存档日期：_____ 存档编号：_____

南昌航空大學

NanChang Hangkong University

本科毕业设计

UNDERGRADUATE DESIGN

设　计　题　目： 基于多平台应用的音乐共享社区设计与实现

姓　　名： 王欣萌

专　　业： 软件工程(东软班-商业智能)

班　级、学　号: 162031 16203133

指　导　教　师： 吕文艳

毕业设计任务书

一、本课题目标：

随着互联网娱乐项目的日益增多，内容也日渐丰富，加之网络便利性的增强，越来越多的用户喜欢在网上听音乐。但是各平台音乐资源残次不齐，也包含了许多假无损音乐，无法满足发烧友们的需求，建立以用户为本的社交型音乐分享平台则可以弥补这一缺陷，这就是本平台搭建的初衷。发烧友用户们非常需要一个跨平台的，并且拥有智能推荐、各端体验一致的音乐分享平台。

本题旨在实现一个开放的跨平台分享系统，音乐则是主要分享对象，所有音乐数据均由用户自行上传，同时还可以发送简单的动态来进行音乐交流，为此初衷为目的制作一个多平台的音乐分享 APP。前端部分采用由 Google 自主开发的基于 Dart 语言的跨平台应用 SDK Flutter，同时兼容 iOS 和 Android 两端。后端服务器采用基于 Java EE 的 Spring Boot 框架搭建，ORM 层使用 Spring Data JPA，数据库使用 MySQL，使用 Shiro 作为权限管理，并且打包为 Docker 镜像方便部署和升级。

二、本课题的任务和要求：

1.课题任务

根据分析本系统 10 个模块 主要分为两个角色：浏览用户、管理员

对于浏览用户，有以下 6 个模块：

（1）动态分享模块

站内分享

可将歌单及歌曲分享到论坛中的各个分区以及自己的个人动态中，站内用户打开后会打开相对应的的歌单或歌曲

站外分享

可将歌曲分享到微信 QQ 等站外软件，用户打开后会跳转到 app 内查看

（2）上传模块

主要用来处理静态资源的上传，包括图片，音乐，和歌词文件

（3）下载模块

主要用来处理静态资源的下载，包括图片，音乐，和歌词文件，并且可以通过用户的权限，拦截部分静态资源的访问

（4）音乐播放模块

用来播放收藏的音乐或歌单，可以拖动进度条，显示专辑封面，如果有歌词可以显示歌词

（5）歌单模块

用户可以从现有曲库中分享歌单给所有用户,并且可以自行对歌单中的歌曲进行修改

（6）评论模块

对分享的音乐进行评论，进行留言和回复，可以收藏和分享自己喜欢的歌单，或者评论歌单

（7）搜索模块

对所有用户上传的音乐资源进行搜索

对于管理员，有以下 3 个模块：

（1）评论审核模块

对用户举报的评论进行审核，如果存在违规行为，管理员进行删除

（2）音乐审核模块

对被举报的的音乐进行审核，如存在违规信息，管理员可以删除

（3）公告模块

在公告栏中发送公告

2.课题要求

本题旨在实现一个开放的跨平台分享系统，音乐则是主要分享对象，所有音乐数据均由用户自行上传，同时还可以发送简单的动态来进行音乐交流，为此初衷为目的制作一个多平台的音乐分享 APP。前端部分采用由 Google 自主开发的基于 Dart 语言的跨平台应用 SDK Flutter，同时兼容 iOS 和 Android 两端。后端服务器采用基于 Java EE 的 Spring Boot 框架搭建，ORM 层使用 Spring Data JPA，数据库使用 MySQL，使用 Shiro 作为权限管理，并且打包为 Docker 镜像方便部署和升级。

App 使用 Flutter 框架进行搭建，Flutter 是一套跨平台的移动 UI 框架，可以快速在 iOS、Android 以及 Fuchsia 上构建高质量的原生用户界面。在 Flutter 中，一切都是部件(Widget) ，所有的界面也是完全使用 Dart 代码进行编写，并且支持热加载。

服务器使用 SpringBoot 进行搭建，是现在应用最广泛的 Java 服务器框架，具有良好的稳定性、扩展性，拥有非常庞大的第三方库。本项目完全遵守 Restful

风格进行 API 开发，通过 Java Doc 自动生成 API 文档，并打包成 Docker 镜像方便部署以及升级。

三、本课题工作内容及进度安排：

起止时间	任务内容
2019 年	
11 月 15 日～ 12 月 8	完成课题选题、指导教师沟通，任务书学
12 月 9 日～ 12 月 30	查阅学习课题相关文献资料，完成外文翻
2020 年	
1 月 1 日～ 2 月 28 日	进行项目设计、编码、调试。参加中期检
3 月 1 日～ 4 月 30 日	进行项目功能完善，完成设计成果验收。
5 月 1 日～ 5 月 30 日	完成毕业设计说明书撰写、修改、定稿，通
6 月 1 日～ 6 月 6 日	参加毕业设计答辩

四、主要参考文献：

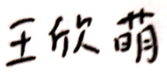
[1] 谭青. 基于用户评论的音乐推荐系统的研究[D].安徽理工大学,2018.
[2] 邓皓瀚.基于 Flutter 的跨平台移动 APP 开发前景研究[J].信息与电脑(理论版),2019(15):197-199.
[3] 王阅蓁. 移动应用的 web 与 native 混合编程模式研究与实现[D].电子科技大学,2015.
[4] 李一然. 基于 React Native 架构的客户端开发与实现[D].北京邮电大学,2019.
[5] Payne R. Developing in Flutter[M]//Beginning App Development with Flutter. Apress, Berkeley, CA, 2019: 9-27.
[6] Guangmang Cui,Xiaojie Ye,Jufeng Zhao,Liyao Zhu,Ying Chen. Multi-frame motion deblurring using coded exposure imaging with complementary fluttering sequences[J]. Optics and Laser Technology,2020,126.
[7] 陈思,冷雪.微信小程序开发方式对比[J].电子制作,2020(02):52-53+22.
[8] React Native 官网 [EB/OL]. https://facebook.github.io/react-native/, 2020-5-16.
[9] Flutter 官网 [EB/OL]. https://flutter.dev/, 2020-5-16.
[10] Django 官网 [EB/OL]. https://www.djangoproject.com/, 2020-5-16.
[11] Spring 官网 [EB/OL]. https://spring.io/, 2020-5-16.
[12] 庄学松,张智,黄可望.基于 SpringBoot 的短信服务的设计与实现[J].无锡职业技术学院学报,2020,19(02):41-44.
[13] 周虎.一种基于 JWT 认证 token 刷新机制研究[J].软件工程,2019,22(12):18-20.
[14] 王杉文.基于 SpringBoot+Shiro 的权限管理实现[J].电脑编程技巧与维护,2019(09):160-161+173.
[15] Django 官网 [EB/OL]. https://www.djangoproject.com/, 2020-5-16.
[16] 请叫我的全名 cv 工程师.spring 上传文件和下载文件[EB/OL].https://www.jianshu.com/p/cba3e0706849,2020-03-08.
[17] Smith-Cruise.Shiro 基于 SpringBoot +JWT 搭建简单的 restful 服务[EB/OL].https://github.com/Smith-Cruise/Spring-Boot-Shiro,2020-02-29.
[18] 软件测试方法和技术[M]. 清华大学出版社有限公司, 2005.

专业教学承办单位意见：

专业教学负责人（签名）：

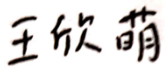
学士学位设计原创性声明

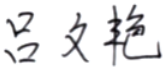
本人声明，所呈交的设计是本人在导师的指导下独立完成的研究成果。除了文中特别加以标注引用的内容外，本设计不包含法律意义上已属于他人的任何形式的研究成果，也不包含本人已用于其他学位申请的论文或成果。对本文的研究作出重要贡献的个人和集体，均已在文中以明确方式表明。本人完全意识到本声明的法律后果由本人承担。

作者签名：  日期：2020 年 6 月 12 日

学位设计版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位设计的规定，同意学校保留并向国家有关部门或机构送交设计的复印件和电子版，允许设计被查阅和借阅。本人授权南昌航空大学可以将本设计的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位设计。

作者签名：  日期：2020 年 6 月 12 日

导师签名：  日期：2020 年 6 月 12 日

摘要

随着互联网娱乐项目的日益增多，内容也日渐丰富，加之网络便利性的增强，越来越多的用户喜欢在网上听音乐。但是各平台音乐资源残次不齐，也包含了许多假无损音乐，无法满足发烧友们的需求，建立以用户为本的社交型音乐分享平台则可以弥补这一缺陷，这就是本平台搭建的初衷。发烧友用户们非常需要一个跨平台的，并且拥有智能推荐、各端体验一致的音乐分享平台。

本题旨在实现一个开放的多平台音乐分享系统，并以音乐为分享目标，同时给用户提供简单的交流平台，所有音乐信息均由注册用户提供，后交由管理员进行验证，为此制作了一个移动端的音乐分享社区 App。

本平台 APP 使用 Google 最新的 Flutter 框架进行搭建，兼容 Android 和 iOS 两端。服务器使用 Spring Boot 框架搭建，数据库使用 MySQL，使用 Shiro 作为权限管理，并且打包为 Docker 镜像方便部署和升级。基于多平台应用的音乐共享社区前端部分主要基于 Flutter 框架进行开发，并实现近似于原生效率的跨平台应用。同时后端的推荐系统会对登录用户的历史听歌记录及收藏的歌曲与歌单进行分析，推荐用户可能喜欢的歌曲。如样本数据过少则会推荐当日热门歌曲。

关键词：Flutter　跨平台　智能推荐　Spring Boot　Docker

Abstract

With the increasing number of Internet entertainment projects, the content is becoming richer, and the convenience of the network has been enhanced. More and more users like to listen to music on the Internet. However, the music resources of various platforms are incomplete, and it also contains many fake non-destructive music, which cannot meet the needs of enthusiasts. Establishing a user-oriented social music sharing platform can make up for this shortcoming. This is the original intention of this platform. Enthusiast users are in great need of a cross-platform music sharing platform with smart recommendations and consistent experience on all ends.

This question aims to achieve an open multi-platform music sharing system, with music as the sharing target, and at the same time provide users with a simple communication platform. All music information is provided by registered users and then submitted to the administrator for verification. A mobile music sharing community app.

This platform APP is built using Google's latest Flutter framework and is compatible with both Android and iOS. The server is built using the Spring Boot framework, the database uses MySQL, Shiro is used as the permission management, and it is packaged as a Docker image for easy deployment and upgrade. The front-end part of the music sharing community based on multi-platform applications is mainly developed based on the Flutter framework and implements cross-platform applications that are similar to native efficiency. At the same time, the back-end recommendation system analyzes the logged-in user's historical listening records and collected songs and song lists, and recommends songs that the user may like. If the sample data is too small, popular songs of the day will be recommended.

Key Words： Flutter Cross-platform smart recommendation Spring Boot Docker

目 录

图清单

表清单

1 绪论

1.1 课题依据及意义

随着互联网娱乐项目的日益增多，内容也日渐丰富，加之网络便利性的增强，越来越多的用户喜欢在网上听音乐。但是各平台音乐资源残次不齐，也包含了许多假无损音乐，无法满足发烧友们的需求，建立以用户为本的社交型音乐分享平台则可以弥补这一缺陷，这就是本平台搭建的初衷。发烧友用户们非常需要一个跨平台的，并且拥有智能推荐、各端体验一致的音乐分享平台[0]。

现有很多跨平台应用，都存在用户体验不一致、性能低下、代码不能多次复用等问题，例如 React Native、Uni-App 等实现方案，他们都是基于 JavaScript 语言实现的跨平台开发框架，多少都需要与原生 API 进行通信，无法实现一处编写处处运行。而由 Google 牵头的 Flutter 采用独立的 Dart 语言进行开发，Flutter 从底层实现了一套自己的框架，同时与原生代码高度契合，全部使用 Widget 进行开发，开发者只需开发 Widget 即可，完成了接近原生的操作体验。

1.2 国内外研究现状

移动端跨平台开发从最初的 Hybrid 到 React-Native，再到最近 Google 新推出的 Flutter 移动 UI 框架，体验和性能越来越接近原生应用[2]。

1.2.1 国外研究现状

(1) Hybrid

以往最早的以 Cordova 为代表的 Hybrid 开发[3]，主要依赖于 WebView。但是 WebView 是一个很重的控件，容易造成内存泄露，而且有些复杂的用户界面会降低 WebView 的显示性能。这是因为 JavaScript 与原生 Native 代码之间需要使用 JSBridge 进行通信，同时还要进行上下文切换，所以性能会因此降低。

(2) React Native

React Native 技术抛开了 WebView，利用 JavaScriptCore 来做桥接[4]，将 JS 调用转为 native 调用，在跨平台开发中只牺牲少量性能，这是一个巨大的进步。但因为仍然存在从 JS 代码转换为本地代码的过程，所以如果界面 UI 操作频繁，则可能导致性能问题。

(3) Flutter

Flutter 对跨平台的实现采用更全面的方案[5]。与采用 WebView 或 JavaScript Core 不同，它不采用 JavaScript Framework，而是实现了自己的 UI 框架，然后在系统的底层绘制系统的 UI 所以他采用的开发语言不是 JS，而是 Dart。据称 Dart 语言可以编成原生代码。

(4) 总结

Hybrid 基于 WebView 容器内部运行的 HTML 和 JS，通过 Cordova 提供的接口与硬件进行通信；因此，它的路径天生就是有限的，并且受不同厂商对 Webkit 内核支持的影响。

由于将 View 编译成本地 View, React Native 的效率大大高于基于 Cordova 的 HTML5，但它仍然存在效率问题。React Native 的渲染机制基于前端框架的考虑，而复杂的 UI 渲染则需要依赖于多种视图叠加。例如，我们呈现了一个复杂的 ListView，每个小控件都是一个静态视图，然后它们互相叠加。考虑一下现在如果我们的 list 需要滑动刷新，那么需要渲染的对象数量是多少，所以它的列表模式并不友好。

Flutter 吸取了前两种方法的教训，并在绘制技术上选择了自己的实现(GDI)，由于更易控制，使用了新的语言 Dart，从而避免了 React Native 通过桥接器与 JavaScript 进行通信带来的效率低下问题，因此在性能方面要优于 React Native [6]。在安卓手机开发者选项中打开显示界面，你会发现 Flutter 的布局是一个整体，表明 Flutter 的渲染没有使用本地控件。

1.2.2 国内研究现状

(1) uni-app

这个框架使用 Vue. js 来开发所有前端应用，开发者编写了一套可以发布到 iOS, Android,H5 的代码，以及各种小程序的代码[7] (包括微信/支付宝/百度/头条/QQ/钉钉/淘宝)，一些平台，如快应用。

(2) 总结

uni-app 遵循的是 Vue. js 的语法规范，基于模板和数据绑定，更加现代化的开发方法，比纯 H5+的代码更少，开发效率更高。

但是 uni-app 不支持 DOM，因为 DOM 受到限制，采用 Vue 模式可以获得更高的性能体验。大量的 web 库不可用，基于 DOM 的库也不能直接使用。

国内还有其他类似的跨平台框架，但大多为了适配小程序，原生开发效率较低。

1.3 论文的主要工作

本题旨在实现一个开放的跨平台分享系统，音乐则是主要分享对象，所有音乐数据均由用户自行上传，同时还可以发送简单的动态来进行音乐交流，为此初衷为目的制作一个多平台的音乐分享 APP。前端部分采用由 Google 自主开发的基于 Dart 语言的跨平台应用 SDK Flutter，同时兼容 iOS 和 Android 两端。后端服务器采用基于 Java EE 的 Spring Boot 框架搭建，ORM 层使用 Spring Data JPA，数据库使用 MySQL，使用 Shiro 作为权限管理，并且打包为 Docker 镜像方便部署和升级。

1.4 论文的组织结构

本文主要基于多平台应用的音乐共享社区 APP 进行简要叙述，文章分为七章节。具体如下：

第一章，绪论。本章节针对本课题在国内外的发展现状、研究目的和意义进行了讨论。

第二章，系统分析。本章节重点分析了共享平台系统的可行性、需求，并对方案选择进行了概括和分享。

第三章，系统的设计。本章节主要分析了系统框架的设计、数据库的设计，重点讲解了系统的主要流程。

第四章，系统的实现。本章节主要展示了实现系统初衷设计的核心代码，重点分析了部分模块的算法。

第五章，系统运行与效果分析。本章节主要对系统的 UI 外观进行了展示，并针对各项功能一一对应解释。

第六章，系统测试。本章节主要使用实例的方式重点解说了测试方案、测试方法等事项。

第七章，总结与展望。本章节主要列示了本系统的不足点，提出了差异性优化方案，并对项目的可拓展性和发展方向给出了展望。

1.5 本章小结

本章节一是对本课题研究意义和目的进行了展示，二是对当今多平台应用框架在国内外的发展进行了描述，三是对论文的整体架构予以总结，重点内容加以阐述。

2 系统分析

基于多平台应用的音乐共享社区前端部分主要基于 Flutter 框架进行开发，并实现近似于原生效率的跨平台应用。同时后端的推荐系统会对登录用户的历史听歌记录及收藏的歌曲与歌单进行分析，推荐用户可能喜欢的歌曲。如样本数据过少则会推荐当日热门歌曲。

2.1 可行性分析

本章节对多平台应用的音乐共享社区的可行性进行了分析和总结，并就其原因进行了详尽阐述。

2.1.1 技术可行性

本系统采用了全栈式的开发方式。系统分为前端的 App 以及后端的服务端。前端 App 使用 Google 开发的跨平台框架 Flutter 进行搭建，兼容 Android 和 iOS 平台，并在未来有望兼容桌面端。服务器使用 Spring Boot + Spring Data JPA + Shiro 权限管理框架搭建，通过 Thymeleaf 模板动态生成后台管理面版，并且系统会定时进行推荐。

此系统使用 Android Studio 和 IntelliJ IDEA 开发工具在 Mac OS 下实施开发，使用 Android Simulator 和 iOS simulator 开展调试，Android 端采用一加 8pro 手、iOS 采用 iPhone11 机型真机调试。通过精密分析，本系统应用结构清楚，使用架框成熟，满足开发环境的要求，故此在技术上可行。

2.1.2 经济可行性

本系统是封闭式资源分享系统，使用开源框架作为技术框架，开发软件均采用 EDU 授权版本，无额外付费软件，同时对硬件环境要求不高，系统开发周期适中，系统整体的开发成本较低，系统稳定性良好且易于维护管理，由于采用了 Spring Boot 进行开发，使项目拓展性大大提高。从经济效益方面讲，本系统仅针对注册用户提供音乐分享，作为交流平台，不存在版权问题。同时，系统还预留了交易接口，今后有望推出付费交流服务，系统可以靠手续费进行营利。综上所述，系统的经济效益大于成本，可获创利润可观，故此在经济上可行。

2.1.3 对环境及可持续性发展的影响

本系统采用前后端分离的方法进行开发，使用 Spring Boot 框架来管理 Java Bean 的生命周期，相对与普通的 MVC 框架比减少了对内存的开支。同时在推荐系统上采用了高效的推荐算法，无需进行大量复杂的运算即可实现可观的效果。同时，本项目还打包成了 Docker 容器，无需复杂的硬件配置，和繁琐的人工配置环境环节，系统即可简单的部署在云主机上，对计算机和人力资源消耗很小。

同时，本系统采用 Flutter 进行开发，Flutter 在渲染技术上，选择了自己实现的（GDI），由于有更好的可控性，使用了新的语言 Dart，避免了 React Native 的那种通过桥接器与 JavaScript 通讯导致效率低下的问题，所以在性能方面比 React Native 更高一筹。可以有效的解决在性能低下设备上的卡顿，并节省开发时间。

综上，本系统对环境影响很小，并具有可持续发展的可行性。

2.2 需求分析

经过一定阶段的市场调研，收集并分析了重要的信息，对本系统的功能和性能上的需求予以确定。

2.2.1 需求描述

本题旨在实现一个开放的多平台音乐分享系统，并以音乐为分享目标，同时给用户提供简单的交流平台，所有音乐信息均由注册用户提供，后交给管理员进行验证，为此制作了一个移动端的音乐分享社区 App。

根据分析本系统 10 个模块 主要分为两个角色：浏览用户、管理员

对于浏览用户，有以下 6 个模块：

（1）动态分享模块

站内分享

可将歌单及歌曲分享到论坛中的各个分区以及自己的个人动态中，站内用户打开后会打开相对应的的歌单或歌曲

站外分享

可将歌曲分享到微信 QQ 等站外软件，用户打开后会跳转到 app 内查看

（2）上传模块

主要用来处理静态资源的上传，包括图片，音乐，和歌词文件

（3）下载模块

主要用来处理静态资源的下载，包括图片，音乐，和歌词文件，并且可以通过用户的权限，拦截部分静态资源的访问

（4）音乐播放模块

用来播放收藏的音乐或歌单，可以拖动进度条，显示专辑封面，如果有歌词可以显示歌词

（5）歌单模块

用户可以从现有曲库中分享歌单给所有用户,并且可以自行对歌单中的歌曲进行修改

（6）评论模块

对分享的音乐进行评论，进行留言和回复，可以收藏和分享自己喜欢的歌单，或者评论歌单

（7）搜索模块

对所有用户上传的音乐资源进行搜索

对于管理员，有以下 3 个模块：

（1）评论审核模块

对用户举报的评论进行审核，如果存在违规行为，管理员进行删除

（2）音乐审核模块

对被举报的的音乐进行审核，如存在违规信息，管理员可以删除

（3）公告模块

在公告栏中发送公告

2.2.2 角色和用例分析建模

（1）用例分析

图 2-1 为系统用例图。系统分为用户和管理员 2 种种角色，因为是封闭式论坛，避免出现版权纠纷，本软件只开放给注册用户。

注册用户可以查看其他用户发送的动态，歌单，音乐等。系统会根据用户的听歌记录来推荐用户可能个喜欢的歌曲，推荐系统仅记录用户的听歌喜好，不会记录敏感数据。用户还可以举报他们认为的违规动态或评论，交给管理员来鉴别。

用户还可以自由的在动态里添加他们想要的元素，例如歌单，音乐，图片等。同时还可以进行搜索，来查找他们喜欢歌曲与歌单，并将它们添加到喜爱列表中。

管理员可以管理所有用户的状态，可以针对性的禁言某一用户。还可以根据用户举报的记录来删除特定的动态和评论

推荐系统则会根据用户平时听歌喜好来推荐贴近用户听歌习惯的音乐或歌单，而所有的用户的听歌记录均做了脱敏处理。

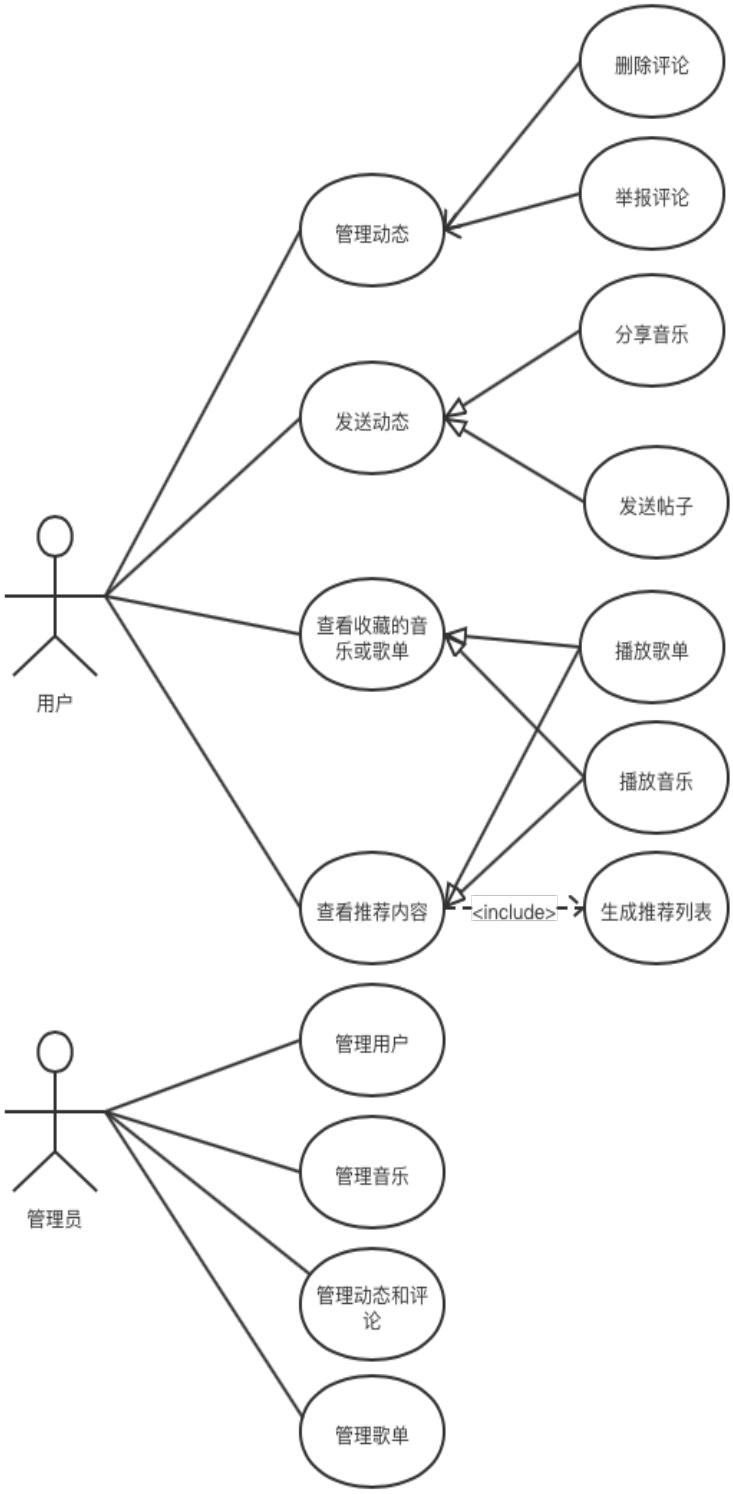


图 2-1 系统用例图

（2）用例描述

表 2-1 动态用例描述

用例名称	动态
描述	用户发送动态
标识符	UC1
角色	注册用户
前置事件流	用户登录
主事件流	1. 注册用户登录账号 2. 用户从首页进入动态页面 3. 用户点击发送按钮 4. 弹出动态发送界面 5. 用户点击发送，提示发送成功 6. 用例结束
其他事件流	1. 网络异常，提示发送失败
后置事件流	1. 对于发送的内容，用户可以在我的页面进行查看

表 2-2 用户注册用例描述

用例名称	用户注册
描述	用户进行注册
标识符	UC2
角色	任何人
前置事件流	无
主事件流	1. 用户点击注册 2. 输入电话号码 3. 等待验证码 4. 输入其他信息 5. 注册完成 6. 用例结束
其他事件流	其他事件流： 1. 点击发送验证码按钮，返回验证码
后置事件流	后置事件流： 1. 用户可以登录

表 2-3 上传音乐用例描述

用例名称	上传音乐
描述	用户上传音乐
标识符	UC3
角色	用户
前置事件流	用户登录且上传音乐
主事件流	1. 用户点击上传音乐 2. 用户根据提示填写上传音乐信息 3. 用户上传音乐 4. 用例结束
其他事件流	其他事件流： 1. 上传的音乐同时发送动态
后置事件流	后置事件流：无

2.2.3 系统非功能需求

（1）性能需求

在用户不多，多数网络请求可在 500ms 内完成响应，大量用户访问时，网络请求可在 2000ms 内完成响应。

（2）兼容性需求

前端 APP 支持 iOS 和 Android 系统，并且应对旧版本系统进行一定的适配。

（3）交互性需求

系统设计和配色均遵循 Metal Design，后端应对大多异常操作进行过处理，前端在网络请求失败，操作失败时应能正确进行错误提示，在进行敏感操作时弹出提示框作为警示，在数据异常时进行一定的展示优化。

2.3 方案比选

目前移动互联网全面普及，近几年发展尤其迅速，在移动终端的开发工具链日渐成熟，相比之前被原生应用通知的移动市场，在大前端时代，原生开发已经失去了昔日的优势，移动 APP 已经被各种跨平台的前端框架所主导。目前大厂用的比较多的前端跨平台框架有 React Native 和 Flutter，后端主流的框架有 Python 的 Django 和 Flask 框架，Java 的 Spring Boot 框架。三者各有其特点。本部分对 React Native + Django 框架和 Flutter + Spring Boot 框架两种方案进行比选。

2.3.1 方案一：React Native + Django

（1）React Native 框架

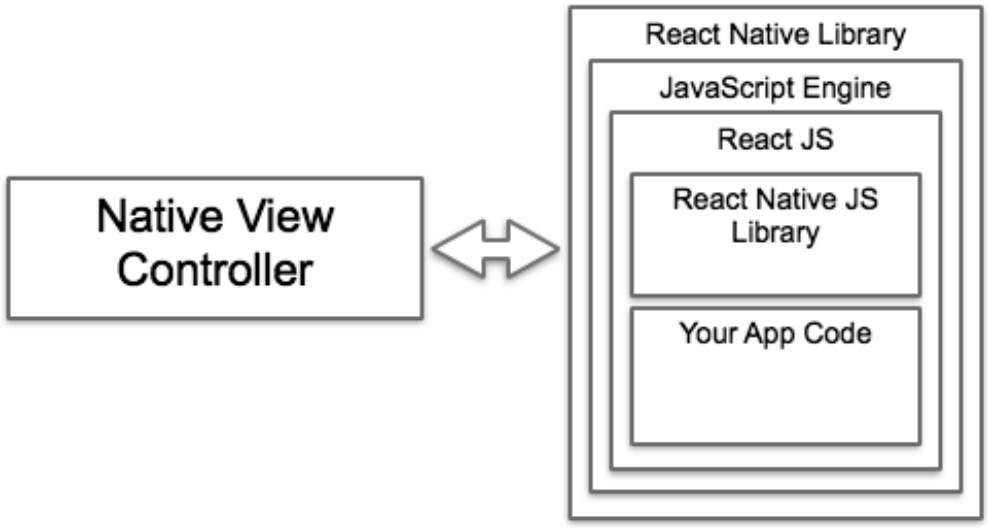


图 2-2 React Native 框架结构图

React Native 是 Facebook 在 2015 年发布的跨平台框架，支持 iOS 和 Android 两种系统。以 React 为基础，他完全继承了 React 组件开发的特点，提高了代码重用率[8]，各个平台功能代码都可以重用，官方数据显示， iOS 和 android 功能代码重用率可达 90%以上。不用 WebView，就可以完全避免交互和性能问题。应用热更新可实现 app 功能升级和问题修复，提高 app 迭代速度和开发效率。由于基于 JS 框架 React，前端程序员不需要太多学习成本即可轻松上手，是目前使用较为广泛的跨平台解决方案。由于 React 社区活跃，开源轮子非常多，很多互联网巨头选择使用 React Native 作为跨平台的选择方案，国外比较著名的案例有 Facebook、Instagram、Airbnb 等，国内比较著名的有京东、QQ、携程。图 2-2 展示了 React Native 的框架结构，从图中可知，内置的 JavaScript 引擎通过编译所写的 JavaScript 代码，并调用原生的视图组件，达到应用构建的效果。

（2）Django 框架

Django 是 Python 编写的 Web 应用框架，它采用开放源代码。Django 最初是为新闻类站点而设计的，它有快速的开发需求，目的是实现简单、快捷的网站开发。

经过十几年的发展和完善， Django 拥有完善的功能和要素，并且在 Django 的 Model 层上自带数据库 ORM 组件，因此，开发人员无需学习其他数据库访问技术。Django 通过正则表达式灵活地管理 URL 映射，并具有丰富的模板语言。除此之外，自带自由后台管理系统，只需简单的几行配置和代码即可实现一个完整的后台数据管理平台。

在 Django 中， Python 的程序开发人员只需编写少量代码就能轻松地完成正式网站所需的大部分工作，并进一步开发全功能 Web 服务 Django 本身是基

于 MVC 模型的， MVC 模式简化了后续程序的修改和扩展，使程序的某个部分得以重复利用[10]。

图 2-3 展示了 Django 工作流程。所有的 HTTP 请求均会被 URLS 模块捕获，并映射到相应的视图，在视图层对模型进行读和写的操作，并响应至前端页面。

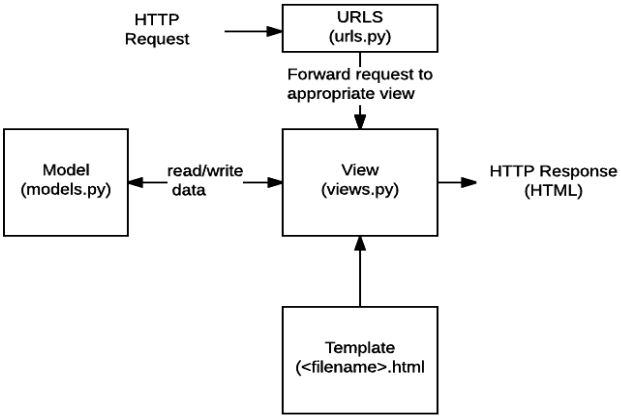


图 2-3 Django 工作流程图

2.3.2 方案二：Flutter + SringBoot

（1）Flutter 框架

Google 开放源码移动应用程序 SDK Flutter，一个代码就能同时生成两个高性能、高保真的应用程序 iOS 和 Android[9]。通过 Dart 语言开发的代码，可以在 iOS, Android 等平台上运行， Web 端现在也可以在 dev 分支上运行， Flutter 团队也宣布不久将支持桌面端。

因为 Flutter 选择 Dart 作为其开发语言，所以 Dart 可以由 AOT (Ahead Of Time)或 JIT (Just In Time)编译，它的 JIT 编译特性使得 Flutter 可以在开发阶段达到亚秒级别的有状态热重载，从而大大提高开发效率。

接口绘制方面，采用自带的高性能渲染引擎 (Skia)自绘制，渲染速度和用户体验都不亚于本机。

Flutter 还内置了 Material Design 和 Cupertino 风格的 widget (iOS 风格)，开发人员可以根据不同的平台需求，快速构建出符合相应平台风格的良好用户界面。

在图 2-4 中， Flutter 正式给出了一个系统框架，我们可以看到， Flutter 框架分为三层：框架层、引擎层和 Embedder 层。

框架层是由 Dart 实现的，它包括了许多 Android Material 风格和 iOS Cupertino 风格的 Widgets 小部件，以及渲染、动画、绘图和手势等等。

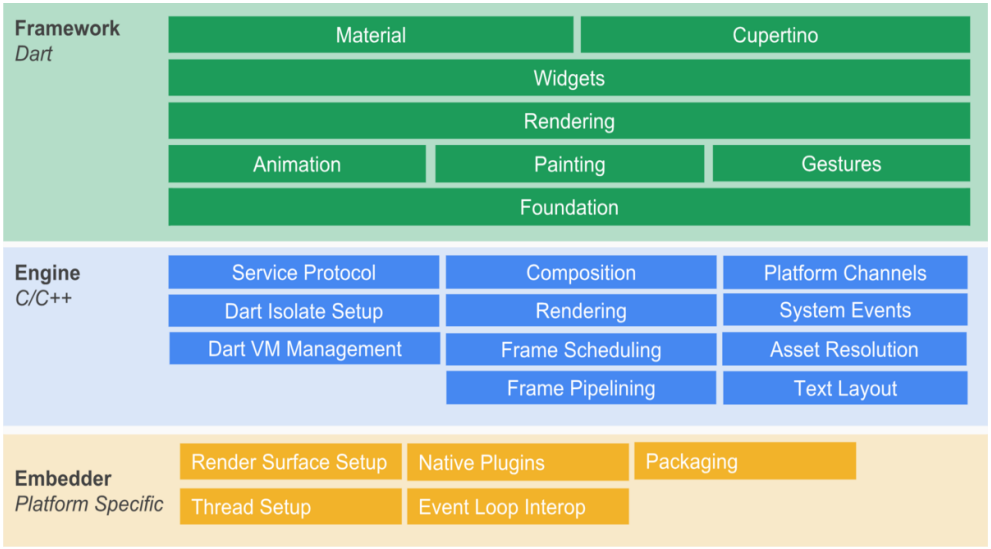


图 2-4 Flutter 框架结构图

使用 C/C++实现的 Engine 层是 Flutter 的核心引擎，主要包括 Skia 图形引擎， Dart 运行时环境 DartVM，文本渲染引擎等等。

Embedder 层主要处理与平台相关的一些操作，比如 Surface 渲染设置，本地插件，打包，线程设置等等。

（2）Spring Boot 框架

SpringBoot 是基于 Spring 开发的所有项目的起始点[11]。与名字一样，SpringBoot 的目的是简化之前 Spring 框架繁琐的配置 xml 过程，采用了惯例大于配置的概念，这与 Mavan 有着异曲同工之处。Spring Boot 官方提供了多种 start 包，如 Spring-web-starter 就包含了 Spring web 开发所需要的 Spring MVC 包

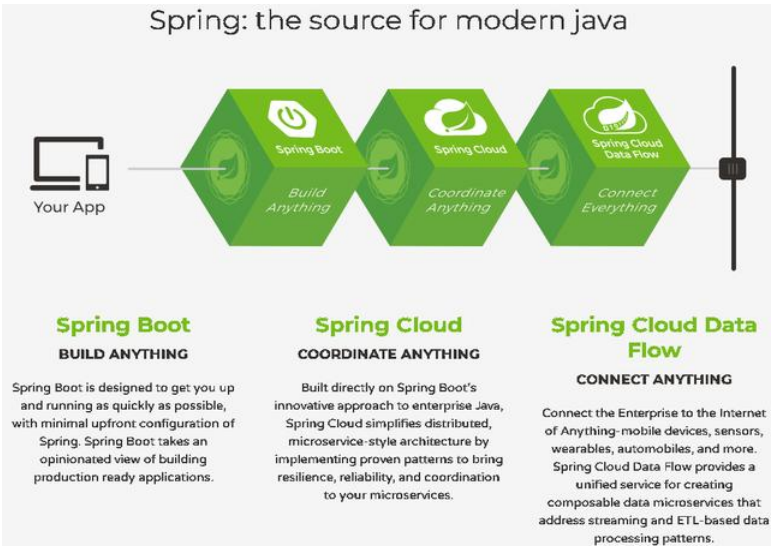


图 2-5 Spring Boot 工作流程图

和其他一些必备的开源包。这就是 Spring 的封装版本，因此理所当然地继承了 Spring 特有的特性，如 IoC (控制反转), DI (依赖注入), AOP (面向切面编程)。

2.3.3 比选结论

（1）Flutter 与 Native 对比

表 2-4 详细的展示了 Flutter 与 React Native 在性能、成熟度等特性方面的对比。

表 2-4 Flutter 与 React Native 对比表

特性	Flutter	React Native
发布时间	2017	2015
语言	Dart	JavaScript
UI 组件	自己实现的 Widget	对应系统的 Native 组件
运行速度	快	中等
热重载	支持	支持
文档	完整	较完整
配置	简单	较复杂
成熟度	较、成熟	成熟
支持平台	iOS、Android、Web 等	iOS 和 Android

（2）Django 与 Spring Boot 对比

表 2-5 详细的展示了 Django 与 Spring Boot 在性能、拓展性等方面的对比。

表 2-5 Django 与 Spring Boot 对比表

	Django	SpringBoot
性能	高	高
可拓展性	强	强
ORM	自带	高
数据库管理	自带	高
插件	丰富	非常丰富
文档	完整	非常完整
配置	简单	一般

（3）结论

表 2-4 针对 Flutter 与 Native 在语言、性能、开发效率和组件等多方面进行了对比。可以得出， Flutter 框架虽然较新，但相对 React Native 并不成熟，但 Flutter 的组件丰富，能够专注于核心业务的开发，同时配置简单，官方文档丰富，支持 Web 甚至是桌面端，并且在性能方面相对 React Native 有较大优势。因此本系统选择 Flutter 作为客户端框架。

表 2-5 针对 Django 和 Spring Boot 在性能、可拓展性和插件等方面进行了对比。虽然 Django 的性能基本与 Spring Boot 相同，但 Spring Boot 具有极高的可拓展性，且各种解决方案非常成熟，有着完整的技术栈和生态圈，熟悉 Java 便于上手。综上所述，本系统选择 Flutter + Spring Boot 框架。

2.4 本章小结

本章节主要阐述了系统的可行性分析与需求性分析，并就系统选用的框架展开对比分析，确定了系统设计方案。

3 系统的设计

3.1 软件体系结构

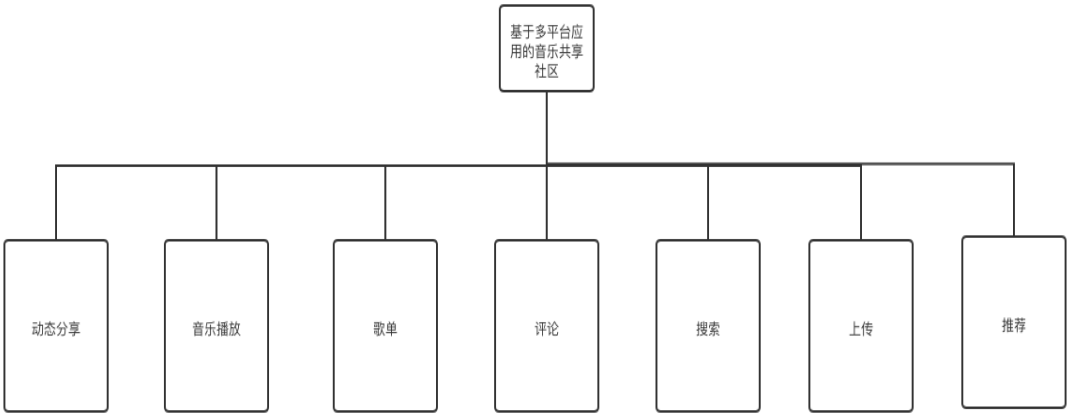


图 3-1 系统功能结构图

图 3-1 对本系统的功能结构进行了展示，把业务进行模块化的拆分之后，各个模块之前的强依赖关系没有了，每个模块实现的业务相互独立了出来，实现对系统功能点的一个解耦。这样每个模块如果要增添新的功能点，无需对其整个系统进行改动，只要修改特定模块接口的实现类即可。

本系统主要由用户动态分享、音乐播放、歌单、评论、搜索、积分、推荐等 6 大模块组成。其中本系统的核心模块推荐模块，只针对注册用户，如果用户之前没有听过任何歌曲，或者用户听取的歌曲样本数量过少，推荐页面会显示系统每天自动生成的公共推荐列表，所有推荐的歌曲和歌单用户均可点击图标进行查看。

3.2 功能设计

（1）类图设计

图 3-2 主要对本系统实体类的属性以及实体类之间的关系进行了展示，系与系统核心功能动态分享和音乐分享相关的实体类有 5 个，分别是 User、Role、Reply 以及 Post，其中用户与动态 Post 为多对多的关系，用户可发送多个动态。每个上传的资源都会视为一个动态，便于管理。

同时，对与音乐线管的各种静态资源也进行了实例化，如图 3-3 将音乐信息分为 Song、Singer、Album、Tag、SongComment 五个实体类。同为静态资源的 Image 则单独拿出来，便于今后搜寻。

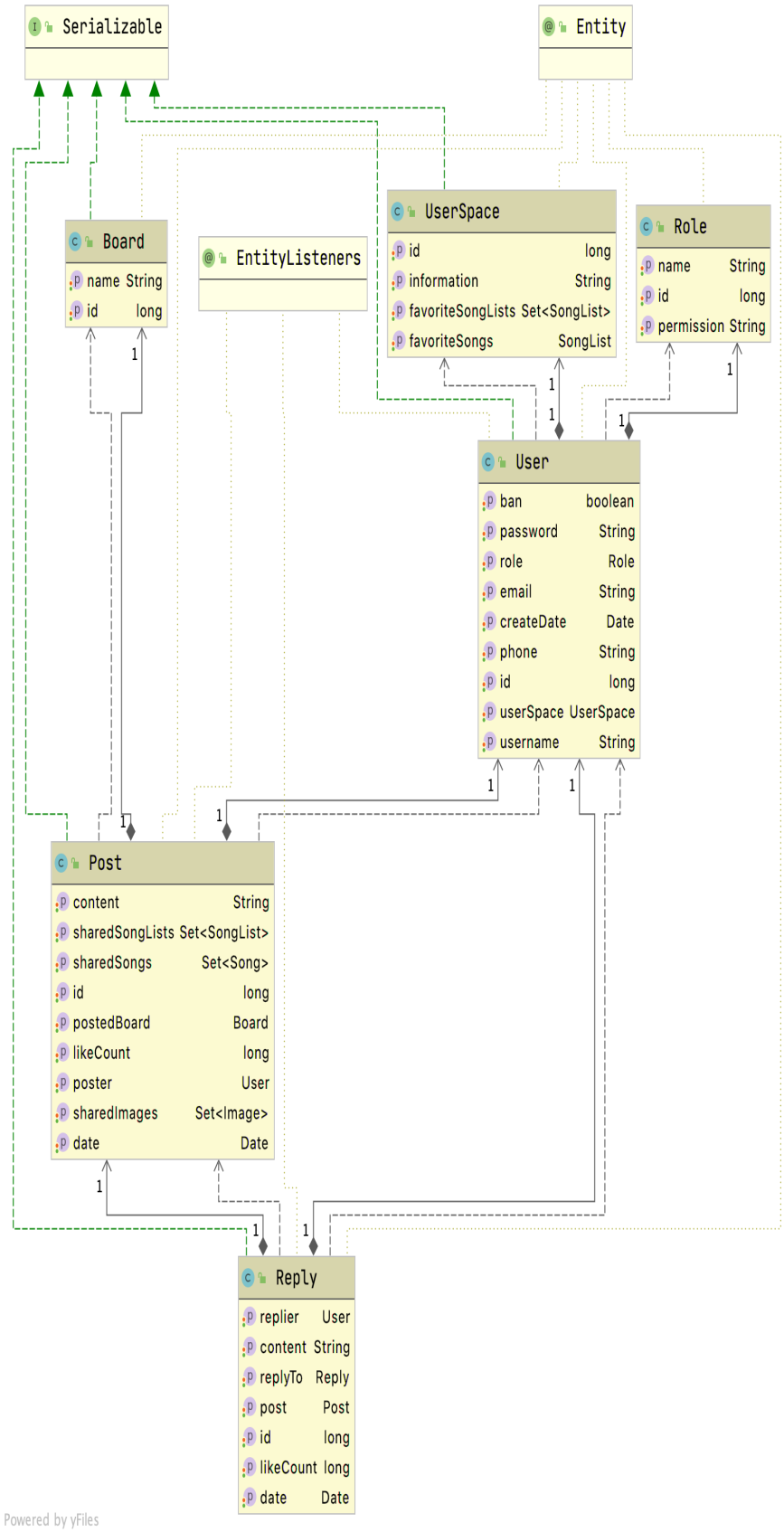

Serializable
Entity
Board
name String
id long
EntityListeners
UserSpace
id long
information String
favoriteSongLists Set<SongList>
favoriteSongs SongList
Role
name String
id long
permission String
User
ban boolean
password String
role Role
email String
createDate Date
phone String
id long
userSpace UserSpace
username String
Post
content String
sharedSongLists Set<SongList>
sharedSongs Set<Song>
id long
postedBoard Board
likeCount long
poster User
sharedImages Set<Image>
date Date
Reply
replier User
content String
replyTo Reply
post Post
id long
likeCount long
date Date
1
Powered by yFiles

图 3-2 论坛类图

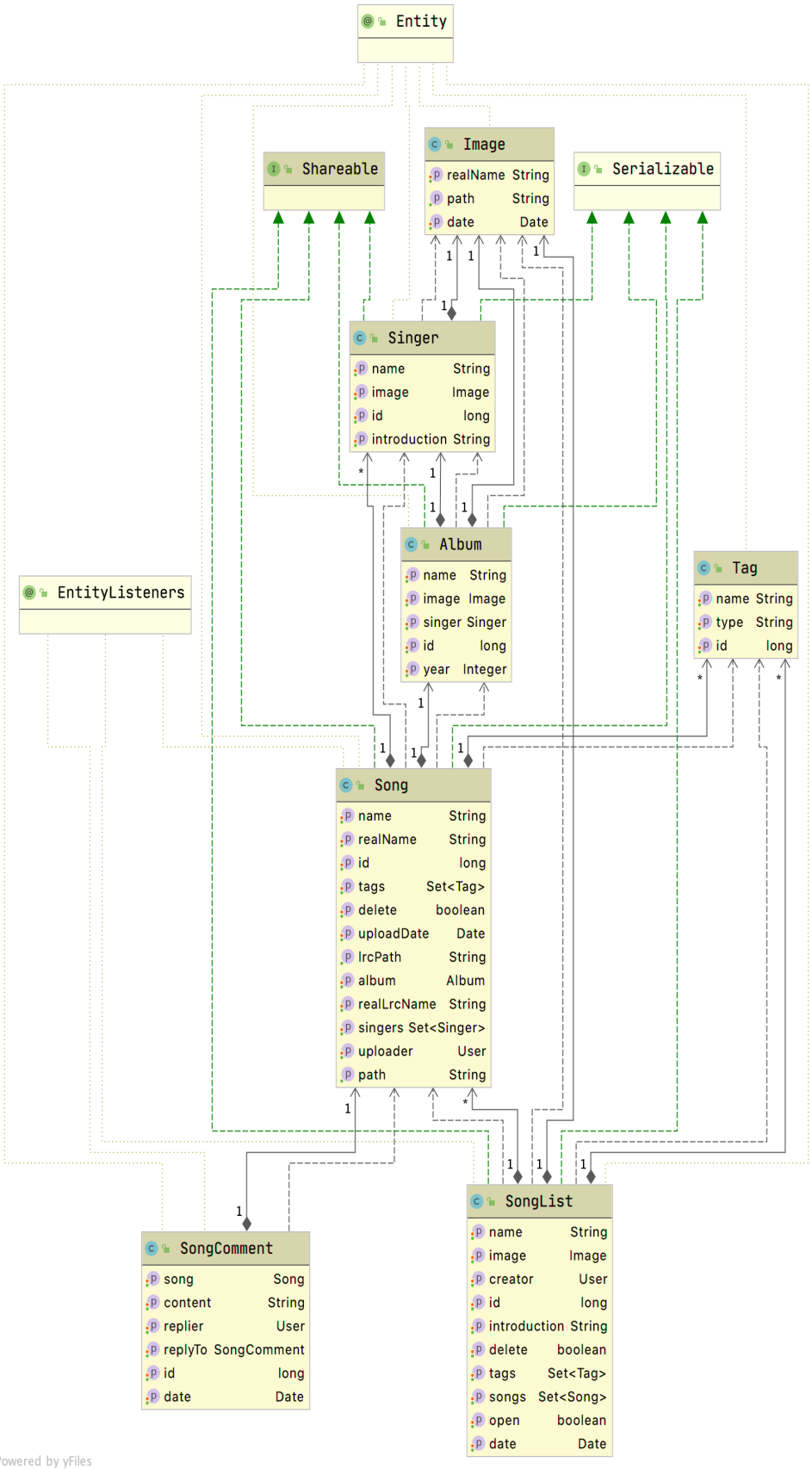

Entity
Shareable
Image
realName String
path String
date Date
Serializable
Singer
name String
image Image
id long
introduction String
Album
name String
image Image
singer Singer
id long
year Integer
Tag
name String
type String
id long
EntityListeners
Song
name String
realName String
id long
tags Set<Tag>
delete boolean
uploadDate Date
lrcPath String
album Album
realLrcName String
singers Set<Singer>
uploader User
path String
SongComment
song Song
content String
replier User
replyTo SongComment
id long
date Date
SongList
name String
image Image
creator User
id long
introduction String
delete boolean
tags Set<Tag>
songs Set<Song>
open boolean
date Date
Powered by yFiles

图 3-3 静态资源类图

（2）时序图设计

系统的亮点是短信验证快捷登录功能。如图 3-4 用户登录时可选择手机快捷登录功能，向服务器请求验证码。后台服务器接到请求后，会调用阿里云 SDK 发送短信验证码，并以手机号为键，验证码为值存入 Redis 缓存中,并设置过期时间 2 分钟[12]。如图 3-5 用户再将验证码和手机号提交到后台的登录接口，后台向 Redis 验证验证码的合法性。

系统的核心功能为歌曲上传系统，用户选择上传的同时会调用服务器的 API。如图 3-6 后台会根据用户上传的内容进行判断是否有相同文件，如没有，则写入用户指定的文件夹。

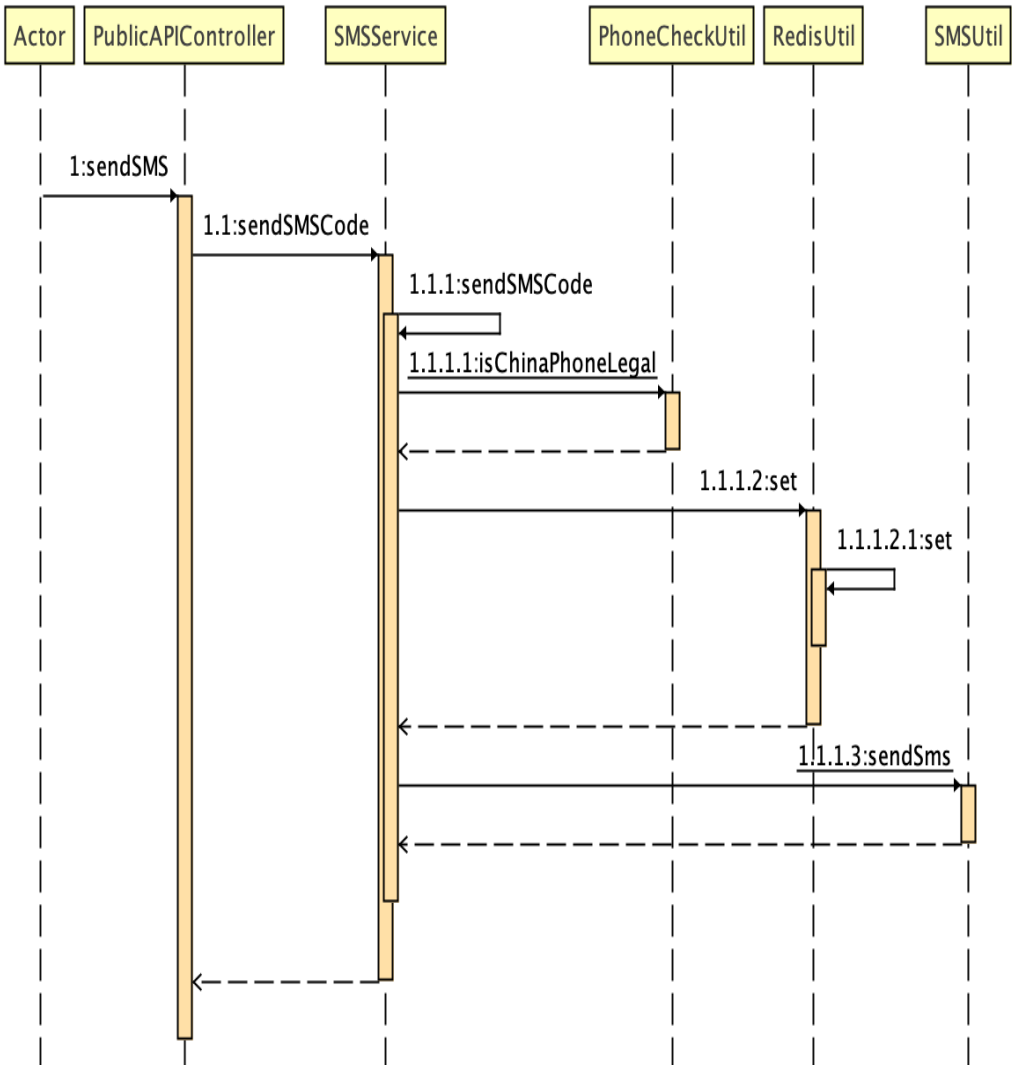


图 3-4 请求验证码时序图

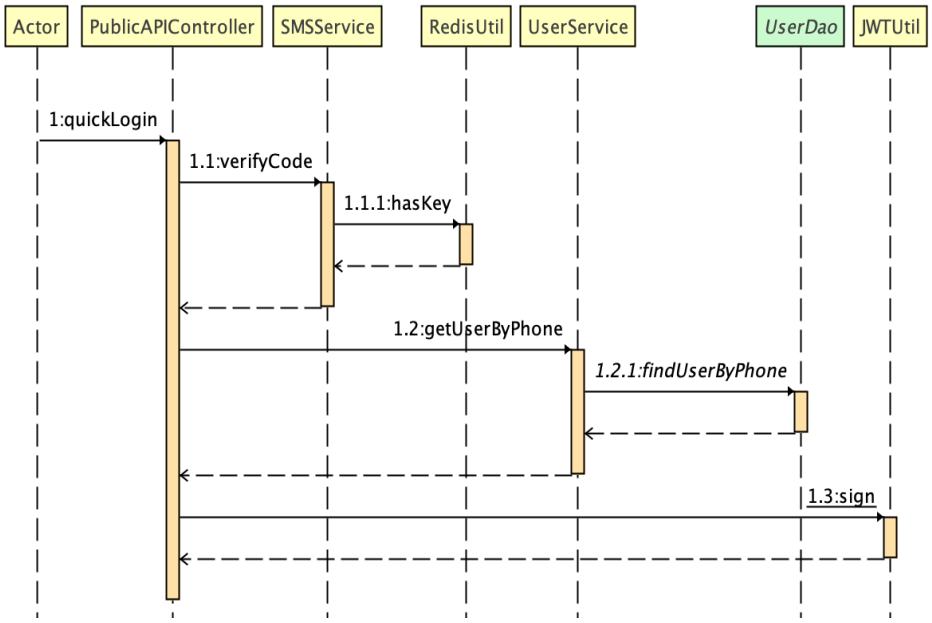


图 3-5 快捷登录时序图

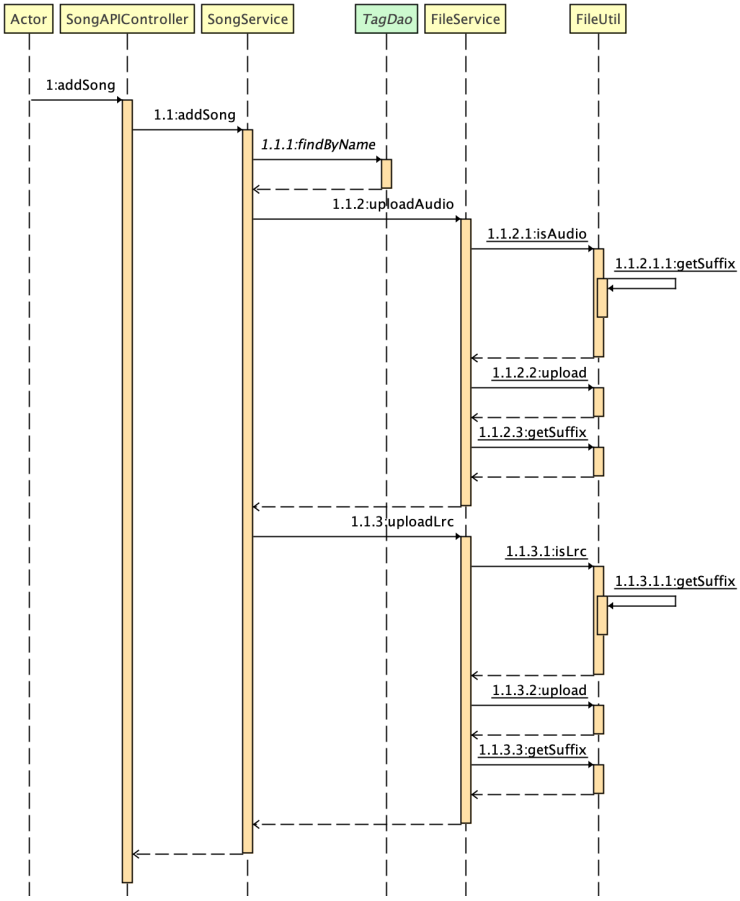


图 3-6 歌曲上传时序图

（3）活动图设计

图 3-4 展示了动态分享的活动图。对于用户的权限验证，本系统使用了 JWT token 进行身份验证，所有的认证信息不会保存在服务端，而是以令牌的形式将 Token 保存在用户端，每次请求时在请求头 Header 中携带该 token 即可，服务端只在用户第一次登录时进行了数据库的读取，之后的请求只需验证 Token 的合法性即可，大大减少了服务器的内存开支，并且实现了服务段的无状态，非常适合 APP 使用[13]。。

同时本项目还使用了 Shiro 作为权限验证框架，在每次请求时基于 Spring AOP 的拦截器都会拦截请求，并通过自己实现的 Realm 进行权限验证。保证了每个接口甚至静态资源不会被无权限的用户访问。

用户选择分享动态时，可以选择图片，音乐甚至是歌单。如果用户选择音乐，或歌单，系统会提示从已存在的歌单或者音乐中选择，如果用户选择新建歌单或音乐，系统会引导用户到新建界面。

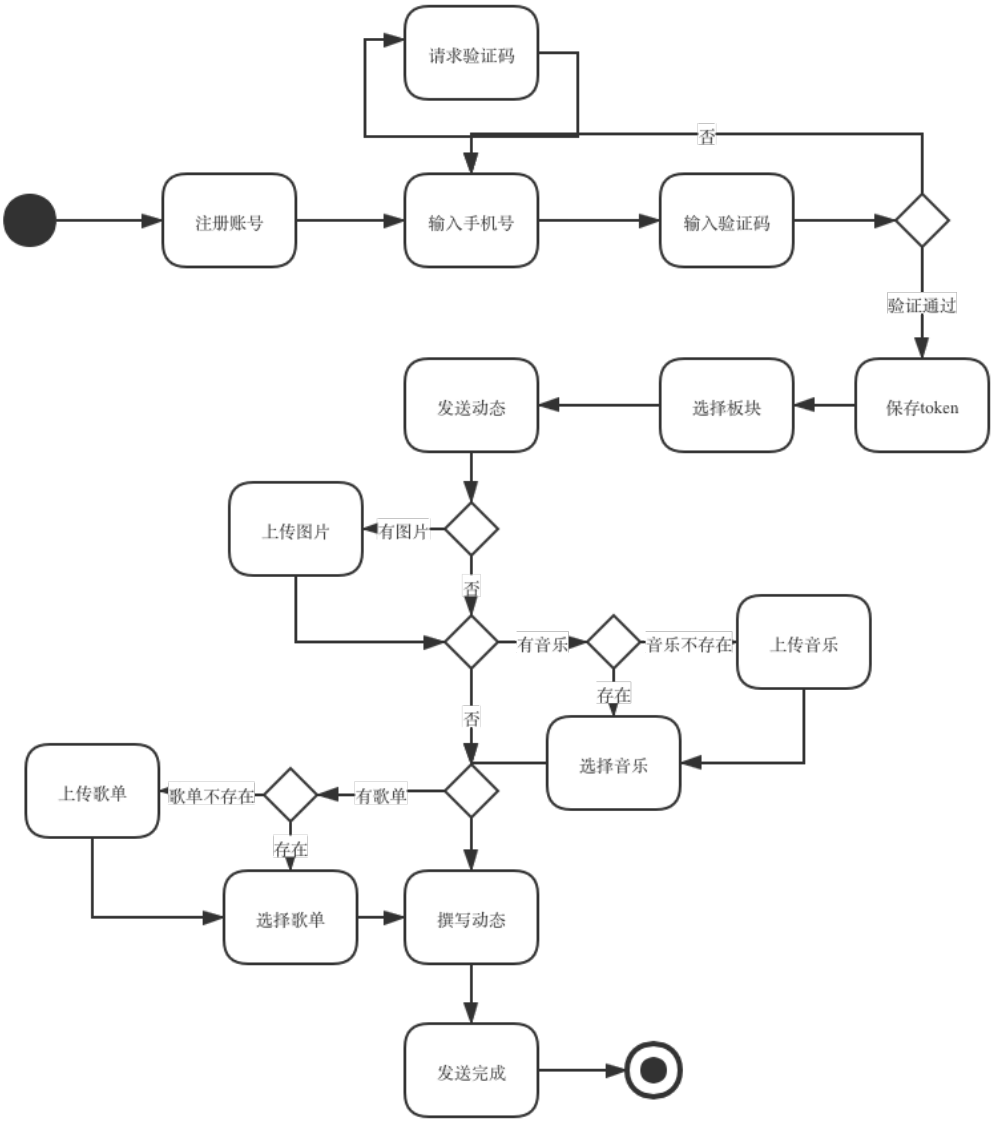


图 3-7 动态模块活动图

3.3 持久化设计

3.3.1 数据库逻辑关系

图 3-8 展示了本系统社交部分的用户、角色、个人空间、动态、回复之间的关系。用户与动态和回复是一对多的关系，体现了系统的核心需求。用户通过发送动态及回复进行系统核心的社交功能。

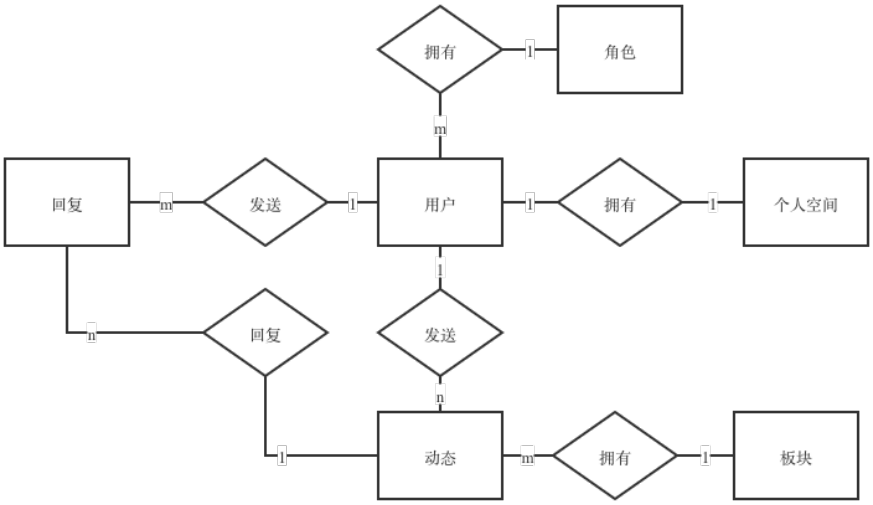


图 3-8 社交系统 E-R 图

3.3.2 数据库表设计

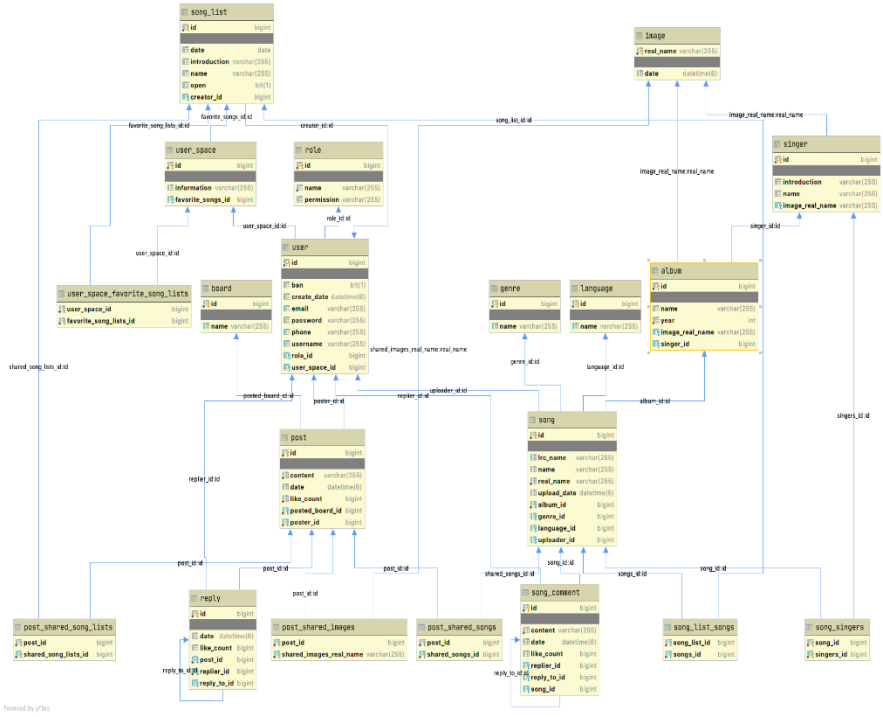


图 3-9 数据库表图

在实现系统时，数据库的设计往往决定了系统性能的上限，好的数据库设计可以有效地简化多表查询的次数，显著提升 QPS 效率。本系统数据库使用的是 MySQL 数据库，由于音乐的数据量较多，为了提升效率，在字段上进行了一些冗余，并且采用了可变长度字符来节约空间。所有字段均采用 long 值作为 id，避免发生键值溢出的情况，也方便 MySQL 在查询的时候自动建立索引。本系统使用 Spring Boot 搭建后台，使用同为 Spring 全家桶的 Spring Data JPA 作为 ORM 框架，数据库表直接由模型生成，同时 JPA 也提供了一些简单的接口，方便开发时节省不必要的基本增删改查 SQL 语句。图 3-9 展示了本系统数据表属性及关联关系。

核心表的详细结构及其字段约束信息如下：

（1）用户表

表 3-1 用户表

列名	数据类型	字段类型	长度	是否为空	默认值	备注
ban	bit(1)	bit		YES	无	是否被 ban
create_date	datetime(6)	datetime		YES	无	创建日期
email	varchar(255)	varchar	255	YES	无	邮箱
password	varchar(255)	varchar	255	YES	无	密码
phone	varchar(255)	varchar	255	YES	无	手机
role_id	bigint	bigint		YES	无	角色 id 外键
user_space_id	bigint	bigint		YES	无	用户空间外键
username	varchar(255)	varchar	255	YES	无	用户名

（2）动态表

表 3-2 动态表

列名	数据类型	字段类型	长度	是否为空	默认值	备注
content	varchar(255)	varchar	255	NO	无	内容
date	datetime(6)	datetime		YES	无	日期
id	bigint	bigint		NO	无	id
like_count	bigint	bigint		NO	无	点赞次数
posted_board_id	bigint	bigint		NO	无	所在板块外键
poster_id	bigint	bigint		NO	无	发送者外键

（3）回复表

表 3-3 回复表

列名	数据类型	字段类型	长度	是否为空	默认值	备注
date	datetime(6)	datetime		YES	无	时间
id	bigint	bigint		NO	无	id
like_count	bigint	bigint		YES	无	点赞次数
replier_id	bigint	bigint		NO	无	回复者 id
reply_to_id	bigint	bigint		YES	无	回复帖子的 id

3.4 针对社会、健康、安全、法律等因素的相关设计

本系统是一个音乐共享平台，系统内容完全由用户上传，不存在违规行为。在社会健康的设计上，本系统作为工具，为使用者提供资源交流和推荐功能，节约用户对时间，用户并不会长期沉迷于应用，保护使用者的身心健康。另外，所有音乐资源均会经管理员审核后进行发布，如有不符合社会主义核心价值观的内容，审核不予通过，不会传播可能对社会造成负面影响的资源。在法律方面，在系统所用音乐资源完全由用户自主上传，本平台仅起到交流及分享作用，如用户上传的任何资源涉及到侵权，本网站会立即删除该资源并对涉嫌用户进行警告处理，严重侵权将会取消其会员资格，不会存在法律纠纷。开发软件均采用 EDU 授权版本，无需对软件额外付费，系统的整个设计均为本人自创，属于正常的社交软件，不存在专利和版权纠纷。同时本系统所含的推荐功能不会收集用户的敏感信息，所有采集的数据均进行脱敏处理，用户的隐私安全可以得到完全保证。

3.5 本章小结

本章节简要介绍了系统的功能结构、功能设计和持久化设计，辅以绘制的图形和文字说明来对本系统的设计思路进行概括，最后对社会健康、文化以及法律相关设计进行阐述。

4 系统的实现

4.1 多平台架构实现

4.1.1 前端架构

（1）设计思路

主要采用 Flutter 框架进行多平台的开发，通过 Android Studio 对 Flutter 整个项目的目录进行管理，由 Flutter 自带的引擎解析 Dart 代码，编译成 Target System 的原生代码。本项目主要编译了 Android 和 iOS 2 种平台的 Native 代码，Android 编译成 Java，BuildTarget=SDK29; iOS 编译成 Swift，MinimizeRequirement= iOS 12.0。

Dart 代码基本遵循了目前前端流行的 MVVM 架构，即 Model-View-ViewModel 的设计方法，设计上参考了开源 App NeteaseCloudMusicd 的思路。具体思路是先独立编写各个页面，即向用户展示的 View，View 之间的跳转通过路由来完成，由于目前版本 Flutter 对路由的支持不是很好，所以采用了自己实现的 RouteUtil 来完成路由功能。

各个页面之间的数据传递使用 Flutter 自带的 Provider 来实现，通过继承 ChangeNotifier 来把 ViewModel 注册成一个通知。当页面需要请求时调用自己实现的 NetUtils 来向后端获取 Model 对象，接收到 Model 对象后即返回到 Provider 封装成 ViewModel，在通过发送通知的方式将 ViewModel 传回前端页面，至此即为前端 App 页面与数据交互的大致思路，图 4-1 为整个交互逻辑的协作图。

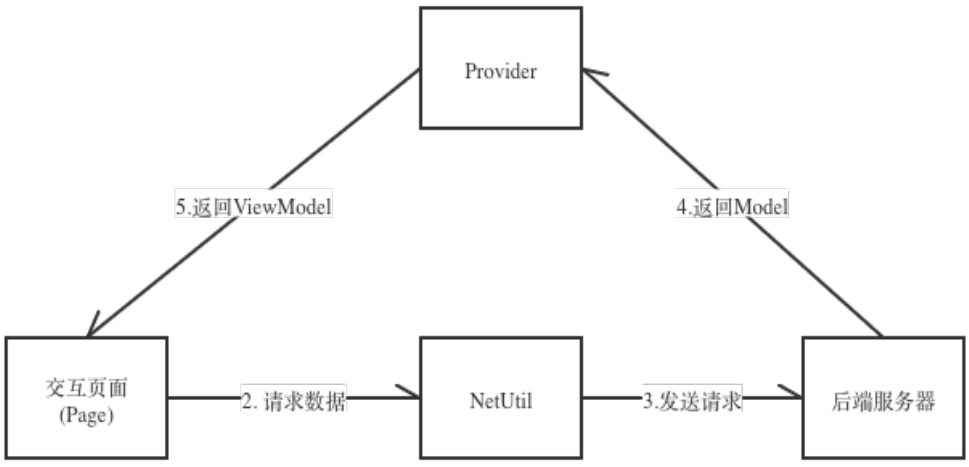


图 4-1 前端交互协作图

（2）实现代码

```
class UserModel with ChangeNotifier {
  User _user;

  String _token;

  User get user => _user;

  /// 初始化 User
  void initUser() {
    if (Application.sp.containsKey('user')) {
      String s = Application.sp.getString('user');
      _user = User.fromJson(json.decode(s));
    }
    if (Application.sp.containsKey('token')){
      String s = Application.sp.getString('token');
      _token=s;
    }
  }

  /// 登录
  Future<User> login(BuildContext context, String phone, String pwd) async {

    ResponseData responseData = await NetUtils1.login(context, phone, pwd);
    if (responseData.code != 200) {
      Utils.showToast(responseData.msg ?? '登录失败，请检查账号密码');
      return null;
    }
    Utils.showToast('登录成功');
    saveToken(responseData.data);
    User user = await NetUtils1.getUserInfo(context);
    _saveUserInfo(user);
    return user;
  }
```

4.1.2 后端架构

（1）设计思路

主要遵循了传统的 MVC 架构进行后端开发，即 Model-View-Controller。在此程度上为了方便今后对 Web 端进行适配，增加接口适用性，所有接口均返回 Json 对象，同时对返回的对象进行了标准化。

整个后端交互的时序图如图 4-2 所示，对所有不分页请求的均返回自己实现的 ResponeData 对象，对于需要分页的请求，为了便于前端进行开发统一返回 ResponeData 的实现类 PageResponeData。这个个对象均封装了 code 状态码、msg 提示消息和 data 数据，PageResponeData 还包括了 page、num、total 等分页相关的数据。通过枚举类实现状态码 code 和提示消息 msg 的规范化，让代码看起来更加整洁，可读性更强。

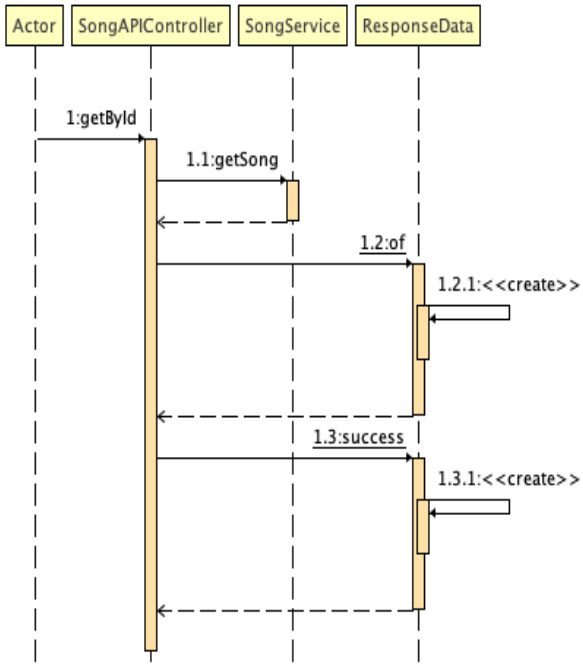


图 4-2 后端交互时序图

4.1 前端跨平台状态管理

本系统采用 Token 的方式来验证用户登录状态，在服务器端不会保存用户的登录状态，减少了服务器内存开支。同时前端组件也大量采用了 StatefulWidget 进行开发，方便代码复用。

4.1.1 登陆态保持及权限管理

（1）登录态算法思路分析

登录态即指登录的状态，目前 web 常用的登录态保持机制有 Session 会话和 JWT Token[13]。

Session 会话是指服务器端对每一个用户都生成一个会话对象，在将 sessionId 写入到用户的 cookie 中来判断用户对应的会话窗口。由于此方式是将所有会话数据保存到服务器端的内存中，当用户量比较庞大时会严重消耗服务器资源，同时 session 只对应一台服务器，当整个服务采用微服务集群的方式部署时，用户的登录态无法在各个服务间传播，所以拓展性很差。JWT token 是用户在第一次登录时获取的一个由服务器端秘钥进行签名的一串字符串，类似于令牌。用户今后的请求只需携带该令牌即可，不会在服务器端额外保存一段数据，可以大大减少不必要的内存开销和数据库读取。所以本客户端选择 Token 来作为验证的方式，用户第一次输入用户密码登录时，将服务器返回的 Token 存储在本地的 SharedPreferences ，每次请求时通过 dart 包 Dio 在请求头附带 Token 信息进行验证。

（2）权限细分思路分析

本系统采用 Shiro 框架进行权限管理，同时整合 JWT，实现 Token 验证的同时，也能进行完整的权限管理。

Apache Shiro 是具有许多功能的全面的程序流安全性体系结构。 Shiro 提供了一个简单而直观的 API，可以轻松解决身份验证，授予管理权，公司会话管理方法和数据加密的问题。此外，Shiro 易于应用和理解，其功能也非常强大，它可以验证用户的真实身份，对用户执行访问控制，区分是否为用户分配了明确的安全角色，以及区分用户是否被允许做某事。在 Shiro 中，Session 的功能是建立相对使用的会话编程范例。范式和工具互不相关，因此，即使没有 Web 或 EJB 组件，Shiro 也适合在所有自然环境中应用 Session API。在身份认证访问控制的整个过程中，Shiro 还可以响应恶意事件，或者在会话的生存期内。Apache shiro 的体系结构包含三个主要概念：主题概念，主题管理器和领域。在这些内容中，主题是一个相对抽象的概念。我们通常将 Subject 对象理解为可以与应用程序交互的任何“用户”，但是它也可以是一个由三方组成的程序，以及可以与系统交互的任何“事物”。“所有主题。使用 Shiro 框架中的 Subject 来完成诸如登录，注销，验证权限和获取会话之类的操作。SecurityManager 是 Shiro 的核心，也是 Shiro 的核心组件；所有特定的安全操作均由 SecurityManager 控制；SecurityManager 管理所有主题，并且与主题相关的所有操作都由 SecurityManager 进行交互；领域充当 Shiro 与应用程序之间的“桥梁”或“连接器”，用于 Shiro 的用户身份验证和授权； Realms 实际上是特定的安全 DAO：封装数据源的连接详细信息，以便能够获取所需的 Shiro 相关数据。[14]。

通过用户自定义实现 AuthorizingRealm 接口，Shiro 可进行粒度更细的权限分割。本项目实现了 doGetAuthenticationInfo 方法，用于提取 token 中的用户信息，通过和数据库中用户拥有的角色作比对，从而获得用户的权限。

同时，本项目还实现了 doGetAuthentizationInfo 方法，将已剥离出的用户角色再进行细分，与数据库每个角色拥有的精确到每部操作的具体权限进行对比，并写入 Shiro 的 AuthenticationInfo，以此完成从角色到具体操作的权限细分[17]。

（3）实现代码

首先实现了一个自己的 realme, 用来做最终的权限和角色验证

```
package wxm.example.comical_music_server.shiro;

@Service
public class MyRealm extends AuthorizingRealm {

    private static final Logger LOGGER = LogManager.getLogger(MyRealm.class);

    @Autowired
    private UserService userService;


    /**
     * 大坑！，必须重写此方法，不然 Shiro 会报错
     */
    @Override
    public boolean supports(AuthenticationToken token) {
        return token instanceof JWTToken;
    }

    /**
     * 只有当需要检测用户权限的时候才会调用此方法，例如 checkRole,checkPermission 之类的
     */
    @Override
```

```
protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) throws AuthenticationException{
    //          String username = JWTUtil.getUsername(principals.toString());
    //          User user = userService.getUser(username);
            SimpleAuthorizationInfo simpleAuthorizationInfo = new SimpleAuthorizationInfo();
    //          Role role=user.getRole();
            User user= (User) principals.getPrimaryPrincipal();
            Role role= user.getRole();
            if (role!=null){
                simpleAuthorizationInfo.addRole(role.getName());
                Set<String> permission = new HashSet<String>(Arrays.asList(role.getPermission().split(",")));
                simpleAuthorizationInfo.addStringPermissions(permission);
            }
            return simpleAuthorizationInfo;
        }
```

4.1.2 前端组件状态管理

（1）算法思路分析

组件化思想在如今的大前端环境中非常流行，像 React、Vue 等前端 JS 框架均支持组件化，可以将许多页面中相同的部分抽离做成组件。但是当页面越来越庞大，组件被不断组合和服用，这使得组件的状态管理变得十分困难。为未来而生的 Flutter 在设计之初就考虑到了这一点，它的核心原则就是“Everything is a Widget”，即“一切皆为组件”。而 Flutter 中只有两种组件，StatefulWidget （状态化组件）和 StatelessWidget （无状态组件），这样的设计使得管理组件状态异常的方便和直观。

本系统使用类似 React 的路由管理系统，通过路由管理各个页面的跳转，并在前端收到后端返回的 JSON 对象后，实时绘制每个部分的组件（Widget）。由此实现动态生成动态帖子的数据。

（2）实现代码

```
import 'package:fluro/fluro.dart';
import 'package:flutter/material.dart';
import 'package:comical_music/pages/login_page.dart';
import 'package:comical_music/route/route_handles.dart';
```

```
class Routes {
  static String root = "/";
  static String home = "/home";
  static String login = "/login";
  static String dailySongs = "/daily_songs";
  static String playList = "/play_list";
  static String topList = "/top_list";
  static String playSongs = "/play_songs";
  static String comment = "/comment";
  static String search = "/search";
  static String lookImg = "/look_img";

  static void configureRoutes(Router router) {
    router.notFoundHandler = new Handler(
        handlerFunc: (BuildContext context, Map<String, List<String>>
params) {
      print("ROUTE WAS NOT FOUND !!!");
      return LoginPage();
    });
    router.define(root, handler: splashHandler);
    router.define(login, handler: loginHandler);
    router.define(home, handler: homeHandler);
    router.define(dailySongs, handler: dailySongsHandler);
    router.define(playList, handler: playListHandler);
    router.define(topList, handler: topListHandler);
    router.define(playSongs, handler: playSongsHandler);
    router.define(comment, handler: commentHandler);
    router.define(search, handler: searchHandler);
    router.define(lookImg, handler: lookImgHandler);
  }
}
```

4.2 数据请求

4.2.1 音乐上传及动态分享请求

（1）算法思路

本系统所有相关数据均来自用户上传，用户再上传音乐的时候可以指定歌手，专辑，歌曲类型。所有接口均严格按照 RESTFUL API 进行编写。用户通过 form-data 进行 post 请求，上传音乐文件刘，同时后端通过 MultipartFile 对象进行解析为 IO 流，生成随机的 UUID 覆盖每个文件原本的文件名。动态分享模块中上传图片也是同样的思路。

（2）实现代码

```
public class FileService {
    //TODO 可将文件 md5 存入缓存，避免重复上传

    @Autowired
    private ImageDao imageDao;
    public Image uploadImg(MultipartFile file){
        if(file.isEmpty()){
            return null;
        }

        String fileName = file.getOriginalFilename();
        if(!FileUtil.isImage(fileName)){
            return null;
        }

        File
realFile=FileUtil.upload(file,Constant.RESOURCE_PATH+Constant.IMG_PATH,
UUID.randomUUID().toString().replace("-","")+"."+FileUtil.getSuffix(fileName));
        if (realFile==null){
            return null;
        }
        Image image=new Image(realFile.getName());
        image=imageDao.saveAndFlush(image);
        return image;
    }
}
```

4.2.2 静态资源请求

（1）算法思路

本系统为音乐分享系统，未避免版权纠纷，所有音乐数据仅为内部交流。需要对静态资源进行权限隔离，未登录的用户将不能访问音乐的静态资源。

总体思路是把用户所有上传的静态资源以 Resource 的形式加载[16]，用 controller 拦截 static 路径下的所有请求，再根据 url 的路径名把真实的静态资源解析过去，返回文件流（类似文件下载服务）。中间鉴权步骤通过 Shiro 自带的工具类实现，读取已经解析好的用户权限信息。

（2）实现代码

```
    @Service
public class FileService {
        public Resource loadFileAsResource(String path) throws IOException{
        String realPath= Constant.RESOURCE_PATH+path;
        try {
            Resource resource = new FileSystemResource(new File(realPath));
            if(resource.exists()) {
                return resource;
            } else {
                throw new IOException("File not found "+ realPath);
            }
        } catch (Exception e){
            throw new IOException("File not found " + realPath);
        }
    }
}

    @RestController
@RequestMapping(Constant.STATIC_URL_PATH)
public class StaticAPIController {
    @Autowired
    private FileService fileService;

    @GetMapping("/{type}/{name}")
    public ResponseEntity<Resource> getStatic(@PathVariable String type,
@PathVariable String name ) throws NotFoundException {
```

```
            try {
                if (type.equals("audio")){//如果是音频，检查是否有权限
                    if(!SecurityUtils.getSubject().isAuthenticated()){
                        throw new AuthorizationException();
                    }
                }
                Resource
resource=fileService.loadFileAsResource("/"+type+"/"+name);
                String contentType=FileUtil.getMimeType(resource.getFile());
                if(contentType == null) {
                    contentType = "application/octet-stream";
                }
                return ResponseEntity.ok()
                        .contentType(MediaType.parseMediaType(contentType))
                        .body(resource);
            } catch (Exception e) {
                e.printStackTrace();
                throw new NotFoundException();
            }
        }
}
```

4.3 后端异常处理

4.3.1 局部异常处理

（1）算法思路

所有接口均在 service 层实现了对异常的处理，对于所有请求在 Controller 层均返回固定的 ResponeData 格式 Json 对象，并使用枚举类对所有可能出现的异常状态进行了枚举并自定义了状态码和错误提示消息。尽可能的 Controller 层对可能出现的异常进行了处理,并根据情况返回特定的状态码。

同时为了弥补 Spring 中@RequestParam 注解只能判断传入参数是否存在，无法判断参数的类型是否合法的 BUG（例如不能判断传入的 int 值是否为空）。本项目基于 AOP 拦截器实现了一个简单的参数验证功能，只需在参入的参数前面加入特定的注解即可检验参数的合法性

（2）实现代码

由于基本所有接口都实现了局部异常处理，此处仅列举用 AOP 注解判断参数是否合法的代码，以及几个典型示范。

通过 AOP 拦截器实现参数检验

```
package wxm.example.comical_music_server.aop;

/**
 * @author Alex Wang
 * @date 2020/05/12
 */
    @Around("checkParam()")
    public Object doAround(ProceedingJoinPoint pjp) throws Throwable {

        MethodSignature signature = ((MethodSignature) pjp.getSignature());
        //得到拦截的方法
        Method method = signature.getMethod();
        //获取方法参数注解，返回二维数组是因为某些参数可能存在多个注解
        Annotation[][] parameterAnnotations = method.getParameterAnnotations();
        if (parameterAnnotations == null || parameterAnnotations.length == 0) {
            return pjp.proceed();
        }
        //获取方法参数名
        String[] paramNames = signature.getParameterNames();
        //获取参数值
        Object[] paranValues = pjp.getArgs();
        //获取方法参数类型
        Class<?>[] parameterTypes = method.getParameterTypes();
        for (int i = 0; i < parameterAnnotations.length; i++) {
            for (int j = 0; j < parameterAnnotations[i].length; j++) {
                //如果该参数前面的注解是 ParamCheck 的实例，并且notNull()=true,则进行非空校验
```

```
                    if (parameterAnnotations[i][j] != null &&
parameterAnnotations[i][j] instanceof ParamCheck && ((ParamCheck)
parameterAnnotations[i][j]).notNull()) {
                        paramIsNull(paramNames[i], paranValues[i],
parameterTypes[i] == null ? null : parameterTypes[i].getName());
                        break;
                    }
                }
            }
            return pjp.proceed();
        }
```

4.3.3 全局异常处理

（1）算法思路

通过 Spring 自带的 RestControllerAdvicer 捕捉特定类型的异常，并同意返回自己实现的 ResponeData 对象[17]。对于特定错误来说的来说，通过 ExceptionHandler 注解进行捕捉异常类，在此处对上一节未在 controller 处理的异常以及 AOP 出现的异常进行捕捉。

（2）实现代码

```
package wxm.example.comical_music_server.controller;
@RestControllerAdvice
public class ExceptionController {
    // 捕捉 UnauthorizedException
    @ResponseStatus(HttpStatus.UNAUTHORIZED)
    @ExceptionHandler(UnauthorizedException.class)
    public ResponseData handle401(UnauthorizedException e) {
        //System.out.println(11);
        return new ResponseData(401, e.getMessage(),null);
    }
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(MultipartException.class)
    public ResponseData handleMultipartException(){
        return new ResponseData(HttpStatus.BAD_REQUEST.value()," 文件
大小超出限制",null);
    }
```

```
@ResponseStatus(HttpStatus.NOT_FOUND)
@ExceptionHandler({NotFoundException.class})
public ResponseData handle404(){
    return new ResponseData(StatusCode.NOT_FOUND,null);
}
// 捕捉其他所有异常
@ExceptionHandler(Exception.class)
@ResponseStatus(HttpStatus.BAD_REQUEST)
public ResponseData globalException(HttpServletRequest request, Throwable ex) {
    //ex.printStackTrace();
    return new ResponseData(getStatus(request).value(), ex.getMessage(), null);
}
```

4.4 智能推荐

4.4.1 音乐推荐

（1）算法思路

本系统实现了分用户推荐歌曲的功能，主要是基于用户喜欢的音乐分类进行推荐。主要根据用户播放过的歌曲标签进行类似歌曲的推荐，哪个种类的歌曲播放越多，在常听标签里占得权值越大。同时进行影响推荐歌曲的其他变量也有歌曲播放量，歌曲是否被用户收藏，和歌曲的上传时间。

采用的计算公式如下：

单个标签权值= （单个标签聆听数 / 总标签聆听数） * 10 （4-1）

歌曲权值= $\Sigma_{\text{所有标签}}$ (标签 * 标签权值) + 歌曲播放总量/100 + （30-上次被播放时距离现在的天数）/10 + (被用户收藏？10 : 0) （4-2）

系统在用户进行请求后，会根据以上公式计算出所有歌曲对该用户的权值，并选取权值最高的 8 首歌曲返回给用户。同时会将该结果存入 Redis 缓存，过期时间为一天，这样用户再一天内多次请求就不会再消耗大量时间进行推荐算法的计算了。这样整个系统就完成了被动生成用户每日推荐的功能。

（2）实现代码

```
public List<Song> getUserRecommandSong(User user) {
    String key = Constant.PREFIX_USER_TAG + user.getId();
    List<Song> songs = songDao.findAllByExist(true);
    Map<Object, Object> map = redisUtil.hmget(key);
    Map<String, Integer> scores = new HashMap<>();
    Map<Song, Integer> songScores = new HashMap<>();
    int total = 0;
    for (Object o :
            map.keySet()) {
        String s = (String) o;
        Integer num = (Integer) map.get(o);
        total += num;
        scores.put(s, num);
    }
    for (String s :
            scores.keySet()) {
        Integer i = scores.get(s);
        scores.put(s, i * 10 / total);
    }

    for (Song s :
            songs) {
        Integer score = 0;
        Set<Tag> ts = s.getTags();
        for (Tag tag : ts) {
            Integer sc = scores.get(tag.getName());
            if (sc != null) {
                score += sc;
            }
        }
        score += (int) s.getPlayCount() / 10;
        songScores.put(s, score);
    }
    Map<Song, Integer> sortedScores=sortMapByValue(songScores);
```

```
}
```

4.5 本章小结

本章对系统状态管理、数据请求、异常处理和音乐推荐的算法进行了分析，并且将其部分核心代码进行展示。

5 系统运行与效果分析

5.1 界面设计概要

本系统遵循 Google 倡导的 Metal Design，使用了 Flutter 原生的 Metal Design 组件进行开发，所有组件均采用 Metal Design 颜色上色。同时部分界面参考了网易云音乐，UI 简洁明了，UX 清晰直观，音乐播放部分和市场上同类产品交互体验类似，大大减少了用户的学习成本。

5.2 用户信息校验

5.2.1 用户登陆

本系统仅限注册用户登录，非注册用户无法使用系统中的任一功能。图 5-1 展示了系统的登录界面，登录时采用 token 验证，app 会将服务器返回的 token 存入本地，每次请求使用 Dio 包在请求头中携带 token 信息，用户只需登录一次即可。

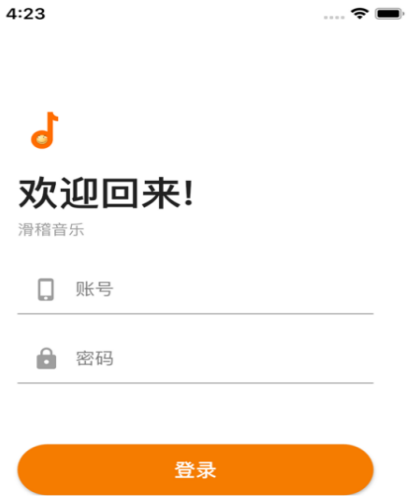


图 5-1 用户登录界面图

5.3 主界面展示

5.3.1 系统动态展示

图 5-2 用户动态图

5.3.2 音乐详情展示

图 5-3 音乐详情界面图

5.3.3 个人详情界面

图 5-4 个人详情图

5.4 本章小结

本章主要通过 app 运行时的系统效果截图进行展示，同时对这些截图进行文字说明，解释各个界面的功能

6 系统测试

6.1 测试方法

常用的测试方法有黑盒测试和白盒测试[18]。

黑盒测试把被测物看成是黑盒子，无法打开。在测试过程中，测试人员完全不需要考虑箱内的逻辑结构和具体操作，只需要根据程序的需求规格说明书，检查程序的功能是否符合其功能说明，检查输出结果是否正确。

白盒测试与黑盒测试相反，它将测试对象视为一个开放的透明盒子。在测试过程中，测试员利用程序内部的逻辑结构和相关信息，通过对各点的程序状态进行检测，来判断程序中的每一条通道是否能够按照预定的要求正常工作。

该系统主要采用黑盒测试，以发现系统设计中的异常

6.2 测试方案及计划

6.2.1 系统功能介绍

本系统是一个音乐分享交流应用，完成音乐分享交流的同时，提供音乐播放功能，同时分析用户的音乐播放行为，进行用户可能感兴趣的音乐推荐。测试部分主要包括用户登录注册、音乐歌单搜索、音乐播放、动态发送、音乐上传、歌单上传。

6.2.2 测试目的

（1）测试交互逻辑是否有不足。

（2）测试重要需求点是否完成。

（3）测试系统整体的安全级别。

（4）测试系统对不同平台的兼容性。

6.2.3 测试范围

针对测试的系统模块，测试范围如表 6-1 所示，其中说明：优先级 1 表示在一期测试，为 2 表示在二期测试。

表 6-1 系统模块测试

模块名称	描述	优先级
用户登录	注册用户输入账号和密码进行登录	1
用户注册	用户注册为论坛的注册用户	1
发送动态	用户发送动态	1

续表 6-1

搜索	用户进行搜索	1
音乐上传	用户上传音乐	1
音乐信息查看	用户查看音乐并播放	2
上传歌单	用户上传歌单	2
音乐推荐	系统推荐音乐	1

6.2.4 测试进度安排

针对测试中不同的阶段，测试进度安排如表 6-2 所示。

表 6-2 测试进度安排表

测试过程	计划开始日期	实际开始日期	实际结束日期
熟悉系统设计需求	20200414	20200414	20200417
制定测试计划	20200417	20200417	20200422
制定测试方案	20200422	20200422	20200428
设计测试用例	20200428	20200428	20200503
模块、集成测试（一期）	20200503	20200503	20200510
模块、集成测试（二期）	20200510	20200510	20200511
系统测试	20200512	220200512	20200513
在线测试	20200513	20200513	20200514
系统测试报告	20200514	20200514	20200515

6.3 测试过程及结果分析

6.3.1 测试用例设计

系统测试用例设计如表 6-3 所示。

表 6-3 系统功能测试用例

测试功能	测试用例	预期结果	结果
注册用户登录	（1）帐号和密码都为空 （2）帐号和密码错误 （3）帐号密码都正确	（1）弹出错误提示框不为空 （2）弹出帐号和密码错误提示框 （3）登陆成功	通过
用户注册	（1）帐号密码为空 （2）手机号格式错误 （3）验证码错误	（1）提示输入帐号密码 （2）提示更正手机号格式 （3）提示验证码错误	通过
动态发送	（1）填写内容为空 （2）无权限发送 （3）用户被 ban	（1）提示内容为空 （2）提示无权限 （3）提示被 ban	通过
搜索	（1）不输入搜索关键字 （2）输入搜索关键字	（1）弹出输入关键字提示框 （2）返回对应的结果	通过
音乐上传	（1）歌曲名重复 （2）未填写歌手或专辑 （3）未上传文件	（1）提示已有同名，是否继续上传 （2）提示填写歌手或专辑 （3）提示上传文件	通过

续表 6-3

	（4）上传文件过大 （5）未选择分类	（4）上传失败 （5）提示选择分类或	
音乐信息查看	（1）查看歌手信息 （2）播放音乐 （3）查看专辑	（1）显示歌手信息 （2）音乐播放 （3）显示专辑信息	通过
上传歌单	（1）选择重复歌曲 （2）不填写介绍 （3）不选择标签	（1）弹出歌曲重复添加提示框 （2）弹出填写提示框 （3）弹出选择标签提示框	未通过

6.3.2 测试结果分析

通过测试，检测到在歌单上传时没有去重，后续将后端实体类中吧歌曲列表的类型由 List 改为 Set，问题解决。

同时系统也有些不足，例如当头像为空时，应该使用占位图片，否则会造成空指针异常。初测之外系统应应该进行压力测试来测试系统 QPS 的上线，并且还要进行性能测试来进行系统最大的资源消耗，同时这也是本系统应该优化的方向。

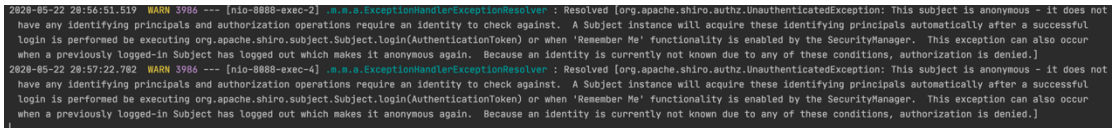


图 6-1 系统异常图

6.4 本章小结

本章详细介绍了系统功能实现和系统测试的方法。首先介绍了测试时使用的测试方法，并且用表格展示了测试的计划和方案，并且设计了测试用例来进行各个模块的功能测试，在最后又对测试结果进行了分析与总结

7 总结与展望

7.1 总结

完全由用户主导的多平台音乐分享系统，用户可以在平台上自由发言，并且上传自己喜欢的歌曲。同时，本系统也实现了类似微博一样的动态分享功能，大家分享音乐的同时，还能进行社交。本文最先介绍了国内外的多平台应用发展情况的不同。并详细分析了系统的业务流盒模型设计，并且对其进行了详细的描述，同时在详细设计部分介绍了系统状态管理与权限管理的实现，在系统运行部分展示了系统各个部分的运行效果，对整个系统采用的 Flutter 框架进行了详细的分析和展示，完全兼容 iOS 和 Android 两端，同时对不同系统的系统交互进型了不同程度的适配，界面简洁，操作友好，实现了查看动态，上传音乐，查看歌曲信息，聆听歌曲和音乐推荐等功能，UI 简洁明了，UX 清晰直观，是个不可多得的优质音乐分享平台。

7.2 展望

本系统是一个全栈式开发项目，涉及多种到技术，同时借用了 github 上的各种开源项目资源。但由于时间和技术问题，有一些还未实现：

（1）应该加入积分收费制度，某些歌曲上传者可以指定价格（积分），就像其他音乐平台的付费歌曲一样，本平台可以用积分代替真实货币。

（2）系统后台有些服务没有采用如 RocketMQ 一样的 Java MQ 队列，当某个查询或请求消耗时间过长时，会造成服务器卡顿。

（3）兼容性不是很好，主要因为 Flutter 在每个平台 build 对应的 TargetSDK 版本较高，希望 Flutter 团队今后能欧改进

（4）推荐算法比较简单，没有针对用户更多的个性化需求进行调整，应该继续进行优化。

（5）界面做得不够友好，应该做的更为美观

参考文献

[1] 谭青. 基于用户评论的音乐推荐系统的研究[D].安徽理工大学,2018.

[2] 邓皓瀚.基于 Flutter 的跨平台移动 APP 开发前景研究[J].信息与电脑(理论版),2019(15):197-199.

[3] 王阅蓁. 移动应用的 web 与 native 混合编程模式研究与实现[D].电子科技大学,2015.

[4] 李一然. 基于 React Native 架构的客户端开发与实现[D].北京邮电大学,2019.

[5] Payne R. Developing in Flutter[M]//Beginning App Development with Flutter. Apress, Berkeley, CA, 2019: 9-27.

[6] Guangmang Cui,Xiaojie Ye,Jufeng Zhao,Liyao Zhu,Ying Chen. Multi-frame motion deblurring using coded exposure imaging with complementary fluttering sequences[J]. Optics and Laser Technology,2020,126.

[7] 陈思,冷雪.微信小程序开发方式对比[J].电子制作,2020(02):52-53+22.

[8] React Native 官网 [EB/OL]. https://facebook.github.io/react-native/, 2020-5-16.

[9] Flutter 官网 [EB/OL]. https://flutter.dev/, 2020-5-16.

[10] Django 官网 [EB/OL]. https://www.djangoproject.com/, 2020-5-16.

[11] Spring 官网 [EB/OL]. https://spring.io/, 2020-5-16.

[12] 庄学松,张智,黄可望.基于 SpringBoot 的短信服务的设计与实现[J].无锡职业技术学院学报,2020,19(02):41-44.

[13] 周虎.一种基于 JWT 认证 token 刷新机制研究[J].软件工程,2019,22(12):18-20.

[14] 王杉文.基于 SpringBoot+Shiro 的权限管理实现[J].电脑编程技巧与维护,2019(09):160-161+173.

[15] Django 官网 [EB/OL]. https://www.djangoproject.com/, 2020-5-16.

[16] 请叫我的全名 cv 工程师.spring 上传文件和下载文件[EB/OL].https://www.jianshu.com/p/cba3e0706849,2020-03-08.

[17] Smith-Cruise.Shiro 基于 SpringBoot +JWT 搭建简单的 restful 服务[EB/OL].https://github.com/Smith-Cruise/Spring-Boot-Shiro,2020-02-29.

[18] 软件测试方法和技术[M]. 清华大学出版社有限公司, 2005.

致谢

时光荏苒，岁月如梭，愉快的四年大学时光转瞬即逝。作为软件学院的一名学生，我更愿意把大学生活比作一段程序。就如同学习每个语言所需的 “Hello World”，刚进入大学的我们还在适应与之前截然不同的学生生活。软件开发中不可或缺的 debug，就像我们平时在学校学习的一点一滴。而软件发布前的健壮性测试，正象征着我们步入社会前，在学校完成的种种考试，这一切的一切都是为了保证软件发布时能够让用户得到最好的体验，也是我们踏入社会后，在职场上为解决我们在工作中遇到的困难打下的基础。

我要感谢南昌航空大学。感谢大一大二曾加入过的 ACM 队，虽然最后没能继续坚持下去，但是我在 ACM 时学到了很多东西，同时也认识了很多非常厉害的学长，甚至是学弟。

我要感谢软件学院的每一位老师。感谢辅导员王超老师，王超老师是一位非常负责任的老师，把每位同学当作自己的孩子一样看待，在生活上与学习上给予我们帮助。我要感谢老师，感谢东软班潘志园老师，让我们能够快速适应东软的生活。感谢段喜龙老师、郑巍老师，还有陈斌全老师，感谢所有教过我的老师们，他们有的严厉，有的风趣，但都教会我很多，这里由衷的感谢！

我还要感谢我们 162031 班上的同学。大一的我还是几乎未出过远门，刚毕业的懵懂少年。但是这 4 年间，同学舍友和我度过的时光使我终身难忘，尤其是 609 寝室。想起我们课余时间一起打游戏，一起玩狼人杀的欢乐时光；临近考试时互相监督，一起在图书馆认认真真学习的辛苦劳累；考试结束后，去餐厅，一起享受火锅的畅快淋漓；那个暑假，我们 7 个人在西安共同度过的青春岁月。我们一起痛苦过，争吵过，也快乐过，共同经历过点点滴滴的小事，也迈过坑坑洼洼的坎坷，也留下过快快乐乐的泪水。很幸庆当时没有听从父母的话选择家乡的大学，我无法想象到没有你们的大学生活会是什么样的，由衷希望每年能见你们几次。

2020，注定是不平凡的一年，新年刚开始全球就笼罩在疫情的阴霾之下，无疑是晴天霹雳。但是经过前线那些白衣天使们的奋战，和一直维持社会的运转普通人的努力，我们现在终于迎来了雨后彩虹。同时对我来说今年也是不平凡的一年，由于疫情一直推迟的雅思考试，可能在下个月终于能够正常开放。希望我接下来的留学生活能够顺利。

最后，用永远离开我们的黑曼巴的一句话，致所有和我同样即将毕业的同学，和那些奋斗在前线的医生和护士们。

Get everything, you have to pay all, conquer all things.

-- Kobe Bean Bryant