



浙大城市学院

ZHEJIANG UNIVERSITY CITY COLLEGE

Springboot 实现步骤

课程编号: C01098

课程名称: 移动互联网应用开发实践

姓 名: 张亦骞、张泽峰

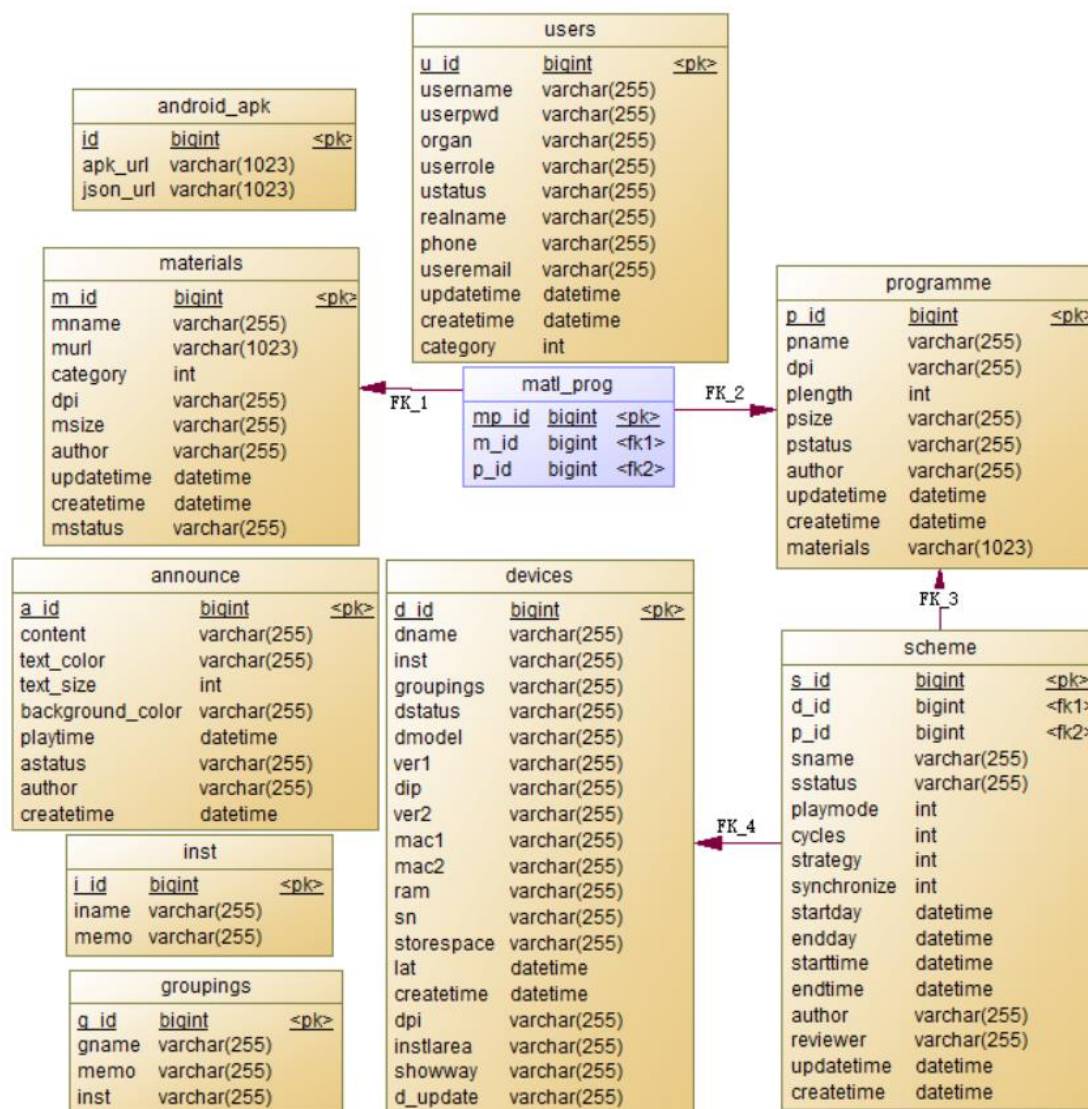
专业班级: 计算 1902

所在学院: 计算学院

报告日期: 2022 年 7 月 7 日

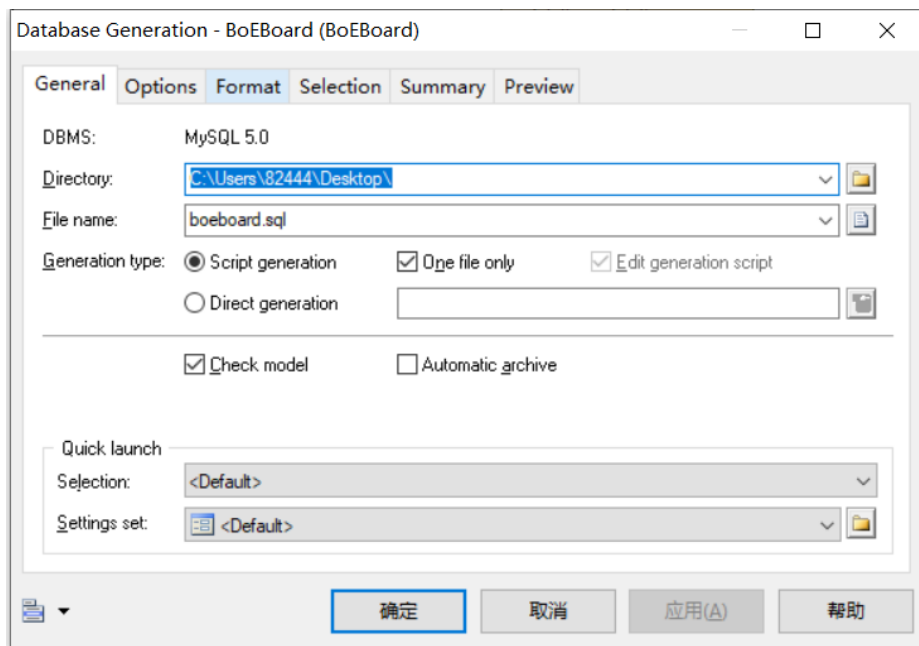
功能实现步骤

一、绘制 er 图



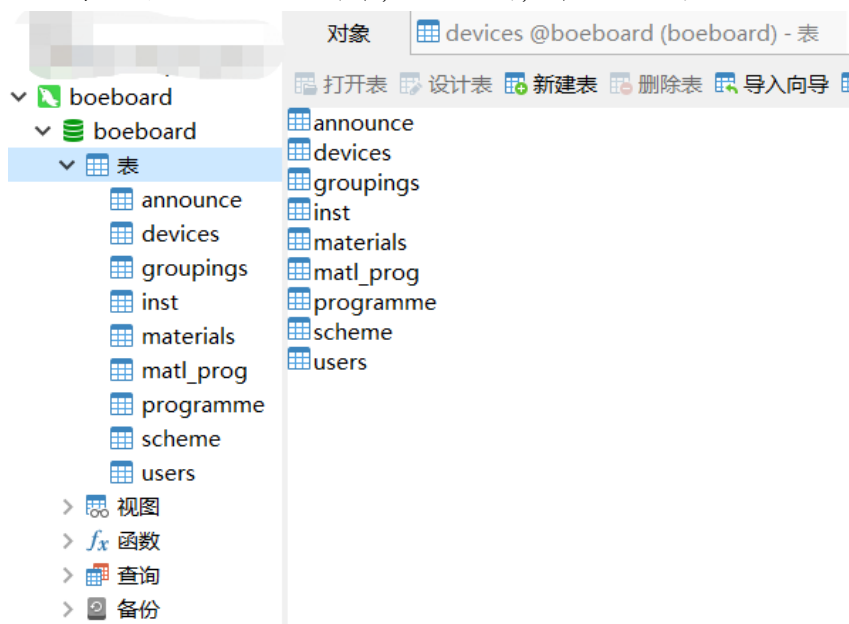
二、er 图转 sql 语句

在 powerdesigner 中，点击 Database->Generate Database:



点击确定，得到 sql 语句。

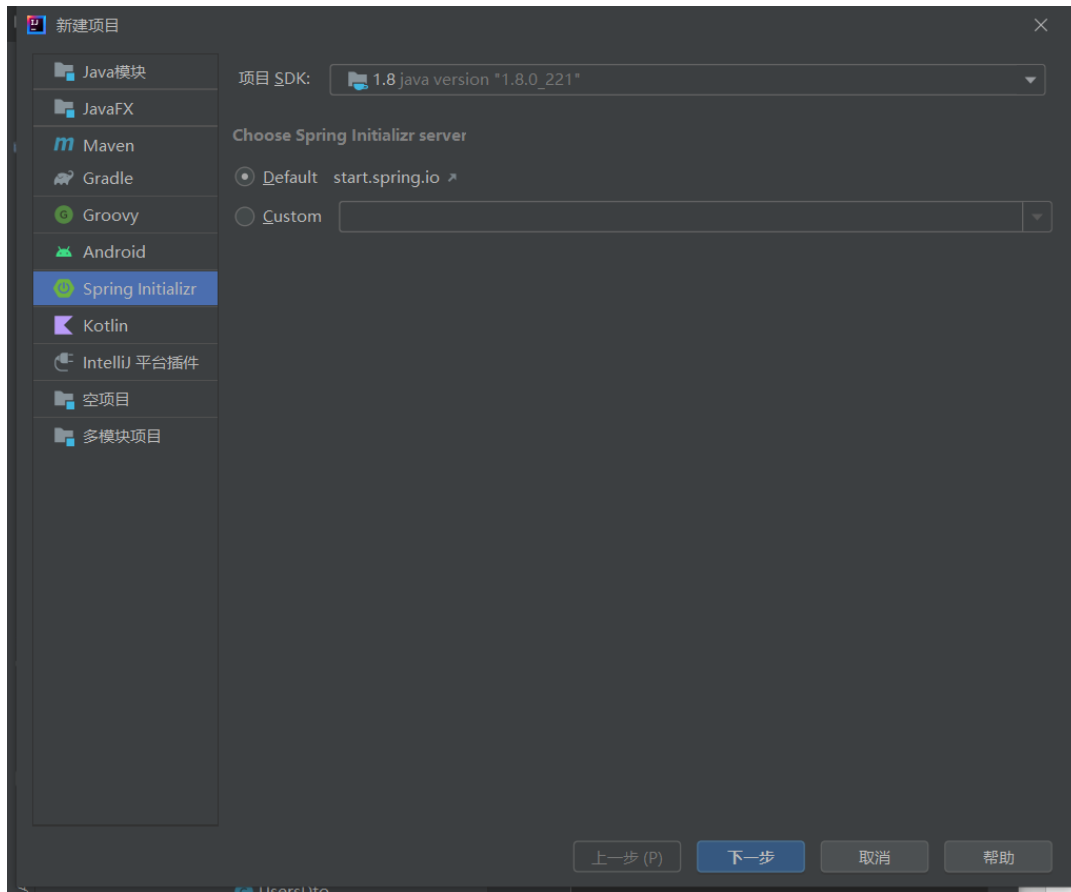
把 sql 语句导入到 navicat 数据库中，执行查询，即建表完成：



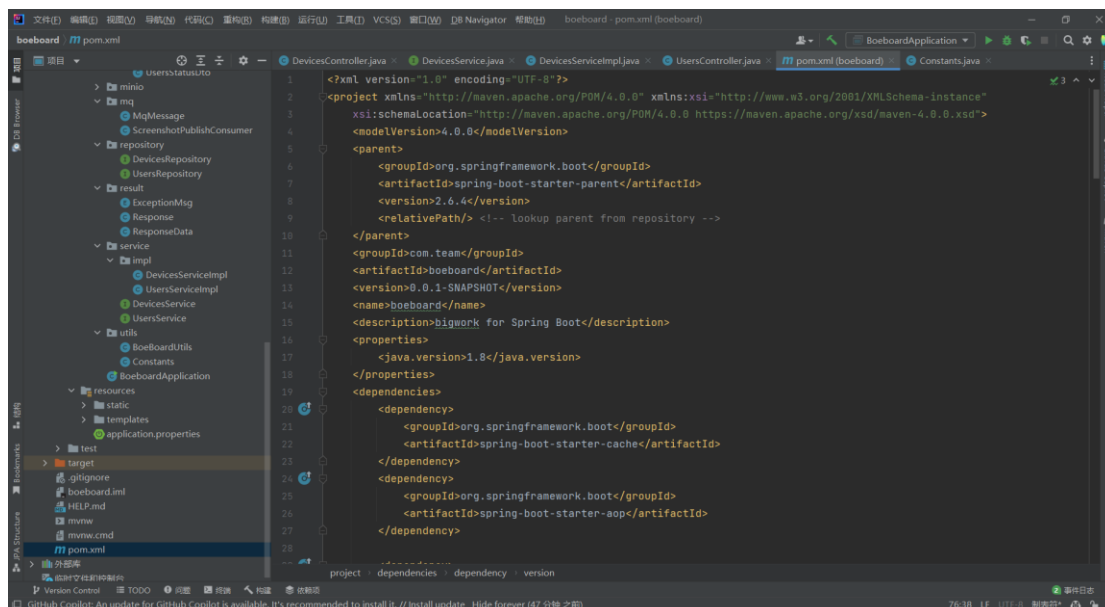
三、框架搭建

使用 springboot2.6.4 版本，使用 IDEA 开发。

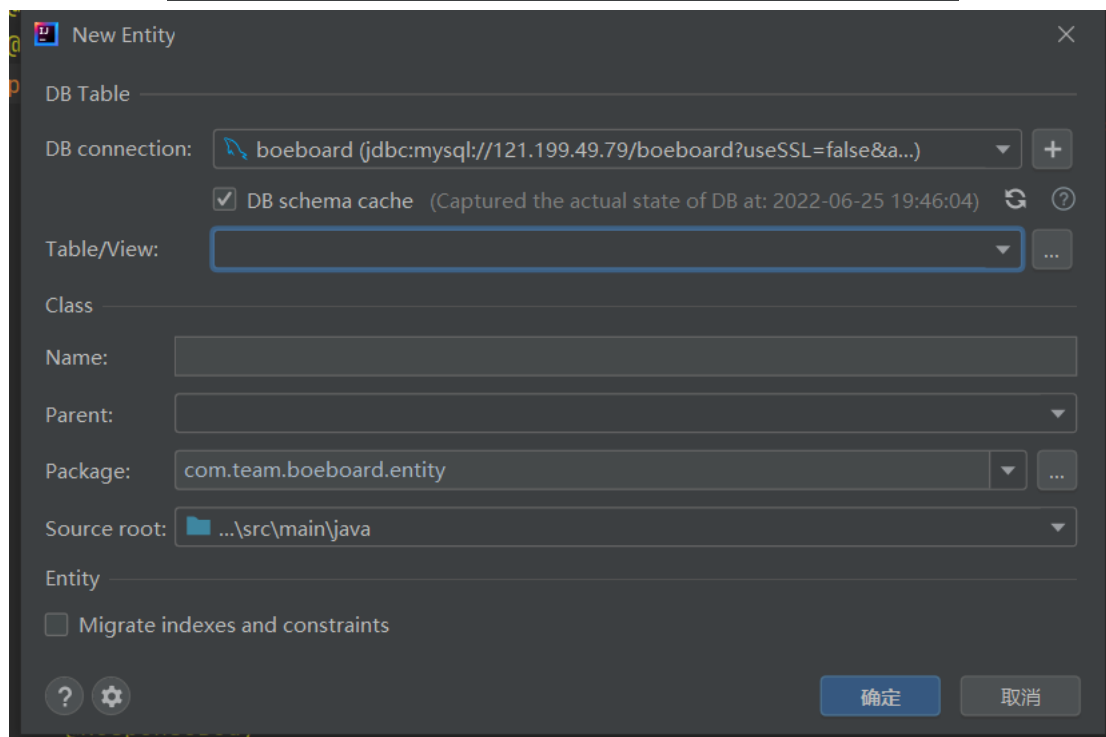
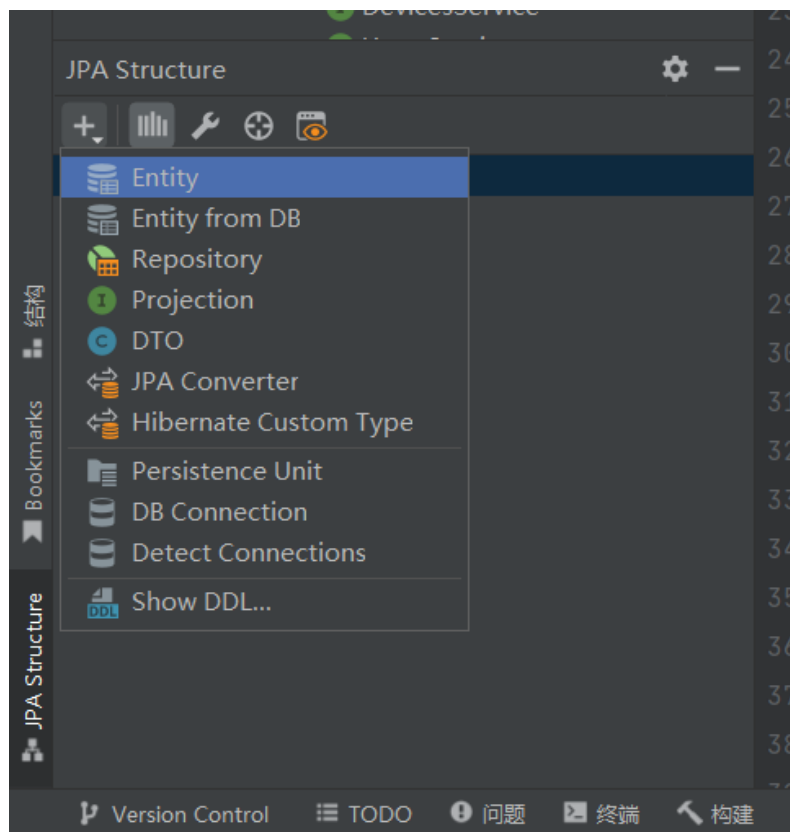
1、创建工程，使用 Spring Initializr 工具创建：



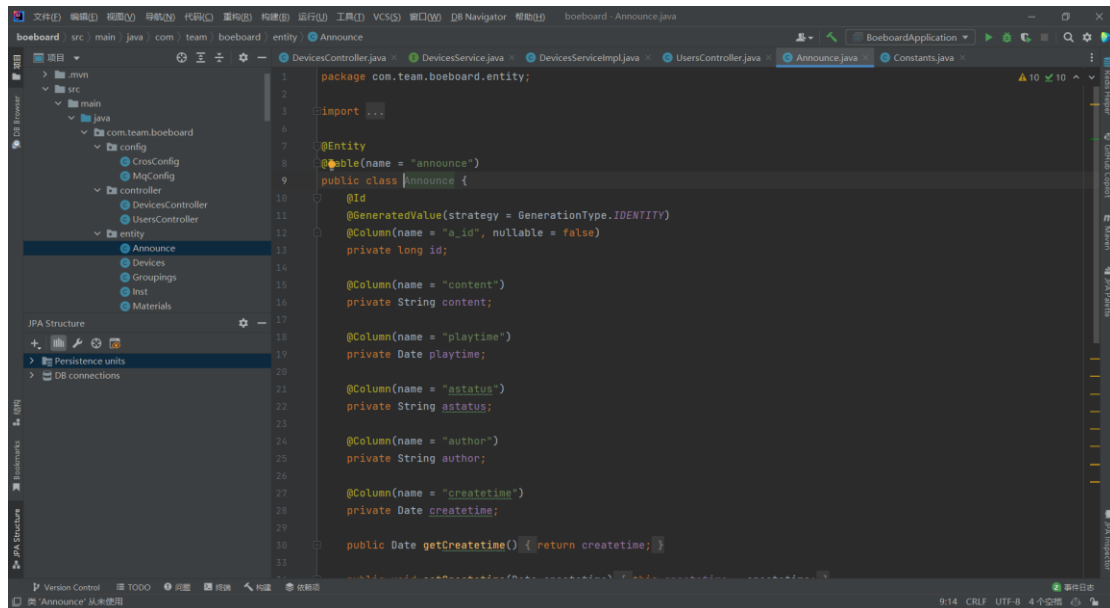
- 2、创建后，在项目中创建 config、controller、entity、exception、form、minio、mq、repository、result、service、utils 等文件夹。目前文件夹都是空的。
- 3、导入相关依赖，包括 json、sa-token、MinIO 等依赖，重新加载项目依赖，使依赖设置得到更新：



- 4、在 JPA Structure 中连接至 mysql 数据库。
- 5、在 JPA Structure 中，选择 Entity from DB 从数据库导入实体。



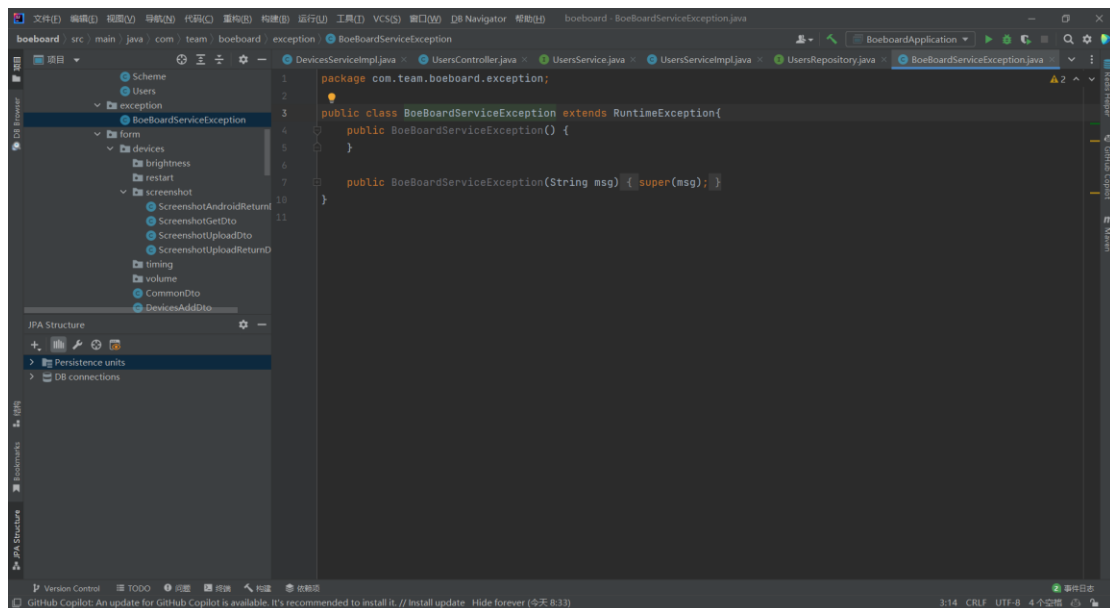
依次导入实体，由于生成的实体会把 `java.util.Date` 类型变成其他类型，故要修改为 `java.util.Date`。



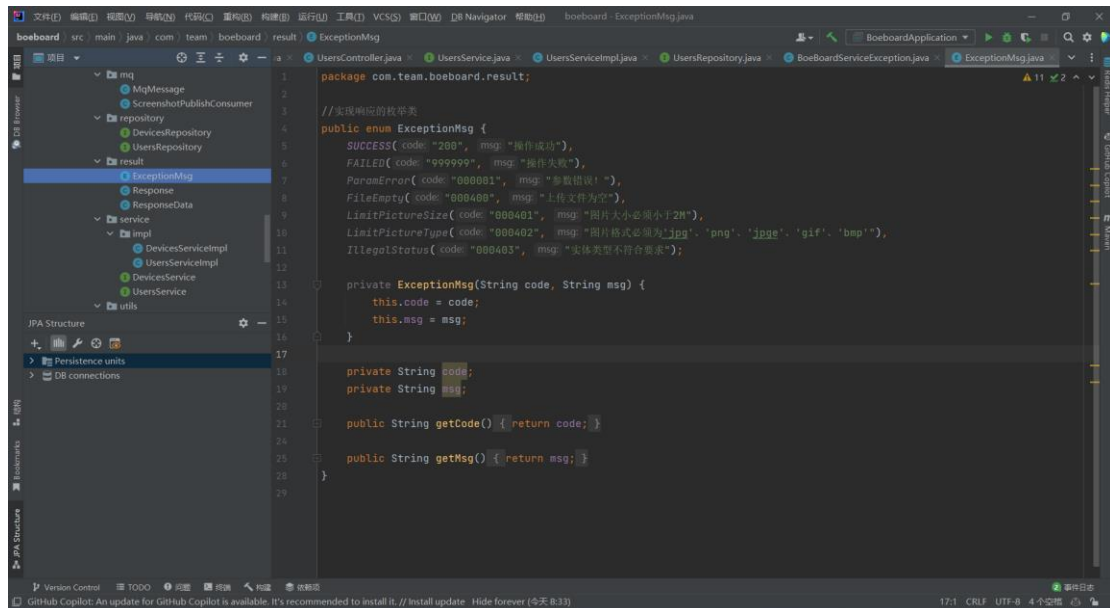
其他实体创建以此类推。

6、创建相关配置类

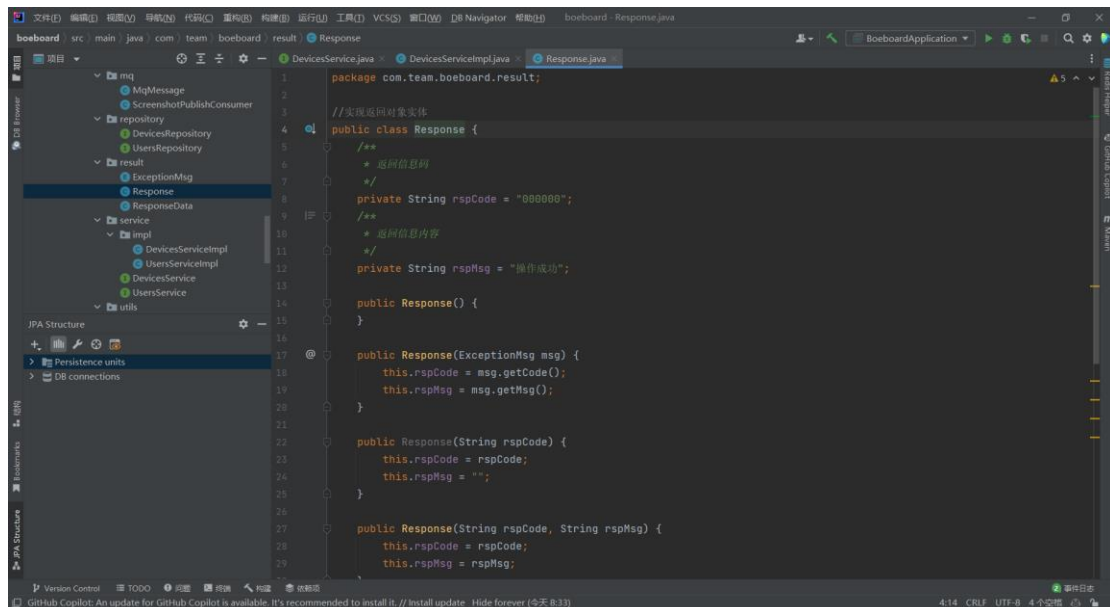
1) 在 exception 文件夹中创建 BoeBoardServiceException.java

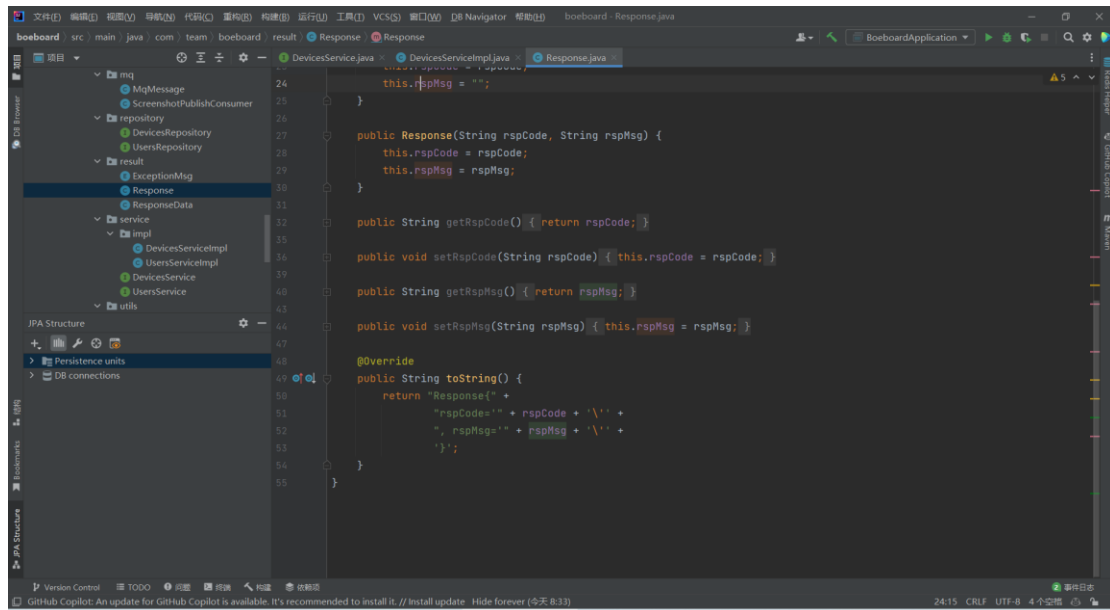


2) 在 result 文件夹中创建 ExceptionMsg.java

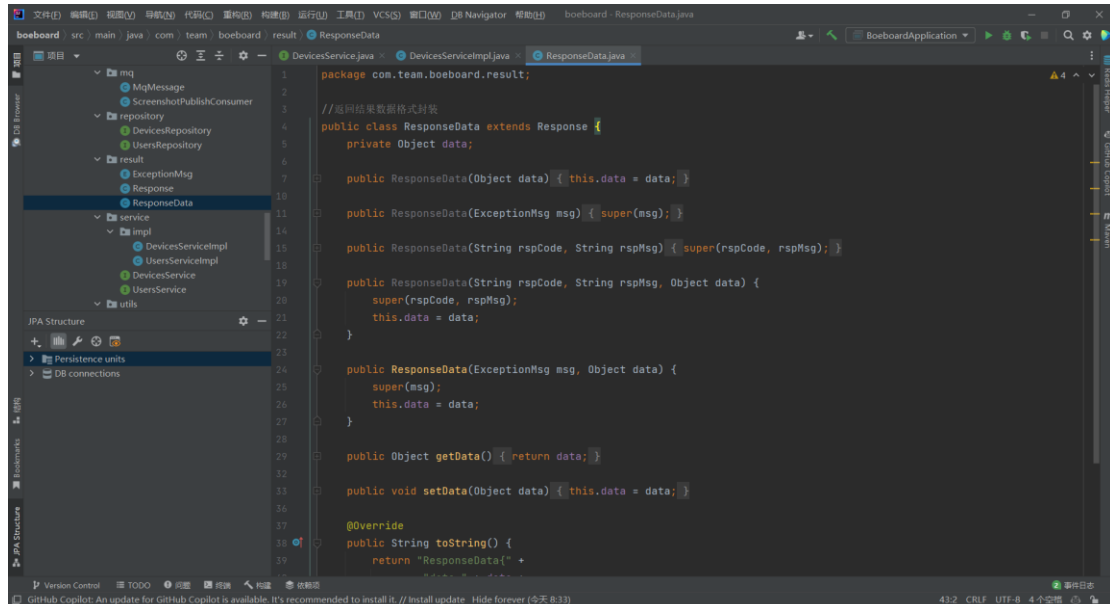


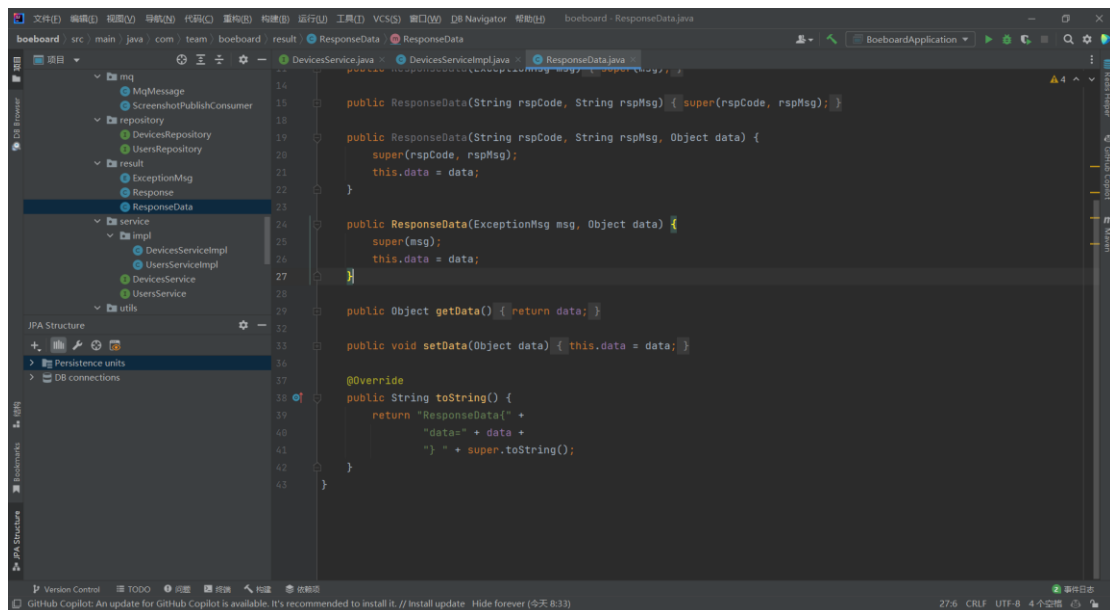
3) 在 result 文件夹中创建 Response.java:



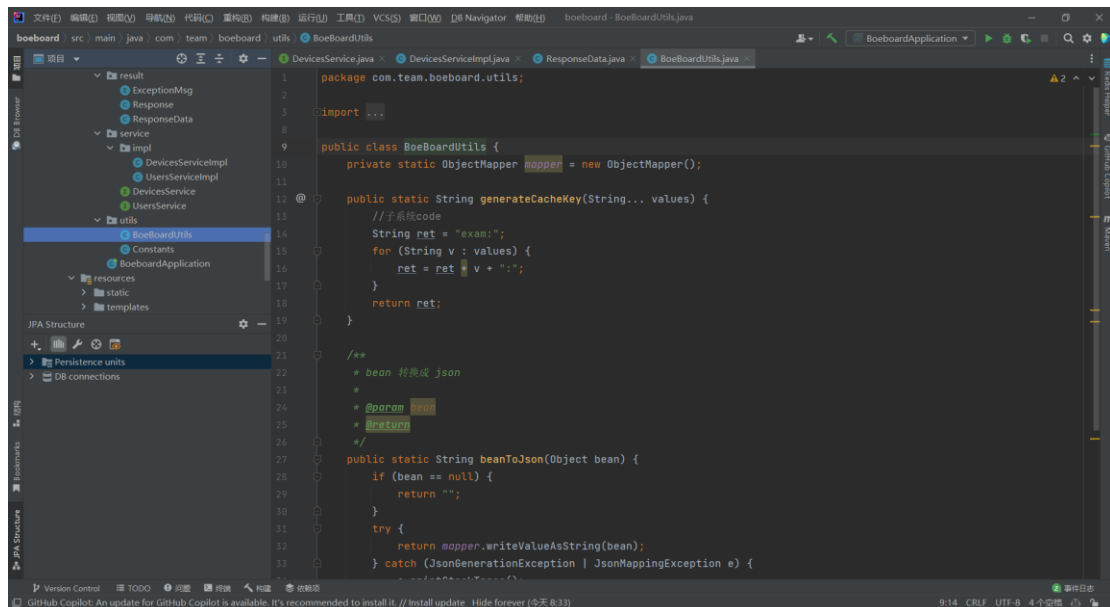


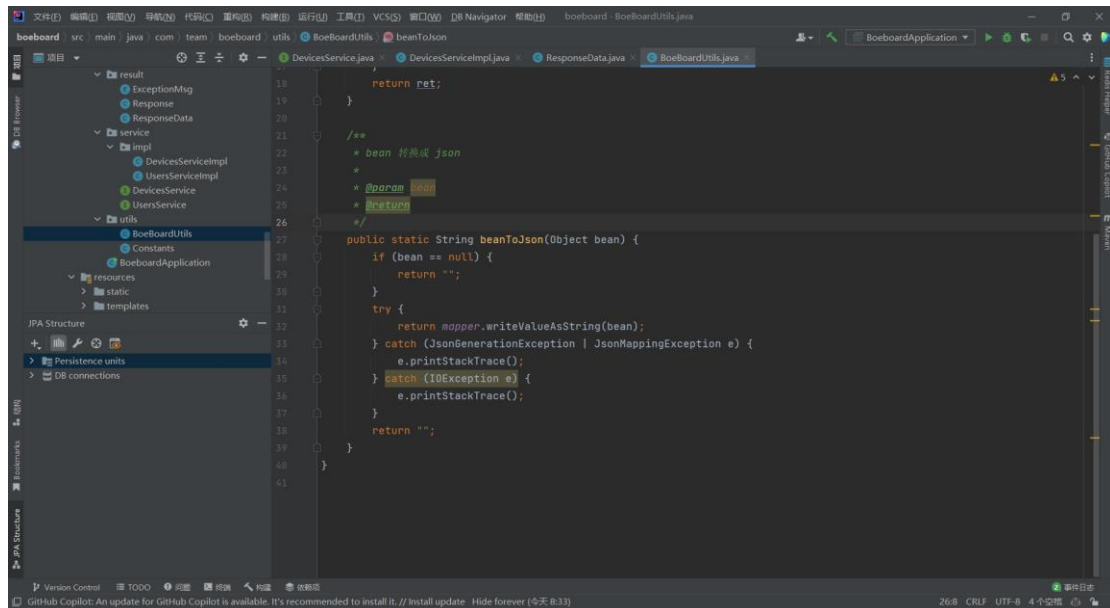
4) 在 result 文件夹中创建 ResponseData.java





5) 在 utils 文件夹中创建 BoeBoardUtils.java, 写有 bean 转 json 等方法。





6) 在 utils 文件夹中创建 Constants. java

```
package com.team.boeboard.utils;

public class Constants {
    //rabbitmq相关常量
    //计划
    public static final String QUE_SCHEME = "scheme";
    //七个设备控制
    public static final String QUE_SCREENSHOT = "screenshot";
    public static final String QUE_TIMING = "timing";
    public static final String QUE_RESTART = "restart";
    public static final String QUE_VOLUME = "volume";
    public static final String QUE_BRIGHTNESS = "brightness";
    public static final String QUE_UPGRADE = "upgrade";
    public static final String QUE_CLEARCACHE = "clearcache";

    //公告消息体
    public static final String QUE_ANNOUNCE = "announce";
}
```

7、以用户管理的添加普通用户为例，编写步骤为：

1) UsersController 中创建方法，标明接口路由，使用 RESTful 风格接口：

```

package com.team.boeboard.controller;

import ...

@RestController
@RequestMapping("/users")
public class UsersController {

    private final Logger logger = LoggerFactory.getLogger(UsersController.class); //日志类

    @Autowired
    private UsersService usersService;

    //TODO 添加普通用户
    @RequestMapping(value = "/register", method = RequestMethod.POST)
    @ResponseBody
    public ResponseData register(@RequestBody RegisterDto registerDto) {
        UsersDto res = usersService.register(registerDto);
        return new ResponseData(ExceptionMsg.SUCCESS, res);
    }
}

```

2) 创建 UsersService:

```

package com.team.boeboard.service;

import ...

@Service
public interface UsersService {

    /**
     * 添加用户
     *
     * @param registerDto
     * @return
     * @throws BoeBoardServiceException
     */
    UsersDto register(RegisterDto registerDto) throws BoeBoardServiceException;
}

```

3) 创建 UsersServiceImpl, 实现接口的内容:

```

package com.team.boeboard.service.impl;

import ...

@Service
public class UsersServiceImpl implements UsersService {
    private final Logger logger = LoggerFactory.getLogger(UsersServiceImpl.class);

    @Autowired
    private UsersRepository usersRepository;

    @Override
    public UsersDto register(RegisterDto registerDto) throws BoeBoardServiceException {
        UsersDto res = new UsersDto(); // 存放结果

        if (registerDto != null) { // vue传入的dto不为空
            // 获取传入的dto的信息
            String username = registerDto.getUsername(); // 账号名
            String userpwd = registerDto.getUserpwd(); // 密码
            String organ = registerDto.getOrgan(); // 所属机构
            String userrole = registerDto.getUserrole(); // 所属角色
            String ustatus = registerDto.getUstatus(); // 账号状态
            String realname = registerDto.getRealname(); // 真实姓名
            String useremail = registerDto.getUseremail(); // 邮箱
            String phone = registerDto.getPhone(); // 手机号

            // 查看是否已经存在该用户名和手机号
            Users user = usersRepository.findIsRegistered(username, phone);
            if (user != null) { // 存在用户，返回错误提示信息

                logger.warn("添加失败，用户已经存在");
                res.setFailed(true);
                res.setId(0);
                res.setMemo("添加失败，用户已经存在");
            } else { // 不存在用户，添加到数据库
                String md5pwd = SaSecureUtil.md5BySalt(userpwd, "boe"); // md5加盐加密处理后的密码
                Date curtime = new Date(System.currentTimeMillis()); // 当前时间
                usersRepository.register(username, md5pwd, organ, userrole, ustatus, realname,
                    phone, useremail, curtime, curtime, "category: 2");

                // 填充返回的dto
                res.setFailed(false); // 没发生异常
                res.setId(1);
                res.setUsername(username);
                res.setUseremail(useremail);
                res.setRealname(realname);
                res.setPhone(phone);
                res.setCategory(2);
                res.setMemo("添加成功");
            }
        }
        return res;
    }
}

```

4) 创建 UsersRepository，其中为要执行的 mysql 语句：

```

package com.team.boeboard.repository;

import ...

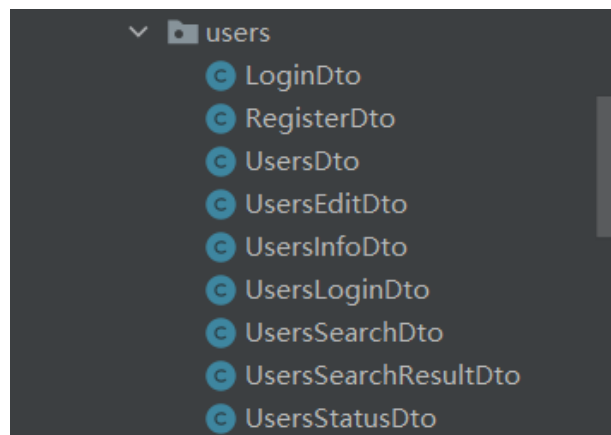
public interface UsersRepository extends JpaRepository<Users, Long>, JpaSpecificationExecutor<Users> {
    Users findById(long id);

    @Query(value = "select * from users where username = ?1 and phone=?2", nativeQuery = true)
    Users findIsRegistered(String username, String phone);

    @Transactional
    @Modifying
    @Query(value = "insert into users(username,userpwd,organ,userrole,ustatus,realname,phone,useremail," +
        "updateTime,createTime,category) values(?1,?2,?3,?4,?5,?6,?7,?8,?9,?10,?11)", nativeQuery = true)
    void register(String username, String userpwd, String organ, String userrole, String ustatus,
        String realname, String phone, String useremail, Date updateTime, Date createTime, int category);
}

```

5) 创建 DTO（用于接收前端传来的数据、返回给前端的数据）:



6) 编写完成后，用 apifox 测试接口是否正确:



- 7) 其他接口均按这种方法以此类推。
- 8) 等 Android、springboot、vue 三端均完成后，连接三端，测试接口功能。

四、使用的工具和框架

1、Springboot

1) 版本：2.6.4

2) mysql 数据库连接所需的配置项：

#数据库部分

```
spring.datasource.url=jdbc:mysql://121.199.49.79:3306/boeboard?useUnicode=true&characterEncoding=utf-8&serverTimezone=Asia/Shanghai&useSSL=true
```

```
spring.datasource.username=root
```

```
spring.datasource.password=123
```

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

```
#spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

#使用的端口

```
server.port = 8080
```

#validate 会验证创建数据库表结构，只会和数据库中的表进行比较，不会创建新表，但是会插入新值，运行程序会校验实体字段与数据库已有的表的字段类型是否相同，不同会报错

```
spring.jpa.hibernate.ddl-auto=validate
```

```
spring.jpa.show-sql=true
```

```
#spring.mvc.pathmatch.matching-strategy=ant_path_matcher
```

3) 使用 IDEA 进行后端代码的编写

2、apifox

1) 使用 apifox 进行 springboot 后端接口的测试

3、Sa-Token 框架

1) Sa-Token 是一个轻量级 Java 权限认证框架，主要解决：登录认证、权限认证、单点登录、OAuth2.0、分布式 Session 会话、微服务网关鉴权等一系列权限相关问题。

2) 所需的配置项

#sa-token 部分

token 名称（同时也是 cookie 名称）

```
sa-token.token-name=boetoken
```

token 有效期，单位 s 默认 30 天，-1 代表永不过期

```
sa-token.timeout=2592000
```

token 临时有效期（指定时间内无操作就视为 token 过期）单位：秒

```
sa-token.activity-timeout=-1
```

是否允许同一账号并发登录（为 true 时允许一起登录，为 false 时新登录挤掉旧登录）

```
sa-token.is-concurrent=true
```

在多人登录同一账号时，是否共用一个 token（为 true 时所有登录共用一个 token，为

false 时每次登录新建一个 token)

```
Sa-token.is-share=false
# token 风格
sa-token.token-style=uuid
# 是否输出操作日志
sa-token.is-log=false
```

3) 所需依赖

<!--Sa-Token 依赖-->

<dependency>

```
<groupId>cn.dev33</groupId>
```

```
<artifactId>sa-token-spring-boot-starter</artifactId>
```

```
<version>1.28.0</version>
```

</dependency>

4、Redis 缓存

1) Redis 是一个远程内存数据库（非关系型数据库），性能强劲，可以存储键值对与 5 种不同类型的值之间的映射，可以将存储在内存的键值对数据持久化到硬盘。

2) 所需的配置项

```
#redis 缓存部分
spring.redis.database=1
spring.redis.host=121.199.49.79
spring.redis.port=6379
spring.redis.password=
spring.redis.pool.max-active=8
spring.redis.pool.max-wait=-1
spring.redis.pool.max-idle=8
spring.redis.pool.min-idle=0
spring.redis.timeout=5000
```

3) 所需依赖

<!--使用 spring cache-->

<dependency>

```
<groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-cache</artifactId>
```

</dependency>

<!--使用 redis-->

<dependency>

```
<groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-data-redis</artifactId>
```

</dependency>

5、RabbitMQ

1) RabbitMQ 是部署最广泛的开源消息代理。

2) 所需的配置项

```
#rabbitmq
spring.rabbitmq.host=121.199.49.79
spring.rabbitmq.port=5672
spring.rabbitmq.username=control
spring.rabbitmq.password=control
spring.rabbitmq.connection-timeout=0
```

3) 所需依赖

```
<!--rabbitmq-->
<!-- https://mvnrepository.com/artifact/org.apache.commons/commons-lang3 -->
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
    <version>3.12.0</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

6、使用配置实现传给前端的内容的时间格式化，配置时区

```
#转为 json 要到的 jackson (用于返回给前端显示)
spring.jackson.date-format=yyyy-MM-dd HH:mm:ss
spring.jackson.time-zone=GMT+8
```

五、用户管理代码

Part1: 接口的实现流程

1、添加普通用户

获取传入的 dto 的信息

- 查看是否已经存在该用户名和手机号
- 如果存在用户，返回错误提示信息；若不存在，就往数据库添加一条记录
- 填充返回的 dto

2、用户登录

获取传入的 dto 中的信息

- 查找数据库判断是否存在该账号
- 若存在，判断输入的密码是否正确（把输入的密码经过 md5 加盐加密处理后和数据库的真正的密码进行比较）；否则返回错误提示信息
- 若输入的密码正确，则使用 `StpUtil.login(user.getId())`；标记当前会话用户登录
- 使用 `StpUtil.getTokenInfo()`；获取当前登录用户的 token
- 把 token 返回

3、获取用户信息

获取传入的 token

- 根据 token 查找用户 id

- 根据用户 id 查找该用户
- 若用户存在，把用户的信息填充到返回的 dto 中；若不存在，返回错误提示信息

4、编辑用户信息

- 获取传入的 dto 的信息
- 根据 username 查找该用户
- 若用户存在，更新数据库中该用户的信息（密码经过加密处理：
`SaSecureUtil.md5BySalt(userpwd, "boe")`）
- 填充返回的 dto

5、启用

- 获取传入的 dto 的信息
- 根据 username 查找该用户
- 若用户存在且用户状态是“停用”，更新数据库中该用户的状态为“启用”
- 填充返回的 dto

6、停用

- 获取传入的 dto 的信息
- 根据 username 查找该用户
- 若用户存在且用户状态是“启用”，更新数据库中该用户的状态为“停用”
- 填充返回的 dto

7、删除用户

- 获取传入的 dto 中的信息
- 根据 username 查找用户
- 若用户存在，更新用户的状态为“已删除”（`updatetime` 字段的值修改为当前时间：
`Date curtime = new Date(System.currentTimeMillis());`）
- 填充返回的 dto

8、查询用户

- 获取传入的 dto 中的信息
- 查询数据库，获取用户列表（根据查询条件执行不同语句，有不限制 `ustatus`、状态为“启用”/“停用”两大种情况）
- 把获得的用户列表 `List<Users>` 构造为 `List<UsersSearchResultDto>`，返回

9、获取用户列表

- 获取传入的 dto 中的信息
- 查询数据库获取所有用户
- 遍历查询到的用户列表 `List<Users> list`，构造返回的 dto 列表

10、用户登出

- 获取传入的用户 token
- 根据 token 找到用户 id
- 根据 `u_id` 找到用户，执行 `StpUtil.logout()`；实现用户登出
- 返回

六、计划管理代码

Part1: 接口的实现流程

1、添加计划（选择设备直接发布）

- 获取传入的作者 token 和 dto 的信息，其中根据传入的 token 找到作者的 name：
`String author = "";`

```

if (StpUtil.getLoginIdByToken(token) != null) {
    String u_id = (String) StpUtil.getLoginIdByToken(token);

    //查找该用户
    Users user = usersRepository.findAUserById(Long.parseLong(u_id));
    if (user != null) {
        author = user.getUsername();
    }
}
String reviewer = author; //审核人默认就是当前用户 author
    → 根据 sname 查询数据库是否存在该计划
    → 若存在该计划，则不能添加，并添加日志信息到缓存中；若不存在该计划，表示可以添加，根据 playmode 播放模式进行划分
        → 若 playmode=1 表示按时段播放，则添加的计划记录包含 startday 开始日期、endday 结束日期、strattime 开始时间、endtime 结束时间、cycles 循环周期，以及其他字段值（创建的计划的状态为“已发布”）
        → 若 playmode=2 表示持续播放，则添加的计划记录不包含 startday 这五个字段的值，其余字段值存在（创建的计划的状态为“已发布”）
        → 更新节目的状态为“使用中”
        → 获取刚刚创建的计划的实体：
long s_id = schemeRepository.searchCreatedScheme();
Scheme scheme1 = schemeRepository.searchSchemeBySid(s_id); //根据 id 查找实体
    → 处理 materials 字段：
String materials = scheme1.getP().getMaterials(); //获取数据库中的 materials 字段
JSONArray jsonArray = JSONArray.parseArray(materials); //string 转 json 数组
    → 遍历 jsonArray，获取每个 url 和种类，并用 video 变量判断是否传的是视频：
List<String> list1 = new ArrayList<>();
List<Integer> list2 = new ArrayList<>();

boolean video = false; //是否是视频

for (int i = 0; i < jsonArray.size(); i++) { //遍历 json 数组
    JSONObject obj = jsonArray.getJSONObject(i); //获取当前那个对象

    String tmp1 = obj.getString("murl"); //获取 url
    int tmp2 = obj.getIntValue("category"); //获取种类

    if (tmp2 == 3) { //是视频
        video = true;
    }
    list1.add(tmp1);
    list2.add(tmp2);
}
    → 如果素材类型是视频，就使用 CATEGORY_SCHEMEVIDEO 发送消息队列的消息；如果素

```

材类型是一张或多张图片，先处理 url 以让 android 端使用访问 url，让多个 url 中间用逗号隔开，使用 CATEGORY_SCHEME 发送消息队列的消息：

```
if (video == true) { //是一个视频
    //和 android 使用消息队列通信，发送计划给 android 【视频】
    MqMessage msg = new MqMessage(MqMessage.CATEGORY_SCHEMEVIDEO);
    msg.appendContent("d_id", d_id); //设备 id
    msg.appendContent("p_id", p_id); //节目 id
    msg.appendContent("plength", scheme1.getP().getPlength()); //节目时长
    msg.appendContent("murl", list1.get(0)); //素材 url 【list1 的第一个元素】
    msg.appendContent("sname", sname); //计划名称
    msg.appendContent("playmode", playmode); //播放模式
    msg.appendContent("cycles", cycles); //循环周期
    msg.appendContent("strategy", strategy); //循环策略
    msg.appendContent("synchronize", synchronize); //多屏同步
    msg.appendContent("startday", startday); //开始日期
    msg.appendContent("endday", endday); //结束日期
    msg.appendContent("starttime", starttime); //开始时间
    msg.appendContent("endtime", endtime); //结束时间
    msg.appendContent("memo", "发送计划"); //备注
    mqService.convertSendAndReceive(Constants.QUE_SCHEMEVIDEO, msg.stringify()); //
    /msg.stringify() 转 json
} else { //是照片
    //处理 url，让多个 url 中间以逗号隔开
    String urls = "";
    for (int i = 0; i < list1.size(); i++) { //list1 和 list2 大小一样，遍历 list1
        素材列表
        String url = list1.get(i);
        if (i == 0) {
            urls += url;
        } else if (i > 0) {
            urls += "," + url;
        }
    }

    //和 android 使用消息队列通信，发送计划给 android 【图片】
    MqMessage msg = new MqMessage(MqMessage.CATEGORY_SCHEME);
    msg.appendContent("d_id", d_id); //设备 id
    msg.appendContent("p_id", p_id); //节目 id
    msg.appendContent("plength", scheme1.getP().getPlength()); //节目时长
    msg.appendContent("murl", urls); //素材 url
    msg.appendContent("sname", sname); //计划名称
    msg.appendContent("playmode", playmode); //播放模式
    msg.appendContent("cycles", cycles); //循环周期
    msg.appendContent("strategy", strategy); //循环策略
}
```

```

        msg.appendContent("synchronize", synchronize);//多屏同步
        msg.appendContent("startday", startday);//开始日期
        msg.appendContent("endday", endday);//结束日期
        msg.appendContent("starttime", starttime);//开始时间
        msg.appendContent("endtime", endtime);//结束时间
        msg.appendContent("memo", "发送计划");//备注
        mqService.convertSendAndReceive(Constants.QUE_SCHEME,msg.stringify());//msg.stringify()转 json
    }

```

→添加日志信息到缓存中：若计划添加成功，就存“计划发布成功”的内容，若添加不成功，就存“计划发布失败”，存到名字叫 scheme_log 的缓存中：

```

//存缓存
++index;
SchemeAddLogDto dto = new SchemeAddLogDto();
dto.setIdx(index);
dto.setTime(new Date(System.currentTimeMillis()));
dto.setContent("[计划]" + res.getSname() + "计划发布成功");
redisTemplate.opsForValue().set(SchemeAddLogDto.cacheKey(index), JSONObject.toJSONString(dto), 2592000, TimeUnit.SECONDS);
logger.warn("[计划]计划发布成功[{}]", SchemeAddLogDto.cacheKey(index));

```

这里 index 是事件日志的下标

→最后把事件日志的数量存到名字叫 cnt_log 的缓存中【事件日志的数量单独用 CntLogDto 表示，在缓存中即名字叫 cnt_log 的缓存，只要增加一条事件日志，当前事件日志的下标就应该是缓存中已经有的数量+1】：

```

//获取缓存中的 cnt
String str = (String) redisTemplate.opsForValue().get(CntLogDto.cacheKey(1));
CntLogDto dto = JSONObject.parseObject(str, CntLogDto.class);//json 转 dto

```

```

int nowCnt = 0;
if (dto != null) {
    nowCnt = dto.getCnt();
}

```

```

//存日志数量
CntLogDto cnt = new CntLogDto();
cnt.setCnt(++nowCnt);
redisTemplate.opsForValue().set(CntLogDto.cacheKey(1), JSONObject.toJSONString(cnt), 2592000, TimeUnit.SECONDS);
logger.warn("[计划]日志数量添加成功[{}]", CntLogDto.cacheKey(1));

```

→把添加的计划的计划的信息填充返回的 dto，返回给 vue 端

2、获取所有计划

int index=1;作为返回的 dto 的下标

→查找数据库，获取所有状态不是“已失效”的计划放到 List<Scheme> list 中

→遍历 list 列表，构造返回的 dto(为了构造 playdate, 形如 2022-06-28~2022-07-01,

使用 SimpleDateFormat 类和 substring 方法:

```
//构造 playdate
Date startday = item.getStartday();
Date endday = item.getEndday();

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
String d1 = (sdf.format(startday)).substring(0, 10);
String d2 = (sdf.format(endday)).substring(0, 10);

String playdate = d1 + "~" + d2;
)
```

→此外还需把素材 url 加到 dto 中:

```
//处理 materials 字段
String materials = item.getP().getMaterials();//获取数据库中的 materials 字段
JSONArray jsonArray = JSONArray.parseArray(materials);//string 转 json 数组
//获取 url、种类
JSONObject obj = jsonArray.getJSONObject(0);//获取第一个对象
vue_url = obj.getString("murl");//获取 url
dto.setMurl(vue_url);

→构造循环时间段 duration:
//TODO 构造 duration
Date starttime = item.getStarttime();
Date endtime = item.getEndtime();
```

```
SimpleDateFormat sdf2 = new SimpleDateFormat("HH:mm:ss");
String t1 = (sdf2.format(starttime)).substring(0, 8);
String t2 = (sdf2.format(endtime)).substring(0, 8);
String duration = t1 + "~" + t2;
dto.setDuration(duration);
```

3、计划详情

获取传入的 dto 中的计划 id

→根据 s_id 在数据库中获得该计划

→若存在该计划,构造返回的 dto,返回 s_id、s_name 等计划的基本信息,并构造 playdate、duration (方法同上一个接口),返回给 vue 端

4、编辑计划

获取传入的 dto 的信息,包括 s_id、sname 等信息

→在数据库中根据计划 id 查找该计划

→若存在该计划,则根据 playmode 播放模式进行划分:若 playmode=1,表示按时段播放,更新数据库中的信息,更新的内容包含 startday 开始日期、endday 结束日期、starttime 开始时间、endtime 结束时间、cycles 循环周期(创建的计划的状态更新为“待发布”);若 playmode=2,表示持续播放,更新数据库中的信息,更新的内容不包含 startday 开始日期等信息(创建的计划的状态更新为“待发布”)

→判断是否使用了其他节目,若使用了其他节目,把原节目状态改为“未使用”,把新节目状态改为“使用中”;若未改变使用的节目,则不更改节目的信息:

```

if (p_id != scheme.getP().getId()) { // 不相等
    programmeRepository.changeOldProgram(curtime, scheme.getP().getId()); // 更改原节目
    programmeRepository.changeNewProgram(curtime, p_id); // 更改新节目
}

```

→ 构造返回的 dto 给 vue 端, 其中处理 materials 字段和获取所有计划接口的是相同的处理方法。

5、删除计划

获取传入的 dto 中的计划 id

→ 在数据库中根据计划 id 查找是否存在该计划, 若存在且计划状态不是“审核中”则进行软删除, 并填充返回的 dto; 否则返回错误提示信息。此外, 更新使用的节目状态为“未使用”:

```

programmeRepository.deleteProgramByDeleteScheme(curtime, scheme.getP().getId());

```

→ 在构造返回的 dto 后, 把事件日志存到缓存中, 并更新事件日志数量【事件日志的数量单独用 CntLogDto 表示, 在缓存中即名字叫 cnt_log 的缓存, 只要增加一条事件日志, 当前事件日志的下标就应该是缓存中已有的数量+1】:

//存缓存

```

++index;

```

```

SchemeAddLogDto dto = new SchemeAddLogDto();

```

```

dto.setId(index);

```

```

dto.setTime(new Date(System.currentTimeMillis()));

```

```

dto.setContent("[计划]" + res.getName() + "计划删除成功");

```

```

redisTemplate.opsForValue().set(SchemeAddLogDto.cacheKey(index), JSONObject.toJSONString(dto), 2592000, TimeUnit.SECONDS);

```

```

logger.warn("[计划]计划删除成功[{}]", SchemeAddLogDto.cacheKey(index));

```

//存缓存

```

++index;

```

```

SchemeAddLogDto dto = new SchemeAddLogDto();

```

```

dto.setId(index);

```

```

dto.setTime(new Date(System.currentTimeMillis()));

```

```

dto.setContent("[计划]" + res.getName() + "计划删除失败");

```

```

redisTemplate.opsForValue().set(SchemeAddLogDto.cacheKey(index), JSONObject.toJSONString(dto), 2592000, TimeUnit.SECONDS);

```

```

logger.warn("[计划]计划删除失败[{}]", SchemeAddLogDto.cacheKey(index));

```

//获取缓存中的 cnt

```

String str = (String) redisTemplate.opsForValue().get(CntLogDto.cacheKey(1));

```

```

CntLogDto dto = JSONObject.parseObject(str, CntLogDto.class); // json 转 dto

```

```

int nowCnt = 0;

```

```

if (dto != null) {

```

```

    nowCnt = dto.getCnt();

```

```

}

```

```
//存日志数量
CntLogDto cnt = new CntLogDto();
cnt.setCnt(++nowCnt);
redisTemplate.opsForValue().set(CntLogDto.cacheKey(1),
JSONObject.toJSONString(cnt),2592000, TimeUnit.SECONDS);
logger.warn("[计划] 日志数量添加成功[{}]", SchemeAddLogDto.cacheKey(1));
```

6、查询计划

获取传入的 dto 的信息，包括计划名称和计划状态

→判断计划的状态

→若未指定 sname，且状态选择为“所有状态”，就获取所有计划，调用 makeReturnDtoList(list)函数，遍历 list，构造返回的 dto

→若指定了 sname，且状态选择为“所有状态”，就根据 sname 进行模糊查询，模糊查询的 sql 语句为：

```
select * from scheme where sname like CONCAT('%',?1,'%')
```

调用 makeReturnDtoList(list)函数，遍历 list，构造返回的 dto

→若未指定 sname，且状态选择为“待发布”或“已发布”，就根据状态进行搜索，调用 makeReturnDtoList(list)函数，遍历 list，构造返回的 dto

→若指定 sname，且状态选择为“待发布”或“已发布”，就根据 sname 和状态进行搜索，调用 makeReturnDtoList(list)函数，遍历 list，构造返回的 dto，返回给 vue

7、复制计划

获取传入的计划 id 和用户的 token

→根据计划 id 查询数据库是否存在该计划，根据 token 先找到该用户：

//查询数据库是否存在该用户

```
Users users = new Users();
if (StpUtil.getLoginIdByToken(token) != null) {
    String u_id = (String) StpUtil.getLoginIdByToken(token);

    //查找该用户
    Users user = usersRepository.findAUserByUId(Long.parseLong(u_id));
    if (user != null) {
        users = user;
    }
}
```

→再查询数据库是否存在该用户

→若两者均存在，根据 playmode 播放模式执行数据库添加一条记录（复制得到的计划的状态为“待发布”）

→返回刚刚复制得到的计划的 id

8、重新发布计划

获取传入的计划 id 和用户 id

→根据计划 id 查询数据库是否存在该计划，根据用户 id 查询数据库是否存在该用户

→若两者均存在，修改数据库的记录，复制得到的计划的状态为“已发布”

→返回计划 id

七、设备管理代码

Part1: 接口的实现流程

1、添加设备

获取传入的 dto 的信息，包括 dname、mac2、inst、sn

→根据无线 mac 地址在数据库中查找是否存在该设备

→若存在，就返回错误提示信息，添加失败

→若不存在，往数据库添加一条设备的记录，构造返回的 dto，把设备的信息返回给 vue

2、控制——屏幕截图

首先，vue 端调用接口 screenshot1

→springboot 通过消息队列把提示 android 端开始截屏的消息发给 android 端

→构造返回给 vue 的 dto

→接着，android 端调用接口 screenshot2，传入 MultipartFile 格式的截屏文件

→把截屏上传到 MinIO 上（使用其他组员的上传 MinIO 的方法）

→把截屏的 url 和文件名称（非完整 url）存到缓存中

→最后，vue 端调用接口 screenshot3，传入获取截屏的指令（一个 String 类型的字符串）

→从缓存中获取截屏信息，构造返回给 vue 的 dto

3、控制——自动开关机

vue 端传入设定的开关机的 starttime、endtime 和 cycles 循环周期

→通过消息队列把这些数据传给 android 端，android 端通过消息队列收到后执行

→构造返回给 vue 的 dto

4、控制——重启系统

vue 端调用接口

→通过消息队列把重启系统的消息传给 android 端，android 端通过消息队列收到后执行

→构造返回给 vue 的 dto

5、控制——音量控制

vue 端调用接口，传入要设定的音量值

→通过消息队列把音量值传给 android 端，android 端通过消息队列收到后执行

→构造返回给 vue 的 dto

6、控制——亮度控制

vue 端调用接口，传入要设定的亮度值

→通过消息队列把音量值传给 android 端，android 端通过消息队列收到后执行

→构造返回给 vue 的 dto

7、控制——系统升级

vue 端调用接口

→通过消息队列把系统升级的消息传给 android 端，android 端通过消息队列收到后执行

→构造返回给 vue 的 dto

8、控制——清除缓存

vue 端调用接口

→通过消息队列把清除缓存的消息传给 android 端，android 端通过消息队列收到后执行

→构造返回给 vue 的 dto

9、编辑设备

获取传入的 dto 的信息，包括设备 id、设备名称、分组

→根据设备 id 在数据库中查找是否存在该设备

→若存在，更新数据库，构造返回的 dto

→若不存在，返回错误提示信息

10、获取所有设备

int index=1;作为返回的 dto 的下标

→查找数据库，获取所有计划放到 List<Devices> list 中

→遍历 list 列表，构造返回的 dto

11、设备上下线状态

定义事件日志在缓存中的下标：

```
public static int index_log = 0; //事件日志下标
```

→获取传入的设备当前状态：off（离线）、relax（空闲）、playing（播放），分别更新数据库中设备的状态

→获取当前使用的设备（因为设定一台设备在使用该系统，故其在数据库中的下标为 1）

→把设备上下线状态更新的操作存到缓存中：

```
++index_log;
```

```
DevicesUpAndDownDto dto = new DevicesUpAndDownDto();
```

```
dto.setIdx(index_log);
```

```
dto.setTime(new Date(System.currentTimeMillis()));
```

```
dto.setContent("[设备]" + devices.getDname() + "上下线状态更新成功");
```

```
redisTemplate.opsForValue().set(DevicesUpAndDownDto.cacheKey(index_log),
```

```
JSONObject.toJSONString(dto), 2592000, TimeUnit.SECONDS);
```

```
logger.warn("[设备]上下线状态更新成功 [{}]",  
DevicesUpAndDownDto.cacheKey(index_log));
```

→更新缓存中的 cnt 【事件日志的数量单独用 CntLogDto 表示，在缓存中即名字叫 cnt_log 的缓存，只要增加一条事件日志，当前事件日志的下标就应该是缓存中已有的数量+1】：

```
//获取缓存中的 cnt
```

```
String str = (String) redisTemplate.opsForValue().get(CntLogDto.cacheKey(1));
```

```
CntLogDto dto = JSONObject.parseObject(str, CntLogDto.class); //json 转 dto
```

```
int nowCnt = 0;
```

```
if (dto != null) {
```

```
    nowCnt = dto.getCnt();
```

```
}
```

```
//存日志数量
```

```
CntLogDto cnt = new CntLogDto();
```

```
cnt.setCnt(++nowCnt);
```

```
redisTemplate.opsForValue().set(CntLogDto.cacheKey(1),
```

```
JSONObject.toJSONString(cnt), 2592000, TimeUnit.SECONDS);
```

```
logger.warn("[设备]上下线状态更新成功 [{}]", CntLogDto.cacheKey(1));
```

八、分组管理代码

Part1: 接口的实现流程

1、添加分组

获取传入的 dto 的信息，包括分组名称、所属机构、分组描述、设备名称

→根据分组名称查找数据库中是否已经存在该名字的分组，若存在则不能添加，返回错误提示信息；若不存在，则：

→根据设备名称找设备的实体，若找到设备，往数据库添加一条分组记录

→更新该设备的分组字段

→获取刚创建的这个分组实体：

```
long g_id = groupingsRepository.searchCreatedGroupings();
```

```
Groupings groupings = groupingsRepository.findGroupingsByGid(g_id); //根据 id 找分组实体
```

→填充返回给 vue 的 dto

2、编辑分组

获取传入的 dto 的信息，包括分组名称、所属机构、分组描述、设备名称

→根据分组名称查找数据库中是否已经存在该名字的分组，若存在则不能添加，返回错误提示信息；若不存在，则：

→根据设备名称找设备的实体，若找到设备，用 String 类型的 name 变量记录设备在修改前的分组名称

→根据该修改前的分组名称，找到要修改的那个分组实体

→若找到，就更新分组，接着更新设备的分组字段，因为一台设备只能在一个分组中，最后构造返回的 dto

→若未找到，返回错误提示信息

3、删除分组

获取传入的分组名称

→根据分组名称查找分组的实体

→若找到分组的实体 curgroup，根据分组的名称在设备中找是否有设备使用该分组，若有就把对应设备的分组置为 null：

```
if (devices != null) { //有设备使用该分组
```

```
    devicesRepository.updateGroupings(null, devices.getDname());
```

```
}
```

→根据 curgroup 的分组 id 删除该分组

→构造返回的 dto

→若不存在分组的实体 curgroup，返回错误提示信息

4、获取所有分组

从数据库中获取所有分组，构造为 List<GroupingsDto>，返回给 vue

九、首页【部分】代码

Part1: 接口的实现流程

1、获取计划数量、获取设备数量、获取设备状态、获取设备分布、获取事件日志列表

该功能实现在 HomeServiceImpl.java 的 GetHomeInfo 接口中。

//TODO 获取计划数量

```
int scheme_num = schemeRepository.countScheme();
```

```

//TODO 获取设备数量
int device_num = devicesRepository.countDevices();

//TODO 获取设备状态
List<DevicesAllDto> list = new ArrayList<>(); //存放所有设备的 dto 结果
int index = 1; //下标

//查找数据库
List<Devices> devices = devicesRepository.findListOfDevices(); //获取所有设备

//遍历 list, 构造返回 dto 列表
for (Devices item : devices) {
    DevicesAllDto dto = new DevicesAllDto();
    dto.setFailed(false);
    dto.setIdx(index++);
    dto.setD_id(item.getId());
    dto.setDname(item.getDname());
    dto.setInst(item.getInst());
    dto.setGroupings(item.getGroupings());
    dto.setMac2(item.getMac2());
    dto.setDpi(item.getDpi());
    dto.setDstatus(item.getDstatus());
    dto.setVer1(item.getVer1());
    dto.setMemo("获取设备列表成功");
    list.add(dto);
}

int cnt1 = 0, cnt2 = 0, cnt3 = 0; //三种状态的设备数量
if (list.size() > 0) {
    //遍历 list
    for (DevicesAllDto item : list) {
        if ((item.getDstatus()).equals("离线")) {
            cnt1++;
        } else if ((item.getDstatus()).equals("播放")) {
            cnt2++;
        } else if ((item.getDstatus()).equals("空闲")) {
            cnt3++;
        }
    }
}

//TODO 获取设备分布
//机构数量
Map<String, Integer> inst_num = devicesRepository.inst();
//分组数量

```

```

Map<String, Integer> groupings_num = devicesRepository.groupings();

//TODO 获取事件日志
Map<Date, String> logs = new HashMap<>();

//获取事件日志数量
int cnt = 0;
String str = (String) redisTemplate.opsForValue().get(CntLogDto.cacheKey(1));
CntLogDto dto = JSONObject.parseObject(str, CntLogDto.class); //json 转 dto
if (dto != null) {
    cnt = dto.getCnt();
}

//把每条日志加到 map 中
for (int i = 1; i <= cnt; i++) {
    String str2 = (String)
redisTemplate.opsForValue().get(SchemeAddLogDto.cacheKey(i));
    SchemeAddLogDto dto2 = JSONObject.parseObject(str2,
SchemeAddLogDto.class); //json 转 dto
    if (dto2 != null) {
        logs.put(dto2.getTime(), dto2.getContent());
    }
}
}

```

十、MinIO服务器搭建

1、相关依赖

```
<!--MinIO-->
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>cn.hutool</groupId>
    <artifactId>hutool-all</artifactId>
    <version>5.5.7</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>io.minio</groupId>
    <artifactId>minio</artifactId>
    <version>7.0.2</version>
</dependency>
```

2、配置文件

将MinIO部署到服务器，设置连接名和密钥

```
#minioåSHYS0SåSS3SPAåCSI%çHTJESA
spring.servlet.multipart.max-file-size=100MB
spring.servlet.multipart.max-request-size=100MB
minio.address=http://47.99.158.248:9000/
minio.accessKey=
minio.secretKey=
minio.bucketName=myfile
```

3、查阅MinIO官网，获得java client API

下载文件接口为例：

```
/**
 * 下载文件
 *
 * @param originalName 文件路径
 */
public InputStream downloadFile(MinioClient minioClient, String originalName,
    HttpServletResponse response) {
    try {
        InputStream file = minioClient.getObject(bucketName, originalName);
```

```

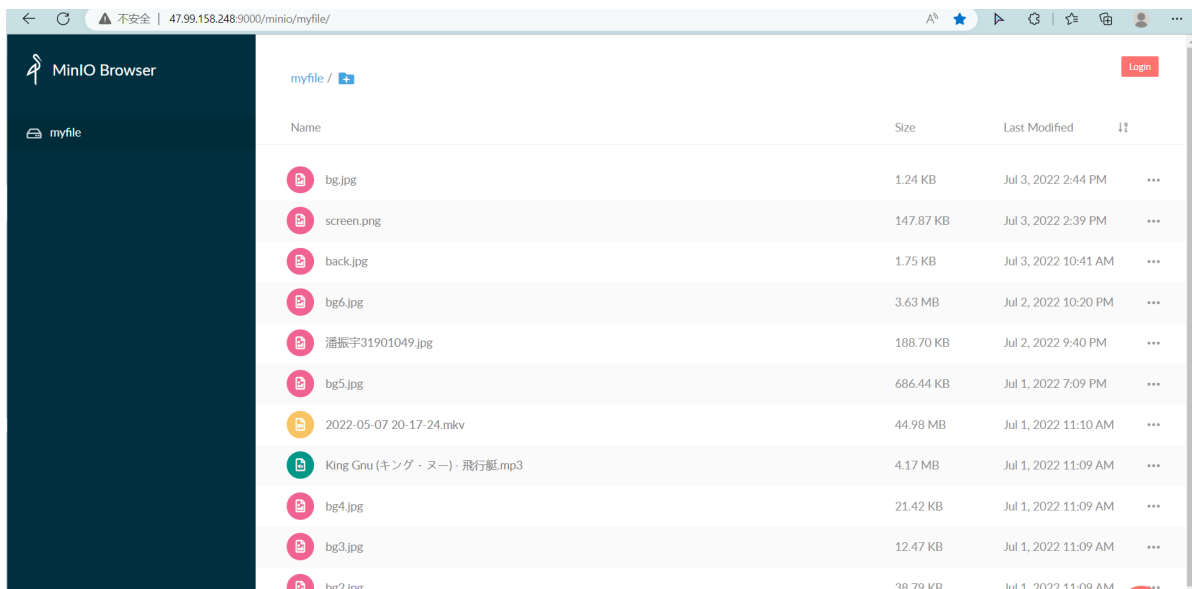
        String filename = new String(originalName.getBytes("ISO8859-1"),
StandardCharsets.UTF_8);
        if (StringUtil.isNotBlank(originalName)) {
            filename = originalName;
        }
        response.setHeader("Content-Disposition", "attachment;filename=" +
filename);
        ServletOutputStream servletOutputStream = response.getOutputStream();
        int len;
        byte[] buffer = new byte[1024];
        while ((len = file.read(buffer)) > 0) {
            servletOutputStream.write(buffer, 0, len);
        }
        servletOutputStream.flush();
        file.close();
        servletOutputStream.close();
        return file;
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
}

```

4、MinIO Browser

访问地址: *****:9000/minio

MinIO服务器固定端口为9000, 前面为自己搭建的服务器地址



十一、素材管理

1、数据库

名	类型	长度	小数点	不是 null	虚拟	键	注释
m_id	bigint			<input checked="" type="checkbox"/>	<input type="checkbox"/>	 1	素材id
mname	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>		素材名称
murl	varchar	1023		<input type="checkbox"/>	<input type="checkbox"/>		素材路径
category	int			<input type="checkbox"/>	<input type="checkbox"/>		素材种类 (1: 图片 2: 音频 3: 视频)
dpi	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>		素材分辨率
msize	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>		素材大小
author	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>		素材作者 (users的真实名称)
updatetime	datetime			<input type="checkbox"/>	<input type="checkbox"/>		素材更新时间
createtime	datetime			<input type="checkbox"/>	<input type="checkbox"/>		素材创建时间
mstatus	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>		素材状态

2、Controller层

增加restfull接口：

实现功能：

- 上传文件
- 下载文件
- 回图片list的JSON字符串
- 按名称模糊查找所有文件
- 删除文件
- 修改文件名称

```
@RestController
@RequestMapping("/material")
public class MaterialController {

    @Autowired
    private MinIOService minIOService;
    @Autowired
    private MaterialService materialService;
    @Autowired
    private UsersRepository usersRepository;

    /**
     * 上传文件
     * @param file
     * @return
     */
    @ResponseBody
    @RequestMapping(value = "/uploadfile", method = RequestMethod.POST)
    public ResponseData uploadFile(@RequestBody MultipartFile file,@RequestParam
String token) throws IOException {
        MinioClient minioClient = minIOService.getMinioClient();

        String id = (String) Stputil.getLoginIdByToken(token);
        String user_name =
usersRepository.findById(Long.parseLong(id)).getUsername();

        if (minioClient == null) {
            return new ResponseData(ExceptionMsg.FAILED,"上传失败，无法连接MinIo服务
器");
        }

        minIOService.uploadFile(minioClient, file,user_name);
    }
}
```

```

        return new ResponseData(ExceptionMsg.SUCCESS, "上传成功");
    }

    /**
     * 返回图片list的JSON字符串
     */
    @ResponseBody
    @RequestMapping(value = "/getall", method = RequestMethod.GET)
    public ResponseData GetAll(){
        List<BackToWebDto> result = materialService.getAll_photos();

        return new ResponseData(ExceptionMsg.SUCCESS, result);
    }

    /**
     * 按名称模糊查找所有图片文件
     */
    @ResponseBody
    @RequestMapping(value = "/select", method = RequestMethod.GET)
    public ResponseData SearchByName(@RequestBody Map map) {
        String search_name = (String)map.get("search_name");
        List<BackToWebDto> result = new ArrayList<>();
        try{
            result = materialService.SelectByName(search_name);
        }catch (BoeBoardServiceException e){
            return new ResponseData(ExceptionMsg.FAILED, e);
        }

        return new ResponseData(ExceptionMsg.SUCCESS, result);
    }

    /**
     * 删除图片
     * @param
     * @return
     */
    @ResponseBody
    @RequestMapping(value = "/delete", method = RequestMethod.POST)
    public ResponseData Delete_Material(@RequestBody Map map){

        String file_name = (String)map.get("file_name");
        MinioClient minioClient = minIOService.getMinioClient();
        if (minioClient == null) {
            return new ResponseData(ExceptionMsg.FAILED, "连接MinIO服务器失败");
        }
        boolean flag = minIOService.deleteFile(minioClient, file_name);
        if(flag == true){
            return new ResponseData(ExceptionMsg.SUCCESS, "删除成功");
        }else {
            return new ResponseData(ExceptionMsg.FAILED, "删除失败");
        }
    }

    /**
     * 修改文件名称

```



```

    * @param map
    *      {
    *          "old_name":"zzz",
    *          "new_name":"zza"
    *      }
    */
    @ResponseBody
    @RequestMapping(value = "/update", method = RequestMethod.POST)
    public ResponseData updateMaterial(@RequestBody Map map){
        // 分为两步
        // 1、操作material 的 service层修改数据库
        // 2、获取inputstream流转化成MultipartFile重新上传，并删除bucket中原先的文件

        String old_name = (String)map.get("old_name");
        String new_name = (String)map.get("new_name");
        MinioClient minioClient = minIOService.getMinioClient();
        InputStream inputStream = minIOService.getObject(minioClient,old_name);
        //获取原文件流

        materialService.updateFile(old_name,new_name);

        return new ResponseData(ExceptionMsg.SUCCESS,"修改成功");
    }

    /**
     * 下载文件
     *
     * @param response 返回请求
     * @param fileName 文件名
     * @return
     */
    @ResponseBody
    @RequestMapping(value = "/downloadFile", method = RequestMethod.GET)
    public String downloadFile(HttpServletResponse response, @RequestParam
    String fileName) {
        MinioClient minioClient = minIOService.getMinioClient();
        if (minioClient == null) {
            return "连接MinIO服务器失败";
        }
        return minIOService.downloadFile(minioClient, fileName, response) !=
        null ? "下载成功" : "下载失败";
    }
}

```

3、Service层

```

@Service
public class MaterialServiceImpl implements MaterialService {

    @Autowired
    private MaterialRepository materialRepository;
    @Autowired
    private UsersRepository usersRepository;

    /**

```

```

    * 上传文件（数据库操作）
    * @param fileName  文件名称
    * @param fileUrl   文件url
    * @param size      文件大小
    * @return
    */
    @Override
    public String upload(String fileName,String fileUrl,long size,String
user_name,String dpi,int category) {

        Materials materials = new Materials();
        int m_id = materialRepository.findMaxId()+1;

        String msize = this.setSize(size);

        materials.setId(m_id);
        materials.setMname(fileName);
        materials.setMurl(fileUrl);
        materials.setCategory(1);
        materials.setCreatetime(new Date());
        materials.setUpdatetime(new Date());
        materials.setMstatus("未删除");
        materials.setMsize(msize);
        materials.setAuthor(user_name);
        materials.setDpi(dpi);
        materials.setCategory(category);

        materialRepository.save(materials);

        return null;
    }

    /**
     * get所有图片信息
     * @return
     */
    @Override
    public List<BackToWebDto> getAll_photos() {
        List<Materials> materials_list = new ArrayList<Materials>();
        materials_list = materialRepository.findAllMaterial();

        List<BackToWebDto> dtolist = this.GetMaterialList(materials_list);

        return dtolist;
    }

    /**
     * 按名称模糊查找所有图片
     * @param name      模糊查找内容
     * @return
     */
    @Override
    public List<BackToWebDto> SelectByName(String name) throws
BoeBoardServiceException {
        List<Materials> materials_list = new ArrayList<Materials>();

```

```

        materials_list = materialRepository.SelectByName(name);
        if(materials_list.size() == 0){
            throw new BoeBoardServiceException("不存在");
        }
        List<BackToWebDto> dtolist = this.GetMaterialList(materials_list);

        return dtolist;
    }

    /**
     * 删除图片
     * @param fileName
     */
    @Override
    public void deleteFile(String fileName) {
        materialRepository.deleteByName(fileName);
    }

    /**
     * 修改文件名
     * @param old_name
     * @param new_name
     */
    @Override
    public void updateFile(String old_name, String new_name) {
        materialRepository.updateMaterial(old_name,new_name,new Date());
    }

    //文件大小转化
    public String setSize(long size) {
        //获取到的size为: 1705230
        int GB = 1024 * 1024 * 1024;//定义GB的计算常量
        int MB = 1024 * 1024;//定义MB的计算常量
        int KB = 1024;//定义KB的计算常量
        DecimalFormat df = new DecimalFormat("0.00");//格式化小数
        String resultSize = "";
        if (size / GB >= 1) {
            //如果当前Byte的值大于等于1GB
            resultSize = df.format(size / (float) GB) + "GB";
        } else if (size / MB >= 1) {
            //如果当前Byte的值大于等于1MB
            resultSize = df.format(size / (float) MB) + "MB";
        } else if (size / KB >= 1) {
            //如果当前Byte的值大于等于1KB
            resultSize = df.format(size / (float) KB) + "KB";
        } else {
            resultSize = size + "B ";
        }
        return resultSize;
    }

    public List<BackToWebDto> GetMaterialList(List<Materials> materialslist){
        List<BackToWebDto> dtolist = new ArrayList<>();

        materialslist.forEach(material ->{

```

```

        BackToWebDto dto = new BackToWebDto();
        dto.setMname(material.getMname());
        dto.setMurl(material.getMurl());
        dto.setMsize(material.getMsize());
        dto.setAuthor(material.getAuthor());
        dto.setUpdatetime(material.getUpdatetime());
        dto.setCreatetime(material.getCreatetime());
        dto.setMstatus(material.getMstatus());
        dto.setDpi(material.getDpi());
        dto.setCategory(material.getCategory());
        dtolist.add(dto);
    } );

    return dtolist;
}
}

```

4、Dto

(1) 传给前端的数据封装

用于前端请求获取单个或者多个素材时，将字段和值封装成dto，以列表返回

```

@Data
public class BackToWebDto implements Serializable {
    private String mname;           //素材名称
    private String murl;           //素材url
    private String msize;          //素材大小
    private String dpi;            //素材dpi
    private String author;         //上传的作者
    private Date updatetime;       //更新时间
    private Date createtime;       //创建的时间
    private String mstatus;        //素材状态
    private int category;          //素材种类
}

```

(2) 素材上传Dto

```

@Data
public class MaterialUploadDto implements Serializable {
    private String file_name;
    private String url;
}

```

十二、节目管理

1、数据库

p_id	bigint			<input checked="" type="checkbox"/>	<input type="checkbox"/>	🔑 1	节目id
pname	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>		节目名称
dpi	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>		节目分辨率
plength	int			<input type="checkbox"/>	<input type="checkbox"/>		节目时长
psize	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>		节目大小
pstatus	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>		节目状态
author	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>		节目作者
updatetime	datetime			<input type="checkbox"/>	<input type="checkbox"/>		节目更新时间
createtime	datetime			<input type="checkbox"/>	<input type="checkbox"/>		节目创建时间
materials	varchar	1023		<input type="checkbox"/>	<input type="checkbox"/>		素材(url+类型)

2、Controller层

实现功能：

- 新建节目
- get所有节目
- 删除节目
- 按input节目名模糊查找节目
- 按dpi查找节目
- 按状态查找节目
- 修改节目名称

```

@RestController
@RequestMapping("/programme")
public class ProgrammeController {

    @Autowired
    private ProgrammeService programmeService;

    /**
     * @param programmeDto    //web端新建节目
     * @param user_token      //用户token
     * @return
     */
    @ResponseBody
    @RequestMapping(value = "/add", method = RequestMethod.POST)
    public ResponseData addProgramme(@RequestBody ProgrammeDto
programmeDto,@RequestParam String user_token) {
        try {
            programmeService.AddProgramme(programmeDto,user_token);
            return new ResponseData(ExceptionMsg.SUCCESS,"添加成功");
        }catch (BoeBoardServiceException e){
            System.out.println(e);
        }
        return new ResponseData(ExceptionMsg.SUCCESS,"添加成功");
    }

    /**
     * get所有节目
     * @return
     */
    @ResponseBody
    @RequestMapping(value = "/getAll",method = RequestMethod.GET)
    public ResponseData GetAllProgramme(){
        List<GetProgrammesDto> result = programmeService.GetAllProgramme();
        return new ResponseData(ExceptionMsg.SUCCESS,result);
    }
}

```

```

}

/**
 * 删除节目
 * @param map
 *      {
 *          "p_name": "zzz"    //要删除的节目名
 *      }
 */
@ResponseBody
@RequestMapping(value = "/delete", method = RequestMethod.POST)
public ResponseData DeleteProgramme(@RequestBody Map map){
    String p_name = (String)map.get("p_name");
    programmeService.DeleteProgramme(p_name);

    return new ResponseData(ExceptionMsg.SUCCESS, "删除成功");
}

/**
 * 按input节目名模糊查找节目
 * @param map
 * @return
 */
@ResponseBody
@RequestMapping(value = "/search/byname", method = RequestMethod.GET)
public ResponseData SearchByName(@RequestBody Map map){

    String search_content = (String)map.get("search_content");
    List<GetProgrammesDto> result = new ArrayList<>();
    try {
        result = programmeService.SearchByName(search_content);
    } catch (BoeBoardServiceException e){
        return new ResponseData(ExceptionMsg.FAILED, e);
    }
    return new ResponseData(ExceptionMsg.SUCCESS, result);
}

/**
 * 按dpi查找节目
 * @param map
 * @return
 */
@ResponseBody
@RequestMapping(value = "/search/bydpi", method = RequestMethod.GET)
public ResponseData SearchByDpi(@RequestBody Map map){

    String search_content = (String)map.get("search_content");
    List<GetProgrammesDto> result = new ArrayList<>();
    try {
        result = programmeService.SearchByName(search_content);
    } catch (BoeBoardServiceException e){
        return new ResponseData(ExceptionMsg.FAILED, e);
    }
    return new ResponseData(ExceptionMsg.SUCCESS, result);
}

```

```

/**
 * 按状态查找节目
 * @param map
 * @return
 */
@ResponseBody
@RequestMapping(value = "/search/bystatus",method = RequestMethod.GET)
public ResponseData SearchByStatus(@RequestBody Map map){

    String search_content = (String)map.get("search_content");
    List<GetProgrammesDto> result = new ArrayList<>();
    try {
        result = programmeService.SearchByName(search_content);
    }catch (BoeBoardServiceException e){
        return new ResponseData(ExceptionMsg.FAILED,e);
    }
    return new ResponseData(ExceptionMsg.SUCCESS,result);
}

/**
 * 修改节目名称
 * @param map
 * @return
 */
@ResponseBody
@RequestMapping(value = "/rename",method = RequestMethod.POST)
public ResponseData RenameProgramme(@RequestBody Map map){
    String old_name = (String)map.get("old_name");
    String new_name = (String)map.get("new_name");
    programmeService.RenameProgramme(old_name,new_name);

    return new ResponseData(ExceptionMsg.SUCCESS,"修改成功");
}
}

```

3、Service层

```

@Service
public class ProgrammeServiceImpl implements ProgrammeService {

    @Autowired
    private UsersRepository usersRepository;
    @Autowired
    private ProgrammeRepository programmeRepository;
    @Autowired
    private MaterialRepository materialRepository;

    double total_size = 0;

    //新增节目
    @Override

```

```

    public void AddProgramme(ProgrammeDto programmeDto,String user_token) throws
    BoeBoardServiceException {

        HomeServiceImpl homeServiceimpl = new HomeServiceImpl();
        MaterialServiceImpl materialService = new MaterialServiceImpl();
        String id = (String) StpUtil.getLoginIdByToken(user_token);
        String user_name =
        usersRepository.findById(Long.parseLong(id)).getUsername();
        //      String user_name = programmeDto.getUser_name();
        int p_id = programmeRepository.findMaxId()+1;

        //素材使用的materials
        List<MaterialsDto> materialsDtos = programmeDto.getMaterials();
        //获取素材大小
        materialsDtos.forEach(materialsDto -> {
            String mname = materialsDto.getMname();
            String single_msize = materialRepository.GetSizeByName(mname);
            String extName = mname.substring(mname.indexOf(".") + 1);
            double material_size = homeServiceimpl.GetSingleSize(single_msize);
            total_size+=material_size;

        });

        String msize = materialService.setSize((long)total_size);

        String p_name = programmeDto.getP_name();
        if(programmeRepository.findByName(p_name) != 0){
            throw new BoeBoardServiceException("节目名已存在，勿重复添加");
        }

        Programme programme = new Programme();
        programme.setId(p_id);
        programme.setPname(programmeDto.getP_name());
        programme.setDpi(programmeDto.getDpi());
        programme.setPlength(programmeDto.getPlength());
        programme.setPsize(programmeDto.getPsize());
        programme.setPstatus("未使用");
        programme.setAuthor(user_name);
        programme.setCreatetime(new Date());
        programme.setUpdatetime(new Date());
        programme.setMaterials(JSON.toJSONString(materialsDtos));
        programme.setPsize(msize);
        programmeRepository.save(programme);

    }

    //获取所有节目
    @Override
    public List<GetProgrammesDto> GetAllProgramme() {
        List<Programme> list = programmeRepository.GetAll();

        List<GetProgrammesDto> result = this.GetProgramme(list);

        return result;
    }

```



```

//删除节目
@Override
public void DeleteProgramme(String p_name) {

    programmeRepository.deleteProgrammeByName(p_name);

}

//模糊查找节目
@Override
public List<GetProgrammesDto> SearchByName(String content) throws
BoeBoardServiceException{
    List<Programme> list = programmeRepository.SelectByName(content);
    if(list.size() == 0){
        throw new BoeBoardServiceException("查找失败");
    }
    List<GetProgrammesDto> result = this.GetProgramme(list);

    return result;
}

//按dpi查找
@Override
public List<GetProgrammesDto> SearchByDpi(String content) {
    List<Programme> list = programmeRepository.SelectByDpi(content);
    if(list.size() == 0){
        throw new BoeBoardServiceException("查找失败");
    }
    List<GetProgrammesDto> result = this.GetProgramme(list);

    return result;
}

//按状态查找
@Override
public List<GetProgrammesDto> SearchByStatus(String content) {
    List<Programme> list = programmeRepository.SelectByStatus(content);
    if(list.size() == 0){
        throw new BoeBoardServiceException("查找失败");
    }
    List<GetProgrammesDto> result = this.GetProgramme(list);

    return result;
}

@Override
public void RenameProgramme(String old_name,String new_name) {
    programmeRepository.Rename(old_name,new_name);
}

//获取对应的program-list
public List<GetProgrammesDto> GetProgramme(List<Programme> list){
    List<GetProgrammesDto> dtolist = new ArrayList<>();

```

```

        list.forEach(programme -> {
            GetProgrammesDto dto = new GetProgrammesDto();
            List<MaterialsDto> materialsDtos =
(List)JSON.parse(programme.getMaterials());
            dto.setP_id(programme.getId());
            dto.setPname(programme.getPname());
            dto.setDpi(programme.getDpi());
            dto.setPlength(programme.getPlength());
            dto.setPsize(programme.getPsize());
            dto.setPstatus(programme.getPstatus());
            dto.setAuthor(programme.getAuthor());
            dto.setCreatetime(programme.getCreatetime());
            dto.setUpdatetime(programme.getUpdatetime());
            dto.setMaterials(materialsDtos);
            dtolist.add(dto);
        });
//        String result = JSON.toJSONString(dtolist);
        return dtolist;
    }
}

```

4、Dto

(1) 数据库映射的节目封装

```

@Data
public class GetProgrammesDto implements Serializable {
    private long p_id;
    private String pname;
    private String dpi;
    private int plength;
    private String psize;
    private String pstatus;
    private String author;
    private Date updatetime;
    private Date createtime;
    private List<MaterialsDto> materials;
}

```

(2) 节目中引用的单个素材封装

```

@Data
public class MaterialsDto implements Serializable {
    private String mname;           //素材名称
    private String murl;           //素材文件访问url
    private int category;          //素材类型
}

```

(3) 传给前端的节目封装


```

public class ProgrammedTo implements Serializable {
    private String p_name;           //节目名称
    private String dpi;              //节目分辨率
    private List<MaterialsDto> materials; //素材url列表
    private int plength;             //节目时长
    private String psize;            //素材大小
}

```

十三、公告管理

1、数据库

名	类型	长度	小数点	不是 null	虚拟	键	注释
a_id	bigint			<input checked="" type="checkbox"/>	<input type="checkbox"/>	 1	公告id
content	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>		公告内容
text_color	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>		字体颜色
text_size	int			<input type="checkbox"/>	<input type="checkbox"/>		字体大小
background_color	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>		背景颜色
start_time	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>		开始时间
finish_time	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>		结束时间
astatus	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>		公告状态
author	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>		公告作者
createtime	datetime			<input type="checkbox"/>	<input type="checkbox"/>		公告创建时间

2、Controller层

实现功能：

- 新增一个公告
- 删除公告
- 获取所有公告

```

@RestController
@RequestMapping("/announce")
public class AnnounceController {

    @Autowired
    private AnnounceService announceService;

    /**
     * 新增一个公告
     * web端返回的数据结构
     * {
     *     "content": "Helloworld", //公告内容
     *     "text_color": "#FFC0CB", //字体颜色
     *     "text_size": 12, //字体大小
     *     "background_color": "#000000" //背景颜色
     *     "start_time": "2022-06-09 20:13:24" //开始时间
     *     "finish_time": "2022-06-10 20:13:24" //结束时间
     * }
     * @param dto
     * @param user_token //用户token
     */
    @ResponseBody
    @RequestMapping(value = "/add", method = RequestMethod.POST)
    public ResponseData AddAnnounce(@RequestBody GetAnnouncedTo
    dto, @RequestParam String user_token) {

```

```

        announceService.AddAnnounce(dto,user_token);

        return new ResponseData(ExceptionMsg.SUCCESS,"添加成功");
    }

    //删除公告
    @ResponseBody
    @RequestMapping(value = "/delete",method = RequestMethod.POST)
    public ResponseData DeleteAnnounce(@RequestBody Map map){
        String content = (String)map.get("content");

        announceService.DeleteAnnounce(content);

        return new ResponseData(ExceptionMsg.SUCCESS,"删除成功");
    }

    /**
     * 获取所有公告
     * @return
     */
    @ResponseBody
    @RequestMapping(value = "/getall",method = RequestMethod.GET)
    public ResponseData GetAllAnnounce(){
        List<BackAnnounceDto> dtos = announceService.GetAllAnnounces();

        return new ResponseData(ExceptionMsg.SUCCESS,dtos);
    }
}

```

3、Service层

```

@Service
public class AnnounceServiceImpl implements AnnounceService {

    @Autowired
    private AnnounceRepository announceRepository;
    @Autowired
    private UsersRepository usersRepository;
    @Autowired
    private AmqpTemplate mqService;//mq

    //增加公告
    @Override
    public void AddAnnounce(GetAnnouncedto dto,String user_token) {

        String id = (String) StpUtil.getLoginIdByToken(user_token);
        String user_name =
        usersRepository.findById(Long.parseLong(id)).getUsername();

        //保存到数据库
        Announce announce = new Announce();
        int a_id = announceRepository.findMaxId()+1;
    }
}

```

```

        announce.setId(a_id);
        announce.setContent(dto.getContent());
        announce.setText_color(dto.getText_color());
        announce.setText_size(dto.getText_size());
        announce.setBackground_color(dto.getBackground_color());
        announce.setAstatus("公告中");
        announce.setAuthor(user_name);
        announce.setCreatetime(new Date());

        announceRepository.save(announce);

        //通过消息队列发送消息给AndroidStudio端
        MqMessage msg = new MqMessage(MqMessage.CATEGORY_ANNOUNCE);
        msg.appendContent("content", dto.getContent());
        msg.appendContent("text_color", dto.getText_color());
        msg.appendContent("text_size", dto.getText_size());
        msg.appendContent("background_color", dto.getBackground_color());
        msg.appendContent("start_time", dto.getStart_time());
        msg.appendContent("finish_time", dto.getFinish_time());
        mqService.convertAndSend(Constants.QUE_ANNOUNCE, msg.stringify());

    }

    //删除公告
    @Override
    public void DeleteAnnounce(String content) {
        announceRepository.deleteByName(content);
    }
}

```

4、Dto

(1) 前端公告列表

```

@Data
public class BackAnnouncedto implements Serializable {
    private String content;
    private String start_time;
    private String astatus;
    private String author;
    private Date createtime;
}

```

(2) 前端生成公告

```

@Data
public class GetAnnouncedto implements Serializable {
    private String content;
    private String text_color;
    private int text_size;
    private String background_color;
    private String start_time;
    private String finish_time;
}

```

5、定时器任务

每隔3秒执行任务，根据公告结束时间刷新数据库表公告状态。

```

@Configuration          //1.主要用于标记配置类，兼备Component的效果。
@EnableScheduling        // 2.开启定时任务
public class SaticScheduleTask {

    @Autowired
    private AnnounceRepository announceRepository;

    //3.添加定时任务
    @Scheduled(cron = "0/5 * * * * ?")
    //或直接指定时间间隔，例如：5秒
    //@Scheduled(fixedRate=5000)
    private void updateAnnounce() {
        TimeHandler timeHandler = new TimeHandler();
        List<Announce> announceList = announceRepository.GetAllAnnounces();

        announceList.forEach(announce -> {
            String astatus = announce.getAstatus();
            if(!astatus.equals("发布结束")) {
                String start_time = announce.getStart_time();
                String finish_time = announce.getFinish_time();
                Date date_start = new Date();
                Date date_finish = new Date();
                Date now = new Date();
                try {
                    date_start = timeHandler.StringToDate(start_time);
                    date_finish = timeHandler.StringToDate(finish_time);
                } catch (ParseException e) {
                    e.printStackTrace();
                }
                int compareToStart = timeHandler.CompareTime(now, date_start);
                int compareToFinish = timeHandler.CompareTime(now, date_finish);

                if (compareToStart < 0) {
                    astatus = "待发布";
                }
                if (compareToStart > 0 && compareToFinish < 0) {
                    astatus = "发布中";
                }
                if (compareToFinish > 0) {
                    astatus = "发布结束";
                }
            }
        });
    }
}

```

```
        announceRepository.updateStatus(astatus, announce.getId());
    }
}
}
```