

Oracle项目

前言

[数据清洗与测试记录文档](#)

[系统设计与实现的相关图表](#)

项目前的准备

[Git管理规范](#)

[基本准备](#)

重要记录

[功能点](#)

[待修改部分记录](#)

[数据库修改记录](#)

成员分工

[初期前端分工](#)

[初期后端分工](#)

[读者分工](#)

[8月分工](#)

[8月25日任务分配](#)

部署

前端进展

[push的流程](#)

[仓库克隆](#)

[前端vue环境搭建](#)

[前端进展](#)

后端进展

[关于小说状态的梳理和思考](#)

[关于交易的梳理和思考](#)

[前后端连通](#)

[后端搭建笔记](#)

[后端api调用问题](#)

学习笔记

[前端学习笔记](#)

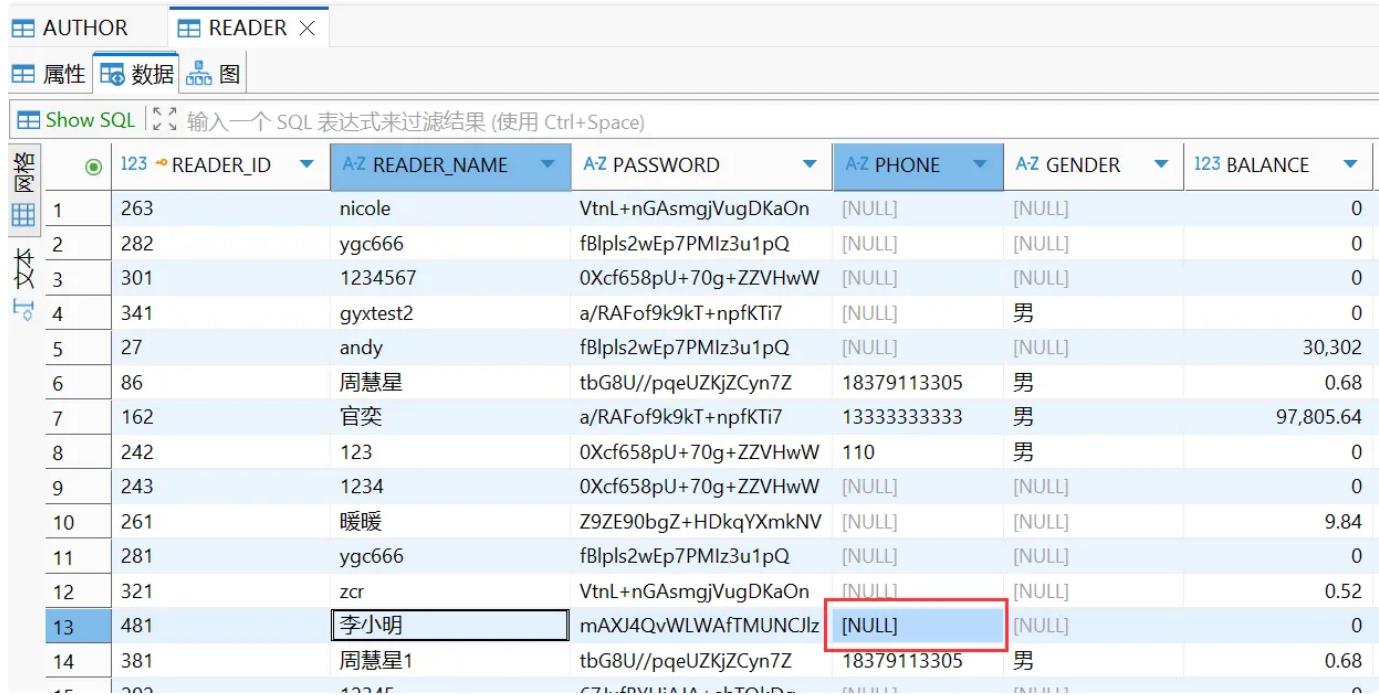
[C#学习笔记](#)

前言

此文档为我们小组暑期为完成数据库项目而创建的共享文档。大致记录我们从课程开始到答辩前的分工与项目心得。

数据清洗与测试记录文档

1. 注册后手机号未存入数据库 加上了



| | 123 ↗ READER_ID | A-Z READER_NAME | A-Z PASSWORD | A-Z PHONE | A-Z GENDER | 123 BALANCE |
|----|-----------------|-----------------|----------------------|-------------|------------|-------------|
| 1 | 263 | nicole | VtnL+nGAsmgjVugDKaOn | [NULL] | [NULL] | 0 |
| 2 | 282 | ygc666 | fBlpls2wEp7PMlz3u1pQ | [NULL] | [NULL] | 0 |
| 3 | 301 | 1234567 | 0Xcf658pU+70g+ZZVHwW | [NULL] | [NULL] | 0 |
| 4 | 341 | gyxtest2 | a/RAFof9k9kT+npfKTi7 | [NULL] | 男 | 0 |
| 5 | 27 | andy | fBlpls2wEp7PMlz3u1pQ | [NULL] | [NULL] | 30,302 |
| 6 | 86 | 周慧星 | tbG8U//pqeUZKjZCyn7Z | 18379113305 | 男 | 0.68 |
| 7 | 162 | 官奕 | a/RAFof9k9kT+npfKTi7 | 13333333333 | 男 | 97,805.64 |
| 8 | 242 | 123 | 0Xcf658pU+70g+ZZVHwW | 110 | 男 | 0 |
| 9 | 243 | 1234 | 0Xcf658pU+70g+ZZVHwW | [NULL] | [NULL] | 0 |
| 10 | 261 | 暖暖 | Z9ZE90bgZ+HDkqYXmkNV | [NULL] | [NULL] | 9.84 |
| 11 | 281 | ygc666 | fBlpls2wEp7PMlz3u1pQ | [NULL] | [NULL] | 0 |
| 12 | 321 | zcr | VtnL+nGAsmgjVugDKaOn | [NULL] | [NULL] | 0.52 |
| 13 | 481 | 李小明 | mAXJ4QvWLWAfTMUNCJz | [NULL] | [NULL] | 0 |
| 14 | 381 | 周慧星1 | tbG8U//pqeUZKjZCyn7Z | 18379113305 | 男 | 0.68 |

2. 仅修改头像无法保存修改 需要号码也填才能保存，修改前展示注册时的号码

修改个人信息



选择头像

姓名：

王小飞

电话：

|

0/11

性别：

男

▼

保存修改

4. 小说的简介在500字之内，也无法修改信息。（是不是因为不能换行，pre的时候需要注意一下，防止出错）

保存失败: Request failed with status code 500 ×

简介

唐门外门弟子唐三，因偷学内门绝学为唐门所不容，跳崖明志时却来到了另一个世界，一个属于武魂的世界。名叫斗罗大陆。
这里没有魔法，没有斗气，没有武术，却有神奇的武魂。这里的每个人，在自己六岁的时候，都会在武魂殿中令武魂觉醒。武魂有动物，有植物，有器物，它们可以辅助人们的日常生活。而其中一些特别出色的武魂却可以用来修炼，这个职业，是斗罗大陆上最为强大也是最重要的职业，——魂师。
当唐门暗器来到斗罗大陆，当唐三武魂觉醒，他能否在这片武魂的世界重塑唐门辉煌？

238/500

封面



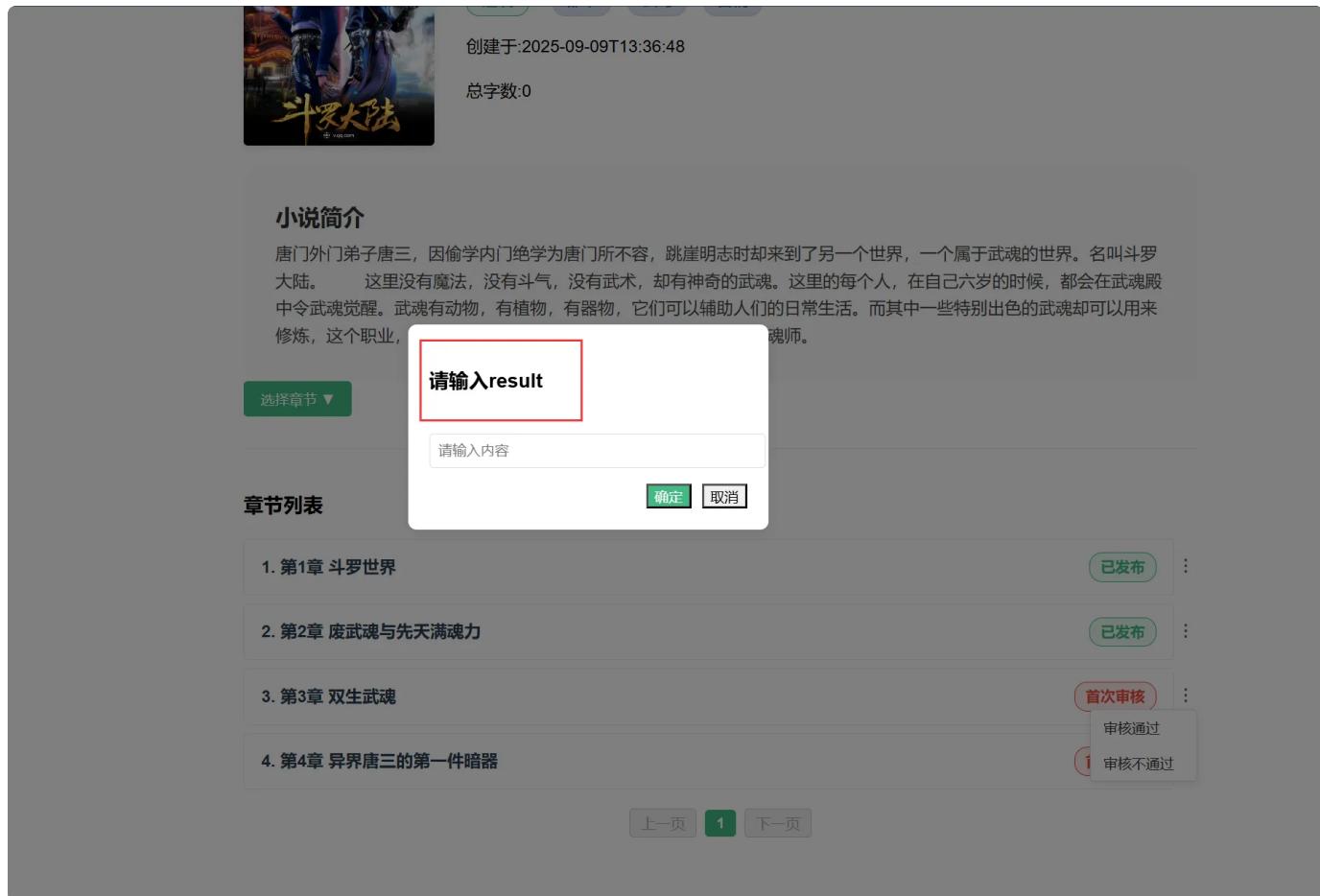
更换封面

注意

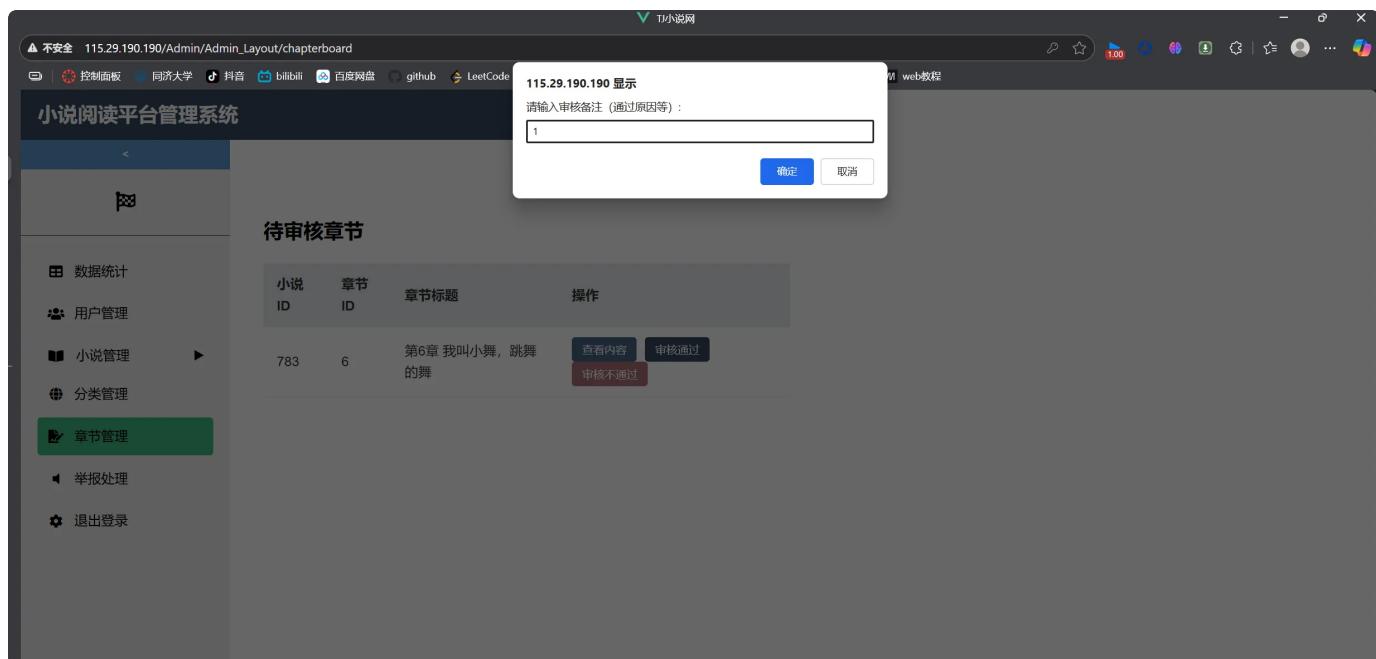
- 对于处于连载或完结状态的小说，修改上传信息后需经管理员审核方可生效。
- 请确保上传的图片及其字体等不侵犯任何人权益，如有侵权须自行承担赔偿等责任！
- 上传600x800像素、不超过5M的JPG/JPEG图片；
- 作品封面应显示作品名和笔名；
- 严禁上传色情、暴力、广告宣传或不适合公众观赏的图片，一经发现即做禁书处理；
- 封面上传后，将在2个工作日内审核完成。

修改信息

5. 审核章节的原因“result”，中英文混杂



并且审核的样式有多种，是否要统一？



6. 有时刷新作者界面，会出现DORA账户（有ysj头像版）/刷新失败（避免pre出现此类错误）

TJ小说作者管理系统


加载失败

- [作品管理](#)
- [数据统计](#)
- [收益中心](#)
- [个人设置](#)
- [退出登录](#)

近期创作活动 >

AI生成

近期征文活动

参与活动赢取创作奖励和推广资源

近期征文活动

参与活动赢取创作奖励和推广资源



我的小说

[+ 创建小说](#)



连载

斗罗大陆

152478字

0

TJ小说作者管理系统


DORA

- [作品管理](#)
- [数据统计](#)
- [收益中心](#)
- [个人设置](#)
- [退出登录](#)

[返回书籍](#)

凡人修仙传 - 章节管理

[+ 写新章节](#)
[保存](#)
[提交更改](#)

搜索章节...

| | |
|----------------------|------------------------------------|
| 第1章 山边小村 | 2162字 已发布 ¥0 2025-09-09 13:56 |
| 审核记录 | |
| 第2章 青牛镇 | 1715字 已发布 ¥0.8575 2025-09-09 13:56 |
| 审核记录 | |
| 第3章 七玄门 | 1484字 已发布 ¥0.742 2025-09-09 13:56 |
| 审核记录 | |
| 第4章 炼骨崖 | 2129字 已发布 ¥1.0645 2025-09-09 13:56 |
| 审核记录 | |

第1章 山边小村

二愣子睁大着双眼，直直望着茅草和烂泥糊成的黑屋顶，身上盖着的旧棉被，已呈深黄色，看不出原来的本来面目，还若有若无的散发着淡淡的霉味。

在他身边紧挨着的另一人，是二哥韩立，酣睡的十分香甜，从他身上不时传来轻重不一的阵阵打呼声。

离床大约半丈远的地方，是一堵黄泥糊成的土墙，因为时间过久，墙壁上裂开了几丝不起眼的细长口子，从这些裂纹中，隐隐约约的传来韩母唠唠叨叨的埋怨声，偶尔还掺杂着韩父，抽旱烟杆的“啪嗒”“啪嗒”吸烟声。

二愣子缓缓的闭上已有些发涩的双目，迫使自己尽早进入深深的睡梦中。他心里非常清楚，再不老实入睡的话，明天就无法早起了，也就无法和其他好的同伴一起进山拣干柴。

二愣子姓韩名立，这么像模像样的名字，他父母可取不出来，这是他父亲用两个粗粮制成的窝头，求村里老张叔给取的名字。

老张叔年轻时，曾经跟城里的有钱人当过几年的伴读书童，是村里唯一认识几个字的读书人，村里小孩子的名字，倒有一多半是他给取的。

韩立被村里人叫作“二愣子”，可人并不是真愣真傻，反而是村中首屈一指的聪明孩子，但就像其他村中的孩子一样，除了家里人外，他就很少听到有人正式叫他名字“韩立”，倒是“二愣子”“二愣子”的称呼一直伴随着他。

而之所以被人起了个“二愣子”的绰号，也只不过是村里已有一个叫“愣子”的孩子了。

这也没啥，村里的其他孩子也是“狗娃”“二蛋”之类的被人一直称呼着，这些名字也不见得比“二愣子”好听到哪里去。

因此，韩立虽然并不喜欢这个称呼，但也只能这样一直的自我安慰着。

韩立外表长得很不起眼，皮肤黑黑的，就是一个普通的农家小孩模样。但他的内心深处，却比同龄人早熟了许多，他从小就向往外面世界的富饶繁华，梦想有一天，他能走出这个巴掌大的村子，去看看老张叔经常所说的外面世界。

7. 只有系统字体首行缩进两字符

读者账号

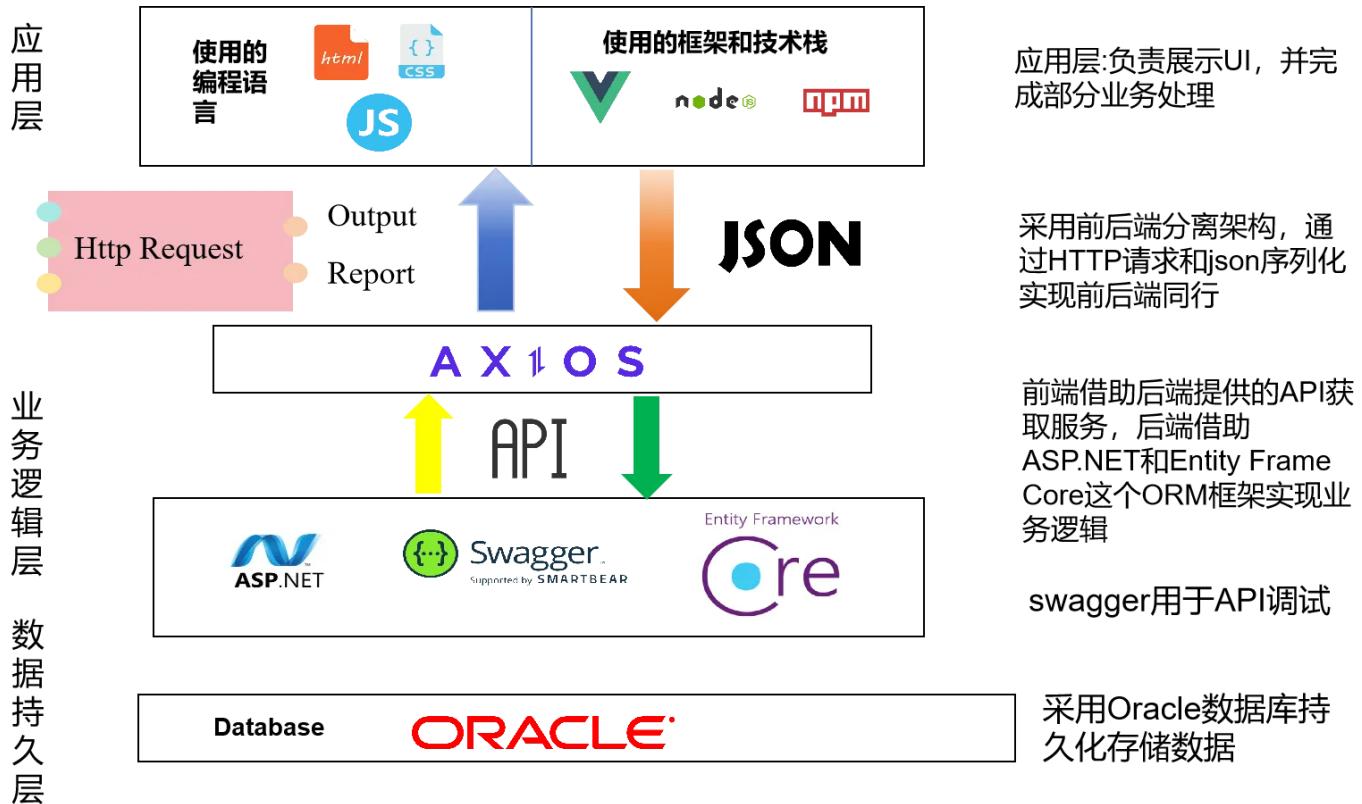
1. 481 李大明
2. 484 赵二花
3. 485 王小飞

作者账号 123456

1. 461 忘语
2. 462 唐家三少
3. 463 千山茶客

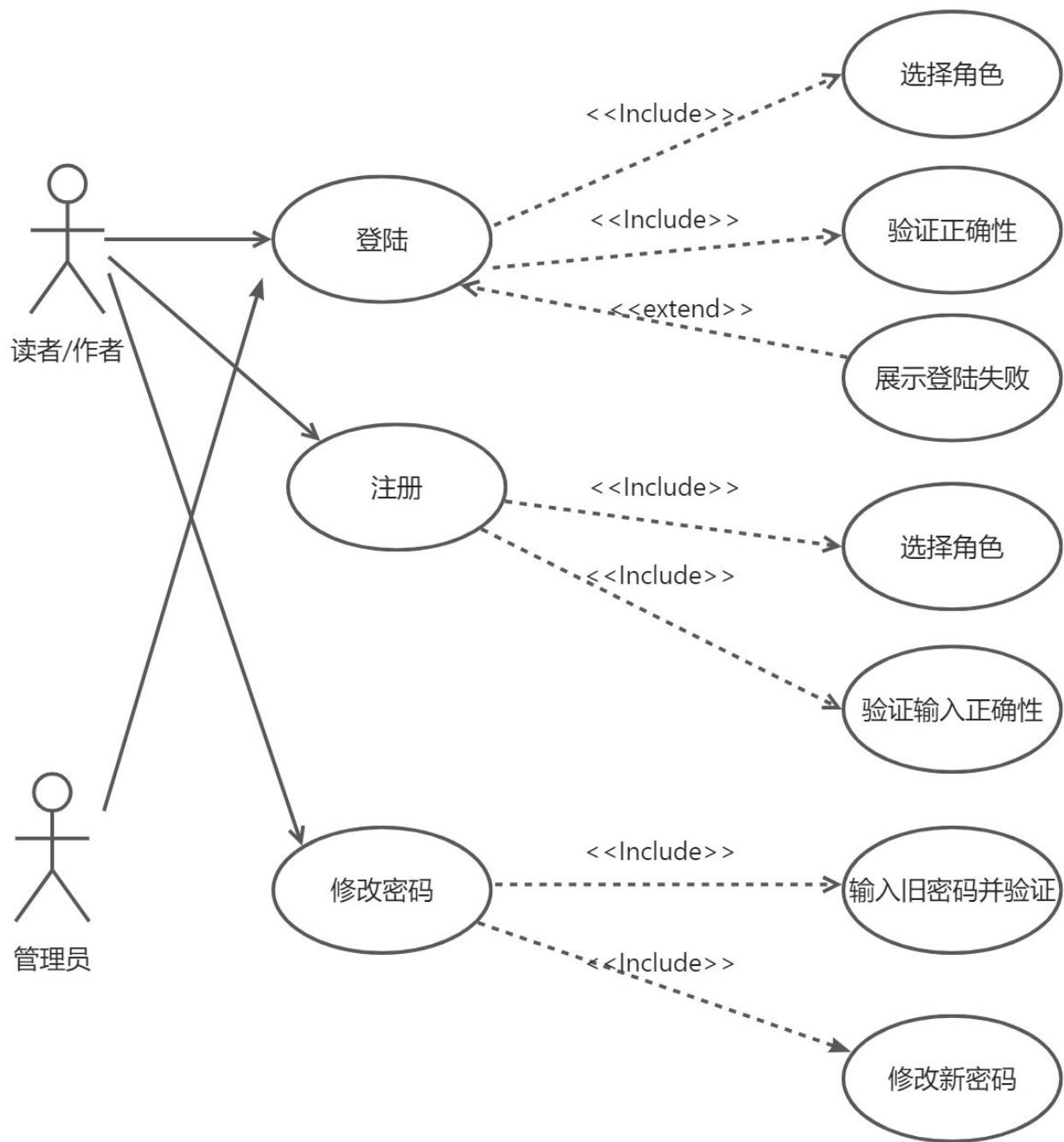
系统设计与实现的相关图表

一. 系统总体设计图

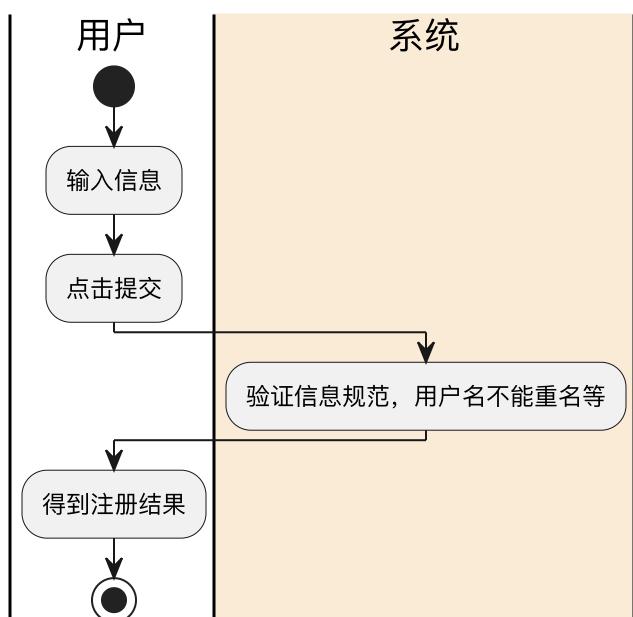
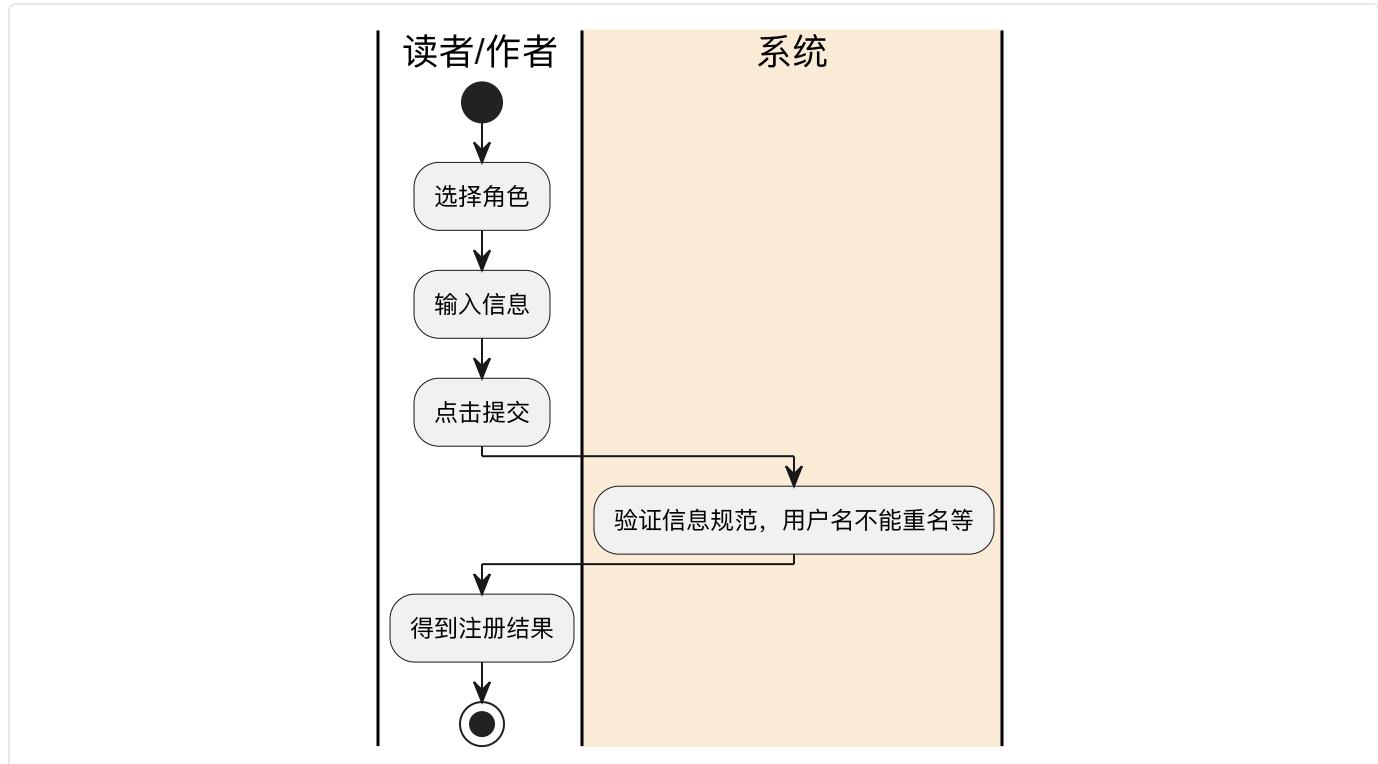


二. 用户登录注册子系统

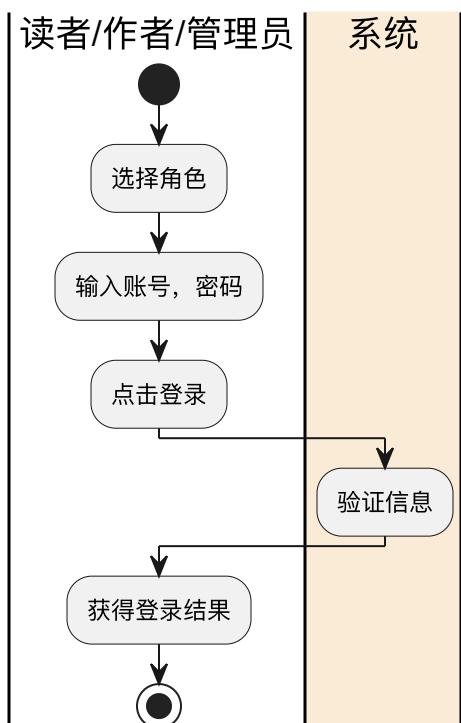
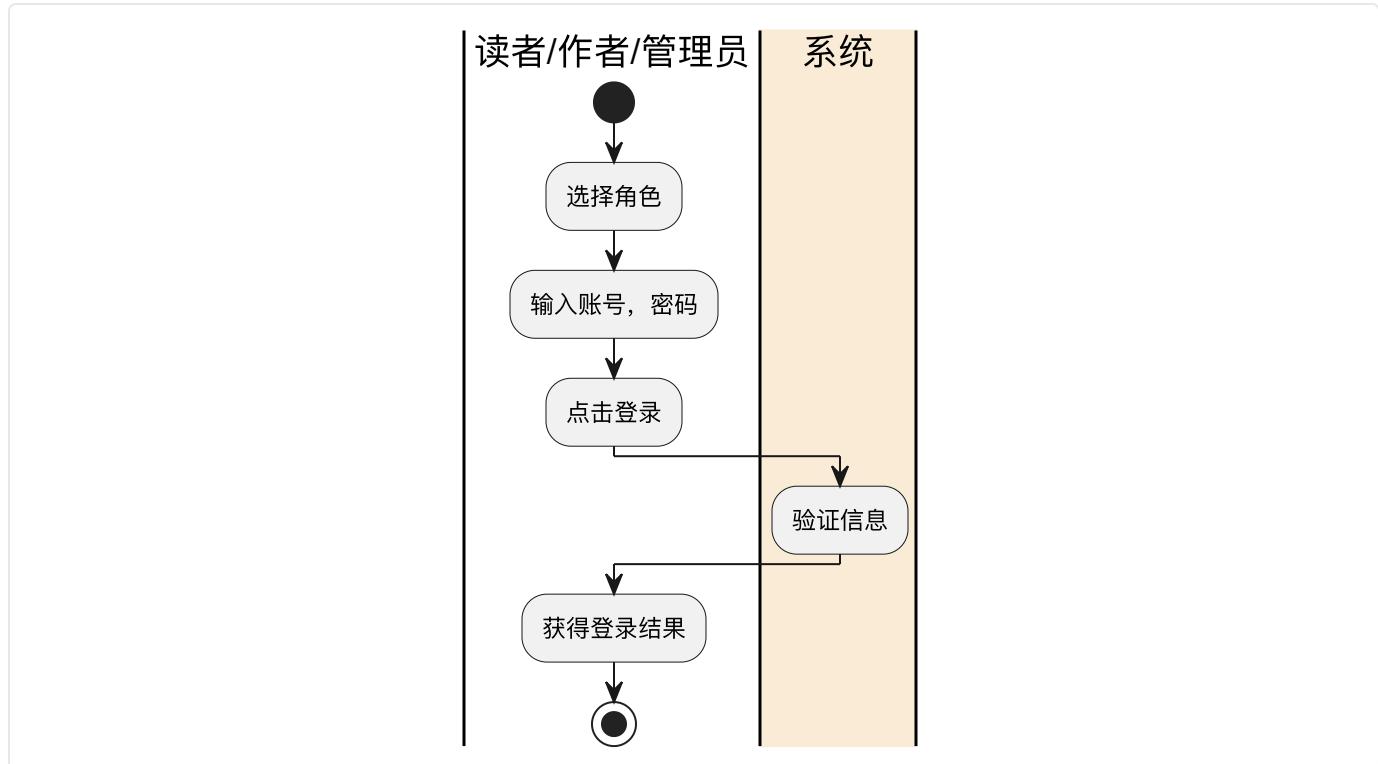
1. 用户登录注册子系统用例图



2. 用户注册用例活动图

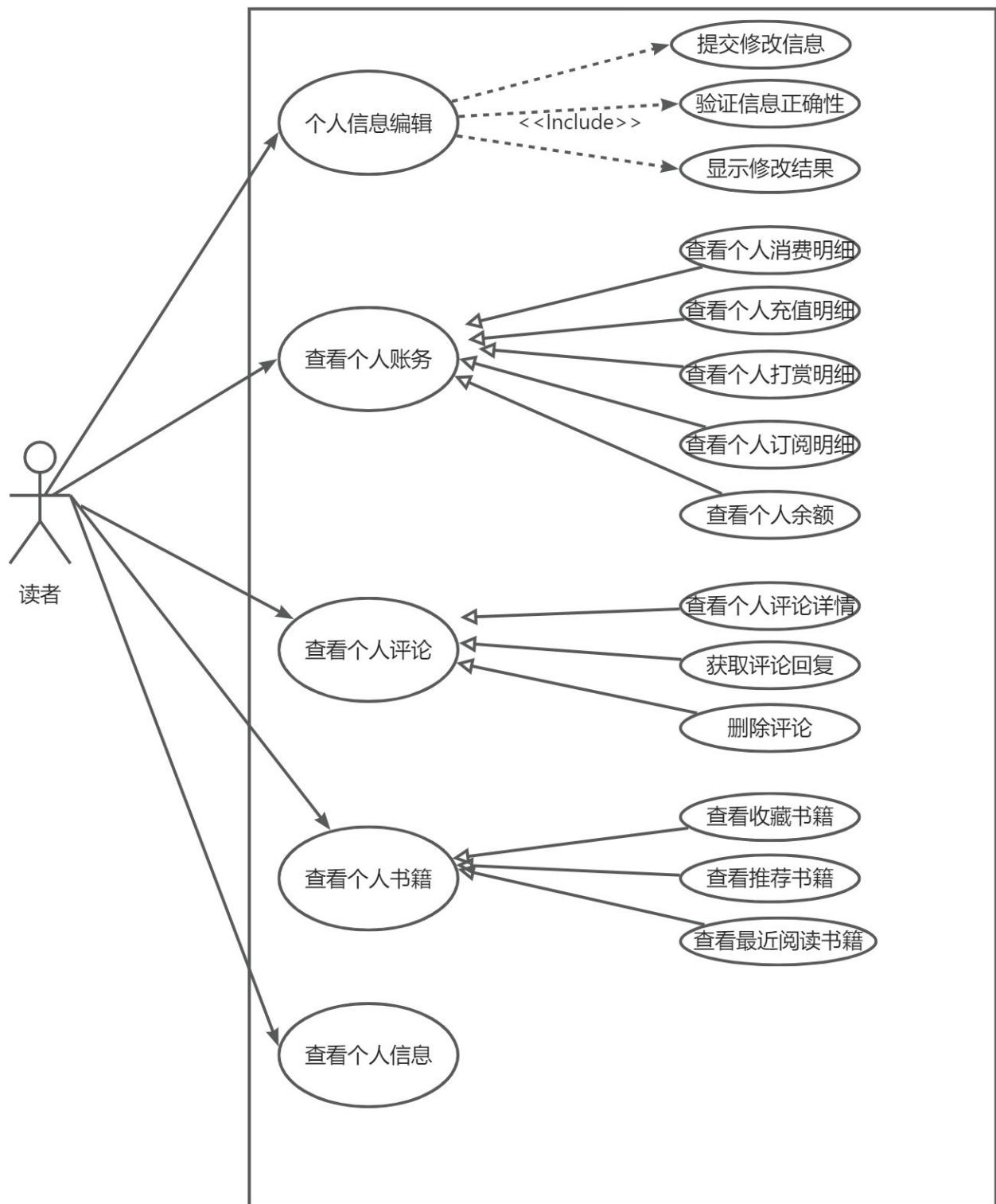


3. 用户登陆用例活动图

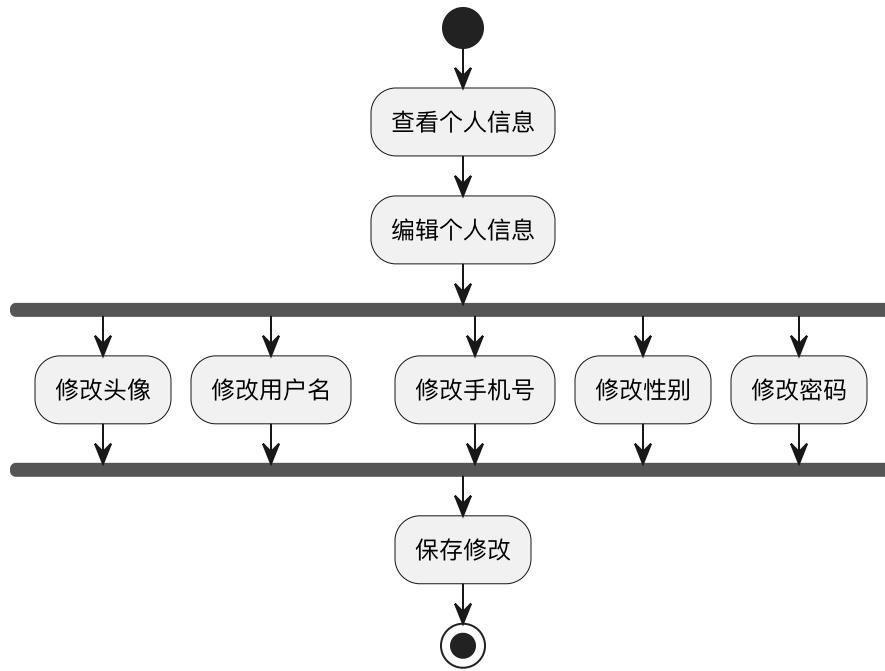


三. 读者个人中心子系统

1. 读者个人中心子系统用例图

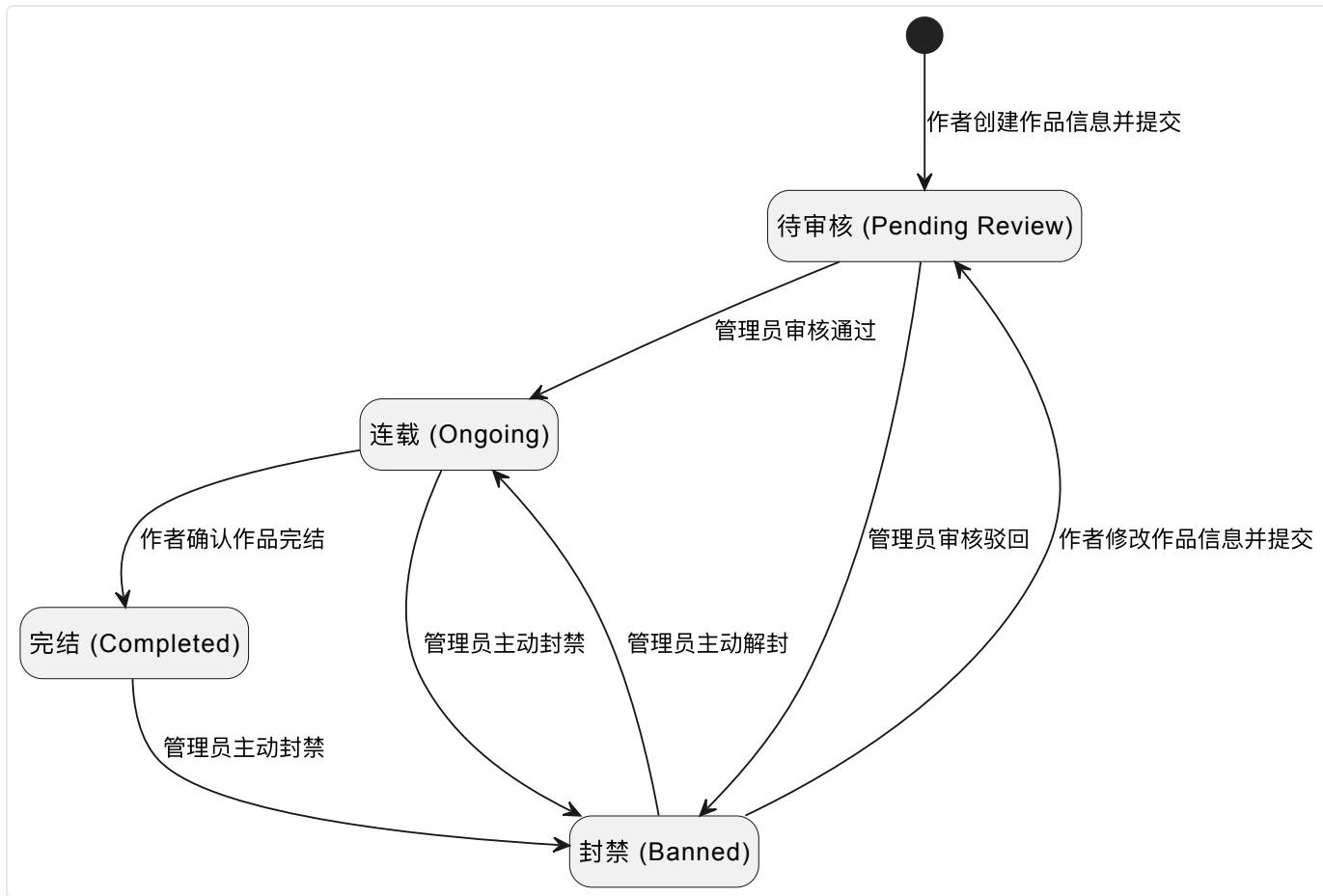


2. 编辑读者个人信息用例流程图

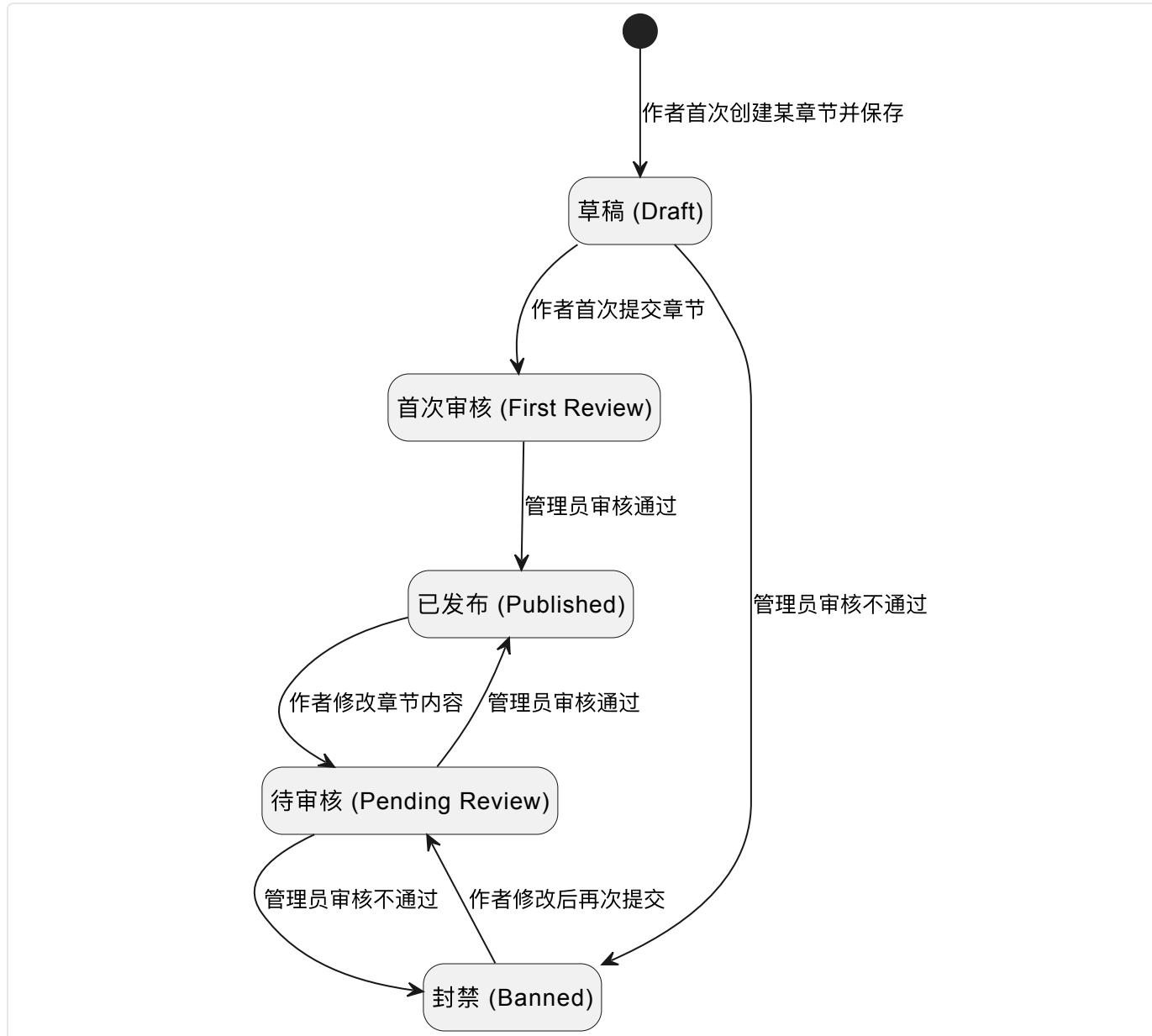


四. 小说子系统

1. 小说作品状态转换图

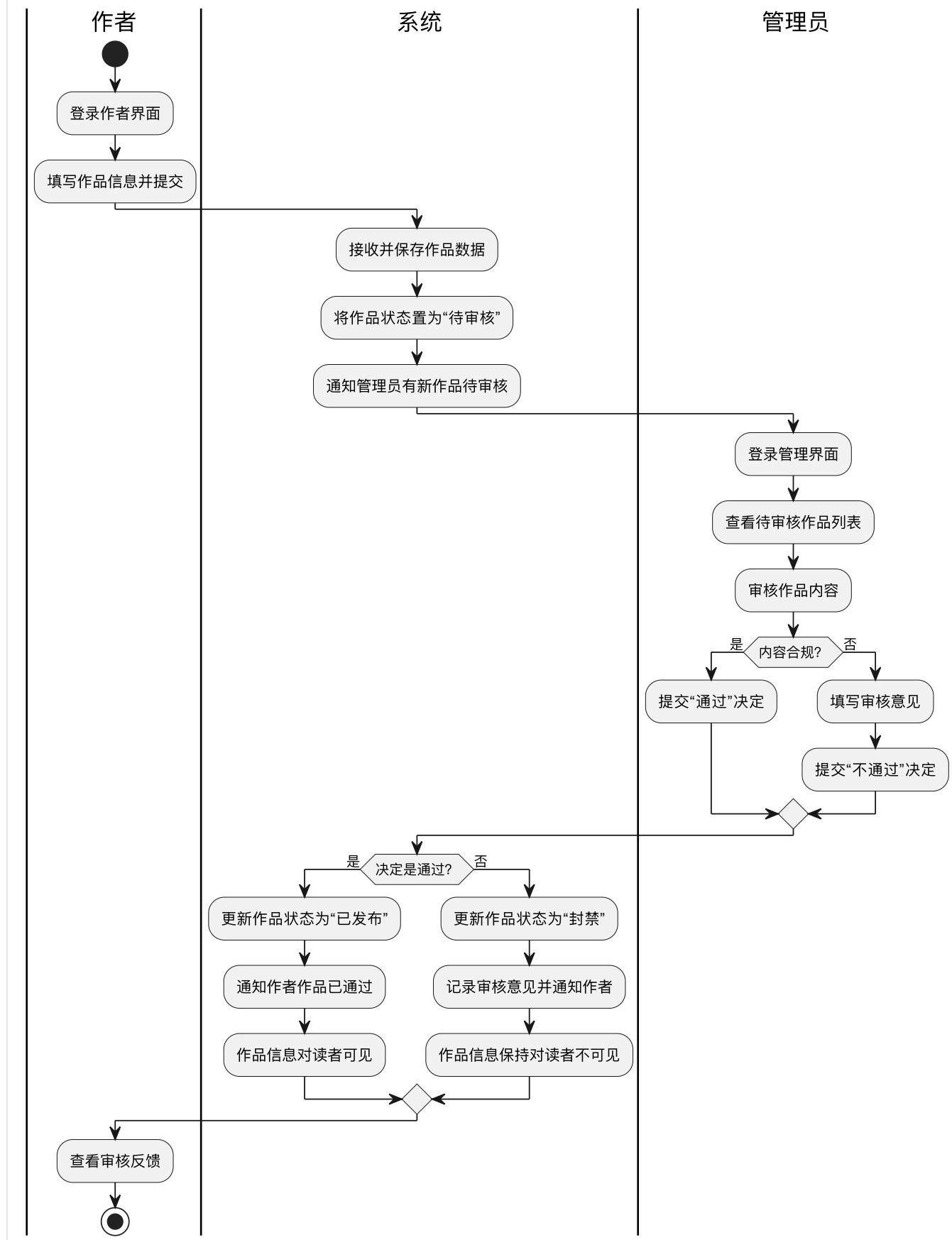


2. 小说章节状态转换图

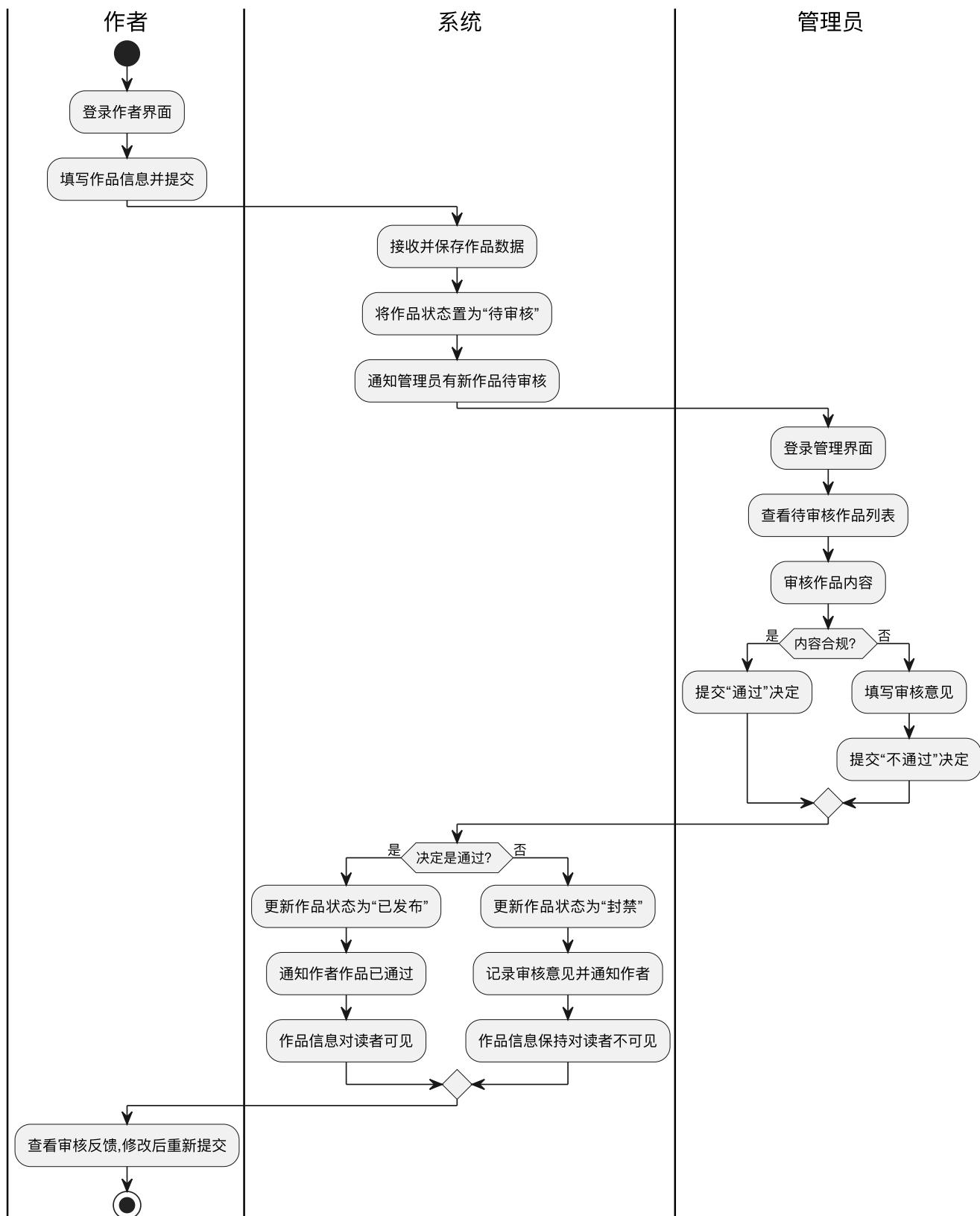


3. 创建作品用例活动图

作品创建与审核流程

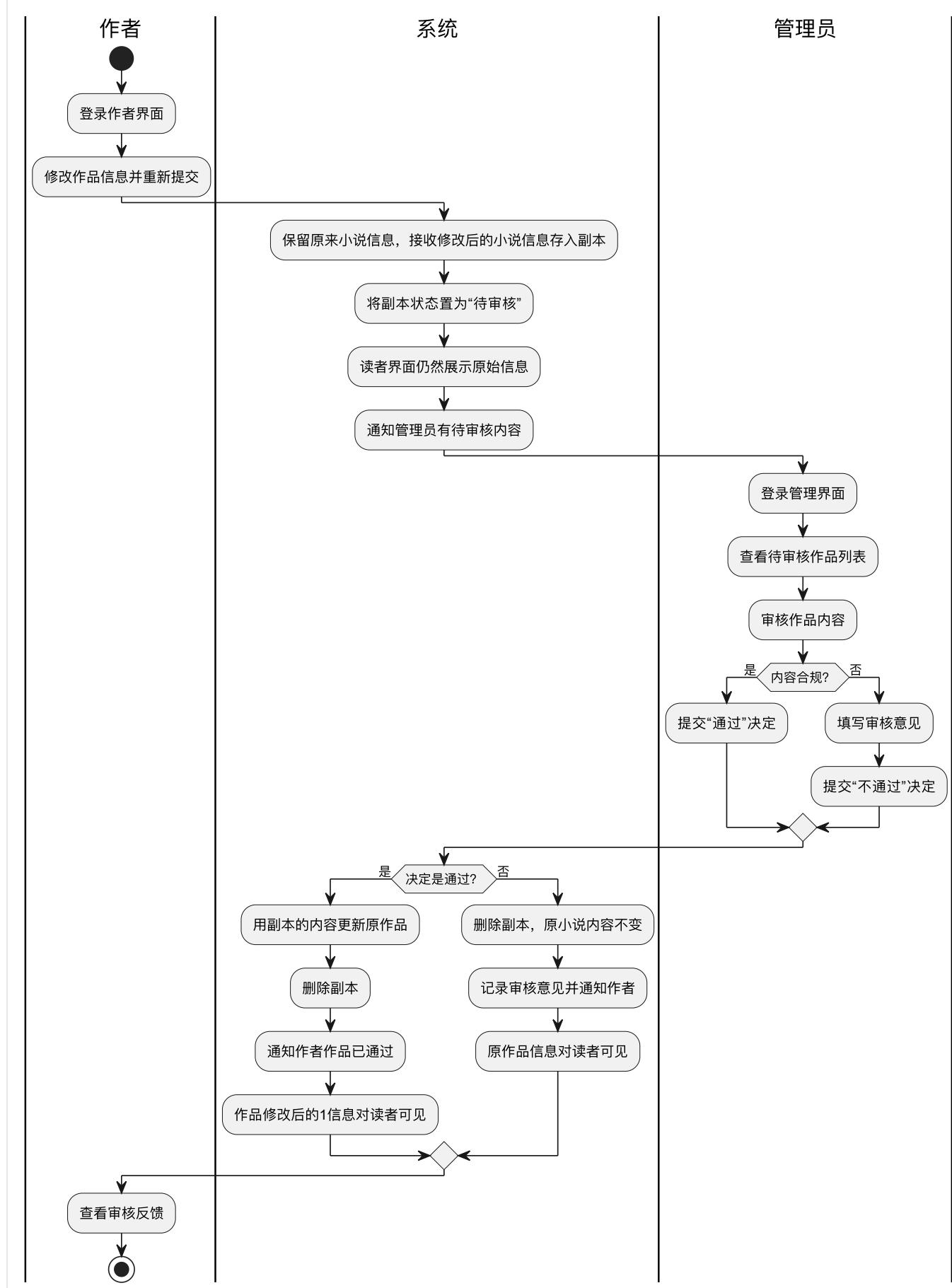


作品创建与审核流程

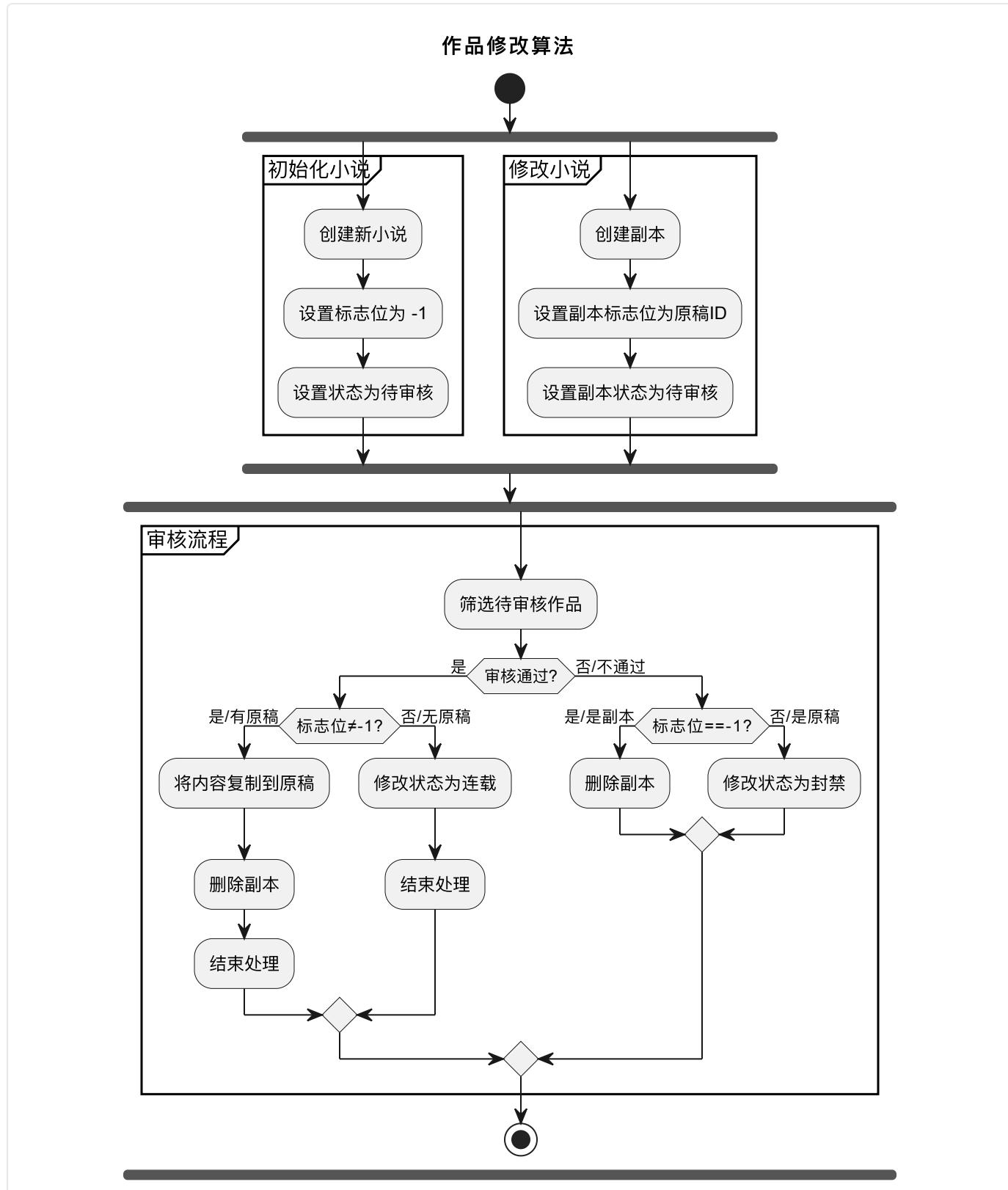


4.修改作品用例

作品修改与审核流程

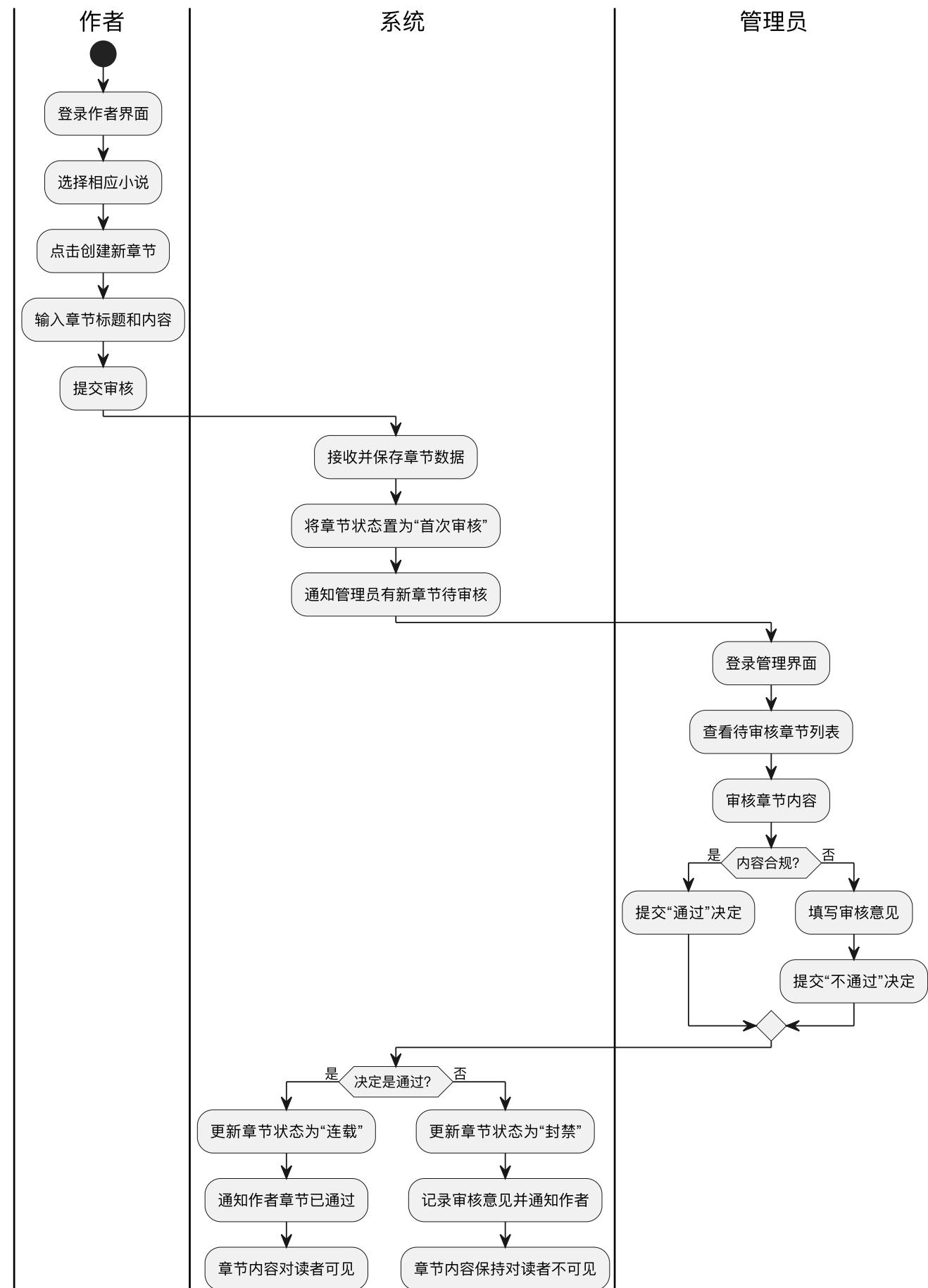


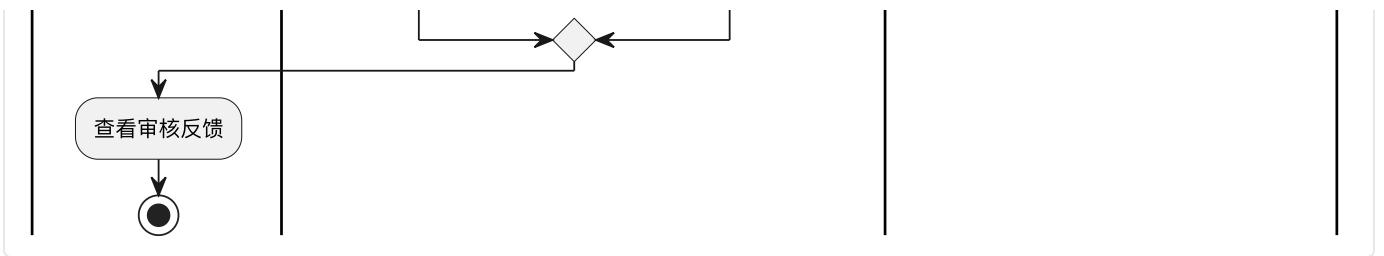
作品更新算法流程图



5.作者创建章节用例流程图

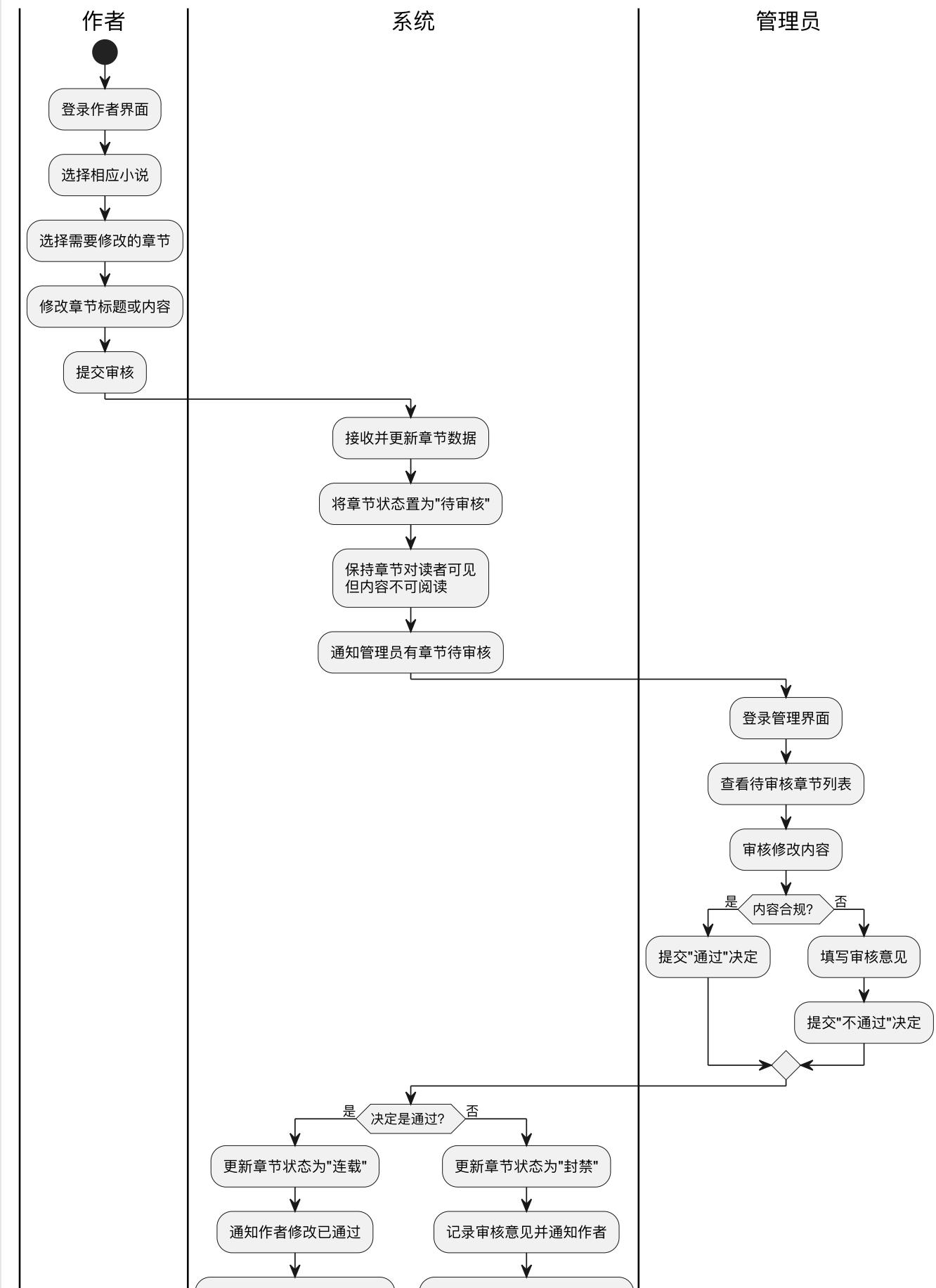
章节创建与审核流程

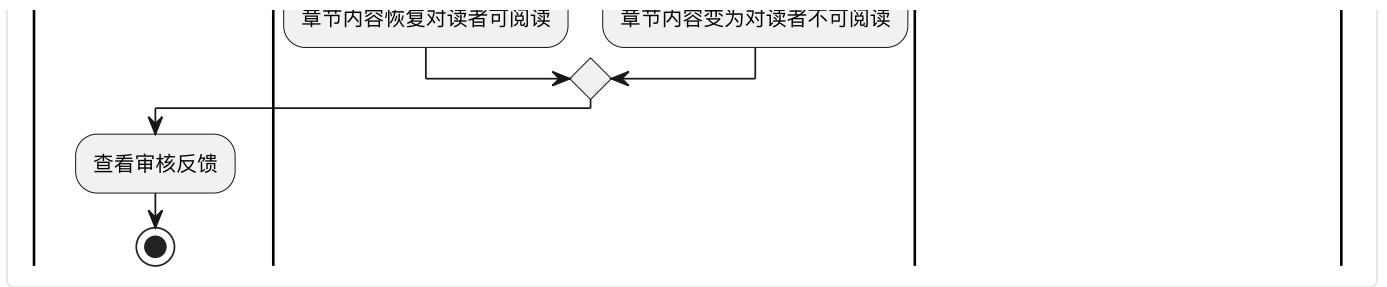




6.作者修改章节用例流程图

章节修改与审核流程





7. 管理员管理小说分类用例

8. 管理员审核小说用例

见上述小说管理图

9. 管理员审核章节用例

见上述章节管理图

10. 读者浏览分类用例

12. 读者查询小说用例

13. 读者查看排行榜用例

14. 读者阅读小说用例

15. 读者收藏小说用例

16. 读者推荐小说用例

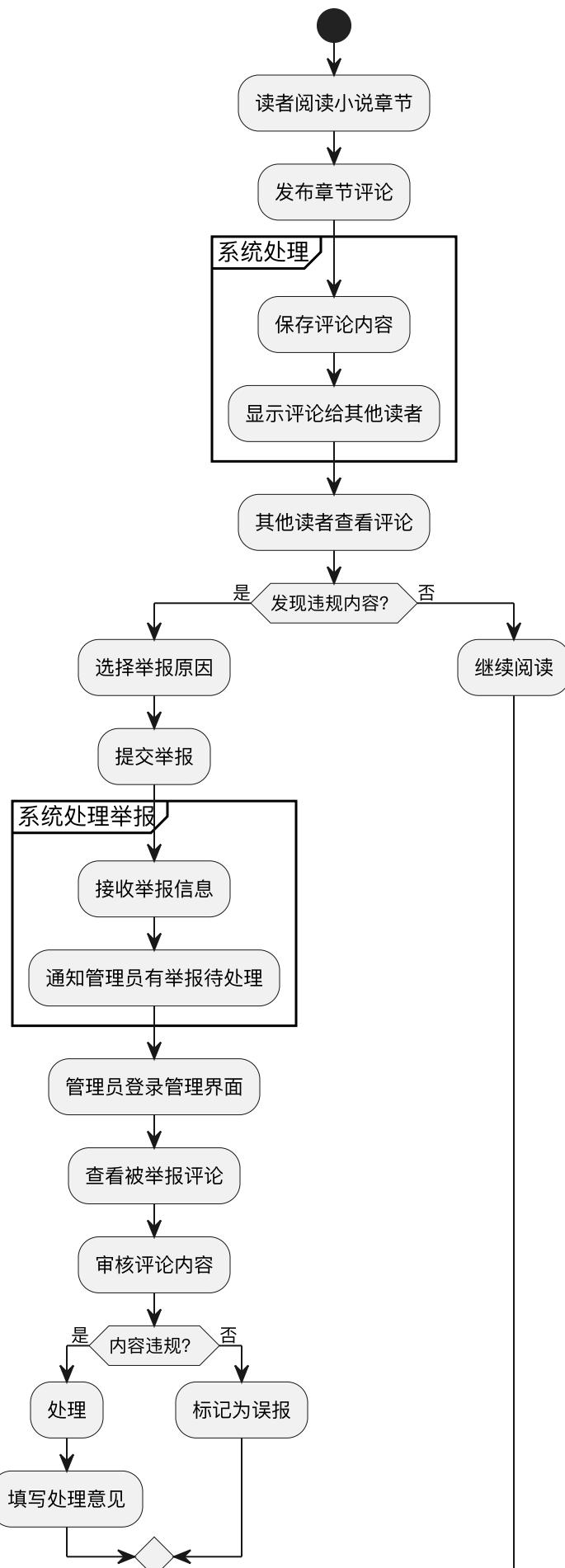
17. 读者评分小说用例

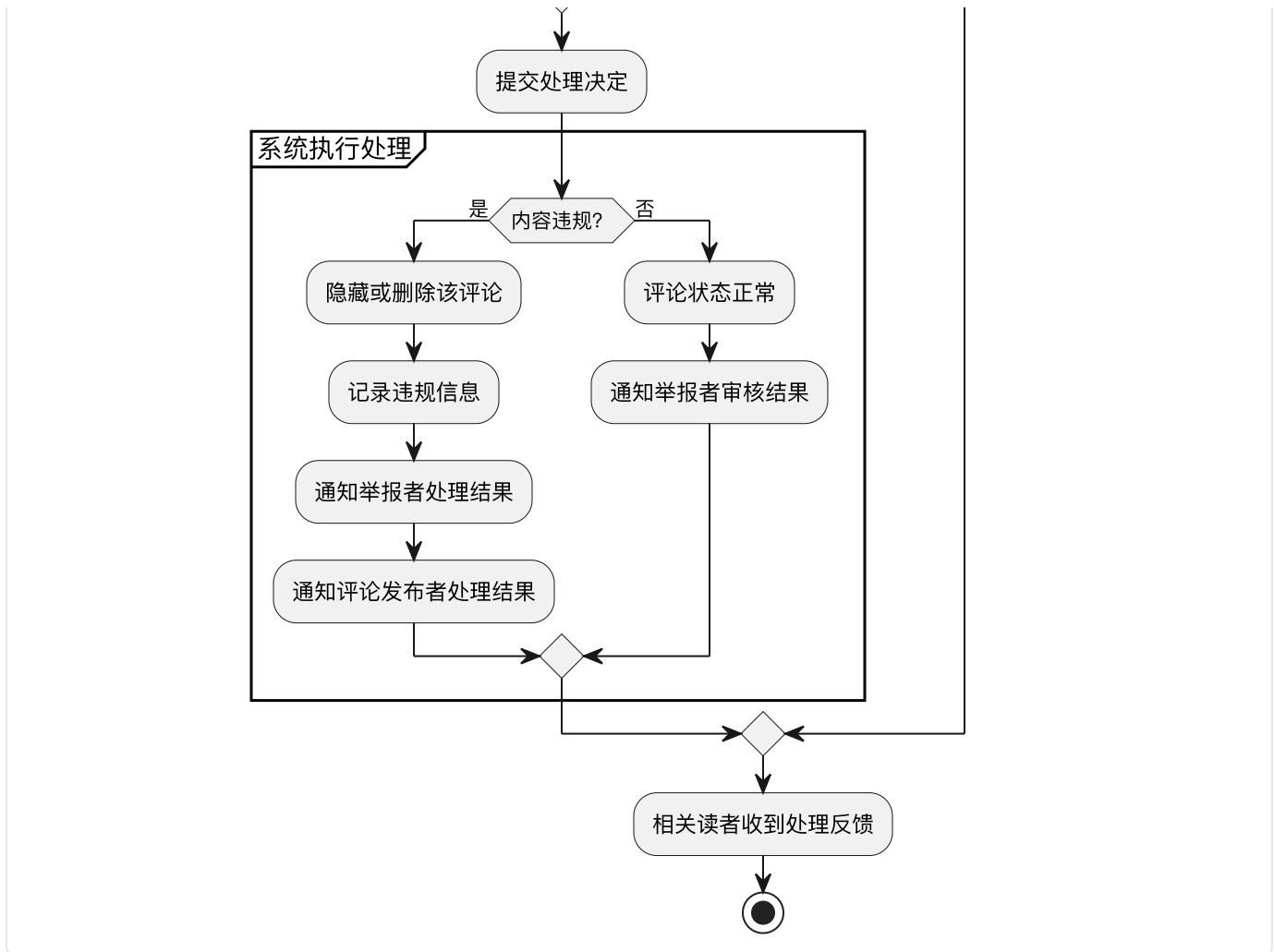
五.用户信息展示子系统

1.作者个人数据展示用例

2.读者个人数据展示用例

六.评论子系统





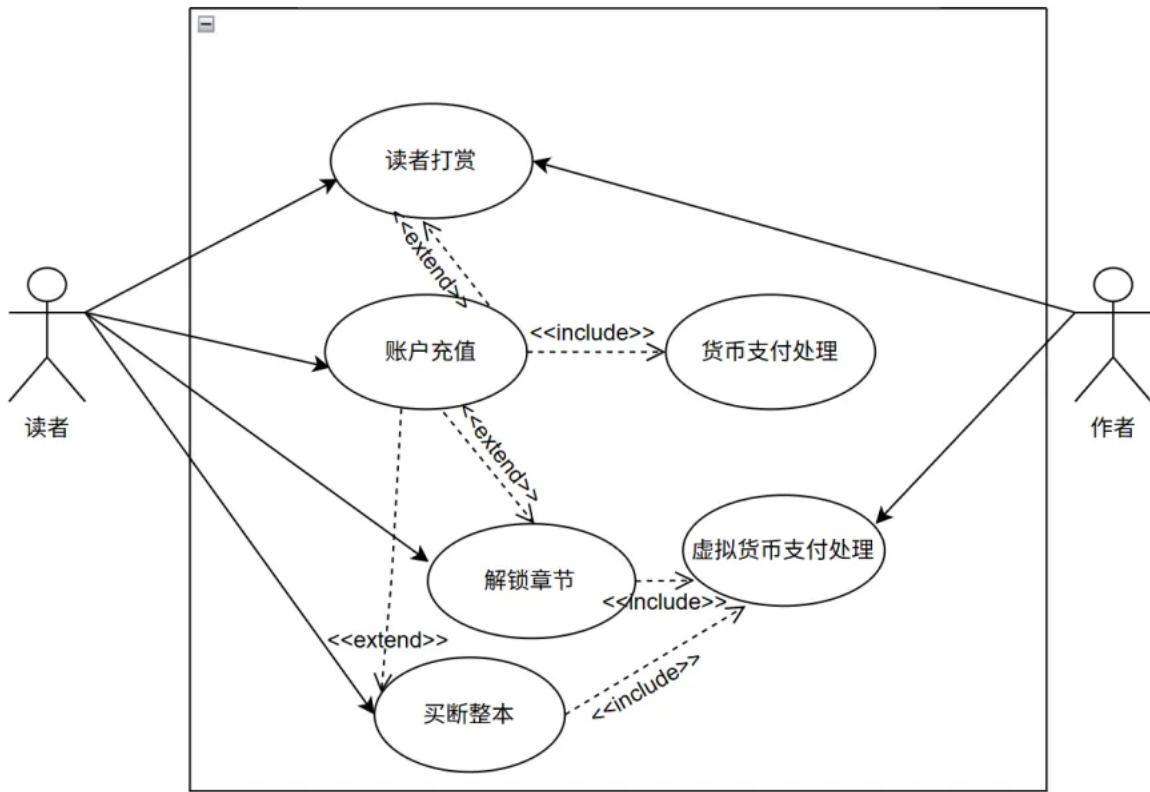
1.读者发布章节评论用例

2.读者举报章节评论用例

3.管理员审核被举报章节评论用例

七.交易子系统

用例图



1.读者充值用例

2.读者购买章节用例

3.读者打赏小说用例

4.作者查看小说收益用例

Git管理规范

1.Git 分支模型

本项目采用的三类分支结构：

1. main 分支 (原 master 分支)

- 生产环境代码，始终保持稳定可发布状态
- 只能通过 dev 分支合并进来
- 每次合并应打上版本标签 (tag)

2. dev 分支 (开发主分支)

- 集成各功能开发成果的稳定开发线
- 通过合并 feature 分支获得新功能
- 经过测试稳定后才能合并到 main

3. feature 分支 (功能开发分支)

- 从 dev 分支创建，命名规范：`feature/功能名称-开发者`，如 `feature/user-auth-zhangsan`
- 各组在各自 feature 分支上自由开发
- 开发完成后通过 Pull Request/Merge Request 申请合并到 dev

2.工作流程示例

```

1
2 1. 从 dev 创建 feature 分支
3 git checkout dev
4 git pull origin dev
5 git checkout -b feature/new-module
6
7 2. 在 feature 分支开发提交
8 git add .
9 git commit -m "feat: 添加用户登录功能"
10 git push origin feature/new-module
11
12 3. 开发完成后发起 Merge Request 到 dev
13
14 4. 代码评审后合并到 dev
15
16 5. 当 dev 稳定后, 合并到 main 并打标签
17 git checkout main
18 git merge dev
19 git tag -a v1.0.0 -m "版本1.0.0发布"
20 git push origin main --tags

```

3. Commit 信息规范

提交信息格式: <类型>: <描述>

| 类型 | 说明 |
|----------|-----------------------------|
| feat | 新增功能或特性 |
| fix | 修复bug |
| docs | 文档更新 (README、CHANGELOG、注释等) |
| style | 代码格式调整 (不影响代码逻辑的空格、分号、格式化等) |
| refactor | 重构代码 (既不是修复bug也不是添加新功能) |
| perf | 性能优化 |
| test | 测试相关 (添加测试用例、修改测试代码) |

| | |
|--------|--|
| build | 构建系统或外部依赖变更 (webpack、gulp、npm等) |
| revert | 回退某个提交 |
| chore | 其他不修改源代码或测试文件的更改 (如.gitignore、.editorconfig等) |

优秀提交信息示例

▼ Markdown

```
1
2 - feat: 添加用户注册功能
3   fix: 修复登录时密码验证失败的问题
4   docs: 更新API接口文档
5   style: 调整首页CSS样式
6   refactor: 重构订单处理逻辑
7   perf: 优化商品列表加载速度
8   test: 添加用户服务单元测试
9   build: 更新webpack至5.0版本
10  revert: 撤销上次的提交
11  chore: 更新.gitignore文件
```

基本准备

1. 前端

①

采用VUE实现，使用VITE构建

负责前端的同学用vite构建项目，构建时选择vue和JavaScript，在第一周主要自己学习相关语法需要实现的目标见文档**前端学习**。

②怎么实现现代化的ui页面

可以直接使用ui库，例如

<https://www.antdv.com/components/overview>

和

<https://element-plus.org/>

<https://vue3-toastify.js-bridge.com/>

③非前端同学如果想要在自己电脑上运行前端界面，请参考文档**前端vue环境搭建**

2. 后端

3. 数据库

①数据库的连接

②可视化工具

推荐使用Dbeaver

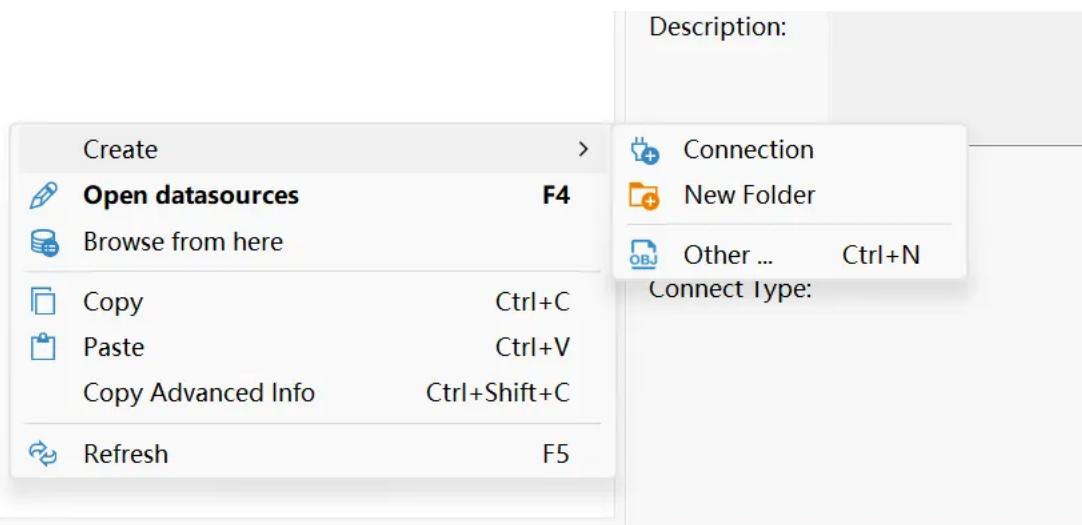
<https://dbeaver.io/download/>

Windows

- [Windows \(installer\)](#)
- [Windows \(zip\)](#)

下载zip文件解压缩后即可运行

进入界面即可连接数据库



右键单击，点击连接

点击oracle，然后next

Connect to a database

Select your database

Create new database connection. Find your database driver in the list below.

Type part of database/driver name to filter Sort by: Title Score

All

Popular

SQL

NoSQL

Analytical

Files

Embedded

Timeseries

Hadoop / BigData

Full-text search

 ORACLE
Oracle

 SQLite

 IBM DB2
Db2 for LUW

 MariaDB
MariaDB

 MySQL

 PostgreSQL

 Microsoft SQL Server
SQL Server

 Altibase

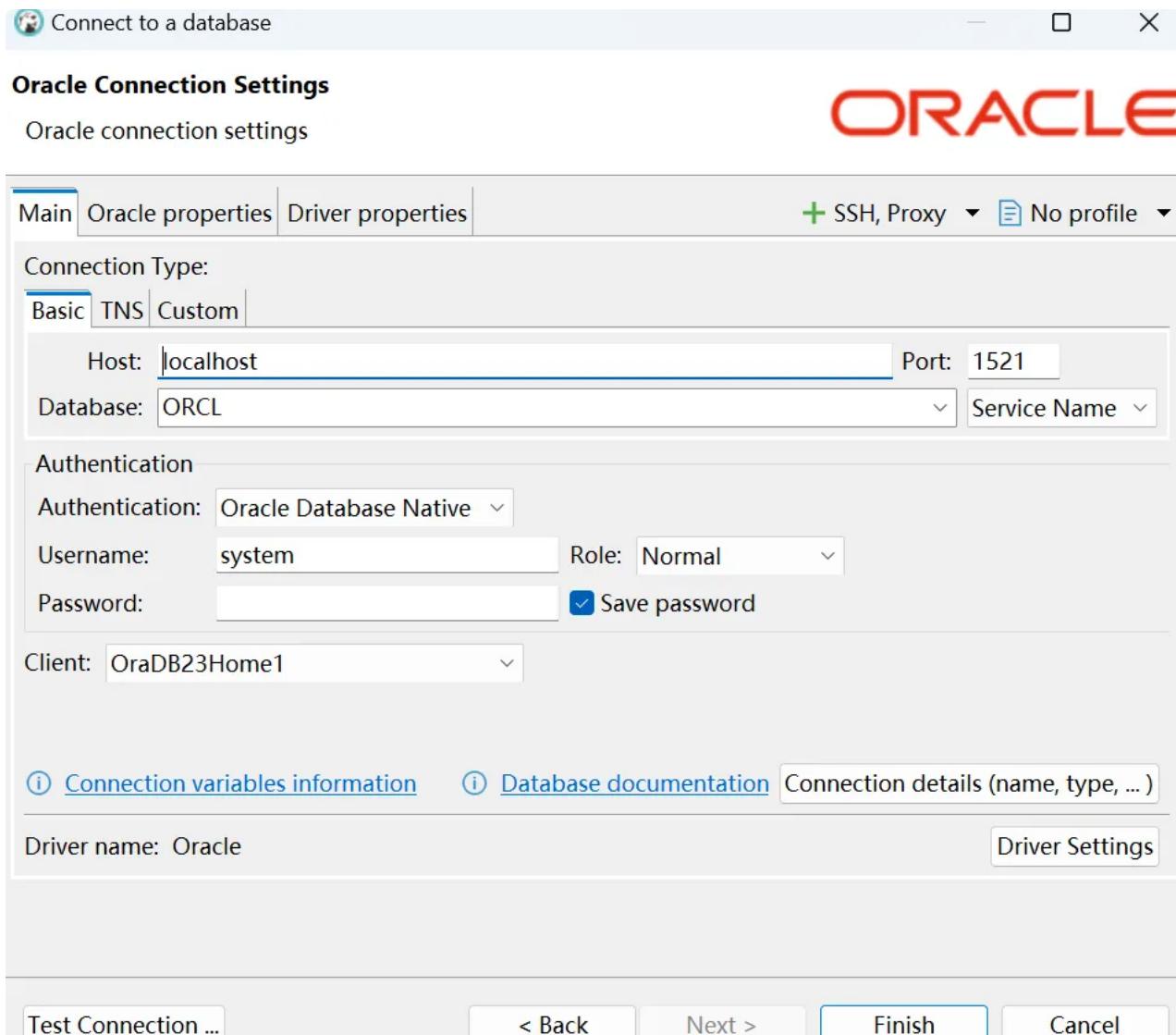
 APACHE DRILL

 Apache Drill

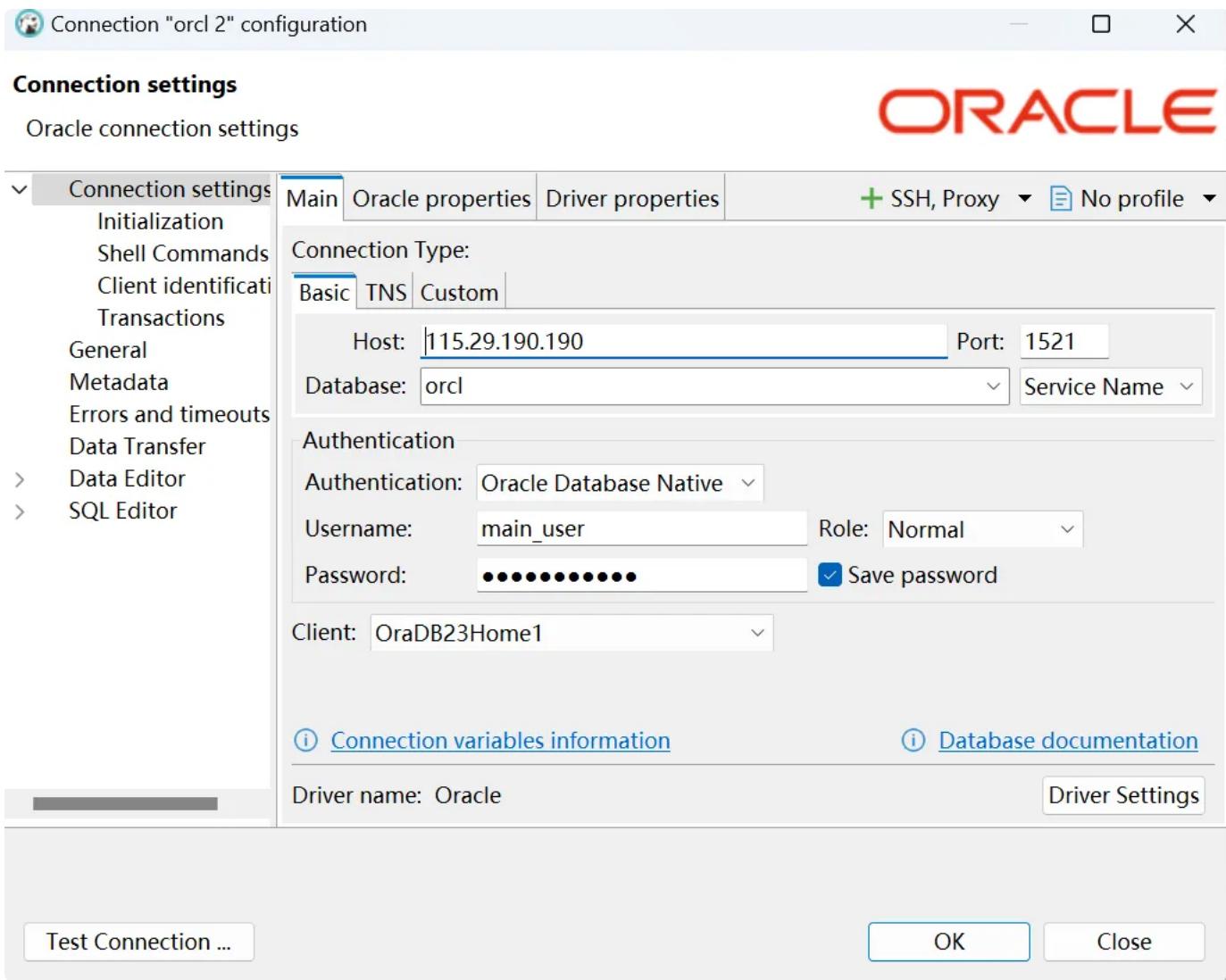
 Apache Drill

Test Connection ... **< Back** **Next >** **Finish** **Cancel**

需要填写如下界面



如图填写，密码是my_password,首次连接注意点击驱动进行下载



点击模式目录下的MAIN_USER， 该目录下的TABLE下存放有我们所有的数据表

- > Schemas
 - > ANONYMOUS
 - > APPQOSSYS
 - > AUDSYS
 - > DBSFWUSER
 - > DIP
 - > DVF
 - > DVSYS
 - > GGSYS
 - > GSMADMIN_INTERNAL
 - > GSMCATUSER
 - > GSMROOTUSER
 - > GSMUSER
 - > LBACSYS
 - > MAIN_USER
 - > Tables
 - > AUTHOR
 - > Columns

其它功能自行探索

4.自动化测试

后续考虑

5.API文档生成

开发调试使用 **swagger**, 写好 API 文档

后端编译运行指定使用 **VS studio**

6.GIT提交

见git管理规范文档

请所有成员严格按照git管理规范文档提交

功能点

1. 登录 登出 注册
 2. 章节
 3. 书籍
 4. 分类
 5. 排行榜
 6. 搜索
 7. 收藏 评分
 8. 举报
 9. 评论 回复 举报 点赞
 10. 管理
 11. 交易 - 充值
 12. 交易 - 解锁 章节
 13. 交易 - 打赏
 14. 个人信息
- 等

待修改部分记录

表示未修改

表示已修改

大家把已存在的但是需要修改的地方汇总在这里，会解决的同学及时解决，解决后打勾

如果是十分紧急的部分直接在群里面说。

待修改：

1.READER数据表的balance字段目前是整型需要修改为两位小数。

`BALANCE NUMBER(18,2) DEFAULT 0`

2.读者id没有设置为自增，导致api出错

1.AUTHOR数据表的earn字段目前是整型需要修改为两位小数。

`NUMBER(18,2) DEFAULT 0`

7.18 后端：

chapter表

chapter状态：草稿，审核中，已发布，封禁（默认草稿）

novel表：

新增OriginalNovelId属性

novel状态：待审核，连载，完结，封禁

management部分：

report修改->comment的status修改

添加result填写机制

report_management表

增加通过举报者的id获取所有举报的result的api

数据库修改记录

注:所有对数据库的修改, 包括新增表, 新增属性, 新增列, 新增触发器等等, 均需要在本文档记录, 另外所有修改使用的sql文件均需存档!!!

文件目录结构

```
Plain Text | ▾

1  database/
2  └── 01_schema/          # 数据库结构定义
3      ├── tables/          # 表结构
4      │   ├── reader.sql    # 读者表
5      │   └── post.sql       # 帖子表
6      ├── views/           # 视图定义
7      └── sequences/        # 序列定义
8          └── reader_seq.sql
9      ├── triggers/         # 触发器
10     └── reader_id_trg.sql
11     └── indexes/          # 索引
12         └── idx_reader_name.sql
13
14 └── 02_data/             # 数据操作
15     ├── seed/             # 初始数据
16     │   └── init_reader_data.sql
17     └── migrations/        # 变更脚本 (按版本)
18         ├── v1.0_create_tables.sql
19         └── v2.0_add_reader_balance.sql
20
21 └── 03_security/         # 权限相关
22     ├── roles.sql          # 角色定义
23     └── grants.sql         # 权限分配
24
25 └── 04_procedures/        # 存储过程/函数
26     ├── sp_get_reader_stats.sql
27     └── fn_calculate_balance.sql
```

1.DLL文件

目前已有的用于创建数据表的sql文件

2.对Reader的修改

对reader新增了序列和触发器

3.对Author的修改

AUTHOR数据表的earn字段目前是整型需要修改为两位小数。

NUMBER(18,2) DEFAULT 0

4.新修改

- ①给所有的外码引用加上了删除级联
- ②删除了原来读者的自增机制，对所有主键设置了identity

5.manger和mangement

将原来的错误的表进行了修改

6.增加作者收入表

```
1 CREATE TABLE author_income (
2     id NUMBER(19) GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
3     author_id NUMBER(19) NOT NULL,
4     type VARCHAR2(20) NOT NULL,
5     amount NUMBER(10,2) NOT NULL,
6     create_time TIMESTAMP DEFAULT SYSTIMESTAMP NOT NULL
7 );
```

初期前端分工

7月14日至7月15日

zyh

作者界面实现

zhx

用户登录与注册界面

api调用

已上传github

zsk, ygc

管理员界面实现

初期后端分工

7月11日至7月20日

前端:

zhx ygc zsk zyh

后端:

gyx gy lyh zcr dyj lxf

①周末首先搭建后端和前端框架，进行代码规范，组织文件结构

小学期那一周详细分配每个同学的任务

②在近两周内后端主要实现各个api，前端主要基于后端api实现管理员界面，可并行推进。

后续由组长指定阶段任务，小组长分配任务，格式如下，完成自行打勾

7月14日至7月15日

任务:A组: 后端--管理员 B组:前端--管理员

A组 lyh zcr

- 小说增删改
- 章节增删改

B组 lxf dyj

- 评论增删改
- 分类增删改

C组 gy gyx

- 用户增删改
- 作者增删改

✓ 图片api的实现

已经上传github。

之后：

| | | | |
|---|--------|--|-------------------------------------|
| A  | 举报评论 | POST /api/comment/{comment_id}/report | report |
| A  | 查看举报列表 | GET /api/reports | report |
| A  | 处理举报 | POST /api/report/{report_id}/process | report_management, management |
| A  | 用户注册 | POST /api/reader/register | reader |
| A  | 用户登录 | POST /api/reader/login | reader |
| A  | 用户退出 | POST /api/reader/logout | 无 |
| A | 修改隐私设置 | PUT /api/reader/{id}/privacy | reader |
| A  | 作者注册 | POST /api/author/register | author |
| A  | 作者登录 | POST /api/author/login | author |
| A  | 作者退出 | POST /api/author/logout | 无 |
| A  | 管理员注册 | POST /api/manager/register | manager |
| A  | 管理员登录 | POST /api/manager/login | manager |
| A  | 小说审核 | POST /api/novel/{id}/review | novel, novel_management, management |

| | | | |
|---|---------------|---|---|
| A  | 章节审核 | POST /api/novel/{id}/chapter/{ chapter_id}/review | chapter, chapter_management, management |
| A  | 评论审核 | | comments, comments_management, management |
| A  | 管理-创建 | | |
| A  | 管理-修改 | | |
| A  | 管理-删除 | | |
| A  | 管理-查看所有 实体 | | |
| B  | 分类管理 - 创建 | POST /api/category | category |
| B  | 分类管理 - 修改 | PUT /api/category/{id} POST/api/Category/rena me (可选) | category |
| B  | 分类管理 - 删除 | DELETE /api/category/{id} | category |
| B  | 分类管理 - 列表 | GET /api/category | category |
| B  | 小说绑定分类 | POST /api/novel/{id}/category/ {category_name} | novel_category |
| B  | 小说移除分类 | DELETE /api/novel/{id}/category/ {category_name} | novel_category |
| B  | 获得全部分类及 小说 | GET/api/NovelCategory | novel_category |

| | | | |
|---|----------------|---|---------------------------------|
| B  | 小说排行(收藏、推荐、评分) | GET /api/rankings, GET /api/search | novel, rate, recommend, collect |
| B | 小说搜索 | GET /api/rankings, GET /api/search | novel |
| B  | 评论发布 | POST /api/novel/{id}/chapter/{chapter_id}/comment | comments |
| B  | 评论列表获取 | GET /api/novel/{id}/chapter/{chapter_id}/comments | comments |
| B  | 删除评论 | DELETE /api/comment/{comment_id} | comments, comment_management |
| B  | 点赞评论 | POST /api/comment/{comment_id}/like | likes |
| B  | 取消点赞 | DELETE /api/comment/{comment_id}/like | likes |
| B  | 评论回复 | POST /api/comment/{comment_id}/reply | comment_reply |
| B  | 获取回复 | GET /api/comment/{comment_id}/replies | comment_reply |
| B  | 收藏小说 | POST /api/collect | collect |
| B  | 取消收藏 | DELETE /api/collect | collect |
| B  | 获取所有收藏记录 | GET /api/collect | collect |

| | | | |
|---|-----------|--|--|
| B  | 推荐小说 | POST /api/recommend | recommend |
| B  | 取消推荐 | DELETE /api/recommend | recommend |
| B  | 获取收藏 | GET /api/recommend | recommend |
| B  | 添加或更新小说评分 | POST /api/rate | rate |
| B  | 删除小说评分 | DELETE /api/rate | rate |
| B  | 获取全部评分 | GET /api/rate | rate |
| C组 | 用户资料获取 | GET /api/reader/{id} | reader |
| C组 | 用户资料修改 | PUT /api/reader/{id} | reader |
| C组 | 作者查看收益 | GET /api/author/{id}/earnings | author, reward, purchase |
| C组 | 管理员列表查看 | GET /api/managers | manager |
| C组 | 管理员信息修改 | PUT /api/manager/{id} | manager |
| C组 | 删除作者 | DELETE /api/author/{id} | author, novel, chapter, comments, collect, recommend, rate, purchase, reward, novel_category, novel_management, chapter_management, comment_management |
| C组 | 删除用户 | DELETE /api/reader/{id} | reader, comments, collect, recommend, rate, purchase, reward, likes, report |
| C组 | 用户充值 | POST /api/reader/{id}/recharge | transaction |
| C组 | 购买章节 | POST /api/novel/{id}/chapter/{chapter_id}/purchase | transaction, purchase |

| | | | |
|----|--------|--------------------------------------|---------------------|
| C组 | 打赏小说 | POST /api/novel/{id}/reward | transaction, reward |
| C组 | 查看交易记录 | GET /api/reader/{id}/transactions | transaction |

读者分工

功能

1. 登录登出注册

2. 章节

一个是现在的目录能不能调整为下面这样 (zcr)

| | | |
|-----------------|-----------------|-----------------|
| 第一章 百世书 | 第二章 顺天易，逆天难 | 第三章 魔门作风 |
| 第四章 防不胜防 | 第五章 这个仇我记下了 | 第六章 魔门的修炼方式 |
| 第七章 神霄御剑真决 | 第八章 牛马修仙传 | 第九章 财帛动人心 |
| 第十章 月黑杀人夜 | 第十一章 杀生咒 | 第十二章 功德博彩，替死阴傀 |
| 第十三章 螳螂捕蝉，黄雀在后 | 第十四章 黑吃黑！ | 第十五章 钓鱼 |
| 第十六章 疯狂的泡沫 | 第十七章 别人恐惧我加仓 | 第十八章 外挂的正确用法 |
| 第十九章 剥皮 | 第二十章 萧石叶 | 第二十一章 无妄之灾 |
| 第二十二章 神光之威 | 第二十三章 先天一炁万灵幡 | 第二十四章 自爆！ |
| 第二十五章 鸿运惊魂 | 第二十六章 不愧是圣宗 | 第二十七章 悟阵 |
| 第二十八章 九变化龙决 | 第二十九章 因果招是非 | 第三十章 补天峰主 |
| 第三十一章 一剑斩长空 | 第三十二章 目标，刘信！ | 第三十三章 先天混元一炁神符 |
| 第三十四章 三河招揽，筑基机缘 | 第三十五章 赵师兄和我有缘啊！ | 第三十六章 骷髅山坊市 |
| 第三十七章 太阴蜕形尸解真法 | 第三十八章 神武门 | 第三十九章 故袭 |
| 第四十章 阵斗真传 | 第四十一章 麽战 | 第四十二章 玄阴摄形，钉头索命 |
| 第四十三章 血衣楼主 | 第四十四章 走投无路的赵旭河 | 第四十五章 福星 |
| 第四十六章 盗天机 | 第四十七章 潜龙不出渊 | 第四十八章 筑基定因果 |

一个打赏按钮 (gy)

3.书籍✓

4.分类✓

5.排行榜✓

6.搜索✓

7.收藏评分(gy)✓

读者对小说评分是不是还没实现，目前只有展示小说的评分。应该简单，加个按钮的事情

Rate 实现读者对小说评分

POST /api/Rate 添加评分 (不可更新, 已存在则报错)



图片加载失败

8.举报(zcr)

举报在评论功能完全完善之后

9.评论回复 举报 点赞 (zcr)✓

10.管理✗

11.交易-充值✓

12.交易-解锁章节(zhx)

13.交易-打赏✓

14.个人详情页

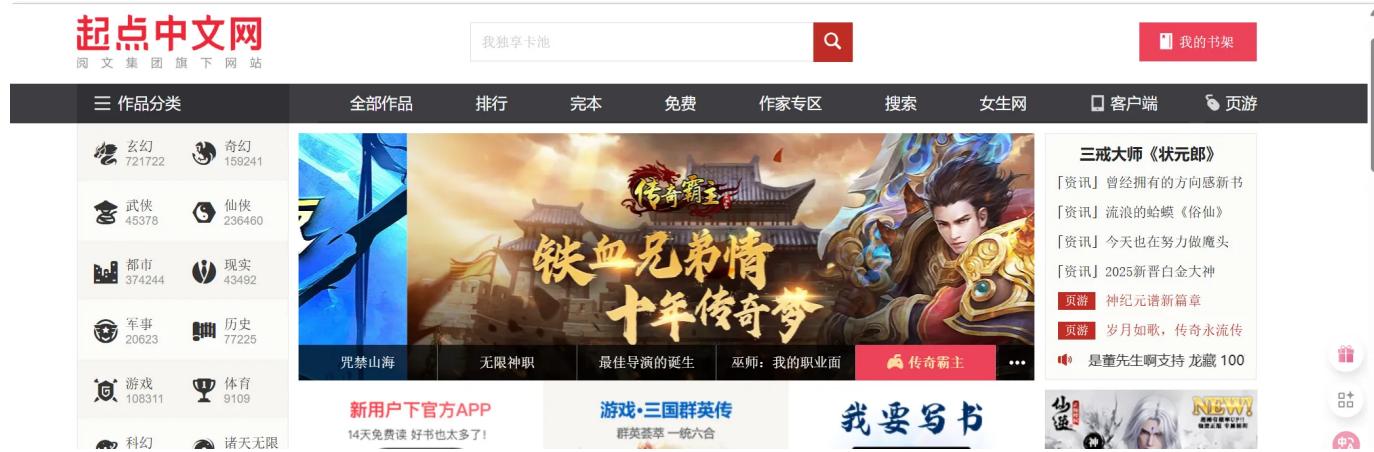
框架gy 交易记录模块:gy

其余部分:gyx

还需要修改加补充

目前的个人主页还需要调整，参考起点小说网

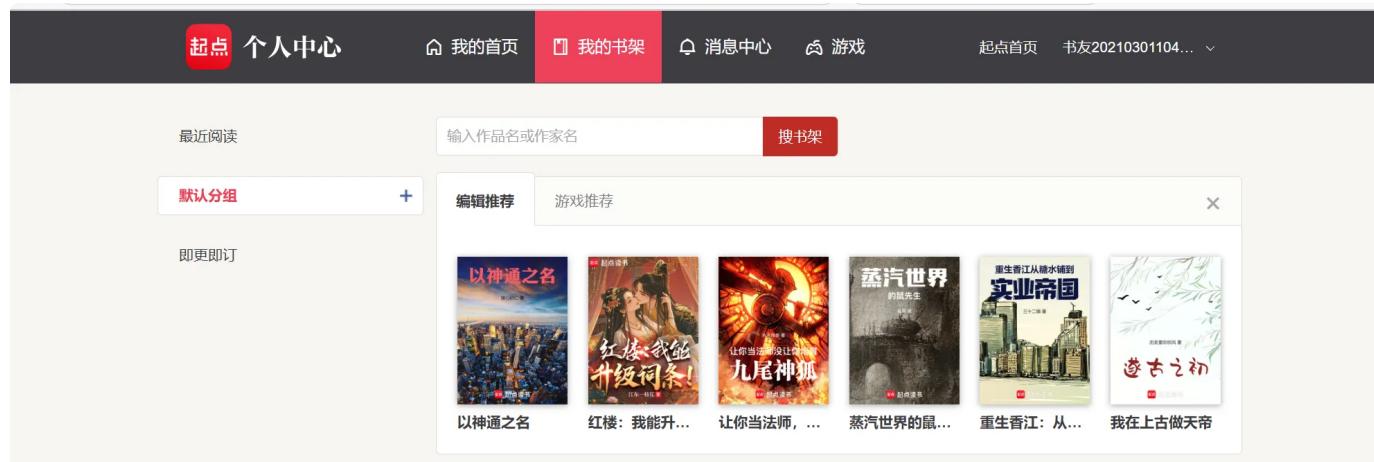
①点击我的书架按钮



The screenshot shows the homepage of Qidian Chinese Network. At the top right, there is a red button labeled '我的书架' (My Shelf). The page features a large banner for '传奇霸主' (Legend Master) with the text '铁血兄弟情 十年传奇梦'. Below the banner are several navigation links: '全部作品' (All Works), '排行' (Ranking), '完本' (Completed), '免费' (Free), '作家专区' (Author Zone), '搜索' (Search), '女生网' (Women's Network), '客户端' (Client), and '页游' (Page Games). On the left, there is a sidebar with '作品分类' (Work Categories) including玄幻、奇幻、武侠、仙侠、都市、现实、军事、历史、游戏、体育、科幻、和诸天无限。At the bottom, there are promotional banners for '新用户下官方APP' (New users download the official app) and '游戏·三国群英传' (Game·Three Kingdoms War).

②进入个人页，需要另开一个窗口

跳转至新窗口，导航栏为我的书架，这部分gyx已经实现，但是样式还要调整



The screenshot shows the user's personal center. The navigation bar at the top has tabs: '个人中心' (Personal Center), '我的首页' (My Home Page), '我的书架' (My Shelf), '消息中心' (Message Center), '游戏' (Games), and '起点首页' (Qidian Home Page). The '我的书架' tab is highlighted with a red background. Below the navigation bar, there is a search bar with the placeholder '输入作品名或作家名' (Enter work name or author name) and a '搜书架' (Search Shelf) button. A '默认分组' (Default Group) button is also visible. A '编辑推荐' (Editor's Recommendation) section displays several book covers: '以神通之名', '红楼：我能升级词条！', '让你当法师，', '蒸汽世界', '重生香江：从', and '造古之初'.

③消息中心，gyx已经实现一部分，在那基础上继续



The screenshot shows the user's personal center with the '消息中心' (Message Center) tab selected. The navigation bar is identical to the previous screenshot. Below the navigation bar, there is a '系统通知' (System Notification) section with a red '系统通知' button. The section displays a table with columns: '来自' (From), '标题' (Title), and '日期' (Date). The table is currently empty.

④我的首页，这部分功能很多，建议布局由一个人设计，功能分散到其它组员

[首页](#)[账务中心](#)[我的票夹](#)[我的红包](#)[我的书评](#)[我的本章说](#)[用户等级](#)[徽章/称号/认证](#)[VIP中心](#)[安全中心](#)[作家专区](#)[风色传说, 勇...](#)[乾坤天地, 英...](#) 书友2021030110411810616 Lv1安全级别  30/100

关注 3 粉丝 0 大神之光 0

[个人主页](#)[账户余额](#)[0 起点币](#)[充值](#)[我的票夹](#)[0 月票 · 3推荐票](#)[立即查看](#)[我的书架](#)[5 本藏书](#)[立即查看](#)[我的书单](#)[0 个关注](#)[立即查看](#)

⑤首页是一个大功能，下面继续说首页

交易记录模块，一个人(gy)

我的资产

账户余额 (起点币)

0

充值

含0赠币 [?](#)

交易记录

交易记录可能存在延时，仅供参考，请以实际金额变化为准

消费记录

充值记录

台币充值订单

获赠记录

当前筛选项累计消费 **0** 起点币 (2017年4月以前打赏筛选项不包括女生网打赏记录)

所有

2025年07月

消费金额

消费时间

类别

备注

目前暂无消费记录

书评模块，一个人

我的书评

| 发表的书评 | 回复的书评 | 收藏的书评 | 收藏的书评区 |
|-------|-------|-------|--------|
| 书评 | 是否精华 | 回复 | 回复时间 |
| | | | 所在书评区 |


扫描前往起点APP查看

还需要支持个人信息修改，下面那个齿轮，这个gyx已经实现，但是还是要调整样式



书友2021030110411810616 Lv1

个人主页 

安全级别  30/100

普通用户

关注 3 粉丝 0 大神之光 0

首页 > 个人信息设置

基本设置 头像设置 隐私设置 收货地址

昵称 书友2021030110411810616 [修改昵称](#) [?](#)

起点ID 215540084

实名认证 [查看实名认证信息](#)

性别 男 女

简介 可以简单的描述自己
0/100

保存

15. 向别人展示的读者信息页

16. 向别人展示的作者页

2025-8-3

读者界面优化相关

个人主页：

1. 提示弹窗统一格式--gy

2. 样式优化--gyx

3. 详情展示跳转api添加--zhx

主页：

1. 初始展示、三端跳转逻辑和按钮位置--zhx

2. 章节解锁相关--zcr

主页美化部分：

分配暂定。

8月分工

①读者界面

(zhx,gyx,gy,zcr)

- 1.完善章节解锁 (zcr) ✓
- 2.首页(首页这里可以做得简单, 也可以做得复杂, 自行发挥) (暂定)
- 3.个人主页美化, 做到风格统一 (gyx, gy)
- 4.处理小说状态问题(后端新的api, 前端调用新的api) (zhx) ✓
- 5.分类界面小说一页做一个分页吧, 后续会增加很多很多小说 (zhx) ✓
- 6.登陆界面, 删除测试界面, 要有一个好看的初始界面 (zhx) ✓
- 7.美化美化美化!!! (一起)
- 8.现在对某个小说目录获取就要加载它的所有章节内容吗? ? 如果是这样, 那需要联系后端进行更改, 只有点击该章节时才获取章节内容。 (之前负责此部分的同学) ✓
- 9.提示弹窗统一格式 (gy)
- 10.个人主页的跳转 (gy,zhx) 完成小说点击跳转 ✓

②作者界面

(zyh)

- 1.解决刷新后数据消失的问题



2. 通过作者界面创建的小说章节没有时间信息？负责作者界面的同学和后端沟通，最好这个默认时间是后端或者数据库生成

第1章 第1章 第一幕 卡塞尔之门 · 一

书名：龙族 作者：官奕 本章字数：2977 更新时间：

最新章节第2章

③管理员界面

尽快完成管理员界面，并进行合并

(ygc,zsk)

④后端与数据库

(lyh, dyj, lxf)

1. 生成所有的ddl文件汇总

2. 需要确保几个人清楚每一处数据库操作的原因，比如为什么要加这个索引，为什么采用这个触发器，是否有更好方案，为什么不采用另一个方案，给出文档说明

3. 继续优化数据库

4. 优化api

5. 分页查询

⑤数据与测试

(怎么分配任务?)

1. 至少50本书，每本20万字，完成完整的数据流动（作者创建，管理员审核，读者阅读）
2. 按业务测试

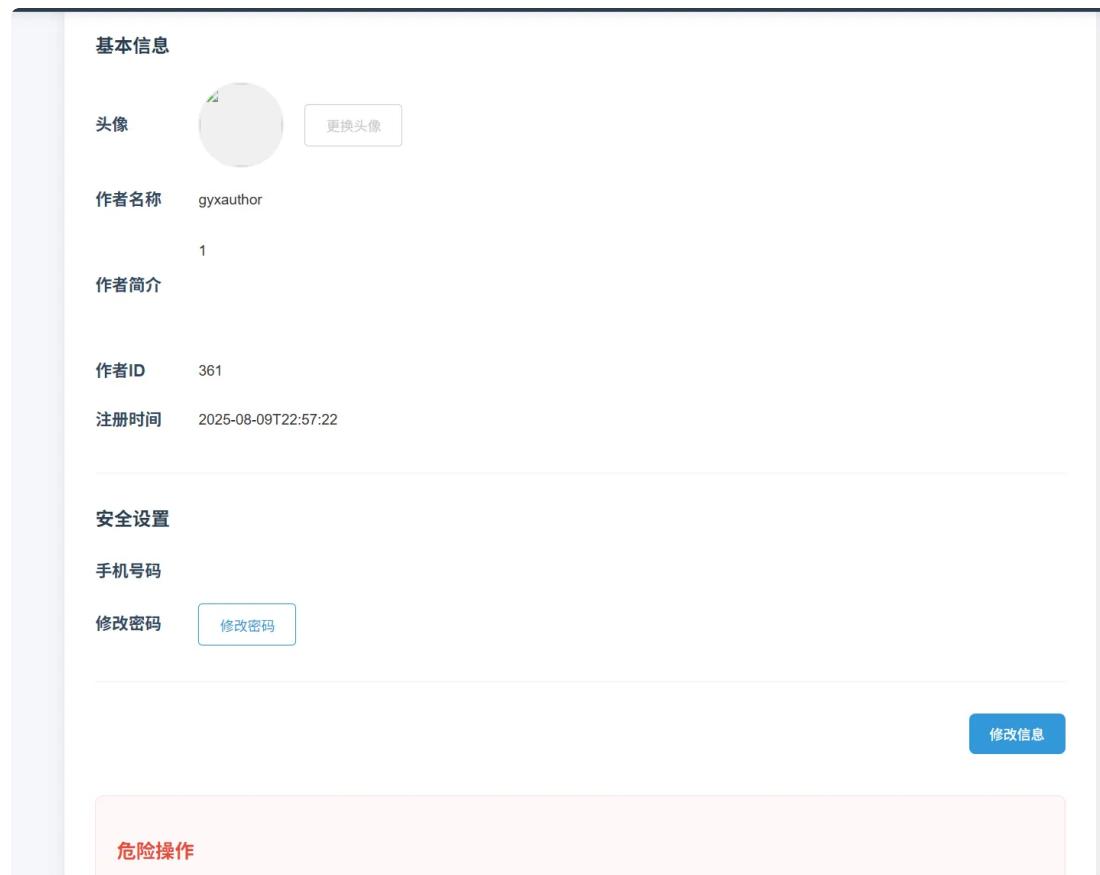
8.10

现有问题：

问题

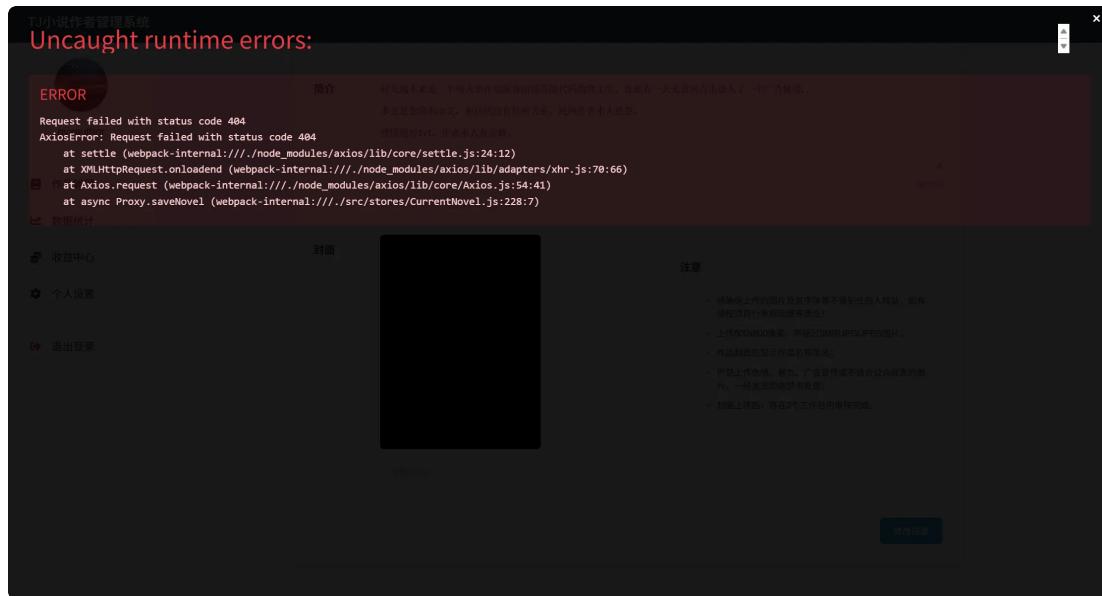
作者页面

1. 修改作者个人信息后头像展示出现问题



(仅出现于刚登陆时修改个人信息，更换头像功能无误)

2. 修改书籍信息弹出来莫名其妙的东西。



同时此处错误也可能导致管理员界面对作者修改简介审核的功能无法测试，管理员界面这一块也展示得不够明显。这个问题可能还得讨论一下。

管理员界面：

1. 用户管理头像问题。

| | | | | |
|--|---------|-------------|---|---------------------|
| | gyx | 13002714567 | 女 | <button>删除</button> |
| | nicole | | | <button>删除</button> |
| | ygc666 | | | <button>删除</button> |
| | 1234567 | | | <button>删除</button> |
| | andy | | | <button>删除</button> |

2. 以下图片是什么意思：

章节内容

加载中或未找到该章节。

返回

同时感觉待审核章节这里属性展示得不够多，比如没有章节号。

3. 审核日志部分全部都没有，这个可以讨论是否需要展示。

读者界面：

1. 有点好奇这个几十万字了会如何显示



2. 上一章下一章显示的是章节加载失败，但也可能是目前没有下一章或者上一章，这里能否处理呢。

3. 如图显示有问题：



5. 评论展示仍然需要优化

同时可以考虑在加载时显示加载中



4. 这个返回键返回是否合理 ✓



5. 无评论时展示有问题

 首页

 账务中心

 我的书评

我发布的评论及回复

 错误: 获取评论失败

 个人信息

6. 默认用户头像展示有误

修改个人信息

 **头像预览**

 **选择头像**

8月25日任务分配

2025.8.24

步入数据库课程设计最后阶段。

原定于8.24完成所有前后端代码编写工作，但是如果有未完成部分视情况分配同学微调。（考虑zhx同学前端做得很好可以让他负责）

前后端服务器部署将即刻展开，部署方案暂定，由gyx和gy负责。

我们原本的服务器应该无法支撑到答辩，为了答辩准备期间和答辩时正常进行，先征求续期一个月意见。zyh同学今天要打开下云平台。

工作目标：9.14数据库答辩和全部材料提交

↳

· 成绩评定：↳

- 考勤：15%↳
- 第一次作业：5%↳
- 提交分组名单，并确定项目选题和系统功能↳
- 第二次作业：15%↳
 1. 提交系统的数据库设计文档↳
 2. 在 Oracle 中创建好数据库↳
- 课程项目：65%↳
 1. 编码实现课程项目，并进行测试，部署或生成可执行程序↳
 2. 撰写系统需求分析、设计与实现文档，撰写答辩 PPT↳
 3. 项目答辩及演示↳

答辩前，组长将每位组员的分数占比交给助教。每位组员的项目分数=每组项目的分数*每个组员的分数占比。↳

答辩前，组长将每位组员的考勤分交给助教，同时提交给出考勤分的依据，如交流记录、github 上的代码上传、修改记录。↳

↳

↳

数据库清洗

(1) 删除已有数据--可以采用数据库直接清除data的方式

(2) 增加新数据：这个必须规划下再完成：

1.拟定三端用于展示的账号，读者x1，作者x1，管理员x1，功能点展示基于这三个账号来。确保这三个账号下所有的东西都齐全。

2.其它账号用于完善小说平台。

3.增加数据最好使用前端完成。

举例：

暂定于上一次安排的后端三位同学完成。

两个文档编写（9.1号至少要出大纲和每个模块至少都要有一定的文字展示）

系统设计与实现文档：（2人）

系统需求分析文档：（2人）

多参考学长学姐的写法，加上我们以前写过的资料，有不清楚的地方群里面沟通。

答辩PPT制作（3人，答辩同学一起总体规划，可以等文档写得差不多了再开始）

9.7-9.10会公布成员比例表，到时候会把每个人的完成任务信息都填在一张表里面。答辩可能需要多台电脑，这个也后面再安排。

部署

文档:如何连接远程桌面 https://www.cnblogs.com/ysocean/p/6675025.html#_label1

🚀 方案 A 部署步骤

1. 确认服务器环境

- 操作系统: 确认是 Windows Server 还是 Linux (不同系统部署方式不一样)。
- Oracle: 已经安装并能用。
- 网络: 能通过公网 IP 访问到服务器。

👉 建议关掉数据库的公网暴露, 只留应用能访问。

2. 部署后端 (ASP.NET Core)

1. 在服务器上安装 .NET SDK/Runtime。

```
1 dotnet --version
```

确认能跑。

2. 在本地项目里发布:

```
1 dotnet publish -c Release -o ./publish
```

3. 把 `publish` 文件夹上传到服务器。

4. 在服务器上运行:

```
1 cd publish
2 dotnet YourApp.dll
```

默认监听 <http://localhost:5000>

5. 为了能让别人能访问：

- Windows Server → 用 IIS 部署，绑定 80/443。
- Linux → 用 Nginx 反向代理，80/443 → 转发到 5000。

3. 部署前端 (Vue)

1. 在本地打包：

```
1 npm run build
```

得到 `dist/` 文件夹。

2. 上传到服务器：

- Windows IIS → 放到 `C:\inetpub\wwwroot\`
- Linux Nginx → 放到 `/usr/share/nginx/html/`

3. 修改前端的 API 地址 (axios/fetch 的 baseURL)，指向你们的服务器域名/IP + 后端端口。

4. 联通性检查

- 打开浏览器访问 `http://服务器IP/` → Vue 前端页面能出来。
- 前端请求 API 时能访问 `http://服务器IP/api/...`。
- 后端能正常连接 Oracle。

5. (可选) 加域名 + HTTPS

- 买一个域名，解析到服务器 IP。

- Linux: 用 Nginx + Certbot 免费证书。
 - Windows: IIS 直接申请或导入证书。
-

📌 架构示意图（方案 A）



push的流程

为方便管理合并代码

1

大家不要直接push到我们前端的主分支frontend上了，自己创建个分支交。

你add和commit完后，执行

```
1  git push -u origin master:frontend-by-zhouhuixing #为方便知道谁的，换成你的名字拼音
```

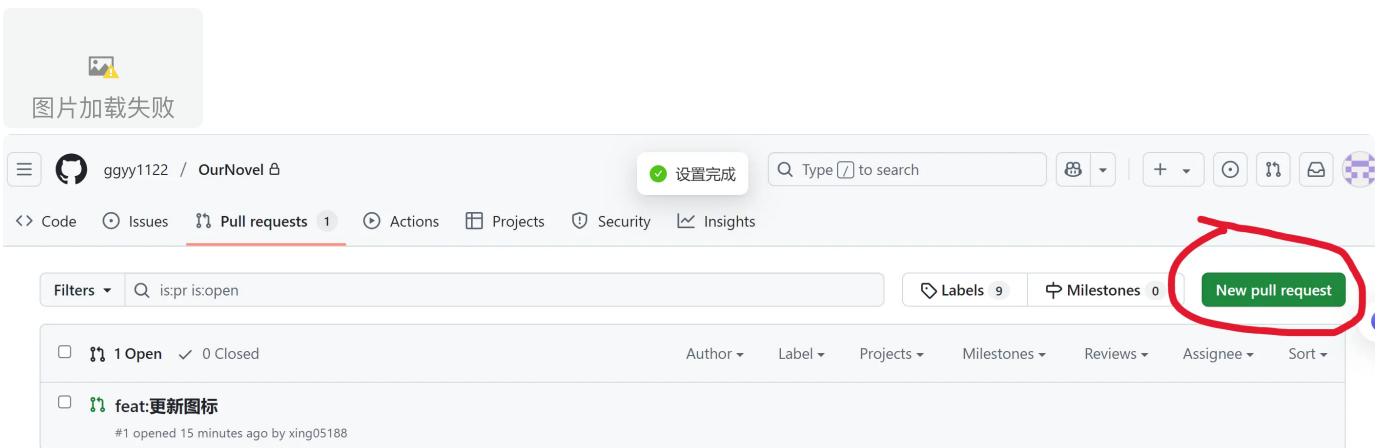
master是你本地主分支，如果是main则换成main

这指令会自动创建分支（如果不存在的话），以后push就只用这个就行了。

2

push之后，打开我们的项目 GitHub 仓库页面

点击 "New Pull Request" 按钮



图片加载失败

ggyy1122 / OurNovel

Code Issues Pull requests 1 Actions Projects Security Insights

Filters Q is:pr is:open Labels 9 Milestones 0 New pull request

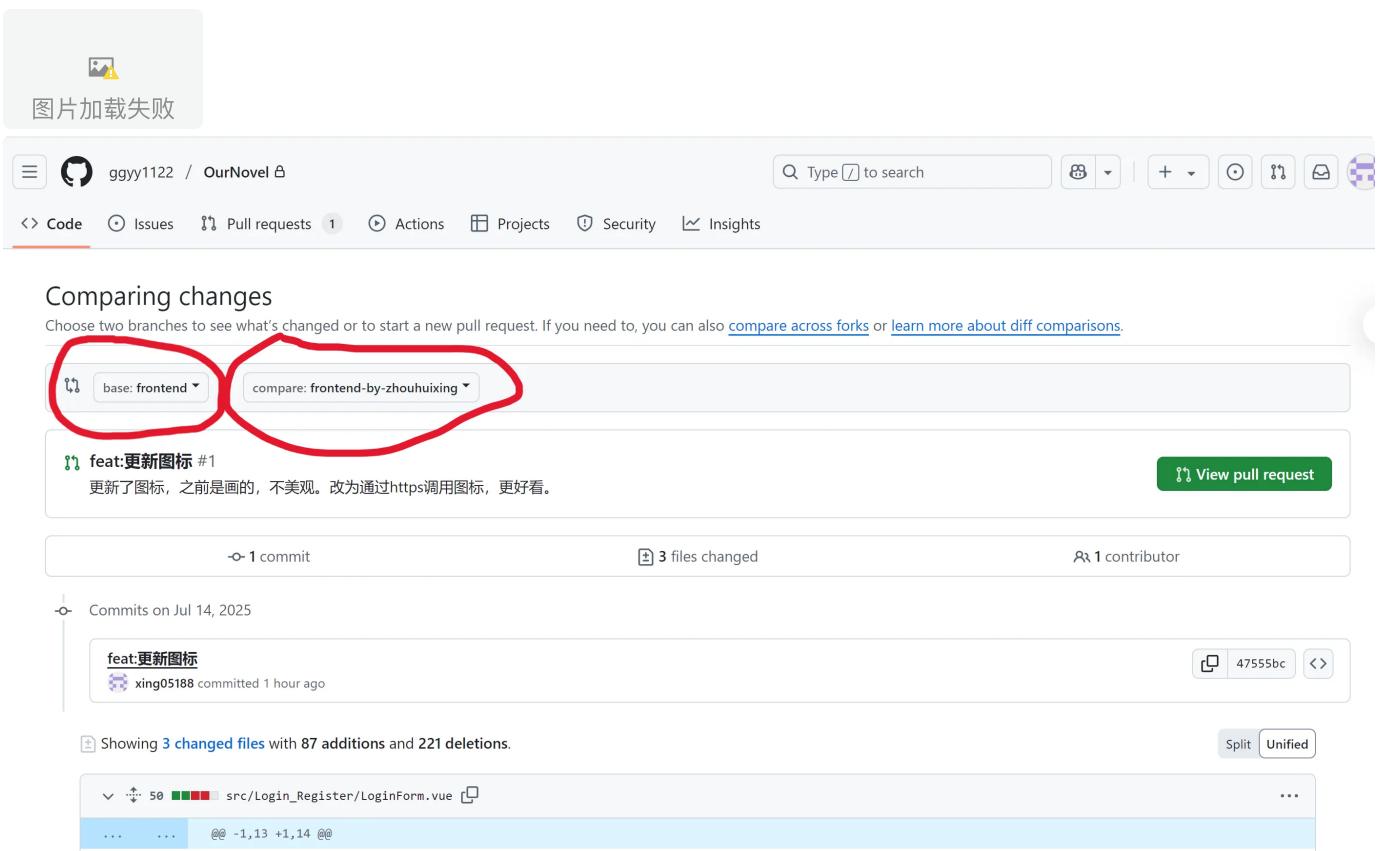
1 Open 0 Closed

feat:更新图标 #1 opened 15 minutes ago by xing05188

ProTip! Mix and match filters to narrow down what you're looking for.

© 2025 GitHub, Inc. Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information

把base换成frontend分支,compare换成你自己的分支, 然后点继续



图片加载失败

ggyy1122 / OurNovel

Code Issues Pull requests 1 Actions Projects Security Insights

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#) or [learn more about diff comparisons](#).

base: frontend compare: frontend-by-zhouhuixing

feat:更新图标 #1

更新了图标, 之前是画的, 不美观. 改为通过https调用图标, 更好看.

View pull request

-o 1 commit 3 files changed 1 contributor

Commits on Jul 14, 2025

feat:更新图标 xing05188 committed 1 hour ago

Showing 3 changed files with 87 additions and 221 deletions.

src/Login_Register/LoginForm.vue

@@ -1,13 +1,14 @@

填写 PR 标题和描述 (建议包含: 改动目的、主要实现方式、相关 Issue 编号 (如果有) 、测试说明)

反正就说下你交了什么新东西。

3

然后，就等待审核，同意后就合并（注意：你不要自己审核合并），找个人来审核，比如组长。

待合并后，其他人就可以执行：

```
1 git pull origin frontend
```

来获取最新的代码了，如果你pull的时候已经改了代码，建议在本地建个分支来放，这样不会影响你正在写的代码，到时候自己在本地合并就好了，然后再按第1步push到远端库。

仓库克隆

参考资料，这别人的前端搭建

[管理员界面](#)

[小说界面](#)

初始化本地Git仓库

创建个空文件夹，用vscode打开，作为你的前端项目根目录。

在你的前端项目根目录（例如 我的 `D:\vue_project\data_project>`）下执行，vscode打开终端即可：

只克隆 frontend 分支

使用 `--single-branch` 参数只克隆指定分支：

```
1  git clone --single-branch --branch frontend https://github.com/ggyy1122/Our  
Novel.git
```

若连不上，用ssh服务，否则跳过：

ssh配置(你https连不上仓库时)

1. 检查是否已有 SSH 密钥（若你之前用过）

```
1  ls -Force ~/.ssh  # 查看是否存在id_ed25519.pub 文件
```

- 若存在，可直接跳到步骤 3。
- 若不存在，需生成新密钥。

2. 生成新的 SSH 密钥

```
1 ssh-keygen -t ed25519 -C "your_email@example.com" #填你github绑定的邮箱
```

按提示完成操作（可直接回车使用默认路径和密码）。

3. 复制公钥到剪贴板

```
1 cat ~/.ssh/id_ed25519.pub | clip # Ed25519 密钥
```

4. 将公钥添加到 GitHub 账户

1. 登录 GitHub → 点击右上角头像 → Settings → SSH and GPG keys。
2. 点击 **New SSH key**。
3. 添加标题（如 `My PC`），粘贴剪贴板中的公钥内容。
4. 点击 **Add SSH key**。

5. 测试 SSH 连接

```
1 ssh -T git@github.com
```

- 首次连接会提示确认指纹，输入 `yes`。
- 若看到 `Hi username! You've successfully authenticated...`，则连接成功。它会显示叉，是对的。

6. 克隆仓库

```
1 git clone --single-branch --branch frontend git@github.com:ggyy1122/OurNove
l.git
```

6. 修改 Git 远程仓库 URL

将 HTTPS 地址替换为 SSH 格式：ba

```
1 # 查看当前远程地址
2 git remote -v
3
4 # 修改为 SSH 地址
5 git remote set-url origin git@github.com:ggyy1122/OurNovel.git
6
7 # 验证修改
8 git remote -v
```

推送代码

你可以修改下test.md验证：

使用 SSH 协议推送：

```
1 git add .
2 git commit -m "test:测试"
3 git push -u origin master:frontend
```

日常开发流程

```
1 # 拉取最新代码
2 git pull origin frontend # 拉取远程前端分支
3
4 # 开发新功能
5 # ...
6
7 # 提交代码
8 git add .
9 git commit -m "信息"
10 git push # 推送到远程
```

配置

首先安装项目需要的依赖：

```
1 npm install axios vue-router vuex element-plus --save
```

Project setup

```
1  npm install
```

Compiles and hot-reloads for development

```
1  npm run serve
```

前端vue环境搭建

- 配置环境
- 使用 VSCode 打开项目文件夹
 - 启动开发服务器
 - 可能遇到的问题
 - 安装插件扩展
- 仓库

配置环境

参考: [CSDN 博客](#)

在安装cnpm时用这个 :

```
1  npm install vue -g --registry=https://registry.npmjs.org --no-optional --strict-ssl=false
```

使用 VSCode 打开项目文件夹

用 VSCode 打开你创建的项目文件夹。代码部分在 `src` 文件夹中。你开始 `npm run server` 看到界面就对应 `App.vue` 。

启动开发服务器

在终端输入以下命令启动开发服务器:

```
1  npm run serve
```

可能遇到的问题

如果遇到如下错误：

```
1 PS D:\vue_project\data_project> npm run server
2 npm : 无法加载文件 D:\VUE-Node-js\npm.ps1, 因为在此系统上禁止运行脚本。有关详细信
息, 请参阅 https://go.microsoft.com/fwlink/?LinkID=135170 中的 about_Execution
 _Policies。
3 所在位置 行:1 字符: 1
4 ● npm run server
5     + CategoryInfo          : SecurityError: (:) [], PSSecurityException
6     + FullyQualifiedErrorId : UnauthorizedAccess
```

在 VS Code 终端中选择“以管理员身份运行”。

1. 关闭 VS Code。
2. 右键点击 VS Code 图标, 选择“以管理员身份运行”。
3. 重新打开项目文件夹并启动终端。

在管理员终端中执行以下命令 (输入 `Y` 确认) :

```
1 Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

验证策略修改成功:

```
1 Get-ExecutionPolicy -List
```

确保 `CurrentUser` 的策略为 `RemoteSigned`。

安装插件扩展

开发 Vue.js 项目时, 推荐安装以下扩展来提升开发体验:

1. vue(official): 官方 Vue.js 扩展, 提供基本的 Vue 支持。
2. ESLint: 帮助你保持代码风格一致并发现潜在错误。
3. Prettier – Code formatter: 自动格式化代码。

仓库

可能要一个人要创建项目后, 放到 GitHub 上, 其他人克隆到本地。

这样大家就可以在本地运行管理相同的项目。

前端进展

进展：

基本结构已弄好：

views中的Home_test.vue

这是为方便测试做的引导界面，比如你想测试个vue，在其中加个按钮，由router切换到你的vue来看测试效果。

store文件夹

利用pinia来管理整个项目的共享资源，比如全局变量。

我添加了个current_state 来定位当前是（读者、作者、管理员）三者之一，我以此对三者的登录和注册进行了合并。

router文件夹

就是定义路径，在各界面切换。

Novels文件夹

实现小说阅读界面（读者），我做了点，现在不用管。

Login_Register文件夹

已实现登录和注册功能，不用管了，以后就只需修改利用router切换的部分（完成管理员界面、作者界面等等）。

assets文件夹

放资源。

API文件夹

定义API，与后端交互。

Admin文件夹

管理员部分，简陋版，你们写完了对文件替换。就是在这文件夹里写。

关于小说状态的梳理和思考

7月18日所说的关于小说的修改如下：

novel表：

新增OriginalNovelId属性

novel状态：待审核，连载，完结，封禁 当时说了还有一个待上架??

下面进行说明

下面再次进行梳理：

首先从前端角度来说：

①对于小说没有提供删除数据库记录的接口

②小说待上架状态，是作者提出创建小说后，小说的最初状态，此状态在作者界面和管理员界面显示相同内容

③待上架的小说经过审核会有两种结果，一，审核通过，小说成为了连载态，连载态在三个平台显示相同内容。二，审核不通过，，小说变成什么状态？待审核？但是好像显示逻辑好像不一样。个人觉得可以再增加一个状态，未上架，该状态和上架状态的显示逻辑是一致的，但是待上架状态不可以再提出上架申请了，而未上架可以修改后继续提交申请。

④对于连载态，三个平台显示一样内容

⑤连载态的小说被修改后会进入待审核状态，待审核状态的显示逻辑是，读者界面显示原内容，作者界面两者都显示还是怎么样??? 管理员界面呢？管理员界面当时好像是说小说栏是显示原内容，审核栏显示新内容，这实现起来都一样的，主要是前端同学觉得从用户界面角度来看哪个合理一些。

⑥完结态，好像仅仅是改一个状态，至于完结小说是否能再修改，这个后续考虑。

⑦封禁态，读者界面不显示，作者和管理员界面显示相同内容。

从前端同学角度来看，读者和作者关于小说的显示应当是简单的，

①读者界面:待上架和未上架的作品均不显示,连载状态的作品直接显示,待审核的作品显示原内容,完结状态直接显示,封禁状态不显示

②作者界面:全部显示最新内容

管理员会相对复杂:

管理员会有一个界面预览所有小说信息,这个界面的预览内容应当是和读者一致的。意思是未上架,待上架,封禁的小说全部不显示,待审核的显示原内容,连载的就显示新内容,和读者的逻辑完全一致。

管理员还有管理界面,作者发出了创建小说等请求,读者发出举报等请求,这些涉及到的小说项都会在这个界面显示,管理员可以选择切换它们的状态

然后从后端和数据库实现角度来说:

(前端同学可以不需要关注实现,但是后端实现同学一定要理清楚)

①数据表设计

a.小说6状态:待上架,未上架,待审核,连载,完结,封禁

b.小说表新增字段:新旧小说标识符,默认值为-1,旧内容该字段均为-1,新内容指向新内容,该字段记录旧内容的id

②业务梳理

a.小说上架业务

作者:

1.在小说表新增一条内容 状态待上架 标识字段为-1, 注意这条记录在待上架时不可以更新, 未上架时可更新。(但是这个由前端实现即可)

管理员:

2.筛选出待上架的小说,开始处理 (不要写一个单独的api,写一个公共的api,用于多种筛选)

3.如果允许上架,修改小说状态为连载,增添小说管理记录,以上操作要有原子性

4.如果不允许上架,修改小说状态为待上架,增加小说管理记录,以上操作要有原子性。

b. 小说更新业务，比如更新id=100的小说

作者：

1. 在小说表新增一条内容，状态为待审核，标志字段记录100(id为100的小说的标志字段为-1)

管理员：

2. 筛选出待审核且标志位不为-1的小说

3. 如果审核通过，就将它的内容复制到标志指向的那一行，并且删除当前行，记得把这个记录加入到管理员记录表中

4. 如果审核不通过，就简单处理，将它恢复到连载，删除当前行，并且将处理记录加入到管理员记录表

读者：

5. 筛选出状态为连载或者待审核且标志位为-1的小说，这些才是读者界面显示的小说。

c. 小说完结业务

作者：

修改状态即可

d. 小说封禁业务

我们好像没有举报小说？？

那封禁是纯管理员操作。

管理员：

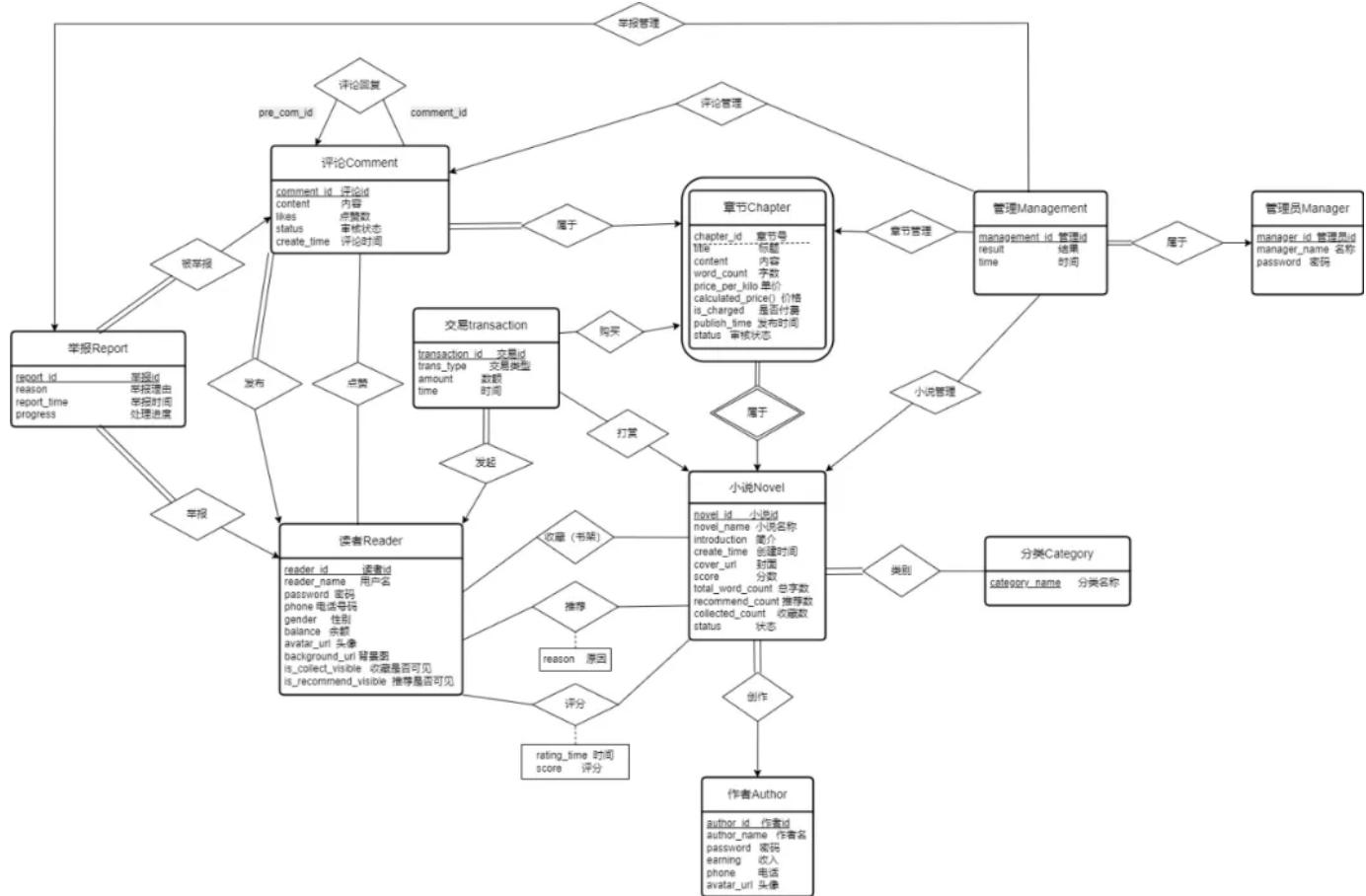
① 修改小说状态，并且将处理记录加入到管理员记录表

作者：

① 修改小说内容，修改状态，状态重新变为待上架状态，重新变为了上架业务。

关于交易的梳理和思考

1.先把ER图摆上



2.再梳理相关数据表

transaction表

| 字段名 | 数据类型 | 长度 | 说明 | 备注 |
|----------------|----------|----|-------|----------------------------|
| reader_id | number | | 读者标识符 | 外码, 参照reader表中的reader_id属性 |
| transaction_id | number | | 交易标识符 | 主码 |
| trans_type | varchar2 | 10 | 交易类型 | “打赏”“充值”“解锁章节” |

| | | | | |
|--------|------|--|------|----|
| amount | int | | 数额 | 非空 |
| time | date | | 交易时间 | |

reader表

| 字段名 | 数据类型 | 长度 | 说明 | 备注 |
|-----------------------|----------|-----|--------|------------|
| reader_id | number | | 读者标识符 | 主码 |
| reader_name | varchar2 | 20 | 用户名 | 非空 |
| password | varchar2 | 20 | 密码 | 非空 |
| phone | char2 | 11 | 电话号码 | |
| gender | char | 2 | 性别 | 只能为“男”或“女” |
| balance | int | | 余额 | 默认0 |
| avatar_url | varchar2 | 255 | 头像 | 存储图片路径 |
| background_url | varchar2 | 255 | 主页背景 | 存储图片路径 |
| is_collect_visible | varchar2 | 2 | 收藏是否展示 | 是/否 |
| is_recommennd_visible | varchar2 | 2 | 推荐是否展示 | 是/否 |

purchase表

| 字段名 | 数据类型 | 长度 | 说明 | 备注 |
|-----|------|----|----|----|
|-----|------|----|----|----|

| | | | | |
|----------------|--------|--|-------|--|
| transaction_id | number | | 交易标识符 | 主码，外码，参照 transaction 表中的 transaction_id 属性 |
| chapter_id | number | | 章节号 | 外码,参照 chapter 表中的 chapter_id 属性 |
| novel_id | number | | 小说标识符 | 外码，参照 novel 表中的 novel_id 属性 |

reward表

| 字段名 | 数据类型 | 长度 | 说明 | 备注 |
|----------------|--------|----|-------|--|
| transaction_id | number | | 交易标识符 | 主码，外码，参照 transaction 表中的 transaction_id 属性 |
| novel_id | number | | 小说标识符 | 外码,参照 novel 表中的 novel_id 属性 |

3. 梳理业务

用户交易

会在交易表中留下交易记录

问题1:这个记录是否在读者界面展示，这个记录是否在管理员界面展示

交易下下细分三种交易:

三个子交易部分都可以用数据库的函数或过程来实现

充值:

用户充值要完成的事:

- ①修改交易表
- ②更新自身余额, 这是修改用户表

以上步骤要有原子性

购买:

用户购买要做的事:

购买是购买具体某本书的某章节,所需参数有用户id, 章节id, 书籍id

- ①修改交易表
- ②更新自身余额, 这是修改用户表, 怎么实现?触发器? 每次修改交易表都会触发
- ③修改购买表
- ④修改章节表, 修改它的状态

以上步骤要有原子性

打赏:

我们是打赏小说

用户打赏要做的事:

- ①修改交易表
- ②更新自身余额, 这是修改用户表, 怎么实现?触发器? 每次修改交易表都会触发
- ③修改打赏表

以上步骤要有原子性

需要处理作者那边的收入更新

问题2:是否要记录作者每次收入的时间, 金额, 这需要额外一个收入表。

其它核心:

①如果展示交易表, 那么一个交易表就涉及多个查询, 对类型, 对时间, 还涉及汇总, 比如一个月的交易额, 这些还需要细化

②可以建立用户打赏视图, 返回用户打赏的书籍, 以及作者, 用于展示打赏过的书籍和作者

③可以建立用户购买视图, 返回用户购买的书籍和作者

④打赏行为对作者收入的更新

方案1: 设计一个存储过程, 每天执行一次, 逻辑是查询一天内所有他的所有书的被打赏的金额, 这样貌似不好, 涉及很多无用的全表扫描

方案2: 实时更新

⑤已加入作者收入表

| 列名 | # 类型 | 类型修饰符 | 非空 | 默认 | 注释 |
|---------------|----------------|-------|-----|--------------------------------------|----|
| 123 ID | 1 NUMBER(19,0) | | [V] | "MAIN_USER"."ISEQ\$\$_74397".nextval | |
| 123 AUTHOR_ID | 2 NUMBER(19,0) | | [V] | | |
| AZ TYPE | 3 VARCHAR2(20) | | [V] | | |
| 123 AMOUNT | 4 NUMBER(10,2) | | [V] | | |
| ⌚ CREATE_TIME | 5 TIMESTAMP | | [V] | SYSTIMESTAMP | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

4.关于充值

1.流程

api 提供充值api 读者id 金额的输入

事务处理流程:

先是调用支付宝api进入支付界面，支付成功会触发后续业务服务
该api要把读者id传入业务服务。

2. 充值api使用流程

api测试前提:

①支付宝沙盒账号的买家账号和密码

大家可以自己注册，大家每个人的支付宝账号都有一个沙盒账号。

②公网暴露

支付宝的支付后的响应需要从支付宝服务器向后端服务器发送响应，但是我们本地机器属于私有网络，支付宝服务器无法送达，在我们正式部署项目前，我们每次测试支付功能都需要将我们本地后端的端口暴露到公网。步骤如下：

1. 下载最新版的ngrok

2. 注册账号，获取认证

3. 在命令行执行命令ngrok http https://localhost:7109，这个7109是我们后端监听的端口，如果大家的机器上后端监听的不是7109就改为自己机器上对应的。

会出现这样的界面，这表示成功暴露到公网

```
ngrok
Decouple policy and sensitive data with vaults: https://ngrok.com/r/secrets

Session Status          online
Region                 ggyy1122 (Plan: Free)
Version                3.23.3
Latency                107ms
Web Interface          http://127.0.0.1:4040
Forwarding             https://0c2b19d96520.ngrok-free.app -> https://localhost:7109

Connections            ttl     opn     rt1     rt5     p50     p90
                        3       0       0.00   0.00   90.15   90.49
```

把这条 <https://0c2b19d96520.ngrok-free.app> 写到后端appsettings.json处，

注意：每次暴露都可能不同，所以测试阶段每次暴露都要写一次。

```
    ],
    "BaseUrl": "https://0c2b19d96520.ngrok-free.app", // 替换为你的实际ngrok地址
    "Allow" : "
```

api正式使用：

①需求:为id=27的读者充值10000元

```
[  
  {  
    "readerId": 27,  
    "readerName": "andy",  
    "password": "f8lpls2wEp7PMIz3u1pQ",  
    "phone": null,  
    "gender": null,  
    "balance": 302,  
    "avatarUrl": "787d524b-f3cc-455e-b3fc-84fd676753d4.jpg",  
    "backgroundUrl": null,  
    "isCollectVisible": null,  
    "isRecommendVisible": null  
  },  
]
```

②调用Recharge api

Recharge

POST /api/Recharge/start 发起充值并返回支付URL

Parameters

No parameters

Request body

application/json

```
{  
  "readerId": 27,  
  "amount": 10000  
}
```

注意:前端有关充值的只需调用这个api, 不需调用其他api

③给前端返回了支付界面的url, 前端打开

Server response

Code Details

200 Response body

```
{  
  "paymentUrl": "https://openapi-sandbox.d1.alipaydev.com/gateway.do?alipay_sdk=alipay-sdk-net-4.9.627.ALL&app_id=20210800149611133&biz_content=%7b%22out_trade_no%22%3a%22recharge_27_20250717213542Z22Zk2Zk2pproduct_code%22%3a%22FAST_INSTANT_TRADE_PAY%22%2c%22qr_code_width%22%3a0%2c%22subject%22%3a%22ke7x94%a8x6eKs%e%72%7e4%bd%99%e9%a2%9d%e5%8x5%e5%80%bc%22%2c%22total_amount%22%3a%2210000.00%22%2d& charset=UTF-8&format=json&method=alipay.trade.page.pay&notify_url=https%3a%2f%2fmc%2fc%2b19d96528.ngrok-free.app%2fapi%2frecharge%2fnotify&return_url=https%3a%2f%2fmc%2fc%2b19d96528.ngrok-free.app%2fapi%2frecharge%2freresult%3fuser_id%3d27&sign_type=RSA2&timestamp=2025-07-17+21%3a35%3a42&version=1.0&sign=0TVxfwv6xGd7hDvZyTfTpQw6alzEbVgX4K5t1n108f71f0q1pskZkU1AgZYhulin04Op%2fbahxq1rC31px2faFkSC%22%2fs%31Pxvh2tlihB5akKk3pcq33XjRDZcB08WlfskX2bQ%2bwrt3KnfWrb9ccpwrmw%2be0Mj06Mkg12DB7s7BrcGn6WosSz%2boZJHgIVuq161d8fmJwUR%2f8EXEYeszBrbVfuxkk%2fssSwefFrhYnHE%2bk%2fp053cm6CfE2MVuLrXF4MeOTQx9Bdg9Agr9Rwtxet2vANRRek0SwVdt27VsxB68Y6wrT1GJN%2f17pnwpzYzecUkx7MkK2uL7H1sqUQ%3d%3d"  
}
```

Download

Response headers

```
content-type: application/json; charset=utf-8  
date: Thu, 17 Jul 2025 13:35:42 GMT  
server: Kestrel
```

Responses

Code Description Links

界面如下, 输入沙盒账号和密码

正在使用即时到账交易 [?]

用户27余额充值 收款方: jtxhgm0678@sandbox...

10000.00 元

扫码支付

登录支付宝账户付款 新用户注册

账户名: 忘记账户名?

手机号码/邮箱

支付密码: 忘记密码?

请输入账户的支付密码, 不是登录密码。

下一步

ICP证: 合字B2-20190046

支付成功，自动调用后端的后续处理机制

支付宝账户: gprcdy9511@sandbox.com | 找人代付 | 帮助

支付宝 | 我的收银台

您已成功付款**10000.00 元**！

对方将立即收到您的付款。
如果您有未付款信息，[查看并继续付款](#)

您可能需要: [查看余额](#) | [消费记录](#) | [我要充值](#)

ICP证: 沪B2-20150087

重新检查id=27的读者

200

Response body

```
[  
  {  
    "readerId": 27,  
    "readerName": "andy",  
    "password": "fBlpls2wEp7PMIz3u1pQ",  
    "phone": null,  
    "gender": null,  
    "balance": 10302,  
    "avatarUrl": "787d524b-f3cc-455e-b3fc-84fd676753d4.jpg",  
    "backgroundUrl": null,  
    "isCollectVisible": null,  
    "isRecommendVisible": null  
  },  
  {  
    "readerId": 27,  
    "readerName": "andy",  
    "password": "fBlpls2wEp7PMIz3u1pQ",  
    "phone": null,  
    "gender": null,  
    "balance": 10302,  
    "avatarUrl": "787d524b-f3cc-455e-b3fc-84fd676753d4.jpg",  
    "backgroundUrl": null,  
    "isCollectVisible": null,  
    "isRecommendVisible": null  
  }]
```

至此，充值业务完成。

前后端连通

仅做测试用

一、项目文件结构规划

```
1 src/
2   └── api/          # API 请求相关
3     ├── index.js    # API 请求基础配置
4     └── oracleApi.js # 具体的 Oracle API 请求方法
5   └── main.js        # Vue 主入口文件
6   └── App.vue        # 主组件 (用于测试 API)
```

二、实现步骤

1. 前端项目安装 axios

```
1 npm install axios
```

2. 配置 API 请求基础文件 (`src/api/index.js`)

```
1 import axios from 'axios'
2
3 // 创建 axios 实例
4 const service = axios.create({
5   baseURL: process.env.VUE_APP_BASE_API, // 从环境变量获取 API 地址
6   timeout: 5000 // 请求超时时间
7 })
8
9 // 请求拦截器
10 service.interceptors.request.use(
11   config => {
12     // 在这里可以添加请求头等信息
13     // 例如添加 token
14     // if (store.getters.token) {
15     //   config.headers['Authorization'] = `Bearer ${store.getters.token}`
16     // }
17     return config
18   },
19   error => {
20     console.log(error)
21     return Promise.reject(error)
22   }
23 )
24
25 // 响应拦截器
26 service.interceptors.response.use(
27   response => {
28     // 对响应数据做处理
29     return response.data
30   },
31   error => {
32     console.log('Error:', error.response)
33     return Promise.reject(error)
34   }
35 )
36
37 export default service
```

3. 创建 Oracle API 请求文件 (`src/api/oracleApi.js`)

```
1 import request from './index'  
2  
3 // 示例 API 请求方法  
4 export function fetchOracleData(params) {  
5   return request({  
6     url: '/api/oracle/data', // 替换为你的实际 API 地址  
7     method: 'get',  
8     params  
9   })  
10 }  
11  
12 // 添加更多 API 方法...
```

4. 修改 `App.vue` 进行测试

```
1  <template>
2    <div id="app">
3      <h1>Oracle API 测试</h1>
4      <button @click="testOracleApi">测试 API 连接</button>
5      <div v-if="apiResponse">
6        <h3>API 响应数据: </h3>
7        <pre>{{ apiResponse }}</pre>
8      </div>
9    </div>
10   </template>
11
12  <script>
13    import { fetchOracleData } from './api/oracleApi'
14
15  export default {
16    name: 'App',
17    data() {
18      return {
19        apiResponse: null
20      }
21    },
22    methods: {
23      async testOracleApi() {
24        try {
25          const response = await fetchOracleData({
26            // 可以添加请求参数
27            testParam: 'test'
28          })
29          this.apiResponse = response
30          console.log('API 响应:', response)
31        } catch (error) {
32          console.error('API 请求错误:', error)
33          this.apiResponse = { error: error.message }
34        }
35      }
36    }
37  }
38  </script>
```

5. 配置环境变量

在项目根目录创建 `.env.development` 文件:

▼

Plain Text |

```
1 VUE_APP_BASE_API=http://your-api-base-url.com
```

6. 运行测试

▼

Plain Text |

```
1 npm run dev
```

点击页面上的按钮测试 API 连接，响应数据会显示在页面上并打印到控制台。

三、API调用相关的问题

①从后端开始解读

打开 `Properties` 目录下的 `launchSettings.json` 文件，这个文件是 `ASP.NET Core` 项目的配置文件，专门用于定义 `开发环境`下的启动行为（如端口、环境变量、启动方式等）。它不会影响生产环境部署，仅在本地开发时生效。

```
1  {
2      "$schema": "http://json.schemastore.org/launchsettings.json",
3      "iisSettings": {
4          "windowsAuthentication": false,
5          "anonymousAuthentication": true,
6          "iisExpress": {
7              "applicationUrl": "http://localhost:31822",
8              "sslPort": 44352
9          }
10     },
11     "profiles": {
12         "http": {
13             "commandName": "Project",
14             "dotnetRunMessages": true,
15             "launchBrowser": true,
16             "launchUrl": "swagger",
17             "applicationUrl": "http://localhost:5037",
18             "environmentVariables": {
19                 "ASPNETCORE_ENVIRONMENT": "Development"
20             }
21         },
22         "https": {
23             "commandName": "Project",
24             "dotnetRunMessages": true,
25             "launchBrowser": true,
26             "launchUrl": "swagger",
27             "applicationUrl": "https://localhost:7109;http://localhost:5037",
28             "environmentVariables": {
29                 "ASPNETCORE_ENVIRONMENT": "Development"
30             }
31         },
32         "IIS Express": {
33             "commandName": "IISExpress",
34             "launchBrowser": true,
35             "launchUrl": "swagger",
36             "environmentVariables": {
37                 "ASPNETCORE_ENVIRONMENT": "Development"
38             }
39         }
40     }
41 }
42 }
```

关键点:

定义了三种启动配置 (Profiles) :

① **http** Profile (Kestrel 服务器, HTTP-only)

```
1 "http": {  
2   "commandName": "Project",          // 使用 Kestrel  
3   "dotnetRunMessages": true,         // 显示 .NET 运行时日志  
4   "launchBrowser": true,            // 启动后自动打开浏览器  
5   "launchUrl": "swagger",          // 浏览器打开的默认路径 (Swagger)  
6   "applicationUrl": "http://localhost:5037", // 监听 5037 端口  
7   "environmentVariables": {  
8     "ASPNETCORE_ENVIRONMENT": "Development" // 强制开发环境  
9   }  
10 }
```

- 访问方式: `http://localhost:5037/swagger`

② **https** Profile (Kestrel 服务器, HTTP+HTTPS)

```
1 "https": {  
2   "commandName": "Project",  
3   "applicationUrl": "https://localhost:7109;http://localhost:5037", // 双协议  
4   // 其他配置同 http...  
5 }
```

- 访问方式:

- HTTPS: `https://localhost:7109/swagger`
- HTTP: `http://localhost:5037/swagger`

③ **IIS Express** Profile

```
1 "IIS Express": {  
2   "commandName": "IISExpress",      // 使用 IIS Express  
3   "launchBrowser": true,  
4   "launchUrl": "swagger",  
5   // 端口继承自 iisSettings (31822/44352)  
6 }
```

- 访问方式: `http://localhost:31822/swagger`

⑤ 重要结论

| 配置项 | 含义 |
|------------|---|
| 默认启动的端口 | <ul style="list-style-type: none"> - <code>http</code> Profile: 5037 (HTTP) - <code>https</code> Profile: 7109 (HTTPS) - <code>IIS Express</code> : 31822 (HTTP) |
| Swagger 入口 | 所有配置均设置 <code>launchUrl: "swagger"</code> , 启动后自动打开 Swagger 页面。 |
| 开发环境强制标记 | <code>ASPNETCORE_ENVIRONMENT=Development</code> , 确保加载 <code>appsettings.Development.json</code> 。 |
| 服务器选择 | <code>commandName: "Project"</code> <code>= Kestrel; "IISExpress"</code> <code>= IIS Express。</code> |

比如说对于一个api, `/api/Reader`, 可以直接在swagger界面测试它, 很直接

Reader 读者控制器，继承基类控制器，实现对 Reader 实体的 CRUD 操作

GET /api/Reader 获取所有实体数据

Parameters

No parameters

Responses

| Code | Description | Links |
|------|-------------|----------|
| 200 | OK | No links |

可以在后端运行时，直接访问 <https://localhost:7109/api/Reader>

```
[{"readerId":1,"readerName":"string","password":"string","phone":"string","gender":"男","balance":0,"avatarUrl":"string","backgroundUrl":"string","isCollectVisible":false,"isRecommendVisible":false}, {"readerId":2,"readerName":"string","password":"string","phone":"string","gender":"男","balance":0,"avatarUrl":"string","backgroundUrl":"string","isCollectVisible":false,"isRecommendVisible":false}]
```

试一下http，<http://localhost:5037/api/Reader>，自动转为https？不知道什么原理

```
[{"readerId":1,"readerName":"string","password":"string","phone":"string","gender":"男","balance":0,"avatarUrl":"string","backgroundUrl":"string","isCollectVisible":false,"isRecommendVisible":false}, {"readerId":2,"readerName":"string","password":"string","phone":"string","gender":"男","balance":0,"avatarUrl":"string","backgroundUrl":"string","isCollectVisible":false,"isRecommendVisible":false}]
```

②前端如何使用后端的api

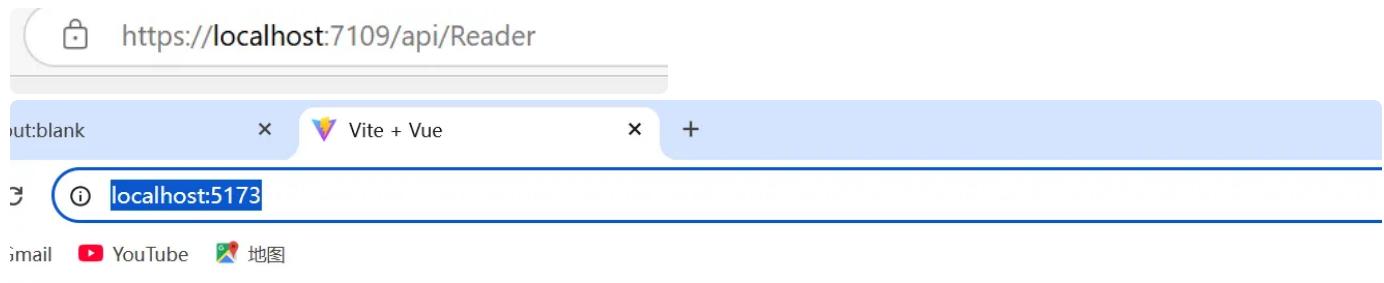
在前面可以看到访问api能直接得到数据，前端访问得到json格式的数据后利用这些数据就可以。我们采用https，就是利用一条条这样的api<https://localhost:7109/api/Reader> 获得数据。

③CORS (Cross-Origin Resource Sharing, 跨域资源共享)

这是前后端分离开发中必须解决的关键问题。完成前两步前端无法直接利用后端api直接获得数据。

1. 为什么需要配置 CORS?

- 场景：前端（Vue）运行在 `http://localhost:5173`，后端（.NET）运行在 `https://localhost:7109`。



Vite + Vue

- 问题: 浏览器会因 **同源策略** 阻止跨域请求, 导致前端调用 API 失败 (错误如 `No 'Access-Control-Allow-Origin' header`) 。

2. 在 .NET 后端配置 CORS

步骤 1: 安装 CORS 服务

确保项目中已安装 `Microsoft.AspNetCore.Cors` 包 (.NET 6+ 默认包含) 。

步骤 2: 修改配置文件

在 `appsettings.json` 添加如下内容

```
1  {
2    "AllowedOrigins": [
3      "http://localhost:5173",
4    ]
5  }
```

步骤3: 在 `Program.cs` 中配置

步骤4: 验证 CORS 是否生效

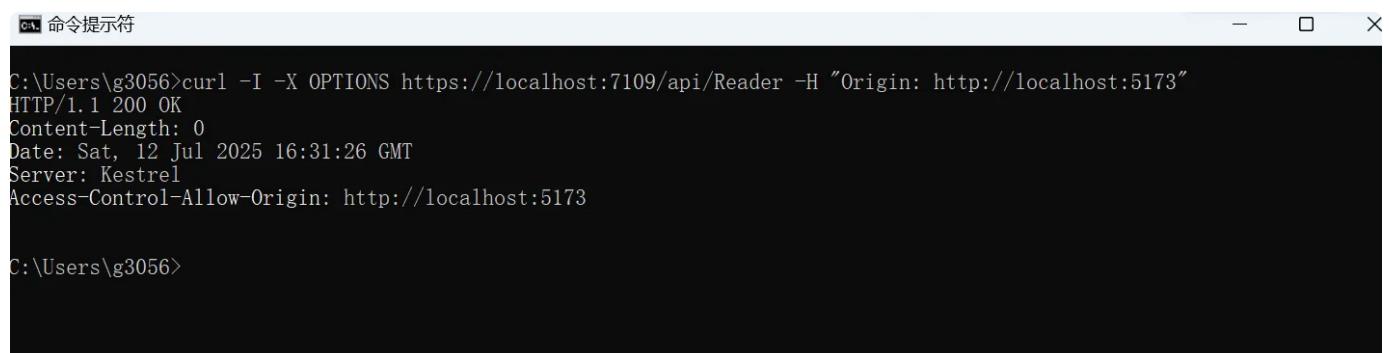
1. 启动后端项目。
2. 由于我们前端还没搭建好用下述命令在命令行验证

```
1 curl -I -X OPTIONS https://localhost:7109/api/Reader -H "Origin: http://localhost:5173"
```

这个 `curl` 命令用于 **手动发送一个 HTTP OPTIONS 请求** 到后端 API, 目的是测试 CORS (跨域资源共享) 配置是否生效。以下是详细分解:

| 部分 | 作用 |
|---|--|
| <code>curl</code> | 命令行工具, 用于发送 HTTP 请求 |
| <code>-I</code> | 仅显示响应头 (不显示响应体) |
| <code>-X OPTIONS</code> | 指定 HTTP 方法为 OPTIONS (预检请求) |
| <code>https://localhost:7109/api/Reader</code> | 目标 API 地址 (.NET 后端) |
| <code>-H "Origin: http://localhost:5173"</code> | 模拟前端页面的来源 (<code>http://localhost:5173</code> 是典型的前端开发地址) |

测试成功



```
C:\Users\g3056>curl -I -X OPTIONS https://localhost:7109/api/Reader -H "Origin: http://localhost:5173"
HTTP/1.1 200 OK
Content-Length: 0
Date: Sat, 12 Jul 2025 16:31:26 GMT
Server: Kestrel
Access-Control-Allow-Origin: http://localhost:5173

C:\Users\g3056>
```

四、一个问题:图像等数据的传输

前面我们已经能够通过api，从前端与数据库进行交互，我们获取数据以json格式获取，我们修改增加数据也同样以json格式调用后端api与数据库进行交互。

但是，图像数据没有存储在json文件里，我们在数据表中存储的是图像的位置信息，下面我们来解决图像信息的存储与访问问题。

后端搭建笔记

注:记录后端搭建的过程

📦 小型 .NET + EF Core 项目结构

```
1 OurNovel/
2   └── Controllers/          # 控制器
3     └── UserController.cs
4
5   └── Data/                 # EF Core 上下文、Migrations
6     └── ApplicationDbContext.cs
7     └── Migrations/
8
9   └── Models/               # 实体类
10    └── User.cs
11
12  └── Repositories/         # 仓储接口和实现
13    └── IRepository.cs
14    └── Repository.cs
15    └── IUserRepository.cs  # 泛型仓储接口
16      # 泛型仓储实现
17      # 特殊方法仓储接口 (如果需要)
18
19  └── Services/              # 业务服务
20    └── UserService.cs
21
22  └── Program.cs            # 启动配置, 依赖注入、管道
23  └── appsettings.json       # 配置文件
24  └── MyProject.csproj        # 项目文件
```

📌 每个文件夹干啥

| 目录 | 内容 |
|--------------|-----------------------|
| Controllers/ | WebAPI 控制器, 处理请求、返回响应 |

| | |
|---------------|---------------------------------|
| Data/ | AppDbContext、EF Core Migrations |
| Models/ | 实体类，映射到数据库表 |
| Repositories/ | 仓储接口和实现，负责数据操作 |
| Services/ | 业务逻辑，调用仓储，供 Controller 用 |

后端api调用问题

- chapter content部分

前端返回json文件时，换行需要转义 '\n' 表示

前端学习笔记

学习可以看官网的详细教程

[Vue.js 官方文档](#)

注：

- ①可以直接在vite构建的vue项目里学习测试各个语法也可以在Vue Playground 在线编写代码片段。
- ②前端同学务必保证风格统一，以及实现的方式统一。
- ③请务必先把每个语法自己过一遍，不要全部依赖ai。可以在共享文档总结一些用法。

一、Vue 基础核心

1. Vue 实例与生命周期 ★

- `new Vue()` 创建的实例是应用根组件
- 生命周期钩子（重点掌握）：

```
▼ JavaScript  
1  created() {}      // 实例创建完成, DOM未生成  
2  mounted() {}     // DOM已挂载 (可操作DOM)  
3  updated() {}     // 数据更新后  
4  destroyed() {}   // 实例销毁 (需手动清理定时器/事件)
```

2. 模板语法 ★

- 插值: `{{ data }}`
- 指令:

```

1  v-if/v-show      // 条件渲染 (区别: v-if销毁DOM, v-show切换CSS)
2  v-for            // 列表渲染 (必须加 :key)
3  v-bind(:)        // 动态绑定属性 (如 :class, :style)
4  v-on(@)          // 事件绑定 (如 @click)
5  v-model          // 表单双向绑定 (语法糖, 本质是 :value + @input)

```

3. 组件基础

- 组件定义: `.vue` 单文件组件 (SFC)
- 组件通信:

```

1  props: ['data'] // 父 → 子 (单向数据流)
2  $emit('event') // 子 → 父 (事件触发)

```

二、进阶必会知识

1. 组件深入

- 插槽 (Slots) :

```

1  <slot></slot>          // 默认插槽
2  <slot name="header">    // 具名插槽

```

- 作用域插槽: 子组件向父组件传递数据

```

1  <slot :user="userData"></slot>

```

2. 状态管理 (Vuex/Pinia)

- Vuex (Vue 2) 核心概念:

```

1 state: { count: 0 }           // 状态数据
2 mutations: { increment() {} } // 同步修改 (通过 commit 调用)
3 actions: { fetchData() {} }  // 异步操作 (通过 dispatch 调用)

```

- **Pinia (Vue 3 推荐) :**

更简洁的 API, 直接定义 `stores` 并支持 Composition API。

3. 路由 (Vue Router)

- 基础配置:

```

1 const routes = [
2   { path: '/', component: Home },
3   { path: '/about', component: About }
4 ]

```

- 导航守卫:

```
1 router.beforeEach((to, from, next) => { /* 权限控制 */ })
```

三、高级特性与优化

1. 响应式原理

- **Vue 2:** 基于 `Object.defineProperty` (无法检测数组/对象新增属性)
- **Vue 3:** 基于 `Proxy` (全面覆盖响应式场景)
- 手动触发更新: `this.$forceUpdate()`

2. Composition API (Vue 3)

- 核心函数:

```

1 ref()      // 基础类型响应式
2 reactive() // 对象类型响应式
3 computed() // 计算属性
4 watch()    // 监听数据变化

```

- 逻辑复用：将代码组织为可复用的函数（替代 Mixins）。

3. 性能优化

- 减少不必要的响应式数据（如冻结大对象：`Object.freeze`）
- 使用 `v-once` 和 `v-memo`（Vue 3）避免重复渲染
- 异步组件：`(() => import('./Component.vue'))`

四、VUE学习记录

1.代码1

```

1  <script setup>
2  import HelloWorld from './components/HelloWorld.vue'
3  </script>
4
5
6  <template>
7    <div>
8      <a href="https://vite.dev" target="_blank">
9        
10     </a>
11     <a href="https://vuejs.org/" target="_blank">
12       
13     </a>
14   </div>
15   <HelloWorld msg="Vite + Vue" />
16 </template>
17
18
19 <style scoped>
20  .logo {
21    height: 6em;
22    padding: 1.5em;
23    will-change: filter;
24    transition: filter 300ms;
25  }
26  .logo:hover {
27    filter: drop-shadow(0 0 2em #646cffaa);
28  }
29  .logo.vue:hover {
30    filter: drop-shadow(0 0 2em #42b883aa);
31  }
32 </style>
33

```

① <script setup> 部分

- **作用**: 这是 Vue 3 的 Composition API 语法糖

1. 变量/数据声明对比

<script setup> 写法 (Vue 3)

```

1 <script setup>
2   const message = "直接暴露给模板"
3 </script>
4 <template>
5   {{ message }} <!-- 直接使用 -->
6 </template>

```

传统写法 (Vue 2)

```

1 <script>
2   export default {
3     data() {           // 必须包裹在 data() 中
4       return {
5         message: "需要手动return" // 必须return才能用
6       }
7     }
8   }
9 </script>
10 <template>
11   {{ message }}
12 </template>

```

2. 函数/方法对比

<script setup> 写法 (Vue 3)

```

1 <script setup>
2   const sayHi = () => alert('Hi!') // 直接声明
3 </script>
4 <template>
5   <button @click="sayHi">点击</button>
6 </template>

```

传统写法 (Vue 2)

```

1
2  <script>
3    export default {
4      methods: { // 必须包裹在 methods 中
5        sayHi() {
6          alert('Hi!')
7        }
8      }
9    }
10   </script>

```

3. 组件注册对比

<script setup> 写法 (Vue 3)

```

1
2  <script setup>
3    import MyComponent from './MyComponent.vue'
4    // 自动注册，无需多余代码
5  </script>
6  <template>
7    <MyComponent />
8  </template>

```

传统写法 (Vue 2)

```

1
2  <script>
3    import MyComponent from './MyComponent.vue'
4
5    export default {
6      components: { // 必须手动注册
7        MyComponent
8      }
9    }
10   </script>

```

核心区别总结

| | | |
|--------------|--|--|
| 特性 | <code><script setup></code> (Vue 3) | 传统写法 (Vue 2) |
| 变量/数据 | 直接声明, 自动暴露 | 需包裹在 <code>data()</code> 中并 <code>return</code> |
| 函数/方法 | 直接声明, 自动绑定 | 需包裹在 <code>methods</code> 中 |
| 组件注册 | 导入即用 | 需在 <code>components</code> 中注册 |
| 代码量 | 减少约 50% | 冗余代码多 |
| TypeScript支持 | 完美支持 | 需要额外类型声明 |

2. 模版语法

功能：在 HTML 中绑定动态数据

```
1  <template>
2    <!-- 1. 文本插值 -->
3    <p>{{ message }}</p>
4
5    <!-- 2. 属性绑定 -->
6    <div :title="dynamicTitle">悬停查看</div>
7
8    <!-- 3. 事件绑定 -->
9    <button @click="handleClick">点击</button>
10   </template>
```

关键符号：

- `{{ }}`：显示 JavaScript 变量
- `:` 或 `v-bind`：动态绑定属性，不再是普通字符串而是js的变量
- `@` 或 `v-on`：绑定事件

3.响应式数据

功能：声明数据，自动同步到模板

```
▼ JavaScript

1 <script setup>
2   import { ref } from 'vue'
3
4 // 1. 基本数据类型用 ref
5 const count = ref(0)
6
7 // 2. 对象/数组用 reactive (或继续用 ref)
8 const user = reactive({ name: 'Alice' })
9
10 // 3. 函数
11 const increment = () => {
12   count.value++ // 注意 .value
13 }
14 </script>
```

核心 API：

- `ref()`：包装基本类型（如数字、字符串）
- `reactive()`：包装对象/数组

在 Vue 中，用 `ref` 和 `reactive` 创建的响应式数据与普通 JavaScript 数据的核心区别在于：响应式数据能自动触发视图更新。以下是具体对比：

1. 普通 JavaScript 数据

```
▼ JavaScript

1 let count = 0
2 const user = { name: 'Alice' }
3
4 function changeData() {
5   count = 1      // 修改值
6   user.name = 'Bob'
7 }
```

特点：

- 数据变化时，不会自动更新网页上的显示
- 纯 JavaScript 行为，与 Vue 无关

2. Vue 响应式数据

```
1 import { ref, reactive } from 'vue'
2
3
4 const count = ref(0) // 响应式数字
5 const user = reactive({ name: 'Alice' }) // 响应式对象
6
7 function changeData() {
8     count.value = 1 // 修改 ref 需要 .value
9     user.name = 'Bob' // 修改 reactive 直接赋值
10 }
```

特点：

- 数据变化时，自动更新所有用到该数据的地方
- 是 Vue 的核心机制

直观对比（修改数据后的效果）

| 操作 | 普通数据 | Vue 响应式数据 |
|--------------------------------|--------------|-------------------|
| 修改数据 | 需手动操作 DOM 更新 | 自动更新所有相关 DOM |
| 模板中显示 <code>{} count {}</code> | 不会变化 | 实时同步最新值 |
| 调试方式 | 控制台直接打印 | 用 Vue Devtools 追踪 |

4. 组件通信（Props & Emits）

①父传子（Props）

```

1
2  <!-- 父组件 -->
3  <ChildComponent :msg="parentMsg" /> //父组件传递msg给子组件
4
5
6
7  <!-- 子组件 -->
8  <script setup>
9  defineProps({
10    msg: String // 子组件声明变量用于接收
11  })
12 </script>

```

②子传父 (Emits)

子组件

```

1
2  <!-- ChildComponent.vue -->
3  <script setup>
4  const emit = defineEmits(['update']) //子组件自定义事件名
5
6  const sendData = () => {
7    emit('update', 123) // 子组件触发事件并传参123
8  }
9 </script>
10 <template>
11   <button @click="sendData">向父组件传值</button><!--点击按钮调用sendData函数
触发事件-->
13 </template>

```

父组件

```

1
2  <!-- App.vue -->
3  <script setup>
4      import ChildComponent from './ChildComponent.vue'
5
6  const handleUpdate = (num) => {
7      alert(`收到子组件的值: ${num}`)
8  } // 接受到子组件中的传参
9  </script>
10
11 <template>
12 <ChildComponent @update="handleUpdate" />
13 </template>

```

5. 生命周期

常用生命周期钩子

```

1
2  <script setup>
3  import { onMounted, onUnmounted } from 'vue'
4
5  onMounted(() => {
6      console.log('组件挂载完成')
7  })
8
9  onUnmounted(() => {
10     console.log('组件卸载')
11 })
12 </script>

```

其他钩子：

- `onUpdated` : 数据更新后
- `onBeforeMount` : DOM 挂载前

Vue 3 的生命周期函数 (对比 Vue 2)

| Vue 3 函数 | 触发时机 | Vue 2 对应写法 |
|-----------------|---------------|---------------|
| onBeforeMount | DOM 挂载前 | beforeMount |
| onMounted | DOM 挂载完成 | mounted |
| onBeforeUpdate | 数据变化, DOM 更新前 | beforeUpdate |
| onUpdated | DOM 更新完成 | updated |
| onBeforeUnmount | 组件销毁前 | beforeDestroy |
| onUnmounted | 组件销毁完成 | destroyed |

6. 状态管理 (Pinia)

① 没有状态管理时的问题:

1. 组件传参繁琐

当多个组件需要同一份数据时, 需要通过 `props` 层层传递

▼ Markdown

1 父组件 → 子组件 → 孙子组件 → 曾孙组件...

2. 数据不同步

某个组件修改数据后, 其他组件无法自动更新

② 状态管理的作用:

- **集中存储数据** (所有组件都能直接访问)
- **自动同步更新** (任何组件修改数据, 其他用到的地方立即更新)

③ 创建 Store

```

1
2 // stores/counter.js
3 import { defineStore } from 'pinia'
4
5 export const useCounterStore = defineStore('counter', {
6   // 数据仓库的状态 (相当于组件的 data)
7   state: () => ({ count: 0 }), // 这里存放全局共享的数据
8   actions: {
9     increment() {
10       this.count++ // 所有使用 count 的组件都会自动更新
11     }
12   }
13 })

```

④在组件中使用

```

1
2 <script setup>
3 import { useCounterStore } from '@/stores/counter'
4 // "接入"共享数据
5 const counter = useCounterStore()
6 </script>
7
8 <template>
9   <!-- 所有组件使用的都是同一个 counter 实例 -->
10  <button @click="counter.increment">
11    {{ counter.count }}
12  </button>
13 </template>

```

何时该用状态管理?

1. 用户登录信息 (多个组件需要显示用户名)
2. 购物车数据 (不同页面操作同一份购物车)
3. 全局主题设置 (切换暗黑模式影响所有组件)
4. 按钮的 hover 状态 (纯组件内部状态)

7.路由 (Vue Router)

基本配置

```
1 // router/index.js
2 import { createRouter } from 'vue-router'
3
4 const router = createRouter({
5   routes: [
6     { path: '/', component: Home },
7     { path: '/about', component: About }
8   ]
9 })
10
```

JavaScript

在组件中使用

```
1 <template>
2   <router-link to="/">首页</router-link>
3   <router-view></router-view>
4 </template>
```

JavaScript

五、基本的一些语法

1.箭头函数

Vue

```
1 const sayHi()=> alert('Hello')
2 //等价于
3 function sayHi(){alert('Hello')}
```

2.按钮绑定

JavaScript

```
1 const cancelRecommend = async () => { ... }
```

HTML

```
1 <button @click="cancelRecommend">取消推荐</button>
```

Vue 的 `@click="cancelRecommend"` 是把这个变量里的函数作为按钮的点击响应事件。

也就是说，每次点击按钮时，Vue 会自动调用 `cancelRecommend` 这个函数。

这里的 `cancelRecommend` 是一个变量，变量的值是定义的箭头函数。可以理解为事件回调函数。

总结来说，按钮绑定的是一个函数。

3.await

JavaScript

```
1 async function test() {
2   const data = await getData(); // 这里会等2秒
3   console.log(data); // “数据来了”
4   console.log('后面的代码等上一行执行完才会继续');
5 }
6 test();
```

1.await 后面要跟一个 Promise (或者异步函数的返回值)。

2.它会暂停当前 `async` 函数的执行，等 `Promise` 有结果 (`resolve` 或 `reject`) 后，才继续往下执行。

3.`await` 只会暂停当前 `async` 函数“内部的顺序”，不会阻塞整个 JS 执行环境，更不会卡死浏览器。

一个例子如下:

```
▼ JavaScript
1  async function foo() {
2    console.log("开始");
3    await new Promise(resolve => setTimeout(resolve, 2000)); // 等2秒
4    console.log("2秒后");
5  }
6
7  foo();
8  console.log("我不会被阻塞");
```

输出结果如下:

```
▼ Markdown
1  开始
2  我不会被阻塞
3  (2秒后)
4  2秒后
```

4.filter

```
▼ JavaScript
1  const arr = [
2    {id: 3, name: 'a'},
3    {id: 7, name: 'b'},
4    {id: 10, name: 'c'}
5  ];
6
7  const result = arr.filter(item => item.id > 5);
8  // result: [{id: 7, name: 'b'}, {id: 10, name: 'c'}]
```

遍历arr, 对于每一个item, 去返回item.id > 5的结果, filter只保留下为true的item, 这样就对arr进行了一次筛选。

```
1 ReaderState.recommendBooks = ReaderState.recommendBooks.filter(item =>
2     item.novel?.novelId !== currentNovelId &&
3     item.novelId !== currentNovelId
4
```

①对 ReaderState.recommendBooks进行筛选后赋值给原数组。

②filter函数体内对数组的每一个item进行检查，箭头函数传入参数item，返回布尔表达式的结果

③使用了可选链操作符，

前半段意思是,item是否有novel对象，没有就返回undefined与currentNovelId比较，有的话就用它的novelId进行比较

后半段直接用novelId进行比较，是为了兼容不同数据结构

都不相等的才被筛选出来。

C#学习笔记

1.C#的属性访问器

```
1 public DbSet<Reader> Reads { get; set; }
```

①

`DbSet<Reader>` :

EF Core 中表示数据库表的集合，对应 `Reader` 实体类映射的数据库表（如 `READERS` 表）。

`get` : 允许外部代码读取（查询）这个表的实体集合。

```
1 var readers = dbContext.Readers; // 调用get
```

`set` : 允许外部代码替换整个 `DbSet` (极少用，通常用于测试或动态切换表)。

②对比

(1) 传统字段+属性 (等价写法)

```
1 private DbSet<Reader> _readers; // 私有字段
2 public DbSet<Reader> Readers
3 {
4     get { return _readers; }
5     set { _readers = value; }
6 }
```

(2) 自动属性 (简化写法)

```
1 public DbSet<Reader> Readers { get; set; } // 推荐! 更简洁
```