西安交通大學

XI'AN JIAOTONG UNIVERSITY

基于 RBAC 的音乐数据管理平台

音乐数据管理平台

课程名称：	软件系统分析与设计
姓名：	凌晨
学院：	软件学院
专业：	软件工程
学号：	**2214414320**

2023 年 10 月 18 日

目录

一、 目的和要求

请基于权限的用户访问控制（RBAC）机制的原理，并在基于监听器模式和事件响应机制下，在事件处理的机制上实现一个 RBAC 的应用实例。其中要求：

(1) 完整的代码实现；

(2) 页面操作的显示，与角色和权限控制的实现；

(3) 对 RBAC 权限控制机制进行分析（例如对页面、对象、功能按钮之间控制的异同）；

扩展：RBAC 是一个模型簇，可以展开对比分析。

二、 RBAC 理论

1. 理论原因

在没有 RBAC 理论前，通常将用户 (User) 与权限 (Privilege) 绑定在一起，耦合度较高，这导致了对于一些现实问题，比如用户的变更，权限的变更，会带来代码的频繁改动，不利维护项目。

而 RBAC 理论出现后，在 RBAC 中，权限与角色相关联，用户通过成为适当角色的成员而得到这些角色的权限。这就极大地简化了权限的管理。在一个组织中，角色是为了完成各种工作而创造，用户则依据它的责任和资格来被指派相应的角色，用户可以很容易地从一个角色被指派到另一个角色。角色可依新的需求和系统的合并而赋予新的权限，而权限也可根据需要而从某角色中回收。角色与角色的关系可以建立起来以囊括更广泛的客观情况。

RBAC 具有多种好处，其支持三个著名的安全原则：最小权限原则，责任分离原则和数据抽象原则。

(1) 最小权限原则之所以被 RBAC 所支持，是因为 RBAC 可以将其角色配置成其完成任务所需要的最小的权限集。

(2) 责任分离原则可以通过调用相互独立互斥的角色来共同完成敏感的任务而体现，比如要求一个计帐员和财务管理员共参与同一过帐。

(3) 数据抽象可以通过权限的抽象来体现，如财务操作用借款、存款等抽象权限，而不用操作系统提供的典型的读、写、执行权限。然而这些原则必须通过 RBAC 各部件的详细配置才能得以体现。

这带来了诸如低耦合性，安全保障，代码编写规范等等好处。

2. 基本概念

RBAC 认为权限授权实际上是 Who、What、How 的问题。在 RBAC 模型中，who、what、how 构成了访问权限三元组, 也就是“Who 对 What(Which) 进行 How 的操作”。

- Who：权限的拥用者或主体（如 Principal、User、Group、Role、Actor 等等）
- What：权限针对的对象或资源（Resource、Class）。
- How：具体的权限（Privilege, 正向授权与负向授权）。
- Operator：操作。表明对 What 的 How 操作。也就是 Privilege+Resource

- Role：角色，一定数量的权限的集合。权限分配的单位与载体, 目的是隔离 User 与 Privilege 的逻辑关系.
- Group：用户组，权限分配的单位与载体。权限不考虑分配给特定的用户而给组。组可以包括组(以实现权限的继承)，也可以包含用户，组内用户继承组的权限。User 与 Group 是多对多的关系。Group 可以层次化，以满足不同层级权限控制的要求。

RBAC 的关注点在于 Role 和 User, Permission 的关系。称为 User assignment(UA) 和 Permission assignment(PA). 关系的左右两边都是 Many-to-Many 关系。就是 user 可以有多个 role，role 可以包括多个 user。

3. 四种模型

(1) RBAC0

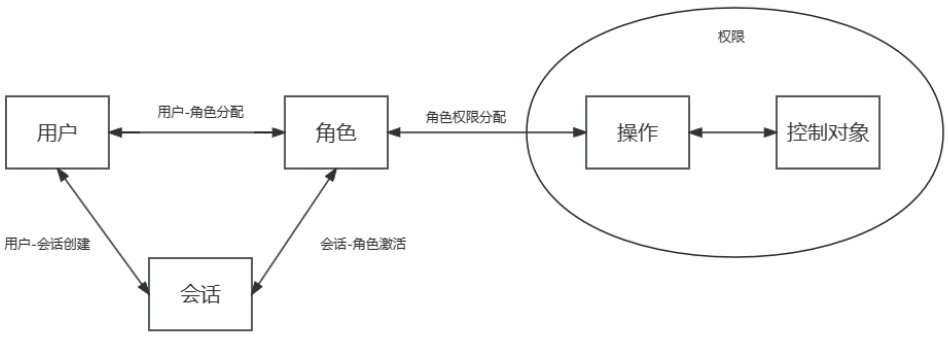
定义：RBAC0 模型由以下描述确定：
U、R、P、S 分别表示用户集合、角色集合、许可权集合和会话集合。

- PA P×R 表示许可权与角色之间多对多的指派关系。
- UA U×R 表示用户与角色之间多对多的指派关系。
- 用户：S→U 每个会话 si 到单个用户 user(si) 的映射函数（常量代表会话的声明周期）。
- 角色：S→2 每个会话 si 到角色子集 roles(si) r|user(si, r') UA（能随时间改变）的映射函数，会话 si 有许可权 Ur roles(si)p|(p,r') PA。

在使用 RBAC0 模型时，应该要求每个许可权和每个用户至少应该被分配给一个角色。两个角色被分配的许可权完全一样是可能的，但仍是两个完全独立的角色，用户也有类似情况。角色可以适当的被看做是一种语义结构，是访问控制策略形式化的基础。

RBAC0 把许可权处理未非解释符号，因为其精确含义只能由实现确定且与系统有关。RBAC0 中的许可权只能应用于数据和资源类客体，但不能应用于模型本身的组件。修改集合 U、R、P 和关系 PA 和 UA 的权限称为管理权限。因此，在 RBAC0 中假定只有安全管理员才能修改这些组件。

会话是由单个用户控制的，在模型中，用户可以创建会话，并有选择的激活用户角色的某些子集。在一个会话中的角色的激活是由用户来决断的，会话的终止也是由用户初始化的。RBAC0 不允许由一个会话去创建另一个会话，会话只能由用户创建。



(2) RBAC1

定义：RBAC1 由以下内容确定

U、R、P、S 分别表示用户集合、角色集合、许可权集合和会话集合。

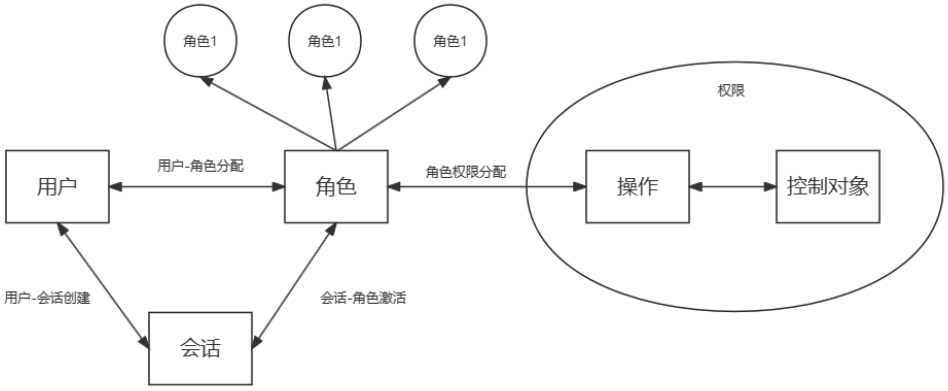
PA P×R 表示许可权与角色之间多对多的指派关系。

UA U×R 表示用户与角色之间多对多的指派关系。

RH R×R 是对 R 的偏序关系，称为角色等级或角色支配关系，也可用 符号表示。

用户：S→U 每个会话 si 到单个用户 user(si) 的映射函数（常量代表会话的声明周期）。

角色：S→2 每个会话 si 到角色子集 roles(si) r|(r' r)[user(si,r') UA]（能随时间改变）的映射函数，会话 si 有许可权 Ur roles(si)p|(r" r)[(p,r") PA]。



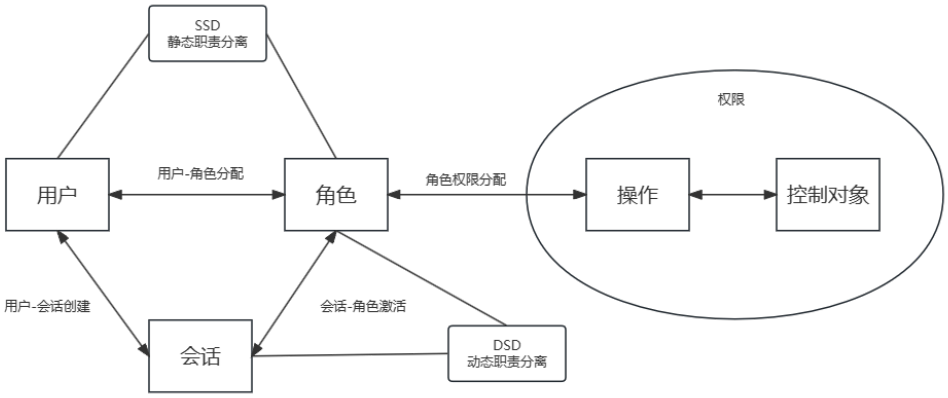
(3) RBAC2

RBAC2，它是 RBAC 的约束模型，RBAC2 也是建立的 RBAC0 的基础之上的，在 RBAC0 基础上假如了约束的概念，主要引入了静态职责分离 SSD (Static Separation of Duty) 和动态职责分离 DSD (Dynamic Separation of Duty)。

SSD 是用户和角色的指派阶段加入的，主要是对用户和角色有如下约束:

- 互斥角色：同一个用户在两个互斥角色中只能选择一个
- 基数约束：一个用户拥有的角色是有限的，一个角色拥有的许可也是有限的
- 先决条件约束：用户想要获得高级角色，首先必须拥有低级角色。

DSD 是会话和角色之间的约束，可以动态的约束用户拥有的角色，如一个用户可以拥有两个角色，但是运行时只能激活一个角色。

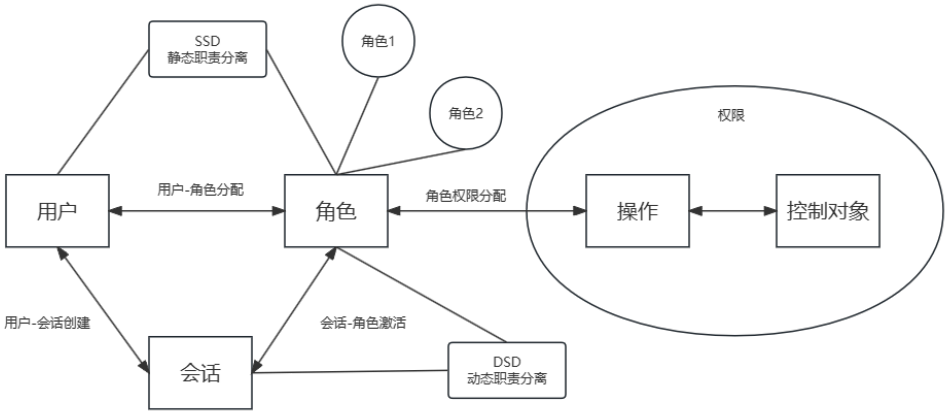


(4) RBAC3

RBAC3 把 RBAC1 和 RBAC2 组合在一起，提供角色的分级和继承的能力。但把这两种概念组合在一起也引起一些新问题。

限制也可以应用于角色等级本身，由于角色间的等级关系满足偏序关系，这种限制对模型而言是本质性的，可能会影响这种偏序关系。例如，附加的限制可能会限制一个给定角色的应有的下级角色的数量。

两个或多个角色由可能被限制成没有公共的上级角色或下级角色。这些类型的限制在概念角色等级的权力已经被分散化的情况下，但是安全主管却希望对所有允许这些改变的方法加以限制。



三、 项目概要

1. 项目背景

随着数字音乐市场的迅速发展，音乐平台成为人们在线获取音乐的主要途径之一。这些平台承载着大量的音乐文件、歌手信息、专辑信息和用户数据等，需要一个高效、可靠的数据管理系统来支持其运营和用户体验。传统的音乐平台数据管理方式可能存在一些问题，如数据存储不规范、数据检索效率低下、用户个性化推荐不足等。此外，随着用户数量和数据量的增加，传统的数据管理方法可能无法满足平台的需求。因此，开发一个专门的音乐平台数据管理项目，能够解决这些问题，提升音乐平台的数据管理能力和用户体验。

2. 项目需求分析

需要注意的时，在进行需求分析，主要开展的是针对该系统的一些特定功能的分析。一般功能不再给出，例如登录，注册，修改个人信息等等。

针对使用者（用户），可以做出如下需求分析：

- I 级用户：即游客，只能查找音乐
- II 级用户：即普通用户，能查找音乐，播放音乐
- III 级用户：即会员，可以查找音乐，播放音乐，下载音乐

针对提供数据的提供者，可以做出如下需求分析：

- I 级提供者：即游客，只能查找音乐
- II 级提供者：即普通提供者，能查找音乐，只能提供相关数据
- III 级提供者：即高级提供者，可以查找音乐，提供数据，更改自己提供的数据

针对审核数据的审核者，可以做出如下需求分析：

- I 级审核者：即游客，只能查找音乐
- II 级审核者：即普通员工，能查找音乐，审核系统分配的数据
- III 级审核者：即高级审核员，可以查找音乐，审核所有数据，但不能修改数据，可以对数据打上标签

针对管理数据的管理员，可以做出如下需求分析：

- I 级管理员：即游客，只能查找音乐
- II 级管理员：即普通管理员，能查找音乐，增加删除所有音乐数据
- III 级管理者：即高级管理员，能查找音乐，增加删除所有数据

我们注意到了以下几件事情：

(1) 部分角色的功能有重复

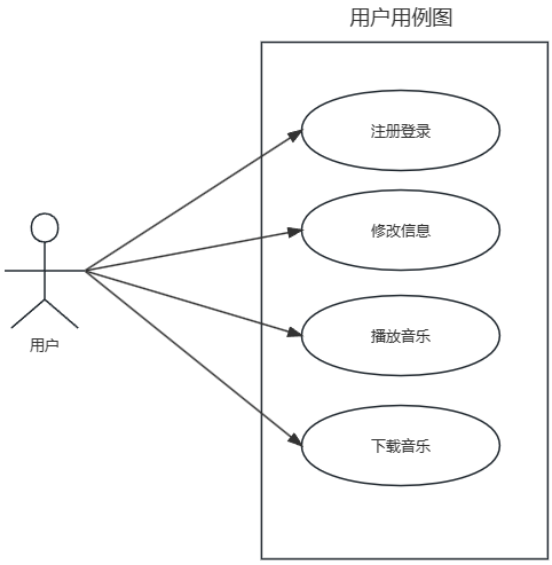
(2) 角色之间存在分级和继承

(3) 角色之间均有游客这一上级角色

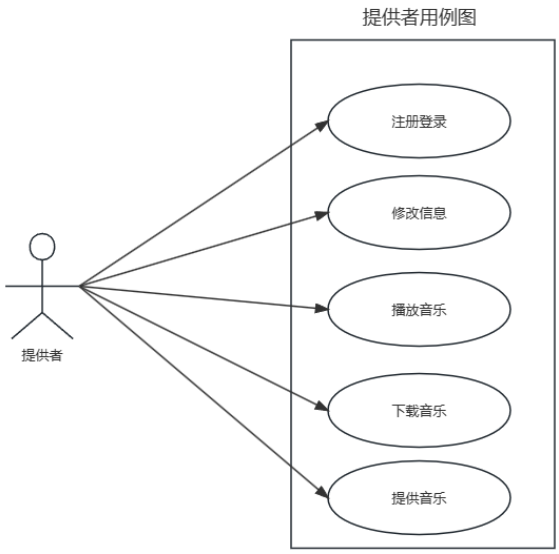
因此，该系统采用 RBAC3 设计。

3. 角色用例图

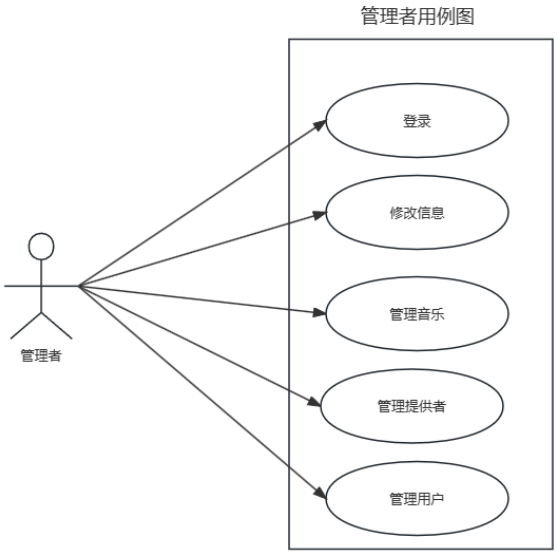
由于角色存在功能的重叠，而且一一展示没有过多的必要，主要是为了展现系统中的部分角色功能。



(a) 用户



(b) 提供者



(c) 管理者

图 1: 用例图

4. 角色结构图

为了压缩篇幅，只给出管理员的角色结构图：

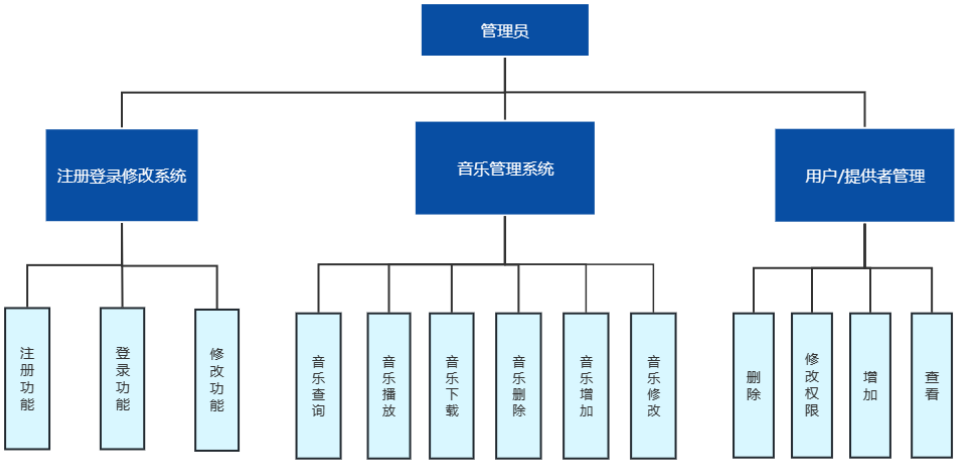


图 2: 管理员结构图

四、 数据库设计

在进行代码实现之前，现要求进行数据库的设计。

1. PowerDesigner

PowerDesigner 是一款功能强大的建模工具和数据管理软件，由 SAP 公司开发。它提供了全面的功能来支持企业架构、数据架构和业务流程的建模、设计和管理。

在 PowerDesigner 软件中，可以快速的建立概念模型和物理模型，可以通过概念模型转化为物理模型，反之亦然。

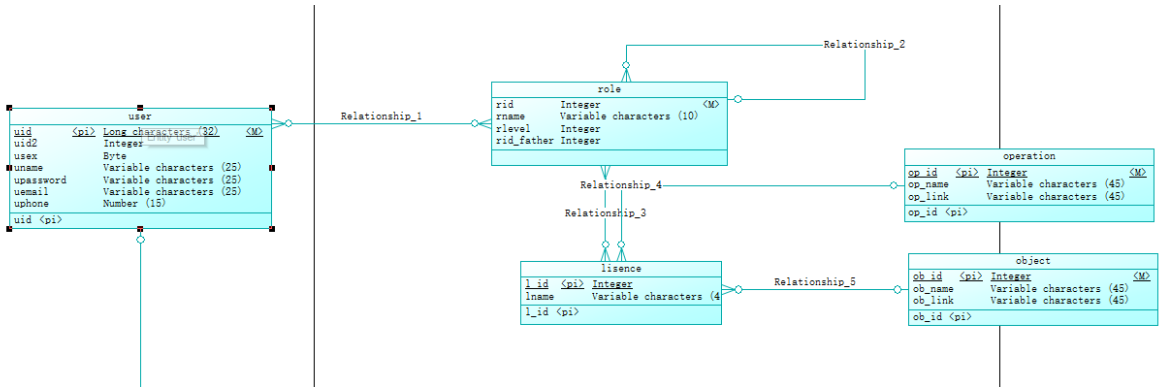
概念模型是数据库设计的高层抽象，它主要关注业务需求和概念结构，而不涉及具体的技术实现。概念模型使用实体-关系图（ER 图）来表示数据实体之间的关系。在概念模型中，你可以定义实体（Entity）、属性（Attribute）、关系（Relationship）等，以及它们之间的约束和规则。概念模型的目标是捕捉业务需求和数据结构之间的关系，为后续的物理设计提供指导和基础。

物理模型是在概念模型的基础上进行的具体实现，它考虑了底层数据库管理系统（如 Oracle、SQL Server 等）的特性和限制。物理模型定义了数据库中的具体表、列、索引、主键、外键等对象，并指定了它们之间的关系和约束。物理模型中使用的表示方法可能会依赖于具体的数据库管理系统，例如，PowerDesigner 提供了针对不同数据库的物理模型模板，可以生成特定数据库的 DDL（数据定义语言）脚本。

因此，在接下来的数据库功能讲解中，我主要以 ER 图为例，来讲解设计数据库的理由。

2. RBAC3 的数据库建立

ER 图如下：

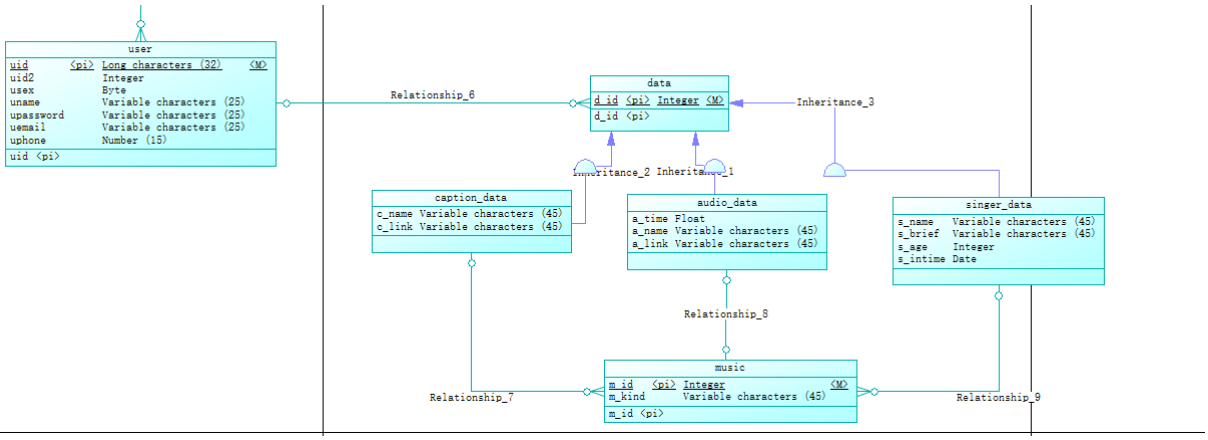


首先，该设计满足了 RBAC 中的基本属性，用户和角色进行了分离，实现了低耦合。然后，角色和许可进行了分离，日后可以通过权限关联表一目了然。同时许可表是在操作和对象的基础上得到的，实现了解耦。

其次，该设计还满足了 RBAC3 中的提供角色的分级和继承的能力，其中分级功能主要是通过 role 表中的 rlevel 属性进行定义。而继承功能主要是通过 role 与 role 表一对多关系进行实现的。

3. 业务数据库

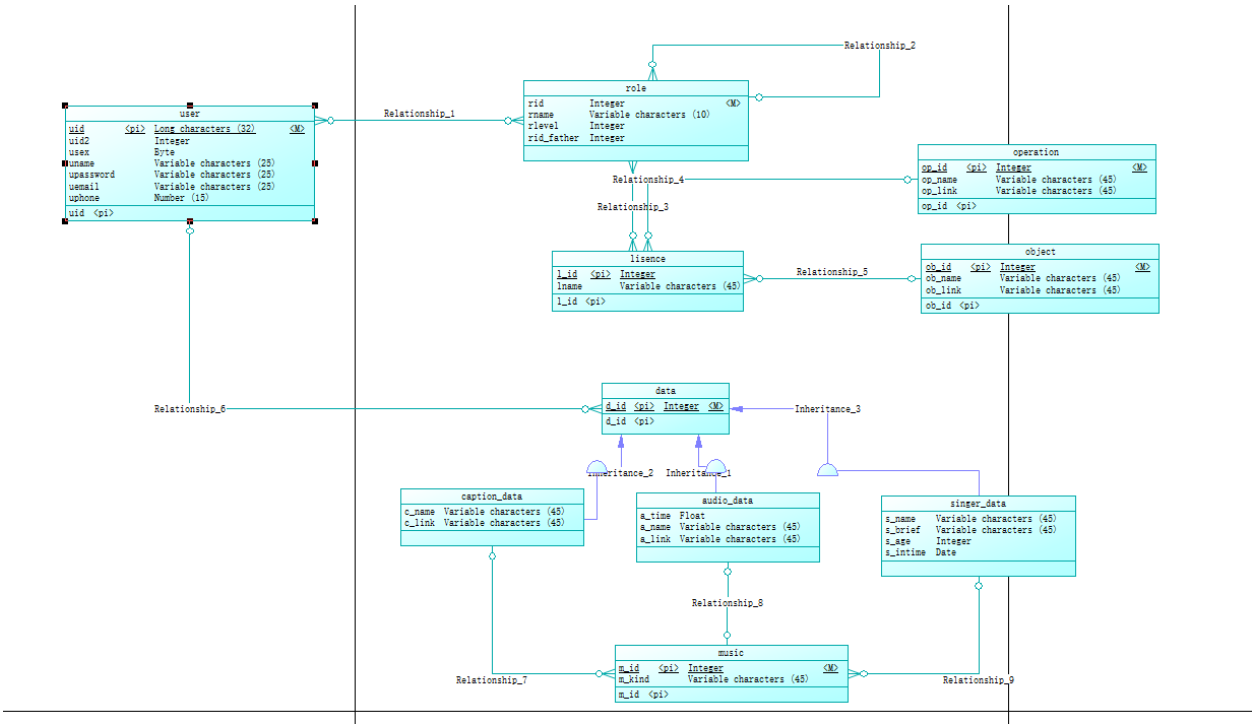
ER 图如下：



该数据库主要是为了满足业务的要求而建立的数据库。主要是对于不同数据的存储，通过继承机制来进行存储。同时，我们将歌曲，音频，字幕，歌手等资源进行了关联处理，关系如图所示，不再赘述。

4. 概念数据库设计

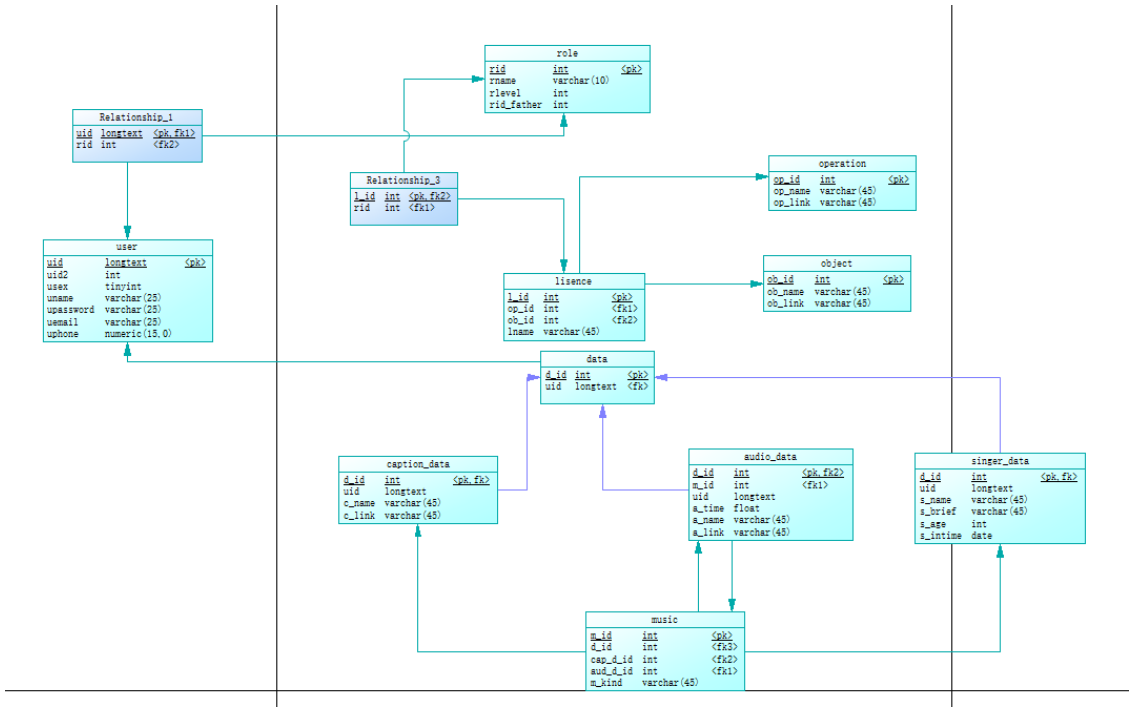
总的概念数据库设计如下：



注意：使用的是 ER+Merise 概念模型

5. 物理数据库设计

总的物理数据库设计如下：



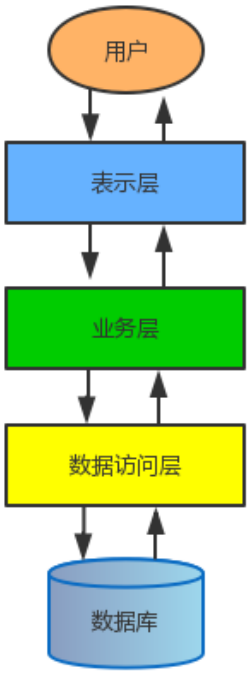
五、 技术栈简述

为了实现 RBAC3，使用的技术栈为发送 http 请求：Apipost，后端 Spring Boot, 数据库：MySQL。整体采取三端分离的思想，我们主要关注的点为**后端代码的编写和数据库的实现**。

为了有基本概念，下面对重点设计思想与技术进行简要介绍：

1. MVC 设计思想

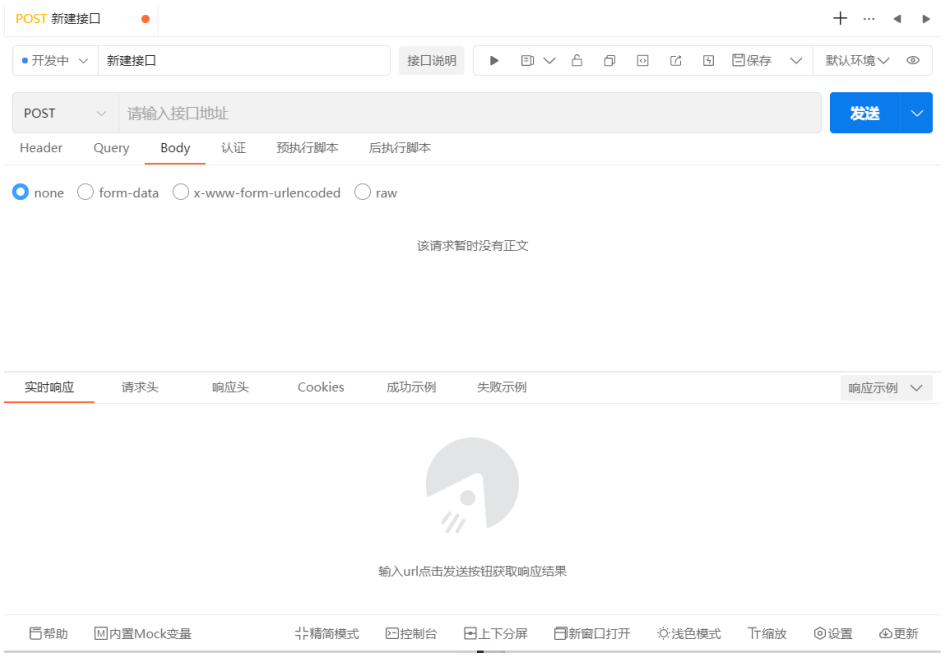
MVC 的全名是 Model View Controller，是模型 (Model) －视图 (View) －控制器 (Controller) 的缩写，是一种设计模式。它是用一种业务逻辑、数据与界面显示分离的方法来组织代码，将众多的业务逻辑聚集到一个部件里面，在需要改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑，达到减少编码的时间，提高代码复用性。



2. Apipost

Apipost 作为 API 研发一体化赋能平台，解决了阿里巴巴在 API 设计、调试、自动化测试等问题，使得企业 API 的版本迭代管理、可拓展性、稳定性、安全性等问题上获得有效解决方案。

下面是该软件的使用界面：



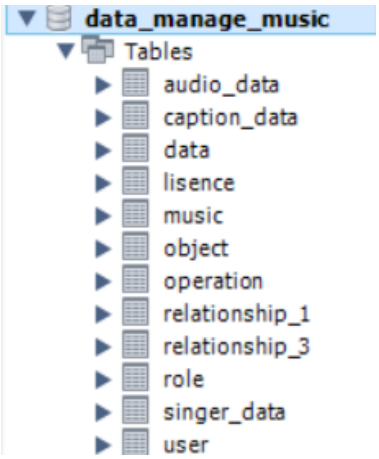
3. Spring Boot

Spring Boot 是一个用于快速构建独立、可部署的 Java 应用程序的开源框架。它是基于 Spring 框架的，但是简化了 Spring 的配置和部署过程，使开发者能够更加专注于业务逻辑的实现。

Spring Boot 提供了许多开箱即用的功能，使得开发者能够快速搭建和运行应用程序，能够简化 Java 应用程序的开发和部署过程，提高开发效率。它广泛应用于各种类型的应用程序开发，包括 Web 应用程序、RESTful API、批处理任务等。

六、 数据库搭建

使用 PowerDesigner 进行 SQL 语句生成，生成的 SQL 语句直接导入 MYSQL 软件中，生成数据库，生成的数据库结构如下：



可以看到，是符合我们在物理数据设计的，表是对应生成的，对于表中的属性不再一一放图片出来

参照了，详情可以查看附件！

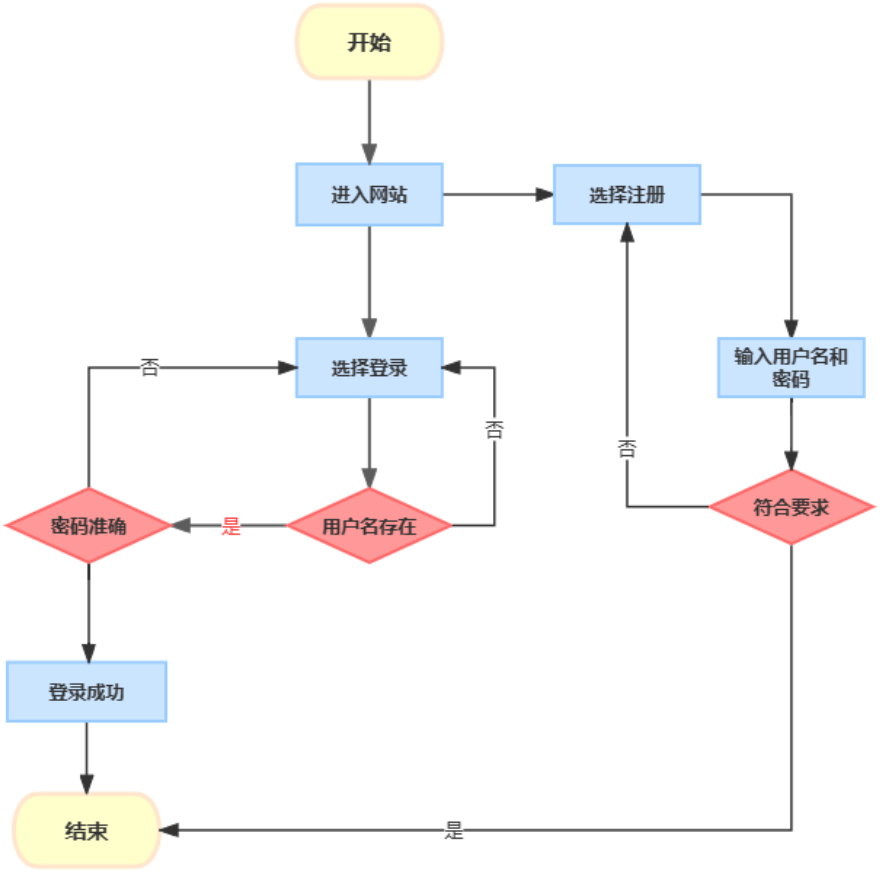
七、 功能设计

处于时间的原因，没有办法完美实现各个功能，我只实现了数据管理中的部分功能，希望老师，助教谅解！

在给出代码前，先对实现的功能进行流程分析，在流程的设计中，可以体现出 RBAC 的设计思想。

1. 注册/登录功能流程图

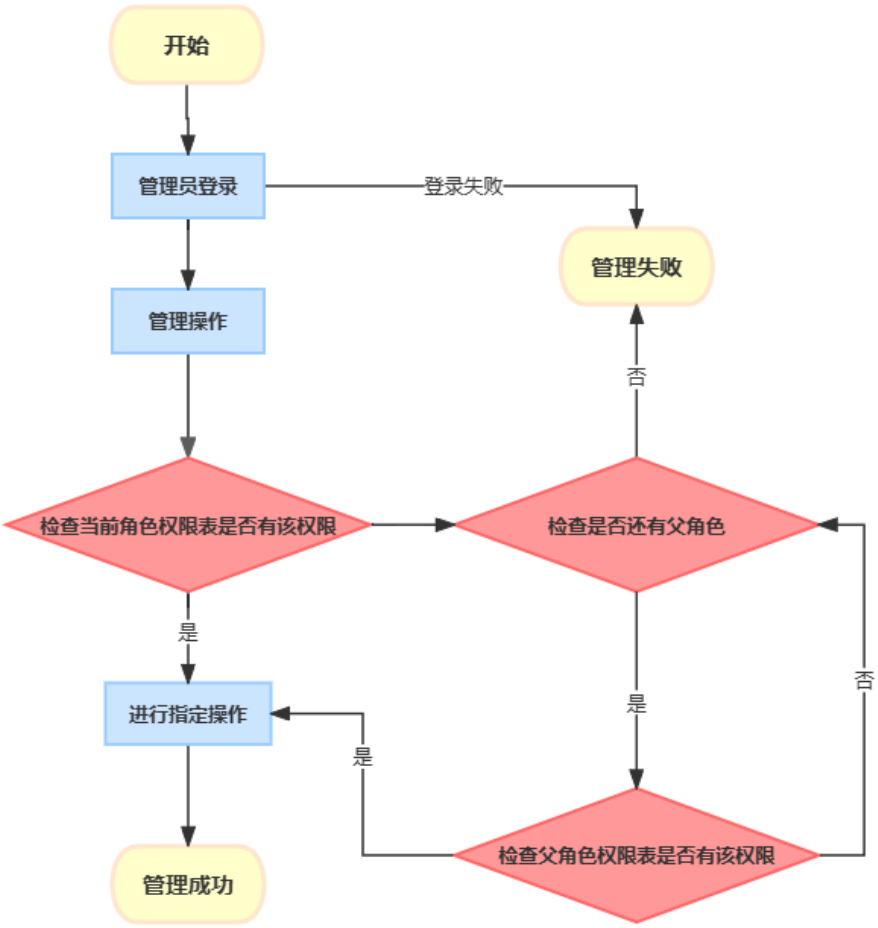
下面是注册/登录功能流程图：



可以看到，对于登录/注册没有体现许可权限，因为这是一项基本功能。

2. 管理数据功能流程图

下面是管理功能流程图：



这里的流程图充分体现了 RBAC3 的设计思想：

(1) 在管理数据前，需要确定用户角色。

(2) 在进行管理时，通过角色权限表，判断是否有权限进行更改。

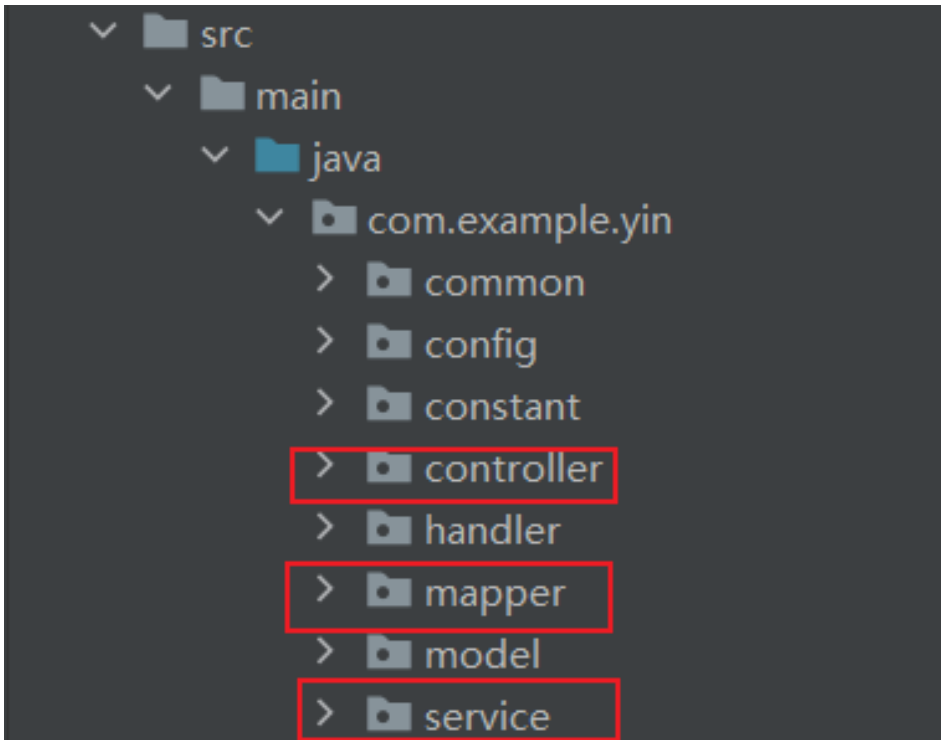
(3) 若无权限，对用户的其他角色或者该角色的父角色进行递归查找权限。

(4) 若递归查找权限时，发现有权限，许可操作。若递归完毕，仍无权限，则终止操作。

该流程图还有结构清晰，实现方便，减少数据冗余等特点，不在一一说明。

八、 代码实现及展示

需要在前面提前说明，框架为 Spring Boot。后端组成如下：



其中最关键的代码实现的三个部分用红框标出来了

- controller: 控制器，就是 MVC 中的 controller，不再赘述。
- mapper: 连接数据库，对数据库进行操作。
- service: 为 controller 和 mapper 的中间层，可以理解为 controller 对 mapper 的映射。

当然，还有一些部分也很重要，给出解释：

- model: 里面存放的为对于数据库的实体属性。
- config: 实现框架必不可少的配置。
- constant：存放着各种全局变量或者全局常量，相当于公共缓存区。

值得一提的是，实现详细功能，我只给出关键代码，因为给全部代码的话。
一、部分代码重复，代码冗长，不利于查看。
二、由于完成时间仓促，代码的完成度比较低，bug 数还是比较多的。所以，代码主要是说明可行性！

1. 登录功能

(1) 代码实现

controller 层代码：

```
@RestController
public class AdminController {
```

```
    @Autowired
    private AdminService adminService;

    // 判断是否登录成功
    @PostMapping("/admin/login/")
    public R loginStatus(@RequestBody AdminRequest adminRequest, HttpSession session) {
        return adminService.verityPasswd(adminRequest, session);
    }
}
```

service 层代码：

```
public interface AdminService extends IService<Admin> {
    // 返回验证结果
    R verityPasswd(AdminRequest adminRequest, HttpSession session);
}

// 下面为实现类
@Service
public class AdminServiceImpl extends ServiceImpl<AdminMapper, Admin> implements
     AdminService {

    @Autowired
    private AdminMapper adminMapper;

    @Override
    public R verityPasswd(AdminRequest adminRequest, HttpSession session) {
        QueryWrapper<Admin> queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("name",adminRequest.getUsername());
        queryWrapper.eq("password",adminRequest.getPassword());
        if (adminMapper.selectCount(queryWrapper) > 0) {
            session.setAttribute("name", adminRequest.getUsername());
            return R.success("登录成功");
        } else {
            return R.error("用户名或密码错误");
        }
    }
}
```

mapper 层代码：

```
@Repository
public interface AdminMapper extends BaseMapper<Admin> {
// 注：BaseMapper为该com.baomidou.mybatisplus.core.mapper包中的类
// 感兴趣可以查看官网的解释!
}
```

至此，登录功能实现完成！

(2)　代码部分讲解

关于三层架构及具体作用不再赘述，下面主要讲解部分可能导致疑惑的函数，类，库。

1、BaseMapper 的作用：BaseMapper 是已经编写好的库，程序员只需要在配置文件中配置对应的版本。

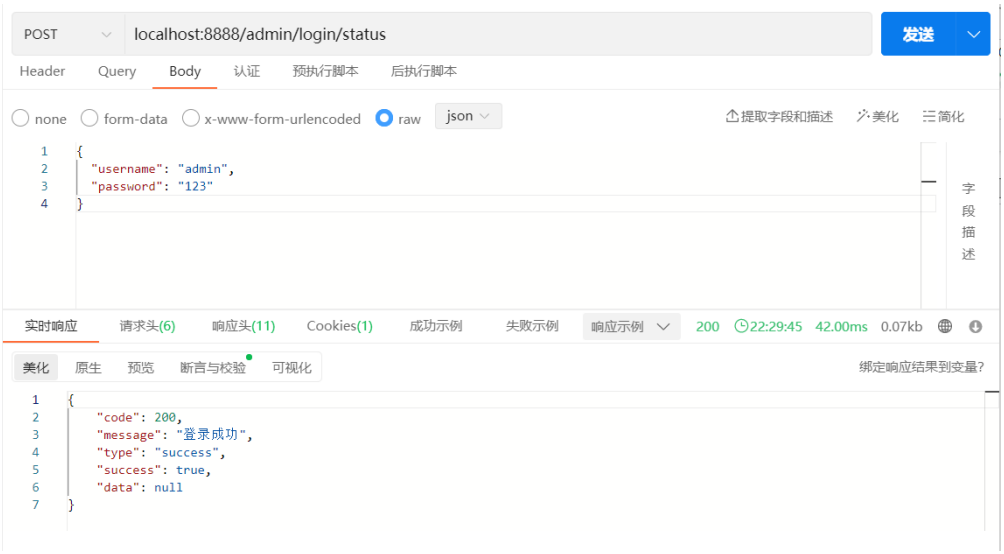
使用步骤如下：将与数据库对应的实体类注入后，BaseMapper 提供了多种增删改查的方法，因此不需要程序员写 SQL 语句对数据库进行操作了。

2、R 类的作用，主要是生成 json 字符串格式，用来返回信息。详情可见附录，注：该类主要为网上借鉴而来，加入了部分自己的理解！

(3)　验证

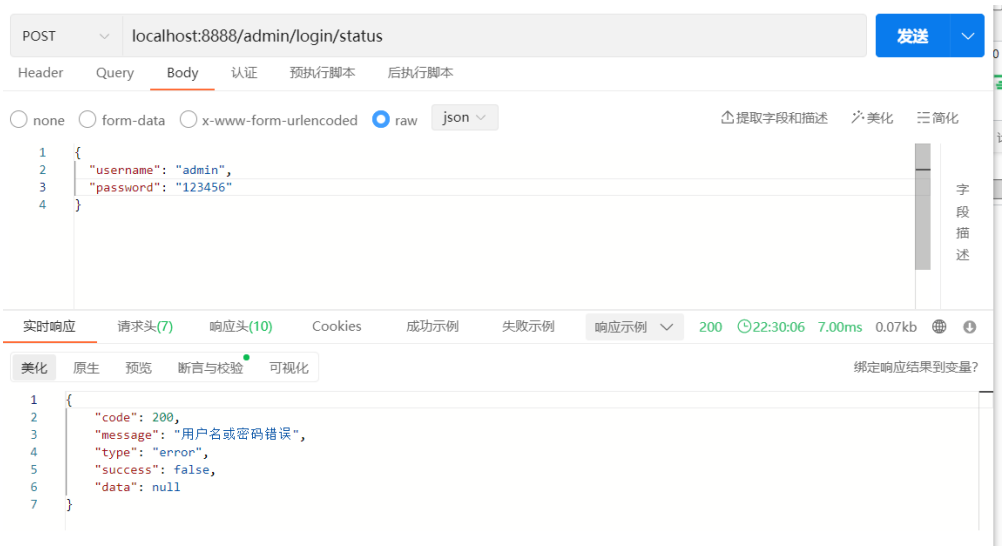
我们将服务器运行至本地 8888 端口，然后使用 Apipost 软件进行验证：

当登录已存在的用户，账号密码均对时：



可以在底下看到返回 200，登录成功！

当登录已存在的用户，密码错误时：



可以在底下看到返回 200，但登录失败！

当登录已存在的用户，用户名存在时：

可以在底下看到返回 200，但登录失败！

2. 增删改查功能

由于增删改查的代码编写逻辑几乎一致，下面只给出**删除功能实现**的代码

(1) 代码实现

controller 层代码：

```
@RestController
public class DeleteController {
    @Autowired
```

```
    private DeleteService deleteService;

    // 判断是否登录成功
    @DeleteMapping("/user/delete")
    public R deleteStatus(@RequestBody DeleteRequest deleteRequest, HttpSession session) {
        return deleteDataResult(deleteRequest, session);
    }
}
```

service 层代码：

```
public interface DeleteService extends IService<DeleteRequest> {
    // 返回删除结果
    R deleteDataResult(DeleteRequest deleteRequest, HttpSession session);
}

// 下面为实现类
@Service
public class DeleteServiceImpl extends ServiceImpl<DeleteMapper, Data> implements
     AdminService {

    @Autowired
    private DeleteMapper deleteMapper;

    @Override
    public R deleteDataResult(DeleteRequest deleteRequest, HttpSession session) {
        QueryWrapper<Data> queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("userId",deleteRequest.getUserId()); // 获得用户的角色
        queryWrapper.eq("objectId",adminRequest.getObjectId()); // 获得用户的想要删除的对象
        int privilege = CheckPrivilege.getPrivilegeByRole(userId,DELETE,objectId); //
            根据角色获取权限
        if (privilrge == PASS) {
            if (deleteMapper.deleteById(objectId) > 0){
                return R.success("删除成功");
            }
            else{
                return R.error("不存在对象，无法删除");
            }
        } else {
            return R.error("权限不够");
        }
    }
}
```

mapper 层代码：

```
@Repository
public interface DeleteMapper extends BaseMapper<Data> {
// 注：BaseMapper为该com.baomidou.mybatisplus.core.mapper包中的类
// 感兴趣可以查看官网的解释!
```

```
}
```

至此，删除功能实现完成！

(2) 代码讲解

对于基本的结构，http 请求和登录功能的写法一致，因此不再赘述，重点讲解如何实现 RBAC3. 关键代码是 service 层的代码：

(1) 首先，获得了用户的角色。

(2) 然后，利用 CheckPrivilege 工具类获得角色是否有对应的权限。

(3) 最后，根据判断执行操作，对应数据库。

下面简要讲解 CheckPrivilege 工具类的设计思想：

- 根据 UserId 中用户对应角色，对查找到的每个角色进行下面步骤。
- 在许可表中查找角色有无对应权限，若有返回 PASS。
- 若无，查看角色是否为最低等级角色，若不是，递归查找。
- 若是最低等级角色，退出该角色。
- 若全部查找完，仍无权限，返回 UNPASS。

至此，实现了 RBAC3。

(3) 测试数据

由于情况过多，只进行成功删除的验证。在验证之前，给出角色-权限表。

	查看音乐数据	下载音乐数据	删除音乐数据	删除所有数据
I 级管理员	Y			
II 级管理员		Y	Y	
III 级管理员				Y

表 1: 角色-权限表

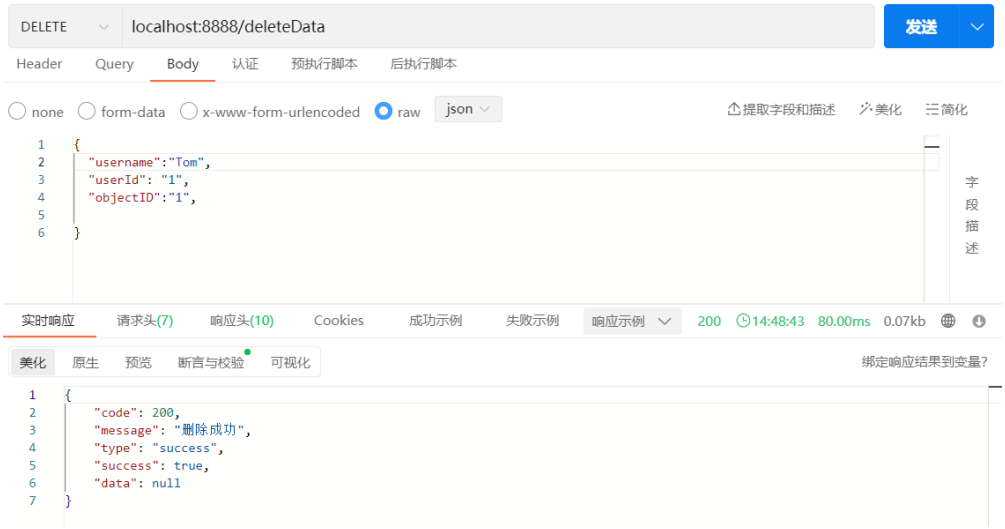
这样子设计许可表可以减少数据充分，只需要查找时，递归查找就可以获得一个角色的全部权限。下面给出用户和对应角色表：

用户名称	角色
(default)	I 级管理员
Bob	II 二级管理员
Tom	III 级管理员

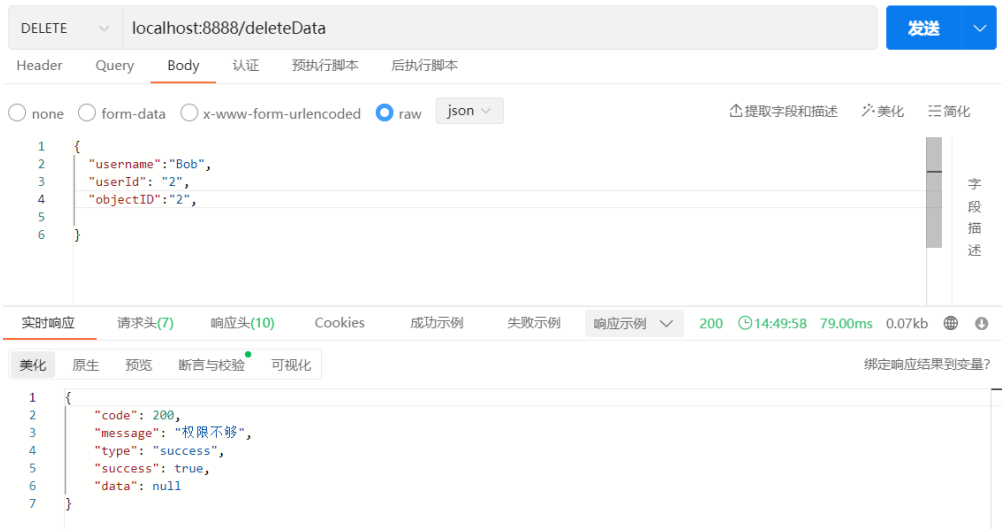
表 2: 用户-角色表

(4) 验证

验证 TOM 能否删除数据：



验证 Bob 能否删除除音乐数据外的数据：



符合预想，代码编写成功！

九、 系统分析

1. 颗粒度

该系统虽然没有做网页，但是可以断言该系统的颗粒度可以控制在按钮组件级别。原因如下：

(1) 每一个组件都可以独立发送 http 请求

(2) 在代码实现上都可以按照相同的逻辑查询角色权限

(3) 根据角色权限来对操作的有效性进行判别

2. 安全性

该系统实现了 RBAC3，保障了业务上的安全性。原因如下：

(1) 每一个用户初始化后都是只赋予最低级别的角色，只能查看。

(2) 更高级别的角色只赋予多余的权限，不再把之前有的权限再次赋予。

(3) 用户只与角色直接关联，不和权限直接关联。

(4) 管理员可以通过管理角色权限保障业务安全，同时可以动态分配角色。

3. 数据库

数据库做到了以下优点：

(1) 达到了 2NF 范式。

(2) 尽可能的减少数据冗余，尤其是在权限授予方面。

十、 实验总结

通过本次实验，我学习到了如下内容：

(1) 深入理解了 RBAC3 权限管理模型，并成功将其应用到实际项目中。

(2) 掌握了使用 Spring Boot 框架构建项目和进行权限验证的方法。

未来改进如下：

(1) 完成前端网页的编写。

(2) 完成全部功能的实现。

参考文献

[1] https://www.cnblogs.com/huangting/p/12654057.html

[2] 星朝. RBAC(基于角色的访问控制）[EB/OL]. 2019[2021-10-26].

[3] 空山鸟语 as. RBAC 权限管理总结 [EB/OL]. 2018[2021-10-26].

[4] https://github.com/Yin-Hongwei/music-website

十一、 附录

为了节约正文内容，将部分较长而且无关紧要的代码放到了附录：

1. SQL 建库

```
/*==============================================================*/
/* DBMS name:    MySQL 5.0                                      */
/* Created on:   2023/10/15 0:35:42                             */
/*==============================================================*/


/*==============================================================*/
/* Table: Relationship_1                                        */
/*==============================================================*/
create table Relationship_1
(
   uid                int not null,
   rid                int
);

alter table Relationship_1
   add primary key (uid);

/*==============================================================*/
/* Table: Relationship_3                                        */
/*==============================================================*/
create table Relationship_3
(
   l_id               int not null,
   rid                int
);

alter table Relationship_3
   add primary key (l_id);

/*==============================================================*/
/* Table: audio_data                                            */
/*==============================================================*/
create table audio_data
(
   d_id               int not null,
   m_id               int,
   uid                varchar(32),
   a_time             float,
   a_name             varchar(45),
   a_link             varchar(45)
);

alter table audio_data
   add primary key (d_id);

/*==============================================================*/
```

```
/* Table: caption_data                                  */
/*==============================================================*/
create table caption_data
(
   d_id              int not null,
   uid               varchar(32),
   c_name            varchar(45),
   c_link            varchar(45)
);

alter table caption_data
   add primary key (d_id);

/*==============================================================*/
/* Table: data                                          */
/*==============================================================*/
create table data
(
   d_id              int not null,
   uid               int
);

alter table data
   add primary key (d_id);

/*==============================================================*/
/* Table: lisence                                       */
/*==============================================================*/
create table lisence
(
   l_id              int not null,
   op_id             int,
   ob_id             int,
   lname             varchar(45)
);

alter table lisence
   add primary key (l_id);

/*==============================================================*/
/* Table: music                                         */
/*==============================================================*/
create table music
(
   m_id              int not null,
   d_id              int,
   cap_d_id          int,
   aud_d_id          int,
   m_kind            varchar(45)
);
```

```
alter table music
   add primary key (m_id);

/*==============================================================*/
/* Table: object                                                */
/*==============================================================*/
create table object
(
   ob_id              int not null,
   ob_name            varchar(45),
   ob_link            varchar(45)
);

alter table object
   add primary key (ob_id);

/*==============================================================*/
/* Table: operation                                             */
/*==============================================================*/
create table operation
(
   op_id              int not null,
   op_name            varchar(45),
   op_link            varchar(45)
);

alter table operation
   add primary key (op_id);

/*==============================================================*/
/* Table: role                                                  */
/*==============================================================*/
create table role
(
   rid                int not null,
   rname              varchar(10),
   rlevel             int,
   rid_father         int
);

alter table role
   add primary key (rid);

/*==============================================================*/
/* Table: singer_data                                           */
/*==============================================================*/
create table singer_data
(
   d_id               int not null,
```

```
   uid               varchar(32),
   s_name            varchar(45),
   s_brief           varchar(45),
   s_age             int,
   s_intime          date
);

alter table singer_data
   add primary key (d_id);

/*==============================================================*/
/* Table: user                                                  */
/*==============================================================*/
create table user
(
   uid               int not null,
   uid2              int,
   usex              tinyint,
   uname             varchar(25),
   upassword         varchar(25),
   uemail            varchar(25),
   uphone            numeric(15,0)
);

alter table user
   add primary key (uid);

alter table Relationship_1 add constraint FK_Relationship_1 foreign key (uid)
      references user (uid) on delete restrict on update restrict;

alter table Relationship_1 add constraint FK_Relationship_2 foreign key (rid)
      references role (rid) on delete restrict on update restrict;

alter table Relationship_3 add constraint FK_Relationship_4 foreign key (rid)
      references role (rid) on delete restrict on update restrict;

alter table Relationship_3 add constraint FK_Relationship_5 foreign key (l_id)
      references lisence (l_id) on delete restrict on update restrict;

alter table audio_data add constraint FK_Inheritance_1 foreign key (d_id)
      references data (d_id) on delete restrict on update restrict;

alter table audio_data add constraint FK_Relationship_11 foreign key (m_id)
      references music (m_id) on delete restrict on update restrict;

alter table caption_data add constraint FK_Inheritance_2 foreign key (d_id)
      references data (d_id) on delete restrict on update restrict;

alter table data add constraint FK_Relationship_8 foreign key (uid)
      references user (uid) on delete restrict on update restrict;
```

```
alter table lisence add constraint FK_Relationship_6 foreign key (op_id)
     references operation (op_id) on delete restrict on update restrict;

alter table lisence add constraint FK_Relationship_7 foreign key (ob_id)
     references object (ob_id) on delete restrict on update restrict;

alter table music add constraint FK_Relationship_10 foreign key (aud_d_id)
     references audio_data (d_id) on delete restrict on update restrict;

alter table music add constraint FK_Relationship_12 foreign key (d_id)
     references singer_data (d_id) on delete restrict on update restrict;

alter table music add constraint FK_Relationship_9 foreign key (cap_d_id)
     references caption_data (d_id) on delete restrict on update restrict;

alter table singer_data add constraint FK_Inheritance_3 foreign key (d_id)
     references data (d_id) on delete restrict on update restrict;
```

2. R 工具类

```
package com.example.yin.common;

import lombok.Data;

@Data
public class R {

    private int code;

    private String message;

    private String type;

    private Boolean success;

    private Object data;

    public static R success(String message) {
        R r = new R();
        r.setCode(200);
        r.setMessage(message);
        r.setSuccess(true);
        r.setType("success");
        r.setData(null);
        return r;
    }
```

```
    public static R success(String message, Object data) {
        R r = success(message);
        r.setData(data);
        return r;
    }

    public static R warning(String message) {
        R r = error(message);
        r.setType("warning");
        return r;
    }

    public static R error(String message) {
        R r = success(message);
        r.setSuccess(false);
        r.setType("error");
        return r;
    }

    public static R fatal(String message) {
        R r = error(message);
        r.setCode(500);
        return r;
    }
}
```