

# IMusic 项目开发日志

开发周期：2025年12月18日 - 2025年12月30日

开发者：邬子阳

项目名称：IMusic 在线音乐播放系统

## 2025年12月18日 星期三

### 项目启动与需求分析

工作内容：

今天正式开始 IMusic 项目的开发工作。

#### 1. 需求梳理

- 在线音乐播放系统
- 用户注册、登录功能
- 歌曲搜索和播放
- 收藏和歌单管理

#### 2. 技术选型

- 前端：React + Vite（现代化构建工具，开发体验好）
- 后端：Spring Boot 微服务架构
- 数据库：MySQL 8.0
- 网关：Spring Cloud Gateway

#### 3. 架构设计

决定采用微服务架构，拆分为以下服务：

- gateway-service：API 网关，统一入口
- user-service：用户管理
- music-service：音乐资源服务
- playlist-service：歌单管理

## 2025年12月19日 星期四

### 环境搭建与项目初始化

上午工作：

#### 1. 开发环境配置

- JDK 17
- Maven 3.6+

- Node.js 18
- MySQL 8.0

## 2. 创建项目结构

```
imusic/
└── gateway-service/
└── user-service/
└── music-service/
└── playlist-service/
└── imusic-frontend/
```

## 3. 初始化 Git 仓库

```
git init
git add .
git commit -m "first commit"
```

下午工作：

### 1. 创建 MySQL 数据库

```
CREATE DATABASE imusic_user;
CREATE DATABASE imusic_music;
CREATE DATABASE imusic_playlist;
CREATE USER 'imusic'@'localhost' IDENTIFIED BY 'imusic123';
```

### 2. 初始化各微服务的 Spring Boot 项目

使用 Spring Initializr 创建项目骨架，添加依赖：

- Spring Web
- Spring Data JPA
- MySQL Driver

---

# 2025年12月20日 星期五

## user-service 开发

工作内容：

### 1. 设计用户表

```
CREATE TABLE users (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL
);
```

### 2. 编写实体类和 Repository

```
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String username;
    private String password;
}
```

### 3. 实现用户服务接口

- o POST /api/user/register - 用户注册
- o POST /api/user/login/password - 用户登录
- o POST /api/user/validate - 验证用户

### 4. 配置 application.properties

```
server.port=8081
spring.datasource.url=jdbc:mysql://localhost:3306/imusic_user
spring.jpa.hibernate.ddl-auto=update
```

遇到的问题：

密码目前是明文存储，后续需要加密处理。暂时先完成基本功能。

---

## 2025年12月21日 星期六

### music-service 开发

上午工作：

#### 1. 设计歌曲表

歌曲信息比较丰富，包含元数据：

- o file\_path：音频文件路径
- o cover\_file\_path：封面路径
- o title、artist、album 等

#### 2. 编写 Song 实体类

```
@Entity
@Table(name = "songs")
public class Song {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String filePath;
    private String coverFilePath;
    private String title;
    private String artist;
    // ... 其他字段
}
```

下午工作：

### 1. 实现歌曲搜索功能

- 支持按标题或艺术家搜索
- 空关键词返回全部歌曲

### 2. 配置静态资源服务

- /cover/\*\* 提供封面图片
- /mp3/\*\* 提供音频文件

### 3. 编写 music\_info.json

用于初始化歌曲数据，包含所有歌曲的元信息。

## 2025年12月22日 星期日

## playlist-service 开发

工作内容：

### 1. 设计歌单相关表

- song\_list: 歌单表
- song\_list\_items: 歌单项目表

### 2. 实现歌单管理接口

- POST /api/user/songLists/show - 获取用户歌单
- POST /api/user/songLists/add - 创建歌单
- POST /api/song/add/tosonglist - 添加歌曲到歌单
- POST /api/song/delete/fromsonglist - 从歌单删除

### 3. 服务间调用

playlist-service 需要调用其他服务：

```
@Component
public class UserServiceClient {
    public boolean validateUser(String username, String password) {
        // 调用 user-service 验证用户
    }
}

@Component
public class MusicServiceClient {
    public SongResponse getSongByInfo(...) {
        // 调用 music-service 获取歌曲信息
    }
}
```

## 2025年12月23日 星期一

### gateway-service 开发

工作内容：

#### 1. 配置 Spring Cloud Gateway

```
spring:
  cloud:
    gateway:
      routes:
        - id: user-service
          uri: http://localhost:8081
          predicates:
            - Path=/api/user/**
        - id: music-service
          uri: http://localhost:8082
          predicates:
            - Path=/api/song/search/all, /cover/**, /mp3/**
        - id: playlist-service
          uri: http://localhost:8083
          predicates:
            - Path=/api/song/**, /api/user/songLists/**
```

#### 2. 配置 CORS

在网关层统一处理跨域：

```
spring:
  cloud:
    gateway:
      globalcors:
        corsConfigurations:
          '[/**]':
            allowedOriginPatterns: "*"
            allowedMethods: "*"
            allowedHeaders: "*"
```

### 3. 测试路由转发

使用 Postman 测试各个路由是否正确转发。

**2025年12月24日 星期二**

## 前端项目初始化

上午工作：

### 1. 创建 React 项目

```
cd imusic-frontend
npm init -y
npm install react react-dom
npm install vite @vitejs/plugin-react -D
```

### 2. 配置 Vite

```
export default defineConfig({
  plugins: [react()],
  server: {
    port: 3000,
    proxy: {
      '/api': { target: 'http://localhost:8080', changeOrigin: true },
      '/cover': { target: 'http://localhost:8080', changeOrigin: true },
      '/mp3': { target: 'http://localhost:8080', changeOrigin: true }
    }
  }
})
```

下午工作：

### 1. 设计页面布局

- 顶部：标题 + 导航 + 登录按钮
- 中间：搜索区域 / 收藏列表
- 底部：播放器控制栏

### 2. 创建组件结构

```
src/components/
├── SearchBar.jsx
├── SongList.jsx
├── Player.jsx
├── Favorites.jsx
├── Login.jsx
├── Toast.jsx
└── LoadingOverlay.jsx
```

## 2025年12月25日 星期三

### 前端核心组件开发

上午工作：

#### 1. 开发 SearchBar 组件

- 搜索输入框
- 搜索按钮
- 回车触发搜索

#### 2. 开发 SongList 组件

- 展示歌曲列表
- 显示封面、标题、艺术家
- 播放和收藏按钮

下午工作：

#### 1. 开发 Player 组件

这是最复杂的组件：

- 歌曲封面和信息
- 播放/暂停控制
- 上一曲/下一曲
- 进度条（可拖拽）
- 音量控制
- 时间显示

#### 2. 实现音频播放

```
const audioRef = useRef(null)

useEffect(() => {
  if (audioRef.current && currentSong) {
    audioRef.current.play()
  }
}, [currentSong])
```

遇到的问题：

进度条拖拽时音频会跳动，需要在拖拽过程中暂停更新。

## 2025年12月26日 星期四

### 前端用户功能开发

工作内容：

#### 1. 开发 Login 组件

- 登录/注册模式切换
- 表单验证
- 错误提示

#### 2. 开发 Favorites 组件

- 展示收藏歌曲
- 支持播放和取消收藏

#### 3. 开发 Toast 组件

- 全局消息提示
- 支持 success/error/warning/info 类型

#### 4. 实现登录状态持久化

```
// 保存到 localStorage
localStorage.setItem('username', username)
localStorage.setItem('isLoggedIn', 'true')

// 页面加载时恢复
useEffect(() => {
  const isLoggedIn = localStorage.getItem('isLoggedIn')
  if (isLoggedIn === 'true') {
    // 恢复登录状态
  }
}, [])
```

## 2025年12月27日 星期五

### 前后端联调

上午工作：

#### 1. API 联调测试

- 用户注册
- 用户登录
- 歌曲搜索

添加收藏

查询收藏

删 除 收藏

## 2. 修复问题

问题	解决方案
封面图片不显示	检查路径拼接
跨域请求失败	配置 Gateway CORS
登录状态丢失	使用 Header 传递用户信息

下午工作：

### 1. 优化用户体验

- 添加加载状态
- 添加操作反馈
- 优化错误处理

### 2. 样式调整

- 统一配色
- 优化按钮样式
- 添加过渡动画

---

## 2025年12月28日 星期六

## 功能完善与优化

上午工作：

### 1. 添加播放动画

播放时专辑封面旋转：

```
.player-cover {  
    animation: rotate 20s linear infinite;  
    animation-play-state: paused;  
}  
.player-cover.playing {  
    animation-play-state: running;  
}
```

### 2. 修复播放按钮居中问题

```
.control-btn-play {  
    text-indent: 2px;  
}
```

下午工作：

### 1. 编写启动脚本

创建 start.sh 一键启动所有服务：

- 检查 MySQL
- 创建数据库
- 启动 4 个后端服务
- 启动前端

### 2. 编写停止脚本

创建 stop.sh 停止所有服务。

## 2025年12月29日 星期日

### 测试与文档

上午工作：

#### 1. 完整功能测试

- 注册新用户
- 登录系统
- 搜索歌曲
- 播放音乐
- 添加收藏
- 管理歌单

#### 2. 修复测试中发现的问题

- 歌单为空时的显示问题
- 搜索结果排序优化

下午工作：

#### 1. 编写系统说明文档

- 项目背景
- 功能介绍
- 技术架构
- 数据库设计
- 部署说明

#### 2. 整理代码

- 删除调试代码
- 添加必要注释
- 格式化代码

2025年12月30日 星期一

## 项目收尾

工作内容：

### 1. 最终测试

- 完整流程测试
- 边界情况测试

### 2. 文档完善

- 更新 README
- 完善开发日志

### 3. 代码提交

```
git add .
git commit -m "Complete IMusic microservices music player"
git push origin main
```

## 开发总结

### 项目完成情况

模块	状态	说明
gateway-service	完成	API 网关、路由转发
user-service	完成	用户注册、登录
music-service	完成	歌曲搜索、播放
playlist-service	完成	歌单、收藏管理
前端	完成	React 单页应用

## 技术收获

### 1. 微服务架构

- 学习了服务拆分原则
- 掌握了 Spring Cloud Gateway 配置
- 理解了服务间通信方式

### 2. Spring Boot 3.x

- 学习了 Spring Data JPA
- 掌握了 RESTful API 设计
- 了解了配置管理

### 3. React 19

- 学习了 Hooks 使用
- 掌握了组件化开发
- 了解了状态管理

## 开发时间统计

日期	工作内容	耗时
12月18日	需求分析、架构设计	6h
12月19日	环境搭建、项目初始化	8h
12月20日	user-service 开发	8h
12月21日	music-service 开发	8h
12月22日	playlist-service 开发	8h
12月23日	gateway-service 开发	6h
12月24日	前端项目初始化	8h
12月25日	前端核心组件	8h
12月26日	前端用户功能	8h
12月27日	前后端联调	8h
12月28日	功能完善	6h
12月29日	测试与文档	6h
12月30日	项目收尾	4h
合计		<b>92h</b>

## 附录：常用命令

### 一键启动

```
./start.sh
```

### 一键停止

```
./stop.sh
```

## 手动启动

```
# 后端服务  
cd user-service && mvn spring-boot:run &  
cd music-service && mvn spring-boot:run &  
cd playlist-service && mvn spring-boot:run &  
cd gateway-service && mvn spring-boot:run &  
  
# 前端  
cd imusic-frontend && npm run dev
```

## 访问地址

- 前端: <http://localhost:3000>
- API 网关: <http://localhost:8080>

## 日志查看

```
tail -f logs/user-service.log  
tail -f logs/music-service.log  
tail -f logs/playlist-service.log  
tail -f logs/gateway-service.log
```

---

文档编写日期: 2025年12月30日

---

## 签名

A handwritten signature in black ink, consisting of three characters: '邵' (Shao), '子' (Zi), and '阳' (Yang), written in a cursive style.