一、创建项目，上传至github

二、数据库

1. 数据库的设计

```sql
/*
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;

-- ----------------------------
-- 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 ('3480515b5567149254133dac6ecdd14b', '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;
```

```sql
-- ----------------------------
-- 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

```xml
<!--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;
```

```java
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;

/**
 * @author 老白
```

```java
 */
@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

```properties
# 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;

/**
 * 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]);
```

```java
            } 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;
        }
    }

    /**
```

```java
     * 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);
    }

    /**
     * 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
     *
```

```java
     * @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;
```

```java
        } 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
 */
```

```java
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;
```

```java
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

```java
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) {
```

```java
        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）

- 测试数据

```
{

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

  "password": "admin",

  "userName": "admin"
}
```

- 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);
    }
```

- 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

- 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);
    }
```

- 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);
```

```java
        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

- 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>
```

- 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());
        }


    }
```

- 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 = 0l;
        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

- 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>
```

- 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;
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");
    }
```

```java
    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();

    }

}
```

- 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);
    }
```

- 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.检查邮箱是否注册

- 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);
    }
```

- 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

- 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);
    }
```

- 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.");

    }
```