

# 目录

**作者 (Github 账号):** Luyabs & 20201jj

第一章 目的、开发环境、技术框架.....	2
第二章 可行性分析.....	5
第三章 需求分析.....	6
第四章 概要设计.....	8
第五章 详细设计.....	17
第六章 程序模块.....	30
第七章 用户手册.....	33

## 一、目的、开发环境、技术框架

### 1. 项目目的

在在线判题网站(即 Online Judge, 下简称为 OJ)进行编程题的训练是学习计算机技术的人们再熟悉不过的事情, 通过解决 OJ 的编程题, 可以帮助人们更好地理解 and 掌握各种常见算法和数据结构, 从而提升编程能力。同时 OJ 的编程题通常涵盖了各种实际问题, 从字符串处理到图算法, 从排序算法到动态规划等等。通过解决这些问题, 我们可以培养解决实际问题的能力, 提高编程实践经验。当然开发领域的面试中也经常出现与 OJ 的编程题十分类似的问题, 通过解决这些问题, 我们可以为技术面试做好准备, 提高通过面试的机会。

不过当前市场上的 OJ 大多只提供对高级语言的支持, 如 C++、Java、Python。却少有支持 SQL 语句判题的 OJ 平台, 而数据库是现代应用程序的核心组成部分, 无论是企业级应用还是网站、移动应用, 都需要与数据库进行交互。在在线评测平台上加入数据库判题功能可以让学生和开发者更好的掌握面向数据库的开发技术, 因此本项目将在提供对高级语言的支持上加入对 SQL 题的判题支持。

此外, 有许多 OJ 平台不支持普通用户上传题目, 而我们将给予用户这样做的权力, 毕竟开放题目上传功能鼓励平台用户参与到社区创作中来。普通用户可能有各种创新的题目想法, 通过上传题目, 他们可以为平台增加多样性的题目内容, 从而丰富平台的题库。当然我们也会设置题目审核。

### 2. 开发环境

#### 2.1 本地开发环境

操作系统: Windows 10

JDK 版本: 17.0

Nodejs 版本: 16.0

## 2.2 服务器环境

服务器信息: 华为云 ECS 1vCPUs | 2GiB | t6.medium.2

操作系统: Centos 8.2

JDK 版本(服务器): 17.0

JDK 版本(Docker 镜像标签): larrypu/jdk17

Nginx 版本(Docker 镜像标签): latest

## 3. 技术框架

后端开发框架: Spring Boot 2.7

前端开发框架: Vue-admin-template (Vue 2.0)

后端鉴权框架: Sa-token

数据库: MySQL 8.0

数据库连接池: Druid

ORM: MybatisPlus + JdbcTemplate

缓存数据库: Redis

反代理服务器: Nginx

镜像服务: Docker

前端组件库: Element-UI

#### 4. 项目代码

我们的项目已开源在 Github 上：

后端： <https://github.com/Luyabs/online-judge-backend>

前端： <https://github.com/Luyabs/online-judge-frontend>

可以通过此地址直接访问项目： <http://124.70.195.38:10000>

## 二、可行性分析

保密

### 三、需求分析

#### 1. 核心需求

- 1.1 本项目主要分为用户与管理员两个角色，管理员可以做任何用户能做的事，且拥有一些额外特权（主要为审核权与某些功能的独特使用权）。
- 1.2 项目为用户提供登录与注册功能，登录与注册都需要填入账号名与密码，并予以验证。
- 1.3 用户在登录后可以查看自己的个人信息，个人信息包括用户名、昵称、密码、个人简介等。并且用户能在某些时刻，如排行榜等情况获取其他用户的非隐私信息，如用户昵称、简介等。
- 1.4 用户可以在登录后，通过题库页面查看题目的详细信息，并选择其中一题进行作答，用户可以通过调试或提交的方式上传作答信息，提交记录经后端判断后记录答题结果（包含是否成功，运行时间等）。
- 1.5 用户可以上传新题目，也可以获取自己已上传的题目，修改或删除已上传的题目。但除了查询操作外，上传、修改与删除需要经管理员审核后才生效，在生效前此题将不可见，直至增加、修改、删除成功或者放弃修改才可见。在用户上传题目时需要提交题目标题、题目内容、测试用例（SQL 题还需提交建表语句与删表语句作为特殊的测试用例）。
- 1.6 管理员可以对提交的审核记录进行审核，在审核通过或不通过之余需要提交审批理由予以用户提示，一旦审核通过后题目将变得可见，可以被任意用户完成。
- 1.7 管理员可以直接增加、修改、删除任意题目，不需要经过审核，但仍会留下修改记录。
- 1.8 管理员可以封禁或解封用户，被封禁的用户将无法再登录。

1.9 用户可以查询自己已完成的统计信息。统计信息包括答题的历史记录（包括正确结果），以及答题的统计信息。

## **2. 未来进一步完善的需求**

2.1 用户可以提交对题目的解答，也可以获取任意题目的题解，也能管理（删改查）自己发布的题解。

2.2 管理员可以置顶题解或删除任意题解。

2.3 用户可以查询由任意题目组成习题集。

2.4 管理员可以发布习题集，并对习题集及习题集的组成做增删改查。

2.5 针对题目用户完成情况与题目偏好推荐练习题。

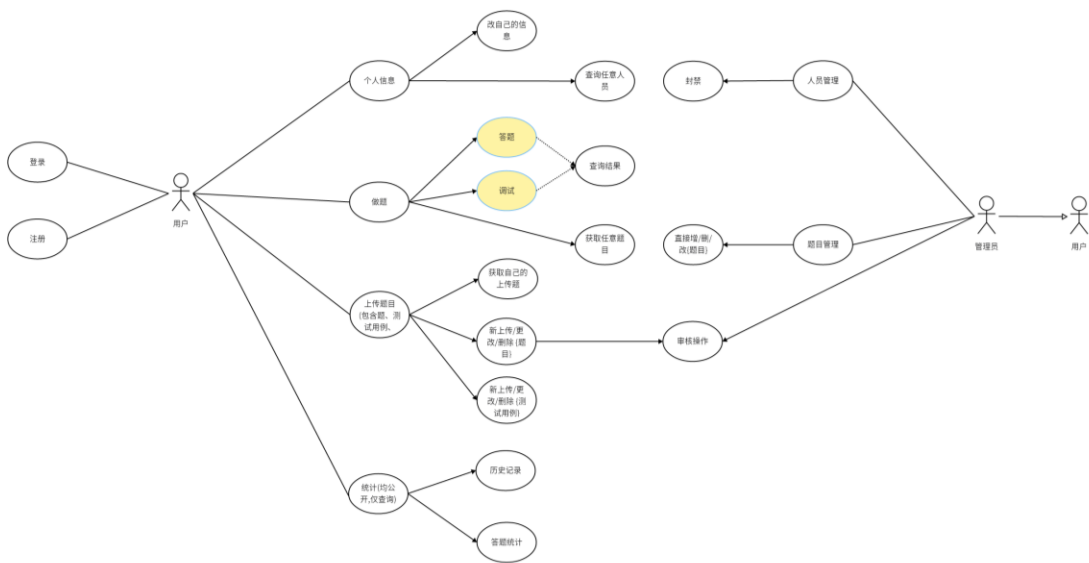
2.6 用户可以向管理员反馈系统问题。

# 四、概要设计

## 1. 用例设计

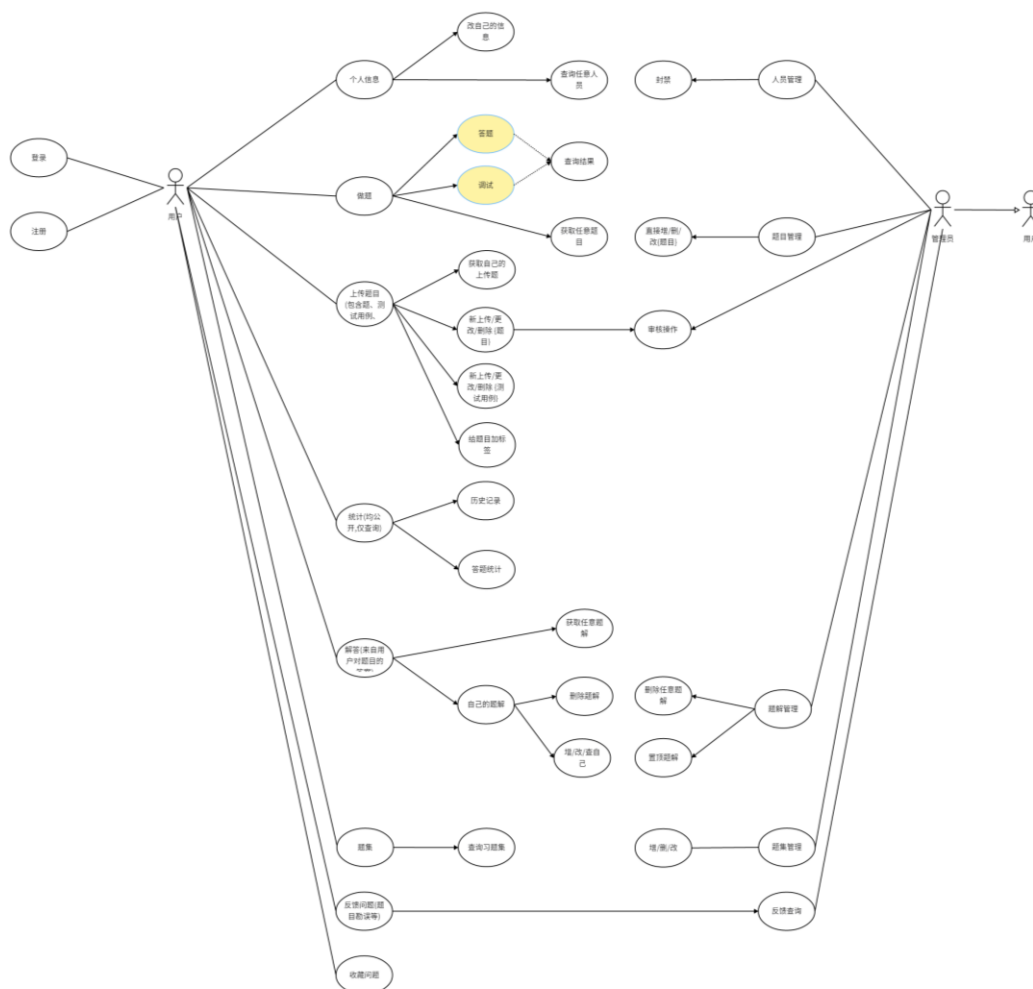
本项目的用例图完全基于需求分析得到，因此省去了冗余的用例描述，可以直观的从图中发掘本项目的功能。我们的开发过程也严格按照此用例图进行。

本项目当前已全部实现的基础用例图如下：



本项目含扩展需求的完整用例图如下：

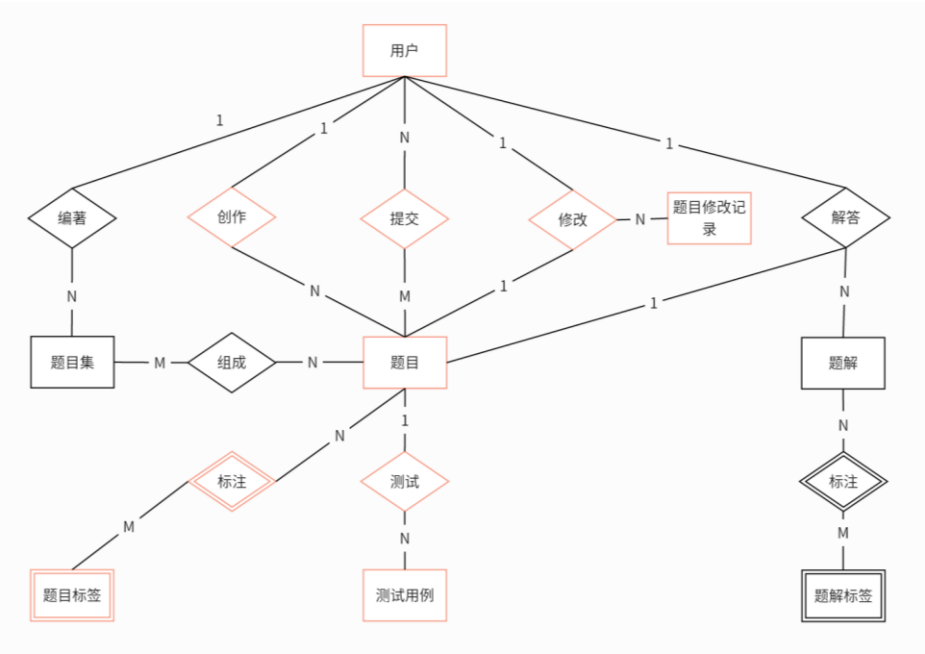




## 2. 数据库设计

本项目的数据库设计，从开发速率与后期维护扩展这两个角度上考虑，平衡了范式的规范性和反范式的易开发性，设计了如下 E-R 图所示的数据库设计，其中红色部分是当前已建成的数据表，黑色部分是对应扩展功能、可能在未来建立的数据表。

由于我们采用 MVC 设计模式开发本项目，因此这张自底向上的 E-R 设计可以帮助我们省略后续绝大部分 CRUD 功能模块的详细设计，以减小我们开发人员的负担画太多图的负担。这点将在后续我们遵循的 MVC 设计模式中进一步阐述。



基于上述概念设计，我们完成了以下完整的数据库逻辑设计（不含扩展

部分）：

	关系名	字段含义	字段名	字段类型	字段约束	后端追加约束
1	用户 user	用户 id	user_id	bigint	主键	自动生成字段 基于雪花算法
		权限	role	integer	非空	枚举关系 normal_user=1,
		账户名	username	varchar	唯一	
		用户登录密码	password	varchar	非空	需检验密码规则
		昵称	nickname	nickname	非空	
		个人简介	introduction	varchar		
		是否被封禁	is_banned	boolean	默认值 false	
		用户信息更新时间	update_time	datetime		自动填充字段
		用户加入时间	insert_time	datetime		自动填充字段
2	题目 problem	题目 id	problem_id	bigint	主键	自动生成字段 基于雪花算法
		作者 id	user_id	bigint	外键 [on delete set null]	
		题目标题	title	varchar	非空	
		题目内容	content	varchar	非空	

		题目类型	type	integer	非空	枚举关系 sql=1, program=
		总尝试次数	attempt_num	integer	默认值 0	新增“提交”记录时需+1
		总通过次数	success_num	integer	默认值 0	新增“提交”记录时视成功状
		难度	difficulty	integer	非空	枚举关系 easy=1, medium=hard=3
		运行时间限制	runtime_limit	double		
		运行内存限制	memory_limit	double		
		状态	status	integer	默认值 0	枚举关系 未审核 = 0, 审核1, 审核中 = 2, 审核未通过历史 = 4
		题目信息更新时间	update_time	datetime		自动填充字段
		题目新增时间	insert_time	datetime		自动填充字段
3	测试用例 test_case	测试用例 id	test_case_id	bigint	主键	自动生成字段 基于雪花算法
		题目 id	problem_id	bigint	外键 [on delete cascade]	
		测试语句	input	varchar		
		预计输出	output	varchar		
		测试描述	description	varchar		
		是否为预处理语句	is_prehandle	boolean	默认值 false	多个用例此项为 true 时按顺序
		是否为后置处理语句	is_posthandle	boolean	默认值 false	多个用例此项为 true 时按顺序
		测试用例顺序	t_order	integer	默认值 0	提供用例执行优先级，相同按 updater_time 降序执行
		<del>是否通过审核</del>	<del>is_verified</del>	<del>boolean</del>	<del>默认值 false</del>	<del>增减“修改记录”时需调整+1后调整</del>
		<del>审核附加信息</del>	<del>verify_message</del>	<del>varchar</del>		<del>此项只有管理员审批时才能</del>
		测试用例更新时间	update_time	datetime		自动填充字段
		测试用例新增时间	insert_time	datetime		自动填充字段

4	修改记录 edit_record	修改记录 id	edit_record_id	bigint	主键	自动生成字段 基于雪花算法
		修改者 id	user_id	bigint	外键 [on delete set null]	
		原始题目 id	original_problem_id	bigint	外键 [on delete cascade]	
		修改行为	change_action	integer	非空	枚举关系 insert(新增)=1, delete(删除)=2, update(更新)
		修改的临时题目 id	edit_problem_id	bigint	外键 [on delete set null]	is_test_case=false & change_type=3 时有效 其余为 null)
		是否由管理员修改	is_admin	boolean	非空	此项只有管理员直接增删改为 true
		修改状态	status	integer	非空	枚举关系 wait=1(审批中), verified=2(通过), failed=3(失败)
		审核附加信息	verify_message	varchar		此项只有管理员审批时才能
		记录更新时间	update_time	datetime		自动填充字段
		记录新增时间	insert_time	datetime		自动填充字段
5	提交记录 submission	提交记录 id	submission_id	bigint	主键	自动生成字段 基于雪花算法
		提交用户 id	user_id	bigint	外键 [on delete set null]	
		题目 id	problem_id	bigint	外键 [on delete cascade]	
		选择语言	language	integer	非空	枚举关系 MYSQL=1, JAVA=2
		提交代码	code	varchar	非空	
		代码运行结果	code_result	varchar		
		是否是调试	is_debug	boolean	非空	
		是否运行成功	is_success	integer		只有判题时才修改此字段
		错误类型	error_type	varchar		is_success=false 时有效 (视为 null) 只有判题时才修改此字段
		通过的测试用例数	pass_test_case_num	integer		只有判题时才修改此字段

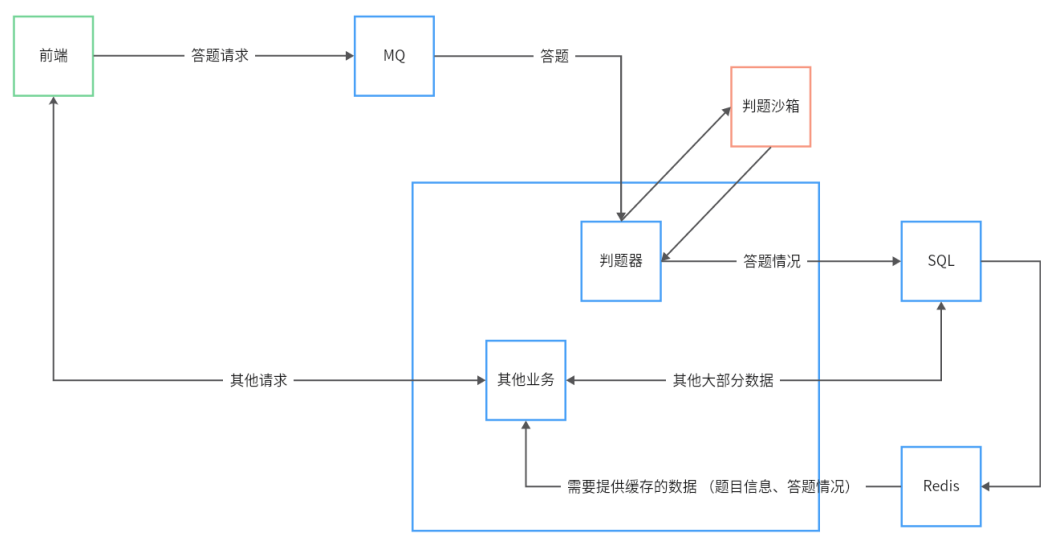
		代码执行时间	runtime	double		只有判题时才修改此字段
		代码消耗内存	memory	double		只有判题时才修改此字段
		提交记录更新时间	update_time	datetime		自动填充字段
		提交记录新增时间	insert_time	datetime		自动填充字段
6	标签库 tag	标签 id	tag_id	bigint	主键	自动生成字段 基于雪花算法
	目前无需为此表提供 CRUD	标签类型	type	varchar		
		标签名	name	varchar	非空	
		标签更新时间	update_time	datetime		自动填充字段
		标签新增时间	insert_time	datetime		自动填充字段
7	题目标签关系 problem_tag	此关系的 id	problem_tag_id	bigint	主键	自动生成字段 基于雪花算法
		题目 id	problem_id	bigint	外键 [on delete cascade]	
		标签 id	tag_id	bigint	外键 [on delete cascade]	
		题目标签关系更新时间	update_time	datetime		自动填充字段
		题目标签关系新增时间	insert_time	datetime		自动填充字段

### 3. 整体功能模块设计

下图为我们项目的整体模块设计。用户在前端答题后，会把题目提交给后端，并在后端的判题沙箱内处理题目，最后将数据记录在数据库中，并同时把答题结果返回给用户。除答题以外的其他用户获取资源请求或更新资源请求将进行 Redis -> SQL 的常规处理。

在上述环节里，考虑到项目实现的复杂度，我们暂缓使用消息队列，同时也

将判题沙箱内置在后端的 Java 代码中。其他部分均如下图模块展示的一致。



#### 4. 前端路径设计

基于用例设计，前端共设计了如下 URL。其中 `constantRoute` 数组存放着所有用户都可访问的 URL，`asyncRoutes` 存放仅开放给管理员访问的 URL。其中 `path` 即为 URL，`children` 即为上一个 `path` 的子 URL。

```
constantRoutes = [  
  path: '/login'  
  path: '/register'  
  path: '/404'  
  
  path: '/'  
  children:  
    path: 'dashboard'  
  
  path: '/problem_set'  
  children:  
    path: 'all'  
    path: 'problem'  
  
  path: '/my_upload'  
  children:  
    path: ''  
]
```

```
        path: '/test_case'

    path: '/statistic'
    children:
        path: ''
        path: 'history'
]

asyncRoutes = [
    path: '/admin'
    children:
        path: 'dashboard'
        path: 'judgement'
        path: 'management/user'
        path: 'management/problem'

    path: '*', redirect: '/404'
]
```

## 5. 项目总体开发设计模式

项目整体采用 MVC 设计模式，具体为前后端分离，后端提供 Model + Controller，前端提供 Model-View 双向绑定的设计模式。

后端抽象上自底向上分为数据接入层 (mapper)，业务处理层 (service)，控制器层 (controller)。

其中 mapper 负责向下通过 jdbc 执行 SQL 语句, 并将返回的数据封装在实体 (entity, 即仅含单表, 且含单表所有数据的类) 中, 并向上层传递实体或数据转换对象 (dto, 即含多表数据的类)。

Service 负责解析复杂的业务逻辑, 如字段检测、异常处理、按顺序处理数据表、与其他中间件连接、向外部平台发请求等。其会向下调用与数据库访问相关的原子操作, 并将其返回的 entity 或 dto 处理后以业务对象 (bo, 即去除了冗余字段后的 entity 与 dto) 传递给上层。

controller 负责为上述设计的 URL 路径, 提供一对一的响应来自客户浏览器发出的请求。并调用下层的业务处理, 最终以带 JSON 数据回复的形式结束这次 HTTP 访问。

这三层大大降低了 CRUD 的耦合度, 因此任何属性 MVC 的开发都能通过数据库设计与用例设计轻松的去一个接一个的实现 CRUD 用例, 并且不再需要额外的详细设计 (非 CRUD 功能仍需要不一样的详细设计)。

前端抽象上分为用户视图, 全局存储, 路由解析, 请求响应四大模块。

用户视图模块直观的提供可操作的界面, 主要还是用 vue + css + html; 全局存储提供对部分值在一整个 session 中的长期存储, 由 vue-x 实现; 路由解析使用的是 vue-element-admin 封装过的 vue-router; 请求响应则使用封装的 axios。以上四块共同支撑起整个前端。



# 五、详细设计

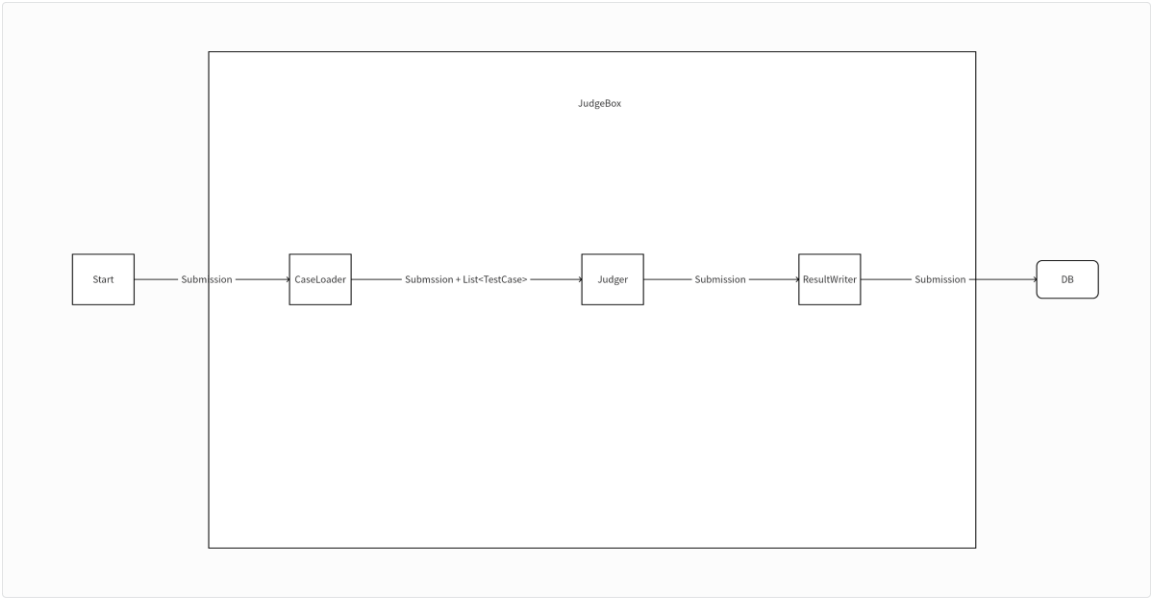
通常，一般的 CRUD 操作并不需要过于详细的设计，只需参照用例设计和数据库设计即可顺利完成。

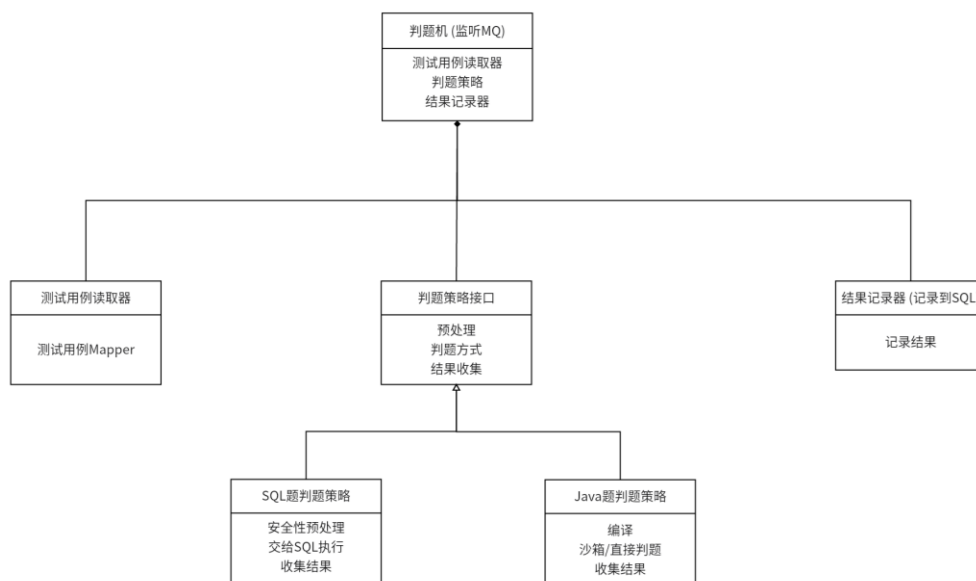
同时，前端也只需要美观和能用即可，因此也不在此处给予详细设计。

## 1. 判题机详细设计

### 1.1 判题机构成

判题机是整体结构如下图所示，当后端接收到答题请求后会调用判题机进行处理，这个处理过程比较线性：首先会根据题目 id 读取题目对应的测试用例并调用对应语言的判题器，接着通关判题器对题目做判断，最后记录数据到数据库中，同时也将结果返回给前端。在此过程中判题机也会照常执行异常处理，并根据异常选择是否要将结果记录到数据库中。





## 1.2 判题机类图

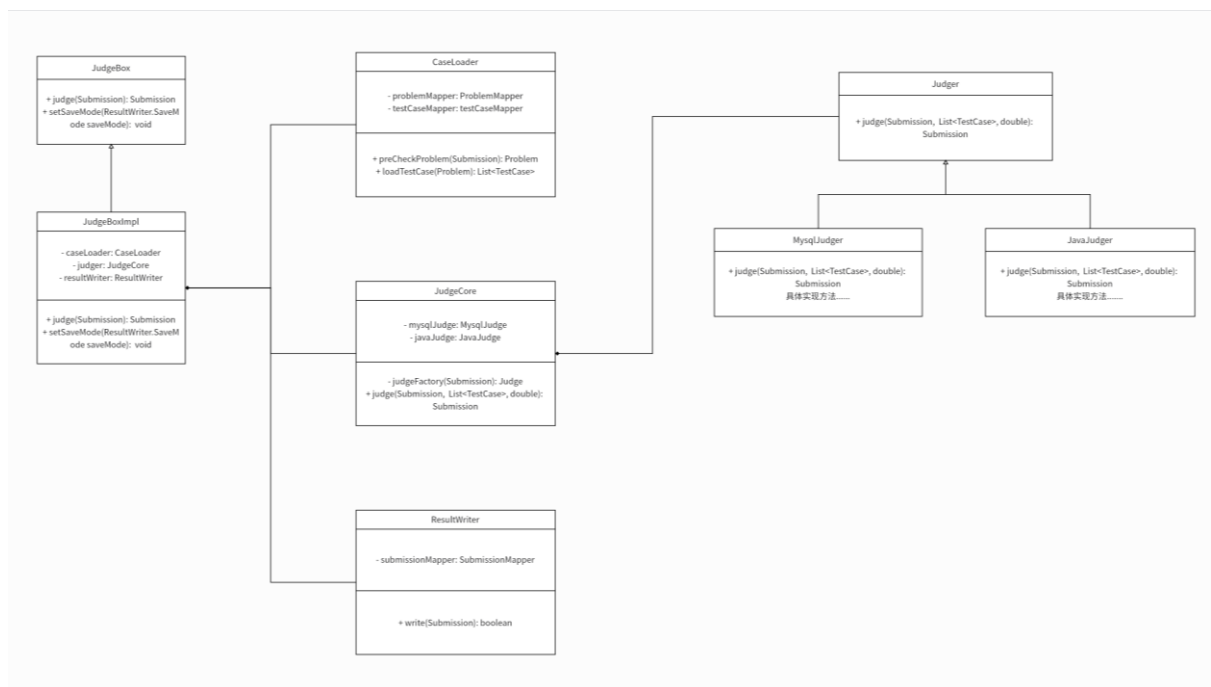
为了体现封装特性，采用判题外观接口 `JudgeBox` 与其实现类 `JudgeBoxImpl` 对整个判题机进行封装，以实现只要简单调用 `judgeBox.judge(submission)` 即可执行判题 (`submission` 是用户提交的解答类，`judgeBox` 是一个 `JudgeBox` 类型的对象)。

这个外观类由测试用例读取器类 (`CaseLoader`)、核心判题器 (`JudgeCore`)、共同组成、结果记录器 (`ResultWirter`) 共同组成。

其中核心判题器采用策略模式，会根据用户提交的解答对应的题目类型选择合适的判题机 (Java 与 MySQL 判题机二选一) 对答案进行批改 (执行一遍代码)。

整个判题机类位于如下 package 中：

`src/main/java/com/example/onlinejudge/judgebox`



### 1.3 测试用例读取器

测试用例读取器类由两个方法组成：

一个是预检方法 `preCheckProblem`，以避免用户提交了错误格式的解答或题目本身出现一定问题时中断此次判题。

另一个是读取测试用例方法 `loadTestCase`，会根据用户提交的解答中的问题 `id`，读取对应问题的测试用例，并将测试用例根据 [是否是前置测试用例 (用于 SQL 题建表插入数据) > 测试用例优先级 > 是否是后置测试用例 (用户 SQL 题删表)] 顺序对测试用例进行排序。同时如果测试用例出现问题 (如不满足一定需求) 也将中断此次判题。

代码位于：

`src/main/java/com/example/onlinejudge/judgebox/core/CaseLoader.java`

### 1.4 判题器

判题器分为 Java 判题器与 MySQL 判题器，分别基于不同的思路进行判题。

#### 1.4.1 Java 判题器

Java 判题器的核心思路是先预处理并保存用户代码，再进行编译代码，最后运行代码并删除所有这个过程中产生的文件。

预处理实际是让用户代码拆散重组的过程。在预处理中首先会检查用户是否引入包，我们明确告知用户不允许引入任何包，取而代之我们帮他们引入，通过字符串检索和拼接实现。其次用正则表达式将用户提交的唯一公共类的类名更换为一个单字母+不会发生重复的 `submission_id`，这样做可以保证因多个用户提交的文件中类名相同而造成并发问题。最后我们引入 `SecurityManager` 安全沙箱，限制 JVM 使用所有系统调用，以确保服务器的绝对安全。最后我们将其保存为文件，文件名与公共类名相同。

编译是件简单事，直接执行指定的 `bash/batch` 命令就行。为了防止操作系统不同带来的问题，我们引入 `apache` 的 `CommandLine` 与相关类来代替 `runtime.exec()` 执行 `bash` 命令。

最后是运行代码，运行代码同样使用了 `CommandLine`，同时也会设置 `WatchDog` 以防止因无限循环带来的超时等问题，而对于内存占用问题，时间限制+禁止系统调用（如禁止创建多线程）+限制了用户提交的代码长度已经很好的解决了代码执行时内存过大的问题，为服务器确保了充分的安全性。

JavaJudge 代码位于

```
src/main/java/com/example/onlinejudge/judgebox/core/judger/JavaJudge.
```

```
java
```

### 1.4.2 MySQL 判题器

对于 MySQL 判题器而言，安全性问题比较简单。只需要几步：①开一个新的数据库(database) ②限制数据库访问者的权限(只允许增删改查数据与建表删表) ③将上述数据作为新的数据源添加到后端(取名为 secondary 数据源)。

具体的 MySQL 判题器的判题原理很直白，就是在数据库中依次执行一遍测试用例与用户提交的代码。如测试用例 1 为 insert...，测试用例 2 为 delete...，用户提交为 select...，就通过 jdbcTemplate 按顺序执行测试用例 1，用户代码，测试用例 2，用户代码。这个过程还是较为简单的。其中我们在 src/main/java/com/example/onlinejudge/common/JdbcTemplateBean.java 中对这种远程执行代码做了封装。

但由于测试用例可能是建表删表语句，增删改或者查询，同时用户提交的代码可能是增删改或查询(我们同样通过代码限制了用户可以执行的操作，同时用户也只能提交一句 SQL)。这是个排列组合问题，因此要检验测试用例是否通过就需要一套独特的规则，以下为我们采用的规则：

输入(INPUT)	预计输出(OUTPUT)	用户代码与 OUTPUT 比较原则
DML DDL	/	不比较
DML DDL	DQL (行得通的 SELECT 语句)	都分别执行，比较 List<List<Object>>是否相等
DML DDL	修改的行数	比较执行完用户代码后修改行数

我们不对测试用例的 input 字段做区分，而对于测试用例的 output 字段做分类讨论。Output 一共分为空、select 语句与一个数值。

其中如果 output 为空，那么无论用户输入什么我们都不比较，这个测试用

例也不记入“通过的用例数”。这通常在测试用例为表增删改数据或建表删表时使用。

如果 output 为 select 语句，那么意味着这是一道考察用户 DQL 水平的题，我们会通过 jdbcTemplate 分别执行 output 与用户提交代码中的语句，并将结果分别存放在二维变长数组 (List<List<Object>>类型) 中。只需要比较这两个数组的值与大小是否相同，即能确定是否通过测试用例。

如果 output 为单个整型数字，即为修改的行数，意味着这是一道考察用户 DML (增删改) 的题，只需要执行用户代码最后统计影响的行数是否正确即可得到结果。（即便有时候会因错误的提交产生正确的答案，但随着测试用例 (增删改表数据) 的增加，蒙对所有答案的概率就趋向 0；同时对于不影响大局的小错误 (比如题目要求 update...set...= 'abc' where...，而用户提交了 update...set...= 'def' where...) 我们也不予深究）。

当然，我们也限制了一句 SQL 的执行时间，防止出现 while 死循环，具体办法是在正式进行用例测试前先做一次测速，以防超时。其中测速通过双线程实现，其中主线程陷入一段时间的 sleep，子线程执行用户提交的 SQL。当主线程按时醒来时意味着 SQL 执行超时，中断子线程；当子线程结束时会提前唤醒主线程，并返回执行时间。

值得提一句，MySQL 判题器不支持并发，因为并发中有可能同时有两个用户提交的数据对一张表做修改，于是我们通过 synchronized 保留字为 MySQLJudge 加了独占锁。这也是未来可以优化之处。

MySQLJudge 代码位于  
src/main/java/com/example/onlinejudge/judgebox/core/judger/MysqlJudge

. java

### 1.4.3 安全性分析

以下为关于安全性的总结分析：

对于以 Java 为代表的高级语言，我们通过设置包/头文件的白名单，引入安全沙箱限制系统调用（在 Java 中有控制系统调用的 SecurityBox），限制代码执行时间。并通过调整类名来实现并发。

对于以 MySQL 为代表的数据库查询语言，我们通过开创独立数据库，限制用户访问数据库权限，限制用户输入的语句，限制代码执行时间。并通过独占锁来避免并发带来的问题。

### 1.4.4 其他可能加入的判题器

事实上编程题绝大部分就分为高级语言与数据库语言两大类，（当然也可能有机器学习题等）。加入其他判题机的操作与我们实现的两个判题机无大差异（尤其是数据库判题机）。

不过仍要注意判题器的安全性，安全性是判题器的基础。这也是决定实现判题器难易程度和可用性的关键，考虑到语言的复杂性，有些语言会在制造安全沙箱上尤为困难，特别是 C++，需要写新程序并用复杂的类来限制系统调用或是拼装代码，这需要大量的 Linux 与 C++ 知识。而有些语言如 Python 又在这方面上较为简单。

当然由于我们将判题器采用了策略模式，因此扩展其他的判题器是件容易的事，除新的判题器外只用修改数行原来的代码即可。

## 1.5 结果记录器

结果记录器只提供一个方法和若干配置参数。会根据参数的不同选择在控制台输出判题结果 / 在数据库中保存判题结果 / 以文件形式保存判题结果。并向上返回判题结果。

代码位于：

`src/main/java/com/example/onlinejudge/judgebox/core/ResultWriter.java`

## 2. 面向切面编程、全局字段校验与全局异常处理

我们使用面向切面编程（AOP）以尽可能的避免高重复度的代码在我们的项目中出现。AOP 即采用代理模式这一设计模式的编程思想，利用注解将高重复度的代码与业务逻辑代码相分离，最后在编译时交给编译器将代码组装到一起。

我们主要在以下两方面通过写注解与解析器实现 AOP：

① 自定义注解 `@Log` 与注解解析器 `LogAspect` 类实现日志打印。以此记录某用户  
在某时刻执行了某一个 service 层的方法，携带了什么参数

② 自定义需要身份验证注解 `@Authority` 与解析器 `AuthorityAspect` 类，身份验证不同于权限验证（或鉴权，下方会进一步提到我们项目的鉴权模式）。用于校验一个方法是否需要管理员权限 (admin) 或验证是否拥有对某物的拥有权 (owner)。其中身份验证较为容易，通过解析用户 token 查 user 表便知。而拥有权需要对每一个类单独实现，我们在

`src/main/java/com/example/onlinejudge/common/authentication` 包内设计了工厂加策略模式的拥有权校验方法，具体思路为分析类名 (记为 A) 与方法的首



个形参(arg0, 统称为一个实体类或 id), 通过反射得到对应 A-mapper 的方法(多表查询表 A 及相关表) 以验证该用户是否拥有对 arg0 (或其映射对象) 的拥有权。

③ 自定义注解@AvoidRepeatableCommit 防止用户重复提交产生相同数据, 由于网络波动导致的数据包重发或者用户连续点击提交可能造成重复提交, 用户连续多次提交解题代码也会占用更多不必要浪费的服务器资源。我们通过 redis 分布式锁对用户访问某接口的行为加以限制(分布式锁), lua 脚本实现原子性, 将@AvoidRepeatableCommit (timeout = \*\*) 注解添加在用户提交题目, 提交题目解答和测试用例等不具有幂等性的接口前, 提高了系统的鲁棒性。

上述代码均位于 src/main/java/com/example/onlinejudge/common/aop

我们主要在以下两方面通过调包使用 AOP:

① 全局异常处理, 我们将所有 controller 层以下抛掷的异常都通过 throw, 返回给了被@ControllerAdvice 标注的全局异常处理类, 以进行统一的异常处理, 并返回给前端。具体分为以下六类: 自定义业务异常处理器、文件异常处理器、MySQL 异常处理器、Token 异常处理器、校验异常处理器与其他全局异常处理器, 作用一致, 只是按处理的异常做了划分。

② 全局字段校验@Validate, 全局字段校验提供了对目标属性的快速检验如限制非空、限制最小值等, 以此来简化代码的重复度。

调用包为 spring-boot-starter-validation

### 3. 应用 redis 中间件实现系统的高并发

#### 3.1 redis 做数据缓存

为了提高项目并发能力，我们将用户最常访问的数据——题目信息，保存在 redis 缓存中。在配置类里面使用 @Bean 标注在方法上给 IoC 容器注册 cacheManager 组件后，可以在业务层前添加 @Cacheable 注解实现项目和 redis 非关系型数据库的结合。

配置文件在：

```
src/main/java/com/example/onlinejudge/config/RedisConfig.java
```

#### 3.2 redis 实现分布式锁

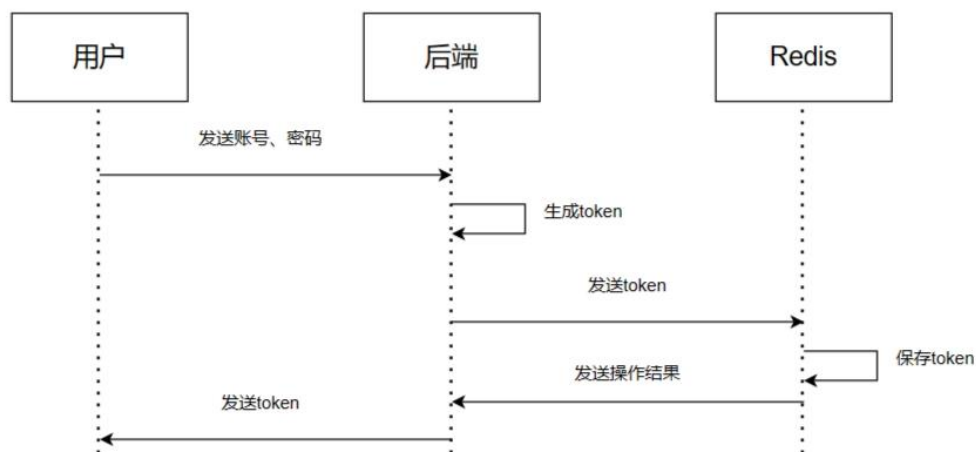
由于在多用户并发操作是，对同一资源的修改会使数据成为脏数据或者用户得到错误的返回信息，比如多个管理员同时审核同一个题目，提交审核结果信息时，或者用户和管理员同时对某一题目做修改，那么有可能会对数据做错误的修改。所以，我们对可能出现并发问题的数据上了锁，因为分布式锁要确保上锁时动作的原子性，所以我们使用了 redisson 组件，业务层的具体实现逻辑为：

业务开始前上锁（在 redis 中添加 key-value：“lock\_editRecord::” + [editRecordId] - [UUID]），然后正常执行业务，最后在 finally 块中删除锁，这样可以很好的保证上锁动作的并发性，以及确保不会出现死锁现象，保证在同一时间只能有同一个进程访问某资源。

### 3.3 redis + token 实现单点登录 + 登录鉴权 + 用户一段时间不操作即下线 + 管理员即时踢人下线

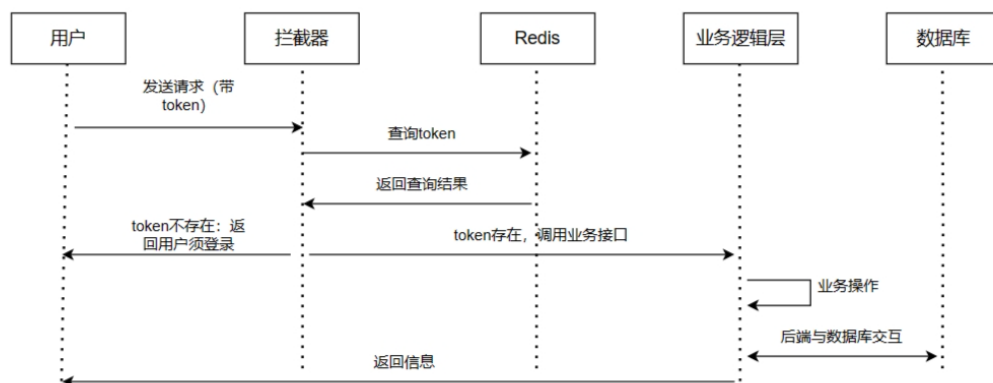
首先，为了确保用户在同一时间只能在一台设备上登录使用平台功能，我们用 redis 存储了用户的 token 信息，当用户在已经登录后，使用第二个页面\终端登录后，系统会将第一次登录时生成的 token 信息从 redis 中删除，同时将第二次登录时新产生的 token 存入 redis。

用户登录：



其次，我们在拦截器中注册有 LoginInterceptor 登录拦截器，拦截器会拦截所有需要登录才能访问的功能，从请求的 header 头中取出 token 查看是否在 redis 中，如不在，则拒绝提供服务（系统认为用户已下线），如果存在，说明用户目前处于在线状态，可以正常提供服务，并同时刷新 redis 中 token 的过期时间。

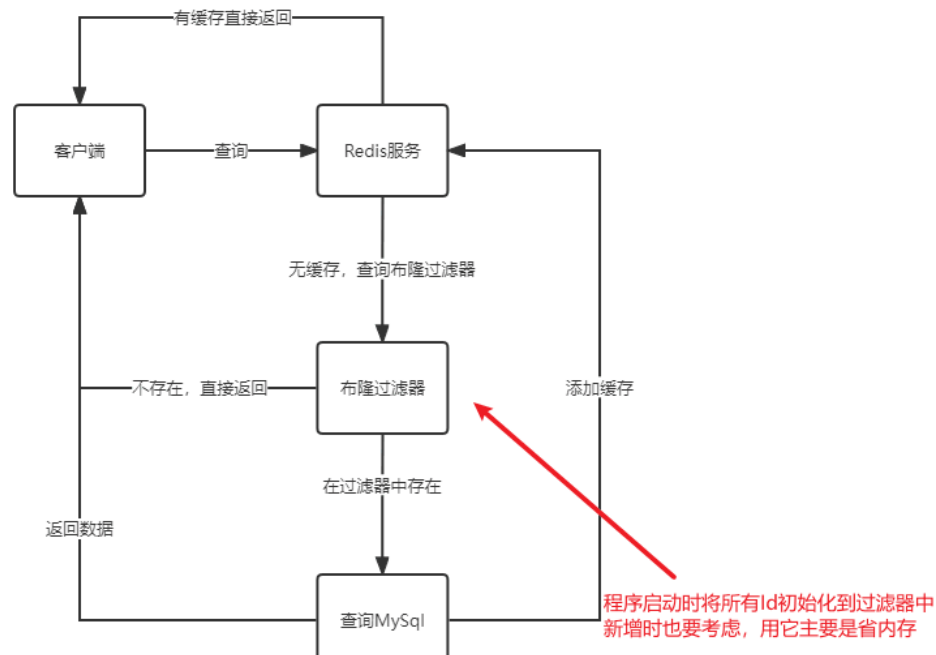
操作登录后才能使用的功能:



基于如上逻辑，我们限制了一个账户同时只能在同一点保持登录状态，重复的登录会使前一次登录失效；使用过滤器拦截所有请求，确保访问限定接口者都是已登录人员；并且拦截器每一次拦截到某账户的请求就会刷新其 token 在 redis 中的过期时间，如果用户长时间不操作就会强制下线；同理，管理员也可以直接删除违规用户的 token, 立刻限制用户的登录行为。

### 3.4 添加布隆过滤器提高系统安全性

布隆过滤器的存在是为了避免缓存穿透的问题，提高项目的安全性。背景如下：当攻击者大量发送访问不存在题目的请求时，由于数据库并无此资源，redis 中自然也没有这个资源，所有大量的请求查询 redis 失败后都会打到数据库上，这会对数据库造成较大的压力，我们在查询 redis 和查询数据库之间添加一个布隆过滤器，这样过滤器可以拦截访问肯定不存在数据的请求。

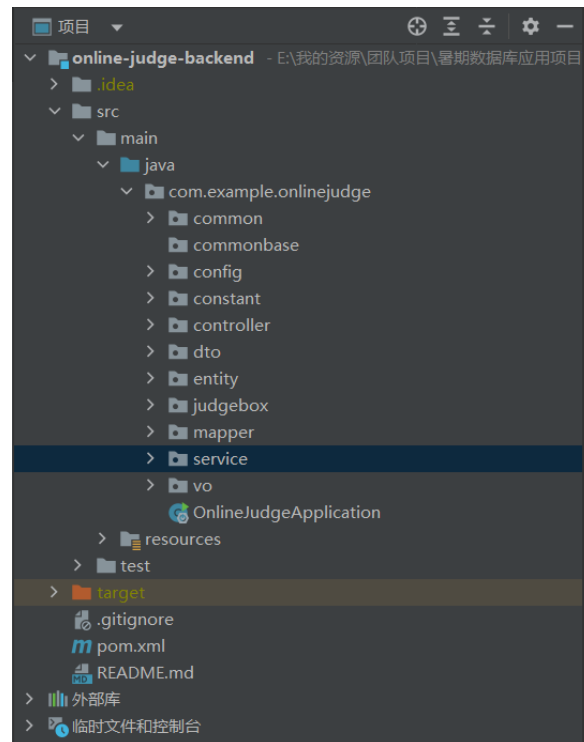


布隆过滤器底层是采用多个无偏 hash 函数+大型的 byte[] 二进制数组来实现的，当向过滤器中添加元素是，会对元素求哈希并将其映射到二进制数组上；当过滤器要判断某元素是否在过滤器中时，只需要对其求 hash，然后查看二进制对应的位是否都为 1，如果为是，则判断在过滤器中。

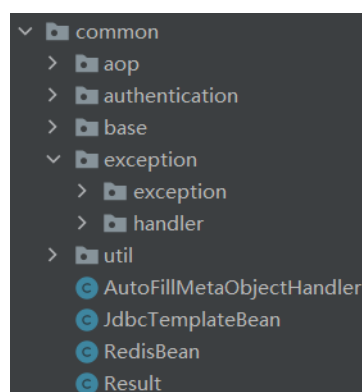
对于避免缓存穿透的问题，其实还有另一种思路，也就是在 redis 存储 key 为资源号，value 为 null 的数据即可，但这样会占据很多的 redis 内存空间，可能会导致命中率下降，最后我们还是选择了布隆过滤器。

## 六、程序模块

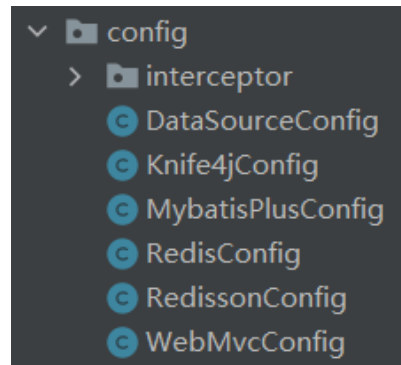
项目文件目录结构大致如下：



1. common：主要存放了切面、注解、鉴权、异常类型及其处理和一些自定义的工具类；



2. config: 设置了拦截器和配置类



3. constant: 业务字典

controller: 控制层

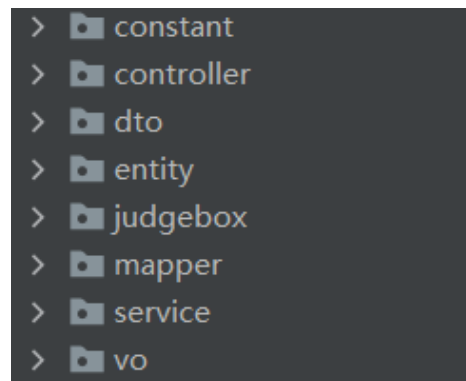
service: 业务层

mapper: 数据库层

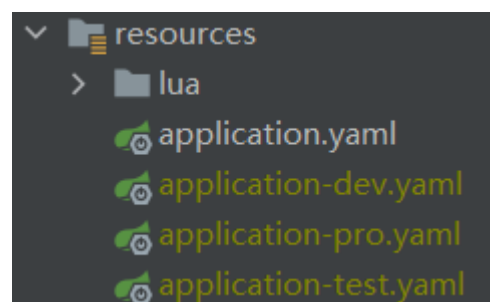
entity: 实体类 - 和表字段一一对应

vo、dto: 数据传输对象

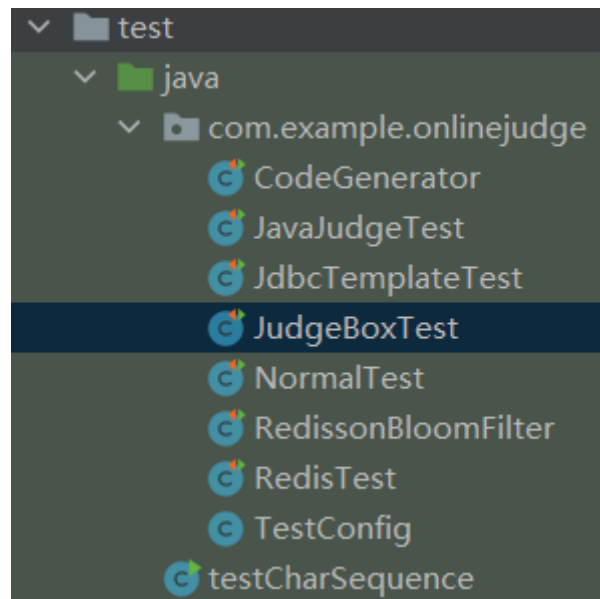
judgebox: 判题沙箱



4. resources: 存放 lua 脚本和 yaml 配置文件



## 5. test: 单元测试和模块间联合测试






## 八、用户手册


### 1. 普通用户使用方式


① 登录 <http://124.70.195.38:10000> (或你部署的前端地址)

② 输入你的用户名与密码，点击蓝色的 Login 登录。如果你还没有账号，请点击红色的 Register 注册一个，我们会以 MD5 方式存储你的密码摘要，任何人将无法从数据库中查看或破解你的密码。

### Login Form

 admin

 .....





LoginRegister


username: admin / u1    password: 123456


密码传输时已采用非对称加密


### Register Form


 Username


 Password



 PasswordAgain

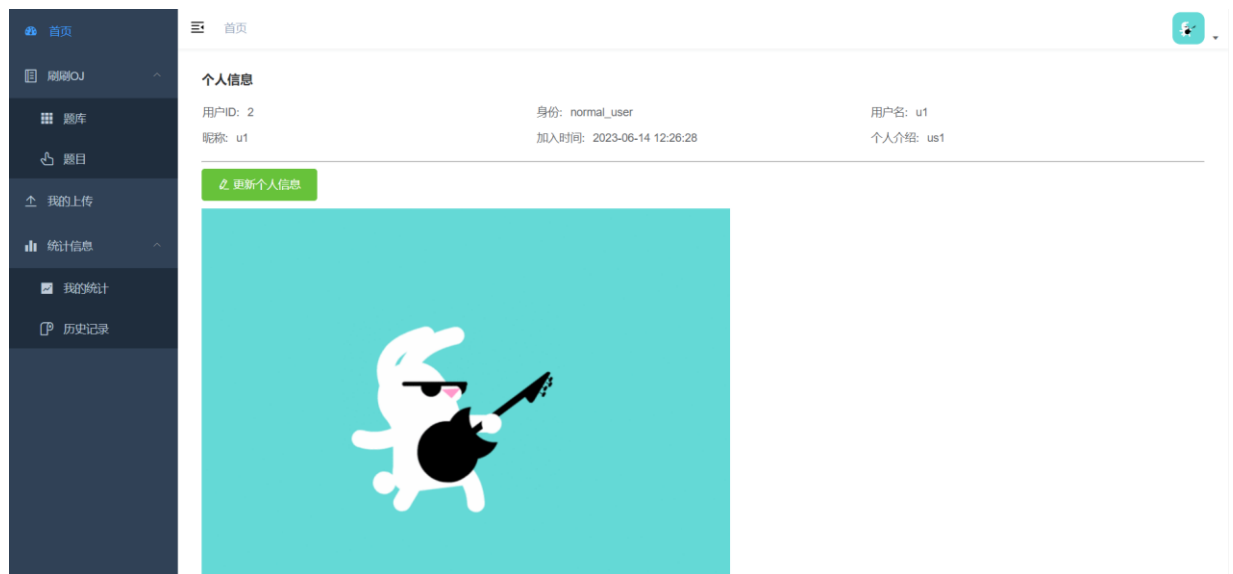


 Nickname

 Introduction

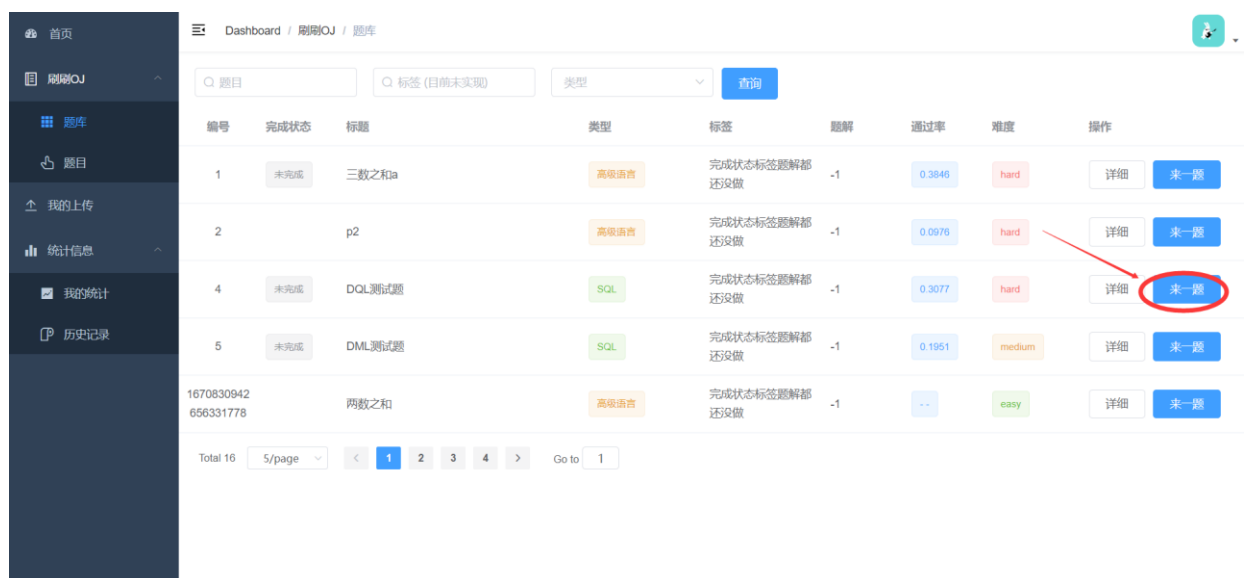
Go BackRegister And Login

③ 登录成功后进入主界面。



④ 接着你可以点击左侧侧边栏，页面顶部面包屑，右上角个人信息栏与页面中央的各种按钮进行操作。

你可以点开题库，选择蓝色的来一题按钮做题：



在题目界面内输入答案并点击调试或提交按钮查看结果。（调试只会执行第一条测试用例，且不计入用户做题结果；测试会执行所有测试用例，并记录结果）

Dashboard / 刷题OJ / 题目

垂直带边框列表

编号	标题	类型	难度
4	DQL测试题	SQL	hard

内容

请查询 test 表中所有字段与记录

注意事项

SQL题只能提交一条语句

MySQL

测试 提交

select \* from test

输入代码以完成题目

提交后查看运行结果

你也可以在我的上传中提交 新题目或管理你已提交的题目及其测试用例。  
(若非管理员，则每次题目的变更需要审核)

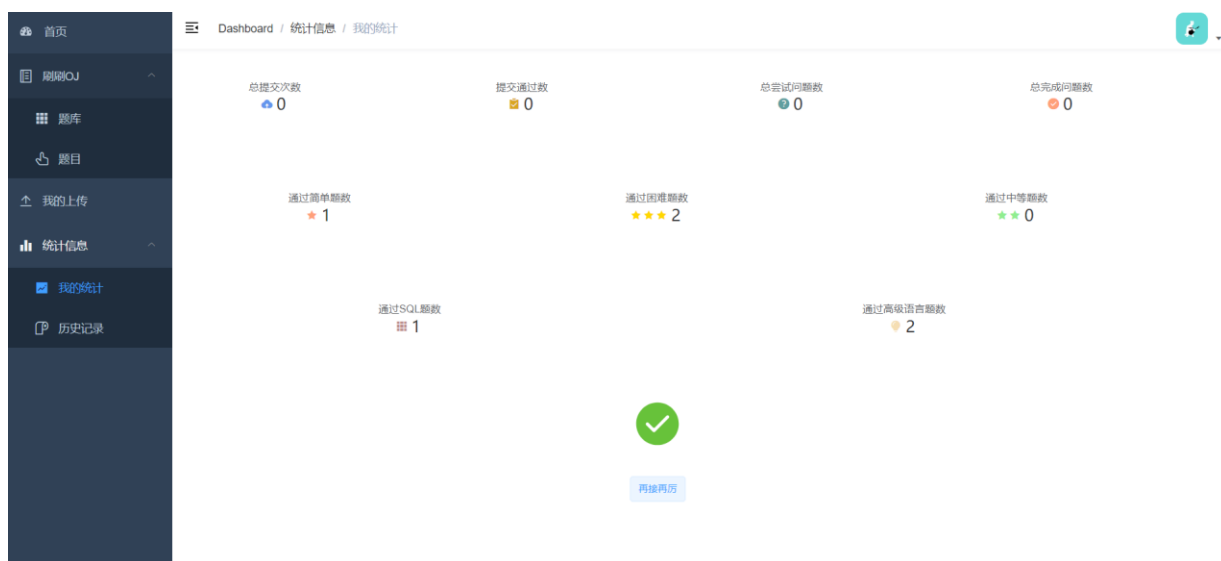
Dashboard / 我的上传

题目 标签 (目前未实现) 类型 查询 上传

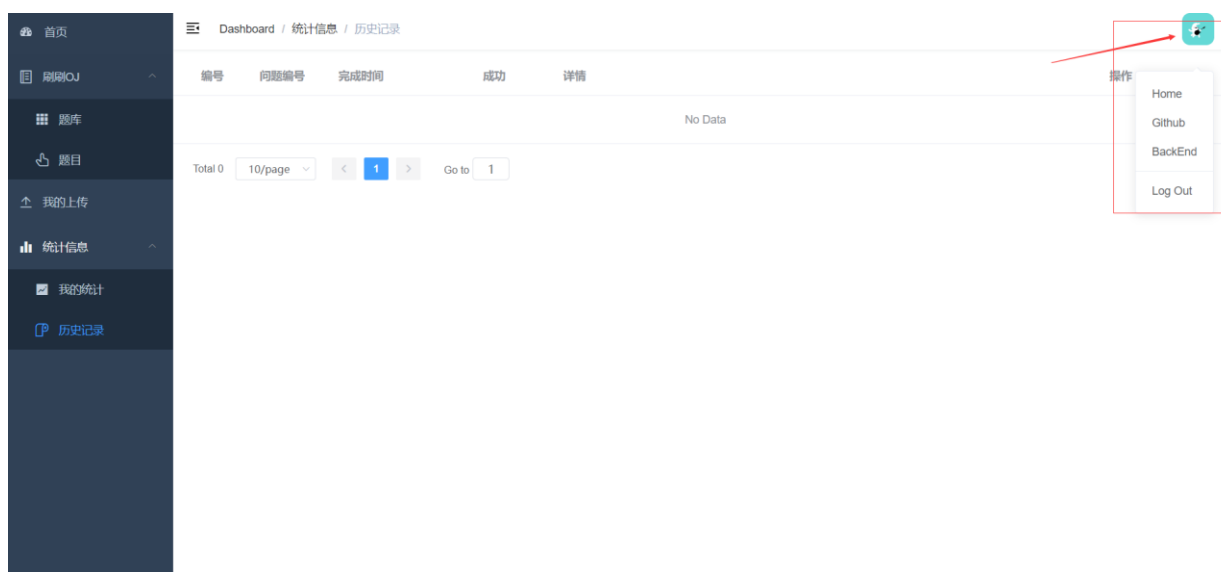
编号	审核状态	标题	类型	标签	通过率	难度	操作
2	正常	p2	高级语言	完成状态标签题解都还没做	0.0976	hard	测试用例 修改 删除
3	审核中	p3	高级语言	完成状态标签题解都还没做	0.1951	hard	测试用例 修改 删除
1670833048 821948417	正常	两数之和	高级语言	完成状态标签题解都还没做	0.1563	easy	测试用例 修改 删除
1672583479 537410049	审核中	最长回文子串	高级语言	完成状态标签题解都还没做	--	medium	测试用例 修改 删除
1672588588 812017666	审核中	最长回文子串	高级语言	完成状态标签题解都还没做	--	easy	测试用例 修改 删除

Total 19 5/page < 1 2 3 4 > Go to 1

当然你也能查看统计信息和历史记录。



你可以点击右上角的头像返回主界面，登出，或查看我们公开在 Github 上的代码：



## 2. 管理员使用方式

管理员的操作方式与普通用户类似，只是多了一部分管理功能。此外管理员无法被注册、封禁等，不能以任何方式创建管理员，

### 3. 开发者使用方式

直接 git clone 代码，对于后端项目还需要在 src/main/resources 添加配置文件 application-dev.yaml，并对下面用中文描述的内容做修改：

```
server:

  port: 8080


spring:

  datasource:

    primary:

      url: jdbc:mysql://你的 MYSQL IP 地址/用到的数据库名

      username: 用户名

      password: 密码

    secondary: # 用于 OJ 不得和 primary 使用同一张表

      url: jdbc:mysql://你的 MYSQL IP 地址/用到的数据库名

      username: 用户名(你需要确保这个用户只有这个数据库的管理权限)

      password: 密码


  redis:

    host: 你的 Redis IP 地址

    port: 你的 Redis 服务端口

    database: 0

    password: 你的 Redis 密码
```

sa-token:

# jwt 密钥 随便填一串字符串

jwt-secret-key: waa65123acbas4bn54634kqaf8706j

#### 4. 运维人员部署方式

可使用本项目开发过程中花了点时间开发的额外自动部署工具部署（地址：<https://github.com/Luyabs/front-end-project-deployment-scripts>），或直接使用命令行的方式打包部署：

## 1. 前端

### 1.1 打包前端文件

npm run build

### 1.2 将前端文件传到服务器上

rm -rf /home/online-judge/frontend/dist

# 在前端项目所在文件夹使用 cmd 操作

scp -r dist/ root@124.70.195.38:/home/online-judge/frontend/dist

### 1.3 docker 运行 nginx

切记把要部署的路径目录映射到容器内`

# 若已经 run 过 nginx 则 docker restart nginx 重启

```
docker run \  
  
--name nginx \  
  
-p 10000:10000 \  
  
-v /home/nginx/conf/nginx.conf:/etc/nginx/nginx.conf \  
  
-v /home/nginx/conf/conf.d:/etc/nginx/conf.d \  
  
-v /home/nginx/log:/var/log/nginx \  
  
-v /home/nginx/html:/usr/share/nginx/html \  
  
-v /home/online-judge/frontend/dist:/home/online-judge/frontend/dist \  
  
-d nginx:latest
```

## ## 2. 后端 (常规部署方式)

`/home/online-judge/backend/` 是 jar 包在服务器上的位置`

### ### 2.1 制作 jar 包

使用 maven

```
mvn package
```

### ### 2.2 清除运行中的 SpringBoot 应用

```
kill -9 $(ps -ef | grep "online-judge-backend-0.0.1-SNAPSHOT.jar" | grep -v grep |
```

```
awk '{print $2}')
```

```
rm -f /home/online-judge/backend/online-judge-backend-0.0.1-SNAPSHOT.jar
```

```
rm -f /home/online-judge/backend/nohup.out
```

### ### 2.3 部署并运行应用

# 先拷贝 (scp/ 其他方式) online-judge-backend-0.0.1-SNAPSHOT.jar 到

/home/online-judge/backend/

# 或者拖动方式上传

cd /home/online-judge/backend

nohup java -jar

/home/online-judge/backend/online-judge-backend-0.0.1-SNAPSHOT.jar

### ## 3. 后端 (容器部署方式)

后续跟进`