一、创建项目，上传至github

1.使用的工具版本

> jdk 8

二、数据库

1. 数据库的设计

```
/*
Navicat MySQL Data Transfer

Source Server         : docker_mysql
Source Server Version : 50732
Source Host           : 192.168.30.133:3306
Source Database       : plusblog

Target Server Type    : MYSQL
Target Server Version : 50732
File Encoding         : 65001

Date: 2021-02-14 13:07:33
*/

SET FOREIGN_KEY_CHECKS=0;

-- ----------------------------
-- Table structure for tb_article
-- ----------------------------
DROP TABLE IF EXISTS `tb_article`;
CREATE TABLE `tb_article` (
  `id` varchar(255) NOT NULL COMMENT 'ID',
  `title` varchar(256) NOT NULL COMMENT '标题',
  `user_id` varchar(255) NOT NULL COMMENT '用户ID',
  `user_avatar` varchar(1024) DEFAULT NULL COMMENT '用户头像',
  `user_name` varchar(255) DEFAULT NULL COMMENT '用户昵称',
  `category_id` varchar(255) NOT NULL COMMENT '分类ID',
  `content` mediumtext NOT NULL COMMENT '文章内容',
  `type` varchar(1) NOT NULL COMMENT '类型（0表示富文本，1表示markdown）',
  `state` varchar(1) NOT NULL COMMENT '状态（0表示已发布，1表示草稿，2表示删除）',
  `summary` text NOT NULL COMMENT '摘要',
  `labels` varchar(128) NOT NULL COMMENT '标签',
  `view_count` int(11) NOT NULL DEFAULT '0' COMMENT '阅读数量',
  `create_time` datetime NOT NULL COMMENT '发布时间',
  `update_time` datetime NOT NULL COMMENT '更新时间',
  `is_delete` int(2) DEFAULT '0' COMMENT '(0表示不删除,1表示删除)',
  `cover` varchar(1024) DEFAULT NULL COMMENT '封面',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- ----------------------------
```

```sql
-- Records of tb_article
-- ----------------------------

-- ----------------------------
-- Table structure for tb_category
-- ----------------------------
DROP TABLE IF EXISTS `tb_category`;
CREATE TABLE `tb_category` (
  `id` varchar(255) NOT NULL COMMENT 'ID',
  `name` varchar(64) NOT NULL COMMENT '分类名称',
  `pinyin` varchar(128) NOT NULL COMMENT '拼音',
  `description` text NOT NULL COMMENT '描述',
  `order` int(11) NOT NULL COMMENT '顺序',
  `status` varchar(1) NOT NULL COMMENT '状态：0表示不使用，1表示正常',
  `create_time` datetime NOT NULL COMMENT '创建时间',
  `update_time` datetime NOT NULL COMMENT '更新时间',
  `is_delete` int(2) NOT NULL DEFAULT '0' COMMENT '(0表示不删除，1表示删除)',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- ----------------------------
-- Records of tb_category
-- ----------------------------

-- ----------------------------
-- Table structure for tb_comment
-- ----------------------------
DROP TABLE IF EXISTS `tb_comment`;
CREATE TABLE `tb_comment` (
  `id` varchar(255) NOT NULL COMMENT 'ID',
  `parent_content` text COMMENT '父内容',
  `article_id` varchar(255) NOT NULL COMMENT '文章ID',
  `content` text NOT NULL COMMENT '评论内容',
  `user_id` varchar(255) NOT NULL COMMENT '评论用户的ID',
  `user_avatar` varchar(1024) DEFAULT NULL COMMENT '评论用户的头像',
  `user_name` varchar(255) DEFAULT NULL COMMENT '评论用户的名称',
  `state` varchar(1) NOT NULL COMMENT '状态（0表示删除，1表示正常）',
  `create_time` datetime NOT NULL COMMENT '创建时间',
  `update_time` datetime NOT NULL COMMENT '更新时间',
  `is_delete` int(2) NOT NULL DEFAULT '0' COMMENT '(0表示不删除，1表示删除)',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- ----------------------------
-- Records of tb_comment
-- ----------------------------

-- ----------------------------
-- Table structure for tb_daily_view_count
-- ----------------------------
DROP TABLE IF EXISTS `tb_daily_view_count`;
CREATE TABLE `tb_daily_view_count` (
  `id` varchar(255) NOT NULL COMMENT 'ID',
  `view_count` int(11) NOT NULL DEFAULT '0' COMMENT '每天浏览量',
  `create_time` datetime NOT NULL COMMENT '创建时间',
  `update_time` datetime NOT NULL COMMENT '更新时间',
  `is_delete` int(2) NOT NULL DEFAULT '0' COMMENT '(0表示不删除，1表示删除)',
  PRIMARY KEY (`id`)
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- ----------------------------
-- Records of tb_daily_view_count
-- ----------------------------

-- ----------------------------
-- Table structure for tb_friend_link
-- ----------------------------
DROP TABLE IF EXISTS `tb_friend_link`;
CREATE TABLE `tb_friend_link` (
  `id` varchar(255) NOT NULL COMMENT 'ID',
  `name` varchar(64) NOT NULL COMMENT '友情链接名称',
  `logo` varchar(1024) NOT NULL COMMENT '友情链接logo',
  `url` varchar(1024) NOT NULL COMMENT '友情链接',
  `order` int(11) NOT NULL DEFAULT '0' COMMENT '顺序',
  `state` varchar(1) NOT NULL COMMENT '友情链接状态:0表示不可用，1表示正常',
  `create_time` datetime NOT NULL COMMENT '创建时间',
  `update_time` datetime NOT NULL COMMENT '更新时间',
  `is_delete` int(2) NOT NULL DEFAULT '0' COMMENT '(0表示不删除，1表示删除)',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- ----------------------------
-- Records of tb_friend_link
-- ----------------------------

-- ----------------------------
-- Table structure for tb_images
-- ----------------------------
DROP TABLE IF EXISTS `tb_images`;
CREATE TABLE `tb_images` (
  `id` varchar(255) NOT NULL COMMENT 'ID',
  `user_id` varchar(255) NOT NULL COMMENT '用户ID',
  `name` varchar(255) NOT NULL COMMENT '图片名称',
  `url` varchar(1024) NOT NULL COMMENT '路径',
  `content_type` varchar(255) NOT NULL COMMENT '图片类型',
  `state` varchar(1) NOT NULL COMMENT '状态（0表示删除，1表正常）',
  `create_time` datetime NOT NULL COMMENT '创建时间',
  `update_time` datetime NOT NULL COMMENT '更新时间',
  `is_delete` int(2) NOT NULL DEFAULT '0' COMMENT '(0表示不删除，1表示删除)',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- ----------------------------
-- Records of tb_images
-- ----------------------------

-- ----------------------------
-- Table structure for tb_img_looper
-- ----------------------------
DROP TABLE IF EXISTS `tb_img_looper`;
CREATE TABLE `tb_img_looper` (
  `id` varchar(255) NOT NULL COMMENT 'ID',
  `title` varchar(128) NOT NULL COMMENT '轮播图标题',
  `order` int(11) NOT NULL DEFAULT '0' COMMENT '顺序',
  `state` varchar(1) NOT NULL COMMENT '状态：0表示不可用，1表示正常',
  `target_url` varchar(1024) DEFAULT NULL COMMENT '目标URL',
```

```
  `image_url` varchar(2014) NOT NULL COMMENT '图片地址',
  `create_time` datetime NOT NULL COMMENT '创建时间',
  `update_time` datetime NOT NULL COMMENT '更新时间',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- ----------------------------
-- Records of tb_img_looper
-- ----------------------------

-- ----------------------------
-- Table structure for tb_labels
-- ----------------------------
DROP TABLE IF EXISTS `tb_labels`;
CREATE TABLE `tb_labels` (
  `id` varchar(255) NOT NULL COMMENT 'ID',
  `name` varchar(32) NOT NULL COMMENT '标签名称',
  `count` int(11) NOT NULL DEFAULT '0' COMMENT '数量',
  `create_time` datetime NOT NULL COMMENT '创建时间',
  `update_time` datetime NOT NULL COMMENT '更新时间',
  `is_delete` int(2) NOT NULL DEFAULT '0' COMMENT '(0表示不删除，1表示删除)',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- ----------------------------
-- Records of tb_labels
-- ----------------------------
INSERT INTO `tb_labels` VALUES ('3480515b55671492541 33dac6ecdd14b', '修改测试4',
'0', '2021-02-04 21:42:14', '2021-02-04 21:45:23', '0');

-- ----------------------------
-- Table structure for tb_refresh_token
-- ----------------------------
DROP TABLE IF EXISTS `tb_refresh_token`;
CREATE TABLE `tb_refresh_token` (
  `id` varchar(255) NOT NULL,
  `refresh_token` text NOT NULL COMMENT 'token',
  `user_id` varchar(255) NOT NULL COMMENT '用户Id',
  `token_key` varchar(255) NOT NULL COMMENT 'token_key ，存放在redis中需要的用到的
key',
  `create_time` datetime NOT NULL COMMENT '发布时间',
  `update_time` datetime NOT NULL COMMENT '更新时间',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- ----------------------------
-- Records of tb_refresh_token
-- ----------------------------

-- ----------------------------
-- Table structure for tb_settings
-- ----------------------------
DROP TABLE IF EXISTS `tb_settings`;
CREATE TABLE `tb_settings` (
  `id` varchar(255) NOT NULL COMMENT 'ID',
  `key` varchar(32) NOT NULL COMMENT '键',
  `value` varchar(512) NOT NULL COMMENT '值',
  `create_time` datetime NOT NULL COMMENT '创建时间',
```

```
  `update_time` datetime NOT NULL COMMENT '更新时间',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- ----------------------------
-- Records of tb_settings
-- ----------------------------
INSERT INTO `tb_settings` VALUES ('b78f3d06276803532bc79c2f68a74804',
'has_manager_init_state', '1', '2021-02-04 22:48:02', '2021-02-04 22:48:02');

-- ----------------------------
-- Table structure for tb_user
-- ----------------------------
DROP TABLE IF EXISTS `tb_user`;
CREATE TABLE `tb_user` (
  `id` varchar(255) NOT NULL COMMENT 'ID',
  `user_name` varchar(32) NOT NULL COMMENT '用户名',
  `password` varchar(255) NOT NULL COMMENT '密码',
  `roles` varchar(100) NOT NULL COMMENT '角色',
  `avatar` varchar(1024) NOT NULL COMMENT '头像地址',
  `email` varchar(100) DEFAULT NULL COMMENT '邮箱地址',
  `sign` varchar(100) DEFAULT NULL COMMENT '签名',
  `state` varchar(1) NOT NULL DEFAULT '1' COMMENT '状态：0表示删除，1表示正常',
  `reg_ip` varchar(32) NOT NULL COMMENT '注册ip',
  `login_ip` varchar(32) NOT NULL COMMENT '登录Ip',
  `create_time` datetime NOT NULL COMMENT '创建时间',
  `update_time` datetime NOT NULL COMMENT '更新时间',
  `is_delete` int(2) NOT NULL DEFAULT '0' COMMENT '(0表示不删除，1表示删除)',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- ----------------------------
-- Records of tb_user
-- ----------------------------
INSERT INTO `tb_user` VALUES ('7406f5e79903bae9a4010ee9e03b7e90', 'admin',
'$2a$10$IyLfBsL6JH2mI9ljiM4tJ.jj5ufIV56394LgPL7mgsBC.L1o.qFU.', 'role_admin',
'https://gimg2.baidu.com/image_search/src=http%3A%2F%2Fc-
ssl.duitang.com%2Fuploads%2Fitem%2F202007%2F01%2F20200701063944_5VaBk.thumb.1000
_0.jpeg&refer=http%3A%2F%2Fc-
ssl.duitang.com&app=2002&size=f9999,10000&q=a80&n=0&g=0n&fmt=jpeg?
sec=1614571386&t=2e68974a8d276943307d75ea32457e3d', '1005777562@qq.com', null,
'1', '0:0:0:0:0:0:0:1', '0:0:0:0:0:0:0:1', '2021-02-04 22:48:02', '2021-02-04
22:48:02', '0');
```

2. 导入mp、逆向工程、逆向工程模板、mysql驱动依赖

```
<!--mybatis plus -->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>3.3.1.tmp</version>
        </dependency>
        <!--构造器-->
        <dependency>
            <groupId>com.baomidou</groupId>
```

```
            <artifactId>mybatis-plus-generator</artifactId>
            <version>3.3.1.tmp</version>
        </dependency>
        <!--逆向生成的模板-->
        <dependency>
            <groupId>org.apache.velocity</groupId>
            <artifactId>velocity</artifactId>
            <version>1.7</version>
        </dependency>
        <!--mysql连接工具-->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.22</version>
        </dependency>
```

3.进行配置数据库连接

```
spring:
  application:
    name: halfmoon
  datasource:
    username: root
    password: root
    url: jdbc:mysql://192.168.30.133:3306/halfmoon?
useUnicode=true&characterEncoding=UTF-
8&serverTimezone=Asia/Shanghai&useSSL=false&characterEncoding=utf-8
    driver-class-name: com.mysql.cj.jdbc.Driver
server:
  port: 8078
```

4.编写mp配置类

```
package com.oldbai.halfmoon.config;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.transaction.annotation.EnableTransactionManagement;

@Configuration
@EnableTransactionManagement
@MapperScan("com.oldbai.halfmoon.mapper")
public class MybatisPlusConfig {

}
```

5.使用逆向工程工具

```
package com.oldbai.halfmoon;

import com.baomidou.mybatisplus.annotation.DbType;
```

```
import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.generator.AutoGenerator;
import com.baomidou.mybatisplus.generator.config.DataSourceConfig;
import com.baomidou.mybatisplus.generator.config.GlobalConfig;
import com.baomidou.mybatisplus.generator.config.PackageConfig;
import com.baomidou.mybatisplus.generator.config.StrategyConfig;
import com.baomidou.mybatisplus.generator.config.rules.DateType;
import com.baomidou.mybatisplus.generator.config.rules.NamingStrategy;
import org.junit.Test;

public class CodeGenerator {

    @Test
    public void run() {

        // 1、创建代码生成器
        AutoGenerator mpg = new AutoGenerator();

        // 2、全局配置
        GlobalConfig gc = new GlobalConfig();
        String projectPath = System.getProperty("user.dir");
        gc.setOutputDir("G:\\MyJava\\HalfMoon\\halfmoon" + "/src/main/java");
        gc.setAuthor("oldbai");
        gc.setOpen(false); //生成后是否打开资源管理器
        gc.setFileOverride(false); //重新生成时文件是否覆盖
        gc.setServiceName("%sService"); //去掉Service接口的首字母I
        gc.setIdType(IdType.ASSIGN_UUID); //主键策略
        gc.setDateType(DateType.ONLY_DATE);//定义生成的实体类中日期类型
        gc.setSwagger2(true);//开启Swagger2模式

        mpg.setGlobalConfig(gc);

        // 3、数据源配置
        DataSourceConfig dsc = new DataSourceConfig();
        dsc.setUrl("jdbc:mysql://192.168.30.133:3306/halfmoon?
serverTimezone=GMT%2B8");
        dsc.setDriverName("com.mysql.cj.jdbc.Driver");
        dsc.setUsername("root");
        dsc.setPassword("root");
        dsc.setDbType(DbType.MYSQL);
        mpg.setDataSource(dsc);

        // 4、包配置
        PackageConfig pc = new PackageConfig();
        pc.setParent("com.oldbai");
        pc.setModuleName("halfmoon"); //模块名
        pc.setController("controller");
        pc.setEntity("entity");
        pc.setService("service");
        pc.setMapper("mapper");
        mpg.setPackageInfo(pc);

        // 5、策略配置
        StrategyConfig strategy = new StrategyConfig();
//        strategy.setInclude("b_comment");
        strategy.setNaming(NamingStrategy.underline_to_camel);//数据库表映射到实体
的命名策略
//        strategy.setTablePrefix(pc.getModuleName() + "_"); //生成实体时去掉表前缀
```

```
//        strategy.setColumnNaming(NamingStrategy.underline_to_camel);
        strategy.setTablePrefix( "tb_");

        strategy.setColumnNaming(NamingStrategy.underline_to_camel);//数据库表字段映射到实体的命名策略
        strategy.setEntityLombokModel(true); // lombok 模型 @Accessors(chain = true) setter链式操作

        strategy.setRestControllerStyle(true); //restful api风格控制器
        strategy.setControllerMappingHyphenStyle(true); //url中驼峰转连字符

        mpg.setStrategy(strategy);


        // 6、执行
        mpg.execute();
    }
}
```

6.导入swagger 依赖

```
<!--swaggui-->
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger-ui</artifactId>
            <version>2.9.2</version>
        </dependency>
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger2</artifactId>
            <version>2.9.2</version>
        </dependency>
```

7.编写swagger 配置类

```
package com.oldbai.halfmoon.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.builders.ApiInfoBuilder;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

/**
 * @author 老白
 */
@Configuration
@EnableSwagger2
public class SwaggerConfig {
```

```
    //版本
    public static final String VERSION = "1.0.0";

    /**
     * 门户api，接口前缀：portal
     *
     * @return
     */
    @Bean
    public Docket portalApi() {
        return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(portalApiInfo())
                .select()
.apis(RequestHandlerSelectors.basePackage("com.oldbai.plusblog.controller.portal
"))
                // 可以根据url路径设置哪些请求加入文档，忽略哪些请求
                .paths(PathSelectors.any())
                .build()
                .groupName("前端门户");
    }

    private ApiInfo portalApiInfo() {
        return new ApiInfoBuilder()
                //设置文档的标题
                .title("月半湾博客系统门户接口文档")
                // 设置文档的描述
                .description("门户接口文档")
                // 设置文档的版本信息-> 1.0.0 Version information
                .version(VERSION)
                .build();
    }


    /**
     * 管理中心api，接口前缀：admin
     *
     * @return
     */
    @Bean
    public Docket adminApi() {
        return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(adminApiInfo())
                .select()
.apis(RequestHandlerSelectors.basePackage("com.oldbai.plusblog.controller.admin"
))
                // 可以根据url路径设置哪些请求加入文档，忽略哪些请求
                .paths(PathSelectors.any())
                .build()
                .groupName("管理中心");
    }


    private ApiInfo adminApiInfo() {
        return new ApiInfoBuilder()
                //设置文档的标题
                .title("月半湾管理中心接口文档")
```

```
                // 设置文档的描述
                .description("管理中心接口")
                // 设置文档的版本信息-> 1.0.0 Version information
                .version(VERSION)
                .build();
    }


    @Bean
    public Docket UserApi() {
        return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(userApiInfo())
                .select()

.apis(RequestHandlerSelectors.basePackage("com.oldbai.plusblog.controller.user")
)
                // 可以根据url路径设置哪些请求加入文档，忽略哪些请求
                .paths(PathSelectors.any())
                .build()
                .groupName("用户中心");
    }

    private ApiInfo userApiInfo() {
        return new ApiInfoBuilder()
                //设置文档的标题
                .title("月半湾博客系统用户接口")
                // 设置文档的描述
                .description("用户接口的接口")
                // 设置文档的版本信息-> 1.0.0 Version information
                .version(VERSION)
                .build();
    }

}
```

8.配制逻辑删除

```
mybatis-plus:
  global-config:
    db-config:
      logic-delete-value: 1
      logic-not-delete-value: 0
      logic-delete-field: isDelete
```

9.自动填充日期

```
package com.oldbai.halfmoon.config;

import com.baomidou.mybatisplus.core.handlers.MetaObjectHandler;
import lombok.extern.slf4j.Slf4j;
import org.apache.ibatis.reflection.MetaObject;
import org.springframework.stereotype.Component;

import java.util.Date;
```

```java
/**
 * @author 老白
 */
@Slf4j
@Component
public class MyMetaObjectHandler implements MetaObjectHandler {
    @Override
    public void insertFill(MetaObject metaObject) {
        log.info("start insert fill ....");
        this.setFieldValByName("createTime", new Date(), metaObject);
        this.setFieldValByName("updateTime", new Date(), metaObject);

    }

    @Override
    public void updateFill(MetaObject metaObject) {
        log.info("start update fill ....");
        // 起始版本 3.3.0(推荐)
        this.setFieldValByName("updateTime", new Date(), metaObject);
    }
}
```

三、Redis

1.导入依赖

```xml
<!--        连接池-->
        <!-- spirng2.X集成redis 所需 common-pool2-->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-pool2</artifactId>
            <version>2.8.0</version>
        </dependency>
        <!-- redis-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
            <version>2.3.2.RELEASE</version>
        </dependency>
```

2.配置连接redis

```
# redis连接
spring.redis.host=192.168.30.133
spring.redis.port=6379
spring.redis.database= 1
spring.redis.timeout=1800000
spring.redis.lettuce.pool.max-active=20
spring.redis.lettuce.pool.max-wait=-1
#最大阻塞等待时间(负数表示没限制)
spring.redis.lettuce.pool.max-idle=5
spring.redis.lettuce.pool.min-idle=0
```

3.redis配置类

```
package com.oldbai.halfmoon.config;

import com.fasterxml.jackson.annotation.JsonAutoDetect;
import com.fasterxml.jackson.annotation.PropertyAccessor;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.oldbai.halfmoon.util.RedisUtil;
import org.springframework.cache.annotation.CachingConfigurerSupport;
import org.springframework.cache.annotation.EnableCaching;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.cache.RedisCacheConfiguration;
import org.springframework.data.redis.cache.RedisCacheManager;
import org.springframework.data.redis.cache.RedisCacheWriter;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;
import org.springframework.data.redis.serializer.RedisSerializationContext;
import org.springframework.data.redis.serializer.RedisSerializer;
import org.springframework.data.redis.serializer.StringRedisSerializer;

import java.time.Duration;


/**
 * redis配制文件
 *
 * @author 老白
 */
@EnableCaching
@Configuration
public class RedisConfig extends CachingConfigurerSupport {

    public static final String REDIS_KEY_DATABASE = "root";

    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory
redisConnectionFactory) {
        RedisSerializer<Object> serializer = redisSerializer();
        RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();
        redisTemplate.setConnectionFactory(redisConnectionFactory);
        redisTemplate.setKeySerializer(new StringRedisSerializer());
        // 设置 redisTemplate 的序列化器
        redisTemplate.setValueSerializer(serializer);
```

```
        redisTemplate.setHashKeySerializer(new StringRedisSerializer());
        redisTemplate.setHashValueSerializer(serializer);
        redisTemplate.afterPropertiesSet();
        return redisTemplate;
    }

    @Bean
    public RedisSerializer<Object> redisSerializer() {
        //创建JSON序列化器
        Jackson2JsonRedisSerializer<Object> serializer = new
Jackson2JsonRedisSerializer<>(Object.class);
        ObjectMapper objectMapper = new ObjectMapper();
        objectMapper.setVisibility(PropertyAccessor.ALL,
JsonAutoDetect.Visibility.ANY);
        objectMapper.enableDefaultTyping(ObjectMapper.DefaultTyping.NON_FINAL);
        serializer.setObjectMapper(objectMapper);
        return serializer;
    }

    @Bean
    public RedisCacheManager redisCacheManager(RedisConnectionFactory
redisConnectionFactory) {
        RedisCacheWriter redisCacheWriter =
RedisCacheWriter.nonLockingRedisCacheWriter(redisConnectionFactory);
        //设置Redis缓存有效期为1天
        RedisCacheConfiguration redisCacheConfiguration =
RedisCacheConfiguration.defaultCacheConfig()
                .serializeValuesWith(RedisSerializationContext.SerializationPair
                        .fromSerializer(redisSerializer()))
                .entryTtl(Duration.ofDays(1));
        return new RedisCacheManager(redisCacheWriter, redisCacheConfiguration);
    }

    /**
     * Redis工具类注入bean
     */
    @Bean
    public RedisUtil createRedisUtil() {
        return new RedisUtil();
    }

}
```

4.redis工具类

```
package com.oldbai.halfmoon.util;

import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Component;
import org.springframework.util.CollectionUtils;

import javax.annotation.Resource;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.TimeUnit;
```

```java
/**
 * redis 工具类
 *
 * @author 老白
 */
@Component
public class RedisUtil {
    @Resource
    private RedisTemplate redisTemplate;

    public void setRedisTemplate(RedisTemplate<String, Object> redisTemplate) {
        this.redisTemplate = redisTemplate;
    }

    /**
     * 指定缓存失效时间
     *
     * @param key  键
     * @param time 时间(秒)
     * @return
     */
    private boolean expire(String key, long time) {
        try {
            if (time > 0) {
                redisTemplate.expire(key, time, TimeUnit.SECONDS);
            }
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 根据key 获取过期时间
     *
     * @param key 键 不能为null
     * @return 时间(秒) 返回0代表为永久有效
     */
    public long getExpire(String key) {
        return redisTemplate.getExpire(key, TimeUnit.SECONDS);
    }

    /**
     * 判断key是否存在
     *
     * @param key 键
     * @return true 存在 false不存在
     */
    public boolean hasKey(String key) {
        try {
            return redisTemplate.hasKey(key);
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }
```

```
    /**
     * 删除缓存
     *
     * @param key 可以传一个值 或多个
     */
    @SuppressWarnings("unchecked")
    public void del(String... key) {
        if (key != null && key.length > 0) {
            if (key.length == 1) {
                redisTemplate.delete(key[0]);
            } else {
                redisTemplate.delete(CollectionUtils.arrayToList(key));
            }
        }
    }

    /**
     * 普通缓存获取
     *
     * @param key 键
     * @return 值
     */
    public Object get(String key) {
        return key == null ? null : redisTemplate.opsForValue().get(key);
    }

    /**
     * 普通缓存放入
     *
     * @param key   键
     * @param value 值
     * @return true成功 false失败
     */
    public boolean set(String key, Object value) {
        try {
            redisTemplate.opsForValue().set(key, value);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }

    }

    /**
     * 普通缓存放入并设置时间
     *
     * @param key   键
     * @param value 值
     * @param time  时间(秒) time要大于0 如果time小于等于0 将设置无限期
     * @return true成功 false 失败
     */
    public boolean set(String key, Object value, long time) {
        try {
            if (time > 0) {
                redisTemplate.opsForValue().set(key, value, time,
TimeUnit.SECONDS);
```

```
            } else {
                set(key, value);
            }
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 递增
     *
     * @param key   键
     * @param delta 要增加几(大于0)
     * @return
     */
    public long incr(String key, long delta) {
        if (delta < 0) {
            throw new RuntimeException("递增因子必须大于0");
        }
        return redisTemplate.opsForValue().increment(key, delta);
    }

    /**
     * 递减
     *
     * @param key   键
     * @param delta 要减少几(小于0)
     * @return
     */
    public long decr(String key, long delta) {
        if (delta < 0) {
            throw new RuntimeException("递减因子必须大于0");
        }
        return redisTemplate.opsForValue().increment(key, -delta);
    }

    /**
     * HashGet
     *
     * @param key  键 不能为null
     * @param item 项 不能为null
     * @return 值
     */
    public Object hget(String key, String item) {
        return redisTemplate.opsForHash().get(key, item);
    }

    /**
     * 获取hashKey对应的所有键值
     *
     * @param key 键
     * @return 对应的多个键值
     */
    public Map<Object, Object> hmget(String key) {
        return redisTemplate.opsForHash().entries(key);
    }
```

```java
    /**
     * HashSet
     *
     * @param key 键
     * @param map 对应多个键值
     * @return true 成功 false 失败
     */
    public boolean hmset(String key, Map<String, Object> map) {
        try {
            redisTemplate.opsForHash().putAll(key, map);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * HashSet 并设置时间
     *
     * @param key  键
     * @param map  对应多个键值
     * @param time 时间(秒)
     * @return true成功 false失败
     */
    private boolean hmset(String key, Map<String, Object> map, long time) {
        try {
            redisTemplate.opsForHash().putAll(key, map);
            if (time > 0) {
                expire(key, time);
            }
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 向一张hash表中放入数据,如果不存在将创建
     *
     * @param key   键
     * @param item  项
     * @param value 值
     * @return true 成功 false失败
     */
    public boolean hset(String key, String item, Object value) {
        try {
            redisTemplate.opsForHash().put(key, item, value);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
```

```
     * 向一张hash表中放入数据,如果不存在将创建
     *
     * @param key   键
     * @param item  项
     * @param value 值
     * @param time  时间(秒)  注意:如果已存在的hash表有时间,这里将会替换原有的时间
     * @return true 成功 false失败
     */
    public boolean hset(String key, String item, Object value, long time) {
        try {
            redisTemplate.opsForHash().put(key, item, value);
            if (time > 0) {
                expire(key, time);
            }
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 删除hash表中的值
     *
     * @param key  键 不能为null
     * @param item 项 可以使多个 不能为null
     */
    public void hdel(String key, Object... item) {
        redisTemplate.opsForHash().delete(key, item);
    }

    /**
     * 判断hash表中是否有该项的值
     *
     * @param key  键 不能为null
     * @param item 项 不能为null
     * @return true 存在 false不存在
     */
    public boolean hHasKey(String key, String item) {
        return redisTemplate.opsForHash().hasKey(key, item);
    }

    /**
     * hash递增 如果不存在,就会创建一个 并把新增后的值返回
     *
     * @param key  键
     * @param item 项
     * @param by   要增加几(大于0)
     * @return
     */
    public double hincr(String key, String item, double by) {
        return redisTemplate.opsForHash().increment(key, item, by);
    }

    /**
     * hash递减
     *
     * @param key  键
```

```
     * @param item 项
     * @param by   要减少记(小于0)
     * @return
     */
    public double hdecr(String key, String item, double by) {
        return redisTemplate.opsForHash().increment(key, item, -by);
    }

    /**
     * 根据key获取Set中的所有值
     *
     * @param key 键
     * @return
     */
    public Set<Object> sGet(String key) {
        try {
            return redisTemplate.opsForSet().members(key);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * 根据value从一个set中查询,是否存在
     *
     * @param key   键
     * @param value 值
     * @return true 存在 false不存在
     */
    public boolean sHasKey(String key, Object value) {
        try {
            return redisTemplate.opsForSet().isMember(key, value);
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 将数据放入set缓存
     *
     * @param key    键
     * @param values 值 可以是多个
     * @return 成功个数
     */
    public long sSet(String key, Object... values) {
        try {
            return redisTemplate.opsForSet().add(key, values);
        } catch (Exception e) {
            e.printStackTrace();
            return 0;
        }
    }

    /**
     * 将set数据放入缓存
     *
```

```
     * @param key    键
     * @param time   时间(秒)
     * @param values 值 可以是多个
     * @return 成功个数
     */
    public long sSetAndTime(String key, long time, Object... values) {
        try {
            Long count = redisTemplate.opsForSet().add(key, values);
            if (time > 0) expire(key, time);
            return count;
        } catch (Exception e) {
            e.printStackTrace();
            return 0;
        }
    }

    /**
     * 获取set缓存的长度
     *
     * @param key 键
     * @return
     */
    public long sGetSetSize(String key) {
        try {
            return redisTemplate.opsForSet().size(key);
        } catch (Exception e) {
            e.printStackTrace();
            return 0;
        }
    }

    /**
     * 移除值为value的
     *
     * @param key    键
     * @param values 值 可以是多个
     * @return 移除的个数
     */
    public long setRemove(String key, Object... values) {
        try {
            Long count = redisTemplate.opsForSet().remove(key, values);
            return count;
        } catch (Exception e) {
            e.printStackTrace();
            return 0;
        }
    }

    /**
     * 获取list缓存的内容
     *
     * @param key   键
     * @param start 开始
     * @param end   结束  0 到 -1代表所有值
     * @return
     */
    public List<Object> lGet(String key, long start, long end) {
        try {
```

```
            return redisTemplate.opsForList().range(key, start, end);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * 获取list缓存的长度
     *
     * @param key 键
     * @return
     */
    public long lGetListSize(String key) {
        try {
            return redisTemplate.opsForList().size(key);
        } catch (Exception e) {
            e.printStackTrace();
            return 0;
        }
    }

    /**
     * 通过索引 获取list中的值
     *
     * @param key   键
     * @param index 索引  index>=0时， 0 表头，1 第二个元素，依次类推；index<0时，-1，表尾，-2倒数第二个元素，依次类推
     * @return
     */
    public Object lGetIndex(String key, long index) {
        try {
            return redisTemplate.opsForList().index(key, index);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * 将list放入缓存
     *
     * @param key   键
     * @param value 值
     * @return
     */
    public boolean lSet(String key, Object value) {
        try {
            redisTemplate.opsForList().rightPush(key, value);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 将list放入缓存
```

```
     *
     * @param key   键
     * @param value 值
     * @param time  时间(秒)
     * @return
     */
    public boolean lSet(String key, Object value, long time) {
        try {
            redisTemplate.opsForList().rightPush(key, value);
            if (time > 0) expire(key, time);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 将list放入缓存
     *
     * @param key   键
     * @param value 值
     * @return
     */
    public boolean lSet(String key, List<Object> value) {
        try {
            redisTemplate.opsForList().rightPushAll(key, value);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 将list放入缓存
     *
     * @param key   键
     * @param value 值
     * @param time  时间(秒)
     * @return
     */
    public boolean lSet(String key, List<Object> value, long time) {
        try {
            redisTemplate.opsForList().rightPushAll(key, value);
            if (time > 0) expire(key, time);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 根据索引修改list中的某条数据
     *
     * @param key   键
     * @param index 索引
```

```
     * @param value 值
     * @return
     */
    public boolean lUpdateIndex(String key, long index, Object value) {
        try {
            redisTemplate.opsForList().set(key, index, value);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 移除N个值为value
     *
     * @param key   键
     * @param count 移除多少个
     * @param value 值
     * @return 移除的个数
     */
    public long lRemove(String key, long count, Object value) {
        try {
            Long remove = redisTemplate.opsForList().remove(key, count, value);
            return remove;
        } catch (Exception e) {
            e.printStackTrace();
            return 0;
        }
    }

    public void expire(int i, int i1, int i2) {
    }
}
```

四、统一返回结果

1.枚举

```
package com.oldbai.halfmoon.response;

/**
 * 枚举的实现
 *
 * @author 老白
 */

public enum ResponseState implements IResponseState {
    SUCCESS(10000, true, "操作成功"),
    FAILED(20000, false, "操作失败"),
    JOIN_IN_SUCCESS(20001, false, "注册成功"),
    PARAMS_ILL(30000, false, "参数错误"),
    PERMISSION_DENIED(40002, false, "权限不够"),
    ACCOUNT_FORBID(40003, false, "账号被禁用"),
```

```
    NOT_LOGIN(50000, false, "账号未登录"),
    ERROE_403(50403, false, "权限不足喔......"),
    ERROE_404(50404, false, "页面找不到了，页面丢失......"),
    ERROE_504(50504, false, "系统繁忙，稍后再试......"),
    ERROE_505(50505, false, "请求错误呀请检查数据是否正确......"),
    LOGIN_SUCCESS(60000, true, "登录成功");


    int code;
    boolean isSuccess;
    String message;

    ResponseState(int code, boolean isSuccess, String message) {
        this.code = code;
        this.isSuccess = isSuccess;
        this.message = message;
    }

    @Override
    public String getMessage() {
        return message;
    }

    @Override
    public boolean isSuccess() {
        return isSuccess;
    }

    @Override
    public int getCode() {
        return code;
    }

}
```

2.接口

```
package com.oldbai.halfmoon.response;

/**
 * 枚举接口
 * @author 老白
 */
public interface IResponseState {

    String getMessage();

    boolean isSuccess();

    int getCode();
}
```

3.返回类

```
package com.oldbai.halfmoon.response;

/**
 * 统一返回结果类
 *
 * @author 老白
 */
public class ResponseResult {

    private String message;
    private boolean success;
    private int code;
    private Object data;

    public ResponseResult(ResponseState commentResult) {
        this.message = commentResult.getMessage();
        this.success = commentResult.isSuccess();
        this.code = commentResult.getCode();
        this.data = null;
    }

    public static ResponseResult GET(ResponseState commentResult) {
        return new ResponseResult(commentResult);
    }

    public static ResponseResult GET(ResponseState commentResult, String
message) {
        ResponseResult get = GET(commentResult);
        get.setMessage(message);
        return get;
    }


    public static ResponseResult JOIN_SUCCESS() {
        return new ResponseResult(ResponseState.JOIN_IN_SUCCESS);
    }

    public static ResponseResult NO_LOGIN() {
        return new ResponseResult(ResponseState.NOT_LOGIN);
    }

    public static ResponseResult NO_LOGIN(String message) {
        ResponseResult nologin = NO_LOGIN();
        nologin.setMessage(message);
        return nologin;
    }

    public static ResponseResult NO_PERMISSION() {
        return new ResponseResult(ResponseState.PERMISSION_DENIED);
    }

    public static ResponseResult NO_PERMISSION(String message) {
        ResponseResult noPermission = NO_PERMISSION();
        noPermission.setMessage(message);
```

```
        return noPermission;
    }


    public static ResponseResult SUCCESS() {
        return new ResponseResult(ResponseState.SUCCESS);
    }

    public static ResponseResult SUCCESS(String message) {
        ResponseResult success = SUCCESS();
        success.setMessage(message);
        return success;
    }

    public static ResponseResult SUCCESS(Object data) {
        ResponseResult success = SUCCESS();
        success.setData(data);
        return success;
    }

    public static ResponseResult SUCCESS(String message, Object data) {
        ResponseResult success = SUCCESS();
        success.setMessage(message);
        success.setData(data);
        return success;
    }

    public static ResponseResult FAILED() {
        return new ResponseResult(ResponseState.FAILED);
    }

    public static ResponseResult FAILED(String message) {
        ResponseResult failed = FAILED();
        failed.setMessage(message);
        return failed;
    }

    public static ResponseResult FAILED(Object data) {
        ResponseResult failed = FAILED();
        failed.setData(data);
        return failed;
    }

    public static ResponseResult FAILED(String message, Object data) {
        ResponseResult failed = FAILED();
        failed.setMessage(message);
        failed.setData(data);
        return failed;
    }


    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
```

```
    public boolean isSuccess() {
        return success;
    }

    public void setSuccess(boolean success) {
        this.success = success;
    }

    public int getCode() {
        return code;
    }

    public void setCode(int code) {
        this.code = code;
    }

    public Object getData() {
        return data;
    }

    public void setData(Object data) {
        this.data = data;
    }
}
```

五、统一日志管理

1.添加配制logback配制文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- 日志级别从低到高分为TRACE < DEBUG < INFO < WARN < ERROR < FATAL，如果设置为WARN，则低于WARN的信息都不会输出 -->
<!-- scan:当此属性设置为true时，配置文件如果发生改变，将会被重新加载，默认值为true -->
<!-- scanPeriod:设置监测配置文件是否有修改的时间间隔，如果没有给出时间单位，默认单位是毫秒。当scan为true时，此属性生效。默认的时间间隔为1分钟。 -->
<!-- debug:当此属性设置为true时，将打印出logback内部日志信息，实时查看logback运行状态。默认值为false。 -->

<!-- 参考文章 https://www.cnblogs.com/zhangjianbing/p/8992897.html -->
<configuration scan="true" scanPeriod="10 seconds">

    <contextName>logback</contextName>

    <!-- 设置属性，文件中使用 ${} 获取属性值 -->
    <!-- 项目名称 -->
    <springProperty scope="context" name="projectName"
source="spring.application.name"/>
    <!-- 日志存放目录；默认从配置文件中读取 "log.path" 的配置值，若没有配置，使用默认值 /var/yuemia-live-logs/ -->
    <springProperty scope="context" name="logPath" source="log.path"
defaultValue="G:\MyJava\HalfMoon\log"/>
    <!-- 日志级别；默认从配置文件中读取 "logging.level.root" 的配置值，若没有配置，使用默认值 info -->
```

```
    <springProperty scope="context" name="logging.level.root"
source="logging.level.root" defaultValue="info"/>

    <!--log日志格式-->
    <property name="PATTERN" value="%d{${LOG_DATEFORMAT_PATTERN:-yyyy-MM-dd
HH:mm:ss.SSS}} ${LOG_LEVEL_PATTERN:-%5p} ${PID:- } --- [%t] %-40.40logger{39} :
%m%n${LOG_EXCEPTION_CONVERSION_WORD:-%wEx}"/>

    <!-- 彩色日志依赖的渲染类 -->
    <conversionRule conversionWord="clr"
converterClass="org.springframework.boot.logging.logback.ColorConverter"/>
    <conversionRule conversionWord="wex"
converterClass="org.springframework.boot.logging.logback.WhitespaceThrowableProx
yConverter"/>
    <conversionRule conversionWord="wEx"
converterClass="org.springframework.boot.logging.logback.ExtendedWhitespaceThrow
ableProxyConverter"/>
    <!-- 彩色日志格式 -->
    <property name="CONSOLE_LOG_PATTERN"
              value="%yellow(%date{yyyy-MM-dd HH:mm:ss}) |%highlight(%-5level)
|%blue(%thread) |%blue(%file:%line) |%green(%logger) |%cyan(%msg%n)"/>


    <!--输出到控制台-->
    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <!--此日志appender是为开发使用，只配置最底级别，控制台输出的日志级别是大于或等于此级
别的日志信息-->
        <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
            <level>${logging.level.root}</level>
        </filter>
        <encoder>
            <pattern>${CONSOLE_LOG_PATTERN}</pattern>
            <!-- 设置字符集 -->
            <charset>UTF-8</charset>
        </encoder>
    </appender>


    <!--输出到文件-->
    <!-- DEBUG 日志 -->
    <appender name="DEBUG_FILE"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${logPath}/${projectName}/log_debug.log</file>
        <encoder>
            <pattern>${PATTERN}</pattern>
            <charset>UTF-8</charset>
        </encoder>
        <!-- 日志记录器的滚动策略，按日期，按大小记录 -->
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <!-- 日志输出文件名路径 -->
            <fileNamePattern>${logPath}/${projectName}/debug/log-debug-%d{yyyy-
MM-dd}.%i.log</fileNamePattern>
            <timeBasedFileNamingAndTriggeringPolicy
class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
                <maxFileSize>100MB</maxFileSize>
            </timeBasedFileNamingAndTriggeringPolicy>
            <!--日志文件保留天数-->
```

```
            <maxHistory>15</maxHistory>
        </rollingPolicy>
        <!-- 此日志文件只记录debug级别的 -->
        <filter class="ch.qos.logback.classic.filter.LevelFilter">
            <level>debug</level>
            <onMatch>ACCEPT</onMatch>
            <onMismatch>DENY</onMismatch>
        </filter>
    </appender>

    <!-- INFO 日志 -->
    <appender name="INFO_FILE"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${logPath}/${projectName}/log_info.log</file>
        <encoder>
            <pattern>${PATTERN}</pattern>
            <charset>UTF-8</charset>
        </encoder>
        <!-- 日志记录器的滚动策略，按日期，按大小记录 -->
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${logPath}/${projectName}/info/log-info-%d{yyyy-MM-
dd}.%i.log</fileNamePattern>
            <timeBasedFileNamingAndTriggeringPolicy
class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
                <maxFileSize>100MB</maxFileSize>
            </timeBasedFileNamingAndTriggeringPolicy>
            <!--日志文件保留天数-->
            <maxHistory>15</maxHistory>
        </rollingPolicy>
        <!-- 此日志文件只记录info级别的 -->
        <filter class="ch.qos.logback.classic.filter.LevelFilter">
            <level>info</level>
            <onMatch>ACCEPT</onMatch>
            <onMismatch>DENY</onMismatch>
        </filter>
    </appender>

    <!-- WARN 日志 -->
    <appender name="WARN_FILE"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${logPath}/${projectName}/log_warn.log</file>
        <encoder>
            <pattern>${PATTERN}</pattern>
            <charset>UTF-8</charset>
        </encoder>
        <!-- 日志记录器的滚动策略，按日期，按大小记录 -->
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${logPath}/${projectName}/warn/log-warn-%d{yyyy-MM-
dd}.%i.log</fileNamePattern>
            <timeBasedFileNamingAndTriggeringPolicy
class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
                <maxFileSize>100MB</maxFileSize>
            </timeBasedFileNamingAndTriggeringPolicy>
            <!--日志文件保留天数-->
            <maxHistory>15</maxHistory>
        </rollingPolicy>
```

```
        <!-- 此日志文件只记录warn级别的 -->
        <filter class="ch.qos.logback.classic.filter.LevelFilter">
            <level>warn</level>
            <onMatch>ACCEPT</onMatch>
            <onMismatch>DENY</onMismatch>
        </filter>
    </appender>


    <!-- ERROR 日志 -->
    <appender name="ERROR_FILE"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${logPath}/${projectName}/log_error.log</file>
        <encoder>
            <pattern>${PATTERN}</pattern>
            <charset>UTF-8</charset>
        </encoder>
        <!-- 日志记录器的滚动策略，按日期，按大小记录 -->
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${logPath}/${projectName}/error/log-error-%d{yyyy-
MM-dd}.%i.log</fileNamePattern>
            <timeBasedFileNamingAndTriggeringPolicy
class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
                <maxFileSize>100MB</maxFileSize>
            </timeBasedFileNamingAndTriggeringPolicy>
            <!--日志文件保留天数-->
            <maxHistory>15</maxHistory>
        </rollingPolicy>
        <!-- 此日志文件只记录ERROR级别的 -->
        <filter class="ch.qos.logback.classic.filter.LevelFilter">
            <level>ERROR</level>
            <onMatch>ACCEPT</onMatch>
            <onMismatch>DENY</onMismatch>
        </filter>
    </appender>

    <!--
        root节点是必选节点，用来指定最基础的日志输出级别，只有一个level属性
        level:用来设置打印级别，大小写无关：TRACE，DEBUG，INFO，WARN，ERROR，ALL 和
OFF，
        不能设置为INHERITED或者同义词NULL。默认是DEBUG
        可以包含零个或多个元素，标识这个appender将会添加到这个logger。

        nico:设置level级别后，磁盘上低级别的日志文件里没有任何内容，控制台也是一样不会输出
    -->
    <root level="INFO">
        <appender-ref ref="CONSOLE"/>
        <appender-ref ref="DEBUG_FILE"/>
        <appender-ref ref="INFO_FILE"/>
        <appender-ref ref="WARN_FILE"/>
        <appender-ref ref="ERROR_FILE"/>
    </root>
</configuration>
```

六、统一异常管理

1.访问异常管理

1).错误页面注册

```
package com.oldbai.halfmoon.config;

import org.springframework.boot.web.server.ErrorPage;
import org.springframework.boot.web.server.ErrorPageRegistrar;
import org.springframework.boot.web.server.ErrorPageRegistry;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpStatus;

/**
 * 页面错误返回处理
 * 错误页面注册
 * @author 老白
 */
@Configuration
public class ErrorPageConfig implements ErrorPageRegistrar {
    @Override
    public void registerErrorPages(ErrorPageRegistry registry) {
        registry.addErrorPages(new ErrorPage(HttpStatus.FORBIDDEN, "/403"));
        registry.addErrorPages(new ErrorPage(HttpStatus.NOT_FOUND, "/404"));
        registry.addErrorPages(new ErrorPage(HttpStatus.GATEWAY_TIMEOUT,
"/504"));
        registry.addErrorPages(new
ErrorPage(HttpStatus.HTTP_VERSION_NOT_SUPPORTED, "/505"));
    }
}
```

2).页面异常返回数据

```
package com.oldbai.blog.controller;

import com.oldbai.blog.response.ResponseResult;
import com.oldbai.blog.response.ResponseState;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UtilController {
    @GetMapping("/403")
    public ResponseResult page403() {
        ResponseResult failed = new ResponseResult(ResponseState.ERROE_403);
        return failed;
    }

    @GetMapping("/404")
    public ResponseResult page404() {
        ResponseResult failed = new ResponseResult(ResponseState.ERROE_404);
        return failed;
```

```
    }

    @GetMapping("/504")
    public ResponseResult page504() {
        ResponseResult failed = new ResponseResult(ResponseState.ERROE_504);
        return failed;
    }

    @GetMapping("/505")
    public ResponseResult page505() {
        ResponseResult failed = new ResponseResult(ResponseState.ERROE_505);
        return failed;
    }
}
```

2.内部异常处理

1).自定义未登录异常处理

```
package com.oldbai.plusblog.exception;

/**
 * 未登录异常处理
 *
 * @author 老白
 */
public class NotLoginException extends Exception {

    @Override
    public String toString() {
        return "not login.";
    }
}
```

2).统一异常处理

```
package com.oldbai.plusblog.controller.utilControll;

import com.oldbai.plusblog.exception.NotLoginException;
import com.oldbai.plusblog.response.ResponseResult;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;

/**
 * 统一异常处理
 * @author 老白
 */
@ControllerAdvice
public class ExceptionController {
```

```
    @ExceptionHandler(NotLoginException.class)
    @ResponseBody
    public ResponseResult handlerNotLoginException(NotLoginException e) {
        e.printStackTrace();
        return ResponseResult.FAILED("账号未登录");
    }

    @ExceptionHandler(Exception.class)
    @ResponseBody
    public ResponseResult handlerException(Exception e) {
        e.printStackTrace();
        return ResponseResult.FAILED("服务器繁忙");
    }

}
```

七、工具类

1.设置全局默认变量

```
package com.oldbai.blog.utils;

/**
 * 设置默认属性
 *
 * @author 老白
 */
public interface Constants {

    /**
     * 用户的初始化
     */
    interface User {
        //初始化管理员角色
        String ROLE_ADMIN = "role_admin";
        String ROLE_NORMAL = "role_normal";
        //头像
        String DEFAULT_AVATAR =
"https://gimg2.baidu.com/image_search/src=http%3A%2F%2Fc-
ssl.duitang.com%2Fuploads%2Fitem%2F202007%2F01%2F20200701063944_5VaBk.thumb.1000
_0.jpeg&refer=http%3A%2F%2Fc-
ssl.duitang.com&app=2002&size=f9999,10000&q=a80&n=0&g=0n&fmt=jpeg?
sec=1614571386&t=2e68974a8d276943307d75ea32457e3d";
        //状态
        String DEFAULT_STATE = "1";
        //以下是redis的key
        //验证码的key
        String KEY_CAPTCHA_CONTENT = "key_captcha_content_";
        //验证码的key
        String KEY_EMAIL_CODE_CONTENT = "key_email_code_content_";
        //email邮件IP地址的key
        String KEY_EMAIL_SEND_IP = "key_email_send_ip_";
        //email邮件发送邮箱地址的key
        String KEY_EMAIL_SEND_ADDRESS = "key_email_send_address_";
```

```
        //md5的token kdy
        String KEY_TOKEN = "key_token_";
        //登陆后返回给cookid的token的名字
        String KEY_TOKEN_NAME = "blog_token";
        //登陆后返回给cookid的名字
        String COOKIE_TOKE_KEY = "blog_token";
        //登陆后返回给cookid的时间
        int COOKIE_TOKE_AGE = 60 * 60;
        //TODO redis存活时间 1 天
        int REDIS_AGE_DAY = 60 * 60;
    }

    /**
     * setting的初始化
     */
    interface Settings {
        //初始化管理员账号
        String HAS_MANAGER_ACCOUNT_INIT_STATE = "has_manager_init_state";
        //网站标题
        String WEB_SIZE_TITLE = "web_size_title";
        //网站描述
        String WEB_SIZE_DESCRIPTION = "web_size_description";
        //键，关键字
        String WEB_SIZE_KEYWORDS = "web_size_keywords";
        //网站浏览量
        String WEB_SIZE_VIEW_COUNT = "web_size_view_count";
    }

    /**
     * redis 时间类
     * 单位 毫秒
     */
    interface RedisTime {
        int init = 1000;
        int MIN = 60 * init;
        int HOUR = 60 * MIN;
        int DAY = 24 * HOUR;
        int WEEK = 7 * DAY;
        int MONTH = 30 * DAY;
        int YEAR = 365 * DAY;
    }

    /**
     * 统一时间
     * 单位 秒
     */
    interface TimeValue {
        int MIN = 60;
        int HOUR = 60 * MIN;
        int DAY = 24 * HOUR;
        int WEEK = 7 * DAY;
        int MONTH = 30 * DAY;
        int YEAR = 365 * DAY;
    }

    /**
     * 分页配制
     */
```

```
    interface Page {
        int DEFAULT_PAGE = 1;
        int MIN_SIZE = 5;
    }

    /**
     * 图片格式限制
     */
    interface ImageType {
        String PREFIX = "image/";
        String TYPE_JGP = "jpg";
        String TYPE_PNG = "png";
        String TYPE_GIF = "gif";
        String TYPE_JGP_WITH_PREFIX = PREFIX + TYPE_JGP;
        String TYPE_PNG_WITH_PREFIX = PREFIX + TYPE_PNG;
        String TYPE_GIF_WITH_PREFIX = PREFIX + TYPE_GIF;
    }

    /**
     * 文章
     */
    interface Article {
        // 0表示删除 、1表示发布 、2表示草稿 、3表示置顶
        String STATE_DELETE = "0";
        String STATE_PUBLISH = "1";
        String STATE_DRAFT = "2";
        String STATE_TOP = "3";
        int TITLE_MAX_LENGTH = 128;
        int SUMMARY_MAX_LENGTH = 256;
    }

    /**
     * 评论的一个通用处理
     */
    interface Comment {
        String STATE_PUBLISH = "";
        String STATE_TOP = "";
    }
}
```

2.常用的验证正则类

```
package com.oldbai.halfmoon.util;

import org.springframework.util.StringUtils;

import java.util.regex.Pattern;

/**
 * 常用的验证正则表达式
 *
 * @author 老白
 */
```

```
public class ValidateUtil {
    /**
     * 手机号规则
     */
    public static final String MOBILE_PATTERN = "^((13[0-9])|(14[0-9])|(15[0-
9])|(17[0-9])|(18[0-9]))(\\d{8})$";
    /**
     * 中国电信号码格式验证 手机段： 133,153,180,181,189,177,1700,173
     **/
    private static final String CHINA_TELECOM_PATTERN = "(?:^(?:\\+86)?
1(?:33|53|7[37]|8[019])\\d{8}$)|(?:^(?:\\+86)?1700\\d{7}$)";
    /**
     * 中国联通号码格式验证 手机段:
130,131,132,155,156,185,186,145,176,1707,1708,1709,175
     **/
    private static final String CHINA_UNICOM_PATTERN = "(?:^(?:\\+86)?1(?:3[0-
2]|4[5]|5[56]|7[56]|8[56])\\d{8}$)|(?:^(?:\\+86)?170[7-9]\\d{7}$)";
    /**
     * 中国移动号码格式验证 手机段:
134,135,136,137,138,139,150,151,152,157,158,159,182,183,184,187,188,147,178,1705
     **/
    private static final String CHINA_MOVE_PATTERN = "(?:^(?:\\+86)?1(?:3[4-
9]|4[7]|5[0-27-9]|7[8]|8[2-478])\\d{8}$)|(?:^(?:\\+86)?1705\\d{7}$)";
    /**
     * 密码规则（6-16位字母、数字）
     */
    public static final String PASSWORD_PATTERN = "^[0-9A-Za-z]{6,16}$";
    /**
     * 固号（座机）规则
     */
    public static final String LANDLINE_PATTERN = "^(?:\\
(\\d{3,4}\\)|\\d{3,4}-)?\\d{7,8}(?:-\\d{1,4})?$";
    /**
     * 邮政编码规则
     */
    public static final String POSTCODE_PATTERN = "[1-9]\\d{5}";
    /**
     * 邮箱规则
     */
    public static final String EMAIL_PATTERN = "^([a-z0-9A-Z]+[-|_|\\.]?)+[a-z0-
9A-Z]@([a-z0-9A-Z]+(-[a-z0-9A-Z]+)?\\.)+[a-zA-Z]{2,}$";
    /**
     * 年龄规则 1-120之间
     */
    public static final String AGE_PATTERN = "^(?:[1-9][0-9]?|1[01][0-9]|120)$";
    /**
     * 身份证规则
     */
    public static final String IDCARD_PATTERN = "^\\d{15}|\\d{18}$";
    /**
     * URL规则，http、www、ftp
     */
    public static final String URL_PATTERN = "http(s)?://([\\w-]+\\.)+[\\w-]+
(/[\\w- ./?%&=]*)?";
    /**
     * QQ规则
     */
    public static final String QQ_PATTERN = "^[1-9][0-9]{4,13}$";
```

```
    /**
     * 全汉字规则
     */
    public static final String CHINESE_PATTERN = "^[\u4E00-\u9FA5]+$";
    /**
     * 全字母规则
     */
    public static final String STR_ENG_PATTERN = "^[A-Za-z]+$";
    /**
     * 整数规则
     */
    public static final String INTEGER_PATTERN = "^-?[0-9]+$";
    /**
     * 正整数规则
     */
    public static final String POSITIVE_INTEGER_PATTERN = "^\\+?[1-9][0-9]*$";


    /**
     * @param mobile 手机号码
     * @return boolean
     * @Description: 验证手机号码格式
     */
    public static boolean validateMobile(String mobile) {
        if (StringUtils.isEmpty(mobile)) {
            return Boolean.FALSE;
        }
        return mobile.matches(MOBILE_PATTERN);
    }

    /**
     * 验证是否是电信手机号,133、153、180、189、177
     *
     * @param mobile 手机号
     * @return boolean
     */
    public static boolean validateTelecom(String mobile) {
        if (StringUtils.isEmpty(mobile)) {
            return Boolean.FALSE;
        }
        return mobile.matches(CHINA_TELECOM_PATTERN);
    }

    /**
     * 验证是否是联通手机号 130,131,132,155,156,185,186,145,176,1707,1708,1709,175
     *
     * @param mobile 电话号码
     * @return boolean
     */
    public static boolean validateUnionMobile(String mobile) {
        if (StringUtils.isEmpty(mobile)) {
            return Boolean.FALSE;
        }
        return mobile.matches(CHINA_UNICOM_PATTERN);
    }

    /**
     * 验证是否是移动手机号
```

```
     *
     * @param mobile 手机号
134,135,136,137,138,139,150,151,152,157,158,159,182,183,184,187,188,147,178,1705
     * @return boolean
     */
    public static boolean validateMoveMobile(String mobile) {
        if (StringUtils.isEmpty(mobile)) {
            return Boolean.FALSE;
        }
        return mobile.matches(CHINA_MOVE_PATTERN);
    }

    /**
     * @param pwd 密码
     * @return boolean
     * @Description: 验证密码格式  6-16 位字母、数字
     */
    public static boolean validatePwd(String pwd) {
        if (StringUtils.isEmpty(pwd)) {
            return Boolean.FALSE;
        }
        return Pattern.matches(PASSWORD_PATTERN, pwd);
    }

    /**
     * 验证座机号码，格式如：58654567,023-58654567
     *
     * @param landline 固话、座机
     * @return boolean
     */
    public static boolean validateLandLine(final String landline) {
        if (StringUtils.isEmpty(landline)) {
            return Boolean.FALSE;
        }
        return landline.matches(LANDLINE_PATTERN);
    }

    /**
     * 验证邮政编码
     *
     * @param postCode 邮政编码
     * @return boolean
     */
    public static boolean validatePostCode(final String postCode) {
        if (StringUtils.isEmpty(postCode)) {
            return Boolean.FALSE;
        }
        return postCode.matches(POSTCODE_PATTERN);
    }

    /**
     * 验证邮箱（电子邮件）
     *
     * @param email 邮箱（电子邮件）
     * @return boolean
     */
    public static boolean validateEamil(final String email) {
        if (StringUtils.isEmpty(email)) {
```

```
            return Boolean.FALSE;
        }
        return email.matches(EMAIL_PATTERN);
    }

    /**
     * 判断年龄，1-120之间
     *
     * @param age 年龄
     * @return boolean
     */
    public static boolean validateAge(final String age) {
        if (StringUtils.isEmpty(age)) {
            return Boolean.FALSE;
        }
        return age.matches(AGE_PATTERN);
    }

    /**
     * 身份证验证
     *
     * @param idCard 身份证
     * @return boolean
     */
    public static boolean validateIDCard(final String idCard) {
        if (StringUtils.isEmpty(idCard)) {
            return Boolean.FALSE;
        }
        return idCard.matches(IDCARD_PATTERN);
    }

    /**
     * URL地址验证
     *
     * @param url URL地址
     * @return boolean
     */
    public static boolean validateUrl(final String url) {
        if (StringUtils.isEmpty(url)) {
            return Boolean.FALSE;
        }
        return url.matches(URL_PATTERN);
    }

    /**
     * 验证QQ号
     *
     * @param qq QQ号
     * @return boolean
     */
    public static boolean validateQq(final String qq) {
        if (StringUtils.isEmpty(qq)) {
            return Boolean.FALSE;
        }
        return qq.matches(QQ_PATTERN);
    }

    /**
```

```
     * 验证字符串是否全是汉字
     *
     * @param str 字符串
     * @return boolean
     */
    public static boolean validateChinese(final String str) {
        if (StringUtils.isEmpty(str)) {
            return Boolean.FALSE;
        }
        return str.matches(CHINESE_PATTERN);
    }

    /**
     * 判断字符串是否全字母
     *
     * @param str 字符串
     * @return boolean
     */
    public static boolean validateStrEnglish(final String str) {
        if (StringUtils.isEmpty(str)) {
            return Boolean.FALSE;
        }
        return str.matches(STR_ENG_PATTERN);
    }

    /**
     * 判断是否是整数，包括负数
     *
     * @param str 字符串
     * @return boolean
     */
    public static boolean validateInteger(final String str) {
        if (StringUtils.isEmpty(str)) {
            return Boolean.FALSE;
        }
        return str.matches(INTEGER_PATTERN);
    }

    /**
     * 判断是否是大于0的正整数
     *
     * @param str 字符串
     * @return boolean
     */
    public static boolean validatePositiveInt(final String str) {
        if (StringUtils.isEmpty(str)) {
            return Boolean.FALSE;
        }
        return str.matches(POSITIVE_INTEGER_PATTERN);
    }
}
```

3.导入hutool工具包

```
<!--        hutool工具包-->
        <dependency>
            <groupId>cn.hutool</groupId>
            <artifactId>hutool-all</artifactId>
            <version>5.5.8</version>
        </dependency>
```

4.初始化页面和大小方法

```
public static int getPage(int page) {
        return page < com.oldbai.blog.utils.Constants.Page.DEFAULT_PAGE ? page :
com.oldbai.blog.utils.Constants.Page.DEFAULT_PAGE;
    }

    public static int getSize(int size) {
        return size < com.oldbai.blog.utils.Constants.Page.MIN_SIZE ?
com.oldbai.blog.utils.Constants.Page.MIN_SIZE : size;
    }
```

5.获取request和response

```
HttpServletRequest request = null;
HttpServletResponse response = null;

public void getRequestAndResponse() {
        this.request = ((ServletRequestAttributes)
RequestContextHolder.getRequestAttributes()).getRequest();
        this.response = ((ServletRequestAttributes)
RequestContextHolder.getRequestAttributes()).getResponse();

    }
```

八、用户中心接口设计

1.初始化管理员账号（注意sql关键字）

1）.测试数据

```
{

  "email": "1005777562@qq.com",

  "password": "admin",

  "userName": "admin"
}
```

2）.接口

```
/**
     * 初始化管理员账号
     * 需要:
     * username 账号
     * password 密码
     * email 邮箱
     *
     * @return
     */
    @ApiOperation("初始化管理员账号")
    @PostMapping("/admin_account")
    public ResponseResult initManagerAccount(@RequestBody User user) {
        log.info(user.getUserName());
        log.info(user.getPassword());
        log.info(user.getEmail());
        return userService.initManagerAccount(user);
    }
```

3）.方法

```
@Override
    public ResponseResult initManagerAccount(User user) {
        getRequestAndResponse();
        //检查是否有初始化管理员账号
        //TODO
        QueryWrapper<Settings> wrapper = new QueryWrapper<>();
        wrapper.eq("`key`", Constants.Settings.HAS_MANAGER_ACCOUNT_INIT_STATE);
        Settings managerAccountState = settingsMapper.selectOne(wrapper);
        if (!StringUtils.isEmpty(managerAccountState)) {
            return ResponseResult.FAILED("已经初始化过管理员账号了！");
        }
        //检查数据
        if (StringUtils.isEmpty(user.getUserName())) {
            return ResponseResult.FAILED("用户名不能为空");
        }
        if (StringUtils.isEmpty(user.getPassword())) {
            return ResponseResult.FAILED("密码不能为空");
        }
        if (StringUtils.isEmpty(user.getEmail())) {
            return ResponseResult.FAILED("邮箱不能为空");
        }
        //加密密码
        user.setPassword(BCrypt.hashpw(user.getPassword()));
        //补充数据
        //角色
        user.setRoles(Constants.User.ROLE_ADMIN);
        //头像
        user.setAvatar(Constants.User.DEFAULT_AVATAR);
        //默认状态
        user.setState(Constants.User.DEFAULT_STATE);
        //注册IP
        user.setRegIp(request.getRemoteAddr());
        //登陆IP
        user.setLoginIp(request.getRemoteAddr());
        //创建时间
        user.setCreateTime(new Date());
```

```
        //更新时间
        user.setUpdateTime(new Date());
        //保存到数据库中
        //更新标记
        int insert = userMapper.insert(user);
        Settings settings = new Settings();
        settings.setKey(Constants.Settings.HAS_MANAGER_ACCOUNT_INIT_STATE);
        settings.setValue("1");
        settingsMapper.insert(settings);
        if (insert == 0) {
            return ResponseResult.FAILED("初始化失败");
        } else {
            return ResponseResult.SUCCESS("初始化成功");
        }
    }
```

2.注册用户

1）.测试数据

```
{

  "email": "934105499@qq.com",

  "password": "oldbai",

  "userName": "oldbai"
}
```

2）.接口

```
/**
     * 注册
     * 需要用户输入：
     * 邮箱地址（邮箱验证码）
     * 邮箱验证码
     * 昵称(用户名)
     * 密码
     * 人类验证码（图灵验证码）
     *
     * @param user
     * @return
     */
    @ApiOperation("注册")
    @PostMapping("/register")
    public ResponseResult register(@RequestBody User user,
                                   @RequestParam("verify_code") String
emailCode,
                                   @RequestParam("captcha") String captcha,
                                   @RequestParam("captcha_key") String
captchaKey) {

        return userService.register(user, emailCode, captcha, captchaKey);
```

```
    }
```

3）.方法

```
 /**
     * 注册功能
     *
     * @param user
     * @param emailCode
     * @param captcha
     * @param captchaKey
     * @return
     */
    @Override
    public ResponseResult register(User user, String emailCode, String captcha,
String captchaKey) {
        getRequestAndResponse();
        //1.检查当前用户名是否已经注册，或者为不为空
        String username = user.getUserName();
        if (StringUtils.isEmpty(username)) {
            return ResponseResult.FAILED("用户名不可以为空！");
        }
        QueryWrapper<User> wrapper = new QueryWrapper<>();
        wrapper.eq("user_name", username);
        User one = userMapper.selectOne(wrapper);
        if (!StringUtils.isEmpty(one)) {
            return ResponseResult.FAILED("该用户已注册！");
        }
        //2.检查邮箱格式是否正确
        String email = user.getEmail();
        if (!ValidateUtil.validateEamil(email)) {
            return ResponseResult.FAILED("邮箱格式不正确！");
        }
        if (StringUtils.isEmpty(email)) {
            return ResponseResult.FAILED("邮箱地址不可以为空！");
        }
        //3.检查该邮箱是否已经注册
        QueryWrapper<User> userQueryWrapper = new QueryWrapper<>();
        userQueryWrapper.eq("email", email);
        User oneByEmail = userMapper.selectOne(userQueryWrapper);
        if (!StringUtils.isEmpty(oneByEmail)) {
            return ResponseResult.FAILED("该邮箱地址已被注册！");
        }
        //4.检查邮箱验证码是否正确
        String code = (String)
redisUtil.get(Constants.User.KEY_EMAIL_CODE_CONTENT + email);
        if (StringUtils.isEmpty(code)) {
            return ResponseResult.FAILED("验证码已经过期了.....");
        }
        if (!emailCode.equals(code)) {
            return ResponseResult.FAILED("邮箱验证码不正确.....");
        } else {
            //正确 ， 删除redis里的内容
            redisUtil.del(Constants.User.KEY_EMAIL_CODE_CONTENT + email);
        }
```

```
        //5.检查图灵验证码是否正确
        String key = (String) redisUtil.get(Constants.User.KEY_CAPTCHA_CONTENT +
captchaKey);
        log.info("captcha" + captcha);
        if (StringUtils.isEmpty(key)) {
            return ResponseResult.FAILED("图灵验证码已经过期了......");
        }
        if (!key.equals(captcha)) {
            return ResponseResult.FAILED("图灵验证码不正确......");
        } else {
            redisUtil.del(Constants.User.KEY_CAPTCHA_CONTENT + captchaKey);
        }
        //达到可以注册条件
        //6.对密码进行加密
        String password = user.getPassword();
        if (StringUtils.isEmpty(password)) {
            return ResponseResult.FAILED("密码不可以为空......");
        }
        user.setPassword(BCrypt.hashpw(password));
        //7.补全数据
        //包括：注册IP，登陆IP，角色（普通角色），头像，创建时间，更新时间
        //角色
        user.setRoles(Constants.User.ROLE_NORMAL);
        //头像
        user.setAvatar(Constants.User.DEFAULT_AVATAR);
        //默认状态
        user.setState(Constants.User.DEFAULT_STATE);
        //注册IP
        user.setRegIp(request.getRemoteAddr());
        //登陆IP
        user.setLoginIp(request.getRemoteAddr());
        //创建时间
        user.setCreateTime(new Date());
        //更新时间
        user.setUpdateTime(new Date());
        //8.保存到数据库中
        userMapper.insert(user);
        //9.返回结果
        return ResponseResult.JOIN_SUCCESS();
    }
```

3.获取图灵验证码

1）.导入工具依赖

```
<!--        图灵验证码生成依赖-->
        <dependency>
            <groupId>com.github.whvcse</groupId>
            <artifactId>easy-captcha</artifactId>
            <version>1.6.2</version>
        </dependency>
```

2）.接口

```
/**
     * 获取图灵验证码
     * 使用：
     * 1.用户前端请求一个验证码
     * 2.后台生成验证码，并且保存在 session 中
     * 3.用户提交注册信息（携带了用户所输入的图灵验证码内容）
     * 4.从 session 中拿出存储的验证码内容跟用户输入的验证码进行比较
     * 5.返回结果
     * 6.在前后端分离下，可以把验证码存入 redis 中。设置有效期为10分钟。
     * 7.存 redis 中时，key 可以由前端生成随机数，然后以参数的形式添加到请求图灵验证码的URL
下
     * 8.后台生成图灵验证码，返回并且保存到 redis 中， 以 key - value 形式
     * 9.用户提交注册信息（携带了用户所输入的图灵验证码内容 + key）
     * 10.从 redis 中 根据 key 拿出图灵验证码跟用户输入的验证码进行比较
     * 11.返回结果。正确进行注册并删除redis里的记录，失败返回结果给前端。
     *
     * @return 访问路径 http://localhost:8058/user/captcha
     */
    @ApiOperation("获取图灵验证码")
    @GetMapping("/captcha")
    public void captcha(@RequestParam("captcha_key") String captchaKey) {

        try {
            userService.createCaptcha(captchaKey);
        } catch (Exception e) {
            log.error(e.toString());
        }


    }
```

3）.方法

```
/**
     * 图灵验证码
     *
     * @param captchaKey
     * @throws Exception
     */
@Override
    public void createCaptcha(String captchaKey) throws IOException,
FontFormatException {
        getRequestAndResponse();
        //进行判断是否传入key
        if (StringUtils.isEmpty(captchaKey) || !(captchaKey.length() < 13)) {
            return;
        }
        long key = 01;
        try {
            key = Long.parseLong(captchaKey);
        } catch (Exception e) {
            return;
```

```
        }
        //可以用了

        // 设置请求头为输出图片类型
        response.setContentType("image/gif");
        response.setHeader("Pragma", "No-cache");
        response.setHeader("Cache-Control", "no-cache");
        response.setDateHeader("Expires", 0);

        // 三个参数分别为宽、高、位数
        SpecCaptcha specCaptcha = new SpecCaptcha(200, 60, 5);
        // 设置字体
        // specCaptcha.setFont(new Font("Verdana", Font.PLAIN, 32));  // 有默认字体，可以不用设置
        specCaptcha.setFont(Captcha.FONT_1);
        // 设置类型，纯数字、纯字母、字母数字混合
        specCaptcha.setCharType(Captcha.TYPE_DEFAULT);

        String content = specCaptcha.text().toLowerCase();
        log.info("captcha content == > " + content);
        // 验证码存入redis , 5 分钟内有效
        //删除时机
        //1.自然过期，比如 10 分钟后过期
        //2.验证码用完后删除
        //3.用完的情况：有get的地方
        redisUtil.set(Constants.User.KEY_CAPTCHA_CONTENT + key, content, 60 * 5);
        // 输出图片流
        specCaptcha.out(response.getOutputStream());
    }
```

4.获取邮箱验证码

1）.导入邮箱依赖

```
<!--email相关-->
        <dependency>
            <groupId>com.sun.mail</groupId>
            <artifactId>javax.mail</artifactId>
            <version>1.6.2</version>
        </dependency>
```

2）.邮箱工具类

```
package com.oldbai.halfmoon.util;

import lombok.extern.slf4j.Slf4j;

import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.mail.*;
import javax.mail.internet.*;
import java.io.File;
```

```java
import java.net.URL;
import java.util.*;


/**
 * 发送邮件工具类
 *
 * @author 老白
 */
@Slf4j
public class EmailSender {
    private static final String TAG = "EmailSender";
    private static Session session;
    private static String user;

    private MimeMessage msg;
    private String text;
    private String html;
    private List<MimeBodyPart> attachments = new ArrayList<MimeBodyPart>();

    /**
     * IMAP/SMTP 服务授权码
     * vrrshutytedmbffc
     */
    private EmailSender() {
        EmailSender.config(EmailSender.SMTP_QQ(false), "xxxxxxxx@qq.com",
"password");
    }

    public static Properties defaultConfig(Boolean debug) {
        Properties props = new Properties();
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.ssl.enable", "true");
        props.put("mail.transport.protocol", "smtp");
        props.put("mail.debug", null != debug ? debug.toString() : "false");
        props.put("mail.smtp.timeout", "10000");
        props.put("mail.smtp.port", "465");
        return props;
    }

    /**
     * smtp entnterprise qq
     *
     * @param debug
     * @return
     */
    public static Properties SMTP_ENT_QQ(boolean debug) {
        Properties props = defaultConfig(debug);
        props.put("mail.smtp.host", "smtp.exmail.qq.com");
        props.put("mail.smtp.ssl.enable", "true");
        return props;
    }

    /**
     * smtp qq
     *
     * @param debug enable debug
     * @return
```

```
     */
    public static Properties SMTP_QQ(boolean debug) {
        Properties props = defaultConfig(debug);
        props.put("mail.smtp.host", "smtp.qq.com");
        props.put("mail.smtp.ssl.enable", "true");
        return props;
    }

    /**
     * smtp 163
     *
     * @param debug enable debug
     * @return
     */
    public static Properties SMTP_163(Boolean debug) {
        Properties props = defaultConfig(debug);
        props.put("mail.smtp.host", "smtp.163.com");
        return props;
    }

    /**
     * config username and password
     *
     * @param props    email property config
     * @param username email auth username
     * @param password email auth password
     */
    public static void config(Properties props, final String username, final
String password) {
        props.setProperty("username", username);
        props.setProperty("password", password);
        config(props);
    }

    public static void config(Properties props) {
        final String username = props.getProperty("username");
        final String password = props.getProperty("password");
        user = username;
        session = Session.getInstance(props, new Authenticator() {
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username, password);
            }
        });
    }

    /**
     * set email subject
     *
     * @param subject subject title
     */
    public static EmailSender subject(String subject) {
        EmailSender EmailSender = new EmailSender();
        EmailSender.msg = new MimeMessage(session);
        try {
            EmailSender.msg.setSubject(subject, "UTF-8");
        } catch (Exception e) {
            log.info(TAG, e + "");
```

```
        }
        return EmailSender;
    }

    /**
     * set email from
     *
     * @param nickName from nickname
     */
    public EmailSender from(String nickName) {
        return from(nickName, user);
    }

    /**
     * set email nickname and from user
     *
     * @param nickName from nickname
     * @param from     from email
     */
    public EmailSender from(String nickName, String from) {
        try {
            String encodeNickName = MimeUtility.encodeText(nickName);
            msg.setFrom(new InternetAddress(encodeNickName + " <" + from +
">"));
        } catch (Exception e) {
            e.printStackTrace();
        }
        return this;
    }

    public EmailSender replyTo(String... replyTo) {
        String result = Arrays.asList(replyTo).toString().replaceAll("(^\\
[|\\]$)", "").replace(", ", ",");
        try {
            msg.setReplyTo(InternetAddress.parse(result));
        } catch (Exception e) {
            e.printStackTrace();
        }
        return this;
    }

    public EmailSender replyTo(String replyTo) {
        try {
            msg.setReplyTo(InternetAddress.parse(replyTo.replace(";", ",")));
        } catch (Exception e) {
            e.printStackTrace();
        }
        return this;
    }

    public EmailSender to(String... to) throws MessagingException {
        return addRecipients(to, Message.RecipientType.TO);
    }

    public EmailSender to(String to) throws MessagingException {
        return addRecipient(to, Message.RecipientType.TO);
    }
```

```
    public EmailSender cc(String... cc) throws MessagingException {
        return addRecipients(cc, Message.RecipientType.CC);
    }

    public EmailSender cc(String cc) throws MessagingException {
        return addRecipient(cc, Message.RecipientType.CC);
    }

    public EmailSender bcc(String... bcc) throws MessagingException {
        return addRecipients(bcc, Message.RecipientType.BCC);
    }

    public EmailSender bcc(String bcc) throws MessagingException {
        return addRecipient(bcc, Message.RecipientType.BCC);
    }

    private EmailSender addRecipients(String[] recipients, Message.RecipientType
type) throws MessagingException {
        String result = Arrays.asList(recipients).toString().replace("(^\\
[|\\]$)", "").replace(", ", ",");
        msg.setRecipients(type, InternetAddress.parse(result));
        return this;
    }

    private EmailSender addRecipient(String recipient, Message.RecipientType
type) throws MessagingException {
        msg.setRecipients(type, InternetAddress.parse(recipient.replace(";",
",")));
        return this;
    }

    public EmailSender text(String text) {
        this.text = text;
        return this;
    }

    public EmailSender html(String html) {
        this.html = html;
        return this;
    }

    public EmailSender attach(File file) {
        attachments.add(createAttachment(file, null));
        return this;
    }

    public EmailSender attach(File file, String fileName) {
        attachments.add(createAttachment(file, fileName));
        return this;
    }

    public EmailSender attachURL(URL url, String fileName) {
        attachments.add(createURLAttachment(url, fileName));
        return this;
    }

    private MimeBodyPart createAttachment(File file, String fileName) {
        MimeBodyPart attachmentPart = new MimeBodyPart();
```

```
        FileDataSource fds = new FileDataSource(file);
        try {
            attachmentPart.setDataHandler(new DataHandler(fds));
            attachmentPart.setFileName(null == fileName ?
MimeUtility.encodeText(fds.getName()) : MimeUtility.encodeText(fileName));
        } catch (Exception e) {
            e.printStackTrace();
        }
        return attachmentPart;
    }

    private MimeBodyPart createURLAttachment(URL url, String fileName) {
        MimeBodyPart attachmentPart = new MimeBodyPart();

        DataHandler dataHandler = new DataHandler(url);
        try {
            attachmentPart.setDataHandler(dataHandler);
            attachmentPart.setFileName(null == fileName ?
MimeUtility.encodeText(fileName) : MimeUtility.encodeText(fileName));
        } catch (Exception e) {
            e.printStackTrace();
        }
        return attachmentPart;
    }

    public void send() {
        if (text == null && html == null) {
            throw new IllegalArgumentException("At least one context has to be
provided: Text or Html");
        }

        MimeMultipart cover;
        boolean usingAlternative = false;
        boolean hasAttachments = attachments.size() > 0;

        try {
            if (text != null && html == null) {
                // TEXT ONLY
                cover = new MimeMultipart("mixed");
                cover.addBodyPart(textPart());
            } else if (text == null && html != null) {
                // HTML ONLY
                cover = new MimeMultipart("mixed");
                cover.addBodyPart(htmlPart());
            } else {
                // HTML + TEXT
                cover = new MimeMultipart("alternative");
                cover.addBodyPart(textPart());
                cover.addBodyPart(htmlPart());
                usingAlternative = true;
            }

            MimeMultipart content = cover;
            if (usingAlternative && hasAttachments) {
                content = new MimeMultipart("mixed");
                content.addBodyPart(toBodyPart(cover));
            }
```

```
            for (MimeBodyPart attachment : attachments) {
                content.addBodyPart(attachment);
            }

            msg.setContent(content);
            msg.setSentDate(new Date());
            Transport.send(msg);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private MimeBodyPart toBodyPart(MimeMultipart cover) throws
MessagingException {
        MimeBodyPart wrap = new MimeBodyPart();
        wrap.setContent(cover);
        return wrap;
    }

    private MimeBodyPart textPart() throws MessagingException {
        MimeBodyPart bodyPart = new MimeBodyPart();
        bodyPart.setText(text);
        return bodyPart;
    }

    private MimeBodyPart htmlPart() throws MessagingException {
        MimeBodyPart bodyPart = new MimeBodyPart();
        bodyPart.setContent(html, "text/html; charset=utf-8");
        return bodyPart;
    }

    /**
     * 一个发送邮箱验证码的静态方法
     *
     * @param code
     * @param emailAddress
     * @throws Exception
     */
    public static void sendRegisterVerifyCode(String code, String emailAddress)
throws Exception {
        //        邮件

        EmailSender
                .subject("月半湾博客系统注册验证码")
                .from("月半湾博客系统")
                .text("您的验证码是：" + code + "，请在5分钟有效期内完成注册。若非本人操
作，请忽略操作。")
                .to(emailAddress)
                .send();

    }

}
```

3）.接口

```
/**
     * 发送邮件，获取邮箱验证码
     * 通过参数的方式获取。不用result风格
     * <p>
     * 使用场景：注册、找回密码、修改邮箱（输入新的邮箱）
     * 注册：如果已经注册过，就提示该邮箱注册过了
     * 找回密码：如果没有注册过，就提示该邮箱没有注册
     * 修改邮箱（输入新的邮箱）：如果已经注册过，就提示该邮箱注册过了
     *
     * @param emailAddress 防止同一IP轰炸
     * @return
     */
    @ApiOperation("发送邮件，获取邮箱验证码")
    @GetMapping("/send/verify_code")
    public ResponseResult sendVerifyCode(@RequestParam("email") String
emailAddress,
                                         @RequestParam("type") String type) {
        log.info(emailAddress);
        return userService.sendEmail(type, emailAddress);
    }
```

4）.方法

```
/**
     * 发送邮件验证码
     * 使用场景：注册、找回密码、修改邮箱（输入新的邮箱）
     * 注册(register)：如果已经注册过，就提示该邮箱注册过了
     * 找回密码(forget)：如果没有注册过，就提示该邮箱没有注册
     * 修改邮箱（输入新的邮箱）(update)：如果已经注册过，就提示该邮箱注册过了
     *
     * @param type
     * @param emailAddress
     * @return
     */
    @Override
    public ResponseResult sendEmail(String type, String emailAddress) {
        getRequestAndResponse();
        if (StringUtils.isEmpty(emailAddress)) {
            return ResponseResult.FAILED("邮箱地址不可以为空");
        }
        QueryWrapper<User> wrapper =null;
        //根据类型进行查询邮箱是否存在。
        if ("register".equals(type) || "update".equals(type)) {
            wrapper = new QueryWrapper<>();
            wrapper.eq("email",emailAddress);
            User oneByEmail = userMapper.selectOne(wrapper);
            if (!StringUtils.isEmpty(oneByEmail)) {
                return ResponseResult.FAILED("该邮箱已被注册！...");
            }
        } else if ("forget".equals(type)) {
            wrapper = new QueryWrapper<>();
            wrapper.eq("email",emailAddress);
```

```
            User oneByEmail = userMapper.selectOne(wrapper);
            if (StringUtils.isEmpty(oneByEmail)) {
                return ResponseResult.FAILED("该邮箱未被注册！...");
            }
        } else {
            return ResponseResult.FAILED("未标明操作类型");
        }
        //1.防止暴力发送，就是不断发生：同一个邮箱 ，间隔要超过30s ,同一个ip , 1h 最多只能发10次（短信，你最多发3次）。
        //TODO 获取不到IP
        String remoteAddr = request.getRemoteAddr();
//        String remoteAddr = getIpAddr(request);
        if (!StringUtils.isEmpty(remoteAddr)) {
            remoteAddr = remoteAddr.replace(":", "_");
        }
        log.info("sendEmail ==> ip ==> " + remoteAddr);
        //从redis拿出来，如果没有，那就过了 。主要判断 IP 地址 和 发送邮箱地址次数
        Integer ipSendTime = null;

        String o = (String) redisUtil.get(Constants.User.KEY_EMAIL_SEND_IP + remoteAddr);
        if (StringUtils.isEmpty(o)) {
            ipSendTime = null;
        } else {
            ipSendTime = Integer.valueOf(o);
        }

        log.info("ipSendTime : " + (String) redisUtil.get(Constants.User.KEY_EMAIL_SEND_IP + remoteAddr));
        if (!StringUtils.isEmpty(ipSendTime) && ipSendTime > 10) {
            //如果有，判断次数
            return ResponseResult.FAILED("ip...请不要发送太频繁！这验证码还没来得及产生...");
        }
        Object addressSendTime = redisUtil.get(Constants.User.KEY_EMAIL_SEND_ADDRESS + emailAddress);
        log.info((String) redisUtil.get(Constants.User.KEY_EMAIL_SEND_ADDRESS + emailAddress));
        if (!StringUtils.isEmpty(addressSendTime)) {
            return ResponseResult.FAILED("address...请不要发送太频繁！这验证码还没来得及产生...");
        }
        //2.检查邮箱地址是否正确
        boolean isEamil = ValidateUtil.validateEamil(emailAddress);
        if (!isEamil) {
            return ResponseResult.FAILED("邮箱格式不正确");
        }
        //3.发送验证码 , 6位数 ： 100000-999999
        Random random = new Random();
        int code = random.nextInt(999999);
        if (code < 100000) {
            code += 100000;
        }
        try {
            taskService.sendEmailVerifyCode(String.valueOf(code), emailAddress);
        } catch (Exception e) {
            return ResponseResult.FAILED("验证码发送失败,请稍后再试");
        }
```

```
        //4.做记录
        //发送记录：code
        if (StringUtils.isEmpty(ipSendTime)) {
            ipSendTime = 0;
        } else {
            ipSendTime = ipSendTime + 1;
        }
        // 五分钟有效期
        redisUtil.set(Constants.User.KEY_EMAIL_SEND_IP + remoteAddr,
String.valueOf(ipSendTime), 60 * 5);
        //30秒内不让重新发送
        redisUtil.set(Constants.User.KEY_EMAIL_SEND_ADDRESS + emailAddress,
"true", 30);
        //保存code
        redisUtil.set(Constants.User.KEY_EMAIL_CODE_CONTENT + emailAddress,
String.valueOf(code), 60 * 5);
        return ResponseResult.SUCCESS("发送成功！");
    }
```

5）.开启异步操作配置类

```
package com.oldbai.halfmoon.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.EnableAsync;
import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;

import java.util.concurrent.Executor;

/**
 * 开启异步操作
 *
 * @author 老白
 */
@Configuration
@EnableAsync
public class AsyncConfig {

    @Bean
    public Executor asyncExecutor() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        //核心数
        executor.setCorePoolSize(2);
        //最大线程池
        executor.setMaxPoolSize(10);
        //线程名称
        executor.setThreadNamePrefix("bai-hui-hui-email");
        //容量，队列容量
        executor.setQueueCapacity(30);
        //初始化
        executor.initialize();
        return executor;
    }
}
```

6）.异步操作服务类

```
package com.oldbai.halfmoon.util;

import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Service;

/**
 * 专做异步操作的服务类
 * @author 老白
 */
@Service
public class TaskService {
    @Async
    public void sendEmailVerifyCode(String verifyCode, String emailAddress)
throws Exception {
        EmailSender.sendRegisterVerifyCode(verifyCode, emailAddress);
    }
}
```

5.检查邮箱是否注册

1）.接口

```
/**
     * 检查该Email是否已经注册
     *
     * @param email 邮箱地址
     * @return SUCCESS -- > 已经注册了，FAILED ===> 没有注册
     */
    @ApiOperation("检查该Email是否已经注册")
    @ApiResponses({
            @ApiResponse(code = 20000, message = "表示当前邮箱已经注册了"),
            @ApiResponse(code = 40000, message = "表示当前邮箱未注册")
    })
    @GetMapping("/email")
    public ResponseResult checkEmail(@RequestParam("email") String email) {
        return userService.checkEmail(email);
    }
```

2）.方法

```
 @Override
    public ResponseResult checkEmail(String email) {
        QueryWrapper<User> userQueryWrapper = new QueryWrapper<>();
        userQueryWrapper.eq("email", email);
        User oneByEmail = userMapper.selectOne(userQueryWrapper);
        return oneByEmail == null ? ResponseResult.FAILED("该邮箱未注册.") :
ResponseResult.SUCCESS("该邮箱已经注册.");
    }
```

6.检查用户名是否注册

1）.接口

```
/**
     * 检查该用户是否已经注册
     *
     * @param userName 用户名
     * @return SUCCESS -- > 已经注册了，FAILED ===> 没有注册
     */
    @ApiOperation("检查该用户是否已经注册")
    @ApiResponses({
            @ApiResponse(code = 20000, message = "表示用户名已经注册了"),
            @ApiResponse(code = 40000, message = "表示用户名未注册")
    })
    @GetMapping("/check/username")
    public ResponseResult checkUserName(@RequestParam("userName") String
userName) {
        return userService.checkUserName(userName);
    }
```

2）.方法

```
 @Override
    public ResponseResult checkUserName(String userName) {
        QueryWrapper<User> userQueryWrapper = new QueryWrapper<>();
        userQueryWrapper.eq("user_name", userName);
        User oneByEmail = userMapper.selectOne(userQueryWrapper);
        return oneByEmail == null ? ResponseResult.FAILED("该用户名未注册.") :
ResponseResult.SUCCESS("该用户名已经注册.");

    }
```

7.登陆账号

1）.接口

```
 /**
     * 登陆 sign-up
```

```
     * 1.用户提交数据：用户名(邮箱地址)，密码，图灵验证码,图灵验证码的key
     * 2.检查验证码是否正确。
     * 3.通过用户名查找用户，用户是否存在
     * 4.不存在通过邮箱查找，用户是否存在
     * 5.存在，则判断密码是否正确
     * 6.生成token，存入redis,返回登陆结果
     * <p>
     * 客户端想要解析
     * 1.算出token的 md5 值，返回这个 md5 值给客户端，就是 key ,将这个 token 保存到
redis 中。
     * 2.解析过程，用户访问的时候，携带 md5key ，我们从 redis 中拿到 token ， 解析token
就知道用户是否有效，角色是什么......
     * 3.设置有效期
     *
     * @param user        用户对象
     * @param captcha     验证码
     * @param captchaKey 验证码key
     * @return
     */

    @ApiOperation("登陆")
    @PostMapping("/login/{captcha}/{captcha_key}")
    public ResponseResult login(@RequestBody User user,
                                @PathVariable("captcha") String captcha,
                                @PathVariable("captcha_key") String captchaKey)
{

        return userService.login(captcha, captchaKey, user);
    }
```

2）.导入jwt依赖生成token

```
<!--        jwt-->
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.9.1</version>
        </dependency>
```

3）.jwt工具类

```
package com.oldbai.halfmoon.util;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.JwtBuilder;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;

import java.util.Date;
import java.util.Map;

/**
 * jwt生成工具类
 *
 * @author 老白
 */
```

```java
public class JwtUtil {
    /**
     * 盐值,也叫秘钥
     */
    private static String key = "ad128433d8e3356e7024009bf6add2ab";

    //单位是毫秒
//    private static long ttl = Constants.TimeValueInMillions.HOUR_2;//2个小时
    /**
     * 一个小时
     */
    private static long ttl = 60 * 60 * 1000;

    public String getKey() {
        return key;
    }

    public void setKey(String key) {
        JwtUtil.key = key;
    }

    public long getTtl() {
        return ttl;
    }

    public void setTtl(long ttl) {
        JwtUtil.ttl = ttl;
    }

    /**
     * double token 的解决方案。
     * 续费token...
     * 我们可能通过一个  RefreshToken (1个月有效期，保存在数据库里)来生成新的token (2个小时有效期，保存在redis中)
     *
     * @param userId
     * @param ttl
     * @return
     */
    public static String createRefreshToken(String userId, long ttl) {
        long nowMillis = System.currentTimeMillis();
        Date now = new Date(nowMillis);
        JwtBuilder builder = Jwts.builder().
                setId(userId)
                .setIssuedAt(now)
                .signWith(SignatureAlgorithm.HS256, key);
        if (ttl > 0) {
            builder.setExpiration(new Date(nowMillis + ttl));
        }
        return builder.compact();
    }

    /**
     * @param claims 载荷内容
     * @param ttl    有效时长
     * @return
     */
    public static String createToken(Map<String, Object> claims, long ttl) {
```

```
        JwtUtil.ttl = ttl;
        return createToken(claims);
    }

    public static String createToken(String id, String username, Map<String,
Object> claims, long ttl) {
        JwtUtil.ttl = ttl;
        return createToken(id, username, claims);
    }


    /**
     * @param claims 载荷
     * @return token
     */
    public static String createToken(Map<String, Object> claims) {

        long nowMillis = System.currentTimeMillis();
        Date now = new Date(nowMillis);
        JwtBuilder builder = Jwts.builder()
                .setIssuedAt(now)
                .signWith(SignatureAlgorithm.HS256, key);

        if (claims != null) {
            builder.setClaims(claims);
        }

        if (ttl > 0) {
            builder.setExpiration(new Date(nowMillis + ttl));
        }
        return builder.compact();
    }

    public static String createToken(String id, String username, Map<String,
Object> claims) {

        long nowMillis = System.currentTimeMillis();
        Date now = new Date(nowMillis);
        JwtBuilder builder = Jwts.builder()
                .setId(id)
                .setSubject(username)
                .setIssuedAt(now)
                .signWith(SignatureAlgorithm.HS256, key);

        if (claims != null) {
            builder.setClaims(claims);
        }

        if (ttl > 0) {
            builder.setExpiration(new Date(nowMillis + ttl));
        }
        return builder.compact();
    }

    /**
     * 解析token
     *
     * @param jwtStr
```

```
     * @return
     */
    public static Claims parseJWT(String jwtStr) {
        return Jwts.parser()
                .setSigningKey(key)
                .parseClaimsJws(jwtStr)
                .getBody();
    }
}
```

4）.ClaimsUtils工具类

```
package com.oldbai.halfmoon.util;

import com.oldbai.halfmoon.entity.User;
import io.jsonwebtoken.Claims;

import java.util.HashMap;
import java.util.Map;

/**
 * token 的 Claims 工具类
 *
 * @author 老白
 */
public class ClaimsUtils {


    public static final String ID = "id";
    public static final String USERNAME = "user_name";
    public static final String ROLES = "roles";
    public static final String AVATAR = "avatar";
    public static final String EMAIL = "email";
    public static final String SIGN = "sign";

    /**
     * 封装user信息
     * 得到一个jwt载荷
     *
     * @return
     */
    public static Map<String, Object> User2Claims(User user) {
        Map<String, Object> claims = new HashMap<>();
        claims.put(ID, user.getId());
        claims.put(USERNAME, user.getUserName());
        claims.put(ROLES, user.getRoles());
        claims.put(AVATAR, user.getAvatar());
        claims.put(EMAIL, user.getEmail());
        claims.put(SIGN, user.getSign());
        return claims;
    }

    /**
     * 解析jwt载荷
     * 得到user对象
     *
```

```
     * @param claims
     * @return
     */
    public static User claims2ToUser(Claims claims) {
        User user = new User();
        user.setId((String) claims.get(ID));
        user.setUserName((String) claims.get(USERNAME));
        user.setRoles((String) claims.get(ROLES));
        user.setAvatar((String) claims.get(AVATAR));
        user.setEmail((String) claims.get(EMAIL));
        user.setSign((String) claims.get(SIGN));
        return user;
    }
}
```

5）.生成token方法（双token方法）

```
  /**
     * 生成token,抽取出来的方法
     *
     * @param oneByUsername
     * @return
     */
    @Transactional
    String createToken(User oneByUsername) {
        getRequestAndResponse();
        QueryWrapper<RefreshToken> refreshTokenQueryWrapper = new QueryWrapper<>
();
        refreshTokenQueryWrapper.eq("user_id", oneByUsername.getId());
        int i = refreshTokenMapper.delete(refreshTokenQueryWrapper);
        log.info("createNew+" + i);
        //TODO 生成token
        Map<String, Object> cliams = new HashMap<>();
        //默认一个小时
        cliams.put("id", oneByUsername.getId());
        cliams.put("user_name", oneByUsername.getUserName());
        cliams.put("roles", oneByUsername.getRoles());
        cliams.put("avatar", oneByUsername.getAvatar());
        cliams.put("email", oneByUsername.getEmail());
        cliams.put("sign", oneByUsername.getSign());
        String token = JwtUtil.createToken(cliams);
        // 返回一个token 的 md5 值 key ，保存在redis中。
        String tokenKey = DigestUtils.md5DigestAsHex(token.getBytes());
        //保存在redis 中 ， 1个小时有效期 ， key 是 tokenKey
        redisUtil.set(Constants.User.KEY_TOKEN + tokenKey, token,
Constants.TimeValue.HOUR);
        //把token 写入 cookies 里
        Cookie cookie = new Cookie(Constants.User.COOKIE_TOKE_KEY, tokenKey);
        //TODO 动态获取域名，可以从request里获取
        cookie.setDomain("localhost");
        //设置存活时间
        cookie.setPath("/");
        cookie.setMaxAge(Constants.User.COOKIE_TOKE_AGE);
        response.addCookie(cookie);
        // 前端访问时，携带 md5 key ，后端从 redis中取出 token
```

```
        //TODO 生成 refreshToken
        String refreshToken = JwtUtil.createRefreshToken(oneByUsername.getId(),
Constants.TimeValue.MONTH);
        //TODO 保存到数据库中
        // refreshToken tokenKey 用户ID 创建时间 更新时间
        RefreshToken refreshTokenBean = new RefreshToken();
        refreshTokenBean.setId(String.valueOf(idWorker.nextId()));
        refreshTokenBean.setRefreshToken(refreshToken);
        refreshTokenBean.setTokenKey(tokenKey);
        refreshTokenBean.setUserId(oneByUsername.getId());
        refreshTokenBean.setCreateTime(new Date());
        refreshTokenBean.setUpdateTime(new Date());
        refreshTokenMapper.insert(refreshTokenBean);
        return tokenKey;
    }
```

6）.解析token获取user对象

```
/**
     * 通过tokenKey 解析 token 获取 user对象
     *
     * @param tokenKey
     * @return
     */
    private User parseByTokenKey(String tokenKey) {
        String token = (String) redisUtil.get(Constants.User.KEY_TOKEN +
tokenKey);
        if (!StringUtils.isEmpty(token)) {
            try {
                Claims claims = JwtUtil.parseJWT(token);
                return ClaimsUtils.claims2ToUser(claims);
            } catch (Exception e) {
                return null;
            }
        }
        return null;
    }
```

7）.检查用户信息，为security提供检查

```
@Transactional
    @Override
    public User checkUser() {
        getRequestAndResponse();
        //1.拿到tokenKey
        String tokenKey = CookieUtils.getCookie(request,
Constants.User.COOKIE_TOKE_KEY);
        User parseByTokenKey = parseByTokenKey(tokenKey);

        if (StringUtils.isEmpty(parseByTokenKey)) {
            //根据 refreshToken 去判断是否已经登陆过了
            //说明出错了，过期了
            //1.去数据库查询 refreshToken
            QueryWrapper<RefreshToken> tokenQueryWrapper = new QueryWrapper<>();
            tokenQueryWrapper.eq("token_key", tokenKey);
```

```
            RefreshToken refreshToken =
refreshTokenMapper.selectOne(tokenQueryWrapper);
            //2.如果不存在，就重新登陆
            if (StringUtils.isEmpty(refreshToken)) {
                return null;
            }
            //3.如果存在，就解析 refreshToken
            try {
                Claims claims =
JwtUtil.parseJWT(refreshToken.getRefreshToken());
                //5.如果 refreshToken 有效，创建新的token 和新的 refreshToken
                QueryWrapper<User> userQueryWrapper = new QueryWrapper<>();
                userQueryWrapper.eq("id", refreshToken.getUserId());
                User user = userMapper.selectOne(userQueryWrapper);
                //千万别这么干，事务还没提交，这样做数据库密码就没了
                // user.setPassword("");
                // 删掉 refreshToken 的记录
                tokenQueryWrapper = new QueryWrapper<>();
                tokenQueryWrapper.eq("id", refreshToken.getId());
                refreshTokenMapper.delete(tokenQueryWrapper);
                String tokenKey1 = createToken(user);
                //返回token
                return parseByTokenKey(tokenKey1);
            } catch (Exception exception) {
                //4.如果 refreshToken 过期了，就当前访问没有登录，提示用户登录
                return null;
            }
        }

        return parseByTokenKey;
    }
```

8）.开始登陆

```
/**
     * 登陆
     *
     * @param captcha
     * @param captchaKey
     * @param user
     * @return
     */
    @Override
    public ResponseResult login(String captcha, String captchaKey, User user) {
        getRequestAndResponse();
        log.info(Constants.User.KEY_CAPTCHA_CONTENT);
        //1.验证图灵验证码
        String code = (String) redisUtil.get(Constants.User.KEY_CAPTCHA_CONTENT
+ captchaKey);
        if (!captcha.equals(code)) {
            return ResponseResult.FAILED("登陆验证码错误......");
        }
        //验证成功，删除
        redisUtil.del(Constants.User.KEY_CAPTCHA_CONTENT + captchaKey);
        //2.用户名
        String username = user.getUserName();
        if (StringUtils.isEmpty(username)) {
```

```
            return ResponseResult.FAILED("登陆用户名不为空......");
        }
        //3.密码
        String password = user.getPassword();
        if (StringUtils.isEmpty(password)) {
            return ResponseResult.FAILED("登陆密码不为空......");
        }
        //4.查用户，有可能是账号，有可能是邮箱
        QueryWrapper<User> wrapper = new QueryWrapper<>();
        wrapper.eq("user_name", username);
        User oneByUsername = userMapper.selectOne(wrapper);
        if (StringUtils.isEmpty(oneByUsername)) {
            wrapper = new QueryWrapper<>();
            wrapper.eq("email", user.getEmail());
            oneByUsername = userMapper.selectOne(wrapper);
            if (StringUtils.isEmpty(oneByUsername)) {
                return ResponseResult.FAILED("用户名或密码错误......");
            }
        }
        //用户存在
        //对比密码
        boolean matches = BCrypt.checkpw(password, oneByUsername.getPassword());
        if (!matches) {
            return ResponseResult.FAILED("用户名或密码错误......");
        }
        //密码正确
        //判断用户状态是否正常。
        if (!oneByUsername.getState().equals("1")) {
            return ResponseResult.GET(ResponseState.ACCOUNT_FORBID);
        }

        createToken(oneByUsername);

        return ResponseResult.SUCCESS("登陆成功");
    }
```

9）.登陆逻辑较为复杂，以下是缕缕思路

> # 一、登陆思路：
>
> 1.首先进行验证码的验证
>
> 2.验证成功验证码后将存在redis里的验证码删除
>
> 3.检查用户名
>
> 4.检查密码
>
> 5.然后用用户名进行数据库查询，如果用户名差不多就用邮箱查，都查不到就是用户名不正确。
>
> 6.如果用户存在，那么咱们就进行密码比对。
>
> 7.密码正确咱们就看看当前用户是否可以使用
>
> 8.然后生成token。
>
> # 二、生成token思路：

1.需要传入一个用户名

2.我们需要知道是不是有一个**refreshToken**是存在数据库的，所以我们先查数据库看看是不是有这个**refreshToken**。

3.如果有这个**refreshToken**，我们就把这个token给删除了。

4.我们就需要重新生成这个**refreshToken**

5.使用cliams工具类生成存储信息的主体

6.然后生成一个**token**存入redis

7.将这个**token**存入cookie返回给前端，注意呀，返回给前端的是token-key

8.然后生成一个**refreshToken**存入数据库

三、检查用户信息思路：

1.首先先从请求里面取出cookie，看看cookie里面是否存有token。我们此时从cookie里得到的是一个tokenkey。

2.我们通过这个tokenkey从redis里找看看有没有这个token

3.注意喔，我们存入redis的是token，但是传给前端的是tokenkey，我们需要通过这个key从redis中取出token

4.如果我们拿到了这个token，然后进行解析。看看它过期了没有。

5.如果它没过期我们就能得到这个用户信息，返回给前端了。

5.如果它过期了，那么我们就去数据库查，看看有没有这个token

6.如果我们找不到，那表示你就真的登陆过期了。

7.如果找到了，解析它，拿到这个储存信息的**claims**。主要是判断过没过期。

8.我们就通过数据库的**refreshToken**拿到用户id。

9.通过用户id得到用户信息然后重新生成存在redis的**token**和数据库的**refreshToken**

10.最后我们将这个token里存的用户信息封装成一个user返回回去。

8.修改密码

1）.接口

```
/**
     * 修改密码password
     * 普通做法：通过旧密码对比来更新密码
     * <p>
     * 找回密码：既可以找回密码又可以修改密码
     * 发送验证码到邮箱/手机-->判断验证码是否正确来判断
     * 对应的邮箱/手机号码所注册的账号是否属于你
     * <p>
     * 步骤
     * 1.用户填写邮箱
     * 2.获取验证码 type = forget
     * 3.用户填写新的密码
     * 4.填写验证码
     * 5.提交数据
```

```
     * <p>
     * 需要提交的参数
     * 1.邮箱
     * 2.新密码
     * 3.验证码
     * <p>
     * 如果验证码正确，所有邮箱注册的账号就是你的，可以修改密码
     *
     * @param user
     * @return
     */
    @ApiOperation("修改密码")
    @PostMapping("/update/password/{verifyCode}")
    public ResponseResult updatePassword(@PathVariable("verifyCode") String
verifyCode,
                                         @RequestBody User user) {
        return userService.updateUserPassword(verifyCode, user);
    }
```

2）.方法

```
/**
     * 更新密码
     *
     * @param verifyCode
     * @param user
     * @return
     */
    @Override
    public ResponseResult updateUserPassword(String verifyCode, User user) {
        getRequestAndResponse();
        //检查邮箱是否有填写
        if (StringUtils.isEmpty(user.getEmail())) {
            return ResponseResult.FAILED("邮箱不可以为空");
        }
        //根据邮箱去 rdis 拿验证码
        String code = (String)
redisUtil.get(Constants.User.KEY_EMAIL_CODE_CONTENT + user.getEmail());
        //进行比对
        if (StringUtils.isEmpty(verifyCode) || !verifyCode.equals(code)) {
            return ResponseResult.FAILED("验证码错误......");
        }
        //干掉资源,删除redis里的验证码
        redisUtil.del(Constants.User.KEY_EMAIL_CODE_CONTENT + user.getEmail());
        //修改密码
        QueryWrapper<User> wrapper = new QueryWrapper<User>();
        wrapper.eq("id", user.getId()).eq("email",user.getEmail());
        user.setPassword(BCrypt.hashpw(user.getPassword()));
        int i = userMapper.update(user, wrapper);
        return i > 0 ? ResponseResult.SUCCESS("密码修改成功") :
ResponseResult.FAILED("密码修改失败");
    }
```

3）.测试数据

```
{
"id":"0feed2d15abb871103ba9ee846aca914",
  "email": "1005777562@qq.com",
  "password": "admin",
  "userName": "admin"
}
```

9.获取用户信息

1）.接口

```
/**
     * 获取用户信息
     *
     * @return
     */
    @ApiOperation("获取用户信息")
    @GetMapping("/get/user_info/{userId}")
    public ResponseResult getUserInfo(@PathVariable("userId") String userId) {
        //1.用户ID
        return userService.getUserInfo(userId);
    }
```

2）.方法

```
/**
     * 获取用户信息
     *
     * @param userId
     * @return
     */
    @Override
    public ResponseResult getUserInfo(String userId) {
        getRequestAndResponse();
        //1.从数据库获取
        QueryWrapper<User> userQueryWrapper = new QueryWrapper<>();
        userQueryWrapper.eq("id", userId);

        User one = userMapper.selectOne(userQueryWrapper);
        //判断结果
        if (StringUtils.isEmpty(one)) {
            //如果不存在
            return ResponseResult.FAILED("用户不存在......");
        }
        //如果存在，就复制对象清空密码、email、登陆IP 注册IP
        String userJson = gson.toJson(one);
        User newUser = gson.fromJson(userJson, User.class);
        newUser.setPassword("");
        newUser.setAvatar("");
        newUser.setRegIp("");
        newUser.setLoginIp("");
        //返回结果
        return ResponseResult.SUCCESS("获取成功", newUser);
    }
```

3）.导入GSON依赖，用于复制对象

```
<!--        gson-->
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.8.5</version>
        </dependency>
```

10.修改用户信息

1）.接口

```
/**
     * 修改用户信息
     * <p>
     * 允许用户修改的内容:
     * 1.头像 ()
     * 2.用户名 (唯一的)
     * 3.密码 (单独更新)
     * 4.签名 ()
     * 5.用户邮箱 (单独修改，唯一的)
     *
     * @param user
     * @return
     */
    @ApiOperation("修改用户信息")
    @PostMapping("/update/user_info/{userId}")
    public ResponseResult updateUserInfo(@RequestBody User user,
                                         @PathVariable("userId") String userId)
{
        return userService.updateUserInfo(userId, user);
    }
```

2）.方法

```
  /**
     * 修改用户信息
     * TODO 还需要改进。比如说，只修改某个属性
     *
     * @param userId
     * @param user
     * @return
     */
    @Transactional
    @Override
    public ResponseResult updateUserInfo(String userId, User user) {
        getRequestAndResponse();
        //1.检查用户是否登录
        User checkUserKey = checkUser();
        if (StringUtils.isEmpty(checkUserKey)) {
            return ResponseResult.NO_LOGIN("用户未登录.....");
        }
        //从数据库中获取
```

```
        QueryWrapper<User> wrapper = new QueryWrapper<>();
        wrapper.eq("id", checkUserKey.getId());
        User checkUser = userMapper.selectOne(wrapper);
        //2.判断用户的Id是否一致，如果一致才可以修改
        if (checkUser.equals(userId)) {
            return ResponseResult.NO_PERMISSION("无权修改......");
        }
        //3.可以修改
        //可以修改的位置
        //用户名 ，用户名不为空且数据库无改用户名注册过
        wrapper = new QueryWrapper<>();
        wrapper.eq("user_name", user.getUserName());
        if (!StringUtils.isEmpty(user.getUserName()) &&
StringUtils.isEmpty(userMapper.selectOne(wrapper))) {
            checkUser.setUserName(user.getUserName());
        } else {
            return ResponseResult.FAILED("该用户名已被注册无法修改......");
        }
        //头像
        if (!StringUtils.isEmpty(user.getAvatar())) {
            checkUser.setAvatar(user.getAvatar());
        }
        //签名，可以为空
        checkUser.setSign(user.getSign());
        wrapper = new QueryWrapper<>();
        wrapper.eq("id", checkUser.getId());
        userMapper.update(checkUser, wrapper);
        //TODO 干掉redis里的token ,下一次请求，跟用户有关的需要解析token，就会根据
refreshToken创建一个
        //拿到key
        String tokenKey = CookieUtils.getCookie(request,
Constants.User.COOKIE_TOKE_KEY);
        redisUtil.del(Constants.User.KEY_TOKEN + tokenKey);
        return ResponseResult.SUCCESS("修改成功......");
    }
```

3）.测试数据

```
{
    "userName": "admin",
    "avatar": "",
    "email": "1005777562@qq.com",
    "sign": "我就是我"
}
```

11.获取用户集合

1）.接口

```
/**
     * 获取用户集合
     * 分页查询
     * 需要管理员权限
     *
     * @param page
```

```
     * @param size
     * @return ------
     * 权限注解 @PreAuthorize("@permission.adminPermission()")
     */
    @PreAuthorize("@permission.adminPermission()")
    @ApiOperation("获取用户集合")
    @GetMapping("/user/list")
    public ResponseResult listUser(@RequestParam("page") int page,
                                   @RequestParam("size") int size
    ) {
        return userService.listUsers(page, size);
    }
```

2）.方法

```
/**
     * 获取用户列表，需要管理员权限
     *
     * @param page
     * @param size
     * @return
     */
    @Transactional
    @Override
    public ResponseResult listUsers(int page, int size) {
        getRequestAndResponse();
        //可以获取用户列表
        //分页查询
        //判断page
        page = Utils.getPage(page);
        //size也限制一下,每一页不小于5个
        size = Utils.getSize(size);
        //分页开始
        //根据注册日期排序
        Page<UserView> pageObj = new Page<>(page,size);
        QueryWrapper<UserView> viewQueryWrapper = new QueryWrapper<>();
        viewQueryWrapper.orderByDesc("create_time");
        Page<UserView> all = userViewMapper.selectPage(pageObj,
viewQueryWrapper);
        //TODO 处理密码问题，在这使用了视图
        return ResponseResult.SUCCESS("获取用户列表成功", all);
    }
```

3）.mybatis plus 开启分页配制

```
 @Bean
    public PaginationInterceptor paginationInterceptor() {
        PaginationInterceptor paginationInterceptor = new
PaginationInterceptor();
        // 设置请求的页面大于最大页后操作， true调回到首页，false 继续请求  默认false
        // paginationInterceptor.setOverflow(false);
        // 设置最大单页限制数量，默认 500 条，-1 不受限制
        // paginationInterceptor.setLimit(500);
        // 开启 count 的 join 优化,只针对部分 left join
        paginationInterceptor.setCountSqlParser(new
JsqlParserCountOptimize(true));
        return paginationInterceptor;
    }
```

4）.小记录

> orderByDesc 从最新（大）开始排序
>
> orderByAsc 从最旧（小）开始排序

12.删除用户（假删除）

1）.接口

```
/**
     * 删除用户
     * 需要管理员权限
     *
     * @param userId
     * @return
     */
    @PreAuthorize("@permission.adminPermission()")
    @ApiOperation("删除用户")
    @GetMapping("/delete/user/{userId}")
    public ResponseResult deleteUser(@PathVariable("userId") String userId) {
        //判断当前操作的用户是谁
        //根据用户角色判断是否可以删除
        //TODO :通过注解的方式来控制权限

        return userService.deleteUserById(userId);
    }
```

2）.方法

```
@Override
    public ResponseResult deleteUserById(String userId) {
        getRequestAndResponse();
        //可以删除用户了......
        int result = userMapper.deleteById(userId);
        if (result > 0) {
            return ResponseResult.SUCCESS("删除成功");
        } else {

            return ResponseResult.FAILED("用户不存在......");
        }
    }
```

13.更新邮箱

1）.接口

```
/**
     * 1、必须已经登录了
     * 2、新的邮箱没有注册过
     * <p>
     * 用户的步骤：
     * 1、已经登录
     * 2、输入新的邮箱地址
     * 3、获取验证码 type=update
     * 4、输入验证码
     * 5、提交数据
     * <p>
     * 需要提交的数据
     * 1、新的邮箱地址
     * 2、验证码
     * 3、其他信息我们可以token里获取
     *
     * @return
     */
    @ApiOperation("更新邮箱")
    @PostMapping("/update/email")
    public ResponseResult updateEmail(@RequestParam("email") String email,
                                      @RequestParam("verify_code") String
verifyCode) {
        return userService.updateEmail(email, verifyCode);
    }
```

2）.方法

```
@Override
    public ResponseResult updateEmail(String email, String verifyCode) {
        getRequestAndResponse();
        //1、确保用户已经登录了
        User sobUser = checkUser();
        //没有登录
        if (sobUser == null) {
            return ResponseResult.NO_LOGIN("没有登录");
        }
```

```
        //2、对比验证码，确保新的邮箱地址是属于当前用户的
        String redisVerifyCode = (String)
redisUtil.get(Constants.User.KEY_EMAIL_CODE_CONTENT + email);
        if (StringUtils.isEmpty(redisVerifyCode) ||
!redisVerifyCode.equals(verifyCode)) {
            return ResponseResult.FAILED("验证码错误");
        }
        //验证完成,删除验证码
        redisUtil.del(Constants.User.KEY_EMAIL_CODE_CONTENT + email);
        //2768011423
        //可以修改邮箱
        User user = new User();
        user.setId(sobUser.getId());
        user.setEmail(email);
        int result = userMapper.updateById(user);
        return result > 0 ? ResponseResult.SUCCESS("邮箱修改成功") :
ResponseResult.FAILED("邮箱修改失败");
    }
```

14.退出登录

1）.接口

```
/**
     * 退出登录
     * <p>
     * 拿到token_key
     * -> 删除redis里对应的token
     * -> 删除mysql里对应的refreshToken
     * -> 删除cookie里的token_key
     *
     * @return
     */
    @ApiOperation("退出登录")
    @GetMapping("/logout")
    public ResponseResult logout() throws NotLoginException {
        return userService.doLogout();
    }
```

2）.方法

```
/**
     * 退出登录
     *
     * @return
     */
    @Override
    public ResponseResult doLogout() {
        getRequestAndResponse();
        //拿到token_key
        String tokenKey = CookieUtils.getCookie(request,
Constants.User.COOKIE_TOKE_KEY);
        if (StringUtils.isEmpty(tokenKey)) {
            return ResponseResult.NO_LOGIN();
        }
```

```
        //删除redis里的token
        redisUtil.del(Constants.User.KEY_TOKEN + tokenKey);
        //删除mysql里的refreshToken
        QueryWrapper<RefreshToken> wrapper = new QueryWrapper<>();
        wrapper.eq("token_key", tokenKey);
        refreshTokenMapper.delete(wrapper);
        //删除cookie里的token_key
        CookieUtils.deleteCookie(response, Constants.User.COOKIE_TOKE_KEY);
        return ResponseResult.SUCCESS("退出登录成功.");
    }
```

九、整合Security，只进行最简单的角色认证。

1）.导入依赖

```
<!--        securtiy-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
            <version>2.3.2.RELEASE</version>
        </dependency>
```

2）.编写配置文件

```
package com.oldbai.halfmoon.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.method.configuration.EnableGlobal
MethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecuri
ty;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConf
igurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;


/**
 * SpringSecurity的配置
 *
```

```
 * @author 老白
 */
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //先把所有先放行
        http.authorizeRequests()
                .antMatchers("/**").permitAll()
                .anyRequest().authenticated()
                .and()
                .csrf().disable();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

3）.编写一个认证的过滤器

```
package com.oldbai.halfmoon.util;

import com.oldbai.blog.utils.Constants;
import com.oldbai.halfmoon.entity.User;
import com.oldbai.halfmoon.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.util.StringUtils;
import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.context.request.ServletRequestAttributes;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * 简单版权限管理
 *
 * @author 老白
 */
@Service("permission")
public class PermissionCheckService {

    @Autowired
    private UserService userService;

    /**
     * 判断是不是管理员
     *
     * @return
     */
    public boolean adminPermission() {
```

```
        //拿到 request  response
        // 获取到当前权限所有的角色，进行角色对比即可确定权限
        HttpServletRequest request = ((ServletRequestAttributes)
RequestContextHolder.getRequestAttributes()).getRequest();
        HttpServletResponse response = ((ServletRequestAttributes)
RequestContextHolder.getRequestAttributes()).getResponse();
        //如果token返回false
        String token = CookieUtils.getCookie(request,
Constants.User.COOKIE_TOKE_KEY);
        //没有令牌的key ，没有登录，不用往下执行了
        if (StringUtils.isEmpty(token)) {
            return false;
        }
        User sobUser = userService.checkUser();
        if (sobUser == null || StringUtils.isEmpty(sobUser.getRoles())) {
            return false;
        }
        if (Constants.User.ROLE_ADMIN.equals(sobUser.getRoles())) {
            return true;
        }
        return false;
    }

}
```

4）.未登录的异常处理(在统一异常管理内已配置)

```
package com.oldbai.halfmoon.exception;

/**
 * 未登录异常处理
 *
 * @author 老白
 */
public class NotLoginException extends Exception {

    @Override
    public String toString() {
        return "not login.";
    }
}
```

5）.cookie工具类

```
package com.oldbai.blog.utils;

import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * cookies 工具类
 *
 * @author 老白
 */
```

```java
public class CookieUtils {
    /**
     * 存活时间
     */
    public static final int default_age = Constants.User.COOKIE_TOKE_AGE;

    public static final String domain = "localhost";

    /**
     * 设置cookie值
     *
     * @param response
     * @param key
     * @param value
     */
    public static void setUpCookie(HttpServletResponse response, String key,
String value) {
        setUpCookie(response, key, value, default_age);
    }

    public static void setUpCookie(HttpServletResponse response, String key,
String value, int age) {
        Cookie cookie = new Cookie(key, value);
        cookie.setPath("/");
        cookie.setDomain(domain);
        cookie.setMaxAge(age);
        response.addCookie(cookie);
    }

    /**
     * 删除cookie
     *
     * @param response
     * @param key
     */
    public static void deleteCookie(HttpServletResponse response, String key) {
        setUpCookie(response, key, null, 0);
    }

    /**
     * 获取cookie
     *
     * @param request
     * @param key
     * @return
     */
    public static String getCookie(HttpServletRequest request, String key) {
        Cookie[] cookies = request.getCookies();
        //TODO cookie为空的状况？
        for (Cookie cookie : cookies) {
            if (key.equals(cookie.getName())) {
                return cookie.getValue();
            }
        }
        return null;
    }
}
```

十、管理中心分类模块API

1.添加分类

1）.接口

```
/**
     * 添加分类
     *
     * @return
     */
    @ApiOperation("添加分类")
    @PreAuthorize("@permission.adminPermission()")
    @PostMapping("/add_categories")
    public ResponseResult addCategory(@RequestBody Category category) {
        return categoryService.addCategory(category);
    }
```

2）.方法

```
@Transactional
    @Override
    public ResponseResult addCategory(Category category) {
        //TODO 还可以进行完善
        //先检查数据
        // 必须的数据有：
        //分类名称、分类的pinyin、顺序、描述
        if (StringUtils.isEmpty(category.getName())) {
            return ResponseResult.FAILED("分类名称不可以为空.");
        }
        if (StringUtils.isEmpty(category.getPinyin())) {
            return ResponseResult.FAILED("分类拼音不可以为空.");
        }
        if (StringUtils.isEmpty(category.getDescription())) {
            return ResponseResult.FAILED("分类描述不可以为空.");
        }
        //补全数据
//        category.setId(idWorker.nextId() + "");
        category.setStatus("1");
        category.setOrder(0);
//        category.setCreateTime(new Date());
//        category.setUpdateTime(new Date());
        //保存数据
        categoryMapper.insert(category);
        //返回结果
        return ResponseResult.SUCCESS("添加分类成功");
    }
```

3）.测试数据

```
{
  "description": "java编程",
  "name": "Java",
  "pinyin": "java"
}
```

2.删除分类

1）.接口

```
/**
     * 删除分类
     *
     * @param categoryId
     * @return
     */
    @PreAuthorize("@permission.adminPermission()")
    @ApiOperation("删除分类")
    @GetMapping("/delete_categories/{categoryId}")
    public ResponseResult deleteCategory(@PathVariable("categoryId") String
categoryId) {
        return categoryService.deleteCategory(categoryId);
    }
```

2）.方法

```
@Override
    public ResponseResult deleteCategory(String categoryId) {
        int result = categoryMapper.deleteById(categoryId);
        if (result == 0) {
            return ResponseResult.FAILED("该分类不存在.");
        }
        return ResponseResult.SUCCESS("删除分类成功.");
    }
```

3.更新分类

1）.接口

```
/**
     * 更新分类
     *
     * @param category
     * @param categoryId
     * @return
     */
    @ApiOperation("更新分类")
    @PreAuthorize("@permission.adminPermission()")
    @PostMapping("/update_categories/{categoryId}")
    public ResponseResult updateCategory(@RequestBody Category category,
```

```
                                        @PathVariable("categoryId") String
categoryId) {
        return categoryService.updateCategory(categoryId, category);
    }
```

2）.方法

```
@Override
    public ResponseResult updateCategory(String categoryId, Category category) {
        //1.找出来
        Category one = categoryMapper.selectById(categoryId);
        if (StringUtils.isEmpty(one)) {
            return ResponseResult.FAILED("分类不存在");
        }
        //2.对内容进行判断
        if (!StringUtils.isEmpty(category.getName())) {
            one.setName(category.getName());
        }
        if (!StringUtils.isEmpty(category.getDescription())) {
            one.setDescription(category.getDescription());
        }
        if (!StringUtils.isEmpty(category.getPinyin())) {
            one.setPinyin(category.getPinyin());
        }
        if (!StringUtils.isEmpty(category.getOrder())) {
            one.setOrder(category.getOrder());
        }
        //3.保存数据
        categoryMapper.updateById(one);
        //4.返回结果
        return ResponseResult.SUCCESS("保存成功...");
    }
```

3）.测试数据

```
{

    "name": "spring",
    "pinyin": "spring",
    "description": "spring框架，非常重要",
    "order": 0
  }
```

4.获取分类

1）.接口

```
/**
     * 获取分类
     * <p>
     * 使用的case:修改的时候，获取一下。填充弹窗
     * 不获取也是可以的，从列表里获取数据
```

```
     * <p>
     * 权限：管理员权限
     *
     * @param categoryId
     * @return
     */
    @PreAuthorize("@permission.adminPermission()")
    @ApiOperation("获取分类")
    @GetMapping("/get_categories/{categoryId}")
    public ResponseResult getCategory(@PathVariable("categoryId") String
categoryId) {
        return categoryService.getCategory(categoryId);
    }
```

2）.方法

```
@Override
    public ResponseResult getCategory(String categoryId) {
        Category category = categoryMapper.selectById(categoryId);
        if (category == null) {
            return ResponseResult.FAILED("分类不存在.");
        }
        return ResponseResult.SUCCESS("获取分类成功.", category);
    }
```

5.获取分类列表

1）.接口

```
/**
     * 获取分类列表
     * <p>
     * 权限：管理员权限
     *
     * @return
     */
    @ApiOperation("获取分类列表")
    @PreAuthorize("@permission.adminPermission()")
    @GetMapping("/list")
    public ResponseResult listCategory() {
        return categoryService.listCategories();
    }
```

2）.方法

```
@Override
    public ResponseResult listCategories() {
        //创建条件

        //判断用户角色，普通用户 未登录用户 只能获取到正常的category
        //管理员账号 可以拿到所有的分类
        User checkUser = userService.checkUser();
        List<Category> all = null;
```

```
        if (StringUtils.isEmpty(checkUser) ||
!com.oldbai.blog.utils.Constants.User.ROLE_ADMIN.equals(checkUser.getRoles())) {
            //只能获取到正常的category
            QueryWrapper<Category> queryWrapper = new QueryWrapper<>();
            queryWrapper.eq("status", 1).orderByDesc("update_time");
            all = categoryMapper.selectList(queryWrapper);
        } else {
            //查询
            all = categoryMapper.selectList(null);
        }
        //返回结果
        return ResponseResult.SUCCESS("获取分类列表成功.", all);
    }
```

十一、管理中心友情链接模块API

1.添加友情链接

1）.接口

```
/**
     * 增
     *
     * @return
     */
    @ApiOperation("添加友情链接")
    @PreAuthorize("@permission.adminPermission()")
    @PostMapping("/upload")
    public ResponseResult uploadFriendLink(@RequestBody FriendLink friendLink) {
        return friendLinkService.addFriendLink(friendLink);
    }
```

2）.方法

```
 /**
     * 添加友情连接
     *
     * @param friendLink
     * @return
     */
    @Override
    public ResponseResult addFriendLink(FriendLink friendLink) {
        //判断数据
        String url = friendLink.getUrl();
        if (StringUtils.isEmpty(url)) {
            return ResponseResult.FAILED("链接Url不可以为空.");
        }
        String logo = friendLink.getLogo();
        if (StringUtils.isEmpty(logo)) {
            return ResponseResult.FAILED("logo不可以为空.");
        }
        String name = friendLink.getName();
        if (StringUtils.isEmpty(name)) {
            return ResponseResult.FAILED("对方网站名不可以为空.");
```

```
        }
        //补全数据
//        friendLink.setId(idWorker.nextId() + "");
//        friendLink.setUpdateTime(new Date());
//        friendLink.setCreateTime(new Date());
        friendLink.setState("1");
        //保存数据
        friendLinkMapper.insert(friendLink);
        //返回结果
        return ResponseResult.FAILED("添加成功.");
    }
```

3）.测试数据

```
{

  "logo": "null",
  "name": "baiblog12345678910",
  "order": 0,
  "url": "oldbai.top"
}
```

2.删除友情链接

1）.接口

```
/**
     * 删
     *
     * @param friendLinkId
     * @return
     */
    @ApiOperation("删除友情链接")
    @PreAuthorize("@permission.adminPermission()")
    @GetMapping("/delete/{friendLinkId}")
    public ResponseResult deleteFriendLink(@PathVariable("friendLinkId") String
friendLinkId) {
        return friendLinkService.deleteFriendLink(friendLinkId);
    }
```

2）.方法

```
@Override
    public ResponseResult deleteFriendLink(String friendLinkId) {
        int result = friendLinkMapper.deleteById(friendLinkId);
        if (result == 0) {
            return ResponseResult.FAILED("删除失败.");
        }
        return ResponseResult.SUCCESS("删除成功.");
    }
```

3.更新友情链接

1）.接口

```
/**
     * 改
     *
     * @param friendLinkId
     * @return
     */
    @ApiOperation("更新友情链接")
    @PreAuthorize("@permission.adminPermission()")
    @PostMapping("/update/{friendLinkId}")
    public ResponseResult updateFriendLink(@PathVariable("friendLinkId") String
friendLinkId,
                                           @RequestBody FriendLink friendLink) {
        return friendLinkService.updateFriendLink(friendLinkId, friendLink);
    }
```

2）.方法

```
 /**
     * 更新内容有什么：
     * logo
     * 对方网站的名称
     * url
     * order
     *
     * @param friendLinkId
     * @param friendLink
     * @return
     */
    @Override
    public ResponseResult updateFriendLink(String friendLinkId, FriendLink
friendLink) {
        FriendLink friendLinkFromDb = friendLinkMapper.selectById(friendLinkId);
        if (friendLinkFromDb == null) {
            return ResponseResult.FAILED("更新失败.");
        }
        String logo = friendLink.getLogo();
        if (!StringUtils.isEmpty(logo)) {
            friendLinkFromDb.setLogo(logo);
        }
        String name = friendLink.getName();
        if (!StringUtils.isEmpty(name)) {
            friendLinkFromDb.setName(name);
        }
        String url = friendLink.getUrl();
        if (!StringUtils.isEmpty(url)) {
            friendLinkFromDb.setUrl(url);
        }
        friendLinkFromDb.setOrder(friendLink.getOrder());
        friendLinkFromDb.setUpdateTime(new Date());
        //保存数据
        friendLinkMapper.updateById(friendLinkFromDb);
```

```
        return ResponseResult.SUCCESS("更新成功.");
    }
```

3）.测试数据

```
{
    "name": "abaiblog1234567",
    "logo": "null",
    "url": "oldbai.top"
  }
```

4.获取友情链接

1）.接口

```
/**
     * 查
     *
     * @param friendLinkId
     * @return
     */
    @ApiOperation("获取友情链接")
    @PreAuthorize("@permission.adminPermission()")
    @GetMapping("/get_image/{friendLinkId}")
    public ResponseResult getFriendLink(@PathVariable("friendLinkId") String
friendLinkId) {
        return friendLinkService.getFriendLink(friendLinkId);
    }
```

2）.方法

```
 @Override
    public ResponseResult getFriendLink(String friendLinkId) {
        FriendLink friendLink = friendLinkMapper.selectById(friendLinkId);
        if (friendLink == null) {
            return ResponseResult.FAILED("友情链接不存");
        }
        return ResponseResult.SUCCESS("获取成功", friendLink);
    }
```

5.获取友情链接集合

1）.接口

```
/**
     * 获取集合
     *
     * @return
     */
    @ApiOperation("获取友情链接集合")
    @PreAuthorize("@permission.adminPermission()")
    @GetMapping("/list")
    public ResponseResult listFriendLinks() {
        return friendLinkService.listFriendLinks();
    }
```

2）.方法

```
@Override
    public ResponseResult listFriendLinks() {
        //创建条件
        List<FriendLink> all;
        User sobUser = userService.checkUser();
        QueryWrapper<FriendLink> friendLinkQueryWrapper = null;
        if (sobUser == null ||
!com.oldbai.blog.utils.Constants.User.ROLE_ADMIN.equals(sobUser.getRoles())) {
            //只能获取到正常的category
            friendLinkQueryWrapper = new QueryWrapper<>();
            friendLinkQueryWrapper.eq("state","1").orderByDesc("update_time");
            all = friendLinkMapper.selectList(friendLinkQueryWrapper);
        } else {
            //查询
            friendLinkQueryWrapper = new QueryWrapper<>();
            friendLinkQueryWrapper.orderByDesc("update_time");
            all = friendLinkMapper.selectList(friendLinkQueryWrapper);
        }
        return ResponseResult.SUCCESS("获取列表成功.", all);
    }
```

十二、管理中心图片模块API

1.使用阿里云OSS云存储

1）.导入依赖

```
<!--        上传图片 时间依赖包-->
        <dependency>
            <groupId>joda-time</groupId>
            <artifactId>joda-time</artifactId>
            <version>2.10.1</version>
        </dependency>
        <!-- 阿里云oss上传图片依赖 -->
        <dependency>
            <groupId>com.aliyun.oss</groupId>
            <artifactId>aliyun-sdk-oss</artifactId>
            <version>3.10.2</version>
        </dependency>
```

2）.添加阿里云配制

```
#阿里云 OSS
#不同的服务器，地址不同
aliyun.oss.file.endpoint=
aliyun.oss.file.keyid=
aliyun.oss.file.keysecret=
#bucket可以在控制台创建，也可以使用java代码创建
aliyun.oss.file.bucketname=oldbai-flie
spring.servlet.multipart.maxFileSize=30MB
spring.servlet.multipart.maxRequestSize=30MB
```

3）.编写上传工具服务类

```
package com.oldbai.halfmoon.oss;

import org.springframework.beans.factory.InitializingBean;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

/**
 * 当项目已启动，spring接口，spring加载之后，执行接口一个方法
 */
@Component
public class ConstantPropertiesUtils implements InitializingBean {

    /**
     * 读取配置文件内容
     */
    @Value("${aliyun.oss.file.endpoint}")
    private String endpoint;

    @Value("${aliyun.oss.file.keyid}")
    private String keyId;

    @Value("${aliyun.oss.file.keysecret}")
    private String keySecret;

    @Value("${aliyun.oss.file.bucketname}")
    private String bucketName;

    /**
```

```
     * 定义公开静态常量
     */
    public static String END_POIND;
    public static String ACCESS_KEY_ID;
    public static String ACCESS_KEY_SECRET;
    public static String BUCKET_NAME;

    @Override
    public void afterPropertiesSet() throws Exception {
        END_POIND = endpoint;
        ACCESS_KEY_ID = keyId;
        ACCESS_KEY_SECRET = keySecret;
        BUCKET_NAME = bucketName;
    }
}
```

```
package com.oldbai.halfmoon.oss;

import org.springframework.web.multipart.MultipartFile;

public interface OssService {
    /**
     * 上传头像到oss
     *
     * @param file
     * @return
     */
    String uploadFileAvatar(MultipartFile file);
}
```

```
package com.oldbai.halfmoon.oss;


import com.aliyun.oss.OSS;
import com.aliyun.oss.OSSClientBuilder;
import org.joda.time.DateTime;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.InputStream;
import java.util.UUID;

/**
 * @author 老白
 */
@Service
public class OssServiceImpl implements OssService {

    /**
     * 上传头像到oss
     *
     * @param file
     * @return
     */
```

```
    @Override
    public String uploadFileAvatar(MultipartFile file) {
        // 工具类获取值
        String endpoint = ConstantPropertiesUtils.END_POIND;
        String accessKeyId = ConstantPropertiesUtils.ACCESS_KEY_ID;
        String accessKeySecret = ConstantPropertiesUtils.ACCESS_KEY_SECRET;
        String bucketName = ConstantPropertiesUtils.BUCKET_NAME;

        try {
            // 创建OSS实例。
            OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId,
accessKeySecret);

            //获取上传文件输入流
            InputStream inputStream = file.getInputStream();
            //获取文件名称
            String fileName = file.getOriginalFilename();

            //1 在文件名称里面添加随机唯一的值
            String uuid = UUID.randomUUID().toString().replaceAll("-", "");
            // yuy76t5rew01.jpg
            fileName = uuid + fileName;

            //2 把文件按照日期进行分类
            //获取当前日期
            //   2019/11/12
            String datePath = new DateTime().toString("yyyy/MM/dd");
            //拼接
            //  2019/11/12/ewtqr313401.jpg
            fileName = datePath + "/" + fileName;

            //调用oss方法实现上传
            //第一个参数  Bucket名称
            //第二个参数  上传到oss文件路径和文件名称   aa/bb/1.jpg
            //第三个参数  上传文件输入流
            ossClient.putObject(bucketName, fileName, inputStream);

            // 关闭OSSClient。
            ossClient.shutdown();

            //把上传之后文件路径返回
            //需要把上传到阿里云oss路径手动拼接出来
            //  https://edu-guli-1010.oss-cn-beijing.aliyuncs.com/01.jpg
            String url = "https://" + bucketName + "." + endpoint + "/" +
fileName;
            return url;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

2.上传图片

1）.接口

```
/**
     * 增
     * 关于图片上传：
     * 一般来说，现在比较常用的是对象存储 --> 很简单，看文档就会了
     * 使用 Nginx + fastDFS ==>  fastDFS ==> 处理文件上传，Nginx --> 复制处理文件访问
     * 直接进行文件操作
     * <p>
     * 返回url给前端
     *
     * @return
     */
    @ApiOperation("上传图片")
    @PostMapping("/upload")
    public ResponseResult uploadImage(@RequestParam("file") MultipartFile file)
{

        return imageService.uploadImage(file);
    }
```

2）.方法

```
/**
     * 上传图片文件
     *
     * @param file
     * @return
     */
    @Override
    public ResponseResult uploadImage(MultipartFile file) {
        //判断是否有文件
        if (file == null) {
            return ResponseResult.FAILED("图片不可以为空.");
        }
        //判断文件类型，我们只支持图片上传，比如说：png，jpg，gif
        String contentType = file.getContentType();
        String fileName = file.getOriginalFilename();

        log.info("contentType == > " + contentType);
        boolean isType = false;
        if (StringUtils.isEmpty(contentType)) {
            return ResponseResult.FAILED("图片格式错误.");
        } else if (Constants.ImageType.TYPE_JGP_WITH_PREFIX.equals(contentType))
{
            isType = true;
        } else if (Constants.ImageType.TYPE_GIF_WITH_PREFIX.equals(contentType))
{
            isType = true;
        } else if (Constants.ImageType.TYPE_PNG_WITH_PREFIX.equals(contentType))
{
            isType = true;
        }else if (Constants.ImageType.TYPE_JPEG_WITH_PREFIX.equals(contentType))
{
            isType = true;
        }
```

```
        if (!isType) {
            return ResponseResult.FAILED("图片格式错误.");
        }

        //获取上传文件  MultipartFile
        //返回上传到oss的路径
        String url = ossService.uploadFileAvatar(file);
        Images image = new Images();
        User checkUser = userService.checkUser();
        image.setName(fileName);
        image.setContentType(contentType);
        image.setUserId(checkUser.getId());
        image.setUrl(url);
        image.setState("1");
        imagesMapper.insert(image);
        Map<String, String> map = new HashMap<>();
        map.put("url", url);
        //返回结果
        return ResponseResult.SUCCESS("上传成功", map);
    }
```

3.删除图片

1）.接口

```
/**
     * 删
     *
     * @param imageId
     * @return
     */
    @ApiOperation("删除图片")
    @PreAuthorize("@permission.adminPermission()")
    @GetMapping("/delete/{imageId}")
    public ResponseResult deleteImage(@PathVariable("imageId") String imageId) {
        return  imageService.deleteById(imageId);
    }
```

2）.方法

```
@Override
    public ResponseResult deleteById(String imageId) {
        int result = imagesMapper.deleteById(imageId);
        if (result > 0) {
            return ResponseResult.SUCCESS("删除成功.");
        }
        return ResponseResult.FAILED("图片不存在.");
    }
```

4.查图片

1）.接口

```
/**
     * 查
     * 获取单张图片，前端进行解析一下得到 URL 然后展示出来
     *
     * @param imageId
     * @return
     */
    @ApiOperation("查图片")
    @GetMapping("/get_image/{imageId}")
    public ResponseResult getImage(@PathVariable("imageId") String imageId) {
        return imageService.viewImage(imageId);
    }
```

2）.方法

```
@Override
    public ResponseResult viewImage(String imageId) {
        Images oneById = imagesMapper.selectById(imageId);
        if (StringUtils.isEmpty(oneById)) {
            return ResponseResult.FAILED("图片不存在......");
        }
        return ResponseResult.SUCCESS(oneById);
    }
```

5.获取图片列表

1）.接口

```
/**
     * 获取集合
     *
     * @param page
     * @param size
     * @return
     */
    @ApiOperation("获取图片列表")
    @GetMapping("/list")
    public ResponseResult listImage(@RequestParam("page") int page,
                                    @RequestParam("size") int size) {
        return imageService.listImages(page, size);
    }
```

2）.方法

```
/**
     * 获取图片列表
     *
     * @param page
     * @param size
     * @return
     */
    @Override
```

```
    public ResponseResult listImages(int page, int size) {
        User checkUser = userService.checkUser();
        page = Utils.getPage(page);
        size = Utils.getSize(size);
        //创建条件
        Page<Images> imagesPage = new Page<>(page, size);
        QueryWrapper<Images> wrapper = new QueryWrapper<>();
        wrapper.orderByDesc("create_time").eq("user_id",
checkUser.getId()).eq("state", "1");
        Page<Images> all = imagesMapper.selectPage(imagesPage, wrapper);
        return ResponseResult.SUCCESS("获取成功", all);
    }
```

十三、管理中心网站信息模块API

1.获取网站标题

1）.接口

```
/**
     * 获取网站标题
     * <p>
     * 网站的标题，就是tab标签栏上显示的标题
     * <p>
     * 这个标题一般只显示在首页，如果是文章页面，则会显示：文章的名称+网站名称
     *
     * @return
     */
    @ApiOperation("获取网站标题")
    @PreAuthorize("@permission.adminPermission()")
    @GetMapping("/title")
    public ResponseResult getWebSizeTitle() {
        return iWebSizeInfoService.getWebSizeTitle();
    }
```

2）.方法

```
@Transactional
    @Override
    public ResponseResult getWebSizeTitle() {
        QueryWrapper<Settings> queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("`key`", Constants.Settings.WEB_SIZE_TITLE);
        Settings title = settingsMapper.selectOne(queryWrapper);
        return ResponseResult.SUCCESS("获取网站title成功.", title);
    }
```

2.更新网站标题

1）.接口

```
/**
     * 更新网站标题
     *
     * @return
     */
    @ApiOperation("更新网站标题")
    @PreAuthorize("@permission.adminPermission()")
    @GetMapping("/updateTitle/{title}")
    public ResponseResult uploadWebSizeInfoTitle(@PathVariable("title") String
title) {
        return iWebSizeInfoService.putWebSizeTitle(title);
    }
```

2）.方法（只做了添加，没做更新）

```
@Transactional
    @Override
    public ResponseResult putWebSizeTitle(String title) {
        if (StringUtils.isEmpty(title)) {
            return ResponseResult.FAILED("网站标题不可以为空.");
        }
        QueryWrapper<Settings> queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("`key`", Constants.Settings.WEB_SIZE_TITLE);
        Settings titleFromDb = settingsMapper.selectOne(queryWrapper);
        if (titleFromDb == null) {
            titleFromDb = new Settings();
            titleFromDb.setKey(Constants.Settings.WEB_SIZE_TITLE);
        }
        titleFromDb.setValue(title);
        settingsMapper.insert(titleFromDb);
        return ResponseResult.SUCCESS("网站Title更新成功.");
    }
```

3.获取网站信息

1）.接口

```
/**
     * 获取网站信息
     * <p>
     * 网站SEO信息，其实包括：描述、关键字和标题
     * <p>
     * 标题我们单独获取了，这里的话，我们就获取关键字和描述信息，主要是首页。
     * <p>
     * 如果是文章页面的话，就使用文章的摘要作为描述，用标签作为关键字即可。
     *
     * @return
     */
    @ApiOperation("获取网站信息")
    @PreAuthorize("@permission.adminPermission()")
    @GetMapping("/seo")
    public ResponseResult getSeoInfo() {
        return iWebSizeInfoService.getSeoInfo();
    }
```

2）.方法

```
@Transactional
    @Override
    public ResponseResult getSeoInfo() {
        QueryWrapper<Settings> queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("`key`", Constants.Settings.WEB_SIZE_DESCRIPTION);
        Settings description = settingsMapper.selectOne(queryWrapper);
        queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("`key`", Constants.Settings.WEB_SIZE_KEYWORDS);
        Settings keyWords = settingsMapper.selectOne(queryWrapper);
        Map<String, String> result = new HashMap<>();
        if (description != null && keyWords != null) {
            result.put(description.getKey(), description.getValue());
            result.put(keyWords.getKey(), keyWords.getValue());
        }
        return ResponseResult.SUCCESS("获取SEO信息成功.", result);
    }
```

4.修改网站信息

1）.接口

```
/**
     * 修改网站信息
     * <p>
     * 更新网站的SEO信息，不建议经常更新哦，一般定了就别改了。
     *
     * @return
     */
    @ApiOperation("修改网站信息")
    @PreAuthorize("@permission.adminPermission()")
    @PostMapping("/update/seo")
    public ResponseResult UpdateSeoInfo(@RequestParam("keywords") String
keywords,
                                        @RequestParam("description") String
description) {
        return iWebSizeInfoService.putSeoInfo(keywords, description);
    }
```

2）.方法（只做了添加，没做更新）

```
@Transactional
    @Override
    public ResponseResult putSeoInfo(String keywords, String description) {
        //判断
        if (StringUtils.isEmpty(keywords)) {
            return ResponseResult.FAILED("关键字不可以为空.");
        }
        if (StringUtils.isEmpty(description)) {
            return ResponseResult.FAILED("网站描述不可以为空.");
        }
```

```
        QueryWrapper<Settings> queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("`key`", Constants.Settings.WEB_SIZE_DESCRIPTION);
        Settings descriptionFromDb = settingsMapper.selectOne(queryWrapper);
        if (descriptionFromDb == null) {
            descriptionFromDb = new Settings();
            descriptionFromDb.setKey(Constants.Settings.WEB_SIZE_DESCRIPTION);
        }
        descriptionFromDb.setValue(description);
        settingsMapper.insert(descriptionFromDb);
        queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("`key`", Constants.Settings.WEB_SIZE_KEYWORDS);
        Settings keyWordsFromDb = settingsMapper.selectOne(queryWrapper);
        if (keyWordsFromDb == null) {
            keyWordsFromDb = new Settings();
            keyWordsFromDb.setKey(Constants.Settings.WEB_SIZE_KEYWORDS);
        }
        keyWordsFromDb.setValue(keywords);
        settingsMapper.insert(keyWordsFromDb);
        return ResponseResult.SUCCESS("更新SEO信息成功.");
    }
```

5.获取浏览网站信息

1）.接口

```
/**
     * 获取浏览网站信息
     * <p>
     * 网站的浏览量，
     * <p>
     * 更新时机：用户页面加载完以后，向统计接口提交一下。
     * <p>
     * 获取时机：管理中心获取访问量/前端如果有需要的话也可以获取
     *
     * @return
     */
    @ApiOperation("获取浏览网站信息")
    @GetMapping("/view_count")
    public ResponseResult getWebSizeViewCount() {
        return iWebSizeInfoService.getSizeViewCount();
    }
```

2）.方法

```
 /**
     * 这个是全网站的访问量，要做得细一点，还得分来源
     * 这里只统计浏览量，只统计文章的浏览量，提供一个浏览量的统计接口（页面级的）
     *
     * @return 浏览量
     */
    @Transactional
    @Override
    public ResponseResult getSizeViewCount() {
        //先从redis里拿出来
```

```
        String viewCountStr = (String)
redisUtil.get(Constants.Settings.WEB_SIZE_VIEW_COUNT);
        QueryWrapper<Settings> queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("`key`", Constants.Settings.WEB_SIZE_VIEW_COUNT);
        Settings viewCount = settingsMapper.selectOne(queryWrapper);
        if (viewCount == null) {
            viewCount = this.initViewItem();
            settingsMapper.insert(viewCount);
        }
        if (StringUtils.isEmpty(viewCountStr)) {
            viewCountStr = viewCount.getValue();
            redisUtil.set(Constants.Settings.WEB_SIZE_VIEW_COUNT, viewCountStr);
        } else {
            //把redis里的更新到数据里
            viewCount.setValue(viewCountStr);
            settingsMapper.updateById(viewCount);
        }
        Map<String, Integer> result = new HashMap<>();
        result.put(viewCount.getKey(), Integer.valueOf(viewCount.getValue()));
        return ResponseResult.SUCCESS("获取网站浏览量成功.", result);
    }

    private Settings initViewItem() {
        Settings settings = new Settings();
        settings.setKey(Constants.Settings.WEB_SIZE_VIEW_COUNT);
        settings.setValue("1");
        return settings;
    }
```

十四、管理中心轮播图模块API

1.添加轮播图

1）.接口

```
    /**
     * 增
     * 对于增删改查的套路就是：
     * 权限有没有
     * 不能为空的数据进行检查
     * 数据格式进行检查
     * 补充数据
     * 保存数据
     * 返回结果
     * <p>
     * 添加轮播图前，要上传图片，然后返回访问的ID，然后拼接成url
     * <p>
     * 这样子创建轮播图的bean，就可以提交了。
     *
     * @return
     */
    @ApiOperation("添加轮播图")
    @PreAuthorize("@permission.adminPermission()")
    @PostMapping("/add_loop")
    public ResponseResult uploadLoop(@RequestBody ImgLooper looper) {
```

```
        return loopService.addLoop(looper);
    }
```

2）.方法

```
@Override
    public ResponseResult addLoop(ImgLooper looper) {
        //检查数据
        String title = looper.getTitle();
        if (StringUtils.isEmpty(title)) {
            return ResponseResult.FAILED("标题不可以为空.");
        }
        String imageUrl = looper.getImageUrl();
        if (StringUtils.isEmpty(imageUrl)) {
            return ResponseResult.FAILED("图片不可以为空.");
        }
        String targetUrl = looper.getTargetUrl();
        if (StringUtils.isEmpty(targetUrl)) {
            return ResponseResult.FAILED("跳转链接不可以为空.");
        }
        if (StringUtils.isEmpty(looper.getOrder())) {
            looper.setOrder(0);
        }
        if (StringUtils.isEmpty(looper.getState())) {
            looper.setState("1");
        }
        //补充数据
        //保存数据
        looperMapper.insert(looper);
        //返回结果
        return ResponseResult.SUCCESS("轮播图添加成功.");
    }
```

3）.测试数据

```
{
  "imageUrl": "20210217154726.jpg",
  "order": 0,
  "targetUrl": "www.baidu.com",
  "title": "轮播图1234567"
}
```

2.删除轮播图

1）.接口

```
    /**
     * 删
     * 话不多说，直接删除，前端可以弹窗警告
     *
     * @param loopId
```

```
     * @return
     */
    @ApiOperation("删除轮播图")
    @PreAuthorize("@permission.adminPermission()")
    @GetMapping("/delete/{loopId}")
    public ResponseResult deleteLoop(@PathVariable("loopId") String loopId) {
        return loopService.deleteLoop(loopId);
    }
```

2）.方法

```
@Override
    public ResponseResult deleteLoop(String loopId) {
        looperMapper.deleteById(loopId);
        return ResponseResult.SUCCESS("删除成功.");
    }
```

3.修改轮播图

1）.接口

```
  /**
     * 改
     *
     * @param loopId
     * @param looper
     * @return
     */
    @ApiOperation("修改轮播图")
    @PreAuthorize("@permission.adminPermission()")
    @PostMapping("/update/{loopId}")
    public ResponseResult updateLoop(@PathVariable("loopId") String loopId,
                                     @RequestBody ImgLooper looper) {
        return loopService.updateLoop(loopId, looper);
    }
```

2）.方法

```
@Override
    public ResponseResult updateLoop(String loopId, ImgLooper looper) {
        //找出来
        ImgLooper loopFromDb = looperMapper.selectById(loopId);
        if (loopFromDb == null) {
            return ResponseResult.FAILED("轮播图不存在.");
        }
        //不可以为空的，要判空
        String title = looper.getTitle();
        if (!StringUtils.isEmpty(title)) {
            loopFromDb.setTitle(title);
        }
        String targetUrl = looper.getTargetUrl();
        if (!StringUtils.isEmpty(targetUrl)) {
            loopFromDb.setTargetUrl(targetUrl);
```

```
        }
        String imageUrl = looper.getImageUrl();
        if (!StringUtils.isEmpty(imageUrl)) {
            loopFromDb.setImageUrl(imageUrl);
        }
        if (!StringUtils.isEmpty(looper.getState())) {
            loopFromDb.setState(looper.getState());
        }
        loopFromDb.setOrder(looper.getOrder());
        //可以为空的直接设置
        //保存回去
        looperMapper.updateById(loopFromDb);
        return ResponseResult.SUCCESS("轮播图更新成功.");
    }
```

3）.测试数据

```
{
      "title": "轮播图12345678",

      "targetUrl": "www.baidu.com",
      "imageUrl": "20210217154726.jpg"
    }
```

4.查询轮播图

1）.接口

```
 /**
     * 查
     * 获取轮播图，这个是获取单个
     * <p>
     * 其实，有没有都可以的
     * <p>
     * 使用场景就是获取完轮播图列表，如果你要更新轮播图，你可以调用此接口获取到它的内容。
     * <p>
     * 如果不获取也可以，直接从列表里获取，因为我们的列表内容是全的。
     *
     * @param loopId
     * @return
     */
    @ApiOperation("查询轮播图")
    @PreAuthorize("@permission.adminPermission()")
    @GetMapping("/get_loop/{loopId}")
    public ResponseResult getLoop(@PathVariable("loopId") String loopId) {
        return loopService.getLoop(loopId);
    }
```

2）.方法

```
@Override
    public ResponseResult getLoop(String loopId) {
        ImgLooper loop = looperMapper.selectById(loopId);
        if (loop == null) {
            return ResponseResult.FAILED("轮播图不存在.");
        }
        return ResponseResult.SUCCESS("轮播图获取成功.", loop);
    }
```

5.获取轮播图列表

1）.接口

```
 /**
     * 获取集合
     * 除了前端，后台也是要获取轮播图列表的，因为我们要管理它呀
     *
     * @return
     */
    @ApiOperation("获取轮播图列表")
    @GetMapping("/list")
    public ResponseResult listLoops() {
        return loopService.listLoops();
    }
```

2）.方法

```
@Override
    public ResponseResult listLoops() {
        User checkUser = userService.checkUser();
        List<ImgLooper> all = null;
        QueryWrapper<ImgLooper> wrapper = null;
        if (StringUtils.isEmpty(checkUser) ||
!Constants.User.ROLE_ADMIN.equals(checkUser.getRoles())) {
            //只能获取到正常的
            wrapper = new QueryWrapper<>();
            wrapper.eq("state", "1").orderByDesc("`order`");
            all = looperMapper.selectList(wrapper);
        } else {
            //查询
            wrapper = new QueryWrapper<>();
            wrapper.orderByDesc("update_time");
            all = looperMapper.selectList(wrapper);
        }
        return ResponseResult.SUCCESS("获取轮播图列表成功.", all);
    }
```

十五、管理中心文章模块API

1.发表文章

1）.接口

```
/**
     * 发表文章
     * <p>
     * 后期可以去做一些定时发布的功能
     * 如果是多人博客系统，得考虑审核的问题--->成功,通知，审核不通过，也可通知
     * <p>
     * 保存成草稿
     * 1、用户手动提交：会发生页面跳转-->提交完即可
     * 2、代码自动提交，每隔一段时间就会提交-->不会发生页面跳转-->多次提交-->如果没有唯一标识，会就重添加到数据库里
     * <p>
     * 不管是哪种草稿-->必须有标题
     * <p>
     * 方案一：每次用户发新文章之前-->先向后台请求一个唯一文章ID
     * 如果是更新文件，则不需要请求这个唯一的ID
     * <p>
     * 方案二：可以直接提交，后台判断有没有ID,如果没有ID，就新创建，并且ID作为此次返回的结果
     * 如果有ID，就修改已经存在的内容。
     * <p>
     * 推荐做法：
     * 自动保存草稿，在前端本地完成，也就是保存在本地。
     * 如果是用户手动提交的，就提交到后台
     *
     *
     * <p>
     * 防止重复提交（网络卡顿的时候，用户点了几次提交）：
     * 可以通过ID的方式
     * 通过token_key的提交频率来计算，如果30秒之内有多次提交，只有最前的一次有效
     * 其他的提交，直接return,提示用户不要太频繁操作.
     * <p>
     * 前端的处理：点击了提交以后，禁止按钮可以使用，等到有响应结果，再改变按钮的状态.
     *
     * @param article
     * @return
     */
    @PreAuthorize("@permission.adminPermission()")
    @ApiOperation("发表文章")
    @PostMapping("/add_article")
    public ResponseResult addArticle(@RequestBody Article article) {


        return articleService.postArticle(article);
    }
```

2）.方法

```
@Transactional
    @Override
    public ResponseResult postArticle(Article article) {
        //检查用户，获取到用户对象
        User sobUser = userService.checkUser();
        //未登录
        if (sobUser == null) {
            return ResponseResult.NO_LOGIN();
        }
        //检查数据
```

```
        //title、分类ID、内容、类型、摘要、标签
        String title = article.getTitle();
        if (StringUtils.isEmpty(title)) {
            return ResponseResult.FAILED("标题不可以为空.");
        }

        //2种，草稿和发布
        //获取文章状态
        String state = article.getState();
        //只接受两种状态 发布和草稿
        if (!Constants.Article.STATE_PUBLISH.equals(state) &&
                !Constants.Article.STATE_DRAFT.equals(state)) {
            //不支持此操作
            return ResponseResult.FAILED("不支持此操作");
        }
        //获取文章类型
        String type = article.getType();
        if (StringUtils.isEmpty(type)) {
            return ResponseResult.FAILED("类型不可以为空.");
        }
        //（0表示富文本，1表示markdown）
        if (!"0".equals(type) && !"1".equals(type)) {
            return ResponseResult.FAILED("类型格式不对.");
        }
        //以下检查是发布的检查，草稿不需要检查
        if (Constants.Article.STATE_PUBLISH.equals(state)) {
            //检查标题
            if (title.length() > Constants.Article.TITLE_MAX_LENGTH) {
                return ResponseResult.FAILED("文章标题不可以超过" +
Constants.Article.TITLE_MAX_LENGTH + "个字符");
            }
            //检查内容
            String content = article.getContent();
            if (StringUtils.isEmpty(content)) {
                return ResponseResult.FAILED("内容不可为空.");
            }
            //检查摘要
            String summary = article.getSummary();
            if (StringUtils.isEmpty(summary)) {
                return ResponseResult.FAILED("摘要不可以为空.");
            }
            if (summary.length() > Constants.Article.SUMMARY_MAX_LENGTH) {
                return ResponseResult.FAILED("摘要不可以超出" +
Constants.Article.SUMMARY_MAX_LENGTH + "个字符.");
            }
            //检查标签
            String labels = article.getLabels();
            //标签-标签1-标签2
            if (StringUtils.isEmpty(labels)) {
                return ResponseResult.FAILED("标签不可以为空.");
            }
        }
        //获取文章的ID
        String articleId = article.getId();
        if (StringUtils.isEmpty(articleId)) {
            //新内容,数据里没有的
            //补充数据：ID、创建时间、用户ID、更新时间
            article.setId(IdUtil.randomUUID() + "");
```

```
//            article.setCreateTime(new Date());
            article.setUserId(sobUser.getId());
            articleMapper.insert(article);
        } else {
            //更新内容，对状态进行处理，如果已经是发布的，则不能再保存为草稿
            Article articleFromDb = articleMapper.selectById(articleId);
            if (Constants.Article.STATE_PUBLISH.equals(articleFromDb.getState())
&&
                    Constants.Article.STATE_DRAFT.equals(state)) {
                //已经发布了，只能更新，不能保存草稿
                return ResponseResult.FAILED("已发布文章不支持成为草稿.");
            }
        }
        article.setUserId(sobUser.getId());
//        article.setUpdateTime(new Date());
        //保存到数据库里
        articleMapper.updateById(article);
        //TODO:保存到搜索的数据库里
        //打散标签，入库，统计
        this.setupLabels(article.getLabels());
        //返回结果,只有一种case使用到这个ID
        //如果要做程序自动保存成草稿（比如说每30秒保存一次，就需要加上这个ID了，否则会创建多个
Item）
        return
ResponseResult.SUCCESS(Constants.Article.STATE_DRAFT.equals(state) ? "草稿保存成
功" :
                "文章发表成功.", article.getId());
    }
```

3）.测试数据

```
      {
      "categoryId": "805487266016395264",
      "content": "测试内容1",
      "labels": "测试标签-文章",
      "state": "1",
      "summary": "测试啊",
      "title": "测试啊1",
      "type": "0"
      }
```

```
{
"id":"486f19c9-26b4-4575-b7ce-78b421fddd41",
      "categoryId": "b316fb0a5dd63ec177dc572f28f24b90",
      "content": "测试内容草稿变文章",
      "labels": "测试标签-文章-python-C++-spring",
      "state": "2",
      "summary": "测试啊1234567123",
      "title": "测试啊草稿变文章",
      "type": "0"
      }
```

2.删除文章

1）.接口

```
 /**
     * 如果是多用户，用户不可以删除，删除只是修改状态
     * 管理可以删除
     * <p>
     * 做成真的删除
     *
     * @param articleId
     * @return
     */
    @PreAuthorize("@permission.adminPermission()")
    @ApiOperation("删除文章")
    @GetMapping("/delete/{articleId}")
    public ResponseResult deleteArticle(@PathVariable("articleId") String
articleId) {
        return articleService.deleteArticleById(articleId);
    }
```

2）.方法

```
@Transactional
    @Override
    public ResponseResult deleteArticleById(String articleId) {
        Article article = articleMapper.selectById(articleId);
        this.deleteLabels(article.getLabels());
        QueryWrapper<Comment> queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("article_id", articleId);
        commentMapper.delete(queryWrapper);
        int result = articleMapper.deleteById(articleId);
        if (result > 0) {
            return ResponseResult.SUCCESS("文章删除成功.");
        }
        return ResponseResult.FAILED("文章不存在.");
    }

    @Transactional
    void deleteLabels(String labels) {
        List<String> labelList = new ArrayList<>();
        if (labels.contains("-")) {
            labelList.addAll(Arrays.asList(labels.split("-")));
        } else {
            labelList.add(labels);
        }
        //入库 并统计
        for (String label : labelList) {
            int result = labelsMapper.deleteCountByName(label);
        }
    }
```

3.更新文章内容

1）.接口

```
/**
     * 更新文章内容
     * <p>
     * 该接口只支持修改内容：标题、内容、标签、分类，摘要
     *
     * @param articleId 文章ID
     * @param article   文章
     * @return
     */
    @PreAuthorize("@permission.adminPermission()")
    @ApiOperation("更新文章")
    @PostMapping("/update/{articleId}")
    public ResponseResult updateArticle(@PathVariable("articleId") String
articleId,
                                        @RequestBody Article article) {
        return articleService.updateArticle(articleId, article);
    }
```

2）.方法

```
/**
     * 更新文章内容
     * <p>
     * 该接口只支持修改内容：标题、内容、标签、分类，摘要
     *
     * @param articleId 文章ID
     * @param article   文章
     * @return
     */
    @Transactional
    @Override
    public ResponseResult updateArticle(String articleId, Article article) {
        //先找出来
        Article articleFromDb = articleMapper.selectById(articleId);
        if (articleFromDb == null) {
            return ResponseResult.FAILED("文章不存在.");
        }
        //内容修改
        String title = article.getTitle();
        if (!StringUtils.isEmpty(title)) {
            articleFromDb.setTitle(title);
        }

        String summary = article.getSummary();
        if (!StringUtils.isEmpty(summary)) {
            articleFromDb.setSummary(summary);
        }

        String content = article.getContent();
        if (!StringUtils.isEmpty(content)) {
            articleFromDb.setContent(content);
        }

        String label = article.getLabels();
```

```
        if (!StringUtils.isEmpty(label)) {
            //TODO 字符串切割 。找到不一样的标签，并且添加进去，获取删除
            articleFromDb.setLabels(label);
        }

        String categoryId = article.getCategoryId();
        if (!StringUtils.isEmpty(categoryId)) {
            articleFromDb.setCategoryId(categoryId);
        }
        articleFromDb.setCover(article.getCover());
        articleMapper.updateById(articleFromDb);
        //返回结果
        return ResponseResult.SUCCESS("文章更新成功.");
    }
```

3）.测试数据

```
{
    "title": "测试啊1234567",
    "categoryId": "b316fb0a5dd63ec177dc572f28f24b90",
    "content": "测试内容我最牛",
    "summary": "测试啊1234567",
    "labels": "测试标签-文章-Java",
    "cover": null
  }
```

4.获取文章

1.接口

```
/**
     * 获取文章
     * <p>
     * 获取文章详情，我们要设置一下viewCount，另外则是做缓存处理。
     * <p>
     * 缓存我们一般会统一处理。因为要考虑添加缓存和删除缓存
     * <p>
     * 比如说，我们在访问文章的时候，添加缓存，先从缓存中获取，如果没有再去数据库中获取，获取到了再添加到缓存里。
     * <p>
     * 如果我们更新文章、置顶文章、删除文章、就需要去清除缓存。等待下次获取文章详情的时候，重新加入缓存里。
     * <p>
     * 还要注意的是权限，管理员可以拿到任何状态的文章，作者可以拿到除了删除状态的文章，其他人只能拿到发布，或者置顶的文章。
     *
     * @param articleId
     * @return
     */
    @PreAuthorize("@permission.adminPermission()")
    @ApiOperation("获取文章")
    @GetMapping("/get_article/{articleId}")
```

```
    public ResponseResult getArticle(@PathVariable("articleId") String
articleId) {
        return articleService.getArticleById(articleId);
    }
```

2）.方法

```
 /**
     * 如果有审核机制：审核中的文章-->只有管理员和作者自己可以获取
     * 有草稿、删除、置顶的、已经发布的
     * 删除的不能获取、其他都可以获取
     *
     * @param articleId
     * @return
     */
    @Transactional
    @Override
    public ResponseResult getArticleById(String articleId) {
        //查询出文章
        Article article = articleMapper.selectById(articleId);
        if (article == null) {
            return ResponseResult.FAILED("文章不存在.");
        }
        //判断文章状态
        String state = article.getState();
        if (Constants.Article.STATE_PUBLISH.equals(state) ||
                Constants.Article.STATE_TOP.equals(state)) {
            //可以返回
            return ResponseResult.SUCCESS("获取文章成功.", article);
        }
        //如果是删除/草稿，需要管理角色
        User sobUser = userService.checkUser();
        if (sobUser == null ||
!Constants.User.ROLE_ADMIN.equals(sobUser.getRoles())) {
            return ResponseResult.NO_PERMISSION();
        }
        //返回结果
        return ResponseResult.SUCCESS("获取文章成功.", article);
    }
```

5.获取文章集合

1）.接口

```
  /**
     * 获取文章集合
     * 这里的条件包括状态呀，标题搜索，分类这些。
     *
     * @param page
     * @param size
     * @return
     */
    @PreAuthorize("@permission.adminPermission()")
```

```
    @ApiOperation("获取文章集合")
    @GetMapping("/list/{page}/{size}")
    public ResponseResult listArticles(@PathVariable("page") int page,
                                       @PathVariable("size") int size,
                                       @RequestParam(value = "state", required =
false) String state,
                                       @RequestParam(value = "keyword", required
= false) String keyword,
                                       @RequestParam(value = "categoryId",
required = false) String categoryId) {
        return articleService.listArticles(page, size, keyword, categoryId,
state);
    }
```

2）.方法

```
/**
     * 这里管理中，获取文章列表
     *
     * @param page       页码
     * @param size       每一页数量
     * @param keyword    标题关键字（搜索关键字）
     * @param categoryId 分类ID
     * @param state      状态：已经删除、草稿、已经发布的、置顶的
     * @return
     */
    @Transactional
    @Override
    public ResponseResult listArticles(int page, int size, String keyword,
String categoryId, String state) {
        //TODO 可以不把文章内容返回回去
        //处理一下size 和page
        page = Utils.getPage(page);
        size = Utils.getSize(size);
        //创建分页和排序条件
        QueryWrapper<ArticleView> queryWrapper = new QueryWrapper<>();
        if (!StringUtils.isEmpty(state)) {
            queryWrapper.eq("state", state);
        }
        if (!StringUtils.isEmpty(categoryId)) {
            queryWrapper.eq("category_id", categoryId);
        }
        if (!StringUtils.isEmpty(keyword)) {
            queryWrapper.like("title", keyword);
        }
        queryWrapper.orderByDesc("create_time");
        Page<ArticleView> viewPage = new Page<>(page, size);
        //开始查询
        Page<ArticleView> all = articleViewMapper.selectPage(viewPage,
queryWrapper);
        //处理查询条件
        //返回结果
        return ResponseResult.SUCCESS("获取文章列表成功.", all);
    }
```

6.更新文章状态

1）.接口

```
/**
     * 单独更新
     * 更新状态
     *
     * @return
     */
    @ApiOperation("更新文章状态")
    @PreAuthorize("@permission.adminPermission()")
    @PostMapping("/state/{articleId}/{state}")
    public ResponseResult updateArticleState(@PathVariable("articleId") String
articleId) {
        return articleService.deleteArticleByState(articleId);
    }
```

2）.方法

```
@Transactional
    @Override
    public ResponseResult deleteArticleByState(String articleId) {
        Article article = articleMapper.selectById(articleId);
        article.setState("0");
        int result = articleMapper.updateById(article);
        if (result > 0) {
            return ResponseResult.SUCCESS("文章删除成功.");
        }
        return ResponseResult.FAILED("文章不存在.");
    }
```

7.文章置顶

1）.接口

```
 /**
     * 文章置顶
     * <p>
     * 文章置顶是独立开来的，因为我们希望不管用户翻到哪一页内容，我们都把这些文章置顶。
     * <p>
     * 多次调用：文章置顶会取消，取消了的则会置顶
     *
     * @return
     */
    @ApiOperation("文章置顶")
    @PreAuthorize("@permission.adminPermission()")
    @PostMapping("/top/{articleId}")
    public ResponseResult topArticleState(@PathVariable("articleId") String
articleId) {
        return articleService.topArticle(articleId);
    }
```

2）.方法

```
@Transactional
    @Override
    public ResponseResult topArticle(String articleId) {
        //必须已经发布的，才可以置顶
        Article article = articleMapper.selectById(articleId);
        if (article == null) {
            return ResponseResult.FAILED("文章不存在.");
        }
        String state = article.getState();
        if (Constants.Article.STATE_PUBLISH.equals(state)) {
            article.setState(Constants.Article.STATE_TOP);
            articleMapper.updateById(article);
            return ResponseResult.SUCCESS("文章置顶成功.");
        }
        if (Constants.Article.STATE_TOP.equals(state)) {
            article.setState(Constants.Article.STATE_PUBLISH);
            articleMapper.updateById(article);
            return ResponseResult.SUCCESS("已取消置顶.");
        }
        return ResponseResult.FAILED("不支持该操作.");
    }
```

十六、管理中心评论模块API

1.删除评论

1）.接口

```
/**
     * 删
     * <p>
     * 删除评论的话，直接物理删除即可
     * <p>
     * 门户可以删除，管理中心也可以删除。
     * <p>
     * 管理员可以删除任意的，但是，门户的用户，只能删除自己的。
     *
     * @param commentId
     * @return
     */
    @ApiOperation("删除评论")
    @PreAuthorize("@permission.adminPermission()")
    @GetMapping("/delete/{commentId}")
    public ResponseResult deleteComment(@PathVariable("commentId") String
commentId) {
        return commentService.deleteCommentById(commentId);
    }
```

2）.方法

```
@Override
    public ResponseResult deleteCommentById(String commentId) {
        //检查用户角色
        User sobUser = userService.checkUser();
        if (sobUser == null) {
            return ResponseResult.NO_LOGIN();
        }
        //把评论找出来，比对用户权限
        Comment comment = commentMapper.selectById(commentId);
        if (comment == null) {
            return ResponseResult.FAILED("评论不存在.");
        }
        if (sobUser.getId().equals(comment.getUserId()) ||
                //登录要判断角色
                Constants.User.ROLE_ADMIN.equals(sobUser.getRoles())) {
            commentMapper.deleteById(commentId);
            return ResponseResult.SUCCESS("评论删除成功.");
        } else {
            //没有权限
            return ResponseResult.NO_PERMISSION();
        }
    }
```

2.获取评论列表

1）.接口

```
/**
     * 获取集合
     * <p>
     * 这个获取只需要根据时间排序即可，如果同学们要做得全一点，可以跟前面的文章一样，条件查询即
可。
     * <p>
     * 根据条件获取评论列表，比如说按时间，比如说按文章，比如说按状态.
     *
     * @param page
     * @param size
     * @return
     */
    @ApiOperation("获取评论列表")
    @PreAuthorize("@permission.adminPermission()")
    @GetMapping("/list")
    public ResponseResult listComments(@RequestParam("page") int page,
                                       @RequestParam("size") int size) {
        return commentService.listComments(page, size);
    }
```

2）.方法

```
@Override
    public ResponseResult listComments(int page, int size) {
        page = Utils.getPage(page);
        size = Utils.getSize(size);
        Page<Comment> commentPage = new Page<>(page, size);
        QueryWrapper<Comment> queryWrapper = new QueryWrapper<>();
        queryWrapper.orderByDesc("create_time");
        Page<Comment> all = commentMapper.selectPage(commentPage, queryWrapper);
        return ResponseResult.SUCCESS("获取评论列表成功.", all);
    }
```

3.评论置顶

1）.接口

```
 /**
     * 评论置顶
     * <p>
     * 这里是后台，没发表评论，可以操作评论。
     * <p>
     * 这个是置顶，评论内容由门户那这发表。
     * <p>
     * 评论置顶的话，查询的时候，我们就需要以这个作为排序了
     *
     * @return
     */
    @ApiOperation("评论置顶")
    @PreAuthorize("@permission.adminPermission()")
    @GetMapping("/top/{commentId}")
    public ResponseResult topComments(@PathVariable("commentId") String
commentId) {
        return commentService.topComment(commentId);
    }
```

2）.方法

```
 @Override
    public ResponseResult topComment(String commentId) {
        Comment comment = commentMapper.selectById(commentId);
        if (comment == null) {
            return ResponseResult.FAILED("评论不存在.");
        }
        String state = comment.getState();
        if (Constants.Comment.STATE_PUBLISH.equals(state)) {
            comment.setState(Constants.Comment.STATE_TOP);
            return ResponseResult.SUCCESS("置顶成功.");
        } else if (Constants.Comment.STATE_TOP.equals(state)) {
            comment.setState(Constants.Comment.STATE_PUBLISH);
            return ResponseResult.SUCCESS("取消置顶.");
        } else {
            return ResponseResult.FAILED("评论状态非法.");
        }
    }
```

十七、门户-网站

1.获取分类

1）.接口

```
/**
     * 获取分类
     *
     * @return
     */
    @ApiOperation("获取分类")
    @GetMapping("/categories")
    public ResponseResult getCategories() {
        return categoryService.listCategories();
    }
```

2）.方法

```
@Override
    public ResponseResult listCategories() {
        //创建条件

        //判断用户角色，普通用户 未登录用户 只能获取到正常的category
        //管理员账号 可以拿到所有的分类
        User checkUser = userService.checkUser();
        List<Category> all = null;
        QueryWrapper<Category> queryWrapper = null;
        if (StringUtils.isEmpty(checkUser) ||
!Constants.User.ROLE_ADMIN.equals(checkUser.getRoles())) {
            //只能获取到正常的category
            queryWrapper = new QueryWrapper<>();
            queryWrapper.eq("status", 1).orderByDesc("update_time");
            all = categoryMapper.selectList(queryWrapper);
        } else {
            //查询
            queryWrapper = new QueryWrapper<>();
            queryWrapper.orderByDesc("update_time");
            all = categoryMapper.selectList(queryWrapper);
        }
        //返回结果
        return ResponseResult.SUCCESS("获取分类列表成功.", all);
    }
```

2.获取网站标题

1）.接口

```
/**
     * 获取网站标题
     *
     * @return
     */
    @ApiOperation("获取网站标题")
    @GetMapping("/title")
    public ResponseResult getTitle() {
        return sizeInfoService.getWebSizeTitle();
    }
```

2）.方法

```
 @Transactional
    @Override
    public ResponseResult getWebSizeTitle() {
        QueryWrapper<Settings> queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("`key`", Constants.Settings.WEB_SIZE_TITLE);
        Settings title = settingsMapper.selectOne(queryWrapper);
        return ResponseResult.SUCCESS("获取网站title成功.", title);
    }
```

3.更新访问量

1）.接口

```
/**
     * 统计访问页，每个页面都统一次，PV，page view.
     * 直接增加一个访问量，可以刷量
     * 根据ip进行一些过滤，可以集成第三方的一个统计工具
     * //
     * 递增的统计
     * 统计信息，通过redis来统计，数据也会保存在mysql里
     * 不会每次都更新到Mysql里，当用户去获取访问量的时候，会更新一次
     * 平时的调用，只增加redis里的访问量
     * <p>
     * redis时机：每个页面访问的时候，如果不在从mysql中读取数据，写到redis里
     * 如果，就自增
     * <p>
     * mysql的时机，用户读取网站总访问量的时候，我们就读取一redis的，并且更新到mysql中
     * 如果redis里没有，那就读取mysql写到reds里的
     * <p>
     * TODO  更新访问量没完成
     */
    @ApiOperation("更新访问量")
    @GetMapping("/view_count")
    public void updateViewCount() {
        sizeInfoService.updateViewCount();
    }
```

2）.方法

```
/**
     * 1、并发量
     * 2、过滤相通的IP/ID
     * 3、防止攻击，比如太频繁的访问，就提示请稍后重试.
     */
    @Override
    public void updateViewCount() {
        //redis的更新时机：
        String viewCount = (String)
redisUtil.get(Constants.Settings.WEB_SIZE_VIEW_COUNT);
        if (StringUtils.isEmpty(viewCount)) {
            QueryWrapper<Settings> queryWrapper = new QueryWrapper<>();
            queryWrapper.eq("`key`", Constants.Settings.WEB_SIZE_VIEW_COUNT);
            Settings setting = settingsMapper.selectOne(queryWrapper);
            if (setting == null) {
                setting = this.initViewItem();
                settingsMapper.insert(setting);
            }
            redisUtil.set(Constants.Settings.WEB_SIZE_VIEW_COUNT,
setting.getValue());
        } else {
            //自增
            Integer integer = Integer.valueOf(viewCount);
            integer = integer + 1;
            redisUtil.set(Constants.Settings.WEB_SIZE_VIEW_COUNT,
String.valueOf(integer));
        }
    }
```

4.获取访问量

1）.接口

```
@ApiOperation("获取访问量")
    @GetMapping("/get/view_count")
    public ResponseResult getViewCount() {
        return sizeInfoService.getSizeViewCount();
    }
```

2）.方法

```
/**
     * 这个是全网站的访问量，要做得细一点，还得分来源
     * 这里只统计浏览量，只统计文章的浏览量，提供一个浏览量的统计接口（页面级的）
     *
     * @return 浏览量
     */
    @Transactional
    @Override
    public ResponseResult getSizeViewCount() {
        //先从redis里拿出来
        String viewCountStr = (String)
redisUtil.get(Constants.Settings.WEB_SIZE_VIEW_COUNT);
        QueryWrapper<Settings> queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("`key`", Constants.Settings.WEB_SIZE_VIEW_COUNT);
```

```
        Settings viewCount = settingsMapper.selectOne(queryWrapper);
        if (viewCount == null) {
            viewCount = this.initViewItem();
            settingsMapper.insert(viewCount);
        }
        if (StringUtils.isEmpty(viewCountStr)) {
            viewCountStr = viewCount.getValue();
            redisUtil.set(Constants.Settings.WEB_SIZE_VIEW_COUNT, viewCountStr);
        } else {
            //把redis里的更新到数据里
            viewCount.setValue(viewCountStr);
            settingsMapper.updateById(viewCount);
        }
        Map<String, Integer> result = new HashMap<>();
        result.put(viewCount.getKey(), Integer.valueOf(viewCount.getValue()));
        return ResponseResult.SUCCESS("获取网站浏览量成功.", result);
    }
```

5.获取网站SEO信息

1）.接口

```
/**
     * 获取网站SEO信息
     *
     * @return
     */
    @ApiOperation("获取网站SEO信息")
    @GetMapping("/seo")
    public ResponseResult getSeo() {
        return sizeInfoService.getSeoInfo();
    }
```

2）.方法

```
 @Transactional
    @Override
    public ResponseResult getSeoInfo() {
        QueryWrapper<Settings> queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("`key`", Constants.Settings.WEB_SIZE_DESCRIPTION);
        Settings description = settingsMapper.selectOne(queryWrapper);
        queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("`key`", Constants.Settings.WEB_SIZE_KEYWORDS);
        Settings keyWords = settingsMapper.selectOne(queryWrapper);
        Map<String, String> result = new HashMap<>();
        if (description != null && keyWords != null) {
            result.put(description.getKey(), description.getValue());
            result.put(keyWords.getKey(), keyWords.getValue());
        }
        return ResponseResult.SUCCESS("获取SEO信息成功.", result);
    }
```

6.获取轮播图信息

1）.接口

```
/**
     * 获取轮播图信息
     *
     * @return
     */
    @ApiOperation("获取轮播图信息")
    @GetMapping("/loop")
    public ResponseResult getLoops() {
        return loopService.listLoops();
    }
```

2）.方法

```
 @Override
    public ResponseResult listLoops() {
        User checkUser = userService.checkUser();
        List<ImgLooper> all = null;
        QueryWrapper<ImgLooper> wrapper = null;
        if (StringUtils.isEmpty(checkUser) ||
!Constants.User.ROLE_ADMIN.equals(checkUser.getRoles())) {
            //只能获取到正常的
            wrapper = new QueryWrapper<>();
            wrapper.eq("state", "1").orderByDesc("`order`");
            all = looperMapper.selectList(wrapper);
        } else {
            //查询
            wrapper = new QueryWrapper<>();
            wrapper.orderByDesc("update_time");
            all = looperMapper.selectList(wrapper);
        }
        return ResponseResult.SUCCESS("获取轮播图列表成功.", all);
    }
```

7.获取友情链接信息

1）.接口

```
/**
     * 获取友情链接信息
     *
     * @return
     */
    @ApiOperation("获取友情链接信息")
    @GetMapping("/friend_link")
    public ResponseResult getFriendLinks() {
        return friendLinkService.listFriendLinks();
    }
```

2）.方法

```
    @Override
    public ResponseResult getFriendLink(String friendLinkId) {
        FriendLink friendLink = friendLinkMapper.selectById(friendLinkId);
        if (friendLink == null) {
            return ResponseResult.FAILED("友情链接不存");
        }
        return ResponseResult.SUCCESS("获取成功", friendLink);
    }
```

十八、门户-文章

1.获取文章详情

1）.接口

```
 /**
     * 获取文章详情
     * 权限：任意用户
     * <p>
     * 内容过滤：只允许拿置顶的，或者已经发布的
     * 其他的获取：比如说草稿、只能对应用户获取。已经删除的，只有管理员才可以获取.
     *
     * @param articleId
     * @return
     */
    @ApiOperation("获取文章详情")
    @GetMapping("/get_article/{articleId}")
    public ResponseResult getArticle(@PathVariable("articleId") String
articleId) {
        return articleService.getArticleById(articleId);
    }
```

2）.方法

```
 /**
     * 如果有审核机制：审核中的文章-->只有管理员和作者自己可以获取
     * 有草稿、删除、置顶的、已经发布的
     * 删除的不能获取、其他都可以获取
     *
     * @param articleId
     * @return
     */
    @Transactional
    @Override
    public ResponseResult getArticleById(String articleId) {
        //查询出文章
        Article article = articleMapper.selectById(articleId);
        if (article == null) {
            return ResponseResult.FAILED("文章不存在.");
        }
        //判断文章状态
        String state = article.getState();
        if (Constants.Article.STATE_PUBLISH.equals(state) ||
```

```
                Constants.Article.STATE_TOP.equals(state)) {
            //可以返回
            return ResponseResult.SUCCESS("获取文章成功.", article);
        }
        //如果是删除/草稿，需要管理角色
        User sobUser = userService.checkUser();
        if (sobUser == null ||
!Constants.User.ROLE_ADMIN.equals(sobUser.getRoles())) {
            return ResponseResult.NO_PERMISSION();
        }
        //返回结果
        return ResponseResult.SUCCESS("获取文章成功.", article);
    }
```

2.获取文章列表

1）.接口

```
/**
     * 获取文章列表
     * 权限,所有用户
     * 状态:必须已经发布的,置顶的由另外一个接口获取,其他的不可以从此接口获取
     *
     * @param page
     * @param size
     * @return
     */
    @ApiOperation("获取文章列表")
    @GetMapping("/list/{page}/{size}")
    public ResponseResult listArticles(@PathVariable("page") int page,
                                       @PathVariable("size") int size) {
        return articleService.listArticles(page, size, null, null,
Constants.Article.STATE_PUBLISH);
    }
```

2）.方法

```
/**
     * 这里管理中，获取文章列表
     *
     * @param page       页码
     * @param size       每一页数量
     * @param keyword    标题关键字（搜索关键字）
     * @param categoryId 分类ID
     * @param state      状态：已经删除、草稿、已经发布的、置顶的
     * @return
     */
    @Transactional
    @Override
    public ResponseResult listArticles(int page, int size, String keyword,
String categoryId, String state) {
        //TODO 可以不把文章内容返回回去
        //处理一下size 和page
        page = Utils.getPage(page);
        size = Utils.getSize(size);
```

```
        //创建分页和排序条件
        QueryWrapper<ArticleView> queryWrapper = new QueryWrapper<>();
        if (!StringUtils.isEmpty(state)) {
            queryWrapper.eq("state", state);
        }
        if (!StringUtils.isEmpty(categoryId)) {
            queryWrapper.eq("category_id", categoryId);
        }
        if (!StringUtils.isEmpty(keyword)) {
            queryWrapper.like("title", keyword);
        }
        queryWrapper.orderByDesc("create_time");
        Page<ArticleView> viewPage = new Page<>(page, size);
        //开始查询
        Page<ArticleView> all = articleViewMapper.selectPage(viewPage,
queryWrapper);
        //处理查询条件
        //返回结果
        return ResponseResult.SUCCESS("获取文章列表成功.", all);
    }
```

3.获取文章列表置顶

1）.接口

```
/**
     * 获取文章列表置顶
     *
     * @return
     */
    @ApiOperation("获取文章列表置顶")
    @GetMapping("/top")
    public ResponseResult getTopArticle() {
        return articleService.listTopArticles();
    }
```

2）.方法

```
   @Transactional
    @Override
    public ResponseResult listTopArticles() {
        QueryWrapper<ArticleView> articleViewQueryWrapper = new QueryWrapper<>
();
        articleViewQueryWrapper.eq("state", Constants.Article.STATE_TOP);
        List<ArticleView> result =
articleViewMapper.selectList(articleViewQueryWrapper);
        return ResponseResult.SUCCESS("获取置顶文章列表成功.", result);
    }
```

4.获取标签云，用户点击标签，就会通过标签获取相关的文章列表

1）.接口

```
/**
     * 获取标签云，用户点击标签，就会通过标签获取相关的文章列表
     * 任意用户
     *
     * @param size
     * @return
     */
    @ApiOperation("获取标签云，用户点击标签，就会通过标签获取相关的文章列表")
    @GetMapping("/label/{size}")
    public ResponseResult getLabels(@PathVariable("size") int size) {
        return articleService.listLabels(size);
    }
```

2）.方法

```
@Transactional
    @Override
    public ResponseResult listLabels(int size) {
        size = Utils.getSize(size);
        QueryWrapper<Labels> wrapper = new QueryWrapper<>();
        wrapper.orderByDesc("count");
        Page<Labels> labelsPage = new Page<>(1, size);
        Page<Labels> all = labelsMapper.selectPage(labelsPage, wrapper);
        return ResponseResult.SUCCESS("获取标签列表成功.", all);
    }
```

5.获取推荐文章

1）.接口

```
/**
     * 获取推荐文章
     * 通过标签来计算这个匹配度
     * 标签：有一个，或者多个（5个以内，包含5个）
     * 从里面随机拿一个标签出来--->每一次获取的推荐文章，不那么雷同，种一样就雷同了
     * 通过标签去查询类似的文章，所包含此标签的文章
     * 如果没有相关文章，则从数据中获取最新的文章的
     *
     * @param articleId
     * @return
     */
    @ApiOperation("获取推荐文章")
    @GetMapping("/recommend/{articleId}/{size}")
    public ResponseResult getRecommendArticles(@PathVariable("articleId") String
articleId, @PathVariable("size") int size) {
        return articleService.listRecommendArticle(articleId, size);
    }
```

2）.方法

```java
/**
     * 获取推荐文章，通过标签来计算
     *
     * @param articleId
     * @param size
     * @return
     */

    @Transactional
    @Override
    public ResponseResult listRecommendArticle(String articleId, int size) {
        Random random = new Random();

        //查询文章，不需要文章，只需要标签
        QueryWrapper<ArticleView> articleViewQueryWrapper = new QueryWrapper<>
();
        articleViewQueryWrapper.select("labels").eq("id", articleId);
        String labels =
articleViewMapper.selectOne(articleViewQueryWrapper).getLabels();
        //打散标签
        List<String> labelList = new ArrayList<>();
        if (!labels.contains("-")) {
            labelList.add(labels);
        } else {
            labelList.addAll(Arrays.asList(labels.split("-")));
        }
        //从列表中随即获取一标签，查询与此标签相似的文章
        String targetLabel = labelList.get(random.nextInt(labelList.size()));
        log.info("targetLabel == > " + targetLabel);
        articleViewQueryWrapper = new QueryWrapper<>();
        articleViewQueryWrapper
                .like("labels", targetLabel)
                .ne("id", articleId)
                .and(i -> i.eq("state", "1").or().eq("state", "3"))
                .orderByDesc("create_time");
        Page<ArticleView> viewPage = new Page<>(1, size);
        Page<ArticleView> likeResultList =
articleViewMapper.selectPage(viewPage, articleViewQueryWrapper);
        //判断它的长度
        if (likeResultList.getTotal() < size) {
            //说明不够数量，获取最新的文章作为补充
            int dxSize = (int) (size - likeResultList.getTotal());
            articleViewQueryWrapper = new QueryWrapper<>();
            articleViewQueryWrapper
                    .ne("id", articleId)
                    .and(i -> i.eq("state", "1").or().eq("state", "3"))
                    .orderByDesc("create_time");
            Page<ArticleView> viewPage1 = new Page<>(1, dxSize);
            Page<ArticleView> dxList = articleViewMapper.selectPage(viewPage1,
articleViewQueryWrapper);
            //这个写法有一定的弊端，会把可能前面找到的也加进来，概率比较小，如果文章比较多
            List<ArticleView> resultList = new ArrayList<>
(likeResultList.getRecords());
            resultList.addAll(new ArrayList<>(dxList.getRecords()));
            likeResultList.setRecords(resultList);
        }
        return ResponseResult.SUCCESS("获取推荐文章成功.", likeResultList);
    }
```

6.根据标签获取文章列表

1）.接口

```
/**
     * 根据标签获取文章列表
     *
     * @param label
     * @param page
     * @param size
     * @return
     */
    @ApiOperation("根据标签获取文章列表")
    @GetMapping("/list/label/{label}/{page}/{size}")
    public ResponseResult listArticleByLabel(@PathVariable("label") String
label,
                                             @PathVariable("page") int page,
@PathVariable("size") int size) {
        return articleService.listArticlesByLabel(page, size, label);
    }
```

2）.方法

```
@Transactional
    @Override
    public ResponseResult listArticlesByLabel(int page, int size, String label)
{
        page = Utils.getPage(page);
        size = Utils.getSize(size);
        Page<ArticleView> articleViewPage = new Page<>(page, size);
        QueryWrapper<ArticleView> wrapper = new QueryWrapper<>();
        wrapper.like("labels", label)
                .and(i -> i.eq("state", "1").or().eq("state", "3"))
                .orderByDesc("create_time");
        Page<ArticleView> all = articleViewMapper.selectPage(articleViewPage,
wrapper);
        return ResponseResult.SUCCESS("获取文章列表成功.", all);
    }
```

7.获取分类内容

1）.接口

```
/**
     * 获取分类内容
     *
     * @return
     */
    @ApiOperation("获取分类内容")
    @GetMapping("/categories")
    public ResponseResult getCategories() {
        return categoryService.listCategories();
    }
```

2）.方法

```

```

8.根据分类获取文章列表

1）.接口

```
/**
     * 根据分类获取文章列表
     *
     * @param categoryId
     * @param page
     * @param size
     * @return
     */
    @ApiOperation("根据分类获取文章列表")
    @GetMapping("/list/{categoryId}/{page}/{size}")
    public ResponseResult listArticleByCategoryId(@PathVariable("categoryId")
String categoryId,
                                                  @PathVariable("page") int
page,
                                                  @PathVariable("size") int
size) {
        return articleService.listArticles(page, size, null, categoryId,
Constants.Article.STATE_PUBLISH);
    }
```

2）.方法

```

```

9.获取图片

1）.接口

```
/**
     * 获取图片
     *
     * @param response
     * @param imageId
     */
    @ApiOperation("获取图片")
    @GetMapping("/{imageId}")
    public ResponseResult getImage(HttpServletResponse response,
@PathVariable("imageId") String imageId) {
        return imagesService.viewImage(imageId);
    }
```

2）.方法

```
```

十九、门户-评论

1.添加评论

1）.接口

```
 /**
     * 增
     *
     * @return
     */
    @ApiOperation("添加评论")
    @PostMapping("/add_comment")
    public ResponseResult addComment(@RequestBody Comment comment) {
        return commentService.postComment(comment);
    }
```

2）.方法

```
/**
     * 发表评论
     *
     * @param comment 评论
     * @return
     */
    @Override
    public ResponseResult postComment(Comment comment) {
        //检查用户是否有登录
        User sobUser = userService.checkUser();
        if (sobUser == null) {
            return ResponseResult.NO_LOGIN();
        }
        //检查内容
        String articleId = comment.getArticleId();
        if (StringUtils.isEmpty(articleId)) {
```

```
            return ResponseResult.FAILED("文章ID不可以为空.");
        }
        ArticleView article = articleViewMapper.selectById(articleId);
        if (article == null) {
            return ResponseResult.FAILED("文章不存在.");
        }
        String content = comment.getContent();
        if (StringUtils.isEmpty(content)) {
            return ResponseResult.FAILED("评论内容不可以为空.");
        }
        //补全内容
        comment.setUserAvatar(sobUser.getAvatar());
        comment.setUserName(sobUser.getUserName());
        comment.setUserId(sobUser.getId());
        //保存入库
        commentMapper.insert(comment);
        //返回结果
        return ResponseResult.SUCCESS("评论成功");
    }
```

3）.测试数据

```
{
  "articleId": "486f19c9-26b4-4575-b7ce-78b421fddd41",
  "content": "测试评论1"
}
```

2.删除评论

1）.接口

```
/**
     * 删
     *
     * @param commentId
     * @return
     */
    @ApiOperation("删除评论")
    @GetMapping("/delete/{commentId}")
    public ResponseResult deleteComment(@PathVariable("commentId") String
commentId) {
        return commentService.deleteCommentById(commentId);
    }
```

2）.方法

```

```

3.获取评论集合

1）.接口

```
 /**
     * 获取评论集合
     *
     * @param articleId
     * @return
     */
    @ApiOperation("获取评论集合")
    @GetMapping("/list/{articleId}/{page}/{size}")
    public ResponseResult listComments(@PathVariable("articleId") String
articleId, @PathVariable("page") int page,@PathVariable("size") int size) {
        return commentService.listCommentByArticleId(articleId, page, size);
    }
```

2）.方法

```
 /**
     * 获取文章的评论
     * 评论的排序策略：
     * 最基本的就按时间排序-->升序和降序-->先发表的在前面或者后发表的在前面
     * <p>
     * 置顶的：一定在前最前面
     * <p>
     * 后发表的：前单位时间内会排在前面，过了此单位时间，会按点赞量和发表时间进行排序
     *
     * @param articleId
     * @param page
     * @param size
     * @return
     */
    @Override
    public ResponseResult listCommentByArticleId(String articleId, int page, int
size) {
        page = Utils.getPage(page);
        size = Utils.getSize(size);
        Page<Comment> commentPage = new Page<>(page, size);
        QueryWrapper<Comment> queryWrapper = new QueryWrapper<>();
        queryWrapper.orderByDesc("state", "create_time");
        Page<Comment> all = commentMapper.selectPage(commentPage, queryWrapper);

        return ResponseResult.SUCCESS("评论列表获取成功.", all);
    }
```

二十、使用dokcer安装solr安装

- 将资源复制进入 /root/docker 下
- 资源下载
- > 链接：https://pan.baidu.com/s/1c8BYsfoCg6_HKbFcgyk1ww
  > 提取码：tl3e
  > 复制这段内容后打开百度网盘手机App，操作更方便哦
- 使用以下命令安装

- `sudo docker-compose up -d`
- 然后就可以使用了。
- 具体配置参考
  -
- 时机

> 添加搜索内容（发表文章）
>
> 删除搜索内容（删除文章）
>
> 修改搜索内容（访问量改变）
>
> 查询->搜索（分页、过滤、条件）

1.Solr使用总结

>

二十一、使用Solr

1.导入依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-solr</artifactId>
</dependency>
```

2.进行配置

```
spring:
  data:
    solr:
      host: http://47.106.234.164:8983/solr/sob_blog_core
```

3.测试添加、更新、删除

```
package com.oldbai.halfmoon.solr;

import io.swagger.annotations.Api;
import org.apache.solr.client.solrj.SolrClient;
import org.apache.solr.client.solrj.SolrServerException;
import org.apache.solr.common.SolrInputDocument;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.io.IOException;
import java.util.Date;

@Service
public class SolrTestService {
```

```
    @Autowired
    private SolrClient solrClient;

    public void solrAdd() {
        SolrInputDocument doc = new SolrInputDocument();
        doc.addField("id", "005f490b2ff5da2483bd5398ae140b1d");
        doc.addField("blog_view_count", 10);
        doc.addField("blog_title", "测试啊12345");
        doc.addField("blog_content", "测试内容12345");
        doc.addField("blog_create_timme", new Date());
        doc.addField("blog_labels", "测试标签-文章");
        doc.addField("blog_url", "www.baidu.com");
        doc.addField("blog_category_id", "b316fb0a5dd63ec177dc572f28f24b90");
        try {
            solrClient.add(doc);
            solrClient.commit();
        } catch (SolrServerException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void solrUpdate() {
        //只要ID一样就是修改
        SolrInputDocument doc = new SolrInputDocument();
        doc.addField("id", "005f490b2ff5da2483bd5398ae140b1d");
        doc.addField("blog_view_count", 10);
        doc.addField("blog_title", "修改了测试啊12345");
        doc.addField("blog_content", "修改了测试内容12345");
        doc.addField("blog_create_timme", new Date());
        doc.addField("blog_labels", "测试标签-文章");
        doc.addField("blog_url", "www.baidu.com");
        doc.addField("blog_category_id", "b316fb0a5dd63ec177dc572f28f24b90");
        try {
            solrClient.add(doc);
            solrClient.commit();
        } catch (SolrServerException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }


    }

    public void solrDelete() {
        //单独删除一条
        try {
            solrClient.deleteById("005f490b2ff5da2483bd5398ae140b1d");
            solrClient.commit();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

4.向solr添加数据库中所有文章，删除所有

```
public void solrDeleteAll() {
        try {
            //删除所有
            solrClient.deleteByQuery("*");
            solrClient.commit();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Autowired
    private ArticleMapper articleMapper;

    /**
     * 向solr添加数据库中所有文章
     */
    public void importAll() {
        List<Article> articleList = articleMapper.selectList(null);
        SolrInputDocument doc = null;
        for (Article article : articleList) {
            doc = new SolrInputDocument();
            doc.addField("id", article.getId());
            doc.addField("blog_view_count", article.getViewCount());
            doc.addField("blog_title", article.getTitle());
            //对内容进行处理，去掉标签，提取纯文本内容
            //第一种是由markdown写的内容 ==> type = 1
            //第二种是有富文本内容 ==> type = 0
            //如果type等于 1 ，需要转换成 html
            //再由 html 转换成 纯文本
            //如果type等于 0 ，提取出纯文本
            String type = article.getType();
            String html = null;
            if (Constants.Article.TYPE_MARKDOWN.equals(type)) {
                //转成HTML
                // markdown to html
                MutableDataSet options = new
MutableDataSet().set(Parser.EXTENSIONS, Arrays.asList(
                        TablesExtension.create(),
                        JekyllTagExtension.create(),
                        TocExtension.create(),
                        SimTocExtension.create()
                ));
                Parser parser = Parser.builder(options).build();
                HtmlRenderer renderer = HtmlRenderer.builder(options).build();
                Node document = parser.parse(article.getContent());
                html = renderer.render(document);
            } else if (Constants.Article.TYPE_RICH_TEXT.equals(type)) {
                //富文本
                html = article.getContent();
            }
            //到了这里不管是什么都变成了 html
            //HTML转Text
```

```
            String content = Jsoup.parse(html).text();
            doc.addField("blog_content", content);
            doc.addField("blog_create_timme", article.getCreateTime());
            doc.addField("blog_labels", article.getLabels());
            doc.addField("blog_url", "null");
            doc.addField("blog_category_id", article.getCategoryId());
            try {
                solrClient.add(doc);
                solrClient.commit();
            } catch (SolrServerException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

5.去除HTML标签

1）.markdown转成html

```
<!--markdown to html-->
<dependency>
    <groupId>com.vladsch.flexmark</groupId>
    <artifactId>flexmark-all</artifactId>
    <version>0.50.44</version>
</dependency>
```

2）.代码

```
//转成HTML
                // markdown to html
                MutableDataSet options = new
MutableDataSet().set(Parser.EXTENSIONS, Arrays.asList(
                        TablesExtension.create(),
                        JekyllTagExtension.create(),
                        TocExtension.create(),
                        SimTocExtension.create()
                ));
                Parser parser = Parser.builder(options).build();
                HtmlRenderer renderer = HtmlRenderer.builder(options).build();
                Node document = parser.parse(article.getContent());
                html = renderer.render(document);
```

3）.html转text

```
<dependency>
    <groupId>org.jsoup</groupId>
    <artifactId>jsoup</artifactId>
    <version>1.12.1</version>
</dependency>
```

4）.html转文字

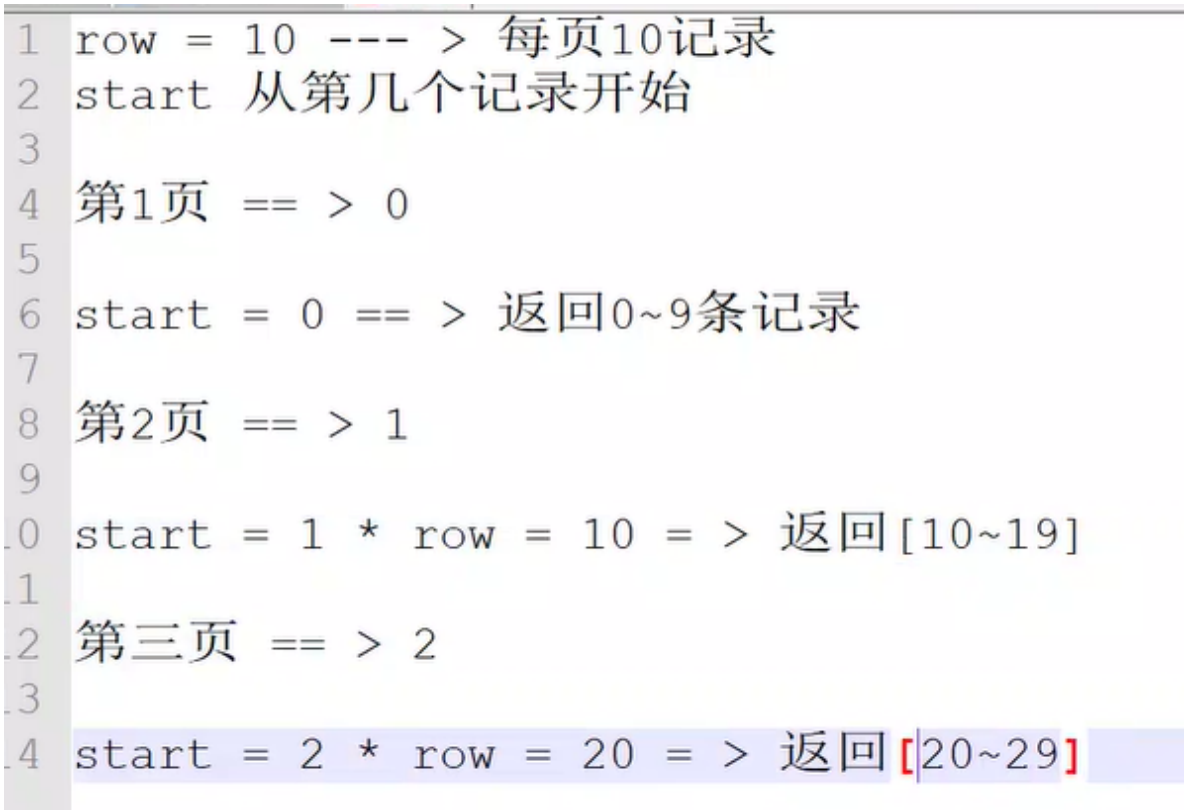
```
String content = Jsoup.parse(html).text();
```

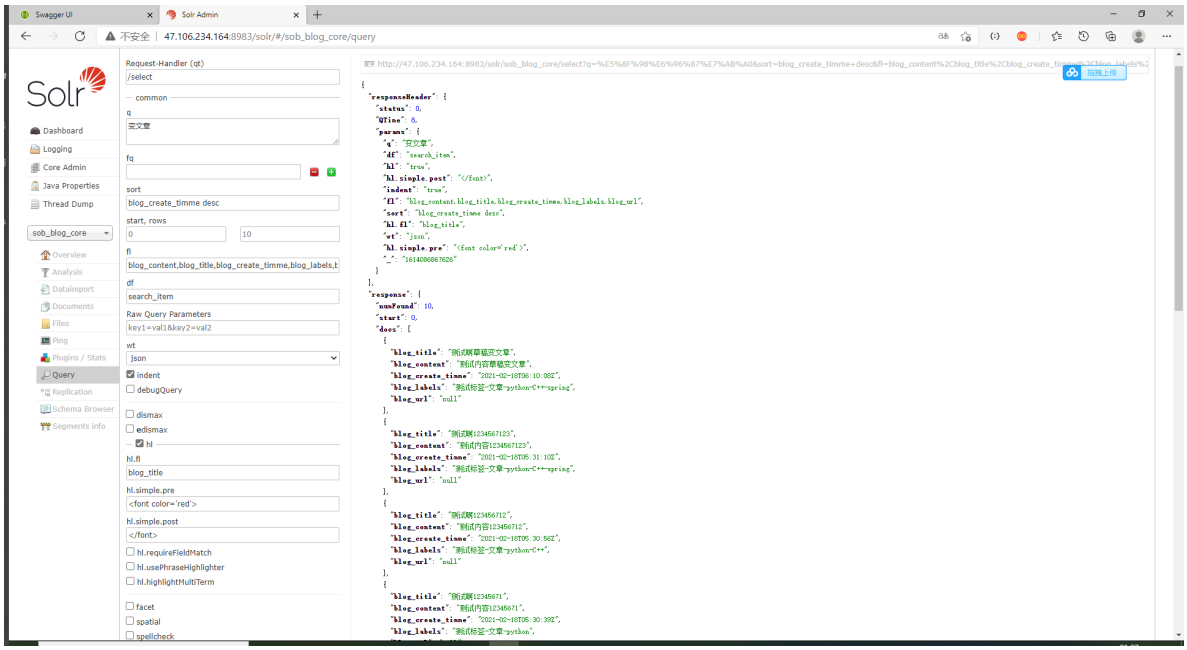
6.测试接口

```
package com.oldbai.halfmoon.controller.portal;

import com.oldbai.halfmoon.response.ResponseResult;
import com.oldbai.halfmoon.solr.SolrTestService;
import io.swagger.annotations.Api;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RequestMapping("/portal/search")
@CrossOrigin
@RestController
@Api(description = "门户-搜索")
public class SearchPortalController {

    @Autowired
    private SolrTestService solrTestService;

    @GetMapping("/test/solr/add")
    public ResponseResult solrAddTest() {
        solrTestService.solrAdd();
        return ResponseResult.SUCCESS("添加成功");
    }

    @GetMapping("/test/solr/addAll")
    public ResponseResult solrAddAllTest() {
        solrTestService.importAll();
        return ResponseResult.SUCCESS("添加所有成功");
    }

    @GetMapping("/test/solr/update")
    public ResponseResult solrUpdateTest() {
        solrTestService.solrUpdate();
        return ResponseResult.SUCCESS("更新成功");
    }

    @GetMapping("/test/solr/delete")
    public ResponseResult solrDeleteTest() {
        solrTestService.solrDelete();
        return ResponseResult.SUCCESS("删除成功");
    }

    @GetMapping("/test/solr/deleteAll")
    public ResponseResult solrDeleteAllTest() {
        solrTestService.solrDeleteAll();
        return ResponseResult.SUCCESS("删除所有成功");
    }
}
```

7.查询

- q：query查询
- fq：filter query 过滤查询
- sort：排序
- start,rows：分页
- fl：返回字段
- dl：默认的查询参考字段
- hl：高亮





1）.查询方法

- 定义返回结果实体类

```
package com.oldbai.halfmoon.vo;
```

```
import lombok.Data;
import org.apache.solr.client.solrj.beans.Field;

import java.io.Serializable;
import java.util.Date;

@Data
public class SearchResult implements Serializable {
    //
blog_content,blog_create_time,blog_labels,blog_url,blog_title,blog_view_coun
t
    @Field("id")
    private String id;
    @Field("blog_content")
    private String blogContent;
    @Field("blog_create_timme")
    private Date blogCreateTime;
    @Field("blog_labels")
    private String blogLabels;
    @Field("blog_url")
    private String blogUrl;
    @Field("blog_title")
    private String blogTitle;
    @Field("blog_view_count")
    private int blogViewCount;
}
```

- 定义返回结果自定义分页实体类

```
package com.oldbai.halfmoon.vo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.util.StringUtils;

import java.io.Serializable;
import java.util.List;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class PageList<T> implements Serializable {
    //做分页要多少数据
    //当前页码
    private int currentPage;
    //总数量
    private long totalCount;
    //每一页多少数量
    private long pageSize;
    //总页数 = 总的数量 / 每页数量
    private long totalPage;
    // 数据
```

```
    private List<T> contents;
    //是否第一页
    private boolean isFirst;
    //是否尾页
    private boolean isLast;

    public PageList(int currentPage, long totalCount, long pageSize) {
        this.currentPage = currentPage;
        this.totalCount = totalCount;
        this.pageSize = pageSize;
        if (this.totalCount % this.pageSize == 0) {
            this.totalPage = this.totalCount / this.pageSize;
        } else {
            this.totalPage = (this.totalCount / this.pageSize) + 1;
        }
        // 计算总的页数
        // 是否为第一页 / 是否为最后一页
        // 第一页为 0 ，最后一页为总的页码
        // 10 ， 每一页 有10 ==> 1
        // 100 ， 每一页 有10 ==> 10
        this.isFirst = this.currentPage == 1;
        this.isLast = this.currentPage == totalPage;
    }
}
```

- 编写搜索方法

```
@Override
    public ResponseResult doSearch(String keyword, int page, int size, String
categoryId, Integer sort) {
        //1、检查page和size
        page = Utils.getPage(page);
        size = Utils.getSize(size);
        SolrQuery solrQuery = new SolrQuery();
        //2、分页设置
        //先设置每页的数量
        solrQuery.setRows(size);
        //设置开始的位置
        //找个规律
        //第1页 -- > 0
        //第2页 == > size
        //第3页 == > 2*size
        //第4页 == > 3*size
        //第n页 == > (n-1)*size
        int start = (page - 1) * size;
        solrQuery.setStart(start);
        //solrQuery.set("start", start);
        //3、设置搜索条件
        //关键字
        solrQuery.set("df", "search_item");
        //条件过滤
        if (StringUtils.isEmpty(keyword)) {
            solrQuery.set("q", "*");
        } else {
            solrQuery.set("q", keyword);
        }
```

```
        //排序
        //排序有四个：根据时间的升序（1）和降序（2），根据浏览量的升序（3）和降序（4）
        if (sort != null) {
            if (sort == 1) {
                solrQuery.setSort("blog_create_timme", SolrQuery.ORDER.asc);
            } else if (sort == 2) {
                solrQuery.setSort("blog_create_timme", SolrQuery.ORDER.desc);
            } else if (sort == 3) {
                solrQuery.setSort("blog_view_count", SolrQuery.ORDER.asc);
            } else if (sort == 4) {
                solrQuery.setSort("blog_view_count", SolrQuery.ORDER.desc);
            }
        }
        //分类
        if (!StringUtils.isEmpty(categoryId)) {
            solrQuery.setFilterQueries("blog_category_id:" + categoryId);
        }
        //关键字高亮
        solrQuery.setHighlight(true);
        solrQuery.addHighlightField("blog_title,blog_content");
        solrQuery.setHighlightSimplePre("<font color='red'>");
        solrQuery.setHighlightSimplePost("</font>");
        solrQuery.setHighlightFragsize(500);
        //设置返回字段

 //blog_content,blog_create_time,blog_labels,blog_url,blog_title,blog_view_count

 solrQuery.addField("id,blog_content,blog_create_timme,blog_labels,blog_url,blog
_title,blog_view_count");
        //
        //4、搜索
        try {
            //4.1、处理搜索结果
            QueryResponse result = solrClient.query(solrQuery);
            //获取到高亮内容
            Map<String, Map<String, List<String>>> highlighting =
result.getHighlighting();
            //把数据转成bean类
            List<SearchResult> resultList = result.getBeans(SearchResult.class);
            //结果列表
            for (SearchResult item : resultList) {
                Map<String, List<String>> stringListMap =
highlighting.get(item.getId());
                List<String> blogContent = stringListMap.get("blog_content");
                if (!StringUtils.isEmpty(blogContent)) {
                    item.setBlogContent(blogContent.get(0));
                }
                List<String> blogTitle = stringListMap.get("blog_title");
                if (!StringUtils.isEmpty(blogTitle)) {
                    item.setBlogTitle(blogTitle.get(0));
                }
            }
            //5、返回搜索结果
            //包含内容
            //列表、页面、每页数量
            long numFound = result.getResults().getNumFound();
            PageList<SearchResult> pageList = new PageList<>(page, numFound,
size);
```

```
            pageList.setContents(resultList);
            //返回结果
            return ResponseResult.SUCCESS("搜索成功.", pageList);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return ResponseResult.FAILED("搜索失败，请稍后重试.");
    }
```

- 搜索接口

```
@ApiOperation("搜索功能")
    @GetMapping("/solr/select")
    public ResponseResult solrSelectAllTest(@RequestParam("keyword") String
keyword,
                                            @RequestParam("page") int page,
                                            @RequestParam("size") int size,
                                            @RequestParam(value = "categoryId",
required = false) String categoryId,
                                            @RequestParam(value = "sort",
required = false) Integer sort) {

        return articleService.doSearch(keyword, page, size, categoryId, sort);
    }
```

2）.添加文章时添加搜索内容

- 完善添加文章方法

```
/**
     * {
     * "categoryId": "805487266016395264",
     * "content": "测试内容1",
     * "labels": "测试标签-文章",
     * "state": "1",
     * "summary": "测试啊",
     * "title": "测试啊1",
     * "type": "0"
     * }
     */
    @Transactional
    @Override
    public ResponseResult postArticle(Article article) {
        //检查用户，获取到用户对象
        User sobUser = userService.checkUser();
        //未登录
        if (sobUser == null) {
            return ResponseResult.NO_LOGIN();
        }
        //检查数据
        //title、分类ID、内容、类型、摘要、标签
        String title = article.getTitle();
        if (StringUtils.isEmpty(title)) {
            return ResponseResult.FAILED("标题不可以为空.");
        }
```

```
        //2种，草稿和发布
        //获取文章状态
        String state = article.getState();
        //只接受两种状态 发布和草稿
        if (!Constants.Article.STATE_PUBLISH.equals(state) &&
                !Constants.Article.STATE_DRAFT.equals(state)) {
            //不支持此操作
            return ResponseResult.FAILED("不支持此操作");
        }
        //获取文章类型
        String type = article.getType();
        if (StringUtils.isEmpty(type)) {
            return ResponseResult.FAILED("类型不可以为空.");
        }
        //（0表示富文本，1表示markdown）
        if (!"0".equals(type) && !"1".equals(type)) {
            return ResponseResult.FAILED("类型格式不对.");
        }
        //以下检查是发布的检查，草稿不需要检查
        if (Constants.Article.STATE_PUBLISH.equals(state)) {
            //检查标题
            if (title.length() > Constants.Article.TITLE_MAX_LENGTH) {
                return ResponseResult.FAILED("文章标题不可以超过" +
Constants.Article.TITLE_MAX_LENGTH + "个字符");
            }
            //检查内容
            String content = article.getContent();
            if (StringUtils.isEmpty(content)) {
                return ResponseResult.FAILED("内容不可为空.");
            }
            //检查摘要
            String summary = article.getSummary();
            if (StringUtils.isEmpty(summary)) {
                return ResponseResult.FAILED("摘要不可以为空.");
            }
            if (summary.length() > Constants.Article.SUMMARY_MAX_LENGTH) {
                return ResponseResult.FAILED("摘要不可以超出" +
Constants.Article.SUMMARY_MAX_LENGTH + "个字符.");
            }
            //检查标签
            String labels = article.getLabels();
            //标签-标签1-标签2
            if (StringUtils.isEmpty(labels)) {
                return ResponseResult.FAILED("标签不可以为空.");
            }
        }
        //获取文章的ID
        String articleId = article.getId();
        if (StringUtils.isEmpty(articleId)) {
            //新内容,数据里没有的
            //补充数据：ID、创建时间、用户ID、更新时间
            article.setId(IdUtil.randomUUID() + "");
//            article.setCreateTime(new Date());
            article.setUserId(sobUser.getId());
            articleMapper.insert(article);
        } else {
            //更新内容，对状态进行处理，如果已经是发布的，则不能再保存为草稿
            Article articleFromDb = articleMapper.selectById(articleId);
```

```
            if (Constants.Article.STATE_PUBLISH.equals(articleFromDb.getState())
&&
                    Constants.Article.STATE_DRAFT.equals(state)) {
                //已经发布了，只能更新，不能保存草稿
                return ResponseResult.FAILED("已发布文章不支持成为草稿.");
            }
        }
        article.setUserId(sobUser.getId());
//        article.setUpdateTime(new Date());
        //保存到数据库里
        articleMapper.updateById(article);
        //TODO:保存到搜索的数据库里
        solrService.addArticle(article);
        //打散标签，入库，统计
        this.setupLabels(article.getLabels());
        //返回结果,只有一种case使用到这个ID
        //如果要做程序自动保存成草稿（比如说每30秒保存一次，就需要加上这个ID了，否则会创建多个
Item）
        return
ResponseResult.SUCCESS(Constants.Article.STATE_DRAFT.equals(state) ? "草稿保存成
功" :
                "文章发表成功.", article.getId());
    }
```

- 具体实现方法(需要进行判空处理)

```
@Override
    public void addArticle(Article article) {
        SolrInputDocument doc = new SolrInputDocument();
        doc.addField("id", article.getId());
        if (StringUtils.isEmpty(article.getViewCount())) {
            article.setViewCount(0);
        }
        doc.addField("blog_view_count", article.getViewCount());
        doc.addField("blog_title", article.getTitle());
        //对内容进行处理，去掉标签，提取纯文本内容
        //第一种是由markdown写的内容 ==> type = 1
        //第二种是有富文本内容 ==> type = 0
        //如果type等于 1 ，需要转换成 html
        //再由 html 转换成 纯文本
        //如果type等于 0 ，提取出纯文本
        String type = article.getType();
        String html = null;
        if (Constants.Article.TYPE_MARKDOWN.equals(type)) {
            //转成HTML
            // markdown to html
            MutableDataSet options = new MutableDataSet().set(Parser.EXTENSIONS,
Arrays.asList(
                    TablesExtension.create(),
                    JekyllTagExtension.create(),
                    TocExtension.create(),
                    SimTocExtension.create()
            ));
            Parser parser = Parser.builder(options).build();
            HtmlRenderer renderer = HtmlRenderer.builder(options).build();
            Node document = parser.parse(article.getContent());
            html = renderer.render(document);
```

```
        } else if (Constants.Article.TYPE_RICH_TEXT.equals(type)) {
            //富文本
            html = article.getContent();
        }
        //到了这里不管是什么都变成了 html
        //HTML转Text
        String content = Jsoup.parse(html).text();
        doc.addField("blog_content", content);
        if (StringUtils.isEmpty(article.getCreateTime())) {
            article.setCreateTime(new Date());
        }
        doc.addField("blog_create_timme", article.getCreateTime());
        doc.addField("blog_labels", article.getLabels());
        doc.addField("blog_url", "null");
        doc.addField("blog_category_id", article.getCategoryId());
        try {
            solrClient.add(doc);
            solrClient.commit();
        } catch (SolrServerException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

3）.删除时删除搜索内容

- 完善删除文章方法

```
@Transactional
    @Override
    public ResponseResult deleteArticleById(String articleId) {
        Article article = articleMapper.selectById(articleId);
        this.deleteLabels(article.getLabels());
        QueryWrapper<Comment> queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("article_id", articleId);
        commentMapper.delete(queryWrapper);
        int result = articleMapper.deleteById(articleId);
        if (result > 0) {
            redisUtil.del(Constants.Article.KEY_ARTICLE_CACHE + articleId);
            redisUtil.del(Constants.Article.KEY_ARTICLE_VIEW_COUNT + articleId);
            solrService.deleteArticle(articleId);
            return ResponseResult.SUCCESS("文章删除成功.");
        }
        return ResponseResult.FAILED("文章不存在.");
    }
```

- 具体方法实现

```
@Override
    public void deleteArticle(String id) {
        //单独删除一条
        try {
            solrClient.deleteById(id);
            solrClient.commit();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

4）.更新时更新搜索内容

- 完善更新文章方法

```
 /**
     * 如果有审核机制：审核中的文章-->只有管理员和作者自己可以获取
     * 有草稿、删除、置顶的、已经发布的
     * 删除的不能获取、其他都可以获取
     * <p>
     * 统计文章的阅读量：
     * 精确点，对ip 进行处理
     * 对同一个 ip 那就不保存
     * <p>
     * 先把阅读量保存到 redis 中，文章也会在 redis中缓存一份。
     * 比如说：
     * 十分钟，当文章没的时候，从 mysql 中取，同时更新阅读量
     * 十分钟后，在下一次访问的时候更新一次。
     *
     * @param articleId
     * @return
     */
    @Transactional
    @Override
    public ResponseResult getArticleById(String articleId) {
        //先从redis里获取文章
        //如果没有再从mysql中取文章
        Article article = (Article)
redisUtil.get(Constants.Article.KEY_ARTICLE_CACHE + articleId);
        if (!StringUtils.isEmpty(article)) {
            //阅读量 + 1
            Integer count = (Integer)
redisUtil.get(Constants.Article.KEY_ARTICLE_VIEW_COUNT + articleId);
            if (StringUtils.isEmpty(count)) {
                count = 1;
            } else {
                count = count + 1;
            }
            redisUtil.set(Constants.Article.KEY_ARTICLE_VIEW_COUNT + articleId,
count);
            article.setViewCount(count);
            articleMapper.updateById(article);
            //可以返回
            return ResponseResult.SUCCESS("获取文章成功.", article);
        } else {
            //查询出文章
```

```
            article = articleMapper.selectById(articleId);
        }

        if (article == null) {
            return ResponseResult.FAILED("文章不存在.");
        }
        //判断文章状态
        String state = article.getState();
        if (Constants.Article.STATE_PUBLISH.equals(state) ||
                Constants.Article.STATE_TOP.equals(state)) {
            //正常发布的状态才能增加阅读量
            redisUtil.set(Constants.Article.KEY_ARTICLE_CACHE + articleId,
article, Constants.TimeValue.MIN * 5);
            //设置阅读量的Key ，先从redis里面拿，如果redis里面没有，就从 article 中获取，
并且添加到redis中
            Integer viewCount = (Integer)
redisUtil.get(Constants.Article.KEY_ARTICLE_VIEW_COUNT + articleId);
            if (StringUtils.isEmpty(viewCount)) {
                int newCount = article.getViewCount() + 1;
                redisUtil.set(Constants.Article.KEY_ARTICLE_VIEW_COUNT +
article.getId(), newCount);
            } else {
                //有的话更新到数据库中
                int newCount = viewCount;
                article.setViewCount(newCount);
            }
            articleMapper.updateById(article);
            solrService.updateArticle(article);
            //可以返回
            return ResponseResult.SUCCESS("获取文章成功.", article);
        }
        //如果是删除/草稿，需要管理角色
        User sobUser = userService.checkUser();
        if (sobUser == null ||
!Constants.User.ROLE_ADMIN.equals(sobUser.getRoles())) {
            return ResponseResult.NO_PERMISSION();
        }
        //返回结果
        return ResponseResult.SUCCESS("获取文章成功.", article);
    }
```

- 具体方法实现

```
@Override
    public void updateArticle(Article article) {
        this.addArticle(article);
    }
```