**课程资料自主学习系统**

**程序设计与编码**

**所属系统：课程资料自主学习系统**

**文件类型：程序设计与编码**

**编 写 者：唐聪**

**com.egao.common.config:**

@Configuration

public class BeetlConfiguration {

@Value("${beetl.templatesPath:templates}")

private String templatesPath; // 模板根目录

@Value("${beetl.suffix:html}")

private String suffix; // 模板后缀

@Value("${beetl.dev:true}")

private boolean dev; // 开启热加载

@Bean(name = "beetlConfig")

public BeetlGroupUtilConfiguration getBeetlGroupUtilConfiguration() {

BeetlGroupUtilConfiguration bguc = new BeetlGroupUtilConfiguration();

Properties extProperties = new Properties();

extProperties.put("RESOURCE.autoCheck", dev ? "true" : "false");

extProperties.put("HTML\_TAG\_FLAG", ":");

bguc.setConfigProperties(extProperties);

ClassLoader loader = Thread.currentThread().getContextClassLoader();

if (loader == null) loader = BeetlConfiguration.class.getClassLoader();

bguc.setResourceLoader(new ClasspathResourceLoader(loader, templatesPath));

bguc.init();

// 增加自定义函数

bguc.getGroupTemplate().registerFunctionPackage("so", new ShiroExt());

// 增加自定义标签

bguc.getGroupTemplate().registerTag("include", org.beetl.ext.tag.html.IncludeResourceHtmlTag.class);

bguc.getGroupTemplate().registerTag("layout", org.beetl.ext.tag.html.LayoutResourceHtmlTag.class);

bguc.getGroupTemplate().registerTag("set", org.beetl.ext.tag.html.SetHtmlTag.class);

bguc.getGroupTemplate().registerTag("if", org.beetl.ext.tag.html.IfHtmlTag.class);

bguc.getGroupTemplate().registerTag("for", org.beetl.ext.tag.html.ForeachHtmlTag.class);

return bguc;

}

@Bean(name = "beetlViewResolver")

public BeetlSpringViewResolver getBeetlSpringViewResolver(@Qualifier("beetlConfig") BeetlGroupUtilConfiguration bguc) {

BeetlSpringViewResolver bsvr = new BeetlSpringViewResolver();

bsvr.setContentType("text/html;charset=UTF-8");

bsvr.setViewNames("\*." + suffix, "\*." + suffix + "#\*");

bsvr.setOrder(0);

bsvr.setConfig(bguc);

return bsvr;

}

}

@Component

public class DateConverterConfig implements Converter<String, Date> {

private static final List<String> formats = new ArrayList<>();

/\*

\* 以下几种时间格式自动转成Date类型

\*/

static {

formats.add("yyyy-MM-dd");

formats.add("yyyy-MM-dd HH:mm");

formats.add("yyyy-MM-dd HH:mm:ss");

}

@Override

public Date convert(String s) {

if (s == null || s.trim().isEmpty()) {

return null;

} else if (s.matches("^\\d{4}-\\d{1,2}-\\d{1,2}$")) {

return parseDate(s, formats.get(0));

} else if (s.matches("^\\d{4}-\\d{1,2}-\\d{1,2} {1}\\d{1,2}:\\d{1,2}$")) {

return parseDate(s, formats.get(1));

} else if (s.matches("^\\d{4}-\\d{1,2}-\\d{1,2} {1}\\d{1,2}:\\d{1,2}:\\d{1,2}$")) {

return parseDate(s, formats.get(2));

} else {

throw new IllegalArgumentException("DateConverterConfig: Invalid date value '" + s + "'");

}

}

private Date parseDate(String date, String format) {

SimpleDateFormat sdf = new SimpleDateFormat(format);

try {

return sdf.parse(date);

} catch (ParseException e) {

e.printStackTrace();

}

return null;

}

}

@EnableTransactionManagement

@Configuration

public class MybatisPlusConfig {

/\*\*

\* 分页插件

\*/

@Bean

public PaginationInterceptor paginationInterceptor() {

return new PaginationInterceptor().setLimit(1000);

}

}

@Configuration

public class WebMvcConfig implements WebMvcConfigurer {

/\*\*

\* 上传文件路径-映射

\*/

@Override

public void addResourceHandlers(ResourceHandlerRegistry registry) {

//如果是Windows系统

registry.addResourceHandler("/getFileURL/\*\*").addResourceLocations("file:D:/");

}

/\*\*

\* 支持跨域访问

\*/

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/api/\*\*").maxAge(3600)

.allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")

.allowedHeaders("Content-Type", "x-requested-with", "X-Custom-Header", "Authorization");

}

/\*\*

\* 支持PUT、DELETE请求

\*/

@Bean

public FormContentFilter httpPutFormContentFilter() {

return new FormContentFilter();

}

}

**com.egao.common.shiro:**

public class AdminRealm extends AuthorizingRealm {

@Lazy

@Autowired

private UserService userService;

@Autowired

private UserRoleService userRoleService;

/\*\*

\* 认证

\*/

@Override

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws AuthenticationException {

String username = (String) authenticationToken.getPrincipal();

Boolean flag=false;

Integer[] roles=userRoleService.getRoleIds(userService.getByUsername(username).getUserId().toString());

for (Integer i :roles){

if (i==1||i==2) flag=true;

}

User user = userService.getByUsername(username);

if (!flag) throw new UnknownAccountException(); // 账号不存在

if (user.getState() != 0) throw new LockedAccountException(); // 账号被锁定

return new SimpleAuthenticationInfo(user, user.getPassword(), getName());

}

/\*\*

\* 授权

\*/

@Override

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {

User user = (User) SecurityUtils.getSubject().getPrincipal();

SimpleAuthorizationInfo authorizationInfo = new SimpleAuthorizationInfo();

userService.selectRoleAndAuth(user);

// 角色

Set<String> roles = new HashSet<>();

for (Role r : user.getRoles()) {

if (r.getDeleted() == 0) roles.add(r.getRoleCode());

}

authorizationInfo.setRoles(roles);

// 权限

Set<String> permissions = new HashSet<>();

for (String auth : user.getAuthorities()) {

if (auth != null && !auth.trim().isEmpty()) permissions.add(auth);

}

authorizationInfo.setStringPermissions(permissions);

return authorizationInfo;

}

}

public class CustomizedModularRealmAuthenticator extends ModularRealmAuthenticator{

@Override

protected AuthenticationInfo doAuthenticate(AuthenticationToken authenticationToken)

throws AuthenticationException {

// 判断getRealms()是否返回为空

assertRealmsConfigured();

// 强制转换回自定义的CustomizedToken

CustomizedToken customizedToken = (CustomizedToken) authenticationToken;

// 登录类型

String loginType = customizedToken.getLoginType();

// 所有Realm

Collection<Realm> realms = getRealms();

// 登录类型对应的所有Realm

Collection<Realm> typeRealms = new ArrayList<>();

for (Realm realm : realms) {

if (realm.getName().contains(loginType))

typeRealms.add(realm);

}

// 判断是单Realm还是多Realm

if (typeRealms.size() == 1)

return doSingleRealmAuthentication(typeRealms.iterator().next(), customizedToken);

else

return doMultiRealmAuthentication(typeRealms, customizedToken);

}

}

public class CustomizedToken extends UsernamePasswordToken{

//登录类型，判断是普通用户登录，教师登录还是管理员登录

private String loginType;

public CustomizedToken(final String username,final String password,final Boolean remember, String loginType) {

super(username,password,remember);

this.loginType = loginType;

}

public String getLoginType() {

return loginType;

}

public void setLoginType(String loginType) {

this.loginType = loginType;

}

}

@Configuration

public class ShiroConfig {

/\*\*

\* shiro过滤器

\*/

@Bean(name = "shiroFilter")

public ShiroFilterFactoryBean shiroFilter(DefaultWebSecurityManager securityManager) {

ShiroFilterFactoryBean shiroFilter = new ShiroFilterFactoryBean();

shiroFilter.setSecurityManager(securityManager);

// 登录配置

shiroFilter.setLoginUrl("/login");

shiroFilter.setSuccessUrl("/");

shiroFilter.setUnauthorizedUrl("/error?code=403");

// 自定义过滤器

Map<String, Filter> filtersMap = new LinkedHashMap<>();

filtersMap.put("slf", new ShiroLoginFilter());

shiroFilter.setFilters(filtersMap);

// 拦截配置

Map<String, String> filterChainDefinitions = new LinkedHashMap<>();

filterChainDefinitions.put("/assets/\*\*", "anon");

filterChainDefinitions.put("/favicon.ico", "anon");

filterChainDefinitions.put("/api/\*\*", "anon");

filterChainDefinitions.put("/error", "anon");

filterChainDefinitions.put("/login", "anon");

filterChainDefinitions.put("/default/\*\*", "anon");

filterChainDefinitions.put("/users/\*\*", "anon");

filterChainDefinitions.put("/getFileURL/\*\*", "anon");

filterChainDefinitions.put("/logout", "logout");

//filterChainDefinitions.put("/\*\*", "slf,authc");

filterChainDefinitions.put("/\*\*", "slf,user"); // 记住密码也能访问

shiroFilter.setFilterChainDefinitionMap(filterChainDefinitions);

return shiroFilter;

}

/\*\*

\* adminRealm

\*/

@Bean(name = "adminRealm")

@DependsOn("lifecycleBeanPostProcessor")

public AdminRealm adminRealm() {

AdminRealm adminRealm = new AdminRealm();

// 密码凭证器

adminRealm.setCredentialsMatcher(new HashedCredentialsMatcher("md5"));

return adminRealm;

}

/\*\*

\* userRealm

\*/

@Bean(name = "userRealm")

@DependsOn("lifecycleBeanPostProcessor")

public UserRealm userRealm() {

UserRealm userRealm = new UserRealm();

// 密码凭证器

userRealm.setCredentialsMatcher(new HashedCredentialsMatcher("md5"));

return userRealm;

}

/\*\*

\* 安全管理器

\*/

@Bean(name = "securityManager")

public DefaultWebSecurityManager securityManager() {

DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();

//设置realm.

securityManager.setAuthenticator(modularRealmAuthenticator());

List<Realm> realms = new ArrayList<>();

realms.add(userRealm());

realms.add(adminRealm());

securityManager.setRealms(realms);

securityManager.setCacheManager(cacheManager());

securityManager.setRememberMeManager(cookieRememberMeManager());

return securityManager;

}

/\*\*

\* 系统自带的Realm管理，主要针对多realm

\* \*/

@Bean

public ModularRealmAuthenticator modularRealmAuthenticator(){

//自己重写的ModularRealmAuthenticator

CustomizedModularRealmAuthenticator modularRealmAuthenticator = new CustomizedModularRealmAuthenticator();

modularRealmAuthenticator.setAuthenticationStrategy(new AtLeastOneSuccessfulStrategy());

return modularRealmAuthenticator;

}

/\*\*

\* 缓存管理器

\*/

@Bean(name = "cacheManager")

public EhCacheManager cacheManager() {

EhCacheManager cacheManager = new EhCacheManager();

net.sf.ehcache.CacheManager cm = net.sf.ehcache.CacheManager.getCacheManager("shirocache");

if (cm == null) {

String configFile = "classpath:ehcache/ehcache-shiro.xml";

InputStream is = null;

try {

is = ResourceUtils.getInputStreamForPath(configFile);

cm = new net.sf.ehcache.CacheManager(is);

} catch (IOException e) {

throw new IllegalStateException("Unable to obtain input stream for cacheManagerConfigFile [" + configFile + "]", e);

} finally {

ResourceUtils.close(is);

}

}

cacheManager.setCacheManager(cm);

return cacheManager;

}

/\*\*

\* 记住密码cookie

\*/

@Bean

public SimpleCookie rememberMeCookie() {

SimpleCookie simpleCookie = new SimpleCookie("rememberMe");

simpleCookie.setMaxAge(60 \* 60 \* 24 \* 7); // 过期时间7天

return simpleCookie;

}

/\*\*

\* 记住密码cookie管理器

\*/

@Bean

public CookieRememberMeManager cookieRememberMeManager() {

CookieRememberMeManager manager = new CookieRememberMeManager();

manager.setCookie(rememberMeCookie());

manager.setCipherKey(Base64.decode("BT7lf0hw4W/QMxpS/Rb+Ng=="));

return manager;

}

/\*\*

\* 开启shiro注解功能

\*/

@Bean

public AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor() {

AuthorizationAttributeSourceAdvisor advisor = new AuthorizationAttributeSourceAdvisor();

advisor.setSecurityManager(securityManager());

return advisor;

}

@Bean(name = "lifecycleBeanPostProcessor")

public LifecycleBeanPostProcessor lifecycleBeanPostProcessor() {

return new LifecycleBeanPostProcessor();

}

}

public class ShiroLoginFilter extends AccessControlFilter {

@Override

protected boolean isAccessAllowed(ServletRequest servletRequest, ServletResponse servletResponse, Object o) {

return false;

}

@Override

protected boolean onAccessDenied(ServletRequest servletRequest, ServletResponse servletResponse) throws Exception {

Subject subject = getSubject(servletRequest, servletResponse);

if (!subject.isAuthenticated() && !subject.isRemembered()) {

if (isAjax((HttpServletRequest) servletRequest)) {

servletResponse.setContentType("application/json;charset=UTF-8");

PrintWriter out = servletResponse.getWriter();

out.write("{\"msg\": \"登录过期，请重新登录\", \"code\": 401}");

out.flush();

return false;

}

}

return true;

}

/\*\*

\* 判断是不是ajax请求

\*/

private boolean isAjax(HttpServletRequest request) {

String xHeader = request.getHeader("X-Requested-With");

return (xHeader != null && xHeader.contains("XMLHttpRequest"));

}

}

public class UserRealm extends AuthorizingRealm {

@Lazy

@Autowired

private UserService userService;

@Autowired

private UserRoleService userRoleService;

/\*\*

\* 认证

\*/

@Override

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws AuthenticationException {

String username = (String) authenticationToken.getPrincipal();

Boolean flag=false;

Integer[] roles=userRoleService.getRoleIds(userService.getByUsername(username).getUserId().toString());

for (Integer i :roles){

if (i==3) flag=true;

}

User user = userService.getByUsername(username);

if (!flag) throw new UnknownAccountException(); // 账号不存在

if (user.getState() != 0) throw new LockedAccountException(); // 账号被锁定

return new SimpleAuthenticationInfo(user, user.getPassword(), getName());

}

@Override

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {

// TODO Auto-generated method stub

return null;

}

}

**com.egao.common.config:**

public class CoreUtil {

private static String[] chars = new String[]{"a", "b", "c", "d", "e", "f",

"g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s",

"t", "u", "v", "w", "x", "y", "z", "0", "1", "2", "3", "4", "5",

"6", "7", "8", "9", "A", "B", "C", "D", "E", "F", "G", "H", "I",

"J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V",

"W", "X", "Y", "Z"};

/\*\*

\* toString，为null返回空白字符

\*/

public static String toString(Object obj) {

return obj == null ? "" : obj.toString();

}

/\*\*

\* 连接多个字符串，null自动过滤

\*/

public static String connect(Object... objects) {

StringBuilder sb = new StringBuilder();

for (Object obj : objects) sb.append(toString(obj));

return sb.toString();

}

/\*\*

\* 首字母大写

\*/

public static String upperHead(String str) {

if (str == null || str.length() == 0) return str;

if (str.length() == 1) return str.toUpperCase();

return str.substring(0, 1).toUpperCase() + str.substring(1);

}

/\*\*

\* 生成8位uuid

\*/

public static String randomUUID8() {

StringBuffer buffer = new StringBuffer();

String uuid = UUID.randomUUID().toString().replace("-", "");

for (int i = 0; i < 8; i++) {

String str = uuid.substring(i \* 4, i \* 4 + 4);

int x = Integer.parseInt(str, 16);

buffer.append(chars[x % 0x3E]);

}

return buffer.toString();

}

/\*\*

\* 生成16位uuid

\*/

public static String randomUUID16() {

StringBuffer buffer = new StringBuffer();

String uuid = UUID.randomUUID().toString().replace("-", "");

for (int i = 0; i < 16; i++) {

String str = uuid.substring(i \* 2, i \* 2 + 2);

int x = Integer.parseInt(str, 16);

buffer.append(chars[x % 0x3E]);

}

return buffer.toString();

}

/\*\*

\* 把对象转成Map

\*/

public static <T> Map<String, Object> objectToMap(T t) {

return objectToMap(t, null);

}

/\*\*

\* 把对象集合转成Map集合

\*/

public static <T> List<Map<String, Object>> objectToMap(List<T> ts) {

return objectToMap(ts, null);

}

/\*\*

\* 把对象转成Map，只包含指定字段

\*

\* @param t 对象

\* @param fields 包含的字段

\* @return Map

\*/

public static <T> Map<String, Object> objectToMap(T t, String[] fields) {

if (t == null) return null;

List<String> fieldList = null;

if (fields != null) fieldList = Arrays.asList(fields);

Map<String, Object> map = new HashMap<>();

Field[] fieldArray = t.getClass().getDeclaredFields();

for (Field field : fieldArray) {

field.setAccessible(true);

if (fieldList == null || fieldList.contains(field.getName())) {

try {

map.put(field.getName(), field.get(t));

} catch (IllegalAccessException e) {

e.printStackTrace();

}

}

}

return map;

}

/\*\*

\* 把对个对象集合转成Map集合，只包含指定字段

\*

\* @param ts 对象集合

\* @param fields 包含的字段

\* @return List<Map>

\*/

public static <T> List<Map<String, Object>> objectToMap(List<T> ts, String[] fields) {

List<Map<String, Object>> rs = new ArrayList<>();

for (T t : ts) {

Map<String, Object> map = objectToMap(t, fields);

if (map != null) rs.add(map);

}

return rs;

}

/\*\*

\* 复制父类的属性的值到子类

\*

\* @param f 父类对象

\* @param c 子类对象

\*/

public static <F, C extends F> void copyAttribute(F f, C c) {

for (Field field : f.getClass().getDeclaredFields()) {

try {

field.setAccessible(true);

Field cf = c.getClass().getField(field.getName());

cf.setAccessible(true);

cf.set(c, field.get(f));

} catch (Exception e) {

e.printStackTrace();

}

}

}

/\*\*

\* 把父类克隆为子类

\*

\* @param father 父类对象

\* @param clazz 子类类型

\*/

public static <F, C extends F> C cloneToChild(F father, Class<C> clazz) {

try {

C child = clazz.newInstance();

copyAttribute(father, child);

return child;

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

/\*\*

\* 判断excel某列是否有空值

\*/

public static String excelCheckBlank(List<List<Object>> list, int startRow, int... cols) {

StringBuilder sb = new StringBuilder();

for (int col : cols) {

for (int i = 0; i < list.size(); i++) {

Object value = list.get(i).get(col);

if (value == null || StrUtil.isBlank(value.toString())) {

if (sb.length() != 0) sb.append("\r\n");

sb.append("第").append(i + startRow + 1).append("行第");

sb.append(col + 1).append("列不能为空.");

}

}

}

return sb.toString();

}

/\*\*

\* 判断excel某列是否有重复值

\*/

public static String excelCheckRepeat(List<List<Object>> list, int startRow, int... cols) {

StringBuilder sb = new StringBuilder();

for (int col : cols) {

for (int i = 0; i < list.size(); i++) {

Object value = list.get(i).get(col);

for (int j = 0; j < list.size(); j++) {

if (i != j && value != null && value.equals(list.get(j).get(col))) {

if (sb.length() != 0) sb.append("\r\n");

sb.append("第").append(i + startRow + 1).append("行第").append(col + 1).append("列与第");

sb.append(j + startRow + 1).append("行第").append(col + 1).append("列重复.");

}

}

}

}

return sb.toString();

}

/\*\*

\* 检查list集合中元素字段是否有重复

\*

\* @param list 集合

\* @param field 字段名称

\* @return 返回重复的元素

\*/

public static <T> T listCheckRepeat(List<T> list, String field) {

for (int i = 0; i < list.size(); i++) {

for (int j = 0; j < list.size(); j++) {

if (i != j) {

Object value = getFieldValue(list.get(i), field);

if (value != null && value.equals(getFieldValue(list.get(j), field))) {

return list.get(j);

}

}

}

}

return null;

}

/\*\*

\* 检查list集合中元素字段是否有重复

\*

\* @param list 集合

\* @param field 字段名称

\* @param field 字段中文名称，用于错误提示

\* @return 返回错误提示信息

\*/

public static <T> String listCheckRepeat(List<T> list, String field, String zhName) {

StringBuilder sb = new StringBuilder();

for (int i = 0; i < list.size(); i++) {

for (int j = 0; j < list.size(); j++) {

if (i != j) {

Object value = getFieldValue(list.get(i), field);

if (value != null && value.equals(getFieldValue(list.get(j), field))) {

if (sb.length() != 0) sb.append("\r\n");

sb.append("第").append(i + 1).append("条与第").append(j + 1).append("条数据的").append(zhName).append("重复.");

}

}

}

}

return sb.toString();

}

/\*\*

\* 检查list集合中元素字段是否有空值

\*

\* @param list 集合

\* @param field 字段名称

\* @return 返回为空的元素

\*/

public static <T> T listCheckBlank(List<T> list, String field) {

for (int i = 0; i < list.size(); i++) {

Object value = getFieldValue(list.get(i), field);

if (value == null || StrUtil.isBlank(value.toString())) {

return list.get(i);

}

}

return null;

}

/\*\*

\* 检查list集合中元素字段是否有空值

\*

\* @param list 集合

\* @param field 字段名称

\* @param field 字段中文名称，用于错误提示

\* @return 返回错误提示信息

\*/

public static <T> String listCheckBlank(List<T> list, String field, String zhName) {

StringBuilder sb = new StringBuilder();

for (int i = 0; i < list.size(); i++) {

Object value = getFieldValue(list.get(i), field);

if (value == null || StrUtil.isBlank(value.toString())) {

if (sb.length() != 0) sb.append("\r\n");

sb.append("第").append(i + 1).append("条数据的").append(zhName).append("不能为空.");

}

}

return sb.toString();

}

/\*\*

\* 获取某个对象的某个字段的值

\*/

public static Object getFieldValue(Object t, String field) {

if (t == null || field == null) return null;

try {

Field clazzField = t.getClass().getDeclaredField(field);

clazzField.setAccessible(true);

return clazzField.get(t);

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

}

public class FileUploadUtil {

// 文件上传的目录

public static final String UPLOAD\_FILE\_DIR = Constants.UPLOAD\_DIR + "file/";

// 缩略图存放的目录

public static final String UPLOAD\_SM\_DIR = Constants.UPLOAD\_DIR + "thumbnail/";

public static final String UPLOAD\_PATH = "http://localhost:8081/getFileURL/file/";

/\*\*

\* 上传文件

\*

\* @param file MultipartFile

\* @return 示例：{"code": 0, "msg": "", "url": "", "fileName": ""}

\*/

public static JsonResult upload(MultipartFile file) {

String path; // 文件保存路径

// 文件原始名称

String orgName = file.getOriginalFilename();

if (orgName == null) return JsonResult.error("上传失败");

File outFile;

String suffix = orgName.substring(orgName.lastIndexOf(".") + 1); // 获取文件后缀

if (Constants.UPLOAD\_UUID\_NAME) { // uuid命名

path = getDateDir() + UUID.randomUUID().toString().replaceAll("-", "") + "." + suffix;

outFile = new File(UPLOAD\_FILE\_DIR, path);

} else { // 使用原名称，存在相同着加(1)

path = getDateDir() + orgName;

outFile = new File(UPLOAD\_FILE\_DIR, path);

int sameSize = 1;

String prefix = orgName.substring(0, orgName.lastIndexOf(".")); // 获取文件名称

while (outFile.exists()) {

path = getDateDir() + prefix + "(" + sameSize + ")." + suffix;

outFile = new File(UPLOAD\_FILE\_DIR, path);

sameSize++;

}

}

try {

if (!outFile.getParentFile().exists()) {

if (!outFile.getParentFile().mkdirs()) return JsonResult.error("上传失败");

}

file.transferTo(outFile);

return JsonResult.ok("上传成功").put("url", path).put("fileName", orgName);

} catch (Exception e) {

e.printStackTrace();

return JsonResult.error("上传失败").put("error", e.getMessage());

}

}

public static JsonResult edit\_upload(MultipartFile file) {

String path; // 文件保存路径

// 文件原始名称

String orgName = file.getOriginalFilename(), dir = getDateDir();

if (orgName == null) return JsonResult.error("上传失败");

File outFile;

String suffix = orgName.substring(orgName.lastIndexOf(".") + 1); // 获取文件后缀

if (Constants.UPLOAD\_UUID\_NAME) { // uuid命名

path = dir + UUID.randomUUID().toString().replaceAll("-", "") + "." + suffix;

outFile = new File(UPLOAD\_FILE\_DIR, path);

} else { // 使用原名称，存在相同着加(1)

path = dir + orgName;

outFile = new File(UPLOAD\_FILE\_DIR, path);

int sameSize = 1;

String prefix = orgName.substring(0, orgName.lastIndexOf(".")); // 获取文件名称

while (outFile.exists()) {

path = dir + prefix + "(" + sameSize + ")." + suffix;

outFile = new File(UPLOAD\_FILE\_DIR, path);

sameSize++;

}

}

try {

if (!outFile.getParentFile().exists()) {

if (!outFile.getParentFile().mkdirs()) return JsonResult.error("上传失败");

}

file.transferTo(outFile);

Map<String,Object> map = new HashMap<String,Object>();

map.put("src", UPLOAD\_PATH+path);

map.put("title", orgName);

JsonResult jsonResult = JsonResult.ok("上传成功").put("data", map);

return jsonResult;

} catch (Exception e) {

e.printStackTrace();

return JsonResult.error("上传失败").put("error", e.getMessage());

}

}

/\*\*

\* 上传文件base64格式

\*

\* @param base64 base64编码字符

\* @return 示例：{"code": 0, "msg": "", "url": ""}

\*/

public static JsonResult upload(String base64) {

if (base64 == null || base64.trim().isEmpty()) return JsonResult.error("上传失败");

String suffix = base64.substring(11, base64.indexOf(";")); // 获取文件格式

String path = getDateDir() + UUID.randomUUID().toString().replaceAll("-", "") + "." + suffix;

File outFile = new File(UPLOAD\_FILE\_DIR, path);

if (!outFile.getParentFile().exists()) {

if (!outFile.getParentFile().mkdirs()) return JsonResult.error("上传失败");

}

try {

byte[] bytes = Base64.getDecoder().decode(base64.substring(base64.indexOf(";") + 8).getBytes());

IoUtil.write(new FileOutputStream(outFile), true, bytes);

} catch (Exception e) {

e.printStackTrace();

return JsonResult.error("上传失败").put("error", e.getMessage());

}

return JsonResult.ok("上传成功").put("url", path);

}

/\*\*

\* 按照日期分存放目录

\*/

public static String getDateDir() {

SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMdd/");

return sdf.format(new Date());

}

/\*\*

\* 预览文件

\*/

public static void preview(File file, HttpServletResponse response) {

if (file == null || !file.exists()) {

outNotFund(response);

return;

}

// 支持word、excel预览

if (OpenOfficeUtil.canConverter(file.getName())) {

File pdfFile = OpenOfficeUtil.converterToPDF(file.getAbsolutePath());

if (pdfFile != null) file = pdfFile;

}

String contentType = getFileType(file); // 获取文件类型

if (contentType != null) {

response.setContentType(contentType);

} else {

setDownloadHeader(response, file.getName());

}

try {

output(file, response.getOutputStream());

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* 预览源文件

\*/

public static void preview(String path, HttpServletResponse response) {

preview(new File(UPLOAD\_FILE\_DIR, path), response);

}

/\*\*

\* 下载源文件

\*/

public static void download(String path, HttpServletResponse response) {

File file = new File(UPLOAD\_FILE\_DIR, path);

if (!file.exists()) {

outNotFund(response);

return;

}

setDownloadHeader(response, file.getName());

try {

output(file, response.getOutputStream());

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* 预览缩略图

\*/

public static void thumbnail(String path, HttpServletResponse response) {

// 如果是图片并且缩略图不存在则生成

File file = new File(UPLOAD\_FILE\_DIR, path);

File smFile = new File(UPLOAD\_SM\_DIR, path);

if (!smFile.exists() && isImgFile(file)) {

// 大于100kb生成100kb的缩略图

long fileSize = file.length();

if ((fileSize / 1024) > 100) {

try {

if (smFile.getParentFile().mkdirs()) {

ImgUtil.scale(file, smFile, 100f / (fileSize / 1024f));

}

} catch (Exception e) {

e.printStackTrace();

}

} else {

preview(file, response);

return;

}

}

preview(smFile, response);

}

/\*\*

\* 输出文件流

\*/

public static void output(File file, OutputStream os) {

try {

output(new FileInputStream(file), os);

} catch (FileNotFoundException e) {

e.printStackTrace();

}

}

public static void output(InputStream in, OutputStream os) {

BufferedInputStream is = null;

try {

is = new BufferedInputStream(in);

byte[] bytes = new byte[1024 \* 256];

int len;

while ((len = is.read(bytes)) != -1) {

os.write(bytes, 0, len);

}

os.flush();

} catch (Exception e) {

e.printStackTrace();

} finally {

if (os != null) {

try {

os.close();

} catch (IOException e) {

e.printStackTrace();

}

}

if (is != null) {

try {

is.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

/\*\*

\* 删除文件

\*/

public static boolean delete(String path) {

File file = new File(UPLOAD\_FILE\_DIR, path);

if (file.delete()) {

return new File(UPLOAD\_SM\_DIR, path).delete();

}

return false;

}

/\*\*

\* 获取文件列表

\*/

public static List<Map<String, Object>> list(String dir) {

dir = dir == null ? "" : dir;

List<Map<String, Object>> list = new ArrayList<>();

File file = new File(UPLOAD\_FILE\_DIR + dir);

File[] files = file.listFiles();

if (files == null) return list;

for (File f : files) {

Map<String, Object> map = new HashMap<>();

map.put("name", f.getName());

map.put("size", f.length());

map.put("isDir", f.isDirectory());

if (!f.isDirectory()) {

map.put("url", dir + "/" + f.getName());

map.put("smUrl", "thumbnail" + dir + "/" + f.getName());

}

map.put("updateTime", f.lastModified());

list.add(map);

}

return list;

}

/\*\*

\* 获取文件类型

\*/

public static String getFileType(File file) {

String contentType = null;

try {

contentType = new Tika().detect(file);

} catch (IOException e) {

e.printStackTrace();

}

return contentType;

}

/\*\*

\* 判断是否是图片类型

\*/

public static boolean isImgFile(File file) {

String contentType = getFileType(file);

return contentType != null && contentType.startsWith("image/");

}

/\*\*

\* 设置下载文件的header

\*/

public static void setDownloadHeader(HttpServletResponse response, String fileName) {

response.setContentType("application/force-download");

try {

fileName = URLEncoder.encode(fileName, "utf-8");

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

}

response.setHeader("Content-Disposition", "attachment;fileName=" + fileName);

}

/\*\*

\* 输出文件不存在

\*/

public static void outNotFund(HttpServletResponse response) {

response.setStatus(HttpServletResponse.SC\_NOT\_FOUND);

outMessage("404 Not Found", response);

}

/\*\*

\* 输出错误信息

\*/

public static void outMessage(String message, HttpServletResponse response) {

response.setContentType("text/html;charset=UTF-8");

try {

PrintWriter writer = response.getWriter();

writer.write("<!doctype html>");

writer.write("<title>" + message + "</title>");

writer.write("<h1 style=\"text-align: center\">" + message + "</h1>");

writer.write("<hr/><p style=\"text-align: center\">Egao File Server</p>");

writer.flush();

} catch (IOException e) {

e.printStackTrace();

}

}

}

public enum LoginType {

USER("User"), ADMIN("Admin");

private String type;

private LoginType(String type) {

this.type = type;

}

@Override

public String toString() {

return this.type.toString();

}

}

**com.egao.common.web:**

public class BaseController {

/\*\*

\* 获取当前登录的user

\*/

public User getLoginUser() {

Subject subject = SecurityUtils.getSubject();

if (subject == null) return null;

Object object = subject.getPrincipal();

if (object != null) return (User) object;

return null;

}

/\*\*

\* 获取当前登录的userId

\*/

public Integer getLoginUserId() {

User loginUser = getLoginUser();

return loginUser == null ? null : loginUser.getUserId();

}

}

public class BatchParam<T> implements Serializable {

private static final long serialVersionUID = 1L;

/\*\*

\* 主键id集合

\*/

private List<Serializable> ids;

/\*\*

\* 批量修改多个字段

\*/

private T data;

/\*\*

\* 通用批量修改方法

\*

\* @param service IService

\* @param idName id字段名称

\* @return boolean

\*/

public boolean update(IService<T> service, String idName) {

if (data == null) return false;

return service.update(data, new UpdateWrapper<T>().in(idName, this.getIds()));

}

public List<Serializable> getIds() {

return ids;

}

public void setIds(List<Serializable> ids) {

this.ids = ids;

}

public T getData() {

return data;

}

public void setData(T data) {

this.data = data;

}

}

public class JsonResult extends HashMap<String, Object> {

private static final long serialVersionUID = 1L;

private static final String CODE\_NAME = "code"; // 状态码字段名称

private static final String MSG\_NAME = "msg"; // 状态信息字段名称

private static final String DATA\_NAME = "data"; // 数据字段名称

private static final int DEFAULT\_OK\_CODE = Constants.RESULT\_OK\_CODE; // 默认成功码

private static final int DEFAULT\_ERROR\_CODE = Constants.RESULT\_ERROR\_CODE; // 默认失败码

private static final String DEFAULT\_OK\_MSG = "操作成功"; // 默认成功msg

private static final String DEFAULT\_ERROR\_MSG = "操作失败"; // 默认失败msg

private JsonResult() {

}

/\*\*

\* 返回成功

\*/

public static JsonResult ok() {

return ok(null);

}

/\*\*

\* 返回成功

\*/

public static JsonResult ok(String message) {

return ok(DEFAULT\_OK\_CODE, message);

}

/\*\*

\* 返回成功

\*/

public static JsonResult ok(int code, String message) {

if (message == null) message = DEFAULT\_OK\_MSG;

JsonResult jsonResult = new JsonResult();

jsonResult.put(CODE\_NAME, code);

jsonResult.put(MSG\_NAME, message);

return jsonResult;

}

/\*\*

\* 返回失败

\*/

public static JsonResult error() {

return error(null);

}

/\*\*

\* 返回失败

\*/

public static JsonResult error(String message) {

return error(DEFAULT\_ERROR\_CODE, message);

}

/\*\*

\* 返回失败

\*/

public static JsonResult error(int code, String message) {

if (message == null) message = DEFAULT\_ERROR\_MSG;

return ok(code, message);

}

public JsonResult setCode(Integer code) {

return put(CODE\_NAME, code);

}

public JsonResult setMsg(String message) {

return put(MSG\_NAME, message);

}

public JsonResult setData(Object object) {

return put(DATA\_NAME, object);

}

public Integer getCode(int code) {

Object o = get(CODE\_NAME);

return o == null ? null : Integer.parseInt(o.toString());

}

public String getMsg() {

Object o = get(MSG\_NAME);

return o == null ? null : o.toString();

}

public Object getData() {

return get(DATA\_NAME);

}

@Override

public JsonResult put(String key, Object object) {

if (key != null && object != null) super.put(key, object);

return this;

}

}

public class PageParam<T> extends Page<T> {

private static final String FILED\_PAGE = "page"; // 第几页参数名称

private static final String FILED\_LIMIT = "limit"; // 每页显示数量参数名称

private static final String FILED\_SORT = "sort"; // 排序字段参数名称

private static final String FILED\_ORDER = "order"; // 排序方式参数名称

private static final String VALUE\_ORDER\_ASC = "asc"; // 表示升序的值

private static final String VALUE\_ORDER\_DESC = "desc"; // 表示降序的值

private static final Pattern HUMP\_PATTERN = Pattern.compile("[A-Z]"); // 驼峰转下划线正则匹配

/\*\*

\* 除分页、排序外的其他参数

\*/

private Map<String, Object> pageData;

/\*\*

\* 是否把字段名称驼峰转下划线

\*/

private boolean needToLine = true;

public PageParam() {

super();

}

public PageParam(HttpServletRequest request) {

init(request);

}

public PageParam(HttpServletRequest request, boolean needToLine) {

setNeedToLine(needToLine);

init(request);

}

public boolean isNeedToLine() {

return needToLine;

}

public PageParam<T> setNeedToLine(boolean needToLine) {

this.needToLine = needToLine;

return this;

}

public Map<String, Object> getPageData() {

return pageData;

}

public void setPageData(Map<String, Object> data) {

this.pageData = data;

}

/\*\*

\* 从request中获取参数并填充到PageParam中

\*/

public PageParam<T> init(HttpServletRequest request) {

String sortValue = null, orderValue = null;

Map<String, Object> map = new HashMap<>();

Enumeration<String> names = request.getParameterNames();

while (names.hasMoreElements()) {

String name = names.nextElement();

String value = request.getParameter(name);

if (value == null || value.isEmpty() || value.replace(" ", "").isEmpty()) {

continue;

}

if (FILED\_PAGE.equals(name)) {

setCurrent(Long.parseLong(value));

} else if (FILED\_LIMIT.equals(name)) {

setSize(Long.parseLong(value));

} else if (FILED\_SORT.equals(name)) {

sortValue = (needToLine ? humpToLine(value) : value);

} else if (FILED\_ORDER.equals(name)) {

orderValue = value;

} else {

map.put(name, value);

}

}

setPageData(map);

// 同步排序方式到MyBatisPlus中

if (sortValue != null) {

if (VALUE\_ORDER\_DESC.equals(orderValue)) {

setOrder(sortValue, false);

} else/\* if (VALUE\_ORDER\_ASC.equals(orderValue))\*/ {

setOrder(sortValue, true);

}

}

return this;

}

/\*\*

\* 获取升序排序的字段

\*/

public List<String> getOrderAscs() {

List<String> ascs = new ArrayList<>();

List<OrderItem> orders = getOrders();

if (orders != null) {

for (OrderItem order : orders) {

if (order.isAsc()) ascs.add(order.getColumn());

}

}

return ascs;

}

/\*\*

\* 获取降序排序的字段

\*/

public List<String> getOrderDescs() {

List<String> descs = new ArrayList<>();

List<OrderItem> orders = getOrders();

if (orders != null) {

for (OrderItem order : orders) {

if (!order.isAsc()) descs.add(order.getColumn());

}

}

return descs;

}

/\*\*

\* 增加asc排序方式

\*/

public PageParam<T> addOrderAsc(String... ascs) {

if (ascs != null) {

List<String> tAscs = getOrderAscs();

List<OrderItem> orderItems = new ArrayList<>();

for (String column : ascs) {

if (!tAscs.contains(column)) {

OrderItem orderItem = new OrderItem();

orderItem.setAsc(true);

orderItem.setColumn(column);

orderItems.add(orderItem);

}

}

if (getOrders() == null) {

setOrders(orderItems);

} else {

getOrders().addAll(orderItems);

}

}

return this;

}

/\*\*

\* 增加desc排序方式

\*/

public PageParam<T> addOrderDesc(String... descs) {

if (descs != null) {

List<String> tDescs = getOrderDescs();

List<OrderItem> orderItems = new ArrayList<>();

for (String column : descs) {

if (!tDescs.contains(column)) {

OrderItem orderItem = new OrderItem();

orderItem.setAsc(false);

orderItem.setColumn(column);

orderItems.add(orderItem);

}

}

if (getOrders() == null) {

setOrders(orderItems);

} else {

getOrders().addAll(orderItems);

}

}

return this;

}

/\*\*

\* 移除某个排序字段

\*/

public PageParam<T> removeOrder(String order, Boolean isAsc) {

List<OrderItem> orderItems = getOrders();

if (orderItems != null) {

Iterator<OrderItem> iterator = orderItems.iterator();

while (iterator.hasNext()) {

OrderItem item = iterator.next();

if (isAsc == null || isAsc == item.isAsc()) {

if (order.equals(item.getColumn())) iterator.remove();

}

}

}

return this;

}

/\*\*

\* 设置排序方式

\*/

public PageParam<T> setOrder(String order, boolean isAsc) {

List<OrderItem> orderItems = new ArrayList<>();

if (order != null) {

OrderItem orderItem = new OrderItem();

orderItem.setAsc(isAsc);

orderItem.setColumn(order);

orderItems.add(orderItem);

}

setOrders(orderItems);

return this;

}

/\*\*

\* 设置默认排序方式

\*/

public PageParam<T> setDefaultOrder(String[] ascs, String[] descs) {

List<OrderItem> orderItems = getOrders();

if (orderItems == null || orderItems.size() == 0) {

addOrderAsc(ascs);

addOrderDesc(descs);

}

return this;

}

/\*\*

\* 往pageData里面增加参数

\*/

public PageParam<T> put(String key, Object value) {

if (pageData == null) pageData = new HashMap<>();

pageData.put(key, value);

return this;

}

/\*\*

\* 获取pageData里面参数

\*/

public Object get(String key) {

return pageData.get(key);

}

public String getString(String key) {

Object o = pageData.get(key);

if (o != null) return String.valueOf(o);

return null;

}

public Integer getInt(String key) {

String str = getString(key);

if (str != null) return Integer.parseInt(str);

return null;

}

public Long getLong(String key) {

String str = getString(key);

if (str != null) return Long.parseLong(str);

return null;

}

public Float getFloat(String key) {

String str = getString(key);

if (str != null) return Float.parseFloat(str);

return null;

}

public Double getDouble(String key) {

String str = getString(key);

if (str != null) return Double.parseDouble(str);

return null;

}

public Boolean getBoolean(String key) {

String str = getString(key);

if (str != null) return Boolean.parseBoolean(str);

return null;

}

/\*\*

\* 构建查询条件

\*

\* @param excludes 不包含的参数

\* @return QueryWrapper

\*/

public QueryWrapper<T> getWrapper(String... excludes) {

List<String> exList = Arrays.asList(excludes);

QueryWrapper<T> queryWrapper = new QueryWrapper<T>();

for (String key : pageData.keySet()) {

if (!exList.contains(key)) buildWrapper(queryWrapper, key, getString(key));

}

return queryWrapper;

}

/\*\*

\* 构建查询条件

\*

\* @param columns 只包含的参数

\* @return QueryWrapper

\*/

public QueryWrapper<T> getWrapperWith(String... columns) {

List<String> keyList = Arrays.asList(columns);

QueryWrapper<T> queryWrapper = new QueryWrapper<T>();

for (String key : pageData.keySet()) {

if (keyList.contains(key)) buildWrapper(queryWrapper, key, getString(key));

}

return queryWrapper;

}

/\*\*

\* 逐个构建QueryWrapper

\*/

private void buildWrapper(QueryWrapper<T> queryWrapper, String key, String value) {

if (value == null || "deleted".equals(key)) return;

if (Arrays.asList("id", "sortNumber", "state").contains(key) || key.endsWith("Id")) {

queryWrapper.eq(needToLine ? humpToLine(key) : key, value);

} else if ("createTimeStart".equals(key)) {

queryWrapper.ge("create\_time", value);

} else if ("createTimeEnd".equals(key)) {

queryWrapper.le("create\_time", value);

} else {

queryWrapper.like(needToLine ? humpToLine(key) : key, value);

}

}

/\*\*

\* 构建排序的QueryWrapper

\*

\* @param queryWrapper 搜索条件的wrapper

\* @return QueryWrapper

\*/

public QueryWrapper<T> getOrderWrapper(QueryWrapper<T> queryWrapper) {

if (queryWrapper == null) queryWrapper = new QueryWrapper<T>();

queryWrapper.orderByAsc(getOrderAscs().toArray(new String[0]));

queryWrapper.orderByDesc(getOrderDescs().toArray(new String[0]));

return queryWrapper;

}

/\*\*

\* 构建包含排序的查询条件

\*

\* @return QueryWrapper

\*/

public QueryWrapper<T> getOrderWrapper() {

return getOrderWrapper(getWrapper());

}

/\*\*

\* 获取不分页的参数，包含排序和pageData

\*

\* @return Map

\*/

public Map<String, Object> getNoPageParam() {

Map<String, Object> map = new HashMap<>();

map.put("pageData", this.pageData);

List<OrderItem> orders = getOrders();

if (orders != null && orders.size() > 0) map.put("orders", orders);

return map;

}

/\*\*

\* 获取查询结果中第一条数据

\*/

public T getOne(List<T> records) {

return records == null || records.size() == 0 ? null : records.get(0);

}

/\*\*

\* 代码排序查询结果，实现类似SQL语句的排序效果

\*/

public List<T> sortRecords(List<T> records) {

if (records == null || records.size() <= 1) return records;

List<OrderItem> orderItems = getOrders();

if (orderItems != null) {

Comparator<T> comparator = null;

for (OrderItem order : orderItems) {

if (order.getColumn() != null) {

Function keyExtractor = t -> getFieldValue(t, needToLine ? lineToHump(order.getColumn()) : order.getColumn());

if (comparator == null) {

if (order.isAsc()) {

comparator = Comparator.comparing(keyExtractor);

} else {

comparator = Comparator.comparing(keyExtractor, Comparator.reverseOrder());

}

} else {

if (order.isAsc()) {

comparator.thenComparing(keyExtractor);

} else {

comparator.thenComparing(keyExtractor, Comparator.reverseOrder());

}

}

}

}

if (comparator != null) {

return records.stream().sorted(comparator).collect(Collectors.toList());

}

}

return records;

}

/\*\*

\* 驼峰转下划线

\*/

public static String humpToLine(String str) {

if (str == null) return null;

Matcher matcher = HUMP\_PATTERN.matcher(str);

StringBuffer sb = new StringBuffer();

while (matcher.find()) {

matcher.appendReplacement(sb, "\_" + matcher.group(0).toLowerCase());

}

matcher.appendTail(sb);

return sb.toString();

}

/\*\*

\* 下划线转驼峰

\*/

public static String lineToHump(String str) {

if (str == null) return null;

StringBuilder sb = new StringBuilder();

String[] ss = str.split("\_");

sb.append(ss[0]);

for (int i = 1; i < ss.length; i++) {

if (ss[i].length() > 0) sb.append(ss[i].substring(0, 1).toUpperCase());

if (ss[i].length() > 1) sb.append(ss[i].substring(1));

}

return sb.toString();

}

/\*\*

\* 获取对象某个字段的值

\*

\* @param t 对象

\* @param field 字段

\* @return Object

\*/

public static Object getFieldValue(Object t, String field) {

if (t == null || field == null) return null;

try {

Field clazzField = t.getClass().getDeclaredField(field);

clazzField.setAccessible(true);

return clazzField.get(t);

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

}

public class PageResult<T> implements Serializable {

private static final long serialVersionUID = 1L;

/\*\*

\* 状态码

\*/

private int code = 0;

/\*\*

\* 提示信息

\*/

private String msg;

/\*\*

\* 总数量

\*/

private long count;

/\*\*

\* 当前页数据

\*/

private List<T> data;

public PageResult() {

}

public PageResult(List<T> rows) {

this(rows, rows == null ? 0 : rows.size());

}

public PageResult(List<T> rows, long total) {

this.count = total;

this.data = rows;

}

public PageResult(int code, List<T> rows, long total) {

this.code = code;

this.count = total;

this.data = rows;

}

public int getCode() {

return code;

}

public PageResult setCode(int code) {

this.code = code;

return this;

}

public String getMsg() {

return msg;

}

public PageResult setMsg(String msg) {

this.msg = msg;

return this;

}

public long getCount() {

return count;

}

public PageResult setCount(long count) {

this.count = count;

return this;

}

public List<T> getData() {

return data;

}

public PageResult setData(List<T> data) {

this.data = data;

return this;

}

}

**com.egao.common.system.controller:**

@Controller

@RequestMapping("/file")

public class FileController {

@RequiresPermissions("sys:file:view")

@RequestMapping("/manage")

public String view() {

return "system/file.html";

}

/\*\*

\* 上传文件

\*/

@OperLog(value = "文件管理", desc = "上传文件", param = false, result = true)

@ResponseBody

@PostMapping("/upload")

public JsonResult upload(@RequestParam MultipartFile file) {

return FileUploadUtil.upload(file);

}

/\*\*

\* 上传base64文件

\*/

@OperLog(value = "文件管理", desc = "上传base64文件", param = false, result = true)

@ResponseBody

@PostMapping("/upload/base64")

public JsonResult uploadBase64(String base64) {

return FileUploadUtil.upload(base64);

}

/\*\*

\* 预览文件

\*/

@GetMapping("/{dir}/{name:.+}")

public void file(@PathVariable("dir") String dir, @PathVariable("name") String name, HttpServletResponse response) {

FileUploadUtil.preview(dir + "/" + name, response);

}

/\*\*

\* 下载文件

\*/

@GetMapping("/download/{dir}/{name:.+}")

public void downloadFile(@PathVariable("dir") String dir, @PathVariable("name") String name, HttpServletResponse response) {

FileUploadUtil.download(dir + "/" + name, response);

}

/\*\*

\* 查看缩略图

\*/

@GetMapping("/thumbnail/{dir}/{name:.+}")

public void smFile(@PathVariable("dir") String dir, @PathVariable("name") String name, HttpServletResponse response) {

FileUploadUtil.thumbnail(dir + "/" + name, response);

}

/\*\*

\* 删除文件

\*/

@OperLog(value = "文件管理", desc = "删除文件", result = true)

@RequiresPermissions("sys:file:remove")

@ResponseBody

@RequestMapping("/remove")

public JsonResult remove(String path) {

if (path == null || path.trim().isEmpty()) {

return JsonResult.error("参数不能为空");

}

if (FileUploadUtil.delete(path)) {

return JsonResult.ok("删除成功");

}

return JsonResult.error("删除失败");

}

/\*\*

\* 查询文件列表

\*/

@OperLog(value = "文件管理", desc = "查询全部")

@RequiresPermissions("sys:file:list")

@ResponseBody

@RequestMapping("/list")

public JsonResult list(String dir) {

List<Map<String, Object>> list = FileUploadUtil.list(dir);

list.sort(new Comparator<Map<String, Object>>() {

@Override

public int compare(Map<String, Object> o1, Map<String, Object> o2) {

return ((Long) o2.get("updateTime")).compareTo((Long) o1.get("updateTime"));

}

});

return JsonResult.ok().setData(list);

}

}

@Controller

@RequestMapping("/sys/loginRecord")

public class LoginRecordController extends BaseController {

@Autowired

private LoginRecordService loginRecordService;

@RequiresPermissions("sys:login\_record:view")

@RequestMapping()

public String view() {

return "system/login-record.html";

}

/\*\*

\* 分页查询登录日志

\*/

@OperLog(value = "登录日志", desc = "分页查询")

@RequiresPermissions("sys:login\_record:view")

@ResponseBody

@RequestMapping("/page")

public PageResult<LoginRecord> page(HttpServletRequest request) {

PageParam<LoginRecord> pageParam = new PageParam<>(request);

pageParam.setDefaultOrder(null, new String[]{"create\_time"});

return loginRecordService.listPage(pageParam);

}

/\*\*

\* 查询全部登录日志

\*/

@OperLog(value = "登录日志", desc = "查询全部")

@RequiresPermissions("sys:login\_record:view")

@ResponseBody

@RequestMapping("/list")

public JsonResult list(HttpServletRequest request) {

PageParam<LoginRecord> pageParam = new PageParam<>(request);

List<LoginRecord> records = loginRecordService.listAll(pageParam.getNoPageParam());

return JsonResult.ok().setData(pageParam.sortRecords(records));

}

/\*\*

\* 根据id查询登录日志

\*/

@OperLog(value = "登录日志", desc = "根据id查询")

@RequiresPermissions("sys:login\_record:view")

@ResponseBody

@RequestMapping("/get")

public JsonResult get(Integer id) {

PageParam<LoginRecord> pageParam = new PageParam<>();

pageParam.put("id", id);

List<LoginRecord> records = loginRecordService.listAll(pageParam.getNoPageParam());

return JsonResult.ok().setData(pageParam.getOne(records));

}

}

@Controller

public class MainController extends BaseController implements ErrorController {

@Autowired

private MenuService menuService;

@Autowired

private LoginRecordService loginRecordService;

/\*\*

\* 用户登录

\*/

@ResponseBody

@PostMapping("/login")

public JsonResult login(String username, String password, String code, Boolean remember, HttpServletRequest request) {

if (username == null || username.trim().isEmpty()) return JsonResult.error("请输入账号");

if (!CaptchaUtil.ver(code, request)) {

loginRecordService.saveAsync(username, LoginRecord.TYPE\_ERROR, "验证码错误", request);

return JsonResult.error("验证码不正确");

}

try {

if (remember == null) remember = false;

SecurityUtils.getSubject().login(new CustomizedToken(username, password, remember,"Admin"));

loginRecordService.saveAsync(username, request);

return JsonResult.ok("登录成功");

} catch (IncorrectCredentialsException ice) {

loginRecordService.saveAsync(username, LoginRecord.TYPE\_ERROR, "密码错误", request);

return JsonResult.error("密码错误");

} catch (UnknownAccountException uae) {

loginRecordService.saveAsync(username, LoginRecord.TYPE\_ERROR, "账号不存在", request);

return JsonResult.error("账号不存在");

} catch (LockedAccountException e) {

loginRecordService.saveAsync(username, LoginRecord.TYPE\_ERROR, "账号被锁定", request);

return JsonResult.error("账号被锁定");

} catch (ExcessiveAttemptsException eae) {

loginRecordService.saveAsync(username, LoginRecord.TYPE\_ERROR, "操作频繁", request);

return JsonResult.error("操作频繁，请稍后再试");

}

}

/\*\*

\* 登录页

\*/

@GetMapping("/login")

public String login() {

if (getLoginUser() != null) return "redirect:index";

return "login.html";

}

/\*\*

\* 主页

\*/

@RequestMapping({"/", "/index"})

public String index(Model model) {

// 左侧菜单

List<Menu> menus = menuService.getUserMenu(getLoginUserId(), Menu.TYPE\_MENU);

model.addAttribute("menus", menuService.toMenuTree(menus, 0));

return "index.html";

}

/\*\*

\* 图形验证码

\*/

@RequestMapping("/assets/captcha")

public void captcha(HttpServletRequest request, HttpServletResponse response) {

try {

CaptchaUtil.out(5, request, response);

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* 主页弹窗页面

\*/

@RequestMapping("/tpl/{name}")

public String tpl(@PathVariable("name") String name) {

return "index/" + name + ".html";

}

/\*\*

\* 错误页

\*/

@RequestMapping("/error")

public String error() {

return "error/404.html";

}

@Override

public String getErrorPath() {

return "/error";

}

}

com.egao.common.system.entity:

@TableName("sys\_user")

public class User implements Serializable {

private static final long serialVersionUID = 242146703513492331L;

/\*\*

\* 用户id

\*/

@TableId(value = "user\_id", type = IdType.AUTO)

private Integer userId;

/\*\*

\* 账号

\*/

private String username;

/\*\*

\* 密码

\*/

private String password;

/\*\*

\* 昵称

\*/

private String nickName;

/\*\*

\* 头像

\*/

private String avatar;

/\*\*

\* 性别

\*/

private Integer sex;

/\*\*

\* 手机号

\*/

private String phone;

/\*\*

\* 邮箱

\*/

private String email;

/\*\*

\* 邮箱是否验证,0否,1是

\*/

private Integer emailVerified;

/\*\*

\* 真实姓名

\*/

private String trueName;

/\*\*

\* 身份证号

\*/

private String idCard;

/\*\*

\* 出生日期

\*/

private Date birthday;

/\*\*

\* 个人简介

\*/

private String introduction;

/\*\*

\* 机构id

\*/

private Integer organizationId;

/\*\*

\* 状态，0正常，1冻结

\*/

private Integer state;

/\*\*

\* 注册时间

\*/

private Date createTime;

/\*\*

\* 修改时间

\*/

private Date updateTime;

/\*\*

\* 是否删除,0否,1是

\*/

@TableLogic

private Integer deleted;

/\*\*

\* 权限列表

\*/

@TableField(exist = false)

private List<String> authorities;

/\*\*

\* 角色列表

\*/

@TableField(exist = false)

private List<Role> roles;

/\*\*

\* 角色id

\*/

@TableField(exist = false)

private List<Integer> roleIds;

/\*\*

\* 机构名称

\*/

@TableField(exist = false)

private String organizationName;

/\*\*

\* 性别名称

\*/

@TableField(exist = false)

private String sexName;

public Integer getUserId() {

return userId;

}

public void setUserId(Integer userId) {

this.userId = userId;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getNickName() {

return nickName;

}

public void setNickName(String nickName) {

this.nickName = nickName;

}

public String getAvatar() {

return avatar;

}

public void setAvatar(String avatar) {

this.avatar = avatar;

}

public Integer getSex() {

return sex;

}

public void setSex(Integer sex) {

this.sex = sex;

}

public String getPhone() {

return phone;

}

public void setPhone(String phone) {

this.phone = phone;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public Integer getEmailVerified() {

return emailVerified;

}

public void setEmailVerified(Integer emailVerified) {

this.emailVerified = emailVerified;

}

public String getTrueName() {

return trueName;

}

public void setTrueName(String trueName) {

this.trueName = trueName;

}

public String getIdCard() {

return idCard;

}

public void setIdCard(String idCard) {

this.idCard = idCard;

}

public Date getBirthday() {

return birthday;

}

public void setBirthday(Date birthday) {

this.birthday = birthday;

}

public String getIntroduction() {

return introduction;

}

public void setIntroduction(String introduction) {

this.introduction = introduction;

}

public Integer getOrganizationId() {

return organizationId;

}

public void setOrganizationId(Integer organizationId) {

this.organizationId = organizationId;

}

public Integer getState() {

return state;

}

public void setState(Integer state) {

this.state = state;

}

public Date getCreateTime() {

return createTime;

}

public void setCreateTime(Date createTime) {

this.createTime = createTime;

}

public Date getUpdateTime() {

return updateTime;

}

public void setUpdateTime(Date updateTime) {

this.updateTime = updateTime;

}

public Integer getDeleted() {

return deleted;

}

public void setDeleted(Integer deleted) {

this.deleted = deleted;

}

public List<String> getAuthorities() {

return authorities;

}

public void setAuthorities(List<String> authorities) {

this.authorities = authorities;

}

public List<Role> getRoles() {

return roles;

}

public void setRoles(List<Role> roles) {

this.roles = roles;

}

public List<Integer> getRoleIds() {

return roleIds;

}

public void setRoleIds(List<Integer> roleIds) {

this.roleIds = roleIds;

}

public String getOrganizationName() {

return organizationName;

}

public void setOrganizationName(String organizationName) {

this.organizationName = organizationName;

}

public String getSexName() {

return sexName;

}

public void setSexName(String sexName) {

this.sexName = sexName;

}

@Override

public String toString() {

return "User{" +

", userId=" + userId +

", username=" + username +

", password=" + password +

", nickName=" + nickName +

", avatar=" + avatar +

", sex=" + sex +

", phone=" + phone +

", email=" + email +

", emailVerified=" + emailVerified +

", trueName=" + trueName +

", idCard=" + idCard +

", birthday=" + birthday +

", introduction=" + introduction +

", organizationId=" + organizationId +

", state=" + state +

", createTime=" + createTime +

", updateTime=" + updateTime +

", deleted=" + deleted +

", organizationName=" + organizationName +

"}";

}

}

@TableName("sys\_user\_role")

public class UserRole implements Serializable {

private static final long serialVersionUID = 1L;

/\*\*

\* 主键id

\*/

@TableId(value = "id", type = IdType.AUTO)

private Integer id;

/\*\*

\* 用户id

\*/

private Integer userId;

/\*\*

\* 角色id

\*/

private Integer roleId;

/\*\*

\* 创建时间

\*/

private Date createTime;

/\*\*

\* 修改时间

\*/

private Date updateTime;

/\*\*

\* 角色名称

\*/

@TableField(exist = false)

private String roleName;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public Integer getUserId() {

return userId;

}

public void setUserId(Integer userId) {

this.userId = userId;

}

public Integer getRoleId() {

return roleId;

}

public void setRoleId(Integer roleId) {

this.roleId = roleId;

}

public Date getCreateTime() {

return createTime;

}

public void setCreateTime(Date createTime) {

this.createTime = createTime;

}

public Date getUpdateTime() {

return updateTime;

}

public void setUpdateTime(Date updateTime) {

this.updateTime = updateTime;

}

public String getRoleName() {

return roleName;

}

public void setRoleName(String roleName) {

this.roleName = roleName;

}

@Override

public String toString() {

return "UserRole{" +

", id=" + id +

", userId=" + userId +

", roleId=" + roleId +

", createTime=" + createTime +

", updateTime=" + updateTime +

", roleName=" + roleName +

"}";

}

}

@TableName("sys\_user")

public class User implements Serializable {

private static final long serialVersionUID = 242146703513492331L;

/\*\*

\* 用户id

\*/

@TableId(value = "user\_id", type = IdType.AUTO)

private Integer userId;

/\*\*

\* 账号

\*/

private String username;

/\*\*

\* 密码

\*/

private String password;

/\*\*

\* 昵称

\*/

private String nickName;

/\*\*

\* 头像

\*/

private String avatar;

/\*\*

\* 性别

\*/

private Integer sex;

/\*\*

\* 手机号

\*/

private String phone;

/\*\*

\* 邮箱

\*/

private String email;

/\*\*

\* 邮箱是否验证,0否,1是

\*/

private Integer emailVerified;

/\*\*

\* 真实姓名

\*/

private String trueName;

/\*\*

\* 身份证号

\*/

private String idCard;

/\*\*

\* 出生日期

\*/

private Date birthday;

/\*\*

\* 个人简介

\*/

private String introduction;

/\*\*

\* 机构id

\*/

private Integer organizationId;

/\*\*

\* 状态，0正常，1冻结

\*/

private Integer state;

/\*\*

\* 注册时间

\*/

private Date createTime;

/\*\*

\* 修改时间

\*/

private Date updateTime;

/\*\*

\* 是否删除,0否,1是

\*/

@TableLogic

private Integer deleted;

/\*\*

\* 权限列表

\*/

@TableField(exist = false)

private List<String> authorities;

/\*\*

\* 角色列表

\*/

@TableField(exist = false)

private List<Role> roles;

/\*\*

\* 角色id

\*/

@TableField(exist = false)

private List<Integer> roleIds;

/\*\*

\* 机构名称

\*/

@TableField(exist = false)

private String organizationName;

/\*\*

\* 性别名称

\*/

@TableField(exist = false)

private String sexName;

public Integer getUserId() {

return userId;

}

public void setUserId(Integer userId) {

this.userId = userId;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getNickName() {

return nickName;

}

public void setNickName(String nickName) {

this.nickName = nickName;

}

public String getAvatar() {

return avatar;

}

public void setAvatar(String avatar) {

this.avatar = avatar;

}

public Integer getSex() {

return sex;

}

public void setSex(Integer sex) {

this.sex = sex;

}

public String getPhone() {

return phone;

}

public void setPhone(String phone) {

this.phone = phone;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public Integer getEmailVerified() {

return emailVerified;

}

public void setEmailVerified(Integer emailVerified) {

this.emailVerified = emailVerified;

}

public String getTrueName() {

return trueName;

}

public void setTrueName(String trueName) {

this.trueName = trueName;

}

public String getIdCard() {

return idCard;

}

public void setIdCard(String idCard) {

this.idCard = idCard;

}

public Date getBirthday() {

return birthday;

}

public void setBirthday(Date birthday) {

this.birthday = birthday;

}

public String getIntroduction() {

return introduction;

}

public void setIntroduction(String introduction) {

this.introduction = introduction;

}

public Integer getOrganizationId() {

return organizationId;

}

public void setOrganizationId(Integer organizationId) {

this.organizationId = organizationId;

}

public Integer getState() {

return state;

}

public void setState(Integer state) {

this.state = state;

}

public Date getCreateTime() {

return createTime;

}

public void setCreateTime(Date createTime) {

this.createTime = createTime;

}

public Date getUpdateTime() {

return updateTime;

}

public void setUpdateTime(Date updateTime) {

this.updateTime = updateTime;

}

public Integer getDeleted() {

return deleted;

}

public void setDeleted(Integer deleted) {

this.deleted = deleted;

}

public List<String> getAuthorities() {

return authorities;

}

public void setAuthorities(List<String> authorities) {

this.authorities = authorities;

}

public List<Role> getRoles() {

return roles;

}

public void setRoles(List<Role> roles) {

this.roles = roles;

}

public List<Integer> getRoleIds() {

return roleIds;

}

public void setRoleIds(List<Integer> roleIds) {

this.roleIds = roleIds;

}

public String getOrganizationName() {

return organizationName;

}

public void setOrganizationName(String organizationName) {

this.organizationName = organizationName;

}

public String getSexName() {

return sexName;

}

public void setSexName(String sexName) {

this.sexName = sexName;

}

@Override

public String toString() {

return "User{" +

", userId=" + userId +

", username=" + username +

", password=" + password +

", nickName=" + nickName +

", avatar=" + avatar +

", sex=" + sex +

", phone=" + phone +

", email=" + email +

", emailVerified=" + emailVerified +

", trueName=" + trueName +

", idCard=" + idCard +

", birthday=" + birthday +

", introduction=" + introduction +

", organizationId=" + organizationId +

", state=" + state +

", createTime=" + createTime +

", updateTime=" + updateTime +

", deleted=" + deleted +

", organizationName=" + organizationName +

"}";

}

}

com.egao.common.system.mapper:

public interface UserMapper extends BaseMapper<User> {

/\*\*

\* 分页查询

\*/

List<User> listPage(@Param("page") PageParam<User> page);

/\*\*

\* 查询全部

\*/

List<User> listAll(@Param("page") Map<String, Object> page);

}

public interface UserRoleMapper extends BaseMapper<UserRole> {

/\*\*

\* 批量添加用户角色

\*/

int insertBatch(@Param("userId") Integer userId, @Param("roleIds") List<Integer> roleIds);

/\*\*

\* 批量查询用户角色

\*/

List<Role> listByUserIds(@Param("userIds") List<Integer> userIds);

/\*\*

\* 查询某个用户的角色

\*/

List<Role> listByUserId(@Param("userId") Integer userId);

}

public interface MenuMapper extends BaseMapper<Menu> {

/\*\*

\* 分页查询

\*/

List<Menu> listPage(@Param("page") PageParam<Menu> page);

/\*\*

\* 根据用户id查询

\*/

List<Menu> listByUserId(@Param("userId") Integer userId, @Param("menuType") Integer menuType);

}

**com.egao.common.system.service:**

public interface UserService extends IService<User> {

/\*\*

\* 根据账号查询用户

\*/

User getByUsername(String username);

/\*\*

\* 根据id查询用户(关联查询)

\*/

User getFullById(Integer userId);

/\*\*

\* 查询用户角色和权限

\*/

User selectRoleAndAuth(User user);

/\*\*

\* 关联分页查询用户

\*/

PageResult<User> listPage(PageParam<User> page);

/\*\*

\* 关联查询全部用户

\*/

List<User> listAll(Map<String, Object> page);

/\*\*

\* 添加用户(包含角色)

\*/

boolean saveUser(User user);

/\*\*

\* 修改用户(包含角色)

\*/

boolean updateUser(User user);

/\*\*

\* 比较用户密码

\*

\* @param dbPsw 数据库存储的密码

\* @param inputPsw 用户输入的密码

\* @return boolean

\*/

boolean comparePsw(String dbPsw, String inputPsw);

/\*\*

\* md5加密用户密码

\*/

String encodePsw(String psw);

}

public interface UserRoleService extends IService<UserRole> {

/\*\*

\* 查询用户的角色id

\*/

Integer[] getRoleIds(String userId);

}

public interface MenuService extends IService<Menu> {

/\*\*

\* 关联分页查询菜单

\*/

PageResult<Menu> listPage(PageParam<Menu> pageParam);

/\*\*

\* 根据用户id查询菜单列表

\*

\* @param userId 用户id

\* @param menuType 菜单类型，为null不筛选

\* @return List<Menu>

\*/

List<Menu> getUserMenu(Integer userId, Integer menuType);

/\*\*

\* 转化为树形结构

\*

\* @param menus 菜单list

\* @param parentId 最顶级id

\* @return List<Menu>

\*/

List<Menu> toMenuTree(List<Menu> menus, Integer parentId);

}

**com.egao.common.test.controller:**

courseService;

@Autowired

private ClassifyService classifyService;

@Autowired

private TeacherService teacherService;

@RequestMapping()

public String view(Model model) {

model.addAttribute("classifyList", classifyService.list());

model.addAttribute("teacherList", teacherService.list());

return "course/course.html";

}

@RequestMapping("/ggkc")

public String ggkc() {

return "course/ggkc.html";

}

@RequestMapping("/zykc")

public String zykc() {

return "course/zykc.html";

}

@RequestMapping("/mfkc")

public String mfkc() {

return "course/mfkc.html";

}

/\*\*

\* 分页查询

\*/

@ResponseBody

@RequestMapping("/page")

public PageResult<Course> page(HttpServletRequest request) {

PageParam<Course> pageParam = new PageParam<>(request);

return new PageResult<>(courseService.page(pageParam, pageParam.getWrapper()).getRecords(), pageParam.getTotal());

}

/\*\*

\* 查询全部

\*/

@ResponseBody

@RequestMapping("/list")

public JsonResult list(HttpServletRequest request) {

PageParam<Course> pageParam = new PageParam<>(request);

return JsonResult.ok().setData(courseService.list(pageParam.getOrderWrapper()));

//List<Course> records = courseService.listAll(pageParam.getNoPageParam()); // 使用关联查询

//return JsonResult.ok().setData(pageParam.sortRecords(records));

}

/\*\*

\* 根据id查询

\*/

@ResponseBody

@RequestMapping("/get")

public JsonResult get(Integer id) {

return JsonResult.ok().setData(courseService.getById(id));

// 使用关联查询

//PageParam<Course> pageParam = new PageParam<>();

//pageParam.put("id", id);

//List<Course> records = courseService.listAll(pageParam.getNoPageParam());

//return JsonResult.ok().setData(pageParam.getOne(records));

}

/\*\*

\* 添加

\*/

@ResponseBody

@RequestMapping("/save")

public JsonResult save(Course course) {

course.setCreateTime(new Date());

if (courseService.save(course)) {

return JsonResult.ok("添加成功");

}

return JsonResult.error("添加失败");

}

/\*\*

\* 修改

\*/

@ResponseBody

@RequestMapping("/update")

public JsonResult update(Course course) {

if (courseService.updateById(course)) {

return JsonResult.ok("修改成功");

}

return JsonResult.error("修改失败");

}

@ResponseBody

@RequestMapping("/updateggkc")

public JsonResult updateggkc(Integer id,String type) {

Course course = courseService.getById(id);

if(type.equals("add")) {

course.setGgkc(1);

}else {

course.setGgkc(0);

}

if (courseService.updateById(course)) {

if(type.equals("add")) {

return JsonResult.ok("添加成功");

}else {

return JsonResult.ok("移除成功");

}

}

if(type.equals("add")) {

return JsonResult.error("添加失败");

}else {

return JsonResult.error("移除失败");

}

}

@ResponseBody

@RequestMapping("/updatezykc")

public JsonResult updatezykc(Integer id,String type) {

Course course = courseService.getById(id);

if(type.equals("add")) {

course.setZykc(1);

}else {

course.setZykc(0);

}

if (courseService.updateById(course)) {

if(type.equals("add")) {

return JsonResult.ok("添加成功");

}else {

return JsonResult.ok("移除成功");

}

}

if(type.equals("add")) {

return JsonResult.error("添加失败");

}else {

return JsonResult.error("移除失败");

}

}

@ResponseBody

@RequestMapping("/updatemfkc")

public JsonResult updatemfkc(Integer id,String type) {

Course course = courseService.getById(id);

if(type.equals("add")) {

course.setMfkc(1);

}else {

course.setMfkc(0);

}

if (courseService.updateById(course)) {

if(type.equals("add")) {

return JsonResult.ok("添加成功");

}else {

return JsonResult.ok("移除成功");

}

}

if(type.equals("add")) {

return JsonResult.error("添加失败");

}else {

return JsonResult.error("移除失败");

}

}

/\*\*

\* 删除

\*/

@ResponseBody

@RequestMapping("/remove")

public JsonResult remove(Integer id) {

if (courseService.removeById(id)) {

return JsonResult.ok("删除成功");

}

return JsonResult.error("删除失败");

}

/\*\*

\* 批量添加

\*/

@ResponseBody

@RequestMapping("/saveBatch")

public JsonResult saveBatch(@RequestBody List<Course> list) {

if (courseService.saveBatch(list)) {

return JsonResult.ok("添加成功");

}

return JsonResult.error("添加失败");

}

/\*\*

\* 批量修改

\*/

@ResponseBody

@RequestMapping("/updateBatch")

public JsonResult updateBatch(@RequestBody BatchParam<Course> batchParam) {

if (batchParam.update(courseService, "id")) {

return JsonResult.ok("修改成功");

}

return JsonResult.error("修改失败");

}

/\*\*

\* 批量删除

\*/

@ResponseBody

@RequestMapping("/removeBatch")

public JsonResult removeBatch(@RequestBody List<Integer> ids) {

if (courseService.removeByIds(ids)) {

return JsonResult.ok("删除成功");

}

return JsonResult.error("删除失败");

}

@ResponseBody

@PostMapping("/edit\_upload")

public JsonResult edit\_upload(@RequestParam MultipartFile file) {

return FileUploadUtil.edit\_upload(file);

}

}

@Controller

@RequestMapping("/course/teacher")

public class TeacherController extends BaseController {

@Autowired

private TeacherService teacherService;

@Autowired

private UserRoleService userRoleService;

@Autowired

private UserService userService;

@RequestMapping()

public String view() {

return "course/teacher.html";

}

/\*\*

\* 分页查询

\*/

@ResponseBody

@RequestMapping("/page")

public PageResult<Teacher> page(HttpServletRequest request) {

PageParam<Teacher> pageParam = new PageParam<>(request);

return new PageResult<>(teacherService.page(pageParam, pageParam.getWrapper()).getRecords(), pageParam.getTotal());

//return teacherService.listPage(pageParam); // 使用关联查询

}

/\*\*

\* 查询全部

\*/

@ResponseBody

@RequestMapping("/list")

public JsonResult list(HttpServletRequest request) {

PageParam<Teacher> pageParam = new PageParam<>(request);

return JsonResult.ok().setData(teacherService.list(pageParam.getOrderWrapper()));

//List<Teacher> records = teacherService.listAll(pageParam.getNoPageParam()); // 使用关联查询

//return JsonResult.ok().setData(pageParam.sortRecords(records));

}

/\*\*

\* 查询全部

\*/

@ResponseBody

@RequestMapping("/listuser")

public JsonResult listuser() {

List<UserRole> userRoles = userRoleService.list(new QueryWrapper<UserRole>().eq("role\_id", 2));

String ids = "";

for(UserRole userrole : userRoles) {

ids += userrole.getUserId()+",";

}

ids = ids.substring(0,ids.length() - 1);

List<User> users = userService.list(new QueryWrapper<User>().in("user\_id", ids).eq("state", 0).eq("deleted", 0));

return JsonResult.ok().setData(users);

}

/\*\*

\* 根据id查询

\*/

@ResponseBody

@RequestMapping("/get")

public JsonResult get(Integer id) {

return JsonResult.ok().setData(teacherService.getById(id));

// 使用关联查询

//PageParam<Teacher> pageParam = new PageParam<>();

//pageParam.put("id", id);

//List<Teacher> records = teacherService.listAll(pageParam.getNoPageParam());

//return JsonResult.ok().setData(pageParam.getOne(records));

}

/\*\*

\* 根据id查询

\*/

@ResponseBody

@RequestMapping("/getById")

public String getById(Integer id) {

Teacher teacher = teacherService.getById(id);

return teacher.getNickName();

}

/\*\*

\* 根据id查询

\*/

@ResponseBody

@RequestMapping("/userId")

public String userId(Integer userId) {

User user = userService.getById(userId);

return user.getUsername();

}

@ResponseBody

@RequestMapping("/updateByuserId")

public JsonResult updateByuserId(Integer id,Integer userId) {

Teacher teacher = teacherService.getById(id);

teacher.setUserId(userId);

if (teacherService.updateById(teacher)) {

return JsonResult.ok("授权成功");

}

return JsonResult.error("授权失败");

}

/\*\*

\* 添加

\*/

@ResponseBody

@RequestMapping("/save")

public JsonResult save(Teacher teacher) {

if (teacherService.save(teacher)) {

return JsonResult.ok("添加成功");

}

return JsonResult.error("添加失败");

}

/\*\*

\* 修改

\*/

@ResponseBody

@RequestMapping("/update")

public JsonResult update(Teacher teacher) {

if (teacherService.updateById(teacher)) {

return JsonResult.ok("修改成功");

}

return JsonResult.error("修改失败");

}

/\*\*

\* 删除

\*/

@ResponseBody

@RequestMapping("/remove")

public JsonResult remove(Integer id) {

if (teacherService.removeById(id)) {

return JsonResult.ok("删除成功");

}

return JsonResult.error("删除失败");

}

/\*\*

\* 批量添加

\*/

@ResponseBody

@RequestMapping("/saveBatch")

public JsonResult saveBatch(@RequestBody List<Teacher> list) {

if (teacherService.saveBatch(list)) {

return JsonResult.ok("添加成功");

}

return JsonResult.error("添加失败");

}

/\*\*

\* 批量修改

\*/

@ResponseBody

@RequestMapping("/updateBatch")

public JsonResult updateBatch(@RequestBody BatchParam<Teacher> batchParam) {

if (batchParam.update(teacherService, "id")) {

return JsonResult.ok("修改成功");

}

return JsonResult.error("修改失败");

}

/\*\*

\* 批量删除

\*/

@ResponseBody

@RequestMapping("/removeBatch")

public JsonResult removeBatch(@RequestBody List<Integer> ids) {

if (teacherService.removeByIds(ids)) {

return JsonResult.ok("删除成功");

}

return JsonResult.error("删除失败");

}

@ResponseBody

@PostMapping("/upload")

public JsonResult upload(@RequestParam MultipartFile file) {

return FileUploadUtil.upload(file);

}

}

@Controller

@RequestMapping("/course/video")

public class VideoController extends BaseController {

@Autowired

private VideoService videoService;

@RequestMapping()

public String view() {

return "course/video.html";

}

/\*\*

\* 分页查询

\*/

@ResponseBody

@RequestMapping("/page")

public PageResult<Video> page(HttpServletRequest request) {

PageParam<Video> pageParam = new PageParam<>(request);

return new PageResult<>(videoService.page(pageParam, pageParam.getWrapper()).getRecords(), pageParam.getTotal());

//return classifyService.listPage(pageParam); // 使用关联查询

}

/\*\*

\* 查询全部

\*/

@ResponseBody

@RequestMapping("/list")

public JsonResult list(HttpServletRequest request) {

PageParam<Video> pageParam = new PageParam<>(request);

return JsonResult.ok().setData(videoService.list(pageParam.getOrderWrapper()));

//List<Classify> records = classifyService.listAll(pageParam.getNoPageParam()); // 使用关联查询

//return JsonResult.ok().setData(pageParam.sortRecords(records));

}

/\*\*

\* 根据id查询

\*/

@ResponseBody

@RequestMapping("/get")

public JsonResult get(Integer id) {

return JsonResult.ok().setData(videoService.getById(id));

// 使用关联查询

//PageParam<Classify> pageParam = new PageParam<>();

//pageParam.put("id", id);

//List<Classify> records = classifyService.listAll(pageParam.getNoPageParam());

//return JsonResult.ok().setData(pageParam.getOne(records));

}

/\*\*

\* 添加

\*/

@ResponseBody

@RequestMapping("/save")

public JsonResult save(Video video) {

video.setCreateTime(new Date());

if (videoService.save(video)) {

return JsonResult.ok("添加成功");

}

return JsonResult.error("添加失败");

}

/\*\*

\* 修改

\*/

@ResponseBody

@RequestMapping("/update")

public JsonResult update(Video video) {

if (videoService.updateById(video)) {

return JsonResult.ok("修改成功");

}

return JsonResult.error("修改失败");

}

/\*\*

\* 删除

\*/

@ResponseBody

@RequestMapping("/remove")

public JsonResult remove(Integer id) {

if (videoService.removeById(id)) {

return JsonResult.ok("删除成功");

}

return JsonResult.error("删除失败");

}

/\*\*

\* 批量添加

\*/

@ResponseBody

@RequestMapping("/saveBatch")

public JsonResult saveBatch(@RequestBody List<Video> list) {

if (videoService.saveBatch(list)) {

return JsonResult.ok("添加成功");

}

return JsonResult.error("添加失败");

}

/\*\*

\* 批量修改

\*/

@ResponseBody

@RequestMapping("/updateBatch")

public JsonResult updateBatch(@RequestBody BatchParam<Video> batchParam) {

if (batchParam.update(videoService, "id")) {

return JsonResult.ok("修改成功");

}

return JsonResult.error("修改失败");

}

/\*\*

\* 批量删除

\*/

@ResponseBody

@RequestMapping("/removeBatch")

public JsonResult removeBatch(@RequestBody List<Integer> ids) {

if (videoService.removeByIds(ids)) {

return JsonResult.ok("删除成功");

}

return JsonResult.error("删除失败");

}

}

com.egao.common.test.entity:

@TableName("t\_video")

public class Video implements Serializable {

private static final long serialVersionUID = 1L;

/\*\*

\* 主键

\*/

@TableId(value = "id", type = IdType.AUTO)

private Integer id;

/\*\*

\* 名称

\*/

private String name;

/\*\*

\* 名称

\*/

private String url;

/\*\*

\* 排序

\*/

private Integer sort;

/\*\*

\* 创建时间

\*/

private Date createTime;

private Integer courseId;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Integer getSort() {

return sort;

}

public void setSort(Integer sort) {

this.sort = sort;

}

public String getUrl() {

return url;

}

public void setUrl(String url) {

this.url = url;

}

public Date getCreateTime() {

return createTime;

}

public void setCreateTime(Date createTime) {

this.createTime = createTime;

}

public Integer getCourseId() {

return courseId;

}

public void setCourseId(Integer courseId) {

this.courseId = courseId;

}

@Override

public String toString() {

return "Classify{" +

", id=" + id +

", name=" + name +

", sort=" + sort +

", url=" + url +

", createTime=" + createTime +

", courseId=" + courseId +

"}";

}

}

@TableName("t\_teacher")

public class Teacher implements Serializable {

private static final long serialVersionUID = 1L;

/\*\*

\* 主键

\*/

@TableId(value = "id", type = IdType.AUTO)

private Integer id;

/\*\*

\* 真实姓名

\*/

@TableField("nickName")

private String nickName;

/\*\*

\* 备注

\*/

private String remark;

/\*\*

\* 图片

\*/

@TableField("imageName")

private String imageName;

private Integer userId;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getNickName() {

return nickName;

}

public void setNickName(String nickName) {

this.nickName = nickName;

}

public String getRemark() {

return remark;

}

public void setRemark(String remark) {

this.remark = remark;

}

public String getImageName() {

return imageName;

}

public void setImageName(String imageName) {

this.imageName = imageName;

}

public Integer getUserId() {

return userId;

}

public void setUserId(Integer userId) {

this.userId = userId;

}

@Override

public String toString() {

return "Teacher{" +

", id=" + id +

", nickName=" + nickName +

", remark=" + remark +

", imageName=" + imageName +

", userId=" + userId +

"}";

}

}

@TableName("t\_course")

public class Course implements Serializable {

private static final long serialVersionUID = 1L;

/\*\*

\* 主键

\*/

@TableId(value = "id", type = IdType.AUTO)

private Integer id;

/\*\*

\* 名称

\*/

private String name;

/\*\*

\* 缩略图

\*/

private String imageUrl;

/\*\*

\* 类别

\*/

private String type;

/\*\*

\* 内容

\*/

private String content;

/\*\*

\* 教师id

\*/

private Integer teacherId;

/\*\*

\* 创建时间

\*/

private Date createTime;

/\*\*

\* 分类id

\*/

private Integer classifyId;

private Integer ggkc;

private Integer zykc;

private Integer mfkc;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getImageUrl() {

return imageUrl;

}

public void setImageUrl(String imageUrl) {

this.imageUrl = imageUrl;

}

public String getType() {

return type;

}

public void setType(String type) {

this.type = type;

}

public String getContent() {

return content;

}

public void setContent(String content) {

this.content = content;

}

public Integer getTeacherId() {

return teacherId;

}

public void setTeacherId(Integer teacherId) {

this.teacherId = teacherId;

}

public Date getCreateTime() {

return createTime;

}

public void setCreateTime(Date createTime) {

this.createTime = createTime;

}

public Integer getClassifyId() {

return classifyId;

}

public void setClassifyId(Integer classifyId) {

this.classifyId = classifyId;

}

public Integer getGgkc() {

return ggkc;

}

public void setGgkc(Integer ggkc) {

this.ggkc = ggkc;

}

public Integer getZykc() {

return zykc;

}

public void setZykc(Integer zykc) {

this.zykc = zykc;

}

public Integer getMfkc() {

return mfkc;

}

public void setMfkc(Integer mfkc) {

this.mfkc = mfkc;

}

@Override

public String toString() {

return "Course{" +

", id=" + id +

", name=" + name +

", imageUrl=" + imageUrl +

", type=" + type +

", content=" + content +

", teacherId=" + teacherId +

", createTime=" + createTime +

", classifyId=" + classifyId +

", ggkc=" + ggkc +

", zykc=" + zykc +

", mfkc=" + mfkc +

"}";

}

}

com.egao.common.test.mapper:

public interface VideoMapper extends BaseMapper<Video> {

/\*\*

\* 分页查询

\*/

List<Video> listPage(@Param("page") PageParam<Video> page);

/\*\*

\* 查询全部

\*/

List<Video> listAll(@Param("page") Map<String, Object> page);

}

public interface TeacherMapper extends BaseMapper<Teacher> {

/\*\*

\* 分页查询

\*/

List<Teacher> listPage(@Param("page") PageParam<Teacher> page);

/\*\*

\* 查询全部

\*/

List<Teacher> listAll(@Param("page") Map<String, Object> page);

}

public interface LunboMapper extends BaseMapper<Lunbo> {

/\*\*

\* 分页查询

\*/

List<Lunbo> listPage(@Param("page") PageParam<Lunbo> page);

/\*\*

\* 查询全部

\*/

List<Lunbo> listAll(@Param("page") Map<String, Object> page);

}

**com.egao.common.test.service:**

public interface VideoService extends IService<Video> {

/\*\*

\* 分页查询

\*/

PageResult<Video> listPage(PageParam<Video> page);

/\*\*

\* 查询所有

\*/

List<Video> listAll(Map<String, Object> page);

}

public interface TeacherService extends IService<Teacher> {

/\*\*

\* 分页查询

\*/

PageResult<Teacher> listPage(PageParam<Teacher> page);

/\*\*

\* 查询所有

\*/

List<Teacher> listAll(Map<String, Object> page);

}

public interface MenurService extends IService<Menur> {

/\*\*

\* 分页查询

\*/

PageResult<Menur> listPage(PageParam<Menur> page);

/\*\*

\* 查询所有

\*/

List<Menur> listAll(Map<String, Object> page);

}

**com.egao.common.test.service.impl:**

@Service

public class UserServiceImpl extends ServiceImpl<UserMapper, User> implements UserService {

@Autowired

private UserRoleMapper userRoleMapper;

@Autowired

private MenuMapper menuMapper;

@Override

public User getByUsername(String username) {

return baseMapper.selectOne(new QueryWrapper<User>().eq("username", username));

}

@Override

public User getFullById(Integer userId) {

List<User> userList = baseMapper.listAll(new PageParam<User>().put("userId", userId).getNoPageParam());

if (userList == null || userList.size() == 0) return null;

return selectRoleAndAuth(userList.get(0));

}

@Override

public User selectRoleAndAuth(User user) {

user.setRoles(userRoleMapper.listByUserId(user.getUserId()));

List<Menu> menus = menuMapper.listByUserId(user.getUserId(), null);

List<String> auths = new ArrayList<>();

for (Menu menu : menus) {

auths.add(menu.getAuthority());

}

user.setAuthorities(auths);

return user;

}

@Override

public PageResult<User> listPage(PageParam<User> page) {

List<User> users = baseMapper.listPage(page);

// 查询用户的角色

selectUserRoles(users);

return new PageResult<>(users, page.getTotal());

}

@Override

public List<User> listAll(Map<String, Object> page) {

List<User> users = baseMapper.listAll(page);

// 查询用户的角色

selectUserRoles(users);

return users;

}

@Transactional

@Override

public boolean saveUser(User user) {

if (user.getUsername() != null && baseMapper.selectCount(new QueryWrapper<User>()

.eq("username", user.getUsername())) > 0) {

throw new BusinessException("账号已存在");

}

if (user.getPhone() != null && baseMapper.selectCount(new QueryWrapper<User>()

.eq("phone", user.getPhone())) > 0) {

throw new BusinessException("手机号已存在");

}

if (user.getEmail() != null && baseMapper.selectCount(new QueryWrapper<User>()

.eq("email", user.getEmail())) > 0) {

throw new BusinessException("邮箱已存在");

}

boolean result = baseMapper.insert(user) > 0;

if (result && user.getRoleIds() != null) {

addUserRoles(user.getUserId(), user.getRoleIds(), false);

}

return result;

}

@Transactional

@Override

public boolean updateUser(User user) {

if (user.getUsername() != null && baseMapper.selectCount(new QueryWrapper<User>()

.eq("username", user.getUsername()).ne("user\_id", user.getUserId())) > 0) {

throw new BusinessException("账号已存在");

}

if (user.getPhone() != null && baseMapper.selectCount(new QueryWrapper<User>()

.eq("phone", user.getPhone()).ne("user\_id", user.getUserId())) > 0) {

throw new BusinessException("手机号已存在");

}

if (user.getEmail() != null && baseMapper.selectCount(new QueryWrapper<User>()

.eq("email", user.getEmail()).ne("user\_id", user.getUserId())) > 0) {

throw new BusinessException("邮箱已存在");

}

boolean result = baseMapper.updateById(user) > 0;

if (result && user.getRoleIds() != null) {

addUserRoles(user.getUserId(), user.getRoleIds(), true);

}

return result;

}

@Override

public boolean comparePsw(String dbPsw, String inputPsw) {

return dbPsw != null && dbPsw.equals(encodePsw(inputPsw));

}

@Override

public String encodePsw(String psw) {

if (psw == null) return null;

return DigestUtils.md5DigestAsHex(psw.getBytes());

}

/\*\*

\* 查询用户的角色

\*/

private void selectUserRoles(List<User> users) {

if (users != null && users.size() > 0) {

List<Integer> userIds = new ArrayList<>();

for (User one : users) {

userIds.add(one.getUserId());

}

List<Role> userRoles = userRoleMapper.listByUserIds(userIds);

for (User user : users) {

List<Role> roles = new ArrayList<>();

for (Role userRole : userRoles) {

if (user.getUserId().equals(userRole.getUserId())) {

roles.add(userRole);

}

}

user.setRoles(roles);

}

}

}

/\*\*

\* 添加用户角色

\*/

private void addUserRoles(Integer userId, List<Integer> roleIds, boolean deleteOld) {

if (deleteOld) {

userRoleMapper.delete(new UpdateWrapper<UserRole>().eq("user\_id", userId));

}

if (roleIds.size() > 0) {

if (userRoleMapper.insertBatch(userId, roleIds) < roleIds.size()) {

throw new BusinessException("操作失败");

}

}

}

}

@Service

public class UserRoleServiceImpl extends ServiceImpl<UserRoleMapper, UserRole> implements UserRoleService {

@Override

public Integer[] getRoleIds(String userId) {

List<UserRole> userRoles = baseMapper.selectList(new QueryWrapper<UserRole>().eq("user\_id", userId));

Integer[] roleIds = new Integer[userRoles.size()];

for (int i = 0; i < userRoles.size(); i++) {

roleIds[i] = userRoles.get(i).getRoleId();

}

return roleIds;

}

}

@Service

public class MenuServiceImpl extends ServiceImpl<MenuMapper, Menu> implements MenuService {

@Override

public PageResult<Menu> listPage(PageParam<Menu> pageParam) {

return new PageResult<>(baseMapper.listPage(pageParam), pageParam.getTotal());

}

@Override

public List<Menu> getUserMenu(Integer userId, Integer menuType) {

return baseMapper.listByUserId(userId, menuType);

}

@Override

public List<Menu> toMenuTree(List<Menu> menus, Integer parentId) {

List<Menu> list = new ArrayList<>();

for (Menu menu : menus) {

if (parentId.equals(menu.getParentId())) {

menu.setChildren(toMenuTree(menus, menu.getMenuId()));

list.add(menu);

}

}

return list;

}

}