目录

[项目简介 3](#_Toc149828338)

[需求分析 3](#_Toc149828339)

[引言 3](#_Toc149828340)

[用户角色 3](#_Toc149828341)

[功能需求 3](#_Toc149828342)

[非功能需求 4](#_Toc149828343)

[假设与约束 5](#_Toc149828344)

[项目架构 6](#_Toc149828345)

[业务流程 7](#_Toc149828346)

[组织形式 8](#_Toc149828347)

[团队成员 8](#_Toc149828348)

[人员分工 9](#_Toc149828349)

[组织沟通与协作 9](#_Toc149828350)

[项目进度安排 9](#_Toc149828351)

[风险管理 10](#_Toc149828352)

[功能模块设计 12](#_Toc149828353)

[爬虫设计 12](#_Toc149828354)

[数据源选择 13](#_Toc149828355)

[需求分析 14](#_Toc149828356)

[总体设计 17](#_Toc149828357)

[详细设计 19](#_Toc149828358)

[编码实现 23](#_Toc149828359)

[数据去重和性能调优 28](#_Toc149828360)

[运行和维护 34](#_Toc149828361)

[算法设计 42](#_Toc149828362)

[背景及意义 42](#_Toc149828363)

[总体设计 42](#_Toc149828364)

[详细设计 44](#_Toc149828365)

[编码实现 44](#_Toc149828366)

[Mook设计 46](#_Toc149828367)

[Mook 46](#_Toc149828368)

[Mook数据的方法 47](#_Toc149828369)

[Python Mook 48](#_Toc149828370)

[总体设计 49](#_Toc149828371)

[SpringAPI 51](#_Toc149828372)

[框架选择 51](#_Toc149828373)

[需求分析 52](#_Toc149828374)

[总体设计 53](#_Toc149828375)

[详细设计 54](#_Toc149828376)

[编码实现 56](#_Toc149828377)

[运行和维护 62](#_Toc149828378)

[Web前后端 65](#_Toc149828379)

[接口设计 65](#_Toc149828380)

[开发过程 72](#_Toc149828381)

[过程调度 97](#_Toc149828382)

[Azkaban 97](#_Toc149828383)

[调度计划 99](#_Toc149828384)

[使用Azkaban实现过程调度 99](#_Toc149828385)

[监控报警 102](#_Toc149828386)

[Prometheus 102](#_Toc149828387)

[使用Prometheus实现指标监控 103](#_Toc149828388)

[使用Python脚本监控Java进程 107](#_Toc149828389)

[测试报告 109](#_Toc149828390)

[爬虫测试 109](#_Toc149828391)

[单元测试 109](#_Toc149828392)

[单元划分 110](#_Toc149828393)

[手工单元测试 111](#_Toc149828394)

[算法模块测试 118](#_Toc149828395)

[SpringAPI测试 123](#_Toc149828396)

[测试方案介绍 123](#_Toc149828397)

[测试用例设计 124](#_Toc149828398)

[测试报告 126](#_Toc149828399)

[Web前后端测试 137](#_Toc149828400)

[流程测试 145](#_Toc149828401)

[总结 148](#_Toc149828402)

# 项目简介

本项目是一个基于Spark的分布式音乐推荐系统，旨在通过大数据算法和分布式计算技术，提供高效准确的音乐推荐服务，以满足用户的个性化需求。

## 需求分析

### 引言

本需求分析报告旨在详细描述基于Spark的分布式音乐推荐系统的功能和特性，以明确用户需求并为系统设计和开发提供指导。本系统旨在为用户提供个性化的音乐推荐服务，通过分析用户的历史行为和偏好，提供符合其口味的音乐推荐，提升用户体验。

### 用户角色

1 普通用户：使用系统提供的音乐搜索和播放功能，获取个性化推荐的音乐。

2 管理员用户：管理系统的用户信息、音乐库和推荐算法等，具有系统配置和维护的权限。

### 功能需求

**1 用户注册和登录**

* 用户可以注册新账号，并通过登录验证身份。
* 用户登录后可享受个性化推荐和其他功能。

**2 音乐搜索**

* 用户可以通过关键词搜索音乐，系统将返回相关的音乐结果。
* 可以支持搜索结果的按照不同条件进行排序和过滤。

**3 音乐播放**

* 用户可以点击音乐结果，进入音乐播放页面进行音乐的在线播放。

**4 个性化推荐**

* 系统根据用户的历史行为和偏好，为用户推荐符合其口味的音乐。
* 推荐算法应考虑音乐分类、用户喜好度等因素，提供个性化推荐结果。

**5 用户喜好管理**

* 用户可以收藏喜欢的音乐，方便快速查找和播放。
* 用户可以取消收藏，管理个人喜好和播放历史。

**6 管理员权限管理**

* 管理员用户可以管理用户的账号信息，包括修改、删除等操作。
* 管理员用户可以管理音乐库，包括添加、编辑、删除音乐信息。

### 非功能需求

#### 性能要求

* 系统需要具备高并发处理能力，能够快速响应用户请求。
* 音乐搜索和推荐功能需要在短时间内生成结果并返回给用户。

#### 可用性要求

* 系统应具备良好的用户界面设计，易于使用和操作。
* 音乐播放功能应稳定可靠，能够流畅地进行音乐播放。

#### 可扩展性要求

* 系统应具备良好的扩展性，能够支持未来的功能扩展和系统优化。

#### 安全性要求

* 用户的个人信息应得到保护，系统应采取相应的安全措施。
* 管理员的权限管理应具备安全性，防止未经授权的访问和操作。

### 假设与约束

* 系统需要连接互联网进行数据的爬取和推荐算法的更新。
* 系统需要具备足够的存储容量来存储音乐库和用户数据。

## 项目架构

本项目采用了分布式计算技术，基于Spark框架，实现了数据的分布式处理和计算。具体架构如下：

数据采集模块：采用爬虫技术获取音乐相关的数据，包括歌曲信息、评论信息、用户信息等。

数据处理模块：对采集到的数据进行清洗和处理，将处理后的数据进行存储和索引。

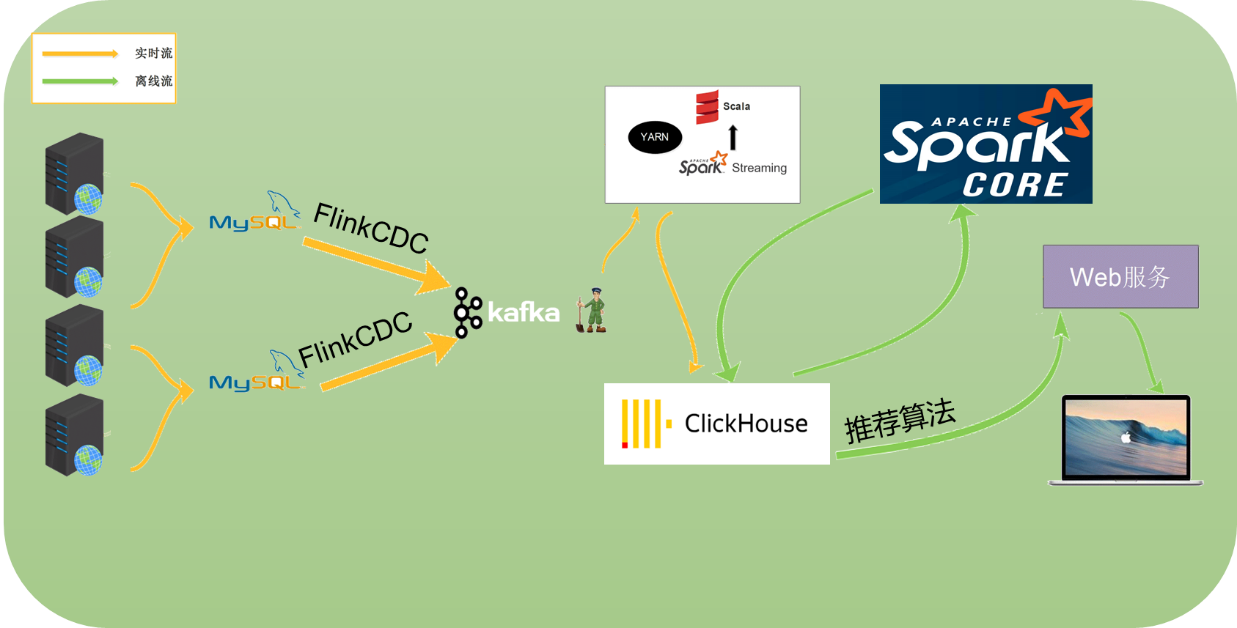
算法模块：采用机器学习算法和推荐算法，对用户的历史行为和偏好进行分析，为用户推荐符合其偏好的音乐。

web前后端模块：提供用户接口，用户可以通过该接口进行音乐的搜索和播放。



系统架构图

## 业务流程



# 组织形式

本项目采用敏捷开发模式，每个迭代周期为两周，包含需求分析、设计、编码、测试、部署和迭代回顾等环节。咱们团队采用周例会和迭代计划会议进行沟通和协调，以保证项目的进展和质量。

该组织架构下包含三种角色，分别是产品负责人（Product Owner）、敏捷教练（Scrum Master）、团队（team）。

产品负责人（Product Owner）简称PO，仅能由一个人来担任，类似产品经理，担任项目的管理者，但不需要管理团队。负责制定产品策略，参与产品架构和技术讨论，负责产品迭代计划的审核与推进。

敏捷教练（Scrum Master）简称SM，仅能由一个人来担任，作为敏捷开发的推动者，需要管理团队，擅长组织和协调。负责组织会议，整理会议纪要，制定迭代计划，推进产品开发。

团队（team）是产品的创造者，是真正的技术执行者。通常包括前端开发工程师、后端开发工程师、测试工程师、UI设计师、运维工程师等。团队队员之间没有头衔、上下级之分，成员之间平等沟通和协作。

## 团队成员

张浩 产品经理

赵龙 敏捷教练

苏文杰 团队成员

李勇江 团队成员

## 人员分工

1. 张浩：负责爬虫部分的开发和调度监控的部署；

2. 赵龙：负责数据库设计和算法部分的开发以及整理文档；

3. 苏文杰：负责Web前后端部分的开发和CDC的开发；

4. 李勇江：负责服务器环境部署和算法部分的开发。

## 组织沟通与协作

为了实现高效的团队沟通与协作，我们将采用以下方式：

1. 每周举行一次团队例会，在例会上分享项目进展、讨论遇到的问题并寻找解决方案。

2. 使用在线沟通工具（如钉钉、微信等）建立团队沟通渠道，方便随时交流和解决问题。

3. 每个迭代周期开始前，进行迭代计划会议，明确本周期的目标和任务分配。

4. 团队成员之间保持持续的沟通和协作，协助彼此解决遇到的问题，确保项目进展顺利。

5. 定期进行迭代回顾，总结经验教训，不断优化工作流程和团队效率。

## 项目进度安排

根据项目需求和团队成员的能力分配，我们拟定以下进度安排：

1. 第一周：进行项目需求分析和功能规划，制定初步的项目计划和迭代周期。

2. 第二周至第四周：进行数据采集模块的开发，包括爬虫技术的研发和数据清洗处理的设计与实现。

3. 第五周至第七周：着手开发算法模块，包括机器学习算法和推荐算法的实现和优化。

4. 第八周至第十周：进行web前后端模块的开发，包括用户接口的设计和实现。

5. 第十一周至第十二周：进行系统的测试和调试，确保各个模块的功能完善和稳定。

6. 第十三周：项目部署和上线准备，包括服务器环境部署、性能优化和安全性测试等。

7. 第十四周：逐步上线并进行用户反馈收集，持续进行迭代优化和功能迭代。

## 风险管理

在项目开发过程中，我们将要面临一些潜在的风险，以确保项目进展顺利：

1. 需求变更风险：对于长期项目，需求的变更是难以避免的，我们将确保及时沟通和调整，尽量减少变更带来的影响。

2. 技术挑战风险：分布式计算和大数据算法是本项目的核心，在技术实现中可能会遇到一些挑战，我们将及时进行技术调研和解决方案讨论。

3. 进度延误风险：对于每个迭代周期，我们会进行合理的时间规划和任务分配，确保能够及时完成计划的工作。

4. 团队协作风险：团队成员之间的沟通和协作是项目成功的关键，我们将重视团队合作，妥善解决团队成员之间的沟通和协作问题。

# 功能模块设计

爬虫设计

在音乐推荐系统的开发过程中，设计爬虫获取数据的阶段具有重要的意义。该阶段的主要目标是通过爬取各种音乐平台、社交媒体和网站上的数据，建立一个全面、多样化的音乐数据库。

这个阶段的核心效果是获取大量的音乐相关数据，包括歌曲信息、艺术家资料、歌词、歌曲标签、用户评论等。通过将这些数据组织和分析，可以构建一个全面的音乐库，为后续的推荐算法提供丰富的素材和依据。

这个阶段对后续开发的影响非常重要。首先，一个全面、多样化的数据集是进行推荐算法训练的基础。只有拥有丰富的数据样本，算法才能更好地理解用户的偏好和行为，从而提供个性化的推荐。

其次，爬虫获取的数据也可以用于数据清洗和预处理。通过对数据进行清洗和标准化处理，可以提高数据的质量和准确性，减少后续算法开发中可能出现的问题。

此外，数据获取阶段还可以帮助发现新的音乐趋势和热门曲目，为推荐系统的更新和优化提供参考。通过观察和分析数据，可以及时调整推荐策略，使系统的推荐结果更符合用户的兴趣和口味。

总之，设计爬虫获取数据的阶段是音乐推荐系统开发中不可或缺的一步。它为后续算法训练、数据清洗和优化提供了基础，并且对于打造更好的用户体验和个性化推荐非常关键。

数据源选择

QQ音乐和网易云音乐都是知名的音乐平台，它们各自具有不同的特点和优势。在选择唯一数据来源时，需要考虑这些因素并做出合适的选择。下面是对两个数据源的特点和网易云音乐的优势的分析：

1. **QQ音乐的特点和优势：**
   * 数据相对齐全：QQ音乐作为中国领先的音乐平台之一，拥有广泛的音乐资源和大量用户。它提供了丰富的音乐库，包括热门流行歌曲、专辑、歌手等。
   * 音乐版权：QQ音乐在版权方面相对较强，与国内外多家音乐公司有合作关系，因此可以提供具有版权的高质量音乐资源。
2. **网易云音乐的特点和优势：**
   * 开源数据接口服务器：网易云音乐提供了开源数据接口服务器，这是对开发者非常友好的特点。开发者可以通过这些接口获取音乐数据，便于数据的获取和整合。
   * 用户互动和社交性：网易云音乐注重用户互动和社交功能的建设，用户可以在平台上评论、点赞、分享音乐，形成良好的社区氛围。这些用户行为数据可以为个性化推荐提供更多的依据。

**综合考虑，选择网易云音乐作为唯一的数据来源有以下几点论证：**

* 开源数据接口服务器：网易云音乐的开源数据接口服务器对开发者友好，提供了便捷的数据获取方式，有利于系统的开发和维护。
* 用户互动和社交性：网易云音乐的用户互动功能和社交性更加突出，这意味着可以获取到更多的用户行为数据，从而为个性化推荐算法提供更准确的依据。
* 数据相对齐全：虽然QQ音乐在数据方面有一定的优势，但网易云音乐也拥有广泛的音乐资源，并且其开源数据接口服务器可以通过其他渠道补充数据，以弥补潜在的不足。

因此，在综合考虑开发者友好性、用户互动和社交性以及数据资源等因素后，选择网易云音乐作为唯一的数据来源是合理的决策。

需求分析

1. 目标：获取音乐信息和对应标签的数据，以满足后续的推荐服务需求。
2. 数据源：选择网易云音乐作为唯一的数据来源。
3. 数据获取方式：
   * 利用网易云音乐的开源数据接口服务器，通过 API 请求，获取音乐数据。
   * 可以根据不同的需求，获取特定类型的音乐数据，如热门榜单、推荐歌单、某个歌手的专辑等。
   * 获取的音乐数据应该包括音乐ID和对应的标签。
4. 音乐ID和标签的存储：
   * 设计数据库模式，存储音乐ID和相应的标签。
   * 数据库中的音乐ID应该是唯一的，方便后续的数据检索和操作。
   * 标签可以根据需求进行分类和组织，例如根据流派、风格、专题等进行标签分类。
5. 数据更新和维护：
   * 设置定时任务，定期更新数据库中的音乐数据，保证数据的及时性和准确性。
   * 对于新增的音乐数据，需要通过爬虫或API请求获取对应的音乐ID和标签，并插入到数据库中。
   * 可以考虑建立自动化流程，实现数据爬取、处理和存储的自动化操作。
6. 数据查询和检索：
   * 提供合适的接口和查询功能，便于根据标签或其他条件进行音乐数据的查询和检索。
   * 根据用户的请求和需求，通过查询数据库获取相应的音乐ID和标签数据。
7. 数据质量和准确性：
   * 数据获取过程中，需要处理重复、缺失或错误的数据，确保数据的质量和准确性。
   * 可以通过数据清洗和验证等方式，对获取到的音乐ID和标签进行处理和校验，确保数据的完整性和一致性。

总结：以上需求分析主要从数据源选择、数据获取方式、数据存储、数据更新维护、数据查询和检索以及数据质量方面对获取音乐ID和对应标签的需求进行了分析。根据这些需求，可以设计和实现相应的音乐数据获取系统，以满足后续的推荐服务的需求。

**技术栈**

1. **网络爬虫：**使用网络爬虫技术从网易云音乐获取音乐数据。网络爬虫模块可以使用Python中的各种爬虫框架，如Scrapy等。
2. **Scrapy：**Scrapy是一个强大的Python爬虫框架，它提供了一套丰富的API和工具，使得爬虫开发更加高效和灵活。
3. **XPath：**XPath是一种用于在XML和HTML文档中进行导航和提取数据的查询语言。在爬虫过程中，XPath可以用于从网页中提取指定的数据。
4. **JSON：**JSON是一种常用的数据交换格式，它以易于阅读和编写的方式存储数据。在爬虫过程中，爬取的数据通常以JSON格式进行处理和存储。
5. **CSV：**CSV是一种常用的纯文本格式，以逗号作为字段的分隔符。在数据处理过程中，可以使用CSV格式将数据导出或导入到文件中，方便进行数据存储和交换。
6. **Pandas：**Pandas是一个强大的数据处理和分析库，它提供了高性能的数据结构和数据分析工具。在数据处理和分析阶段，可以使用Pandas进行数据的清洗、转换和分析。
7. **MySQL：**MySQL是一种流行的关系型数据库管理系统，提供了高效可靠的数据存储和查询功能。在数据存储的阶段，可以使用MySQL作为数据库管理系统，存储爬取的音乐ID和标签等数据。

综合上述技术栈，可以使用网络爬虫框架Scrapy进行网易云音乐数据的爬取，使用XPath进行数据提取，将爬取的数据以JSON格式进行处理和存储。在数据分析和处理阶段，可以使用Pandas进行数据的清洗、转换和分析，并将数据导出或导入到CSV格式的文件中。最终，可以使用MySQL作为数据库管理系统，存储爬取的音乐ID和对应的标签数据。

总体设计

**数据表设计**

分类表

|  |  |  |
| --- | --- | --- |
| 字段名 | 字段属性 | 注释 |
| id | Unsigned int, Primary key | 自增主键 |
| Cate | Varchar(255) | 主分类 |
| Scate | Varchar(255) | 子分类 |

音乐分类表

|  |  |  |
| --- | --- | --- |
| 字段名 | 字段属性 | 注释 |
| id | Unsigned bigint, Primary key | 自增主键 |
| url | Varchar(255) | 音乐地址，网易云音乐id |
| title | Varchar(255) | 音乐标题 |
| cate | Varchar(255) | 音乐子分类 |

音乐信息表

|  |  |  |
| --- | --- | --- |
| 字段名 | 字段属性 | 注释 |
| id | Unsigned bigint, Primary key | 自增主键 |
| music\_id | Varchar(255) | 网易云音乐id |
| music\_name | Varchar(255) | 音乐标题 |
| url | Varchar(255) | 音乐地址 |

艺术家表

|  |  |  |
| --- | --- | --- |
| 字段名 | 字段属性 | 注释 |
| id | Unsigned bigint, Primary key | 自增主键 |
| artist | Varchar(255) | 网易云艺术家id |
| name | Varchar(255) | 艺术家名 |

音乐-艺术家表

|  |  |  |
| --- | --- | --- |
| 字段名 | 字段属性 | 注释 |
| id | Unsigned bigint, Primary key | 自增主键 |
| music\_id | Varchar(255) | 网易云音乐id |
| atrist\_id | Varchar(255) | 艺术家id |

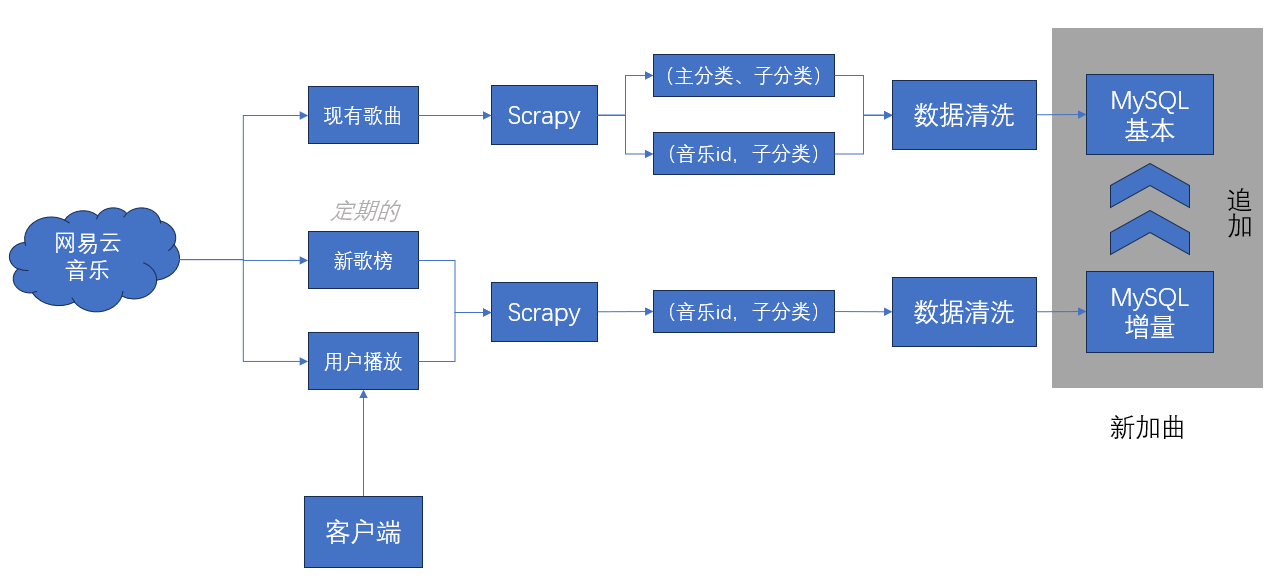
专辑表

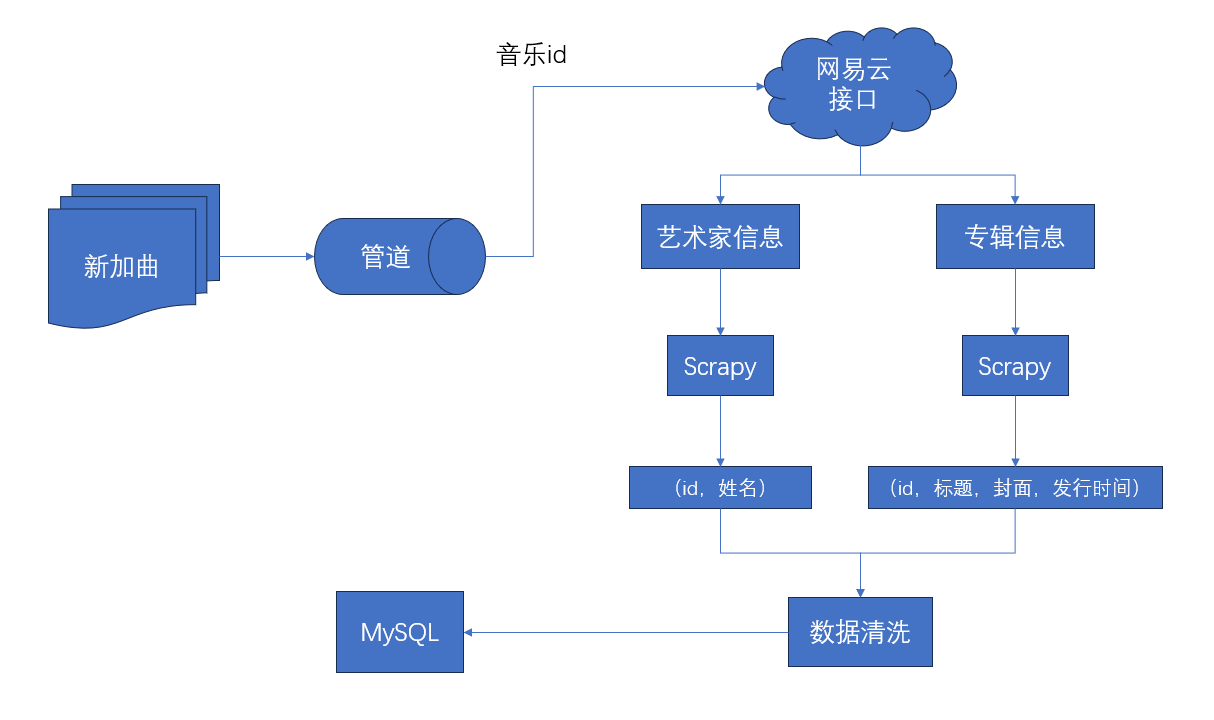
|  |  |  |
| --- | --- | --- |
| 字段名 | 字段属性 | 注释 |
| id | Unsigned bigint, Primary key | 自增主键 |
| album | Varchar(255) | 网易云专辑id |
| title | Varchar(255) | 专辑标题 |
| pic | Varchar(255) | 专辑封面地址 |
| time | Datetime | 发布时间 |

音乐-艺术家表

|  |  |  |
| --- | --- | --- |
| 字段名 | 字段属性 | 注释 |
| id | Unsigned bigint, Primary key | 自增主键 |
| music\_id | Varchar(255) | 网易云音乐id |
| album\_id | Varchar(255) | 专辑id |

**业务流程**





详细设计

**基本数据获取**

这个流程使用Scrapy作为网络请求工具。Scrapy是一个强大的Python爬虫框架，它提供了方便的异步网络请求和XPath解析功能，非常适合用于爬取网易云音乐的歌单数据。

具体的流程如下：

1. 创建一个Scrapy项目，并配置合适的爬虫设置。
2. 在爬虫类中定义start\_urls，设置为网易云音乐主页地址。
3. 在爬虫类的parse方法中，解析主页的HTML代码，使用XPath提取歌单列表页链接。可以使用Scrapy提供的Selector和xpath方法来实现。
4. 在parse方法中，访问歌单列表页链接，在回调函数中处理列表页的HTML源码。
5. 在回调函数中，使用XPath解析歌单列表页，提取出歌单信息和分类方式的相关数据。将分类方式写入CSV文件，并依次访问歌单链接。
6. 在回调函数中，继续使用XPath从歌单页的HTML源码中解析出歌单标签和音乐列表的相关信息。
7. 将解析得到的音乐地址、音乐标题和标签整理成字典形式，并使用Scrapy提供的yield语句将数据传递给管道进行处理。
8. 创建管道类，在管道类的process\_item方法中，将音乐数据写入CSV文件。
9. 使用Pandas加载CSV文件，进行去重和去空值的清洗操作，生成清洗后的结果。
10. 连接到MySQL数据库，在数据库中创建表格用于存储音乐数据。使用Python的MySQL库，将清洗后的音乐数据插入到MySQL数据库表格中。

以上流程利用Scrapy作为网络请求工具，实现了从网易云音乐获取歌单数据并存储到MySQL数据库的功能，同时使用XPath进行HTML解析，并利用Pandas进行数据清洗操作。

**增量数据获取**

这个流程用Scrapy作为网络请求工具。Scrapy是一个强大的Python爬虫框架，它提供了方便的异步网络请求和XPath解析功能，非常适合用于爬取网易云音乐的歌曲数据。

具体的流程如下：

1. 创建一个Scrapy项目，并配置合适的爬虫设置。
2. 在爬虫类中定义start\_urls，设置为网易云音乐主页地址。
3. 在爬虫类的parse方法中，解析主页的HTML代码，使用XPath提取榜单页链接。可以使用Scrapy提供的Selector和xpath方法来实现。
4. 在parse方法中，访问榜单页链接，在回调函数中处理榜单页的HTML源码。
5. 在回调函数中，使用XPath解析榜单页，提取出新歌榜的链接，访问新歌榜的链接。
6. 在回调函数中，使用XPath解析新歌榜，依次请求新歌链接，获取音乐页的HTML源码；对于用户播放记录，直接请求到对应音乐页的HTML源码。
7. 在回调函数中，使用XPath解析音乐页源码，提取出包含该歌曲的歌单信息和标签。
8. 将解析得到的音乐地址、音乐标题和标签整理成字典形式，并使用Scrapy提供的yield语句将数据传递给管道进行处理。
9. 创建管道类，在管道类的process\_item方法中，将音乐数据写入MySQL数据库，并在写入之前确保该记录在数据表中不存在，以避免重复记录。
10. 对于没有被任何歌单收录的歌曲，手动打上另类、独立的标签。

以上流程利用Scrapy作为网络请求工具，实现了从网易云音乐获取新歌数据并存储到MySQL数据库的功能，同时使用XPath进行HTML解析，并在写入数据之前进行去重判断。使用Scrapy框架的异步请求和优雅的数据流处理机制来处理大量的数据，可以让爬虫更加高效和稳定。

**音乐信息获取**

这个流程分为以下几个步骤：

1. 从磁盘读取新加曲编号列表：首先，从磁盘上读取包含新加曲编号的文件。这个文件可以是一个文本文件或者其他适合存储编号的文件格式。读取文件并获取编号列表。
2. 使用网易云接口查询歌曲、艺术家和专辑信息：使用网易云的接口，将每个编号作为参数，逐个查询歌曲的艺术家和专辑信息。发送请求至网易云接口，并解析返回的数据，获取艺术家和专辑信息。
3. 检查数据是否存在：在将数据保存到数据库之前，需要进行检查以确保在数据库表中没有重复记录。可以根据艺术家编号、专辑编号以及歌曲编号来进行检查。
4. 保存艺术家信息：将查询到的艺术家信息（艺术家编号，艺术家名）保存到艺术家信息表中。如果艺术家已存在于表中，则不进行插入操作。
5. 保存专辑信息：将查询到的专辑信息（专辑编号，标题，封面，发行时间）保存到专辑信息表中。同样地，如果专辑已存在于表中，则不进行插入操作。
6. 保存歌曲-艺术家关联信息：将歌曲的编号和艺术家的编号保存到歌曲-艺术家关联表中。检查关联关系是否已存在于表中，避免重复插入。
7. 保存歌曲-专辑关联信息：将歌曲的编号和专辑的编号保存到歌曲-专辑关联表中。同样地，检查关联关系是否已存在于表中，避免重复插入。

以上是整个流程的步骤描述，从磁盘读取新加曲编号，通过网易云接口查询歌曲、艺术家和专辑信息，并将这些信息保存到对应的数据库表中，同时在写入之前确保记录在数据表中不存在。通过这个流程，可以实现从网易云接口获取详细歌曲信息并进行持久化存储的功能。

编码实现

此部分介绍使用Python语言基于Scrapy框架编写爬虫的实现步骤和相关代码示例。通过编写Scrapy爬虫，可以实现从目标网站获取数据，并将数据存储到指定的存储介质中。

**环境准备**

使用以下命令安装Scrapy框架：

|  |
| --- |
| **pip install scrapy** |

确保已安装Python和Scrapy框架，并保证所需的依赖库已安装。

**创建Scrapy项目**

使用Scrapy提供的命令行工具创建一个新的Scrapy项目：

|  |
| --- |
| **scrapy startproject project\_name** |

**定义爬虫**

在Scrapy项目中，进入爬虫项目目录，创建一个爬虫文件，并打开编辑：

|  |
| --- |
| **cd project\_name**  **scrapy genspider spider\_name website\_url** |

**编写爬虫代码**

在打开的爬虫文件中，根据目标网站的页面结构和数据需求，编写爬虫代码。主要包括以下几个部分：

* 定义爬虫类：继承自scrapy.Spider，定义爬虫名称、起始URL等信息。
* 编写请求和解析函数：在爬虫类中，编写start\_requests和parse函数，用于发送请求和解析页面数据。
* 编写数据提取规则：使用XPath或CSS选择器等方法，提取目标数据的规则，将数据提取为字典或Scrapy Item对象。
* 处理下一页和多页面：在解析函数中，根据需要编写代码来处理翻页和处理多个页面的情况。
* 数据存储：定义数据存储方式，可以选择将数据存储到本地文件、数据库或其他外部存储介质中。

以下是使用Scrapy框架编写某一部分爬虫的具体细节，用于从网易云音乐抓取音乐更新信息：

1. 创建Spider类：

|  |
| --- |
| class MusicUpdateSpider(scrapy.Spider):      name = "music\_update"      allowed\_domains = ["music.163.com"]      start\_urls = ["https://music.163.com"] |

这个Spider类继承自Scrapy的Spider类，并定义了爬虫的名称（name）、抓取的目标域名（allowed\_domains）和起始URL（start\_urls）。

1. 解析起始URL：

|  |
| --- |
| def parse(self, response):      next\_url = '/discover/toplist'      au = response.urljoin(next\_url)      yield scrapy.Request(au, callback=self.parse\_new) |

在这个方法中，解析起始URL的响应，提取下一个URL，并构造新的请求（使用scrapy.Request），将请求传递给parse\_new方法进行处理。

1. 解析新歌榜URL：

|  |
| --- |
| def parse\_new(self, response):      next\_url = response.xpath('//div[@id="toplist"]//img[@alt="新歌榜"]/../../a/@href').get()      au = response.urljoin(next\_url)      yield scrapy.Request(au, callback=self.parse\_music\_list) |

在这个方法中，解析新歌榜URL页面的响应，提取音乐列表URL，并构造新的请求，将请求传递给parse\_music\_list方法进行处理。

1. 解析音乐列表URL：

|  |
| --- |
| def parse\_music\_list(self, response):      li\_music\_url = response.xpath('//div[@id="song-list-pre-cache"]//li/a/@href').getall()      for url in li\_music\_url:          au = response.urljoin(url)          yield scrapy.Request(au, callback=self.parse\_music) |

在这个方法中，解析音乐列表URL页面的响应，提取每首歌曲的URL，并构造新的请求，将请求传递给parse\_music方法进行处理。

1. 解析音乐信息：

|  |
| --- |
| def parse\_music(self, response):      url = f'/song?id={parse\_qs(urlparse(response.url).query)["id"][0]}'      title = response.xpath('//meta[@property="og:title"]/@content').get()      li\_playlist\_url = response.xpath('//span[text()="包含这首歌的歌单"]/../../ul//div[@class="info"]//a[@data-res-type="13"]/@href').getall()      if not li\_playlist\_url:          for tag in ('另类', '独立'):              item = Music163Item()              item['url'] = url              item['title'] = title              item['cate'] = tag              yield item      for next\_url in li\_playlist\_url:          au = response.urljoin(next\_url)          yield scrapy.Request(au, callback=lambda x: self.parse\_music\_cate(x, url, title)) |

在这个方法中，解析音乐URL页面的响应，提取音乐的URL、标题和包含该首歌的歌单URL。如果没有找到包含歌单的URL，则创建包含默认分类标签的Music163Item对象并返回。如果找到了包含歌单的URL，则对每个URL构造新的请求，将请求传递给parse\_music\_cate方法进行处理。

1. 解析歌单分类信息：

|  |
| --- |
| def parse\_music\_cate(self, response, url, title):      li\_tags= response.xpath('//div[@class="tags f-cb"]/a//text()').getall()      for tag in li\_tags:          item = Music163Item()          item['url'] = url          item['title'] = title          item['cate'] = tag          yield item |

在这个方法中，解析包含歌单URL页面的响应，提取歌单的分类标签，并创建包含分类标签的Music163Item对象并返回。

以上代码主要通过不同的解析方法逐层读取和处理目标网页的内容，将抓取到的信息组装成项（Music163Item），并通过yield返回给Scrapy框架进行处理和存储。接下来可以根据实际需求，进一步处理和存储这些项的信息。

**配置项目和管道**

在Scrapy项目中，修改settings.py文件，配置项目相关设置，包括User-Agent、代理、延时等。同时，配置数据存储的管道(Pipeline)，决定数据如何处理和存储。

**运行爬虫**

进入Scrapy项目的根目录，执行以下命令来运行爬虫：

|  |
| --- |
| **scrapy crawl spider\_name** |

其中，spider\_name是在定义爬虫时创建的爬虫的名称。

**数据存储与验证**

根据在编写爬虫代码时定义的数据存储方式，在Scrapy项目的管道中实现数据存储的逻辑。同时，进行数据验证和一致性检查，确保获取的数据准确、完整。

**日志记录与错误处理**

通过Scrapy框架提供的日志功能，可以记录爬取过程中的日志信息和错误信息。根据需要，编写错误处理机制来处理爬取过程中的异常情况和错误。

通过上述步骤，可以基于Scrapy框架使用Python语言编写爬虫，实现从目标网站获取数据并将数据存储到指定的存储介质中。在编码实现过程中，需要根据目标网站的页面结构和数据需求，编写相应的爬虫代码来实现数据的抓取、解析和存储。

数据去重和性能调优

数据去重和性能调优在数据处理和数据库管理中具有重要的意义。以下是它们的主要作用和好处：

**数据去重的意义**

- 数据一致性：数据去重可以消除重复的数据，确保数据的一致性和准确性。这对于数据分析、决策制定和业务运营都是至关重要的。

- 存储空间节省：重复的数据占据了存储空间，通过去重可以减少存储成本，提高存储效率。

- 数据质量提升：去除重复数据可以提高数据的质量，减少错误和冗余。

- 提高查询效率：去除重复数据可以减少不必要的数据读取和处理操作，从而提高查询和分析的效率。

**性能调优的意义**

- 提升系统响应速度：通过性能调优，可以减少查询响应时间，提升系统的实时性和用户体验。

- 提高系统并发能力：性能调优可以提高系统的并发处理能力，使其能够同时处理更多的请求。

- 减少资源消耗：通过优化数据库和代码，可以减少对计算资源、内存和存储的占用，降低运维成本。

- 增强可扩展性：性能调优可以使系统更具可扩展性，支持更大规模的数据和用户量的处理。

综上所述，数据去重可以提高数据质量、节省存储空间并提高查询效率，而性能调优则可以提升系统的响应速度、并发能力和资源利用效率。这些都是优化数据处理和数据库管理的重要手段，对于提高工作效率、降低成本和提升用户体验都具有积极的影响。

**数据去重**

由于原始的音乐信息数据中存在有重复记录，为确保数据的准确性和一致性，同时为后续的数据处理和分析打下良好的基础，需要对数据进行规范化、去重处理。以下是对MySQL音乐表进行数据去重操作的过程：

SQL

|  |
| --- |
| INSERT INTO music(music\_id, music\_name, url)  SELECT DISTINCT SUBSTRING(music\_cate.url,10) AS music\_id, title AS music\_name, CONCAT('https://music.163.com', music\_cate.url) AS url  FROM music\_cate; |

这条语句从 music\_cate 表中选择数据，并将其插入到 music 表中的相应字段中。具体来说，它会：

- 使用 DISTINCT 关键字确保选择的结果中没有重复的记录；

- 使用 SUBSTRING 函数截取 music\_cate.url 字段的子串，从第 10 个字符开始截取，并将其作为 music\_id 的值；

- 将 music\_cate 表中的 title 字段的值赋给 music\_name 字段；

- 使用 CONCAT 函数拼接字符串 'https://music.163.com' 和 music\_cate.url 字段的值，并将结果作为 url 字段的值。

这样就能得到去重后结果，并将其保存在music表中。

去重效果

|  |  |
| --- | --- |
| 去重前 | 去重后 |
|  |  |

189146条重复数据被去除

**性能调优**

数据库性能调优是指通过对数据库系统进行优化和调整，以提高其处理能力、查询性能和响应速度的过程。以下是数据库性能调优的一般步骤和相关策略：

1. 评估当前性能：首先需要对当前数据库系统进行评估和分析，包括数据库结构、查询语句、索引使用、系统配置等方面。可以使用性能监控工具或执行性能测试来获取相关指标和数据。

2. 优化数据库结构：通过规范化、去除冗余数据和设计合理的数据模型，可以提升数据库的查询性能和存储效率。优化表结构、字段类型和关系设计，减少数据冗余和不必要的联接操作。

3. 索引优化：通过创建适当的索引，可以加速数据库的查询操作。需要根据查询的频率和类型来选择合适的索引策略，包括单列索引、组合索引、覆盖索引等，避免过多或不必要的索引。

4. 查询优化：对查询语句进行优化，包括选择合适的查询方式、使用优化器提示、避免全表扫描等。可以利用查询分析工具分析查询的执行计划，根据情况进行优化。

5. 优化系统配置：调整数据库的系统参数和配置，包括缓冲区大小、并发连接数、日志记录等。根据具体的数据库系统，进行性能调优相关配置。

6. 优化存储引擎：根据具体的数据库系统，选择合适的存储引擎，如InnoDB、MyISAM等，并根据具体需求进行相关优化设置。

7. 数据库分区和分片：对庞大的数据进行分区或分片，可以提高数据库的查询和存储效率，加快响应速度。根据具体业务需求和数据特点，进行合理的分区和分片策略。

8. 定期维护和监控：定期执行数据库维护任务，如数据清理、空间释放、索引重建等，保持数据库的健康状态。同时，实施数据库性能监控，并进行性能分析和优化。

这里考虑对music表中的music\_name字段建立全文索引来加速数据库的查询操作。

SQL

|  |
| --- |
| ALTER TABLE music ADD FULLTEXT music\_name (music\_name); |

这句语句会在 music 表的 music\_name 字段上创建一个全文索引。通过 ADD FULLTEXT 子句可以在指定字段上创建全文索引。

全文索引

全文索引是一种用于有效搜索和匹配文本数据的技术。在SQL中，可以通过使用 `MATCH` 和 `AGAINST` 关键字来进行全文索引查询。

以下是一个示例查询，演示如何使用全文索引查询数据：

|  |
| --- |
| SELECT music\_id, music\_name, url  FROM music  WHERE MATCH(music\_name) AGAINST('关键词'); |

在上述查询中，假设我们想要通过 `music\_name` 字段进行全文索引搜索，关键词是 `**关键词**`。

- **MATCH(music\_name)**：用于指定需要进行全文索引搜索的字段。

- **AGAINST('关键词')**：指定了要匹配的关键词。

这条查询语句将返回符合关键词匹配的结果集，包括 `music\_id`、`music\_name` 和 `url` 字段的值。

为了使全文索引起作用，需要确保数据库表上的相应字段已经创建了全文索引。在上述代码中，我们使用 `ALTER TABLE` 语句在 `music` 表的 `music\_name` 字段上创建了全文索引。

另外，对于全文索引的匹配行为，还可以使用一些额外的参数和修饰符，例如设置最小词长度、布尔方式匹配等。具体取决于所使用的数据库管理系统和全文索引实现。

**性能对比**

进行性能对比的目的是评估和比较不同优化方案的效果，以确定哪种方案对系统性能产生了最显著的改进。具体来说，性能对比可以帮助我们实现以下目标：

1. 评估优化方案的有效性：通过对比不同优化方案之间的性能差异，可以确定哪种方案对系统性能的提升效果最好。这有助于选择最佳的优化策略并避免不必要的开销。

2. 指导性能调优工作：通过性能对比，我们可以确定系统的瓶颈所在，找出需要优化的关键区域。这有助于针对具体问题进行有针对性的调优工作，提高整体系统性能。

3. 为决策提供依据：性能对比可以提供数据支持，帮助决策者在不同的优化方案之间做出明智的选择。通过定量的性能指标和数据分析结果，决策者可以更加准确地评估投资回报率，并决定是否值得进行特定的优化工作。

4. 监控系统变化：进行性能对比，可以跟踪系统改进的效果并监控系统的性能变化。这样可以及时发现问题和异常，并采取相应的措施进行修复和调整。

总而言之，性能对比的目的是为了评估优化方案、指导性能调优工作、为决策提供依据和监控系统变化。通过定量的对比分析，我们可以获得有关系统性能改进的宝贵信息，以便做出明智的决策并持续优化系统的性能。

测试方案

编写SQL语句，查询同一内容，对比不同方式下查询效率之高低。查询效率通过响应时间衡量，响应时间越短则效率越高。

确保数据库连接正常，确保每个查询都是第一次调用，避免MySQL读取缓存影响测试结果。

测试环境

* Windows 11家庭版
* MySQL 8.0.33

查询SQL

未去重的查询SQL

SELECT DISTINCT \* FROM music WHERE music\_name LIKE '%等一分钟%'

去重后的查询SQL

SELECT \* FROM music WHERE music\_name LIKE '%等一分钟%'

不使用全文索引的查询SQL

SELECT \* FROM music WHERE music\_name LIKE '%等一分钟%'

全文索引查询SQL

SELECT \* FROM music WHERE MATCH(music\_name) AGAINST('等一分钟')

测试结果

|  |  |
| --- | --- |
| 去重前 | 去重后 |
| SELECT DISTINCT \* FROM music WHERE music\_name LIKE '%等一分钟%'  > OK  > 查询时间: 1.381s | SELECT \* FROM music WHERE music\_name LIKE '%等一分钟%'  > OK  > 查询时间: 0.374s |

性能提升近300%

|  |  |
| --- | --- |
| 未使用全文索引 | 使用全文索引 |
| SELECT \* FROM music WHERE music\_name LIKE '%等一分钟%'  > OK  > 查询时间: 0.376s | SELECT \* FROM music WHERE MATCH(music\_name) AGAINST('等一分钟')  > OK  > 查询时间: 0.077s |

性能提升近400%

运行和维护

在运行和维护阶段，主要工作包括使用调度器调度爬虫的两个流程（分类信息获取和音乐信息获取）定期执行，并确保爬虫能够持续稳定地抓取所需的数据。以下是该阶段的主要工作内容和解决方案的简述：

1. 调度器定期执行爬虫流程：

* 通过配置调度器，设置定期运行爬虫的时间间隔或触发条件，以确保在预定时间内启动爬虫。
* 确保爬虫的运行时间不会与目标网站的维护时间冲突，导致异常或无法访问的情况。
* 定期检查爬虫的状态和性能，如是否正常运行、处理速度是否满足需求等。
* 根据需要，调整调度器的配置和参数，以最大限度地提高爬虫的吞吐量和稳定性。

1. 运行失败时向开发者发送告警：

* 实现异常处理机制，当爬虫运行过程中出现问题或错误时，及时捕获并向开发者发送告警信息。
* 可通过邮件、短信或即时通知等方式，将错误日志和异常信息发送给开发者，以便进行及时处理。
* 分析和处理告警信息，找出问题的根本原因，并尽快修复和改进代码，以确保爬虫的稳定和连续运行。

通过定期执行和调度，以及及时处理异常和错误，可以保证爬虫的稳定性和持续性，从而确保能够持续稳定地抓取所需的数据。同时，通过分析问题原因和持续改进措施，可以提高爬虫的健壮性和适应能力，以应对目标网站的变化和需求的变更。

**调度计划**

**任务流程名称:** 音乐数据更新

**执行时间:** 每天凌晨12点

**任务顺序:**

1. 音乐分类信息获取
   * 调度时间: 每天凌晨12点
   * 预计执行时间: 约30分钟
2. 音乐信息获取
   * 调度时间: 音乐分类信息获取任务结束后，即开始时间为音乐分类信息获取任务的结束时间
   * 预计执行时间: 约1小时

**任务说明:**

* 音乐分类信息获取任务负责更新音乐的分类信息。
* 音乐信息获取任务需要在音乐分类信息获取任务完成之后进行，以便基于更新的分类信息进行音乐信息的抓取和更新。

**任务注意事项:**

* 确保每个任务在预计执行时间内完成，以免影响后续任务的执行。
* 根据实际情况和任务的复杂度，适当调整任务的执行时间和预计执行时间，以确保任务能够按时完成。
* 监控任务的执行状态和运行时间，如有异常情况或延迟，及时调整任务调度、优化代码或增加资源等。
* 在任务执行过程中，记录任务执行日志和异常情况，以便后续分析和排查问题。

以上调度计划基于每天凌晨12点执行两个过程的顺序调度，确保音乐信息获取任务在音乐分类信息获取任务完成后进行。根据实际需求和情况，可以自行调整任务的执行时间和预计执行时间，以适应特定的业务需求。

**通过Windows任务计划程序实现调度**

Windows任务计划程序

Windows提供的任务计划程序功能可以帮助我们实现过程调度。任务计划程序是Windows操作系统的一个内置工具，它允许用户在特定时间或事件触发时自动执行任务。

通过任务计划程序，操作系统用户可以按照特定的时间计划、触发器条件和操作来创建和管理定时任务。还可以设置任务的重复间隔、执行时间和日期、执行操作的用户名和密码等。

任务计划程序还提供了灵活的触发器选项，例如：

* 指定任务在每天（如每天凌晨12点）的特定时间触发。
* 指定任务在特定日期（如每周的某一天）触发。
* 指定任务在特定事件（如特定的系统事件）触发。

通过任务计划程序，用户可以执行各种任务，包括执行脚本、运行应用程序、执行系统维护任务等。我们可以根据自己的需求和想要实现的过程调度，在任务计划程序中创建相应的任务并配置相应的操作。

但值得注意的是，任务计划程序功能可能会因Windows操作系统的不同版本而有所不同。因此在使用任务计划程序时，熟悉使用的Windows操作系统版本的相关文档和说明是必要的。

调度实现

以下是通过Windows任务计划程序实现过程调度，每天凌晨12点自动执行爬虫任务（获取音乐分类信息，再获取音乐信息）的步骤：

1. 打开任务计划程序：
   * 打开Windows的"开始"菜单，搜索并打开"任务计划程序"。
   * 在任务计划程序窗口中，点击"创建基本任务"来创建新的定时任务。
2. 配置基本任务信息：
   * 在弹出的向导中，输入任务名称和描述，然后点击"下一步"。
   * 选择"每天"来指定任务的触发频率，然后点击"下一步"。
   * 选择任务应该运行的日期和时间（凌晨12点），然后点击"下一步"。
3. 配置操作：
   * 选择"启动程序"，然后点击"下一步"。
   * 在"程序/脚本"字段中，指定执行爬虫任务的脚本路径，例如：“scrapy crawl name-of-spider”。注意，这里的名称应该是实际的爬虫中定义的名称。
   * 在"起始于"字段中，指定爬虫脚本所在的目录路径。
   * 在"添加参数"字段中，指定任何额外的参数，例如爬虫脚本需要的音乐分类信息等。
   * 点击"下一步"。
4. 配置其他设置（可选）：
   * 根据需要配置任务的其他设置，例如设置任务的优先级、电源管理选项等。
   * 点击"下一步"，然后点击"完成"。

现在，我们已经成功创建了一个定时任务，用于每天凌晨12点自动执行你的爬虫任务（获取音乐分类信息，再获取音乐信息）。

注意，进行这个步骤之前需要相应的爬虫脚本已经编写完成，并且已经安装了Python环境以及Scrapy等相关的依赖库。另外，确保在任务计划程序中正确配置了Python的路径，以便能够正确执行爬虫脚本。

批处理流程串联

进行调度时，可以考虑提前编写批处理脚本来实现流程的串联，然后在任务计划程序中调用该批处理脚本。

批处理脚本是一种在Windows中运行的脚本文件，用于批量执行命令和操作。通过编写批处理脚本，您可以按照需要将多个命令和操作组合起来，实现流程的串联和自动化执行。

以下是在批处理脚本中实现增量更新流程串联的示例：

|  |
| --- |
| **@echo off**  **echo 获取音乐分类信息**  **scrapy crawl music\_update**  **echo 获取音乐信息**  **scrapy crawl music\_profile** |

上述批处理脚本中，使用了@echo off命令关闭输出提示信息，然后使用scrapy命令调用相应的爬虫脚本来获取音乐分类信息和音乐信息。开发者可以根据实际需要编写对应的脚本和命令。

接下来，将该批处理脚本保存为.bat文件，例如music\_crawler.bat。

然后，在任务计划程序中，创建一个新任务，将该批处理脚本的路径作为操作，并按照需求配置触发器和其他设置。任务计划程序会在指定的时间自动执行该批处理脚本，从而实现流程的串联。

使用批处理脚本可以方便地将多个操作组合在一起，但需要确保在脚本中正确配置所有所需的环境变量、命令路径等。此外，确保在任务计划程序中正确配置了Python的路径，以便能够正确执行相应的Python脚本。

**通过Scrapy-mail实现异常告警**

要设置Scrapy爬虫在执行失败后发送邮件通知给开发人员，可以借助Scrapy的中间件和邮件发送库。以下是具体实现方式：

1. 首先，安装相关依赖库。使用命令行工具pip来安装所需的库。在命令行中执行以下命令：

|  |
| --- |
| **pip install scrapy**  **pip install scrapy-mail** |

1. 在Scrapy项目中，创建一个名为middlewares.py的文件，并在其中定义一个新的中间件类作为邮件发送的处理器。这个中间件类的作用是在爬虫出错时触发邮件发送。例如：

|  |
| --- |
| from scrapy.mail import MailSender  class EmailFailureMiddleware:      def \_\_init\_\_(self, mailer):          self.mailer = mailer      @classmethod      def from\_crawler(cls, crawler):          mailer = MailSender.from\_settings(crawler.settings)          return cls(mailer)      def process\_spider\_exception(self, response, exception, spider):          subject = 'Scrapy爬虫执行失败'          body = f'爬虫 {spider.name} 在 {response.url} 时发生异常：{exception}'          self.mailer.send(to=['your@email.com'], subject=subject, body=body) |

1. 在项目的settings.py文件中启用该中间件，并配置邮箱相关的设置。找到DOWNLOADER\_MIDDLEWARES并添加中间件的路径。然后，在文件的底部加入如下的邮件设置，修改为发送邮件的SMTP服务器和账户信息：

|  |
| --- |
| DOWNLOADER\_MIDDLEWARES = {     "music163.middlewares.Music163DownloaderMiddleware": 543,     "music163.middlewares.EmailFailureMiddleware": 900  }  # 邮件设置  MAIL\_HOST = 'your-smtp-server'  MAIL\_PORT = 25  # 邮件服务器端口  MAIL\_FROM = 'your-email@example.com'  # 发件人邮箱  MAIL\_USER = 'your-email@example.com'  # 邮箱账户  MAIL\_PASS = 'your-email-password'  # 邮箱密码  MAIL\_TLS = True  # 是否使用TLS |

1. 当爬虫发生异常时，此中间件会调用邮件发送功能，发送包含出错信息的邮件给预先设定的开发人员邮箱。

开发过程中根据实际需求和环境进行了相应的修改和配置，例如更详细的错误信息、多个收件人等。此外，需要确保运行Scrapy爬虫的主机具备发送邮件的权限和网络连接。

以上是一种基本的设置来实现在Scrapy爬虫执行失败后发邮件通知开发人员的方式。开发者会根据需求进行修改和扩展，以满足具体的应用场景。

## 算法设计

### 背景及意义

背景：需要实现个性推荐的功能，需要使用算法分析各个用户的喜好，并根据其喜好给该用户推荐类似的音乐

意义：实现对用户的精准推荐

### 总体设计

#### 相关数据库表设计

1. 音乐推荐表

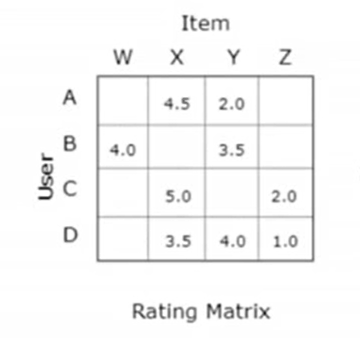
|  |  |  |
| --- | --- | --- |
| 字段名 | 数据类型 | 注释 |
| user\_id | Int64 | 用户id（primary key） |
| recommended\_music | Array(String) | 推荐的音乐id |

#### 算法的选择

1、协同过滤算法：

基于用户的协同过滤（User-Based Collaborative Filtering）：该方法通过分析用户之间的相似性来进行推荐。当一个用户喜欢某个项目时，系统会找到与该用户兴趣相似的其他用户，并将这些用户喜欢的项目推荐给当前用户。

基于物品的协同过滤（Item-Based Collaborative Filtering）：该方法通过分析项目之间的相似性来进行推荐。当一个用户喜欢某个项目时，系统会找到与该项目相似的其他项目，并将这些相似项目推荐给当前用户。SVD就是其中一种。



2、ALS算法：

ALS算法是一种用于解决协同过滤问题的优化算法。它通过交替优化用户向量和项目向量来求解最佳的推荐模型。

ALS算法的核心思想是通过最小化损失函数来学习用户和项目的隐含特征向量，使得预测评分尽可能接近实际评分。在每次迭代中，先固定用户向量，通过最小二乘法求解最佳的项目向量；然后再固定项目向量，通过最小二乘法求解最佳的用户向量。通过多次迭代，逐步优化模型参数，直到收敛为止。

ALS算法的优点是可以处理稀疏数据，且能够并行计算，适用于大规模推荐系统。

#### 推荐流程

通过协同过滤和ALS相结合的方式，为该用户推荐符合其喜好的音乐，将推荐结果去重后写入对应表中，前端读取后展示音乐详情在界面

**热门音乐推荐**

从clickhouse的music\_partition表得到最新一天用户对音乐的数据（播放量，收藏量，点赞量，评论次数，分享次数），将这些数据相加得到用户对某音乐的总体得分，选择得分最高的20首音乐作为热门音乐推荐，写入music\_popular表中。

**新歌推荐**

从clickhouse的music\_profile表获得音乐详情，根据时间戳排序获得最新的20首歌曲进行推荐，写入music\_new表中。

**热门歌手推荐**

从clickhouse的music\_partition表得到最新一天用户对音乐的数据（播放量，收藏量，点赞量，评论次数，分享次数），将这些数据相加得到某音乐的总体得分，再与music\_profile根据music\_id进行连接获得各个音乐的歌手，再根据歌手进行分组后将音乐得分求和，获得某歌手的总得分（歌手各个音乐的得分的和），选择歌手的总得分前几名，最后与artist表相连接得到热门歌手的详细信息，写入singer\_popular表中。

**用户活跃量**

读取mysql的new\_user\_action的数据，用filter得到昨天的全部用户操作数据，再根据uid进行去重，可以获得昨天的用户活跃量，将其数据加上日期写入clickhouse的user\_active表中。

**Vip复购率**

读取mysql的new\_user\_action的数据，用filter得到上个月的全部用户操作数据且操作字段（oc）为‘b’的数据（表示购买操作），根据uid进行分组，获得每个人该月的购买次数，将购买次数大于一的人数除以总的购买人数，获得该月的复购率，写入vip\_repurchase表。

**个性化音乐推荐**

从clickhouse的uid\_music表获得各个用户对某音乐的播放次数，以及个人的总听歌次数，用某音乐的播放次数除以个人总听歌次数获得某用户对某音乐的评分。之后通过协同过滤算法推算得到各个用户对所有音乐的评分，推荐评分最高的5首音乐，过程中通过svd降维矩阵进行降维来加快运算。

### 详细设计

（1）协同过滤

1、采用基于物品的推荐

2、以用户播放某首音乐的次数，除以该用户的总音乐播放次数为评分

3、基于上述评分建立音乐-评分矩阵

4、通过机器学习内置算法推导其他音乐评分

5、选定评分高的音乐作为推荐音乐

6、将推荐的音乐写入数据库

（2）ALS算法

1、采用基于用户的推荐

2、以用户对某首音乐的正向操作次数为评分

3、基于上述评分建立音乐-评分矩阵

4、通过spark内置算法推导其他音乐评分

5、选定评分高的音乐作为推荐音乐

6、将推荐的音乐写入数据库

### 编码实现

1、环境准备

IDEA,Spark,Python3,Zookeeper,Clickhouse

2、创建Maven项目

在IDEA中手动创建Maven项目，并修改maven库为本地仓库

3、编写代码

Gitee托管

详细代码请访问：

https://gitee.com/suwenjiewensu/music-recommendation-system

主要涉及到的项目目录为：

Spark-Streaming-realtime/src/main/scala/com.gzu.realtime/app（算法实现）

和Spark-Streaming-realtime/src/main/scala/com.gzu.realtime/util（工具类）

和Spark-Streaming-realtime/src/main/scala/com.gzu.realtime/bean

1. 代码运行

编写shell命令

调度shell命令调度运行

5、数据存储

数据将存入Clickhouse的对应表中

## Mook设计

### Mook

"Mook"是一种在开发早期阶段用于模拟数据的技术。它允许我们生成随机的、虚拟的数据，以模拟真实数据的流通性和行为。使用Mook数据进行测试的原因有以下几点：

1. 模拟真实环境：在音乐推荐系统开发初期，往往缺少实际用户生成的海量行为数据。使用Mook数据可以为我们提供模拟的用户行为和数据流，以模拟真实环境中的数据交互和流通情况。
2. 快速测试原型：在系统开发的早期阶段，通常会有各种新想法和概念需要验证。使用Mook数据可以快速创建原型并进行测试，而无需等待真实用户的数据产生。这使得我们可以更快地迭代和调整系统，在保证一定准确性的前提下进行快速试验和原型构建。
3. 覆盖边缘情况：使用Mook数据可以使我们模拟各种边缘情况，例如极端用户行为、异常数据或是不常见的使用场景等。这可以帮助我们在开发初期就发现潜在的问题并提前做出相应的优化或调整。
4. 数据一致性和控制：使用Mook数据，我们可以精确控制生成的数据和数据流，以保持数据的一致性。我们可以定制化生成特定类型的数据，生成符合特定测试需求的数据集。这样可以更好地进行系统测试，并且可以固定和重复特定的测试场景。

当然，在使用Mook数据进行测试时，我们也要清楚地意识到它只是模拟真实数据的一种近似方法。因此，在系统开发成熟后，我们仍然需要通过真实用户的行为数据来验证和调整系统。-·

### Mook数据的方法

在开发和测试过程中，可以使用不同的方法来生成Mook数据。以下是几种常见的方式：

1. 随机生成数据：通过编写代码或使用工具，可以生成随机的、虚拟的数据。例如，对于音乐推荐系统，可以通过随机生成用户、歌曲、评论等信息来模拟真实环境下的数据。
2. 使用伪造数据生成器：有一些开源工具和库可以帮助生成伪造数据，这些数据可以具有一定的结构和规则。例如，Faker是一个流行的Python库，可以生成各种随机的、虚拟的数据，包括姓名、地址、日期等。
3. 实际数据采样和变换：当有一些真实数据可用时，可以从这些数据中采样并进行变换，生成基于真实数据的Mook数据。例如，可以对真实用户行为数据进行采样，并进行一些变换（如随机修改、用户行为重复）来生成模拟数据集。
4. 真实数据的子集：如果有一部分真实数据可用，可以选择使用其中的一部分作为Mook数据。根据需要，可以选择特定条件下的数据子集，以覆盖不同的测试场景。

使用哪种方式生成Mook数据取决于具体的需求和资源可用性。在选择方法时，需要考虑数据的准确性、数据量的充分性以及对测试场景的覆盖程度。同时，需要确保生成的Mook数据可以有效地模拟真实环境中的数据流通性和用户行为。

这里考虑使用Python脚本生成Mook数据。

### Python Mook

使用Python脚本生成Mook数据有以下几个原因：

1. 简单易学：Python是一门易于学习和使用的编程语言，具有简洁的语法和直观的语言结构。这使得编写Python脚本生成Mook数据相对容易，无论是对于新手还是有经验的开发人员来说，都能快速上手并编写出功能强大的脚本。
2. 丰富的数据处理库和工具：Python拥有许多强大的数据处理库和工具，例如NumPy、Pandas和faker等。这些库提供了丰富的功能和方法，使得生成各种类型的Mook数据变得更加容易。例如，使用faker库可以轻松生成具有各种特定格式的虚拟数据，如姓名、地址、电子邮件等。
3. 可拓展性和灵活性：Python是一门高度可拓展的语言，可以轻松地集成其他第三方库或工具。这意味着您可以使用Python来生成复杂和多样化的Mook数据，以满足不同的测试需求。无论是生成大量的随机数据、模拟特定的用户行为还是定制化生成特定格式的数据，Python都可以提供相应的解决方案。
4. 生态系统的支持：Python拥有活跃的生态系统，有许多开源工具和社区资源可供使用。这意味着您可以从社区中获取海量的示例代码、文档和指导，以帮助您更快地编写生成Mook数据的Python脚本。无论您遇到什么问题，几乎可以肯定会有其他开发人员在Python社区中分享相关经验和解决方案。

综上所述，Python的轻量简单易学的特性、丰富的数据处理库和工具、可拓展性和灵活性，以及强大的社区支持，使其成为生成Mook数据的理想选择。使用Python脚本能够帮助开发人员以高效和灵活的方式生成各种类型的Mook数据，以用于测试和开发过程中的数据模拟和验证。

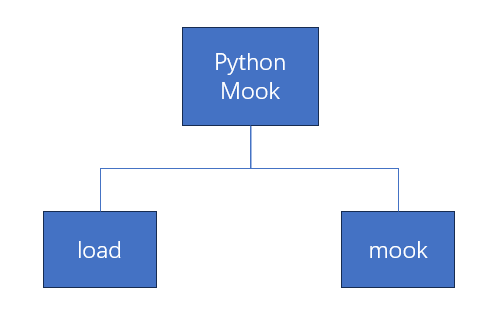
### 总体设计

Python Mook的设计包含两个模块：load和mook，它们分别负责加载预置数据和处理Mook数据生成的逻辑。以下是对这两个模块的详细描述：

1. load模块：
   * load 模块负责加载预置数据，包括音乐列表、操作类型列表等。这些预置数据是在Mook数据生成过程中所需的基础数据，用于构建生成Mook数据所需的上下文和参考。
   * 在这个模块中，可以通过读取文件、调用API或从数据库中获取预置数据。根据具体需求，可以采用不同的方法和数据源来加载预置数据。
   * 一旦数据加载完成，它们可以被传递到 mook 模块中作为参数，以供后续的数据生成和处理使用。
2. mook 模块：
   * mook 模块是 Python Mook 的核心部分，负责处理Mook数据生成的逻辑。
   * 在这个模块中，可以编写生成Mook数据的算法、规则和逻辑。根据项目的需求，可以定义生成数据的条件、数据结构和生成策略。
   * 使用预置数据作为输入，mook 模块可以根据指定的规则和算法生成符合条件的Mook数据。这些数据可以包括模拟用户行为、生成假数据、构建特定场景等。
   * 在生成数据时，可以使用 Python的基本数据结构、列表、字典等，并借助第三方库来处理数据和生成特定格式的数据。

通过将预置数据加载和Mook数据生成的逻辑分离成两个模块，Python Mook的架构更加模块化和可扩展。这种设计模式使得每个模块都有明确的功能和责任，并可以单独测试和调试，提高了代码的可读性和可维护性。

总而言之，load 模块负责加载预置数据，mook 模块负责处理Mook数据生成的逻辑。两个模块的协作能够实现灵活且高效的Mook数据生成功能。



Python Mook 架构图

## SpringAPI

在音乐推荐系统中，推荐结果通常根据不同的算法和数据源而存储在各个数据库的不同表中。这样的存储方式给前端获取推荐结果带来了一定的困扰，需要分别访问和处理不同的数据库和表。为了简化前端对推荐结果的访问和展示，需要提供一个统一的接口。

### 框架选择

Springboot是面向该场景的一个可行的解决方案。使用Springboot进行API设计有以下理由和意义：

* **多数据源支持：**Springboot框架提供了多数据源配置的能力，可以同时连接多个数据库。这样可以轻松地访问和处理不同数据库的推荐结果，避免了繁琐的数据库连接和操作过程。
* **简化前端访问：**通过提供统一的接口，前端只需调用一个API即可获取推荐结果，无需关心具体的数据库和表结构。这样简化了前端的开发流程，提高了开发效率。
* **用户交互处理：**Springboot可以通过处理用户交互事件，如点击、收藏等，来实时更新推荐结果和用户行为数据。这样可以提供个性化的推荐体验，并持续优化推荐结果。
* **运行监控：**Springboot提供了监控和管理应用程序的功能，可以实时监控API的运行情况和性能指标，帮助开发团队及时发现和解决问题，确保系统的稳定性和可靠性。

### 需求分析

#### 功能需求

* 实现一个统一接口，用于前端获取音乐推荐结果并进行页面展示。
* 提供以下五个功能接口：
  + 热门音乐接口：用于获取热门音乐的推荐结果。
  + 热门艺术家接口：用于获取热门艺术家的推荐结果。
  + 个性推荐接口：根据用户id获取个性化的音乐推荐结果。
  + 搜索接口：根据搜索类型和关键词在曲库内搜索对应内容。
  + 用户操作接口：用于处理用户的操作事件。

#### 数据需求

* 音乐推荐结果保存在不同数据库的不同表中，需要通过配置多数据源，实现数据源的切换。
* 需要根据不同的推荐类型（热门音乐、热门艺术家、个性推荐等）从相应的数据库表中查询结果。

#### 用户界面

* 前端需要与后端进行交互，并通过提供的统一接口获取推荐结果进行页面展示。
* 前端可以根据用户的交互事件，调用用户操作接口上传用户操作至服务器。

#### 非功能需求

* 性能要求：接口需要在合理的时间内返回推荐结果，保证用户的流畅体验。
* 可扩展性：系统需要支持在后续进行新的音乐推荐接口的扩展。
* 可靠性和稳定性：系统需要保证数据的准确性和可靠性，避免因系统故障导致数据丢失或异常。

#### 运行环境

* 使用Spring Boot作为后端开发框架，结合多数据源的配置，以支持不同数据库的访问。
* 部署在适当的服务器环境中，同时配置合适的监控机制，以确保系统正常运行。

#### 总结

根据需求，需要设计和实现一个基于Spring Boot的API，提供统一接口以获取音乐推荐结果并进行页面展示。通过配置多数据源，实现不同数据库的访问，提供热门音乐、热门艺术家、个性推荐、搜索、用户操作等五个接口。通过这些接口，实现与前端的交互，并满足性能、可扩展性、可靠性和稳定性等非功能需求。

### 总体设计

连接池：HikariPool

数据源：MySQL+ClickHouse

提供接口：

* 热门音乐
* 热门艺术家
* 个性推荐
* 搜索
* 用户操作

### 详细设计

#### 热门音乐

**说明：**调用此接口可获取热门音乐。

**接口地址：**/recommend/pop

**请求方式：**GET

**请求参数：**无

#### 热门艺术家

**说明：**调用此接口可获取热门歌手

**请求地址：**/recommend/pop/artist/

**请求方式：**GET

**请求参数：**无

#### 个性推荐

**说明：**调用此接口，传入用户id，可获取个性推荐

**请求地址：**/recommend/self/

**请求方式：**GET

**请求参数：**

uid: 用户id

#### 搜索

**说明：**调用此接口，传入搜索类型和关键词，可在曲库内搜索对应内容

**请求地址：**/search/

**请求地址：**GET

**请求参数：**

type: 搜索类型，支持音乐搜索（song)，模糊搜索（songs)，艺术家搜索（artist)，专辑搜索（album)

kw: 搜索关键词

#### 用户操作

**说明：**调用此接口，传入用户id、音乐id和操作类型，上传用户操作至服务器

**请求地址：**/uop/music/

**请求方式：**POST

**请求参数：**

uid: 用户id

music\_id: 音乐id

oc: 执行操作

（l->点赞, p->播放, c->评论, s->分享, f->收藏, ul->取消点赞, uf->取消收藏, uc->取消评论, b->购买单曲）

### 编码实现

以下是实现SpringAPI设计的一般步骤：

* 配置多数据源：根据各个数据库的配置，使用Springboot的多数据源功能进行配置，在配置文件中指定不同的数据库连接信息。
* 定义统一接口：使用Springboot的RESTful API设计原则，定义统一的接口用于查询和返回推荐结果。
* 编写业务逻辑：根据具体业务需求，编写处理推荐结果的业务逻辑，包括查询不同数据库的数据、数据处理和数据封装。
* 实现用户交互处理：根据需求，编写处理用户交互事件的代码，如点击、收藏等，更新相关推荐结果和用户行为数据。
* 运行监控配置：根据需要，配置Springboot的监控和管理功能，如使用Actuator监控API的运行情况和使用性能。

通过上述步骤，可以实现SpringAPI设计，在音乐推荐系统中提供一个统一的接口，简化前端访问和展示推荐结果的流程，同时支持用户交互处理和运行监控，提高系统的性能和可靠性。

#### MyBatis

MyBatis是一个开源的持久层框架，它封装了JDBC操作和SQL语句的编写，可以帮助开发者更便捷地处理数据库读写操作。MyBatis基于Java的反射机制，通过Mapper文件定义和配置SQL语句，然后将SQL语句与Java方法绑定，实现数据库的读写操作。

以下是使用MyBatis处理数据库读写的编码实现细节：

1. 配置MyBatis：
   * 在Spring Boot的配置文件中，配置MyBatis的相关属性，包括数据库连接、用户名、密码等信息。
   * 在pom.xml文件中添加MyBatis的依赖。
   * 创建MyBatis的配置类，使用@Configuration注解，配置数据源和MyBatis的一些属性。
2. 编写Mapper接口：
   * 创建Mapper接口，使用@Mapper注解，定义数据库操作的方法。
   * 在Mapper接口中定义与数据库表对应的方法，包括增删改查等操作。
3. 编写Mapper XML文件：
   * 在resources目录下创建Mapper XML文件，与Mapper接口相对应，用于配置SQL语句以及查询参数等信息。
   * 在XML文件中定义与Mapper接口方法对应的SQL语句和参数配置。
4. 注入Mapper：
   * 在Spring Boot的配置类中，使用@MapperScan注解，扫描Mapper接口所在的包路径，并将Mapper接口注入到Spring容器中。
5. 编写Service层：
   * 创建Service层，调用Mapper接口中的方法，对数据库进行读写操作。
   * 在Service层中处理业务逻辑，并调用Mapper接口中定义的方法。
6. 编写Controller层：
   * 创建Controller层，接收前端的请求，调用Service层进行处理，并返回结果。
   * 在Controller层中定义路由和请求方法，处理前端的各种操作和交互。

通过以上步骤，就可以使用MyBatis处理数据库的读写操作。在编码实现过程中，需要配置MyBatis的相关属性和依赖，编写Mapper接口和Mapper XML文件，注入Mapper到Spring容器中，并在Service和Controller层中调用Mapper接口的方法，完成数据库操作和业务逻辑的处理。这样就可以实现基于Spring Boot和MyBatis的API设计，完成数据库的读写操作，并提供统一的接口供前端进行使用。

#### @RestController

@RestController是Spring框架中的一个注解，用于标识一个类是RESTful API的控制器。使用@RestController注解的类会自动将所有处理请求的方法的返回值转换为适当的响应体，并以JSON或XML格式返回给客户端。

以下是使用@RestController设计和实现Spring API的一般步骤和细节：

1. 导入相关依赖：在Maven或Gradle配置文件中添加Spring Web和其他必要的依赖。
2. 创建Controller类：创建一个新的Java类，并使用@RestController注解标记它。你可以定义多个处理请求的方法，每个方法对应不同的请求路径和HTTP方法。例如：

|  |
| --- |
| @RestController  @RequestMapping("/api")  public class MyController {      @GetMapping("/users")      public List<User> getUsers() {          // 处理获取用户列表的逻辑          // 返回List<User>      }      @PostMapping("/users")      public ResponseEntity<User> createUser(@RequestBody User user) {          // 处理创建用户的逻辑          // 返回ResponseEntity<User>      }      // 其他处理请求的方法  } |

1. 定义请求路径和HTTP方法：使用@RequestMapping注解定义方法处理的请求路径和HTTP方法。可以根据需要设置URL路径、请求参数、请求体等条件。例如，在上述代码中，@GetMapping("/users")指示处理GET请求的"/api/users"路径。
2. 处理请求和生成响应：编写方法体来处理请求，并生成合适的响应。可以通过方法参数来获取请求中的参数、路径变量、请求体等。可以使用任何业务逻辑和数据访问的代码来处理请求，并返回适当的数据结构作为响应。
3. 处理异常情况：可以使用异常处理注解（如@ExceptionHandler或@ControllerAdvice）来处理异常情况，并返回定义的错误信息，以确保良好的错误处理和可靠的API响应。
4. 添加其他注解：根据需要，可以添加其他注解来提供更多的功能和配置，如@PathVariable、@RequestBody、@ResponseStatus等注解。

需要注意的是，还需要对项目进行正确的配置，包括Spring的配置文件、连接数据库、配置对象映射等，以及在启动类中添加相关注解（如@EnableWebMvc）以启用Spring Web功能。

通过以上步骤，即可使用@RestController注解来设计和实现一个基于Spring的RESTful API。开发者定义了不同的请求路径和HTTP方法的处理方法，并根据业务逻辑和需求来处理请求和生成响应。此外，Spring还提供了许多其他功能和注解，可用于进一步增强和定制SpringAPI。

#### 启用Prometheus支持

Spring Boot提供了对Prometheus的支持，可以方便地启动Prometheus进行监控。

Spring Boot通过Micrometer库集成了多个监控系统，包括Prometheus。Micrometer是一个应用程序度量工具，可以将应用程序的各种指标（如请求次数、错误率、响应时间等）导出到不同的监控系统，包括Prometheus。

在Spring Boot中启用Prometheus进行监控的编码实现细节如下：

1. 添加Prometheus依赖：
   * 在pom.xml文件中添加Prometheus的依赖。例如：

|  |
| --- |
|  |

1. 配置Prometheus：

* 在application.properties或application.yml文件中添加Prometheus的相关配置。例如：

|  |
| --- |
|  |

1. 编写自定义指标：

* 在应用程序中，根据需要，使用Micrometer库提供的API，定义自定义的指标。例如：

|  |
| --- |
|  |

1. 启动应用程序：

* 运行Spring Boot应用程序，Prometheus会自动在/metrics端点暴露应用程序的监控指标。可以通过访问http://localhost:8080/来验证指标是否正常显示。

通过以上步骤，我们可以在Spring Boot应用程序中启用Prometheus进行监控。配置Prometheus依赖，添加相关配置，编写自定义指标，并启动应用程序。Prometheus会自动暴露应用程序的监控指标，开发人员可以通过配置Prometheus Server和Grafana进行数据的收集、存储、查询和可视化。这样就可以实现对应用程序的监控和运维。

### 运行和维护

此部分介绍将SpringAPI部署到Docker容器中运行，并使用Prometheus监控运行指标的运行和维护方案。

#### Docker简介和工作原理

Docker是一种容器化技术，可以通过容器的方式部署和运行应用程序。Docker将应用程序及其依赖项打包在一个独立的容器中，使其可以在不同的环境中进行部署和运行，而不会受到底层操作系统的差异影响。Docker利用容器的隔离性和轻量级特性，实现了快速部署、弹性扩展和环境一致性等优势。

#### Docker运维方案

* 安装和配置Docker：根据操作系统的不同，安装和配置Docker运行环境，并确保Docker服务正常启动。
* 构建Docker镜像：根据SpringAPI的代码，创建Dockerfile文件，定义Docker镜像的构建过程。在Dockerfile中指定基础镜像（openjdk:17）、安装依赖项、复制应用程序代码等步骤，并构建Docker镜像。
* 运行Docker容器：使用Docker命令运行Docker容器，基于构建的Docker镜像启动SpringAPI应用程序。配置容器的网络、端口映射等参数，使应用程序可以在容器内外进行访问。
* 实现持久化存储：根据需要，将应用程序需要依赖的文件和数据进行持久化存储，可以使用Docker的数据卷或者网络存储等技术。
* 监控和日志收集：通过配置Docker容器的日志驱动，将应用程序的日志输出到标准输出或者日志文件，并使用日志收集工具（如ELK、EFK等）进行管理和分析。使用Prometheus作为监控系统，收集Docker容器的运行指标（如CPU、内存、网络等）以及应用程序的自定义指标，并进行可视化和报警。

#### Prometheus简介和工作原理

Prometheus是一个开源的监控系统，可以收集和存储各种服务和系统的指标数据，并通过查询和可视化方式进行展示。Prometheus使用Pull方式，通过定期抓取被监控对象的指标数据，并存储在本地的时间序列数据库中。它支持多维度和灵活的查询语言，可以用于监控和警报。Prometheus还可以与Grafana等工具集成，提供强大的监控和可视化能力。

#### Prometheus运维方案

* 安装和配置Prometheus：根据Prometheus的官方文档，安装和配置Prometheus服务，并确保服务正常运行。
* 配置Prometheus监控目标：在Prometheus的配置文件中，添加需要监控的Docker容器的IP地址和端口信息。
* 定义告警规则：根据应用程序的需求，定义相应的告警规则，并设置告警策略和通知方式，以及处理告警的流程。
* 可视化和报表：借助Grafana等工具，配置Prometheus的数据源，并创建监控仪表盘，以实现指标数据的可视化和报表展示。
* 定期备份和升级：定期备份Prometheus和应用程序的数据，并进行系统和应用程序的升级和更新，以确保系统的稳定性和安全性。

通过使用Docker部署SpringAPI和Prometheus，可以实现应用程序的快速部署、环境一致性和灵活的扩展性。同时，通过配置Prometheus进行监控，可以及时了解运行指标和性能状况，做出相应的优化和调整。通过以上运维方案，可以确保应用程序的正常运行，并实现对应用程序和Docker容器的监控和管理。

## Web前后端

### 接口设计

欢迎阅读SpringBoot接口文档

作者：苏文杰

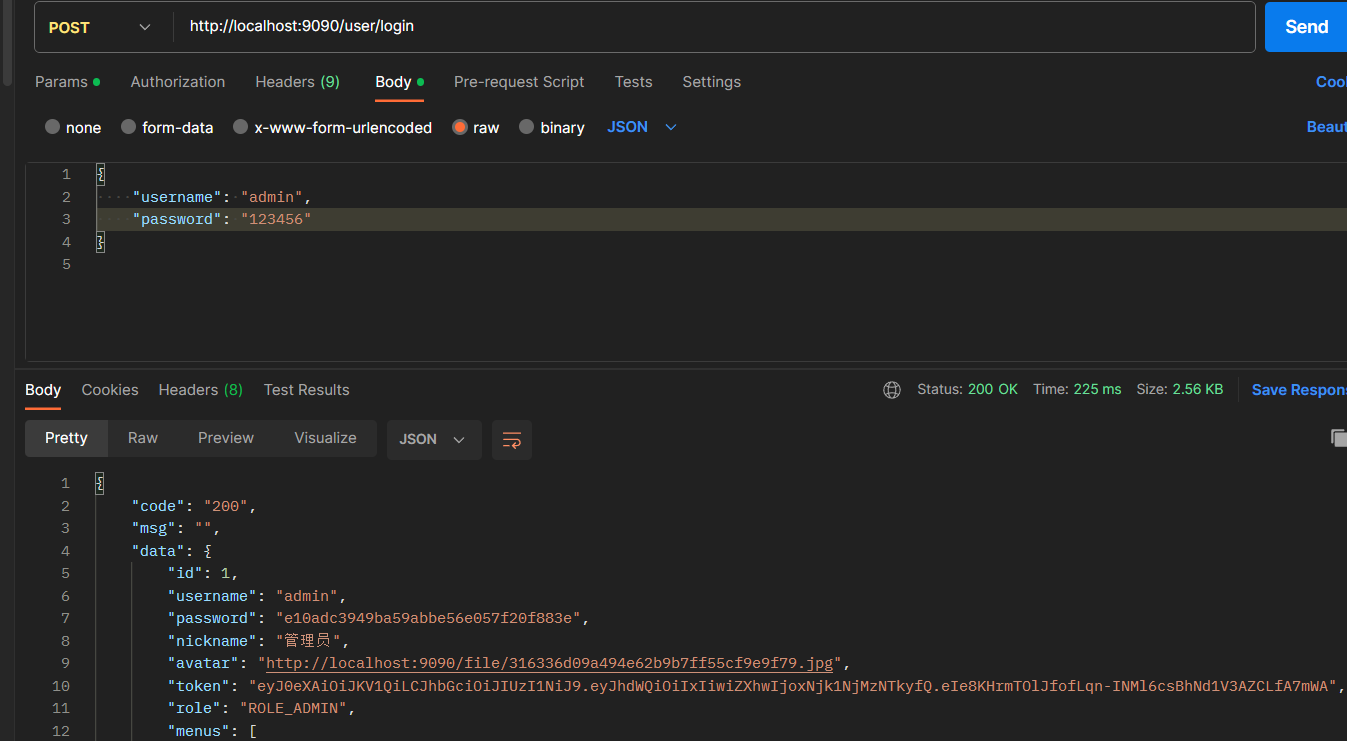
时间：2023-9-25

1.UserController

1.1 login

请求方式：Post

参数 UserDto

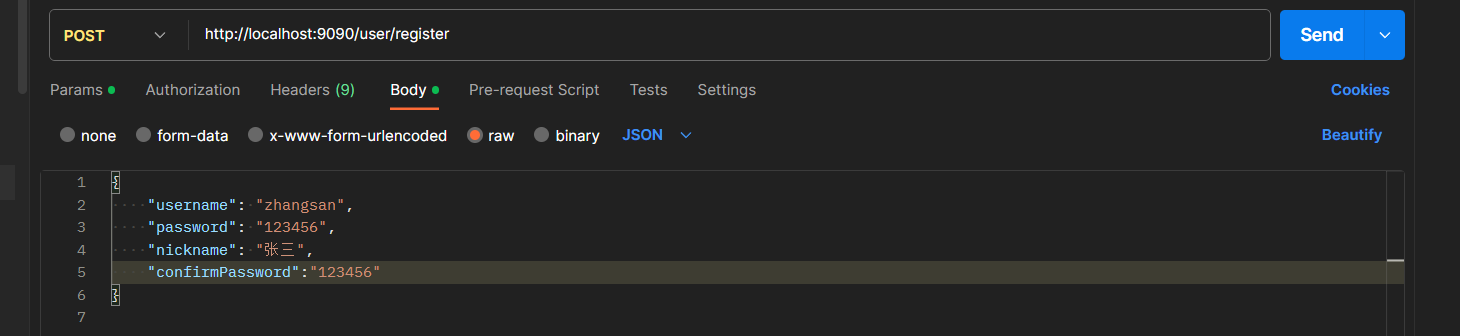
地址：/user/login

返回了token、除了拦截器放行的路径、其余路径发起请求都需要携带token

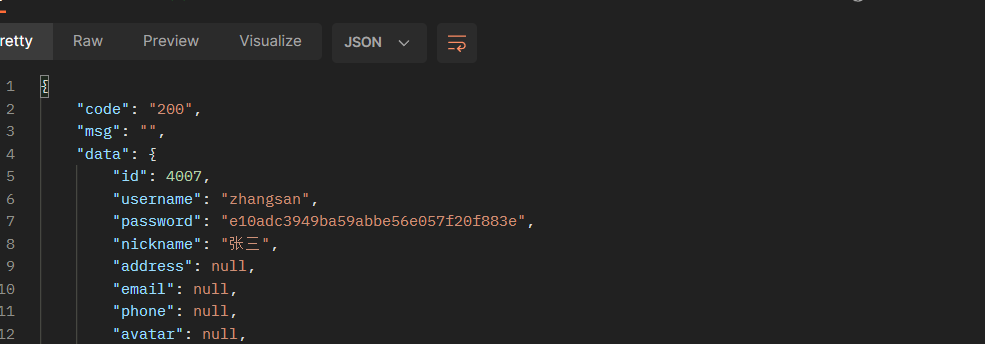
1.2 注册接口

请求方式：Post

URL: /user/register

参数：UserDto

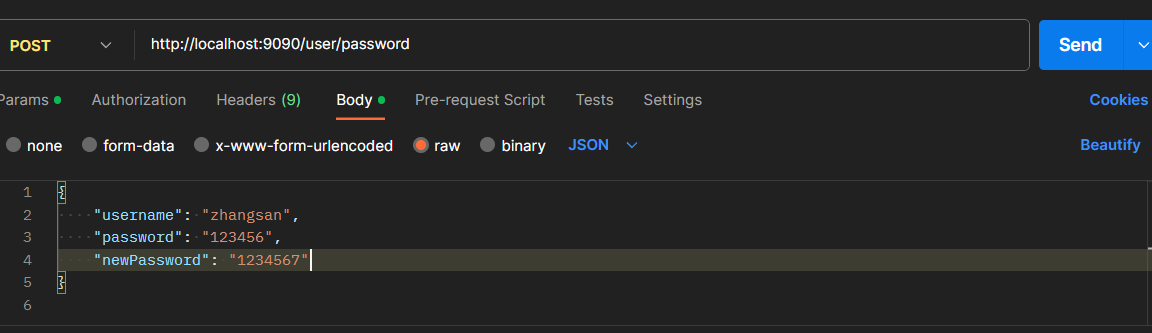
返回信息

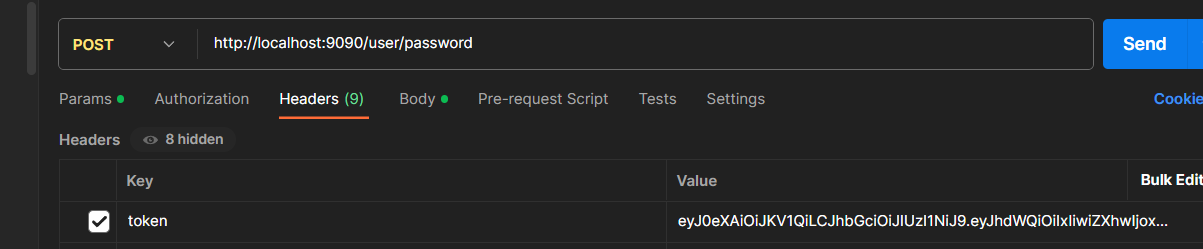
密码通过MD5 加密存储与数据库中

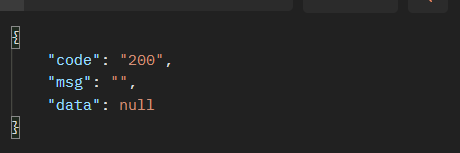
1.3 修改密码接口

请求方式：Post

参数：PasswordDto

路径：/user/password

携带token  
返回：

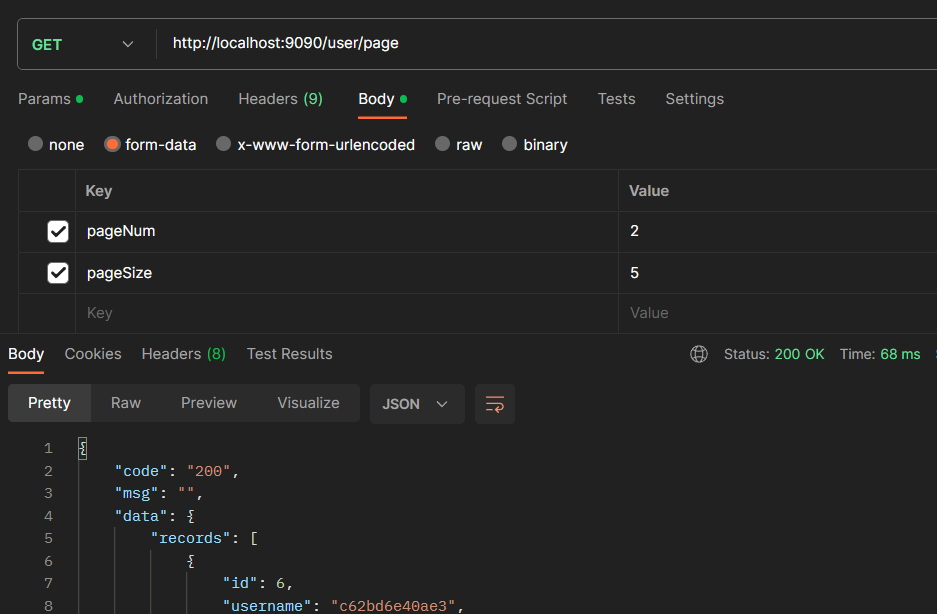


1.4 分页查询

请求方式：GET

url: /user/page

返回：



1.5 修改和新增接口

请求方式：Post

参数: User

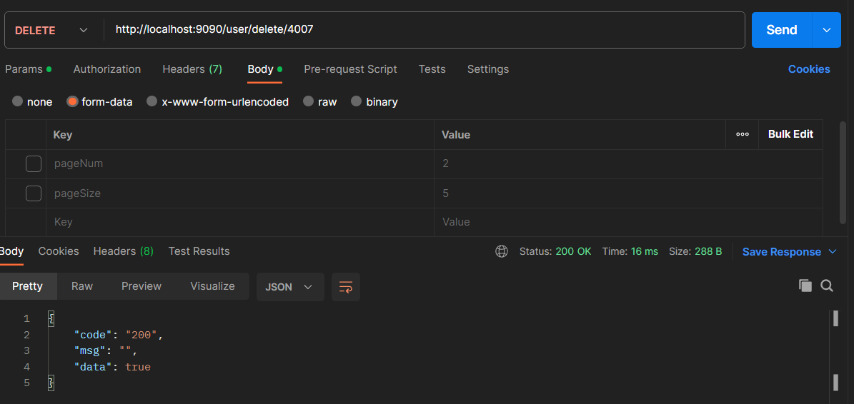
URL：/user/save

1.6 删除接口

请求方式： Delete

参数：ID

URL: /delete/{id}



1.7 批量删除接口

请求方式：POST

参数: List<Integer> IDS

URL：/delete/batch

1.8 根据username查询用户

请求方式：GET

参数：String username

URL: /user/username/{username}

以/user/username/admin为例

返回：

{

    "code": "200",

    "msg": "",

    "data": {

        "id": 1,

        "username": "admin",

        "password": "e10adc3949ba59abbe56e057f20f883e",

        "nickname": "管理员",

        "address": "贵州",

        "email": "123qq1.com",

        "phone": "18786903900",

        "avatar": "http://localhost:9090/file/316336d09a494e62b9b7ff55cf9e9f79.jpg",

        "createTime": "2023-09-21T10:16:36.000+00:00",

        "role": "ROLE\_ADMIN",

        "sex": "男",

        "age": 18

    }

}

1.9 导出接口

请求方式：GET

参数 HttpServletResponse response

返回信息

1.10 获取用户各个阶段的年龄

请求方式：GET

参数：无

URL: /user/getAgeGroup

1.11 获取各个地区的用户数

请求方式：GET

参数：无

URL: /user/getAddressCount

1.12 获取性别比例  
请求方式：GET

参数：无

URL: /user/getSexCount

2.用户操作接口

2.1 用户操作保存

请求方式：POST

请求参数：UserActionDto

URL：/uop/music-play

2.2 上月用户信息统计

请求方式：GET

请求参数：无

URL: /uop/LastMonthDate

2.3 近五天用户操作  
请求方式：GET

请求参数：无

URL：/uop/user-record

3.音乐接口

3.1 音乐新增或修改

请求方式：POST

请求参数：SysMusic

URL: /sys-music/save

3.2 音乐删除

请求方式：DELETE

请求参数：id（音乐ID）

URL: /sys-music/delete/{id}

3.3 音乐删除

请求方式：DELETE

请求参数：id（音乐ID）

URL: /sys-music/delete/{id}

3.4 音乐批量删除

请求方式：POST

请求参数：ids（音乐ID列表）

URL: /sys-music/delete/batch

3.5 分页查询

请求方式：GET

请求参数：PageNum,pageSize,musicName

[URL: /sys-music/page](URL:/sys-music/page)

3.6 通过musicId查询音乐

请求方式:GET

请求参数：musicId

URL：/getOneByMusic/{musicId}

3.7 热门音乐查询

请求方式：GET

请求参数：无

URL: /singerPopular/getSingerPopular

3.8 个性推荐查询

请求方式：GET

请求参数：无

URL: /musicRecommend/getMusic

3.9 获取详细10条音乐信息

请求方式：GET

请求参数：无

URL：/musicProfile/list

3.10 通过音乐id获取音乐详细信息

请求方式：GET

请求参数：无

URL：/musicProfile/getByMusicId/{musicId}

3.11 通过一组id获取音乐详细信息

请求方式：GET

请求参数：无

URL：/musicProfile/getByMusicIds

3.12 获取原创歌曲

请求方式：GET

请求参数：无

URL: /musicProfile/original

3.13 获取新歌信息

请求方式：GET

请求参数：无

URL：/musicNew/getNewMusic

3.14 获取新歌榜前10

请求方式：GET

请求参数：无

URL：/musicNew/getNewMusicTop10

4 文件接口

4.1 文件上传接口

请求方式：Post

参数：MultipartFile file

URL: /file/upload

返回：文件地址

4.2 文件下载接口

请求方式: GET

参数：无

URL: /file/{fileUUID}

4.3 保存接口

请求方式：POST

参数：Files

URL：/file/save

4.3 删除接口

请求方式：DELETE

参数：Integer id

URL: /file/delete/{id}

4.4 批量删除接口

请求方式：Post

参数：List<Interger> id

URL: /file/delete/batch

4.5 分页接口

请求方式：GET

参数：pageNum,pageSize,filename

URL: /file/page

5 专辑接口

5.1 save接口

请求方式：/album/save

参数：AlbumDto

URL：/album/save

5.2 删除接口

请求方式：DELETE

参数：id

URL：/album/delete/{id}

5.3 批量删除接口

请求方式：Post

参数：List<Integer> ids

URL: /album/delete/batch

5.4 分页接口

请求方式：GET

参数：pageNum，pageSize, albumName

URL: /album/page

6.艺术家接口

6.1 save接口

请求方式：/artist/save

参数：ArtistDto

URL：/ artist /save

6.2 删除接口

请求方式：DELETE

参数：id

URL：/ artist /delete/{id}

6.3 批量删除接口

请求方式：Post

参数：List<Integer> ids

URL: /artist/delete/batch

6.4 分页接口

请求方式：GET

参数：pageNum，pageSize, albumName

URL: /artist/page

### 开发过程

**音乐推荐系统开发文档**

**版本: 1.0**

**作者: [苏文杰]**

**发布日期: [2023-9-24]**

欢迎阅读音乐推荐系统的开发文档。本文档旨在提供有关系统架构、功能、技术栈以及其他重要信息的详细说明，以便您能够了解和使用本系统。

#### 简介

随着数字音乐产业的快速发展以及大数据时代的到来、音乐推荐系统已经成为了用户发现和探索音乐的最重要途径。如何能够高效的根据用户行为推荐出用户喜爱的音乐、根据用户行为画像成了现如今大数据时代的任务。因此，我们开发了这套音乐推荐系统，希望能够给用户带来良好的使用体验。本系统分为前台的音乐推荐以及后台管理。前台页面首页展示了热门推荐、这是我们通过播放量、点赞、收藏等指标进行计算，然后得到热门推荐歌曲，也提供了新歌展示。同时对登陆用户、我们还会基于其操作进行个性化推荐。为了能够稳定的支持音乐推荐系统的运营和数据处理，我们还开发了后台管理系统，后台管理系统用于管理音乐推荐系统数据,管理员登陆后可以通过后台对音乐、用户进行管理，同时也可以看到实时数据的统计量、通过echats展示用户活跃度。

#### 技术栈

后端: Springboot 2 + Mybatis + Mybatis Plus + JWT + hutool

前端: Vue 2 + Vue x + elementUI + echarts + axios

数据库: Mysql + ClickHouse + Redis

##### 选用Spring Boot+mybatis的好处

**3.1快速开发**：Spring Boot 简化了应用程序的搭建和配置过程、提供了自动装配和默认设置、使开发人员能够更快速的创建应用程序原型并迅速进入开发阶段。

**3.2简化的配置**：Spring Boot采用约定大于配置的原则，减少了繁琐的配置。它通过注解和熟悉文件等机制来管理配置，使配置更加清晰简单

**3.3内嵌式容器:**Spring Boot指出多种内嵌式容器（如Tomcat,jetty等），这使得将应3用程序打包成可执行的Jar文件非常容易，无需外部的Web服务器。

**3.4自动化配置：**Spring Boot提供了大量自动配置选项，根据应用程序的依赖关系自动装配Spring应用程序上下文，从而减少了手动配置的需求。

**3.5集成测试支持：**Spring Boot提供了用于编写集成测试的工具和库，使得测试变得更加容易。

**3.6.监控和管理：**Spring Boot集成了Actuator,可以轻松地监控和管理应用程序的健康状况、性能和运行时信息。

**3.7.生态管理：**Spring Boot具有强大的生态系统，有大量的第三方库和插件可用，可用于解决各种常见的问题。

**3.8.Mybatis Plus整合：**Mybatis Plus是Mybatis的增强工具包，简化了数据库操作和SQL编写、提供了便利的方法，可以更加轻松地执行了数据库操作，并且能够充分利用Spring Boot的特性，如事务管理和数据源配置。

**3.9.代码生成：**Mybatis Plus提高了代码生成器，可以自动生成实体类、Mapper接口、XML文件、Service，减少了手动编写重复代码的工作，大大提高了工作效率

**3.10.分页支持：**Mybatis Plus内置了强大的分页查询功能，使得分页查询变得更加简单。

**3.11.内置SQL注入防护：**Mybatis Plus通过自动处理SQL注入攻击，提高了应用程序的安全性。

**3.12.可拓展性：**Spring Boot 和Mybatis Plus都具有高度可拓展性，可以根据项目的需求定义和扩展功能。

##### 环境配置

JDK1.8 + Mysql 5.7+ + IntrlliJ IDEA + Node 16 + Maven 3.9.1

#### 可行性研究

进行可行性研究的目的是对音乐推荐系统的一系列问题进行分析，结合自身技术来制定开发计划、避免在开发过程中出现一些不必要的错误。

**5.1经济可行性：**

本系统由小组系统设计、投入资金较小、主要费用包括硬件设备：软件开发的人员费用。

**5.2技术可行性：**

在Windows操作系统下、我们通过IDEA进行开发、选取Java语言、Spring Boot后端框架以及Vue 前端框架、数据库采用Mysql和ClickHouse以及Redis。这些都是比较成熟的语言、框架、数据库。无论从安全性还是可行性都是可行的。同时我们还要在项目中体现大数据风格。

**5.3操作可行性：**本系统采用简洁明了的可视化界面，用户只需鼠标点击和键盘操作、就可以完成对相关信息的修改。

#### 需求分析

**6.1功能需求**

**6.1.1后台管理系统：**

1.后台管理员的登陆。

2.超级管理员对普通管理员的注册功能

3.注册为普通用户、由超级管理员审核，批准为普通管理员

4.登陆日志记录功能

5.用户管理(用户的增删改查)

6.音乐管理(音乐的增删改查、导入、导出)

7.权限菜单：不同管理员的权限菜单不同，超级管理员可以分配菜单。

8.文件管理：由后端提供文件上传接口，后台对所上传的文件进行管理。

9.数据统计功能：用于统计音乐播放量、点赞量、评论量等数据，并提供多维度筛选和对比功能。

10.数据分析功能:用于对音乐推荐系统运营情况分析，包括用户访问量、转化率等多方面的指标统计与分析。

11.个人信息管理：该模块包括修改个人信息、上传、修改密码等功能。

**6.1.2推荐系统前台：**

1.用户的登陆与注册修改密码功能。

2.个人信息的完善与修改、包括头像上传、地址、邮箱、电话等信息。

3.热门推荐、新碟上架的展示、对处理好的热门推荐、新歌上架展示给用户。

4.音乐库、用户能够根据歌曲名对数据库中几十万条数据进行搜索查询、然后点击可以进入歌曲详情页面、然后可以对歌曲进行播放、喜欢、收藏的模拟操作。

5.热门歌手推荐、展示根据算法得出来的艺术家排名。

6.个性推荐：根据用户id以及用户操作、会为用户推荐个性化音乐、展示到前端页面。

7.推荐排行榜、对算出来的热门歌曲、新歌进行排行榜展示。

**6.2.安全性需求分析**

6.2.1账户权限控制: 不同角色拥有不同的权限控制，超级管理员可以通过角色权限来限制对数据的访问和修改。

6.2.2防止SQL注入攻击:再开发过程中，防止恶意攻击者利用SQL漏洞进行攻击。

6.2.3数据加密:敏感数据需要采取加密措施，如用户密码。

**6.3.用户体验需求分析:**

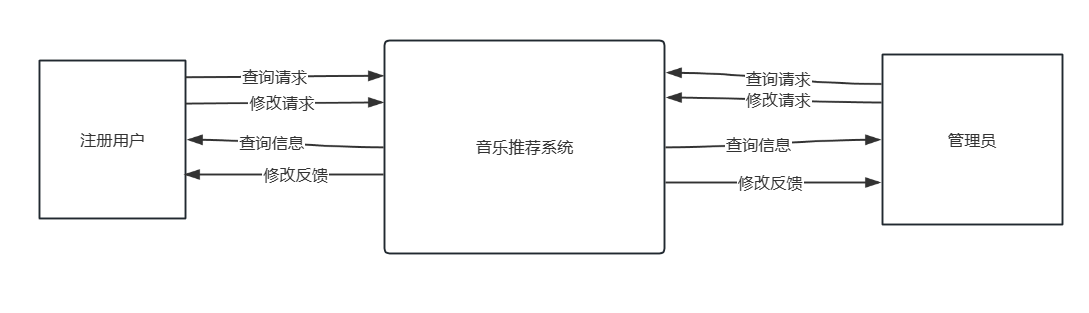
**6.3.**1界面设计：使用户操作简单方便，采用响应式布局或使用简单直观的图表展示数据。

6.3.2反馈机制：进行关键操作时、例如删除等功能时给出确认对话框、错误提示等。

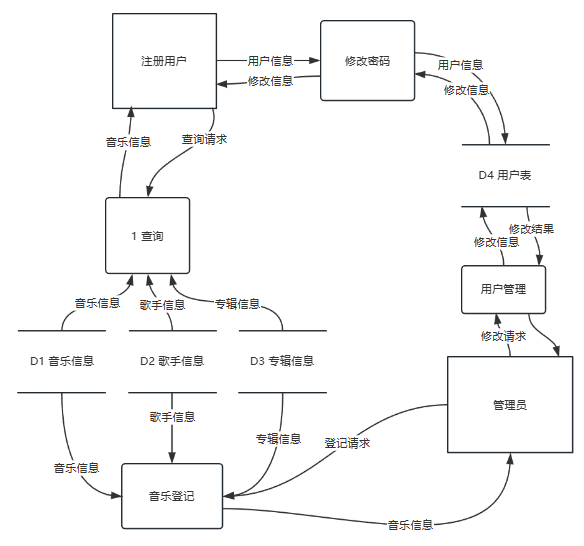
6.3.3响应速度：系统须保证快速响应、以提高用户体验和工作效率。

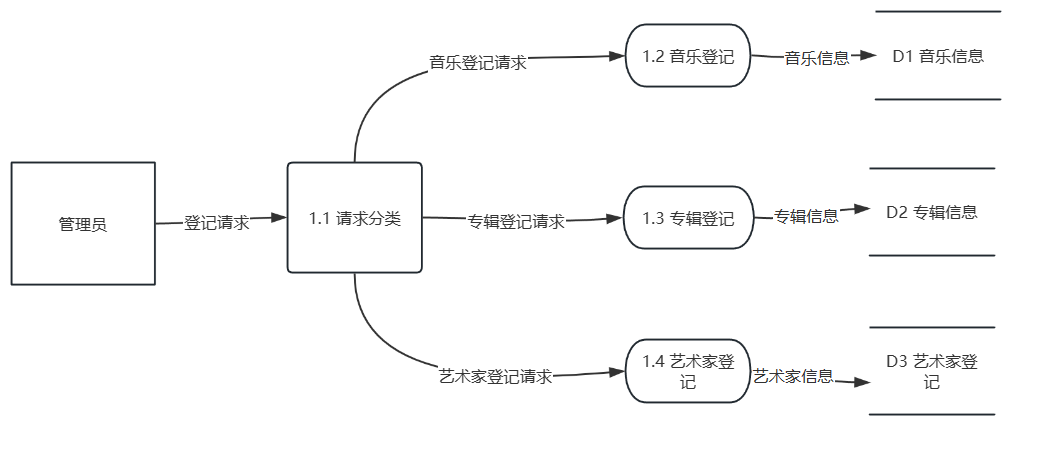
**6.4.数据流图：**

数据流图的简称为DFD，是通过使用图形的方法对系统所具备的逻辑功能进行阐述，描述系统数据的流向和逻辑变换。由于该方法能将难以阐述的问题进行表述，因此被开发者广泛应用，是当前应用范围较为广泛的结构化系统分析方法。

**6.4.1顶层数据流图：**

**6.4.2 一层DFD**



**6.4.3管理员登记音乐DFD**

**6.5数据字典：**

6.5.1数据项

|  |  |  |  |
| --- | --- | --- | --- |
| Id | 数据项名称 | 说明 | 定义 |
| username | 注册用户账号 | 注册用户账号、唯一索引 | 3{数字|字符}20 |
| password | 注册用户密码 | 注册用户密码 | 3{数字|字符}20 |
| nickname | 用户昵称 | 用户的昵称 | 2{字符}10 |
| address | 用户地址 | 地址、枚举类型 | [北京|上海|…] |
| email | 用户邮箱 | 用户邮箱 | 5{字符}20 |
| phone | 用户电话号 | 用户的电话 | 11{字符}11 |
| avatar | 用户头像 | 用户头像 | 一个url链接 |
| create\_time | 注册时间 | 用户注册时的时间 | time |
| role | 角色 | 枚举型分为Role\_Admin和Role\_User | [Role\_Admin|Role\_User] |
| sex | 性别 | 用户的性别enum{男性、女性、其他} | [男性|女性|其他] |
| age | 年龄 | 用户的年龄 | 数字 |
| menu\_name | 菜单名称 | 为用户分配的菜单名 | 5{字符}10 |
| path | 前端页面路径 | 前端页面路径 | url |
| icon | 图标 | 图标 | 字符 |
| description | 描述 | 菜单的描述 | 字符 |
| pid | 父级菜单 | 父级菜单 | 数字 |
| role\_name | 角色名 | 角色名 | 字符 |
| role\_flag | 用户标识 | enum[ROLE\_ADMIN|ROLE\_USER] | 枚举类型 |
| file\_name | 文件名 | 文件名称 | 字符 |
| file\_type | 文件类型 | 文件类型 | 字符 |
| file\_size | 文件大小 | 文件大小 | 数字 |
| file\_srl | 文件链接 | 文件的链接 | url |
| Music\_id | 音乐id | 音乐id | 数字 |
| music\_name | 音乐名称 | 音乐名称 | 字符 |
| music\_url | 音乐链接 | 音乐链接 | url |
| artist\_id | 艺术家Id | 艺术家id | 数字 |
| artist\_name | 艺术家名称 | 艺术家名称 | 字符 |
| album\_id | 专辑id | 专辑id | 数字 |
| album\_name | 专辑名称 | 专辑名称 | 字符 |

6.5.2数据结构：

|  |  |  |
| --- | --- | --- |
| 数据结构名 | 含义说明 | 组成 |
| 注册用户 | 注册用户基本信息 | username,nikename,password,address,avatar,  phone,sex,age,role |
| 音乐 | 音乐 | music\_id，music\_name,url |
| 专辑 | 专辑 | album\_id,album\_name,time |
| 艺术家 | 艺术家 | artist\_id,artist\_name |

6.5.3数据流

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据流名 | 含义说明 | 数据流来源 | 数据流去向 | 组成 | 流量 |
| 登记请求 | 管理员登记信息的请求 | 管理员 | 请求分类 | 音乐、专辑、歌手 | 随机 |
| 登记音乐请求 | 请求分类根据请求类型进行登记 | 请求分类 | 音乐登记 | 音乐信息 | 随机 |
| 登记专辑请求 | 请求分类根据请求类型进行登记 | 请求分类 | 专辑登记 | 专辑信息 | 随机 |
| 艺术家登记请求 | 请求分类根据请求类型进行登记 | 请求分类 | 艺术家登记 | 艺术家信息 | 随机 |
| 查询请求 | 用户进行查询 | 用户 | 查询 | 查询信息 | 随机 |
| 修改密码请求 | 用户提供用户信息和修改信息修改密码 | 用户 | 修改密码 | 用户信息 | 随机 |
| 用户管理请求 | 管理员提供修改请求、对用户表进行修改 | 管理员 | 用户管理 | 修改信息 | 随机 |

6.5.4数据存储

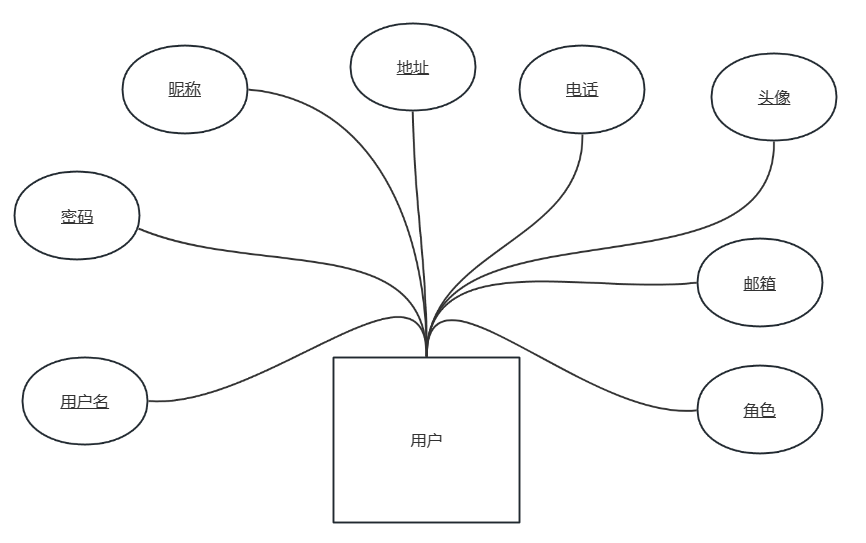
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 编号 | 数据存储名 | 说明 | 流入的数据流 | 流出的数据流 | 组成 |
| D1 | 音乐信息 | 包含音乐表的所有信息 | 管理员登记 | 用户查询信息 | 音乐 |
| D2 | 歌手信息 | 包含歌手表的信息 | 管理员登记 | 用户查询信息 | 歌手 |
| D3 | 专辑信息 | 包含专辑表的信息 | 管理员登记 | 用户查询信息 | 专辑 |
| D4 | 用户表 | 包含了用户信息 | 用户修改密码、管理员管理用户 | 用户修改结果、管理修改结果 | 用户 |

6.5.5数据处理

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 编号 | 处理过程名 | 说明 | 输入数据流 | 输出数据流 | 加工逻辑 |
| 1.1 | 请求分类 | 对管理员的登记请求分类处理 | 管理员 | 音乐登记、专辑登记、艺术家登记 | IF  管理员发出登记请求  THEN  请求分类  END IF |
| 1.2 | 音乐登记 | 登记请求为音乐 | 请求分类 | 音乐信息 | IF  音乐登记请求  THEN  音乐登记 |
| 1.3 | 专辑登记 | 登记请求为专辑 | 请求分类 | 专辑信息 | IF  专辑登记请求  THEN  专辑登记 |
| 1.4 | 艺术家登记 | 登记请求为艺术家 | 请求分类 | 艺术家信息 | IF  艺术家登记请求  THEN  艺术家登记 |

**6.6 实体-联系图（E-R图）**

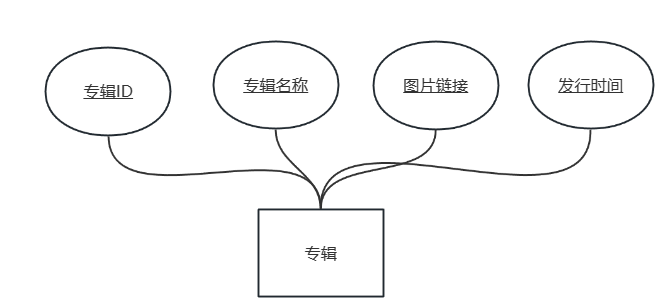
**6.6.1 用户E-R图**



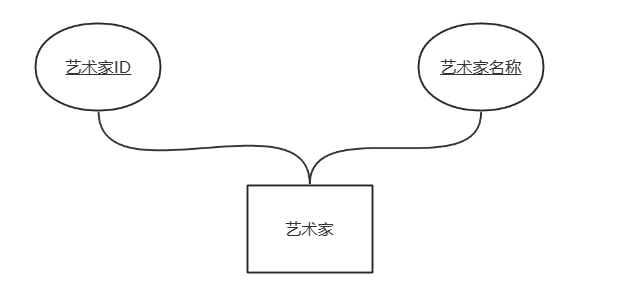
**6.6.2 音乐E-R图**



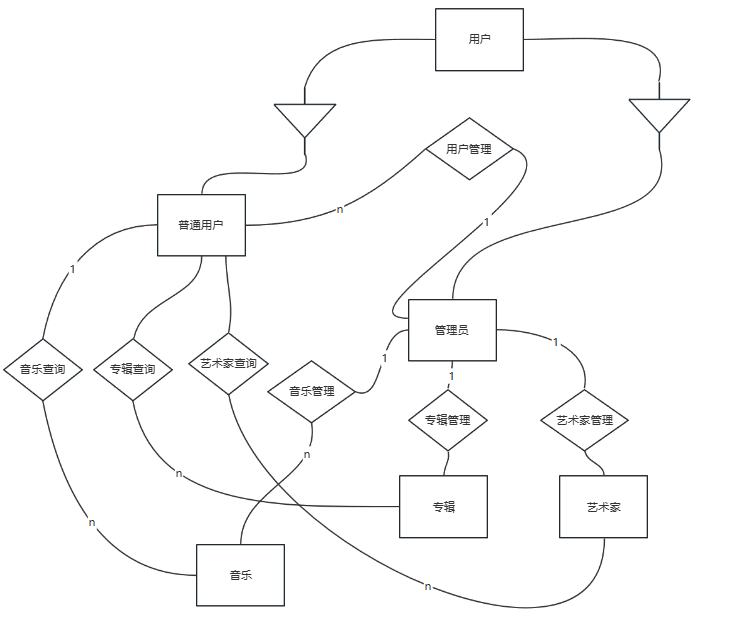
**6.6.3 专辑E-R图**



**6.6.4 艺术家E-R图**

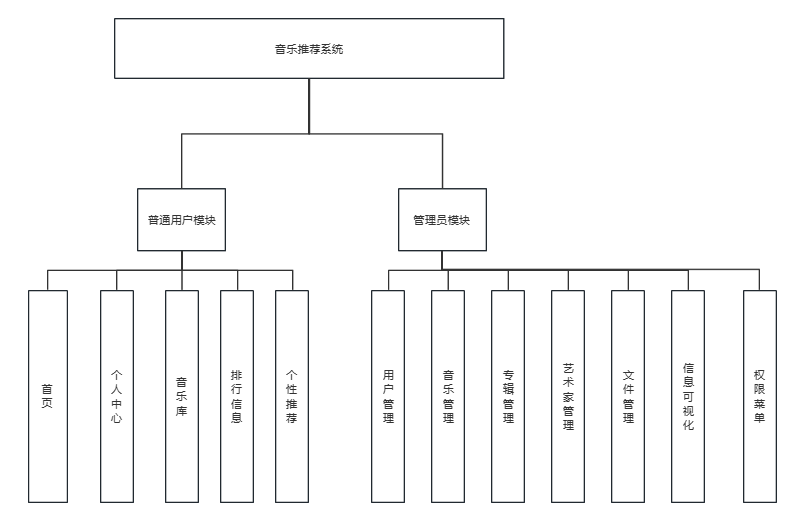


**6.6.5 总体E-R图**

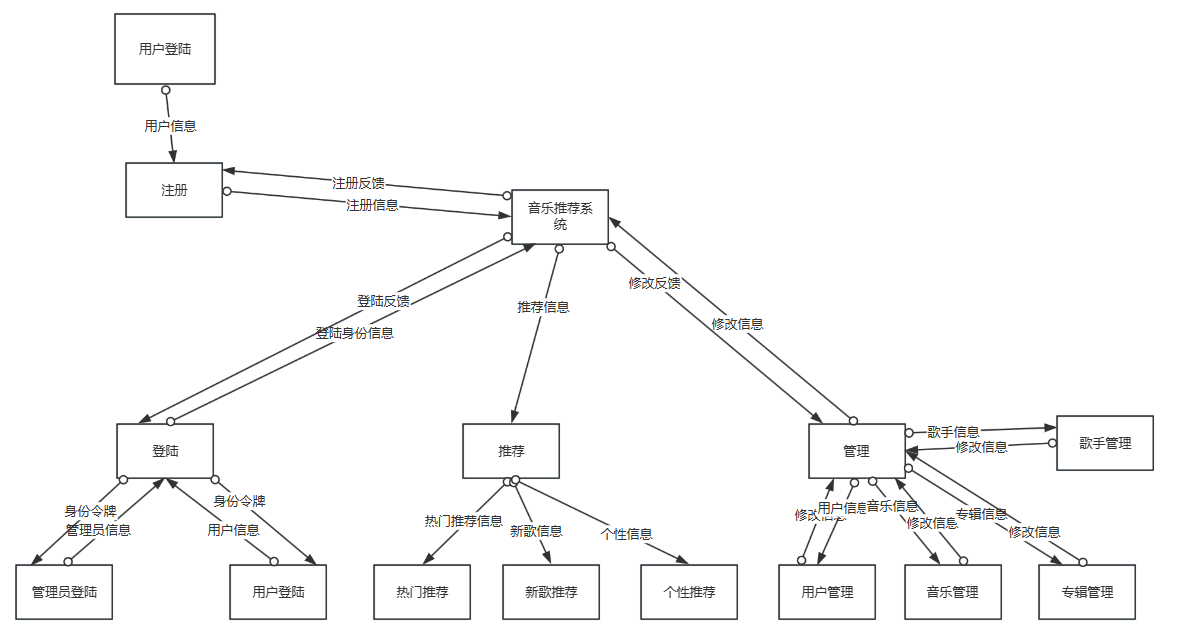


#### 总体设计

**7.1系统架构图**

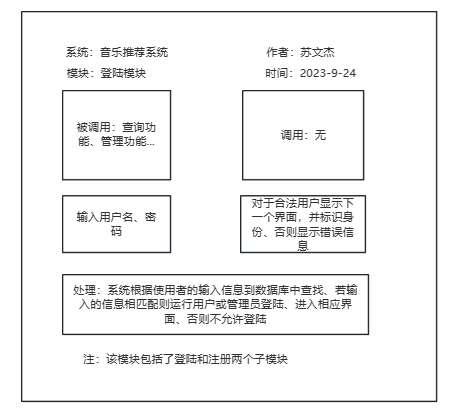


**7.1.2 系统功能图**

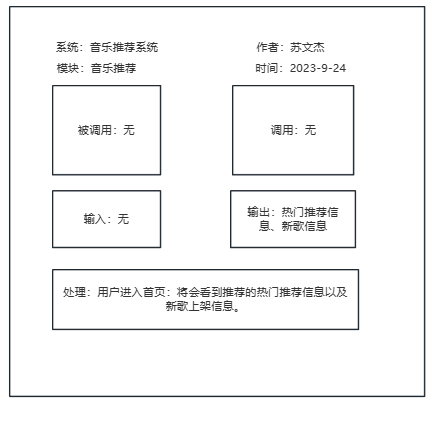


**7.1.3 模块的简要描述**

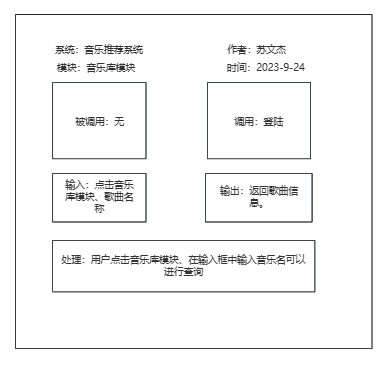
1.登陆功能IPO表



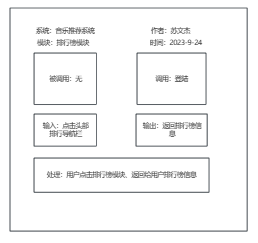
**2.首页模块IPO表**



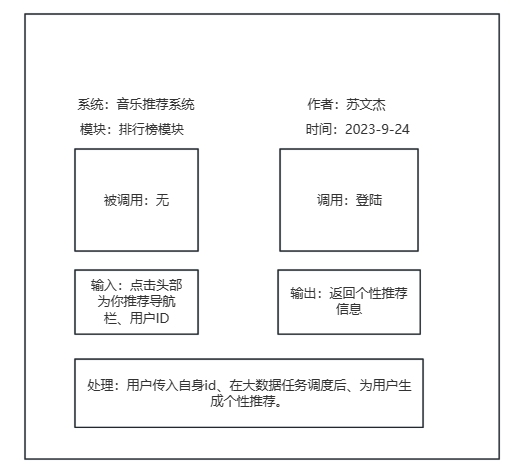
**3.音乐库模块IPO表**



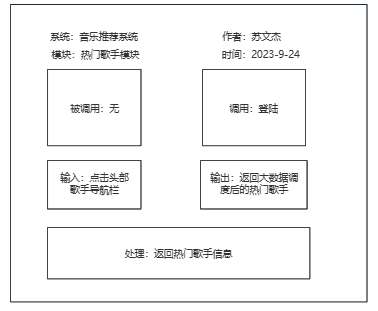
**4.排行榜模块IPO表**



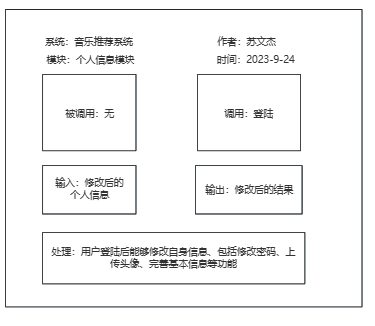
**5.个性推荐模块IPO表**



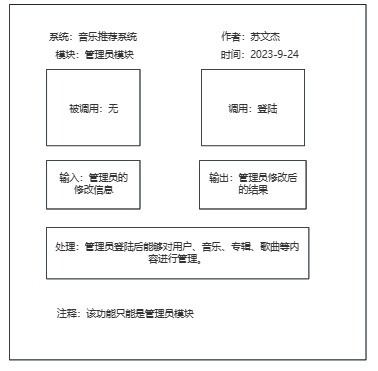
**6.热门歌手IPO表**



**7.个人信息管理IPO表**

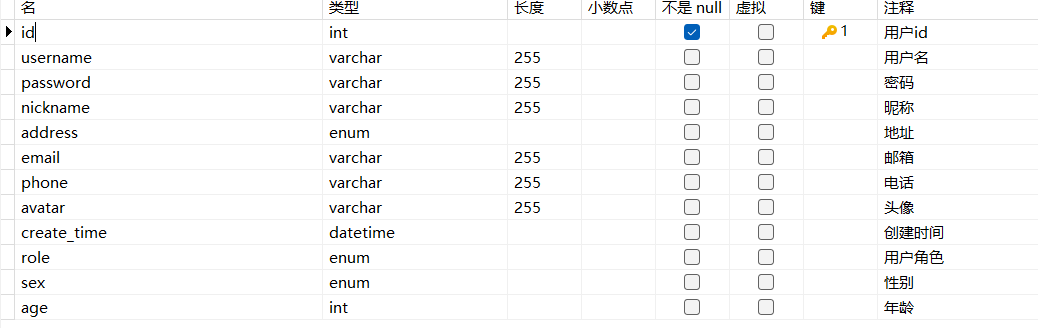


**8.管理员管理模块IPO表**



**7.1.4 数据库表结构设计**

**1.用户表（user表）**



**2.音乐表（music表）**



**3.专辑表（album表）**



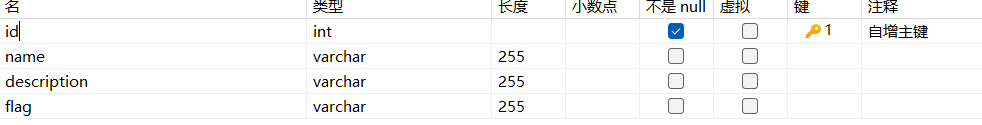
**4.艺术家表（artist表）**



**5.菜单表（menu）**



**6.角色表（role）**



**7.文件表（sys\_file）**



**8.用户操作表（user\_action）**

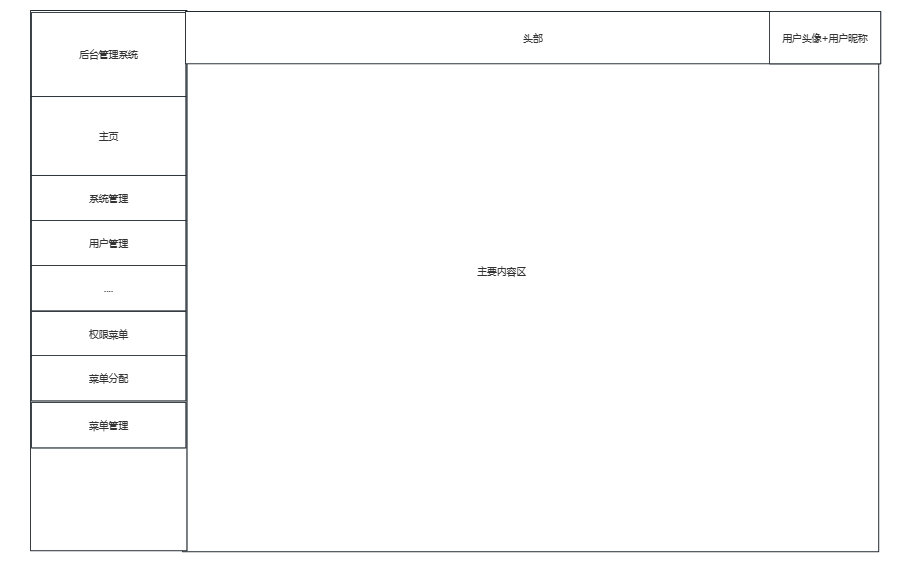
#### 详细设计

**8.1 管理员登陆框：**



**8.2 注册框**



**8.3 管理员界面UI**

**8.4 普通用户登陆界面UI设计**



#### 编码实现

**8.1 所选语言特点**

**前端： Vue 框架**

1. **轻量级：** Vue.js非常轻巧，文件大小小，因此易于学习和集成到现有项目中。
2. **双向数据绑定：** Vue提供了双向数据绑定，使数据模型和视图之间的同步变得非常简单。当数据发生变化时，视图会自动更新，反之亦然。
3. **组件化：** Vue鼓励将应用程序拆分成可重用的组件。每个组件都有自己的状态、模板和逻辑，这样可以更容易地维护和扩展应用程序。
4. **虚拟DOM：** Vue使用虚拟DOM来优化性能。它会在内存中创建一个虚拟的DOM树，然后通过比较前后两个虚拟DOM树的差异来最小化实际DOM操作，从而提高渲染性能。
5. **指令：** Vue提供了一系列指令，用于处理DOM元素。例如，v-for用于循环渲染元素，v-bind用于绑定属性，v-on用于绑定事件等。
6. **生命周期钩子：** Vue组件具有一组生命周期钩子函数，允许你在组件的不同生命周期阶段执行自定义逻辑。这些钩子包括created、mounted、updated和destroyed等。
7. **路由管理：** Vue可以与Vue Router集成，从而实现客户端路由管理。这允许你构建单页面应用（SPA），在不刷新整个页面的情况下切换视图。
8. **状态管理：** Vue可以与Vuex集成，提供了一种用于管理应用程序状态的集中式解决方案。这对于大型应用程序的状态管理非常有用。
9. **模板引擎：** Vue使用基于HTML的模板语法，允许你声明式地构建用户界面。模板可以包含简单的表达式、条件语句和循环结构。
10. **社区支持：** Vue拥有一个庞大的社区和生态系统，有丰富的第三方库和插件可供选择。这使得Vue成为一个强大而受欢迎的前端框架。

**后端框架：SpringBoot**

1. **简化配置：** Spring Boot使用约定优于配置的原则，提供了默认配置，减少了繁琐的XML配置文件。大多数情况下，你只需要很少的配置即可开始开发。
2. **内嵌式Web服务器：** Spring Boot内置了多个常用的Web服务器（如Tomcat、Jetty和Undertow），可以将应用程序打包成可执行的JAR文件或WAR文件，轻松部署和运行。
3. **自动配置：** Spring Boot根据你的项目中使用的库和依赖自动配置应用程序。这大大减少了手动配置的需求，提高了开发效率。
4. **自动化构建：** Spring Boot项目通常使用Maven或Gradle进行构建，这些构建工具可以自动管理依赖项，并生成可执行的JAR文件或WAR文件。
5. **Spring生态系统：** Spring Boot是建立在Spring框架之上的，可以轻松集成Spring的各种功能，如Spring Data、Spring Security和Spring Cloud等。
6. **生产就绪：** Spring Boot提供了一系列功能，如健康检查、监控、度量和外部配置，使应用程序更容易在生产环境中部署和管理。
7. **大型社区和生态系统：** Spring Boot拥有庞大的社区支持和丰富的生态系统，你可以轻松找到各种插件、扩展和解决方案，以满足不同项目的需求。
8. **测试支持：** Spring Boot提供了强大的测试支持，包括单元测试、集成测试和端到端测试。测试可以帮助确保应用程序的稳定性和可靠性。
9. **外部化配置：** Spring Boot允许你将配置信息从应用程序代码中分离出来，并在不同环境中进行配置，从而使你的应用程序更具可配置性。
10. **微服务支持：** Spring Boot与Spring Cloud集成，为构建微服务架构提供了强大的支持，包括服务注册和发现、负载均衡、断路器模式等。

**数据库选用：**

**MySQL 、ClickHouse、Redis**

Mysql作为业务数据库、存储web产生的业务数据、ClickHouse对大数据计算产生的数据进行存储、Redis会对一些首页数据进行缓存、以便查询因为Redis是基于内存、因此读写效率会很快。

**数据库连接框架：Mybatis Plus**

**Mybatis Plus**是一个基于MyBatis框架之上的开源持久层框架，旨在简化和增强MyBatis的功能。它提供了一些额外的功能和工具，以提高数据库操作的开发效率和可维护性。

**8.2 编码：**

**8.2.1 entity 实体类**

实体类用于表示应用程序中的数据模型，通常与数据库中的表或文档对应。它们是纯粹的Java对象，用于封装数据。

**${**swagger**}**

**${**entityLombokModel**}**  
**${**chainModel**}**

**8.2.2 Mapper层（数据访问层）**

Mapper层用于定义与数据库的交互，包括查询、插入、更新和删除数据。它通常使用ORM（对象-关系映射）框架（如Hibernate或MyBatis）来执行数据库操作。Mapper层核心代码。

<mapper namespace="**${**package.Mapper**}**.**${**table.mapperName**}**">  
<!-- 开启二级缓存 -->  
 <cache type="**${**cacheClassName**}**"/>  
 <!-- 通用查询映射结果 -->  
 <resultMap id="BaseResultMap" type="**${**package.Entity**}**.**${**entity**}**">

<result column="**${**field.name**}**"property="**${**field.propertyName**}**" />

</resultMap>  
<!-- 通用查询结果列 -->  
 <sql id="Base\_Column\_List"></sql></mapper>

**8.2.3 Service层**

Service层包含应用程序的业务逻辑，负责处理业务规则、验证数据、协调不同的数据访问操作，并对外提供服务接口。

public interface **${**table.serviceName**}** extends **${**superServiceClass**}**<**${**entity**}**> {  
// 标准化接口  
}  
**#end**

**ServiceImpl**

@Servicepublic class **${**table.serviceImplName**}** extends **${**superServiceImplClass**}**<**${**table.mapperName**}**, **${**entity**}**> implements **${**table.serviceName**}** {  
}// 实现并拓展  
**#end**

**8.2.4 Controller层**

Controller层负责处理HTTP请求和响应，它接收来自客户端的请求，调用Service层进行业务处理，然后将结果返回给客户端。

// 处理del请求  
 @DeleteMapping("/{id}")  
// 处理post请求  
 @PostMapping("/del/batch")  
// 处理get请求  
 @GetMapping  
**#end**

8.2.5 代码生成器

MyBatis Plus的代码生成器是一个非常有用的工具，它可以根据数据库表自动生成实体类、Mapper接口以及XML映射文件，从而加速开发过程，减少手动编写重复的持久层代码的工作。上述代码即使代码生成器模板，可以根据mysql数据库生成代码。

// 导入MyBatis Plus的代码生成器  
import com.baomidou.mybatisplus.generator.FastAutoGenerator;  
import com.baomidou.mybatisplus.generator.config.OutputFile;

8.2.6 配置类：

1.跨域配置：

@Configuration  
public class CorsConfig {  
  
 // 当前跨域请求最大有效时长。这里默认1天  
 private static final long *MAX\_AGE* = 24 \* 60 \* 60;  
// 1 设置访问源地址  
 ("http://localhost:8080");   
 ("http://localhost:8081");  
 ("http://192.168.43.105:8080");  
 ("http://192.168.43.105:8081");  
 ("http://192.168.43.105:80");

// 2 设置访问源请求头  
 addAllowedHeader("\*");

// 3 设置访问源请求方法  
 addAllowedMethod("\*");

// 4 设置跨域请求最大有效时长  
 setMaxAge(*MAX\_AGE*);

// 5 对接口配置跨域设置  
 source.registerCorsConfiguration("/\*\*", corsConfiguration);   
 return new CorsFilter(source);  
  
}

2.拦截器配置：

@Configuration  
public class InterceptorConfig implements WebMvcConfigurer {  
  
 @Override  
 public void addInterceptors(InterceptorRegistry registry) {  
// 拦截所有请求，通过判断token是否合法来决定是否需要登录  
// 放行静态文件  
 }  
 @Bean  
 public JwtInterceptor jwtInterceptor(){  
 return new JwtInterceptor();  
 }  
  
}

3.Mybatis Plus 分页设置

// 最新版

import PaginationInnerInterceptor;

@Configuration  
@MapperScan("com.gzu.backstage.mapper")  
public class MybatisPlusConfig {   
 @Bean  
 public MybatisPlusInterceptor mybatisPlusInterceptor() {}  
  
}

8.2.7 返回规则：定义了一系列返回代码。

public interface Constants {  
 String *CODE\_200* = "200"; //成功  
 String *CODE\_400* = "400"; //参数错误  
  
 String *CODE\_401* = "401"; //权限不足  
 String *CODE\_500* = "500"; //系统错误  
 String *CODE\_600* = "600"; //其他业务异常  
  
 String *DICT\_TYPE\_ICON* = "icon";  
  
 String *FILES\_KEY* = "FILES\_FRONT\_ALL";  
}

定义了一个Result类用于统一返回Http请求。

@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class Result {  
 private String code;  
 private String msg;  
 private Object data;  
}

8.3.1 登陆逻辑

用户在前端填写表单、向后端api发送请求。

<template>  
 登 录

登录

注册  
</template>  
  
<script>  
 username: [  
 '请输入用户名',长度在 3 到 5 个字符'   
 ],  
 password: [  
 '请输入密码', '长度在 1 到 20 个字符'  
 ]   
 mounted() {  
 ***document***.title = "登陆";  
 },  
 methods: {  
 login() {  
 // 表单校验合法  
 // 存储用户信息到浏览器  
 // 动态设置当前用户路由

// 处理服务器返回结果

// 成功  
 this.$message.success("登陆成功")  
 this.$router.push("/")  
 // 失败   
 this.$message.error("暂无权限登陆后台")

// 提示其他错误  
 this.$message.error(res.msg)   
  
</script>

后端接收：

@RestController  
@RequestMapping("/user")  
public class UserController {  
  
 @Resource  
 private IUserService userService;  
/\*\* 其他代码省略 \*\*/  
 @PostMapping("/login")  
 public Result login(@RequestBody UserDTO userDTO){  
 String username = userDTO.getUsername();  
 String password = userDTO.getPassword();  
 if(StrUtil.*isBlank*(username) || StrUtil.*isBlank*(password)){  
 return Result.*error*(Constants.*CODE\_400*,"参数错误");  
 }  
 UserDTO dto = userService.login(userDTO);  
 return Result.*success*(dto);  
 }  
}

由UserService处理登陆逻辑

@Service  
 @Override  
 public UserDTO login(UserDTO userDTO) {

// 处理登陆失败  
 // 用户密码使用MD5加密  
 // 设置token  
 //增加一条用户登陆信息  
 // 设置用户的菜单列表  
 }

如果用户输入了正确的信息会得到一个token、然后用户会根据请求头去请求其他数据。

@Component  
public class TokenUtils {  
  
 // 将 user id 保存到 token 里面,作为载荷  
 // 2小时后token过期  
 // 以 password 作为 token 的密钥

}

当Token无效也会对用户请求进行拦截：

throw new ServiceException(Constants.*CODE\_401*, "无token，请重新登录");  
throw new ServiceException(Constants.*CODE\_401*, "token验证失败，请重新登录");  
throw new ServiceException(Constants.*CODE\_401*, "用户不存在，请重新登录");  
throw new ServiceException(Constants.*CODE\_401*, "token验证失败，请重新登录");

#### 个人总结

时间过得很快、为期两个月的实训即将接近尾声，在这八周的实训过程中，我们每天都过得很充实、我们从0到1搭建出了我们的音乐推荐系统、虽然结果可能不如预期、但是我们却收获很多，从项目的需求分析、调研、进度计划到数据库、架构的设计、再到编码实现。通过规范的软件开发流程让我受益匪浅。

这两个月对我最大的收获就是学会使用了SpringBoot 和vue框架来进行Web系统的开发，同时我也学会了多技术栈的配合使用。要对大数据处理的结果进行展示交互。在开发过程中也遇到了很多难题、自己独立去解决问题的能力也得到了提升。这次开发深深的让我意识到、要做大型项目的开发必须要有清晰的开发思路、要有规范的开发流程、单凭一两个人是难以实现复杂项目的。

## 过程调度

此部分介绍使用Azkaban实现各过程联合调度的过程。

### Azkaban

Azkaban是一个开源的批处理工作流调度系统，用于管理和调度作业流程。它提供了一个直观的用户界面和强大的调度功能，能够帮助用户实现高效的作业调度和任务管理。下面是关于Azkaban的简要介绍以及调度部署流程的描述：

1. Azkaban简介：
   * Azkaban提供了一个用户友好的Web界面，使用户可以轻松创建、编排、调度和监控批处理作业。
   * 它支持多种类型的作业，如Shell脚本、Java程序、Hadoop作业等，并提供了灵活的调度策略和可视化监控。
   * Azkaban具有良好的可扩展性和可靠性，适用于大规模的数据处理和调度场景。
2. 调度部署流程：

a. 安装和配置Azkaban服务器：

* + 下载Azkaban服务器，并解压到指定目录。
  + 在azkaban.properties配置文件中，配置数据库、执行器和邮件通知等相关参数。
  + 配置SSL证书（可选），以提供安全的Web访问。

b. 启动Azkaban服务器：

* + 使用启动脚本或命令行启动Azkaban服务器。
  + 在浏览器中访问Azkaban Web界面，确保服务器正常运行。

c. 创建项目和任务：

* + 在Azkaban Web界面中，创建一个新项目，并指定项目的名称和描述等信息。
  + 在项目中创建任务流程，添加需要执行的作业任务和依赖关系。

d. 配置执行器：

* + 在Azkaban服务器上，配置执行器（Executor）来执行任务作业。
  + 配置执行器的名称、属性和运行环境等相关参数。

e. 设置调度：

* + 在Azkaban Web界面中，配置任务的调度策略，如定时调度、依赖调度等。
  + 设置作业的调度时间和频率，以达到期望的调度效果。

f. 监控调度任务：

* + 使用Azkaban的监控功能，实时查看任务的执行结果和日志输出。
  + 根据需求设置告警通知，以便及时发现和处理任务执行中的异常情况。

g. 执行和管理任务：

* + 在Azkaban Web界面中，手动触发任务的执行，或等待系统按照预定的调度时间自动执行。
  + 监控任务的执行状态和结果，如有需要，可以手动终止或重新执行任务。

h. 定期维护和优化：

* + 定期检查和更新Azkaban服务器，确保版本为最新，并修复已知的问题和漏洞。
  + 优化调度策略和作业配置，以提高整体的任务执行效率和性能。

在完成以上调度部署流程后，我们将能够通过Azkaban进行作业调度和任务管理。根据实际需求和业务场景，可以进一步扩展和调整Azkaban的功能和配置，以满足更复杂的任务调度和管理需求。

### 调度计划

每天凌晨12点按顺序调度：

1. 统计三天操作数据并计算音乐热度评分
2. 统计两天操作变化率
3. 统计用户活跃度
4. 处理热门音乐
5. 获取登录用户ID并进行个性推荐

每周六凌晨12点按顺序调度：

1. 同步音乐表
2. 统计并推荐热门艺术家

每月1号凌晨12点执行：

1. 统计VIP复购率

### 使用Azkaban实现过程调度

使用Azkaban可以很方便地实现上述调度计划。以下是Azkaban实现调度计划的示例：

要开始一个Azkaban项目，首先我们需要创建一个azkaban.project文件，用来定义项目的基本配置信息和流程。

之后创建一个basic.flow文件，定义工作节点和节点之间的依赖关系。

将以上两个文件和需要使用的依赖打包成zip文件，在Azkaban的Web界面中新建项目并上传，Azkaban便会自动解析该文件并生成工作流以备调度。

接下来：

1. 创建一个每天凌晨12点调度的Project（music\_recommend\_1d）：
   * 添加一个并行执行的Job节点，按顺序包含以下任务：
     + 统计三天操作数据并计算音乐热度评分（Job节点mika）
     + 统计两天操作变化率（Job节点hennka）
     + 统计用户活跃度（Job节点useralive）
     + 处理热门音乐（Job节点popmusic）
     + 获取登录用户ID并进行个性推荐（Job节点personalre）
2. 创建一个每周六凌晨12点调度的Project（music\_recommend\_1w）：
   * 添加一个并行执行的Job节点，按顺序包含以下任务：
     + 同步音乐表（Job节点shinnkyoku）
     + 统计并推荐热门艺术家（Job节点popArtist）
3. 创建一个每月1号凌晨12点调度的Project（music\_recommend\_1m）：
   * 添加一个Job节点，用于统计VIP复购率（Job节点shinnkyoku）

通过Azkaban的Web界面，我们可以设置每个Job节点的调度时间和依赖关系，以确保任务按照预定的顺序和时间触发和执行。同时还可以查看任务的执行历史、日志和状态，通过Azkaban提供的监控和报警功能来实时监控任务的执行情况。

## 监控报警

此部分主要介绍使用Prometheus实现运行指标监控与告警的过程。

### Prometheus

Prometheus是一个开源的监控和时间序列数据库系统，它能够帮助我们收集、存储和分析应用程序和系统的监控数据。以下是Prometheus的一些主要功能和能力：

1. 数据采集和存储：Prometheus可以通过各种插件和客户端库来采集应用程序和系统的监控数据。它支持多种数据模型，能够存储和保留采集的时间序列数据，并提供内置的本地存储引擎。
2. 指标查询和展示：Prometheus提供了强大的查询语言（PromQL），使开发者能够对存储的时间序列数据进行丰富的查询和分析。开发者可以使用PromQL查询指标数据，构建灵活的仪表盘和图表，以展示系统的状态和性能。
3. 警报和通知：Prometheus支持设置自定义的警报规则和阈值，当指标数据达到或超过设定的阈值时，会触发警报并发送通知。开发者可以选择通过电子邮件、Slack、PagerDuty等方式接收警报通知，以及定义自定义的警报动作。
4. 自动发现和服务发现：Prometheus支持自动发现目标，它可以自动从配置的目标集合中发现和连接到运行的应用程序和服务。它还支持多种服务发现机制，如标签发现、服务发现和DNS发现，以简化配置管理和监视的过程。
5. 可扩展性和集成性：Prometheus具有可扩展的架构，可以适应大规模和高频率的度量采集。它支持与其他工具和系统的集成，如Grafana、Alertmanager、Kubernetes、Prometheus Operator等，以提供更全面的监控解决方案。

总的来说，Prometheus提供了一个完整的监控解决方案，可以帮助我们收集、存储、查询和展示系统和应用程序的监控数据。它具有强大的查询语言和丰富的可视化能力，能够帮助我们实时监控系统性能、发现潜在问题并采取相应的行动。

### 使用Prometheus实现指标监控

#### 使用Prometheus监控MySQL

按照以下步骤进行设置，可实现使用 Prometheus 监控 MySQL：

1. 安装和配置 Prometheus：
   * 下载 Prometheus 并解压：在 Prometheus 的官方网站下载适用于操作系统的版本，并解压到合适的位置。可通过Docker容器快速开始Prometheus的使用。
   * 修改配置文件：在 Prometheus 的配置文件 prometheus.yml 中添加以下内容，以便于监控 MySQL 数据库：

|  |
| --- |
| scrape\_configs:  - job\_name: 'mysql'  static\_configs:  - targets: ['spark101:9104'] |

* + 启动 Prometheus：运行 Prometheus 的可执行文件，它将自动加载配置文件并开始监控。

1. 安装和配置 mysqld\_exporter：
   * 下载 mysqld\_exporter：在 mysqld\_exporter 的 GitHub 仓库中下载适用于操作系统的版本。
   * 启动 mysqld\_exporter：运行 mysqld\_exporter 的可执行文件，它将默认监听在端口 9104，并提供 MySQL 监控指标。
2. 配置 MySQL 数据库：
   * 确保 MySQL 允许远程访问：在 MySQL 的配置文件中（例如 /etc/my.cnf 或 /etc/mysql/mysql.conf.d/mysqld.cnf），设置 bind-address 配置为 0.0.0.0，以允许远程连接。
   * 配置 MySQL 用户和权限：创建一个专门用于监控的 MySQL 用户，并授予该用户适当的权限，例如 REPLICATION CLIENT、PROCESS 等。
   * 重启 MySQL 服务：根据操作系统，使用适当的命令重启 MySQL 服务，以使配置更改生效。
3. 验证监控配置：
   * 在浏览器中打开 Prometheus 的 Web 界面，根据我们集群部署的情况，Prometheus地址为 http://spark102:9090。
   * 在查询输入框中输入 mysql\_up 执行查询，确保 mysqld\_exporter 已经成功与 MySQL 建立连接并提供数据。
4. Grafana 可视化：
   * 安装和配置 Grafana：根据 Grafana 的官方网站下载适用于您操作系统的版本，并完成配置。可通过Docker容器快速开始Granafa的使用。
   * 添加数据源：在Grafana的Web界面中创建仪表盘，并添加Prometheus作为数据源。可选择通过ID导入在线仪表盘。Grafana服务器部署时默认启用3000端口，根据我们的集群部署情况，Grafana的地址为http://spark102:3000。

通过以上步骤将使 Prometheus 开始抓取MySQL的指标数据，并将其存储在 Prometheus 中，并通过Grafana进行指标的可视化。我们可以在Grafana中设定报警规则监控特定指标，当指标发生异动的时候Grafana会第一时间发邮件提醒开发者引起注意。

#### 使用Prometheus监控Spark应用

要监控 Spark 程序并将其与 Prometheus 集成，需要执行以下步骤：

1. 启用 Spark 的 Prometheus 监控：
   * 在 Spark 的启动脚本中，添加以下配置参数：

|  |
| --- |
| **--conf "spark.metrics.conf=<path-to-prometheus-configuration-file>"**  **--conf "spark.metrics.namespace=<your-metrics-namespace>"**  **--conf "spark.metrics.prometheus.enabled=true"** |

其中，<path-to-prometheus-configuration-file> 是 Prometheus 配置文件的路径，<your-metrics-namespace> 是用于区分 Spark 指标的命名空间。

1. 配置 Prometheus 的抓取器：
   * 在 Prometheus 的配置文件 prometheus.yml 中添加以下配置，以定期抓取 Spark 的指标：

|  |
| --- |
| scrape\_configs:  - job\_name: 'spark'  static\_configs:  - targets: ['<spark-master>:4040'] # 将 <spark-master> 替换为 Spark 主节点的地址 |

1. 重新启动 Spark 程序以应用更改，并启动 Prometheus。
2. 在 Prometheus Web 界面中，确认 Spark 的指标已经添加并开始抓取。
3. 使用 Grafana：
   * 安装和配置 Grafana，设置 Prometheus 数据源。
   * 创建自定义的监控面板，使用 Prometheus 提供的 Spark 指标展示 Spark 应用程序的关键信息。

以上步骤将使 Prometheus 开始抓取 Spark 程序的指标数据，并将其存储在 Prometheus 中供查询和可视化。开发者可以根据 Prometheus 和 Grafana 的文档进一步设置和调整监控环境。

#### 使用Prometheus监控YARN

要监控 YARN 使用 Prometheus，可以按照以下步骤进行设置：

1. 安装和配置 NodeExporter：
   * 下载 Prometheus NodeExporter：在 Prometheus 的官方网站下载适用于操作系统的版本，并解压到合适的位置。
   * 启动 NodeExporter：运行 NodeExporter 的可执行文件，它将默认监听在端口 9100，并公开本机节点的机器级指标。
2. 配置 Prometheus 的抓取器：
   * 在 Prometheus 的配置文件 prometheus.yml 中添加以下配置，以定期抓取 YARN 的指标：

|  |
| --- |
| scrape\_configs:  - job\_name: 'yarn'  static\_configs:  - targets: ['<yarn-resourcemanager>:<node-exporter-port>'] # 将 <yarn-resourcemanager> 替换为 YARN 资源管理器的地址，将 <node-exporter-port> 替换为 NodeExporter 的端口（默认为 9100） |

1. 重新启动 Prometheus。
2. 在 Prometheus Web 界面中，确认 YARN 的指标已经添加并开始抓取。
3. 使用 Grafana：
   * 安装和配置 Grafana，设置 Prometheus 数据源。
   * 创建自定义的监控面板，使用 Prometheus 提供的 YARN 指标展示 YARN 集群的关键信息。

以上步骤将使 Prometheus 开始抓取 YARN 的指标数据，并将其存储在 Prometheus 中供查询和可视化。开发者可以根据 Prometheus 和 Grafana 的文档进一步设置和调整监控环境。

### 使用Python脚本监控Java进程

为了使推荐系统的运行更加稳健，监控报警需要覆盖到更广的层面。对此我们引入了一段Python脚本监控Java进程，在Java进程异常退出的时候会发送邮件通知开发者进行处理。脚本的具体运行过程如下：

* 获取进程号：这里会通过Java的进程名获取对应的进程号。具体实现方式是利用Java Development Kit（JDK）提供的jps命令，执行该命令会返回当前正在运行的Java进程的进程号和对应的进程名。
* 捕捉进程状态：进程状态变为STOPPED时表明进程已停止运行，下一步需要获取退出代码决定是否告警。
* 通过SMTP发送报警邮件：当检测到非0代码退出时立即发送告警邮件通知开发者。

这个脚本将在后台持续运行，对推荐系统的稳健运行负责，为推荐系统保驾护航。可以在监控的过程中动态检测Java进程的变化情况，当需要监控的进程被创建时自动将该进程纳入监控。

# 测试报告

## 爬虫测试

此部分介绍使用手工测试方式对爬虫进行单元测试的步骤和场景。通过手工测试，可以验证爬虫的功能是否正常运行，数据是否准确获取和存储。

### 单元测试

进行单元测试是一种测试策略，旨在测试软件中的最小功能单元（即模块或类）的正确性和可靠性。对于爬虫这样的功能组件，进行单元测试具有重要的目的和意义：

1. 确保正确性：单元测试能够验证爬虫功能的正确性，即确保爬虫能够按照预期行为来进行数据的抓取和处理。通过编写测试用例并执行测试，可以验证爬虫是否正确处理不同的输入和边界条件，以及是否返回了预期的输出结果。
2. 提高可靠性：通过单元测试，可以发现和修复潜在的错误和缺陷，从而提高爬虫的可靠性和稳定性。单元测试能够及早地检测到问题，并帮助开发人员迅速迭代和修复bug，以提供更可靠的爬虫功能。
3. 简化调试和排除问题：当爬虫出现问题时，单元测试可以作为调试工具来定位问题所在。通过针对特定功能单元的测试，可以迅速排除影响因素，缩小故障范围，并有针对性地进行错误定位、修复和验证。
4. 支持持续集成和自动化测试：单元测试是持续集成和自动化测试的基础。通过编写可重复执行的测试用例，可以将单元测试纳入持续集成流水线中，实现自动化测试框架的开发和运行，从而提高整个开发流程的效率和质量。
5. 促进代码质量和可维护性：编写单元测试通常需要优秀的代码架构和结构，以便模块化测试和模拟测试环境。通过编写可测试的代码，可以促进编码质量和可维护性，并帮助开发人员更好地理解和组织软件组件。

综上所述，单元测试对于爬虫这样的功能组件具有重要的目的和意义，可以帮助确保爬虫功能的正确性和可靠性，提高调试和排查问题的效率，以及支持持续集成和自动化测试流程的建立。通过合理编写和执行单元测试，可以提高爬虫的质量、可维护性和稳定性。

### 单元划分

基于在编码实现部分展示的爬虫代码，将划分如下测试单元，以便在进行手动测试时可以有一个明确的测试范围：

1. **解析起始URL单元测试：**
   * 测试是否能正确解析起始URL的响应。
   * 测试是否能正确提取下一个URL。
   * 测试是否正确构造新的请求。
2. **解析新歌榜URL单元测试：**
   * 测试是否能正确解析新歌榜URL页面的响应。
   * 测试是否能正确提取音乐列表URL。
   * 测试是否正确构造新的请求。
3. **解析音乐列表URL单元测试：**
   * 测试是否能正确解析音乐列表URL页面的响应。
   * 测试是否能正确提取每首歌曲的URL。
   * 测试是否正确构造新的请求。
4. **解析音乐信息单元测试：**
   * 测试是否能正确解析音乐URL页面的响应。
   * 测试是否能正确提取音乐的URL、标题和包含该首歌的歌单URL。
   * 测试是否正确构造新的请求。
5. **解析歌单分类信息单元测试：**
   * 测试是否能正确解析包含歌单URL页面的响应。
   * 测试是否能正确提取歌单的分类标签。
6. **整体功能测试：**
   * 测试整个爬虫流程是否能够正常运行，包括从起始URL解析到最终的歌单分类信息的获取。
   * 测试是否能够正确处理各种异常情况，如网络问题、页面结构变化等。

在进行手动测试时，可以根据以上测试单元的划分，逐个测试各个单元的功能是否正确和完整。测试单元的划分有助于更好地组织测试工作，并确保每个功能点都经过了充分的测试。可以根据需要进一步对每个测试单元进行细分或添加其他测试单元，以覆盖更广泛的功能和场景。

### 手工单元测试

根据上一步骤的单元划分，设计测试用例、执行测试、比较测试结果和预期，验证爬虫的功能是否正常运行，数据是否准确获取和存储。

#### 测试环境

* Windows 11家庭版
* Python 3.11.4
* Scrapy 2.9.0
* MySQL 8.0.33

#### 测试方案

运用IDE的断点功能控制程序执行进度，检查关键节点控制台输出或日志是否与预期输出相符。相符则认为测试通过，否则认为测试不通过。

进行整体功能测试时需要检查写入数据库的记录是否正确无差错，重复记录、异常记录是否被正确处理。

#### 测试用例

1. **解析起始URL单元测试：**

输入：无

预期：能够正确解析起始URL响应，能够正确提取下一个URL，能够正确构造新的请求，并能在日志中找到。

1. **解析新歌榜URL单元测试：**

输入：无

预期：能够正确解析新歌榜URL页面的响应，能够正确提取音乐列表URL，能够正确构造新的请求，并能在日志中找到。

1. **解析音乐列表URL单元测试：**

输入：无

预期：能够正确解析音乐列表URL页面的响应，能够正确提取每首歌曲的URL，能够正确构造新的请求，并能在日志中找到。

1. **解析音乐信息单元测试：**

输入：无

预期：能够正确解析音乐URL页面的响应，能够正确提取音乐的URL、标题和包含该首歌的歌单URL，能够正确构造新的请求，并能在日志中找到。

1. **解析歌单分类信息单元测试：**

输入：无

预期：能正确解析包含歌单URL页面的响应，能正确提取歌单的分类标签，并能在日志中找到。

1. **整体功能测试：**

输入：无

预期：整个爬虫流程能够正常运行，包括从起始URL解析到最终的歌单分类信息的获取。并能在日志中找到。写入数据库中的记录正确无差错，重复记录、异常记录能够被正确处理。

#### 测试结果分析

1. **解析起始URL单元测试：**

日志信息

|  |
| --- |
| **DEBUG: Crawled (200) <GET https://music.163.com> (referer: None)**  **Crawled (200) <GET https://music.163.com/discover/toplist> (referer: https://music.163.com)** |

分析：从日志信息中可以看到，起始URL响应能够正确解析，下一个URL能够正确提取，能够正确构造新的请求。

结论：测试通过

1. **解析新歌榜URL单元测试：**

日志信息

|  |
| --- |
| **Crawled (200) <GET https://music.163.com/discover/toplist?id=3779629> (referer: https://music.163.com/discover/toplist)**  **Crawled (200) <GET https://music.163.com/song?id=2082700997> (referer: https://music.163.com/discover/toplist?id=3779629)**  **Crawled (200) <GET https://music.163.com/song?id=2081773508> (referer: https://music.163.com/discover/toplist?id=3779629)** |

分析：从日志信息中可以看到，新歌榜URL页面的响应能够正确解析，音乐列表URL能够正确提取，能够正确构造新的请求。

结论：测试通过

1. **解析音乐列表URL单元测试：**

日志信息

|  |
| --- |
| **Crawled (200) <GET https://music.163.com/song?id=2081587825> (referer: https://music.163.com/discover/toplist?id=3779629)**  **Crawled (200) <GET https://music.163.com/song?id=2079246062> (referer: https://music.163.com/discover/toplist?id=3779629)**  **Crawled (200) <GET https://music.163.com/song?id=2081196527> (referer: https://music.163.com/discover/toplist?id=3779629)** |

分析：从日志信息中可以看到，音乐列表URL页面的响应能够正确解析，每首歌曲的URL能够正确提取，能够正确构造新的请求。

结论：测试通过

1. **解析音乐信息单元测试：**

日志信息

|  |
| --- |
| **Crawled (200) <GET https://music.163.com/playlist?id=487958524> (referer:** [**https://music.163.com/song?id=2081773508**](https://music.163.com/song?id=2081773508)**)**  **Crawled (200) <GET https://music.163.com/playlist?id=6785387087> (referer: https://music.163.com/song?id=2081196527)** |

分析：从日志信息中可以看到，音乐URL页面的响应能够正确解析，音乐的URL、标题和包含该首歌的歌单URL能够正确提取，能够正确构造新的请求。

结论：测试通过

1. **解析歌单分类信息单元测试：**

日志信息

|  |
| --- |
| **Scraped from <200 https://music.163.com/playlist?id=7659356137>**  **{'cate': '欧美', 'title': '爱的回归线 (Live版)', 'url': '/song?id=2079246062'}**  **Scraped from <200 https://music.163.com/playlist?id=7659356137>**  **{'cate': '流行', 'title': '爱的回归线 (Live版)', 'url': '/song?id=2079246062'}**  **Scraped from <200 https://music.163.com/playlist?id=487958524>**  **{'cate': '华语', 'title': 'Red', 'url': '/song?id=2081773508'}** |

分析：从日志信息中可以看到，包含歌单URL页面的响应能正确解析，歌单的分类标签能正确提取。

结论：测试通过

1. **整体功能测试：**

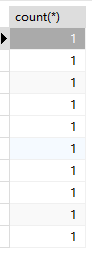
日志信息

|  |
| --- |
| **[scrapy.statscollectors] INFO: Dumping Scrapy stats:**  **{'downloader/request\_bytes': 150712,**  **'downloader/request\_count': 338,**  **'downloader/request\_method\_count/GET': 338,**  **'downloader/response\_bytes': 11727821,**  **'downloader/response\_count': 338,**  **'downloader/response\_status\_count/200': 338,**  **'dupefilter/filtered': 58,**  **'elapsed\_time\_seconds': 15.445583,**  **'finish\_reason': 'finished',**  **'finish\_time': datetime.datetime(2023, 9, 19, 9, 33, 48, 36113),**  **'httpcompression/response\_bytes': 44060630,**  **'httpcompression/response\_count': 338,**  **'item\_dropped\_count': 182,**  **'item\_dropped\_reasons\_count/DropItem': 182,**  **'item\_scraped\_count': 435,**  **'log\_count/DEBUG': 777,**  **'log\_count/INFO': 10,**  **'log\_count/WARNING': 182,**  **'request\_depth\_max': 4,**  **'response\_received\_count': 338,**  **'scheduler/dequeued': 338,**  **'scheduler/dequeued/memory': 338,**  **'scheduler/enqueued': 338,**  **'scheduler/enqueued/memory': 338,**  **'start\_time': datetime.datetime(2023, 9, 19, 9, 33, 32, 590530)}** |

数据库记录



*SELECT \* FROM music\_cate ORDER BY id DESC LIMIT 10*



*SELECT count(\*) FROM music\_cate GROUP BY url, cate LIMIT 10*

分析：整个爬虫流程，包括从起始URL解析到最终的歌单分类信息的获取均能够正常运行。

根据日志显示，整个爬虫过程中执行了338个请求，其中所有请求都是通过GET方法发送的；收到了338个响应，所有响应的状态码都是200。所有请求都得到了正确响应。

在处理过程中，有58个重复的请求被过滤掉。

在爬虫的执行过程中，总共抓取到了435个item，有182个item被丢弃，原因是DropItem。这是管道处理数据过程中，在数据库中检测到重复记录时抛出的异常类型。

数据库查询结果显示写入数据库中的记录正确无差错，重复记录、异常记录能够被正确处理。

结论：测试通过

## 算法模块测试

1.个性化音乐推荐：根据用户对音乐的评分来推荐相关音乐

uid\_music表测试数据：

┌─uid─┬─music\_id───┬─plays─┬─total\_plays─┐

│ 0 │ 1004186 │ 2 │ 3 │

│ 1 │ 1003747 │ 2 │ 2 │

│ 11 │ 1408763802 │ 1 │ 2 │

│ 11 │ 1836100414 │ 1 │ 2 │

│ 14 │ 1957475466 │ 1 │ 3 │

│ 14 │ 544151691 │ 2 │ 3 │

│ 17 │ 473202141 │ 1 │ 1 │

└─────┴────────────┴───

Uid：0的个性化推荐表(推荐5首):

─user\_id─┬─recommended\_music─┐

│ 0 │ ['1004186'] │

│ 0 │ ['1003747'] │

│ 0 │ ['1408763802'] │

│ 0 │ ['1836100414'] │

│ 0 │ ['1957475466'] │

结果：成功推荐

Uid：11

│ 11 │ ['1408763802'] │

│ 11 │ ['1836100414'] │

│ 11 │ ['1004186'] │

│ 11 │ ['1003747'] │

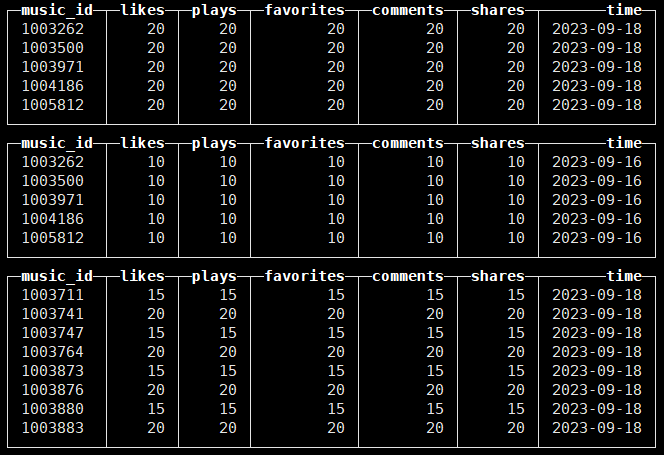
│ 11 │ ['1957475466'] │

└─────────┴──────────────────

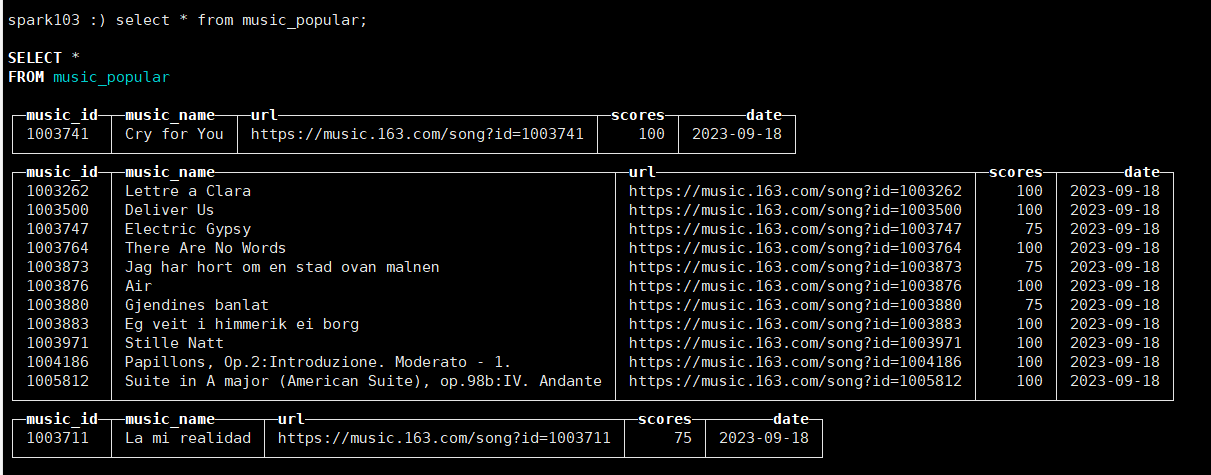
Uid为11的未在表music\_recommend中，在删除复写时未报

2.热门音乐推荐：获得前一天得分最高的20条音乐的数据

从music\_partition表中获得数据：

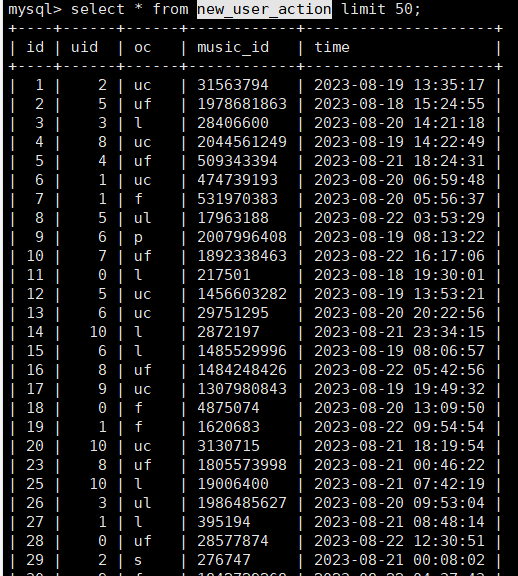


运行PopularMusic，从中获得9.18号的数据处理后写入clickhouse



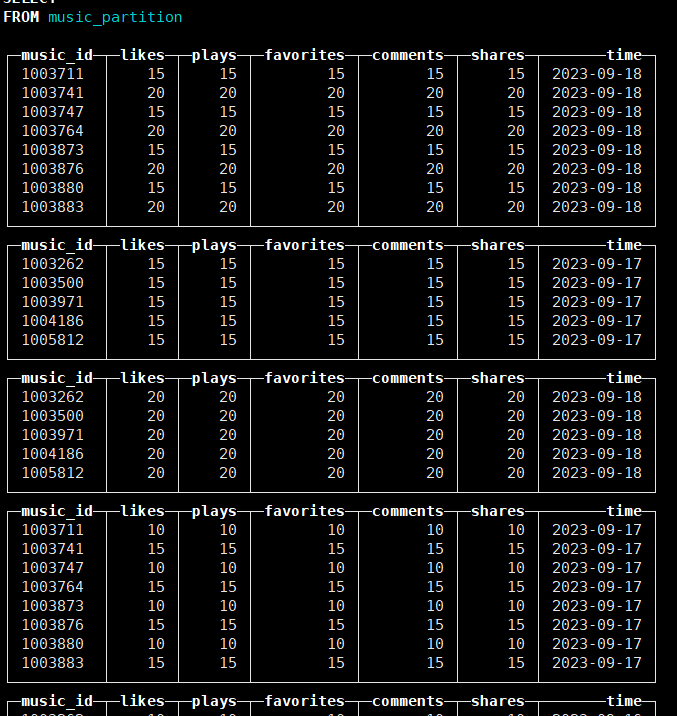
3.用户活跃量统计：统计前一天进行操作的用户的数量

从mysql的new\_user\_action获得用户行为数据，



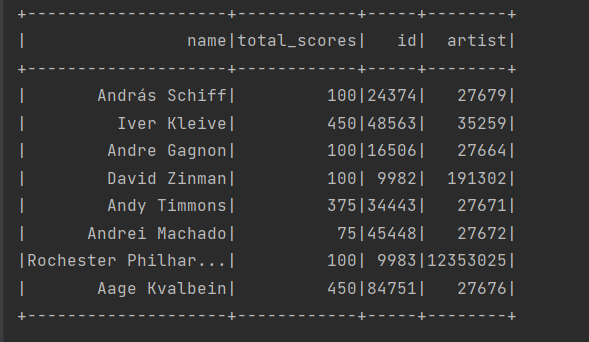
1. 热门歌手推荐：统计每天播放量最高的歌手

从partition获取音乐播放数据



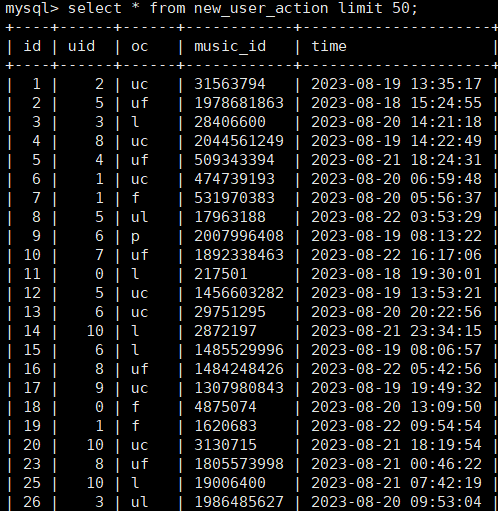
将最新一天的数据进行相加，之后与music\_profile连表后获得歌手信息，再根据歌手id分组进行统计获得分数，提取最高分数的几位歌手

运行代码后获得歌手数据：



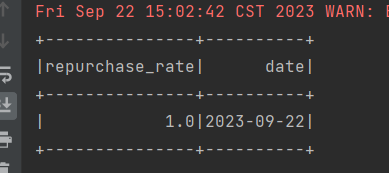
1. vip复购率：统计每个月 购买大于俩次的人数/总购买的人数

读取mysql的new\_user\_action表的数据



1. 筛选出oc字段为"b"且时间在本月内的数据，统计每个月 购买大于俩次的人数/总购买的人数

运行代码后获得：



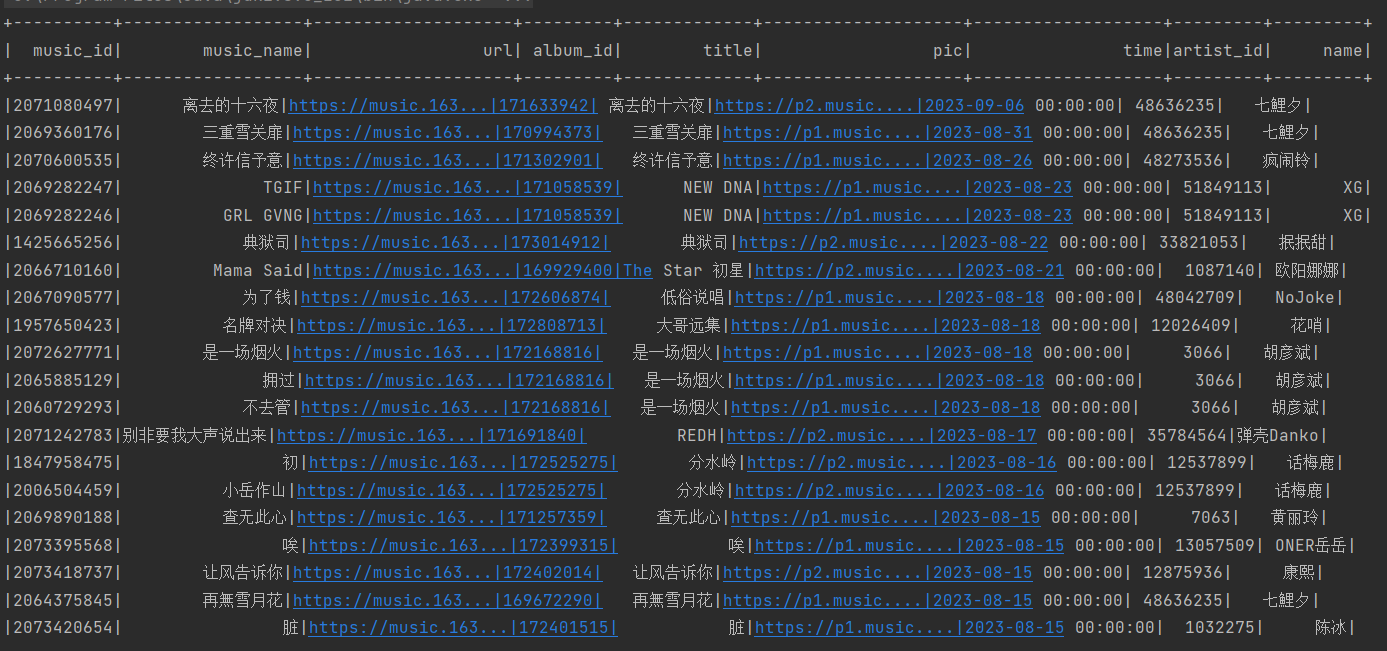
1. 新音乐：最新发布的20首歌曲

读取music\_profile表数据，



将time字段转换为时间戳，选择最大的几首歌曲

运行代码后得到：



## SpringAPI测试

### 测试方案介绍

**测试方式：**手工黑盒测试

手工黑盒测试是一种测试方法，通过模拟最终用户的行为和操作，对系统的功能进行验证和评估。测试人员不需要了解系统的内部实现细节，只需根据需求和规格说明书，针对每个功能点设计测试用例，并执行测试用例来验证系统的行为和结果。

**测试原理：**

1. 功能验证：手工黑盒测试主要用于验证系统的功能是否按照需求和规格说明书进行实现。测试人员根据功能需求，设计合适的测试用例，并执行测试用例来验证系统是否满足功能要求。
2. 边界测试：手工黑盒测试可以帮助发现系统在边界条件下的问题。测试人员可以设计测试用例，包括输入的边界值、边界情况等，验证系统对于边界条件的处理是否正确。
3. 用户体验评估：手工黑盒测试可以通过模拟用户的操作行为，对系统的用户体验进行评估。测试人员可以验证系统的界面友好性、操作便捷性、响应时间等，以提高用户的满意度和体验。

**测试方案步骤：**

1. 确定测试范围和目标：明确要测试的系统模块和功能，确定测试的目标和要求。
2. 创建测试用例：根据需求和规格说明书，设计测试用例。测试用例包括输入数据、操作步骤、预期结果等。确保测试用例覆盖功能的各种情况和可能的边界条件。
3. 执行测试用例：按照设计的测试用例执行测试。模拟用户的操作行为，输入数据，执行操作步骤，并记录实际结果。
4. 验证结果：验证实际结果是否与预期结果一致。根据实际结果与预期结果的差异，判断测试是否通过或失败。
5. 记录问题和缺陷：如发现问题和缺陷，及时记录并报告给开发团队。包括问题的描述、复现步骤、发生的环境等详细信息。
6. 进行回归测试：在修复问题和缺陷后，进行回归测试，确保修复后的系统不会引入新的问题。
7. 更新测试用例：根据测试的结果和发现的问题，及时更新测试用例，以提高测试的覆盖率和效果。

通过手工黑盒测试，可以有效验证系统的功能、边界条件和用户体验，并及时发现和解决问题。该测试方式相对简单易行，适用于各个阶段的测试，可以提高系统的质量和可靠性。但需要测试人员具备一定的领域知识和测试技巧，以保证测试的准确性和有效性。

### 测试用例设计

测试接口：热门音乐

1. 输入正确的请求地址和请求方式（GET），验证返回的状态码是否为200，且返回的数据中包含热门音乐的相关信息。
2. 输入错误的请求地址或请求方式（POST、PUT、DELETE等），验证返回的状态码是否为404或其他合适的错误码。

测试接口：热门艺术家

1. 输入正确的请求地址和请求方式（GET），验证返回的状态码是否为200，且返回的数据中包含热门艺术家的相关信息。
2. 输入错误的请求地址或请求方式（POST、PUT、DELETE等），验证返回的状态码是否为404或其他合适的错误码。

测试接口：个性推荐

1. 输入正确的请求地址和请求方式（GET），同时提供有效的用户id，验证返回的状态码是否为200，且返回的数据中包含个性推荐的相关信息。
2. 输入正确的请求地址和请求方式（GET），但未提供用户id，验证返回的状态码是否为400或其他合适的错误码。

测试接口：搜索

1. 输入正确的请求地址和请求方式（GET），提供有效的搜索类型和关键词，验证返回的状态码是否为200，且返回的数据中包含符合搜索条件的结果。
2. 输入正确的请求地址和请求方式（GET），但未提供搜索类型或关键词，验证返回的状态码是否为400或其他合适的错误码。

测试接口：用户操作

1. 输入正确的请求地址和请求方式（POST），同时提供有效的用户id、音乐id和操作类型，验证返回的状态码是否为200，且返回的数据中包含操作成功的相关信息。
2. 输入正确的请求地址和请求方式（POST），但未提供用户id、音乐id或操作类型，验证返回的状态码是否为400或其他合适的错误码。

### 测试报告

#### 测试概述

测试对象：Spring API

测试类型：黑盒测试

测试日期：2023/9/21

测试人员：张浩

#### 测试环境

操作系统：Windows 11 家庭版

浏览器：Microsoft Edge

工具/框架：Postman

#### 测试用例及结果分析

3.1 热门音乐接口测试

* **用例1：验证获取热门音乐**

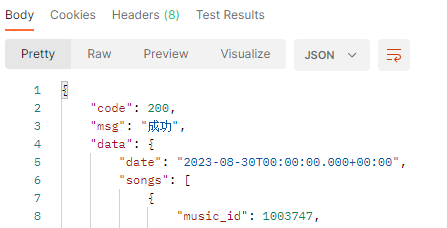
请求地址：/recommend/pop

请求方式：GET

预期结果：返回状态码为200，响应数据中包含热门音乐的相关信息

测试结果：





结论：测试通过

响应时间：418ms

* **用例2：验证请求方式**

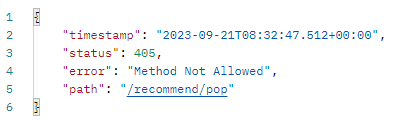
请求地址：/recommend/pop

请求方式：POST

预期结果：返回状态码为400或其他合适的错误码

测试结果：





结论：测试通过

响应时间：98ms

3.2 热门艺术家接口测试

* **用例3：验证获取热门艺术家**

请求地址：/recommend/pop/artist/

请求方式：GET

预期结果：返回状态码为200，响应数据中包含热门艺术家的相关信息

测试结果：





结论：测试通过

响应时间：121ms

* **用例4：验证请求方式**

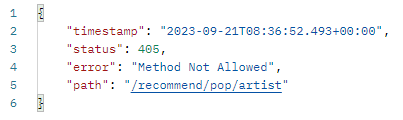
请求地址：/recommend/pop/artist/

请求方式：POST

预期结果：返回状态码为400或其他合适的错误码

测试结果：





结论：测试通过

响应时间：129ms

3.3 个性推荐接口测试

* **用例5：验证获取个性推荐**

请求地址：/recommend/self/

请求方式：GET

请求参数：uid=1

预期结果：返回状态码为200，响应数据中包含个性推荐的相关信息

测试结果：





结论：测试通过

响应时间：240ms

* **用例6：验证未提供用户id**

请求地址：/recommend/self/

请求方式：GET

预期结果：返回状态码为400或其他合适的错误码

测试结果：





结论：测试通过

响应时间：164ms

* **用例7：验证用户id不存在**

请求地址：/recommend/self/

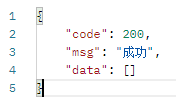
请求方式：GET

请求参数：uid=12345

预期结果：无法查询到个性推荐的相关信息

测试结果：





结论：测试通过

响应时间：84ms

3.4 搜索接口测试

* **用例8：验证音乐搜索**

请求地址：/search/

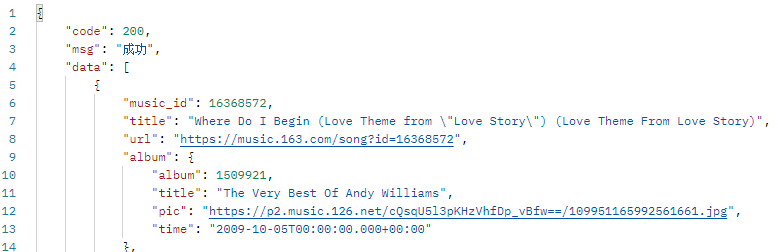
请求方式：GET

请求参数：type=song&kw=love

预期结果：返回状态码为200，响应数据中包含符合搜索条件的音乐结果

测试结果：





结论：测试通过

响应时间：293ms

* **用例9：验证专辑搜索**

请求地址：/search/

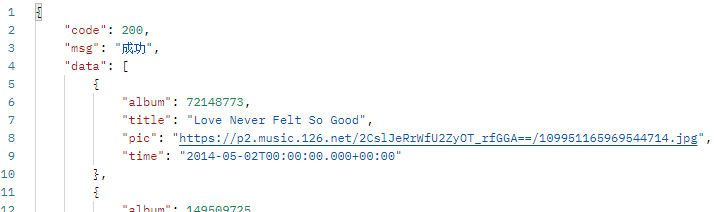
请求方式：GET

请求参数：type=album&kw=love

预期结果：返回状态码为200，响应数据中包含符合搜索条件的专辑结果

测试结果：





结论：测试通过

响应时间：213ms

* **用例10：验证歌手搜索**

请求地址：/search/

请求方式：GET

请求参数：type=artist&kw=love

预期结果：返回状态码为200，响应数据中包含符合搜索条件的艺术家结果

测试结果：





结论：测试通过

响应时间：269ms

* **用例11：验证未提供搜索类型或关键词**

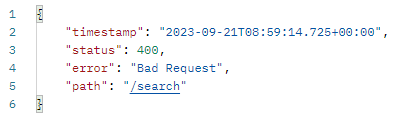
请求地址：/search/

请求方式：GET

预期结果：返回状态码为400或其他合适的错误码

测试结果：





结论：测试通过

响应时间：75ms

* **用例12：验证错误搜索类型**

请求地址：/search/

请求方式：GET

请求参数：type=adult&kw=love

预期结果：返回状态码为400或其他合适的错误码

测试结果：





结论：测试通过

响应时间：297ms

3.5 用户操作接口测试

* **用例13：验证用户操作**

请求地址：/uop/music/

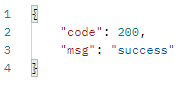
请求方式：POST

请求参数：uid=12345&music\_id=67890&oc=l

预期结果：返回状态码为200，响应数据中包含操作成功的相关信息

测试结果：





结论：测试通过

响应时间：387ms

* **用例14：验证未提供用户id、音乐id或操作类型**

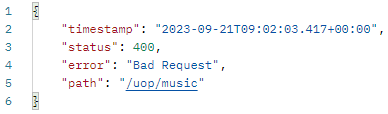
请求地址：/uop/music/

请求方式：POST

预期结果：返回状态码为400或其他合适的错误码

测试结果：





结论：测试通过

响应时间：109ms

* **用例15：验证错误操作类型**

请求地址：/uop/music/

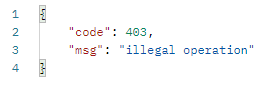
请求方式：POST

请求参数：uid=12345&music\_id=67890&oc=114

预期结果：返回状态码为400或其他合适的错误码

测试结果：





结论：测试通过

响应时间：266ms

#### 测试总结

本轮测试覆盖的模块和接口：

* + 热门音乐
  + 热门艺术家
  + 个性推荐
  + 搜索
  + 用户操作

测试过程中遇到的问题和解决方案：测试通过，功能正常

对整体功能的评价和建议：测试通过，功能正常

## Web前后端测试

**版本: 1.0**

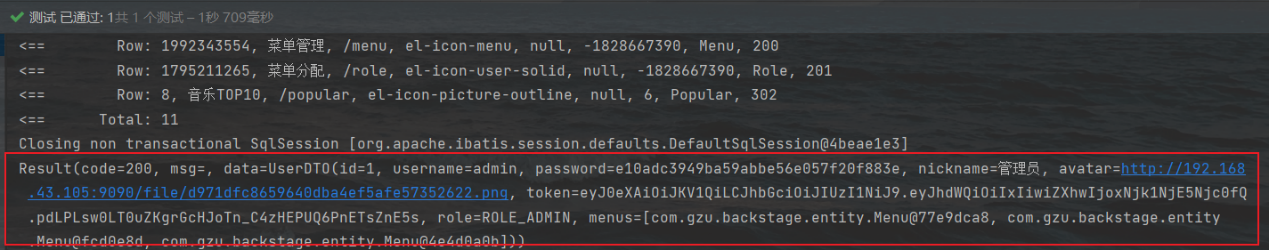
**作者: [苏文杰]**

**发布日期: [2023-9-24]**

8.3.1 查询测试

以用户表为例：使用SpringBootTest测试指定用例数据。

测试结果：

这里得到了一串用户信息、并且生成了一个token、用户可以根据这个token对其他后端控制器方法进行请求。

结论：通过

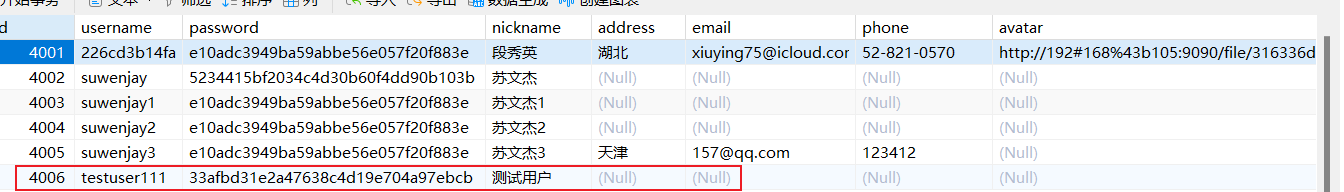
2.增加测试：

我们这里注册一个用户来进行单元测试：

@Test  
public void registerTest(){  
 UserDTO userDTO = new UserDTO();  
 userDTO.setUsername("testuser111");  
 userDTO.setPassword("woshitest111");  
 userDTO.setNickname("测试用户");  
 Result register = userController.register(userDTO);  
 System.*out*.println(register);  
}

控制台日志：

已经新增用户、我们查询数据库表

看到我们新增的用户：

结论：通过

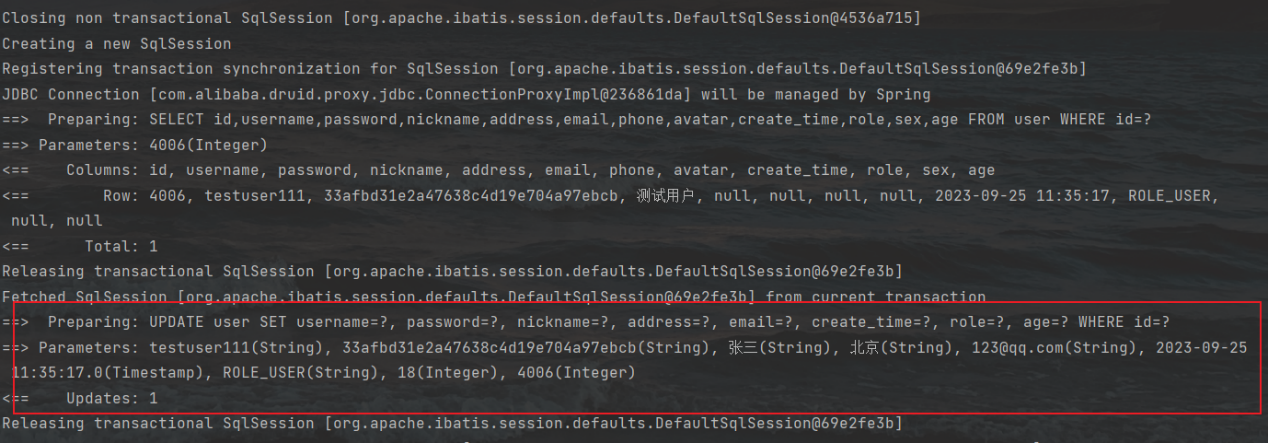
3.修改信息测试：

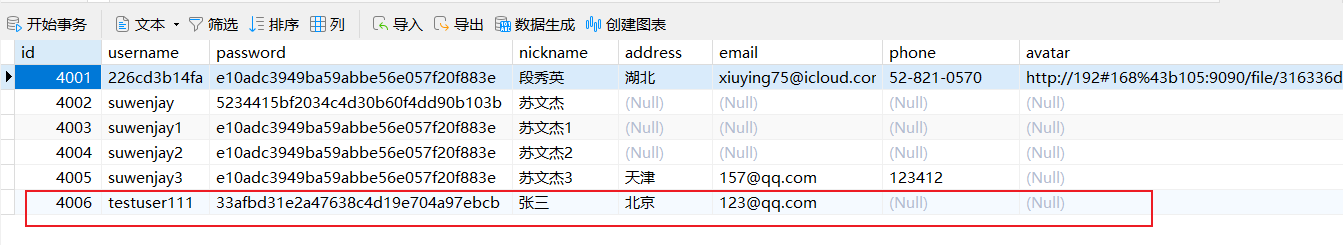
我们现在对该用户进行信息的完善

@Autowired  
IUserService userService;

@Test  
public void updateTest(){  
 User testUser = userService.getById(4006);  
 testUser.setAddress("北京");  
 testUser.setNickname("张三");  
 testUser.setAge(18);  
 testUser.setEmail("123@qq.com");  
 userController.save(testUser);  
}

控制台打印消息：

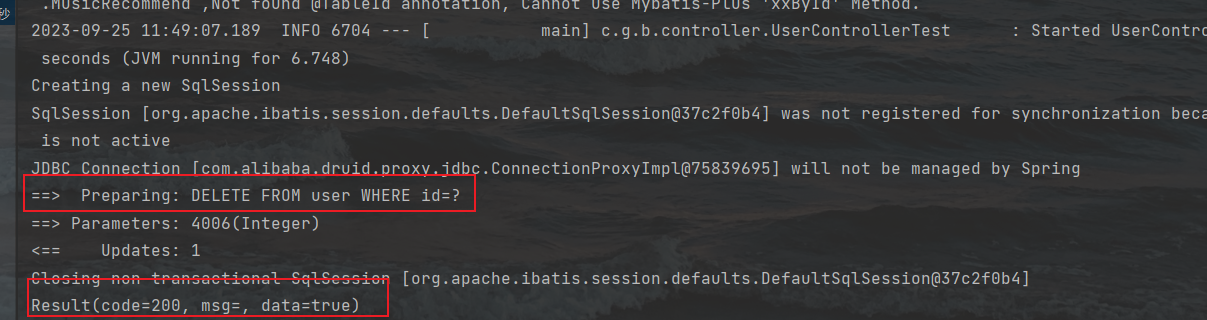
表明我们用户已经修改，我们查看数据库观察信息。

这里已经对用户进行修改了。

结论：通过

4.删除测试

@Test  
public void deleteTest(){  
 Result deleteResult = userController.delete(4006);  
 System.*out*.println(deleteResult);  
}

控制台打印信息  
结论：通过

其他的增删改查接口也是类似。

**9.功能测试：**

9.1 管理员登陆测试：

9.1.1 不输入任何账号和密码

预期：不能登录

结果：不能登录

结论：通过

9.1.2 输入错误账号和密码

预期：不能登录

结果：不能登录

结论：通过

9.1.3 账号输入不符合规则

预期：不能登录

结果：不能登录

结论：通过

9.1.4 输入正确账号密码

预期：登陆成功

结果：登陆成功并返回首页

结论：通过

9.2.1 用户登陆测试：

1.不做任何输入

预期：不能登录

结果：不能登录

结论：通过

2.输入错误的账号密码

预期：不能登录

结果：不能登录

结论：通过

3.输入不合法的长度

预期：不能登录

结果：不能登录

结论：通过

4.输入正确的输入

预期：登陆成功

结果：先提示登陆成功随后右上角的登陆变为用户头像和用户名

结论：通过

9.3 用户注册测试：

9.3.1 不输入任何信息

预期：不能注册

结果：不能注册

结论：通过

9.3.2 两次密码不一样

预期：不能注册

结果：不能注册

结论：通过

9.3.3 输入不符合规范

预期：不能注册

结果：不能注册

结论：通过

9.3.4 输入正确的信息

预期：注册成功

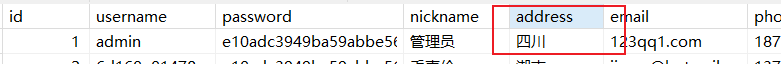
结果：提示注册成功、关闭注册框。

结论：通过

9.4 用户个人信息的修改

修改地址为四川

查看数据库

已经修改

结论：通过

9.5 密码的修改

9.5.1 原密码错误

预期：不能修改

结果：不能修改

结论：通过

9.5.2 新密码为空

预期：不能修改

结果：不能修改

结论：通过

9.5.3 两次输入密码不同

预期：不能修改

结果：不能修改

结论：通过

9.5.4 正确输入

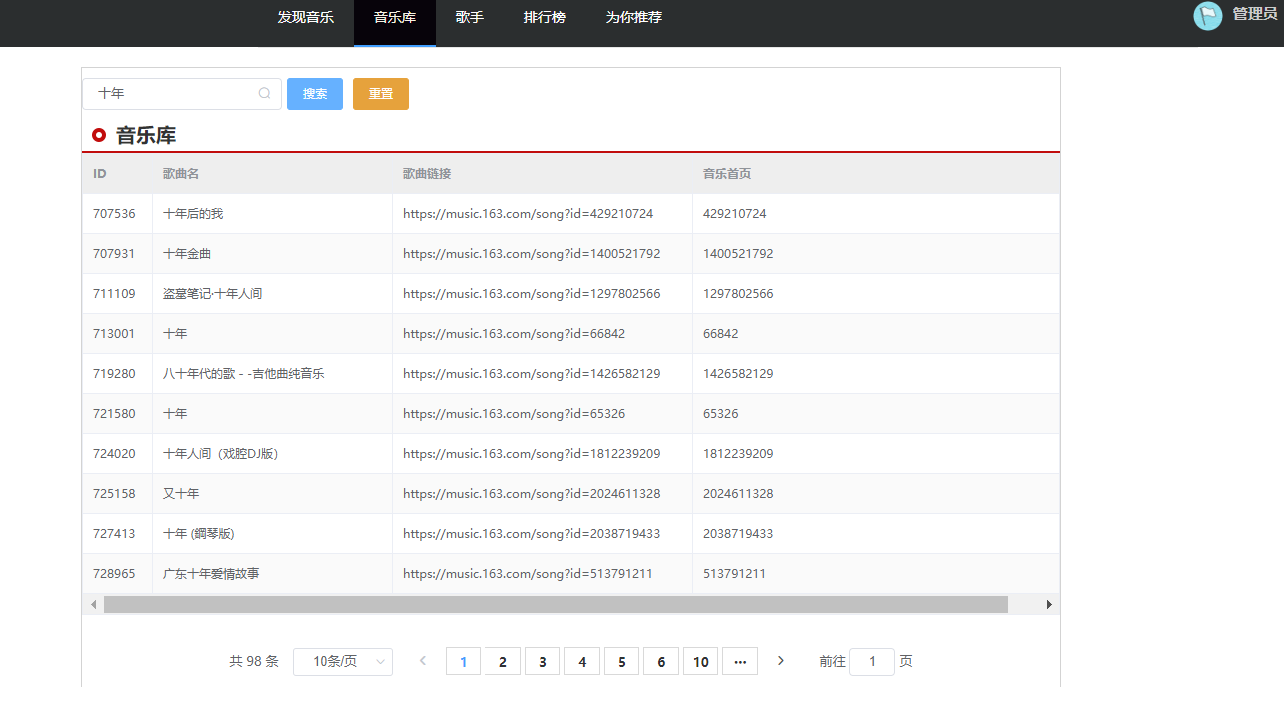
预期：成功修改

结果：成功修改

结论：通过

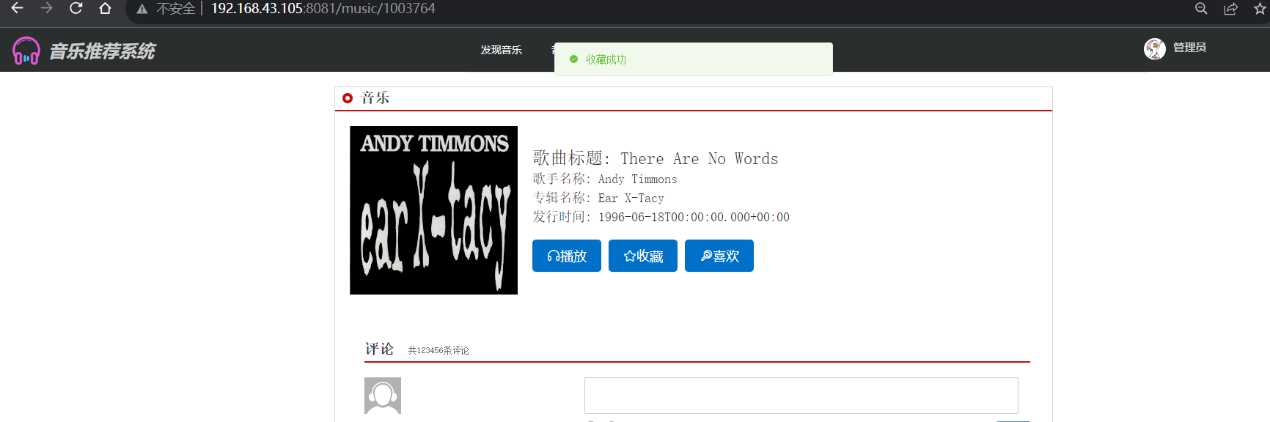
9.6 音乐库测试

关键字：十年

搜索结果：

结论：通过

9.11 用户点击歌曲进入歌曲首页进行操作测试

点击播放，提示播放成功。

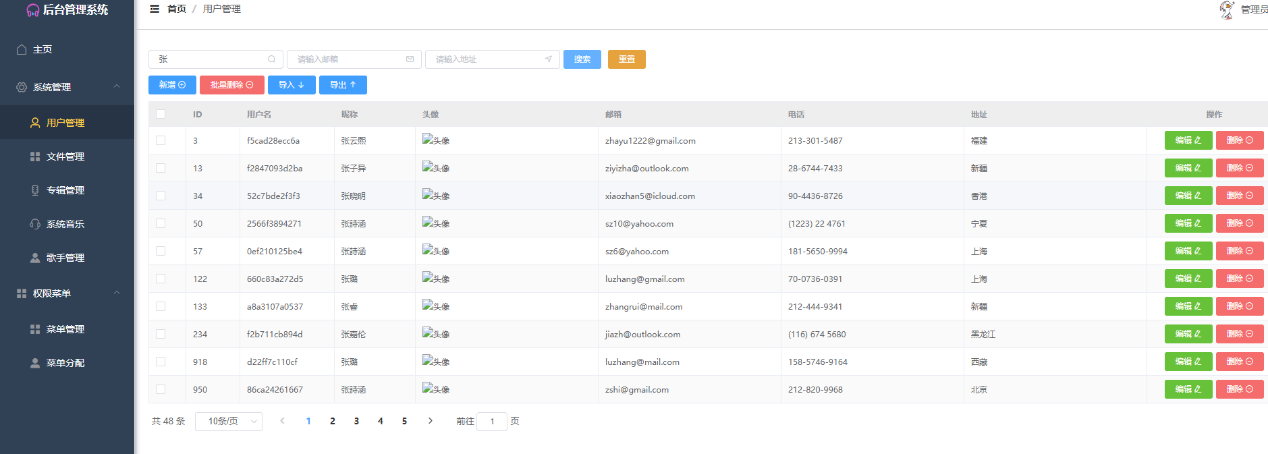
点击收藏提示收藏成功

点击喜欢提示喜欢成功。

结论：通过

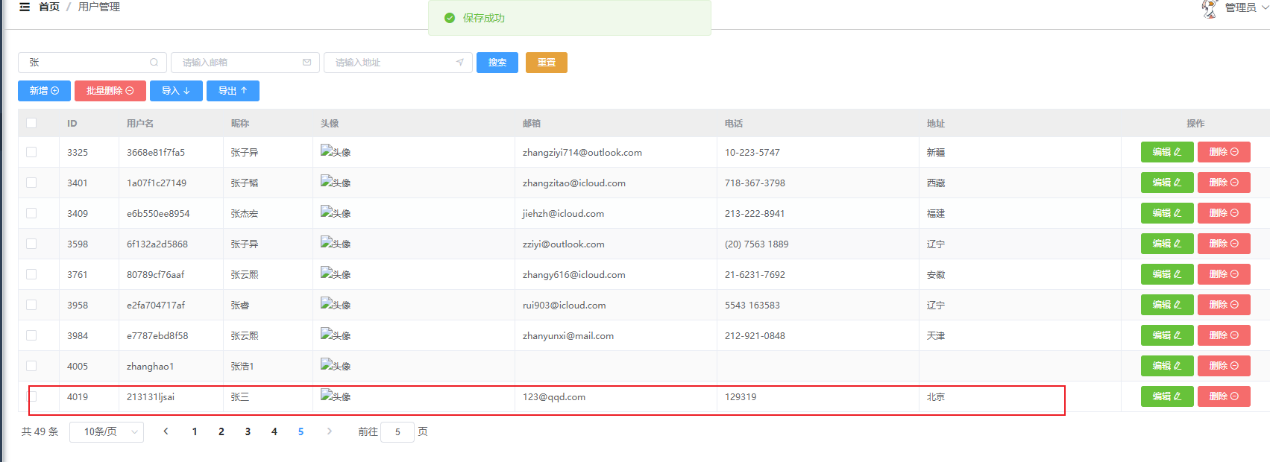
9.12 用户管理测试

9.12.1 搜索测试、搜索昵称为张

9.12.2添加用户测试

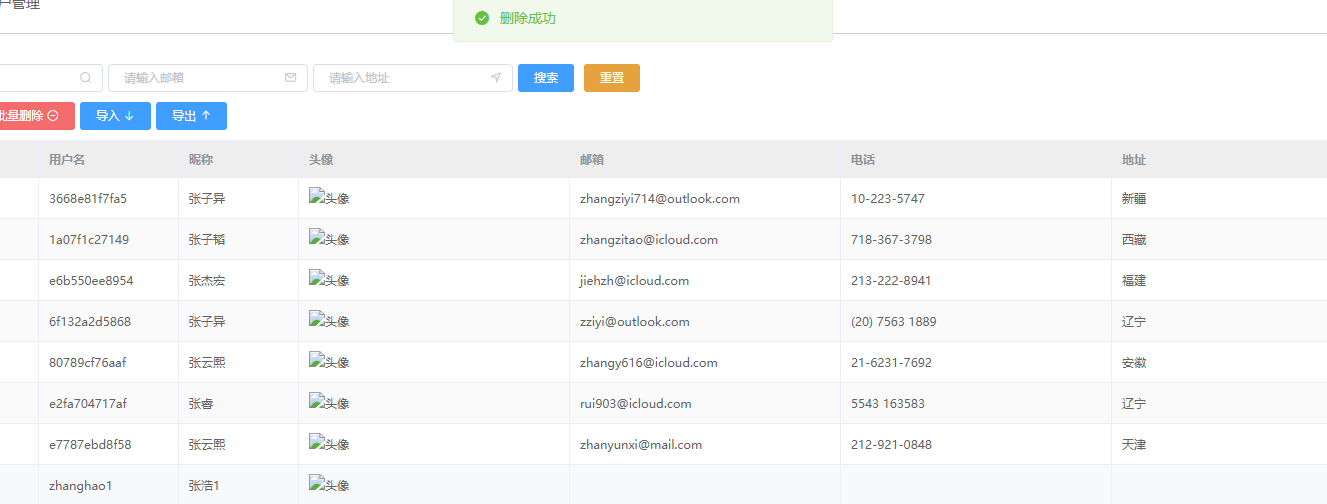


新增加了一条

9.12.3 删除测试：

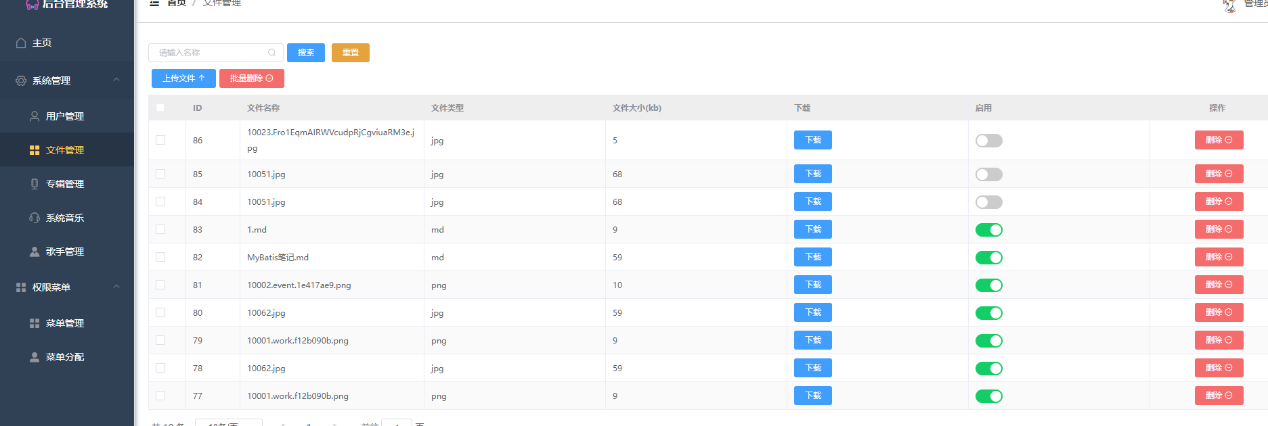


点击确定

用户张三被删除

结论：通过

9.13 文件测试

9.13.1上传文件测试：

9.13.2 删除文件测试：



结论：通过

同时进行了专辑管理测试、系统音乐管理测试、歌手管理测试、 菜单管理测试、菜单分配测试

结论：通过

## 流程测试

一、实时流程一：

FlinkCDC监控MySQL=>Kafka=>DwdPartitionApp=>Clickhouse

1、测试用例数据：

insert into new\_user\_action (uid, oc, music\_id, `time`) values

(21, 'p', '1006281', '2023-09-23T05:12:49'),

(21, 'p', '1006386', '2023-09-23T18:54:49'),

(22, 'p', '1006749', '2023-09-23T11:50:15'),

(22, 'l', '1006753', '2023-09-23T02:52:58'),

(21, 'f', '1006281', '2023-09-23T03:43:32'),

(21, 'p', '1006386', '2023-09-23T22:23:27'),

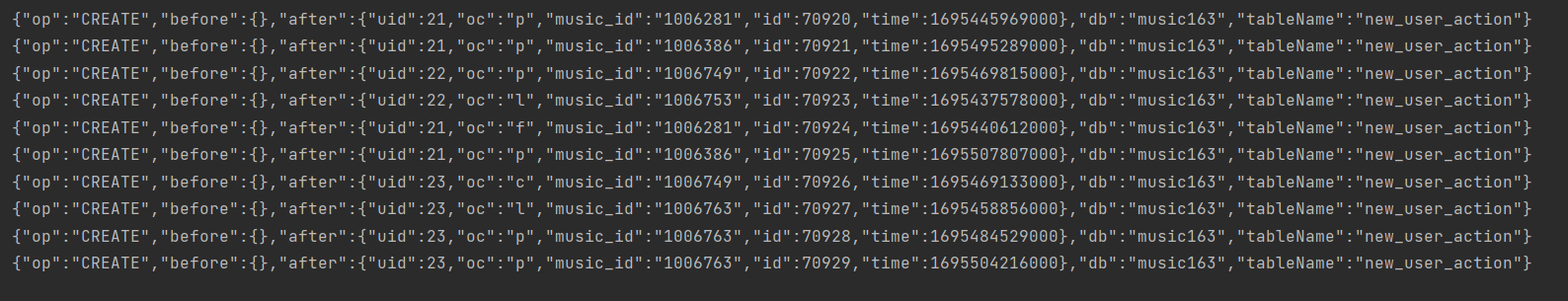
(23, 'c', '1006749', '2023-09-23T11:38:53'),

(23, 'l', '1006763', '2023-09-23T08:47:36'),

(23, 'p', '1006763', '2023-09-23T15:55:29'),

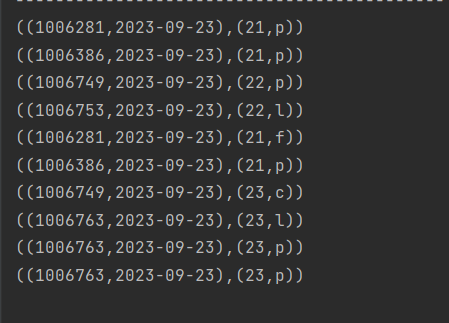
(23, 'p', '1006763', '2023-09-23T21:23:36');

2、FlinkCDC监控数据：

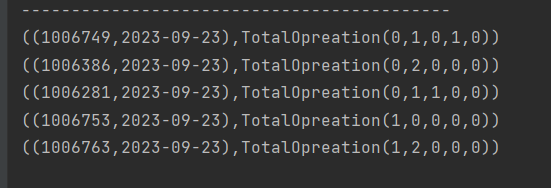


1. DwdPartitionApp处理数据：

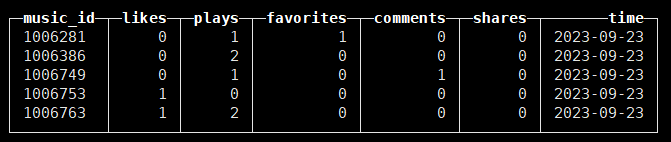
转换格式后的数据：



统计后的结果：



1. 查看Click house中数据与预期是否一致：



二、实时流程二：

FlinkCDC监控MySQL=>Kafka=>DwdPersonalityApp=>Clickhouse

1、测试用例数据：

(21, 'p', '1006281', '2023-09-23T05:12:49'),

(21, 'p', '1006386', '2023-09-23T18:54:49'),

(22, 'p', '1006749', '2023-09-23T11:50:15'),

(22, 'l', '1006753', '2023-09-23T02:52:58'),

(21, 'f', '1006281', '2023-09-23T03:43:32'),

(21, 'p', '1006386', '2023-09-23T22:23:27'),

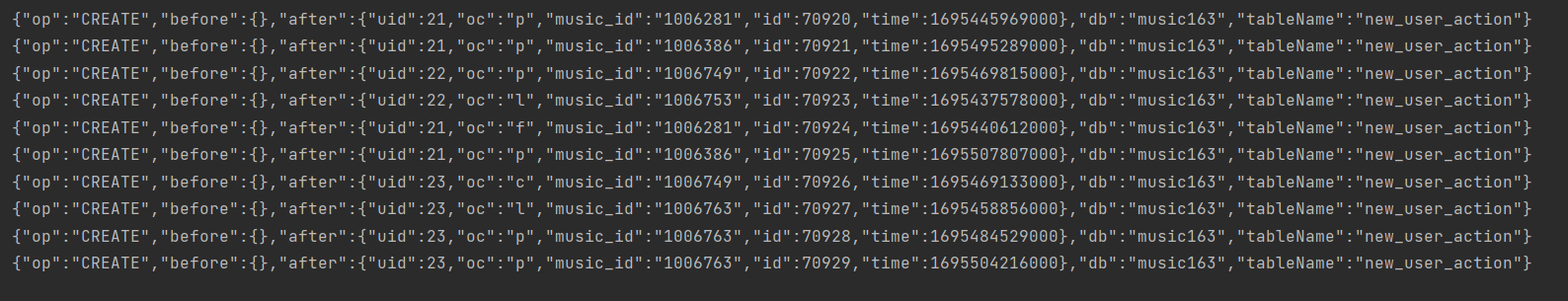
(23, 'c', '1006749', '2023-09-23T11:38:53'),

(23, 'l', '1006763', '2023-09-23T08:47:36'),

(23, 'p', '1006763', '2023-09-23T15:55:29'),

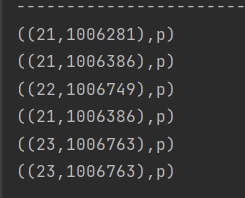
(23, 'p', '1006763', '2023-09-23T21:23:36');

2、FlinkCDC监控数据：

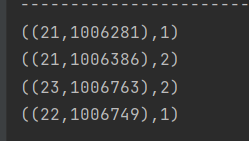


3、DwdPersonality处理数据：

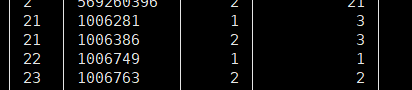
转换格式后的数据:



统计后的结果



4、查看Click house中数据与预期是否一致：



# 总结

该项目旨在通过基于Spark的分布式音乐推荐系统，提供高效准确的音乐推荐服务。我们的团队成员在各自领域有着丰富的经验和技能，通过敏捷开发模式，我们将高效地合作开发各个模块，确保项目按时交付，并不断优化和迭代。我们相信这个项目将为用户提供出色的音乐推荐体验，取得良好的市场反响。