# 项目实战-前后端分离博客系统

# 项目实战-前后端分离博客系统

## 1.课程介绍

* 纯后端讲解

* 完整的前台后台代码编写

* 主流技术栈（SpringBoot,MybatisPlus,SpringSecurity,EasyExcel,Swagger2,Redis,Echarts,Vue,ElementUI....）

* 完善细致的需求分析

* 由易到难循序渐进

## 2.创建工程

我们有前台和后台两套系统。两套系统的前端工程都已经提供好了。所以我们只需要写两套系统的后端。

但是大家思考下，实际上两套后端系统的很多内容是可能重复的。这里如果我们只是单纯的创建两个后端工程。那么就会有大量的重复代码，并且需要修改的时候也需要修改两次。这就是代码复用性不高。

所以我们需要创建多模块项目，两套系统可能都会用到的代码可以写到一个公共模块中，让前台系统和后台系统分别取依赖公共模块。

① 创建父模块

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.sangeng</groupId>  
 <artifactId>SGBlog</artifactId>  
 <packaging>pom</packaging>  
 <version>1.0-SNAPSHOT</version>  
 <modules>  
 <module>sangeng-framework</module>  
 <module>sangeng-admin</module>  
 <module>sangeng-blog</module>  
 </modules>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <java.version>1.8</java.version>  
 </properties>  
 <dependencyManagement>  
  
  
 <dependencies>  
 <!-- SpringBoot的依赖配置-->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-dependencies</artifactId>  
 <version>2.5.0</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 <!--fastjson依赖-->  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>fastjson</artifactId>  
 <version>1.2.33</version>  
 </dependency>  
 <!--jwt依赖-->  
 <dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt</artifactId>  
 <version>0.9.0</version>  
 </dependency>  
 <!--mybatisPlus依赖-->  
 <dependency>  
 <groupId>com.baomidou</groupId>  
 <artifactId>mybatis-plus-boot-starter</artifactId>  
 <version>3.4.3</version>  
 </dependency>  
  
 <!--阿里云OSS-->  
 <dependency>  
 <groupId>com.aliyun.oss</groupId>  
 <artifactId>aliyun-sdk-oss</artifactId>  
 <version>3.10.2</version>  
 </dependency>  
  
  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>easyexcel</artifactId>  
 <version>3.0.5</version>  
 </dependency>  
  
 <dependency>  
 <groupId>io.springfox</groupId>  
 <artifactId>springfox-swagger2</artifactId>  
 <version>2.9.2</version>  
 </dependency>  
 <dependency>  
 <groupId>io.springfox</groupId>  
 <artifactId>springfox-swagger-ui</artifactId>  
 <version>2.9.2</version>  
 </dependency>  
 </dependencies>  
  
  
 </dependencyManagement>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.1</version>  
 <configuration>  
 <source>${java.version}</source>  
 <target>${java.version}</target>  
 <encoding>${project.build.sourceEncoding}</encoding>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
</project>

②创建公共子模块 sangeng-framework

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <parent>  
 <artifactId>SGBlog</artifactId>  
 <groupId>com.sangeng</groupId>  
 <version>1.0-SNAPSHOT</version>  
 </parent>  
 <modelVersion>4.0.0</modelVersion>  
  
 <artifactId>sangeng-framework</artifactId>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <!--lombk-->  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <optional>true</optional>  
 </dependency>  
 <!--junit-->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <!--SpringSecurity启动器-->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 </dependency>  
 <!--redis依赖-->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-redis</artifactId>  
 </dependency>  
 <!--fastjson依赖-->  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>fastjson</artifactId>  
 </dependency>  
 <!--jwt依赖-->  
 <dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt</artifactId>  
 </dependency>  
 <!--mybatisPlus依赖-->  
 <dependency>  
 <groupId>com.baomidou</groupId>  
 <artifactId>mybatis-plus-boot-starter</artifactId>  
 </dependency>  
 <!--mysql数据库驱动-->  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 </dependency>  
  
 <!--阿里云OSS-->  
 <dependency>  
 <groupId>com.aliyun.oss</groupId>  
 <artifactId>aliyun-sdk-oss</artifactId>  
 </dependency>  
  
 <!--AOP-->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-aop</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>easyexcel</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>io.springfox</groupId>  
 <artifactId>springfox-swagger2</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>io.springfox</groupId>  
 <artifactId>springfox-swagger-ui</artifactId>  
 </dependency>  
  
 </dependencies>  
</project>

③创建博客后台模块sangeng-admin

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <parent>  
 <artifactId>SGBlog</artifactId>  
 <groupId>com.sangeng</groupId>  
 <version>1.0-SNAPSHOT</version>  
 </parent>  
 <modelVersion>4.0.0</modelVersion>  
  
 <artifactId>sangeng-admin</artifactId>  
  
 <dependencies>  
 <dependency>  
 <groupId>com.sangeng</groupId>  
 <artifactId>sangeng-framework</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 </dependency>  
 </dependencies>  
</project>

④创建博客前台模块sangeng-blog

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <parent>  
 <artifactId>SGBlog</artifactId>  
 <groupId>com.sangeng</groupId>  
 <version>1.0-SNAPSHOT</version>  
 </parent>  
 <modelVersion>4.0.0</modelVersion>  
  
 <artifactId>sangeng-blog</artifactId>  
  
 <dependencies>  
 <dependency>  
 <groupId>com.sangeng</groupId>  
 <artifactId>sangeng-framework</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 </dependency>  
 </dependencies>  
  
</project>

## 3.博客前台

### 3.0 准备工作

#### 3.1 SpringBoot和MybatisPuls整合配置测试

①创建启动类

/\*\*  
 \* @Author 三更 B站： https://space.bilibili.com/663528522  
 \*/  
@SpringBootApplication  
@MapperScan("com.sangeng.mapper")  
public class SanGengBlogApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.run(SanGengBlogApplication.class,args);  
 }  
}

②创建application.yml配置文件

server:  
 port: 7777  
spring:  
 datasource:  
 url: jdbc:mysql://localhost:3306/sg\_blog?characterEncoding=utf-8&serverTimezone=Asia/Shanghai  
 username: root  
 password: root  
 driver-class-name: com.mysql.cj.jdbc.Driver  
 servlet:  
 multipart:  
 max-file-size: 2MB  
 max-request-size: 5MB  
mybatis-plus:  
 configuration:  
 # 日志  
 log-impl: org.apache.ibatis.logging.stdout.StdOutImpl  
 global-config:  
 db-config:  
 logic-delete-field: delFlag  
 logic-delete-value: 1  
 logic-not-delete-value: 0  
 id-type: auto

③ SQL语句

SQL脚本：SGBlog\资源\SQL\sg\_article.sql

④ 创建实体类，Mapper，Service

注意思考这些文件应该写在哪个模块下？

@SuppressWarnings("serial")  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
@TableName("sg\_article")  
public class Article {  
 @TableId  
 private Long id;  
 //标题  
 private String title;  
 //文章内容  
 private String content;  
 //文章类型:1 文章 2草稿  
 private String type;  
 //文章摘要  
 private String summary;  
 //所属分类id  
 private Long categoryId;  
 //缩略图  
 private String thumbnail;  
 //是否置顶（0否，1是）  
 private String isTop;  
 //状态（0已发布，1草稿）  
 private String status;  
 //评论数  
 private Integer commentCount;  
 //访问量  
 private Long viewCount;  
 //是否允许评论 1是，0否  
 private String isComment;  
   
 private Long createBy;  
   
 private Date createTime;  
   
 private Long updateBy;  
   
 private Date updateTime;  
 //删除标志（0代表未删除，1代表已删除）  
 private Integer delFlag;  
  
}

public interface ArticleMapper extends BaseMapper<Article> {  
  
  
}

public interface ArticleService extends IService<Article> {  
}

@Service  
public class ArticleServiceImpl extends ServiceImpl<ArticleMapper, Article> implements ArticleService {  
  
}

⑤ 创建Controller测试接口

注意思考这些文件应该写在哪个模块下？

@RestController  
@RequestMapping("/article")  
public class ArticleController {  
  
 @Autowired  
 private ArticleService articleService;  
  
 @GetMapping("/list")  
 public List<Article> test(){  
 return articleService.list();  
 }  
}

我们可以暂时先注释掉sangeng-framework中的SpringSecurity依赖方便测试

### 3.1 热门文章列表

#### 3.1.0 文章表分析

通过需求去分析需要有哪些字段。

#### 3.1.1 需求

需要查询浏览量最高的前10篇文章的信息。要求展示文章标题和浏览量。把能让用户自己点击跳转到具体的文章详情进行浏览。

注意：不能把草稿展示出来，不能把删除了的文章查询出来。要按照浏览量进行降序排序。

#### 3.1.2 接口设计

见接口文档

#### 3.1.3 基础版本代码实现

①准备工作

统一响应类和响应枚举

package com.sangeng.domain;  
  
import com.fasterxml.jackson.annotation.JsonInclude;  
import com.sangeng.enums.AppHttpCodeEnum;  
  
import java.io.Serializable;  
  
@JsonInclude(JsonInclude.Include.NON\_NULL)  
public class ResponseResult<T> implements Serializable {  
 private Integer code;  
 private String msg;  
 private T data;  
  
 public ResponseResult() {  
 this.code = AppHttpCodeEnum.SUCCESS.getCode();  
 this.msg = AppHttpCodeEnum.SUCCESS.getMsg();  
 }  
  
 public ResponseResult(Integer code, T data) {  
 this.code = code;  
 this.data = data;  
 }  
  
 public ResponseResult(Integer code, String msg, T data) {  
 this.code = code;  
 this.msg = msg;  
 this.data = data;  
 }  
  
 public ResponseResult(Integer code, String msg) {  
 this.code = code;  
 this.msg = msg;  
 }  
  
 public static ResponseResult errorResult(int code, String msg) {  
 ResponseResult result = new ResponseResult();  
 return result.error(code, msg);  
 }  
 public static ResponseResult okResult() {  
 ResponseResult result = new ResponseResult();  
 return result;  
 }  
 public static ResponseResult okResult(int code, String msg) {  
 ResponseResult result = new ResponseResult();  
 return result.ok(code, null, msg);  
 }  
  
 public static ResponseResult okResult(Object data) {  
 ResponseResult result = setAppHttpCodeEnum(AppHttpCodeEnum.SUCCESS, AppHttpCodeEnum.SUCCESS.getMsg());  
 if(data!=null) {  
 result.setData(data);  
 }  
 return result;  
 }  
  
 public static ResponseResult errorResult(AppHttpCodeEnum enums){  
 return setAppHttpCodeEnum(enums,enums.getMsg());  
 }  
  
 public static ResponseResult errorResult(AppHttpCodeEnum enums, String msg){  
 return setAppHttpCodeEnum(enums,msg);  
 }  
  
 public static ResponseResult setAppHttpCodeEnum(AppHttpCodeEnum enums){  
 return okResult(enums.getCode(),enums.getMsg());  
 }  
  
 private static ResponseResult setAppHttpCodeEnum(AppHttpCodeEnum enums, String msg){  
 return okResult(enums.getCode(),msg);  
 }  
  
 public ResponseResult<?> error(Integer code, String msg) {  
 this.code = code;  
 this.msg = msg;  
 return this;  
 }  
  
 public ResponseResult<?> ok(Integer code, T data) {  
 this.code = code;  
 this.data = data;  
 return this;  
 }  
  
 public ResponseResult<?> ok(Integer code, T data, String msg) {  
 this.code = code;  
 this.data = data;  
 this.msg = msg;  
 return this;  
 }  
  
 public ResponseResult<?> ok(T data) {  
 this.data = data;  
 return this;  
 }  
  
 public Integer getCode() {  
 return code;  
 }  
  
 public void setCode(Integer code) {  
 this.code = code;  
 }  
  
 public String getMsg() {  
 return msg;  
 }  
  
 public void setMsg(String msg) {  
 this.msg = msg;  
 }  
  
 public T getData() {  
 return data;  
 }  
  
 public void setData(T data) {  
 this.data = data;  
 }  
  
  
  
}

package com.sangeng.enums;  
  
public enum AppHttpCodeEnum {  
 // 成功  
 SUCCESS(200,"操作成功"),  
 // 登录  
 NEED\_LOGIN(401,"需要登录后操作"),  
 NO\_OPERATOR\_AUTH(403,"无权限操作"),  
 SYSTEM\_ERROR(500,"出现错误"),  
 USERNAME\_EXIST(501,"用户名已存在"),  
 PHONENUMBER\_EXIST(502,"手机号已存在"), EMAIL\_EXIST(503, "邮箱已存在"),  
 REQUIRE\_USERNAME(504, "必需填写用户名"),  
 LOGIN\_ERROR(505,"用户名或密码错误");  
 int code;  
 String msg;  
  
 AppHttpCodeEnum(int code, String errorMessage){  
 this.code = code;  
 this.msg = errorMessage;  
 }  
  
 public int getCode() {  
 return code;  
 }  
  
 public String getMsg() {  
 return msg;  
 }  
}

② 代码实现

@RestController  
@RequestMapping("/article")  
public class ArticleController {  
  
 @Autowired  
 private ArticleService articleService;  
   
 @GetMapping("/hotArticleList")  
 public ResponseResult hotArticleList(){  
  
 ResponseResult result = articleService.hotArticleList();  
 return result;  
 }  
}

public interface ArticleService extends IService<Article> {  
 ResponseResult hotArticleList();  
}

@Service  
public class ArticleServiceImpl extends ServiceImpl<ArticleMapper, Article> implements ArticleService {  
  
 @Override  
 public ResponseResult hotArticleList() {  
 //查询热门文章 封装成ResponseResult返回  
 LambdaQueryWrapper<Article> queryWrapper = new LambdaQueryWrapper<>();  
 //必须是正式文章  
 queryWrapper.eq(Article::getStatus,0);  
 //按照浏览量进行排序  
 queryWrapper.orderByDesc(Article::getViewCount);  
 //最多只查询10条  
 Page<Article> page = new Page(1,10);  
 page(page,queryWrapper);  
  
 List<Article> articles = page.getRecords();  
 return ResponseResult.okResult(articles);  
 }  
}

③ 解决跨域问题

@Configuration  
public class WebConfig implements WebMvcConfigurer {  
  
 @Override  
 public void addCorsMappings(CorsRegistry registry) {  
 // 设置允许跨域的路径  
 registry.addMapping("/\*\*")  
 // 设置允许跨域请求的域名  
 .allowedOriginPatterns("\*")  
 // 是否允许cookie  
 .allowCredentials(true)  
 // 设置允许的请求方式  
 .allowedMethods("GET", "POST", "DELETE", "PUT")  
 // 设置允许的header属性  
 .allowedHeaders("\*")  
 // 跨域允许时间  
 .maxAge(3600);  
 }  
  
}

#### 3.1.4 使用VO优化

目前我们的响应格式其实是不符合接口文档的标准的，多返回了很多字段。这是因为我们查询出来的结果是Article来封装的，Article中字段比较多。

我们在项目中一般最后还要把VO来接受查询出来的结果。一个接口对应一个VO，这样即使接口响应字段要修改也只要改VO即可。

@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class HotArticleVo {  
 private Long id;  
 //标题  
 private String title;  
  
 //访问量  
 private Long viewCount;  
}

@Service  
public class ArticleServiceImpl extends ServiceImpl<ArticleMapper, Article> implements ArticleService {  
  
 @Override  
 public ResponseResult hotArticleList() {  
 //查询热门文章 封装成ResponseResult返回  
 LambdaQueryWrapper<Article> queryWrapper = new LambdaQueryWrapper<>();  
 //必须是正式文章  
 queryWrapper.eq(Article::getStatus,0);  
 //按照浏览量进行排序  
 queryWrapper.orderByDesc(Article::getViewCount);  
 //最多只查询10条  
 Page<Article> page = new Page(1,10);  
 page(page,queryWrapper);  
  
 List<Article> articles = page.getRecords();  
 //bean拷贝  
 List<HotArticleVo> articleVos = new ArrayList<>();  
 for (Article article : articles) {  
 HotArticleVo vo = new HotArticleVo();  
 BeanUtils.copyProperties(article,vo);  
 articleVos.add(vo);  
 }  
  
 return ResponseResult.okResult(articleVos);  
 }  
}

#### 3.1.5 字面值处理

实际项目中都不允许直接在代码中使用字面值。都需要定义成常量来使用。这种方式有利于提高代码的可维护性。

public class SystemConstants  
{  
 /\*\*  
 \* 文章是草稿  
 \*/  
 public static final int ARTICLE\_STATUS\_DRAFT = 1;  
 /\*\*  
 \* 文章是正常分布状态  
 \*/  
 public static final int ARTICLE\_STATUS\_NORMAL = 0;  
   
}

@Service  
public class ArticleServiceImpl extends ServiceImpl<ArticleMapper, Article> implements ArticleService {  
  
 @Override  
 public ResponseResult hotArticleList() {  
 //查询热门文章 封装成ResponseResult返回  
 LambdaQueryWrapper<Article> queryWrapper = new LambdaQueryWrapper<>();  
 //必须是正式文章  
 queryWrapper.eq(Article::getStatus, SystemConstants.ARTICLE\_STATUS\_NORMAL);  
 //按照浏览量进行排序  
 queryWrapper.orderByDesc(Article::getViewCount);  
 //最多只查询10条  
 Page<Article> page = new Page(1,10);  
 page(page,queryWrapper);  
  
 List<Article> articles = page.getRecords();  
 //bean拷贝  
 List<HotArticleVo> articleVos = new ArrayList<>();  
 for (Article article : articles) {  
 HotArticleVo vo = new HotArticleVo();  
 BeanUtils.copyProperties(article,vo);  
 articleVos.add(vo);  
 }  
  
 return ResponseResult.okResult(articleVos);  
 }  
}

### 3.2 Bean拷贝工具类封装

public class BeanCopyUtils {  
  
 private BeanCopyUtils() {  
 }  
  
 public static <V> V copyBean(Object source,Class<V> clazz) {  
 //创建目标对象  
 V result = null;  
 try {  
 result = clazz.newInstance();  
 //实现属性copy  
 BeanUtils.copyProperties(source, result);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 //返回结果  
 return result;  
 }  
 public static <O,V> List<V> copyBeanList(List<O> list,Class<V> clazz){  
 return list.stream()  
 .map(o -> copyBean(o, clazz))  
 .collect(Collectors.toList());  
 }  
}

### 3.2 查询分类列表

#### 3.2.0 分类表分析

通过需求去分析需要有哪些字段。

建表SQL及初始化数据见：SGBlog\资源\SQL\sg\_category.sql

#### 3.2.1 需求

[MISSING IMAGE: , ]

页面上需要展示分类列表，用户可以点击具体的分类查看该分类下的文章列表。

注意： ①要求只展示有发布正式文章的分类 ②必须是正常状态的分类

#### 3.2.2 接口设计

见接口文档

#### 3.2.3 EasyCode代码模板

##导入宏定义  
$!{define.vm}  
  
##保存文件（宏定义）  
#save("/entity", ".java")  
  
##包路径（宏定义）  
#setPackageSuffix("entity")  
  
##自动导入包（全局变量）  
$!{autoImport.vm}  
  
import java.io.Serializable;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
import com.baomidou.mybatisplus.annotation.TableId;  
import com.baomidou.mybatisplus.annotation.TableName;  
##表注释（宏定义）  
#tableComment("表实体类")  
@SuppressWarnings("serial")  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
@TableName("$!{tableInfo.obj.name}")  
public class $!{tableInfo.name} {  
#foreach($column in $tableInfo.pkColumn)  
 #if(${column.comment})//${column.comment}#end  
@TableId  
 private $!{tool.getClsNameByFullName($column.type)} $!{column.name};  
#end  
  
#foreach($column in $tableInfo.otherColumn)  
 #if(${column.comment})//${column.comment}#end  
  
 private $!{tool.getClsNameByFullName($column.type)} $!{column.name};  
#end  
  
  
  
}

##导入宏定义  
$!{define.vm}  
  
##设置表后缀（宏定义）  
#setTableSuffix("Mapper")  
  
##保存文件（宏定义）  
#save("/mapper", "Mapper.java")  
  
##包路径（宏定义）  
#setPackageSuffix("mapper")  
  
import com.baomidou.mybatisplus.core.mapper.BaseMapper;  
  
  
##表注释（宏定义）  
#tableComment("表数据库访问层")  
public interface $!{tableName} extends BaseMapper<$!tableInfo.name> {  
  
}

##导入宏定义  
$!{define.vm}  
  
##设置表后缀（宏定义）  
#setTableSuffix("Service")  
  
##保存文件（宏定义）  
#save("/service", "Service.java")  
  
##包路径（宏定义）  
#setPackageSuffix("service")  
  
import com.baomidou.mybatisplus.extension.service.IService;  
  
  
##表注释（宏定义）  
#tableComment("表服务接口")  
public interface $!{tableName} extends IService<$!tableInfo.name> {  
  
}

##导入宏定义  
$!{define.vm}  
  
##设置表后缀（宏定义）  
#setTableSuffix("ServiceImpl")  
  
##保存文件（宏定义）  
#save("/service/impl", "ServiceImpl.java")  
  
##包路径（宏定义）  
#setPackageSuffix("service.impl")  
  
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;  
import org.springframework.stereotype.Service;  
  
##表注释（宏定义）  
#tableComment("表服务实现类")  
@Service("$!tool.firstLowerCase($tableInfo.name)Service")  
public class $!{tableName} extends ServiceImpl<$!{tableInfo.name}Mapper, $!{tableInfo.name}> implements $!{tableInfo.name}Service {  
  
}

#### 3.2.4 代码实现

@RestController  
@RequestMapping("/category")  
public class CategoryController {  
  
 @Autowired  
 private CategoryService categoryService;  
  
 @GetMapping("/getCategoryList")  
 public ResponseResult getCategoryList(){  
 return categoryService.getCategoryList();  
 }  
}

public interface CategoryService extends IService<Category> {  
  
  
 ResponseResult getCategoryList();  
  
}

@Service("categoryService")  
public class CategoryServiceImpl extends ServiceImpl<CategoryMapper, Category> implements CategoryService {  
  
 @Autowired  
 private ArticleService articleService;  
  
 @Override  
 public ResponseResult getCategoryList() {  
 //查询文章表 状态为已发布的文章  
 LambdaQueryWrapper<Article> articleWrapper = new LambdaQueryWrapper<>();  
 articleWrapper.eq(Article::getStatus,SystemConstants.ARTICLE\_STATUS\_NORMAL);  
 List<Article> articleList = articleService.list(articleWrapper);  
 //获取文章的分类id，并且去重  
 Set<Long> categoryIds = articleList.stream()  
 .map(article -> article.getCategoryId())  
 .collect(Collectors.toSet());  
  
 //查询分类表  
 List<Category> categories = listByIds(categoryIds);  
 categories = categories.stream().  
 filter(category -> SystemConstants.STATUS\_NORMAL.equals(category.getStatus()))  
 .collect(Collectors.toList());  
 //封装vo  
 List<CategoryVo> categoryVos = BeanCopyUtils.copyBeanList(categories, CategoryVo.class);  
  
 return ResponseResult.okResult(categoryVos);  
 }  
}

### 3.3 分页查询文章列表

#### 3.3.1 需求

在首页和分类页面都需要查询文章列表。

首页：查询所有的文章

分类页面：查询对应分类下的文章

要求：①只能查询正式发布的文章 ②置顶的文章要显示在最前面

#### 3.3.2 接口设计

见文档

#### 3.3.3 代码实现

MP支持分页配置

/\*\*  
 \* @Author 三更 B站： https://space.bilibili.com/663528522  
 \*/  
@Configuration  
public class MbatisPlusConfig {  
  
 /\*\*  
 \* 3.4.0之后版本  
 \* @return  
 \*/  
 @Bean  
 public MybatisPlusInterceptor mybatisPlusInterceptor(){  
 MybatisPlusInterceptor mybatisPlusInterceptor = new MybatisPlusInterceptor();  
 mybatisPlusInterceptor.addInnerInterceptor(new PaginationInnerInterceptor());  
 return mybatisPlusInterceptor;  
 }  
}

在ArticleController中

@GetMapping("/articleList")  
 public ResponseResult articleList(Integer pageNum,Integer pageSize,Long categoryId){  
 return articleService.articleList(pageNum,pageSize,categoryId);  
 }

在ArticleService中

ResponseResult articleList(Integer pageNum, Integer pageSize, Long categoryId);

在ArticleServiceImpl中

@Service  
public class ArticleServiceImpl extends ServiceImpl<ArticleMapper, Article> implements ArticleService {  
  
 @Autowired  
 private CategoryService categoryService;  
  
 @Override  
 public ResponseResult hotArticleList() {  
 //查询热门文章 封装成ResponseResult返回  
 LambdaQueryWrapper<Article> queryWrapper = new LambdaQueryWrapper<>();  
 //必须是正式文章  
 queryWrapper.eq(Article::getStatus, SystemConstants.ARTICLE\_STATUS\_NORMAL);  
 //按照浏览量进行排序  
 queryWrapper.orderByDesc(Article::getViewCount);  
 //最多只查询10条  
 Page<Article> page = new Page(1,10);  
 page(page,queryWrapper);  
  
 List<Article> articles = page.getRecords();  
 //bean拷贝  
// List<HotArticleVo> articleVos = new ArrayList<>();  
// for (Article article : articles) {  
// HotArticleVo vo = new HotArticleVo();  
// BeanUtils.copyProperties(article,vo);  
// articleVos.add(vo);  
// }  
 List<HotArticleVo> vs = BeanCopyUtils.copyBeanList(articles, HotArticleVo.class);  
 return ResponseResult.okResult(vs);  
 }  
  
 @Override  
 public ResponseResult articleList(Integer pageNum, Integer pageSize, Long categoryId) {  
 //查询条件  
 LambdaQueryWrapper<Article> lambdaQueryWrapper = new LambdaQueryWrapper<>();  
 // 如果 有categoryId 就要 查询时要和传入的相同  
 lambdaQueryWrapper.eq(Objects.nonNull(categoryId)&&categoryId>0 ,Article::getCategoryId,categoryId);  
 // 状态是正式发布的  
 lambdaQueryWrapper.eq(Article::getStatus,SystemConstants.ARTICLE\_STATUS\_NORMAL);  
 // 对isTop进行降序  
 lambdaQueryWrapper.orderByDesc(Article::getIsTop);  
  
 //分页查询  
 Page<Article> page = new Page<>(pageNum,pageSize);  
 page(page,lambdaQueryWrapper);  
  
 List<Article> articles = page.getRecords();  
 //查询categoryName  
 articles.stream()  
 .map(article -> article.setCategoryName(categoryService.getById(article.getCategoryId()).getName()))  
 .collect(Collectors.toList());  
 //articleId去查询articleName进行设置  
// for (Article article : articles) {  
// Category category = categoryService.getById(article.getCategoryId());  
// article.setCategoryName(category.getName());  
// }  
  
 //封装查询结果  
 List<ArticleListVo> articleListVos = BeanCopyUtils.copyBeanList(page.getRecords(), ArticleListVo.class);  
  
 PageVo pageVo = new PageVo(articleListVos,page.getTotal());  
 return ResponseResult.okResult(pageVo);  
 }  
}

PageVo

@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class PageVo {  
 private List rows;  
 private Long total;  
}

ArticleListVo

@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class ArticleListVo {  
  
 private Long id;  
 //标题  
 private String title;  
 //文章摘要  
 private String summary;  
 //所属分类名  
 private String categoryName;  
 //缩略图  
 private String thumbnail;  
  
  
 //访问量  
 private Long viewCount;  
  
 private Date createTime;  
  
  
}

在Article中增加一个字段

@TableField(exist = false)  
 private String categoryName;

#### 3.3.4 FastJson配置

@Bean//使用@Bean注入fastJsonHttpMessageConvert  
 public HttpMessageConverter fastJsonHttpMessageConverters() {  
 //1.需要定义一个Convert转换消息的对象  
 FastJsonHttpMessageConverter fastConverter = new FastJsonHttpMessageConverter();  
 FastJsonConfig fastJsonConfig = new FastJsonConfig();  
 fastJsonConfig.setSerializerFeatures(SerializerFeature.PrettyFormat);  
 fastJsonConfig.setDateFormat("yyyy-MM-dd HH:mm:ss");  
   
 SerializeConfig.globalInstance.put(Long.class, ToStringSerializer.instance);  
  
 fastJsonConfig.setSerializeConfig(SerializeConfig.globalInstance);  
 fastConverter.setFastJsonConfig(fastJsonConfig);  
 HttpMessageConverter<?> converter = fastConverter;  
 return converter;  
 }  
  
 @Override  
 public void configureMessageConverters(List<HttpMessageConverter<?>> converters) {  
 converters.add(fastJsonHttpMessageConverters());  
 }

### 3.4 文章详情接口

#### 3.4.1 需求

要求在文章列表点击阅读全文时能够跳转到文章详情页面，可以让用户阅读文章正文。

要求：①要在文章详情中展示其分类名

#### 3.4.2 接口设计

|  |  |
| --- | --- |
| 请求方式 | 请求路径 |
| Get | /article/{id} |

响应格式：

{  
 "code": 200,  
 "data": {  
 "categoryId": "1",  
 "categoryName": "java",  
 "content": "内容",  
 "createTime": "2022-01-23 23:20:11",  
 "id": "1",  
 "isComment": "0",  
 "title": "SpringSecurity从入门到精通",  
 "viewCount": "114"  
 },  
 "msg": "操作成功"  
}

#### 3.4.3 代码实现

ArticleController中新增

@GetMapping("/{id}")  
 public ResponseResult getArticleDetail(@PathVariable("id") Long id){  
 return articleService.getArticleDetail(id);  
 }

Service

ResponseResult getArticleDetail(Long id);

ServiceImpl

@Override  
 public ResponseResult getArticleDetail(Long id) {  
 //根据id查询文章  
 Article article = getById(id);  
 //转换成VO  
 ArticleDetailVo articleDetailVo = BeanCopyUtils.copyBean(article, ArticleDetailVo.class);  
 //根据分类id查询分类名  
 Long categoryId = articleDetailVo.getCategoryId();  
 Category category = categoryService.getById(categoryId);  
 if(category!=null){  
 articleDetailVo.setCategoryName(category.getName());  
 }  
 //封装响应返回  
 return ResponseResult.okResult(articleDetailVo);  
 }

### 3.5 友链查询

#### 3.5.0 友链表分析

通过需求去分析需要有哪些字段。

建表SQL及初始化数据见：SGBlog\资源\SQL\sg\_link.sql

#### 3.5.1 需求

在友链页面要查询出所有的审核通过的友链。

#### 3.5.2 接口设计

|  |  |
| --- | --- |
| 请求方式 | 请求路径 |
| Get | /link/getAllLink |

响应格式：

{  
 "code": 200,  
 "data": [  
 {  
 "address": "https://www.baidu.com",  
 "description": "sda",  
 "id": "1",  
 "logo": "图片url1",  
 "name": "sda"  
 },  
 {  
 "address": "https://www.qq.com",  
 "description": "dada",  
 "id": "2",  
 "logo": "图片url2",  
 "name": "sda"  
 }  
 ],  
 "msg": "操作成功"  
}

#### 3.5.3 代码实现

Controller

@RestController  
@RequestMapping("/link")  
public class LinkController {  
  
 @Autowired  
 private LinkService linkService;  
  
 @GetMapping("/getAllLink")  
 public ResponseResult getAllLink(){  
 return linkService.getAllLink();  
 }  
}

Service

public interface LinkService extends IService<Link> {  
  
 ResponseResult getAllLink();  
}

ServiceImpl

@Service("linkService")  
public class LinkServiceImpl extends ServiceImpl<LinkMapper, Link> implements LinkService {  
  
 @Override  
 public ResponseResult getAllLink() {  
 //查询所有审核通过的  
 LambdaQueryWrapper<Link> queryWrapper = new LambdaQueryWrapper<>();  
 queryWrapper.eq(Link::getStatus, SystemConstants.LINK\_STATUS\_NORMAL);  
 List<Link> links = list(queryWrapper);  
 //转换成vo  
 List<LinkVo> linkVos = BeanCopyUtils.copyBeanList(links, LinkVo.class);  
 //封装返回  
 return ResponseResult.okResult(linkVos);  
 }  
}

SystemConstants

/\*\*  
 \* 友链状态为审核通过  
 \*/  
 public static final String LINK\_STATUS\_NORMAL = "0";

### 3.6 登录功能实现

使用我们前台和后台的认证授权统一都使用SpringSecurity安全框架来实现。

#### 3.6.0 需求

需要实现登录功能

有些功能必须登录后才能使用，未登录状态是不能使用的。

#### 3.6.1 接口设计

|  |  |
| --- | --- |
| 请求方式 | 请求路径 |
| POST | /login |

请求体：

{  
 "userName":"sg",  
 "password":"1234"  
}

响应格式：

{  
 "code": 200,  
 "data": {  
 "token": "eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI0ODBmOThmYmJkNmI0NjM0OWUyZjY2NTM0NGNjZWY2NSIsInN1YiI6IjEiLCJpc3MiOiJzZyIsImlhdCI6MTY0Mzg3NDMxNiwiZXhwIjoxNjQzOTYwNzE2fQ.ldLBUvNIxQCGemkCoMgT\_0YsjsWndTg5tqfJb77pabk",  
 "userInfo": {  
 "avatar": "https://gimg2.baidu.com/image\_search/src=http%3A%2F%2Fi0.hdslb.com%2Fbfs%2Farticle%2F3bf9c263bc0f2ac5c3a7feb9e218d07475573ec8.gi",  
 "email": "23412332@qq.com",  
 "id": 1,  
 "nickName": "sg333",  
 "sex": "1"  
 }  
 },  
 "msg": "操作成功"  
}

#### 3.6.2 表分析

建表SQL及初始化数据见：SGBlog\资源\SQL\sys\_user.sql

顺便生成下User和UserMapper后面会用到

#### 3.6.3 思路分析

登录

①自定义登录接口

调用ProviderManager的方法进行认证 如果认证通过生成jwt

把用户信息存入redis中

②自定义UserDetailsService

在这个实现类中去查询数据库

注意配置passwordEncoder为BCryptPasswordEncoder

校验：

①定义Jwt认证过滤器

获取token

解析token获取其中的userid

从redis中获取用户信息

存入SecurityContextHolder

#### 3.6.4 准备工作

①添加依赖

注意放开Security依赖的注释

<!--redis依赖-->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-redis</artifactId>  
 </dependency>  
 <!--fastjson依赖-->  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>fastjson</artifactId>  
 <version>1.2.33</version>  
 </dependency>  
 <!--jwt依赖-->  
 <dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt</artifactId>  
 <version>0.9.0</version>  
 </dependency>

②工具类和相关配置类

见 ：SGBlog\资源\登录功能所需资源

#### 3.6.5 登录接口代码实现

##### BlogLoginController

@RestController  
public class BlogLoginController {  
 @Autowired  
 private BlogLoginService blogLoginService;  
  
 @PostMapping("/login")  
 public ResponseResult login(@RequestBody User user){  
 return blogLoginService.login(user);  
 }  
}

##### BlogLoginService

public interface BlogLoginService {  
 ResponseResult login(User user);  
}

##### SecurityConfig

@Configuration  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
 @Bean  
 public PasswordEncoder passwordEncoder(){  
 return new BCryptPasswordEncoder();  
 }  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 //关闭csrf  
 .csrf().disable()  
 //不通过Session获取SecurityContext  
 .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)  
 .and()  
 .authorizeRequests()  
 // 对于登录接口 允许匿名访问  
 .antMatchers("/login").anonymous()  
 // 除上面外的所有请求全部不需要认证即可访问  
 .anyRequest().permitAll();  
  
  
 http.logout().disable();  
 //允许跨域  
 http.cors();  
 }  
 @Override  
 @Bean  
 public AuthenticationManager authenticationManagerBean() throws Exception {  
 return super.authenticationManagerBean();  
 }  
}

##### BlogLoginServiceImpl

@Service  
public class BlogLoginServiceImpl implements BlogLoginService {  
  
 @Autowired  
 private AuthenticationManager authenticationManager;  
  
 @Autowired  
 private RedisCache redisCache;  
  
 @Override  
 public ResponseResult login(User user) {  
 UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(user.getUserName(),user.getPassword());  
 Authentication authenticate = authenticationManager.authenticate(authenticationToken);  
 //判断是否认证通过  
 if(Objects.isNull(authenticate)){  
 throw new RuntimeException("用户名或密码错误");  
 }  
 //获取userid 生成token  
 LoginUser loginUser = (LoginUser) authenticate.getPrincipal();  
 String userId = loginUser.getUser().getId().toString();  
 String jwt = JwtUtil.createJWT(userId);  
 //把用户信息存入redis  
 redisCache.setCacheObject("bloglogin:"+userId,loginUser);  
  
 //把token和userinfo封装 返回  
 //把User转换成UserInfoVo  
 UserInfoVo userInfoVo = BeanCopyUtils.copyBean(loginUser.getUser(), UserInfoVo.class);  
 BlogUserLoginVo vo = new BlogUserLoginVo(jwt,userInfoVo);  
 return ResponseResult.okResult(vo);  
 }  
}

##### UserDetailServiceImpl

@Service  
public class UserDetailsServiceImpl implements UserDetailsService {  
  
 @Autowired  
 private UserMapper userMapper;  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 //根据用户名查询用户信息  
 LambdaQueryWrapper<User> queryWrapper = new LambdaQueryWrapper<>();  
 queryWrapper.eq(User::getUserName,username);  
 User user = userMapper.selectOne(queryWrapper);  
 //判断是否查到用户 如果没查到抛出异常  
 if(Objects.isNull(user)){  
 throw new RuntimeException("用户不存在");  
 }  
 //返回用户信息  
 // TODO 查询权限信息封装  
 return new LoginUser(user);  
 }  
}

##### LoginUser

@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class LoginUser implements UserDetails {  
  
 private User user;  
  
  
 @Override  
 public Collection<? extends GrantedAuthority> getAuthorities() {  
 return null;  
 }  
  
 @Override  
 public String getPassword() {  
 return user.getPassword();  
 }  
  
 @Override  
 public String getUsername() {  
 return user.getUserName();  
 }  
  
 @Override  
 public boolean isAccountNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isAccountNonLocked() {  
 return true;  
 }  
  
 @Override  
 public boolean isCredentialsNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isEnabled() {  
 return true;  
 }  
}

##### BlogUserLoginVo

@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class BlogUserLoginVo {  
  
 private String token;  
 private UserInfoVo userInfo;  
}

##### UserInfoVo

@Data  
@Accessors(chain = true)  
public class UserInfoVo {  
 /\*\*  
 \* 主键  
 \*/  
 private Long id;  
  
 /\*\*  
 \* 昵称  
 \*/  
 private String nickName;  
  
 /\*\*  
 \* 头像  
 \*/  
 private String avatar;  
  
 private String sex;  
  
 private String email;  
  
  
}

#### 3.6.6 登录校验过滤器代码实现

##### 思路

①定义Jwt认证过滤器

获取token

解析token获取其中的userid

从redis中获取用户信息

存入SecurityContextHolder

##### JwtAuthenticationTokenFilter

@Component  
public class JwtAuthenticationTokenFilter extends OncePerRequestFilter {  
  
 @Autowired  
 private RedisCache redisCache;  
  
 @Override  
 protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {  
 //获取请求头中的token  
 String token = request.getHeader("token");  
 if(!StringUtils.hasText(token)){  
 //说明该接口不需要登录 直接放行  
 filterChain.doFilter(request, response);  
 return;  
 }  
 //解析获取userid  
 Claims claims = null;  
 try {  
 claims = JwtUtil.parseJWT(token);  
 } catch (Exception e) {  
 e.printStackTrace();  
 //token超时 token非法  
 //响应告诉前端需要重新登录  
 ResponseResult result = ResponseResult.errorResult(AppHttpCodeEnum.NEED\_LOGIN);  
 WebUtils.renderString(response, JSON.toJSONString(result));  
 return;  
 }  
 String userId = claims.getSubject();  
 //从redis中获取用户信息  
 LoginUser loginUser = redisCache.getCacheObject("bloglogin:" + userId);  
 //如果获取不到  
 if(Objects.isNull(loginUser)){  
 //说明登录过期 提示重新登录  
 ResponseResult result = ResponseResult.errorResult(AppHttpCodeEnum.NEED\_LOGIN);  
 WebUtils.renderString(response, JSON.toJSONString(result));  
 return;  
 }  
 //存入SecurityContextHolder  
 UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(loginUser,null,null);  
 SecurityContextHolder.getContext().setAuthentication(authenticationToken);  
  
 filterChain.doFilter(request, response);  
 }  
  
  
}

##### SecurityConfig

@Configuration  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
 @Override  
 @Bean  
 public AuthenticationManager authenticationManagerBean() throws Exception {  
 return super.authenticationManagerBean();  
 }  
  
 @Autowired  
 private JwtAuthenticationTokenFilter jwtAuthenticationTokenFilter;  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 //关闭csrf  
 .csrf().disable()  
 //不通过Session获取SecurityContext  
 .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)  
 .and()  
 .authorizeRequests()  
 // 对于登录接口 允许匿名访问  
 .antMatchers("/login").anonymous()  
 //jwt过滤器测试用，如果测试没有问题吧这里删除了  
 .antMatchers("/link/getAllLink").authenticated()  
 // 除上面外的所有请求全部不需要认证即可访问  
 .anyRequest().permitAll();  
  
  
 http.logout().disable();  
 //把jwtAuthenticationTokenFilter添加到SpringSecurity的过滤器链中  
 http.addFilterBefore(jwtAuthenticationTokenFilter, UsernamePasswordAuthenticationFilter.class);  
 //允许跨域  
 http.cors();  
 }  
  
 @Bean  
 public PasswordEncoder passwordEncoder(){  
 return new BCryptPasswordEncoder();  
 }  
}

### 3.7 认证授权失败处理

目前我们的项目在认证出错或者权限不足的时候响应回来的Json是Security的异常处理结果。但是这个响应的格式肯定是不符合我们项目的接口规范的。所以需要自定义异常处理。

AuthenticationEntryPoint 认证失败处理器

AccessDeniedHandler 授权失败处理器

@Component  
public class AuthenticationEntryPointImpl implements AuthenticationEntryPoint {  
  
 @Override  
 public void commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException authException) throws IOException, ServletException {  
 authException.printStackTrace();  
 //InsufficientAuthenticationException  
 //BadCredentialsException  
 ResponseResult result = null;  
 if(authException instanceof BadCredentialsException){  
 result = ResponseResult.errorResult(AppHttpCodeEnum.LOGIN\_ERROR.getCode(),authException.getMessage());  
 }else if(authException instanceof InsufficientAuthenticationException){  
 result = ResponseResult.errorResult(AppHttpCodeEnum.NEED\_LOGIN);  
 }else{  
 result = ResponseResult.errorResult(AppHttpCodeEnum.SYSTEM\_ERROR.getCode(),"认证或授权失败");  
 }  
 //响应给前端  
 WebUtils.renderString(response, JSON.toJSONString(result));  
 }  
}

@Component  
public class AccessDeniedHandlerImpl implements AccessDeniedHandler {  
 @Override  
 public void handle(HttpServletRequest request, HttpServletResponse response, AccessDeniedException accessDeniedException) throws IOException, ServletException {  
 accessDeniedException.printStackTrace();  
 ResponseResult result = ResponseResult.errorResult(AppHttpCodeEnum.NO\_OPERATOR\_AUTH);  
 //响应给前端  
 WebUtils.renderString(response, JSON.toJSONString(result));  
 }  
}

配置Security异常处理器

@Configuration  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
 @Override  
 @Bean  
 public AuthenticationManager authenticationManagerBean() throws Exception {  
 return super.authenticationManagerBean();  
 }  
  
 @Autowired  
 private JwtAuthenticationTokenFilter jwtAuthenticationTokenFilter;  
 @Autowired  
 AuthenticationEntryPoint authenticationEntryPoint;  
 @Autowired  
 AccessDeniedHandler accessDeniedHandler;  
  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 //关闭csrf  
 .csrf().disable()  
 //不通过Session获取SecurityContext  
 .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)  
 .and()  
 .authorizeRequests()  
 // 对于登录接口 允许匿名访问  
 .antMatchers("/login").anonymous()  
 //jwt过滤器测试用，如果测试没有问题吧这里删除了  
 .antMatchers("/link/getAllLink").authenticated()  
 // 除上面外的所有请求全部不需要认证即可访问  
 .anyRequest().permitAll();  
  
 //配置异常处理器  
 http.exceptionHandling()  
 .authenticationEntryPoint(authenticationEntryPoint)  
 .accessDeniedHandler(accessDeniedHandler);  
  
 http.logout().disable();  
 //把jwtAuthenticationTokenFilter添加到SpringSecurity的过滤器链中  
 http.addFilterBefore(jwtAuthenticationTokenFilter, UsernamePasswordAuthenticationFilter.class);  
 //允许跨域  
 http.cors();  
 }  
  
 @Bean  
 public PasswordEncoder passwordEncoder(){  
 return new BCryptPasswordEncoder();  
 }  
}

### 3.8 统一异常处理

实际我们在开发过程中可能需要做很多的判断校验，如果出现了非法情况我们是期望响应对应的提示的。但是如果我们每次都自己手动去处理就会非常麻烦。我们可以选择直接抛出异常的方式，然后对异常进行统一处理。把异常中的信息封装成ResponseResult响应给前端。

SystemException

/\*\*  
 \* @Author 三更 B站： https://space.bilibili.com/663528522  
 \*/  
public class SystemException extends RuntimeException{  
  
 private int code;  
  
 private String msg;  
  
 public int getCode() {  
 return code;  
 }  
  
 public String getMsg() {  
 return msg;  
 }  
  
 public SystemException(AppHttpCodeEnum httpCodeEnum) {  
 super(httpCodeEnum.getMsg());  
 this.code = httpCodeEnum.getCode();  
 this.msg = httpCodeEnum.getMsg();  
 }  
   
}

GlobalExceptionHandler

@RestControllerAdvice  
@Slf4j  
public class GlobalExceptionHandler {  
  
 @ExceptionHandler(SystemException.class)  
 public ResponseResult systemExceptionHandler(SystemException e){  
 //打印异常信息  
 log.error("出现了异常！ {}",e);  
 //从异常对象中获取提示信息封装返回  
 return ResponseResult.errorResult(e.getCode(),e.getMsg());  
 }  
  
  
 @ExceptionHandler(Exception.class)  
 public ResponseResult exceptionHandler(Exception e){  
 //打印异常信息  
 log.error("出现了异常！ {}",e);  
 //从异常对象中获取提示信息封装返回  
 return ResponseResult.errorResult(AppHttpCodeEnum.SYSTEM\_ERROR.getCode(),e.getMessage());  
 }  
}

### 3.9 退出登录接口

#### 3.9.1 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| POST | /logout | 需要token请求头 |

响应格式:

{  
 "code": 200,  
 "msg": "操作成功"  
}

#### 3.9.2 代码实现

要实现的操作：

删除redis中的用户信息

BlogLoginController

@PostMapping("/logout")  
 public ResponseResult logout(){  
 return blogLoginService.logout();  
 }

BlogLoginService

ResponseResult logout();

BlogLoginServiceImpl

@Override  
 public ResponseResult logout() {  
 //获取token 解析获取userid  
 Authentication authentication = SecurityContextHolder.getContext().getAuthentication();  
 LoginUser loginUser = (LoginUser) authentication.getPrincipal();  
 //获取userid  
 Long userId = loginUser.getUser().getId();  
 //删除redis中的用户信息  
 redisCache.deleteObject("bloglogin:"+userId);  
 return ResponseResult.okResult();  
 }

SecurityConfig

要关闭默认的退出登录功能。并且要配置我们的退出登录接口需要认证才能访问

@Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 //关闭csrf  
 .csrf().disable()  
 //不通过Session获取SecurityContext  
 .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)  
 .and()  
 .authorizeRequests()  
 // 对于登录接口 允许匿名访问  
 .antMatchers("/login").anonymous()  
 //注销接口需要认证才能访问  
 .antMatchers("/logout").authenticated()  
 //jwt过滤器测试用，如果测试没有问题吧这里删除了  
 .antMatchers("/link/getAllLink").authenticated()  
 // 除上面外的所有请求全部不需要认证即可访问  
 .anyRequest().permitAll();  
  
 //配置异常处理器  
 http.exceptionHandling()  
 .authenticationEntryPoint(authenticationEntryPoint)  
 .accessDeniedHandler(accessDeniedHandler);  
 //关闭默认的注销功能  
 http.logout().disable();  
 //把jwtAuthenticationTokenFilter添加到SpringSecurity的过滤器链中  
 http.addFilterBefore(jwtAuthenticationTokenFilter, UsernamePasswordAuthenticationFilter.class);  
 //允许跨域  
 http.cors();  
 }

### 3.10 查询评论列表接口

#### 3.10.1 需求

文章详情页面要展示这篇文章下的评论列表。

效果如下:

[MISSING IMAGE: , ]

#### 3.10.2 评论表分析

通过需求去分析需要有哪些字段。

建表SQL及初始化数据见：SGBlog\资源\SQL\sg\_comment.sql

**顺便生成下对应的代码**

#### 3.10.3 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| GET | /comment/commentList | 不需要token请求头 |

Query格式请求参数：

articleId:文章id

pageNum: 页码

pageSize: 每页条数

响应格式:

{  
 "code": 200,  
 "data": {  
 "rows": [  
 {  
 "articleId": "1",  
 "children": [  
 {  
 "articleId": "1",  
 "content": "你说啥？",  
 "createBy": "1",  
 "createTime": "2022-01-30 10:06:21",  
 "id": "20",  
 "rootId": "1",  
 "toCommentId": "1",  
 "toCommentUserId": "1",  
 "toCommentUserName": "sg333",  
 "username": "sg333"  
 }  
 ],  
 "content": "asS",  
 "createBy": "1",  
 "createTime": "2022-01-29 07:59:22",  
 "id": "1",  
 "rootId": "-1",  
 "toCommentId": "-1",  
 "toCommentUserId": "-1",  
 "username": "sg333"  
 }  
 ],  
 "total": "15"  
 },  
 "msg": "操作成功"  
}

#### 3.10.4 代码实现

##### 3.10.4.1 不考虑子评论

CommentController

@RestController  
@RequestMapping("/comment")  
public class CommentController {  
  
 @Autowired  
 private CommentService commentService;  
  
 @GetMapping("/commentList")  
 public ResponseResult commentList(Long articleId,Integer pageNum,Integer pageSize){  
 return commentService.commentList(articleId,pageNum,pageSize);  
 }  
}

CommentService

public interface CommentService extends IService<Comment> {  
  
 ResponseResult commentList(Long articleId, Integer pageNum, Integer pageSize);  
}

CommentServiceImpl

@Service("commentService")  
public class CommentServiceImpl extends ServiceImpl<CommentMapper, Comment> implements CommentService {  
  
 @Autowired  
 private UserService userService;  
  
 @Override  
 public ResponseResult commentList(Long articleId, Integer pageNum, Integer pageSize) {  
 //查询对应文章的根评论  
 LambdaQueryWrapper<Comment> queryWrapper = new LambdaQueryWrapper<>();  
 //对articleId进行判断  
 queryWrapper.eq(Comment::getArticleId,articleId);  
 //根评论 rootId为-1  
 queryWrapper.eq(Comment::getRootId,-1);  
  
 //分页查询  
 Page<Comment> page = new Page(pageNum,pageSize);  
 page(page,queryWrapper);  
  
 List<CommentVo> commentVoList = toCommentVoList(page.getRecords());  
  
 return ResponseResult.okResult(new PageVo(commentVoList,page.getTotal()));  
 }  
  
 private List<CommentVo> toCommentVoList(List<Comment> list){  
 List<CommentVo> commentVos = BeanCopyUtils.copyBeanList(list, CommentVo.class);  
 //遍历vo集合  
 for (CommentVo commentVo : commentVos) {  
 //通过creatyBy查询用户的昵称并赋值  
 String nickName = userService.getById(commentVo.getCreateBy()).getNickName();  
 commentVo.setUsername(nickName);  
 //通过toCommentUserId查询用户的昵称并赋值  
 //如果toCommentUserId不为-1才进行查询  
 if(commentVo.getToCommentUserId()!=-1){  
 String toCommentUserName = userService.getById(commentVo.getToCommentUserId()).getNickName();  
 commentVo.setToCommentUserName(toCommentUserName);  
 }  
 }  
 return commentVos;  
 }  
}

CommentVo

@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class CommentVo {  
 private Long id;  
 //文章id  
 private Long articleId;  
 //根评论id  
 private Long rootId;  
 //评论内容  
 private String content;  
 //所回复的目标评论的userid  
 private Long toCommentUserId;  
 private String toCommentUserName;  
 //回复目标评论id  
 private Long toCommentId;  
  
 private Long createBy;  
  
 private Date createTime;  
  
 private String username;  
}

##### 3.10.4.2 查询子评论

CommentVo在之前的基础上增加了  private List children;

@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class CommentVo {  
 private Long id;  
 //文章id  
 private Long articleId;  
 //根评论id  
 private Long rootId;  
 //评论内容  
 private String content;  
 //所回复的目标评论的userid  
 private Long toCommentUserId;  
 private String toCommentUserName;  
 //回复目标评论id  
 private Long toCommentId;  
  
 private Long createBy;  
  
 private Date createTime;  
  
 private String username;  
  
 private List<CommentVo> children;  
}

CommentServiceImpl

@Service("commentService")  
public class CommentServiceImpl extends ServiceImpl<CommentMapper, Comment> implements CommentService {  
  
 @Autowired  
 private UserService userService;  
  
 @Override  
 public ResponseResult commentList(Long articleId, Integer pageNum, Integer pageSize) {  
 //查询对应文章的根评论  
 LambdaQueryWrapper<Comment> queryWrapper = new LambdaQueryWrapper<>();  
 //对articleId进行判断  
 queryWrapper.eq(Comment::getArticleId,articleId);  
 //根评论 rootId为-1  
 queryWrapper.eq(Comment::getRootId,-1);  
  
 //分页查询  
 Page<Comment> page = new Page(pageNum,pageSize);  
 page(page,queryWrapper);  
  
 List<CommentVo> commentVoList = toCommentVoList(page.getRecords());  
  
 //查询所有根评论对应的子评论集合，并且赋值给对应的属性  
 for (CommentVo commentVo : commentVoList) {  
 //查询对应的子评论  
 List<CommentVo> children = getChildren(commentVo.getId());  
 //赋值  
 commentVo.setChildren(children);  
 }  
  
 return ResponseResult.okResult(new PageVo(commentVoList,page.getTotal()));  
 }  
  
 /\*\*  
 \* 根据根评论的id查询所对应的子评论的集合  
 \* @param id 根评论的id  
 \* @return  
 \*/  
 private List<CommentVo> getChildren(Long id) {  
  
 LambdaQueryWrapper<Comment> queryWrapper = new LambdaQueryWrapper<>();  
 queryWrapper.eq(Comment::getRootId,id);  
 queryWrapper.orderByAsc(Comment::getCreateTime);  
 List<Comment> comments = list(queryWrapper);  
  
 List<CommentVo> commentVos = toCommentVoList(comments);  
 return commentVos;  
 }  
  
 private List<CommentVo> toCommentVoList(List<Comment> list){  
 List<CommentVo> commentVos = BeanCopyUtils.copyBeanList(list, CommentVo.class);  
 //遍历vo集合  
 for (CommentVo commentVo : commentVos) {  
 //通过creatyBy查询用户的昵称并赋值  
 String nickName = userService.getById(commentVo.getCreateBy()).getNickName();  
 commentVo.setUsername(nickName);  
 //通过toCommentUserId查询用户的昵称并赋值  
 //如果toCommentUserId不为-1才进行查询  
 if(commentVo.getToCommentUserId()!=-1){  
 String toCommentUserName = userService.getById(commentVo.getToCommentUserId()).getNickName();  
 commentVo.setToCommentUserName(toCommentUserName);  
 }  
 }  
 return commentVos;  
 }  
}

### 3.11 发表评论接口

#### 3.11.1 需求

用户登录后可以对文章发表评论，也可以对评论进行回复。

用户登录后也可以在友链页面进行评论。

#### 3.11.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| POST | /comment | 需要token头 |

##### 请求体：

回复了文章：

{"articleId":1,"type":0,"rootId":-1,"toCommentId":-1,"toCommentUserId":-1,"content":"评论了文章"}

回复了某条评论：

{"articleId":1,"type":0,"rootId":"3","toCommentId":"3","toCommentUserId":"1","content":"回复了某条评论"}

如果是友链评论，type应该为1

##### 响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

#### 3.11.3 代码实现

CommentController

@PostMapping  
 public ResponseResult addComment(@RequestBody Comment comment){  
 return commentService.addComment(comment);  
 }

CommentService

ResponseResult addComment(Comment comment);

CommentServiceImpl

@Override  
 public ResponseResult addComment(Comment comment) {  
 //评论内容不能为空  
 if(!StringUtils.hasText(comment.getContent())){  
 throw new SystemException(AppHttpCodeEnum.CONTENT\_NOT\_NULL);  
 }  
 save(comment);  
 return ResponseResult.okResult();  
 }

SecurityUtils

/\*\*  
 \* @Author 三更 B站： https://space.bilibili.com/663528522  
 \*/  
public class SecurityUtils  
{  
  
 /\*\*  
 \* 获取用户  
 \*\*/  
 public static LoginUser getLoginUser()  
 {  
 return (LoginUser) getAuthentication().getPrincipal();  
 }  
  
 /\*\*  
 \* 获取Authentication  
 \*/  
 public static Authentication getAuthentication() {  
 return SecurityContextHolder.getContext().getAuthentication();  
 }  
  
 public static Boolean isAdmin(){  
 Long id = getLoginUser().getUser().getId();  
 return id != null && 1L == id;  
 }  
  
 public static Long getUserId() {  
 return getLoginUser().getUser().getId();  
 }  
}

配置MP字段自动填充

@Component  
public class MyMetaObjectHandler implements MetaObjectHandler {  
 @Override  
 public void insertFill(MetaObject metaObject) {  
 Long userId = null;  
 try {  
 userId = SecurityUtils.getUserId();  
 } catch (Exception e) {  
 e.printStackTrace();  
 userId = -1L;//表示是自己创建  
 }  
 this.setFieldValByName("createTime", new Date(), metaObject);  
 this.setFieldValByName("createBy",userId , metaObject);  
 this.setFieldValByName("updateTime", new Date(), metaObject);  
 this.setFieldValByName("updateBy", userId, metaObject);  
 }  
  
 @Override  
 public void updateFill(MetaObject metaObject) {  
 this.setFieldValByName("updateTime", new Date(), metaObject);  
 this.setFieldValByName(" ", SecurityUtils.getUserId(), metaObject);  
 }  
}

用注解标识哪些字段在什么情况下需要自动填充

/\*\*  
 \* 创建人的用户id  
 \*/  
 @TableField(fill = FieldFill.INSERT)  
 private Long createBy;  
 /\*\*  
 \* 创建时间  
 \*/  
 @TableField(fill = FieldFill.INSERT)  
 private Date createTime;  
 /\*\*  
 \* 更新人  
 \*/  
 @TableField(fill = FieldFill.INSERT\_UPDATE)  
 private Long updateBy;  
 /\*\*  
 \* 更新时间  
 \*/  
 @TableField(fill = FieldFill.INSERT\_UPDATE)  
 private Date updateTime;

### 3.12 友联评论列表

#### 3.12.1 需求

友链页面也需要查询对应的评论列表。

#### 3.12.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| GET | /comment/linkCommentList | 不需要token请求头 |

Query格式请求参数：

pageNum: 页码

pageSize: 每页条数

响应格式:

{  
 "code": 200,  
 "data": {  
 "rows": [  
 {  
 "articleId": "1",  
 "children": [  
 {  
 "articleId": "1",  
 "content": "回复友链评论3",  
 "createBy": "1",  
 "createTime": "2022-01-30 10:08:50",  
 "id": "23",  
 "rootId": "22",  
 "toCommentId": "22",  
 "toCommentUserId": "1",  
 "toCommentUserName": "sg333",  
 "username": "sg333"  
 }  
 ],  
 "content": "友链评论2",  
 "createBy": "1",  
 "createTime": "2022-01-30 10:08:28",  
 "id": "22",  
 "rootId": "-1",  
 "toCommentId": "-1",  
 "toCommentUserId": "-1",  
 "username": "sg333"  
 }  
 ],  
 "total": "1"  
 },  
 "msg": "操作成功"  
}

#### 3.12.3 代码实现

CommentController 修改了之前的文章评论列表接口，并且增加了新的友联评论接口

@GetMapping("/commentList")  
 public ResponseResult commentList(Long articleId,Integer pageNum,Integer pageSize){  
 return commentService.commentList(SystemConstants.ARTICLE\_COMMENT,articleId,pageNum,pageSize);  
 }   
 @GetMapping("/linkCommentList")  
 public ResponseResult linkCommentList(Integer pageNum,Integer pageSize){  
 return commentService.commentList(SystemConstants.LINK\_COMMENT,null,pageNum,pageSize);  
 }

SystemConstants增加了两个常量

/\*\*  
 \* 评论类型为：文章评论  
 \*/  
 public static final String ARTICLE\_COMMENT = "0";  
 /\*\*  
 \* 评论类型为：友联评论  
 \*/  
 public static final String LINK\_COMMENT = "1";

CommentService修改了commentList方法，增加了一个参数commentType

ResponseResult commentList(String commentType, Long articleId, Integer pageNum, Integer pageSize);

CommentServiceImpl修改commentList方法的代码，必须commentType为0的时候才增加articleId的判断，并且增加了一个评论类型的添加。

@Override  
 public ResponseResult commentList(String commentType, Long articleId, Integer pageNum, Integer pageSize) {  
 //查询对应文章的根评论  
 LambdaQueryWrapper<Comment> queryWrapper = new LambdaQueryWrapper<>();  
 //对articleId进行判断  
 queryWrapper.eq(SystemConstants.ARTICLE\_COMMENT.equals(commentType),Comment::getArticleId,articleId);  
 //根评论 rootId为-1  
 queryWrapper.eq(Comment::getRootId,-1);  
  
 //评论类型  
 queryWrapper.eq(Comment::getType,commentType);  
  
 //分页查询  
 Page<Comment> page = new Page(pageNum,pageSize);  
 page(page,queryWrapper);  
  
 List<CommentVo> commentVoList = toCommentVoList(page.getRecords());  
  
 //查询所有根评论对应的子评论集合，并且赋值给对应的属性  
 for (CommentVo commentVo : commentVoList) {  
 //查询对应的子评论  
 List<CommentVo> children = getChildren(commentVo.getId());  
 //赋值  
 commentVo.setChildren(children);  
 }  
  
 return ResponseResult.okResult(new PageVo(commentVoList,page.getTotal()));  
 }

### 3.13 个人信息查询接口

#### 3.13.1 需求

进入个人中心的时候需要能够查看当前用户信息

#### 3.13.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| GET | /user/userInfo | 需要token请求头 |

不需要参数

响应格式:

{  
 "code":200,  
 "data":{  
 "avatar":"https://gimg2.baidu.com/image\_search/src=http%3A%2F%2Fi0.hdslb.com%2Fbfs%2Farticle%2F3bf9c263bc0f2ac5c3a7feb9e218d07475573ec8.gi",  
 "email":"23412332@qq.com",  
 "id":"1",  
 "nickName":"sg333",  
 "sex":"1"  
 },  
 "msg":"操作成功"  
}

#### 3.13.3 代码实现

UserController

@RestController  
@RequestMapping("/user")  
public class UserController {  
  
 @Autowired  
 private UserService userService;  
  
 @GetMapping("/userInfo")  
 public ResponseResult userInfo(){  
 return userService.userInfo();  
 }  
}

UserService增加方法定义

public interface UserService extends IService<User> {  
  
 ResponseResult userInfo();  
  
}

UserServiceImpl实现userInfo方法

@Override  
 public ResponseResult userInfo() {  
 //获取当前用户id  
 Long userId = SecurityUtils.getUserId();  
 //根据用户id查询用户信息  
 User user = getById(userId);  
 //封装成UserInfoVo  
 UserInfoVo vo = BeanCopyUtils.copyBean(user,UserInfoVo.class);  
 return ResponseResult.okResult(vo);  
 }

SecurityConfig配置该接口必须认证后才能访问

@Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 //关闭csrf  
 .csrf().disable()  
 //不通过Session获取SecurityContext  
 .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)  
 .and()  
 .authorizeRequests()  
 // 对于登录接口 允许匿名访问  
 .antMatchers("/login").anonymous()  
 //注销接口需要认证才能访问  
 .antMatchers("/logout").authenticated()  
 //个人信息接口必须登录后才能访问  
 .antMatchers("/user/userInfo").authenticated()  
 // 除上面外的所有请求全部不需要认证即可访问  
 .anyRequest().permitAll();  
  
 //配置异常处理器  
 http.exceptionHandling()  
 .authenticationEntryPoint(authenticationEntryPoint)  
 .accessDeniedHandler(accessDeniedHandler);  
 //关闭默认的注销功能  
 http.logout().disable();  
 //把jwtAuthenticationTokenFilter添加到SpringSecurity的过滤器链中  
 http.addFilterBefore(jwtAuthenticationTokenFilter, UsernamePasswordAuthenticationFilter.class);  
 //允许跨域  
 http.cors();  
 }

### 3.14 头像上传接口

#### 3.14.1 需求

在个人中心点击编辑的时候可以上传头像图片。上传完头像后，可以用于更新个人信息接口。

#### 3.14.2 OSS

##### 3.14.2.1 为什么要使用OSS

因为如果把图片视频等文件上传到自己的应用的Web服务器，在读取图片的时候会占用比较多的资源。影响应用服务器的性能。

所以我们一般使用OSS(Object Storage Service对象存储服务)存储图片或视频。

##### 3.14.2.2 七牛云基本使用测试

[MISSING IMAGE: , ]

[MISSING IMAGE: , ]

秘钥

[MISSING IMAGE: , ]

[MISSING IMAGE: , ]

##### 3.14.2.3 七牛云测试代码编写

①添加依赖

<dependency>  
 <groupId>com.qiniu</groupId>  
 <artifactId>qiniu-java-sdk</artifactId>  
 <version>[7.7.0, 7.7.99]</version>  
 </dependency>

②复制修改案例代码

application.yml

oss:  
 accessKey: xxxx  
 secretKey: xxxx  
 bucket: sg-blog

OSSTest.java

@SpringBootTest  
@ConfigurationProperties(prefix = "oss")  
public class OSSTest {  
  
 private String accessKey;  
 private String secretKey;  
 private String bucket;  
  
 public void setAccessKey(String accessKey) {  
 this.accessKey = accessKey;  
 }  
  
 public void setSecretKey(String secretKey) {  
 this.secretKey = secretKey;  
 }  
  
 public void setBucket(String bucket) {  
 this.bucket = bucket;  
 }  
  
 @Test  
 public void testOss(){  
 //构造一个带指定 Region 对象的配置类  
 Configuration cfg = new Configuration(Region.autoRegion());  
 //...其他参数参考类注释  
  
 UploadManager uploadManager = new UploadManager(cfg);  
 //...生成上传凭证，然后准备上传  
// String accessKey = "your access key";  
// String secretKey = "your secret key";  
// String bucket = "sg-blog";  
  
 //默认不指定key的情况下，以文件内容的hash值作为文件名  
 String key = "2022/sg.png";  
  
 try {  
// byte[] uploadBytes = "hello qiniu cloud".getBytes("utf-8");  
// ByteArrayInputStream byteInputStream=new ByteArrayInputStream(uploadBytes);  
  
  
 InputStream inputStream = new FileInputStream("C:\\Users\\root\\Desktop\\Snipaste\_2022-02-28\_22-48-37.png");  
 Auth auth = Auth.create(accessKey, secretKey);  
 String upToken = auth.uploadToken(bucket);  
  
 try {  
 Response response = uploadManager.put(inputStream,key,upToken,null, null);  
 //解析上传成功的结果  
 DefaultPutRet putRet = new Gson().fromJson(response.bodyString(), DefaultPutRet.class);  
 System.out.println(putRet.key);  
 System.out.println(putRet.hash);  
 } catch (QiniuException ex) {  
 Response r = ex.response;  
 System.err.println(r.toString());  
 try {  
 System.err.println(r.bodyString());  
 } catch (QiniuException ex2) {  
 //ignore  
 }  
 }  
 } catch (Exception ex) {  
 //ignore  
 }  
  
 }  
}

#### 3.14.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| POST | /upload | 需要token |

参数：

img,值为要上传的文件

请求头：

Content-Type ：multipart/form-data;

响应格式:

{  
 "code": 200,  
 "data": "文件访问链接",  
 "msg": "操作成功"  
}

#### 3.14.3 代码实现

@RestController  
public class UploadController {  
 @Autowired  
 private UploadService uploadService;  
  
 @PostMapping("/upload")  
 public ResponseResult uploadImg(MultipartFile img){  
 return uploadService.uploadImg(img);  
 }  
}

public interface UploadService {  
 ResponseResult uploadImg(MultipartFile img);  
}

@Service  
@Data  
@ConfigurationProperties(prefix = "oss")  
public class OssUploadService implements UploadService {  
 @Override  
 public ResponseResult uploadImg(MultipartFile img) {  
 //判断文件类型  
 //获取原始文件名  
 String originalFilename = img.getOriginalFilename();  
 //对原始文件名进行判断  
 if(!originalFilename.endsWith(".png")){  
 throw new SystemException(AppHttpCodeEnum.FILE\_TYPE\_ERROR);  
 }  
  
 //如果判断通过上传文件到OSS  
 String filePath = PathUtils.generateFilePath(originalFilename);  
 String url = uploadOss(img,filePath);// 2099/2/3/wqeqeqe.png  
 return ResponseResult.okResult(url);  
 }  
  
 private String accessKey;  
 private String secretKey;  
 private String bucket;  
  
  
 private String uploadOss(MultipartFile imgFile, String filePath){  
 //构造一个带指定 Region 对象的配置类  
 Configuration cfg = new Configuration(Region.autoRegion());  
 //...其他参数参考类注释  
 UploadManager uploadManager = new UploadManager(cfg);  
 //默认不指定key的情况下，以文件内容的hash值作为文件名  
 String key = filePath;  
 try {  
 InputStream inputStream = imgFile.getInputStream();  
 Auth auth = Auth.create(accessKey, secretKey);  
 String upToken = auth.uploadToken(bucket);  
 try {  
 Response response = uploadManager.put(inputStream,key,upToken,null, null);  
 //解析上传成功的结果  
 DefaultPutRet putRet = new Gson().fromJson(response.bodyString(), DefaultPutRet.class);  
 System.out.println(putRet.key);  
 System.out.println(putRet.hash);  
 return "http://r7yxkqloa.bkt.clouddn.com/"+key;  
 } catch (QiniuException ex) {  
 Response r = ex.response;  
 System.err.println(r.toString());  
 try {  
 System.err.println(r.bodyString());  
 } catch (QiniuException ex2) {  
 //ignore  
 }  
 }  
 } catch (Exception ex) {  
 //ignore  
 }  
 return "www";  
 }  
}

PathUtils

/\*\*  
 \* @Author 三更 B站： https://space.bilibili.com/663528522  
 \*/  
public class PathUtils {  
  
 public static String generateFilePath(String fileName){  
 //根据日期生成路径 2022/1/15/  
 SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd/");  
 String datePath = sdf.format(new Date());  
 //uuid作为文件名  
 String uuid = UUID.randomUUID().toString().replaceAll("-", "");  
 //后缀和文件后缀一致  
 int index = fileName.lastIndexOf(".");  
 // test.jpg -> .jpg  
 String fileType = fileName.substring(index);  
 return new StringBuilder().append(datePath).append(uuid).append(fileType).toString();  
 }  
}

### 3.15 更新个人信息接口

#### 3.15.1 需求

在编辑完个人资料后点击保存会对个人资料进行更新。

#### 3.15.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| PUT | /user/userInfo | 需要token请求头 |

参数

请求体中json格式数据：

{  
 "avatar":"https://sg-blog-oss.oss-cn-beijing.aliyuncs.com/2022/01/31/948597e164614902ab1662ba8452e106.png",  
 "email":"23412332@qq.com",  
 "id":"1",  
 "nickName":"sg333",  
 "sex":"1"  
}

响应格式:

{  
 "code":200,  
 "msg":"操作成功"  
}

#### 3.15.3 代码实现

UserController

@PutMapping("/userInfo")  
 public ResponseResult updateUserInfo(@RequestBody User user){  
 return userService.updateUserInfo(user);  
 }

UserService

ResponseResult updateUserInfo(User user);

UserServiceImpl

@Override  
 public ResponseResult updateUserInfo(User user) {  
 updateById(user);  
 return ResponseResult.okResult();  
 }

### 3.16 用户注册

#### 3.16.1 需求

要求用户能够在注册界面完成用户的注册。要求用户名，昵称，邮箱不能和数据库中原有的数据重复。如果某项重复了注册失败并且要有对应的提示。并且要求用户名，密码，昵称，邮箱都不能为空。

注意:密码必须密文存储到数据库中。

#### 3.16.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| POST | /user/register | 不需要token请求头 |

参数

请求体中json格式数据：

{  
 "email": "string",  
 "nickName": "string",  
 "password": "string",  
 "userName": "string"  
}

响应格式:

{  
 "code":200,  
 "msg":"操作成功"  
}

#### 3.16.3 代码实现

UserController

@PostMapping("/register")  
 public ResponseResult register(@RequestBody User user){  
 return userService.register(user);  
 }

UserService

ResponseResult register(User user);

UserServiceImpl

@Autowired  
 private PasswordEncoder passwordEncoder;  
 @Override  
 public ResponseResult register(User user) {  
 //对数据进行非空判断  
 if(!StringUtils.hasText(user.getUserName())){  
 throw new SystemException(AppHttpCodeEnum.USERNAME\_NOT\_NULL);  
 }  
 if(!StringUtils.hasText(user.getPassword())){  
 throw new SystemException(AppHttpCodeEnum.PASSWORD\_NOT\_NULL);  
 }  
 if(!StringUtils.hasText(user.getEmail())){  
 throw new SystemException(AppHttpCodeEnum.EMAIL\_NOT\_NULL);  
 }  
 if(!StringUtils.hasText(user.getNickName())){  
 throw new SystemException(AppHttpCodeEnum.NICKNAME\_NOT\_NULL);  
 }  
 //对数据进行是否存在的判断  
 if(userNameExist(user.getUserName())){  
 throw new SystemException(AppHttpCodeEnum.USERNAME\_EXIST);  
 }  
 if(nickNameExist(user.getNickName())){  
 throw new SystemException(AppHttpCodeEnum.NICKNAME\_EXIST);  
 }  
 //...  
 //对密码进行加密  
 String encodePassword = passwordEncoder.encode(user.getPassword());  
 user.setPassword(encodePassword);  
 //存入数据库  
 save(user);  
 return ResponseResult.okResult();  
 }

public enum AppHttpCodeEnum {  
 // 成功  
 SUCCESS(200,"操作成功"),  
 // 登录  
 NEED\_LOGIN(401,"需要登录后操作"),  
 NO\_OPERATOR\_AUTH(403,"无权限操作"),  
 SYSTEM\_ERROR(500,"出现错误"),  
 USERNAME\_EXIST(501,"用户名已存在"),  
 PHONENUMBER\_EXIST(502,"手机号已存在"), EMAIL\_EXIST(503, "邮箱已存在"),  
 REQUIRE\_USERNAME(504, "必需填写用户名"),  
 CONTENT\_NOT\_NULL(506, "评论内容不能为空"),  
 FILE\_TYPE\_ERROR(507, "文件类型错误，请上传png文件"),  
 USERNAME\_NOT\_NULL(508, "用户名不能为空"),  
 NICKNAME\_NOT\_NULL(509, "昵称不能为空"),  
 PASSWORD\_NOT\_NULL(510, "密码不能为空"),  
 EMAIL\_NOT\_NULL(511, "邮箱不能为空"),  
 NICKNAME\_EXIST(512, "昵称已存在"),  
 LOGIN\_ERROR(505,"用户名或密码错误");  
 int code;  
 String msg;  
  
 AppHttpCodeEnum(int code, String errorMessage){  
 this.code = code;  
 this.msg = errorMessage;  
 }  
  
 public int getCode() {  
 return code;  
 }  
  
 public String getMsg() {  
 return msg;  
 }  
}

### 3.17 AOP实现日志记录

#### 3.17.1 需求

需要通过日志记录接口调用信息。便于后期调试排查。并且可能有很多接口都需要进行日志的记录。

接口被调用时日志打印格式如下：

[MISSING IMAGE: , ]

#### 3.17.2 思路分析

相当于是对原有的功能进行增强。并且是批量的增强，这个时候就非常适合用AOP来进行实现。

#### 3.17.3 代码实现

日志打印格式

log.info("=======Start=======");  
 // 打印请求 URL  
 log.info("URL : {}",);  
 // 打印描述信息  
 log.info("BusinessName : {}", );  
 // 打印 Http method  
 log.info("HTTP Method : {}", );  
 // 打印调用 controller 的全路径以及执行方法  
 log.info("Class Method : {}.{}", );  
 // 打印请求的 IP  
 log.info("IP : {}",);  
 // 打印请求入参  
 log.info("Request Args : {}",);  
 // 打印出参  
 log.info("Response : {}", );  
 // 结束后换行  
 log.info("=======End=======" + System.lineSeparator());

### 3.18 更新浏览次数

#### 3.18.1 需求

在用户浏览博文时要实现对应博客浏览量的增加。

#### 3.18.2 思路分析

我们只需要在每次用户浏览博客时更新对应的浏览数即可。

但是如果直接操作博客表的浏览量的话，在并发量大的情况下会出现什么问题呢？

如何去优化呢？

①在应用启动时把博客的浏览量存储到redis中

②更新浏览量时去更新redis中的数据

③每隔10分钟把Redis中的浏览量更新到数据库中

④读取文章浏览量时从redis读取

#### 3.18.3 铺垫知识

##### 3.18.3.1 CommandLineRunner实现项目启动时预处理

如果希望在SpringBoot应用启动时进行一些初始化操作可以选择使用CommandLineRunner来进行处理。

我们只需要实现CommandLineRunner接口，并且把对应的bean注入容器。把相关初始化的代码重新到需要重新的方法中。

这样就会在应用启动的时候执行对应的代码。

@Component  
public class TestRunner implements CommandLineRunner {  
 @Override  
 public void run(String... args) throws Exception {  
 System.out.println("程序初始化");  
 }  
}

##### 3.18.3.2 定时任务

定时任务的实现方式有很多，比如XXL-Job等。但是其实核心功能和概念都是类似的，很多情况下只是调用的API不同而已。

这里就先用SpringBoot为我们提供的定时任务的API来实现一个简单的定时任务，让大家先对定时任务里面的一些核心概念有个大致的了解。

实现步骤

① 使用@EnableScheduling注解开启定时任务功能

我们可以在配置类上加上[@EnableScheduling](/EnableScheduling%20)

@SpringBootApplication  
@MapperScan("com.sangeng.mapper")  
@EnableScheduling  
public class SanGengBlogApplication {  
 public static void main(String[] args) {  
 SpringApplication.run(SanGengBlogApplication.class,args);  
 }  
}

② 确定定时任务执行代码，并配置任务执行时间

使用@Scheduled注解标识需要定时执行的代码。注解的cron属性相当于是任务的执行时间。目前可以使用 0/5   ? 进行测试，代表从0秒开始，每隔5秒执行一次。

注意：对应的bean要注入容器，否则不会生效。

@Component  
public class TestJob {  
  
 @Scheduled(cron = "0/5 \* \* \* \* ?")  
 public void testJob(){  
 //要执行的代码  
 System.out.println("定时任务执行了");  
 }  
}

**3.18.3.2.1 cron 表达式语法**

cron表达式是用来设置定时任务执行时间的表达式。

很多情况下我们可以用 ： [在线Cron表达式生成器](https://www.bejson.com/othertools/cron/) 来帮助我们理解cron表达式和书写cron表达式。

但是我们还是有需要学习对应的Cron语法的，这样可以更有利于我们书写Cron表达式。

如上我们用到的 0/5   ? \*，cron表达式由七部分组成，中间由空格分隔，这七部分从左往右依次是：

秒（059），分钟（059），小时（0~23），日期（1-月最后一天），月份（1-12），星期几（1-7,1表示星期日），年份（一般该项不设置，直接忽略掉，即可为空值）

通用特殊字符：, - \* /  (可以在任意部分使用)

星号表示任意值，例如：

\* \* \* \* \* ?

表示 “ 每年每月每天每时每分每秒 ” 。

,

可以用来定义列表，例如 ：

1,2,3 \* \* \* \* ?

表示 “ 每年每月每天每时每分的每个第1秒，第2秒，第3秒 ” 。

定义范围，例如：

1-3 \* \* \* \* ?

表示 “ 每年每月每天每时每分的第1秒至第3秒 ”。

/

每隔多少，例如

5/10 \* \* \* \* ?

表示 “ 每年每月每天每时每分，从第5秒开始，每10秒一次 ” 。即 “ / ” 的左侧是开始值，右侧是间隔。如果是从 “ 0 ” 开始的话，也可以简写成 “ /10 ”

日期部分还可允许特殊字符： ? L W

星期部分还可允许的特殊字符: ? L #

?

只可用在日期和星期部分。表示没有具体的值，使用?要注意冲突。日期和星期两个部分如果其中一个部分设置了值，则另一个必须设置为 “ ? ”。

例如：

0\\* \* \* 2 \* ?  
 和  
0\\* \* \* ? \* 2

同时使用?和同时不使用?都是不对的

例如下面写法就是错的

\* \* \* 2 \* 2  
 和  
\* \* \* ? \* ?

W

只能用在日期中，表示当月中最接近某天的工作日

0 0 0 31W \* ?

表示最接近31号的工作日，如果31号是星期六，则表示30号，即星期五，如果31号是星期天，则表示29号，即星期五。如果31号是星期三，则表示31号本身，即星期三。

L

表示最后（Last）,只能用在日期和星期中

在日期中表示每月最后一天，在一月份中表示31号，在六月份中表示30号

也可以表示每月倒是第N天。例如： L-2表示每个月的倒数第2天

0 0 0 LW \* ?LW可以连起来用，表示每月最后一个工作日，即每月最后一个星期五

在星期中表示7即星期六

0 0 0 ? \* L  
表示每个星期六  
0 0 0 ? \* 6L  
若前面有其他值的话，则表示最后一个星期几，即每月的最后一个星期五

只能用在星期中，表示第几个星期几

0 0 0 ? \* 6#3  
表示每个月的第三个星期五。

#### 3.18.4 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| PUT | /article/updateViewCount/{id} | 不需要token请求头 |

参数

请求路径中携带文章id

响应格式:

{  
 "code":200,  
 "msg":"操作成功"  
}

#### 3.18.5 代码实现

##### ①在应用启动时把博客的浏览量存储到redis中

实现CommandLineRunner接口，在应用启动时初始化缓存。

@Component  
public class ViewCountRunner implements CommandLineRunner {  
  
 @Autowired  
 private ArticleMapper articleMapper;  
  
 @Autowired  
 private RedisCache redisCache;  
  
 @Override  
 public void run(String... args) throws Exception {  
 //查询博客信息 id viewCount  
 List<Article> articles = articleMapper.selectList(null);  
 Map<String, Integer> viewCountMap = articles.stream()  
 .collect(Collectors.toMap(article -> article.getId().toString(), article -> {  
 return article.getViewCount().intValue();//  
 }));  
 //存储到redis中  
 redisCache.setCacheMap("article:viewCount",viewCountMap);  
 }  
}

##### ②更新浏览量时去更新redsi中的数据

RedisCache增加方法

public void incrementCacheMapValue(String key,String hKey,long v){  
 redisTemplate.boundHashOps(key).increment(hKey, v);  
 }

ArticleController中增加方法更新阅读数

@PutMapping("/updateViewCount/{id}")  
 public ResponseResult updateViewCount(@PathVariable("id") Long id){  
 return articleService.updateViewCount(id);  
 }

ArticleService中增加方法

ResponseResult updateViewCount(Long id);

ArticleServiceImpl中实现方法

@Override  
 public ResponseResult updateViewCount(Long id) {  
 //更新redis中对应 id的浏览量  
 redisCache.incrementCacheMapValue("article:viewCount",id.toString(),1);  
 return ResponseResult.okResult();  
 }

##### ③定时任务每隔10分钟把Redis中的浏览量更新到数据库中

Article中增加构造方法

public Article(Long id, long viewCount) {  
 this.id = id;  
 this.viewCount = viewCount;  
 }

@Component  
public class UpdateViewCountJob {  
  
 @Autowired  
 private RedisCache redisCache;  
  
 @Autowired  
 private ArticleService articleService;  
  
 @Scheduled(cron = "0/5 \* \* \* \* ?")  
 public void updateViewCount(){  
 //获取redis中的浏览量  
 Map<String, Integer> viewCountMap = redisCache.getCacheMap("article:viewCount");  
  
 List<Article> articles = viewCountMap.entrySet()  
 .stream()  
 .map(entry -> new Article(Long.valueOf(entry.getKey()), entry.getValue().longValue()))  
 .collect(Collectors.toList());  
 //更新到数据库中  
 articleService.updateBatchById(articles);  
  
 }  
}

##### ④读取文章浏览量时从redis读取

@Override  
 public ResponseResult getArticleDetail(Long id) {  
 //根据id查询文章  
 Article article = getById(id);  
 //从redis中获取viewCount  
 Integer viewCount = redisCache.getCacheMapValue("article:viewCount", id.toString());  
 article.setViewCount(viewCount.longValue());  
 //转换成VO  
 ArticleDetailVo articleDetailVo = BeanCopyUtils.copyBean(article, ArticleDetailVo.class);  
 //根据分类id查询分类名  
 Long categoryId = articleDetailVo.getCategoryId();  
 Category category = categoryService.getById(categoryId);  
 if(category!=null){  
 articleDetailVo.setCategoryName(category.getName());  
 }  
 //封装响应返回  
 return ResponseResult.okResult(articleDetailVo);  
 }

## 4. Swagger2

### 4.1 简介

Swagger 是一套基于 OpenAPI 规范构建的开源工具，可以帮助我们设计、构建、记录以及使用 Rest API。

### 4.2 为什么使用Swagger

   当下很多公司都采取前后端分离的开发模式，前端和后端的工作由不同的工程师完成。在这种开发模式下，维持一份及时更新且完整的 Rest API 文档将会极大的提高我们的工作效率。传统意义上的文档都是后端开发人员手动编写的，相信大家也都知道这种方式很难保证文档的及时性，这种文档久而久之也就会失去其参考意义，反而还会加大我们的沟通成本。而 Swagger 给我们提供了一个全新的维护 API 文档的方式，下面我们就来了解一下它的优点：

1.代码变，文档变。只需要少量的注解，Swagger 就可以根据代码自动生成 API 文档，很好的保证了文档的时效性。2.跨语言性，支持 40 多种语言。3.Swagger UI 呈现出来的是一份可交互式的 API 文档，我们可以直接在文档页面尝试 API 的调用，省去了准备复杂的调用参数的过程。

### 4.3 快速入门

#### 4.3.1 引入依赖

<dependency>  
 <groupId>io.springfox</groupId>  
 <artifactId>springfox-swagger2</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>io.springfox</groupId>  
 <artifactId>springfox-swagger-ui</artifactId>  
 </dependency>

#### 4.3.2 启用Swagger2

在启动类上或者配置类加 [@EnableSwagger2](/EnableSwagger2%20)  注解

@SpringBootApplication  
@MapperScan("com.sangeng.mapper")  
@EnableScheduling  
@EnableSwagger2  
public class SanGengBlogApplication {  
 public static void main(String[] args) {  
 SpringApplication.run(SanGengBlogApplication.class,args);  
 }  
}

#### 4.3.3 测试

访问：<http://localhost:7777/swagger-ui.html>  注意其中localhost和7777要调整成实际项目的域名和端口号。

### 4.4 具体配置

#### 4.4.1 Controller配置

##### 4.4.1 [@Api](/Api%20) 注解

属性介绍：

tags  设置标签

description 设置描述信息

@RestController  
@RequestMapping("/comment")  
@Api(tags = "评论",description = "评论相关接口")  
public class CommentController {  
}

#### 4.4.2 接口配置

##### 4.4.2.1 接口描述配置[@ApiOperation](/ApiOperation%20)

@GetMapping("/linkCommentList")  
 @ApiOperation(value = "友链评论列表",notes = "获取一页友链评论")  
 public ResponseResult linkCommentList(Integer pageNum,Integer pageSize){  
 return commentService.commentList(SystemConstants.LINK\_COMMENT,null,pageNum,pageSize);  
 }

##### 4.4.2.2 接口参数描述

[@ApiImplicitParam](/ApiImplicitParam%20)  用于描述接口的参数，但是一个接口可能有多个参数，所以一般与 [@ApiImplicitParams](/ApiImplicitParams%20)  组合使用。

@GetMapping("/linkCommentList")  
 @ApiOperation(value = "友链评论列表",notes = "获取一页友链评论")  
 @ApiImplicitParams({  
 @ApiImplicitParam(name = "pageNum",value = "页号"),  
 @ApiImplicitParam(name = "pageSize",value = "每页大小")  
 }  
 )  
 public ResponseResult linkCommentList(Integer pageNum,Integer pageSize){  
 return commentService.commentList(SystemConstants.LINK\_COMMENT,null,pageNum,pageSize);  
 }

#### 4.4.3 实体类配置

##### 4.4.3.1 实体的描述配置[@ApiModel](/ApiModel%20)

@ApiModel用于描述实体类。

@Data  
@AllArgsConstructor  
@NoArgsConstructor  
@ApiModel(description = "添加评论dto")  
public class AddCommentDto{  
 //..  
}

##### 4.4.3.2 实体的属性的描述配置[@ApiModelProperty](/ApiModelProperty%20)

@ApiModelProperty用于描述实体的属性

@ApiModelProperty(notes = "评论类型（0代表文章评论，1代表友链评论）")  
 private String type;

#### 4.4.4 文档信息配置

@Configuration  
public class SwaggerConfig {  
 @Bean  
 public Docket customDocket() {  
 return new Docket(DocumentationType.SWAGGER\_2)  
 .apiInfo(apiInfo())  
 .select()  
 .apis(RequestHandlerSelectors.basePackage("com.sangeng.controller"))  
 .build();  
 }  
  
 private ApiInfo apiInfo() {  
 Contact contact = new Contact("团队名", "http://www.my.com", "my@my.com");  
 return new ApiInfoBuilder()  
 .title("文档标题")  
 .description("文档描述")  
 .contact(contact) // 联系方式  
 .version("1.1.0") // 版本  
 .build();  
 }  
}

## 5. 博客后台

### 5.0 准备工作

前端工程启动

npm install

npm run dev

①创建启动类

/\*\*  
 \* @Author 三更 B站： https://space.bilibili.com/663528522  
 \*/  
@SpringBootApplication  
@MapperScan("com.sangeng.mapper")  
public class BlogAdminApplication {  
 public static void main(String[] args) {  
 SpringApplication.run(BlogAdminApplication.class, args);  
 }  
}

②创建application.yml配置文件

server:  
 port: 8989  
spring:  
 datasource:  
 url: jdbc:mysql://localhost:3306/sg\_blog?characterEncoding=utf-8&serverTimezone=UTC  
 username: root  
 password: root  
 driver-class-name: com.mysql.cj.jdbc.Driver  
 servlet:  
 multipart:  
 max-file-size: 2MB  
 max-request-size: 5MB  
  
mybatis-plus:  
 configuration:  
 # 日志  
 log-impl: org.apache.ibatis.logging.stdout.StdOutImpl  
 global-config:  
 db-config:  
 logic-delete-field: delFlag  
 logic-delete-value: 1  
 logic-not-delete-value: 0  
 id-type: auto

③ SQL语句

SQL脚本：SGBlog\资源\SQL\sg\_tag.sql

④ 创建实体类，Mapper，Service

注意思考这些文件应该写在哪个模块下？

Tag

@SuppressWarnings("serial")  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
@TableName("sg\_tag")  
public class Tag {  
 @TableId  
 private Long id;  
  
   
 private Long createBy;  
   
 private Date createTime;  
   
 private Long updateBy;  
   
 private Date updateTime;  
 //删除标志（0代表未删除，1代表已删除）  
 private Integer delFlag;  
 //备注  
 private String remark;  
 //标签名  
 private String name;  
  
  
  
}

TagMapper

/\*\*  
 \* 标签(Tag)表数据库访问层  
 \*  
 \* @author makejava  
 \* @since 2022-07-19 22:33:35  
 \*/  
public interface TagMapper extends BaseMapper<Tag> {  
  
}

TagService

/\*\*  
 \* 标签(Tag)表服务接口  
 \*  
 \* @author makejava  
 \* @since 2022-07-19 22:33:38  
 \*/  
public interface TagService extends IService<Tag> {  
  
}

TagServiceImpl

/\*\*  
 \* 标签(Tag)表服务实现类  
 \*  
 \* @author makejava  
 \* @since 2022-07-19 22:33:38  
 \*/  
@Service("tagService")  
public class TagServiceImpl extends ServiceImpl<TagMapper, Tag> implements TagService {  
  
}

⑤ 创建Controller测试接口

注意思考这些文件应该写在哪个模块下？

TagController    /content/tag

@RestController  
@RequestMapping("/content/tag")  
public class TagController {  
 @Autowired  
 private TagService tagService;  
  
 @GetMapping("/list")  
 public ResponseResult list(){  
 return ResponseResult.okResult(tagService.list());  
 }  
}

⑥添加security相关类

@Configuration  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
 @Override  
 @Bean  
 public AuthenticationManager authenticationManagerBean() throws Exception {  
 return super.authenticationManagerBean();  
 }  
  
 @Autowired  
 private JwtAuthenticationTokenFilter jwtAuthenticationTokenFilter;  
 @Autowired  
 AuthenticationEntryPoint authenticationEntryPoint;  
 @Autowired  
 AccessDeniedHandler accessDeniedHandler;  
  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 //关闭csrf  
 .csrf().disable()  
 //不通过Session获取SecurityContext  
 .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)  
 .and()  
 .authorizeRequests()  
 // 对于登录接口 允许匿名访问  
// .antMatchers("/login").anonymous()  
// //注销接口需要认证才能访问  
// .antMatchers("/logout").authenticated()  
// .antMatchers("/user/userInfo").authenticated()  
// .antMatchers("/upload").authenticated()  
 // 除上面外的所有请求全部不需要认证即可访问  
 .anyRequest().permitAll();  
  
 //配置异常处理器  
 http.exceptionHandling()  
 .authenticationEntryPoint(authenticationEntryPoint)  
 .accessDeniedHandler(accessDeniedHandler);  
 //关闭默认的注销功能  
 http.logout().disable();  
 //把jwtAuthenticationTokenFilter添加到SpringSecurity的过滤器链中  
 http.addFilterBefore(jwtAuthenticationTokenFilter, UsernamePasswordAuthenticationFilter.class);  
 //允许跨域  
 http.cors();  
 }  
  
 @Bean  
 public PasswordEncoder passwordEncoder(){  
 return new BCryptPasswordEncoder();  
 }  
}

@Component  
public class JwtAuthenticationTokenFilter extends OncePerRequestFilter {  
  
 @Autowired  
 private RedisCache redisCache;  
  
 @Override  
 protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {  
 //获取请求头中的token  
 String token = request.getHeader("token");  
 if(!StringUtils.hasText(token)){  
 //说明该接口不需要登录 直接放行  
 filterChain.doFilter(request, response);  
 return;  
 }  
 //解析获取userid  
 Claims claims = null;  
 try {  
 claims = JwtUtil.parseJWT(token);  
 } catch (Exception e) {  
 e.printStackTrace();  
 //token超时 token非法  
 //响应告诉前端需要重新登录  
 ResponseResult result = ResponseResult.errorResult(AppHttpCodeEnum.NEED\_LOGIN);  
 WebUtils.renderString(response, JSON.toJSONString(result));  
 return;  
 }  
 String userId = claims.getSubject();  
 //从redis中获取用户信息  
 LoginUser loginUser = redisCache.getCacheObject("login:" + userId);  
 //如果获取不到  
 if(Objects.isNull(loginUser)){  
 //说明登录过期 提示重新登录  
 ResponseResult result = ResponseResult.errorResult(AppHttpCodeEnum.NEED\_LOGIN);  
 WebUtils.renderString(response, JSON.toJSONString(result));  
 return;  
 }  
 //存入SecurityContextHolder  
 UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(loginUser,null,null);  
 SecurityContextHolder.getContext().setAuthentication(authenticationToken);  
  
 filterChain.doFilter(request, response);  
 }  
  
  
}

### 5.1 后台登录

后台的认证授权也使用SpringSecurity安全框架来实现。

#### 5.1.0 需求

需要实现登录功能

后台所有功能都必须登录才能使用。

#### 5.1.1 接口设计

|  |  |
| --- | --- |
| 请求方式 | 请求路径 |
| POST | /user/login |

请求体：

{  
 "userName":"sg",  
 "password":"1234"  
}

响应格式：

{  
 "code": 200,  
 "data": {  
 "token": "eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI0ODBmOThmYmJkNmI0NjM0OWUyZjY2NTM0NGNjZWY2NSIsInN1YiI6IjEiLCJpc3MiOiJzZyIsImlhdCI6MTY0Mzg3NDMxNiwiZXhwIjoxNjQzOTYwNzE2fQ.ldLBUvNIxQCGemkCoMgT\_0YsjsWndTg5tqfJb77pabk"  
 },  
 "msg": "操作成功"  
}

#### 5.1.2 思路分析

登录

①自定义登录接口

调用ProviderManager的方法进行认证 如果认证通过生成jwt

把用户信息存入redis中

②自定义UserDetailsService

在这个实现类中去查询数据库

注意配置passwordEncoder为BCryptPasswordEncoder

校验：

①定义Jwt认证过滤器

获取token

解析token获取其中的userid

从redis中获取用户信息

存入SecurityContextHolder

#### 5.1.3 准备工作

①添加依赖

前面已经添加过相关依赖，不需要做什么处理

<!--redis依赖-->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-redis</artifactId>  
 </dependency>  
 <!--fastjson依赖-->  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>fastjson</artifactId>  
 <version>1.2.33</version>  
 </dependency>  
 <!--jwt依赖-->  
 <dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt</artifactId>  
 <version>0.9.0</version>  
 </dependency>

#### 5.1.4 登录接口代码实现

##### LoginController

复制一份BlogLoginController ，命名为LoginController，其中注入 LoginService

请求地址修改为/user/login即可

@RestController  
public class LoginController {  
 @Autowired  
 private LoginService loginService;  
  
 @PostMapping("/user/login")  
 public ResponseResult login(@RequestBody User user){  
 if(!StringUtils.hasText(user.getUserName())){  
 //提示 必须要传用户名  
 throw new SystemException(AppHttpCodeEnum.REQUIRE\_USERNAME);  
 }  
 return loginService.login(user);  
 }  
  
}

##### LoginService

复制一份BlogLoginService命名为LoginService即可

public interface LoginService {  
 ResponseResult login(User user);  
  
}

##### SecurityConfig

之前已经复制过了

##### SystemLoginServiceImpl

复制一份，LoginServiceImpl，命名为SystemLoginServiceImpl  实现 LoginService

login方法中存redis的key的前缀修改为login

返回的数据中只要返回token

@Service  
public class SystemLoginServiceImpl implements LoginService {  
  
 @Autowired  
 private AuthenticationManager authenticationManager;  
  
 @Autowired  
 private RedisCache redisCache;  
  
 @Override  
 public ResponseResult login(User user) {  
 UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(user.getUserName(),user.getPassword());  
 Authentication authenticate = authenticationManager.authenticate(authenticationToken);  
 //判断是否认证通过  
 if(Objects.isNull(authenticate)){  
 throw new RuntimeException("用户名或密码错误");  
 }  
 //获取userid 生成token  
 LoginUser loginUser = (LoginUser) authenticate.getPrincipal();  
 String userId = loginUser.getUser().getId().toString();  
 String jwt = JwtUtil.createJWT(userId);  
 //把用户信息存入redis  
 redisCache.setCacheObject("login:"+userId,loginUser);  
  
 //把token封装 返回  
 Map<String,String> map = new HashMap<>();  
 map.put("token",jwt);  
 return ResponseResult.okResult(map);  
 }  
}

##### UserDetailServiceImpl

复用原来的即可

##### LoginUser

复用原来的即可

### 5.2 后台权限控制及动态路由

#### 需求

后台系统需要能实现不同的用户权限可以看到不同的功能。

   用户只能使用他的权限所允许使用的功能。

#### 功能设计

之前在我的SpringSecurity的课程中就介绍过RBAC权限模型。没有学习过的可以去看下 [RBAC权限模型](https://www.bilibili.com/video/BV1mm4y1X7Hc?p=28)  。这里我们就是在RBAC权限模型的基础上去实现这个功能。

#### 表分析

通过需求去分析需要有哪些字段。

建表SQL及初始化数据见：SGBlog\资源\SQL\sg\_menu.sql

#### 接口设计

##### getInfo接口

是

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| GET | /getInfo | 需要token请求头 |

请求参数：

无

响应格式:

如果用户id为1代表管理员，roles 中只需要有admin，permissions中需要有所有菜单类型为C或者F的，状态为正常的，未被删除的权限

{  
 "code":200,  
 "data":{  
 "permissions":[  
 "system:user:list",  
 "system:role:list",  
 "system:menu:list",  
 "system:user:query",  
 "system:user:add"  
 //此次省略1000字  
 ],  
 "roles":[  
 "admin"  
 ],  
 "user":{  
 "avatar":"http://r7yxkqloa.bkt.clouddn.com/2022/03/05/75fd15587811443a9a9a771f24da458d.png",  
 "email":"23412332@qq.com",  
 "id":1,  
 "nickName":"sg3334",  
 "sex":"1"  
 }  
 },  
 "msg":"操作成功"  
}

##### getRouters接口

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| GET | /getRouters | 需要token请求头 |

请求参数：

无

响应格式:

前端为了实现动态路由的效果，需要后端有接口能返回用户所能访问的菜单数据。

注意：**返回的菜单数据需要体现父子菜单的层级关系**

如果用户id为1代表管理员，menus中需要有所有菜单类型为C或者M的，状态为正常的，未被删除的权限

数据格式如下：

{  
 "code":200,  
 "data":{  
 "menus":[  
 {  
 "children":[],  
 "component":"content/article/write/index",  
 "createTime":"2022-01-08 11:39:58",  
 "icon":"build",  
 "id":2023,  
 "menuName":"写博文",  
 "menuType":"C",  
 "orderNum":"0",  
 "parentId":0,  
 "path":"write",  
 "perms":"content:article:writer",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "children":[  
 {  
 "children":[],  
 "component":"system/user/index",  
 "createTime":"2021-11-12 18:46:19",  
 "icon":"user",  
 "id":100,  
 "menuName":"用户管理",  
 "menuType":"C",  
 "orderNum":"1",  
 "parentId":1,  
 "path":"user",  
 "perms":"system:user:list",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "children":[],  
 "component":"system/role/index",  
 "createTime":"2021-11-12 18:46:19",  
 "icon":"peoples",  
 "id":101,  
 "menuName":"角色管理",  
 "menuType":"C",  
 "orderNum":"2",  
 "parentId":1,  
 "path":"role",  
 "perms":"system:role:list",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "children":[],  
 "component":"system/menu/index",  
 "createTime":"2021-11-12 18:46:19",  
 "icon":"tree-table",  
 "id":102,  
 "menuName":"菜单管理",  
 "menuType":"C",  
 "orderNum":"3",  
 "parentId":1,  
 "path":"menu",  
 "perms":"system:menu:list",  
 "status":"0",  
 "visible":"0"  
 }  
 ],  
 "createTime":"2021-11-12 18:46:19",  
 "icon":"system",  
 "id":1,  
 "menuName":"系统管理",  
 "menuType":"M",  
 "orderNum":"1",  
 "parentId":0,  
 "path":"system",  
 "perms":"",  
 "status":"0",  
 "visible":"0"  
 }  
 ]  
 },  
 "msg":"操作成功"  
}

#### 代码实现

##### 准备工作

生成menu和role表对于的类

##### getInfo接口

@Data  
@Accessors(chain = true)  
@AllArgsConstructor  
@NoArgsConstructor  
public class AdminUserInfoVo {  
  
 private List<String> permissions;  
  
 private List<String> roles;  
  
 private UserInfoVo user;  
}

@RestController  
public class LoginController {  
 @Autowired  
 private LoginService loginService;  
  
 @Autowired  
 private MenuService menuService;  
  
 @Autowired  
 private RoleService roleService;  
  
 @PostMapping("/user/login")  
 public ResponseResult login(@RequestBody User user){  
 if(!StringUtils.hasText(user.getUserName())){  
 //提示 必须要传用户名  
 throw new SystemException(AppHttpCodeEnum.REQUIRE\_USERNAME);  
 }  
 return loginService.login(user);  
 }  
  
 @GetMapping("getInfo")  
 public ResponseResult<AdminUserInfoVo> getInfo(){  
 //获取当前登录的用户  
 LoginUser loginUser = SecurityUtils.getLoginUser();  
 //根据用户id查询权限信息  
 List<String> perms = menuService.selectPermsByUserId(loginUser.getUser().getId());  
 //根据用户id查询角色信息  
 List<String> roleKeyList = roleService.selectRoleKeyByUserId(loginUser.getUser().getId());  
  
 //获取用户信息  
 User user = loginUser.getUser();  
 UserInfoVo userInfoVo = BeanCopyUtils.copyBean(user, UserInfoVo.class);  
 //封装数据返回  
  
 AdminUserInfoVo adminUserInfoVo = new AdminUserInfoVo(perms,roleKeyList,userInfoVo);  
 return ResponseResult.okResult(adminUserInfoVo);  
 }  
  
}

RoleServiceImpl  selectRoleKeyByUserId方法

@Service("menuService")  
public class MenuServiceImpl extends ServiceImpl<MenuMapper, Menu> implements MenuService {  
  
 @Override  
 public List<String> selectPermsByUserId(Long id) {  
 //如果是管理员，返回所有的权限  
 if(id == 1L){  
 LambdaQueryWrapper<Menu> wrapper = new LambdaQueryWrapper<>();  
 wrapper.in(Menu::getMenuType,SystemConstants.MENU,SystemConstants.BUTTON);  
 wrapper.eq(Menu::getStatus,SystemConstants.STATUS\_NORMAL);  
 List<Menu> menus = list(wrapper);  
 List<String> perms = menus.stream()  
 .map(Menu::getPerms)  
 .collect(Collectors.toList());  
 return perms;  
 }  
 //否则返回所具有的权限  
 return getBaseMapper().selectPermsByUserId(id);  
 }  
}

MenuMapper

/\*\*  
 \* 菜单权限表(Menu)表数据库访问层  
 \*  
 \* @author makejava  
 \* @since 2022-08-09 22:32:07  
 \*/  
public interface MenuMapper extends BaseMapper<Menu> {  
  
 List<String> selectPermsByUserId(Long userId);  
}

<?xml version="1.0" encoding="UTF-8" ?>  
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >  
<mapper namespace="com.sangeng.mapper.MenuMapper">  
  
 <select id="selectPermsByUserId" resultType="java.lang.String">  
 SELECT  
 DISTINCT m.perms  
 FROM  
 `sys\_user\_role` ur  
 LEFT JOIN `sys\_role\_menu` rm ON ur.`role\_id` = rm.`role\_id`  
 LEFT JOIN `sys\_menu` m ON m.`id` = rm.`menu\_id`  
 WHERE  
 ur.`user\_id` = #{userId} AND  
 m.`menu\_type` IN ('C','F') AND  
 m.`status` = 0 AND  
 m.`del\_flag` = 0  
 </select>  
</mapper>

MenuServiceImpl  selectPermsByUserId方法

@Service("roleService")  
public class RoleServiceImpl extends ServiceImpl<RoleMapper, Role> implements RoleService {  
  
 @Override  
 public List<String> selectRoleKeyByUserId(Long id) {  
 //判断是否是管理员 如果是返回集合中只需要有admin  
 if(id == 1L){  
 List<String> roleKeys = new ArrayList<>();  
 roleKeys.add("admin");  
 return roleKeys;  
 }  
 //否则查询用户所具有的角色信息  
 return getBaseMapper().selectRoleKeyByUserId(id);  
 }  
}

public interface RoleMapper extends BaseMapper<Role> {  
  
 List<String> selectRoleKeyByUserId(Long userId);  
}

<?xml version="1.0" encoding="UTF-8" ?>  
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >  
<mapper namespace="com.sangeng.mapper.RoleMapper">  
 <select id="selectRoleKeyByUserId" resultType="java.lang.String">  
 SELECT  
 r.`role\_key`  
 FROM  
 `sys\_user\_role` ur  
 LEFT JOIN `sys\_role` r ON ur.`role\_id` = r.`id`  
 WHERE  
 ur.`user\_id` = #{userId} AND  
 r.`status` = 0 AND  
 r.`del\_flag` = 0  
 </select>  
</mapper>

##### getRouters接口

RoutersVo

@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class RoutersVo {  
  
 private List<Menu> menus;  
}

LoginController

@GetMapping("getRouters")  
 public ResponseResult<RoutersVo> getRouters(){  
 Long userId = SecurityUtils.getUserId();  
 //查询menu 结果是tree的形式  
 List<Menu> menus = menuService.selectRouterMenuTreeByUserId(userId);  
 //封装数据返回  
 return ResponseResult.okResult(new RoutersVo(menus));  
 }

MenuService

public interface MenuService extends IService<Menu> {  
  
 List<String> selectPermsByUserId(Long id);  
  
 List<Menu> selectRouterMenuTreeByUserId(Long userId);  
}

MenuServiceImpl

@Override  
 public List<Menu> selectRouterMenuTreeByUserId(Long userId) {  
 MenuMapper menuMapper = getBaseMapper();  
 List<Menu> menus = null;  
 //判断是否是管理员  
 if(SecurityUtils.isAdmin()){  
 //如果是 获取所有符合要求的Menu  
 menus = menuMapper.selectAllRouterMenu();  
 }else{  
 //否则 获取当前用户所具有的Menu  
 menus = menuMapper.selectRouterMenuTreeByUserId(userId);  
 }  
  
 //构建tree  
 //先找出第一层的菜单 然后去找他们的子菜单设置到children属性中  
 List<Menu> menuTree = builderMenuTree(menus,0L);  
 return menuTree;  
 }  
  
 private List<Menu> builderMenuTree(List<Menu> menus, Long parentId) {  
 List<Menu> menuTree = menus.stream()  
 .filter(menu -> menu.getParentId().equals(parentId))  
 .map(menu -> menu.setChildren(getChildren(menu, menus)))  
 .collect(Collectors.toList());  
 return menuTree;  
 }  
  
 /\*\*  
 \* 获取存入参数的 子Menu集合  
 \* @param menu  
 \* @param menus  
 \* @return  
 \*/  
 private List<Menu> getChildren(Menu menu, List<Menu> menus) {  
 List<Menu> childrenList = menus.stream()  
 .filter(m -> m.getParentId().equals(menu.getId()))  
 .map(m->m.setChildren(getChildren(m,menus)))  
 .collect(Collectors.toList());  
 return childrenList;  
 }

MenuMapper.java

List<Menu> selectAllRouterMenu();  
  
 List<Menu> selectRouterMenuTreeByUserId(Long userId);

MenuMapper.xml

<select id="selectAllRouterMenu" resultType="com.sangeng.domain.entity.Menu">  
 SELECT  
 DISTINCT m.id, m.parent\_id, m.menu\_name, m.path, m.component, m.visible, m.status, IFNULL(m.perms,'') AS perms, m.is\_frame, m.menu\_type, m.icon, m.order\_num, m.create\_time  
 FROM  
 `sys\_menu` m  
 WHERE  
 m.`menu\_type` IN ('C','M') AND  
 m.`status` = 0 AND  
 m.`del\_flag` = 0  
 ORDER BY  
 m.parent\_id,m.order\_num  
 </select>  
 <select id="selectRouterMenuTreeByUserId" resultType="com.sangeng.domain.entity.Menu">  
 SELECT  
 DISTINCT m.id, m.parent\_id, m.menu\_name, m.path, m.component, m.visible, m.status, IFNULL(m.perms,'') AS perms, m.is\_frame, m.menu\_type, m.icon, m.order\_num, m.create\_time  
 FROM  
 `sys\_user\_role` ur  
 LEFT JOIN `sys\_role\_menu` rm ON ur.`role\_id` = rm.`role\_id`  
 LEFT JOIN `sys\_menu` m ON m.`id` = rm.`menu\_id`  
 WHERE  
 ur.`user\_id` = #{userId} AND  
 m.`menu\_type` IN ('C','M') AND  
 m.`status` = 0 AND  
 m.`del\_flag` = 0  
 ORDER BY  
 m.parent\_id,m.order\_num  
 </select>

查询的列：

SELECT DISTINCT m.id, m.parent\_id, m.menu\_name, m.path, m.component, m.visible, m.status, IFNULL(m.perms,'') AS perms, m.is\_frame,  m.menu\_type, m.icon, m.order\_num, m.create\_time

注意需要按照parent\_id和order\_num排序

### 5.3 退出登录接口

#### 5.3.1 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| POST | /user/logout | 需要token请求头 |

响应格式:

{  
 "code": 200,  
 "msg": "操作成功"  
}

#### 5.3.2 代码实现

要实现的操作：

删除redis中的用户信息

LoginController

@PostMapping("/user/logout")  
 public ResponseResult logout(){  
 return loginServcie.logout();  
 }

LoginService

ResponseResult logout();

SystemLoginServiceImpl

@Override  
 public ResponseResult logout() {  
 //获取当前登录的用户id  
 Long userId = SecurityUtils.getUserId();  
 //删除redis中对应的值  
 redisCache.deleteObject("login:"+userId);  
 return ResponseResult.okResult();  
 }

SecurityConfig

要关闭默认的退出登录功能。并且要配置我们的退出登录接口需要认证才能访问

@Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 //关闭csrf  
 .csrf().disable()  
 //不通过Session获取SecurityContext  
 .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)  
 .and()  
 .authorizeRequests()  
 // 对于登录接口 允许匿名访问  
 .antMatchers("/user/login").anonymous()  
// //注销接口需要认证才能访问  
// .antMatchers("/logout").authenticated()  
// .antMatchers("/user/userInfo").authenticated()  
// .antMatchers("/upload").authenticated()  
 // 除上面外的所有请求全部不需要认证即可访问  
 .anyRequest().authenticated();  
  
 //配置异常处理器  
 http.exceptionHandling()  
 .authenticationEntryPoint(authenticationEntryPoint)  
 .accessDeniedHandler(accessDeniedHandler);  
 //关闭默认的注销功能  
 http.logout().disable();  
 //把jwtAuthenticationTokenFilter添加到SpringSecurity的过滤器链中  
 http.addFilterBefore(jwtAuthenticationTokenFilter, UsernamePasswordAuthenticationFilter.class);  
 //允许跨域  
 http.cors();  
 }

### 5.4 查询标签列表

#### 5.4.0 需求

为了方便后期对文章进行管理，需要提供标签的功能，一个文章可以有多个标签。

   在后台需要分页查询标签功能，要求能根据标签名进行分页查询。 **后期可能会增加备注查询等需求**。

注意：不能把删除了的标签查询出来。

#### 5.4.1 标签表分析

通过需求去分析需要有哪些字段。

#### 5.4.2 接口设计

|  |  |
| --- | --- |
| 请求方式 | 请求路径 |
| Get | content/tag/list |

Query格式请求参数：

pageNum: 页码

pageSize: 每页条数

name：标签名

remark：备注

响应格式：

{  
 "code":200,  
 "data":{  
 "rows":[  
 {  
 "id":4,  
 "name":"Java",  
 "remark":"sdad"  
 }  
 ],  
 "total":1  
 },  
 "msg":"操作成功"  
}

#### 5.4.3 代码实现

Controller

@RestController  
@RequestMapping("/content/tag")  
public class TagController {  
 @Autowired  
 private TagService tagService;  
  
 @GetMapping("/list")  
 public ResponseResult<PageVo> list(Integer pageNum, Integer pageSize, TagListDto tagListDto){  
 return tagService.pageTagList(pageNum,pageSize,tagListDto);  
 }  
}

Service

public interface TagService extends IService<Tag> {  
  
 ResponseResult<PageVo> pageTagList(Integer pageNum, Integer pageSize, TagListDto tagListDto);  
}

@Service("tagService")  
public class TagServiceImpl extends ServiceImpl<TagMapper, Tag> implements TagService {  
  
 @Override  
 public ResponseResult<PageVo> pageTagList(Integer pageNum, Integer pageSize, TagListDto tagListDto) {  
 //分页查询  
 LambdaQueryWrapper<Tag> queryWrapper = new LambdaQueryWrapper<>();  
 queryWrapper.eq(StringUtils.hasText(tagListDto.getName()),Tag::getName,tagListDto.getName());  
 queryWrapper.eq(StringUtils.hasText(tagListDto.getRemark()),Tag::getRemark,tagListDto.getRemark());  
  
 Page<Tag> page = new Page<>();  
 page.setCurrent(pageNum);  
 page.setSize(pageSize);  
 page(page, queryWrapper);  
 //封装数据返回  
 PageVo pageVo = new PageVo(page.getRecords(),page.getTotal());  
 return ResponseResult.okResult(pageVo);  
 }  
}

### 5.5 新增标签

#### 5.5.0 需求

点击标签管理的新增按钮可以实现新增标签的功能。

#### 5.5.1 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| POST | /content/tag | 需要token请求头 |

请求体格式:

{"name":"c#","remark":"c++++"}

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

#### 5.5.2 测试

测试时注意，添加到数据库中的记录有没有 创建时间，更新时间，创建人，更新人字段。

### 5.6 删除标签

#### 5.6.1 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| DELETE | /content/tag/{id} | 需要token请求头 |

请求参数在path中

例如：content/tag/6   代表删除id为6的标签数据

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

#### 5.6.2 测试

注意测试删除后在列表中是否查看不到该条数据

数据库中该条数据还是存在的，只是修改了逻辑删除字段的值

### 5.7 修改标签

#### 5.7.1 接口设计

##### 5.7.1.1 获取标签信息

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| GET | /content/tag/{id} | 需要token请求头 |

请求参数在path中

例如：content/tag/6   代表获取id为6的标签数据

响应格式：

{  
 "code":200,  
 "data":{  
 "id":4,  
 "name":"Java",  
 "remark":"sdad"  
 },  
 "msg":"操作成功"  
}

##### 5.7.1.2 修改标签接口

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| PUT | /content/tag | 需要token请求头 |

请求体格式：

{"id":7,"name":"c#","remark":"c++++"}

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

### 5.8 写博文

#### 5.8.1 需求

需要提供写博文的功能，写博文时需要关联分类和标签。

可以上传缩略图，也可以在正文中添加图片。

文章可以直接发布，也可以保存到草稿箱。

#### 5.8.2 表分析

标签和文章需要关联所以需要一张关联表。

SQL脚本：SGBlog\资源\SQL\sg\_article\_tag.sql

#### 5.8.2 接口设计

思考下需要哪些接口才能实现这个功能？

##### 5.8.2.1 查询所有分类接口

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| GET | /content/category/listAllCategory | 需要token请求头 |

请求参数：

无

响应格式：

{  
 "code":200,  
 "data":[  
 {  
 "description":"wsd",  
 "id":1,  
 "name":"java"  
 },  
 {  
 "description":"wsd",  
 "id":2,  
 "name":"PHP"  
 }  
 ],  
 "msg":"操作成功"  
}

##### 5.8.2.2 查询所有标签接口

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| GET | /content/tag/listAllTag | 需要token请求头 |

请求参数:

无

响应格式：

{  
 "code":200,  
 "data":[  
 {  
 "id":1,  
 "name":"Mybatis"  
 },  
 {  
 "id":4,  
 "name":"Java"  
 }  
 ],  
 "msg":"操作成功"  
}

##### 5.8.2.3 上传图片

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| POST | /upload | 需要token请求头 |

参数：

img,值为要上传的文件

请求头：

Content-Type ：multipart/form-data;

响应格式:

{  
 "code": 200,  
 "data": "文件访问链接",  
 "msg": "操作成功"  
}

##### 5.8.2.4 新增博文

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| POST | /content/article | 需要token请求头 |

请求体格式：

{  
 "title":"测试新增博文",  
 "thumbnail":"https://sg-blog-oss.oss-cn-beijing.aliyuncs.com/2022/08/21/4ceebc07e7484beba732f12b0d2c43a9.png",  
 "isTop":"0",  
 "isComment":"0",  
 "content":"# 一级标题\n## 二级标题\n![Snipaste\_20220228\_224837.png](https://sg-blog-oss.oss-cn-beijing.aliyuncs.com/2022/08/21/c3af554d4a0f4935b4073533a4c26ee8.png)\n正文",  
 "tags":[  
 1,  
 4  
 ],  
 "categoryId":1,  
 "summary":"哈哈",  
 "status":"1"  
}

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

#### 5.8.3 代码实现

##### 5.8.3.1 查询所有分类接口

CategoryController

/\*\*  
 \* @Author 三更 B站： https://space.bilibili.com/663528522  
 \*/  
@RestController  
@RequestMapping("/content/category")  
public class CategoryController {  
 @Autowired  
 private CategoryService categoryService;  
   
 @GetMapping("/listAllCategory")  
 public ResponseResult listAllCategory(){  
 List<CategoryVo> list = categoryService.listAllCategory();  
 return ResponseResult.okResult(list);  
 }  
  
   
}

CategoryVo修改,增加description属性

@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class CategoryVo {  
  
 private Long id;  
 private String name;  
 //描述  
 private String description;  
}

CategoryService增加listAllCategory方法

public interface CategoryService extends IService<Category> {  
  
  
 ResponseResult getCategoryList();  
  
 List<CategoryVo> listAllCategory();  
}

SystemConstants中增加常量

/\*\* 正常状态 \*/  
 public static final String NORMAL = "0";

CategoryServiceImpl增加方法

@Override  
 public List<CategoryVo> listAllCategory() {  
 LambdaQueryWrapper<Category> wrapper = new LambdaQueryWrapper<>();  
 wrapper.eq(Category::getStatus, SystemConstants.NORMAL);  
 List<Category> list = list(wrapper);  
 List<CategoryVo> categoryVos = BeanCopyUtils.copyBeanList(list, CategoryVo.class);  
 return categoryVos;  
 }

##### 5.8.3.2 查询所有标签接口

TagVo

@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class TagVo {  
 private Long id;  
  
 //标签名  
 private String name;  
  
  
  
}

TagController

@GetMapping("/listAllTag")  
 public ResponseResult listAllTag(){  
 List<TagVo> list = tagService.listAllTag();  
 return ResponseResult.okResult(list);  
 }

TagService 增加listAllTag方法

List<TagVo> listAllTag();

TagServiceImpl

@Override  
 public List<TagVo> listAllTag() {  
 LambdaQueryWrapper<Tag> wrapper = new LambdaQueryWrapper<>();  
 wrapper.select(Tag::getId,Tag::getName);  
 List<Tag> list = list(wrapper);  
 List<TagVo> tagVos = BeanCopyUtils.copyBeanList(list, TagVo.class);  
 return tagVos;  
 }

##### 5.8.3.3 上传图片接口

在sangeng-admin中增加UploadController

/\*\*  
 \* @Author 三更 B站： https://space.bilibili.com/663528522  
 \*/  
@RestController  
public class UploadController {  
  
 @Autowired  
 private UploadService uploadService;  
  
 @PostMapping("/upload")  
 public ResponseResult uploadImg(@RequestParam("img") MultipartFile multipartFile) {  
 try {  
 return uploadService.uploadImg(multipartFile);  
 } catch (IOException e) {  
 e.printStackTrace();  
 throw new RuntimeException("文件上传上传失败");  
 }  
 }  
}

##### 5.8.3.4 新增博文接口

ArticleController

/\*\*  
 \* @Author 三更 B站： https://space.bilibili.com/663528522  
 \*/  
@RestController  
@RequestMapping("/content/article")  
public class ArticleController {  
  
 @Autowired  
 private ArticleService articleService;  
  
 @PostMapping  
 public ResponseResult add(@RequestBody AddArticleDto article){  
 return articleService.add(article);  
 }  
  
  
}

AddArticleDto

注意增加tags属性用于接收文章关联标签的id

@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class AddArticleDto {  
  
 private Long id;  
 //标题  
 private String title;  
 //文章内容  
 private String content;  
 //文章摘要  
 private String summary;  
 //所属分类id  
 private Long categoryId;  
  
 //缩略图  
 private String thumbnail;  
 //是否置顶（0否，1是）  
 private String isTop;  
 //状态（0已发布，1草稿）  
 private String status;  
 //访问量  
 private Long viewCount;  
 //是否允许评论 1是，0否  
 private String isComment;  
 private List<Long> tags;  
  
}

Article 修改这样创建时间创建人修改时间修改人可以自动填充

@TableField(fill = FieldFill.INSERT)  
 private Long createBy;  
 @TableField(fill = FieldFill.INSERT)  
 private Date createTime;  
 @TableField(fill = FieldFill.INSERT\_UPDATE)  
 private Long updateBy;  
 @TableField(fill = FieldFill.INSERT\_UPDATE)  
 private Date updateTime;

ArticleService增加方法

ResponseResult add(AddArticleDto article);

创建ArticleTag表相关的实体类，mapper，service,serviceimpl等

@TableName(value="sg\_article\_tag")  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class ArticleTag implements Serializable {  
 private static final long serialVersionUID = 625337492348897098L;  
   
 /\*\*  
 \* 文章id  
 \*/  
 private Long articleId;  
 /\*\*  
 \* 标签id  
 \*/  
 private Long tagId;  
  
  
  
}

ArticleServiceImpl增加如下代码

@Autowired  
 private ArticleTagService articleTagService;  
  
 @Override  
 @Transactional  
 public ResponseResult add(AddArticleDto articleDto) {  
 //添加 博客  
 Article article = BeanCopyUtils.copyBean(articleDto, Article.class);  
 save(article);  
  
  
 List<ArticleTag> articleTags = articleDto.getTags().stream()  
 .map(tagId -> new ArticleTag(article.getId(), tagId))  
 .collect(Collectors.toList());  
  
 //添加 博客和标签的关联  
 articleTagService.saveBatch(articleTags);  
 return ResponseResult.okResult();  
 }

### 5.9 导出所有分类到Excel

#### 5.9.1 需求

在分类管理中点击导出按钮可以把所有的分类导出到Excel文件中。

#### 5.9.2 技术方案

使用EasyExcel实现Excel的导出操作。

<https://github.com/alibaba/easyexcel>

[https://easyexcel.opensource.alibaba.com/docs/current/quickstart/write#示例代码-1](https://easyexcel.opensource.alibaba.com/docs/current/quickstart/write#%E7%A4%BA%E4%BE%8B%E4%BB%A3%E7%A0%81-1)

#### 5.9.3 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求地址 | 请求头 |
| GET | /content/category/export | 需要token请求头 |

请求参数：

无

响应格式：

成功的话可以直接导出一个Excel文件

失败的话响应格式如下：

{  
 "code":500,  
 "msg":"出现错误"  
}

#### 5.9.4 代码实现

工具类方法修改

WebUtils

public static void setDownLoadHeader(String filename, HttpServletResponse response) throws UnsupportedEncodingException {  
 response.setContentType("application/vnd.openxmlformats-officedocument.spreadsheetml.sheet");  
 response.setCharacterEncoding("utf-8");  
 String fname= URLEncoder.encode(filename,"UTF-8").replaceAll("\\+", "%20");  
 response.setHeader("Content-disposition","attachment; filename="+fname);  
 }

CategoryController

@GetMapping("/export")  
 public void export(HttpServletResponse response){  
 try {  
 //设置下载文件的请求头  
 WebUtils.setDownLoadHeader("分类.xlsx",response);  
 //获取需要导出的数据  
 List<Category> categoryVos = categoryService.list();  
  
 List<ExcelCategoryVo> excelCategoryVos = BeanCopyUtils.copyBeanList(categoryVos, ExcelCategoryVo.class);  
 //把数据写入到Excel中  
 EasyExcel.write(response.getOutputStream(), ExcelCategoryVo.class).autoCloseStream(Boolean.FALSE).sheet("分类导出")  
 .doWrite(excelCategoryVos);  
  
 } catch (Exception e) {  
 //如果出现异常也要响应json  
 ResponseResult result = ResponseResult.errorResult(AppHttpCodeEnum.SYSTEM\_ERROR);  
 WebUtils.renderString(response, JSON.toJSONString(result));  
 }  
 }

ExcelCategoryVo

@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class ExcelCategoryVo {  
 @ExcelProperty("分类名")  
 private String name;  
 //描述  
 @ExcelProperty("描述")  
 private String description;  
  
 //状态0:正常,1禁用  
 @ExcelProperty("状态0:正常,1禁用")  
 private String status;  
}

### 5.10 权限控制

#### 5.10.1 需求

需要对导出分类的接口做权限控制。

sg   eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiJkZGJkNjM5MWJiZTA0NmMzOTc4NDg1ZTcxNWQ3YjQ0MSIsInN1YiI6IjEiLCJpc3MiOiJzZyIsImlhdCI6MTY2MjI0NDE4NywiZXhwIjoxNjYyMzMwNTg3fQ.z4JGwFN3lWyVbOCbhikCe-O4D6SvCQFEE5eQY3jDJkw

sangeng

eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI0Y2I1ZjhmMTc0Mjk0NzM0YjI4Y2M1NTQzYjQ2Yjc1YyIsInN1YiI6IjYiLCJpc3MiOiJzZyIsImlhdCI6MTY2MjI0NDQzMywiZXhwIjoxNjYyMzMwODMzfQ.yEkbyGYXBp5ndnyq-3acdgpvqx2mnI8B9fK9f3Y6Jco

#### 5.10.2 代码实现

SecurityConfig

@EnableGlobalMethodSecurity(prePostEnabled = true)

UserDetailsServiceImpl

@Service  
public class UserDetailsServiceImpl implements UserDetailsService {  
  
 @Autowired  
 private UserMapper userMapper;  
  
 @Autowired  
 private MenuMapper menuMapper;  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 //根据用户名查询用户信息  
 LambdaQueryWrapper<User> queryWrapper = new LambdaQueryWrapper<>();  
 queryWrapper.eq(User::getUserName,username);  
 User user = userMapper.selectOne(queryWrapper);  
 //判断是否查到用户 如果没查到抛出异常  
 if(Objects.isNull(user)){  
 throw new RuntimeException("用户不存在");  
 }  
 //返回用户信息  
 if(user.getType().equals(SystemConstants.ADMAIN)){  
 List<String> list = menuMapper.selectPermsByUserId(user.getId());  
 return new LoginUser(user,list);  
 }  
 return new LoginUser(user,null);  
 }  
}

LoginUser

增加属性

private List<String> permissions;

PermissionService

hasPermisson

@Service("ps")  
public class PermissionService {  
  
 /\*\*  
 \* 判断当前用户是否具有permission  
 \* @param permission 要判断的权限  
 \* @return  
 \*/  
 public boolean hasPermission(String permission){  
 //如果是超级管理员 直接返回true  
 if(SecurityUtils.isAdmin()){  
 return true;  
 }  
 //否则 获取当前登录用户所具有的权限列表 如何判断是否存在permission  
 List<String> permissions = SecurityUtils.getLoginUser().getPermissions();  
 return permissions.contains(permission);  
 }  
}

CategoryController

@PreAuthorize("@ps.hasPermission('content:category:export')")  
 @GetMapping("/export")  
 public void export(HttpServletResponse response){  
 try {  
 //设置下载文件的请求头  
 WebUtils.setDownLoadHeader("分类.xlsx",response);  
 //获取需要导出的数据  
 List<Category> categoryVos = categoryService.list();  
  
 List<ExcelCategoryVo> excelCategoryVos = BeanCopyUtils.copyBeanList(categoryVos, ExcelCategoryVo.class);  
 //把数据写入到Excel中  
 EasyExcel.write(response.getOutputStream(), ExcelCategoryVo.class).autoCloseStream(Boolean.FALSE).sheet("分类导出")  
 .doWrite(excelCategoryVos);  
  
 } catch (Exception e) {  
 //如果出现异常也要响应json  
 ResponseResult result = ResponseResult.errorResult(AppHttpCodeEnum.SYSTEM\_ERROR);  
 WebUtils.renderString(response, JSON.toJSONString(result));  
 }  
 }

### 5.11 文章列表

#### 5.10.1 需求

为了对文章进行管理，需要提供文章列表，

   在后台需要分页查询文章功能，要求能根据标题和摘要**模糊查询**。

注意：不能把删除了的文章查询出来

#### 5.10.2  接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| Get | /content/article/list | 是 |

Query格式请求参数：

pageNum: 页码

pageSize: 每页条数

title：文章标题

summary：文章摘要

响应格式：

{  
 "code":200,  
 "data":{  
 "rows":[  
 {  
 "categoryId":"1",  
 "content":"嘻嘻嘻嘻嘻嘻",  
 "createTime":"2022-01-24 07:20:11",  
 "delFlag":0,  
 "id":"1",  
 "isComment":"0",  
 "isTop":"1",  
 "status":"0",  
 "summary":"SpringSecurity框架教程-Spring Security+JWT实现项目级前端分离认证授权",  
 "thumbnail":"https://sg-blog-oss.oss-cn-beijing.aliyuncs.com/2022/01/31/948597e164614902ab1662ba8452e106.png",  
 "title":"SpringSecurity从入门到精通",  
 "viewCount":"161"  
 }  
 ],  
 "total":"1"  
 },  
 "msg":"操作成功"  
}

### 5.12 修改文章

#### 5.12.1 需求

点击文章列表中的修改按钮可以跳转到写博文页面。回显示该文章的具体信息。

用户可以在该页面修改文章信息。点击更新按钮后修改文章。

#### 5.12.2 分析

这个功能的实现首先需要能够根据文章id查询文章的详细信息这样才能实现文章的回显。

如何需要提供更新文章的接口。

#### 5.12.3 接口设计

##### 5.12.3.1 查询文章详情接口

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| Get | content/article/{id} | 是 |

Path格式请求参数：

id: 文章id

响应格式：

{  
 "code":200,  
 "data":{  
 "categoryId":"1",  
 "content":"xxxxxxx",  
 "createBy":"1",  
 "createTime":"2022-08-28 15:15:46",  
 "delFlag":0,  
 "id":"10",  
 "isComment":"0",  
 "isTop":"1",  
 "status":"0",  
 "summary":"啊实打实",  
 "tags":[  
 "1",  
 "4",  
 "5"  
 ],  
 "thumbnail":"https://sg-blog-oss.oss-cn-beijing.aliyuncs.com/2022/08/28/7659aac2b74247fe8ebd9e054b916dbf.png",  
 "title":"委屈饿驱蚊器",  
 "updateBy":"1",  
 "updateTime":"2022-08-28 15:15:46",  
 "viewCount":"0"  
 },  
 "msg":"操作成功"  
}

##### 5.12.3.2 更新文章接口

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| PUT | content/article | 是 |

请求体参数格式：

{  
 "categoryId":"1",  
 "content":"![Snipaste\_20220228\_224837.png](https://sg-blog-oss.oss-cn-beijing.aliyuncs.com/2022/08/28/f3938a0368c540ee909ba7f7079a829a.png)\n\n# 十大\n## 时代的",  
 "createBy":"1",  
 "createTime":"2022-08-28 15:15:46",  
 "delFlag":0,  
 "id":"10",  
 "isComment":"0",  
 "isTop":"1",  
 "status":"0",  
 "summary":"啊实打实2",  
 "tags":[  
 "1",  
 "4",  
 "5"  
 ],  
 "thumbnail":"https://sg-blog-oss.oss-cn-beijing.aliyuncs.com/2022/08/28/7659aac2b74247fe8ebd9e054b916dbf.png",  
 "title":"委屈饿驱蚊器",  
 "updateBy":"1",  
 "updateTime":"2022-08-28 15:15:46",  
 "viewCount":"0"  
}

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

### 5.13 删除文章

#### 5.13.1 需求

点击文章后面的删除按钮可以删除该文章

注意：是逻辑删除不是物理删除

#### 5.13.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| DELETE | content/article/{id} | 是 |

Path请求参数：

id：要删除的文章id

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

### 5.14 菜单列表

#### 5.14.1 需求

需要展示菜单列表，不需要分页。

可以针对菜单名进行模糊查询

也可以针对菜单的状态进行查询。

菜单要按照父菜单id和orderNum进行排序

#### 5.14.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| GET | system/menu/list | 是 |

Query请求参数：

status  : 状态

menuName： 菜单名

响应格式：

{  
 "code":200,  
 "data":[  
 {  
 "component":"content/article/write/index",  
 "icon":"build",  
 "id":"2023",  
 "isFrame":1,  
 "menuName":"写博文",  
 "menuType":"C",  
 "orderNum":0,  
 "parentId":"0",  
 "path":"write",  
 "perms":"content:article:writer",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "icon":"system",  
 "id":"1",  
 "isFrame":1,  
 "menuName":"系统管理",  
 "menuType":"M",  
 "orderNum":1,  
 "parentId":"0",  
 "path":"system",  
 "perms":"",  
 "remark":"系统管理目录",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "icon":"table",  
 "id":"2017",  
 "isFrame":1,  
 "menuName":"内容管理",  
 "menuType":"M",  
 "orderNum":4,  
 "parentId":"0",  
 "path":"content",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"system/user/index",  
 "icon":"user",  
 "id":"100",  
 "isFrame":1,  
 "menuName":"用户管理",  
 "menuType":"C",  
 "orderNum":1,  
 "parentId":"1",  
 "path":"user",  
 "perms":"system:user:list",  
 "remark":"用户管理菜单",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"system/role/index",  
 "icon":"peoples",  
 "id":"101",  
 "isFrame":1,  
 "menuName":"角色管理",  
 "menuType":"C",  
 "orderNum":2,  
 "parentId":"1",  
 "path":"role",  
 "perms":"system:role:list",  
 "remark":"角色管理菜单",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"system/menu/index",  
 "icon":"tree-table",  
 "id":"102",  
 "isFrame":1,  
 "menuName":"菜单管理",  
 "menuType":"C",  
 "orderNum":3,  
 "parentId":"1",  
 "path":"menu",  
 "perms":"system:menu:list",  
 "remark":"菜单管理菜单",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"",  
 "icon":"#",  
 "id":"1001",  
 "isFrame":1,  
 "menuName":"用户查询",  
 "menuType":"F",  
 "orderNum":1,  
 "parentId":"100",  
 "path":"",  
 "perms":"system:user:query",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"",  
 "icon":"#",  
 "id":"1002",  
 "isFrame":1,  
 "menuName":"用户新增",  
 "menuType":"F",  
 "orderNum":2,  
 "parentId":"100",  
 "path":"",  
 "perms":"system:user:add",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"",  
 "icon":"#",  
 "id":"1003",  
 "isFrame":1,  
 "menuName":"用户修改",  
 "menuType":"F",  
 "orderNum":3,  
 "parentId":"100",  
 "path":"",  
 "perms":"system:user:edit",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"",  
 "icon":"#",  
 "id":"1004",  
 "isFrame":1,  
 "menuName":"用户删除",  
 "menuType":"F",  
 "orderNum":4,  
 "parentId":"100",  
 "path":"",  
 "perms":"system:user:remove",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"",  
 "icon":"#",  
 "id":"1005",  
 "isFrame":1,  
 "menuName":"用户导出",  
 "menuType":"F",  
 "orderNum":5,  
 "parentId":"100",  
 "path":"",  
 "perms":"system:user:export",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"",  
 "icon":"#",  
 "id":"1006",  
 "isFrame":1,  
 "menuName":"用户导入",  
 "menuType":"F",  
 "orderNum":6,  
 "parentId":"100",  
 "path":"",  
 "perms":"system:user:import",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"",  
 "icon":"#",  
 "id":"1007",  
 "isFrame":1,  
 "menuName":"重置密码",  
 "menuType":"F",  
 "orderNum":7,  
 "parentId":"100",  
 "path":"",  
 "perms":"system:user:resetPwd",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"",  
 "icon":"#",  
 "id":"1008",  
 "isFrame":1,  
 "menuName":"角色查询",  
 "menuType":"F",  
 "orderNum":1,  
 "parentId":"101",  
 "path":"",  
 "perms":"system:role:query",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"",  
 "icon":"#",  
 "id":"1009",  
 "isFrame":1,  
 "menuName":"角色新增",  
 "menuType":"F",  
 "orderNum":2,  
 "parentId":"101",  
 "path":"",  
 "perms":"system:role:add",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"",  
 "icon":"#",  
 "id":"1010",  
 "isFrame":1,  
 "menuName":"角色修改",  
 "menuType":"F",  
 "orderNum":3,  
 "parentId":"101",  
 "path":"",  
 "perms":"system:role:edit",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"",  
 "icon":"#",  
 "id":"1011",  
 "isFrame":1,  
 "menuName":"角色删除",  
 "menuType":"F",  
 "orderNum":4,  
 "parentId":"101",  
 "path":"",  
 "perms":"system:role:remove",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"",  
 "icon":"#",  
 "id":"1012",  
 "isFrame":1,  
 "menuName":"角色导出",  
 "menuType":"F",  
 "orderNum":5,  
 "parentId":"101",  
 "path":"",  
 "perms":"system:role:export",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"",  
 "icon":"#",  
 "id":"1013",  
 "isFrame":1,  
 "menuName":"菜单查询",  
 "menuType":"F",  
 "orderNum":1,  
 "parentId":"102",  
 "path":"",  
 "perms":"system:menu:query",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"",  
 "icon":"#",  
 "id":"1014",  
 "isFrame":1,  
 "menuName":"菜单新增",  
 "menuType":"F",  
 "orderNum":2,  
 "parentId":"102",  
 "path":"",  
 "perms":"system:menu:add",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"",  
 "icon":"#",  
 "id":"1015",  
 "isFrame":1,  
 "menuName":"菜单修改",  
 "menuType":"F",  
 "orderNum":3,  
 "parentId":"102",  
 "path":"",  
 "perms":"system:menu:edit",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"",  
 "icon":"#",  
 "id":"1016",  
 "isFrame":1,  
 "menuName":"菜单删除",  
 "menuType":"F",  
 "orderNum":4,  
 "parentId":"102",  
 "path":"",  
 "perms":"system:menu:remove",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"content/article/index",  
 "icon":"build",  
 "id":"2019",  
 "isFrame":1,  
 "menuName":"文章管理",  
 "menuType":"C",  
 "orderNum":0,  
 "parentId":"2017",  
 "path":"article",  
 "perms":"content:article:list",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"content/category/index",  
 "icon":"example",  
 "id":"2018",  
 "isFrame":1,  
 "menuName":"分类管理",  
 "menuType":"C",  
 "orderNum":1,  
 "parentId":"2017",  
 "path":"category",  
 "perms":"content:category:list",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"content/link/index",  
 "icon":"404",  
 "id":"2022",  
 "isFrame":1,  
 "menuName":"友链管理",  
 "menuType":"C",  
 "orderNum":4,  
 "parentId":"2017",  
 "path":"link",  
 "perms":"content:link:list",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "component":"content/tag/index",  
 "icon":"button",  
 "id":"2021",  
 "isFrame":1,  
 "menuName":"标签管理",  
 "menuType":"C",  
 "orderNum":6,  
 "parentId":"2017",  
 "path":"tag",  
 "perms":"content:tag:index",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "icon":"#",  
 "id":"2028",  
 "isFrame":1,  
 "menuName":"导出分类",  
 "menuType":"F",  
 "orderNum":1,  
 "parentId":"2018",  
 "path":"",  
 "perms":"content:category:export",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "icon":"#",  
 "id":"2024",  
 "isFrame":1,  
 "menuName":"友链新增",  
 "menuType":"F",  
 "orderNum":0,  
 "parentId":"2022",  
 "path":"",  
 "perms":"content:link:add",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "icon":"#",  
 "id":"2025",  
 "isFrame":1,  
 "menuName":"友链修改",  
 "menuType":"F",  
 "orderNum":1,  
 "parentId":"2022",  
 "path":"",  
 "perms":"content:link:edit",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "icon":"#",  
 "id":"2026",  
 "isFrame":1,  
 "menuName":"友链删除",  
 "menuType":"F",  
 "orderNum":1,  
 "parentId":"2022",  
 "path":"",  
 "perms":"content:link:remove",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 {  
 "icon":"#",  
 "id":"2027",  
 "isFrame":1,  
 "menuName":"友链查询",  
 "menuType":"F",  
 "orderNum":2,  
 "parentId":"2022",  
 "path":"",  
 "perms":"content:link:query",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 }  
 ],  
 "msg":"操作成功"  
}

### 5.15 新增菜单

#### 5.15.1 需求

可以新增菜单

#### 5.15.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| POST | content/article | 是 |

请求体参数：

Menu类对应的json格式

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

### 5.16 修改菜单

#### 5.16.1 需求

能够修改菜单，但是修改的时候不能把父菜单设置为当前菜单，如果设置了需要给出相应的提示。并且修改失败。

#### 5.16.2 接口设计

##### 5.16.2.1 根据id查询菜单数据

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| Get | system/menu/{id} | 是 |

Path格式请求参数：

id: 菜单id

响应格式：

{  
 "code":200,  
 "data":{  
 "icon":"table",  
 "id":"2017",  
 "menuName":"内容管理",  
 "menuType":"M",  
 "orderNum":"4",  
 "parentId":"0",  
 "path":"content",  
 "remark":"",  
 "status":"0",  
 "visible":"0"  
 },  
 "msg":"操作成功"  
}

##### 5.16.2.2 更新菜单

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| PUT | system/menu | 是 |

请求体参数：

Menu类对应的json格式

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

如果把父菜单设置为当前菜单:

{  
 "code":500,  
 "msg":"修改菜单'写博文'失败，上级菜单不能选择自己"  
}

### 5.17 删除菜单

#### 5.17.1 需求

能够删除菜单，但是如果要删除的菜单有子菜单则提示：存在子菜单不允许删除 并且删除失败。

#### 5.17.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| DELETE | content/article/{menuId} | 是 |

Path参数：

menuId：要删除菜单的id

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

如果要删除的菜单有子菜单则

{  
 "code":500,  
 "msg":"存在子菜单不允许删除"  
}

### 5.18 角色列表

#### 5.18.1 需求

需要有角色列表分页查询的功能。

要求能够针对角色名称进行模糊查询。

要求能够针对状态进行查询。

要求按照role\_sort进行升序排列。

#### 5.18.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| GET | system/role/list | 是 |

Query格式请求参数：

pageNum: 页码

pageSize: 每页条数

roleName：角色名称

status：状态

响应格式：

{  
 "code":200,  
 "data":{  
 "rows":[  
 {  
 "id":"12",  
 "roleKey":"link",  
 "roleName":"友链审核员",  
 "roleSort":"1",  
 "status":"0"  
 }  
 ],  
 "total":"1"  
 },  
 "msg":"操作成功"  
}

### 5.19 改变角色状态

#### 5.19.1 需求

要求能够修改角色的停启用状态

#### 5.19.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| PUT | system/role/changeStatus | 是 |

请求体：

{"roleId":"11","status":"1"}

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

### 5.20 新增角色

#### 5.20.1 需求

需要提供新增角色的功能。新增角色时能够直接设置角色所关联的菜单权限。

#### 5.20.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| POST | system/role | 是 |

请求体：

{  
 "roleName":"测试新增角色",  
 "roleKey":"wds",  
 "roleSort":0,  
 "status":"0",  
 "menuIds":[  
 "1",  
 "100"  
 ],  
 "remark":"我是角色备注"  
}

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

### 5.21 修改角色

#### 5.21.1 需求

需要提供修改角色的功能。修改角色时可以修改角色所关联的菜单权限

#### 5.21.2 接口设计

##### 5.21.2.1 角色信息回显接口

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| Get | content/role/{id} | 是 |

Path格式请求参数：

id: 角色id

响应格式：

{  
 "code":200,  
 "data":{  
 "id":"11",  
 "remark":"嘎嘎嘎",  
 "roleKey":"aggag",  
 "roleName":"嘎嘎嘎",  
 "roleSort":"5",  
 "status":"0"  
 },  
 "msg":"操作成功"  
}

##### 5.21.2.2 加载对应角色菜单列表树接口

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| Get | /system/menu/roleMenuTreeselect/{id} | 是 |

Path格式请求参数：

id: 角色id

响应格式：

字段介绍

menus：菜单树。

checkedKeys：角色所关联的菜单权限id列表。

{  
 "code":200,  
 "data":{  
 "menus":[  
 {  
 "children":[],  
 "id":"2023",  
 "label":"写博文",  
 "parentId":"0"  
 },  
 {  
 "children":[  
 {  
 "children":[  
 {  
 "children":[],  
 "id":"1001",  
 "label":"用户查询",  
 "parentId":"100"  
 },  
 {  
 "children":[],  
 "id":"1002",  
 "label":"用户新增",  
 "parentId":"100"  
 },  
 {  
 "children":[],  
 "id":"1003",  
 "label":"用户修改",  
 "parentId":"100"  
 },  
 {  
 "children":[],  
 "id":"1004",  
 "label":"用户删除",  
 "parentId":"100"  
 },  
 {  
 "children":[],  
 "id":"1005",  
 "label":"用户导出",  
 "parentId":"100"  
 },  
 {  
 "children":[],  
 "id":"1006",  
 "label":"用户导入",  
 "parentId":"100"  
 },  
 {  
 "children":[],  
 "id":"1007",  
 "label":"重置密码",  
 "parentId":"100"  
 }  
 ],  
 "id":"100",  
 "label":"用户管理",  
 "parentId":"1"  
 },  
 {  
 "children":[  
 {  
 "children":[],  
 "id":"1008",  
 "label":"角色查询",  
 "parentId":"101"  
 },  
 {  
 "children":[],  
 "id":"1009",  
 "label":"角色新增",  
 "parentId":"101"  
 },  
 {  
 "children":[],  
 "id":"1010",  
 "label":"角色修改",  
 "parentId":"101"  
 },  
 {  
 "children":[],  
 "id":"1011",  
 "label":"角色删除",  
 "parentId":"101"  
 },  
 {  
 "children":[],  
 "id":"1012",  
 "label":"角色导出",  
 "parentId":"101"  
 }  
 ],  
 "id":"101",  
 "label":"角色管理",  
 "parentId":"1"  
 },  
 {  
 "children":[  
 {  
 "children":[],  
 "id":"1013",  
 "label":"菜单查询",  
 "parentId":"102"  
 },  
 {  
 "children":[],  
 "id":"1014",  
 "label":"菜单新增",  
 "parentId":"102"  
 },  
 {  
 "children":[],  
 "id":"1015",  
 "label":"菜单修改",  
 "parentId":"102"  
 },  
 {  
 "children":[],  
 "id":"1016",  
 "label":"菜单删除",  
 "parentId":"102"  
 }  
 ],  
 "id":"102",  
 "label":"菜单管理",  
 "parentId":"1"  
 }  
 ],  
 "id":"1",  
 "label":"系统管理",  
 "parentId":"0"  
 },  
 {  
 "children":[  
 {  
 "children":[],  
 "id":"2019",  
 "label":"文章管理",  
 "parentId":"2017"  
 },  
 {  
 "children":[  
 {  
 "children":[],  
 "id":"2028",  
 "label":"导出分类",  
 "parentId":"2018"  
 }  
 ],  
 "id":"2018",  
 "label":"分类管理",  
 "parentId":"2017"  
 },  
 {  
 "children":[  
 {  
 "children":[],  
 "id":"2024",  
 "label":"友链新增",  
 "parentId":"2022"  
 },  
 {  
 "children":[],  
 "id":"2025",  
 "label":"友链修改",  
 "parentId":"2022"  
 },  
 {  
 "children":[],  
 "id":"2026",  
 "label":"友链删除",  
 "parentId":"2022"  
 },  
 {  
 "children":[],  
 "id":"2027",  
 "label":"友链查询",  
 "parentId":"2022"  
 }  
 ],  
 "id":"2022",  
 "label":"友链管理",  
 "parentId":"2017"  
 },  
 {  
 "children":[],  
 "id":"2021",  
 "label":"标签管理",  
 "parentId":"2017"  
 }  
 ],  
 "id":"2017",  
 "label":"内容管理",  
 "parentId":"0"  
 }  
 ],  
 "checkedKeys":[  
 "1001"   
 ]  
 },  
 "msg":"操作成功"  
}

##### 5.21.2.3 更新角色信息接口

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| PUT | system/role | 是 |

请求体：

{  
 "id":"13",  
 "remark":"我是角色备注",  
 "roleKey":"wds",  
 "roleName":"测试新增角色",  
 "roleSort":0,  
 "status":"0",  
 "menuIds":[  
 "1",  
 "100",  
 "1001"  
 ]  
}

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

### 5.22 删除角色

#### 5.22.1 需求

删除固定的某个角色（逻辑删除）

#### 5.22.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| DELETE | system/role/{id} | 是 |

Path请求参数：

id：要删除的角色id

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

### 5.23 用户列表

#### 5.23.1 需求

需要用户分页列表接口。

可以根据用户名模糊搜索。

可以进行手机号的搜索。

可以进行状态的查询。

#### 5.23.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| GET | system/user/list | 是 |

Query格式请求参数：

pageNum: 页码

pageSize: 每页条数

userName：用户名

phonenumber：手机号

status:状态

响应格式：

{  
 "code":200,  
 "data":{  
 "rows":[  
 {  
 "avatar":"http://r7yxkqloa.bkt.clouddn.com/2022/03/05/75fd15587811443a9a9a771f24da458d.png",  
 "createTime":"2022-01-05 17:01:56",  
 "email":"23412332@qq.com",  
 "id":"1",  
 "nickName":"sg3334",  
 "phonenumber":"18888888888",  
 "sex":"1",  
 "status":"0",  
 "updateBy":"1",  
 "updateTime":"2022-03-13 21:36:22",  
 "userName":"sg"  
 }  
 ],  
 "total":"1"  
 },  
 "msg":"操作成功"  
}

### 5.24 新增用户

#### 5.24.1 需求

需要新增用户功能。新增用户时可以直接关联角色。

注意：新增用户时注意密码加密存储。

用户名不能为空，否则提示：必需填写用户名

用户名必须之前未存在，否则提示：用户名已存在

   手机号必须之前未存在，否则提示：手机号已存在

邮箱必须之前未存在，否则提示：邮箱已存在

#### 5.24.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| POST | system/user | 是 |

请求体：

{  
 "userName":"wqeree",  
 "nickName":"测试新增用户",  
 "password":"1234343",  
 "phonenumber":"18889778907",  
 "email":"233@sq.com",  
 "sex":"0",  
 "status":"0",  
 "roleIds":[  
 "2"  
 ]  
}

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

### 5.25 删除用户

#### 5.25.1 需求

删除固定的某个用户（逻辑删除）

#### 5.25.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| DELETE | /system/user/{id} | 是 |

Path请求参数：

id：要删除的用户id

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

### 5.26 修改用户

#### 5.26.1 需求

需要提供修改用户的功能。修改用户时可以修改用户所关联的角色。

#### 5.26.2 接口设计

##### 5.26.2.1 根据id查询用户信息回显接口

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| Get | /system/user/{id} | 是 |

Path格式请求参数：

id: 用户id

响应格式：

roleIds：用户所关联的角色id列表

roles：所有角色的列表

user：用户信息

{  
 "code":200,  
 "data":{  
 "roleIds":[  
 "11"  
 ],  
 "roles":[  
 {  
 "createBy":"0",  
 "createTime":"2021-11-12 18:46:19",  
 "delFlag":"0",  
 "id":"1",  
 "remark":"超级管理员",  
 "roleKey":"admin",  
 "roleName":"超级管理员",  
 "roleSort":"1",  
 "status":"0",  
 "updateBy":"0"  
 },  
 {  
 "createBy":"0",  
 "createTime":"2021-11-12 18:46:19",  
 "delFlag":"0",  
 "id":"2",  
 "remark":"普通角色",  
 "roleKey":"common",  
 "roleName":"普通角色",  
 "roleSort":"2",  
 "status":"0",  
 "updateBy":"0",  
 "updateTime":"2022-01-02 06:32:58"  
 },  
 {  
 "createTime":"2022-01-06 22:07:40",  
 "delFlag":"0",  
 "id":"11",  
 "remark":"嘎嘎嘎",  
 "roleKey":"aggag",  
 "roleName":"嘎嘎嘎",  
 "roleSort":"5",  
 "status":"0",  
 "updateBy":"1",  
 "updateTime":"2022-09-11 20:34:49"  
 },  
 {  
 "createTime":"2022-01-16 14:49:30",  
 "delFlag":"0",  
 "id":"12",  
 "roleKey":"link",  
 "roleName":"友链审核员",  
 "roleSort":"1",  
 "status":"0",  
 "updateTime":"2022-01-16 16:05:09"  
 }  
 ],  
 "user":{  
 "email":"weq@2132.com",  
 "id":"14787164048663",  
 "nickName":"sg777",  
 "sex":"0",  
 "status":"0",  
 "userName":"sg777"  
 }  
 },  
 "msg":"操作成功"  
}

##### 5.26.2.2 更新用户信息接口

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| PUT | /system/user | 是 |

请求体：

{  
 "email":"weq@2132.com",  
 "id":"14787164048663",  
 "nickName":"sg777",  
 "sex":"1",  
 "status":"0",  
 "userName":"sg777",  
 "roleIds":[  
 "11"  
 ]  
}

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

### 5.27 分页查询分类列表

#### 5.27.1 需求

需要分页查询分类列表。

能根据分类名称进行模糊查询。

能根据状态进行查询。

#### 5.27.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| GET | content/category/list | 是 |

Query格式请求参数：

pageNum: 页码

pageSize: 每页条数

name：分类名

status:状态

响应格式：

{  
 "code":200,  
 "data":{  
 "rows":[  
 {  
 "description":"wsd",  
 "id":"1",  
 "name":"java",  
 "status":"0"  
 },  
 {  
 "description":"wsd",  
 "id":"2",  
 "name":"PHP",  
 "status":"0"  
 }  
 ],  
 "total":"2"  
 },  
 "msg":"操作成功"  
}

### 5.28 新增分类

#### 5.28.1 需求

需要新增分类功能

#### 5.28.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| POST | /content/category | 是 |

请求体：

{  
 "name":"威威",  
 "description":"是的",  
 "status":"0"  
}

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

### 5.29 修改分类

#### 5.29.1 需求

需要提供修改分类的功能

#### 5.29.2 接口设计

##### 5.29.2.1 根据id查询分类

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| Get | content/category/{id} | 是 |

Path格式请求参数：

id: 分类id

响应格式：

{  
 "code":200,  
 "data":{  
 "description":"qwew",  
 "id":"4",  
 "name":"ww",  
 "status":"0"  
 },  
 "msg":"操作成功"  
}

##### 5.29.2.2 更新分类

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| PUT | /content/category | 是 |

请求体：

{  
 "description":"是的",  
 "id":"3",  
 "name":"威威2",  
 "status":"0"  
}

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

### 5.30 删除分类

#### 5.30.1 需求

删除某个分类（逻辑删除）

#### 5.30.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| DELETE | /content/category/{id} | 是 |

Path请求参数：

id：要删除的分类id

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

### 5.31 分页查询友链列表

#### 5.31.1 需求

需要分页查询友链列表。

能根据友链名称进行模糊查询。

能根据状态进行查询。

#### 5.31.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| GET | /content/link/list | 是 |

Query格式请求参数：

pageNum: 页码

pageSize: 每页条数

name：友链名

status:状态

响应格式：

{  
 "code":200,  
 "data":{  
 "rows":[  
 {  
 "address":"https://www.baidu.com",  
 "description":"sda",  
  
 "name":"sda",  
 "status":"0"  
 }  
 ],  
 "total":"1"  
 },  
 "msg":"操作成功"  
}

### 5.32 新增友链

#### 5.32.1 需求

需要新增友链功能

#### 5.32.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| POST | /content/link | 是 |

请求体：

{  
 "name":"sda",  
 "description":"weqw",  
 "address":"wewe",  
 "logo":"weqe",  
 "status":"2"  
}

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

### 5.33 修改友链

#### 5.33.1 需求

需要提供修改友链的功能

#### 5.33.2 接口设计

##### 5.33.2.1 根据id查询友联

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| Get | content/link/{id} | 是 |

Path格式请求参数：

id: 友链id

响应格式：

{  
 "code":200,  
 "data":{  
 "address":"wewe",  
 "description":"weqw",  
 "id":"4",  
 "logo":"weqe",  
 "name":"sda",  
 "status":"2"  
 },  
 "msg":"操作成功"  
}

##### 5.33.2.2 修改友链

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| PUT | /content/link | 是 |

请求体：

{  
 "address":"https://www.qq.com",  
 "description":"dada2",  
 "id":"2",  
  
 "name":"sda",  
 "status":"0"  
}

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}

### 5.34 删除友链

#### 5.34.1 需求

删除某个友链（逻辑删除）

#### 5.34.2 接口设计

|  |  |  |
| --- | --- | --- |
| 请求方式 | 请求路径 | 是否需求token头 |
| DELETE | /content/link/{id} | 是 |

Path请求参数：

id：要删除的友链id

响应格式：

{  
 "code":200,  
 "msg":"操作成功"  
}