吉首大学



码

客

之

家

详

细

设

计

文

档

|  |  |
| --- | --- |
| 成 员： | 舒镐、周鑫、于亮 |
| 班 级： | 20计科一班 |
| 学 号： | 2020404102、2020404101、2022200565 |
| 时 间： | 2023年6月30日 |

目录

[1. 系统架构设计 3](#_Toc514710119)

[1.1系统架构风格 3](#_Toc668839917)

[1.1.1 调研方案 3](#_Toc1263076621)

[1.1.2 预选型方案 3](#_Toc652918552)

[1.1.3 预选型分析 4](#_Toc2108150941)

[1.1.4 预选型理由 4](#_Toc360173534)

[1.2主要组件 4](#_Toc1827668692)

[1.2.1前端客户端 4](#_Toc21619756)

[1.2.2后端服务端 5](#_Toc438502749)

[1.2.3运维测试部署 6](#_Toc1899309586)

[1.2.4数据库 7](#_Toc1499282894)

[1.3系统架构图 8](#_Toc2021969207)

[1.4 DevOps流程图 9](#_Toc1455231921)

[2. 模块以及类设计 9](#_Toc391640564)

[2.1 ArticleController类设计 9](#_Toc265581093)

[2.2 QuestionController类设计 10](#_Toc1150411585)

[2.1 CommentController类设计 12](#_Toc1172235154)

[2.4 FeedbackController类设计 13](#_Toc741255700)

[2.5 ReportController类设计 14](#_Toc731913653)

[2.6 UserController类设计 15](#_Toc486435955)

[3. UI设计 16](#_Toc58851556)

[4. 数据库设计 16](#_Toc1275624072)

[5. 接口设计 16](#_Toc1084530103)

1. 系统架构设计

1.1系统架构风格

系统初步采用B/S（浏览器-服务器）架构模式。

1.1.1 调研方案

（1）客户端-服务器-数据库

（2）客户端-web服务器-应用服务器-数据库

（3）客户端-负载均衡器-中间服务器-应用服务器-数据库

选型分析：

首先我们本次的技术选型为为客户端-负载均衡器-应用服务器-数据库。系统初始架构规模使用该方案足够满足需求，后续可进行升级变迁。

（扩展：后续在开发进度完成后，同时将继续进行其他设备、客户端的开发。以下为移动端技术选型方案调研）

1.1.2 预选型方案

（1）Native APP（原生开发）

对于对性能和用户体验要求非常高的应用，可以选择原生开发，即使用各平台提供的开发语言和工具进行开发。如使用Swift和Objective-C进行iOS开发，使用Java和Kotlin进行Android开发。

（2）Web APP（H5开发）

Web APP使用HTML、CSS和JavaScript等前端技术进行开发。这种开发方式可以在各种平台上运行，包括桌面浏览器和移动浏览器。Web APP具有跨平台的特性，一次开发可以在不同的设备上使用，减少了开发成本和时间。然而，相比原生开发和混合开发，Web APP的性能和用户体验通常较差。

（3）Hybrid APP（混合开发）

混合开发通常通过使用WebView组件来实现，该组件允许将Web内容嵌入原生应用中。开发人员可以使用框架和工具，如Flutter和React Native，来进行混合开发。

1.1.3 预选型分析

首先本次我们将会采用响应式布局的设计，来兼容Web APP（H5）移动端的适配，通过使用灵活的布局、流体网格和媒体查询等技术，使内容能够根据设备的特性自动适应和重排。同时我们将会在后面采用React Native来实现混合开发移动端应用。结合了Web技术和原生开发，可以兼顾跨平台特性和性能优势，提供较好的用户体验。从而实现跨端开发。

1.1.4 预选型理由

（1）响应式设计：实现多设备兼容性，响应式设计可以让使用不同设备的用户都得到良好的体验，无论是台式机、平板电脑还是移动设备；维护成本降低，通过响应式设计，只需要维护一个代码库就可以适应不同的设备，而不是为每个设备单独开发和维护应用程序，这样可以减少开发成本和维护工作量；以及确保应用在不同设备上保持同步，同时可以提供更好的扩展性，可以为未来的设备和屏幕尺寸做好准备，随着新设备的出现和技术的发展，应用要能够自动适应新的设备，并提供一致的体验给用户。

（2）混合开发：Learn once, write anywhere. React Native允许开发人员使用JavaScript语言编写一次代码，并将其部署到多个平台，如iOS和Android。这样可以减少开发工作量和时间，并使应用能够同时在不同平台上运行。同时我们的前端技术栈主要就是采用以React为基础的各种库以及框架，相较于Flutter还需要学习Dart一门新的语言可以减少学习成本。

1.2主要组件

1.2.1前端客户端

前端客户端前台采用基于React库的Nextjs服务端渲染框架。后台采用Vuejs框架。

组件选型理由：

首先我们明确的认知到我们的系统是一个技术交流平台，那么最重要的就是文章与问题这类内容的展示，以及用户的阅读体验，那么传统的CSR（客户端渲染）会导致首次渲染的白屏时间过长，由于所有 JS 都打包在一个文件中，在这个 JS 加载完成之前，在页面上是看不到任何东西，这就会让用户感受到白屏，同时对于搜索引擎而言，只能在DOM中发现一个DOM节点，那么由于搜索引擎不解析Javascript代码，故不利于SEO。故我们采用SSR服务端渲染的框架技术。

组件选型分析：

1. 简化开发：Nextjs提供了简化的开发模式，约定优于配置（Convention Over Configuration）的原则可以快速搭建和开发应用程序。它提供了许多开箱即用的功能和组件，减少了开发人员的重复工作，使开发过程更加高效
2. 服务端渲染（SSR）：Next.js提供了服务端渲染的能力，将React组件在服务器上预渲染成HTML，并将其发送到客户端。这样可以改善初始加载性能，减少客户端渲染所需的时间，并提供更好的搜索引擎优化（SEO）和用户体验。
3. 更好的首屏加载性能：由于Next.js采用了服务端渲染，用户在首次访问页面时可以直接获取到已经渲染好的HTML内容，而无需等待客户端加载和渲染。这大大减少了首屏加载时间，提供了更快的用户体验。
4. 支持静态站点生成（SSG）：Next.js不仅支持服务端渲染，还提供了静态站点生成的能力。这意味着可以预先生成静态HTML页面，并将其部署到任何静态主机上，从而实现更快的加载速度和更佳的可扩展性。

1.2.2后端服务端

后端服务端采用基于Java语言的SpringBoot服务端框架。

组件选型理由：

我们最初的系统是一个单体架构的系统，后端所有的服务都在一个服务器/容器中，但是不排除我们的系统有使用分布式架构的需求可能，所以在最初的设计中，我们就希望能够兼容以后可能会使用的微服务以及分布式架构。那么Spring及其全家桶就有很好的微服务支持，可以配合一系列工具实现完美的分布式架构。

组件选型分析：

1. 简化开发：Spring Boot也提供了简化的开发模式，约定优于配置（Convention Over Configuration）的原则可以快速搭建和开发应用程序。它提供了许多开箱即用的功能和组件，减少了开发人员的重复工作，使开发过程更加高效。
2. 易于上手：Spring Boot建立在Spring框架之上，继承了Spring框架的优秀特性和生态系统，但相对于传统的Spring应用，Spring Boot提供了更简单、更易于上手的编程模型。
3. 生态系统支持：Spring Boot拥有庞大的生态系统和活跃的社区支持。它集成了许多流行的第三方库和框架，如Spring Data、Spring Security、Spring Cloud等，可以方便地与其他组件和服务进行集成。
4. 微服务支持：Spring Boot天生支持微服务架构的开发。它提供了各种特性和工具，如内嵌的服务器、服务注册与发现、负载均衡、断路器等，可以轻松构建和部署分布式系统和微服务应用。

1.2.3运维测试部署

采用DevOps方法论，采用Git分布式版本控制系统存储拉去代码，采用自动化构建工具Jenkins实现项目的持续集成，持续部署。采用Docker实现容器化部署。

组件选型理由：

我们希望通过功能模块的划分来实现并发的开发，所以git这种分布式版本管理系统对代码的管理，以及团队成员的协作有很好的帮助，同时我们希望省去手动打包构建，测试部署的重复性操作，故采用Jenkins这种自动化构建工具，来减少不必要的重复工作量，加快系统的开发与发布。同时，我们希望现今的单体式架构能够和以后可能扩展的微服务与分布式架构相兼容，所以使用Docker来进行容器化部署，来实现应用程序在不同环境中的一致性和可移植性，确保每个应用的隔离性，同时可以免去生产环境的搭建。

组件选型分析：

1. DevOps方法论：DevOps方法论可以将我们的开发团队和运维团队进行无缝集成，通过自动化流程、持续集成和持续部署等实践，实现快速交付高质量的软件。采用DevOps方法论可以提高团队的协作效率、减少错误和故障，并实现持续改进。
2. Git分布式版本控制系统：Git的分布式特性可以让我们的团队成员在本地独立工作，减少了对中央服务器的依赖，并通过强大的分支管理能力，来支持并行开发和版本回退等操作。
3. Jenkins自动化构建工具：采用Jenkins实现项目的持续集成，自动构建、测试和部署代码。Jenkins提供了丰富的插件生态系统，可以与其他工具集成，如代码静态分析、自动化测试和部署到各种环境等。
4. Docker容器化部署：Docker通过将应用程序及其依赖打包成独立的容器，实现了应用程序在不同环境中的一致性和可移植性。采用Docker可以快速部署、扩展和管理应用程序，同时提供了隔离性、易于管理和资源利用效率高的优势。

1.2.4数据库

系统使用MySQL作为主要的关系型数据库管理系统，用于存储和管理数据。

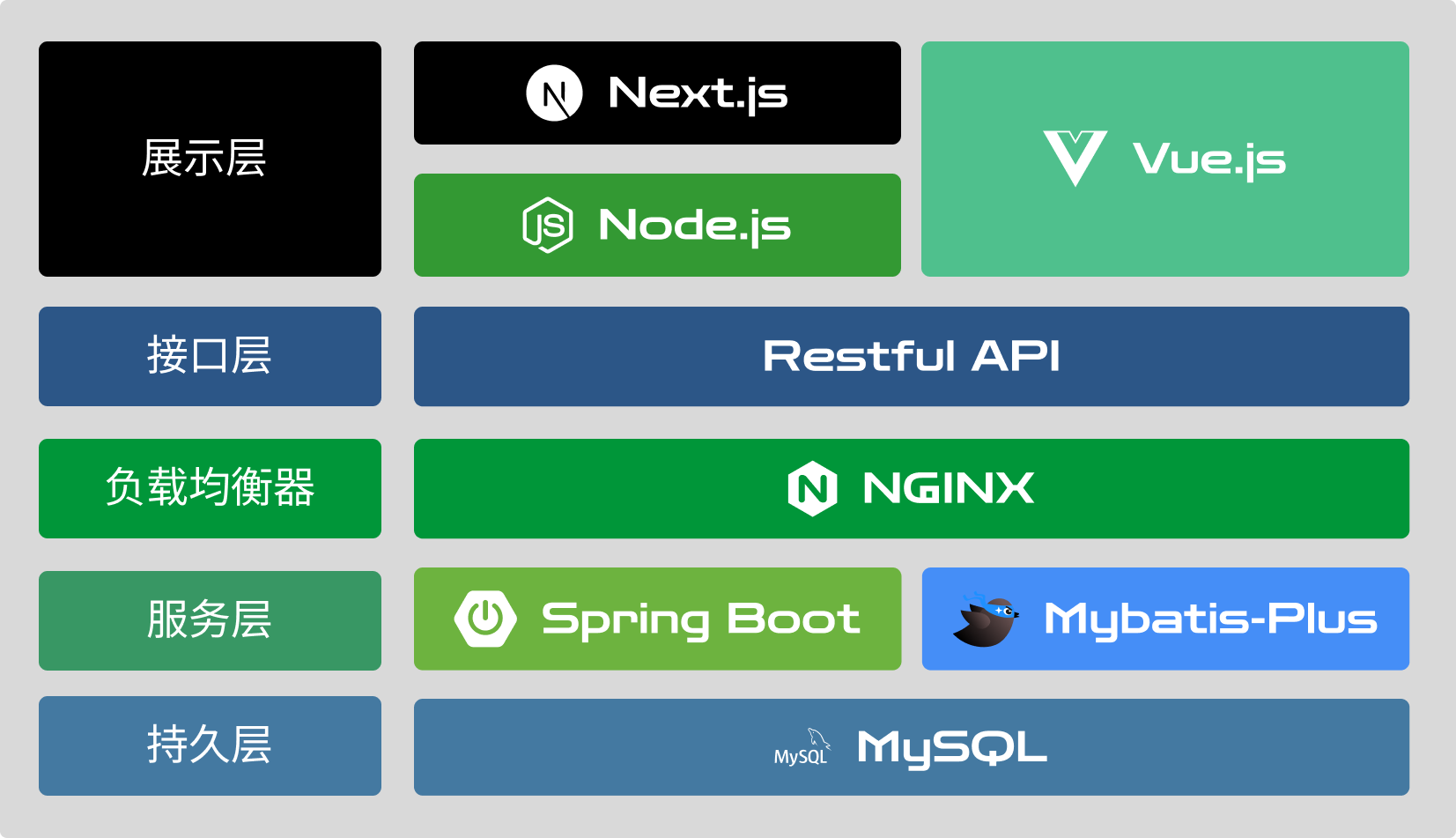
组件选型理由：

我们存储的数据大多都是是结构化的文本数据，而并非要存储和处理图片、音频、视频等非结构化数据，如果采用NoSQL数据库，那么就比较脱离真正的业务需求，所以关系型数据库是最佳的选择，那么在众多的关系型数据库之中，我们之所以选择MySQL，首先是由于MySQL的开源性。甲骨文自身另外的Oracle，虽然功能强，稳定性好，安全性高，但是收费昂贵。我们目前的项目规模、需求，采用Oracle也是一种对资源的浪费。且相比较与PostgreSQL，我们当前的项目对于数据的完整性以及复杂性要求，都没有上升到更高的程度，我们只需要一个简单，性能较强且相对轻量的数据库。

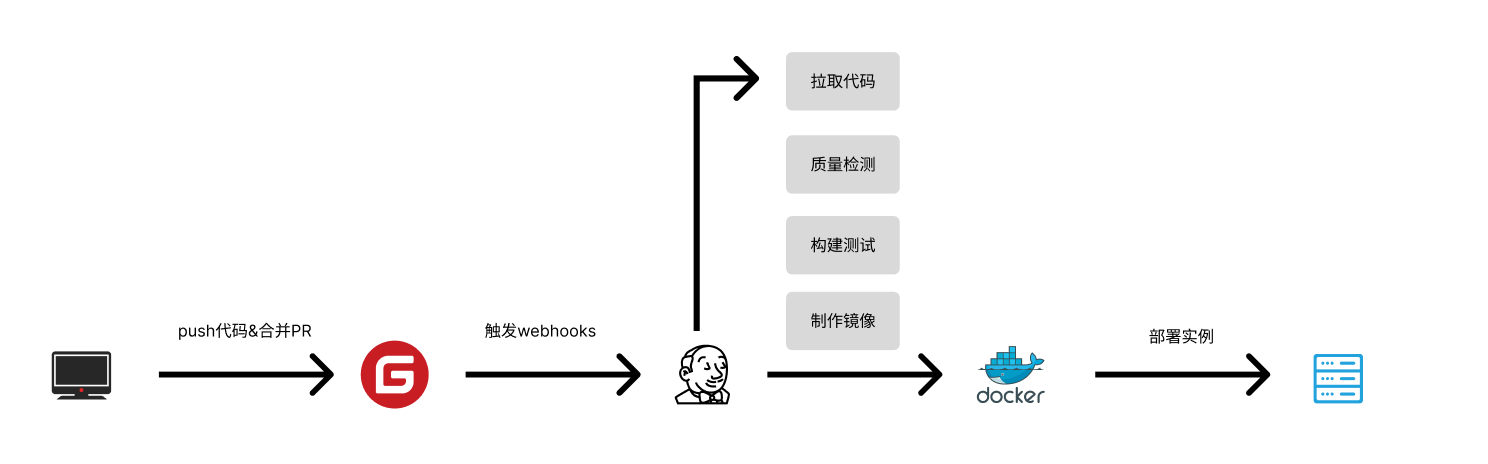
组件选型分析：

1. 开源性：MySQL是开源的，开放源代码且无版权制约，自主性强，使用成本低，可以免费使用。
2. 社区支持和生态系统：MySQL具有庞大的开源社区支持。我们可以轻松获取使用帮助，并且可以从社区中获得丰富的文档、教程和支持。
3. 跨平台：MySQL可在多个操作系统上运行，包括Windows、Linux、macOS等，这使得它非常灵活，并且可以方便地部署到各种环境中。
4. 可扩展性：MySQL具有良好的可扩展性，可以在需要处理大量数据和高并发访问的情况下进行水平扩展。它支持主从复制、分区和集群等技术，可以实现高性能和高可用性。

1.3系统架构图



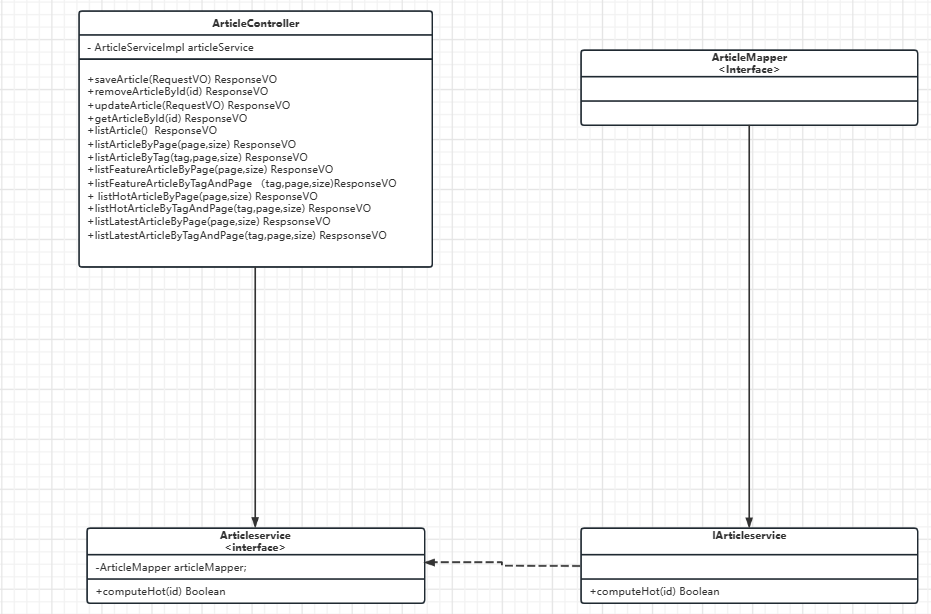
1.4 DevOps流程图



1. 模块以及类设计

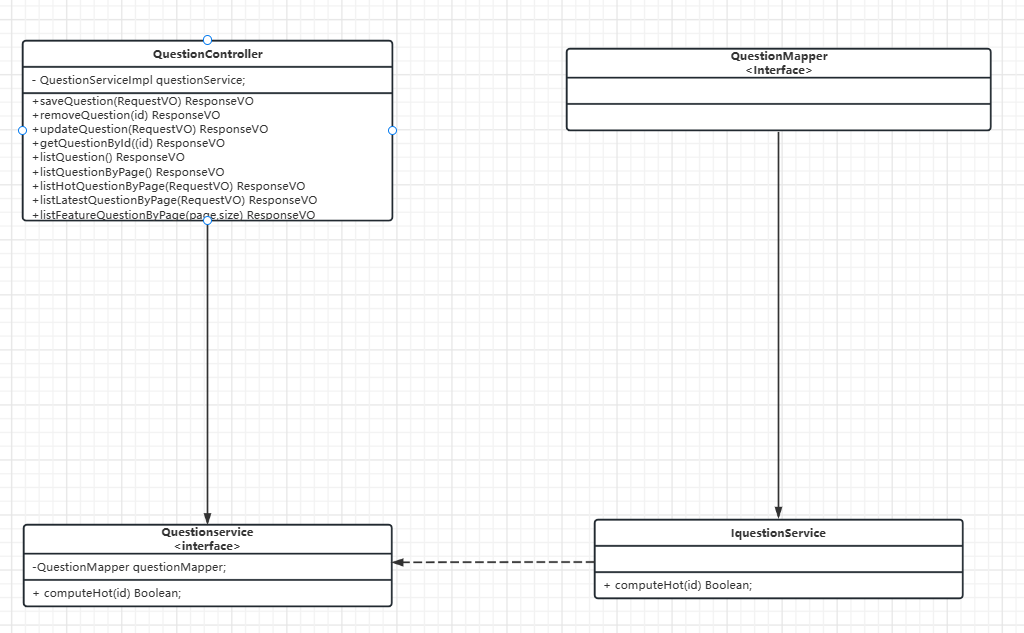
2.1 ArticleController类设计

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 方法名 | 等级 | 功能描述 | 参数 | 返回值 |
| getArticleById | public | 根据id获取单篇文章 | int id | ResponseVO |
| listArticle | public | 获取所有文章 | null | ResponseVO |
| listArticleByPage | public | 获取所有文章+分页 | int page, int size | ResponseVO |
| listArticleByTag | public | 根据标签获取单篇文章 | int page, int size | ResponseVO |
| listFeatureArticleByPage | public | 获取精选文章+分页 | int page, int size | ResponseVO |
| listFeatureArticleByTagAndPage | public | 根据标签获取精选文章+分页 | String tag, int page, int size | ResponseVO |
| listHotArticleByPage | public | 获取热门文章+分页 | int page, int size | ResponseVO |
| listHotArticleByTagAndPage | public | 根据标签获取热门文章+分页 | String tag, int page, int size | ResponseVO |
| listLatestArticleByPage | public | 获取最新文章+分页 | int page, int size | ResponseVO |
| listLatestArticleByTagAndPage | public | 根据标签获取最新文章+分页 | String tag, int page, int size | ResponseVO |
| saveArticle | public | 新增文章 | RequestVO | ResponseVO |
| removeArticleById | public | 通过id删除文章 | int id | ResponseVO |
| updateArticle | public | 修改文章 | RequestVO | ResponseVO |



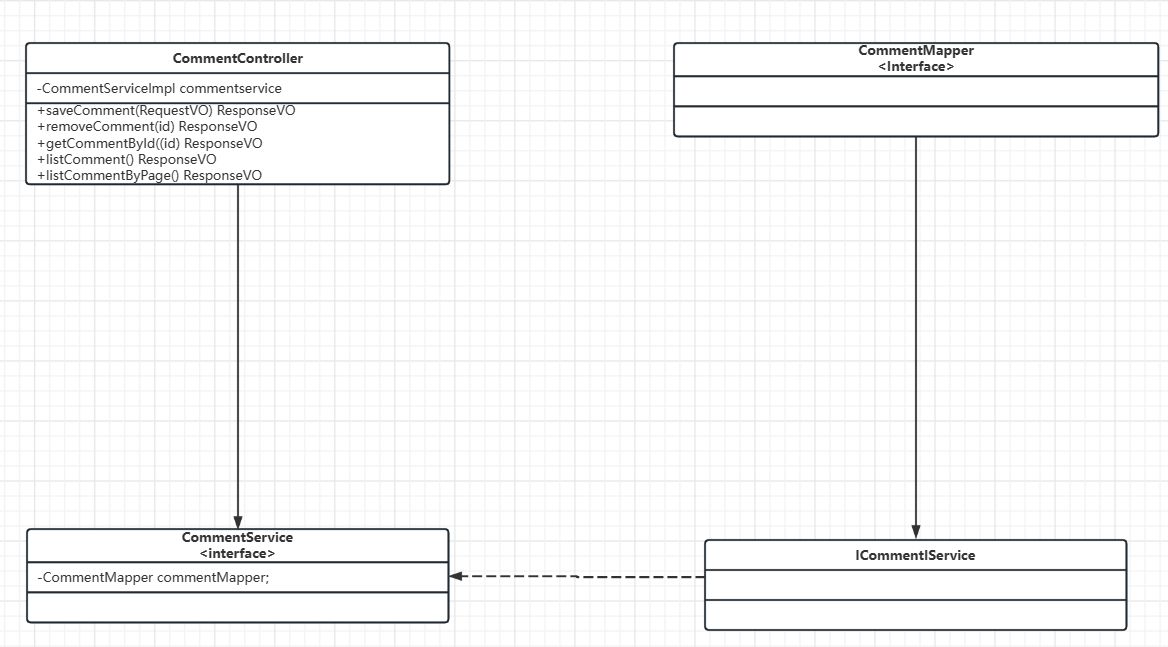
2.2 QuestionController类设计

| 方法名 | 等级 | 功能描述 | 参数 | 返回值 |
| --- | --- | --- | --- | --- |
| getQuestionById | public | 根据id获取单个问题 | int id | ResponseVO |
| listQuestion | public | 获取所有问题 | null | ResponseVO |
| listQuestionByPage | public | 获取所有问题+分页 | int page, int size | ResponseVO |
| listFeatureQuestionByPage | public | 获取精选问题+分页 | int page, int size | ResponseVO |
| listHotQuestionByPage | public | 获取热门问题+分页 | int page, int size | ResponseVO |
| listLatestQuestionByPage | public | 获取最新问题+分页 | int page, int size | ResponseVO |
| removeQuestionById | public | 根据id删除问题 | int id | ResponseVO |
| saveQuestion | public | 新增问题 | RequestVO | ResponseVO |
| updateQuestion | public | 修改问题 | RequestVO | ResponseVO |



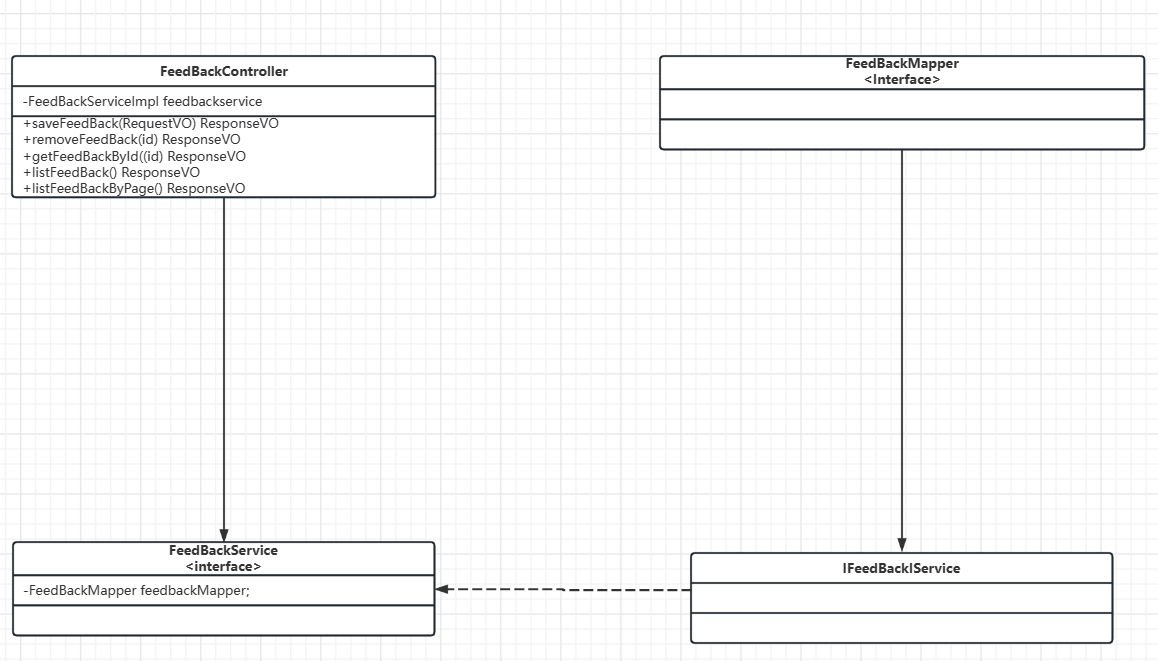
* 1. CommentController类设计

| 方法名 | 等级 | 功能描述 | 参数 | 返回值 |
| --- | --- | --- | --- | --- |
| getCommentById | public | 根据id获取单条评论 | int id | ResponseVO |
| listComment | public | 获取所有评论 | null | ResponseVO |
| listCommentByPage | public | 获取所有评论+分页 | int page,int size | ResponseVO |
| saveComment | public | 新增评论 | RequestVO | ResponseVO |
| removeCommentById | public | 根据id删除评论 | int id | ResponseVO |
| updateComment | public | 修改评论 | RequestVO | ResponseVO |



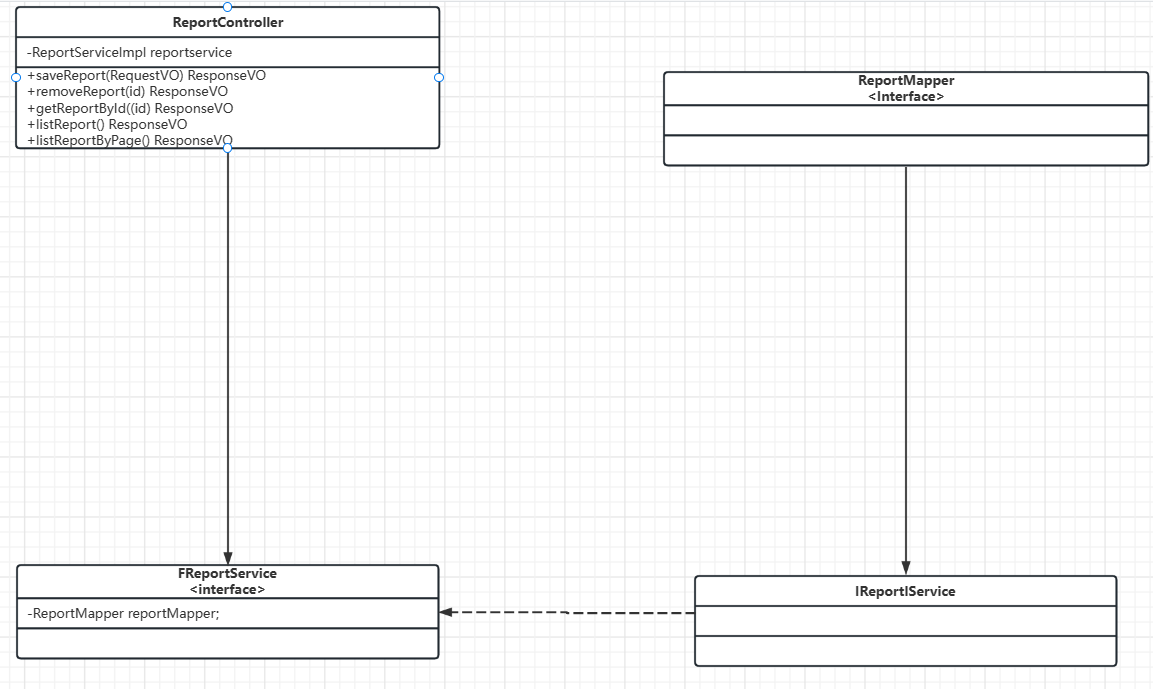
2.4 FeedbackController类设计

| **方法名** | **等级** | **功能描述** | **参数** | **返回值** |
| --- | --- | --- | --- | --- |
| getFeedBackById | public | 根据id获取单条反馈 | int id | ResponseVO |
| listFeedBack | public | 获取所有反馈 | null | ResponseVO |
| listFeedBackByPage | public | 获取所有反馈+分页 | int page, int  size | ResponseVO |
| saveFeedBack | public | 增加反馈 | RequestVO | ResponseVO |
| removeFeedBackById | public | 根据id删除反馈 | int id | ResponseVO |



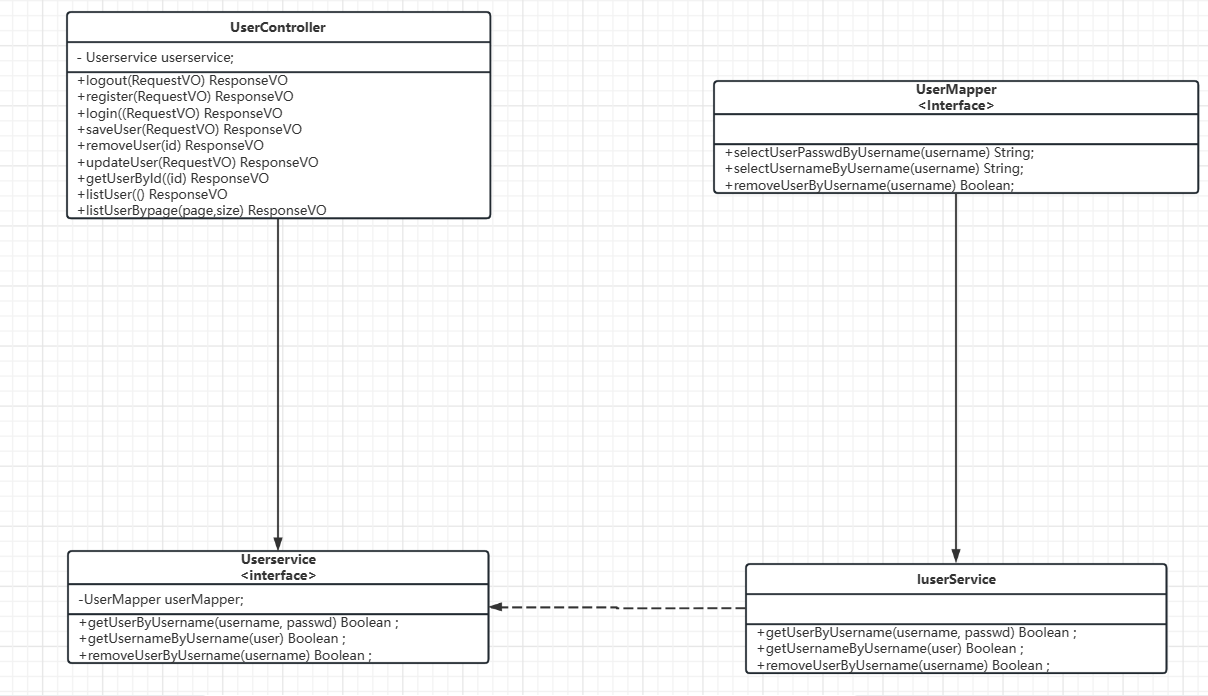
2.5 ReportController类设计

| **方法名** | **等级** | **功能描述** | **参数** | **返回值** |
| --- | --- | --- | --- | --- |
| getReportById | public | 根据id获取单个举报 | int id | ResponseVO |
| listReport | public | 获取所有举报 | null | ResponseVO |
| listReportByPage | public | 获取所有举报+分页 | int page, int  size | ResponseVO |
| saveReport | public | 新增举报 | RequestVO | ResponseVO |
| removeReportById | public | 根据id删除举报 | int id | ResponseVO |



2.6 UserController类设计

| **方法名** | **等级** | **功能描述** | **参数** | **返回值** |
| --- | --- | --- | --- | --- |
| getUserById | public | 根据id获取单个用户 | int id | ResponseVO |
| listUser | public | 获取所有用户信息 | null | ResponseVO |
| listUserByPage | public | 获取所有用户信息+分页 | int page, int  size | ResponseVO |
| saveUser | public | 增加用户 | RequestVO | ResponseVO |
| removeUserById | public | 根据id删除用户 | int id | ResponseVO |
| updateUser | public | 修改用户信息 | RequestVO | ResponseVO |
| login | public | 登录 | RequestVO | ResponseVO |
| register | public | 注册 | RequestVO | ResponseVO |
| loginout | public | 注销 | RequestVO | ResponseVO |



1. UI设计

略，详见UI设计文档

1. 数据库设计

略，详见数据库设计文档

1. 接口设计

略，详见接口设计文档