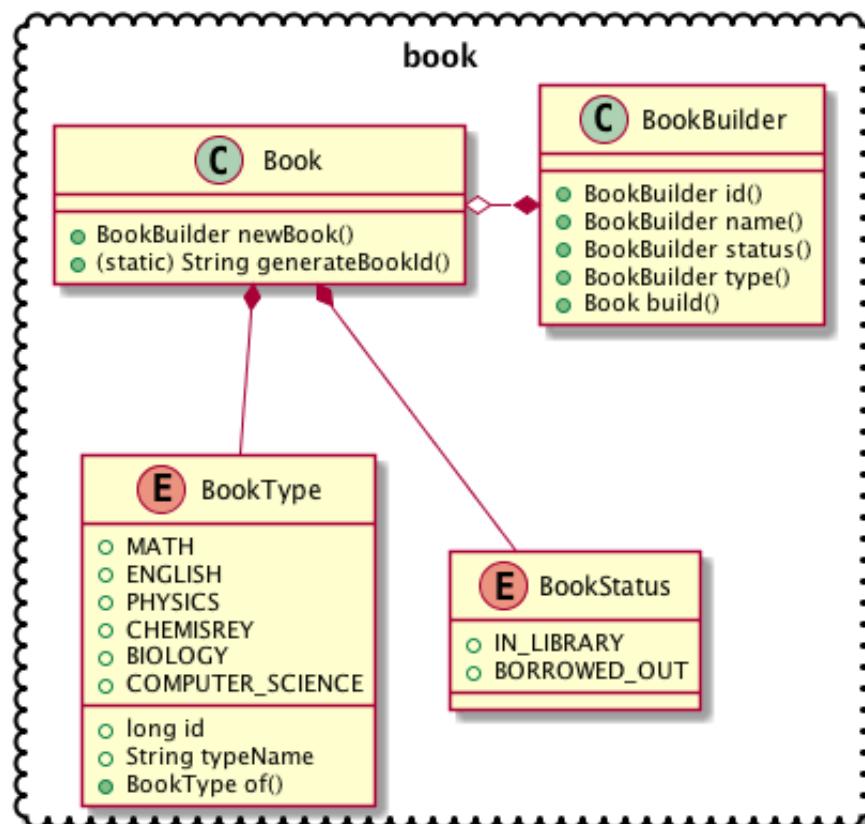


完整类图

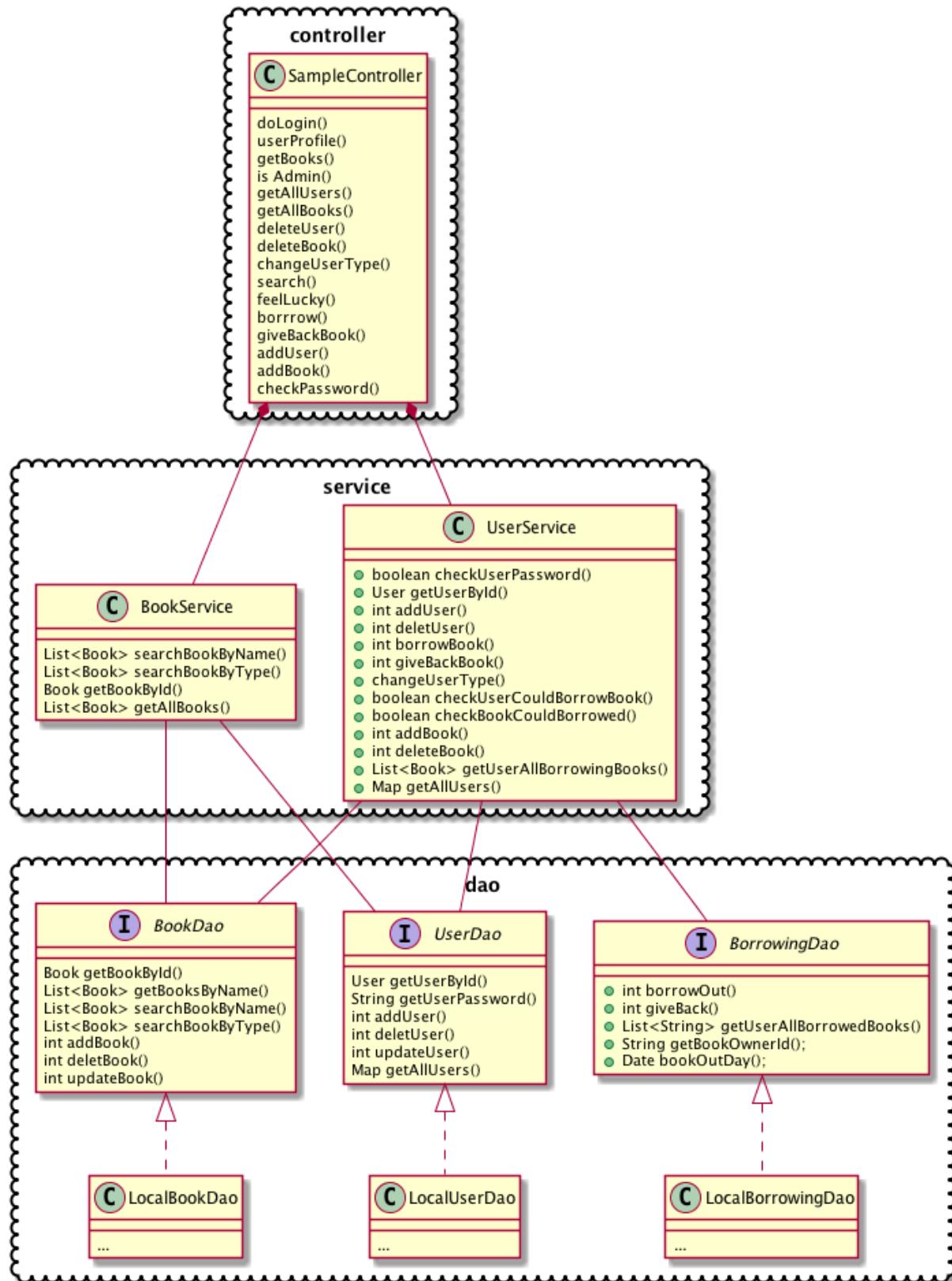
user 包类图



book 包类图



web 包类图



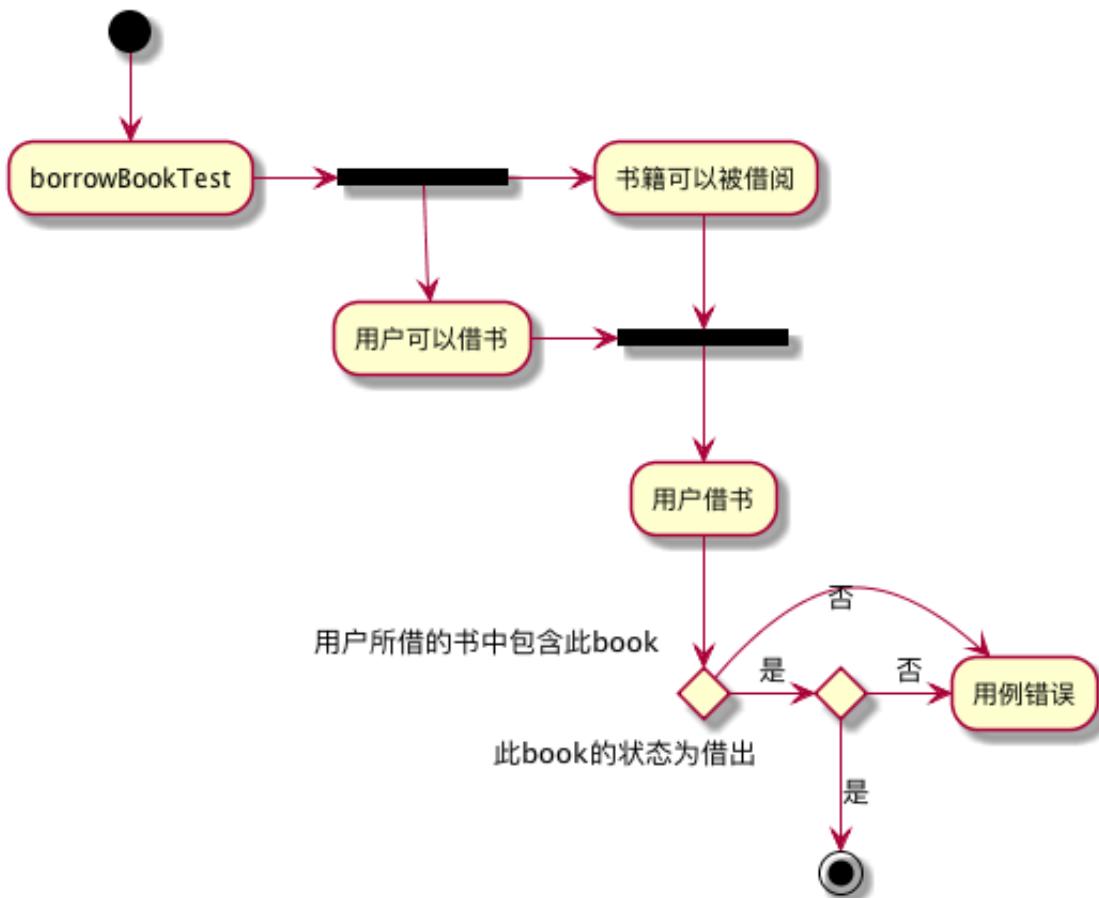
项目实现

详见 [GitHub 项目地址](#)

测试

借书测试及活动图

```
@Test  
public void borrowBookTest() {  
    userService.borrowBook("1", "CS121");  
    Book book = bookService.getBookById("CS121");  
    List ls = userService.getUserAllBorrowingBooks("1");  
    Assert.assertTrue(ls.contains(book));  
    Assert.assertTrue(book.getBookStatus().equals(BookStatus.IN_LIBRARY));  
}
```

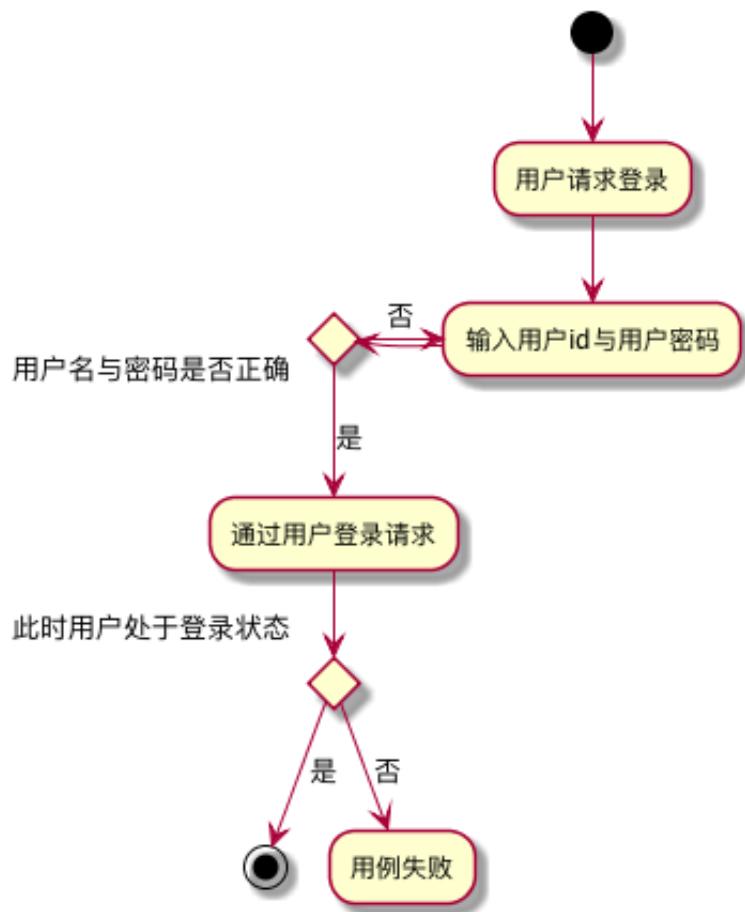


用户登录测试及活动图

```

@Test
public void userLogInTest(){
    String id = "1";
    String password = "123";
    Result r = mainController.doLogin(id, password);
    User u = (User) r.get();
    Assert.assertEquals(u, userService.getUserById(id));
    Assert.assertTrue(userService.isOnline(u));
}

```



可扩展性分析

加入修改图书信息的功能

功能描述：管理员可以修改图书的信息, 如图书 name, type 等

解决方案

1. 页面提供一个功能的入口, 让用户提交图书 id 和图书的新信息
2. Controller 收到信息后调用 UserService 检查用户是否有修改图书信息的权限
3. Controller 调用 BookService 的 updateBookInfo(Book book) 方法
4. BookService 调用 BookDao 的 updateBook(Book book) 方法
5. 修改图书信息成功

