校园帮帮网
软件系统设计说明书

软工实践第二小组

2020 年 4 月 12 日

目 录

0. 文档介绍

0.1 文档目的与范围

0.1.1 目的

编写校园帮帮网软件系统设计说明书的主要目的是对校园帮帮网进行详细设计，在概要设计的基础上进一步明确系统结构，详细地介绍系统的各个模块，并对项目功能进行了具体描述，对前后端界面和功能实现提出了具体要求以指导前后端开发，让开发人员能够更好地投入到软件开发中。

0.1.2 范围

校园帮帮网主要的面向群体是在校生，在校生学习生活的特征是自学要求较高，课余生活丰富，因此该部分群体在学习生活中会遇到较多的问题，针对在校生的需求校园帮帮网推 出了以下几个主要业务：

①查看近期热门问题

②在线提问/回答

③根据关键词搜索相关问题

④进入个人中心查看我的提问/回答/关注问题/消息

⑤使用积分兑换奖励

0.2 读者对象

读者对象有用户（主要是在校生）、需求分析人员、项目开发人员、测试人员、文档编写人员等

0.3 参考文献

[1] 邹欣，构建之法（第三版），人民邮电出版社，2017

1. 系统概述

同学们在日常生活中往往会有一些关于学校资讯方面的疑惑（比如选课、学分、绩点、学术方面的问题等等），而百度、知乎等方式很难获取到这些方面信息，而寻找学长学姐帮忙作为同学们的主要途径不仅效率较低，而且对于很多性格较为内向的同学不太合适。在此背景下我们提出了一个专注于校园内容的问答网站——校园帮帮网来解决同学们的需求。

2. 文档设计规范

一级标题：宋体三号
二级标题：宋体四号
三级标题：宋体小四
正文：宋体五号

3. 开发、测试与运行环境

提示：*说明本系统应当在什么样的环境下开发和运行，有什么强制要求和建议？*

类别	标准配置	用途
开发环境	IDEA2019	后端开发
	Visual Studio Code	前端开发
	5.5.62 MySQL	数据库
测试环境	Postman	后端测试
	谷歌浏览器	前端测试
运行环境	谷歌浏览器	进入网页
	火狐浏览器	

4. 软件系统结构图

4.1 体系结构设计图

4.1.1 采用原因

本系统采用 MVC 设计架构，目的是为了使得前后端可以完全分离，后端团队在于前端团队确定好接口和数据结构后二者可以并行开发，从而提高效率。同时采用 MVC 架构使得前端视图和后端业务相分离，更容易改变程序的业务规则，提高了项目开发的灵活性和控制性。

4.1.2 架构说明

（1）Model(实体层)

本层主要用于存放与数据库相对应的实体类

（2）View(视图层)

本层的数据类主要是对实体层的实体类进行封装，从而以前端页面所需要的格式由 Controller(控制器)层发送给前端，避免前端人员对数据进行大量操作，可以直接读取所需要的数据，减少前端人员一些不必要的工作。

（3）Controller(控制层)

本层主要用于接收前端的请求，并通过调用 Service(服务层)方法对请求进行处理，并将 Service 层封装好的 View 层数据发送至前端。本层的设计结构是对应功能模块的需求，每个独立的功能模块都有对应的 Conrtoller 层类处理需求。

（4）Service(服务层)

本层主要是对于 Controller(控制层)处理的具体实现，包含主要的逻辑处理和数据处理和存储操作，在处理结束后将封装好的 View 数据类返回给 Controller 对象，由 Controller 对象返回前端。

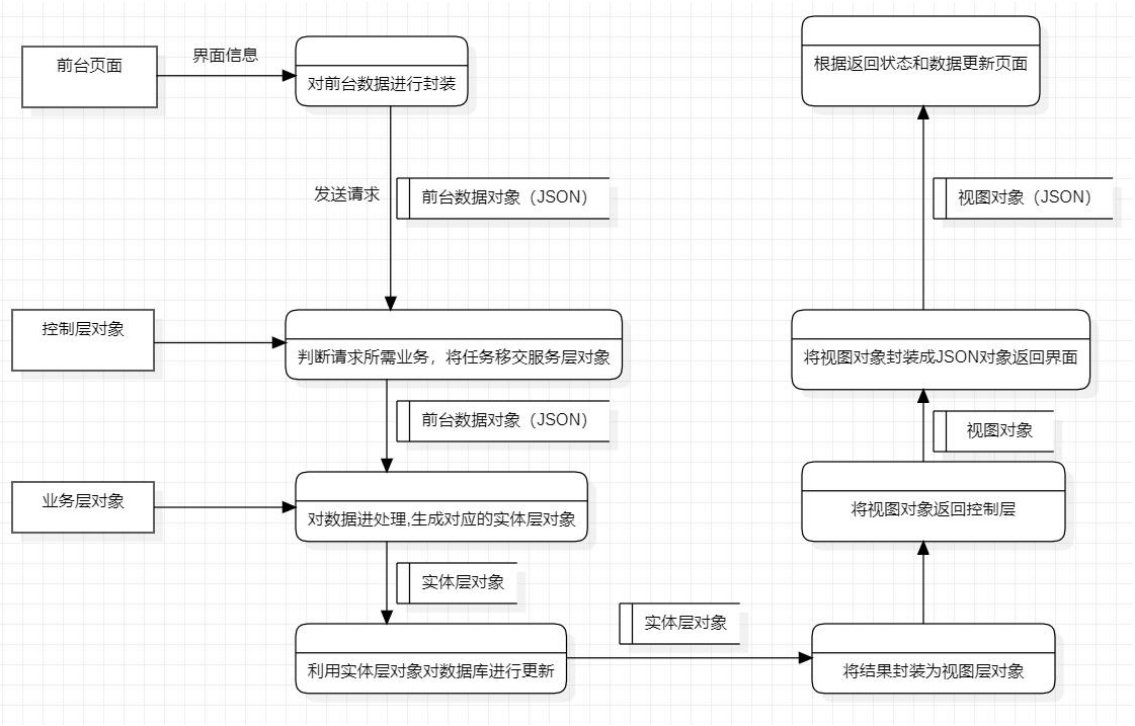
4.1.3 各层次之间配合流程

（1）Controller 层：获取前端数据，并判断应调用的 Service 对象/函数

（2）Service 层：对 Controller 层传入的参数进行处理，将数据转换为 Model 层对象，并对数据库进行增删查改操作，在处理结束后将前端所需的 Model 数据封装成 View 对象返回给 Controller 层

（3）Controller 层：将 Service 层返回的对象加上状态码等信息进一步进行封装，并返回给前端。

体系结构设计图



数据流图

4.2 功能模块层次图

4.2.1 划分依据与原因

本项目主要分成 7 个模块（用户账号管理、问题管理、回复管理、临时板块管理、投诉信息管理、奖励管理和消息管理）。划分的主要依据是按照实际的功能划分，模块的功能尽可能的单一，操作对象也尽可能单一，其目的是为了使得各个模块之间尽可能相互独立，在开发分工中可以并行开发不同的模块，避免因耦合度太高而带来合作难度的提高。

4.2.2 模块介绍

（1）用户账号管理（实现类：LoginController、UserController）

该模块主要是针对用户账户增删查改操作的管理，所包含的功能如下：

1） 登录校验

2） 密码修改

3） 密码重置

4） 用户增减

a. 逐个增加

b. 批量增加

c. 用户删除

i) 删除单个用户

ii) 批量删除用户

5） 用户列表获取

（2）问题管理（实现类：QuestionController）

1) 创建提问
2) 问题删除
    a. 逐个删除
    b. 批量删除
3) 问题信息更新
4) 问题列表获取
    a. 按热度排序（对应“首页”问题列表）
    b. 按时间由近到远排序（对应“等你来答”页面问题列表）
    c. 关键字获取（对应搜索框搜索结果和临时板块问题列表）
    d. 举报问题列表（对应后台举报管理问题列表）
    e. 特定用户问题列表（对应“个人主页”用户问题列表）
    f. 用户关注问题列表（对应“个人主页”关注问题列表）

（3）回复管理（实现类：ResponseController）

1) 新增回复
2) 删除回复
    a. 逐条删除
    b. 批量删除
3) 更新回复数据（点赞数、踩灭数、举报数）
4) 后台举报信息管理回复列表获取
    a. “问题详情”页面问题回复列表
    b. 后台举报信息管理问题列表

（4）临时板块管理（实现类：SectionController）

1) 新增临时板块
2) 撤销临时板块

（5）投诉信息管理（实现类：ReportController）

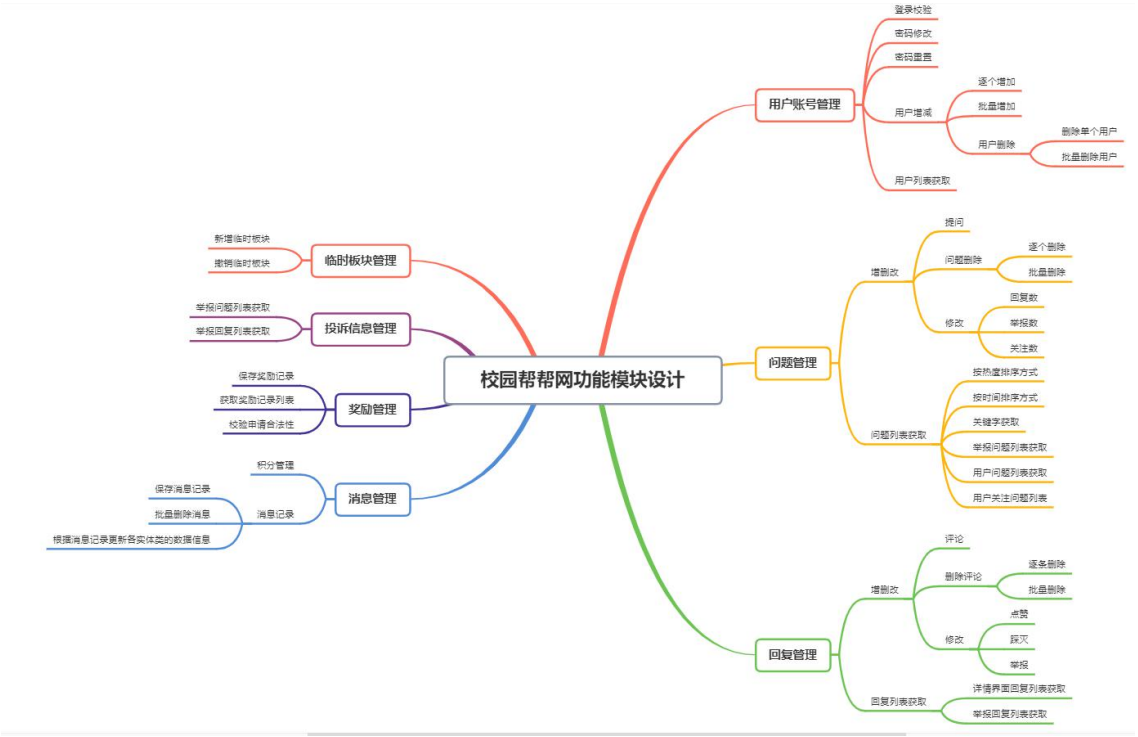
1) 举报问题列表获取
2) 举报回复列表获取

（6）奖励信息管理（实现类：RewardController）

1) 保存奖励记录
2) 获取奖励记录列表
3) 检验申请合法性

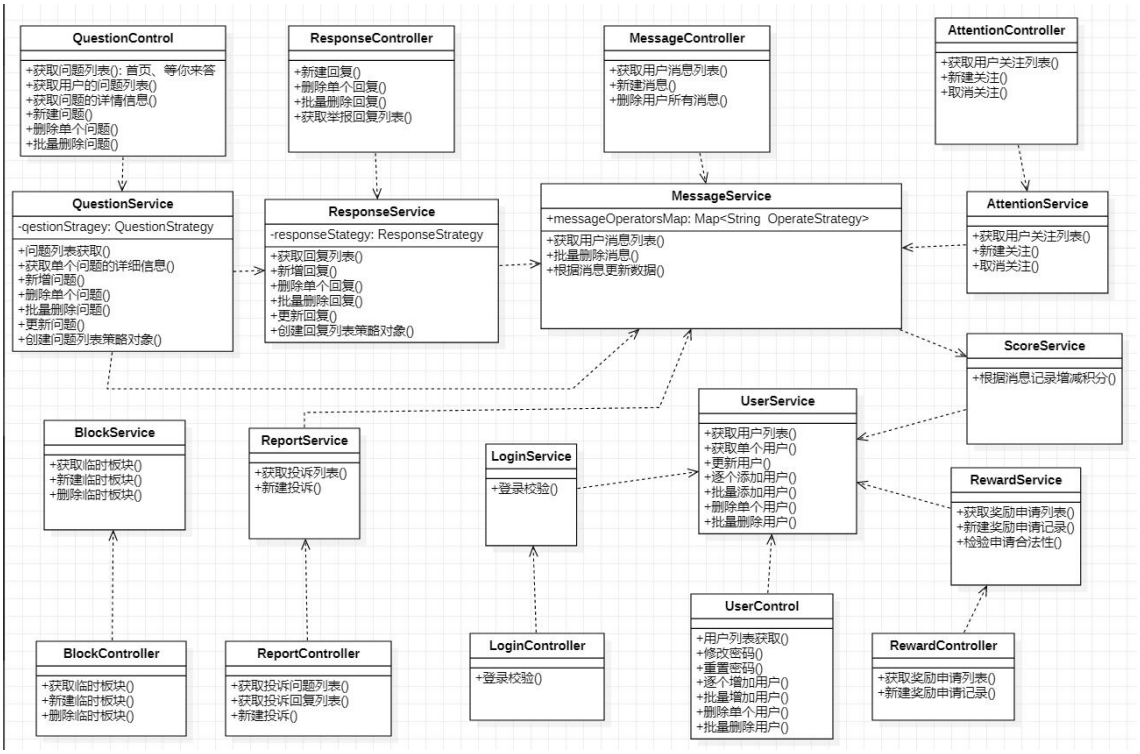
（7）消息管理（实现类：MessageController）

1) 积分管理
2) 保存消息记录
3) 批量删除消息记录
4) 根据消息更新实体类数据信息



功能模块层次图

4.3 设计类图

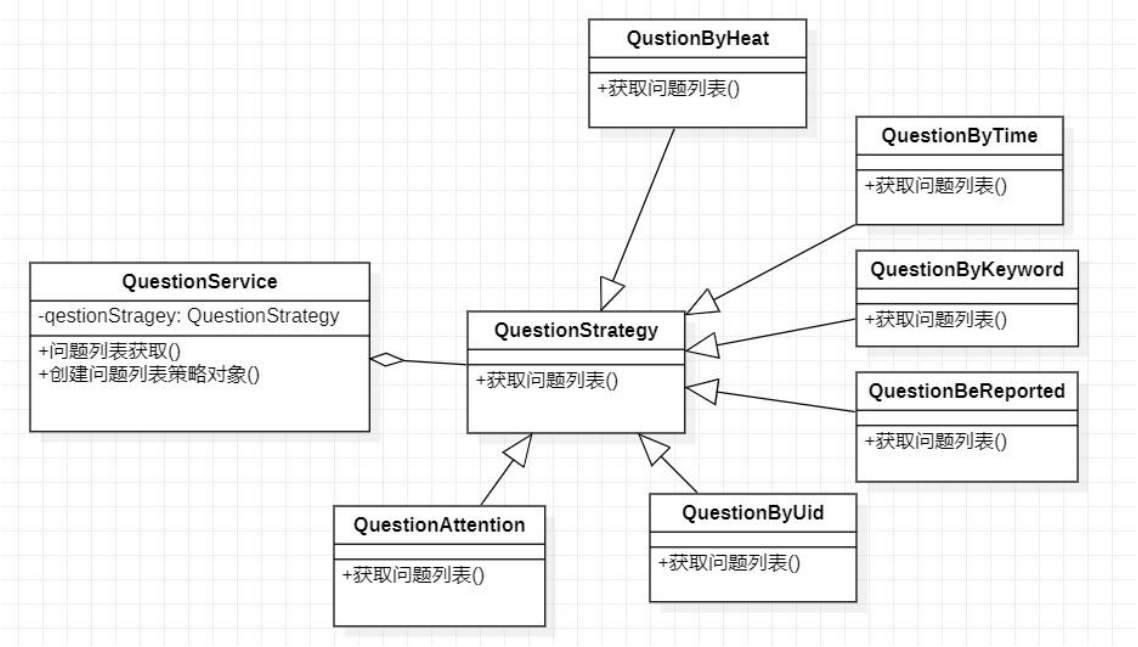


Controller 类与 Service 类依赖关系图

在该类图中我们主要体现的是控制层和服务层之间的关系，

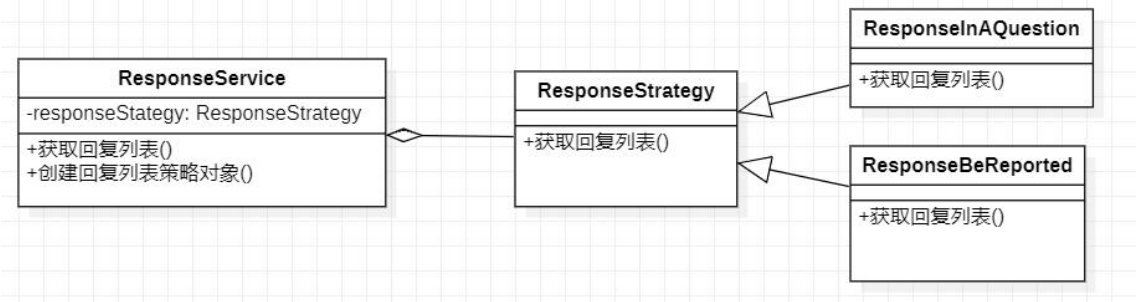
第 1、4 行是系统的控制层，主要为前端提供数据接口，接受前端的数据请求，并经过判断后调用相应的服务层对象对数据进行处理，最后对服务层返回的数据结果进行包装返回前端。控制层的设计是根据功能模块的划分，即每个功能模块都有自己对应的控制类，共包含 9 个控制类分别为：QuestionController、ResponseController、MessageController、AttentionController、BlockController、ReportController、LoginController、UserController、RewardController

第 2、3 行对应的是服务层对象，作用是为控制层的功能进行具体实现，负责对数据的处理，数据库的更新，并将处理后的结果返回包装成对应的视图层对象返回给控制层。



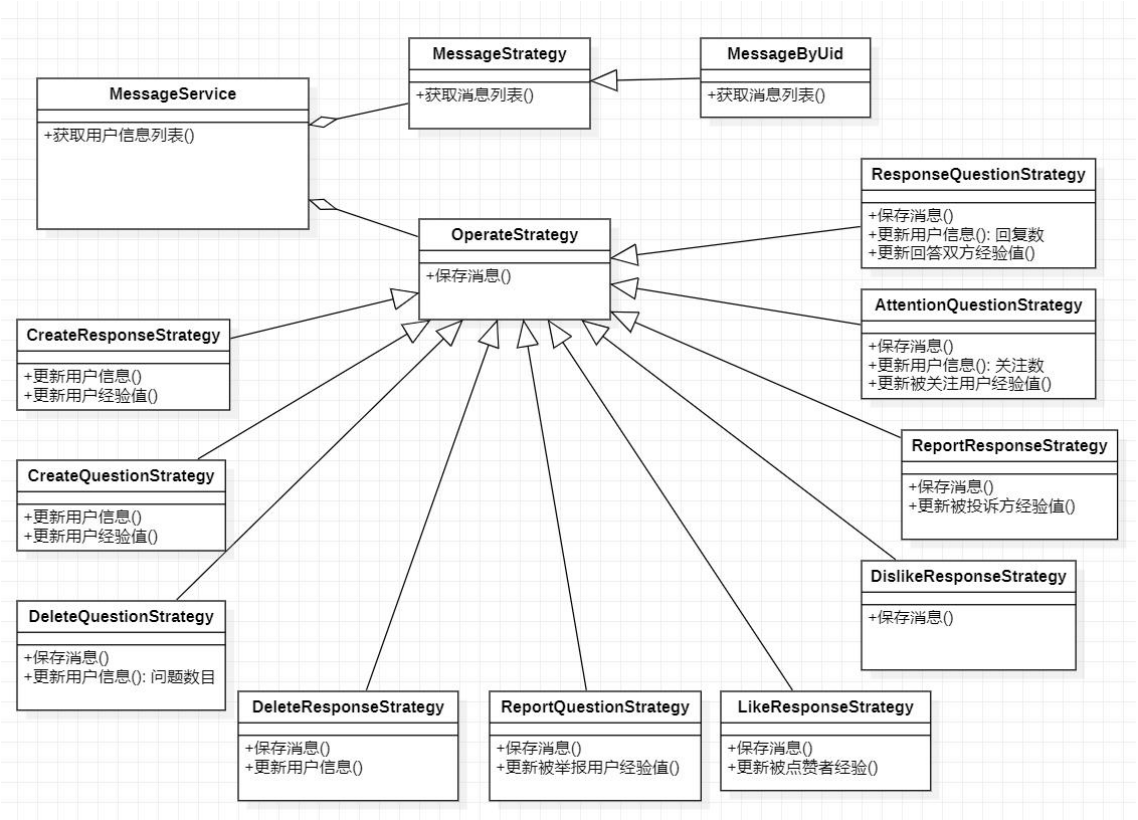
QuestionService 细节类图

在 QuestionService 类的设计中我们采用了策略模式对类的列表获取方式算法进行封装，因为在系统的前台页面中有多个界面都要获取问题列表，但是排序方式却不尽相同，若将这些算法全部都写在 QuestionService 类内部则会导致类十分的臃肿，增加了类的设计和实现难度，同时也给后期维护带来了困难。因此我们采用策略模式对类中获取列表方式的算法进行封装，通过继承公共类的方式使得相互之间可以替换，从而实现更换获取算法却不影响客户端的效果。



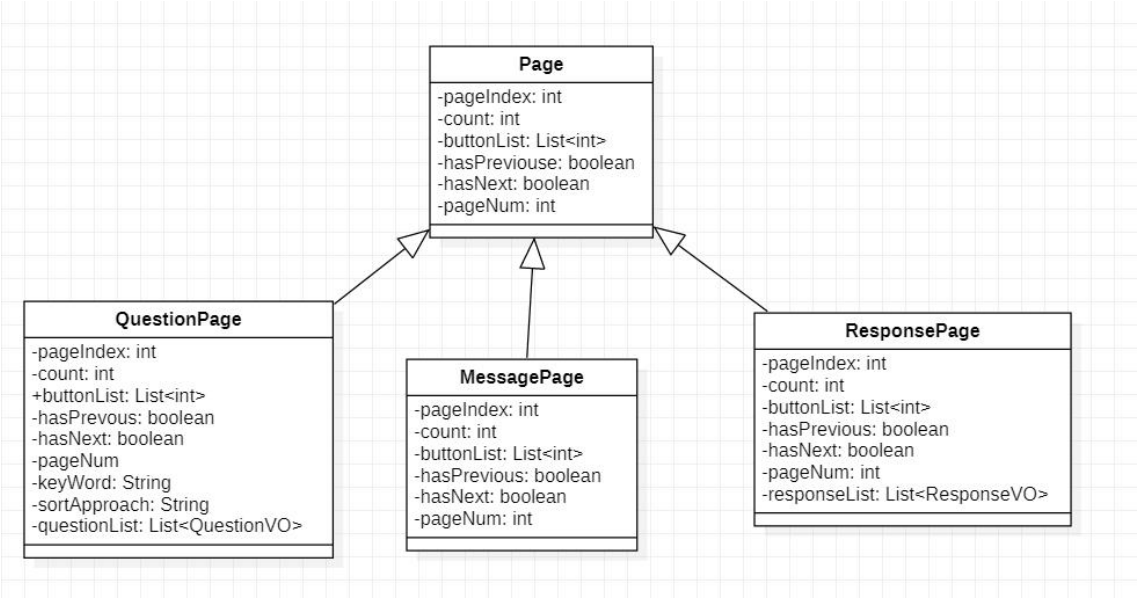
ResponseService 细节类图

ResponseService 类设计与 QuestionService 类似，是对获取回复列表的方式进行封装。

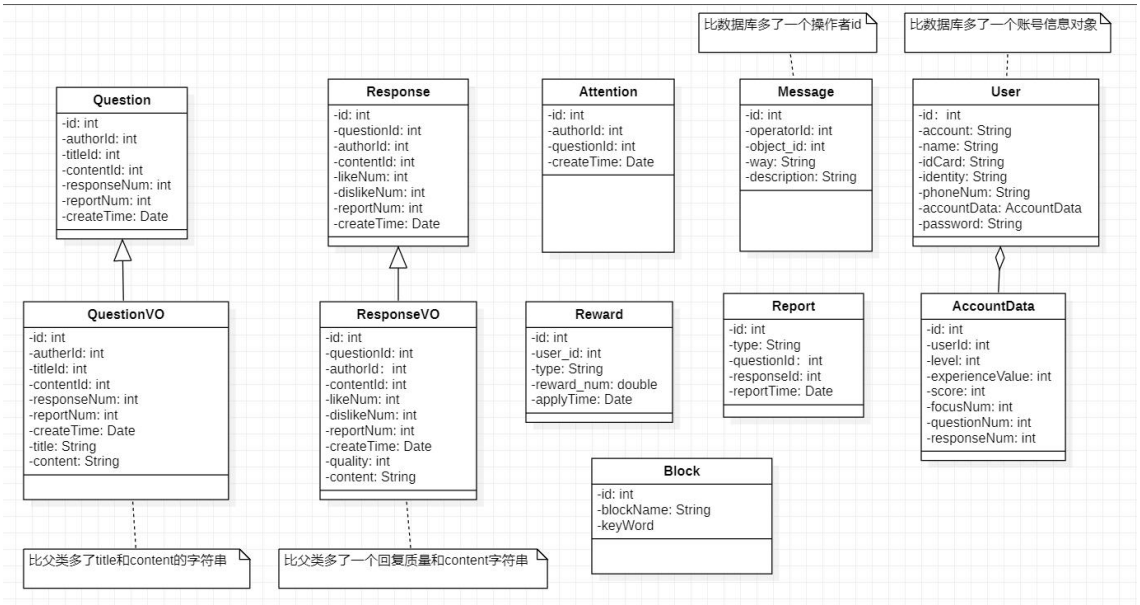


MessageService 细节类图

MessageService 类也采取了策略模式，主要是因为针对不同的消息系统有不同的处理方式，不同的处理方式写在不同的策略类中会提高系统的可扩展性。同时因为本系统有涉及积分的计算，很多的操作都会影响用户的积分，如：写问题，回复问题等，若将积分的处理分散在系统的各个地方则会给后期维护带来困难，因此我将积分的处理也放在了策略类中。 设计带来的好处：1.提高了类的可扩展性 (开闭原则)2.提高了程序的可维护性 (单一职责原则)3.使用聚合的方式设计策略类，有利于其他类对策略类的复用(合成复用原则)4.使逻辑更加清晰，处理方法可以独立变化不相互影响 5.简化了 MessageService 的设计和实现难度。


Page
-pageIndex: int
-count: int
-buttonList: List<int>
-hasPreviouse: boolean
-hasNext: boolean
-pageNum: int
QuestionPage
-pageIndex: int
-count: int
+buttonList: List<int>
-hasPrevous: boolean
-hasNext: boolean
-pageNum
-keyWord: String
-sortApproach: String
-questionList: List<QuestionVO>
MessagePage
-pageIndex: int
-count: int
-buttonList: List<int>
-hasPrevious: boolean
-hasNext: boolean
-pageNum: int
ResponsePage
-pageIndex: int
-count: int
-buttonList: List<int>
-hasPrevious: boolean
-hasNext: boolean
-pageNum: int
-responseList: List<ResponseVO>

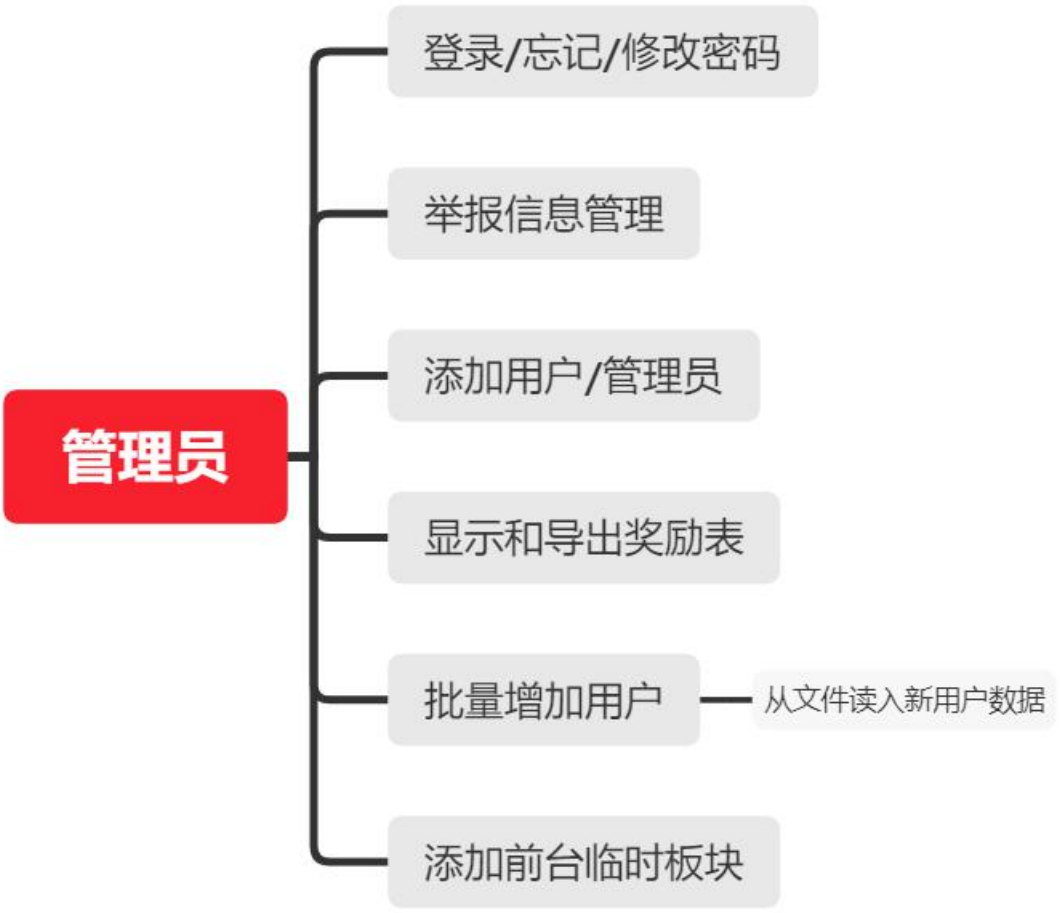
视图类


比数据库多了一个操作者id
比数据库多了一个账号信息对象
Question
-id: int
-authorId: int
-titleId: int
-contentId: int
-responseNum: int
-reportNum: int
-createTime: Date
Response
-id: int
-questionId: int
-authorId: int
-contentId: int
-likeNum: int
-dislikeNum: int
-reportNum: int
-createTime: Date
Attention
-id: int
-authorId: int
-questionId: int
-createTime: Date
Message
-id: int
-operatorId: int
-object_id: int
-way: String
-description: String
User
-id：int
-account: String
-name: String
-idCard: String
-identity: String
-phoneNum: String
-accountData: AccountData
-password: String
QuestionVO
-id: int
-autherId: int
-titleId: int
-contentId: int
-responseNum: int
-reportNum: int
-createTime: Date
-title: String
-content: String
ResponseVO
-id: int
-questionId: int
-authorId：int
-contentId: int
-likeNum: int
-dislikeNum: int
-reportNum: int
-createTime: Date
-quality: int
-content: String
Reward
-id: int
-user_id: int
-type: String
-reward_num: double
-applyTime: Date
Report
-id: int
-type: String
-questionId：int
-responseId: int
-reportTime: Date
AccountData
-id: int
-userId: int
-level: int
-experienceValue: int
-score: int
-focusNum: int
-questionNum: int
-responseNum: int
Block
-id: int
-blockName: String
-keyWord
比父类多了title和content的字符串
比父类多了一个回复质量和content字符串

实体类

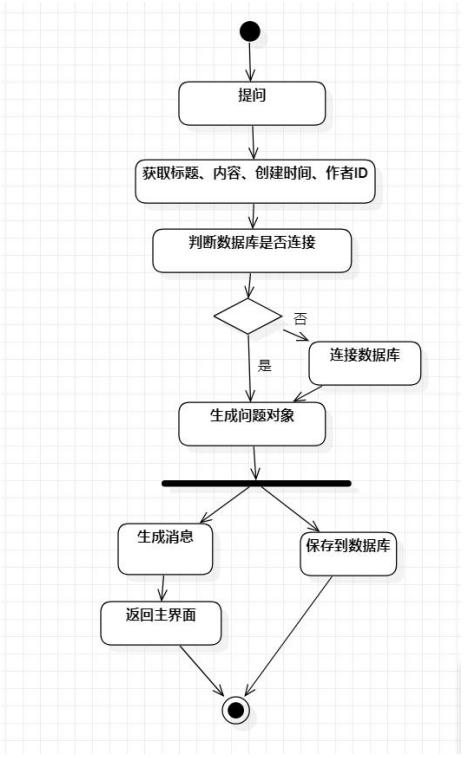
4.4 用例分析

普通用户用例

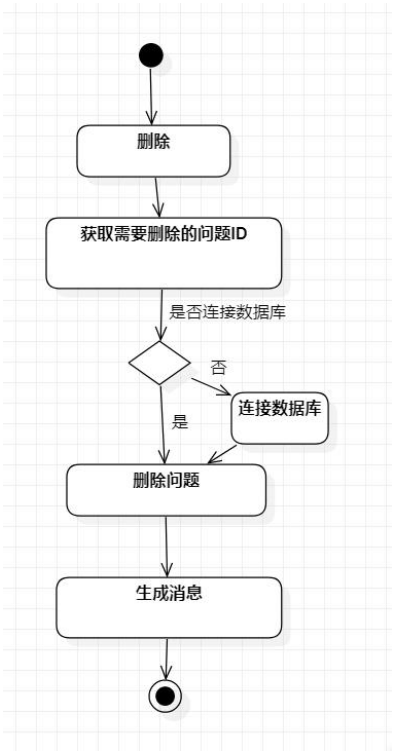

管理员
登录/忘记/修改密码
举报信息管理
添加用户/管理员
显示和导出奖励表
批量增加用户
从文件读入新用户数据
添加前台临时板块

管理员用例

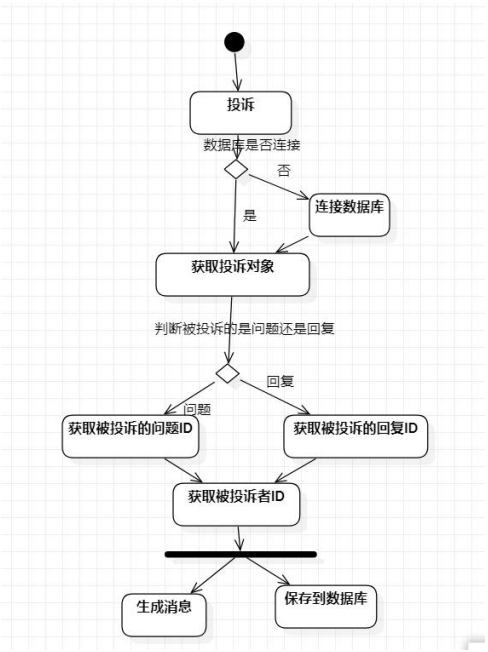
4.5 活动图



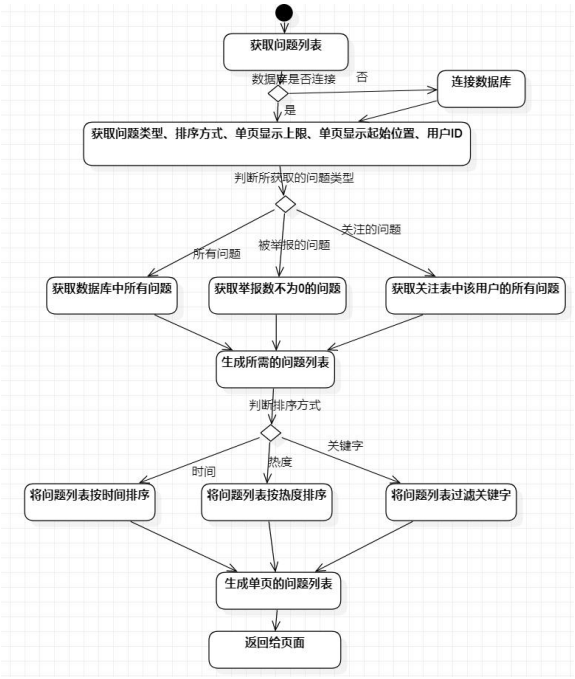
提问活动图



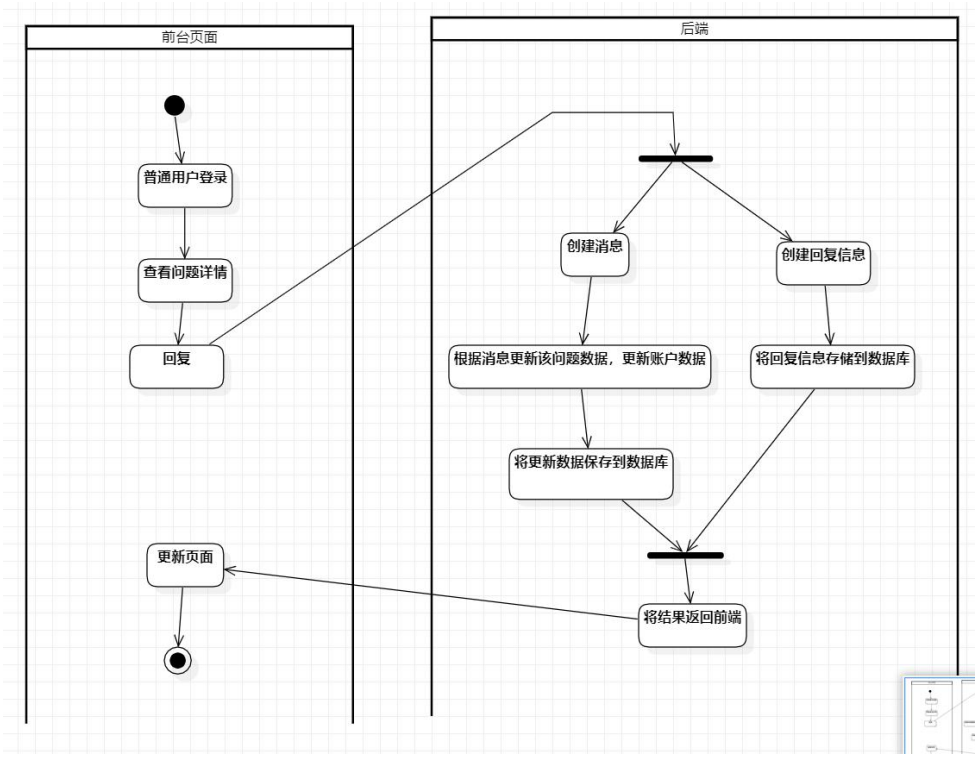
删除问题活动图



投诉活动图



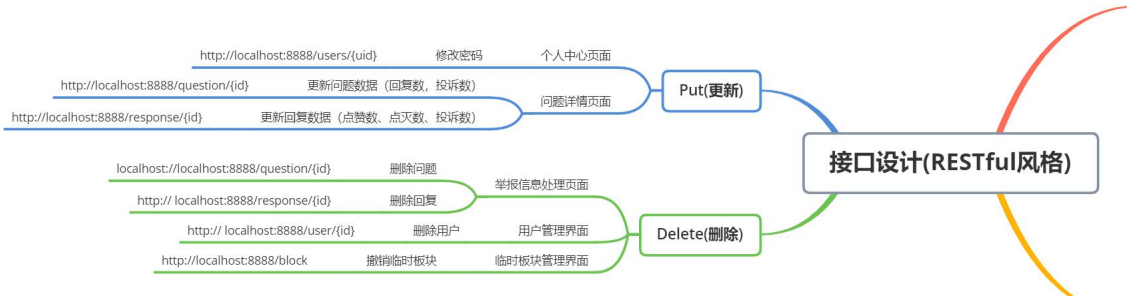
获取问题列表活动图



回复活动图

5. 接口设计设计概述

5.1 接口设计图



5.2 接口设计介绍

本项目的接口设计采用 RESTful 风格设计接口，根绝对数据的处理操作类型划分请求方式，GET(获取)、DELETE(删除)、POST(创建)、PUT(更新)，并制定以下规约：

1. 不在 URL 中出现动词,用名词体现出所需要的资源
2. HTTP 方法体现出对资源的操作
3. 通过 HTTP 状态码体现操作结果

该设计方案带来的好处：

1. 看 Url 就知道要什么资源
2. 看 http method 就知道针对资源干什么
3. 看 http status code 就知道结果如何
4. 为前端后端交互降低了接口方面的交流成本，是约定大于配置的体现

对于接口传输数据格式的详细说明请参考《接口设计说明》

6. 系统安全性和权限设计概述

6.1 安全性设计说明

针对系统的安全性，我们考虑了以下几种常见的 web 攻击方式，并制定了一下几种应对策略

1. CSRF(跨站请求伪造)

   CSRF 是指在用户登录后系统会将用户信息保存到用户浏览器存储，若用户同时一些恶意的网站，这些网站可能会利用存储的用户信息进入系统，从而可以对数据进行读取和破坏

   解决方案：

   在过滤器中进行 referer 识别，若 referer 为空或不属于自身域及子域，则拒绝后续操作

2. SQL 注入

SQL 注入是指某些用户在输入某些数据的时候会故意加上一些 SQL 语句，若没有经过处理直接拿这些数据进行数据库操作可能会导致跨越权限或对数据造成损坏。

解决方案

登录账号在进行数据查询前会进行字符检查，账号只能由字母和数字组成。

例：输入为“123 or ‘1’ = ‘1’”的账号将不会进行数据库查询，直接判为账号错误

3. SSX(跨站脚本攻击)

SSX 是指某些用户存入数据库的信息（如系统中的问题标题等）中夹杂 js 代码，若使用 jsp 等方式直接将值嵌入 html 语句中有可能在页面加载时执行该部分语句达到某些目的。

解决方案

前端内容赋值全部通过 js 改变 dom 对象的 innerHTML 属性可以避免该情况的发生

其他的数据安全性设计

在前后端数据传输的过程中对用户的数据进行加密处理（具体的加密算法还没有决定，等到项目进行安全性部分时再决定具体采用那种加密算法）

6.2 系统权限设计说明

本系统共有两种用户，三种角色每种角色对应的权限如下：

管理员(管理员用户)：

登录进入的是后台页面，能够对所有用户具有增删查改权限；同时也对文章和评论具有查看、删除权限；对奖励记录具有查看和删除权限；

老师(普通用户)：

进入前台页面，具有对问题、回复进行新建和查找权限；具有查看和修改本用户信息的权限；相较于另一个普通用户老师没有兑换奖励的权限

学生(普通用户)：

进入前台页面，对问题、回复具有新建和查看权限，同时具有兑换奖励权限；用户操作只具备查看和修改本用户的权限

7. 数据库表设计概述

在数据库设计方面我们采用的关系型数据库，共设计了 10 张表，每个实体类都有对应的数据库存储表。其中为了提高问题表的查询和更新效率，我们将问题的标题和内容分别单独提取出来设置一张表来存储。

用户表：存储用户的基本个人信息

账户信息表：存储用户的账户信息，如经验值，等级等

问题表：用于存储问题的数据信息，如回复数，投诉数等

回复表：用于存储回复的数据信息，如点赞数，点灭数等

标题表：用于将问题的标题单独放到一张表中存储

内容表：用于将问题和回复的内容单独放到一张表里存储。

关注表：用于记录用户的关注问题

消息表：用于存储用户的消息记录

奖励申请表：用于记录所有用户的奖励申请记录

临时板块表：用于保存前台临时板块的信息

具体更为细节的设计说明请参考《数据库设计说明书》