@SpringBootApplication

@ConfigurationPropertiesScan

@EnableTransactionManagement

public class StcJcglApplication {

public static void main(String[] args) {

SpringApplication.run(StcJcglApplication.class, args);

}

}

/\*\*

\* 跨域过滤器

\*

\* @author kong

\*/

@Component

@Order(-200)

public class CorsFilter implements Filter {

static final String OPTIONS = "OPTIONS";

@Override

public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)

throws IOException, ServletException {

HttpServletRequest request = (HttpServletRequest) req;

HttpServletResponse response = (HttpServletResponse) res;

// 允许指定域访问跨域资源

response.setHeader("Access-Control-Allow-Origin", "\*");

// 允许所有请求方式

response.setHeader("Access-Control-Allow-Methods", "POST, GET, OPTIONS, DELETE");

// 有效时间

response.setHeader("Access-Control-Max-Age", "3600");

// 允许的header参数

response.setHeader("Access-Control-Allow-Headers", "x-requested-with,satoken");

// 如果是预检请求，直接返回

if (OPTIONS.equals(request.getMethod())) {

response.getWriter().print("");

return;

}

// System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*过滤器被使用\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

chain.doFilter(req, res);

}

@Override

public void init(FilterConfig filterConfig) {

}

@Override

public void destroy() {

}

}

/\*\*

\* gradle\_home

\*

\* @author Rodger.Young

\* 自定义异常类

\*\*/

@RestControllerAdvice

public class CustomErrorHandler extends BasicErrorController {

public CustomErrorHandler() {

super(new DefaultErrorAttributes(), new ErrorProperties());

}

@ExceptionHandler({MethodArgumentTypeMismatchException.class, MissingServletRequestParameterException.class})

public Object handleExceptionS(@NotNull Exception ex) {

return SaResult.error(ex.getMessage()).setCode(400);

}

@ExceptionHandler({HttpRequestMethodNotSupportedException.class})

public Object handleException405(@NotNull Exception ex) {

return SaResult.error(ex.getMessage()).setCode(405);

}

@ExceptionHandler(JcglException.class)

public SaResult handlerJcglException(JcglException e) {

e.printStackTrace();

return SaResult.ok(e.getMessage()).setCode(500);

}

@ExceptionHandler({Exception.class})

public Object runtimeException(@NotNull Exception ex) {

ex.printStackTrace();

return SaResult.error("请联系系统管理员解决问题").setCode(500);

}

@ExceptionHandler(MethodArgumentNotValidException.class)

public SaResult handleMethodArgumentNotValidException(MethodArgumentNotValidException e) {

// 默认的提示

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

e.getAllErrors().forEach(objectError -> {

msg.add(objectError.getDefaultMessage());

});

return SaResult.ok(Joiner.on(",").join(msg)).setCode(500);

}

@ExceptionHandler(NotLoginException.class)

public SaResult handlerSaTokenException(NotLoginException e) {

// 默认的提示

return SaResult.ok("请登录").setCode(401);

}

@ExceptionHandler(SaTokenException.class)

public SaResult handlerSaTokenException(SaTokenException e) {

// 默认的提示

return SaResult.ok(e.getMessage()).setCode(e.getCode());

}

@Override

public ResponseEntity<Map<String, Object>> error(HttpServletRequest request) {

Map<String, Object> body = getErrorAttributes(request, getErrorAttributeOptions(request, MediaType.ALL));

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

map.put("code", body.get("status"));

map.put("msg", body.get("error"));

map.put("data", null);

return new ResponseEntity<>(map, getStatus(request));

}

@Override

public ModelAndView errorHtml(HttpServletRequest request, HttpServletResponse response) {

return super.errorHtml(request, response);

}

}

@Configuration

public class DruidConfig {

/\*\*

\* 设置数据源

\*

\* @param targetDataSources 备选数据源集合

\* @param sourceName 数据源名称

\* @param beanName bean名称

\*/

public void setDataSource(Map<Object, Object> targetDataSources, String sourceName, String beanName) {

try {

DataSource dataSource = SpringUtil.getBean(beanName);

targetDataSources.put(sourceName, dataSource);

} catch (Exception ignored) {

}

}

/\*\*

\* 去除监控页面底部的广告

\*/

@SuppressWarnings({"rawtypes", "unchecked"})

@Bean

@ConditionalOnProperty(name = "spring.datasource.druid.statViewServlet.enabled", havingValue = "true")

public FilterRegistrationBean removeDruidFilterRegistrationBean(DruidStatProperties properties) {

// 获取web监控页面的参数

DruidStatProperties.StatViewServlet config = properties.getStatViewServlet();

// 提取common.js的配置路径

String pattern = config.getUrlPattern() != null ? config.getUrlPattern() : "/druid/\*";

String commonJsPattern = pattern.replaceAll("\\\*", "js/common.js");

final String filePath = "support/http/resources/js/common.js";

// 创建filter进行过滤

Filter filter = new Filter() {

@Override

public void init(javax.servlet.FilterConfig filterConfig) throws ServletException {

}

@Override

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)

throws IOException, ServletException {

chain.doFilter(request, response);

// 重置缓冲区，响应头不会被重置

response.resetBuffer();

// 获取common.js

String text = Utils.readFromResource(filePath);

// 正则替换banner, 除去底部的广告信息

text = text.replaceAll("<a.\*?banner\"></a><br/>", "");

text = text.replaceAll("powered.\*?shrek.wang</a>", "");

response.getWriter().write(text);

}

@Override

public void destroy() {

}

};

FilterRegistrationBean registrationBean = new FilterRegistrationBean();

registrationBean.setFilter(filter);

registrationBean.addUrlPatterns(commonJsPattern);

return registrationBean;

}

}

/\*\*

\* flowable配置----为放置生成的流程图中中文乱码

\*

\* @author wanggang

\* @date 2023/4/12 14:10

\*/

@Configuration

public class FlowableConfig implements EngineConfigurationConfigurer<SpringProcessEngineConfiguration> {

@Override

public void configure(SpringProcessEngineConfiguration engineConfiguration) {

engineConfiguration.setActivityFontName("宋体");

engineConfiguration.setLabelFontName("宋体");

engineConfiguration.setAnnotationFontName("宋体");

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/3/10 11:58

\*/

@Configuration

public class LocalDateTimeSerializerConfig {

@Value("${spring.jackson.date-format:yyyy-MM-dd HH:mm:ss}")

private String pattern;

public LocalDateTimeSerializer localDateTimeDeserializer() {

return new LocalDateTimeSerializer(DateTimeFormatter.ofPattern(pattern));

}

@Bean

public Jackson2ObjectMapperBuilderCustomizer jackson2ObjectMapperBuilderCustomizer() {

return builder -> builder.serializerByType(LocalDateTime.class, localDateTimeDeserializer());

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/3/6 15:00

\*/

@ConfigurationProperties(prefix = "minio")

@Data

public class MinioConfig {

private String endpoint;

private String accessKey;

private String secretKey;

private String bucketName;

/\*\*

\* 注入minio 客户端

\*

\* @return

\*/

@Bean

public MinioClient minioClient() {

return MinioClient.builder()

.endpoint(endpoint)

.credentials(accessKey, secretKey)

.build();

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/3/10 10:58

\*/

@Configuration

@MapperScan(basePackages = "com.cstc.jcgl.mapper")

public class MybatisPlusConfig {

private final String[] ignoreTables = {"flow\_base\_info","sys\_menu", "sys\_log", "sys\_dict", "DUAL", "sys\_tenant", "sys\_tenant\_pkg", "sys\_tenant\_pkg\_menu","sys\_flow\_role","sys\_user\_flow\_role"};

@Autowired

ISysUserService sysUserService;

@Bean

public MybatisPlusInterceptor mybatisPlusInterceptor() {

MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();

//添加分页插件

interceptor.addInnerInterceptor(new TenantLineInnerInterceptor(new TenantLineHandler() {

@Override

public Expression getTenantId() {

TenantContextInfo tenantContextInfo = TenantContextHolder.getContext().getTenantContextInfo();

return new StringValue(tenantContextInfo.getTenantId());

}

// 这是 default 方法,默认返回 false 表示所有表都需要拼多租户条件

@Override

public boolean ignoreTable(String tableName) {

TenantContextInfo tenantContextInfo = TenantContextHolder.getContext().getTenantContextInfo();

return tableName.startsWith("act\_")||tableName.startsWith("flw\_")||Arrays.asList(ignoreTables).contains(tableName) || tenantContextInfo.getIgnore();

}

@Override

public String getTenantIdColumn() {

return "`tenant\_id`";

}

}));

// 添加分页插件

PaginationInnerInterceptor paginationInnerInterceptor = new PaginationInnerInterceptor(DbType.MYSQL);

paginationInnerInterceptor.setOptimizeJoin(false);

interceptor.addInnerInterceptor(paginationInnerInterceptor);

return interceptor;

}

}

/\*\*

\* @author Clinton Begin

\*/

@SuppressWarnings("all")

public class MybatisPlusStdOutImpl implements Log {

public MybatisPlusStdOutImpl(String clazz) {

// Do Nothing

}

@Override

public boolean isDebugEnabled() {

return true;

}

@Override

public boolean isTraceEnabled() {

return true;

}

@Override

public void error(String s, Throwable e) {

System.err.println(s);

e.printStackTrace(System.err);

}

@Override

public void error(String s) {

System.err.println(s);

}

@Override

public void debug(String s) {

System.out.println(s.replaceAll("\\n"," "));

}

@Override

public void trace(String s) {

System.out.println(s);

}

@Override

public void warn(String s) {

System.out.println(s);

}

}

/\*\*

\* rides配置类

\*

\* @author 15594

\*/

@EnableCaching

@Configuration

public class RidesConfig extends CachingConfigurerSupport {

@Bean

public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory factory) {

RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();

redisTemplate.setConnectionFactory(factory);

// 设置Key的序列化器为StringRedisSerializer

redisTemplate.setKeySerializer(new StringRedisSerializer());

// 设置Hash Key的序列化器为StringRedisSerializer

redisTemplate.setHashKeySerializer(new StringRedisSerializer());

// 设置Value的序列化器为Jackson2JsonRedisSerializer

redisTemplate.setValueSerializer(jackson2JsonRedisSerializer());

// 设置Hash Value的序列化器为Jackson2JsonRedisSerializer

redisTemplate.setHashValueSerializer(jackson2JsonRedisSerializer());

redisTemplate.afterPropertiesSet();

//是否开启事务，true为开启

redisTemplate.setEnableTransactionSupport(true);

return redisTemplate;

}

private Jackson2JsonRedisSerializer<Object> jackson2JsonRedisSerializer() {

ObjectMapper objectMapper = new ObjectMapper();

// 注册Java 8日期时间模块

objectMapper.registerModule(new JavaTimeModule());

// 关闭日期时间作为时间戳的序列化

objectMapper.configure(SerializationFeature.WRITE\_DATES\_AS\_TIMESTAMPS, false);

Jackson2JsonRedisSerializer<Object> serializer = new Jackson2JsonRedisSerializer<>(Object.class);

serializer.setObjectMapper(objectMapper);

return serializer;

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/22 11:55

\*/

@Component

public class SaTokenConfigure implements WebMvcConfigurer {

@Autowired

ISysUserService sysUserService;

// 注册 Sa-Token 拦截器，打开注解式鉴权功能

@Override

public void addInterceptors(InterceptorRegistry registry) {

// 注册 Sa-Token 拦截器，打开注解式鉴权功能

registry.addInterceptor(new SaInterceptor(handle -> {

TenantContextHolder.clear();

StpUtil.checkLogin();

String username=StpUtil.getLoginId().toString();

SysUser sysUser=sysUserService.findSysUserByUsername(username);

TenantContextInfo tenantContextInfo= TenantContextHolder.getContext().getTenantContextInfo();

tenantContextInfo.setSysUser(sysUser);

tenantContextInfo.setTenantId(sysUser.getTenantId());

TenantContextHolder.getContext().setTenantContextInfo(tenantContextInfo);

}))

.addPathPatterns("/\*\*")

.excludePathPatterns("/sso/\*\*")

.excludePathPatterns("/doc.html\*\*")

.excludePathPatterns("/webjars/\*\*")

.excludePathPatterns("/\*\*/api-docs/\*\*")

.excludePathPatterns("/swagger-resources/\*\*")

.excludePathPatterns("/swagger-ui.html\*\*")

.excludePathPatterns("/user/login\*\*")

.excludePathPatterns("/user/wechat/login")

;

}

}

/\*\*

\* @Description: 权限角色

\* @Author: yb

\* @Date: 2023/2/28 15:04

\*/

@Configuration

public class StpInterfaceImpl implements StpInterface {

@Autowired

ISysUserService sysUserService;

@Override

public List<String> getPermissionList(Object loginId, String loginType) {

return sysUserService.findMenuCodesByUsername(loginId.toString());

}

@Override

public List<String> getRoleList(Object loginId, String loginType) {

return sysUserService.findRoleCodeByUsername(loginId.toString());

}

}

/\*\*

\* 线程池配置

\*

\*\*/

@Configuration

public class ThreadPoolConfig{

// 核心线程池大小

private final static int corePoolSize = 50;

// 最大可创建的线程数

private final static int maxPoolSize = 200;

// 队列最大长度

private final static int queueCapacity = 1000;

// 线程池维护线程所允许的空闲时间

private final static int keepAliveSeconds = 300;

@Bean(name = "threadPoolTaskExecutor")

public ThreadPoolTaskExecutor threadPoolTaskExecutor() {

ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();

executor.setMaxPoolSize(maxPoolSize);

executor.setCorePoolSize(corePoolSize);

executor.setQueueCapacity(queueCapacity);

executor.setKeepAliveSeconds(keepAliveSeconds);

// 线程池对拒绝任务(无线程可用)的处理策略

executor.setRejectedExecutionHandler(new ThreadPoolExecutor.CallerRunsPolicy());

return executor;

}

/\*\*

\* 租户使用线程池

\* @return

\*/

@Bean(name = "tenantExecutor")

public Executor tenantExecutor(){

return TtlExecutors.getTtlExecutor(threadPoolTaskExecutor());

}

}

@Aspect

@Component

public class LogAop {

@Autowired

ISysLogService sysLogService;

@Autowired

Executor tenantExecutor;

@Pointcut("execution(\* com.cstc.jcgl.controller..\*.\*(..))")

public void aopLoggerAspect() {

}

@Around("aopLoggerAspect()")

public Object doAround(ProceedingJoinPoint point) {

Object result;

long startTime = System.currentTimeMillis();

MethodSignature signature = (MethodSignature) point.getSignature();

Method method = signature.getMethod();

RequestAttributes requestAttributes = RequestContextHolder.getRequestAttributes();

ServletRequestAttributes servletRequestAttributes = (ServletRequestAttributes) requestAttributes;

SysLog sysLog = new SysLog();

if (servletRequestAttributes != null) {

HttpServletRequest request = servletRequestAttributes.getRequest();

sysLog.setRequestUri(request.getRequestURI());

sysLog.setRemoteAddr(request.getHeader("x-forwarded-for"));

sysLog.setUserAgent(request.getHeader("user-agent"));

}

//获取操作方法名

String className = point.getTarget().getClass().getName();

String methodName = method.getName();

sysLog.setModule(className);

sysLog.setCreateTime(new Date());

if (TenantContextHolder.getContext().getTenantContextInfo().getSysUser() != null) {

sysLog.setCreateBy(TenantContextHolder.getContext().getTenantContextInfo().getSysUser().getId());

}

sysLog.setMethod(methodName);

sysLog.setParams(Arrays.toString(point.getArgs()));

sysLog.setTime(new Date());

sysLog.setExStatus(0);

try {

result = point.proceed();

} catch (RuntimeException e) {

sysLog.setEx(e.toString());

sysLog.setExStatus(1);

throw e;

} catch (Throwable e) {

sysLog.setEx(e.toString());

sysLog.setExStatus(1);

throw new RuntimeException(e);

} finally {

try {

sysLog.setExecTime(System.currentTimeMillis() - startTime);

tenantExecutor.execute(() -> sysLogService.save(sysLog));

} catch (Exception ex) {

ex.printStackTrace();

}

}

return result;

}

}

@Target({ElementType.METHOD,ElementType.TYPE})

@Retention(RetentionPolicy.RUNTIME)

@Inherited

public @interface TenantIgnore {

}

@Aspect

@Component

@Slf4j

public class TenantIgnoreAspect {

@Around("@annotation(tenantIgnore)||@within(tenantIgnore)")

public Object around(ProceedingJoinPoint joinPoint, TenantIgnore tenantIgnore) throws Throwable {

TenantContextInfo tenantContextInfo=TenantContextHolder.getContext().getTenantContextInfo();

try {

if (tenantContextInfo!=null){

tenantContextInfo.setIgnore(true);

TenantContextHolder.getContext().setTenantContextInfo(tenantContextInfo);

}

// 执行逻辑

return joinPoint.proceed();

} finally {

if (tenantContextInfo!=null){

tenantContextInfo.setIgnore(false);

TenantContextHolder.getContext().setTenantContextInfo(tenantContextInfo);

}

}

}

}

/\*\*

\* @Description: 通用创建修改字段填充处理器

\* @Author: yb

\* @Date: 2023/3/9 9:00

\*/

@Slf4j

@Configuration

public class CommonMetaObjectHandler implements MetaObjectHandler {

@Override

public void insertFill(MetaObject metaObject) {

SysUser sysUser = TenantContextHolder.getContext().getTenantContextInfo().getSysUser();

if (!Objects.isNull(sysUser)) {

// 或者

this.strictInsertFill(metaObject, "createBy", sysUser::getId, String.class); // 起始版本 3.3.3(推荐)

this.strictInsertFill(metaObject, "createName", sysUser::getEmpName, String.class); // 起始版本 3.3.3(推荐)

}

this.setFieldValByName("createTime", new Date(), metaObject);

}

@Override

public void updateFill(MetaObject metaObject) {

SysUser sysUser = TenantContextHolder.getContext().getTenantContextInfo().getSysUser();

if (!Objects.isNull(sysUser)) {

this.strictUpdateFill(metaObject, "updateBy", sysUser::getId, String.class); // 起始版本 3.3.3(推荐)

this.strictUpdateFill(metaObject, "updateName", sysUser::getEmpName, String.class); // 起始版本 3.3.3(推荐)

}

// 或者

this.setFieldValByName("updateTime", new Date(), metaObject);

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/5/9 15:28

\*/

public interface TenantContext extends Serializable {

TenantContextInfo getTenantContextInfo();

void setTenantContextInfo(TenantContextInfo tenantContextInfo);

}

public class TenantContextHolder {

// 创建一个线程变量，类型为BaseContext

private static final ThreadLocal<TenantContext> contextHolder = new TransmittableThreadLocal<>(){

@Override

protected TenantContext initialValue() {

TenantContext tenantContext=new TenantContextImpl();

TenantContextInfo tenantContextInfo=new TenantContextInfo();

tenantContextInfo.setIgnore(false);

tenantContext.setTenantContextInfo(tenantContextInfo);

return tenantContext;

}

};

// 设置变量

public static void setContext(TenantContext baseContext) { contextHolder.set(baseContext); }

public static TenantContext getContext() {

TenantContext obj = contextHolder.get();

if (obj == null) {

throw new NullPointerException("不存在的用户上下文");

}

return obj;

}

public static void clear() {

contextHolder.remove();

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/5/9 15:36

\*/

public class TenantContextImpl implements TenantContext{

@Serial

private static final long serialVersionUID = -7020860718908760031L;

// 用户类

private TenantContextInfo tenantContextInfo = new TenantContextInfo();

@Override

public TenantContextInfo getTenantContextInfo() {

return tenantContextInfo;

}

@Override

public void setTenantContextInfo(TenantContextInfo tenantContextInfo) {

this.tenantContextInfo=tenantContextInfo;

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/5/9 15:34

\*/

@Data

public class TenantContextInfo {

//忽略租户

private Boolean ignore;

//租户Id

private String tenantId;

private SysUser sysUser;

}

public class CacheConstants {

}

public class DateTimeConstant {

public static final String YEAR = "yyyy";

public static final String YEAR\_MONTH = "yyyy-MM";

public static final String DATE = "yyyy-MM-dd";

public static final String TIME = "HH:mm:ss";

public static final String DATE\_TIME = "yyyy-MM-dd HH:mm:ss";

public static final String DAY\_START\_TIME = " 00:00:00";

public static final String DAY\_END\_TIME = " 23:59:59";

public class SystemConstant {

private SystemConstant() {

}

// redis键的前缀，全局通用

public static final String REDIS\_KEY\_PREFIX = "jcgl-";

public static final String SYSTEM\_TENANT\_ID = "1657945176121942018";

public static final String TREE\_TITLE = "title";

public static final String TREE\_VALUE = "value";

public static final String TREE\_CHILDREN = "children";

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/10 8:51

\*/

@RestController

public class BaseWebController {

@Autowired

ISysUserService sysUserService;

@Autowired

ISysOrgService sysOrgService;

@Autowired

ISysUserOrgService sysUserOrgService;

@Autowired

ISysFlowRoleService sysFlowRoleService;

public SysUser getUserDetail() {

return TenantContextHolder.getContext().getTenantContextInfo().getSysUser();

}

public String getTenantId() {

return TenantContextHolder.getContext().getTenantContextInfo().getSysUser().getTenantId();

}

/\*\*

\* 获取授权组织

\*

\* @return

\*/

public List<String> getAuthSysOrgIds() {

//查询当前组织以及以下

List<String> authSysOrgIds = sysUserService.findAuthSysOrgIds(getUserDetail().getSysOrgId());

return authSysOrgIds;

}

/\*\*

\* 获取当前用户组织和

\*

\* @return

\*/

public List<String> getOrgIds() {

SysUser sysUser = getUserDetail();

List<String> sysOrgIds = sysUserOrgService.list(new QueryWrapper<SysUserOrg>().lambda().eq(SysUserOrg::getSysUserId, sysUser.getId()).select(SysUserOrg::getSysOrgId)).stream().map(SysUserOrg::getSysOrgId).collect(Collectors.toList());

if (!sysOrgIds.isEmpty()) {

sysOrgIds.add(sysUser.getSysOrgId());

return sysOrgIds;

} else {

return Collections.singletonList(sysUser.getSysOrgId());

}

}

/\*\*

\* 获取主流程角色

\*

\* @return

\*/

public List<SysFlowRole> getMainFlowRoles() {

SysUser sysUser = getUserDetail();

//获取主组织中路程角色编码

return sysUserService.findFlowRoleCodes(sysUser.getId());

}

/\*\*

\* 获取兼职流程角色

\*

\* @return

\*/

public List<SysUserOrg> getSlaveFlowRoles() {

SysUser sysUser = getUserDetail();

//获取兼职

List<SysUserOrg> sysUserOrgs = sysUserOrgService.list(new QueryWrapper<SysUserOrg>().lambda().eq(SysUserOrg::getSysUserId, sysUser.getId()));

if (sysUserOrgs == null) {

sysUserOrgs = new ArrayList<>();

}

sysUserOrgs.forEach(sysUserOrg -> {

if (sysUserOrg.getSysFlowRoleId() != null && !sysUserOrg.getSysFlowRoleId().isEmpty()) {

sysUserOrg.setSysFlowRoles(sysFlowRoleService.list(new QueryWrapper<SysFlowRole>().lambda().in(SysFlowRole::getId, sysUserOrg.getSysFlowRoleId())));

}

});

return sysUserOrgs;

}

public Set<String> getAllFlowRoleCodes() {

Set<String> mainFlowRoleCodes = getMainFlowRoles().stream().map(SysFlowRole::getCode).collect(Collectors.toSet());

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

getSlaveFlowRoles().forEach(sysUserOrg -> {

if (sysUserOrg.getSysFlowRoles() != null && !sysUserOrg.getSysFlowRoles().isEmpty()) {

slaveFlowRoleCodes.addAll(sysUserOrg.getSysFlowRoles().stream().map(SysFlowRole::getCode).collect(Collectors.toSet()));

}

});

return CollUtil.unionDistinct(mainFlowRoleCodes,slaveFlowRoleCodes);

}

//获取全部的组织包括兼职组织

public Set<String> getAllOrgIds(){

SysUser sysUser = getUserDetail();

Set<String> orgIds= sysUserOrgService.list(new QueryWrapper<SysUserOrg>().lambda()

.select(SysUserOrg::getSysOrgId)

.eq(SysUserOrg::getSysUserId,sysUser.getId())).stream().map(SysUserOrg::getSysOrgId).collect(Collectors.toSet());

orgIds.add(sysUser.getSysOrgId());

return orgIds;

}

public void checkFileSize(MultipartFile file){

if (file == null) return;

// 检测文件大小 1048576 == 1MB

if (file.getSize() > 52428800) {

throw new ServiceException("导入数据不能大于50M");

}

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/22 12:37

\*/

@RestController

@RequestMapping("/user")

@TenantIgnore

@Api(tags = "登录")

public class LoginController {

private static final Logger log = LoggerFactory.getLogger(LoginController.class);

@Autowired

ISysUserService sysUserService;

@Resource

private RedisUtil redisUtil;

@Resource

private WechatUtil wechatUtil;

@PostMapping("login")

@ApiOperation("登录接口")

public SaResult saResult(@Validated @RequestBody LoginDTO loginDTO) {

SysUser sysUser = sysUserService.login(loginDTO.getUsername(), loginDTO.getPassword());

if (sysUser != null) {

StpUtil.login(sysUser.getUsername());

return SaResult.ok().setData(StpUtil.getTokenInfo().tokenValue);

}

return SaResult.error("登录失败,请检查用户密码");

}

@GetMapping("wechat/login")

@ApiOperation(value = "微信登录接口", notes = "<a href=\"https://developers.weixin.qq.com/miniprogram/dev/framework/open-ability/getPhoneNumber.html\">手机号获取凭证</a>")

@ApiImplicitParam(name = "code", value = "手机号获取凭证", required = true)

public SaResult wechatLogin(@RequestParam(name = "code") String code) {

String phone;

PhoneInfoResponse info = wechatUtil.getPhoneNumber(code);

log.info("获取手机号码信息：{}", info.toString());

switch (info.getErrcode()) {

case -1:

throw new ServiceException("微信系统繁忙，请稍候再试");

case 40029, 47001:

throw new ServiceException("js\_code【" + code + "】无效,请联系系统管理员");

case 45009:

throw new ServiceException("获取手机号码超过调用额度请联系系统管理员解决。(" + info.getErrmsg() + ")");

case 40001:

// access\_token 失效或者不是最新的，因为我们在系统外面重新获取了 access\_token，导致缓存失效需要手动触发重新获取

String tokenKey = SystemConstant.REDIS\_KEY\_PREFIX + "access\_token";

redisUtil.deleteObject(tokenKey);

info = wechatUtil.getPhoneNumber(code);

break;

case 0:

break;

default:

throw new ServiceException("获取手机号码失败请联系系统管理员解决。(" + info.getErrcode() + "," + info.getErrmsg() + ")");

}

if (info.getErrcode() != 0) {

log.error("获取手机号码失败：{}", info.toString());

throw new ServiceException("获取手机号码失败请联系系统管理员解决(" + info.getErrcode() + "," + info.getErrmsg() + ")");

}

phone = info.getPhoneInfo().getPurePhoneNumber();

if (StringUtils.isBlank(phone)) {

throw new ServiceException("从微信获取手机号码失败！");

}

SysUser sysUser = sysUserService.findSysUserByPhoneNumber(phone);

// 使用手机号码查找对应的用户然后登录

if (sysUser != null) {

StpUtil.login(sysUser.getUsername());

return SaResult.ok().setData(StpUtil.getTokenInfo().tokenValue);

} else {

return SaResult.error("登录失败,请绑定用户数据，该用户数据不唯一");

}

}

}

/\*\*

\* 设备资产信息记录

\*

\* @author wanggang

\* @date 2023/5/6 14:46

\*/

@RestController

@RequestMapping("/DeviceAssetInfoRecord")

@RequiredArgsConstructor

@Api(tags = "设备资产信息记录接口")

public class DeviceAssetInfoRecordController extends BaseWebController {

private final IDeviceAssetInfoRecordService iDeviceAssetInfoRecordService;

@GetMapping("/selectDeviceAssetInfoRecordPage")

@ApiOperation("分页查询设备资产信息记录接口")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceAssetInfoRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceAssetInfoRecordPage(@ApiParam("设备资产信息记录查询条件实体类") DeviceAssetInfoRecordSelectDTO deviceAssetInfoRecordSelectDTO) {

Page<DeviceAssetInfoRecordVO> deviceAssetInfoRecordPage = iDeviceAssetInfoRecordService.selectDeviceAssetInfoRecordPage(deviceAssetInfoRecordSelectDTO);

return SaResult.ok().setData(deviceAssetInfoRecordPage);

}

@GetMapping("/selectDeviceAssetInfoRecordById")

@ApiOperation("根据资产信息ID查询设备资产信息记录数据")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceAssetInfoRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceAssetInfoRecordById(@ApiParam("资产信息ID") @RequestParam("id") String id) {

DeviceAssetInfoRecordVO deviceAssetInfoRecordVO = iDeviceAssetInfoRecordService.selectDeviceAssetInfoRecordById(id);

return SaResult.ok().setData(deviceAssetInfoRecordVO);

}

@PostMapping("/addDeviceAssetInfoRecord")

@ApiOperation("新增设备资产信息记录")

public SaResult addDeviceAssetInfoRecord(@ApiParam("设备资产信息记录DTO") @RequestBody @Valid DeviceAssetInfoRecordDTO deviceAssetInfoRecordDTO) {

return iDeviceAssetInfoRecordService.addDeviceAssetInfoRecord(deviceAssetInfoRecordDTO) ? SaResult.ok() : SaResult.error();

}

@PutMapping("/updateDeviceAssetInfoRecord")

@ApiOperation("修改设备资产信息记录")

public SaResult updateDeviceAssetInfoRecord(@ApiParam("设备资产信息记录DTO") @RequestBody @Valid DeviceAssetInfoRecordDTO deviceAssetInfoRecordDTO) {

return iDeviceAssetInfoRecordService.updateDeviceAssetInfoRecord(deviceAssetInfoRecordDTO) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteDeviceAssetInfoRecord")

@ApiOperation("逻辑删除设备资产信息记录")

public SaResult deleteDeviceAssetInfoRecord(@ApiParam("设备资产信息记录id") @RequestParam("id") String id) {

return iDeviceAssetInfoRecordService.deleteDeviceAssetInfoRecord(id) ? SaResult.ok() : SaResult.error();

}

}

/\*\*

\* 设备授权记录

\*

\* @author wanggang

\* @date 2023/6/9 9:46

\*/

@RestController

@RequestMapping("/deviceAuthorization")

@RequiredArgsConstructor

@Api(tags = "设备授权接口")

public class DeviceAuthorizationController extends BaseWebController {

private final IDeviceAuthorizationService iDeviceAuthorizationService;

@GetMapping("/selectDeviceAuthorizationPage")

@ApiOperation("分页查询设备授权")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceAuthorizationVO.class, message = "查询成功")

})

public SaResult selectDeviceAuthorizationPage(@ApiParam("设备授权查询条件实体类") DeviceAuthorizationSelectDTO deviceAuthorizationSelectDTO) {

Page<DeviceAuthorizationVO> deviceAuthorizationPage = iDeviceAuthorizationService.selectDeviceAuthorizationPage(deviceAuthorizationSelectDTO);

return SaResult.ok().setData(deviceAuthorizationPage);

}

@GetMapping("/selectDeviceAuthorizationById")

@ApiOperation("根据设备授权ID查询修改回显数据")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceAuthorizationVO.class, message = "查询成功")

})

public SaResult selectDeviceAuthorizationById(@ApiParam("设备授权ID") @RequestParam("id") String id) {

DeviceAuthorizationVO deviceAuthorizationVO = iDeviceAuthorizationService.selectDeviceAuthorizationById(id);

return SaResult.ok().setData(deviceAuthorizationVO);

}

@PostMapping("/addDeviceAuthorizationBatch")

@ApiOperation("批量新增设备授权")

public SaResult addDeviceAuthorizationBatch(@ApiParam("设备授权DTO") @RequestBody List<DeviceAuthorizationDTO> deviceAuthorizationDTOList) {

if (CollUtil.isEmpty(deviceAuthorizationDTOList)) {

throw new ParameterException("新增参数缺失");

}

return iDeviceAuthorizationService.addDeviceAuthorizationBatch(deviceAuthorizationDTOList) ? SaResult.ok() : SaResult.error();

}

@PutMapping("/updateDeviceAuthorization")

@ApiOperation("修改设备授权")

public SaResult updateDeviceAuthorization(@ApiParam("设备授权DTO") @RequestBody @Valid DeviceAuthorizationDTO deviceAuthorizationDTO) {

return iDeviceAuthorizationService.updateDeviceAuthorization(deviceAuthorizationDTO) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteDeviceAuthorization")

@ApiOperation("逻辑删除设备授权")

public SaResult deleteDeviceAuthorization(@ApiParam("设备授权id") @RequestParam("id") String id) {

return iDeviceAuthorizationService.deleteDeviceAuthorization(id) ? SaResult.ok() : SaResult.error();

}

}

/\*\*

\* 设备借用记录

\*

\* @author wanggang

\* @date 2023/5/22 14:46

\*/

@RestController

@RequestMapping("/deviceBorrowRecord")

@RequiredArgsConstructor

@Api(tags = "设备借用记录接口")

public class DeviceBorrowRecordController extends BaseWebController {

private final IDeviceBorrowRecordService iDeviceBorrowRecordService;

@GetMapping("/selectDeviceBorrowRecordPage")

@ApiOperation("分页查询设备借用记录")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceBorrowRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceBorrowRecordPage(@ApiParam("设备借用记录查询条件实体类") DeviceBorrowRecordSelectDTO deviceBorrowRecordSelectDTO) {

Page<DeviceBorrowRecordVO> deviceBorrowRecordPage = iDeviceBorrowRecordService.selectDeviceBorrowRecordPage(deviceBorrowRecordSelectDTO);

return SaResult.ok().setData(deviceBorrowRecordPage);

}

@GetMapping("/selectDeviceBorrowRecordById")

@ApiOperation("根据设备借用记录ID查询修改回显数据")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceBorrowRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceBorrowRecordById(@ApiParam("设备借用记录ID") @RequestParam("id") String id) {

DeviceBorrowRecordVO deviceBorrowRecordVO = iDeviceBorrowRecordService.selectDeviceBorrowRecordById(id);

return SaResult.ok().setData(deviceBorrowRecordVO);

}

@PostMapping("/addDeviceBorrowRecord")

@ApiOperation("新增设备借用记录")

public SaResult addDeviceBorrowRecord(@ApiParam("设备借用记录DTO") @RequestBody @Valid DeviceBorrowRecordDTO deviceBorrowRecordDTO) {

return iDeviceBorrowRecordService.addDeviceBorrowRecord(deviceBorrowRecordDTO) ? SaResult.ok() : SaResult.error();

}

@PutMapping("/updateDeviceBorrowRecord")

@ApiOperation("修改设备借用记录")

public SaResult updateDeviceBorrowRecord(@ApiParam("设备借用记录DTO") @RequestBody @Valid DeviceBorrowRecordDTO deviceBorrowRecordDTO) {

return iDeviceBorrowRecordService.updateDeviceBorrowRecord(deviceBorrowRecordDTO) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteDeviceBorrowRecord")

@ApiOperation("逻辑删除设备借用记录")

public SaResult deleteDeviceBorrowRecord(@ApiParam("设备借用记录id") @RequestParam("id") String id) {

return iDeviceBorrowRecordService.deleteDeviceBorrowRecord(id) ? SaResult.ok() : SaResult.error();

}

}

/\*\*

\* 设备流转记录

\*

\* @author wanggang

\* @date 2023/5/10 14:46

\*/

@RestController

@RequestMapping("/deviceCirculationRecord")

@RequiredArgsConstructor

@Api(tags = "设备流转记录接口")

public class DeviceCirculationRecordController extends BaseWebController {

private final IDeviceCirculationRecordService iDeviceCirculationRecordService;

@GetMapping("/selectDeviceCirculationRecordPage")

@ApiOperation("分页查询设备流转记录")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceCirculationRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceCirculationRecordPage(@ApiParam("设备流转记录查询条件实体类") DeviceCirculationRecordSelectDTO deviceCirculationRecordSelectDTO) {

Page<DeviceCirculationRecordVO> deviceCirculationRecordPage = iDeviceCirculationRecordService.selectDeviceCirculationRecordPage(deviceCirculationRecordSelectDTO);

return SaResult.ok().setData(deviceCirculationRecordPage);

}

@GetMapping("/selectDeviceCirculationRecordById")

@ApiOperation("根据设备设备流转记录ID查询修改回显数据")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceCirculationRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceCirculationRecordById(@ApiParam("设备流转记录ID") @RequestParam("id") String id) {

DeviceCirculationRecordVO deviceCirculationRecordVO = iDeviceCirculationRecordService.selectDeviceCirculationRecordById(id);

return SaResult.ok().setData(deviceCirculationRecordVO);

}

@PostMapping("/addDeviceCirculationRecord")

@ApiOperation("新增设备流转记录")

public SaResult addDeviceCirculationRecord(@ApiParam("设备流转记录DTO") @RequestBody @Valid DeviceCirculationRecordDTO deviceCirculationRecordDTO) {

return iDeviceCirculationRecordService.addDeviceCirculationRecord(deviceCirculationRecordDTO) ? SaResult.ok() : SaResult.error();

}

@PutMapping("/updateDeviceCirculationRecord")

@ApiOperation("修改设备流转记录")

public SaResult updateDeviceCirculationRecord(@ApiParam("设备流转记录DTO") @RequestBody @Valid DeviceCirculationRecordDTO deviceCirculationRecordDTO) {

return iDeviceCirculationRecordService.updateDeviceCirculationRecord(deviceCirculationRecordDTO) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteDeviceCirculationRecord")

@ApiOperation("逻辑删除设备流转记录")

public SaResult deleteDeviceCirculationRecord(@ApiParam("设备流转记录id") @RequestParam("id") String id) {

return iDeviceCirculationRecordService.deleteDeviceCirculationRecord(id) ? SaResult.ok() : SaResult.error();

}

}

@RestController

@RequestMapping("/device/classify")

@Api(tags = "设备分类管理接口")

@RequiredArgsConstructor

@Validated

public class DeviceClassifyInfoController extends BaseWebController {

private final DeviceClassifyInfoService deviceClassifyInfoService;

@GetMapping("/select")

@ApiOperation("查询设备分类信息")

public SaResult selectDeviceClassifyInfo(DeviceClassifyInfoSelectDTO dto) {

dto.setTenantId(this.getTenantId());

return SaResult.ok().setData(deviceClassifyInfoService.findByPage(dto, true));

}

@GetMapping("/selectById")

@ApiOperation("根据主键查询设备分类信息")

@ApiImplicitParam(name = "id", value = "主键", required = true, example = "1658755598444441601")

public SaResult selectDeviceClassifyInfo(String id) {

return SaResult.ok().setData(deviceClassifyInfoService.selectById(id));

}

@GetMapping("/selectClassifyNames")

@ApiOperation("查询设备分类名称数据")

public SaResult selectClassifyNames() {

return SaResult.ok().setData(deviceClassifyInfoService.selectDeviceClassifyNames(this.getTenantId()));

}

// TODO 设备编号接口需要提供

@PostMapping("/add")

@ApiOperation("添加设备分类信息、复制功能")

public SaResult addDeviceClassifyInfo(@RequestBody DeviceClassifyInfoDTO add) {

return deviceClassifyInfoService.add(add) ? SaResult.ok() : SaResult.error();

}

@PostMapping("/update")

@ApiOperation("修改设备分类信息")

public SaResult updateDeviceClassifyInfo(@RequestBody DeviceClassifyInfoDTO update) {

return deviceClassifyInfoService.update(update) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteByIds")

@ApiOperation("删除设备分类信息")

public SaResult deleteDeviceClassifyInfo(String[] ids) {

deviceClassifyInfoService.logicDelete(ids);

return SaResult.ok().setData("删除成功！");

}

@PostMapping("/import")

@ApiOperation("导入设备分类信息")

@ApiImplicitParams({

@ApiImplicitParam(name = "file", value = "文件", required = true),

@ApiImplicitParam(name = "isOverride", value = "是否覆盖已经存在的数据"),

@ApiImplicitParam(name = "isFirstDelete", value = "导入之前是否删除数据")

})

public SaResult upload(@RequestParam(value = "file") MultipartFile file, Boolean isOverride, Boolean isFirstDelete) {

// 检测文件大小 1048576 == 1MB

if (file.getSize() > 52428800) {

return SaResult.error("导入数据不能大于50M");

}

if ((Boolean.TRUE.equals(isOverride) && Boolean.TRUE.equals(isFirstDelete)) ||

Boolean.FALSE.equals(isOverride) && Boolean.FALSE.equals(isFirstDelete)) {

return SaResult.error("是否覆盖和是否删除参数不能同时选中或不选");

}

try {

List<DeviceClassifyInfoVO> data = EasyExcelUtil.readExcel(file.getInputStream(), DeviceClassifyInfoVO.class, new EasyExcelUtil.ExcelListener<>());

deviceClassifyInfoService.importData(data, isOverride, isFirstDelete);

} catch (IOException e) {

throw new ServiceException(e);

}

return SaResult.ok();

}

@GetMapping("/downloadTemplate")

@ApiOperation("下载模板设备分类信息")

public void downloadDeviceClassifyInfo(HttpServletResponse response) {

try {

EasyExcelUtil.writeExcel(response, DeviceClassifyInfoVO.class, Collections.emptyList(), "设备分类信息模板" + System.currentTimeMillis() + ".xlsx");

} catch (IOException | IllegalAccessException | NoSuchFieldException e) {

throw new ServiceException(e);

}

}

@GetMapping("/download")

@ApiOperation("导出设备分类信息")

public void downloadDeviceClassifyInfo(DeviceClassifyInfoSelectDTO dto, HttpServletResponse response) {

deviceClassifyInfoService.exportData(dto, response);

}

}

/\*\*

\* 设备信息

\*

\* @author wanggang

\* @date 2023/5/6 14:46

\*/

@RestController

@RequestMapping("/deviceInfo")

@RequiredArgsConstructor

@Api(tags = "设备信息接口")

public class DeviceInfoController extends BaseWebController {

private final IDeviceInfoService iDeviceInfoService;

@GetMapping("/selectDevicePage")

@ApiOperation("分页查询设备信息接口")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceInfoVO.class, message = "查询成功")

})

public SaResult selectDevicePage(@ApiParam("设备信息查询条件实体类") DeviceInfoSelectDTO deviceInfoSelectDTO) {

Page<DeviceInfoVO> deviceInfoPage = iDeviceInfoService.selectDevicePage(deviceInfoSelectDTO);

return SaResult.ok().setData(deviceInfoPage);

}

@GetMapping("/selectDeviceInfoById")

@ApiOperation("根据设备信息ID查询修改回显数据")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceInfoVO.class, message = "查询成功")

})

public SaResult selectDeviceInfoById(@ApiParam("设备信息ID") @RequestParam("id") String id) {

DeviceInfoVO deviceInfoVO = iDeviceInfoService.selectDeviceInfoById(id);

return SaResult.ok().setData(deviceInfoVO);

}

@PostMapping("/addDeviceInfo")

@ApiOperation("新增设备信息")

public SaResult addDeviceInfo(@ApiParam("设备信息DTO") @RequestBody @Valid DeviceInfoDTO deviceInfoDTO) {

return iDeviceInfoService.addDeviceInfo(deviceInfoDTO) ? SaResult.ok() : SaResult.error();

}

@PutMapping("/updateDeviceInfo")

@ApiOperation("修改设备信息")

public SaResult updateDeviceInfo(@ApiParam("设备信息DTO") @RequestBody @Valid DeviceInfoDTO deviceInfoDTO) {

return iDeviceInfoService.updateDeviceInfo(deviceInfoDTO) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteDeviceInfo")

@ApiOperation("逻辑删除设备信息")

public SaResult deleteDeviceInfo(@ApiParam("设备信息id集") @RequestParam("id") String id) {

return iDeviceInfoService.deleteDeviceInfo(id) ? SaResult.ok() : SaResult.error();

}

@PostMapping("/importDeviceInfo")

@ApiOperation("导入设备信息")

public SaResult importDeviceInfo(@ApiParam("导入的文件") @RequestParam(value = "file") MultipartFile file,

@ApiParam("是否覆盖更新字段(true是覆盖更新false先删除后插入)") @RequestParam(value = "isOverride") Boolean isOverride) {

// 检测文件大小 1048576 == 1MB

if (file.getSize() > 52428800) {

return SaResult.error("导入数据不能大于50M");

}

iDeviceInfoService.importDeviceInfo(file, isOverride);

return SaResult.ok();

}

@GetMapping("/exportDeviceInfo")

@ApiOperation("导出设备信息")

public void exportDeviceInfo(@ApiParam("设备信息查询条件实体类") DeviceInfoSelectDTO deviceInfoSelectDTO, HttpServletResponse response) {

iDeviceInfoService.exportDeviceInfo(deviceInfoSelectDTO, response);

}

}

@RestController

@RequestMapping("/device/place")

@Api(tags = "场所管理接口")

@RequiredArgsConstructor

@Validated

public class DevicePlaceController extends BaseWebController {

private final DevicePlaceService devicePlaceService;

@GetMapping("/select")

@ApiOperation("分页查询场所信息")

@ApiResponse(code = 200, response = DevicePlaceVO.class, message = "查询成功")

public SaResult select(CommonSelectEntity dto) {

dto.setTenantId(this.getTenantId());

return SaResult.ok().setData(devicePlaceService.findByPage(dto, true));

}

@GetMapping("/selectAllInfo")

@ApiOperation("查询当前租户下场所信息")

@ApiResponse(code = 200, response = DevicePlaceVO.class, message = "查询成功")

public SaResult select() {

CommonSelectEntity dto = new CommonSelectEntity();

dto.setTenantId(this.getTenantId());

return SaResult.ok().setData(devicePlaceService.findByPage(dto, false).getRecords());

}

@GetMapping("/treePlaceAndRoom")

@ApiOperation("查询所有的场所和下面的存放室，以树状结构返回")

public SaResult treePlaceAndRoom() {

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

List<DevicePlaceTreeVO> devicePlaceTreeVOS = devicePlaceService.treePlaceAndRoom();

for (DevicePlaceTreeVO first : devicePlaceTreeVOS) {

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

result.add(firstMap);

firstMap.put(SystemConstant.TREE\_TITLE, first.getPlaceName());

firstMap.put("placeCode", first.getPlaceCode());

firstMap.put(SystemConstant.TREE\_VALUE, first.getId());

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

firstMap.put(SystemConstant.TREE\_CHILDREN, subList);

for (DeviceRoomVO deviceRoomVO : first.getDeviceRoomVOS()) {

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

subList.add(secondMap);

secondMap.put(SystemConstant.TREE\_TITLE, deviceRoomVO.getRoomName());

secondMap.put("roomCode", deviceRoomVO.getRoomCode());

secondMap.put(SystemConstant.TREE\_VALUE, deviceRoomVO.getId());

secondMap.put(SystemConstant.TREE\_CHILDREN, Collections.emptyList());

}

}

return SaResult.ok().setData(result);

}

@GetMapping("/treePlaceAndRoomAndShelves")

@ApiOperation("查询所有的场所和下面的存放室以及存放架，以树状结构返回")

public SaResult treePlaceAndRoomAndShelves() {

return SaResult.ok().setData(devicePlaceService.treePlaceAndRoomAndShelves());

}

@GetMapping("/selectById")

@ApiOperation("根据主键查询场所信息")

@ApiImplicitParam(name = "id", value = "主键", required = true)

public SaResult selectById(String id) {

return SaResult.ok().setData(devicePlaceService.selectById(id));

}

@GetMapping("/init")

@ApiOperation("初始化场所信息")

public SaResult addDeviceClassifyInfo() {

devicePlaceService.add();

return SaResult.ok();

}

@PostMapping("/update")

@ApiOperation("修改场所信息")

public SaResult updateDeviceClassifyInfo(@RequestBody DevicePlaceDTO update) {

return devicePlaceService.update(update) ? SaResult.ok() : SaResult.error();

}

@PostMapping("/batchUpdate")

@ApiOperation("批量修改场所信息")

public SaResult updateDeviceClassifyInfo(@RequestBody List<DevicePlaceDTO> update) {

update.forEach(devicePlaceService::update);

return SaResult.ok();

}

@DeleteMapping("/deleteByIds")

@ApiOperation("删除场所信息(没有使用)")

public SaResult deleteDeviceClassifyInfo(String[] ids) {

devicePlaceService.logicDeleteByIds(ids, getTenantId());

return SaResult.ok().setData("删除成功！");

}

@GetMapping("/surplusCode")

@ApiOperation("查询场所剩余编码")

public SaResult selectSurplusPlaceCode() {

List<Character> characters = devicePlaceService.selectSurplusPlaceCode();

List<Map<String, Character>> result = new ArrayList<>(characters.size());

for (Character v : characters) {

Map<String, Character> temp = new HashMap<>(1);

temp.put("label", v);

temp.put("value", v);

result.add(temp);

}

return SaResult.data(result);

}

}

/\*\*

\* 设备保养维修记录

\*

\* @author wanggang

\* @date 2023/5/10 14:46

\*/

@RestController

@RequestMapping("/deviceRepairMaintenanceRecord")

@RequiredArgsConstructor

@Api(tags = "设备保养维修记录接口")

public class DeviceRepairMaintenanceRecordController extends BaseWebController {

private final IDeviceRepairMaintenanceRecordService iDeviceRepairMaintenanceRecordService;

@GetMapping("/selectDeviceRepairMaintenanceRecordPage")

@ApiOperation("分页查询设备保养维修记录")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceRepairMaintenanceRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceRepairMaintenanceRecordPage(@ApiParam("设备保养维修记录查询条件实体类") DeviceRepairMaintenanceRecordSelectDTO deviceRepairMaintenanceRecordSelectDTO) {

Page<DeviceRepairMaintenanceRecordVO> deviceRepairMaintenanceRecordPage = iDeviceRepairMaintenanceRecordService.selectDeviceRepairMaintenanceRecordPage(deviceRepairMaintenanceRecordSelectDTO);

return SaResult.ok().setData(deviceRepairMaintenanceRecordPage);

}

@GetMapping("/selectDeviceRepairMaintenanceRecordById")

@ApiOperation("根据设备设备保养维修记录ID查询修改回显数据")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceRepairMaintenanceRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceRepairMaintenanceRecordById(@ApiParam("设备保养维修记录ID") @RequestParam("id") String id) {

DeviceRepairMaintenanceRecordVO deviceRepairMaintenanceRecordVO = iDeviceRepairMaintenanceRecordService.selectDeviceRepairMaintenanceRecordById(id);

return SaResult.ok().setData(deviceRepairMaintenanceRecordVO);

}

@PostMapping("/addDeviceRepairMaintenanceRecord")

@ApiOperation("新增设备保养维修记录")

public SaResult addDeviceRepairMaintenanceRecord(@ApiParam("设备保养维修记录DTO") @RequestBody @Valid DeviceRepairMaintenanceRecordDTO deviceRepairMaintenanceRecordDTO) {

return iDeviceRepairMaintenanceRecordService.addDeviceRepairMaintenanceRecord(deviceRepairMaintenanceRecordDTO) ? SaResult.ok() : SaResult.error();

}

@PutMapping("/updateDeviceRepairMaintenanceRecord")

@ApiOperation("修改设备保养维修记录")

public SaResult updateDeviceRepairMaintenanceRecord(@ApiParam("设备保养维修记录DTO") @RequestBody @Valid DeviceRepairMaintenanceRecordDTO deviceRepairMaintenanceRecordDTO) {

return iDeviceRepairMaintenanceRecordService.updateDeviceRepairMaintenanceRecord(deviceRepairMaintenanceRecordDTO) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteDeviceRepairMaintenanceRecord")

@ApiOperation("逻辑删除设备保养维修记录")

public SaResult deleteDeviceRepairMaintenanceRecord(@ApiParam("设备保养维修记录id") @RequestParam("id") String id) {

return iDeviceRepairMaintenanceRecordService.deleteDeviceRepairMaintenanceRecord(id) ? SaResult.ok() : SaResult.error();

}

}

@RestController

@RequestMapping("/device/room")

@Api(tags = "存放室管理接口")

@RequiredArgsConstructor

@Validated

public class DeviceRoomController extends BaseWebController {

private final DeviceRoomService deviceRoomService;

@GetMapping("/select")

@ApiOperation("分页查询存放室信息")

@ApiResponse(code = 200, response = DeviceRoomVO.class, message = "查询成功")

public SaResult select(CommonSelectEntity dto) {

dto.setTenantId(this.getTenantId());

return SaResult.ok().setData(deviceRoomService.findByPage(dto, true));

}

@GetMapping("/selectById")

@ApiOperation("根据主键查询存放室信息")

@ApiImplicitParam(name = "id", value = "主键", required = true)

public SaResult selectById(String id) {

return SaResult.ok().setData(deviceRoomService.selectById(id));

}

@PostMapping("/add")

@ApiOperation("添加存放室信息")

public SaResult addDeviceClassifyInfo(@RequestBody DeviceRoomDTO add) {

deviceRoomService.add(add);

return SaResult.ok();

}

@PostMapping("/update")

@ApiOperation("修改存放室信息")

public SaResult updateDeviceClassifyInfo(@RequestBody DeviceRoomDTO update) {

return deviceRoomService.update(update) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteByIds")

@ApiOperation("删除存放室信息")

public SaResult deleteDeviceClassifyInfo(String[] ids) {

deviceRoomService.logicDeleteByIds(ids, getTenantId());

return SaResult.ok().setData("删除成功！");

}

}

@RestController

@RequestMapping("/device/shelves")

@Api(tags = "货架管理")

@RequiredArgsConstructor

@Validated

public class DeviceShelvesInfoController extends BaseWebController {

private final DeviceShelvesInfoService deviceShelvesInfoService;

@GetMapping("/select")

@ApiOperation("分页查询货架信息")

@ApiResponse(code = 200, response = DeviceShelvesInfoVO.class, message = "查询成功")

public SaResult select(DeviceShelvesInfoDTO dto) {

dto.setTenantId(this.getTenantId());

return SaResult.ok().setData(deviceShelvesInfoService.findByPage(dto, true));

}

@GetMapping("/selectById")

@ApiOperation("根据id查询货架信息")

@ApiResponse(code = 200, response = SupplierContactsInfoVO.class, message = "查询成功")

public SaResult selectById(String id) {

return SaResult.ok().setData(deviceShelvesInfoService.selectById(id));

}

@PostMapping("/add")

@ApiOperation(value = "添加货架信息")

public SaResult add(@RequestBody DeviceShelvesAddInfoDTO add) {

return deviceShelvesInfoService.add(add) ? SaResult.ok() : SaResult.error();

}

@PostMapping("/update")

@ApiOperation("修改货架信息")

public SaResult update(@RequestBody DeviceShelvesInfoUpdateDTO update) {

return deviceShelvesInfoService.update(update) ? SaResult.ok() : SaResult.error();

}

@GetMapping("/deleteByIds")

@ApiOperation("删除货架信息")

public SaResult deleteByIds(@RequestParam("ids") String[] ids) {

deviceShelvesInfoService.logicDeleteByIds(ids, null);

return SaResult.ok().setData("删除成功！");

}

@GetMapping("/download")

@ApiOperation("导出货架信息")

public void downloadDeviceClassifyInfo(DeviceShelvesInfoDTO dto, HttpServletResponse response) {

Page<DeviceShelvesInfoVO> data = deviceShelvesInfoService.findByPage(dto, false);

dto.setTenantId(this.getTenantId());

try {

EasyExcelUtil.writeExcel(response, DeviceShelvesInfoVO.class, data.getRecords(), "货架信息" + System.currentTimeMillis() + ".xlsx");

} catch (IOException | IllegalAccessException | NoSuchFieldException e) {

throw new ServiceException(e);

}

}

}

/\*\*

\* 设备检定记录

\*

\* @author wanggang

\* @date 2023/5/10 15:46

\*/

@RestController

@RequestMapping("/deviceVerificationRecord")

@RequiredArgsConstructor

@Api(tags = "设备检定记录接口")

public class DeviceVerificationRecordController extends BaseWebController {

private final IDeviceVerificationRecordService iDeviceVerificationRecordService;

@GetMapping("/selectDeviceVerificationRecordPage")

@ApiOperation("分页查询设备检定记录")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceVerificationRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceVerificationRecordPage(@ApiParam("设备检定记录查询条件实体类") DeviceVerificationRecordSelectDTO deviceVerificationRecordSelectDTO) {

Page<DeviceVerificationRecordVO> deviceVerificationRecordPage = iDeviceVerificationRecordService.selectDeviceVerificationRecordPage(deviceVerificationRecordSelectDTO);

return SaResult.ok().setData(deviceVerificationRecordPage);

}

@GetMapping("/selectDeviceVerificationRecordById")

@ApiOperation("根据设备检定记录ID查询修改回显数据")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceVerificationRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceVerificationRecordById(@ApiParam("设备检定记录ID") @RequestParam("id") String id) {

DeviceVerificationRecordVO deviceVerificationRecordVO = iDeviceVerificationRecordService.selectDeviceVerificationRecordById(id);

return SaResult.ok().setData(deviceVerificationRecordVO);

}

@PostMapping("/addDeviceVerificationRecord")

@ApiOperation("新增设备检定记录")

public SaResult addDeviceVerificationRecord(@ApiParam("设备检定记录DTO") @RequestBody @Valid DeviceVerificationRecordDTO deviceVerificationRecordDTO) {

return iDeviceVerificationRecordService.addDeviceVerificationRecord(deviceVerificationRecordDTO) ? SaResult.ok() : SaResult.error();

}

@PutMapping("/updateDeviceVerificationRecord")

@ApiOperation("修改设备检定记录")

public SaResult updateDeviceVerificationRecord(@ApiParam("设备检定记录DTO") @RequestBody @Valid DeviceVerificationRecordDTO deviceVerificationRecordDTO) {

return iDeviceVerificationRecordService.updateDeviceVerificationRecord(deviceVerificationRecordDTO) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteDeviceVerificationRecord")

@ApiOperation("逻辑删除设备检定记录")

public SaResult deleteDeviceVerificationRecord(@ApiParam("设备检定记录id") @RequestParam("id") String id) {

return iDeviceVerificationRecordService.deleteDeviceVerificationRecord(id) ? SaResult.ok() : SaResult.error();

}

}

/\*\*

\* 设备运行记录

\*

\* @author wanggang

\* @date 2023/5/9 14:46

\*/

@RestController

@RequestMapping("/deviceWorkingRecord")

@RequiredArgsConstructor

@Api(tags = "设备运行记录接口")

public class DeviceWorkingRecordController extends BaseWebController {

private final IDeviceWorkingRecordService iDeviceWorkingRecordService;

@GetMapping("/selectDeviceWorkingRecordPage")

@ApiOperation("分页查询设备运行记录")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceWorkingRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceWorkingRecordPage(@ApiParam("设备运行记录查询条件实体类") DeviceWorkingRecordSelectDTO deviceWorkingRecordSelectDTO) {

Page<DeviceWorkingRecordVO> deviceWorkingRecordPage = iDeviceWorkingRecordService.selectDeviceWorkingRecordPage(deviceWorkingRecordSelectDTO);

return SaResult.ok().setData(deviceWorkingRecordPage);

}

@GetMapping("/selectDeviceWorkingRecordById")

@ApiOperation("根据设备运行记录ID查询修改回显数据")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceWorkingRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceWorkingRecordById(@ApiParam("设备运行记录ID") @RequestParam("id") String id) {

DeviceWorkingRecordVO deviceWorkingRecordVO = iDeviceWorkingRecordService.selectDeviceWorkingRecordById(id);

return SaResult.ok().setData(deviceWorkingRecordVO);

}

@PostMapping("/addDeviceWorkingRecord")

@ApiOperation("新增设备运行记录")

public SaResult addDeviceWorkingRecord(@ApiParam("设备运行记录DTO") @RequestBody @Valid DeviceWorkingRecordDTO deviceWorkingRecordDTO) {

return iDeviceWorkingRecordService.addDeviceWorkingRecord(deviceWorkingRecordDTO) ? SaResult.ok() : SaResult.error();

}

@PutMapping("/updateDeviceWorkingRecord")

@ApiOperation("修改设备运行记录")

public SaResult updateDeviceWorkingRecord(@ApiParam("设备运行记录DTO") @RequestBody @Valid DeviceWorkingRecordDTO deviceWorkingRecordDTO) {

return iDeviceWorkingRecordService.updateDeviceWorkingRecord(deviceWorkingRecordDTO) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteDeviceWorkingRecord")

@ApiOperation("逻辑删除设备运行记录")

public SaResult deleteDeviceWorkingRecord(@ApiParam("设备运行记录id") @RequestParam("id") String id) {

return iDeviceWorkingRecordService.deleteDeviceWorkingRecord(id) ? SaResult.ok() : SaResult.error();

}

}

@RestController

@RequestMapping("/supplier/contacts")

@Api(tags = "厂商联系人管理")

@RequiredArgsConstructor

@Validated

public class SupplierContactsInfoController extends BaseWebController {

private final SupplierContactsInfoService supplierContactsInfoService;

private final ISysDictCategoryService sysDictCategoryService;

@GetMapping("/select")

@ApiOperation("分页查询厂商联系人信息")

@ApiResponse(code = 200, response = SupplierContactsInfoVO.class, message = "查询成功")

public SaResult select(SupplierContactsInfoSelectDTO dto) {

dto.setTenantId(this.getTenantId());

return SaResult.ok().setData(supplierContactsInfoService.findByPage(dto, true));

}

@GetMapping("/selectById")

@ApiOperation("根据id查询 联系人信息")

@ApiResponse(code = 200, response = SupplierContactsInfoVO.class, message = "查询成功")

public SaResult selectById(@RequestParam("id") String id) {

return SaResult.ok().setData(supplierContactsInfoService.selectById(id));

}

@PostMapping("/add")

@ApiOperation("添加厂商联系人信息")

public SaResult add(@RequestBody SupplierContactsInfoDTO add) {

return supplierContactsInfoService.addSupplierContactsInfo(add) ? SaResult.ok() : SaResult.error();

}

@PostMapping("/update")

@ApiOperation("修改厂商联系人信息")

public SaResult update(@RequestBody SupplierContactsInfoDTO update) {

return supplierContactsInfoService.updateSupplierContactsInfo(update) ? SaResult.ok() : SaResult.error();

}

@GetMapping("/deleteByIds")

@ApiOperation("删除厂商联系人信息")

public SaResult deleteByIds(@RequestParam("ids") String[] ids) {

supplierContactsInfoService.logicDeleteByIds(ids, null, null);

return SaResult.ok().setData("删除成功！");

}

// // 提示：此接口没有使用、测试。没有使用场景。暂时保留

@PostMapping("/import")

@ApiOperation(value = "导入厂商联系人信息" , hidden = true)

public SaResult upload(@RequestParam(value = "file") MultipartFile file) {

this.checkFileSize(file);

try {

List<SupplierContactsInfoDTO> excelData = EasyExcelUtil.readExcel(file.getInputStream(), SupplierContactsInfoDTO.class, new EasyExcelUtil.ExcelListener<>());

if (CollUtil.isEmpty(excelData)) {

return SaResult.data("导入成功！");

}

// 批量导入数据

List<SupplierContactsInfo> collect = excelData.stream().map(SupplierContactsInfoDTO::transferEntity).toList();

supplierContactsInfoService.saveBatch(collect);

} catch (IOException e) {

throw new ServiceException(e);

}

return SaResult.ok();

}

@GetMapping("/downloadTemplate")

@ApiOperation("下载模板厂商联系人信息")

public void downloadTemplate(HttpServletResponse response) {

try {

EasyExcelUtil.writeExcel(response, SupplierContactsInfoDTO.class, Collections.emptyList(), "厂商联系人信息模板" + System.currentTimeMillis() + ".xlsx");

} catch (IOException | IllegalAccessException | NoSuchFieldException e) {

throw new ServiceException(e);

}

}

@GetMapping("/download")

@ApiOperation("导出厂商联系人信息")

public void downloadDeviceClassifyInfo(SupplierContactsInfoSelectDTO dto, HttpServletResponse response) {

Page<SupplierContactsInfoVO> deviceClassifyByPage = supplierContactsInfoService.findByPage(dto, false);

dto.setTenantId(this.getTenantId());

// 获取性别字典

Map<String, String> gender = sysDictCategoryService.selectDictBycategoryKey(null, "gender" , true);

if (CollectionUtils.isEmpty(gender)) {

throw new ServiceException("性别字典没有数据，请联系管理员添加");

}

List<SupplierContactsInfoDTO> collect = deviceClassifyByPage.getRecords().stream().map(c -> {

SupplierContactsInfoDTO data = c.transferDto();

data.setGender(gender.get(String.valueOf(data.getGender())));

return data;

}).toList();

try {

EasyExcelUtil.writeExcel(response, SupplierContactsInfoDTO.class, collect, "厂商联系人信息" + System.currentTimeMillis() + ".xlsx");

} catch (IOException | IllegalAccessException | NoSuchFieldException e) {

throw new ServiceException(e);

}

}

}

@RestController

@RequestMapping("/supplier")

@Api(tags = "厂商管理")

@RequiredArgsConstructor

@Validated

public class SupplierInfoController extends BaseWebController {

private final SupplierInfoService supplierInfoService;

private final SupplierContactsInfoService contactInfoService;

@GetMapping("/select")

@ApiOperation(value = "分页查询厂商和联系人信息", notes = "分页查询，响应参数省略了分页数据")

@ApiResponse(code = 200, response = SupplierInfoVO.class, message = "查询成功")

public SaResult select(SupplierInfoSelectDTO dto) {

dto.setTenantId(this.getTenantId());

return SaResult.ok().setData(supplierInfoService.findByPage(dto, true));

}

@GetMapping("/selectSupplier")

@ApiOperation(value = "分页查询厂商信息", notes = "分页查询")

@ApiResponse(code = 200, response = SupplierInfoVO.class, message = "查询成功")

public SaResult selectSupplier(SupplierInfoSelectDTO dto) {

dto.setTenantId(this.getTenantId());

return SaResult.ok().setData(supplierInfoService.findSupplierByPage(dto, true));

}

@GetMapping("/selectById")

@ApiOperation("根据id查询 厂商和联系人信息")

@ApiResponse(code = 200, response = SupplierInfoDTO.class, message = "查询成功")

@ApiImplicitParam(value = "id", example = "1656847984220512257", required = true)

public SaResult selectById(@NotBlank(message = "主键不能为空") String id) {

SupplierInfo byId = supplierInfoService.getById(id);

SupplierInfoDTO dto = BeanUtil.copyProperties(byId, SupplierInfoDTO.class);

dto.setContactList(contactInfoService.selectListBySupplierId(id));

return SaResult.ok().setData(dto);

}

@PostMapping("/add")

@ApiOperation("添加厂商和联系人信息")

@ApiResponse(code = 200, response = SaResult.class, message = "查询成功")

public SaResult add(@RequestBody SupplierInfoDTO add) {

return supplierInfoService.addSupplierInfo(add) ? SaResult.ok() : SaResult.error();

}

@PostMapping("/update")

@ApiOperation(value = "修改厂商信息", notes = "注意这个接口只能修改厂商信息")

@ApiResponse(code = 200, response = SaResult.class, message = "查询成功")

public SaResult update(@RequestBody SupplierInfoUpdateDTO update) {

return supplierInfoService.updateSupplierInfo(update) ? SaResult.ok() : SaResult.error();

}

@GetMapping("/deleteByIds")

@ApiOperation("删除厂商和联系人信息")

@ApiResponse(code = 200, response = SaResult.class, message = "查询成功")

@ApiImplicitParam(name = "ids", example = "1656847984220512257,1656847984220512290")

public SaResult deleteByIds(@RequestParam(name = "ids") String[] ids) {

if (ids == null || ids.length == 0) {

return SaResult.ok().setData("删除成功！");

}

supplierInfoService.logicDelete(ids);

return SaResult.ok().setData("删除成功！");

}

@PostMapping("/import")

@ApiOperation(value = "导入厂商和联系人分类信息", notes = "isOverride 和 isFirstDelete 参数只能有一个为true")

@ApiImplicitParams({

@ApiImplicitParam(name = "file", value = "文件", required = true),

@ApiImplicitParam(name = "isOverride", value = "是否覆盖已经存在的数据"),

@ApiImplicitParam(name = "isFirstDelete", value = "导入之前是否删除数据")

})

public SaResult upload(@RequestParam(value = "file") MultipartFile file, Boolean isOverride, Boolean isFirstDelete) {

this.checkFileSize(file);

if ((Boolean.TRUE.equals(isOverride) && Boolean.TRUE.equals(isFirstDelete)) ||

Boolean.FALSE.equals(isOverride) && Boolean.FALSE.equals(isFirstDelete)) {

return SaResult.error("是否覆盖和是否删除参数不能同时选中或不选");

}

try {

List<SupplierInfoVO> excelData = EasyExcelUtil.readExcel(file.getInputStream(), SupplierInfoVO.class, new EasyExcelUtil.ExcelListener<>());

supplierInfoService.importData(excelData, isOverride, isFirstDelete);

} catch (IOException e) {

throw new ServiceException(e);

}

return SaResult.ok();

}

@GetMapping("/downloadTemplate")

@ApiOperation("下载模板厂商和联系人")

public void downloadTemplate(HttpServletResponse response) {

try {

EasyExcelUtil.writeExcel(response, SupplierInfoVO.class, Collections.emptyList(), "厂商和联系人信息模板" + System.currentTimeMillis() + ".xlsx");

} catch (IOException | IllegalAccessException | NoSuchFieldException e) {

throw new ServiceException(e);

}

}

@GetMapping("/download")

@ApiOperation("导出厂商和联系人数据")

public void download(SupplierInfoSelectDTO dto, HttpServletResponse response) {

supplierInfoService.exportData(dto, response);

}

}

/\*\*

\* 历史信息服务

\* @author wanggang

\* @date 2023/4/15 10:30

\*

\*/

@Api(tags = "历史信息服务")

@RestController

@RequestMapping(value = "/flowableHistory")

public class FlowableHistoryController {

@Autowired

IFlowableHistoryService iFlowableHistoryService;

@ApiOperation("查询所有历史任务信息")

@GetMapping(value = "/selectHistoryTask")

public SaResult selectHistoryTask() {

List<HistoricTaskInstanceDTO> historicTaskInstanceDTOList = iFlowableHistoryService.selectHistoryTask();

return SaResult.ok().setData(historicTaskInstanceDTOList);

}

@ApiOperation("查询已完成历史任务信息")

@GetMapping(value = "/selectHistoryTaskFinish")

public SaResult selectHistoryTaskFinish() {

List<HistoricTaskInstanceDTO> historicTaskInstanceDTOList = iFlowableHistoryService.selectHistoryTaskFinish();

return SaResult.ok().setData(historicTaskInstanceDTOList);

}

}

/\*\*

\* 初始化用户服务

\*

\* @author wanggang

\* @date 2023/4/15 10:30

\*/

@Api(tags = "初始化用户服务")

@RestController

@RequestMapping(value = "/flowableIdentity")

public class FlowableIdentityController {

@Autowired

IFlowableIdentityService iFlowableIdentityService;

/\*\*

\* 新增组

\*

\* @param groupDTO

\* @return void

\* @author wanggang

\* @date 2023/4/14 17:10

\*/

@ApiOperation("新增用户组")

@PostMapping("/addGroup")

public SaResult addGroup(@ApiParam("组DTO") @Valid @RequestBody GroupDTO groupDTO) {

iFlowableIdentityService.addGroup(groupDTO);

return SaResult.ok();

}

/\*\*

\* 查询组

\*

\* @param param

\* @return void

\* @author wanggang

\* @date 2023/4/14 17:10

\*/

@ApiOperation("查询组")

@GetMapping("/selectGroup")

public SaResult selectGroup(@ApiParam("查询参数(可以传组ID、组Name、用户ID、不传为全查询)")

@RequestParam(value = "param", required = false) String param) {

List<GroupDTO> groups = iFlowableIdentityService.selectGroup(param);

return SaResult.ok().setData(groups);

}

/\*\*

\* 添加用户和组的关联

\*

\* @param groupAndUserDTOList

\* @return void

\* @author wanggang

\* @date 2023/4/17 10:32

\*/

@ApiOperation("添加用户和组的关联")

@PostMapping("/addUserAndGroup")

public SaResult addUserAndGroupRelationBatch(@ApiParam("用户和组") @RequestBody List<GroupAndUserDTO> groupAndUserDTOList) {

if (null == groupAndUserDTOList || groupAndUserDTOList.isEmpty()) {

throw new ParameterException("参数缺失");

}

iFlowableIdentityService.addUserAndGroupRelationBatch(groupAndUserDTOList);

return SaResult.ok();

}

/\*\*

\* 修改组名信息

\*

\* @param groupDTO

\* @return void

\* @author wanggang

\* @date 2023/4/17 17:10

\*/

@ApiOperation("修改组信息")

@PutMapping("/updateGroup")

public SaResult updateGroup(@ApiParam("组DTO") @Valid @RequestBody GroupDTO groupDTO) {

iFlowableIdentityService.updateGroup(groupDTO);

return SaResult.ok();

}

/\*\*

\* 删除组

\*

\* @param groupId

\* @return void

\* @author wanggang

\* @date 2023/4/17 17:10

\*/

@ApiOperation("删除组")

@DeleteMapping("/deleteGroup")

public SaResult deleteGroup(@ApiParam("组Id") @RequestParam(value = "groupId") String groupId) {

iFlowableIdentityService.deleteGroup(groupId);

return SaResult.ok();

}

/\*\*

\* 新增用户

\*

\* @param userDTO

\* @return void

\* @author wanggang

\* @date 2023/4/17 17:10

\*/

@ApiOperation("新增用户")

@PostMapping("/addUser")

public SaResult addUser(@ApiParam("用户DTO") @Valid @RequestBody UserDTO userDTO) {

iFlowableIdentityService.addUser(userDTO);

return SaResult.ok();

}

/\*\*

\* 修改除密码外的其他信息

\*

\* @param userDTO

\* @return void

\* @author wanggang

\* @date 2023/4/19 16:41

\*/

@ApiOperation("修改用户")

@PutMapping("/updateUser")

public SaResult updateUser(@ApiParam("用户DTO") @Valid @RequestBody UserDTO userDTO) {

iFlowableIdentityService.updateUser(userDTO);

return SaResult.ok();

}

/\*\*

\* 删除用户

\*

\* @param userId

\* @return void

\* @author wanggang

\* @date 2023/4/19 16:46

\*/

@ApiOperation("删除用户")

@DeleteMapping("/deleteUser")

public SaResult deleteUser(@ApiParam("用户Id") @RequestParam(value = "userId") String userId) {

iFlowableIdentityService.deleteUser(userId);

return SaResult.ok();

}

/\*\*

\* 查询用户

\*

\* @param param

\* @return void

\* @author wanggang

\* @date 2023/4/18 9:10

\*/

@ApiOperation("查询用户")

@GetMapping("/selectUser")

public SaResult selectUser(@ApiParam("查询参数(可以传用户ID、用户显示名、不传为全查询)")

@RequestParam(value = "param", required = false) String param) {

List<UserDTO> users = iFlowableIdentityService.selectUser(param);

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

}

/\*\*

\* 删除用户和组的关联关系

\*

\* @param groupAndUserDTO

\* @return void

\* @author wanggang

\* @date 2023/4/19 16:46

\*/

@ApiOperation("删除用户和组的关联关系")

@DeleteMapping("/deleteUserAndGroupRelation")

public SaResult deleteUserAndGroupRelation(@ApiParam("用户和组DTO") @Valid @RequestBody GroupAndUserDTO groupAndUserDTO) {

iFlowableIdentityService.deleteUserAndGroupRelation(groupAndUserDTO);

return SaResult.ok();

}

}

/\*\*

\* 流程定义和部署服务

\*

\* @author wanggang

\* @date 2023/4/15 10:33

\*/

@Api(tags = "流程定义和部署服务")

@RestController

@RequestMapping(value = "/flowableRepository")

public class FlowableRepositoryController {

@Autowired

IFlowableRepositoryService iFlowableRepositoryService;

@ApiOperation("根据flowId和resourceName查询流程图或xml")

@GetMapping(value = "/loadFlowXml")

public void loadFlowXml(HttpServletResponse httpServletResponse,

@ApiParam("流程Id") @RequestParam(value = "flowId") String flowId,

@ApiParam("流程图名") @RequestParam(value = "resourceName") String resourceName) {

iFlowableRepositoryService.loadFlowXml(httpServletResponse, flowId, resourceName);

}

// @ApiOperation("下载资源文件")

// @GetMapping(value = "/downloadFlowXml")

// public void downloadFlowXml(@ApiParam("流程Id") @RequestParam(value = "flowId") String flowId,

// @ApiParam("流程图名") @RequestParam(value = "resourceName") String resourceName,

// HttpServletRequest request, HttpServletResponse response) {

// iFlowableRepositoryService.downloadFlowXml(flowId, resourceName, request, response);

// }

@ApiOperation("查询流程定义信息(不传参数为全查询)")

@GetMapping(value = "/selectProcessDefinition")

public SaResult selectProcessDefinition(@ApiParam("流程名称") @RequestParam(value = "name", required = false) String name,

@ApiParam("流程key") @RequestParam(value = "key", required = false) String key) {

List<ProcessDefinitionDTO> processDefinitions = iFlowableRepositoryService.selectProcessDefinition(name, key);

return SaResult.ok().setData(processDefinitions);

}

}

@Api(tags = "流程相关操作接口")

@RestController

@RequestMapping(value = "/flowableTask")

public class FlowableTaskController extends BaseWebController {

@Autowired

IFlowableTaskService flowableTaskService;

@Autowired

IFlowFactory flowFactory;

@Autowired

ISysOrgService sysOrgService;

@Autowired

IFlowBaseInfoService flowBaseInfoService;

@Autowired

ISysUserService sysUserService;

@ApiOperation("查询待办任务列表")

@GetMapping(value = "/findMyTask")

@ApiResponses({

@ApiResponse(code = 200,message = "返回成功",response = FlowBaseInfo.class)

})

public SaResult getFlowableMyTaskByUserId(@RequestParam Integer current,

@RequestParam Integer pageSize,

String searchKey,

String processKey,

String processState,

String startTime,

String endTime

) {

SysUser sysUser = getUserDetail();

IPage<FlowBaseInfo> page = flowableTaskService.findPageFlowBaseInfo(current, pageSize, sysUser, getMainFlowRoles(), getSlaveFlowRoles(),searchKey,processKey,processState,startTime,endTime);

return SaResult.ok().setData(page);

}

@ApiOperation("审核通过任务")

@PostMapping(value = "/completeTask")

public SaResult completeTask(@Validated @RequestBody CompleteTaskDTO completeTaskDTO) {

//

SysUser sysUser=getUserDetail();

if (Strings.isNullOrEmpty(sysUser.getSignatureBase64())){

return SaResult.error("您还未设置签名，无法审核，请到右上角个人中心设置签名");

}

flowableTaskService.completeTask(completeTaskDTO, getMainFlowRoles(), getSlaveFlowRoles(), getAllFlowRoleCodes());

return SaResult.ok().setMsg("审核成功");

}

@ApiOperation("拒绝任务，驳回到任务发起人")

@PostMapping(value = "/rejectTask")

public SaResult myCompletedProcessPage(@Validated @RequestBody RejectTaskDTO rejectTaskDTO) {

//

//判断当前流程的审核人和发起人都是登登录人才能修改

if (!flowableTaskService.canApproval(rejectTaskDTO.getProcessId(), getMainFlowRoles(), getSlaveFlowRoles(), getAllFlowRoleCodes())) {

return SaResult.error("任务审核人不是您,无法拒绝");

}

flowableTaskService.rejectTask(rejectTaskDTO);

return SaResult.data(true).setMsg("拒绝成功,已回到申请人节点");

}

@ApiOperation("查询已经办理的流程")

@GetMapping("myCompletedProcessPage")

@ApiResponses({

@ApiResponse(code = 200,message = "返回成功",response = FlowBaseInfo.class)

})

public SaResult myCompletedProcessPage(@RequestParam Integer current,

@RequestParam Integer pageSize,

String searchKey,

@ApiParam("流程key") String processKey,

@ApiParam("流程状态") String processState,

String startTime, String endTime) {

//

SysUser sysUser = TenantContextHolder.getContext().getTenantContextInfo().getSysUser();

IPage<FlowBaseInfo> flowBaseInfoIPage = flowBaseInfoService.findCompletedProcessPage(new Page<>(current, pageSize), sysUser.getId()

,processState,processKey,searchKey,startTime,endTime);

return SaResult.data(flowBaseInfoIPage).setMsg("查询成功");

}

@ApiOperation("查询我发起的流程")

@GetMapping("myStartProcessPage")

@ApiResponses({

@ApiResponse(code = 200,message = "返回成功",response = FlowBaseInfo.class)

})

public SaResult myStartProcessPage(@RequestParam Integer current,

@RequestParam Integer pageSize,

String searchKey,

@ApiParam("流程key") String processKey,

@ApiParam("流程状态") String processState,String startTime, String endTime) {

SysUser sysUser = TenantContextHolder.getContext().getTenantContextInfo().getSysUser();

IPage<FlowBaseInfo> flowBaseInfoIPage = flowBaseInfoService.page(new Page<>(current, pageSize), new QueryWrapper<FlowBaseInfo>() .lambda()

.eq(!Strings.isNullOrEmpty(processState)&&ProcessState.match(processState)!=null,FlowBaseInfo::getState,ProcessState.match(processState))

.eq(!Strings.isNullOrEmpty(processKey),FlowBaseInfo::getProcessKey,processKey)

.eq(BaseEntity::getCreateBy, sysUser.getId()).orderByDesc(BaseEntity::getCreateTime)

.and(!Strings.isNullOrEmpty(searchKey), flowBaseInfoLambdaQueryWrapper -> flowBaseInfoLambdaQueryWrapper.like(FlowBaseInfo::getProcessId, searchKey)

.or().like(FlowBaseInfo::getProcessName, searchKey)

.or().like(FlowBaseInfo::getCreateName, searchKey)

.or().like(FlowBaseInfo::getSysOrgFullName, searchKey))

.gt(!Strings.isNullOrEmpty(startTime),BaseEntity::getCreateTime,startTime)

.lt(!Strings.isNullOrEmpty(endTime),BaseEntity::getCreateTime,endTime)

);

return SaResult.data(flowBaseInfoIPage).setMsg("查询成功");

}

@ApiOperation("查询流程的审核记录")

@GetMapping("processHis/{processId}")

@ApiResponses({

@ApiResponse(code = 200,message = "返回成功",response = FlowHistVO.class)

})

public SaResult processHis(@PathVariable String processId) {

List<FlowHistVO> flowHistVOS =flowableTaskService.findFlowHistVOs(processId);

return SaResult.data(flowHistVOS).setMsg("查询成功");

}

/\*\*

\* 获取流程图并显示当前流程

\*

\* @param httpServletResponse, processId

\* @return void

\* @author wanggang

\* @date 2023/4/18 11:13

\*/

@ApiOperation("获取流程图并显示当前流程")

@GetMapping(value = "/processDiagram")

public void genProcessDiagram(HttpServletResponse httpServletResponse,

@ApiParam("流程Id") @RequestParam(value = "processId") String processId) throws Exception {

flowableTaskService.genProcessDiagram(httpServletResponse, processId);

}

// /\*\*

// \* 根据任务Id获取流程图并显示当前流程

// \*

// \* @param httpServletResponse, taskId

// \* @return void

// \* @author wanggang

// \* @date 2023/4/18 11:13

// \*/

// @ApiOperation("根据任务Id获取流程图并显示当前流程")

// @GetMapping(value = "/processDiagramByTaskId")

// public void genProcessDiagramByTaskId(HttpServletResponse httpServletResponse,

// @ApiParam("任务Id") @RequestParam(value = "taskId") String taskId) throws Exception {

// flowableTaskService.genProcessDiagramByTaskId(httpServletResponse, taskId);

// }

}

/\*\*

\* 检定校验申请记录

\*

\* @author wanggang

\* @date 2023/6/2 14:46

\*/

@RestController

@RequestMapping("/flowDeviceVerificationRecord")

@Api(tags = "检定校验申请记录接口")

public class FlowDeviceVerificationRecordController extends BaseWebController {

@Resource

IFlowDeviceVerificationRecordService iFlowDeviceVerificationRecordService;

@Resource

IFlowDeviceVerificationService iFlowDeviceVerificationService;

@GetMapping("/selectFlowDeviceVerificationRecordPage")

@ApiOperation("分页查询检定校准申请流水记录")

@ApiResponses({

@ApiResponse(code = 200, response = FlowDeviceVerificationVO.class, message = "查询成功")

})

public SaResult selectFlowDeviceVerificationRecordPage(@ApiParam("检定校准申请流水记录查询条件实体类") FlowDeviceVerificationRecordSelectDTO flowDeviceVerificationRecordSelectDTO) {

Page<FlowDeviceVerificationVO> flowDeviceVerificationPage = iFlowDeviceVerificationService.selectFlowDeviceVerificationRecordPage(flowDeviceVerificationRecordSelectDTO);

return SaResult.ok().setData(flowDeviceVerificationPage);

}

@PostMapping("/storageFlowDeviceVerificationRecord")

@ApiOperation("暂存检定校验申请记录")

public SaResult storageFlowDeviceVerificationRecord(@ApiParam("检定校验申请记录DTO") @RequestBody List<FlowDeviceVerificationDTO> flowDeviceVerificationDTOList) {

if (CollUtil.isEmpty(flowDeviceVerificationDTOList)) {

throw new ParameterException("修改参数缺失!");

}

return iFlowDeviceVerificationService.updateFlowDeviceVerificationBatch(flowDeviceVerificationDTOList) ? SaResult.ok() : SaResult.error();

}

@GetMapping("/selectByProcessRecordId")

@ApiOperation("根据流程记录ID查询明细")

@ApiResponses({

@ApiResponse(code = 200, response = FlowDeviceVerificationVO.class, message = "查询成功")

})

public SaResult selectByProcessRecordId(@ApiParam("流程记录ID") @RequestParam("processRecordId") String processRecordId) {

List<FlowDeviceVerificationVO> flowDeviceVerificationVO = iFlowDeviceVerificationService.selectByProcessRecordId(processRecordId);

return SaResult.ok().setData(flowDeviceVerificationVO);

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/6/2 15:13

\*/

@RestController

@RequestMapping("flowProcessBiz")

@Api(tags = "流程相关操作接口")

public class FlowProcessBizController extends BaseWebController {

@Autowired

FlowBaseHandlerFactory flowBaseHandlerFactory;

@Autowired

IFlowFactory flowFactory;

@Autowired

IFlowableTaskService flowableTaskService;

@Autowired

IFlowBaseInfoService flowBaseInfoService;

/\*\*

\* @return

\*/

@ApiOperation(value = "创建流程传入参数格式{processKey:'',processName:'',......}")

@PostMapping("createProcess")

public SaResult createProcess(@RequestBody @Validated JSONObject jsonObject) {

if (Strings.isNullOrEmpty(jsonObject.getString("processKey"))) {

return SaResult.error("请选择需要发起的流程key");

}

FlowBaseFactory flowBaseFactory = flowBaseHandlerFactory.createHandler(jsonObject.getString("processKey"));

flowBaseFactory.createProcess(jsonObject, null);

return SaResult.data(true).setMsg("流程创建成功");

}

@ApiOperation("获取流程业务数据")

@GetMapping("getProcessBiz/{processId}")

@ApiResponses({

@ApiResponse(code = 200, response = FlowBizVO.class, message = "流程业务数据")

})

public SaResult getProcessBiz(@PathVariable String processId) {

String processKey = "";

boolean isEnd=false;

ProcessInstance processInstance = flowFactory.getRuntimeService().createProcessInstanceQuery().processInstanceId(processId).singleResult();

if (processInstance == null) {

HistoricProcessInstance historicProcessInstance = flowFactory.getHistoryService().createHistoricProcessInstanceQuery().processInstanceId(processId).singleResult();

if (historicProcessInstance == null) {

return SaResult.error("不存在的流程");

} else {

isEnd=true;

processKey = historicProcessInstance.getProcessDefinitionKey();

}

} else {

processKey = processInstance.getProcessDefinitionKey();

}

FlowBaseFactory flowBaseFactory = flowBaseHandlerFactory.createHandler(processKey);

FlowBizVO flowBizVO = new FlowBizVO();

flowBizVO.setCanUpdate(flowableTaskService.canUpdate(processId, getMainFlowRoles(), getSlaveFlowRoles(), getAllFlowRoleCodes()));

FlowBaseInfo flowBaseInfo = flowBaseInfoService.getOne(new QueryWrapper<FlowBaseInfo>().lambda().eq(FlowBaseInfo::getProcessId, processId));

flowBizVO.setCanDelete(getUserDetail().getId().equals(flowBaseInfo.getCreateBy())&& !isEnd);

flowBizVO.setCanApproval(flowableTaskService.canApproval(processId, getMainFlowRoles(), getSlaveFlowRoles(), getAllFlowRoleCodes()));

flowBizVO.setCanReject(flowBizVO.getCanApproval() && !flowBizVO.getCanDelete());

flowBizVO.setBizObj(flowBaseFactory.getProcessBiz(processId));

flowBizVO.setProcessKey(ProcessKey.match(processKey));

return SaResult.data(flowBizVO).setMsg("查询成功");

}

@ApiOperation(value = "修改业务数据,传入格式为{processId:'xxxx',.....}")

@PutMapping("updateProcessBiz")

public SaResult updateProcessBiz(@RequestBody @Validated JSONObject flowBaseInfoUpdateDTO) {

SysUser sysUser=getUserDetail();

if (Strings.isNullOrEmpty(sysUser.getSignatureBase64())){

return SaResult.error("您还未设置签名，请到右上角个人中心设置签名");

}

ProcessInstance processInstance = flowFactory.getRuntimeService().createProcessInstanceQuery().processInstanceId(flowBaseInfoUpdateDTO.getString("processId")).singleResult();

if (processInstance == null) {

return SaResult.error("不存在的流程");

}

//判断当前流程的审核人和发起人都是登登录人才能修改

if (!flowableTaskService.canUpdate(processInstance.getProcessInstanceId(), getMainFlowRoles(), getSlaveFlowRoles(), getAllFlowRoleCodes())) {

return SaResult.error("您不是当前流程的发起人且任务审核人不是您,无法修改");

}

FlowBaseFactory flowBaseFactory = flowBaseHandlerFactory.createHandler(processInstance.getProcessDefinitionKey());

flowBaseFactory.updateProcessBiz(flowBaseInfoUpdateDTO, processInstance.getProcessInstanceId());

CompleteTaskDTO completeTaskDTO=new CompleteTaskDTO();

completeTaskDTO.setProcessId(processInstance.getProcessInstanceId());

completeTaskDTO.setComment("再次发起申请");

flowableTaskService.completeTask(completeTaskDTO, getMainFlowRoles(), getSlaveFlowRoles(), getAllFlowRoleCodes());

return SaResult.data(null).setMsg("修改成功");

}

@ApiOperation("废弃流程")

@PostMapping("deleteProcess")

@Transactional

public SaResult deleteProcess(@RequestBody @Validated DeleteProcessDTO deleteProcessDTO) {

ProcessInstance processInstance = flowFactory.getRuntimeService().createProcessInstanceQuery().processInstanceId(deleteProcessDTO.getProcessId()).singleResult();

if (processInstance==null){

return SaResult.error("流程已经结束或不存在的流程！");

}

FlowBaseInfo flowBaseInfo = flowBaseInfoService.getOne(new QueryWrapper<FlowBaseInfo>().lambda().eq(FlowBaseInfo::getProcessId, deleteProcessDTO.getProcessId()));

if (!getUserDetail().getId().equals(flowBaseInfo.getCreateBy())) {

return SaResult.error("您不是当前流程发起人，无法废弃流程！");

}

//

flowFactory.getRuntimeService().deleteProcessInstance(deleteProcessDTO.getProcessId(), deleteProcessDTO.getDeleteReason());

flowBaseInfo.setState(ProcessState.REVOKE);

flowBaseInfoService.updateById(flowBaseInfo);

return SaResult.data(true).setMsg("废弃成功");

}

@ApiOperation("查询全部流程定义")

@GetMapping("flowProcessKeys")

public SaResult flowProcessKeys() {

return SaResult.data(ProcessKey.values());

}

@ApiOperation("查询全部流程状态定义")

@GetMapping("flowProcessStates")

public SaResult flowProcessStates() {

return SaResult.data(ProcessState.values());

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/27 17:10

\*/

@RestController

public class H5Controller {

@Autowired

ISysUserService sysUserService;

// 当前是否登录

@RequestMapping("/sso/isLogin")

@TenantIgnore

public Object isLogin() {

return SaResult.data(StpUtil.isLogin());

}

// 返回SSO认证中心登录地址

@RequestMapping("/sso/getSsoAuthUrl")

@TenantIgnore

public SaResult getSsoAuthUrl(String clientLoginUrl, String back) {

String serverAuthUrl = SaSsoUtil.buildServerAuthUrl(clientLoginUrl, back);

return SaResult.data(serverAuthUrl);

}

// 根据ticket进行登录

@RequestMapping("/sso/doLoginByTicket")

@TenantIgnore

public SaResult doLoginByTicket(String ticket) {

String username = (String) SaSsoProcessor.instance.checkTicket(ticket, "/sso/doLoginByTicket");

if (username != null) {

//查询在当前系统是否存在相同账号

SysUser sysUser = sysUserService.findSysUserByUsername(username);

if (sysUser == null) {

return SaResult.error("当前系统无认证中心统一账户");

}

StpUtil.login(username);

return SaResult.data(StpUtil.getTokenValue());

}

return SaResult.error("无效ticket：" + ticket);

}

}

/\*\*

\* Sa-Token-SSO Client端 Controller

\*

\* @author kong

\*/

@RestController

@Slf4j

public class SsoClientController {

/\*

\* SSO-Client端：处理所有SSO相关请求

\* http://{host}:{port}/sso/login -- Client端登录地址，接受参数：back=登录后的跳转地址

\* http://{host}:{port}/sso/logout -- Client端单点注销地址（isSlo=true时打开），接受参数：back=注销后的跳转地址

\* http://{host}:{port}/sso/logoutCall -- Client端单点注销回调地址（isSlo=true时打开），此接口为框架回调，开发者无需关心

\*/

@RequestMapping("/sso/\*")

@TenantIgnore

public Object ssoRequest() {

return SaSsoProcessor.instance.clientDister();

}

// 配置SSO相关参数

@Autowired

@TenantIgnore

private void configSso(SaSsoConfig sso) {

// 配置Http请求处理器

sso.setSendHttp(url -> {

log.debug("------ 发起请求：" + url);

return Forest.get(url).executeAsString();

});

}

// 查询我的账号信息

@RequestMapping("/sso/myinfo")

@TenantIgnore

public Object myinfo() {

Object userinfo = SaSsoUtil.getUserinfo(StpUtil.getLoginId());

log.debug("--------info：" + userinfo);

return userinfo;

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/3/17 11:21

\*/

@RestController

@RequestMapping("file")

public class FileUploadController {

@Autowired

MinioUtil minioUtil;

@PostMapping("upload")

public Object upload(@RequestParam(value = "file") MultipartFile file) throws IOException {

String url = minioUtil.upload(file);

return SaResult.data(url);

}

@PostMapping("/uploadFile")

public SaResult uploadFile(@RequestParam(value = "file") MultipartFile file) {

return SaResult.ok().setData(minioUtil.uploadFile(file));

}

@GetMapping("down")

public void down(String fileName, HttpServletResponse response) {

minioUtil.download(fileName, response);

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/5/15 15:57

\*/

@RestController

@RequestMapping("sysDictCategory")

@Api(tags = "字典分类")

@RequiredArgsConstructor

public class SysDictCategoryController extends BaseWebController {

private final ISysDictCategoryService sysDictCategoryService;

private final ISysDictService sysDictService;

@GetMapping("listPage")

public Object listPage(@RequestParam(defaultValue = "1") Integer current,

@RequestParam(defaultValue = "10") Integer pageSize,

String categoryName,

String categoryKey) {

// String tenantId = this.getTenantId();

LambdaQueryWrapper<SysDictCategory> queryWrapper = new QueryWrapper<SysDictCategory>().lambda()

.like(!Strings.isNullOrEmpty(categoryName), SysDictCategory::getCategoryName, categoryName)

.like(!Strings.isNullOrEmpty(categoryKey), SysDictCategory::getCategoryKey, categoryKey)

.orderByDesc(SysDictCategory::getCreateTime);

// if (!SystemConstant.SYSTEM\_TENANT\_ID.equals(tenantId)) {

// queryWrapper.eq(true, SysDictCategory::getTenantId, tenantId);

// }

Page<SysDictCategory> page = sysDictCategoryService.page(new Page<>(current, pageSize), queryWrapper);

return SaResult.data(page);

}

/\*\*

\* 添加数据字典

\*

\* @return

\*/

@PostMapping("add")

public SaResult add(@RequestBody SysDictCategory sysDictCategory) {

long count=sysDictCategoryService.count(new QueryWrapper<SysDictCategory>().lambda()

.eq(SysDictCategory::getCategoryKey,sysDictCategory.getCategoryKey())

);

if (count>0){

throw new JcglException("当前字典类别key不能重复");

}

sysDictCategory.setTenantId(this.getTenantId());

return SaResult.ok("添加成功").setData(sysDictCategoryService.save(sysDictCategory));

}

@ApiOperation("修改数据字典分类")

@PostMapping("update")

@Transactional(rollbackFor = JcglException.class)

public SaResult update(@RequestBody SysDictCategory sysDictCategory) {

sysDictCategoryService.updateById(sysDictCategory);

long count=sysDictCategoryService.count(new QueryWrapper<SysDictCategory>().lambda()

.eq(SysDictCategory::getCategoryKey,sysDictCategory.getCategoryKey())

);

if (count>1){

throw new JcglException("当前字典类别key不能重复");

}

List<SysDict> sysDictList = sysDictService.list(new QueryWrapper<SysDict>().lambda().eq(SysDict::getSysDictCategoryId, sysDictCategory.getId()));

if (sysDictList != null && !sysDictList.isEmpty()) {

sysDictList.forEach(sysDict -> {

sysDict.setType(sysDictCategory.getCategoryKey());

sysDictService.updateById(sysDict);

});

}

return SaResult.ok("修改成功");

}

@DeleteMapping("delete/{id}")

public SaResult update(@PathVariable String id) {

if (sysDictService.count(new QueryWrapper<SysDict>().lambda().eq(SysDict::getSysDictCategoryId, id)) > 0) {

throw new JcglException("当前数据字典存在不能删除!");

}

return SaResult.ok("删除成功").setData(sysDictCategoryService.removeById(id));

}

@GetMapping("/initDictData")

@ApiOperation("初始化当前租户字典数据或者系统升级")

public SaResult initdefault() {

String tenantId = TenantContextHolder.getContext().getTenantContextInfo().getTenantId();

sysDictCategoryService.initDictData(tenantId);

return SaResult.ok();

}

@GetMapping("/dictDataReset")

@ApiOperation("字典数据初始化为系统数据")

public SaResult dictDataReset() {

String tenantId = TenantContextHolder.getContext().getTenantContextInfo().getTenantId();

sysDictCategoryService.dictDataReset(tenantId);

return SaResult.ok();

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/15 9:17

\*/

@RestController

@RequestMapping("sysDict")

@RequiredArgsConstructor

@Api(tags = "数据字典")

public class SysDictController extends BaseWebController {

private final ISysDictService sysDictService;

private final ISysDictCategoryService sysDictCategoryService;

/\*\*

\* 查询全部数据字典

\*

\* @return

\*/

@ApiOperation("查询全部")

@GetMapping("listAll")

public Object listAll() {

List<SysDict> sysDictList=sysDictService.findListAll(getTenantId());

return SaResult.data(sysDictList);

}

@ApiOperation("分页查询")

@GetMapping("listPage")

public Object listPage(@RequestParam(defaultValue = "1") Integer current,

@RequestParam(defaultValue = "10") Integer pageSize,

String sysDictCategoryId) {

Page<SysDict> page = sysDictService.page(new Page<>(current, pageSize), new QueryWrapper<SysDict>().lambda()

.eq(SysDict::getSysDictCategoryId,sysDictCategoryId)

.orderByAsc(SysDict::getSort)

);

return SaResult.data(page);

}

@ApiOperation("添加数据字典")

@PostMapping("add")

public SaResult add(@RequestBody SysDict sysDict) {

long count=sysDictService.count(new QueryWrapper<SysDict>().lambda()

.eq(SysDict::getVal,sysDict.getVal())

.eq(SysDict::getSysDictCategoryId,sysDict.getSysDictCategoryId()));

if (count>0){

throw new JcglException("当前字典类目下值不能重复");

}

sysDict.setType(sysDictCategoryService.getById(sysDict.getSysDictCategoryId()).getCategoryKey());

return SaResult.ok("添加成功").setData(sysDictService.save(sysDict));

}

@ApiOperation("修改数据字典")

@PostMapping("update")

@Transactional(rollbackFor = JcglException.class)

public SaResult update(@RequestBody SysDict sysDict) {

sysDict.setType(sysDictCategoryService.getById(sysDict.getSysDictCategoryId()).getCategoryKey());

sysDictService.updateById(sysDict);

long count=sysDictService.count(new QueryWrapper<SysDict>().lambda()

.eq(SysDict::getVal,sysDict.getVal())

.eq(SysDict::getSysDictCategoryId,sysDict.getSysDictCategoryId()));

if (count>1){

throw new JcglException("当前字典类目下值不能重复");

}

return SaResult.ok("修改成功");

}

@DeleteMapping("delete/{id}")

public SaResult update(@PathVariable String id) {

return SaResult.ok("删除成功").setData(sysDictService.removeById(id));

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/5/30 13:50

\*/

@Api(tags = "用户流程角色")

@RestController

@RequestMapping("/sysFlowRole")

public class SysFlowRoleController extends BaseWebController {

@Autowired

ISysFlowRoleService sysFlowRoleService;

@ApiOperation("分页查询流程角色")

@GetMapping("listPage")

public SaResult listPage(@RequestParam Integer current,

@RequestParam Integer pageSize,

String name,

String code) {

IPage<SysFlowRole> sysFlowRoleIPage = sysFlowRoleService.page(

new Page<>(current, pageSize),

new QueryWrapper<SysFlowRole>().lambda()

.like(!Strings.isNullOrEmpty(code), SysFlowRole::getCode, code)

.like(!Strings.isNullOrEmpty(name), SysFlowRole::getName, name));

return SaResult.data(sysFlowRoleIPage);

}

@ApiOperation("查询全部流程角色")

@GetMapping("listAll")

public SaResult listAll() {

return SaResult.data(sysFlowRoleService.list());

}

@ApiOperation("新增流程角色")

@PostMapping("add")

public SaResult add(@RequestBody SysFlowRole sysFlowRole) {

if (sysFlowRoleService.getOne(new QueryWrapper<SysFlowRole>().lambda().eq(SysFlowRole::getCode, sysFlowRole.getCode())) != null) {

throw new JcglException("角色编码不能重复");

}

return SaResult.ok("添加成功").setData(sysFlowRoleService.save(sysFlowRole));

}

@ApiOperation("修改流程角色")

@PostMapping("update")

public SaResult update(@RequestBody SysFlowRole sysFlowRole) {

return SaResult.ok("修改成功").setData(sysFlowRoleService.updateById(sysFlowRole));

}

@ApiOperation("删除流程角色")

@DeleteMapping("delete/{id}")

public SaResult delete(@PathVariable String id) {

return SaResult.ok("删除成功").setData(sysFlowRoleService.removeById(id));

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/14 8:56

\*/

@RestController

@RequestMapping("sysMenu")

public class SysMenuController extends BaseWebController {

@Autowired

ISysMenuService sysMenuService;

@GetMapping("menuListTree")

public Object menuListTree(String title,String menuSys) {

return SaResult.data(sysMenuService.findMenuListTree(title,menuSys));

}

/\*\*

\* 新增

\*

\* @param sysMenu

\* @return

\*/

@PostMapping("add")

public Object add(@RequestBody @Validated SysMenu sysMenu) {

if (Strings.isNullOrEmpty(sysMenu.getParentId())) {

sysMenu.setParentId("0");

}

return SaResult.ok("保存成功").setData(sysMenuService.save(sysMenu));

}

/\*\*

\* 删除菜单

\*

\* @param id

\* @return

\*/

@DeleteMapping("delete/{id}")

public Object delete(@PathVariable String id) {

return SaResult.ok("删除成功").setData(sysMenuService.removeById(id));

}

/\*\*

\* 删除菜单

\*

\* @param sysMenu

\* @return

\*/

@PostMapping("update")

public Object update(@RequestBody SysMenu sysMenu) {

if (Strings.isNullOrEmpty(sysMenu.getParentId())) {

sysMenu.setParentId("0");

}

return SaResult.ok("修改成功").setData(sysMenuService.updateById(sysMenu));

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/22 16:48

\*/

@RestController

@RequestMapping("sysOrg")

public class SysOrgController extends BaseWebController {

@Autowired

ISysOrgService sysOrgService;

/\*\*

\* 查询组织树

\*

\* @return

\*/

@GetMapping("myTree")

public SaResult myTree() {

SysUser sysUser = getUserDetail();

SysOrg sysOrg = sysOrgService.getById(sysUser.getSysOrgId());

if (sysOrg == null) {

return SaResult.ok().setData(new ArrayList<>());

}

List<SysOrg> sysOrgs = sysOrgService.findTreeByOrg(sysOrg);

return SaResult.ok().setData(sysOrgs);

}

@GetMapping("subOrg")

public SaResult subOrg(String sysOrgId,

@RequestParam(defaultValue = "1") Integer current,

@RequestParam(defaultValue = "10") Integer pageSize) {

SysUser sysUser = getUserDetail();

SysOrg psysOrg = sysOrgService.getById(sysUser.getSysOrgId());

if (psysOrg == null) {

return SaResult.error("当前登录用户组织信息不对");

}

if (Strings.isNullOrEmpty(sysOrgId)) {

sysOrgId = psysOrg.getId();

}

//判断当前新增的父级ID是否在登录人权限内

Set<String> authOrgIds = sysOrgService.findAuthIdsByOrgId(psysOrg.getParentId());

if (authOrgIds == null || authOrgIds.isEmpty()) {

return SaResult.error("当前所选父级组织不在权限范围内");

} else {

if (!authOrgIds.contains(sysOrgId)) {

return SaResult.error("当前所选父级组织不在权限范围内");

}

}

return SaResult.data(sysOrgService.page(new Page<>(current, pageSize),

new QueryWrapper<SysOrg>().lambda().eq(SysOrg::getParentId, sysOrgId).orderByAsc(SysOrg::getSort)));

}

@PostMapping("add")

public SaResult add(@RequestBody SysOrg sysOrg) {

SysUser sysUser = getUserDetail();

SysOrg psysOrg = sysOrgService.getById(sysUser.getSysOrgId());

if (psysOrg == null) {

return SaResult.error("当前所选父级组织不在权限范围内");

}

//判断当前新增的父级ID是否在登录人权限内

Set<String> authOrgIds = sysOrgService.findAuthIdsByOrgId(psysOrg.getParentId());

if (authOrgIds == null || authOrgIds.isEmpty()) {

return SaResult.error("当前所选父级组织不在权限范围内");

} else {

if (!authOrgIds.contains(sysOrg.getParentId())) {

return SaResult.error("当前所选父级组织不在权限范围内");

}

}

//顶级节点第一级才是公司租户级别

if ("0".equals(sysUser.getSysOrgId())) {

sysOrg.setOrgType(1);

} else {

sysOrg.setOrgType(2);

}

sysOrg.setCreateBy(sysUser.getId());

sysOrg.setCreateTime(new Date());

sysOrgService.save(sysOrg);

//设置分级列表ID和完整部门路径

SupOrgListVO supOrgListVO = sysOrgService.findSupOrgListVOByOrgId(sysOrg.getId());

sysOrg.setAncestors(supOrgListVO.getAncestors() != null ? Joiner.on(",").join(supOrgListVO.getAncestors()) : null);

sysOrg.setFullOrgName(supOrgListVO.getPOrgNames() != null ? Joiner.on("-").join(supOrgListVO.getPOrgNames()) : null);

sysOrgService.updateById(sysOrg);

return SaResult.ok("保存成功");

}

@PostMapping("update")

public SaResult update(@RequestBody SysOrg sysOrg) {

SysUser sysUser = getUserDetail();

SysOrg psysOrg = sysOrgService.getById(sysUser.getSysOrgId());

if (psysOrg == null) {

return SaResult.error("当前所选父级组织不在权限范围内");

}

//判断当前新增的父级ID是否在登录人权限内

Set<String> authOrgIds = sysOrgService.findAuthIdsByOrgId(psysOrg.getParentId());

if (authOrgIds == null || authOrgIds.isEmpty()) {

return SaResult.error("当前所选父级组织不在权限范围内");

} else {

if (!authOrgIds.contains(sysOrg.getParentId())) {

return SaResult.error("当前所选父级组织不在权限范围内");

}

}

//顶级节点第一级才是公司租户级别

if ("0".equals(sysUser.getSysOrgId())) {

sysOrg.setOrgType(1);

} else {

sysOrg.setOrgType(2);

}

sysOrg.setUpdateBy(sysUser.getId());

sysOrg.setUpdateTime(new Date());

sysOrgService.updateById(sysOrg);

SupOrgListVO supOrgListVO = sysOrgService.findSupOrgListVOByOrgId(sysOrg.getId());

sysOrg.setAncestors(supOrgListVO.getAncestors() != null ? Joiner.on(",").join(supOrgListVO.getAncestors()) : null);

sysOrg.setFullOrgName(supOrgListVO.getPOrgNames() != null ? Joiner.on("-").join(supOrgListVO.getPOrgNames()) : null);

sysOrgService.updateById(sysOrg);

return SaResult.ok("保存成功");

}

@DeleteMapping("delete/{sysOrgId}")

public SaResult delete(@PathVariable String sysOrgId) {

SysUser sysUser = getUserDetail();

SysOrg psysOrg = sysOrgService.getById(sysUser.getSysOrgId());

//判断当前新增的父级ID是否在登录人权限内

Set<String> authOrgIds = sysOrgService.findAuthIdsByOrgId(psysOrg.getParentId());

if (authOrgIds == null || authOrgIds.isEmpty() || !authOrgIds.contains(sysOrgId)) {

return SaResult.error("没有删除该组织的权限");

}

sysOrgService.removeById(sysOrgId);

return SaResult.ok("删除成功");

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/23 17:07

\*/

@RestController

@RequestMapping("sysPost")

public class SysPostController {

@Autowired

ISysPostService sysPostService;

@GetMapping("listPage")

public SaResult listPage(@RequestParam Integer current,

@RequestParam Integer pageSize, String postName) {

return SaResult.data(sysPostService.page(new Page<>(current, pageSize), new QueryWrapper<SysPost>().lambda()

.like(!Strings.isNullOrEmpty(postName), SysPost::getPostName, postName)));

}

@GetMapping("listAll")

public SaResult listAll() {

return SaResult.data(sysPostService.list());

}

@PostMapping("add")

public SaResult add(@RequestBody SysPost sysPost) {

return SaResult.ok("添加成功").setData(sysPostService.save(sysPost));

}

@PostMapping("update")

public SaResult update(@RequestBody SysPost sysPost) {

return SaResult.ok("添加成功").setData(sysPostService.updateById(sysPost));

}

@DeleteMapping("delete/{postId}")

public SaResult delete(@PathVariable String postId) {

return SaResult.ok("删除成功").setData(sysPostService.removeById(postId));

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/23 13:34

\*/

@RestController

@RequestMapping("sysRole")

public class SysRoleController extends BaseWebController {

@Autowired

ISysRoleService sysRoleService;

@Autowired

ISysRoleMenuService sysRoleMenuService;

@GetMapping("listPage")

public SaResult listPage(@RequestParam Integer current, @RequestParam Integer pageSize, String name) {

return SaResult.data(sysRoleService.page(new Page<>(current, pageSize), new QueryWrapper<SysRole>().lambda()

.like(!Strings.isNullOrEmpty(name), SysRole::getRoleName, name)));

}

@GetMapping("listAll")

public SaResult listAll() {

return SaResult.data(sysRoleService.list());

}

@PostMapping("add")

public SaResult add(@RequestBody SysRole sysRole) {

if (sysRoleService.getOne(new QueryWrapper<SysRole>().lambda().eq(SysRole::getCode,sysRole.getCode()))!=null){

throw new JcglException("角色编码不能重复");

}

return SaResult.ok("添加成功").setData(sysRoleService.save(sysRole));

}

@PostMapping("update")

public SaResult update(@RequestBody SysRole sysRole) {

return SaResult.ok("修改成功").setData(sysRoleService.updateById(sysRole));

}

@DeleteMapping("delete/{roleId}")

public SaResult delete(@PathVariable String roleId) {

return SaResult.ok("删除成功").setData(sysRoleService.removeById(roleId));

}

@PostMapping("roleAuthMenusSave")

public SaResult roleAuthMenusSave(@RequestBody RoleAuthMenusDTO roleAuthMenusDTO) {

SysUser sysUser = getUserDetail();

sysRoleMenuService.remove(new QueryWrapper<SysRoleMenu>().lambda().eq(SysRoleMenu::getSysRoleId, roleAuthMenusDTO.getRoleId()));

List<SysRoleMenu> sysRoleMenuList = new ArrayList<>();

roleAuthMenusDTO.getMenuIds().forEach(menuId -> {

SysRoleMenu sysRoleMenu = new SysRoleMenu();

sysRoleMenu.setSysRoleId(roleAuthMenusDTO.getRoleId());

sysRoleMenu.setSysMenuId(menuId);

sysRoleMenu.setCreateTime(new Date());

sysRoleMenu.setCreateBy(sysUser.getId());

sysRoleMenuList.add(sysRoleMenu);

});

sysRoleMenuService.saveBatch(sysRoleMenuList);

//当拥有当前角色的用户踢下线

List<String> usernames = sysRoleService.findUsernameByRoleId(roleAuthMenusDTO.getRoleId());

if (usernames != null && !usernames.isEmpty()) {

usernames.forEach(StpUtil::logout);

}

return SaResult.ok("保存成功");

}

@GetMapping("menuIdsByRoleId/{roleId}")

public SaResult roleAuthMenus(@PathVariable String roleId) {

List<SysRoleMenu> sysRoleMenuList = sysRoleMenuService.list(new QueryWrapper<SysRoleMenu>().lambda().eq(SysRoleMenu::getSysRoleId, roleId));

List<String> menuIds = sysRoleMenuList.stream().map(SysRoleMenu::getSysMenuId).collect(Collectors.toList());

return SaResult.ok("保存成功").setData(menuIds);

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/5/9 17:21

\*/

@Api(tags = "租户列表")

@RestController

@RequestMapping("sysTenant")

@RequiredArgsConstructor

public class SysTenantController {

private final ISysTenantService sysTenantService;

@GetMapping("listPage")

@ApiOperation("分页查询")

@ApiResponses({

@ApiResponse(code = 200, response = SysTenant.class, message = "查询成功")

})

public SaResult listPage(@Validated SysTenantPageDTO sysTenantPageDTO) {

Page<SysTenantVO> sysTenantVOPage = sysTenantService.findPage(new Page<>(sysTenantPageDTO.getCurrent(), sysTenantPageDTO.getPageSize()), sysTenantPageDTO.getTenantName());

return SaResult.data(sysTenantVOPage);

}

@PostMapping("add")

@ApiOperation("新增租户")

@ApiResponses({

@ApiResponse(code = 200, response = SaResult.class, message = "返回成功")

})

public SaResult add(@Validated @RequestBody SysTenantAddDTO sysTenantAddDTO) {

sysTenantService.addSysTenant(sysTenantAddDTO);

return SaResult.ok("新增成功");

}

@PutMapping("update")

@ApiOperation("修改租户")

@ApiResponses({

@ApiResponse(code = 200, response = SaResult.class, message = "返回成功")

})

public SaResult update(@Validated @RequestBody SysTenant sysTenant) {

sysTenantService.updateTenant(sysTenant);

return SaResult.ok("修改成功");

}

@DeleteMapping("delete/{id}")

@ApiOperation("删除租户")

@ApiResponses({

@ApiResponse(code = 200, response = SaResult.class, message = "返回成功")

})

public SaResult delete(@PathVariable String id) {

sysTenantService.removeTenant(id);

return SaResult.ok("删除成功");

}

}

/\*\*

\* @Description: 租户套餐管理

\* @Author: yb

\* @Date: 2023/5/10 9:58

\*/

@RestController

@RequestMapping("sysTenantPkg")

@Api(tags = "租户套餐")

@RequiredArgsConstructor

public class SysTenantPkgController {

private final ISysTenantPkgService sysTenantPkgService;

@ApiOperation("分页查询")

@GetMapping("listPage")

@ApiResponses({

@ApiResponse(code = 200,response = SysTenantPkgVO.class,message = "查询成功")

})

public SaResult listPage(@Validated SysTenantPkgPageDTO sysTenantPkgPageDTO){

Page<SysTenantPkgVO> sysTenantPkgVOPage=sysTenantPkgService.findPage(new Page<>(sysTenantPkgPageDTO.getCurrent(), sysTenantPkgPageDTO.getPageSize()),sysTenantPkgPageDTO.getPkgName());

return SaResult.data(sysTenantPkgVOPage);

}

@ApiOperation("查询全部套餐")

@GetMapping("listAll")

@ApiResponses({

@ApiResponse(code = 200,response = SysTenantPkg.class,message = "查询成功")

})

public SaResult listAll(){

return SaResult.data(sysTenantPkgService.list());

}

@ApiOperation("新增套餐")

@PostMapping("add")

@ApiResponses({

@ApiResponse(code = 200,response = SaResult.class,message = "操作成功")

})

public SaResult add(@Validated @RequestBody SysTenantPkgDTO sysTenantPkgDTO){

sysTenantPkgDTO.setId(null);

sysTenantPkgService.savePkg(sysTenantPkgDTO);

return SaResult.ok("新增成功");

}

@ApiOperation("修改套餐")

@PutMapping("update")

@ApiResponses({

@ApiResponse(code = 200,response = SaResult.class,message = "操作成功")

})

public SaResult update(@Validated @RequestBody SysTenantPkgDTO sysTenantPkgDTO){

//修改套餐

sysTenantPkgService.updatePkg(sysTenantPkgDTO);

return SaResult.ok("修改成功");

}

@ApiOperation("删除套餐")

@DeleteMapping("delete/{id}")

@ApiResponses({

@ApiResponse(code = 200,response = SaResult.class,message = "操作成功")

})

public SaResult add(@PathVariable String id){

sysTenantPkgService.removePkg(id);

return SaResult.ok("删除成功");

}

}

@RestController

@RequestMapping("sysUser")

@Api(tags = "用户管理")

public class SysUserController extends BaseWebController {

@Autowired

ISysUserService sysUserService;

@Autowired

ISysMenuService sysMenuService;

@Autowired

ISysUserRoleService sysUserRoleService;

@Autowired

ISysUserFlowRoleService sysUserFlowRoleService;

@Autowired

ISysUserOrgService sysUserOrgService;

@GetMapping("listPage")

@ApiResponses({@ApiResponse(code = 200,message = "返回成功",response =SysUserVO.class )})

public SaResult listPage(@RequestParam Integer current,

@RequestParam Integer pageSize,

String sysOrgId,

String empName,

String mobile) {

if (Strings.isNullOrEmpty(sysOrgId)){

sysOrgId=getUserDetail().getSysOrgId();

}

Page<SysUserVO> page = sysUserService.findUserPage(new Page<>(current, pageSize), sysOrgId, empName, mobile);

page.getRecords().forEach(sysUserVO -> {

List<SysRole> sysRoles=sysUserService.findRolesByUserId(sysUserVO.getId());

if (sysRoles!=null&&!sysRoles.isEmpty()){

sysUserVO.setRoleNames(Joiner.on(",").join(sysRoles.stream().map(SysRole::getRoleName).collect(Collectors.toList())));

sysUserVO.setRoleIds(sysRoles.stream().map(SysRole::getId).collect(Collectors.toList()));

}

sysUserVO.setFlowRoleIds(sysUserFlowRoleService.list(new QueryWrapper<SysUserFlowRole>().lambda()

.eq(SysUserFlowRole::getSysUserId,sysUserVO.getId())

.select(SysUserFlowRole::getSysFlowRoleId)).stream().map(SysUserFlowRole::getSysFlowRoleId).collect(Collectors.toList()));

//查询兼职组织

sysUserVO.setUserOrgs(sysUserOrgService.list(new QueryWrapper<SysUserOrg>().lambda().eq(SysUserOrg::getSysUserId,sysUserVO.getId())));

});

return SaResult.data(page);

}

/\*\*

\* 查询我的信息

\*

\* @return

\*/

@GetMapping("myUserInfo")

public Object myUserInfo() {

SysUserVO sysUserVO = sysUserService.findUserVOByUserId(getUserDetail().getId());

return SaResult.data(sysUserVO);

}

@GetMapping("getMyMenusPc")

public Object getMyMenusPc( ) {

SysUser sysUser = getUserDetail();

String userId = sysUser.getId();

List<SysMenu> sysMenuList = sysMenuService.findMyMenuListTree(userId,"pc");

return SaResult.data(sysMenuList);

}

@GetMapping("getMyMenusApp")

public Object getMyMenusApp( ) {

SysUser sysUser = getUserDetail();

String userId = sysUser.getId();

List<SysMenu> sysMenuList = sysMenuService.findMyMenuListTree(userId,"app");

return SaResult.data(sysMenuList);

}

@ApiOperation("新增用户")

@PostMapping("add")

@Transactional

public SaResult add(@Validated @RequestBody SysUserAddDTO sysUserAddDTO) {

SysUser sysUser = new SysUser();

BeanUtils.copyProperties(sysUserAddDTO, sysUser);

String password = RandomUtil.randomString(8);

sysUser.setPassword(BCrypt.hashpw(password));

sysUserService.save(sysUser);

//功能角色新增

List<SysUserRole> sysUserRoles=new ArrayList<>();

sysUserAddDTO.getRoleIds().forEach(roleId->{

SysUserRole sysUserRole=new SysUserRole();

sysUserRole.setSysRoleId(roleId);

sysUserRole.setSysUserId(sysUser.getId());

sysUserRoles.add(sysUserRole);

});

if(!sysUserRoles.isEmpty()){

sysUserRoleService.saveBatch(sysUserRoles);

}

//流程角色新增

List<SysUserFlowRole> sysUserFlowRoles=new ArrayList<>();

sysUserAddDTO.getFlowRoleIds().forEach(flowRoleId->{

SysUserFlowRole sysUserFlowRole=new SysUserFlowRole();

sysUserFlowRole.setSysFlowRoleId(flowRoleId);

sysUserFlowRole.setSysUserId(sysUser.getId());

sysUserFlowRoles.add(sysUserFlowRole);

});

if (!sysUserFlowRoles.isEmpty()){

sysUserFlowRoleService.saveBatch(sysUserFlowRoles);

}

//兼职组织添加

if (sysUserAddDTO.getUserOrgs()!=null&&!sysUserAddDTO.getUserOrgs().isEmpty()){

List<SysUserOrg> sysUserOrgs=new ArrayList<>();

sysUserAddDTO.getUserOrgs().forEach(sysUserOrg -> {

sysUserOrg.setSysUserId(sysUser.getId());

sysUserOrgs.add(sysUserOrg);

});

if (!sysUserOrgs.isEmpty()){

sysUserOrgService.saveBatch(sysUserOrgs);

}

}

return SaResult.ok("新增成功:初始密码为:" + password);

}

@ApiOperation("修改用户信息")

@PostMapping("update")

@Transactional

public SaResult update(@Validated @RequestBody SysUserUpdateDTO sysUserUpdateDTO) {

SysUser sysUser = sysUserService.getById(sysUserUpdateDTO.getId());

BeanUtils.copyProperties(sysUserUpdateDTO, sysUser);

sysUserService.updateById(sysUser);

sysUserRoleService.remove(new QueryWrapper<SysUserRole>().lambda().eq(SysUserRole::getSysUserId,sysUser.getId()));

sysUserFlowRoleService.remove(new QueryWrapper<SysUserFlowRole>().lambda().eq(SysUserFlowRole::getSysUserId,sysUser.getId()));

//功能角色新增

List<SysUserRole> sysUserRoles=new ArrayList<>();

sysUserUpdateDTO.getRoleIds().forEach(roleId->{

SysUserRole sysUserRole=new SysUserRole();

sysUserRole.setSysRoleId(roleId);

sysUserRole.setSysUserId(sysUser.getId());

sysUserRoles.add(sysUserRole);

});

if(!sysUserRoles.isEmpty()){

sysUserRoleService.saveBatch(sysUserRoles);

}

//流程角色新增

List<SysUserFlowRole> sysUserFlowRoles=new ArrayList<>();

sysUserUpdateDTO.getFlowRoleIds().forEach(flowRoleId->{

SysUserFlowRole sysUserFlowRole=new SysUserFlowRole();

sysUserFlowRole.setSysFlowRoleId(flowRoleId);

sysUserFlowRole.setSysUserId(sysUser.getId());

sysUserFlowRoles.add(sysUserFlowRole);

});

if (!sysUserFlowRoles.isEmpty()){

sysUserFlowRoleService.saveBatch(sysUserFlowRoles);

}

//兼职组织删除

sysUserOrgService.remove(new QueryWrapper<SysUserOrg>().lambda().eq(SysUserOrg::getSysUserId,sysUser.getId()));

//兼职组织添加

if (sysUserUpdateDTO.getUserOrgs()!=null&&!sysUserUpdateDTO.getUserOrgs().isEmpty()){

List<SysUserOrg> sysUserOrgs=new ArrayList<>();

sysUserUpdateDTO.getUserOrgs().forEach(sysUserOrg -> {

sysUserOrg.setSysUserId(sysUser.getId());

sysUserOrgs.add(sysUserOrg);

});

if (!sysUserOrgs.isEmpty()){

sysUserOrgService.saveBatch(sysUserOrgs);

}

}

StpUtil.logout(sysUser.getUsername());

return SaResult.ok("修改成功,当前用户已经强制下线");

}

//删除用户

@DeleteMapping("delete/{sysUserId}")

@Transactional

public SaResult delete(@PathVariable String sysUserId) {

SysUser sysUser = sysUserService.getById(sysUserId);

if (sysUser==null){

throw new JcglException("不存在的用户");

}

sysUserService.removeById(sysUserId);

sysUserRoleService.remove(new QueryWrapper<SysUserRole>().lambda().eq(SysUserRole::getSysUserId,sysUserId));

sysUserFlowRoleService.remove(new QueryWrapper<SysUserFlowRole>().lambda().eq(SysUserFlowRole::getSysUserId,sysUserId));

StpUtil.logout(sysUser.getUsername());

return SaResult.ok("删除成功,当前用户已经强制下线");

}

//停用用户

@PostMapping("stop/{sysUserId}")

public SaResult stop(@PathVariable String sysUserId) {

sysUserService.updateEnabled(sysUserId, 0);

//员工状态1 在职2 离职

sysUserService.updateEmpStatus(sysUserId, 2);

SysUser sysUser = sysUserService.getById(sysUserId);

StpUtil.logout(sysUser.getUsername());

return SaResult.ok("停用成功,当前用户已经强制下线");

}

//启用用户

@PostMapping("start/{sysUserId}")

public SaResult start(@PathVariable String sysUserId) {

sysUserService.updateEnabled(sysUserId, 1);

sysUserService.updateEmpStatus(sysUserId, 1);

return SaResult.ok("启用成功");

}

/\*\*

\* 重置密码

\* @param resetPasswordDTO

\* @return

\*/

@PostMapping("resetPassword")

public SaResult resetPassword(@RequestBody @Validated ResetPasswordDTO resetPasswordDTO) {

SysUser sysUser = sysUserService.getById(resetPasswordDTO.getSysUserId());

if (sysUser == null) {

return SaResult.error("用户不存在");

} else {

sysUser.setPassword(BCrypt.hashpw(resetPasswordDTO.getPassword()));

sysUserService.updateById(sysUser);

}

//重置成功后把该用户踢下线

StpUtil.logout(sysUser.getUsername());

return SaResult.ok("重置成功,当前用户已经强制下线");

}

/\*\*

\* 查询用户拥有的角色

\*

\* @param sysUserId

\* @return

\*/

@GetMapping("rolesByUerId/{sysUserId}")

public SaResult getRolesByUserId(@PathVariable String sysUserId) {

return SaResult.ok("查询成功").setData(sysUserRoleService.list(new QueryWrapper<SysUserRole>()

.lambda().eq(SysUserRole::getSysUserId, sysUserId)));

}

//重新授权角色

@PostMapping("saveUserRoles")

public SaResult saveUserRoles(@RequestBody @Validated SaveUserRolesDTO saveUserRolesDTO) {

SysUser sysUser = getUserDetail();

//获取授权用户

SysUser sysUserAuth = sysUserService.getById(saveUserRolesDTO.getSysUserId());

StpUtil.logout(sysUserAuth.getUsername());

sysUserRoleService.remove(new QueryWrapper<SysUserRole>().lambda().eq(SysUserRole::getSysUserId, saveUserRolesDTO.getSysUserId()));

List<SysUserRole> sysUserRoles = new ArrayList<>();

saveUserRolesDTO.getRoleIds().forEach(roleId -> {

SysUserRole sysUserRole = new SysUserRole();

sysUserRole.setSysUserId(saveUserRolesDTO.getSysUserId());

sysUserRole.setSysRoleId(roleId);

sysUserRole.setCreateTime(new Date());

sysUserRole.setCreateBy(sysUser.getId());

sysUserRoles.add(sysUserRole);

});

sysUserRoleService.saveBatch(sysUserRoles);

return SaResult.ok("保存成功,关联角色的用户已经强制下线").setData(true);

}

@PostMapping("updateMyInfo")

public SaResult updateMyInfo(@RequestBody UpdateMyInfoDTO updateMyInfoDTO) {

SysUser sysUser = getUserDetail();

BeanUtils.copyProperties(updateMyInfoDTO, sysUser);

return SaResult.ok("保存成功").setData(sysUserService.updateById(sysUser));

}

@PostMapping("exitLogin")

public SaResult exitLogin() {

StpUtil.logout();

return SaResult.ok("退出成功");

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/3/9 9:05

\*/

@Data

public class BaseEntity {

@TableField(fill = FieldFill.INSERT)

@ApiModelProperty(value = "创建时间",position = 100)

private Date createTime;

@TableField(fill = FieldFill.INSERT)

@ApiModelProperty(value = "创建人ID",position = 101)

private String createBy;

@TableField(fill = FieldFill.INSERT)

@ApiModelProperty(value = "创建人姓名",position = 102)

private String createName;

@TableField(fill = FieldFill.UPDATE)

@ApiModelProperty(value = "修改时间",position = 103)

private Date updateTime;

@TableField(fill = FieldFill.UPDATE)

@ApiModelProperty(value = "修改人ID",position = 104)

private String updateBy;

@TableField(fill = FieldFill.UPDATE)

@ApiModelProperty(value = "修改人姓名",position = 105)

private String updateName;

}

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(description = "通用分类查询条件实体类")

public class CommonSelectEntity extends PageEntity {

@ApiModelProperty(value = "检索条件")

private String vagueField;

private String tenantId;

@ApiModelProperty(value = "创建日期开始时间，格式：" + DateTimeConstant.DATE\_TIME, dataType = "LocalDateTime", example = "2023-01-01 00:00:01")

@JsonFormat(pattern = DateTimeConstant.DATE\_TIME, timezone = "GMT+8", locale = "zh\_CN")

@DateTimeFormat(pattern = DateTimeConstant.DATE\_TIME)

private LocalDateTime startTime;

@ApiModelProperty(hidden = true)

private String startTimeString;

@ApiModelProperty(value = "创建日期开始时间，格式：" + DateTimeConstant.DATE\_TIME, dataType = "LocalDateTime", example = "2023-12-31 00:00:00")

@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss", timezone = "GMT+8", locale = "zh\_CN")

@DateTimeFormat(pattern = "yyyy-MM-dd HH:mm:ss")

private LocalDateTime endTime;

@ApiModelProperty(hidden = true)

private String endTimeString;

public void dealTime() {

if (this.startTime != null) {

this.startTimeString = this.startTime.format(DateTimeFormatter.ofPattern(DateTimeConstant.DATE))

+ DateTimeConstant.DAY\_START\_TIME;

}

if (this.endTime != null) {

this.endTimeString = this.endTime.format(DateTimeFormatter.ofPattern(DateTimeConstant.DATE))

+ DateTimeConstant.DAY\_END\_TIME;

}

}

}

@Data

@ApiModel("表格错误信息")

public class ExcelErrorInfoEntity {

@ApiModelProperty("表格错误信息")

private String line;

@ApiModelProperty("表格错误信息")

private String column;

@ApiModelProperty("表格错误信息")

private String oldValue;

@ApiModelProperty("表格错误信息")

private String errorMessage;

}

@Data

public abstract class PageEntity {

@ApiModelProperty(value = "当前页面,大于等于1", name = "current", dataType = "int", example = "1")

@Min(1)

private int current = 1;

@ApiModelProperty(value = "页面大小", name = "pageSize", dataType = "int", example = "10")

@Min(1)

@Max(200)

private int pageSize = 10;

}

/\*\*

\* 设备资产信息记录表

\* @author wanggang

\* @date 2023/5/15 10:48

\* @TableName device\_asset\_info\_record

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@TableName(value ="device\_asset\_info\_record")

public class DeviceAssetInfoRecord extends BaseEntity implements Serializable {

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 6564768135328843016L;

/\*\*

\* ID

\*/

@TableId(value = "id")

private String id;

/\*\*

\* 租户ID

\*/

@TableField(value = "tenant\_id")

private String tenantId;

/\*\*

\* 设备信息ID

\*/

@TableField(value = "device\_info\_id")

private String deviceInfoId;

/\*\*

\* 资产编号

\*/

@TableField(value = "asset\_code")

private String assetCode;

/\*\*

\* 资产名称

\*/

@TableField(value = "asset\_name")

private String assetName;

/\*\*

\* 在账状态（1:在库;2:在账;3:报废;）

\*/

@TableField(value = "on\_account\_status")

private String onAccountStatus;

/\*\*

\* 登记时间

\*/

@TableField(value = "register\_date")

private Date registerDate;

/\*\*

\* 设备价格

\*/

@TableField(value = "device\_price")

private BigDecimal devicePrice;

/\*\*

\* 备注

\*/

@TableField(value = "remarks")

private String remarks;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

@TableLogic(value = "1", delval = "0")

@TableField(value = "del\_flag")

private Integer delFlag;

/\*\*

\* 设备授权表

\*

\* @author wanggang

\* @date 2023/6/8 16:22

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@TableName(value = "device\_authorization")

public class DeviceAuthorization extends BaseEntity implements Serializable {

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 2743592770815249783L;

/\*\*

\* ID

\*/

@TableId(value = "id")

private String id;

/\*\*

\* 租户ID

\*/

@TableField(value = "tenant\_id")

private String tenantId;

/\*\*

\* 设备信息ID

\*/

@TableField(value = "device\_info\_id")

private String deviceInfoId;

/\*\*

\* 授权人

\*/

@TableField(value = "authorization\_person")

private String authorizationPerson;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

@TableLogic(value = "1", delval = "0")

@TableField(value = "del\_flag")

private Integer delFlag;

/\*\*

\* 设备借用记录表

\*

\* @author wanggang

\* @date 2023/5/22 16:07

\* @TableName device\_borrow\_record

\*/

@Data

@EqualsAndHashCode(callSuper = true)

@TableName(value = "device\_borrow\_record")

public class DeviceBorrowRecord extends BaseEntity implements Serializable {

@Serial

@TableField(exist = false)

private static final long serialVersionUID = -7032352246487273177L;

/\*\*

\* ID

\*/

@TableId(value = "id")

private String id;

/\*\*

\* 租户ID

\*/

@TableField(value = "tenant\_id")

private String tenantId;

/\*\*

\* 设备信息ID

\*/

@TableField(value = "device\_info\_id")

private String deviceInfoId;

/\*\*

\* 借出时间

\*/

@TableField(value = "lend\_time")

private Date lendTime;

/\*\*

\* 借出员工ID

\*/

@TableField(value = "lend\_staff\_id")

private String lendStaffId;

/\*\*

\* 借出部门ID

\*/

@TableField(value = "lend\_department\_id")

private String lendDepartmentId;

/\*\*

\* 使用项目ID

\*/

@TableField(value = "use\_project\_id")

private String useProjectId;

/\*\*

\* 借出设备状态

\*/

@TableField(value = "lend\_device\_status")

private String lendDeviceStatus;

/\*\*

\* 归还时间

\*/

@TableField(value = "return\_time")

private Date returnTime;

/\*\*

\* 归还人员ID

\*/

@TableField(value = "return\_staff\_id")

private String returnStaffId;

/\*\*

\* 归还设备状态

\*/

@TableField(value = "return\_device\_status")

private String returnDeviceStatus;

/\*\*

\* 附件Id

\*/

@TableField(value = "file\_id")

private String fileId;

/\*\*

\* 校准结果(1:合格;0:不合格)

\*/

@TableField(value = "calibration\_result")

private Integer calibrationResult;

/\*\*

\* 备注

\*/

@TableField(value = "remarks")

private String remarks;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

@TableLogic(value = "1", delval = "0")

@TableField(value = "del\_flag")

private Integer delFlag;

/\*\*

\* 设备流转记录表

\*

\* @author wanggang

\* @date 2023/5/22 10:19

\* @TableName device\_circulation\_record

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@TableName(value = "device\_circulation\_record")

public class DeviceCirculationRecord extends BaseEntity implements Serializable {

@Serial

@TableField(exist = false)

private static final long serialVersionUID = -6364225803912691713L;

/\*\*

\* ID

\*/

@TableId(value = "id")

private String id;

/\*\*

\* 租户ID

\*/

@TableField(value = "tenant\_id")

private String tenantId;

/\*\*

\* 设备信息ID

\*/

@TableField(value = "device\_info\_id")

private String deviceInfoId;

/\*\*

\* 流转类型

\*/

@TableField(value = "circulation\_type")

private String circulationType;

/\*\*

\* 发出地点

\*/

@TableField(value = "issue\_place")

private String issuePlace;

/\*\*

\* 流转日期

\*/

@TableField(value = "circulation\_date")

private Date circulationDate;

/\*\*

\* 接收人ID

\*/

@TableField(value = "recipient\_id")

private String recipientId;

/\*\*

\* 接收地点

\*/

@TableField(value = "receive\_place")

private String receivePlace;

/\*\*

\* 发出设备状态

\*/

@TableField(value = "issue\_device\_situation")

private String issueDeviceSituation;

/\*\*

\* 接收设备状态

\*/

@TableField(value = "receive\_device\_situation")

private String receiveDeviceSituation;

/\*\*

\* 备注

\*/

@TableField(value = "remarks")

private String remarks;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

@TableLogic(value = "1", delval = "0")

@TableField(value = "del\_flag")

private Integer delFlag;

/\*\*

\* 附件Id

\*/

@TableField(value = "file\_id")

private String fileId;

/\*\*

\* "设备分类信息表

\*

\* @TableName device\_classify\_info

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@TableName(value = "device\_classify\_info")

public class DeviceClassifyInfo extends BaseEntity implements Serializable {

@Serial

private static final long serialVersionUID = 6805311429578380014L;

/\*\*

\* ID

\*/

private String id;

/\*\*

\* 租户id

\*/

private String tenantId;

/\*\*

\* 设备分类编号

\*/

private String deviceClassifyNumber;

/\*\*

\* 设备分类名称

\*/

private String deviceClassifyName;

/\*\*

\* 设备分类简称

\*/

private String shortName;

/\*\*

\* 设备名称

\*/

private String deviceName;

/\*\*

\* 规格

\*/

private String specifications;

/\*\*

\* 型号

\*/

private String model;

/\*\*

\* 测量范围(量程)

\*/

private String measureRange;

/\*\*

\* 准确度等级

\*/

private String accuracyLevel;

/\*\*

\* 不确定度

\*/

private String uncertainty;

/\*\*

\* 设备分类（1:A;2:B;3:C;4:S;）

\*/

private String deviceClassify;

/\*\*

\* 资产分类(1:固定资产;2:耗材;)

\*/

private String assetClassify;

/\*\*

\* 归属权(1:北域;2:总站;)

\*/

private String ownership;

/\*\*

\* 精度

\*/

private String precision;

/\*\*

\* 设备原值(元)

\*/

private BigDecimal deviceOriginalValue;

/\*\*

\* 数量

\*/

private Integer quantity;

/\*\*

\* 生产厂家ID（对应厂商ID）

\*/

private String productionId;

/\*\*

\* 供货商ID（对应厂商ID）

\*/

private String supplierId;

/\*\*

\* 备注

\*/

private String remarks;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

private Integer delFlag;

/\*\*

\* 设备管理表

\*

\* @author wanggang

\* @date 2023/5/8 17:06

\*/

@Data

@TableName(value = "device\_info")

@EqualsAndHashCode(callSuper = false)

public class DeviceInfo extends BaseEntity implements Serializable {

@Serial

@TableField(exist = false)

private static final long serialVersionUID = -2791565559990160827L;

/\*\*

\* 设备ID

\*/

@TableId(value = "id")

private String id;

/\*\*

\* 租户ID

\*/

@TableField(value = "tenant\_id")

private String tenantId;

/\*\*

\* 设备编号

\*/

@TableField(value = "device\_number")

private String deviceNumber;

/\*\*

\* 使用状态（1:使用;0:停用;-1:报废;）

\*/

@TableField(value = "use\_status")

private String useStatus;

/\*\*

\* 设备状态

\*/

@TableField(value = "device\_status")

private String deviceStatus;

/\*\*

\*流程状态

\*/

@TableField(value = "flow\_status")

private String flowStatus;

/\*\*

\* 检定状态（1:已检定;2:待检定;）

\*/

@TableField(value = "verification\_status")

private String verificationStatus;

/\*\*

\* 购置日期

\*/

@TableField(value = "purchase\_time")

private Date purchaseTime;

/\*\*

\* 出厂编号

\*/

@TableField(value = "exworks\_number")

private String exworksNumber;

/\*\*

\* 检定周期（1:3个月;2:6个月;）

\*/

@TableField(value = "verification\_cycle")

private String verificationCycle;

/\*\*

\* 备注

\*/

@TableField(value = "remarks")

private String remarks;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

@TableLogic(value = "1", delval = "0")

@TableField(value = "del\_flag")

private Integer delFlag;

/\*\*

\* 设备分类ID

\*/

@TableField(value = "device\_classify\_info\_id")

private String deviceClassifyInfoId;

/\*\*

\* 部门ID

\*/

@TableField(value = "department\_id")

private String departmentId;

/\*\*

\* 量值溯源

\*/

@TableField(value = "asset\_source")

private String assetSource;

/\*\*

\* 责任人

\*/

@TableField(value = "person\_liable")

private String personLiable;

/\*\*

\* 是否自动采集(0否;1是)

\*/

@TableField(value = "automatic\_collection")

private Integer automaticCollection;

/\*\*

\* 出厂日期

\*/

@TableField(value = "exworks\_time")

private Date exworksTime;

/\*\*

\* 设备存放货架信息ID

\*/

@TableField(value = "device\_shelves\_info\_id")

private String deviceShelvesInfoId;

/\*\*

\* 设备编号(交通行业)

\*/

@TableField(value = "traffic\_device\_number")

private String trafficDeviceNumber;

/\*\*

\* 设备名称(交通行业)

\*/

@TableField(value = "traffic\_device\_name")

private String trafficDeviceName;

/\*\*

\* 场所信息

\*

\* @TableName device\_place

\*/

@TableName(value = "device\_place")

@Data

@EqualsAndHashCode(callSuper=false)

public class DevicePlace extends BaseEntity implements Serializable {

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 7009817421627027559L;

/\*\*

\* 主键

\*/

@TableId

private String id;

/\*\*

\* 租户id

\*/

private String tenantId;

/\*\*

\* 组织id

\*/

private String orgId;

/\*\*

\* 场所名称(二级组织名称)

\*/

private String placeName;

/\*\*

\* 场所编码(一个字母)

\*/

private String placeCode;

/\*\*

\* 场所位置及描述

\*/

private String placeDesc;

/\*\*

\* 备注

\*/

private String remarks;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

private Integer delFlag;

/\*\*

\* 设备保养维修记录表

\*

\* @author wanggang

\* @date 2023/5/10 10:28

\* @TableName device\_repair\_maintenance\_record

\*/

@Data

@TableName(value = "device\_repair\_maintenance\_record")

@EqualsAndHashCode(callSuper = false)

public class DeviceRepairMaintenanceRecord extends BaseEntity implements Serializable {

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 6408182651538687656L;

/\*\*

\* ID

\*/

@TableId(value = "id")

private String id;

/\*\*

\* 设备信息ID

\*/

@TableField(value = "device\_info\_id")

private String deviceInfoId;

/\*\*

\* 租户ID

\*/

@TableField(value = "tenant\_id")

private String tenantId;

/\*\*

\* 使用部门ID

\*/

@TableField(value = "use\_department\_id")

private String useDepartmentId;

/\*\*

\* 负责人ID

\*/

@TableField(value = "head\_id")

private String headId;

/\*\*

\* 参与人员

\*/

@TableField(value = "involved\_staff")

private String involvedStaff;

/\*\*

\* 分类(1:保养;2:维修)

\*/

@TableField(value = "classify")

private String classify;

/\*\*

\* 发生时间

\*/

@TableField(value = "happen\_time")

private Date happenTime;

/\*\*

\* 情况说明

\*/

@TableField(value = "situation\_description")

private String situationDescription;

/\*\*

\* 校准结果(1:合格;0:不合格)

\*/

@TableField(value = "calibration\_result")

private Integer calibrationResult;

/\*\*

\* 运行使用状态

\*/

@TableField(value = "run\_use\_status")

private String runUseStatus;

/\*\*

\* 附件Id

\*/

@TableField(value = "file\_id")

private String fileId;

/\*\*

\* 备注

\*/

@TableField(value = "remarks")

private String remarks;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

@TableLogic(value = "1", delval = "0")

@TableField(value = "del\_flag")

private Integer delFlag;

/\*\*

\* 设备存放室

\*

\* @TableName device\_room

\*/

@EqualsAndHashCode(callSuper = true)

@TableName(value = "device\_room")

@Data

public class DeviceRoom extends BaseEntity implements Serializable {

@Serial

@TableField(exist = false)

private static final long serialVersionUID = -4997143640562861768L;

/\*\*

\* 主键

\*/

@TableId

private String id;

/\*\*

\* 租户id

\*/

private String tenantId;

/\*\*

\* 场所id

\*/

private String placeId;

/\*\*

\* 场所名称

\*/

private String placeName;

/\*\*

\* 存放室名称

\*/

private String roomName;

/\*\*

\* 房间编号(场所编号+2位数字)

\*/

private String roomCode;

/\*\*

\* 描述

\*/

private String roomDesc;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

private Integer delFlag;

/\*\*

\* 设备存放货架信息表

\*

\* @TableName device\_shelves\_info

\*/

@TableName(value = "device\_shelves\_info")

@EqualsAndHashCode(callSuper =false)

@Data

public class DeviceShelvesInfo extends BaseEntity implements Serializable {

@Serial

@TableField(exist = false)

private static final long serialVersionUID = -3641383449597684827L;

/\*\*

\* ID

\*/

private String id;

/\*\*

\* 租户ID

\*/

private String tenantId;

/\*\*

\* 设备信息ID

\*/

private String deviceInfoId;

/\*\*

\* 设备编号

\*/

private String deviceNumber;

/\*\*

\* 存放编号

\*/

private String storageCode;

/\*\*

\* 货架名称

\*/

private String shelvesName;

/\*\*

\* 货架编号

\*/

private String shelvesCode;

/\*\*

\* 货架层号

\*/

private String layerCode;

/\*\*

\* 位置序号

\*/

private String positionCode;

/\*\*

\* 位置描述

\*/

private String positionDesc;

/\*\*

\* 存放室id

\*/

private String roomId;

/\*\*

\* 存放室名称

\*/

private String roomName;

/\*\*

\* 房间编号(场所编号+2位数字)

\*/

private String roomCode;

/\*\*

\* 场所id

\*/

private String placeId;

/\*\*

\* 场所名称(二级组织名称)

\*/

private String placeName;

/\*\*

\* 场所编码(一个字母)

\*/

private String placeCode;

/\*\*

\* 购买时间

\*/

private LocalDateTime purchaseTime;

/\*\*

\* 备注

\*/

private String remarks;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

private Integer delFlag;

/\*\*

\* 设备检定记录表

\*

\* @author wanggang

\* @date 2023/5/10 14:14

\* @TableName device\_verification\_record

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@TableName(value = "device\_verification\_record")

public class DeviceVerificationRecord extends BaseEntity implements Serializable {

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 6119202029037174892L;

/\*\*

\* 检定ID

\*/

@TableId(value = "id")

private String id;

/\*\*

\* 设备信息ID

\*/

@TableField(value = "device\_info\_id")

private String deviceInfoId;

/\*\*

\* 租户ID

\*/

@TableField(value = "tenant\_id")

private String tenantId;

/\*\*

\* 证书编号

\*/

@TableField(value = "certificate\_number")

private String certificateNumber;

/\*\*

\* 证书类型

\*/

@TableField(value = "certificate\_classification")

private String certificateClassification;

/\*\*

\* 检定单位

\*/

@TableField(value = "verification\_company")

private String verificationCompany;

/\*\*

\* 检定日期

\*/

@TableField(value = "verification\_date")

private Date verificationDate;

/\*\*

\* 有效开始日期

\*/

@TableField(value = "valid\_start")

private Date validStart;

/\*\*

\* 有效结束日期

\*/

@TableField(value = "valid\_end")

private Date validEnd;

/\*\*

\* 检定结果(1:合格;0:不合格)

\*/

@TableField(value = "verification\_result")

private Integer verificationResult;

/\*\*

\* 备注

\*/

@TableField(value = "remarks")

private String remarks;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

@TableLogic(value = "1", delval = "0")

@TableField(value = "del\_flag")

private Integer delFlag;

/\*\*

\* 附件Id

\*/

@TableField(value = "file\_id")

private String fileId;

/\*\*

\* 检定方式

\*/

@TableField(value = "verification\_style")

private String verificationStyle;

/\*\*

\* 设备运行记录表

\*

\* @author wanggang

\* @date 2023/5/9 11:43

\* @TableName device\_working\_record

\*/

@Data

@TableName(value = "device\_working\_record")

@EqualsAndHashCode(callSuper = false)

public class DeviceWorkingRecord extends BaseEntity implements Serializable {

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 5929004034667493547L;

/\*\*

\* ID

\*/

@TableId(value = "id")

private String id;

/\*\*

\* 租户ID

\*/

@TableField(value = "tenant\_id")

private String tenantId;

/\*\*

\* 设备信息ID

\*/

@TableField(value = "device\_info\_id")

private String deviceInfoId;

/\*\*

\* 报告编号

\*/

@TableField(value = "report\_number")

private String reportNumber;

/\*\*

\* 运行开始时间

\*/

@TableField(value = "working\_start\_time")

private Date workingStartTime;

/\*\*

\* 运行结束时间

\*/

@TableField(value = "working\_end\_time")

private Date workingEndTime;

/\*\*

\* 使用前状态（1:正常;2:异常;）

\*/

@TableField(value = "use\_front\_status")

private String useFrontStatus;

/\*\*

\* 使用后状态（1:正常;2:异常;）

\*/

@TableField(value = "use\_end\_status")

private String useEndStatus;

/\*\*

\* 检测人员ID

\*/

@TableField(value = "detection\_staff\_id")

private String detectionStaffId;

/\*\*

\* 异常状况描述

\*/

@TableField(value = "exception\_situation")

private String exceptionSituation;

/\*\*

\* 审核人员ID

\*/

@TableField(value = "review\_person")

private String reviewPerson;

/\*\*

\* 审核意见

\*/

@TableField(value = "review\_comments")

private String reviewComments;

/\*\*

\* 备注

\*/

@TableField(value = "remarks")

private String remarks;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

@TableLogic(value = "1", delval = "0")

@TableField(value = "del\_flag")

private Integer delFlag;

/\*\*

\* "厂商联系人信息表

\*

\* @TableName supplier\_contacts\_info

\*/

@TableName(value = "supplier\_contacts\_info")

@Data

@EqualsAndHashCode(callSuper = false)

public class SupplierContactsInfo extends BaseEntity implements Serializable {

@Serial

@TableField(exist = false)

private static final long serialVersionUID = -1323021059367421636L;

/\*\*

\* ID

\*/

@TableId(value = "id", type = IdType.ASSIGN\_ID)

private String id;

/\*\*

\* 租户ID

\*/

@TableField(value = "tenant\_id")

private String tenantId;

/\*\*

\* 厂商ID

\*/

@TableField(value = "supplier\_id")

private String supplierId;

/\*\*

\* 姓名

\*/

@TableField(value = "name")

private String name;

/\*\*

\* 别名

\*/

@TableField(value = "alias")

private String alias;

/\*\*

\* 职位

\*/

@TableField(value = "position")

private String position;

/\*\*

\* 部门名称

\*/

@TableField(value = "department\_name")

private String departmentName;

/\*\*

\* 性别（0:未知;1:女;2:男;）

\*/

@TableField(value = "gender")

private String gender;

/\*\*

\* 微信

\*/

@TableField(value = "wechat")

private String wechat;

/\*\*

\* 企业邮箱

\*/

@TableField(value = "mailbox")

private String mailbox;

/\*\*

\* QQ

\*/

@TableField(value = "qq")

private String qq;

/\*\*

\* 住址

\*/

@TableField(value = "address")

private String address;

/\*\*

\* 电话

\*/

@TableField(value = "telephone")

private String telephone;

/\*\*

\* 备注

\*/

@TableField(value = "remarks")

private String remarks;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

@TableField(value = "del\_flag")

private Integer delFlag;

/\*\*

\* 厂商信息表

\*

\* @TableName supplier\_info

\*/

@EqualsAndHashCode(callSuper = true)

@TableName(value = "supplier\_info")

@Data

public class SupplierInfo extends BaseEntity implements Serializable {

@TableField(exist = false)

@Serial

private static final long serialVersionUID = -3263251265019821799L;

/\*\*

\* 厂商ID

\*/

@TableId(type = IdType.ASSIGN\_ID)

private String id;

/\*\*

\* 租户ID

\*/

@TableField(value = "tenant\_id")

private String tenantId;

/\*\*

\* 名称

\*/

@TableField(value = "name")

private String name;

/\*\*

\* 简称

\*/

@TableField(value = "abbreviation")

private String abbreviation;

/\*\*

\* 地址

\*/

@TableField(value = "address")

private String address;

/\*\*

\* 行业

\*/

@TableField(value = "industry")

private String industry;

/\*\*

\* 类型（1:厂商;2:供应商;）

\*/

@TableField(value = "type")

private String type;

/\*\*

\* 座机

\*/

@TableField(value = "contact\_telephone")

private String contactTelephone;

/\*\*

\* 企业邮箱

\*/

@TableField(value = "mailbox")

private String mailbox;

/\*\*

\* 厂商联系人ID

\*/

@TableField(value = "supplier\_contact\_id")

private String supplierContactId;

/\*\*

\* 简介

\*/

@TableField(value = "profile")

private String profile;

/\*\*

\* 备注

\*/

@TableField(value = "remarks")

private String remarks;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

@TableField(value = "del\_flag")

private Integer delFlag;

/\*\*

\* 设备资产信息记录表

\*

\* @author wanggang

\* @date 2023/5/15 10:57

\*/

@Data

@ApiModel(value = "设备资产信息记录表")

public class DeviceAssetInfoRecordDTO {

/\*\*

\* 设备编号

\*/

@ApiModelProperty(value = "设备编号", hidden = true)

private String deviceNumber;

/\*\*

\* 设备名称

\*/

@ApiModelProperty(value = "设备名称", hidden = true)

private String deviceName;

/\*\*

\* ID

\*/

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "ID")

private String id;

/\*\*

\* 设备信息ID

\*/

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "设备信息ID", required = true, example = "1656847984220512212")

private String deviceInfoId;

/\*\*

\* 资产编号

\*/

@NotBlank(message = "[资产编号]不能为空")

@Length(min = 12, max = 64, message = "资产编号长度要超过12且不能超过64")

@ApiModelProperty(value = "资产编号", required = true, example = "Z12345678987654")

private String assetCode;

/\*\*

\* 资产名称

\*/

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "资产名称", required = true, example = "资产名称")

private String assetName;

/\*\*

\* 资产分类

\*/

@ApiModelProperty(value = "资产分类", required = true, example = "1")

private String assetClassify;

/\*\*

\* 在账状态（1:在库;2:在账;3:报废;）

\*/

@NotNull(message = "[在账状态（1:在库;2:在账;3:报废;）]不能为空")

@ApiModelProperty(value = "在账状态（1:在用;2:在库;0:报废;）", required = true, example = "1")

private String onAccountStatus;

/\*\*

\* 登记日期

\*/

@NotNull(message = "[登记日期]不能为空")

@JsonFormat(pattern = "yyyy-MM-dd")

@ApiModelProperty(value = "登记日期", required = true, example = "2023-05-12")

private Date registerDate;

/\*\*

\* 设备价格

\*/

@NotNull(message = "[设备价格]不能为空")

@ApiModelProperty(value = "设备价格", required = true, example = "50000.25")

private BigDecimal devicePrice;

/\*\*

\* 备注

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "备注", example = "备注")

private String remarks;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

@ApiModelProperty(value = "删除标志（1:有效;0:删除;默认1）", hidden = true)

private Integer delFlag;

}

/\*\*

\* 设备资产查询条件实体类

\*

\* @author wanggang

\* @date 2023/5/18 9:03

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(value = "设备资产查询条件实体类")

public class DeviceAssetInfoRecordSelectDTO extends PageEntity {

/\*\*

\* 模糊字段

\*/

@ApiModelProperty(value = "设备编号、设备名称、资产编号、资产名称", name = "vagueField", dataType = "String")

private String vagueField;

/\*\*

\* 资产分类

\*/

@ApiModelProperty(value = "资产分类", name = "assetClassify", dataType = "String")

private String assetClassify;

/\*\*

\* 在账状态

\*/

@ApiModelProperty(value = "在账状态", name = "onAccountStatus", dataType = "String")

private String onAccountStatus;

/\*\*

\* 登记日期开始时间

\*/

@ApiModelProperty(value = "登记日期开始时间,格式：yyyy-MM-dd", name = "startTime", dataType = "String")

private String startTime;

/\*\*

\* 登记日期结束时间

\*/

@ApiModelProperty(value = "登记日期结束时间,格式：yyyy-MM-dd", name = "endTime", dataType = "String")

private String endTime;

}

/\*\*

\* 设备授权表

\*

\* @author wanggang

\* @date 2023/6/8 16:29

\*/

@Data

@ApiModel(value = "设备授权DTO")

public class DeviceAuthorizationDTO {

/\*\*

\* ID

\*/

@NotBlank(message = "[ID]不能为空")

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "ID")

private String id;

/\*\*

\* 设备信息ID

\*/

@NotBlank(message = "[设备信息ID]不能为空")

@ApiModelProperty("设备信息ID")

@Length(max = 64, message = "编码长度不能超过64")

private String deviceInfoId;

/\*\*

\* 授权人

\*/

@Length(max = 1000, message = "编码长度不能超过1000")

@ApiModelProperty(value = "授权人", hidden = true)

private String authorizationPerson;

/\*\*

\* 授权人ID集

\*/

@ApiModelProperty(value = "授权人ID集")

private List<String> authorizationPersonList;

}

/\*\*

\* 设备授权查询条件实体类

\*

\* @author wanggang

\* @date 2023/5/18 9:03

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(value = "设备授权查询条件实体类")

public class DeviceAuthorizationSelectDTO extends PageEntity {

/\*\*

\* 模糊字段

\*/

@ApiModelProperty(value = "设备编号、设备名称、规格型号", name = "vagueField", dataType = "String")

private String vagueField;

/\*\*

\* 设备状态

\*/

@ApiModelProperty(value = "设备状态", name = "deviceStatus", dataType = "String")

private String deviceStatus;

/\*\*

\* 资产分类

\*/

@ApiModelProperty(value = "资产分类", name = "assetClassify", dataType = "String")

private String assetClassify;

/\*\*

\* 使用部门

\*/

@ApiModelProperty(value = "使用部门", name = "departmentId", dataType = "String")

private String departmentId;

/\*\*

\* 责任人

\*/

@ApiModelProperty(value = "责任人", name = "personLiable", dataType = "String")

private String personLiable;

// /\*\*

// \* 证书到期开始时间

// \*/

// @ApiModelProperty(value = "证书到期开始时间,格式：yyyy-MM-dd", name = "startTime", dataType = "String")

// private String startTime;

//

// /\*\*

// \* 证书到期结束时间

// \*/

// @ApiModelProperty(value = "证书到期结束时间,格式：yyyy-MM-dd", name = "endTime", dataType = "String")

// private String endTime;

}

/\*\*

\* 设备借用记录表

\*

\* @author wanggang

\* @date 2023/5/22 16:19

\*/

@Data

@ApiModel(value = "设备借用记录表")

public class DeviceBorrowRecordDTO {

/\*\*

\* ID

\*/

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "ID")

private String id;

/\*\*

\* 设备信息ID

\*/

@NotBlank(message = "[设备信息ID]不能为空")

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "设备信息ID", required = true, example = "1656950797366423554")

private String deviceInfoId;

/\*\*

\* 设备编号

\*/

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "设备编号", hidden = true)

private String deviceNumber;

/\*\*

\* 借出时间

\*/

@NotNull(message = "[借出时间]不能为空")

@JsonFormat(pattern = "yyyy-MM-dd")

@ApiModelProperty(value = "借出时间", required = true, example = "2023-05-22")

private Date lendTime;

/\*\*

\* 借出员工ID

\*/

@NotBlank(message = "[借出员工ID]不能为空")

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "借出员工ID", required = true, example = "1656950797366423554")

private String lendStaffId;

/\*\*

\* 借出员工名

\*/

@ApiModelProperty(value = "借出员工名")

private String lendStaffName;

/\*\*

\* 借出部门ID

\*/

@NotBlank(message = "[借出部门ID]不能为空")

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "借出部门ID", required = true, example = "1656950797366423554")

private String lendDepartmentId;

/\*\*

\* 借出部门名

\*/

@ApiModelProperty(value = "借出部门名")

private String lendDepartmentName;

/\*\*

\* 使用项目ID

\*/

@NotBlank(message = "[使用项目ID]不能为空")

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "使用项目ID", required = true, example = "1656950797366423554")

private String useProjectId;

/\*\*

\* 使用项目名

\*/

@ApiModelProperty("使用项目名")

private String useProjectName;

/\*\*

\* 借出设备状态

\*/

@NotNull(message = "[借出设备状态]不能为空")

@ApiModelProperty(value = "借出设备状态", required = true, example = "2")

private String lendDeviceStatus;

/\*\*

\* 归还时间

\*/

@ApiModelProperty(value = "归还时间", example = "2023-05-22")

@JsonFormat(pattern = "yyyy-MM-dd")

private Date returnTime;

/\*\*

\* 归还人员ID

\*/

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "归还人员ID", example = "1656950797366423554")

private String returnStaffId;

/\*\*

\* 归还人员名

\*/

@ApiModelProperty("归还人员名")

private String returnStaffName;

/\*\*

\* 归还设备状态

\*/

@ApiModelProperty(value = "归还设备状态", example = "2")

private String returnDeviceStatus;

/\*\*

\* 备注

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "备注", example = "备注")

private String remarks;

/\*\*

\* 附件Id

\*/

@Length(max = 500, message = "附件Id不能超过500")

@ApiModelProperty(value = "附件Id", example = "1640520011505266689")

private String fileId;

/\*\*

\* 附件Id集合

\*/

@ApiModelProperty(value = "附件Id集合")

private List<String> fileIdList;

/\*\*

\* 校准结果(1:合格;0:不合格)

\*/

@ApiModelProperty(value = "校准结果(1:合格;0:不合格)", example = "1")

private Integer calibrationResult;

}

/\*\*

\* 设备借用记录查询条件实体类

\*

\* @author wanggang

\* @date 2023/5/22 14:07

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(value = "设备借用记录查询条件实体类")

public class DeviceBorrowRecordSelectDTO extends PageEntity {

/\*\*

\* 模糊字段

\*/

@ApiModelProperty(value = "设备编号、设备名称、使用项目", name = "vagueField", dataType = "String")

private String vagueField;

/\*\*

\* 借出设备状态

\*/

@ApiModelProperty(value = "借出设备状态", name = "lendDeviceStatus", dataType = "String")

private String lendDeviceStatus;

/\*\*

\* 归还设备状态

\*/

@ApiModelProperty(value = "归还设备状态", name = "returnDeviceStatus", dataType = "String")

private String returnDeviceStatus;

/\*\*

\* 借出日期开始时间

\*/

@ApiModelProperty(value = "借出日期开始时间,格式：yyyy-MM-dd", name = "startTime", dataType = "String")

private String startTime;

/\*\*

\* 归还日期结束时间

\*/

@ApiModelProperty(value = "归还日期结束时间,格式：yyyy-MM-dd", name = "endTime", dataType = "String")

private String endTime;

}

/\*\*

\* 设备流转记录表

\*

\* @author wanggang

\* @date 2023/5/10 9:28

\*/

@Data

@ApiModel(value = "设备流转记录表")

public class DeviceCirculationRecordDTO {

/\*\*

\* ID

\*/

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "ID")

private String id;

/\*\*

\* 设备编号

\*/

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "设备编号", hidden = true)

private String deviceNumber;

/\*\*

\* 流转类型

\*/

@NotNull(message = "[流转类型]不能为空")

@ApiModelProperty(value = "流转类型", required = true, example = "1")

private String circulationType;

/\*\*

\* 发出地点

\*/

@NotBlank(message = "[发出地点]不能为空")

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "发出地点", required = true, example = "茶花路222号茶园主场所")

private String issuePlace;

/\*\*

\* 流转日期

\*/

@NotNull(message = "[流转日期]不能为空")

@JsonFormat(pattern = "yyyy-MM-dd")

@ApiModelProperty(value = "流转日期", required = true, example = "2023-05-22")

private Date circulationDate;

/\*\*

\* 接收人ID

\*/

@NotBlank(message = "[接收人ID]不能为空")

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "接收人ID", required = true, example = "1659396771864166402")

private String recipientId;

/\*\*

\* 接收人名

\*/

@ApiModelProperty(value = "接收人名")

private String recipientName;

/\*\*

\* 接收地点

\*/

@NotBlank(message = "[接收地点]不能为空")

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "接收地点", required = true, example = "万州区万象路222号渝东北分中心")

private String receivePlace;

/\*\*

\* 发出设备状态

\*/

@NotNull(message = "[发出设备状态]不能为空")

@ApiModelProperty(value = "发出设备状态(1不正常2正常)", required = true, example = "2")

private String issueDeviceSituation;

/\*\*

\* 接收设备状态

\*/

@NotNull(message = "[接收设备状态]不能为空")

@ApiModelProperty(value = "接收设备状态(1不正常2正常)", required = true, example = "2")

private String receiveDeviceSituation;

/\*\*

\* 备注

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "备注", example = "备注")

private String remarks;

/\*\*

\* 设备信息ID

\*/

@NotBlank(message = "[设备信息ID]不能为空")

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "设备信息ID", required = true, example = "1659396771864166402")

private String deviceInfoId;

/\*\*

\* 附件Id

\*/

@Length(max = 500, message = "附件Id不能超过500")

@ApiModelProperty(value = "附件Id", example = "1640520011505266689")

private String fileId;

/\*\*

\* 附件Id集合

\*/

@ApiModelProperty(value = "附件Id集合")

private List<String> fileIdList;

}

/\*\*

\* 设备流转记录查询条件实体类

\*

\* @author wanggang

\* @date 2023/5/22 14:07

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(value = "设备流转记录查询条件实体类")

public class DeviceCirculationRecordSelectDTO extends PageEntity {

/\*\*

\* 模糊字段

\*/

@ApiModelProperty(value = "设备编号、设备名称、发出地点、接收地点、接收人", name = "vagueField", dataType = "String")

private String vagueField;

/\*\*

\* 流转类型

\*/

@ApiModelProperty(value = "流转类型", name = "circulationType", dataType = "String")

private String circulationType;

/\*\*

\* 发出设备状态

\*/

@ApiModelProperty(value = "发出设备状态", name = "issueDeviceSituation", dataType = "String")

private String issueDeviceSituation;

/\*\*

\* 接收设备状态

\*/

@ApiModelProperty(value = "接收设备状态", name = "receiveDeviceSituation", dataType = "String")

private String receiveDeviceSituation;

/\*\*

\* 流转日期开始时间

\*/

@ApiModelProperty(value = "流转日期开始时间,格式：yyyy-MM-dd", name = "startTime", dataType = "String")

private String startTime;

/\*\*

\* 流转日期结束时间

\*/

@ApiModelProperty(value = "流转日期结束时间,格式：yyyy-MM-dd", name = "endTime", dataType = "String")

private String endTime;

}

@Data

@ExcelIgnoreUnannotated

@EqualsAndHashCode

@ApiModel(description = "设备分类添加、修改数据传输类")

public class DeviceClassifyInfoDTO {

@ApiModelProperty(name = "id" , value = "主键")

private String id;

@ApiModelProperty(value = "租户ID")

private String tenantId;

@ApiModelProperty(value = "device\_classify\_number" , example = "20231101")

private String deviceClassifyNumber;

@ApiModelProperty(value = "设备分类名称" , required = true, example = "测试类型设备")

@NotBlank(message = "设备分类名称必填")

@Length(max = 32, message = "设备分类名称不能超过32个字符")

private String deviceClassifyName;

@ApiModelProperty(value = "设备分类简称" , example = "测试")

private String shortName;

@ApiModelProperty(value = "设备名称" , required = true, example = "测试设备")

@NotBlank(message = "设备名称必填")

@Length(max = 32, message = "设备名称不能超过32个字符")

private String deviceName;

@ApiModelProperty(value = "规格型号" , required = true, example = "ZX01")

@NotBlank(message = "设备规格型号必填")

@Length(max = 20, message = "设备规格型号不能超过20个字符")

private String model;

@ApiModelProperty(value = "测量范围(量程)" , example = "100.1001")

private String measureRange;

@ApiModelProperty(value = "精度" , example = "Z")

private String precision;

@ApiModelProperty(value = "准确度等级" , example = "1")

private String accuracyLevel;

@ApiModelProperty(value = "不确定度" , example = "0.006")

private String uncertainty;

@ApiModelProperty(value = "设备分类（1:A;2:B;3:C;4:S;）" , required = true, example = "1")

@NotBlank(message = "设备分类必填")

private String deviceClassify;

@ApiModelProperty(value = "资产分类(1:固定资产;2:耗材;)" , required = true, example = "1")

@NotBlank(message = "资产分类必填")

private String assetClassify;

@ApiModelProperty(value = "设备归属权(1:北域;2:总站;)" , example = "1")

@NotBlank(message = "归属权必填")

private String ownership;

@ApiModelProperty(value = "设备原值(元)" , example = "100.009")

private BigDecimal deviceOriginalValue;

@ApiModelProperty(value = "数量" , hidden = true)

private Integer quantity;

@ApiModelProperty(value = "生产厂家ID（对应厂商ID）" , example = "1658385301195239426")

@NotBlank(message = "生产厂家必填")

private String productionId;

@ApiModelProperty(value = "供货商ID（对应厂商ID）" , example = "1658401660834721793")

@NotBlank(message = "供货商必填")

private String supplierId;

@ApiModelProperty(value = "备注" , example = "测试数据")

@Length(max = 100, message = "备注不能超过100个字符")

private String remarks;

public DeviceClassifyInfo transferEntity() {

DeviceClassifyInfo entity = new DeviceClassifyInfo();

entity.setId(this.id);

entity.setDeviceClassifyNumber(this.deviceClassifyNumber);

entity.setDeviceClassifyName(this.getDeviceClassifyName());

entity.setShortName(this.shortName);

entity.setDeviceName(this.deviceName);

entity.setModel(this.model);

entity.setMeasureRange(this.measureRange);

entity.setAccuracyLevel(this.accuracyLevel);

entity.setUncertainty(this.uncertainty);

entity.setDeviceClassify(this.deviceClassify);

entity.setAssetClassify(this.assetClassify);

entity.setOwnership(this.ownership);

entity.setPrecision(this.precision);

entity.setDeviceOriginalValue(this.deviceOriginalValue);

entity.setQuantity(this.quantity);

entity.setProductionId(this.productionId);

entity.setSupplierId(this.supplierId);

entity.setRemarks(this.remarks);

return entity;

}

}

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(description = "设备分类查询条件实体类")

public class DeviceClassifyInfoSelectDTO extends PageEntity {

@ApiModelProperty(value = "检索条件：分类编号、分类名称、设备名称、规格型号")

private String vagueField;

private String tenantId;

@ApiModelProperty(value = "创建日期开始时间，格式：" + DateTimeConstant.DATE\_TIME,dataType = "LocalDateTime", example = "2023-05-12 00:00:00")

@JsonFormat(pattern = DateTimeConstant.DATE\_TIME, timezone = "GMT+8", locale = "zh\_CN")

@DateTimeFormat(pattern = DateTimeConstant.DATE\_TIME)

private LocalDateTime startTime;

@ApiModelProperty(hidden = true)

private String startTimeString;

@ApiModelProperty(value = "创建日期开始时间，格式：" + DateTimeConstant.DATE\_TIME,dataType = "LocalDateTime", example = "2023-05-12 00:00:00")

@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss", timezone = "GMT+8", locale = "zh\_CN")

@DateTimeFormat(pattern = "yyyy-MM-dd HH:mm:ss")

private LocalDateTime endTime;

@ApiModelProperty(hidden = true)

private String endTimeString;

}

/\*\*

\* 设备管理表

\*

\* @author wanggang

\* @date 2023/5/8 9:51

\*/

@Data

@ApiModel(value = "设备管理表")

@ExcelIgnoreUnannotated

public class DeviceInfoDTO {

//设备分类的字段--------------------

/\*\*

\* 设备分类编号

\*/

@ApiModelProperty(value = "设备分类编号", hidden = true)

@ExcelProperty(value = "分类编号")

private String deviceClassifyNumber;

/\*\*

\* 设备分类名称

\*/

@ApiModelProperty(value = "设备分类名称", hidden = true)

private String deviceClassifyName;

/\*\*

\* 设备名称

\*/

@ApiModelProperty(value = "设备名称", hidden = true)

private String deviceName;

/\*\*

\* 规格/型号

\*/

@ApiModelProperty(value = "规格/型号", hidden = true)

private String model;

/\*\*

\* 精度

\*/

@ApiModelProperty(value = "精度", hidden = true)

private String precision;

/\*\*

\* 测量范围(量程)

\*/

@ApiModelProperty(value = "测量范围(量程)", hidden = true)

private String measureRange;

/\*\*

\* 准确度等级

\*/

@ApiModelProperty(value = "准确度等级", hidden = true)

private String accuracyLevel;

/\*\*

\* 不确定度

\*/

@ApiModelProperty(value = "不确定度", hidden = true)

private String uncertainty;

/\*\*

\* 设备分类

\*/

@ApiModelProperty(value = "设备分类（1:A;2:B;3:C;4:S;）", hidden = true)

private String deviceClassify;

/\*\*

\* 资产分类

\*/

@ApiModelProperty(value = "资产分类(1:固定资产;2:耗材;)", hidden = true)

private String assetClassify;

/\*\*

\* 设备归属

\*/

@ApiModelProperty(value = "设备归属权(1:北域;2:总站;)", hidden = true)

private String ownership;

/\*\*

\* 生产厂家ID（对应厂商ID）

\*/

@ApiModelProperty(value = "生产厂家ID（对应厂商ID）", hidden = true)

private String productionId;

/\*\*

\* 生产厂家

\*/

@ApiModelProperty(value = "生产厂家", hidden = true)

private String productionName;

/\*\*

\* 供货商

\*/

@ApiModelProperty(value = "供货商", hidden = true)

private String supplierName;

/\*\*

\* 设备原值(元)

\*/

@ApiModelProperty(value = "设备原值(元)", hidden = true)

private Double deviceOriginalValue;

/\*\*

\* 联系人

\*/

@ApiModelProperty(value = "联系人", hidden = true)

private String supplierContactsName;

/\*\*

\* 联系电话

\*/

@ApiModelProperty(value = "联系电话", hidden = true)

private String supplierContactPhone;

//设备信息字段--------------------

/\*\*

\* 设备分类ID

\*/

@NotBlank(message = "[设备分类ID]不能为空")

@Length(max = 64, message = "设备分类ID长度不能超过64")

@ApiModelProperty(value = "设备分类ID", required = true, example = "1656847984220512212")

private String deviceClassifyInfoId;

/\*\*

\* 部门ID

\*/

@NotBlank(message = "[部门ID]不能为空")

@Length(max = 64, message = "部门ID长度不能超过64")

@ApiModelProperty(value = "部门ID", required = true, example = "1656847984220512212")

private String departmentId;

/\*\*

\* 部门名称

\*/

@ApiModelProperty(value = "部门名称")

@ExcelProperty(value = "使用部门")

private String departmentName;

/\*\*

\* 设备ID

\*/

@Length(max = 64, message = "设备ID长度不能超过64")

@ApiModelProperty(value = "设备ID")

private String id;

/\*\*

\* 设备编号

\*/

@Length(max = 64, message = "设备编号长度不能超过64")

@ExcelProperty(value = "设备编号")

@ApiModelProperty(value = "设备编号", example = "系统自动生成")

private String deviceNumber;

/\*\*

\* 出厂编号

\*/

@NotBlank(message = "[出厂编号]不能为空")

@Length(max = 64, message = "出厂编号长度不能超过20")

@ExcelProperty(value = "出厂编号")

@ApiModelProperty(value = "出厂编号", required = true, example = "C123")

private String exworksNumber;

/\*\*

\* 量值溯源

\*/

@NotBlank(message = "[量值溯源]不能为空")

@Length(max = 64, message = "量值溯源长度不能超过64")

@ExcelProperty(value = "量值溯源")

@ApiModelProperty(value = "量值溯源", required = true, example = "1")

private String assetSource;

/\*\*

\* 责任人

\*/

@NotBlank(message = "[责任人]不能为空")

@Length(max = 64, message = "责任人长度不能超过64")

@ApiModelProperty(value = "责任人", required = true, example = "1656847984220512212")

private String personLiable;

/\*\*

\* 责任人名

\*/

@ApiModelProperty(value = "责任人")

@ExcelProperty(value = "责任人")

private String personLiableName;

/\*\*

\* 购置日期

\*/

@NotNull(message = "[购置日期]不能为空")

@JsonFormat(pattern = "yyyy-MM-dd")

@ApiModelProperty(value = "购置日期", required = true, example = "2023-5-12")

private Date purchaseTime;

/\*\*

\* 购置日期

\*/

@ApiModelProperty(value = "购置日期", hidden = true)

@ExcelProperty(value = "购置日期")

private String purchaseTimeTran;

/\*\*

\* 使用状态（1:使用;0:停用;-1:报废;）

\*/

@NotNull(message = "[使用状态（1:使用;0:停用;-1:报废;）]不能为空")

@ApiModelProperty(value = "使用状态（1:使用;0:停用;-1:报废;）", required = true, example = "1")

private String useStatus;

/\*\*

\* 使用状态（1:使用;0:停用;-1:报废;）

\*/

@ApiModelProperty(value = "使用状态（1:使用;0:停用;-1:报废;）", hidden = true)

@ExcelProperty(value = "使用状态")

private String useStatusDisplay;

/\*\*

\* 设备状态

\*/

@ApiModelProperty(value = "设备状态", hidden = true)

private String deviceStatus;

/\*\*

\* 流程状态

\*/

@ApiModelProperty(value = "流程状态", hidden = true)

private String flowStatus;

/\*\*

\* 设备存放货架信息ID

\*/

@NotBlank(message = "[设备存放货架信息ID]不能为空")

@Length(max = 64, message = "设备存放货架信息ID长度不能超过64")

@ApiModelProperty(value = "设备存放货架信息ID", required = true, example = "1656847984220512212")

private String deviceShelvesInfoId;

/\*\*

\* 存放编号

\*/

@ExcelProperty(value = "存放编号")

@ApiModelProperty(value = "存放编号")

private String storageCode;

/\*\*

\* 设备存放室ID

\*/

@ApiModelProperty(value = "设备存放室ID", hidden = true)

private String deviceRoomId;

/\*\*

\* 存放室名称

\*/

@ExcelProperty(value = "存放位置")

@ApiModelProperty(value = " 存放室名称")

private String roomName;

/\*\*

\* 是否自动采集(0否;1是)

\*/

@NotNull(message = "[是否自动采集(0否;1是)]不能为空")

@ApiModelProperty(value = "是否自动采集(0否;1是)", required = true, example = "1")

private Integer automaticCollection;

/\*\*

\* 是否自动采集(0否;1是)

\*/

@ApiModelProperty(value = "是否自动采集(0否;1是)", hidden = true)

@ExcelProperty(value = "是否自动采集")

private String automaticCollectionTran;

/\*\*

\* 检定周期（1:3个月;2:6个月;）

\*/

@ExcelProperty(value = "检定周期")

@ApiModelProperty(value = "检定周期（1:3个月;2:6个月;）", example = "1")

private String verificationCycle;

/\*\*

\* 出厂日期

\*/

@NotNull(message = "出厂日期不能为空")

@JsonFormat(pattern = "yyyy-MM-dd")

@ApiModelProperty(value = "出厂日期", required = true, example = "2023-5-12")

private Date exworksTime;

/\*\*

\* 出厂日期

\*/

@ExcelProperty(value = "出厂日期")

@ApiModelProperty(value = "出厂日期", hidden = true)

private String exworksTimeTran;

/\*\*

\* 备注

\*/

@Length(max = 500, message = "备注不能超过500")

@ExcelProperty(value = "备注")

@ApiModelProperty(value = "备注", example = "备注")

private String remarks;

/\*\*

\* 检定状态

\*/

@NotEmpty(message = "检定状态不能为空")

@ApiModelProperty(value = "检定状态", required = true, example = "1")

private String verificationStatus;

/\*\*

\* 在账状态（1:在库;2:在账;3:报废;）

\*/

@ExcelProperty(value = "在账状态")

@ApiModelProperty(value = "在账状态（1:在用;2:在库;0:报废;）", hidden = true)

private String onAccountStatusDisplay;

/\*\*

\* 资产分类

\*/

@ApiModelProperty(value = "资产分类", hidden = true)

private String assetClassifyDisplay;

/\*\*

\* 资产编号

\*/

@ExcelProperty(value = "资产编号")

@ApiModelProperty(value = "资产编号", hidden = true)

private String assetCode;

/\*\*

\* 登记时间

\*/

@ApiModelProperty(value = "登记时间", hidden = true)

private Date registerDate;

/\*\*

\* 登记时间

\*/

@ExcelProperty(value = "登记日期")

@ApiModelProperty(value = "登记时间", hidden = true)

private String registerDateTran;

/\*\*

\* 设备价格

\*/

@ExcelProperty(value = "设备价格（元）")

@ApiModelProperty(value = "设备价格", hidden = true)

private BigDecimal devicePrice;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

@ApiModelProperty(value = "删除标志（1:有效;0:删除;默认1）", hidden = true)

private Integer delFlag;

/\*\*

\* 设备编号(交通行业)

\*/

@ExcelProperty(value = "设备编号(交通行业)")

@ApiModelProperty(value = "设备编号(交通行业)", example = "JT123456")

private String trafficDeviceNumber;

/\*\*

\* 设备名称(交通行业)

\*/

@ExcelProperty(value = "设备名称(交通行业)")

@ApiModelProperty(value = "设备名称(交通行业)", example = "交通设备名称")

private String trafficDeviceName;

}

/\*\*

\* 设备管理查询条件实体类

\*

\* @author wanggang

\* @date 2023/5/8 16:03

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(value = "设备管理查询条件实体类")

public class DeviceInfoSelectDTO extends PageEntity {

/\*\*

\* 设备管理ID

\*/

@ApiModelProperty(value = "设备管理ID", name = "id", dataType = "String")

private String id;

/\*\*

\* 模糊字段

\*/

@ApiModelProperty(value = "设备编号、设备名称、型号", name = "vagueField", dataType = "String")

private String vagueField;

/\*\*

\* 使用状态

\*/

@ApiModelProperty(value = "使用状态", name = "useStatus", dataType = "String")

private String useStatus;

/\*\*

\* 设备状态

\*/

@ApiModelProperty(value = "设备状态", name = "deviceStatus", dataType = "String")

private String deviceStatus;

/\*\*

\* 流程状态

\*/

@ApiModelProperty(value = "流程状态", name = "flowStatus", dataType = "String")

private String flowStatus;

/\*\*

\* 检定状态

\*/

@ApiModelProperty(value = "检定状态", name = "verificationStatus", dataType = "String")

private String verificationStatus;

/\*\*

\* 证书到期开始时间

\*/

@ApiModelProperty(value = "证书到期开始时间,格式：yyyy-MM-dd", name = "startTime", dataType = "String")

private String startTime;

/\*\*

\* 证书到期结束时间

\*/

@ApiModelProperty(value = "证书到期结束时间,格式：yyyy-MM-dd", name = "endTime", dataType = "String")

private String endTime;

}

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(description = "场所视图类")

public class DevicePlaceDTO {

@ApiModelProperty(value = "主键")

private String id;

@ApiModelProperty(value = " 组织id")

private String orgId;

@ApiModelProperty(value = " 场所名称(二级组织名称)")

private String placeName;

@ApiModelProperty(value = " 场所编码(一个字母)")

private String placeCode;

@ApiModelProperty(value = " 场所位置及描述")

private String placeDesc;

@ApiModelProperty(value = " 备注")

private String remarks;

@ApiModelProperty(value = " 删除标志（1:有效;0:删除;默认1）")

private Integer delFlag;

}

/\*\*

\* 设备的记录查询条件实体类

\*

\* @author wanggang

\* @date 2023/5/9 16:03

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(value = "设备运行记录查询条件实体类")

public class DeviceRecordSelectDTO extends PageEntity {

/\*\*

\* 设备信息表ID

\*/

@ApiModelProperty(value = "设备信息表ID", dataType = "String", required = true)

private String deviceInfoId;

/\*\*

\* 分类(1:保养;2:维修)

\*/

@ApiModelProperty(value = "设备保养和维修的分类(分类(1:保养;2:维修))", dataType = "String")

private Integer classify;

}

/\*\*

\* 设备保养维修记录表

\*

\* @author wanggang

\* @date 2023/5/10 10:31

\* @TableName device\_repair\_maintenance\_record

\*/

@Data

@ApiModel(value = "设备保养维修记录表")

public class DeviceRepairMaintenanceRecordDTO {

/\*\*

\* ID

\*/

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "ID")

private String id;

/\*\*

\* 设备编号

\*/

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "设备编号", hidden = true)

private String deviceNumber;

/\*\*

\* 使用部门ID

\*/

@NotBlank(message = "[使用部门ID]不能为空")

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "使用部门ID", required = true, example = "1628617301980041217")

private String useDepartmentId;

/\*\*

\* 负责人ID

\*/

@NotBlank(message = "[负责人ID]不能为空")

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "负责人ID", required = true, example = "1628267730502344706")

private String headId;

/\*\*

\* 参与人员

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "参与人员", hidden = true)

private String involvedStaff;

/\*\*

\* 参与人员ID集

\*/

@ApiModelProperty(value = "参与人员ID集")

private List<String> involvedStaffList;

/\*\*

\* 分类(1:保养;2:维修)

\*/

@NotNull(message = "[分类(1:保养;2:维修)]不能为空")

@ApiModelProperty(value = "分类(1:保养;2:维修)", required = true, example = "1")

private String classify;

/\*\*

\* 发生时间

\*/

@NotNull(message = "[发生时间]不能为空")

@JsonFormat(pattern = "yyyy-MM-dd")

@ApiModelProperty(value = "发生时间", required = true, example = "2023-05-23")

private Date happenTime;

/\*\*

\* 情况说明

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "情况说明", example = "情况说明")

private String situationDescription;

/\*\*

\* 校准结果(1:合格;0:不合格)

\*/

@ApiModelProperty(value = "校准结果(1:合格;0:不合格)", required = true, example = "1")

private Integer calibrationResult;

/\*\*

\* 运行使用状态

\*/

@ApiModelProperty(value = "运行使用状态", required = true, example = "1")

private String runUseStatus;

/\*\*

\* 附件Id

\*/

@Length(max = 500, message = "附件Id不能超过500")

@ApiModelProperty(value = "附件Id", example = "1640520011505266689")

private String fileId;

/\*\*

\* 附件Id集合

\*/

@ApiModelProperty(value = "附件Id集合")

private List<String> fileIdList;

/\*\*

\* 备注

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "备注", example = "备注")

private String remarks;

/\*\*

\* 设备信息ID

\*/

@NotBlank(message = "[设备信息ID]不能为空")

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "设备信息ID", required = true, example = "1628267730502344706")

private String deviceInfoId;

}

/\*\*

\* 设备保养维修记录查询条件实体类

\*

\* @author wanggang

\* @date 2023/5/23 14:07

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(value = "设备保养维修记录查询条件实体类")

public class DeviceRepairMaintenanceRecordSelectDTO extends PageEntity {

/\*\*

\* 模糊字段

\*/

@ApiModelProperty(value = "设备编号、设备名称、情况说明、文件名称", name = "vagueField", dataType = "String")

private String vagueField;

/\*\*

\* 保养维修分类(1:保养;2:维修)

\*/

@ApiModelProperty(value = "保养维修分类(1:保养;2:维修) ps:必传", required = true, name = "classify", dataType = "String")

private String classify;

/\*\*

\* 使用部门

\*/

@ApiModelProperty(value = "使用部门", name = "useDepartmentId", dataType = "String")

private String useDepartmentId;

/\*\*

\* 运行使用状态

\*/

@ApiModelProperty(value = "运行使用状态", name = "runUseStatus", dataType = "String")

private String runUseStatus;

/\*\*

\* 校准结果

\*/

@ApiModelProperty(value = "校准结果", name = "calibrationResult", dataType = "Integer")

private Integer calibrationResult;

/\*\*

\* 发生日期开始时间

\*/

@ApiModelProperty(value = "发生日期开始时间,格式：yyyy-MM-dd", name = "startTime", dataType = "String")

private String startTime;

/\*\*

\* 发生日期结束时间

\*/

@ApiModelProperty(value = "发生日期结束时间,格式：yyyy-MM-dd", name = "endTime", dataType = "String")

private String endTime;

}

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(description = "设备存放室视图类")

public class DeviceRoomDTO {

@ApiModelProperty(value = "主键")

private String id;

@ApiModelProperty(value = "场所id", required = true, example = "1660863968203886593")

private String placeId;

@ApiModelProperty(value = " 场所名称", example = "客户查询站")

private String placeName;

@ApiModelProperty(value = " 存放室名称", required = true, example = "测试存放室")

private String roomName;

@ApiModelProperty(value = " 房间编号(场所编号+2位数字)")

private String roomCode;

@ApiModelProperty(value = " 描述")

private String roomDesc;

}

@Data

@ApiModel(value = "货架信息添加实体类")

public class DeviceShelvesAddInfoDTO {

@ApiModelProperty(value = "存放室id", required = true)

@NotBlank(message = "存放室不能为空")

private String roomId;

@ApiModelProperty(value = "场所id", required = true)

@NotBlank(message = "场所不能为空")

private String placeId;

@ApiModelProperty(value = "货架名称", required = true, example = "测试货架")

@NotBlank(message = "货架名称不能为空")

@Size(max = 255)

private String shelvesName;

@ApiModelProperty(value = "货架编号【系统自动生成，保留字段】", example = "FG")

private String shelvesCode;

@ApiModelProperty(value = "货架层号开始位置", required = true, example = "1")

@NotBlank(message = "货架层号开始位置不能为空")

private Integer layerStart;

@ApiModelProperty(value = "货架层号结束位置", required = true, example = "4")

@NotBlank(message = "货架层号结束位置不能为空")

private Integer layerEnd;

@ApiModelProperty(value = "位置序号开始位置", required = true, example = "1")

@NotBlank(message = "位置序号开始位置不能为空")

private Integer positionStart;

@ApiModelProperty(value = "位置序号结束位置", required = true, example = "5")

@NotBlank(message = "位置序号结束位置不能为空")

private Integer positionEnd;

}

/\*\*

\* 设备存放货架信息表

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(value = "设备存放货架信息表")

public class DeviceShelvesInfoDTO extends PageEntity {

@ApiModelProperty(value = "检索条件: 货架名称、货架编号、层号", dataType = "String")

private String vagueField;

private String tenantId;

@ApiModelProperty(value = "创建日期开始时间，格式：" + DateTimeConstant.DATE\_TIME,dataType = "LocalDateTime", example = "2023-05-12 00:00:00")

@JsonFormat(pattern = DateTimeConstant.DATE\_TIME, timezone = "GMT+8", locale = "zh\_CN")

@DateTimeFormat(pattern = DateTimeConstant.DATE\_TIME)

private LocalDateTime startTime;

@ApiModelProperty(hidden = true)

private String startTimeString;

@ApiModelProperty(value = "创建日期开始时间，格式：" + DateTimeConstant.DATE\_TIME,dataType = "LocalDateTime", example = "2023-05-12 00:00:00")

@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss", timezone = "GMT+8", locale = "zh\_CN")

@DateTimeFormat(pattern = "yyyy-MM-dd HH:mm:ss")

private LocalDateTime endTime;

@ApiModelProperty(hidden = true)

private String endTimeString;

}

@Data

@ApiModel(value = "设备存放货架更新类")

public class DeviceShelvesInfoUpdateDTO {

@ApiModelProperty(value = "id")

@NotBlank(message = "主键不能为空")

private String id;

@ApiModelProperty(value = "租户ID")

private String tenantId;

@ApiModelProperty(value = "设备信息ID")

private String deviceInfoId;

@ApiModelProperty(value = "设备编号")

private String deviceNumber;

@ApiModelProperty(value = "存放编号")

private String storageCode;

@ApiModelProperty(value = "货架名称")

private String shelvesName;

@ApiModelProperty(value = "货架编号")

private String shelvesCode;

@ApiModelProperty(value = "货架层号")

private String layerCode;

@ApiModelProperty(value = "位置序号")

private String positionCode;

@ApiModelProperty(value = "位置描述")

@NotBlank(message = "位置描述不能为空")

private String positionDesc;

@ApiModelProperty(value = "购买时间")

private LocalDateTime purchaseTime;

@ApiModelProperty(value = "备注")

private String remarks;

}

/\*\*

\* 设备检定记录表

\*

\* @author wanggang

\* @date 2023/5/10 14:18

\* @TableName device\_verification\_record

\*/

@Data

@ApiModel(value = "设备检定记录表")

public class DeviceVerificationRecordDTO {

/\*\*

\* 检定记录ID

\*/

@ApiModelProperty("检定记录ID")

@Length(max = 64, message = "编码长度不能超过64")

private String id;

/\*\*

\* 设备信息ID

\*/

@NotBlank(message = "[设备信息ID]不能为空")

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "设备信息ID", required = true, example = "1656950797366423551")

private String deviceInfoId;

/\*\*

\* 设备编号

\*/

@ApiModelProperty(value = "设备编号", hidden = true)

private String deviceNumber;

/\*\*

\* 设备名称

\*/

@ApiModelProperty(value = "设备名称", hidden = true)

private String deviceName;

/\*\*

\* 证书编号

\*/

@Length(max = 20, message = "编码长度不能超过20")

@ApiModelProperty(value = "证书编号", required = true, example = "ZS123456789")

private String certificateNumber;

/\*\*

\* 检定日期

\*/

@NotNull(message = "[检定日期]不能为空")

@JsonFormat(pattern = "yyyy-MM-dd")

@ApiModelProperty(value = "检定日期", required = true, example = "2023-05-18")

private Date verificationDate;

/\*\*

\* 证书类型

\*/

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "证书类型", required = true, example = "1")

private String certificateClassification;

/\*\*

\* 检定单位

\*/

@NotEmpty(message = "[检定单位]不能为空")

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "检定单位", required = true, example = "1656950797366423551")

private String verificationCompany;

/\*\*

\* 有效开始日期

\*/

@JsonFormat(pattern = "yyyy-MM-dd")

@ApiModelProperty(value = "有效开始日期", required = true, example = "2023-05-18")

private Date validStart;

/\*\*

\* 有效结束日期

\*/

@JsonFormat(pattern = "yyyy-MM-dd")

@ApiModelProperty(value = "有效结束日期", required = true, example = "2023-05-18")

private Date validEnd;

/\*\*

\* 检定结果(1:合格;0:不合格)

\*/

@ApiModelProperty("检定结果(1:合格;0:不合格)")

private Integer verificationResult;

/\*\*

\* 备注

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "备注", example = "备注")

private String remarks;

/\*\*

\* 附件Id

\*/

@Length(max = 500, message = "附件Id不能超过500")

@ApiModelProperty(value = "附件Id", hidden = true)

private String fileId;

/\*\*

\* 附件Id集合

\*/

@ApiModelProperty(value = "附件Id集合")

private List<String> fileIdList;

/\*\*

\* 检定方式

\*/

@ApiModelProperty("检定方式")

private String verificationStyle;

}

/\*\*

\* 设备检定记录查询条件实体类

\*

\* @author wanggang

\* @date 2023/5/18 9:03

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(value = "设备检定记录查询条件实体类")

public class DeviceVerificationRecordSelectDTO extends PageEntity {

/\*\*

\* 模糊字段

\*/

@ApiModelProperty(value = "设备编号、设备名称、证书编号、文件名称", name = "vagueField", dataType = "String")

private String vagueField;

/\*\*

\* 检定单位

\*/

@ApiModelProperty(value = "检定单位", name = "verificationCompany", dataType = "String")

private String verificationCompany;

/\*\*

\* 证书分类

\*/

@ApiModelProperty(value = "证书分类", name = "certificateClassification", dataType = "String")

private String certificateClassification;

/\*\*

\* 检定结果

\*/

@ApiModelProperty(value = "检定结果", name = "verificationResult", dataType = "Integer")

private Integer verificationResult;

/\*\*

\* 检定日期开始时间

\*/

@ApiModelProperty(value = "检定日期开始时间,格式：yyyy-MM-dd", name = "startTime", dataType = "String")

private String startTime;

/\*\*

\* 检定日期结束时间

\*/

@ApiModelProperty(value = "检定日期结束时间,格式：yyyy-MM-dd", name = "endTime", dataType = "String")

private String endTime;

}

/\*\*

\* 设备运行记录表

\*

\* @author wanggang

\* @date 2023/5/9 12:00

\* @TableName device\_working\_record

\*/

@Data

@ApiModel(value = "设备运行记录表")

public class DeviceWorkingRecordDTO {

/\*\*

\* ID

\*/

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "ID", hidden = true)

private String id;

/\*\*

\* 设备编号

\*/

@ApiModelProperty(value = "设备编号", hidden = true)

private String deviceNumber;

/\*\*

\* 设备名称

\*/

@ApiModelProperty(value = "设备名称", hidden = true)

private String deviceName;

/\*\*

\* 设备信息ID

\*/

@NotNull(message = "[运行开始时间]不能为空")

@ApiModelProperty(value = "设备信息ID", required = true, example = "1656950797366423554")

private String deviceInfoId;

/\*\*

\* 报告编号

\*/

@NotEmpty(message = "[报告编号]不能为空")

@ApiModelProperty(value = "报告编号", required = true, example = "BG123456789")

private String reportNumber;

/\*\*

\* 运行开始时间

\*/

@NotNull(message = "[运行开始时间]不能为空")

@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")

@ApiModelProperty(value = "运行开始时间", required = true, example = "2023-05-24 16:14:11")

private Date workingStartTime;

/\*\*

\* 运行结束时间

\*/

@NotNull(message = "[运行结束时间]不能为空")

@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")

@ApiModelProperty(value = "运行结束时间", required = true, example = "2023-05-25 16:14:11")

private Date workingEndTime;

/\*\*

\* 使用前状态（1:不正常;2:正常;）

\*/

@NotEmpty(message = "[使用前状态]不能为空")

@ApiModelProperty(value = "使用前状态（1:不正常;2:正常;）", example = "2")

private String useFrontStatus;

/\*\*

\* 使用后状态（1:不正常;2:正常;）

\*/

@NotEmpty(message = "[使用后状态]不能为空")

@ApiModelProperty(value = "使用后状态（1:不正常;2:正常;）", example = "2")

private String useEndStatus;

/\*\*

\* 检测人员ID

\*/

@NotBlank(message = "[检测人员ID]不能为空")

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "检测人员ID", required = true, example = "1628267730502344706")

private String detectionStaffId;

/\*\*

\* 审核人员ID

\*/

@ApiModelProperty(value = "审核人员ID", example = "1628267730502344706")

private String reviewPerson;

/\*\*

\* 审核意见

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "审核意见", example = "审核意见")

private String reviewComments;

/\*\*

\* 异常状况描述

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "异常状况描述", example = "异常状况描述")

private String exceptionSituation;

/\*\*

\* 备注

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "备注", example = "备注")

private String remarks;

}

/\*\*

\* 设备运行记录查询条件实体类

\*

\* @author wanggang

\* @date 2023/5/18 9:03

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(value = "设备运行记录查询条件实体类")

public class DeviceWorkingRecordSelectDTO extends PageEntity {

/\*\*

\* 模糊字段

\*/

@ApiModelProperty(value = "设备编号、设备名称、报告编号、异常状况", name = "vagueField", dataType = "String")

private String vagueField;

/\*\*

\* 使用前状态

\*/

@ApiModelProperty(value = "使用前状态", name = "useFrontStatus", dataType = "String")

private String useFrontStatus;

/\*\*

\* 使用后状态

\*/

@ApiModelProperty(value = "使用后状态", name = "useEndStatus", dataType = "String")

private String useEndStatus;

/\*\*

\* 创建日期开始时间

\*/

@ApiModelProperty(value = "创建日期开始时间,格式：yyyy-MM-dd HH:mm:ss", name = "startTime", dataType = "String")

private String startTime;

/\*\*

\* 创建日期结束时间

\*/

@ApiModelProperty(value = "创建日期结束时间,格式：yyyy-MM-dd HH:mm:ss", name = "endTime", dataType = "String")

private String endTime;

}

@Data

@ApiModel(description = "厂商联系人数据传输类")

@ExcelIgnoreUnannotated

@EqualsAndHashCode

public class SupplierContactsInfoDTO {

@ApiModelProperty(value = "主键")

private String id;

@ApiModelProperty(value = "厂商ID", required = true, example = "1")

@NotBlank(message = "厂商ID必填")

private String supplierId;

@ApiModelProperty(value = "姓名", required = true, example = "李四")

@NotBlank(message = "联系人姓名必填")

@Length(max = 12, message = "联系人姓名不能超过12个字符")

@ExcelProperty(value = "姓名")

private String name;

@ApiModelProperty(value = "别名")

@Length(max = 12, message = "设联系人别名不能超过12个字符")

@ExcelProperty(value = "别名")

private String alias;

@ApiModelProperty(value = "职位", required = true, example = "经理")

@NotBlank(message = "职位必填")

@Length(max = 12, message = "职位不能超过12个字符")

@ExcelProperty(value = "职位")

private String position;

@ApiModelProperty(value = "部门名称")

@Length(max = 20, message = "部门名称不能超过20个字符")

@ExcelProperty(value = "部门名称")

private String departmentName;

@ApiModelProperty(value = "性别（0:未知;1:男;2:女;），以字典表为准")

@ExcelProperty(value = "性别")

private String gender;

@ApiModelProperty(value = "微信")

@Length(max = 20, message = "微信不能超过20个字符")

@ExcelProperty(value = "微信")

private String wechat;

@ApiModelProperty(value = "企业邮箱", required = true, example = "firefox@email.com")

@Length(max = 32, message = "企业邮箱不能超过32个字符")

@NotBlank(message = "企业邮箱必填")

@ExcelProperty(value = "企业邮箱")

@Email

private String mailbox;

@ApiModelProperty(value = "qq")

@Length(max = 20, message = "QQ不能超过20个字符")

@ExcelProperty(value = "qq")

private String qq;

@ApiModelProperty(value = "住址")

@Length(max = 64, message = "住址不能超过64个字符")

@ExcelProperty(value = "住址")

private String address;

@ApiModelProperty(value = "电话", required = true, example = "11378248711")

@NotBlank(message = "电话号码必填")

@Length(max = 20, message = "电话号码不能超过20个字符")

@ExcelProperty(value = "电话")

private String telephone;

@ApiModelProperty(value = "备注")

@Length(max = 100, message = "备注不能超过100个字符")

@ExcelProperty(value = "备注")

private String remarks;

public SupplierContactsInfo transferEntity() {

SupplierContactsInfo supplier = new SupplierContactsInfo();

supplier.setId(this.id);

supplier.setSupplierId(this.supplierId);

supplier.setName(this.name);

supplier.setAlias(this.alias);

supplier.setPosition(this.position);

supplier.setDepartmentName(this.departmentName);

supplier.setGender(this.gender);

supplier.setWechat(this.wechat);

supplier.setMailbox(this.mailbox);

supplier.setQq(this.qq);

supplier.setAddress(this.address);

supplier.setTelephone(this.telephone);

supplier.setRemarks(this.remarks);

return supplier;

}

}

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(description = "供应商联系人查询条件实体类")

public class SupplierContactsInfoSelectDTO extends PageEntity {

@ApiModelProperty(value = "检索条件：姓名、职位、联系电话、住址", dataType = "String")

private String vagueField;

@ApiModelProperty(value = "公司名称", dataType = "String")

private String companyName;

private String tenantId;

@ApiModelProperty(value = "创建日期开始时间，格式：" + DateTimeConstant.DATE\_TIME, dataType = "LocalDateTime", example = "2023-05-12 00:00:00")

@JsonFormat(pattern = DateTimeConstant.DATE\_TIME, timezone = "GMT+8", locale = "zh\_CN")

@DateTimeFormat(pattern = DateTimeConstant.DATE\_TIME)

private LocalDateTime startTime;

@ApiModelProperty(hidden = true)

private String startTimeString;

@ApiModelProperty(value = "创建日期开始时间，格式：" + DateTimeConstant.DATE\_TIME, dataType = "LocalDateTime", example = "2023-05-12 00:00:00")

@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss", timezone = "GMT+8", locale = "zh\_CN")

@DateTimeFormat(pattern = "yyyy-MM-dd HH:mm:ss")

private LocalDateTime endTime;

@ApiModelProperty(hidden = true)

private String endTimeString;

}

/\*\*

\* 厂商信息表

\*

\* @TableName supplier\_info

\*/

@Data

@ExcelIgnoreUnannotated

@ApiModel(description = "厂商数据传输类")

@EqualsAndHashCode(callSuper = false)

public class SupplierInfoDTO {

@ApiModelProperty(value = "主键")

private String id;

@ApiModelProperty(value = "名称", required = true, example = "大华")

@NotBlank(message = "名称必填")

@Length(max = 32, message = "名称不能超过32个字符")

private String name;

@ApiModelProperty(value = "简称", example = "大华")

@Length(max = 20, message = "简称不能超过20个字符")

private String abbreviation;

@ApiModelProperty(value = "地址", example = "重庆市")

@Length(max = 64, message = "地址不能超过64个字符")

private String address;

@ApiModelProperty(value = "行业", required = true, example = "1")

@NotBlank(message = "行业必填")

private String industry;

@ApiModelProperty(value = "类型：1厂家(默认),2供应商,3检定", required = true, example = "1")

@NotBlank(message = "类型必填")

private String type;

@ApiModelProperty(value = "座机", required = true, example = "35556678")

@NotBlank(message = "座机必填")

@Length(max = 20, message = "座机不能超过20个字符")

private String contactTelephone;

@ApiModelProperty(value = "企业邮箱", required = true, example = "firefox@email.com")

@Length(max = 32, message = "企业邮箱不能超过32个字符")

@NotBlank(message = "企业邮箱必填")

@ExcelProperty(value = "企业邮箱")

@Email

private String mailbox;

private String supplierContactId;

@ApiModelProperty(value = "简介", example = "xxxx")

@Length(max = 200, message = "简介不能超过200个字符")

@ExcelProperty(value = "简介")

private String profile;

@ApiModelProperty(value = "备注", example = "xxxx")

@Length(max = 100, message = "备注不能超过100个字符")

@ExcelProperty(value = "备注")

private String remarks;

@ApiModelProperty(value = "厂商联系人信息")

@ExcelProperty(value = "备注")

private List<SupplierContactsInfoDTO> contactList = Collections.emptyList();

public SupplierInfo transferEntity() {

SupplierInfo supplierInfo = new SupplierInfo();

supplierInfo.setId(this.id);

supplierInfo.setName(this.name);

supplierInfo.setAbbreviation(this.abbreviation);

supplierInfo.setAddress(this.address);

supplierInfo.setIndustry(this.industry);

supplierInfo.setType(this.type);

supplierInfo.setContactTelephone(this.contactTelephone);

supplierInfo.setMailbox(this.mailbox);

supplierInfo.setSupplierContactId(this.supplierContactId);

supplierInfo.setProfile(this.profile);

supplierInfo.setRemarks(this.remarks);

return supplierInfo;

}

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(description = "厂商信息查询条件实体类")

public class SupplierInfoSelectDTO extends PageEntity {

@ApiModelProperty(value = "厂商信息:厂商名称、简称、地址", dataType = "String")

private String supplierInfo;

@ApiModelProperty(value = "联系人:姓名、电话", dataType = "String")

private String concacts;

@ApiModelProperty(hidden = true)

private String tenantId;

@ApiModelProperty("厂商类型")

private String type;

@ApiModelProperty(value = "创建日期开始时间，格式：" + DateTimeConstant.DATE\_TIME,dataType = "LocalDateTime", example = "2023-05-12 00:00:00")

@JsonFormat(pattern = DateTimeConstant.DATE\_TIME, timezone = "GMT+8", locale = "zh\_CN")

@DateTimeFormat(pattern = DateTimeConstant.DATE\_TIME)

private LocalDateTime startTime;

@ApiModelProperty(hidden = true)

private String startTimeString;

@ApiModelProperty(value = "创建日期开始时间，格式：" + DateTimeConstant.DATE\_TIME,dataType = "LocalDateTime", example = "2023-05-12 00:00:00")

@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss", timezone = "GMT+8", locale = "zh\_CN")

@DateTimeFormat(pattern = "yyyy-MM-dd HH:mm:ss")

private LocalDateTime endTime;

@ApiModelProperty(hidden = true)

private String endTimeString;

}

@Data

@EqualsAndHashCode(callSuper = false)

public class SupplierInfoUpdateDTO {

@ApiModelProperty(value = "主键")

private String id;

@ApiModelProperty(value = "名称", required = true, example = "大华")

@NotBlank(message = "名称必填")

@Length(max = 32, message = "名称不能超过32个字符")

@ExcelProperty(value = "名称")

private String name;

@ApiModelProperty(value = "简称", example = "大华")

@Length(max = 20, message = "简称不能超过20个字符")

@ExcelProperty(value = "简称")

private String abbreviation;

@ApiModelProperty(value = "地址", example = "重庆市")

@Length(max = 64, message = "地址不能超过64个字符")

@ExcelProperty(value = "地址")

private String address;

@ApiModelProperty(value = "行业", required = true, example = "1")

@NotBlank(message = "行业必填")

@ExcelProperty(value = "行业")

private String industry;

@ApiModelProperty(value = "类型：1厂家(默认),2供应商,3检定", required = true, example = "1")

@NotBlank(message = "类型必填")

@Size(max = 2)

private String type;

@ApiModelProperty(value = "座机", required = true, example = "35556678")

@NotBlank(message = "座机必填")

@Length(max = 20, message = "座机不能超过20个字符")

@ExcelProperty(value = "座机")

private String contactTelephone;

@ApiModelProperty(value = "企业邮箱", required = true, example = "firefox@email.com")

@Length(max = 32, message = "企业邮箱不能超过32个字符")

@NotBlank(message = "企业邮箱必填")

@ExcelProperty(value = "企业邮箱")

@Email

private String mailbox;

private String supplierContactId;

@ApiModelProperty(value = "简介", example = "xxxx")

@Length(max = 200, message = "简介不能超过200个字符")

@ExcelProperty(value = "简介")

private String profile;

@ApiModelProperty(value = "备注", example = "xxxx")

@Length(max = 100, message = "备注不能超过100个字符")

@ExcelProperty(value = "备注")

private String remarks;

public SupplierInfo transferEntity() {

SupplierInfo supplierInfo = new SupplierInfo();

supplierInfo.setId(this.id);

supplierInfo.setName(this.name);

supplierInfo.setAbbreviation(this.abbreviation);

supplierInfo.setAddress(this.address);

supplierInfo.setIndustry(this.industry);

supplierInfo.setType(this.type);

supplierInfo.setContactTelephone(this.contactTelephone);

supplierInfo.setMailbox(this.mailbox);

supplierInfo.setSupplierContactId(this.supplierContactId);

supplierInfo.setProfile(this.profile);

supplierInfo.setRemarks(this.remarks);

return supplierInfo;

}

}

/\*\*

\* 设备资产信息记录表

\*

\* @author wanggang

\* @date 2023/5/15 10:51

\*/

@Data

@ApiModel(value = "设备资产信息记录查询结果实体")

public class DeviceAssetInfoRecordVO {

/\*\*

\* 设备编号

\*/

@ApiModelProperty(value = "设备编号")

private String deviceNumber;

/\*\*

\* 设备名称

\*/

@ApiModelProperty(value = "设备名称")

private String deviceName;

/\*\*

\* ID

\*/

@ApiModelProperty(value = "ID")

private String id;

/\*\*

\* 设备信息ID

\*/

@ApiModelProperty(value = "设备信息ID")

private String deviceInfoId;

/\*\*

\* 资产编号

\*/

@ApiModelProperty(value = "资产编号")

private String assetCode;

/\*\*

\* 资产名称

\*/

@ApiModelProperty(value = "资产名称")

private String assetName;

/\*\*

\* 资产分类

\*/

@ApiModelProperty(value = "资产分类")

private String assetClassify;

/\*\*

\* 资产分类显示

\*/

@ApiModelProperty(value = "资产分类显示")

private String assetClassifyDisplay;

/\*\*

\* 在账状态（1:在库;2:在账;3:报废;）

\*/

@ApiModelProperty(value = "在账状态（1:在库;2:在账;3:报废;）")

private String onAccountStatus;

/\*\*

\* 在账状态（1:在库;2:在账;3:报废;）

\*/

@ApiModelProperty(value = "在账状态显示")

private String onAccountStatusDisplay;

/\*\*

\* 登记时间

\*/

@ApiModelProperty(value = "登记时间")

private Date registerDate;

/\*\*

\* 设备价格

\*/

@ApiModelProperty(value = "设备价格")

private BigDecimal devicePrice;

/\*\*

\* 创建人

\*/

@ApiModelProperty(value = "创建人")

private String createBy;

/\*\*

\* 创建人名

\*/

@ApiModelProperty(value = "创建人名")

private String createName;

/\*\*

\* 创建时间

\*/

@ApiModelProperty(value = "创建时间")

private Date createTime;

/\*\*

\* 修改人

\*/

@ApiModelProperty(value = "修改人")

private String updateBy;

/\*\*

\* 修改人名

\*/

@ApiModelProperty(value = "修改人名")

private String updateName;

/\*\*

\* 修改时间

\*/

@ApiModelProperty(value = "修改时间")

private Date updateTime;

/\*\*

\* 备注

\*/

@ApiModelProperty(value = "备注")

private String remarks;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

@ApiModelProperty("删除标志（1:有效;0:删除;默认1）")

private Integer delFlag;

}

/\*\*

\* 设备授权表

\*

\* @author wanggang

\* @date 2023/6/8 16:29

\*/

@Data

@ApiModel(value = "设备授权查询结果返回")

public class DeviceAuthorizationVO {

/\*\*

\* ID

\*/

@ApiModelProperty("ID")

private String id;

/\*\*

\* 设备信息ID

\*/

@ApiModelProperty("设备信息ID")

private String deviceInfoId;

/\*\*

\* 设备编号

\*/

@ApiModelProperty("设备编号")

private String deviceNumber;

/\*\*

\* 设备名称

\*/

@ApiModelProperty("设备名称")

private String deviceName;

/\*\*

\* 规格/型号

\*/

@ApiModelProperty("规格/型号")

private String model;

/\*\*

\* 出厂编号

\*/

@ApiModelProperty("出厂编号")

private String exworksNumber;

/\*\*

\* 生产厂家ID（对应厂商ID）

\*/

@ApiModelProperty("生产厂家ID（对应厂商ID）")

private String productionId;

/\*\*

\* 生产厂家

\*/

@ApiModelProperty("生产厂家名")

private String productionName;

/\*\*

\* 授权人

\*/

@ApiModelProperty("授权人")

private String authorizationPerson;

/\*\*

\* 授权人名

\*/

@ApiModelProperty("授权人名")

private String authorizationPersonName;

/\*\*

\* 授权人员集

\*/

@ApiModelProperty("授权人员集")

private List<SysUserVO> authorizationPersonList;

/\*\*

\* 责任人

\*/

@ApiModelProperty("责任人")

private String personLiable;

/\*\*

\* 责任人名

\*/

@ApiModelProperty("责任人名")

private String personLiableName;

/\*\*

\* 部门ID

\*/

@ApiModelProperty("部门ID")

private String departmentId;

/\*\*

\* 部门名称

\*/

@ApiModelProperty("部门名称")

private String departmentName;

/\*\*

\* 设备状态

\*/

@ApiModelProperty("设备状态")

private String deviceStatus;

/\*\*

\* 设备状态显示

\*/

@ApiModelProperty("设备状态显示")

private String deviceStatusDisplay;

/\*\*

\* 设备存放货架信息ID

\*/

@ApiModelProperty(value = "设备存放货架信息ID")

private String deviceShelvesInfoId;

/\*\*

\* 设备存放室ID

\*/

@ApiModelProperty(value = "设备存放室ID")

private String deviceRoomId;

/\*\*

\* 存放室名称

\*/

@ApiModelProperty(value = " 存放室名称")

private String roomName;

/\*\*

\* 资产分类

\*/

@ApiModelProperty(value = "资产分类")

private String assetClassify;

/\*\*

\* 资产分类显示

\*/

@ApiModelProperty(value = "资产分类显示")

private String assetClassifyDisplay;

/\*\*

\* 创建时间

\*/

@ApiModelProperty("创建时间")

private Date createTime;

}

/\*\*

\* 设备借用记录表

\*

\* @author wanggang

\* @date 2023/5/22 16:09

\*/

@Data

@ApiModel(value = "设备借用记录查询结果实体")

public class DeviceBorrowRecordVO {

/\*\*

\* ID

\*/

@ApiModelProperty("ID")

private String id;

/\*\*

\* 设备信息ID

\*/

@ApiModelProperty("设备信息ID")

private String deviceInfoId;

/\*\*

\* 设备编号

\*/

@ApiModelProperty("设备编号")

private String deviceNumber;

/\*\*

\* 设备名称

\*/

@ApiModelProperty(value = "设备名称")

private String deviceName;

/\*\*

\* 借出时间

\*/

@ApiModelProperty("借出时间")

private Date lendTime;

/\*\*

\* 借出员工ID

\*/

@ApiModelProperty("借出员工ID")

private String lendStaffId;

/\*\*

\* 借出员工名

\*/

@ApiModelProperty(value = "借出员工名")

private String lendStaffName;

/\*\*

\* 借出部门ID

\*/

@ApiModelProperty("借出部门ID")

private String lendDepartmentId;

/\*\*

\* 借出部门名

\*/

@ApiModelProperty(value = "借出部门名")

private String lendDepartmentName;

/\*\*

\* 使用项目ID

\*/

@ApiModelProperty("使用项目ID")

private String useProjectId;

/\*\*

\* 使用项目名

\*/

@ApiModelProperty("使用项目名")

private String useProjectName;

/\*\*

\* 借出设备状态

\*/

@ApiModelProperty("借出设备状态")

private String lendDeviceStatus;

/\*\*

\* 借出设备状态

\*/

@ApiModelProperty("借出设备状态显示")

private String lendDeviceStatusDisplay;

/\*\*

\* 归还时间

\*/

@ApiModelProperty("归还时间")

private Date returnTime;

/\*\*

\* 归还人员ID

\*/

@ApiModelProperty("归还人员ID")

private String returnStaffId;

/\*\*

\* 归还人员名

\*/

@ApiModelProperty("归还人员名")

private String returnStaffName;

/\*\*

\* 归还设备状态

\*/

@ApiModelProperty("归还设备状态")

private String returnDeviceStatus;

/\*\*

\* 归还设备状态

\*/

@ApiModelProperty("归还设备状态显示")

private String returnDeviceStatusDisplay;

/\*\*

\* 附件Id

\*/

@ApiModelProperty("附件Id")

private String fileId;

/\*\*

\* 校准结果(1:合格;0:不合格)

\*/

@ApiModelProperty(value = "校准结果(1:合格;0:不合格)")

private Integer calibrationResult;

/\*\*

\* 校准结果(1:合格;0:不合格)

\*/

@ApiModelProperty("校准结果显示")

private String calibrationResultDisplay;

/\*\*

\* 文件名称

\*/

@ApiModelProperty("文件名称")

private String fileName;

/\*\*

\* 文件路径

\*/

@ApiModelProperty("文件路径")

private String filePath;

/\*\*

\* 上传时间

\*/

@ApiModelProperty("上传时间")

private String uploadTime;

/\*\*

\* 附件Id集合

\*/

@ApiModelProperty(value = "附件Id集合")

private List<String> fileIdList;

/\*\*

\* 附件集合

\*/

@ApiModelProperty(value = "附件集合")

private List<FileAttachVO> fileAttachVOList;

/\*\*

\* 备注

\*/

@ApiModelProperty("备注")

private String remarks;

}

/\*\*

\* 设备流转记录表

\*

\* @author wanggang

\* @date 2023/5/10 9:28

\*/

@Data

@ApiModel(value = "设备流转记录查询结果实体")

public class DeviceCirculationRecordVO {

/\*\*

\* ID

\*/

@ApiModelProperty("ID")

private String id;

/\*\*

\* 设备编号

\*/

@ApiModelProperty("设备编号")

private String deviceNumber;

/\*\*

\* 设备名称

\*/

@ApiModelProperty("设备名称")

private String deviceName;

/\*\*

\* 流转类型

\*/

@ApiModelProperty("流转类型")

private String circulationType;

/\*\*

\* 流转类型显示

\*/

@ApiModelProperty("流转类型显示")

private String circulationTypeDisplay;

/\*\*

\* 发出地点

\*/

@ApiModelProperty("发出地点")

private String issuePlace;

/\*\*

\* 流转日期

\*/

@ApiModelProperty("流转日期")

private Date circulationDate;

/\*\*

\* 接收人ID

\*/

@ApiModelProperty("接收人ID")

private String recipientId;

/\*\*

\* 接收人名

\*/

@ApiModelProperty("接收人名")

private String recipientName;

/\*\*

\* 接收地点

\*/

@ApiModelProperty("接收地点")

private String receivePlace;

/\*\*

\* 发出设备状态

\*/

@ApiModelProperty("发出设备状态")

private String issueDeviceSituation;

/\*\*

\* 发出设备状态

\*/

@ApiModelProperty("发出设备状态显示")

private String issueDeviceSituationDisplay;

/\*\*

\* 接收设备状态

\*/

@ApiModelProperty("接收设备状态")

private String receiveDeviceSituation;

/\*\*

\* 接收设备状态

\*/

@ApiModelProperty("接收设备状态显示")

private String receiveDeviceSituationDisplay;

/\*\*

\* 备注

\*/

@ApiModelProperty("备注")

private String remarks;

/\*\*

\* 设备信息ID

\*/

@ApiModelProperty("设备信息ID")

private String deviceInfoId;

/\*\*

\* 附件Id

\*/

@ApiModelProperty("附件Id")

private String fileId;

/\*\*

\* 文件名称

\*/

@ApiModelProperty("文件名称")

private String fileName;

/\*\*

\* 文件路径

\*/

@ApiModelProperty("文件路径")

private String filePath;

/\*\*

\* 上传时间

\*/

@ApiModelProperty("上传时间")

private String uploadTime;

/\*\*

\* 附件Id集合

\*/

@ApiModelProperty(value = "附件Id集合")

private List<String> fileIdList;

/\*\*

\* 附件集合

\*/

@ApiModelProperty(value = "附件集合")

private List<FileAttachVO> fileAttachVOList;

}

@Data

@ExcelIgnoreUnannotated

@EqualsAndHashCode(callSuper = false)

@ApiModel(description = "设备分类信息查询表（包含厂商供应商）")

public class DeviceClassifyInfoVO extends BaseEntity {

@ApiModelProperty(name = "id")

private String id;

@ApiModelProperty(name = "deviceClassifyNumber" , value = "分类编号")

@ExcelProperty("分类编号")

private String deviceClassifyNumber;

@ApiModelProperty(name = "deviceClassifyName" , value = "分类名称")

@ExcelProperty("分类名称(不允许重复)")

private String deviceClassifyName;

@ApiModelProperty(name = "deviceName" , value = "设备名称")

@ExcelProperty("设备名称")

private String deviceName;

@ApiModelProperty(name = "specifications" , value = "规格")

private String specifications;

@ApiModelProperty(name = "model" , value = "规格型号")

@ExcelProperty("规格/型号")

private String model;

@ApiModelProperty(name = "precision" , value = "精度")

@ExcelProperty("精度")

private String precision;

@ApiModelProperty(name = "measureRange" , value = "测量范围(量程)")

@ExcelProperty("测量范围(量程)")

private String measureRange;

@ApiModelProperty(name = "accuracyLevel" , value = "准确度等级")

@ExcelProperty("准确度等级")

private String accuracyLevel;

@ApiModelProperty(name = "uncertainty" , value = "不确定度")

@ExcelProperty("不确定度")

private String uncertainty;

@ApiModelProperty(value = "设备分类（1:A;2:B;3:C;4:S;）")

@ExcelProperty("设备分类")

private String deviceClassify;

@ApiModelProperty(value = "资产分类(1:固定资产;2:耗材;)")

@ExcelProperty("资产分类")

private String assetClassify;

@ApiModelProperty(value = "归属权(1:北域;2:总站;)")

private String ownership;

@ApiModelProperty(name = "deviceOriginalValue" , value = "设备原值(元)")

@ExcelProperty("设备原值(元)")

private BigDecimal deviceOriginalValue;

@ApiModelProperty(value = "厂商id")

private String productionId;

@ApiModelProperty(name = "productionName" , value = "生产厂家名称")

@ExcelProperty("生产厂家")

private String productionName;

@ApiModelProperty(value = "供货商ID")

private String supplierId;

@ApiModelProperty(name = "supplierName" , value = "供应商名称")

@ExcelProperty("供应商")

private String supplierName;

@ApiModelProperty(name = "supplierConcatName" , value = "供应商联系人")

private String supplierContactsName;

@ApiModelProperty(name = "suppliertelephone" , value = "供应商联系人电话")

private String supplierContactPhone;

@ApiModelProperty("备注")

private String remarks;

}

/\*\*

\* 设备管理表

\*

\* @author wanggang

\* @date 2023/5/8 9:46

\*/

@Data

@ApiModel(value = "设备管理查询结果实体")

@ExcelIgnoreUnannotated

public class DeviceInfoVO {

/\*\*

\* 设备编号

\*/

@ApiModelProperty("设备编号")

@ExcelProperty(value = "设备编号", order = 10)

private String deviceNumber;

//设备分类的字段--------------------

/\*\*

\* 设备分类

\*/

@ApiModelProperty("设备分类")

private String deviceClassify;

/\*\*

\* 设备分类名称

\*/

@ApiModelProperty("设备分类名称")

private String deviceClassifyName;

/\*\*

\* 设备名称

\*/

@ApiModelProperty("设备名称")

@ExcelProperty(value = "设备名称", order = 20)

private String deviceName;

/\*\*

\* 精度

\*/

@ApiModelProperty("精度")

private String precision;

/\*\*

\* 测量范围(量程)

\*/

@ApiModelProperty("测量范围(量程)")

private String measureRange;

/\*\*

\* 准确度等级

\*/

@ApiModelProperty("准确度等级")

private String accuracyLevel;

/\*\*

\* 不确定度

\*/

@ApiModelProperty("不确定度")

private String uncertainty;

/\*\*

\* 生产厂家ID（对应厂商ID）

\*/

@ApiModelProperty("生产厂家ID（对应厂商ID）")

private String productionId;

/\*\*

\* 生产厂家

\*/

@ApiModelProperty("生产厂家名")

@ExcelProperty(value = "生产厂家", order = 30)

private String productionName;

/\*\*

\* 规格/型号

\*/

@ApiModelProperty("规格/型号")

@ExcelProperty(value = "规格/型号", order = 40)

private String model;

/\*\*

\* 供货商ID（对应厂商ID）

\*/

@ApiModelProperty("供货商ID（对应厂商ID）")

private String supplierId;

/\*\*

\* 供货商名

\*/

@ApiModelProperty("供货商名")

private String supplierName;

/\*\*

\* 设备原值(元)

\*/

@ApiModelProperty("设备原值(元)")

private Double deviceOriginalValue;

/\*\*

\* 资产分类

\*/

@ApiModelProperty(value = "资产分类(1:固定资产;2:耗材;)")

private String assetClassify;

/\*\*

\* 设备归属

\*/

@ApiModelProperty(value = "设备归属权(1:北域;2:总站;)")

private String ownership;

/\*\*

\* 联系人

\*/

@ApiModelProperty("联系人")

private String supplierContactsName;

/\*\*

\* 联系电话

\*/

@ApiModelProperty("联系电话")

private String supplierContactPhone;

//设备信息字段--------------------

/\*\*

\* 设备分类ID

\*/

@ApiModelProperty("设备分类ID")

private String deviceClassifyInfoId;

/\*\*

\* 设备ID

\*/

@ApiModelProperty("设备ID")

private String id;

/\*\*

\* 设备分类编号

\*/

@ApiModelProperty("设备分类编号")

private String deviceClassifyNumber;

/\*\*

\* 使用状态（1:使用;0:停用;-1:损坏;）

\*/

@ApiModelProperty("使用状态（1:使用;0:停用;-1:损坏;）")

private String useStatus;

/\*\*

\* 使用状态显示（1:使用;0:停用;-1:损坏;）

\*/

@ApiModelProperty("使用状态显示")

@ExcelProperty(value = "使用状态", order = 50)

private String useStatusDisplay;

/\*\*

\* 设备状态

\*/

@ApiModelProperty("设备状态")

private String deviceStatus;

/\*\*

\* 设备状态显示

\*/

@ApiModelProperty("设备状态显示")

private String deviceStatusDisplay;

/\*\*

\* 流程状态

\*/

@ApiModelProperty("流程状态")

private String flowStatus;

/\*\*

\* 流程状态显示

\*/

@ApiModelProperty("流程状态显示")

private String flowStatusDisplay;

/\*\*

\* 部门ID

\*/

@ApiModelProperty("部门ID")

private String departmentId;

/\*\*

\* 部门名称

\*/

@ApiModelProperty("部门名称")

@ExcelProperty(value = "使用部门", order = 60)

private String departmentName;

/\*\*

\* 检定状态

\*/

@ApiModelProperty("检定状态")

private String verificationStatus;

/\*\*

\* 检定状态显示

\*/

@ApiModelProperty("检定状态显示")

@ExcelProperty(value = "检定状态", order = 70)

private String verificationStatusDisplay;

/\*\*

\* 检定时间

\*/

@ApiModelProperty("检定时间")

@ExcelProperty(value = "检定时间", order = 80)

private Date verificationDate;

/\*\*

\* 检定单位

\*/

@ApiModelProperty("检定单位")

@ExcelProperty(value = "检定单位", order = 90)

private String verificationCompany;

/\*\*

\* 证书到期

\*/

@ApiModelProperty("证书到期")

@ExcelProperty(value = "证书到期", order = 100)

private Date validEnd;

/\*\*

\* 购置日期

\*/

@ApiModelProperty("购置日期")

private Date purchaseTime;

/\*\*

\* 出厂编号

\*/

@ApiModelProperty("出厂编号")

private String exworksNumber;

/\*\*

\* 检定周期（1:3个月;2:6个月;）

\*/

@ApiModelProperty("检定周期（1:3个月;2:6个月;）")

private String verificationCycle;

/\*\*

\* 设备存放货架信息ID

\*/

@ApiModelProperty(value = "设备存放货架信息ID")

private String deviceShelvesInfoId;

/\*\*

\* 存放编号(位置编号)

\*/

@ApiModelProperty(value = "位置编号")

private String storageCode;

/\*\*

\* 设备存放室ID

\*/

@ApiModelProperty(value = "设备存放室ID")

private String deviceRoomId;

/\*\*

\* 存放室名称

\*/

@ApiModelProperty(value = " 存放室名称")

@ExcelProperty(value = "存放地点", order = 110)

private String roomName;

/\*\*

\* 备注

\*/

@ApiModelProperty("备注")

private String remarks;

/\*\*

\* 资产来源

\*/

@ApiModelProperty("资产来源")

private String assetSource;

/\*\*

\* 责任人

\*/

@ApiModelProperty("责任人")

private String personLiable;

/\*\*

\* 责任人名

\*/

@ApiModelProperty("责任人名")

@ExcelProperty(value = "使用责任人", order = 120)

private String personLiableName;

/\*\*

\* 是否自动采集(0否;1是)

\*/

@ApiModelProperty("是否自动采集(0否;1是)")

private Integer automaticCollection;

/\*\*

\* 出厂日期

\*/

@ApiModelProperty("出厂日期")

private String exworksTime;

/\*\*

\* 设备编号(交通行业)

\*/

@ApiModelProperty("设备编号(交通行业)")

private String trafficDeviceNumber;

/\*\*

\* 设备名称(交通行业)

\*/

@ApiModelProperty("设备名称(交通行业)")

private String trafficDeviceName;

}

@Data

@EqualsAndHashCode(callSuper = false)

public class DevicePlaceTreeVO extends DevicePlaceVO {

private List<DeviceRoomVO> deviceRoomVOS = Collections.emptyList();

}

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(description = "场所视图类")

public class DevicePlaceVO {

@ApiModelProperty(value = "主键")

private String id;

@ApiModelProperty(value = " 组织id")

private String orgId;

@ApiModelProperty(value = " 场所名称(二级组织名称)")

private String placeName;

@ApiModelProperty(value = " 场所编码(一个字母)")

private String placeCode;

@ApiModelProperty(value = " 场所位置及描述")

private String placeDesc;

@ApiModelProperty(value = " 备注")

private String remarks;

@ApiModelProperty(value = " 删除标志（1:有效;0:删除;默认1）")

private Integer delFlag;

@ApiModelProperty(value = " 修改人")

private String updateBy;

@ApiModelProperty(value = " 修改人名")

private String updateName;

@ApiModelProperty(value = " 修改时间")

private LocalDateTime updateTime;

}

/\*\*

\* 设备保养维修记录表

\*

\* @author wanggang

\* @date 2023/5/10 10:33

\* @TableName device\_repair\_maintenance\_record

\*/

@Data

@ApiModel(value = "设备保养维修记录查询结果实体")

public class DeviceRepairMaintenanceRecordVO {

/\*\*

\* ID

\*/

@ApiModelProperty("ID")

private String id;

/\*\*

\* 设备编号

\*/

@ApiModelProperty("设备编号")

private String deviceNumber;

/\*\*

\* 设备名称

\*/

@ApiModelProperty("设备名称")

private String deviceName;

/\*\*

\* 使用部门ID

\*/

@ApiModelProperty("使用部门ID")

private String useDepartmentId;

/\*\*

\* 使用部门名

\*/

@ApiModelProperty("使用部门名")

private String useDepartmentName;

/\*\*

\* 负责人ID

\*/

@ApiModelProperty("负责人ID")

private String headId;

/\*\*

\* 负责人名

\*/

@ApiModelProperty("负责人名")

private String headName;

/\*\*

\* 参与人员

\*/

@ApiModelProperty("参与人员")

private String involvedStaff;

/\*\*

\* 参与人员名

\*/

@ApiModelProperty("参与人员名")

private String involvedStaffName;

/\*\*

\* 参与人员集

\*/

@ApiModelProperty("参与人员集")

private List<SysUserVO> involvedStaffList;

/\*\*

\* 分类(1:保养;2:维修)

\*/

@ApiModelProperty("分类(1:保养;2:维修)")

private String classify;

/\*\*

\* 发生时间

\*/

@ApiModelProperty("发生时间")

private Date happenTime;

/\*\*

\* 情况说明

\*/

@ApiModelProperty("情况说明")

private String situationDescription;

/\*\*

\* 校准结果(1:合格;0:不合格)

\*/

@ApiModelProperty("校准结果(1:合格;0:不合格)")

private Integer calibrationResult;

/\*\*

\* 校准结果(1:合格;0:不合格)

\*/

@ApiModelProperty("校准结果显示")

private String calibrationResultDisplay;

/\*\*

\* 运行使用状态

\*/

@ApiModelProperty("运行使用状态")

private String runUseStatus;

/\*\*

\* 运行使用状态

\*/

@ApiModelProperty("运行使用状态显示")

private String runUseStatusDisplay;

/\*\*

\* 附件Id

\*/

@ApiModelProperty("附件Id")

private String fileId;

/\*\*

\* 文件名称

\*/

@ApiModelProperty("文件名称")

private String fileName;

/\*\*

\* 文件路径

\*/

@ApiModelProperty("文件路径")

private String filePath;

/\*\*

\* 上传时间

\*/

@ApiModelProperty("上传时间")

private String uploadTime;

/\*\*

\* 附件Id集合

\*/

@ApiModelProperty(value = "附件Id集合")

private List<String> fileIdList;

/\*\*

\* 附件集合

\*/

@ApiModelProperty(value = "附件集合")

private List<FileAttachVO> fileAttachVOList;

/\*\*

\* 备注

\*/

@ApiModelProperty("备注")

private String remarks;

/\*\*

\* 设备信息ID

\*/

@ApiModelProperty("设备信息ID")

private String deviceInfoId;

}

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(description = "设备存放室视图类")

public class DeviceRoomVO {

@ApiModelProperty(value = "主键")

private String id;

@ApiModelProperty(value = "场所id")

private String placeId;

@ApiModelProperty(value = " 场所名称")

private String placeName;

@ApiModelProperty(value = " 场所编码")

private String placeCode;

@ApiModelProperty(value = " 存放室名称")

private String roomName;

@ApiModelProperty(value = " 房间编号(场所编号+2位数字)")

private String roomCode;

@ApiModelProperty(value = " 描述")

private String roomDesc;

@ApiModelProperty(value = " 删除标志（1:有效;0:删除;默认1）")

private Integer delFlag;

@ApiModelProperty(value = " 修改人")

private String updateBy;

@ApiModelProperty(value = " 修改人名")

private String updateName;

@ApiModelProperty(value = " 修改时间")

private LocalDateTime updateTime;

}

/\*\*

\* 设备存放货架信息表

\*/

@Data

@ExcelIgnoreUnannotated

@ApiModel(value = "设备存放货架信息表")

public class DeviceShelvesInfoVO {

@ApiModelProperty(value = "id")

private String id;

@ApiModelProperty(value = "租户ID")

private String tenantId;

@ApiModelProperty(value = "设备信息ID")

private String deviceInfoId;

@ApiModelProperty(value = "设备编号")

private String deviceNumber;

@ApiModelProperty(value = "存放编号")

@ExcelProperty(value = "存放编号")

private String storageCode;

@ApiModelProperty(value = "货架名称")

@ExcelProperty(value = "货架名称")

private String shelvesName;

@ApiModelProperty(value = "货架编号")

@ExcelProperty(value = "货架编号")

private String shelvesCode;

@ApiModelProperty(value = "货架层号")

@ExcelProperty(value = "货架层号")

private String layerCode;

@ApiModelProperty(value = "位置序号")

@ExcelProperty(value = "位置序号")

private String positionCode;

@ApiModelProperty(value = "位置描述")

@ExcelProperty(value = "位置描述")

private String positionDesc;

@ApiModelProperty(value = "存放室id")

private String roomId;

@ApiModelProperty(value = "存放室名称")

private String roomName;

@ApiModelProperty(value = "房间编号(场所编号+2位数字)")

private String roomCode;

@ApiModelProperty(value = "场所id")

private String placeId;

@ApiModelProperty(value = "场所名称(二级组织名称)")

private String placeName;

@ApiModelProperty(value = "场所编码(一个字母)")

private String placeCode;

@ApiModelProperty(value = "购买时间")

private LocalDateTime purchaseTime;

@ApiModelProperty(value = "备注")

@ExcelProperty(value = "备注")

private String remarks;

@ApiModelProperty(value = "删除标志（1:有效;0:删除;默认1）")

private Integer delFlag;

}

/\*\*

\* 设备检定记录表

\*

\* @author wanggang

\* @date 2023/5/10 14:22

\* @TableName device\_verification\_record

\*/

@Data

@ApiModel(value = "设备检定记录查询结果实体")

public class DeviceVerificationRecordVO {

/\*\*

\* 检定记录ID

\*/

@ApiModelProperty("检定记录ID")

private String id;

/\*\*

\* 设备信息ID

\*/

@ApiModelProperty("设备信息ID")

private String deviceInfoId;

/\*\*

\* 设备编号

\*/

@ApiModelProperty("设备编号")

private String deviceNumber;

/\*\*

\* 设备名称

\*/

@ApiModelProperty("设备名称")

private String deviceName;

/\*\*

\* 证书编号

\*/

@ApiModelProperty("证书编号")

private String certificateNumber;

/\*\*

\* 证书类型

\*/

@ApiModelProperty("证书类型")

private String certificateClassification;

/\*\*

\* 证书类型

\*/

@ApiModelProperty("证书类型显示")

private String certificateClassificationDisplay;

/\*\*

\* 检定单位

\*/

@ApiModelProperty("检定单位")

private String verificationCompany;

/\*\*

\* 检定单位

\*/

@ApiModelProperty("检定单位名")

private String verificationCompanyName;

/\*\*

\* 检定日期

\*/

@ApiModelProperty("检定日期")

private Date verificationDate;

/\*\*

\* 有效开始日期

\*/

@ApiModelProperty("有效开始日期")

private Date validStart;

/\*\*

\* 有效结束日期

\*/

@ApiModelProperty("有效结束日期")

private Date validEnd;

/\*\*

\* 检定结果(1:合格;0:不合格)

\*/

@ApiModelProperty("检定结果(1:合格;0:不合格)")

private Integer verificationResult;

/\*\*

\* 检定结果(1:合格;0:不合格)

\*/

@ApiModelProperty("检定结果显示")

private String verificationResultDisplay;

/\*\*

\* 备注

\*/

@ApiModelProperty("备注")

private String remarks;

/\*\*

\* 文件名称

\*/

@ApiModelProperty("文件名称")

private String fileName;

/\*\*

\* 文件路径

\*/

@ApiModelProperty("文件路径")

private String filePath;

/\*\*

\* 上传时间

\*/

@ApiModelProperty("上传时间")

private String uploadTime;

/\*\*

\* 附件Id集合

\*/

@ApiModelProperty(value = "附件Id集合")

private List<String> fileIdList;

/\*\*

\* 附件集合

\*/

@ApiModelProperty(value = "附件集合")

private List<FileAttachVO> fileAttachVOList;

/\*\*

\* 附件Id

\*/

@ApiModelProperty("附件Id")

private String fileId;

/\*\*

\* 检定方式

\*/

@ApiModelProperty("检定方式")

private String verificationStyle;

/\*\*

\* 检定方式显示

\*/

@ApiModelProperty("检定方式显示")

private String verificationStyleDisplay;

}

/\*\*

\* 设备运行记录表

\*

\* @author wanggang

\* @date 2023/5/9 12:00

\*/

@Data

@ApiModel(value = "设备运行记录查询结果实体")

public class DeviceWorkingRecordVO {

/\*\*

\* ID

\*/

@ApiModelProperty("ID")

private String id;

/\*\*

\* 设备编号

\*/

@ApiModelProperty("设备编号")

private String deviceNumber;

/\*\*

\* 设备名称

\*/

@ApiModelProperty("设备名称")

private String deviceName;

/\*\*

\* 设备信息ID

\*/

@ApiModelProperty("设备信息ID")

private String deviceInfoId;

/\*\*

\* 报告编号

\*/

@ApiModelProperty("报告编号")

private String reportNumber;

/\*\*

\* 运行开始时间

\*/

@ApiModelProperty("运行开始时间")

private Date workingStartTime;

/\*\*

\* 运行结束时间

\*/

@ApiModelProperty("运行结束时间")

private Date workingEndTime;

/\*\*

\* 使用前状态（1:不正常;2:正常;）

\*/

@ApiModelProperty("使用前状态（1:不正常;2:正常;）")

private String useFrontStatus;

/\*\*

\* 使用前状态（1:不正常;2:正常;）

\*/

@ApiModelProperty("使用前状态显示")

private String useFrontStatusDisplay;

/\*\*

\* 使用后状态（1:正常;2:异常;）

\*/

@ApiModelProperty("使用后状态（1:正常;2:异常;）")

private String useEndStatus;

/\*\*

\* 使用后状态（1:正常;2:异常;）

\*/

@ApiModelProperty("使用后状态显示")

private String useEndStatusDisplay;

/\*\*

\* 检测人员ID

\*/

@ApiModelProperty("检测人员ID")

private String detectionStaffId;

/\*\*

\* 检测人员名

\*/

@ApiModelProperty("检测人员名")

private String detectionStaffName;

/\*\*

\* 异常状况描述

\*/

@ApiModelProperty("异常状况描述")

private String exceptionSituation;

/\*\*

\* 审核人员ID

\*/

@ApiModelProperty("审核人员ID")

private String reviewPerson;

/\*\*

\* 审核人员名

\*/

@ApiModelProperty("审核人员名")

private String reviewPersonName;

/\*\*

\* 审核意见

\*/

@ApiModelProperty("审核意见")

private String reviewComments;

/\*\*

\* 创建时间

\*/

@ApiModelProperty("创建时间")

private Date createTime;

/\*\*

\* 备注

\*/

@ApiModelProperty("备注")

private String remarks;

}

@Data

@ApiModel(value = "厂商联系人信息表")

public class SupplierContactsInfoVO {

@ApiModelProperty(value = "主键")

private String id;

@ApiModelProperty(value = "厂商ID", required = true, example = "1")

@NotBlank(message = "厂商ID必填")

private String supplierId;

@ApiModelProperty(value = "姓名", required = true, example = "李四")

@NotBlank(message = "联系人姓名必填")

@Length(max = 12, message = "联系人姓名不能超过12个字符")

@ExcelProperty(value = "姓名")

private String contactName;

@ApiModelProperty(value = "别名")

@Length(max = 12, message = "设联系人别名不能超过12个字符")

@ExcelProperty(value = "别名")

private String alias;

@ApiModelProperty(value = "职位", required = true, example = "经理")

@NotBlank(message = "职位必填")

@Length(max = 12, message = "职位不能超过12个字符")

@ExcelProperty(value = "职位")

private String position;

@ApiModelProperty(value = "部门名称")

@Length(max = 20, message = "部门名称不能超过20个字符")

@ExcelProperty(value = "部门名称")

private String departmentName;

@ApiModelProperty(value = "性别（0:未知;1:女;2:男;）")

@ExcelProperty(value = "性别")

private String gender;

@ApiModelProperty(value = "微信")

@Length(max = 20, message = "微信不能超过20个字符")

@ExcelProperty(value = "微信")

private String wechat;

@ApiModelProperty(value = "企业邮箱", required = true, example = "firefox@email.com")

@Length(max = 32, message = "企业邮箱不能超过32个字符")

@NotBlank(message = "企业邮箱必填")

@ExcelProperty(value = "企业邮箱")

@Email

private String mailbox;

@ApiModelProperty(value = "qq")

@Length(max = 20, message = "QQ不能超过20个字符")

@ExcelProperty(value = "qq")

private String qq;

@ApiModelProperty(value = "住址")

@Length(max = 64, message = "住址不能超过64个字符")

@ExcelProperty(value = "住址")

private String address;

@ApiModelProperty(value = "联系人电话", required = true, example = "11378248711")

@NotBlank(message = "电话号码必填")

@Length(max = 20, message = "电话号码不能超过20个字符")

@ExcelProperty(value = "电话")

private String telephone;

@ApiModelProperty(value = "备注")

@Length(max = 100, message = "备注不能超过100个字符")

@ExcelProperty(value = "备注")

private String remarks;

public SupplierContactsInfoDTO transferDto() {

SupplierContactsInfoDTO dto = new SupplierContactsInfoDTO();

dto.setId(this.id);

dto.setName(this.contactName);

dto.setAlias(this.alias);

dto.setPosition(this.position);

dto.setDepartmentName(this.departmentName);

dto.setGender(this.gender);

dto.setWechat(this.wechat);

dto.setMailbox(this.mailbox);

dto.setQq(this.qq);

dto.setAddress(this.address);

dto.setTelephone(this.telephone);

dto.setRemarks(this.remarks);

return dto;

}

}

/\*\*

\* 厂商信息表

\*

\* @TableName supplier\_info

\*/

@Data

@ApiModel(value = "厂商信息表")

@ExcelIgnoreUnannotated

@EqualsAndHashCode(callSuper = false)

public class SupplierInfoVO extends SupplierContactsInfoVO {

@ApiModelProperty(value = "主键")

private String id;

@ApiModelProperty(value = "名称")

@ExcelProperty(value = "名称")

private String supplierName;

@ApiModelProperty(value = "简称")

@ExcelProperty(value = "简称")

private String abbreviation;

@ApiModelProperty(value = "地址")

@ExcelProperty(value = "地址")

private String address;

@ApiModelProperty(value = "行业")

@ExcelProperty(value = "行业")

private String industry;

@ApiModelProperty(value = "类型：1厂家(默认),2供应商,3检定", required = true, example = "1")

@ExcelProperty(value = "类型")

private String type;

@ApiModelProperty(value = "企业电话")

@ExcelProperty(value = "企业电话")

private String contactTelephone;

@ApiModelProperty(value = "邮箱")

private String mailbox;

@ApiModelProperty(value = "微信")

private String wechat;

@ApiModelProperty(value = "QQ")

private String qq;

@ApiModelProperty(value = "厂商联系人ID")

private String supplierContactId;

@ApiModelProperty(value = "简介")

private String profile;

@ApiModelProperty(value = "备注")

private String remarks;

@ApiModelProperty(value = "创建人")

private String createBy;

@ApiModelProperty(value = "创建时间")

private LocalDateTime createTime;

@ApiModelProperty(value = "修改人")

private String updateBy;

@ApiModelProperty(value = "修改时间")

private LocalDateTime updateTime;

public SupplierInfoDTO transferSupplierInfoDto() {

SupplierInfoDTO dto = new SupplierInfoDTO();

dto.setId(this.id);

dto.setName(this.supplierName);

dto.setAbbreviation(this.abbreviation);

dto.setAddress(this.address);

dto.setIndustry(this.industry);

dto.setType(this.type);

dto.setContactTelephone(this.contactTelephone);

dto.setMailbox(this.mailbox);

dto.setSupplierContactId(this.supplierContactId);

dto.setProfile(this.profile);

dto.setRemarks(this.remarks);

return dto;

}

/\*\*

\* 流程基础流水信息表

\* @TableName flow\_base\_info

\*/

@EqualsAndHashCode(callSuper = true)

@TableName(value ="flow\_base\_info")

@Data

public class FlowBaseInfo extends BaseEntity implements Serializable {

/\*\*

\*

\*/

@TableId

private String id;

@ApiModelProperty("流程实例ID")

private String processId;

/\*\*

\* 流程引擎key，枚举类

\*/

@ApiModelProperty("流程引擎key，枚举类")

private ProcessKey processKey;

/\*\*

\* 流程名称

\*/

@ApiModelProperty("流程名称")

private String processName;

/\*\*

\* 流程状态，枚举类

\*/

@ApiModelProperty("流程状态，枚举类")

private ProcessState state;

/\*\*

\* 结束时间

\*/

@ApiModelProperty("结束时间")

private Date endTime;

@ApiModelProperty("发起人组织ID")

private String sysOrgId;//发起人组织ID

@ApiModelProperty("发起人组织名称")

private String sysOrgFullName;//发起人组织ID

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 1L;

}

/\*\*

\* 采购申请明细记录

\* @author wanggang

\* @date 2023/6/12 16:09

\* @TableName flow\_device\_purchase

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@TableName(value ="flow\_device\_purchase")

public class FlowDevicePurchase extends BaseEntity implements Serializable {

@Serial

@TableField(exist = false)

private static final long serialVersionUID = -234499127230943638L;

/\*\*

\* ID

\*/

@TableId(value = "id")

private String id;

/\*\*

\* 租户ID

\*/

@TableField(value = "tenant\_id")

private String tenantId;

/\*\*

\* 资产分类

\*/

@TableField(value = "asset\_classify")

private String assetClassify;

/\*\*

\* 流程记录表ID

\*/

@TableField(value = "process\_record\_id")

private String processRecordId;

/\*\*

\* 设备分类ID

\*/

@TableField(value = "device\_classify\_info\_id")

private String deviceClassifyInfoId;

/\*\*

\* 物品类别

\*/

@TableField(value = "Item\_category")

private String itemCategory;

/\*\*

\* 采购类别

\*/

@TableField(value = "purchase\_category")

private String purchaseCategory;

/\*\*

\* 申请部门

\*/

@TableField(value = "apply\_department")

private String applyDepartment;

/\*\*

\* 单位

\*/

@TableField(value = "unit")

private String unit;

/\*\*

\* 数量

\*/

@TableField(value = "device\_count")

private Integer deviceCount;

/\*\*

\* 单价

\*/

@TableField(value = "unit\_price")

private BigDecimal unitPrice;

/\*\*

\* 总价

\*/

@TableField(value = "total\_price")

private BigDecimal totalPrice;

/\*\*

\* 备注

\*/

@TableField(value = "remarks")

private String remarks;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

@TableLogic(value = "1", delval = "0")

@TableField(value = "del\_flag")

private Integer delFlag;

/\*\*

\* 采购申请流水记录

\*

\* @author wanggang

\* @date 2023/6/12 16:10

\* @TableName flow\_device\_purchase\_record

\*/

@Data

@EqualsAndHashCode(callSuper = true)

@TableName(value = "flow\_device\_purchase\_record")

public class FlowDevicePurchaseRecord extends BaseEntity implements Serializable {

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 3245513430808645307L;

/\*\*

\* ID

\*/

@TableId(value = "id")

private String id;

/\*\*

\* 租户ID

\*/

@TableField(value = "tenant\_id")

private String tenantId;

/\*\*

\* 编号

\*/

@TableField(value = "number")

private String number;

/\*\*

\* 流程ID

\*/

@TableField(value = "process\_id")

private String processId;

/\*\*

\* 流程key

\*/

@TableField(value = "process\_key")

private String processKey;

/\*\*

\* 相关资料

\*/

@TableField(value = "related\_information")

private String relatedInformation;

/\*\*

\* 申请人

\*/

@TableField(value = "apply\_person")

private String applyPerson;

/\*\*

\* 申请理由

\*/

@TableField(value = "apply\_reason")

private String applyReason;

/\*\*

\* 申请时间

\*/

@TableField(value = "apply\_time")

private Date applyTime;

/\*\*

\* 审批人

\*/

@TableField(value = "approval\_person")

private String approvalPerson;

/\*\*

\* 审批人签名

\*/

@TableField(value = "approval\_sign")

private String approvalSign;

/\*\*

\* 审核意见

\*/

@TableField(value = "review\_comments")

private String reviewComments;

/\*\*

\* 申请状态

\*/

@TableField(value = "apply\_status")

private ApplyState applyStatus;

/\*\*

\* 备注

\*/

@TableField(value = "remarks")

private String remarks;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

@TableLogic(value = "1", delval = "0")

@TableField(value = "del\_flag")

private Integer delFlag;

/\*\*

\* 检定校验申请表

\*

\* @author wanggang

\* @date 2023/6/1 16:30

\* @TableName flow\_device\_verification

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@TableName(value = "flow\_device\_verification")

public class FlowDeviceVerification extends BaseEntity implements Serializable {

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 1100405375839450623L;

/\*\*

\* 检定校准申请ID

\*/

@TableId(value = "id")

private String id;

/\*\*

\* 租户ID

\*/

@TableField(value = "tenant\_id")

private String tenantId;

/\*\*

\* 流程记录ID

\*/

@TableField(value = "process\_record\_id")

private String processRecordId;

/\*\*

\* 设备信息ID

\*/

@TableField(value = "device\_info\_id")

private String deviceInfoId;

/\*\*

\* 有效期

\*/

@TableField(value = "validity\_period")

private String validityPeriod;

/\*\*

\* 申请检定时间

\*/

@TableField(value = "application\_verification\_time")

private Date applicationVerificationTime;

/\*\*

\* 检定方式

\*/

@TableField(value = "verification\_style")

private String verificationStyle;

/\*\*

\* 检定地点

\*/

@TableField(value = "verification\_place")

private String verificationPlace;

/\*\*

\* 附送配件名称及数量

\*/

@TableField(value = "parts\_name\_count")

private String partsNameCount;

/\*\*

\* 检定所需工作日

\*/

@TableField(value = "verification\_need\_weekday")

private String verificationNeedWeekday;

/\*\*

\* 技术要求

\*/

@TableField(value = "technical\_require")

private String technicalRequire;

/\*\*

\* 备注

\*/

@TableField(value = "remarks")

private String remarks;

/\*\*

\* 申请状态

\*/

@TableField(value = "apply\_status")

private ApplyState applyStatus;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

@TableLogic(value = "1", delval = "0")

@TableField(value = "del\_flag")

private Integer delFlag;

/\*\*

\* 检定校准申请流水记录

\*

\* @author wanggang

\* @date 2023/6/5 17:05

\* @TableName flow\_device\_verification\_record

\*/

@Data

@EqualsAndHashCode(callSuper = true)

@TableName(value = "flow\_device\_verification\_record")

public class FlowDeviceVerificationRecord extends BaseEntity implements Serializable {

@Serial

@TableField(exist = false)

private static final long serialVersionUID = -6120466429902627599L;

/\*\*

\* 检定校准申请ID

\*/

@TableId(value = "id")

private String id;

/\*\*

\* 租户ID

\*/

@TableField(value = "tenant\_id")

private String tenantId;

/\*\*

\* 流程ID

\*/

@TableField(value = "process\_id")

private String processId;

/\*\*

\* 流程key

\*/

@TableField(value = "process\_key")

private String processKey;

/\*\*

\* 申请人

\*/

@TableField(value = "apply\_person")

private String applyPerson;

/\*\*

\* 申请时间

\*/

@TableField(value = "apply\_time")

private Date applyTime;

/\*\*

\* 审批人

\*/

@TableField(value = "approval\_person")

private String approvalPerson;

/\*\*

\* 审批人签名

\*/

@TableField(value = "approval\_sign")

private String approvalSign;

/\*\*

\* 审核意见

\*/

@TableField(value = "review\_comments")

private String reviewComments;

/\*\*

\* 流水号

\*/

@TableField(value = "serial\_number")

private String serialNumber;

/\*\*

\* 历史流水记录ID

\*/

@TableField(value = "history\_id")

private String historyId;

/\*\*

\* 备注

\*/

@TableField(value = "remarks")

private String remarks;

/\*\*

\* 删除标志（1:有效;0:删除;默认1）

\*/

@TableLogic(value = "1", delval = "0")

@TableField(value = "del\_flag")

private Integer delFlag;

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/6/2 10:59

\*/

@Data

public class CompleteTaskDTO {

@NotBlank(message = "流程实例ID不能为空")

private String processId;

private String comment;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/6/5 10:17

\*/

@Data

public class DeleteProcessDTO {

@ApiModelProperty("流程ID")

@NotBlank(message = "流程实例ID不能为空")

private String processId;

@ApiModelProperty("废弃原因")

private String deleteReason;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/5/31 17:27

\*/

@Data

public class FlowBaseInfoAddDTO {

@ApiModelProperty("流程KEY")

@TableField(exist = false)

@NotBlank(message = "流程key必须输入")

private String processKey;

@TableField(exist = false)

@ApiModelProperty("流程名称")

@NotBlank(message = "流程名称不能为空")

private String processName;

private String processId;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/6/2 15:36

\*/

@Data

public class FlowBaseInfoUpdateDTO {

@NotBlank(message = "流程ID不能为空")

private String processId;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/5/31 17:32

\*/

@EqualsAndHashCode(callSuper = true)

@Data

public class FlowDeviceCheckDTO extends FlowBaseInfoAddDTO {

private String deviceId;

}

/\*\*

\* 采购申请明细记录

\*

\* @author wanggang

\* @date 2023/6/12 16:24

\*/

@Data

@ApiModel(value = "采购申请明细记录")

public class FlowDevicePurchaseDTO {

/\*\*

\* ID

\*/

@NotBlank(message = "[ID]不能为空")

@ApiModelProperty("ID")

@Length(max = 64, message = "编码长度不能超过64")

private String id;

/\*\*

\* 资产分类

\*/

@ApiModelProperty(value = "资产分类", required = true, example = "1")

private String assetClassify;

/\*\*

\* 流程记录表ID

\*/

@NotBlank(message = "[流程记录表ID]不能为空")

@Length(max = 255, message = "编码长度不能超过255")

@ApiModelProperty(value = "流程记录表ID", hidden = true)

private String processRecordId;

/\*\*

\* 设备分类ID

\*/

@NotBlank(message = "[设备分类ID]不能为空")

@Length(max = 255, message = "编码长度不能超过64")

@ApiModelProperty(value = "设备分类ID", required = true, example = "1666654384268062722")

private String deviceClassifyInfoId;

/\*\*

\* 物品类别

\*/

@ApiModelProperty(value = "物品类别")

private String itemCategory;

/\*\*

\* 采购类别

\*/

@ApiModelProperty(value = "采购类别")

private String purchaseCategory;

/\*\*

\* 申请部门

\*/

@Length(max = 255, message = "编码长度不能超过255")

@ApiModelProperty(value = "申请部门", required = true, example = "1666654384268062722")

private String applyDepartment;

/\*\*

\* 单位

\*/

@Length(max = 255, message = "编码长度不能超过255")

@ApiModelProperty(value = "单位", required = true, example = "台")

private String unit;

/\*\*

\* 数量

\*/

@ApiModelProperty(value = "数量", required = true, example = "100")

private Integer deviceCount;

/\*\*

\* 单价

\*/

@ApiModelProperty(value = "单价", required = true, example = "10010.25")

private BigDecimal unitPrice;

/\*\*

\* 总价

\*/

@ApiModelProperty(value = "总价", required = true, example = "50010.25")

private BigDecimal totalPrice;

/\*\*

\* 备注

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "备注", hidden = true)

private String remarks;

}

/\*\*

\* 采购申请流水记录

\*

\* @author wanggang

\* @date 2023/6/12 16:24

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(value = "采购申请明细记录")

public class FlowDevicePurchaseRecordDTO extends FlowBaseInfoAddDTO {

/\*\*

\* ID

\*/

@NotBlank(message = "[ID]不能为空")

@ApiModelProperty("ID")

@Length(max = 64, message = "编码长度不能超过64")

private String id;

/\*\*

\* 编号

\*/

@Length(max = 255, message = "编码长度不能超过255")

@ApiModelProperty(value = "编号", example = "编号")

private String number;

/\*\*

\* 相关资料

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "相关资料", example = "相关资料")

private String relatedInformation;

/\*\*

\* 申请人

\*/

@NotBlank(message = "[申请人]不能为空")

@Length(max = 255, message = "编码长度不能超过255")

@ApiModelProperty(value = "申请人", required = true, example = "1666633897964929026")

private String applyPerson;

/\*\*

\* 申请理由

\*/

@Length(max = 255, message = "编码长度不能超过255")

@ApiModelProperty(value = "申请理由", example = "申请理由")

private String applyReason;

/\*\*

\* 申请时间

\*/

@NotNull(message = "[申请时间]不能为空")

@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")

@ApiModelProperty(value = "申请时间", required = true, example = "2023-06-05 17:15:00")

private Date applyTime;

/\*\*

\* 审批人

\*/

@Length(max = 255, message = "编码长度不能超过255")

@ApiModelProperty(value = "审批人", example = "1666633897964929026")

private String approvalPerson;

/\*\*

\* 审批人签名

\*/

@ApiModelProperty(value = "审批人签名", hidden = true)

private String approvalSign;

/\*\*

\* 审核意见

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "审核意见", hidden = true)

private String reviewComments;

/\*\*

\* 申请状态

\*/

@NotNull(message = "[申请状态]不能为空")

@ApiModelProperty(value = "申请状态", hidden = true)

private ApplyState applyStatus;

/\*\*

\* 备注

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "备注", hidden = true)

private String remarks;

/\*\*

\* 设备申请内容

\*/

@ApiModelProperty(value = "设备申请内容", required = true, example = "[{},{}]")

private List<FlowDevicePurchaseDTO> dtoList;

}

/\*\*

\* 采购申请流水记录

\*

\* @author wanggang

\* @date 2023/6/12 16:24

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(value = "采购申请明细记录(暂存用)")

public class FlowDevicePurchaseRecordStorageDTO {

/\*\*

\* ID

\*/

@NotBlank(message = "[ID]不能为空")

@ApiModelProperty("ID")

@Length(max = 64, message = "编码长度不能超过64")

private String id;

/\*\*

\* 编号

\*/

@Length(max = 255, message = "编码长度不能超过255")

@ApiModelProperty(value = "编号", example = "编号")

private String number;

/\*\*

\* 相关资料

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "相关资料", example = "相关资料")

private String relatedInformation;

/\*\*

\* 申请人

\*/

@NotBlank(message = "[申请人]不能为空")

@Length(max = 255, message = "编码长度不能超过255")

@ApiModelProperty(value = "申请人", required = true, example = "1666633897964929026")

private String applyPerson;

/\*\*

\* 申请理由

\*/

@Length(max = 255, message = "编码长度不能超过255")

@ApiModelProperty(value = "申请理由", example = "申请理由")

private String applyReason;

/\*\*

\* 申请时间

\*/

@NotNull(message = "[申请时间]不能为空")

@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")

@ApiModelProperty(value = "申请时间", required = true, example = "2023-06-05 17:15:00")

private Date applyTime;

/\*\*

\* 审批人

\*/

@Length(max = 255, message = "编码长度不能超过255")

@ApiModelProperty(value = "审批人", example = "1666633897964929026")

private String approvalPerson;

/\*\*

\* 审批人签名

\*/

@ApiModelProperty(value = "审批人签名", hidden = true)

private String approvalSign;

/\*\*

\* 审核意见

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "审核意见", hidden = true)

private String reviewComments;

/\*\*

\* 申请状态

\*/

@NotNull(message = "[申请状态]不能为空")

@ApiModelProperty(value = "申请状态", hidden = true)

private ApplyState applyStatus;

/\*\*

\* 备注

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "备注", hidden = true)

private String remarks;

/\*\*

\* 设备申请内容

\*/

@ApiModelProperty(value = "设备申请内容", required = true, example = "[{},{}]")

private List<FlowDevicePurchaseDTO> dtoList;

}

/\*\*

\* 检定校验申请新增实体

\*

\* @author wanggang

\* @date 2023/6/1 16:32

\*/

@Data

@ApiModel(value = "检定校验申请新增实体")

public class FlowDeviceVerificationDTO {

/\*\*

\* 检定校准申请ID

\*/

@ApiModelProperty(value = "检定校准申请ID")

private String id;

/\*\*

\* 设备信息ID

\*/

@NotBlank(message = "[设备信息ID]不能为空")

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "设备信息ID", required = true, example = "1660563788953858049")

private String deviceInfoId;

/\*\*

\* 流程记录ID

\*/

@NotBlank(message = "[流程ID]不能为空")

@Length(max = 255, message = "编码长度不能超过255")

@ApiModelProperty(value = "流程ID", hidden = true)

private String processRecordId;

/\*\*

\* 设备名称

\*/

@ApiModelProperty(value = "设备名称", hidden = true)

private String deviceName;

/\*\*

\* 规格/型号

\*/

@ApiModelProperty(value = "规格/型号", hidden = true)

private String model;

/\*\*

\* 设备编号

\*/

@ApiModelProperty(value = "设备编号", hidden = true)

private String deviceNumber;

/\*\*

\* 有效期

\*/

@Length(max = 255, message = "编码长度不能超过255")

@ApiModelProperty(value = "有效期", example = "有效期")

private String validityPeriod;

/\*\*

\* 申请检定时间

\*/

@ApiModelProperty(value = "申请检定时间", example = "2022-06-02 16:41:00")

@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")

private Date applicationVerificationTime;

/\*\*

\* 检定方式

\*/

@ApiModelProperty(value = "检定方式", example = "1")

private String verificationStyle;

/\*\*

\* 检定地点

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "检定地点", example = "茶园主场所")

private String verificationPlace;

/\*\*

\* 附送配件名称及数量

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "附送配件名称及数量", example = "检定设备、20个")

private String partsNameCount;

/\*\*

\* 检定所需工作日

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "检定所需工作日", example = "35个工作日")

private String verificationNeedWeekday;

/\*\*

\* 技术要求

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "技术要求", example = "检定技术")

private String technicalRequire;

/\*\*

\* 备注

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "备注", example = "备注")

private String remarks;

/\*\*

\* 申请状态

\*/

@ApiModelProperty(value = "申请状态", hidden = true)

private ApplyState applyStatus;

}

/\*\*

\* 检定校准申请流水记录

\*

\* @author wanggang

\* @date 2023/6/5 17:07

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(value = "检定校准申请流水记录DTO")

public class FlowDeviceVerificationRecordDTO extends FlowBaseInfoAddDTO {

/\*\*

\* 检定校准申请ID

\*/

@Length(max = 64, message = "编码长度不能超过64")

@ApiModelProperty(value = "检定校准申请ID", hidden = true)

private String id;

/\*\*

\* 申请人

\*/

@NotBlank(message = "[申请人ID]不能为空")

@Length(max = 255, message = "编码长度不能超过255")

@ApiModelProperty(value = "申请人", required = true, example = "1660846587764842497")

private String applyPerson;

/\*\*

\* 申请时间

\*/

@NotNull(message = "[申请时间]不能为空")

@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")

@ApiModelProperty(value = "申请时间", required = true, example = "2023-06-05 17:15:00")

private Date applyTime;

/\*\*

\* 审批人

\*/

@ApiModelProperty(value = "审批人", example = "1660846587764842497")

private String approvalPerson;

/\*\*

\* 审批人签名

\*/

@ApiModelProperty(value = "审批人签名", hidden = true)

private String approvalSign;

/\*\*

\* 审核意见

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "审核意见")

private String reviewComments;

/\*\*

\* 流水号

\*/

@ApiModelProperty(value = "流水号", hidden = true)

private String serialNumber;

/\*\*

\* 历史流水记录ID

\*/

@ApiModelProperty(value = "历史流水记录ID", hidden = true)

private String historyId;

/\*\*

\* 备注

\*/

@Length(max = 500, message = "编码长度不能超过500")

@ApiModelProperty(value = "备注", example = "备注")

private String remarks;

/\*\*

\* 检定申请内容

\*/

@ApiModelProperty(value = "检定申请内容", required = true, example = "[{},{}]")

private List<FlowDeviceVerificationDTO> dtoList;

}

/\*\*

\* 检定校准申请流水记录查询条件实体类

\*

\* @author wanggang

\* @date 2023/6/7 14:03

\*/

@Data

@EqualsAndHashCode(callSuper = false)

@ApiModel(value = "检定校准申请流水记录查询条件实体类")

public class FlowDeviceVerificationRecordSelectDTO extends PageEntity {

/\*\*

\* 模糊字段

\*/

@ApiModelProperty(value = "设备编号、设备名称、申请人、检定地点", name = "vagueField", dataType = "String")

private String vagueField;

/\*\*

\* 审批状态

\*/

@ApiModelProperty(value = "审批状态", name = "state", dataType = "String")

private String state;

/\*\*

\* 申请状态

\*/

@ApiModelProperty(value = "申请状态", name = "applyStatus", dataType = "String")

private String applyStatus;

/\*\*

\* 检定方式

\*/

@ApiModelProperty(value = "检定方式", name = "verificationStyle", dataType = "String")

private String verificationStyle;

/\*\*

\* 申请检定日期开始时间

\*/

@ApiModelProperty(value = "申请检定日期开始时间,格式：yyyy-MM-dd", name = "startTime", dataType = "String")

private String startTime;

/\*\*

\* 申请检定日期结束时间

\*/

@ApiModelProperty(value = "申请检定日期结束时间,格式：yyyy-MM-dd", name = "endTime", dataType = "String")

private String endTime;

}

/\*\*

\* 用户和组DTO

\*

\* @author wanggang

\* @date 2023/4/17 9:43

\*/

@Data

@ApiModel(value = "用户和组")

public class GroupAndUserDTO {

/\*\*

\* 组id

\*/

@ApiModelProperty(value = "groupId")

@NotEmpty(message = "组id不能为空")

private String groupId;

/\*\*

\* 用户id

\*/

@ApiModelProperty(value = "userId")

@NotEmpty(message = "用户id不能为空")

private String userId;

}

/\*\*

\* 用户组DTO

\*

\* @author wanggang

\* @date 2023/4/17 9:43

\*/

@Data

@ApiModel(value = "用户组")

@Valid

public class GroupDTO {

/\*\*

\* 组id

\*/

@ApiModelProperty(value = "组Id")

@NotEmpty(message = "id不能为空")

private String id;

/\*\*

\* 组名

\*/

@ApiModelProperty(value = "组名")

@NotEmpty(message = "组名不能为空")

private String name;

/\*\*

\* 类型

\*/

@ApiModelProperty(value = "类型")

private String type;

/\*\*

\* revision 的值为 0 表示这是一个新添加的组。

\*/

@ApiModelProperty(value = "修订参数 revision的值为0表示这是一个新添加的组")

@NotNull(message = "revision值不能为空")

@Max(message = "revision值只能是0或1", value = 1)

private Integer revision;

}

/\*\*

\* 历史任务信息DTO

\*

\* @author wanggang

\* @date 2023/5/4 13:47

\*/

@Data

public class HistoricTaskInstanceDTO {

private String id;

private String name;

private String description;

private int priority;

private String owner;

private String assignee;

private String processInstanceId;

private String executionId;

private String taskDefinitionId;

private String processDefinitionId;

private String scopeId;

private String subScopeId;

private String scopeType;

private String scopeDefinitionId;

private String propagatedStageInstanceId;

private Date createTime;

private String taskDefinitionKey;

private Date dueDate;

private String category;

private String parentTaskId;

private String tenantId;

private String formKey;

private String deleteReason;

private Date startTime;

private Date endTime;

private Long durationInMillis;

private Long workTimeInMillis;

private Date claimTime;

}

/\*\*

\* 初始化处理用户或人

\*

\* @author wanggang

\* @date 2023/4/23 10:26

\*/

@Data

@ApiModel(value = "初始化处理用户")

public class InitDealUser {

@ApiModelProperty(value = "候选组")

private String group;

// @ApiModelProperty(value = "处理用户")

// private String dealUser;

//

// @ApiModelProperty(value = "候选用户")

// private String user;

}

/\*\*

\* 流程定义DTO

\*

\* @author wanggang

\* @date 2023/4/25 16:35

\*/

@Data

@ApiModel(value = "流程定义DTO")

public class ProcessDefinitionDTO {

/\*\*

\* 流程名称

\*/

@ApiModelProperty(value = "流程名称")

private String name;

/\*\*

\* 本地化名称

\*/

@ApiModelProperty(value = "本地化名称")

private String localizedName;

/\*\*

\* 描述

\*/

@ApiModelProperty(value = "描述")

private String description;

/\*\*

\* 本地化描述

\*/

@ApiModelProperty(value = "本地化描述")

private String localizedDescription;

/\*\*

\* 流程key

\*/

@ApiModelProperty(value = "流程key")

private String key;

/\*\*

\* 版本

\*/

@ApiModelProperty(value = "版本")

private Integer version;

/\*\*

\* 流程类别

\*/

@ApiModelProperty(value = "流程类别")

private String category;

/\*\*

\* 流程部署的Id

\*/

@ApiModelProperty(value = "流程部署的Id")

private String deploymentId;

/\*\*

\* 流程资源名

\*/

@ApiModelProperty(value = "流程资源名")

private String resourceName;

/\*\*

\* 租户Id

\*/

@ApiModelProperty(value = "租户Id")

private String tenantId;

/\*\*

\* 历史记录级别

\*/

@ApiModelProperty(value = "历史记录级别")

private String historyLevel;

/\*\*

\* 图表资源名称

\*/

@ApiModelProperty(value = "图表资源名称")

private String diagramResourceName;

/\*\*

\* 是否定义了图形符号

\*/

@ApiModelProperty(value = "是否定义了图形符号")

private boolean isGraphicalNotationDefined;

/\*\*

\* 变量"

\*/

@ApiModelProperty(value = "变量")

private String variables;

/\*\*

\* 是否有开始表单键

\*/

@ApiModelProperty(value = "是否有开始表单键")

private boolean hasStartFormKey;

/\*\*

\* 暂停状态

\*/

@ApiModelProperty(value = "暂停状态")

private Integer suspensionState;

/\*\*

\* 是否初始化身份链接

\*/

@ApiModelProperty(value = "是否初始化身份链接")

private boolean isIdentityLinksInitialized;

/\*\*

\* IO规范

\*/

@ApiModelProperty(value = "IO规范")

private String ioSpecification;

/\*\*

\* 派生的值

\*/

@ApiModelProperty(value = "派生的值")

private String derivedFrom;

/\*\*

\* 派生的根

\*/

@ApiModelProperty(value = "派生的根")

private String derivedFromRoot;

/\*\*

\* 派生的版本

\*/

@ApiModelProperty(value = "派生的版本")

private Integer derivedVersion;

/\*\*

\* 引擎版本

\*/

@ApiModelProperty(value = "引擎版本")

private String engineVersion;

/\*\*

\* 流程id

\*/

@ApiModelProperty(value = "流程id")

private String id;

/\*\*

\* 修正系数

\*/

@ApiModelProperty(value = "修正系数")

private Integer revision;

/\*\*

\* 是否可以插入

\*/

@ApiModelProperty(value = "是否可以插入")

private boolean isInserted;

/\*\*

\* 是否可以更新

\*/

@ApiModelProperty(value = "是否可以更新")

private boolean isUpdated;

/\*\*

\* 是否可以删除

\*/

@ApiModelProperty(value = "是否可以删除")

private boolean isDeleted;

}

/\*\*

\* 流程DTO

\*

\* @author wanggang

\* @date 2023/4/18 10:14

\*/

@Data

@ApiModel(value = "流程DTO")

public class ProcessDTO {

@ApiModelProperty(value = "流程Id")

private String processId;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/6/5 10:47

\*/

@Data

public class RejectTaskDTO {

@NotBlank(message = "流程实例ID不能为空")

private String processId;

private String comment;

}

/\*\*

\* 任务DTO

\*

\* @author wanggang

\* @date 2023/4/19 10:22

\*/

@Data

@ApiModel(value = "任务")

public class TaskDTO {

@ApiModelProperty(value = "任务Id")

private String id;

@ApiModelProperty(value = "任务名")

private String name;

@ApiModelProperty(value = "任务处理人")

private String assignee;

}

/\*\*

\* 用户DTO

\*

\* @author wanggang

\* @date 2023/4/17 10:22

\*/

@Data

@ApiModel(value = "用户")

public class UserDTO {

@ApiModelProperty(value = "id")

@NotEmpty(message = "id不能为空")

private String id;

@ApiModelProperty(value = "姓氏")

private String firstName;

@ApiModelProperty(value = "名字")

private String lastName;

@ApiModelProperty(value = "显示名字")

@NotEmpty(message = "显示名字不能为空")

private String displayName;

@ApiModelProperty(value = "邮箱")

@NotEmpty(message = "邮箱不能为空")

private String email;

@ApiModelProperty(value = "密码")

@NotEmpty(message = "密码不能为空")

private String password;

/\*\*

\* revision 的值为 0 表示这是一个新添加的用户。

\*/

@ApiModelProperty(value = "修订参数 revision的值为0表示这是一个新添加的用户")

@NotNull(message = "revision值不能为空")

@Max(message = "revision值只能是0或1", value = 1)

private Integer revision;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/6/5 10:24

\*/

@Data

public class FlowBizVO {

@ApiModelProperty("能修改数据")

private Boolean canUpdate;

@ApiModelProperty("能废弃流程")

private Boolean canDelete;

@ApiModelProperty("能拒绝流程")

private Boolean canReject;

@ApiModelProperty("能审核流程")

private Boolean canApproval;

@ApiModelProperty("流程key")

private ProcessKey processKey;

@ApiModelProperty("业务数据对象")

private Object bizObj;

}

/\*\*

\* 采购申请流水记录

\*

\* @author wanggang

\* @date 2023/6/12 16:14

\*/

@Data

@ApiModel(value = "采购申请流水记录")

public class FlowDevicePurchaseRecordVO {

/\*\*

\* ID

\*/

@ApiModelProperty("ID")

private String id;

/\*\*

\* 编号

\*/

@ApiModelProperty("编号")

private String number;

/\*\*

\* 流程ID

\*/

@ApiModelProperty("流程ID")

private String processId;

/\*\*

\* 流程key

\*/

@ApiModelProperty("流程key")

private String processKey;

/\*\*

\* 相关资料

\*/

@ApiModelProperty("相关资料")

private String relatedInformation;

/\*\*

\* 申请人

\*/

@ApiModelProperty("申请人")

private String applyPerson;

/\*\*

\* 申请理由

\*/

@ApiModelProperty("申请理由")

private String applyReason;

/\*\*

\* 申请时间

\*/

@ApiModelProperty("申请时间")

private Date applyTime;

/\*\*

\* 审批人

\*/

@ApiModelProperty("审批人")

private String approvalPerson;

/\*\*

\* 审批人签名

\*/

@ApiModelProperty("审批人签名")

private String approvalSign;

/\*\*

\* 审核意见

\*/

@ApiModelProperty("审核意见")

private String reviewComments;

/\*\*

\* 申请状态

\*/

@ApiModelProperty("申请状态")

private ApplyState applyStatus;

/\*\*

\* 备注

\*/

@ApiModelProperty("备注")

private String remarks;

/\*\*

\* 设备申请内容

\*/

@ApiModelProperty(value = "设备申请内容")

private List<FlowDevicePurchaseVO> dtoList;

}

/\*\*

\* 采购申请明细记录

\*

\* @author wanggang

\* @date 2023/6/12 16:14

\*/

@Data

@ApiModel(value = "采购申请明细记录")

public class FlowDevicePurchaseVO {

/\*\*

\* ID

\*/

@ApiModelProperty("ID")

private String id;

/\*\*

\* 资产分类

\*/

@ApiModelProperty(value = "资产分类")

private String assetClassify;

/\*\*

\* 流程记录表ID

\*/

@ApiModelProperty("流程记录表ID")

private String processRecordId;

/\*\*

\* 设备分类ID

\*/

@ApiModelProperty("设备分类ID")

private String deviceClassifyInfoId;

/\*\*

\* 物品类别

\*/

@ApiModelProperty("物品类别")

private String itemCategory;

/\*\*

\* 采购类别

\*/

@ApiModelProperty("采购类别")

private String purchaseCategory;

/\*\*

\* 申请部门

\*/

@ApiModelProperty("申请部门")

private String applyDepartment;

/\*\*

\* 单位

\*/

@ApiModelProperty("单位")

private String unit;

/\*\*

\* 数量

\*/

@ApiModelProperty("数量")

private Integer deviceCount;

/\*\*

\* 单价

\*/

@ApiModelProperty("单价")

private BigDecimal unitPrice;

/\*\*

\* 总价

\*/

@ApiModelProperty("总价")

private BigDecimal totalPrice;

/\*\*

\* 备注

\*/

@ApiModelProperty("备注")

private String remarks;

}

/\*\*

\* 检定校准申请流水记录

\*

\* @author wanggang

\* @date 2023/6/5 17:07

\*/

@Data

@ApiModel(value = "检定校准申请流水记录返回数据")

public class FlowDeviceVerificationRecordVO {

/\*\*

\* 检定校准申请ID

\*/

@ApiModelProperty("检定校准申请ID")

private String id;

/\*\*

\* 流程ID

\*/

@ApiModelProperty("流程ID")

private String processId;

/\*\*

\* 流程key

\*/

@ApiModelProperty("流程key")

private String processKey;

/\*\*

\* 申请人

\*/

@ApiModelProperty("申请人")

private String applyPerson;

/\*\*

\* 申请时间

\*/

@ApiModelProperty("申请时间")

private Date applyTime;

/\*\*

\* 审批人

\*/

@ApiModelProperty("审批人")

private String approvalPerson;

/\*\*

\* 审批人签名

\*/

@ApiModelProperty("审批人签名")

private String approvalSign;

/\*\*

\* 审核意见

\*/

@ApiModelProperty("审核意见")

private String reviewComments;

/\*\*

\* 流水号

\*/

@ApiModelProperty("流水号")

private String serialNumber;

/\*\*

\* 历史流水记录ID

\*/

@ApiModelProperty("历史流水记录ID")

private String historyId;

/\*\*

\* 备注

\*/

@ApiModelProperty("备注")

private String remarks;

/\*\*

\* 检定申请内容

\*/

@ApiModelProperty(value = "检定申请内容")

private List<FlowDeviceVerificationVO> dtoList;

}

/\*\*

\* 检定校验申请返回结果实体

\*

\* @author wanggang

\* @date 2023/6/1 16:32

\*/

@Data

@ApiModel(value = "检定校验申请返回结果实体")

public class FlowDeviceVerificationVO {

/\*\*

\* 检定校准申请ID

\*/

@ApiModelProperty("检定校准申请ID")

private String id;

/\*\*

\* 流程ID

\*/

@ApiModelProperty("流程ID")

private String processId;

/\*\*

\* 流程记录ID

\*/

@ApiModelProperty("流程记录ID")

private String processRecordId;

/\*\*

\* 设备信息ID

\*/

@ApiModelProperty("设备信息ID")

private String deviceInfoId;

/\*\*

\* 设备名称

\*/

@ApiModelProperty(value = "设备名称")

private String deviceName;

/\*\*

\* 规格/型号

\*/

@ApiModelProperty(value = "规格/型号")

private String model;

/\*\*

\* 设备编号

\*/

@ApiModelProperty(value = "设备编号")

private String deviceNumber;

/\*\*

\* 有效期

\*/

@ApiModelProperty("有效期")

private String validityPeriod;

/\*\*

\* 申请检定时间

\*/

@ApiModelProperty("申请检定时间")

private Date applicationVerificationTime;

/\*\*

\* 检定方式

\*/

@ApiModelProperty("检定方式")

private String verificationStyle;

/\*\*

\* 检定方式显示

\*/

@ApiModelProperty("检定方式显示")

private String verificationStyleDisplay;

/\*\*

\* 检定地点

\*/

@ApiModelProperty("检定地点")

private String verificationPlace;

/\*\*

\* 检定地点名

\*/

@ApiModelProperty("检定地点名")

private String verificationPlaceName;

/\*\*

\* 附送配件名称及数量

\*/

@ApiModelProperty("附送配件名称及数量")

private String partsNameCount;

/\*\*

\* 检定所需工作日

\*/

@ApiModelProperty("检定所需工作日")

private String verificationNeedWeekday;

/\*\*

\* 技术要求

\*/

@ApiModelProperty("技术要求")

private String technicalRequire;

/\*\*

\* 申请状态

\*/

@ApiModelProperty("申请状态")

private ApplyState applyStatus;

/\*\*

\* 审批状态

\*/

@ApiModelProperty("审批状态")

private ProcessState state;

/\*\*

\* 备注

\*/

@ApiModelProperty("备注")

private String remarks;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/6/6 9:15

\*/

@Data

@AllArgsConstructor

@NoArgsConstructor

public class FlowHistVO {

@ApiModelProperty("活动名称")

private String activityName;

@ApiModelProperty("活动开始时间")

private Date startTime;

@ApiModelProperty("活动结束时间")

private Date endTime;

@ApiModelProperty("审核人")

private String assigneeEmpName;

@ApiModelProperty("审核意见")

private String comment;

@ApiModelProperty("任务状态")

private FlowTaskState taskState;

@ApiModelProperty("候选人姓名")

private String candidateUsers;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/6/5 14:06

\*/

@Data

public class FlowOrgRoleVO {

private List<String> upDownOrgIds;

private List<FlowRoleCode> roleCodes;

@Data

@AllArgsConstructor

@NoArgsConstructor

public static class FlowRoleCode{

private String flowRoleCode;

private String type;

}

}

@Data

public class BaseResponse {

private int errcode;

private String errmsg;

}

@EqualsAndHashCode(callSuper = true)

@Data

public class LoginResponse extends BaseResponse {

private String openid;

@JsonProperty("session\_key")

private String sessionKey;

private String unionid;

}

@EqualsAndHashCode(callSuper = true)

@Data

public class PhoneInfoResponse extends BaseResponse {

@JsonProperty("phone\_info")

private PhoneInfo phoneInfo;

@Data

public static class PhoneInfo {

private String phoneNumber;

// 没有区号的手机号

private String purePhoneNumber;

// 区号

private int countryCode;

// 数据水印

private Watermark watermark;

}

@Data

public static class Watermark {

private long timestamp;

private String appid;

}

}

/\*\*

\* @TableName file\_list

\*/

@EqualsAndHashCode(callSuper = true)

@TableName(value = "file\_attach")

@Data

public class FileAttach extends BaseEntity implements Serializable {

/\*\*

\*

\*/

@TableId

private String id;

/\*\*

\*

\*/

private String uuid;

/\*\*

\*

\*/

private String filename;

/\*\*

\*

\*/

private String contentType;

/\*\*

\*

\*/

private String md5;

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 1L;

}

/\*\*

\* @TableName sys\_dict

\*/

@EqualsAndHashCode(callSuper = true)

@TableName(value = "sys\_dict")

@Data

public class SysDict extends BaseEntity implements Serializable {

/\*\*

\*

\*/

@TableId

private String id;

/\*\*

\* 字典类型

\*/

private String type;

/\*\*

\* 字典value

\*/

private String val;

/\*\*

\* 字典名称

\*/

private String label;

private Integer sort;

private String sysDictCategoryId;

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 1L;

/\*\*

\* @TableName sys\_dict\_category

\*/

@EqualsAndHashCode(callSuper = true)

@TableName(value = "sys\_dict\_category")

@Data

public class SysDictCategory extends BaseEntity implements Serializable {

@TableId

private String id;

private String tenantId;

/\*\*

\* 类别名称

\*/

private String categoryName;

/\*\*

\* 键值key

\*/

private String categoryKey;

/\*\*

\* 描述

\*/

private String descr;

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 1L;

/\*\*

\* 系统流程角色

\* @TableName sys\_flow\_role

\*/

@EqualsAndHashCode(callSuper = true)

@TableName(value ="sys\_flow\_role")

@Data

public class SysFlowRole extends BaseEntity implements Serializable {

/\*\*

\* 流程类型1 上下级 2全局

\*/

@TableId

private String id;

@ApiModelProperty("流程角色编码")

private String code;

@ApiModelProperty("流程角色名称")

private String name;

@ApiModelProperty("流程类型1 上下级 2全局")

private String type;

@ApiModelProperty("描述")

private String descri;

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 1L;

/\*\*

\* @TableName sys\_log

\*/

@TableName(value = "sys\_log")

@Data

public class SysLog implements Serializable {

/\*\*

\*

\*/

@TableId

private String id;

/\*\*

\* class

\*/

private String module;

/\*\*

\* method

\*/

private String method;

/\*\*

\*

\*/

private String remoteAddr;

/\*\*

\*

\*/

private String userAgent;

/\*\*

\*

\*/

private String requestUri;

/\*\*

\*

\*/

private String params;

/\*\*

\*

\*/

private String ex;

/\*\*

\*

\*/

private Long execTime;

/\*\*

\*

\*/

private Date time;

/\*\*

\* 1异常

\*/

private Integer exStatus;

/\*\*

\*

\*/

private String createBy;

/\*\*

\*

\*/

private Date createTime;

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 1L;

/\*\*

\* 菜单

\*

\* @TableName sys\_menu

\*/

@EqualsAndHashCode(callSuper = true)

@TableName(value = "sys\_menu")

@Data

public class SysMenu extends BaseEntity implements Serializable {

/\*\*

\*

\*/

@TableId

private String id;

/\*\*

\* 父级id

\*/

private String parentId;

/\*\*

\* 菜单标题

\*/

private String title;

/\*\*

\* 菜单编码

\*/

private String code;

/\*\*

\* 1目录 2菜单 3按钮

\*/

private String type;

/\*\*

\* 菜单路径

\*/

private String path;

/\*\*

\* vue路由组件

\*/

private String component;

/\*\*

\* 图标

\*/

private String icon;

/\*\*

\* 排序

\*/

private Integer sort;

private String isHide;

//菜单区分app和web

private String menuSys;

@TableField(exist = false)

private List<SysMenu> children;

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 1L;

/\*\*

\* @TableName sys\_org

\*/

@EqualsAndHashCode(callSuper = true)

@TableName(value = "sys\_org")

@Data

public class SysOrg extends BaseEntity implements Serializable {

/\*\*

\*

\*/

@TableId

private String id;

/\*\*

\* 父级ID

\*/

private String parentId;

/\*\*

\*

\*/

private String orgNo;

/\*\*

\* 分级列表

\*/

private String ancestors;

/\*\*

\* 完整路径

\*/

private String fullOrgName;

/\*\*

\*

\*/

private String orgName;

/\*\*

\* 排序

\*/

private Integer sort;

/\*\*

\* 组织类型1:公司 2部门

\*/

private Integer orgType;

private String tenantId;

@TableField(exist = false)

List<SysOrg> children;

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 1L;

/\*\*

\* @TableName sys\_post

\*/

@EqualsAndHashCode(callSuper = true)

@TableName(value = "sys\_post")

@Data

public class SysPost extends BaseEntity implements Serializable {

/\*\*

\*

\*/

@TableId

private String id;

/\*\*

\*

\*/

private String postName;

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 1L;

/\*\*

\* 角色

\*

\* @TableName sys\_role

\*/

@EqualsAndHashCode(callSuper = true)

@TableName(value = "sys\_role")

@Data

public class SysRole extends BaseEntity implements Serializable {

/\*\*

\*

\*/

@TableId

private String id;

/\*\*

\* 角色编码

\*/

private String code;

/\*\*

\* 对应的组织ID

\*/

private String orgId;

/\*\*

\*

\*/

private String roleName;

private String tenantId;

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 1L;

/\*\*

\* @TableName sys\_role\_menu

\*/

@EqualsAndHashCode(callSuper = true)

@TableName(value = "sys\_role\_menu")

@Data

public class SysRoleMenu extends BaseEntity implements Serializable {

/\*\*

\*

\*/

@TableId

private String id;

/\*\*

\*

\*/

private String sysRoleId;

/\*\*

\*

\*/

private String sysMenuId;

private String tenantId;

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 1L;

/\*\*

\* 租户信息表

\* @TableName sys\_tenant

\*/

@EqualsAndHashCode(callSuper = true)

@Data

@ApiModel("租户信息表")

public class SysTenant extends BaseEntity implements Serializable {

@Serial

private static final long serialVersionUID = 7000167767921345462L;

/\*\*

\* 租户ID

\*/

@NotBlank(message="[租户ID]不能为空")

@Size(max= 64,message="编码长度不能超过64")

@ApiModelProperty("租户ID")

@TableId

private String id;

/\*\*

\* 租户名称

\*/

@Size(max= 20,message="编码长度不能超过20")

@ApiModelProperty("租户名称")

@Length(max= 20,message="编码长度不能超过20")

private String tenantName;

/\*\*

\* 租户套餐ID

\*/

@Size(max= 64,message="编码长度不能超过64")

@ApiModelProperty("租户套餐ID")

@Length(max= 64,message="编码长度不能超过64")

private String packageId;

/\*\*

\* 认证分类（1:企业;2:个人;）

\*/

@ApiModelProperty("认证分类（1:企业;2:个人;）")

private String identifiClassify;

/\*\*

\* 注册方式（0:注册;1:邮箱;2:微信;3:QQ;4:钉钉:5:其他）

\*/

@ApiModelProperty("注册方式（0:注册;1:邮箱;2:微信;3:QQ;4:钉钉:5:其他）")

private String registeMode;

/\*\*

\* 状态（1:未激活;2:激活;3:注销;4:禁用;）

\*/

@ApiModelProperty("状态（1:未激活;2:激活;3:注销;4:禁用;）")

private String status;

/\*\*

\* 备注

\*/

@Size(max= 100,message="编码长度不能超过100")

@ApiModelProperty("备注")

@Length(max= 100,message="编码长度不能超过100")

private String remark;

}

/\*\*

\* 租户套餐表

\*

\* @TableName sys\_tenant\_pkg

\*/

@EqualsAndHashCode(callSuper = true)

@Data

public class SysTenantPkg extends BaseEntity implements Serializable {

@Serial

private static final long serialVersionUID = -5672841994918681398L;

/\*\*

\* 套餐编号

\*/

@TableId

private String id;

/\*\*

\* 套餐名

\*/

@NotBlank(message = "[套餐名]不能为空")

@Size(max = 30, message = "编码长度不能超过30")

@ApiModelProperty("套餐名")

@Length(max = 30, message = "编码长度不能超过30")

private String pkgName;

/\*\*

\* 备注

\*/

@Size(max = 256, message = "编码长度不能超过256")

@ApiModelProperty("备注")

@Length(max = 256, message = "编码长度不能超过256")

private String remark;

}

/\*\*

\* 套餐菜单

\* @TableName sys\_tenant\_pkg\_menu

\*/

@Data

public class SysTenantPkgMenu implements Serializable {

@Serial

private static final long serialVersionUID = -6410022815505479439L;

@TableId

private String id;

/\*\*

\* 套餐ID

\*/

@Size(max= 64,message="编码长度不能超过64")

@ApiModelProperty("套餐ID")

@Length(max= 64,message="编码长度不能超过64")

private String sysTenantPkgId;

/\*\*

\* 菜单ID

\*/

@Size(max= 64,message="编码长度不能超过64")

@ApiModelProperty("菜单ID")

@Length(max= 64,message="编码长度不能超过64")

private String sysMenuId;

}

/\*\*

\* 系统用户

\*

\* @TableName sys\_user

\*/

@EqualsAndHashCode(callSuper = true)

@TableName(value = "sys\_user")

@Data

public class SysUser extends BaseEntity implements Serializable {

/\*\*

\*

\*/

@TableId

private String id;

/\*\*

\* 用户名

\*/

private String username;

/\*\*

\* 密码

\*/

private String password;

/\*\*

\* 员工编号

\*/

private String empNo;

/\*\*

\* 姓名

\*/

private String empName;

/\*\*

\* 性别

\*/

private String gender;

/\*\*

\* 手机

\*/

private String mobile;

/\*\*

\* 出生日期

\*/

private String birthday;

/\*\*

\* 组织Id

\*/

private String sysOrgId;

/\*\*

\* 职位ID

\*/

private String sysPostId;

/\*\*

\* 入职时间

\*/

private String entryDay;

/\*\*

\* 离职时间

\*/

private String leaveDay;

/\*\*

\* 员工状态 1在职 2离职

\*/

private String empStatus;

//签名文件

private String signatureBase64;

/\*\*

\* 账号是否启用

\*/

private String enabled;

private String tenantId;

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 1L;

}

/\*\*

\* 用户流程角色

\* @TableName sys\_user\_flow\_role

\*/

@EqualsAndHashCode(callSuper = true)

@TableName(value ="sys\_user\_flow\_role")

@Data

public class SysUserFlowRole extends BaseEntity implements Serializable {

/\*\*

\*

\*/

@TableId

private String id;

/\*\*

\* 用户ID

\*/

private String sysUserId;

/\*\*

\* 流程角色ID

\*/

private String sysFlowRoleId;

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 1L;

/\*\*

\* @TableName sys\_user\_org

\*/

@EqualsAndHashCode(callSuper = true)

@TableName(value = "sys\_user\_org",autoResultMap = true)

@Data

public class SysUserOrg extends BaseEntity implements Serializable {

/\*\*

\*

\*/

@TableId

private String id;

/\*\*

\*

\*/

private String sysUserId;

/\*\*

\*

\*/

private String sysOrgId;

private String sysPostId;

@TableField(typeHandler = JacksonTypeHandler.class)

private List<String> sysFlowRoleId;

@TableField(exist = false)

private List<SysFlowRole> sysFlowRoles;

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 1L;

/\*\*

\* @TableName sys\_user\_role

\*/

@EqualsAndHashCode(callSuper = true)

@TableName(value = "sys\_user\_role")

@Data

public class SysUserRole extends BaseEntity implements Serializable {

/\*\*

\*

\*/

@TableId

private String id;

/\*\*

\*

\*/

private String sysUserId;

/\*\*

\*

\*/

private String sysRoleId;

private String tenantId;

@Serial

@TableField(exist = false)

private static final long serialVersionUID = 1L;

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/22 12:38

\*/

@Data

@ApiModel("用户的登录")

public class LoginDTO {

@ApiModelProperty(value = "用户名", example = "admin")

@NotBlank(message = "用户名不能为空")

private String username;

@ApiModelProperty(value = "密码", example = "admin123")

@NotBlank(message = "密码不能为空")

private String password;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/24 10:19

\*/

@Data

public class ResetPasswordDTO {

@NotBlank(message = "用户ID不能为空")

private String sysUserId;

//^(?![0-9]+$)(?![a-zA-Z]+$)[0-9A-Za-z]{6,16}$

@NotBlank(message = "密码不能为空")

@Pattern(regexp = "^(?![0-9]+$)(?![a-zA-Z]+$)[0-9A-Za-z]{6,16}$", message = "密码必须是6-18位且包含数字和字母")

private String password;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/23 14:16

\*/

@Data

public class RoleAuthMenusDTO {

private String roleId;

List<String> menuIds;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/27 9:23

\*/

@Data

public class SaveUserRolesDTO {

@NotBlank(message = "用户ID不能为null")

private String sysUserId;

@NotEmpty(message = "角色不能为空")

private List<String> roleIds;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/5/9 17:40

\*/

@Data

@AllArgsConstructor

@NoArgsConstructor

public class SysTenantAddDTO {

/\*\*

\* 租户名称

\*/

@Size(max= 20,message="编码长度不能超过20")

@ApiModelProperty("租户名称")

@Length(max= 20,message="编码长度不能超过20")

private String tenantName;

/\*\*

\* 租户套餐ID

\*/

@Size(max= 64,message="编码长度不能超过64")

@ApiModelProperty("租户套餐ID")

@Length(max= 64,message="编码长度不能超过64")

private String packageId;

/\*\*

\* 认证分类（1:企业;2:个人;）

\*/

@ApiModelProperty("认证分类（1:企业;2:个人;）")

private String identifiClassify;

/\*\*

\* 注册方式（0:注册;1:邮箱;2:微信;3:QQ;4:钉钉:5:其他）

\*/

@ApiModelProperty("注册方式（0:注册;1:邮箱;2:微信;3:QQ;4:钉钉:5:其他）")

private String registeMode;

/\*\*

\* 状态（1:未激活;2:激活;3:注销;4:禁用;）

\*/

@ApiModelProperty("状态（1:未激活;2:激活;3:注销;4:禁用;）")

private String status;

/\*\*

\* 备注

\*/

@Size(max= 100,message="编码长度不能超过100")

@ApiModelProperty("备注")

@Length(max= 100,message="编码长度不能超过100")

private String remark;

@NotBlank(message = "用户名不能为空")

private String username;

@NotBlank(message = "密码不能为空")

private String password;

@NotBlank(message = "手机号不能为空")

private String mobile;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/5/9 17:23

\*/

@EqualsAndHashCode(callSuper = true)

@Data

public class SysTenantPageDTO extends PageEntity {

@ApiModelProperty("租户名称")

private String tenantName;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/5/10 10:04

\*/

@EqualsAndHashCode(callSuper = true)

@Data

@ApiModel("租户套餐新增修改")

public class SysTenantPkgDTO extends SysTenantPkg {

@ApiModelProperty("菜单ID")

@Size(min = 1,message = "PC菜单不能为空")

private List<String> sysMenuIdsPc;

@ApiModelProperty("菜单ID")

@Size(min = 1,message = "APP菜单不能为空")

private List<String> sysMenuIdsApp;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/5/10 10:02

\*/

@EqualsAndHashCode(callSuper = true)

@Data

public class SysTenantPkgPageDTO extends PageEntity {

@ApiModelProperty("套餐名称")

private String pkgName;

}

/\*\*

\* @Description: 新增用户提示

\* @Author: yb

\* @Date: 2023/2/22 13:34

\*/

@Data

public class SysUserAddDTO implements Serializable {

@Serial

private static final long serialVersionUID = -1185183339596430487L;

/\*\*

\* 用户名

\*/

private String username;

/\*\*

\* 员工编号

\*/

private String empNo;

/\*\*

\* 姓名

\*/

private String empName;

/\*\*

\* 性别

\*/

private String gender;

/\*\*

\* 手机

\*/

private String mobile;

/\*\*

\* 出生日期

\*/

private String birthday;

/\*\*

\* 组织Id

\*/

private String sysOrgId;

/\*\*

\* 职位ID

\*/

private String sysPostId;

/\*\*

\* 入职时间

\*/

private String entryDay;

/\*\*

\* 离职时间

\*/

private String leaveDay;

/\*\*

\* 员工状态 1在职 2离职

\*/

private String empStatus;

//签名文件

private String signatureBase64;

/\*\*

\* 账号是否启用 1启用 0停用

\*/

private String enabled;

@NotEmpty(message = "角色不能为空")

private List<String> roleIds;

private List<String> flowRoleIds;

private List<SysUserOrg> userOrgs;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/23 17:30

\*/

@EqualsAndHashCode(callSuper = true)

@Data

public class SysUserUpdateDTO extends SysUserAddDTO {

@Serial

private static final long serialVersionUID = 195150681077165564L;

private String id;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/27 11:12

\*/

@Data

public class UpdateMyInfoDTO {

private String empName;

private String gender;

private String mobile;

private String birthday;

private String signatureBase64;

}

/\*\*

\* 文件实体VO

\*

\* @author wanggang

\* @date 2023/5/29 9:22

\*/

@Data

public class FileAttachVO {

/\*\*

\* 文件Id

\*/

@ApiModelProperty("文件Id")

private String id;

/\*\*

\* 文件路径

\*/

@ApiModelProperty("文件路径")

private String uuid;

/\*\*

\* 文件名称

\*/

@ApiModelProperty("文件名称")

private String filename;

/\*\*

\* 内容类型

\*/

@ApiModelProperty("内容类型")

private String contentType;

/\*\*

\* 加密

\*/

@ApiModelProperty("加密")

private String md5;

/\*\*

\* 上传时间

\*/

@ApiModelProperty("上传时间")

private Date createTime;

/\*\*

\* 上传人

\*/

@ApiModelProperty("上传人")

private String createBy;

/\*\*

\* 上传人名

\*/

@ApiModelProperty("上传人")

private String createName;

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/23 11:26

\*/

@Data

public class SupOrgListVO {

private List<String> ancestors;

private List<String> pOrgNames;

}

/\*\*

\* 数据字典表

\*

\* @author wanggang

\* @date 2023/5/16 14:05

\*/

@Data

@ApiModel(value = "数据字典表")

public class SysDictVO {

/\*\*

\* 类别名称

\*/

@ApiModelProperty("类别名称")

private String categoryName;

/\*\*

\* 类别key

\*/

@ApiModelProperty("类别key")

private String categoryKey;

/\*\*

\* id

\*/

@ApiModelProperty("id")

private String id;

/\*\*

\* 字典类型

\*/

@ApiModelProperty("字典类型")

private String type;

/\*\*

\* 字典value

\*/

@ApiModelProperty("字典value")

private String val;

/\*\*

\* 字典名称

\*/

@ApiModelProperty("字典名称")

private String label;

/\*\*

\* 排序

\*/

@ApiModelProperty("排序")

private Integer sort;

/\*\*

\* 系统词典类别Id

\*/

@ApiModelProperty("系统词典类别Id")

private String sysDictCategoryId;

/\*\*

\* 组织架构表

\*

\* @author wanggang

\* @date 2023/5/16 12:01

\* @TableName sys\_org

\*/

@Data

@ApiModel(value = "组织架构表")

public class SysOrgVO {

/\*\*

\* id

\*/

@ApiModelProperty("id")

private String id;

/\*\*

\* 父级ID

\*/

@ApiModelProperty("父级ID")

private String parentId;

/\*\*

\* 组织架构编号

\*/

@ApiModelProperty("组织架构编号")

private String orgNo;

/\*\*

\* 分级列表

\*/

@ApiModelProperty("分级列表")

private String ancestors;

/\*\*

\* 完整路径

\*/

@ApiModelProperty("完整路径")

private String fullOrgName;

/\*\*

\* 组织架构名

\*/

@ApiModelProperty("组织架构名")

private String orgName;

/\*\*

\* 排序

\*/

@ApiModelProperty("排序")

private Integer sort;

/\*\*

\* 组织类型1:公司 2部门

\*/

@ApiModelProperty("组织类型1:公司 2部门")

private Integer orgType;

@ApiModelProperty("租户ID")

private String tenantId;

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/5/10 10:04

\*/

@EqualsAndHashCode(callSuper = true)

@Data

public class SysTenantPkgVO extends SysTenantPkg {

@ApiModelProperty("pc菜单ID")

private List<String> sysMenuIdsPc;

@ApiModelProperty("pc菜单ID")

private List<String> sysMenuIdsApp;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/5/10 14:51

\*/

@EqualsAndHashCode(callSuper = true)

@Data

public class SysTenantVO extends SysTenant {

private String pkgName;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/23 16:12

\*/

@Data

public class SysUserVO implements Serializable {

@Serial

private static final long serialVersionUID = -8103750087897518616L;

/\*\*

\*

\*/

private String id;

/\*\*

\* 用户名

\*/

private String username;

/\*\*

\* 员工编号

\*/

private String empNo;

/\*\*

\* 姓名

\*/

private String empName;

/\*\*

\* 性别

\*/

private String gender;

/\*\*

\* 手机

\*/

private String mobile;

/\*\*

\* 出生日期

\*/

private String birthday;

/\*\*

\* 组织Id

\*/

private String sysOrgId;

/\*\*

\* 职位ID

\*/

private String sysPostId;

/\*\*

\* 入职时间

\*/

private String entryDay;

/\*\*

\* 离职时间

\*/

private String leaveDay;

/\*\*

\* 员工状态 1在职 2离职

\*/

private String empStatus;

/\*\*

\* 账号是否启用 1启用 0停用

\*/

private String enabled;

/\*\*

\* 创建人id

\*/

private String createBy;

/\*\*

\* 创建时间

\*/

private Date createTime;

//职位名称

private String sysPostName;

//组织全路径

private String fullOrgName;

private String roleNames;

private String signatureBase64;

private List<String> roleIds;

private List<String> flowRoleIds;

private List<SysUserOrg> userOrgs;

}

/\*\*

\* @author wanggang

\* @date 2023/6/6 15:51

\*/

@AllArgsConstructor

@Getter

@JsonFormat(shape = JsonFormat.Shape.OBJECT)

public enum ApplyState {

NOT\_APPLY("1", "未申请"),

APPLY("2", "已申请");

@EnumValue//标记数据库存的值是code

private final String key;

private final String name;

}

/\*\*

\* 是否自动采集(0否;1是)

\*

\* @author wanggang

\* @date 2023/5/11 9:07

\*/

public enum DeviceAutomaticCollectionEnum {

/\*\*

\* 是否自动采集(0否;1是)

\*/

NO(0, "否"),

YES(1, "是");

DeviceAutomaticCollectionEnum(int value, String text) {

this.value = value;

this.text = text;

}

private int value;

private String text;

public int getValue() {

return value;

}

public void setValue(int value) {

this.value = value;

}

public String getText() {

return text;

}

public void setText(String text) {

this.text = text;

}

/\*\*

\* 通过text获取value

\*

\* @param text 枚举值

\* @return

\*/

public static Integer getValueByText(String text) {

// 遍历当前枚举列表

for (DeviceAutomaticCollectionEnum temp : DeviceAutomaticCollectionEnum.values()) {

// value值相等

if (temp.getText().equals(text)) {

return temp.getValue();

}

}

return null;

}

/\*\*

\* 通过value获取text

\*

\* @param value

\* @return

\*/

public static String getTextByValue(int value) {

// 遍历当前枚举列表

for (DeviceAutomaticCollectionEnum temp : DeviceAutomaticCollectionEnum.values()) {

// value值相等

if (temp.getValue() == value) {

return temp.getText();

}

}

return null;

}

}

/\*\*

\* 设备信息里面的结果

\*

\* @author wanggang

\* @date 2023/5/24 9:07

\*/

public enum DeviceResultEnum {

/\*\*

\* 合格/不合格

\*/

UNQUALIFIED(0, "不合格"),

QUALIFIED(1, "合格");

DeviceResultEnum(int value, String text) {

this.value = value;

this.text = text;

}

private int value;

private String text;

public int getValue() {

return value;

}

public void setValue(int value) {

this.value = value;

}

public String getText() {

return text;

}

public void setText(String text) {

this.text = text;

}

/\*\*

\* 通过text获取value

\*

\* @param text 枚举值

\* @return

\*/

public static Integer getValueByText(String text) {

// 遍历当前枚举列表

for (DeviceResultEnum temp : DeviceResultEnum.values()) {

// value值相等

if (temp.getText().equals(text)) {

return temp.getValue();

}

}

return null;

}

/\*\*

\* 通过value获取text

\*

\* @param value

\* @return

\*/

public static String getTextByValue(int value) {

// 遍历当前枚举列表

for (DeviceResultEnum temp : DeviceResultEnum.values()) {

// value值相等

if (temp.getValue() == value) {

return temp.getText();

}

}

return null;

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/6/8 16:18

\*/

@AllArgsConstructor

@Getter

@JsonFormat(shape = JsonFormat.Shape.OBJECT)

public enum FlowTaskState {

NOT\_APPROVE("1", "待审核"),

APPROVED("2", "已通过"),

REJECT("3", "被拒绝");

@EnumValue//标记数据库存的值是code

private final String key;

private final String name;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/5/31 16:44

\*/

@AllArgsConstructor

@Getter

@JsonFormat(shape = JsonFormat.Shape.OBJECT)

public enum ProcessKey {

DEVICE\_CHECK\_VALIDATED("device\_check\_validated","设备检定校准流程"),

DEVICE\_PURCHASE("device\_purchase","采购申请");

@EnumValue//标记数据库存的值是code

private final String key;

private final String name;

public static ProcessKey match(String key) {

ProcessKey result = null;

for (ProcessKey s : values()) {

if (s.getKey().equals(key)) {

result = s;

break;

}

}

return result;

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/5/31 17:10

\*/

@AllArgsConstructor

@Getter

@JsonFormat(shape = JsonFormat.Shape.OBJECT)

public enum ProcessState {

NOT\_APPROVE("1", "审批中"),

COMPLETE("2", "已完成"),

REJECT("3", "被拒绝"),

REVOKE("4", "已撤销");

@EnumValue//标记数据库存的值是code

private final String key;

private final String name;

public static ProcessState match(String key) {

ProcessState result = null;

for (ProcessState s : values()) {

if (s.getKey().equals(key)) {

result = s;

break;

}

}

return result;

}

}

/\*\*

\* 异常类封装

\* @author wanggang

\* @date 2023/4/12 15:53

\*

\*/

@Data

public class CodeMsg {

/\*\*

\* 异常代码

\*/

private int code;

/\*\*

\* 异常信息

\*/

private String msg;

}

/\*\*

\* @author wanggang

\* @descript 异常代码

\* @date 2023/5/8 11:14

\*/

public enum ErrCode {

/\*\*

\* 异常枚举

\*/

SYSTEM\_ERROR(HttpStatus.HTTP\_INTERNAL\_ERROR, "系统异常"),

BUSINESS\_ERROR(HttpStatus.HTTP\_BAD\_REQUEST, "业务异常");

ErrCode(int code, String value) {

this.code = code;

this.value = value;

}

public String getValue() {

return value;

}

public void setValue(String value) {

this.value = value;

}

public int getCode() {

return code;

}

public void setCode(int code) {

this.code = code;

}

private int code;

private String value;

}

/\*\*

\* 异常基类

\*

\* @author: wg

\* @date: 2019/2/25

\*/

public class JcglException extends RuntimeException {

private int code = ErrCode.SYSTEM\_ERROR.getCode();//错误编码，前台根据这个编码可以做相应业务处理，默认500服务内部异常

public JcglException(int code, Exception e) {

super(e);

this.code = code;

}

public JcglException(int code, String msg) {

super(msg);

this.code = code;

}

public JcglException() {

super();

}

public JcglException(Exception e) {

super(e);

}

public JcglException(String msg) {

super(msg);

}

public JcglException(String msg, Exception e) {

super(msg, e);

}

public int getCode() {

return code;

}

public void setCode(int code) {

this.code = code;

}

}

/\*\*

\* 参数异常

\*

\* @author wanggang

\* @date 2023/4/4 11:07

\*/

public class ParameterException extends JcglException {

private static final int code = ErrCode.BUSINESS\_ERROR.getCode();

public ParameterException() {

super(code, "");

}

public ParameterException(Exception e) {

super(code, e);

}

public ParameterException(String msg) {

super(code, msg);

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/6/8 9:59

\*/

public class ProcessException extends JcglException {

public ProcessException(String msg) {

super(msg);

}

}

/\*\*

\* 业务异常

\*

\* @author wanggang

\* @date 2023/4/4 11:07

\*/

public class ServiceException extends JcglException {

private static final int code = ErrCode.BUSINESS\_ERROR.getCode();

public ServiceException() {

super();

}

public ServiceException(Exception e) {

super(e);

}

public ServiceException(String msg) {

super(msg);

}

public ServiceException(int code, Exception e) {

super(code, e);

}

public ServiceException(int code, String msg) {

super(code, msg);

}

public ServiceException(String msg, Exception e) {

super(msg, e);

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/6/1 9:27

\*/

public interface FlowBaseFactory {

/\*\*

\* 设置流程Key

\*

\* @return

\*/

ProcessKey getProcessKey();

/\*\*

\* 创建流程

\*

\* @param flowBaseInfoDTO

\* @param processId

\* @return

\*/

void createProcess(JSONObject flowBaseInfoDTO, String processId) throws ProcessException;

/\*\*

\* 流程结束事件

\* @param processId

\*/

void processEnd(String processId) throws ProcessException;

/\*\*

\* 任务完成操作

\*

\* @param sysUser

\* @param taskEventId 任务标识key

\* @param commentMsg

\*/

void taskComplete(String processId, SysUser sysUser, String taskEventId, String commentMsg) throws ProcessException;

/\*\*

\* 查询流程业务业务数据

\* @param processId

\* @return

\*/

Object getProcessBiz(String processId);

/\*\*

\* 修改流程业务数据

\* @param flowBaseInfoUpdateDTO

\* @param processId

\* @throws ProcessException

\*/

void updateProcessBiz(JSONObject flowBaseInfoUpdateDTO, String processId) throws ProcessException;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/6/1 11:18

\*/

@Component

public class FlowBaseHandlerFactory implements InitializingBean, ApplicationContextAware {

private final Map<String, Supplier<FlowBaseFactory>> MAP = new ConcurrentHashMap<>();

private ApplicationContext applicationContext;

@Autowired

IFlowBaseInfoService flowBaseInfoService;

@Autowired

FlowCreateDecorator flowCreateDecorator;

public FlowBaseFactory createHandler(String processKey) {

Supplier<FlowBaseFactory> p = MAP.get(processKey);

if (p != null) {

flowCreateDecorator.setFlowBaseFactory(p.get());

return flowCreateDecorator;

}

throw new IllegalArgumentException("No such factory by code:" + processKey);

}

@Override

public void afterPropertiesSet() throws Exception {

applicationContext.getBeansOfType(FlowBaseFactory.class)

.values()

.forEach(flowBaseFactory -> {

if (flowBaseFactory.getProcessKey()!=null){

MAP.put(flowBaseFactory.getProcessKey().getKey(), () -> flowBaseFactory);

}

});

}

@Override

public void setApplicationContext(@NotNull ApplicationContext applicationContext) throws BeansException {

this.applicationContext = applicationContext;

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/6/1 12:50

\*/

@Component

public class FlowCreateDecorator implements FlowBaseFactory {

private FlowBaseFactory flowBaseFactory;

@Autowired

IFlowBaseInfoService flowBaseInfoService;

@Autowired

IFlowFactory flowFactory;

@Autowired

ISysOrgService sysOrgService;

public void setFlowBaseFactory(FlowBaseFactory flowBaseFactory) {

this.flowBaseFactory = flowBaseFactory;

}

@Override

public ProcessKey getProcessKey() {

if (flowBaseFactory != null) {

return flowBaseFactory.getProcessKey();

} else {

return null;

}

}

@Override

@Transactional(rollbackFor = Exception.class)

public void createProcess(JSONObject flowBaseInfoDTO, String processId) throws ProcessException {

SysUser sysUser = TenantContextHolder.getContext().getTenantContextInfo().getSysUser();

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

map.put("orgId", sysUser.getSysOrgId());

map.put("INITIATOR", sysUser.getId());

Authentication.setAuthenticatedUserId(sysUser.getId());

ProcessInstance processInstance = flowFactory.getRuntimeService().startProcessInstanceByKey(getProcessKey().getKey(), map);

flowBaseInfoDTO.put("processId", processInstance.getProcessInstanceId());

flowBaseFactory.createProcess(flowBaseInfoDTO, processInstance.getProcessInstanceId());

FlowBaseInfo flowBaseInfo = new FlowBaseInfo();

//装饰器对创建流程设置创建人

flowBaseInfo.setProcessId(processInstance.getProcessInstanceId());

flowBaseInfo.setProcessName(flowBaseInfoDTO.getString("processName"));

flowBaseInfo.setSysOrgId(sysUser.getSysOrgId());

flowBaseInfo.setState(ProcessState.NOT\_APPROVE);

flowBaseInfo.setProcessKey(getProcessKey());

SysOrg sysOrg=sysOrgService.getById(sysUser.getSysOrgId());

flowBaseInfo.setSysOrgFullName(sysOrg.getFullOrgName());

flowBaseInfoService.save(flowBaseInfo);

//发起申请流程完成

Task task = flowFactory.getTaskService().createTaskQuery().processInstanceId(processInstance.getProcessInstanceId())

.taskAssignee(sysUser.getId())

.singleResult();

if (task != null) {

flowFactory.getTaskService().addComment(task.getId(), processInstance.getProcessInstanceId(), "提交申请");

flowFactory.getTaskService().complete(task.getId());

}

}

@Override

@Transactional(rollbackFor = Exception.class)

public void processEnd(String processId) throws ProcessException {

flowBaseFactory.processEnd(processId);

//

FlowBaseInfo flowBaseInfo = flowBaseInfoService.getOne(new QueryWrapper<FlowBaseInfo>().lambda().eq(FlowBaseInfo::getProcessId, processId));

if (flowBaseInfo != null) {

//修改完成状态

flowBaseInfo.setEndTime(new DateTime());

flowBaseInfo.setState(ProcessState.COMPLETE);

flowBaseInfoService.updateById(flowBaseInfo);

}

}

@Override

@Transactional(rollbackFor = Exception.class)

public void taskComplete(String processId, SysUser sysUser, String taskEventId, String commentMsg) throws ProcessException {

FlowBaseInfo flowBaseInfo = flowBaseInfoService.getOne(new QueryWrapper<FlowBaseInfo>().lambda().eq(FlowBaseInfo::getProcessId, processId));

flowBaseInfo.setState(ProcessState.NOT\_APPROVE);

flowBaseInfoService.updateById(flowBaseInfo);

flowBaseFactory.taskComplete(processId, sysUser, taskEventId, commentMsg);

}

@Override

public Object getProcessBiz(String processId) {

return flowBaseFactory.getProcessBiz(processId);

}

@Override

@Transactional(rollbackFor = Exception.class)

public void updateProcessBiz(JSONObject flowBaseInfoUpdateDTO, String processId) throws ProcessException {

flowBaseFactory.updateProcessBiz(flowBaseInfoUpdateDTO, processId);

}

}

/\*\*

\* 采购申请流程

\*

\* @author wanggang

\* @date 2023/6/13 11:36

\*/

@Service

public class FlowDevicePurchaseFactory implements FlowBaseFactory {

@Autowired

IFlowBaseInfoService flowBaseInfoService;

@Resource

IFlowDevicePurchaseService iFlowDevicePurchaseService;

@Resource

IFlowDevicePurchaseRecordService iFlowDevicePurchaseRecordService;

@Resource

FlowDevicePurchaseRecordMapper flowDevicePurchaseRecordMapper;

@Resource

FlowDevicePurchaseMapper flowDevicePurchaseMapper;

@Override

public ProcessKey getProcessKey() {

return ProcessKey.DEVICE\_PURCHASE;

}

@Override

public void createProcess(JSONObject flowBaseInfoDTO, String processId) throws ProcessException {

FlowDevicePurchaseRecordDTO flowDevicePurchaseRecordDTO = BeanUtil.toBean(flowBaseInfoDTO, FlowDevicePurchaseRecordDTO.class);

//判断是否有id 有id则修改状态为已申请 没id则新增数据切状态是未申请

String id = flowDevicePurchaseRecordDTO.getId();

//创建流程后处理流程数据

List<FlowDevicePurchaseDTO> dtoList = flowDevicePurchaseRecordDTO.getDtoList();

flowDevicePurchaseRecordDTO.setApplyStatus(ApplyState.APPLY);

if (StringUtils.isNotEmpty(id)) {

iFlowDevicePurchaseRecordService.updateFlowDevicePurchaseRecord(flowDevicePurchaseRecordDTO);

//修改采购申请详细数据

iFlowDevicePurchaseService.updateFlowDevicePurchaseBatch(dtoList);

} else {

//新增流水记录

String id1 = iFlowDevicePurchaseRecordService.addFlowDevicePurchaseRecord(flowDevicePurchaseRecordDTO);

//新增详细信息

dtoList.forEach(f -> f.setProcessRecordId(id1));

iFlowDevicePurchaseService.addFlowDevicePurchaseBatch(dtoList);

}

}

@Override

public void processEnd(String processId) {

//流程结束时往采购入库表增加数据

//查询明细数据

// List<FlowDeviceVerificationVO> flowDeviceVerificationVOList = flowDeviceVerificationMapper.selectByProcessId(processId);

// for (FlowDeviceVerificationVO flowDeviceVerificationVO : flowDeviceVerificationVOList) {

// String deviceInfoId = flowDeviceVerificationVO.getDeviceInfoId();

// //检定地点和检定单位都是存储的供应商type=3的id

// String verificationPlace = flowDeviceVerificationVO.getVerificationPlace();

// Date applicationVerificationTime = flowDeviceVerificationVO.getApplicationVerificationTime();

// String verificationStyle = flowDeviceVerificationVO.getVerificationStyle();

// DeviceVerificationRecordDTO deviceVerificationRecordDTO = new DeviceVerificationRecordDTO();

// deviceVerificationRecordDTO.setDeviceInfoId(deviceInfoId);

// deviceVerificationRecordDTO.setVerificationCompany(verificationPlace);

// deviceVerificationRecordDTO.setVerificationDate(applicationVerificationTime);

// deviceVerificationRecordDTO.setVerificationStyle(verificationStyle);

// iDeviceVerificationRecordService.addDeviceVerificationRecord(deviceVerificationRecordDTO);

// }

}

@Override

public void taskComplete(String processId, SysUser sysUser, String taskEventId, String commentMsg) throws ProcessException {

//taskEventId对应的审批节点

//设备管理员审批 device\_approval

if ("device\_approval".equals(taskEventId)) {

//sysUser 审批人

if (null != sysUser) {

String id = sysUser.getId();

String signatureBase64 = sysUser.getSignatureBase64();

flowDevicePurchaseRecordMapper.updateByProcessId(id, commentMsg, signatureBase64, processId);

}

}

}

@Override

public Object getProcessBiz(String processId) {

//流程点详情查看数据

//先查询流水数据

FlowDevicePurchaseRecordVO flowDevicePurchaseRecordVO = flowDevicePurchaseRecordMapper.selectByProcessId(processId);

if (null != flowDevicePurchaseRecordVO) {

String id = flowDevicePurchaseRecordVO.getId();

List<FlowDevicePurchaseVO> flowDevicePurchaseVOList = flowDevicePurchaseMapper.selectByProcessRecordId(id);

flowDevicePurchaseRecordVO.setDtoList(flowDevicePurchaseVOList);

}

return flowDevicePurchaseRecordVO;

}

@Override

public void updateProcessBiz(JSONObject flowBaseInfoUpdateDTO, String processId) throws ProcessException {

//被驳回后 修改申请审批流程的数据

FlowDevicePurchaseRecordDTO flowDevicePurchaseRecordDTO = BeanUtil.toBean(flowBaseInfoUpdateDTO, FlowDevicePurchaseRecordDTO.class);

List<FlowDevicePurchaseDTO> dtoList = flowDevicePurchaseRecordDTO.getDtoList();

//修改检定流水数据

iFlowDevicePurchaseRecordService.updateFlowDevicePurchaseRecord(flowDevicePurchaseRecordDTO);

//修改检定申请详细数据

iFlowDevicePurchaseService.updateFlowDevicePurchaseBatch(dtoList);

}

}

/\*\*

\* 仪器设备检定申请

\*

\* @author wanggang

\* @date 2023/6/7 11:36

\*/

@Service

public class FlowDeviceVerificationFactory implements FlowBaseFactory {

@Resource

IFlowDeviceVerificationService iFlowDeviceVerificationService;

@Resource

IFlowDeviceVerificationRecordService iFlowDeviceVerificationRecordService;

@Resource

FlowDeviceVerificationRecordMapper flowDeviceVerificationRecordMapper;

@Resource

FlowDeviceVerificationMapper flowDeviceVerificationMapper;

@Resource

IDeviceVerificationRecordService iDeviceVerificationRecordService;

@Override

public ProcessKey getProcessKey() {

return ProcessKey.DEVICE\_CHECK\_VALIDATED;

}

@Override

public void createProcess(JSONObject flowBaseInfoDTO, String processId) throws ProcessException {

FlowDeviceVerificationRecordDTO flowDeviceVerificationRecordDTO = BeanUtil.toBean(flowBaseInfoDTO, FlowDeviceVerificationRecordDTO.class);

String id1 = flowDeviceVerificationRecordDTO.getId();

//如果id不为空 说明是再申请(被拒绝)

if (StringUtils.isNotEmpty(id1)) {

//把上一条的流水记录的id记录到新的流水记录中

flowDeviceVerificationRecordDTO.setHistoryId(id1);

}

//新增流水记录

String id = iFlowDeviceVerificationRecordService.addFlowDeviceVerificationRecord(flowDeviceVerificationRecordDTO);

//把流水记录和明细表绑定

//创建流程后处理流程数据

List<FlowDeviceVerificationDTO> dtoList = flowDeviceVerificationRecordDTO.getDtoList();

//新增详细信息

for (FlowDeviceVerificationDTO flowDeviceVerificationDTO : dtoList) {

flowDeviceVerificationDTO.setProcessRecordId(id);

flowDeviceVerificationDTO.setApplyStatus(ApplyState.APPLY);

}

//修改检定申请详细数据

iFlowDeviceVerificationService.updateFlowDeviceVerificationBatch(dtoList);

}

@Override

public void processEnd(String processId) {

//流程结束时往检定记录表增加数据

//查询明细数据

List<FlowDeviceVerificationVO> flowDeviceVerificationVOList = flowDeviceVerificationMapper.selectByProcessId(processId);

for (FlowDeviceVerificationVO flowDeviceVerificationVO : flowDeviceVerificationVOList) {

String deviceInfoId = flowDeviceVerificationVO.getDeviceInfoId();

//检定地点和检定单位都是存储的供应商type=3的id

String verificationPlace = flowDeviceVerificationVO.getVerificationPlace();

Date applicationVerificationTime = flowDeviceVerificationVO.getApplicationVerificationTime();

String verificationStyle = flowDeviceVerificationVO.getVerificationStyle();

DeviceVerificationRecordDTO deviceVerificationRecordDTO = new DeviceVerificationRecordDTO();

deviceVerificationRecordDTO.setDeviceInfoId(deviceInfoId);

deviceVerificationRecordDTO.setVerificationCompany(verificationPlace);

deviceVerificationRecordDTO.setVerificationDate(applicationVerificationTime);

deviceVerificationRecordDTO.setVerificationStyle(verificationStyle);

iDeviceVerificationRecordService.addDeviceVerificationRecord(deviceVerificationRecordDTO);

}

}

@Override

public void taskComplete(String processId, SysUser sysUser, String taskEventId, String commentMsg) throws ProcessException {

//taskEventId对应的审批节点

//设备管理员审批 device\_approval

if ("device\_approval".equals(taskEventId)) {

//sysUser 审批人

if (null != sysUser) {

String id = sysUser.getId();

String signatureBase64 = sysUser.getSignatureBase64();

flowDeviceVerificationRecordMapper.updateByProcessId(id, commentMsg, signatureBase64, processId);

}

}

}

@Override

public Object getProcessBiz(String processId) {

//流程点详情查看数据

//先查询流水数据

FlowDeviceVerificationRecordVO flowDeviceVerificationRecordVO = flowDeviceVerificationRecordMapper.selectByProcessId(processId);

if (null != flowDeviceVerificationRecordVO) {

String id = flowDeviceVerificationRecordVO.getId();

List<FlowDeviceVerificationVO> flowDeviceVerificationVOList = flowDeviceVerificationMapper.selectByProcessRecordId(id);

flowDeviceVerificationRecordVO.setDtoList(flowDeviceVerificationVOList);

}

return flowDeviceVerificationRecordVO;

}

@Override

public void updateProcessBiz(JSONObject flowBaseInfoUpdateDTO, String processId) throws ProcessException {

//被驳回后 修改申请审批流程的数据

// FlowDeviceVerificationRecordDTO flowDeviceVerificationRecordDTO = BeanUtil.toBean(flowBaseInfoUpdateDTO, FlowDeviceVerificationRecordDTO.class);

// List<FlowDeviceVerificationDTO> dtoList = flowDeviceVerificationRecordDTO.getDtoList();

// //修改检定流水数据

// iFlowDeviceVerificationRecordService.updateFlowDeviceVerificationRecord(flowDeviceVerificationRecordDTO);

// //修改检定申请详细数据

// iFlowDeviceVerificationService.updateFlowDeviceVerificationBatch(dtoList);

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/6/1 14:31

\*/

@Component(value = "flowTaskExecutionListener")

@Slf4j

public class FlowTaskExecutionListener implements FlowableEventListener {

@Autowired

FlowBaseHandlerFactory flowBaseHandlerFactory;

@Autowired

IFlowFactory flowFactory;

@Override

public void onEvent(FlowableEvent event) {

SysUser sysUser= TenantContextHolder.getContext().getTenantContextInfo().getSysUser();

if (event.getType().name().equals(FlowableEngineEventType.PROCESS\_CREATED.name())){

//流程开始

}

if (event.getType().name().equals(FlowableEngineEventType.PROCESS\_COMPLETED.name())){

//流程结束

ExecutionEntityImpl executionEntity= (ExecutionEntityImpl) ((FlowableEntityEventImpl) event).getEntity();

FlowBaseFactory flowBaseFactory= flowBaseHandlerFactory.createHandler(executionEntity.getProcessDefinitionKey());

flowBaseFactory.processEnd(executionEntity.getProcessInstanceId());

}

if (event.getType().name().equals(FlowableEngineEventType.TASK\_COMPLETED.name())){

TaskEntityImpl task= (TaskEntityImpl) ((FlowableEntityEventImpl) event).getEntity();

String commentMsg="";

List<Comment> comments=flowFactory.getTaskService().getTaskComments(task.getId());

if (!comments.isEmpty()){

commentMsg = Joiner.on(",").join(comments.stream().map(Comment::getFullMessage).collect(Collectors.toList()));

}

ProcessInstance processInstance=flowFactory.getRuntimeService().createProcessInstanceQuery().processInstanceId(task.getProcessInstanceId()).singleResult();

FlowBaseFactory flowBaseFactory= flowBaseHandlerFactory.createHandler(processInstance.getProcessDefinitionKey());

flowBaseFactory.taskComplete(processInstance.getProcessInstanceId(),sysUser,task.getTaskDefinitionKey(),commentMsg);

}

}

@Override

public boolean isFailOnException() {

return true;

}

@Override

public boolean isFireOnTransactionLifecycleEvent() {

return false;

}

@Override

public String getOnTransaction() {

return null;

}

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_asset\_info\_record(设备资产信息记录表)】的数据库操作Mapper

\* @createDate 2023-05-15 10:43:37

\* @Entity com.cstc.jcgl.domain.device.DeviceAssetInfoRecord

\*/

public interface DeviceAssetInfoRecordMapper extends BaseMapper<DeviceAssetInfoRecord> {

/\*\*

\* 分页查询设备资产信息记录

\*

\* @param page

\* @param dto

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceAssetInfoRecordVO>

\* @author wanggang

\* @date 2023/5/18 9:27

\*/

Page<DeviceAssetInfoRecordVO> selectDeviceAssetInfoRecordPage(@Param("page") Page<DeviceAssetInfoRecordVO> page, @Param("dto") DeviceAssetInfoRecordSelectDTO dto);

/\*\*

\* 根据资产信息ID查询设备资产信息记录

\*

\* @param id 资产信息ID

\* @return DeviceAssetInfoRecordVO

\* @author wanggang

\* @date 2023/5/15 13:50

\*/

DeviceAssetInfoRecordVO selectDeviceAssetInfoRecordById(@Param("id") String id);

/\*\*

\* 根据资产编号查询资产信息表ID

\*

\* @param assetCodeList 资产编号集

\* @return List<DeviceAssetInfoRecordVO>

\* @author wanggang

\* @date 2023/5/16 14:00

\*/

List<DeviceAssetInfoRecordVO> selectIdByAssetCode(@Param("assetCodeList") List<String> assetCodeList);

/\*\*

\* 根据资产编号集删除资产信息

\*

\* @param assetCodeList 删除的资产编号集

\* @return Integer

\* @author wanggang

\* @date 2023/5/17 11:03

\*/

Integer deleteDeviceAssetInfoRecordBatch(@Param("assetCodeList") List<String> assetCodeList);

/\*\*

\* 根据资产编号查询资产编号是否存在

\*

\* @param assetCode 设备编号

\* @return DeviceAssetInfoRecordVO

\* @author wanggang

\* @date 2023/5/18 9:47

\*/

DeviceAssetInfoRecordVO selectDeviceAssetInfoRecordByAssetCode(@Param("assetCode") String assetCode);

/\*\*

\* 根据资产编号修改资产信息

\*

\* @param assetCode 资产编号

\* @param deviceAssetInfoRecord 资产表

\* @return Integer

\* @author wanggang

\* @date 2023/5/17 11:03

\*/

Integer updateDeviceAssetInfoRecord(@Param("deviceAssetInfoRecord") DeviceAssetInfoRecord deviceAssetInfoRecord, @Param("assetCode") String assetCode);

/\*\*

\* 根据设备信息ID逻辑删除资产信息

\*

\* @param deviceInfoId 设备ID

\* @return Integer

\* @author wanggang

\* @date 2023/5/24 9:47

\*/

Integer updateDelFlagByDeviceInfoId(@Param("deviceInfoId") String deviceInfoId);

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_authorization(设备授权表)】的数据库操作Mapper

\* @createDate 2023-06-08 16:12:54

\* @Entity com.cstc.jcgl.domain.device.DeviceAuthorization

\*/

public interface DeviceAuthorizationMapper extends BaseMapper<DeviceAuthorization> {

/\*\*

\* 分页查询设备授权表

\*

\* @param page

\* @param dto

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceAuthorizationVO>

\* @author wanggang

\* @date 2023/6/9 11:27

\*/

Page<DeviceAuthorizationVO> selectDeviceAuthorizationPage(@Param("page") Page<DeviceAuthorizationVO> page, @Param("dto") DeviceAuthorizationSelectDTO dto);

/\*\*

\* 分页查询设备授权表

\*

\* @param id

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceAuthorizationVO>

\* @author wanggang

\* @date 2023/6/9 11:27

\*/

DeviceAuthorizationVO selectDeviceAuthorizationById(@Param("id") String id);

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_borrow\_record(设备借用记录表)】的数据库操作Mapper

\* @createDate 2023-05-22 16:04:16

\* @Entity com.cstc.jcgl.domain.device.DeviceBorrowRecord

\*/

public interface DeviceBorrowRecordMapper extends BaseMapper<DeviceBorrowRecord> {

/\*\*

\* 分页查询设备流转记录

\*

\* @param page

\* @param dto

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceCirculationRecordVO>

\* @author wanggang

\* @date 2023/5/22 14:27

\*/

Page<DeviceBorrowRecordVO> selectDeviceBorrowRecordPage(@Param("page") Page<DeviceBorrowRecordVO> page, @Param("dto") DeviceBorrowRecordSelectDTO dto);

/\*\*

\* 根据设备流转记录ID查询修改回显数据

\*

\* @param id

\* @return com.cstc.jcgl.domain.device.vo.DeviceCirculationRecordVO

\* @author wanggang

\* @date 2023/5/22 11:21

\*/

DeviceBorrowRecordVO selectDeviceBorrowRecordById(@Param("id") String id);

/\*\*

\* 根据设备信息ID逻辑删除设备借用信息

\*

\* @param deviceInfoId 设备ID

\* @return Integer

\* @author wanggang

\* @date 2023/5/24 9:47

\*/

Integer updateDelFlagByDeviceInfoId(@Param("deviceInfoId") String deviceInfoId);

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_circulation\_record(设备流转记录表)】的数据库操作Mapper

\* @createDate 2023-05-10 09:18:40

\* @Entity com.cstc.jcgl.domain.device.DeviceCirculationRecord

\*/

public interface DeviceCirculationRecordMapper extends BaseMapper<DeviceCirculationRecord> {

/\*\*

\* 分页查询设备流转记录

\*

\* @param page

\* @param dto

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceCirculationRecordVO>

\* @author wanggang

\* @date 2023/5/22 14:27

\*/

Page<DeviceCirculationRecordVO> selectDeviceCirculationRecordPage(@Param("page") Page<DeviceVerificationRecordVO> page, @Param("dto") DeviceCirculationRecordSelectDTO dto);

/\*\*

\* 根据设备流转记录ID查询修改回显数据

\*

\* @param id

\* @return com.cstc.jcgl.domain.device.vo.DeviceCirculationRecordVO

\* @author wanggang

\* @date 2023/5/22 11:21

\*/

DeviceCirculationRecordVO selectDeviceCirculationRecordById(@Param("id") String id);

/\*\*

\* 根据设备信息ID逻辑删除设备流转记录信息

\*

\* @param deviceInfoId 设备ID

\* @return Integer

\* @author wanggang

\* @date 2023/5/24 9:47

\*/

Integer updateDelFlagByDeviceInfoId(@Param("deviceInfoId") String deviceInfoId);

}

/\*\*

\* @author Administrator

\* @description 针对表【device\_classify\_info("设备分类信息表)】的数据库操作Mapper

\* @createDate 2023-05-06 16:49:30

\* @Entity com.cstc.jcgl.domain.DeviceClassifyInfo

\*/

public interface DeviceClassifyInfoMapper extends BaseMapper<DeviceClassifyInfo> {

/\*\*

\* 分页查询未删除的设备分类信息，包含了厂家和供应商信息

\*

\* @param page 分页信息

\* @param dto 查询条件

\* @return 分页查询之后的设备分类、厂商、供应商信息

\*/

Page<DeviceClassifyInfoVO> findDeviceClassifyByPage(@Param("page") Page<DeviceClassifyInfoVO> page, @Param("dto") DeviceClassifyInfoSelectDTO dto);

List<DeviceClassifyInfoVO> findDeviceClassifyByPage(@Param("dto") DeviceClassifyInfoSelectDTO dto);

/\*\*

\* 逻辑删除设备分类信息

\* 注意：必须至少有一个主键

\*

\* @param ids 主键数组

\* @return 删除个数

\*/

int logicDeleteDeviceClassifyInfo(String[] ids, @Param("tenantId") String tenantId);

/\*\*

\* 根据设备分类ID查询设备数量

\*

\* @param id 主键ID

\* @return int

\* @author wanggang

\* @date 2023/5/12 11:04

\*/

@TenantIgnore

Integer selectDeviceQuantity(@Param("id") String id);

/\*\*

\* 根据设备分类ID修改设备数量

\* 注意：查询参数和数量必须同时传入才能修改成功

\* @param quantity 数量

\* @param id 主键ID

\* @return int

\* @author wanggang

\* @date 2023/5/12 11:04

\*/

@TenantIgnore

Integer updateDeviceQuantity(@Param("quantity") Integer quantity, @Param("id") String id);

/\*\*

\* 根据设备分类编号集查询设备分类表ID

\*

\* @param deviceClassifyNumberList 设备分类编号集

\* @return int

\* @author wanggang

\* @date 2023/5/16 11:04

\*/

List<DeviceClassifyInfoVO> selectIdByDeviceClassifyNumber(@Param("deviceClassifyNumberList") List<String> deviceClassifyNumberList);

/\*\*

\* 查询某个组织下面的设备分类名称

\* 注意：1、不传租户id查询所有数据的设备分类名称

\*

\* @param tenantId 组织id

\* @return {@link String}

\*/

Set<String> selectDeviceClassifyNames(@Param("tenantId") String tenantId);

/\*\*

\* 根据设备分类名称查询，设备分类信息

\* <p>

\* 提示： 可以自行添加条件，但也要修改对应的调用方

\* </p>

\* <p>

\* 注意：租户id最好传入，否则查询所有的设备分类信息，如果是超级管理员可以查询所有的数据

\* </p>

\*

\* @param deviceClassifyName 设备分类名称

\* @param tenantId 租户id

\* @return 返回 {@link List}， {@link List}的属性为 {@link DeviceClassifyInfoVO}

\*/

List<DeviceClassifyInfoVO> selectDeviceClassifyInfo(@Param("deviceClassifyName") String deviceClassifyName

, @Param("tenantId") String tenantId);

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_info(设备管理表)】的数据库操作Mapper

\* @createDate 2023-05-08 09:29:50

\* @Entity com.cstc.jcgl.domain.device.DeviceInfo

\*/

public interface DeviceInfoMapper extends BaseMapper<DeviceInfo> {

/\*\*

\* 分页查询设备管理

\*

\* @param dto

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceInfoVO>

\* @author wanggang

\* @date 2023/5/8 17:27

\*/

Page<DeviceInfoVO> selectDevicePage(@Param("page") Page<DeviceInfoVO> page, @Param("dto") DeviceInfoSelectDTO dto);

/\*\*

\* 根据设备信息ID查询修改回显数据

\*

\* @param id

\* @return com.cstc.jcgl.domain.device.vo.DeviceInfoVO

\* @author wanggang

\* @date 2023/5/11 14:21

\*/

DeviceInfoVO selectDeviceInfoById(@Param("id") String id);

/\*\*

\* 根据设备编号查询设备信息

\*

\* @param deviceNumberList 设备编号集

\* @return List<DeviceInfoVO>

\* @author wanggang

\* @date 2023/5/17 9:47

\*/

List<DeviceInfoVO> selectDeviceByDeviceNumber(@Param("deviceNumberList") List<String> deviceNumberList);

/\*\*

\*

\* @author wanggang

\* @date 2023/5/17 11:03

\* @param deviceNumberList 删除的设备编号集

\* @return Integer

\*/

Integer deleteDeviceInfoBatch(@Param("deviceNumberList") List<String> deviceNumberList);

}

/\*\*

\* @author Administrator

\* @description 针对表【device\_place(场所信息)】的数据库操作Mapper

\* @createDate 2023-05-23 09:23:24

\*/

public interface DevicePlaceMapper extends BaseMapper<DevicePlace> {

/\*\*

\* 分页查询场所信息

\*

\* @param page 分页信息

\* @param dto 查询条件

\* @return {@link Page}

\*/

Page<DevicePlaceVO> findPlaceByPage(@Param("page") Page<DevicePlaceVO> page, @Param("dto") CommonSelectEntity dto);

List<DevicePlaceVO> findPlaceByPage(@Param("dto") CommonSelectEntity dto);

// 更具主键和租户id逻辑删除

int logicDelete(String[] ids, @Param("tenantId") String tenantId);

// 获取组织下面的最大编号值

char findmMaxCode(@Param("tenantId") String tenantId);

List<Character> selectPlaceCode(@Param("tenantId") String tenantId);

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_repair\_maintenance\_record(设备保养维修记录表)】的数据库操作Mapper

\* @createDate 2023-05-10 10:26:14

\* @Entity com.cstc.jcgl.domain.device.DeviceRepairMaintenanceRecord

\*/

public interface DeviceRepairMaintenanceRecordMapper extends BaseMapper<DeviceRepairMaintenanceRecord> {

/\*\*

\* 分页查询设备保养维修记录表

\*

\* @param page

\* @param dto

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceRepairMaintenanceRecordVO>

\* @author wanggang

\* @date 2023/5/23 14:27

\*/

Page<DeviceRepairMaintenanceRecordVO> selectDeviceRepairMaintenanceRecordPage(@Param("page") Page<DeviceRepairMaintenanceRecordVO> page, @Param("dto") DeviceRepairMaintenanceRecordSelectDTO dto);

/\*\*

\* 根据设备维修保养记录ID查询修改回显数据

\*

\* @param id

\* @return com.cstc.jcgl.domain.device.vo.DeviceRepairMaintenanceRecordVO

\* @author wanggang

\* @date 2023/5/23 14:21

\*/

DeviceRepairMaintenanceRecordVO selectDeviceRepairMaintenanceRecordById(@Param("id") String id);

/\*\*

\* 根据设备信息ID逻辑删除设备保养维修记录信息

\*

\* @param deviceInfoId 设备ID

\* @return Integer

\* @author wanggang

\* @date 2023/5/24 9:47

\*/

Integer updateDelFlagByDeviceInfoId(@Param("deviceInfoId") String deviceInfoId);

}

/\*\*

\* @author Administrator

\* @description 针对表【device\_room（存放室信息）】的数据库操作Mapper

\* @createDate 2023-05-23 09:23:35

\* @Entity com.cstc.jcgl.domain.device.DeviceRoom

\*/

public interface DeviceRoomMapper extends BaseMapper<DeviceRoom> {

/\*\*

\* 分页查询存放室信息

\*

\* @param page 分页信息

\* @param dto 查询条件

\* @return {@link Page}

\*/

Page<DeviceRoomVO> findPlaceByPage(@Param("page") Page<DeviceRoomVO> page, @Param("dto") CommonSelectEntity dto);

List<DeviceRoomVO> findPlaceByPage(@Param("dto") CommonSelectEntity dto);

List<DeviceRoomVO> selectByPlaceId(@Param("tenantId") String tenantId,@Param("placeId") String placeId);

/\*\*

\* 逻辑删除存放室信息

\* 注意：没有参数代表逻辑删除所有的数据，tenantId 必传

\*

\* @param ids 主键数组

\* @param tenantId 租号id

\* @return 删除个数

\*/

int logicDelete(String[] ids, @Param("tenantId") String tenantId);

/\*\*

\* 查询设备存放室的最大编码数

\*

\* @param tenantId 租户id

\* @return {@link int}

\*/

Long findMaxCode(@Param("tenantId") String tenantId);

/\*\*

\* 根据存放室名称查询ID

\*

\* @param roomNameList 存放室名称集

\* @return List<DeviceRoomVO>

\* @author wanggang

\* @date 2023/5/24 15:21

\*/

List<DeviceRoomVO> selectIdByRootName(@Param("roomNameList") List<String> roomNameList);

}

/\*\*

\* @author Administrator

\* @description 针对表【device\_shelves\_info(设备存放货架信息表)】的数据库操作Mapper

\* @createDate 2023-05-16 11:26:31

\*/

public interface DeviceShelvesInfoMapper extends BaseMapper<DeviceShelvesInfo> {

/\*\*

\* 分页查询未删除的货架信息

\*

\* @param page 分页信息

\* @param dto 查询条件

\* @return 分页查询之后的货架信息

\*/

Page<DeviceShelvesInfoVO> findByPage(Page<DeviceShelvesInfoVO> page, @Param("dto") DeviceShelvesInfoDTO dto);

List<DeviceShelvesInfoVO> findByPage(@Param("dto") DeviceShelvesInfoDTO dto);

/\*\*

\* 逻辑删除货架信息

\*

\* @param ids 主键数组

\* @param tenantId 租户号

\* @return 删除个数

\*/

int logicDelete(String[] ids, @Param("tenantId") String tenantId);

Set<String> selectStorageCodeByTenantId(@Param("shelvesCode") String shelvesCode, @Param("tenantId") String tenantId);

/\*\*

\* 根据存放编号查询货架表ID

\*

\* @param storageCodeList 存放编号集

\* @return List<DeviceShelvesInfoVO>

\* @author wanggang

\* @date 2023/5/16 17:21

\*/

List<DeviceShelvesInfoVO> selectIdByShelvesCode(@Param("storageCodeList") List<String> storageCodeList);

/\*\*

\* 查询当前租户下最大编码

\*

\* @param tenantId 租户id

\* @return 最大编码

\*/

Long findMaxCode(@Param("tenantId") String tenantId);

/\*\*

\* 查询当前租户下，指定货架的名称

\*

\* @param tenantId 租户id

\* @param shelvesCode 货架编码

\* @return 货架名称

\*/

String selectShelvesNameByCode(@Param("tenantId") String tenantId, @Param("shelvesCode") String shelvesCode);

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_verification\_record(设备检定记录表)】的数据库操作Mapper

\* @createDate 2023-05-10 14:09:52

\* @Entity com.cstc.jcgl.domain.device.DeviceVerificationRecord

\*/

public interface DeviceVerificationRecordMapper extends BaseMapper<DeviceVerificationRecord> {

/\*\*

\* 分页查询设备检定记录

\*

\* @param page

\* @param dto

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceVerificationRecordVO>

\* @author wanggang

\* @date 2023/5/18 17:27

\*/

Page<DeviceVerificationRecordVO> selectDeviceVerificationRecordPage(@Param("page") Page<DeviceVerificationRecordVO> page, @Param("dto") DeviceVerificationRecordSelectDTO dto);

/\*\*

\* 根据设备检定ID查询编辑回显数据

\*

\* @param id 设备检定记录的Id

\* @return DeviceVerificationRecordVO

\* @author wanggang

\* @date 2023/5/18 16:11

\*/

DeviceVerificationRecordVO selectDeviceVerificationRecordById(@Param("id") String id);

/\*\*

\* 根据设备信息ID逻辑删除设备检定记录信息

\*

\* @param deviceInfoId 设备ID

\* @return Integer

\* @author wanggang

\* @date 2023/5/24 9:47

\*/

Integer updateDelFlagByDeviceInfoId(@Param("deviceInfoId") String deviceInfoId);

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_working\_record(设备运行记录表)】的数据库操作Mapper

\* @createDate 2023-05-09 11:37:02

\* @Entity com.cstc.jcgl.domain.device.DeviceWorkingRecord

\*/

public interface DeviceWorkingRecordMapper extends BaseMapper<DeviceWorkingRecord> {

/\*\*

\* 分页查询设备运行记录

\*

\* @param page

\* @param dto

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceWorkingRecordVO>

\* @author wanggang

\* @date 2023/5/24 17:27

\*/

Page<DeviceWorkingRecordVO> selectDeviceWorkingRecordPage(@Param("page") Page<DeviceWorkingRecordVO> page, @Param("dto") DeviceWorkingRecordSelectDTO dto);

/\*\*

\* 根据设备运行记录ID查询修改回显数据

\*

\* @param id

\* @return com.cstc.jcgl.domain.device.vo.DeviceBorrowRecordVO

\* @author wanggang

\* @date 2023/5/24 17:21

\*/

DeviceWorkingRecordVO selectDeviceWorkingRecordById(@Param("id") String id);

/\*\*

\* 根据设备信息ID逻辑删除设备运行记录信息

\*

\* @param deviceInfoId 设备ID

\* @return Integer

\* @author wanggang

\* @date 2023/5/24 9:47

\*/

Integer updateDelFlagByDeviceInfoId(@Param("deviceInfoId") String deviceInfoId);

}

/\*\*

\* @author Administrator

\* @description 针对表【supplier\_contacts\_info("厂商联系人信息表)】的数据库操作Mapper

\* @createDate 2023-05-08 13:53:52

\* @Entity com.cstc.jcgl.domain.device.SupplierContactsInfo

\*/

public interface SupplierContactsInfoMapper extends BaseMapper<SupplierContactsInfo> {

/\*\*

\* 分页查询未删除的厂商联系人信息

\*

\* @param page 分页信息

\* @param dto 查询条件

\* @return 分页查询之后的厂商联系人信息

\*/

Page<SupplierContactsInfoVO> findByPage(@Param("page") Page<SupplierContactsInfoVO> page, @Param("dto") SupplierContactsInfoSelectDTO dto);

List<SupplierContactsInfoVO> findByPage(@Param("dto") SupplierContactsInfoSelectDTO dto);

/\*\*

\* 查询最新的厂商联系人的数据

\*

\* @param supplierId 厂商表id

\* @return {@link SupplierContactsInfoVO}

\*/

SupplierContactsInfoVO selectContactsInfoLatest(@Param("supplierId") String supplierId);

/\*\*

\* 逻辑删除厂商联系人信息

\*

\* @param ids 主键数组

\* @param supplierIds 厂商主键数组

\* @param tenantId 租户号

\* @return 删除个数

\*/

int logicDelete(String[] ids, String[] supplierIds, @Param("tenantId") String tenantId);

/\*\*

\* 根据厂商id查询联系人姓名

\*

\* @param supplierId 厂商id

\* @param tenantId 租户id

\* @return 联系人姓名

\*/

Set<String> selectContactsName(@Param("supplierId") String supplierId, @Param("tenantId") String tenantId);

}

/\*\*

\* @author Administrator

\* @description 针对表【supplier\_info(厂商信息表)】的数据库操作Mapper

\* @createDate 2023-05-08 13:46:24

\* @Entity com.cstc.jcgl.domain.device.SupplierInfo

\*/

public interface SupplierInfoMapper extends BaseMapper<SupplierInfo> {

/\*\*

\* 分页查询未删除厂商、联系人信息

\*

\* @param page 分页信息

\* @param dto 查询条件

\* @return 分页查询之后的设备分类、厂商、供应商信息

\*/

Page<SupplierInfoVO> findByPage(@Param("page") Page<SupplierInfoVO> page, @Param("dto") SupplierInfoSelectDTO dto);

// 不分页查询

List<SupplierInfoVO> findByPage(@Param("dto") SupplierInfoSelectDTO dto);

/\*\*

\* 分页查询未删除厂商信息

\*

\* @param page 分页信息

\* @param dto 查询条件

\* @return 分页查询之后的设备分类、厂商、供应商信息

\*/

Page<SupplierInfoVO> findSupplierByPage(@Param("page") Page<SupplierInfoVO> page, @Param("dto") SupplierInfoSelectDTO dto);

List<SupplierInfoVO> findSupplierByPage(@Param("dto") SupplierInfoSelectDTO dto);

/\*\*

\* 逻辑删除厂商

\* 注意：没有参数代表逻辑删除所有的数据，tenantId 必传

\*

\* @param ids 主键数组

\* @param tenantId 租号id

\* @return 删除个数

\*/

int logicDelete(String[] ids, @Param("tenantId") String tenantId);

/\*\*

\* 查询租户下面供应商名称

\*

\* @param tenantId 租户id

\* @return 供应商名称

\*/

Set<String> selectSupplierName(@Param("tenantId") String tenantId);

}

/\*\*

\* @author yb

\* @description 针对表【flow\_base\_info(流程基础流水信息表)】的数据库操作Mapper

\* @createDate 2023-05-31 16:36:30

\* @Entity com.cstc.jcgl.domain.FlowBaseInfo

\*/

public interface FlowBaseInfoMapper extends BaseMapper<FlowBaseInfo> {

List<FlowBaseInfo> findCompletedProcessPage(Page<FlowBaseInfo> page,

@Param("sysUserId") String sysUserId,

@Param("processState") String processState,

@Param("processKey") String processKey,

@Param("searchKey")String searchKey,

@Param("startTime") String startTime,

@Param("endTime") String endTime);

@TenantIgnore

List<FlowBaseInfo> findMyTaskPage(Page<FlowBaseInfo> page,

@Param("flowOrgRoleVOS")List<FlowOrgRoleVO> flowOrgRoleVOS,

@Param("sysUserId") String sysUserId,

@Param("searchKey")String searchKey,

@Param("processKey")String processKey,

@Param("processState") String processState,

@Param("startTime") String startTime,

@Param("endTime") String endTime);

}

/\*\*

\* @author wanggang

\* @description 针对表【flow\_device\_purchase(采购申请明细记录)】的数据库操作Mapper

\* @createDate 2023-06-12 15:55:33

\* @Entity com.cstc.jcgl.domain.flow.FlowDevicePurchase

\*/

public interface FlowDevicePurchaseMapper extends BaseMapper<FlowDevicePurchase> {

/\*\*

\* 根据流水记录表ID查询 明细数据

\*

\* @param processRecordId

\* @return List<FlowDevicePurchaseVO>

\* @author wanggang

\* @date 2023/6/13 10:34

\*/

List<FlowDevicePurchaseVO> selectByProcessRecordId(@Param("processRecordId") String processRecordId);

}

/\*\*

\* @author wanggang

\* @description 针对表【flow\_device\_purchase\_record(采购申请流水记录)】的数据库操作Mapper

\* @createDate 2023-06-12 15:55:33

\* @Entity com.cstc.jcgl.domain.flow.FlowDevicePurchaseRecord

\*/

public interface FlowDevicePurchaseRecordMapper extends BaseMapper<FlowDevicePurchaseRecord> {

/\*\*

\* 根据流程ID查询流水表数据

\*

\* @param processId 流程ID

\* @return FlowDevicePurchaseRecordVO

\* @author wanggang

\* @date 2023/6/13 9:47

\*/

FlowDevicePurchaseRecordVO selectByProcessId(@Param("processId") String processId);

/\*\*

\* 修改流水表字段

\*

\* @param approvalPerson 审批人

\* @param reviewComments 审核意见

\* @param approvalSign 签名

\* @param processId 流程ID

\* @return Integer

\* @author wanggang

\* @date 2023/6/13 9:47

\*/

Integer updateByProcessId(@Param("approvalPerson") String approvalPerson, @Param("reviewComments") String reviewComments,

@Param("approvalSign") String approvalSign, @Param("processId") String processId);

}

/\*\*

\* @author wanggang

\* @description 针对表【flow\_device\_verification】的数据库操作Mapper

\* @createDate 2023-06-01 16:10:49

\* @Entity com.cstc.jcgl.domain.flow.FlowDeviceVerification

\*/

public interface FlowDeviceVerificationMapper extends BaseMapper<FlowDeviceVerification> {

/\*\*

\* 分页查询设备检定记录

\*

\* @param page

\* @param dto

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.flow.vo.FlowDeviceVerificationVO>

\* @author wanggang

\* @date 2023/6/7 15:27

\*/

Page<FlowDeviceVerificationVO> selectFlowDeviceVerificationPage(@Param("page") Page<FlowDeviceVerificationVO> page, @Param("dto") FlowDeviceVerificationRecordSelectDTO dto);

/\*\*

\* 根据检定流水记录表ID查询 明细数据

\*

\* @param processRecordId

\* @return List<FlowDeviceVerificationVO>

\* @author wanggang

\* @date 2023/6/7 10:34

\*/

List<FlowDeviceVerificationVO> selectByProcessRecordId(@Param("processRecordId") String processRecordId);

/\*\*

\* 根据流程ID查询 明细数据

\*

\* @param processId

\* @return List<FlowDeviceVerificationVO>

\* @author wanggang

\* @date 2023/6/8 10:34

\*/

List<FlowDeviceVerificationVO> selectByProcessId(@Param("processId") String processId);

}

/\*\*

\* @author wanggang

\* @description 针对表【flow\_device\_verification\_record(检定校准申请流水记录)】的数据库操作Mapper

\* @createDate 2023-06-05 17:00:34

\* @Entity com.cstc.jcgl.domain.flow.FlowDeviceVerificationRecord

\*/

public interface FlowDeviceVerificationRecordMapper extends BaseMapper<FlowDeviceVerificationRecord> {

/\*\*

\* 根据流程ID查询流水表数据

\*

\* @param processId 流程ID

\* @return Integer

\* @author wanggang

\* @date 2023/6/7 9:47

\*/

FlowDeviceVerificationRecordVO selectByProcessId(@Param("processId") String processId);

/\*\*

\* 修改流水表字段

\*

\* @param approvalPerson 审批人

\* @param reviewComments 审核意见

\* @param approvalSign 签名

\* @param processId 流程ID

\* @return Integer

\* @author wanggang

\* @date 2023/6/7 9:47

\*/

Integer updateByProcessId(@Param("approvalPerson") String approvalPerson, @Param("reviewComments") String reviewComments,

@Param("approvalSign") String approvalSign, @Param("processId") String processId);

}

/\*\*

\* @author yb

\* @description 针对表【file\_list】的数据库操作Mapper

\* @createDate 2023-03-27 15:55:10

\* @Entity com.cstc.jcgl.domain.FileList

\*/

public interface FileAttachMapper extends BaseMapper<FileAttach> {

/\*\*

\* 根据文件id集查询文件信息

\* @author wanggang

\* @date 2023/5/29 9:25

\* @param fileIdList

\* @return List<FileAttachVO>

\*/

List<FileAttachVO> selectFileByFileId(@Param("fileIdList") List<String> fileIdList);

}

/\*\*

\* @author yb

\* @description 针对表【sys\_dict\_category】的数据库操作Mapper

\* @createDate 2023-05-15 15:51:38

\* @Entity com.cstc.jcgl.domain.SysDictCategory

\*/

public interface SysDictCategoryMapper extends BaseMapper<SysDictCategory> {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_dict】的数据库操作Mapper

\* @createDate 2023-02-15 09:17:11

\* @Entity com.cstc.jcglbusi.domain.SysDict

\*/

public interface SysDictMapper extends BaseMapper<SysDict> {

/\*\*

\* 查询数据字典

\* @param type 数据字典类型 不传为全查询

\* @return List<SysDictVO>

\* @author wanggang

\* @date 2023/5/16 14:13

\*/

List<SysDictVO> selectDict(@Param("type") String type);

List<SysDict> findListAll(@Param("tenantId") String tenantId);

}

/\*\*

\* @author yb

\* @description 针对表【sys\_flow\_role(系统流程角色)】的数据库操作Mapper

\* @createDate 2023-05-30 13:39:26

\* @Entity com.cstc.jcgl.domain.SysFlowRole

\*/

public interface SysFlowRoleMapper extends BaseMapper<SysFlowRole> {

List<SysUser> findSysUserMainByCodeOrg(@Param("flowRoleCode") String flowRoleCode,@Param("orgIds") List<String> upDownOrgIds);

}

/\*\*

\* @author yb

\* @description 针对表【sys\_log】的数据库操作Mapper

\* @createDate 2023-02-10 10:17:11

\* @Entity com.cstc.jcglbusi.domain.SysLog

\*/

public interface SysLogMapper extends BaseMapper<SysLog> {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_menu(菜单)】的数据库操作Mapper

\* @createDate 2023-02-10 10:17:11

\* @Entity com.cstc.jcglbusi.domain.SysMenu

\*/

public interface SysMenuMapper extends BaseMapper<SysMenu> {

List<String> findAuthMenuIdsByUserId(@Param("userId") String userId);

}

/\*\*

\* @author yb

\* @description 针对表【sys\_org】的数据库操作Mapper

\* @createDate 2023-02-22 16:47:13

\* @Entity com.cstc.auth.domain.SysOrg

\*/

public interface SysOrgMapper extends BaseMapper<SysOrg> {

List<String> selectOrgIdByPId(@Param("sysOrgId") String sysOrgId);

/\*\*

\* 根据组织架构名查询组织架构ID

\* @author wanggang

\* @date 2023/5/16 12:04

\* @param sysOrgNameList 组织架构名集合

\* @return List<SysOrgVO>

\*/

List<SysOrgVO> selectOrgByName(@Param("sysOrgNameList") List<String> sysOrgNameList);

}

/\*\*

\* @author yb

\* @description 针对表【sys\_post】的数据库操作Mapper

\* @createDate 2023-02-23 17:08:24

\* @Entity com.cstc.auth.domain.SysPost

\*/

public interface SysPostMapper extends BaseMapper<SysPost> {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_role(角色)】的数据库操作Mapper

\* @createDate 2023-02-10 10:17:11

\* @Entity com.cstc.jcglbusi.domain.SysRole

\*/

public interface SysRoleMapper extends BaseMapper<SysRole> {

List<String> findUsernameByRoleId(@Param("roleId") String roleId);

}

/\*\*

\* @author yb

\* @description 针对表【sys\_role\_menu】的数据库操作Mapper

\* @createDate 2023-02-10 10:17:11

\* @Entity com.cstc.jcglbusi.domain.SysRoleMenu

\*/

public interface SysRoleMenuMapper extends BaseMapper<SysRoleMenu> {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_tenant(租户信息表)】的数据库操作Mapper

\* @createDate 2023-05-09 17:17:23

\* @Entity com.cstc.jcgl.domain.SysTenant

\*/

public interface SysTenantMapper extends BaseMapper<SysTenant> {

/\*\*

\* 分页查询

\* @param sysTenantVOPage

\* @param tenantName

\* @return

\*/

List<SysTenantVO> findPage(Page<SysTenantVO> sysTenantVOPage, @Param("tenantName") String tenantName);

}

/\*\*

\* @author yb

\* @description 针对表【sys\_tenant\_pkg(租户套餐表)】的数据库操作Mapper

\* @createDate 2023-05-09 17:17:23

\* @Entity com.cstc.jcgl.domain.SysTenantPkg

\*/

public interface SysTenantPkgMapper extends BaseMapper<SysTenantPkg> {

/\*\*

\* 分页查询

\* @param sysTenantPkgVOPage

\* @param pkgName

\* @return

\*/

List<SysTenantPkgVO> findPage(Page<SysTenantPkgVO> sysTenantPkgVOPage, @Param("pkgName") String pkgName);

List<String> findMenuIdsBySys(@Param("sysTenantPkgId") String sysTenantPkgId, @Param("menuSys") String menuSys);

}

/\*\*

\* @author yb

\* @description 针对表【sys\_tenant\_pkg\_menu(套餐菜单)】的数据库操作Mapper

\* @createDate 2023-05-09 17:17:23

\* @Entity com.cstc.jcgl.domain.SysTenantPkgMenu

\*/

@Repository

public interface SysTenantPkgMenuMapper extends BaseMapper<SysTenantPkgMenu> {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_user\_flow\_role(用户流程角色)】的数据库操作Mapper

\* @createDate 2023-05-30 13:39:43

\* @Entity com.cstc.jcgl.domain.SysUserFlowRole

\*/

public interface SysUserFlowRoleMapper extends BaseMapper<SysUserFlowRole> {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_user(系统用户)】的数据库操作Mapper

\* @createDate 2023-02-10 10:17:11

\* @Entity com.cstc.jcglbusi.domain.SysUser

\*/

public interface SysUserMapper extends BaseMapper<SysUser> {

/\*\*

\* 分页查询用户信息

\*

\* @param sysUserVOPage

\* @param sysOrgId

\* @param empName

\* @param mobile

\* @return

\*/

Page<SysUserVO> findUserPage(Page<SysUserVO> sysUserVOPage,

@Param("sysOrgId") String sysOrgId,

@Param("empName") String empName,

@Param("mobile") String mobile);

int updateEnabled(@Param("sysUserId") String sysUserId,

@Param("enabled") int enabled);

int updateEmpStatus(@Param("sysUserId") String sysUserId, @Param("status") int status);

SysUserVO findUserVOByUserId(@Param("id") String id);

/\*\*

\* 查询用户菜单编码

\*

\* @param username

\* @return

\*/

List<String> findMenuCodesByUsername(@Param("username") String username);

/\*\*

\* 查询用户角色编码

\*

\* @param username

\* @return

\*/

List<String> findRoleCodeByUsername(@Param("username") String username);

/\*\*

\* 查询当前组织及其以下组织

\*

\* @param sysOrgId

\* @return

\*/

List<String> findAuthSysOrgIds(@Param("sysOrgId") String sysOrgId);

String findTenantId(@Param("userId")String userId);

long setStatusByTenantId(@Param("tenantId")String tenantId, @Param("enabled")String enabled);

/\*\*

\* 根据员工名字查询对应的ID

\* @author wanggang

\* @date 2023/5/16 13:55

\* @param personLiableNameList 员工名字集

\* @return List<SysUserVO>

\*/

List<SysUserVO> selectIdByUserName(@Param("personLiableNameList") List<String> personLiableNameList);

/\*\*

\* 根据员工ID查询对应的名字

\* @author wanggang

\* @date 2023/5/23 13:55

\* @param idList 员工Id集

\* @return List<SysUserVO>

\*/

List<SysUserVO> selectNameByUserId(@Param("idList") List<String> idList);

List<SysFlowRole> findFlowRoleCodes(@Param("sysUserId") String sysUserId);

List<SysRole> findRolesByUserId(@Param("sysUserId")String sysUserId);

}

/\*\*

\* @author yb

\* @description 针对表【sys\_user\_org(用户组织)】的数据库操作Mapper

\* @createDate 2023-05-31 10:45:14

\* @Entity com.cstc.jcgl.domain.SysUserOrg

\*/

public interface SysUserOrgMapper extends BaseMapper<SysUserOrg> {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_user\_role】的数据库操作Mapper

\* @createDate 2023-02-10 10:17:11

\* @Entity com.cstc.jcglbusi.domain.SysUserRole

\*/

public interface SysUserRoleMapper extends BaseMapper<SysUserRole> {

}

/\*\*

\* 代理类

\*

\* @author wanggang

\* @date 2023/4/12 13:54

\*/

public class ManagerTaskHandler implements TaskListener {

@Override

public void notify(DelegateTask delegateTask) {

//主管审批流程创建后 指定主管审批流程的候选组

// delegateTask.addCandidateGroup("manager");

JcglLog.info("监听器触发:主管审批流程");

}

}

/\*\*

\* 代理类

\*

\* @author wanggang

\* @date 2023/4/12 13:54

\*/

public class PersonnelMattersTaskHandler implements TaskListener {

@Override

public void notify(DelegateTask delegateTask) {

//人事审批流程创建后 指定人事审批流程的候选组

// delegateTask.addCandidateGroup("personnelMatters");

JcglLog.info("监听器触发:人事审批流程");

}

}

/\*\*

\* 代理类

\*

\* @author wanggang

\* @date 2023/4/12 13:54

\*/

public class StaffTaskHandler implements TaskListener {

@Override

public void notify(DelegateTask delegateTask) {

//主管审批流程创建后 指定主管审批流程的候选组

// delegateTask.addCandidateGroup("staffTask");

JcglLog.info("监听器触发:员工提交流程");

}

}

/\*\*

\* @author Administrator

\* @description 针对表【device\_classify\_info("设备分类信息表)】的数据库操作Service

\* @createDate 2023-05-06 16:49:30

\*/

public interface DeviceClassifyInfoService extends IService<DeviceClassifyInfo> {

/\*\*

\* 分页查询设备分类信息

\*

\* @param dto 查询参数

\* @param isPage 是否分页

\* @return 分页数据

\*/

Page<DeviceClassifyInfoVO> findByPage(DeviceClassifyInfoSelectDTO dto, boolean isPage);

/\*\*

\* 根据id查询设备分类数据

\* 如果传入的数据为空，那么返回的数据也是空；数据不存在返回的也是空

\*

\* @param id 主键

\* @return {@link DeviceClassifyInfoVO}

\*/

DeviceClassifyInfoVO selectById(@Nullable String id);

/\*\*

\* 查询设备分类名称数据

\*

\* @param tenantId 租户id

\* @return 设备分类名称

\*/

Set<String> selectDeviceClassifyNames(@Nullable String tenantId);

/\*\*

\* 逻辑删除设备信息

\*

\* @param id 主键数组

\* @return 返回逻辑删除数量

\*/

int logicDelete(String[] id);

/\*\*

\* 保存设备分类信息

\*

\* @param add

\* @return

\*/

@Nullable

boolean add(DeviceClassifyInfoDTO add);

/\*\*

\* 更新设备分类信息

\*

\* @param update

\* @return

\*/

@Nullable

boolean update(DeviceClassifyInfoDTO update);

/\*\*

\* 模板类导入设备分类数据

\*

\* @param data excel数据

\* @param isOverride 是否覆盖

\* @param isFirstDelete 是否删除再插入

\*/

void importData(List<DeviceClassifyInfoVO> data, Boolean isOverride, Boolean isFirstDelete);

/\*\*

\* 导出设备分类数据

\*

\* @param dto 查询参数

\* @param response resp

\*/

void exportData(DeviceClassifyInfoSelectDTO dto, HttpServletResponse response);

}

/\*\*

\* @author Administrator

\* @description 针对表【device\_place(场所信息)】的数据库操作Service

\* @createDate 2023-05-23 09:23:24

\*/

public interface DevicePlaceService extends IService<DevicePlace> {

/\*\*

\* 分页查询设备场所信息

\*

\* @param dto 查询参数

\* @param isPage 是否分页

\* @return 分页数据

\*/

Page<DevicePlaceVO> findByPage(CommonSelectEntity dto, boolean isPage);

/\*\*

\* 根据主键查询场所信息

\*

\* @param id 主键

\* @return 场所信息

\*/

@Nullable

DevicePlaceVO selectById(String id);

/\*\*

\* 逻辑删除场所信息

\*

\* @param ids 主键数组

\* @param tenantId 租户号 ,默认当前用户下面的租户号

\* @return 返回逻辑删除数量

\*/

int logicDeleteByIds(String[] ids, String tenantId);

/\*\*

\* 根据\*二级组织机构生成场所信息生成场所

\* 实现幂等性。已经存在的场所不处理，新增的场所添加进去，修改名称的场所优化名称

\*/

void add();

/\*\*

\* 更新设备场所信息

\*

\* @param update 参数

\* @return 布尔值

\*/

boolean update(@Nullable DevicePlaceDTO update);

List<DevicePlaceTreeVO> treePlaceAndRoom();

List<Map<String, Object>> treePlaceAndRoomAndShelves();

/\*\*

\* 查询剩余没有使用的场所编码

\*

\* @return {@link List}

\*/

List<Character> selectSurplusPlaceCode();

}

/\*\*

\* @author Administrator

\* @description 针对表【device\_room(存放室信息)】的数据库操作Service

\* @createDate 2023-05-23 09:23:35

\*/

public interface DeviceRoomService extends IService<DeviceRoom> {

/\*\*

\* 分页查询设备存放室信息

\*

\* @param dto 查询参数

\* @param isPage 是否分页

\* @return 分页数据

\*/

Page<DeviceRoomVO> findByPage(CommonSelectEntity dto, boolean isPage);

/\*\*

\* 根据主键查询存放室信息

\*

\* @param id 主键

\* @return 存放室信息

\*/

@Nullable

DeviceRoomVO selectById(String id);

List<DeviceRoomVO> selectByPlaceId(String placeId);

/\*\*

\* 逻辑删除存放室信息

\*

\* @param ids 主键数组

\* @param tenantId 租户号 ,默认当前用户下面的租户号

\* @return 返回逻辑删除数量

\*/

int logicDeleteByIds(String[] ids, String tenantId);

/\*\*

\* 根据\*二级组织机构生成存放室信息生成场所

\* 实现幂等性。已经存在的场所不处理，新增的场所添加进去，修改名称的场所优化名称

\*

\* @return 布尔值

\*/

boolean add(@Nullable DeviceRoomDTO add);

/\*\*

\* 更新设备存放室信息

\*

\* @param update 参数

\* @return 布尔值

\*/

boolean update(@Nullable DeviceRoomDTO update);

}

/\*\*

\* @author Administrator

\* @description 针对表【device\_shelves\_info(设备存放货架信息表)】的数据库操作Service

\* @createDate 2023-05-16 11:26:31

\*/

public interface DeviceShelvesInfoService extends IService<DeviceShelvesInfo> {

/\*\*

\* 分页查询设备分类信息

\*

\* @param dto 查询参数

\* @param isPage 是否分页

\* @return 分页数据

\*/

Page<DeviceShelvesInfoVO> findByPage(DeviceShelvesInfoDTO dto, boolean isPage);

/\*\*

\* 根据主键查询联系人数据

\*

\* @param id 主键

\* @return 联系人数据

\*/

@Nullable

DeviceShelvesInfoVO selectById(String id);

/\*\*

\* 查询房间里面的货架

\*

\* @param roomId 房间id

\* @return 货架信息

\*/

List<DeviceShelvesInfo> selectByRoomId(String roomId);

/\*\*

\* 逻辑删除设备信息

\* 可以根据主键 或者 厂商主键 （二选一） 和 租户号删除 联系人信息

\*

\* @param ids 主键数组

\* @param tenantId 租户号 ,默认当前用户下面的租户号

\* @return 返回逻辑删除数量

\*/

int logicDeleteByIds(String[] ids, String tenantId);

/\*\*

\* 保存设备分类信息

\*

\* @param add 参数

\* @return 布尔值

\*/

boolean add(@Nullable DeviceShelvesAddInfoDTO add);

/\*\*

\* 更新设备分类信息

\*

\* @param update 参数

\* @return 布尔值

\*/

boolean update(@Nullable DeviceShelvesInfoUpdateDTO update);

/\*\*

\* 根据货架id查询货架名称

\*

\* @param shelvesCode 货架编号

\* @return 货架名称

\*/

String selectShelvesNameByCode(String shelvesCode);

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_asset\_info\_record(设备资产信息记录表)】的数据库操作Service

\* @createDate 2023-05-15 10:43:37

\*/

public interface IDeviceAssetInfoRecordService extends IService<DeviceAssetInfoRecord> {

/\*\*

\* 分页查询设备资产信息记录

\*

\* @param deviceAssetInfoRecordSelectDTO

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceAssetInfoRecordVO>

\* @author wanggang

\* @date 2023/5/18 9:27

\*/

Page<DeviceAssetInfoRecordVO> selectDeviceAssetInfoRecordPage(DeviceAssetInfoRecordSelectDTO deviceAssetInfoRecordSelectDTO);

/\*\*

\* 根据设备信息ID查询设备资产信息记录

\*

\* @param deviceInfoId 设备信息表ID

\* @return com.cstc.jcgl.domain.device.vo.DeviceAssetInfoRecordVO

\* @author wanggang

\* @date 2023/5/15 11:40

\*/

DeviceAssetInfoRecordVO selectDeviceAssetInfoRecordById(String deviceInfoId);

/\*\*

\* 新增设备资产信息记录

\*

\* @param deviceAssetInfoRecordDTO 设备资产信息记录表

\* @return void

\* @author wanggang

\* @date 2023/5/15 10:30

\*/

Boolean addDeviceAssetInfoRecord(DeviceAssetInfoRecordDTO deviceAssetInfoRecordDTO);

/\*\*

\* 修改设备资产信息记录

\*

\* @param deviceAssetInfoRecordDTO 设备资产信息记录表

\* @return void

\* @author wanggang

\* @date 2023/5/15 10:30

\*/

Boolean updateDeviceAssetInfoRecord(DeviceAssetInfoRecordDTO deviceAssetInfoRecordDTO);

/\*\*

\* 逻辑删除设备资产信息记录

\*

\* @param id 设备资产信息Id

\* @return void

\* @author wanggang

\* @date 2023/5/15 11:50

\*/

Boolean deleteDeviceAssetInfoRecord(String id);

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_authorization(设备授权表)】的数据库操作Service

\* @createDate 2023-06-08 16:12:54

\*/

public interface IDeviceAuthorizationService extends IService<DeviceAuthorization> {

/\*\*

\* 分页查询设备授权

\*

\* @param deviceAuthorizationSelectDTO

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceAuthorizationVO>

\* @author wanggang

\* @date 2023/6/9 15:27

\*/

Page<DeviceAuthorizationVO> selectDeviceAuthorizationPage(DeviceAuthorizationSelectDTO deviceAuthorizationSelectDTO);

/\*\*

\* 根据设备授权表ID查询修改回显数据

\*

\* @param id

\* @return com.cstc.jcgl.domain.device.vo.DeviceAuthorizationVO

\* @author wanggang

\* @date 2023/6/9 10:21

\*/

DeviceAuthorizationVO selectDeviceAuthorizationById(String id);

/\*\*

\* 批量新增设备授权记录

\*

\* @param deviceAuthorizationDTOList

\* @return Boolean

\* @author wanggang

\* @date 2023/6/8 16:30

\*/

Boolean addDeviceAuthorizationBatch(List<DeviceAuthorizationDTO> deviceAuthorizationDTOList);

/\*\*

\* 修改设备授权记录

\*

\* @param deviceAuthorizationDTO

\* @return Boolean

\* @author wanggang

\* @date 2023/6/8 16:30

\*/

Boolean updateDeviceAuthorization(DeviceAuthorizationDTO deviceAuthorizationDTO);

/\*\*

\* 逻辑删除设备授权记录

\*

\* @param id

\* @return Boolean

\* @author wanggang

\* @date 2023/6/9 11:30

\*/

Boolean deleteDeviceAuthorization(String id);

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_borrow\_record(设备借用记录表)】的数据库操作Service

\* @createDate 2023-05-22 16:04:16

\*/

public interface IDeviceBorrowRecordService extends IService<DeviceBorrowRecord> {

/\*\*

\* 分页查询设备借用记录

\*

\* @param deviceBorrowRecordSelectDTO 设备借用分页查询实体类

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceBorrowRecordVO>

\* @author wanggang

\* @date 2023/5/22 13:47

\*/

Page<DeviceBorrowRecordVO> selectDeviceBorrowRecordPage(DeviceBorrowRecordSelectDTO deviceBorrowRecordSelectDTO);

/\*\*

\* 根据设备借用记录ID查询修改回显数据

\*

\* @param id 设备借用表ID

\* @return com.cstc.jcgl.domain.device.vo.DeviceBorrowRecordVO

\* @author wanggang

\* @date 2023/5/22 11:21

\*/

DeviceBorrowRecordVO selectDeviceBorrowRecordById(String id);

/\*\*

\* 新增设备借用记录

\*

\* @param deviceBorrowRecordDTO 设备借用实体类

\* @return Boolean

\* @author wanggang

\* @date 2023/5/22 16:30

\*/

Boolean addDeviceBorrowRecord(DeviceBorrowRecordDTO deviceBorrowRecordDTO);

/\*\*

\* 修改设备借用记录

\*

\* @param deviceBorrowRecordDTO 设备借用实体类

\* @return Boolean

\* @author wanggang

\* @date 2023/5/22 16:30

\*/

Boolean updateDeviceBorrowRecord(DeviceBorrowRecordDTO deviceBorrowRecordDTO);

/\*\*

\* 逻辑删除设备借用记录

\*

\* @param id 设备借用表ID

\* @return Boolean

\* @author wanggang

\* @date 2023/5/22 16:30

\*/

Boolean deleteDeviceBorrowRecord(String id);

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_circulation\_record(设备流转记录表)】的数据库操作Service

\* @createDate 2023-05-10 09:18:40

\*/

public interface IDeviceCirculationRecordService extends IService<DeviceCirculationRecord> {

/\*\*

\* 分页查询设备流转记录

\*

\* @param deviceCirculationRecordSelectDTO

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceCirculationRecordVO>

\* @author wanggang

\* @date 2023/5/10 15:27

\*/

Page<DeviceCirculationRecordVO> selectDeviceCirculationRecordPage(DeviceCirculationRecordSelectDTO deviceCirculationRecordSelectDTO);

/\*\*

\* 根据设备流转记录ID查询修改回显数据

\*

\* @param id

\* @return com.cstc.jcgl.domain.device.vo.DeviceCirculationRecordVO

\* @author wanggang

\* @date 2023/5/22 11:21

\*/

DeviceCirculationRecordVO selectDeviceCirculationRecordById(String id);

/\*\*

\* 新增设备流转记录

\*

\* @param deviceCirculationRecordDTO

\* @return Boolean

\* @author wanggang

\* @date 2023/5/10 11:30

\*/

Boolean addDeviceCirculationRecord(DeviceCirculationRecordDTO deviceCirculationRecordDTO);

/\*\*

\* 修改设备流转记录

\*

\* @param deviceCirculationRecordDTO

\* @return Boolean

\* @author wanggang

\* @date 2023/5/10 11:30

\*/

Boolean updateDeviceCirculationRecord(DeviceCirculationRecordDTO deviceCirculationRecordDTO);

/\*\*

\* 逻辑删除设备流转记录

\*

\* @param id

\* @return Boolean

\* @author wanggang

\* @date 2023/5/10 11:30

\*/

Boolean deleteDeviceCirculationRecord(String id);

}

/\*\*

\* 针对表【device\_info(设备管理表)】的数据库操作Service

\*

\* @author wanggang

\* @date 2023/5/8 10:41

\*/

public interface IDeviceInfoService extends IService<DeviceInfo> {

/\*\*

\* 新增设备信息

\*

\* @param deviceInfoDTO 设备信息DTO

\* @return void

\* @author wanggang

\* @date 2023/5/8 11:30

\*/

Boolean addDeviceInfo(DeviceInfoDTO deviceInfoDTO);

/\*\*

\* 根据设备信息ID查询修改回显数据

\*

\* @param id 设备信息ID

\* @return com.cstc.jcgl.domain.device.vo.DeviceInfoVO

\* @author wanggang

\* @date 2023/5/11 14:21

\*/

DeviceInfoVO selectDeviceInfoById(String id);

/\*\*

\* 修改设备信息

\*

\* @param deviceInfoDTO 设备信息DTO

\* @return void

\* @author wanggang

\* @date 2023/5/8 11:30

\*/

Boolean updateDeviceInfo(DeviceInfoDTO deviceInfoDTO);

/\*\*

\* 逻辑删除设备信息

\*

\* @param id 设备信息ID

\* @return void

\* @author wanggang

\* @date 2023/5/8 11:30

\*/

Boolean deleteDeviceInfo(String id);

/\*\*

\* 分页查询设备管理

\*

\* @param deviceInfoSelectDTO 分页查询DTO

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceInfoVO>

\* @author wanggang

\* @date 2023/5/8 17:27

\*/

Page<DeviceInfoVO> selectDevicePage(DeviceInfoSelectDTO deviceInfoSelectDTO);

/\*\*

\* 导入设备信息数据

\*

\* @param file 导入文件

\* @param isOverride 是否覆盖 true是 false 否

\* @return java.lang.Boolean

\* @author wanggang

\* @date 2023/5/15 17:15

\*/

void importDeviceInfo(MultipartFile file, Boolean isOverride);

/\*\*

\* 导出设备信息数据

\*

\* @param deviceInfoSelectDTO 设备导出查询实体

\* @author wanggang

\* @date 2023/5/15 17:15

\*/

void exportDeviceInfo(DeviceInfoSelectDTO deviceInfoSelectDTO,HttpServletResponse response);

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_repair\_maintenance\_record(设备保养维修记录表)】的数据库操作Service

\* @createDate 2023-05-10 10:26:14

\*/

public interface IDeviceRepairMaintenanceRecordService extends IService<DeviceRepairMaintenanceRecord> {

/\*\*

\* 分页查询设备保养维修记录

\*

\* @param deviceRepairMaintenanceRecordSelectDTO 设备保养维修记录查询DTO

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceCirculationRecordVO>

\* @author wanggang

\* @date 2023/5/10 15:27

\*/

Page<DeviceRepairMaintenanceRecordVO> selectDeviceRepairMaintenanceRecordPage(DeviceRepairMaintenanceRecordSelectDTO deviceRepairMaintenanceRecordSelectDTO);

/\*\*

\* 根据设备维修保养记录ID查询修改回显数据

\*

\* @param id

\* @return com.cstc.jcgl.domain.device.vo.DeviceRepairMaintenanceRecordVO

\* @author wanggang

\* @date 2023/5/23 14:21

\*/

DeviceRepairMaintenanceRecordVO selectDeviceRepairMaintenanceRecordById(String id);

/\*\*

\* 新增设备保养维修记录

\*

\* @param deviceRepairMaintenanceRecordDTO 设备保养维修记录DTO

\* @return void

\* @author wanggang

\* @date 2023/5/10 11:30

\*/

Boolean addDeviceRepairMaintenanceRecord(DeviceRepairMaintenanceRecordDTO deviceRepairMaintenanceRecordDTO);

/\*\*

\* 修改设备保养维修记录

\*

\* @param deviceRepairMaintenanceRecordDTO 设备保养维修记录DTO

\* @return void

\* @author wanggang

\* @date 2023/5/10 11:30

\*/

Boolean updateDeviceRepairMaintenanceRecord(DeviceRepairMaintenanceRecordDTO deviceRepairMaintenanceRecordDTO);

/\*\*

\* 逻辑删除设备保养维修记录

\*

\* @param id 设备保养维修记录表ID

\* @return void

\* @author wanggang

\* @date 2023/5/10 11:30

\*/

Boolean deleteDeviceRepairMaintenanceRecord(String id);

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_verification\_record(设备检定记录表)】的数据库操作Service

\* @createDate 2023-05-10 14:09:52

\*/

public interface IDeviceVerificationRecordService extends IService<DeviceVerificationRecord> {

/\*\*

\* 分页查询设备检定记录

\*

\* @param deviceVerificationRecordSelectDTO

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceVerificationRecordVO>

\* @author wanggang

\* @date 2023/5/10 15:27

\*/

Page<DeviceVerificationRecordVO> selectDeviceVerificationRecordPage(DeviceVerificationRecordSelectDTO deviceVerificationRecordSelectDTO);

/\*\*

\* 根据设备检定记录ID查询修改回显数据

\*

\* @param id 设备检定记录ID

\* @return com.cstc.jcgl.domain.device.vo.DeviceVerificationRecordVO

\* @author wanggang

\* @date 2023/5/18 14:21

\*/

DeviceVerificationRecordVO selectDeviceVerificationRecordById(String id);

/\*\*

\* 新增设备检定记录

\*

\* @param deviceVerificationRecordDTO

\* @return void

\* @author wanggang

\* @date 2023/5/10 14:30

\*/

Boolean addDeviceVerificationRecord(DeviceVerificationRecordDTO deviceVerificationRecordDTO);

/\*\*

\* 修改设备检定记录

\*

\* @param deviceVerificationRecordDTO

\* @return void

\* @author wanggang

\* @date 2023/5/10 14:30

\*/

Boolean updateDeviceVerificationRecord(DeviceVerificationRecordDTO deviceVerificationRecordDTO);

/\*\*

\* 逻辑删除设备检定记录

\*

\* @param id

\* @return void

\* @author wanggang

\* @date 2023/5/10 14:30

\*/

Boolean deleteDeviceVerificationRecord(String id);

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_working\_record(设备运行记录表)】的数据库操作Service

\* @createDate 2023-05-09 11:37:02

\*/

public interface IDeviceWorkingRecordService extends IService<DeviceWorkingRecord> {

/\*\*

\* 分页查询设备运行记录

\*

\* @param deviceWorkingRecordSelectDTO

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceWorkingRecordVO>

\* @author wanggang

\* @date 2023/5/9 15:27

\*/

Page<DeviceWorkingRecordVO> selectDeviceWorkingRecordPage(DeviceWorkingRecordSelectDTO deviceWorkingRecordSelectDTO);

/\*\*

\* 根据设备运行记录ID查询修改回显数据

\*

\* @param id

\* @return com.cstc.jcgl.domain.device.vo.DeviceBorrowRecordVO

\* @author wanggang

\* @date 2023/5/24 17:21

\*/

DeviceWorkingRecordVO selectDeviceWorkingRecordById(String id);

/\*\*

\* 新增设备运行记录

\*

\* @param deviceWorkingRecordDTO

\* @return void

\* @author wanggang

\* @date 2023/5/9 11:30

\*/

Boolean addDeviceWorkingRecord(DeviceWorkingRecordDTO deviceWorkingRecordDTO);

/\*\*

\* 修改设备运行记录

\*

\* @param deviceWorkingRecordDTO

\* @return void

\* @author wanggang

\* @date 2023/5/9 11:30

\*/

Boolean updateDeviceWorkingRecord(DeviceWorkingRecordDTO deviceWorkingRecordDTO);

/\*\*

\* 逻辑删除设备运行记录

\*

\* @param id

\* @return void

\* @author wanggang

\* @date 2023/5/9 11:30

\*/

Boolean deleteDeviceWorkingRecord(String id);

}

/\*\*

\* @author Administrator

\* @description 针对表【supplier\_contacts\_info(厂商联系人信息表)】的数据库操作Service

\* @createDate 2023-05-09 10:27:13

\*/

public interface SupplierContactsInfoService extends IService<SupplierContactsInfo> {

/\*\*

\* 分页查询设备分类信息

\*

\* @param dto 查询参数

\* @param isPage 是否分页

\* @return 分页数据

\*/

Page<SupplierContactsInfoVO> findByPage(SupplierContactsInfoSelectDTO dto, boolean isPage);

/\*\*

\* 根据主键查询联系人数据

\*

\* @param id 主键

\* @return 联系人数据

\*/

@Nullable

SupplierContactsInfoVO selectById(String id);

/\*\*

\* 根据厂商id查询 联系人数据

\*

\* @param supplierId 厂商id

\* @return 联系人数据

\*/

@Nullable

List<SupplierContactsInfoDTO> selectListBySupplierId(@Nullable String supplierId);

/\*\*

\* 逻辑删除设备信息

\* 可以根据主键 或者 厂商主键 （二选一） 和 租户号删除 联系人信息

\*

\* @param ids 主键数组

\* @param supplierIds 厂商主键数组

\* @param tenantId 租户号 ,默认当前用户下面的租户号

\* @return 返回逻辑删除数量

\*/

int logicDeleteByIds(String[] ids, String[] supplierIds, String tenantId);

/\*\*

\* 保存设备分类信息

\*

\* @param add 参数

\* @return 布尔值

\*/

boolean addSupplierContactsInfo(@Nullable SupplierContactsInfoDTO add);

/\*\*

\* 更新设备分类信息

\*

\* @param update 参数

\* @return 布尔值

\*/

boolean updateSupplierContactsInfo(@Nullable SupplierContactsInfoDTO update);

/\*\*

\* 批量插入联系人数据

\*

\* @param data 联系人数据

\* @return 布尔值

\*/

boolean batchInserData(List<SupplierContactsInfo> data);

}

/\*\*

\* @author Administrator

\* @description 针对表【supplier\_info(厂商信息表)】的数据库操作Service

\* @createDate 2023-05-09 10:27:17

\*/

public interface SupplierInfoService extends IService<SupplierInfo> {

/\*\*

\* 分页查询<b>厂商和联系人</b>信息表信息

\*

\* @param dto 查询参数

\* @param isPage 是否分页, false 代表不分页

\* @return 返回 {@link Page} 对象，泛型是 {@link SupplierInfoVO}

\*/

Page<SupplierInfoVO> findByPage(SupplierInfoSelectDTO dto, boolean isPage);

/\*\*

\* 分页查询<b>厂商</b>信息表信息

\*

\* @param dto 查询参数

\* @param isPage 是否分页, false 代表不分页

\* @return 返回 {@link Page} 对象，泛型是 {@link SupplierInfoVO}

\*/

Page<SupplierInfoVO> findSupplierByPage(SupplierInfoSelectDTO dto, boolean isPage);

/\*\*

\* 逻辑删除厂商和联系人信息表

\* ids 为空表示删除该租户下所有的 <b>厂商和联系人</b>数据

\*

\* @param ids 主键数组

\* @return 返回逻辑删除数量

\*/

int logicDelete(String[] ids);

/\*\*

\* <p>添加厂商信息和联系人信息</p>

\* <p>注意：参数为空时，返回true</p>

\*

\* @param add {@link SupplierInfoDTO} 参数，传入联系人和厂商数据

\* @return 布尔值

\*/

boolean addSupplierInfo(@Nullable SupplierInfoDTO add);

/\*\*

\* <p>更新<b>厂商信息</b>表</p>

\* <p>注意：参数为空时，返回true</p>

\*

\* @param update {@link SupplierInfoUpdateDTO} 参数

\* @return 布尔值

\*/

boolean updateSupplierInfo(SupplierInfoUpdateDTO update);

/\*\*

\* 根据 {@link SupplierInfoVO} 模板类导入数据

\*

\* @param data excel数据

\* @param isOverride 是否覆盖

\* @param isFirstDelete 是否删除再插入

\*/

void importData(List<SupplierInfoVO> data, Boolean isOverride, Boolean isFirstDelete);

/\*\*

\* 导出数据

\*

\* @param dto 查询参数 {@link SupplierInfoSelectDTO}

\* @param response {@link HttpServletResponse}

\*/

void exportData(SupplierInfoSelectDTO dto, HttpServletResponse response);

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_asset\_info\_record(设备资产信息记录表)】的数据库操作Service实现

\* @createDate 2023-05-15 10:43:37

\*/

@Service

public class DeviceAssetInfoRecordServiceImpl extends ServiceImpl<DeviceAssetInfoRecordMapper, DeviceAssetInfoRecord>

implements IDeviceAssetInfoRecordService {

@Resource

private DeviceAssetInfoRecordMapper deviceAssetInfoRecordMapper;

@Resource

private SysDictMapper sysDictMapper;

/\*\*

\* 分页查询设备资产信息记录

\*

\* @param deviceAssetInfoRecordSelectDTO

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceAssetInfoRecordVO>

\* @author wanggang

\* @date 2023/5/18 9:27

\*/

@Override

public Page<DeviceAssetInfoRecordVO> selectDeviceAssetInfoRecordPage(DeviceAssetInfoRecordSelectDTO deviceAssetInfoRecordSelectDTO) {

if (null == deviceAssetInfoRecordSelectDTO) {

throw new ParameterException("查询参数缺失");

}

Page<DeviceAssetInfoRecordVO> page = new Page<>(deviceAssetInfoRecordSelectDTO.getCurrent(), deviceAssetInfoRecordSelectDTO.getPageSize());

Page<DeviceAssetInfoRecordVO> deviceAssetInfoRecordVO = baseMapper.selectDeviceAssetInfoRecordPage(page, deviceAssetInfoRecordSelectDTO);

List<DeviceAssetInfoRecordVO> records = deviceAssetInfoRecordVO.getRecords();

//资产分类device\_asset\_category

//在账状态device\_on\_account\_status

//查询数据字典

List<SysDictVO> deviceAssetCategory = sysDictMapper.selectDict("device\_asset\_category");

List<SysDictVO> deviceOnAccountStatus = sysDictMapper.selectDict("device\_on\_account\_status");

for (DeviceAssetInfoRecordVO record : records) {

String assetClassify = record.getAssetClassify();

String onAccountStatus = record.getOnAccountStatus();

List<SysDictVO> collect = deviceAssetCategory.stream().filter(f -> null != f.getVal() && f.getVal().equals(assetClassify)).collect(Collectors.toList());

if (CollUtil.isNotEmpty(collect)) {

record.setAssetClassifyDisplay(collect.get(0).getLabel());

}

List<SysDictVO> collect1 = deviceOnAccountStatus.stream().filter(f -> null != f.getVal() && f.getVal().equals(onAccountStatus)).collect(Collectors.toList());

if (CollUtil.isNotEmpty(collect1)) {

record.setOnAccountStatusDisplay(collect1.get(0).getLabel());

}

}

return deviceAssetInfoRecordVO;

}

/\*\*

\* 根据资产信息ID查询设备资产信息记录

\*

\* @param id

\* @return com.cstc.jcgl.domain.device.vo.DeviceAssetInfoRecordVO

\* @author wanggang

\* @date 2023/5/15 11:40

\*/

@Override

public DeviceAssetInfoRecordVO selectDeviceAssetInfoRecordById(String id) {

if (StringUtils.isEmpty(id)) {

throw new ParameterException("查询参数缺失");

}

//根据资产信息ID查询设备资产信息记录

return baseMapper.selectDeviceAssetInfoRecordById(id);

}

/\*\*

\* 新增设备资产信息记录

\*

\* @param deviceAssetInfoRecordDTO

\* @return void

\* @author wanggang

\* @date 2023/5/15 10:30

\*/

@Override

public Boolean addDeviceAssetInfoRecord(DeviceAssetInfoRecordDTO deviceAssetInfoRecordDTO) {

if (null == deviceAssetInfoRecordDTO) {

throw new ParameterException("新增参数缺失");

}

DeviceAssetInfoRecord deviceAssetInfoRecord = new DeviceAssetInfoRecord();

JcglBeanUtil.copyProperties(deviceAssetInfoRecordDTO, deviceAssetInfoRecord, true);

String assetCode = deviceAssetInfoRecordDTO.getAssetCode();

DeviceAssetInfoRecordVO deviceAssetInfoRecordVO = deviceAssetInfoRecordMapper.selectDeviceAssetInfoRecordByAssetCode(assetCode);

if (null != deviceAssetInfoRecordVO) {

throw new ServiceException("新增数据的资产编号已经存在!");

}

return this.save(deviceAssetInfoRecord);

}

/\*\*

\* 修改设备资产信息记录

\*

\* @param deviceAssetInfoRecordDTO

\* @return void

\* @author wanggang

\* @date 2023/5/15 10:30

\*/

@Override

public Boolean updateDeviceAssetInfoRecord(DeviceAssetInfoRecordDTO deviceAssetInfoRecordDTO) {

if (null == deviceAssetInfoRecordDTO || StringUtils.isEmpty(deviceAssetInfoRecordDTO.getId())) {

throw new ParameterException("修改参数缺失");

}

DeviceAssetInfoRecord deviceAssetInfoRecord = new DeviceAssetInfoRecord();

JcglBeanUtil.copyProperties(deviceAssetInfoRecordDTO, deviceAssetInfoRecord, true);

return this.updateById(deviceAssetInfoRecord);

}

/\*\*

\* 逻辑删除设备资产信息记录

\*

\* @param id

\* @return void

\* @author wanggang

\* @date 2023/5/15 11:50

\*/

@Override

public Boolean deleteDeviceAssetInfoRecord(String id) {

if (StringUtils.isEmpty(id)) {

throw new ParameterException("逻辑删除Id不能为空!");

}

return this.removeById(id);

}

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_authorization(设备授权表)】的数据库操作Service实现

\* @createDate 2023-06-08 16:12:54

\*/

@Service

public class DeviceAuthorizationServiceImpl extends ServiceImpl<DeviceAuthorizationMapper, DeviceAuthorization>

implements IDeviceAuthorizationService {

@Resource

SysUserMapper sysUserMapper;

@Resource

SysDictMapper sysDictMapper;

/\*\*

\* 分页查询设备授权

\*

\* @param deviceAuthorizationSelectDTO

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceAuthorizationVO>

\* @author wanggang

\* @date 2023/6/9 15:27

\*/

@Override

public Page<DeviceAuthorizationVO> selectDeviceAuthorizationPage(DeviceAuthorizationSelectDTO deviceAuthorizationSelectDTO) {

if (null == deviceAuthorizationSelectDTO) {

throw new ParameterException("查询参数缺失");

}

Page<DeviceAuthorizationVO> page = new Page<>(deviceAuthorizationSelectDTO.getCurrent(), deviceAuthorizationSelectDTO.getPageSize());

Page<DeviceAuthorizationVO> deviceAuthorizationVO = baseMapper.selectDeviceAuthorizationPage(page, deviceAuthorizationSelectDTO);

List<DeviceAuthorizationVO> records = deviceAuthorizationVO.getRecords();

//查询数据字典

//设备状态 device\_status

List<SysDictVO> deviceStatus = sysDictMapper.selectDict("device\_status");

//资产类别

List<SysDictVO> deviceAssetCategory = sysDictMapper.selectDict("device\_asset\_category");

for (DeviceAuthorizationVO record : records) {

String status = record.getDeviceStatus();

String assetClassify = record.getAssetClassify();

List<SysDictVO> collect = deviceStatus.stream().filter(f -> null != f.getVal() && f.getVal().equals(status)).collect(Collectors.toList());

if (CollUtil.isNotEmpty(collect)) {

record.setDeviceStatusDisplay(collect.get(0).getLabel());

}

List<SysDictVO> collect1 = deviceAssetCategory.stream().filter(f -> null != f.getVal() && f.getVal().equals(assetClassify)).collect(Collectors.toList());

if (CollUtil.isNotEmpty(collect1)) {

record.setAssetClassifyDisplay(collect1.get(0).getLabel());

}

String authorizationPerson = record.getAuthorizationPerson();

List<String> authorizationPersonIdList = Arrays.asList(authorizationPerson.split("、"));

List<String> collect2 = authorizationPersonIdList.stream().distinct().collect(Collectors.toList());

//查询对应的人员名称

List<SysUserVO> sysUserVO = sysUserMapper.selectNameByUserId(collect2);

List<String> nameList = sysUserVO.stream().map(SysUserVO::getEmpName).filter(Objects::nonNull).collect(Collectors.toList());

String authorizationPersonName = nameList.stream().map(String::valueOf).collect(Collectors.joining("、"));

record.setAuthorizationPersonName(authorizationPersonName);

}

return deviceAuthorizationVO;

}

/\*\*

\* 根据设备授权表ID查询修改回显数据

\*

\* @param id

\* @return com.cstc.jcgl.domain.device.vo.DeviceAuthorizationVO

\* @author wanggang

\* @date 2023/6/9 10:21

\*/

@Override

public DeviceAuthorizationVO selectDeviceAuthorizationById(String id) {

if (StringUtils.isEmpty(id)) {

throw new ParameterException("查询参数缺失");

}

//根据备授权表ID查询修改回显数据

DeviceAuthorizationVO deviceAuthorizationVO = baseMapper.selectDeviceAuthorizationById(id);

if (null != deviceAuthorizationVO) {

String authorizationPerson = deviceAuthorizationVO.getAuthorizationPerson();

if (StringUtils.isNotEmpty(authorizationPerson)) {

List<String> authorizationPersonIdList = Arrays.asList(authorizationPerson.split("、"));

List<String> collect = authorizationPersonIdList.stream().distinct().collect(Collectors.toList());

//查询对应的授权人员

List<SysUserVO> sysUserVO = sysUserMapper.selectNameByUserId(collect);

String involvedStaffName = sysUserVO.stream().map(SysUserVO::getEmpName).filter(Objects::nonNull)

.collect(Collectors.joining("、"));

deviceAuthorizationVO.setAuthorizationPersonName(involvedStaffName);

deviceAuthorizationVO.setAuthorizationPersonList(sysUserVO);

}

}

return deviceAuthorizationVO;

}

/\*\*

\* 批量新增设备授权记录

\*

\* @param deviceAuthorizationDTOList

\* @return Boolean

\* @author wanggang

\* @date 2023/6/8 16:30

\*/

@Override

public Boolean addDeviceAuthorizationBatch(List<DeviceAuthorizationDTO> deviceAuthorizationDTOList) {

if (CollUtil.isEmpty(deviceAuthorizationDTOList)) {

throw new ParameterException("新增参数缺失");

}

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

for (DeviceAuthorizationDTO deviceAuthorizationDTO : deviceAuthorizationDTOList) {

List<String> authorizationPersonList = deviceAuthorizationDTO.getAuthorizationPersonList();

String authorizationPerson = String.join("、", authorizationPersonList);

DeviceAuthorization deviceAuthorization = new DeviceAuthorization();

deviceAuthorizationDTO.setAuthorizationPerson(authorizationPerson);

JcglBeanUtil.copyProperties(deviceAuthorizationDTO, deviceAuthorization, true);

list.add(deviceAuthorization);

}

return this.saveBatch(list);

}

/\*\*

\* 修改设备授权记录

\*

\* @param deviceAuthorizationDTO

\* @return Boolean

\* @author wanggang

\* @date 2023/6/8 16:30

\*/

@Override

public Boolean updateDeviceAuthorization(DeviceAuthorizationDTO deviceAuthorizationDTO) {

if (null == deviceAuthorizationDTO || StringUtils.isEmpty(deviceAuthorizationDTO.getId())) {

throw new ParameterException("修改参数缺失");

}

List<String> authorizationPersonList = deviceAuthorizationDTO.getAuthorizationPersonList();

String authorizationPerson = String.join("、", authorizationPersonList);

DeviceAuthorization deviceAuthorization = new DeviceAuthorization();

deviceAuthorizationDTO.setAuthorizationPerson(authorizationPerson);

JcglBeanUtil.copyProperties(deviceAuthorizationDTO, deviceAuthorization, true);

return this.updateById(deviceAuthorization);

}

/\*\*

\* 逻辑删除设备授权记录

\*

\* @param id

\* @return Boolean

\* @author wanggang

\* @date 2023/6/9 11:30

\*/

@Override

public Boolean deleteDeviceAuthorization(String id) {

if (StringUtils.isEmpty(id)) {

throw new ParameterException("逻辑删除Id不能为空!");

}

return this.removeById(id);

}

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_borrow\_record(设备借用记录表)】的数据库操作Service实现

\* @createDate 2023-05-22 16:04:16

\*/

@Service

public class DeviceBorrowRecordServiceImpl extends ServiceImpl<DeviceBorrowRecordMapper, DeviceBorrowRecord>

implements IDeviceBorrowRecordService {

@Resource

private SysDictMapper sysDictMapper;

@Resource

private FileAttachMapper fileAttachMapper;

/\*\*

\* 分页查询设备借用记录

\*

\* @param deviceBorrowRecordSelectDTO

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceBorrowRecordVO>

\* @author wanggang

\* @date 2023/5/22 13:47

\*/

@Override

public Page<DeviceBorrowRecordVO> selectDeviceBorrowRecordPage(DeviceBorrowRecordSelectDTO deviceBorrowRecordSelectDTO) {

if (null == deviceBorrowRecordSelectDTO) {

throw new ParameterException("查询参数缺失");

}

Page<DeviceBorrowRecordVO> page = new Page<>(deviceBorrowRecordSelectDTO.getCurrent(), deviceBorrowRecordSelectDTO.getPageSize());

Page<DeviceBorrowRecordVO> deviceBorrowRecordVO = baseMapper.selectDeviceBorrowRecordPage(page, deviceBorrowRecordSelectDTO);

List<DeviceBorrowRecordVO> records = deviceBorrowRecordVO.getRecords();

//查询数据字典

//借出/归还状态 device\_situation

List<SysDictVO> deviceStatus = sysDictMapper.selectDict("device\_situation");

for (DeviceBorrowRecordVO record : records) {

String lendDeviceStatus = record.getLendDeviceStatus();

String returnDeviceStatus = record.getReturnDeviceStatus();

List<SysDictVO> collect = deviceStatus.stream().filter(f -> null != f.getVal() && f.getVal().equals(lendDeviceStatus)).collect(Collectors.toList());

if (CollUtil.isNotEmpty(collect)) {

record.setLendDeviceStatusDisplay(collect.get(0).getLabel());

}

List<SysDictVO> collect1 = deviceStatus.stream().filter(f -> null != f.getVal() && f.getVal().equals(returnDeviceStatus)).collect(Collectors.toList());

if (CollUtil.isNotEmpty(collect1)) {

record.setReturnDeviceStatusDisplay(collect1.get(0).getLabel());

}

//处理文件信息

String fileId = record.getFileId();

if (StringUtils.isNotEmpty(fileId)) {

List<String> fileIdList = Arrays.asList(fileId.split("、"));

List<String> collect2 = fileIdList.stream().distinct().collect(Collectors.toList());

//查询对应文件信息

List<FileAttachVO> fileAttachVOList = fileAttachMapper.selectFileByFileId(collect2);

record.setFileAttachVOList(fileAttachVOList);

String uuId = fileAttachVOList.stream().map(FileAttachVO::getUuid)

.filter(Objects::nonNull).collect(Collectors.joining("、"));

String fileName = fileAttachVOList.stream().map(FileAttachVO::getFilename)

.filter(Objects::nonNull).collect(Collectors.joining("、"));

List<Date> collect3 = fileAttachVOList.stream().map(FileAttachVO::getCreateTime)

.filter(Objects::nonNull).collect(Collectors.toList());

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

for (Date date : collect3) {

String dateString = DateUtil.dateToStr("yyyy-MM-dd HH:mm:ss", date);

dateList.add(dateString);

}

String uploadTime = String.join("、", dateList);

record.setFileIdList(collect2);

record.setFilePath(uuId);

record.setFileName(fileName);

record.setUploadTime(uploadTime);

}

}

return deviceBorrowRecordVO;

}

/\*\*

\* 根据设备借用记录ID查询修改回显数据

\*

\* @param id

\* @return com.cstc.jcgl.domain.device.vo.DeviceBorrowRecordVO

\* @author wanggang

\* @date 2023/5/22 11:21

\*/

@Override

public DeviceBorrowRecordVO selectDeviceBorrowRecordById(String id) {

if (StringUtils.isEmpty(id)) {

throw new ParameterException("查询参数缺失");

}

//根据设备借用记录ID查询修改回显数据

DeviceBorrowRecordVO deviceBorrowRecordVO = baseMapper.selectDeviceBorrowRecordById(id);

if (null != deviceBorrowRecordVO) {

String fileId = deviceBorrowRecordVO.getFileId();

if (StringUtils.isNotEmpty(fileId)) {

List<String> fileIdList = Arrays.asList(fileId.split("、"));

List<String> collect = fileIdList.stream().distinct().collect(Collectors.toList());

//查询对应文件信息

List<FileAttachVO> fileAttachVOList = fileAttachMapper.selectFileByFileId(collect);

deviceBorrowRecordVO.setFileAttachVOList(fileAttachVOList);

}

}

return deviceBorrowRecordVO;

}

/\*\*

\* 新增设备借用记录

\*

\* @param deviceBorrowRecordDTO

\* @return Boolean

\* @author wanggang

\* @date 2023/5/22 16:30

\*/

@Override

public Boolean addDeviceBorrowRecord(DeviceBorrowRecordDTO deviceBorrowRecordDTO) {

if (null == deviceBorrowRecordDTO) {

throw new ParameterException("新增参数缺失");

}

DeviceBorrowRecord deviceBorrowRecord = new DeviceBorrowRecord();

List<String> fileIdList = deviceBorrowRecordDTO.getFileIdList();

if (CollUtil.isNotEmpty(fileIdList)) {

String fileId = String.join("、", fileIdList);

deviceBorrowRecordDTO.setFileId(fileId);

}

JcglBeanUtil.copyProperties(deviceBorrowRecordDTO, deviceBorrowRecord, true);

return this.save(deviceBorrowRecord);

}

/\*\*

\* 修改设备借用记录

\*

\* @param deviceBorrowRecordDTO

\* @return Boolean

\* @author wanggang

\* @date 2023/5/22 16:30

\*/

@Override

public Boolean updateDeviceBorrowRecord(DeviceBorrowRecordDTO deviceBorrowRecordDTO) {

if (null == deviceBorrowRecordDTO || StringUtils.isEmpty(deviceBorrowRecordDTO.getId())) {

throw new ParameterException("修改参数缺失");

}

DeviceBorrowRecord deviceBorrowRecord = new DeviceBorrowRecord();

List<String> fileIdList = deviceBorrowRecordDTO.getFileIdList();

if (CollUtil.isNotEmpty(fileIdList)) {

String fileId = String.join("、", fileIdList);

deviceBorrowRecordDTO.setFileId(fileId);

}

JcglBeanUtil.copyProperties(deviceBorrowRecordDTO, deviceBorrowRecord, true);

return this.updateById(deviceBorrowRecord);

}

/\*\*

\* 逻辑删除设备借用记录

\*

\* @param id

\* @return Boolean

\* @author wanggang

\* @date 2023/5/22 16:30

\*/

@Override

public Boolean deleteDeviceBorrowRecord(String id) {

if (StringUtils.isEmpty(id)) {

throw new ParameterException("逻辑删除Id不能为空!");

}

return this.removeById(id);

}

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_circulation\_record(设备流转记录表)】的数据库操作Service实现

\* @createDate 2023-05-10 09:18:40

\*/

@Service

public class DeviceCirculationRecordServiceImpl extends ServiceImpl<DeviceCirculationRecordMapper, DeviceCirculationRecord>

implements IDeviceCirculationRecordService {

@Resource

private SysDictMapper sysDictMapper;

@Resource

private FileAttachMapper fileAttachMapper;

/\*\*

\* 分页查询设备流转记录

\*

\* @param deviceCirculationRecordSelectDTO

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceCirculationRecordVO>

\* @author wanggang

\* @date 2023/5/22 13:47

\*/

@Override

public Page<DeviceCirculationRecordVO> selectDeviceCirculationRecordPage(DeviceCirculationRecordSelectDTO deviceCirculationRecordSelectDTO) {

if (null == deviceCirculationRecordSelectDTO) {

throw new ParameterException("查询参数缺失");

}

Page<DeviceVerificationRecordVO> page = new Page<>(deviceCirculationRecordSelectDTO.getCurrent(), deviceCirculationRecordSelectDTO.getPageSize());

Page<DeviceCirculationRecordVO> deviceCirculationRecordVO = baseMapper.selectDeviceCirculationRecordPage(page, deviceCirculationRecordSelectDTO);

List<DeviceCirculationRecordVO> records = deviceCirculationRecordVO.getRecords();

//查询数据字典

//发出/接收状态 device\_situation

List<SysDictVO> deviceStatus = sysDictMapper.selectDict("device\_situation");

List<SysDictVO> circulationTypeList = sysDictMapper.selectDict("device\_circulation\_type");

for (DeviceCirculationRecordVO record : records) {

String issueDeviceSituation = record.getIssueDeviceSituation();

String receiveDeviceSituation = record.getReceiveDeviceSituation();

String circulationType = record.getCirculationType();

List<SysDictVO> collect = deviceStatus.stream().filter(f -> null != f.getVal() && f.getVal().equals(issueDeviceSituation)).collect(Collectors.toList());

if (CollUtil.isNotEmpty(collect)) {

record.setIssueDeviceSituationDisplay(collect.get(0).getLabel());

}

List<SysDictVO> collect1 = deviceStatus.stream().filter(f -> null != f.getVal() && f.getVal().equals(receiveDeviceSituation)).collect(Collectors.toList());

if (CollUtil.isNotEmpty(collect1)) {

record.setReceiveDeviceSituationDisplay(collect1.get(0).getLabel());

}

List<SysDictVO> collect4 = circulationTypeList.stream().filter(f -> null != f.getVal() && f.getVal().equals(circulationType)).collect(Collectors.toList());

if (CollUtil.isNotEmpty(collect4)) {

record.setCirculationTypeDisplay(collect4.get(0).getLabel());

}

//处理文件信息

String fileId = record.getFileId();

if (StringUtils.isNotEmpty(fileId)) {

List<String> fileIdList = Arrays.asList(fileId.split("、"));

List<String> collect2 = fileIdList.stream().distinct().collect(Collectors.toList());

//查询对应文件信息

List<FileAttachVO> fileAttachVOList = fileAttachMapper.selectFileByFileId(collect2);

record.setFileAttachVOList(fileAttachVOList);

String uuId = fileAttachVOList.stream().map(FileAttachVO::getUuid)

.filter(Objects::nonNull).collect(Collectors.joining("、"));

String fileName = fileAttachVOList.stream().map(FileAttachVO::getFilename)

.filter(Objects::nonNull).collect(Collectors.joining("、"));

List<Date> collect3 = fileAttachVOList.stream().map(FileAttachVO::getCreateTime)

.filter(Objects::nonNull).collect(Collectors.toList());

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

for (Date date : collect3) {

String dateString = DateUtil.dateToStr("yyyy-MM-dd HH:mm:ss", date);

dateList.add(dateString);

}

String uploadTime = String.join("、", dateList);

record.setFileIdList(collect2);

record.setFilePath(uuId);

record.setFileName(fileName);

record.setUploadTime(uploadTime);

}

}

return deviceCirculationRecordVO;

}

/\*\*

\* 根据设备流转记录ID查询修改回显数据

\*

\* @param id

\* @return com.cstc.jcgl.domain.device.vo.DeviceCirculationRecordVO

\* @author wanggang

\* @date 2023/5/22 11:21

\*/

@Override

public DeviceCirculationRecordVO selectDeviceCirculationRecordById(String id) {

if (StringUtils.isEmpty(id)) {

throw new ParameterException("查询参数缺失");

}

//根据设备流转记录ID查询修改回显数据

DeviceCirculationRecordVO deviceCirculationRecordVO = baseMapper.selectDeviceCirculationRecordById(id);

if (null != deviceCirculationRecordVO) {

String fileId = deviceCirculationRecordVO.getFileId();

if (StringUtils.isNotEmpty(fileId)) {

List<String> fileIdList = Arrays.asList(fileId.split("、"));

List<String> collect = fileIdList.stream().distinct().collect(Collectors.toList());

//查询对应文件信息

List<FileAttachVO> fileAttachVOList = fileAttachMapper.selectFileByFileId(collect);

deviceCirculationRecordVO.setFileAttachVOList(fileAttachVOList);

}

}

return deviceCirculationRecordVO;

}

/\*\*

\* 新增设备流转记录

\*

\* @param deviceCirculationRecordDTO

\* @return Boolean

\* @author wanggang

\* @date 2023/5/10 11:30

\*/

@Override

public Boolean addDeviceCirculationRecord(DeviceCirculationRecordDTO deviceCirculationRecordDTO) {

if (null == deviceCirculationRecordDTO) {

throw new ParameterException("新增参数缺失");

}

DeviceCirculationRecord deviceCirculationRecord = new DeviceCirculationRecord();

List<String> fileIdList = deviceCirculationRecordDTO.getFileIdList();

if (CollUtil.isNotEmpty(fileIdList)) {

String fileId = String.join("、", fileIdList);

deviceCirculationRecordDTO.setFileId(fileId);

}

JcglBeanUtil.copyProperties(deviceCirculationRecordDTO, deviceCirculationRecord, true);

return this.save(deviceCirculationRecord);

}

/\*\*

\* 修改设备流转记录

\*

\* @param deviceCirculationRecordDTO

\* @return Boolean

\* @author wanggang

\* @date 2023/5/10 11:30

\*/

@Override

public Boolean updateDeviceCirculationRecord(DeviceCirculationRecordDTO deviceCirculationRecordDTO) {

if (null == deviceCirculationRecordDTO || StringUtils.isEmpty(deviceCirculationRecordDTO.getId())) {

throw new ParameterException("修改参数缺失");

}

DeviceCirculationRecord deviceCirculationRecord = new DeviceCirculationRecord();

List<String> fileIdList = deviceCirculationRecordDTO.getFileIdList();

if (CollUtil.isNotEmpty(fileIdList)) {

String fileId = String.join("、", fileIdList);

deviceCirculationRecordDTO.setFileId(fileId);

}

JcglBeanUtil.copyProperties(deviceCirculationRecordDTO, deviceCirculationRecord, true);

return this.updateById(deviceCirculationRecord);

}

/\*\*

\* 逻辑删除设备流转记录

\*

\* @param id

\* @return Boolean

\* @author wanggang

\* @date 2023/5/10 11:30

\*/

@Override

public Boolean deleteDeviceCirculationRecord(String id) {

if (StringUtils.isEmpty(id)) {

throw new ParameterException("逻辑删除Id不能为空!");

}

return this.removeById(id);

}

}

/\*\*

\* @author Administrator

\* @description 针对表【device\_classify\_info(设备分类信息表)】的数据库操作Service实现

\* @createDate 2023-05-06 16:33:10

\*/

@Service

public class DeviceClassifyInfoServiceImpl extends ServiceImpl<DeviceClassifyInfoMapper, DeviceClassifyInfo>

implements DeviceClassifyInfoService {

@Resource

private SupplierInfoService supplierInfoService;

@Resource

private SupplierContactsInfoMapper concatenationInfoMapper;

@Resource

private ISysDictCategoryService sysDictCategoryService;

@Override

public Page<DeviceClassifyInfoVO> findByPage(DeviceClassifyInfoSelectDTO dto, boolean isPage) {

Page<DeviceClassifyInfoVO> page = new Page<>(dto.getCurrent(), dto.getPageSize());

// 处理时间

if (dto.getStartTime() != null) {

dto.setStartTimeString(dto.getStartTime().format(DateTimeFormatter.ofPattern(DateTimeConstant.DATE))

+ DateTimeConstant.DAY\_START\_TIME);

}

if (dto.getEndTime() != null) {

dto.setEndTimeString(dto.getEndTime().format(DateTimeFormatter.ofPattern(DateTimeConstant.DATE))

+ DateTimeConstant.DAY\_END\_TIME);

}

// 是否关闭分页查询的总数统计

page.setSearchCount(isPage);

if (isPage) {

baseMapper.findDeviceClassifyByPage(page, dto);

} else {

page.setRecords(baseMapper.findDeviceClassifyByPage(dto));

}

return page;

}

@Override

public DeviceClassifyInfoVO selectById(String id) {

if (StringUtils.isEmpty(id)) return null;

DeviceClassifyInfo deviceClassifyInfo = baseMapper.selectById(id);

DeviceClassifyInfoVO result = BeanUtil.copyProperties(deviceClassifyInfo, DeviceClassifyInfoVO.class);

// 查询供应商信息

SupplierInfo product = supplierInfoService.getById(deviceClassifyInfo.getProductionId());

result.setProductionName(product.getName());

// 查询供应商信息

SupplierInfo supplier = supplierInfoService.getById(deviceClassifyInfo.getSupplierId());

result.setSupplierName(supplier.getName());

// 查询供应商信息 最新的联系人

SupplierContactsInfoVO contactsInfoVO = concatenationInfoMapper.selectContactsInfoLatest(supplier.getId());

if (contactsInfoVO != null) {

result.setSupplierContactsName(contactsInfoVO.getContactName());

result.setSupplierContactPhone(contactsInfoVO.getTelephone());

}

return result;

}

@Override

public Set<String> selectDeviceClassifyNames(String tenantId) {

return baseMapper.selectDeviceClassifyNames(tenantId);

}

@Override

public int logicDelete(String[] ids) {

return this.baseMapper.logicDeleteDeviceClassifyInfo(ids,

TenantContextHolder.getContext().getTenantContextInfo().getSysUser().getTenantId());

}

private void checkDeviceClassifyInfo(DeviceClassifyInfo info) {

// 1、检测 厂家、供应商id是否存在

SupplierInfo supplierInfo = supplierInfoService.getById(info.getSupplierId());

SupplierInfo providerInfo = supplierInfoService.getById(info.getProductionId());

if (supplierInfo == null || providerInfo == null

|| supplierInfo.getDelFlag() == 0 || providerInfo.getDelFlag() == 0) {

throw new ServiceException("传入的厂家、供应商不存在");

}

if (StringUtils.isEmpty(info.getDeviceClassifyName())) {

throw new ServiceException("设备分类名称不存在");

} else {

// 已经存在的设备分类名称也要查看是否需要修改

List<DeviceClassifyInfoVO> deviceClassifyInfoVOS = baseMapper.selectDeviceClassifyInfo(info.getDeviceClassifyName(),

TenantContextHolder.getContext().getTenantContextInfo().getTenantId());

if (CollUtil.isNotEmpty(deviceClassifyInfoVOS)) {

throw new ServiceException("设备分类名称已经存在");

}

}

if (StringUtils.isEmpty(info.getDeviceName())) {

throw new ServiceException("设备名称不存在");

}

if (StringUtils.isEmpty(info.getModel())) {

throw new ServiceException("规格型号不存在");

}

if (StringUtils.isEmpty(info.getDeviceClassify())) {

throw new ServiceException("设备分类不存在");

}

if (StringUtils.isEmpty(info.getAssetClassify())) {

throw new ServiceException("资产分类不存在");

}

}

@Override

public boolean add(DeviceClassifyInfoDTO add) {

if (add == null) return true;

DeviceClassifyInfo deviceClassifyInfo = add.transferEntity();

this.checkDeviceClassifyInfo(deviceClassifyInfo);

// TODO 生成设备分类编号

deviceClassifyInfo.setDeviceClassifyNumber(String.valueOf(System.currentTimeMillis()));

return baseMapper.insert(deviceClassifyInfo) == 1;

}

@Override

public boolean update(DeviceClassifyInfoDTO update) {

if (update == null) return true;

DeviceClassifyInfo deviceClassifyInfo = update.transferEntity();

if (deviceClassifyInfo.getId() == null) {

throw new ParameterException("请求的主键没有传入");

}

DeviceClassifyInfo old = baseMapper.selectById(deviceClassifyInfo.getId());

if (old == null) {

throw new ParameterException("请求更新的设备分类数据不存在，不能更新");

}

if (StringUtils.isNotEmpty(update.getDeviceClassifyNumber()) && StringUtils.isNotEmpty(old.getDeviceClassifyNumber())

&& !old.getDeviceClassifyNumber().equals(update.getDeviceClassifyNumber())) {

throw new ParameterException("设备分类编号不能更新");

}

return this.updateById(deviceClassifyInfo);

}

private void dealWithDictType(List<DeviceClassifyInfoVO> list, boolean order) {

// 获取对应的字典

Map<String, String> deviceOwnership = sysDictCategoryService.selectDictBycategoryKey(null, "device\_ownership", order);

Map<String, String> deviceClassify = sysDictCategoryService.selectDictBycategoryKey(null, "device\_classify", order);

Map<String, String> assetCategory = sysDictCategoryService.selectDictBycategoryKey(null, "device\_asset\_category", order);

if (org.springframework.util.CollectionUtils.isEmpty(deviceOwnership)) {

throw new ServiceException("设备归属字典没有数据，请联系管理员添加");

}

if (org.springframework.util.CollectionUtils.isEmpty(deviceClassify)) {

throw new ServiceException("设备类型字典没有数据，请联系管理员添加");

}

if (org.springframework.util.CollectionUtils.isEmpty(assetCategory)) {

throw new ServiceException("资产分类字典没有数据，请联系管理员添加");

}

list.forEach(s -> {

s.setDeviceClassify(deviceClassify.get(String.valueOf(s.getDeviceClassify())));

s.setAssetClassify(assetCategory.get(String.valueOf(s.getAssetClassify())));

if (StringUtils.isNotBlank(s.getOwnership())) {

s.setOwnership(deviceOwnership.get(String.valueOf(s.getOwnership())));

}

});

}

@Override

@Transactional(rollbackFor = Exception.class)

public void importData(List<DeviceClassifyInfoVO> data, Boolean isOverride, Boolean isFirstDelete) {

if (CollUtil.isEmpty(data)) {

return;

}

// 查询所有的设备分类名称

Set<String> names = baseMapper.selectDeviceClassifyNames(TenantContextHolder.getContext().getTenantContextInfo().getTenantId());

// 处理枚举类数据

this.dealWithDictType(data, false);

// 覆盖时 执行 下面的操作，一条条的插入，判断

if (Boolean.TRUE.equals(isOverride)) {

this.checkData(data, names, isOverride);

this.updateDeviceClassifyData(isFirstDelete, isOverride, data, names);

} else {

this.insertDeviceClassifyData(isFirstDelete, isOverride, data, names);

}

}

private void checkData(List<DeviceClassifyInfoVO> data, Set<String> nameCollect, Boolean isOverride) {

// 获取 厂商的名称 集合

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

// 判断excel表中是否存在重复的租户号

for (DeviceClassifyInfoVO temp : data) {

// 不覆盖时检测是否重名

String deviceClassifyName = temp.getDeviceClassifyName();

if (nameCollect.contains(deviceClassifyName) && Boolean.FALSE.equals(isOverride)) {

throw new ServiceException("【" + deviceClassifyName + "】设备分类已经导入");

}

if (excelCollect.contains(deviceClassifyName)) {

throw new ServiceException("【" + deviceClassifyName + "】设备分类在表中重复");

} else {

excelCollect.add(deviceClassifyName);

}

}

}

/\*\*

\* 厂商和供应商转换为 id

\*

\* @param data 表格数据

\* @param isCreate true表示创建，必填厂商 和 供应商

\* @return

\*/

private List<DeviceClassifyInfo> dealSupplierAndProduct(List<DeviceClassifyInfoVO> data, boolean isCreate) {

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

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

data.forEach(d -> {

supplierNames.add(d.getSupplierName());

productNames.add(d.getProductionName());

});

String tenantId = TenantContextHolder.getContext().getTenantContextInfo().getTenantId();

if (supplierNames.isEmpty() && isCreate) {

throw new ServiceException("供应商数据必填");

}

if (productNames.isEmpty() && isCreate) {

throw new ServiceException("厂商数据必填");

}

List<SupplierInfo> supplierInfos = supplierInfoService.list(new QueryWrapper<SupplierInfo>().lambda()

.eq(true, SupplierInfo::getDelFlag, 1)

.eq(true, SupplierInfo::getTenantId, tenantId)

.eq(true, SupplierInfo::getType, 2)

.in(!supplierNames.isEmpty(), SupplierInfo::getName, supplierNames)

);

Map<String, String> supplierDict = supplierInfos.stream().collect(Collectors.toMap(SupplierInfo::getName, SupplierInfo::getId));

List<SupplierInfo> producerInfos = supplierInfoService.list(new QueryWrapper<SupplierInfo>().lambda()

.eq(true, SupplierInfo::getDelFlag, 1)

.eq(true, SupplierInfo::getTenantId, tenantId)

.eq(true, SupplierInfo::getType, 1)

.in(!supplierNames.isEmpty(), SupplierInfo::getName, productNames)

);

Map<String, String> producerDict = producerInfos.stream().collect(Collectors.toMap(SupplierInfo::getName, SupplierInfo::getId));

// 处理生产厂家 和 供应商

return data.stream().map(d -> {

DeviceClassifyInfo info = BeanUtil.copyProperties(d, DeviceClassifyInfo.class);

if (supplierDict.containsKey(d.getSupplierName())) {

info.setSupplierId(supplierDict.get(d.getSupplierName()));

} else {

throw new ServiceException("【" + d.getSupplierName() + "】供应商不存在，请先创建供应商信息");

}

if (producerDict.containsKey(d.getProductionName())) {

info.setProductionId(producerDict.get(d.getProductionName()));

} else {

throw new ServiceException("【" + d.getSupplierName() + "】厂商不存在，请先创建厂商信息");

}

return info;

}).toList();

}

private void updateDeviceClassifyData(Boolean isFirstDelete, Boolean isOverride, List<DeviceClassifyInfoVO> data,

Set<String> nameCollect) {

if (nameCollect.isEmpty()) {

this.insertDeviceClassifyData(isFirstDelete, isOverride, data, nameCollect);

} else {

List<DeviceClassifyInfo> deviceClassifyInfos = this.dealSupplierAndProduct(data, false);

String tenantId = TenantContextHolder.getContext().getTenantContextInfo().getTenantId();

deviceClassifyInfos.forEach(d -> {

List<DeviceClassifyInfoVO> vos = baseMapper.selectDeviceClassifyInfo(d.getDeviceClassifyName(), tenantId);

if (CollUtil.isEmpty(vos)) {

baseMapper.insert(d);

} else {

d.setId(vos.get(0).getId());

baseMapper.updateById(d);

}

});

}

}

private void insertDeviceClassifyData(Boolean isFirstDelete, Boolean isOverride, List<DeviceClassifyInfoVO> data, Set<String> nameCollect) {

if (Boolean.TRUE.equals(isFirstDelete)) this.logicDelete(null);

this.checkData(data, nameCollect, isOverride);

List<DeviceClassifyInfo> deviceClassifyInfos = this.dealSupplierAndProduct(data, true);

// TODO 生成分类编号

this.saveBatch(deviceClassifyInfos);

}

@Override

public void exportData(DeviceClassifyInfoSelectDTO dto, HttpServletResponse response) {

Page<DeviceClassifyInfoVO> page = this.findByPage(dto, false);

List<DeviceClassifyInfoVO> records = page.getRecords();

this.dealWithDictType(records, true);

try {

EasyExcelUtil.writeExcel(response, DeviceClassifyInfoVO.class, records, "厂商和联系人信息" + System.currentTimeMillis() + ".xlsx");

} catch (IOException | IllegalAccessException | NoSuchFieldException e) {

throw new ServiceException(e);

}

}

}

/\*\*

\* 针对表【device\_info(设备管理表)】的数据库操作Service实现

\*

\* @author wanggang

\* @date 2023/5/8 10:41

\*/

@Service

@RequiredArgsConstructor

public class DeviceInfoServiceImpl extends ServiceImpl<DeviceInfoMapper, DeviceInfo>

implements IDeviceInfoService {

@Resource

private DeviceClassifyInfoMapper deviceClassifyInfoMapper;

@Resource

private IDeviceAssetInfoRecordService iDeviceAssetInfoRecordService;

@Resource

private SysOrgMapper sysOrgMapper;

@Resource

private SysUserMapper sysUserMapper;

@Resource

private DeviceAssetInfoRecordMapper deviceAssetInfoRecordMapper;

@Resource

private SysDictMapper sysDictMapper;

@Resource

private DeviceShelvesInfoMapper deviceShelvesInfoMapper;

@Resource

private DeviceInfoMapper deviceInfoMapper;

@Resource

private SupplierContactsInfoMapper concatenationInfoMapper;

@Resource

private DeviceBorrowRecordMapper deviceBorrowRecordMapper;

@Resource

private DeviceCirculationRecordMapper deviceCirculationRecordMapper;

@Resource

private DeviceRepairMaintenanceRecordMapper deviceRepairMaintenanceRecordMapper;

@Resource

private DeviceVerificationRecordMapper deviceVerificationRecordMapper;

@Resource

private DeviceWorkingRecordMapper deviceWorkingRecordMapper;

/\*\*

\* 分页查询设备管理

\*

\* @param deviceInfoSelectDTO

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceInfoVO>

\* @author wanggang

\* @date 2023/5/8 17:27

\*/

@Override

public Page<DeviceInfoVO> selectDevicePage(DeviceInfoSelectDTO deviceInfoSelectDTO) {

if (null == deviceInfoSelectDTO) {

throw new ParameterException("查询参数缺失");

}

Page<DeviceInfoVO> page = new Page<>(deviceInfoSelectDTO.getCurrent(), deviceInfoSelectDTO.getPageSize());

Page<DeviceInfoVO> deviceInfoPage = baseMapper.selectDevicePage(page, deviceInfoSelectDTO);

//查询数据字典

List<SysDictVO> deviceUseStatus = sysDictMapper.selectDict("device\_use\_status");

//设备状态 device\_status

List<SysDictVO> deviceStatus = sysDictMapper.selectDict("device\_status");

List<SysDictVO> deviceVerificationStatus = sysDictMapper.selectDict("device\_verification\_status");

String startTime = deviceInfoSelectDTO.getStartTime();

String endTime = deviceInfoSelectDTO.getEndTime();

//查询条件包含这三个的时候 去掉没查询出来的检定数据的设备信息数据

if (StringUtils.isNotEmpty(startTime) || StringUtils.isNotEmpty(endTime)) {

List<DeviceInfoVO> records = deviceInfoPage.getRecords();

List<DeviceInfoVO> collect = records.stream().filter(f -> null != f.getValidEnd()).collect(Collectors.toList());

deviceInfoPage.setRecords(collect);

}

List<DeviceInfoVO> records = deviceInfoPage.getRecords();

for (DeviceInfoVO record : records) {

String useStatus = record.getUseStatus();

String verificationStatus1 = record.getVerificationStatus();

String status = record.getDeviceStatus();

List<SysDictVO> collect = deviceUseStatus.stream().filter(f -> null != f.getVal() && f.getVal().equals(useStatus)).collect(Collectors.toList());

if (CollUtil.isNotEmpty(collect)) {

record.setUseStatusDisplay(collect.get(0).getLabel());

}

List<SysDictVO> collect1 = deviceVerificationStatus.stream().filter(f -> null != f.getVal() && f.getVal().equals(verificationStatus1)).collect(Collectors.toList());

if (CollUtil.isNotEmpty(collect1)) {

record.setVerificationStatusDisplay(collect1.get(0).getLabel());

}

List<SysDictVO> collect2 = deviceStatus.stream().filter(f -> null != f.getVal() && f.getVal().equals(status)).collect(Collectors.toList());

if (CollUtil.isNotEmpty(collect2)) {

record.setDeviceStatusDisplay(collect2.get(0).getLabel());

}

}

return deviceInfoPage;

}

/\*\*

\* 新增设备信息

\*

\* @param deviceInfoDTO

\* @return void

\* @author wanggang

\* @date 2023/5/8 11:30

\*/

@Override

@Transactional(rollbackFor = Exception.class)

public Boolean addDeviceInfo(DeviceInfoDTO deviceInfoDTO) {

if (null == deviceInfoDTO) {

throw new ParameterException("新增参数缺失");

}

DeviceInfo deviceInfo = new DeviceInfo();

JcglBeanUtil.copyProperties(deviceInfoDTO, deviceInfo, true);

//自动生成设备编号

LocalDateTime now = LocalDateTime.now();

String date = now.format(DateTimeFormatter.ofPattern("yyyyMMddHHmmss"));

String deviceNumber = "SB" + date;

deviceInfo.setDeviceNumber(deviceNumber);

boolean save = this.save(deviceInfo);

if (save) {

//设备分类ID

String deviceClassifyInfoId = deviceInfoDTO.getDeviceClassifyInfoId();

//查询设备的数量

Integer quantity = deviceClassifyInfoMapper.selectDeviceQuantity(deviceClassifyInfoId);

if (null != quantity) {

Integer addQuantity = quantity + 1;

//修改设备数量

deviceClassifyInfoMapper.updateDeviceQuantity(addQuantity, deviceClassifyInfoId);

}

}

return save;

}

/\*\*

\* 根据设备信息ID查询修改回显数据

\*

\* @param id

\* @return com.cstc.jcgl.domain.device.vo.DeviceInfoVO

\* @author wanggang

\* @date 2023/5/11 14:21

\*/

@Override

public DeviceInfoVO selectDeviceInfoById(String id) {

if (StringUtils.isEmpty(id)) {

throw new ParameterException("查询参数缺失");

}

//根据设备信息ID查询设备信息表修改回显数据

DeviceInfoVO deviceInfoVO = baseMapper.selectDeviceInfoById(id);

String supplierId = deviceInfoVO.getSupplierId();

// 查询供应商的最新的联系人

SupplierContactsInfoVO contactsInfoVO = concatenationInfoMapper.selectContactsInfoLatest(supplierId);

if (contactsInfoVO != null) {

deviceInfoVO.setSupplierContactsName(contactsInfoVO.getContactName());

deviceInfoVO.setSupplierContactPhone(contactsInfoVO.getTelephone());

}

return deviceInfoVO;

}

/\*\*

\* 修改设备信息

\*

\* @param deviceInfoDTO

\* @return void

\* @author wanggang

\* @date 2023/5/8 11:30

\*/

@Override

@Transactional(rollbackFor = Exception.class)

public Boolean updateDeviceInfo(DeviceInfoDTO deviceInfoDTO) {

if (null == deviceInfoDTO || StringUtils.isEmpty(deviceInfoDTO.getId())) {

throw new ParameterException("修改参数缺失");

}

DeviceInfo deviceInfo = new DeviceInfo();

JcglBeanUtil.copyProperties(deviceInfoDTO, deviceInfo, true);

return this.updateById(deviceInfo);

}

/\*\*

\* 逻辑删除设备信息

\*

\* @param id 逻辑删除Id

\* @return void

\* @author wanggang

\* @date 2023/5/8 11:30

\*/

@Override

@Transactional(rollbackFor = Exception.class)

public Boolean deleteDeviceInfo(String id) {

if (StringUtils.isEmpty(id)) {

throw new ParameterException("逻辑删除Id不能为空!");

}

DeviceInfo deviceInfo = getById(id);

//设备分类ID

String deviceClassifyInfoId = deviceInfo.getDeviceClassifyInfoId();

boolean flag = this.removeById(id);

if (flag) {

//查询设备的数量

Integer quantity = deviceClassifyInfoMapper.selectDeviceQuantity(deviceClassifyInfoId);

if (null != quantity) {

Integer addQuantity = quantity - 1;

//修改设备数量

deviceClassifyInfoMapper.updateDeviceQuantity(addQuantity, deviceClassifyInfoId);

}

//删除设备关联的其他数据

//逻辑删除资产信息

deviceAssetInfoRecordMapper.updateDelFlagByDeviceInfoId(id);

//逻辑删除设备借用信息

deviceBorrowRecordMapper.updateDelFlagByDeviceInfoId(id);

//逻辑删除设备流转记录

deviceCirculationRecordMapper.updateDelFlagByDeviceInfoId(id);

//逻辑删除设备保养维修记录

deviceRepairMaintenanceRecordMapper.updateDelFlagByDeviceInfoId(id);

//逻辑删除设备检定记录

deviceVerificationRecordMapper.updateDelFlagByDeviceInfoId(id);

//逻辑删除设备运行记录

deviceWorkingRecordMapper.updateDelFlagByDeviceInfoId(id);

}

return flag;

}

/\*\*

\* 导入设备信息数据

\*

\* @param file

\* @return java.lang.Boolean

\* @author wanggang

\* @date 2023/5/15 17:15

\*/

@SneakyThrows

@Override

@Transactional(rollbackFor = Exception.class)

public void importDeviceInfo(MultipartFile file, Boolean isOverride) {

List<DeviceInfoDTO> deviceInfoDTOList = new ArrayList<>();

try {

deviceInfoDTOList = EasyExcelUtil.readExcelData(DeviceInfoDTO.class, file);

if (deviceInfoDTOList.isEmpty()) {

throw new ServiceException("导入失败!");

}

} catch (Exception e) {

e.printStackTrace();

}

List<DeviceInfo> deviceInfoList = new ArrayList<>();

List<DeviceAssetInfoRecord> deviceAssetInfoRecordList = new ArrayList<>();

String error = importDetail(deviceInfoDTOList, deviceInfoList, deviceAssetInfoRecordList);

//根据设备编号查询

List<String> deviceNumberList = deviceInfoList.stream().map(DeviceInfo::getDeviceNumber).filter(Objects::nonNull).collect(Collectors.toList());

List<DeviceInfoVO> deviceInfoVO = new ArrayList<>();

if (CollUtil.isNotEmpty(deviceNumberList)) {

deviceInfoVO = deviceInfoMapper.selectDeviceByDeviceNumber(deviceNumberList);

}

//资产编号集

List<String> assetCodeList = deviceAssetInfoRecordList.stream().map(DeviceAssetInfoRecord::getAssetCode).filter(Objects::nonNull).collect(Collectors.toList());

// 如果有错误则抛出

if (error != null) {

throw new ServiceException(error);

}

//判断是覆盖更新(true)还是先删除后插入(false)

if (isOverride) {

//批量更新设备信息 如果有则更新没有则插入

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

DeviceInfo deviceInfo = deviceInfoList.get(i);

String deviceNumber = deviceInfo.getDeviceNumber();

List<DeviceInfoVO> collect = deviceInfoVO.stream().filter(f -> null != f.getDeviceNumber() && f.getDeviceNumber().equals(deviceNumber)).collect(Collectors.toList());

//如果为空则新增这条数据 不为空则更新这条数据

if (collect.isEmpty()) {

//新增资产信息

this.save(deviceInfo);

String id = deviceInfo.getId();

DeviceAssetInfoRecord deviceAssetInfoRecord = deviceAssetInfoRecordList.get(i);

deviceAssetInfoRecord.setDeviceInfoId(id);

iDeviceAssetInfoRecordService.save(deviceAssetInfoRecord);

} else {

DeviceInfoVO deviceInfoVO1 = collect.get(0);

String id = deviceInfoVO1.getId();

deviceInfo.setId(id);

this.updateById(deviceInfo);

//修改资产信息

DeviceAssetInfoRecord deviceAssetInfoRecord = deviceAssetInfoRecordList.get(i);

//根据资产编号修改资产信息

String assetCode = deviceAssetInfoRecord.getAssetCode();

//修改资产数据

deviceAssetInfoRecordMapper.updateDeviceAssetInfoRecord(deviceAssetInfoRecord, assetCode);

}

}

} else {

//删除设备信息

deviceInfoMapper.deleteDeviceInfoBatch(deviceNumberList);

//删除资产信息

deviceAssetInfoRecordMapper.deleteDeviceAssetInfoRecordBatch(assetCodeList);

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

DeviceInfo deviceInfo = deviceInfoList.get(i);

//新增设备信息

this.save(deviceInfo);

//新增资产信息

DeviceAssetInfoRecord deviceAssetInfoRecord = deviceAssetInfoRecordList.get(i);

String id = deviceInfo.getId();

deviceAssetInfoRecord.setDeviceInfoId(id);

iDeviceAssetInfoRecordService.save(deviceAssetInfoRecord);

}

}

}

/\*\*

\* 导入设备信息数据的处理

\*

\* @param deviceInfoDTOList, deviceInfoList

\* @return java.lang.String

\* @author wanggang

\* @date 2023/5/16 13:43

\*/

private String importDetail(List<DeviceInfoDTO> deviceInfoDTOList, List<DeviceInfo> deviceInfoList, List<DeviceAssetInfoRecord> deviceAssetInfoRecordLis) throws IllegalAccessException {

//部门名称

List<String> departmentNameList = deviceInfoDTOList.stream().distinct().map(DeviceInfoDTO::getDepartmentName).filter(Objects::nonNull)

.collect(Collectors.toList());

List<SysOrgVO> sysOrgVO = new ArrayList<>();

if (CollUtil.isNotEmpty(departmentNameList)) {

sysOrgVO = sysOrgMapper.selectOrgByName(departmentNameList);

}

//分类编号

List<String> deviceClassifyNumberList = deviceInfoDTOList.stream().distinct().map(DeviceInfoDTO::getDeviceClassifyNumber).filter(Objects::nonNull)

.collect(Collectors.toList());

List<DeviceClassifyInfoVO> deviceClassifyInfoVO = new ArrayList<>();

if (CollUtil.isNotEmpty(deviceClassifyNumberList)) {

deviceClassifyInfoVO = deviceClassifyInfoMapper.selectIdByDeviceClassifyNumber(deviceClassifyNumberList);

}

//数据字典

List<SysDictVO> sysDictVO = sysDictMapper.selectDict(null);

//责任人

List<String> personLiableNameList = deviceInfoDTOList.stream().distinct().map(DeviceInfoDTO::getPersonLiableName).filter(Objects::nonNull)

.collect(Collectors.toList());

List<SysUserVO> sysUserVO = new ArrayList<>();

if (CollUtil.isNotEmpty(personLiableNameList)) {

sysUserVO = sysUserMapper.selectIdByUserName(personLiableNameList);

}

//存放编号

List<String> storageCodeList = deviceInfoDTOList.stream().distinct().map(DeviceInfoDTO::getStorageCode).filter(Objects::nonNull)

.collect(Collectors.toList());

List<DeviceShelvesInfoVO> deviceShelvesInfoVO = new ArrayList<>();

if (CollUtil.isNotEmpty(storageCodeList)) {

deviceShelvesInfoVO = deviceShelvesInfoMapper.selectIdByShelvesCode(storageCodeList);

}

// 记录错误信息

StringBuilder error = new StringBuilder();

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

DeviceInfoDTO deviceInfoDTO = deviceInfoDTOList.get(i);

DeviceInfo deviceInfo = new DeviceInfo();

DeviceAssetInfoRecord deviceAssetInfoRecord = new DeviceAssetInfoRecord();

int count = 0;

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

f.setAccessible(true);

if (f.get(deviceInfoDTO) == null || "".equals(f.get(deviceInfoDTO))) {

count = count + 1;

}

}

// 行错误信息

StringBuilder tempError = new StringBuilder();

int length = deviceInfoDTO.getClass().getDeclaredFields().length;

count = count - length;

if (count == 0) {

throw new ServiceException("导入的数据存在空格符或其他无效信息，请重新核对后再导入!");

}

//部门

String departmentName = deviceInfoDTO.getDepartmentName();

if (StringUtils.isNotEmpty(departmentName)) {

List<SysOrgVO> collect = sysOrgVO.stream().filter(f -> null != f.getOrgName() && f.getOrgName().equals(departmentName))

.collect(Collectors.toList());

if (collect.isEmpty()) {

tempError.append("没有找到该使用部门");

} else {

String departmentId = collect.get(0).getId();

deviceInfo.setDepartmentId(departmentId);

}

} else {

tempError.append("使用部门不能为空,");

}

//设备编号

String deviceNumber = deviceInfoDTO.getDeviceNumber();

//判断设备编号在excel表格中出现的次数 如果出现超过1次 则设备编号重复 提示用户

List<String> collect = deviceInfoDTOList.stream().map(DeviceInfoDTO::getDeviceNumber)

.filter(Objects::nonNull).collect(Collectors.toList());

int frequency = Collections.frequency(collect, deviceNumber);

if (frequency > 1) {

tempError.append("导入模板中设备编号").append(deviceNumber).append("重复了");

}

if (StringUtils.isNotEmpty(deviceNumber)) {

deviceInfo.setDeviceNumber(deviceNumber);

} else {

tempError.append("设备编号不能为空,");

}

//分类编号

String deviceClassifyNumber = deviceInfoDTO.getDeviceClassifyNumber();

if (StringUtils.isNotEmpty(deviceClassifyNumber)) {

List<DeviceClassifyInfoVO> collect1 = deviceClassifyInfoVO.stream().filter(f -> null != f.getDeviceClassifyNumber()

&& f.getDeviceClassifyNumber().equals(deviceClassifyNumber)).collect(Collectors.toList());

if (collect1.isEmpty()) {

tempError.append("没有找到该设备分类,");

} else {

String id = collect1.get(0).getId();

deviceInfo.setDeviceClassifyInfoId(id);

}

} else {

tempError.append("设备分类编号不能为空,");

}

//出厂编号

String exWorksNumber = deviceInfoDTO.getExworksNumber();

if (StringUtils.isNotEmpty(exWorksNumber)) {

deviceInfo.setExworksNumber(exWorksNumber);

} else {

tempError.append("出厂编号不能为空,");

}

//量值溯源(device\_asset\_source)

String assetSource = deviceInfoDTO.getAssetSource();

if (StringUtils.isNotEmpty(assetSource)) {

List<SysDictVO> collect1 = sysDictVO.stream().filter(f -> null != f.getCategoryKey() && "device\_asset\_source".equals(f.getCategoryKey()))

.filter(f -> null != f.getLabel() && f.getLabel().equals(assetSource)).collect(Collectors.toList());

if (collect1.isEmpty()) {

tempError.append("没有找到该量值溯源的值,");

} else {

String val = collect1.get(0).getVal();

deviceInfo.setAssetSource(val);

}

} else {

tempError.append("量值溯源不能为空,");

}

//责任人

String personLiableName = deviceInfoDTO.getPersonLiableName();

if (StringUtils.isNotEmpty(personLiableName)) {

List<SysUserVO> collect1 = sysUserVO.stream().filter(f -> null != f.getEmpName() && f.getEmpName().equals(personLiableName))

.collect(Collectors.toList());

if (collect1.isEmpty()) {

tempError.append("该责任人不存在,");

} else {

String id = collect1.get(0).getId();

deviceInfo.setPersonLiable(id);

}

} else {

tempError.append("责任人不能为空,");

}

//购置日期

String purchaseTime = deviceInfoDTO.getPurchaseTimeTran();

if (StringUtils.isNotEmpty(purchaseTime)) {

Date date = DateUtil.strToDate("yyyy-MM-dd", purchaseTime.replace("/", "-"));

deviceInfo.setPurchaseTime(date);

} else {

tempError.append("购置日期不能为空,");

}

//使用状态(device\_use\_status)

String useStatusDisplay = deviceInfoDTO.getUseStatusDisplay();

if (StringUtils.isNotEmpty(useStatusDisplay)) {

List<SysDictVO> collect1 = sysDictVO.stream().filter(f -> null != f.getCategoryKey() && "device\_use\_status".equals(f.getCategoryKey()))

.filter(f -> null != f.getLabel() && f.getLabel().equals(useStatusDisplay)).collect(Collectors.toList());

if (collect1.isEmpty()) {

tempError.append("没有找到该使用状态的值,");

} else {

String val = collect1.get(0).getVal();

deviceInfo.setUseStatus(val);

}

} else {

tempError.append("使用状态不能为空,");

}

//存放编号

String storageCode = deviceInfoDTO.getStorageCode();

String roomName = deviceInfoDTO.getRoomName();

if (StringUtils.isNotEmpty(storageCode)) {

List<DeviceShelvesInfoVO> collect1 = deviceShelvesInfoVO.stream().filter(f -> null != f.getStorageCode()

&& f.getStorageCode().equals(storageCode)).filter(f -> null != f.getRoomName()

&& f.getRoomName().equals(roomName)).collect(Collectors.toList());

if (collect1.isEmpty()) {

tempError.append("没有找到该货架,");

} else {

String id = collect1.get(0).getId();

deviceInfo.setDeviceShelvesInfoId(id);

}

}

//是否自动采集(0否;1是)

String automaticCollection = deviceInfoDTO.getAutomaticCollectionTran();

if (StringUtils.isNotEmpty(automaticCollection)) {

if ("是".equals(automaticCollection)) {

deviceInfo.setAutomaticCollection(1);

} else if ("否".equals(automaticCollection)) {

deviceInfo.setAutomaticCollection(0);

} else {

tempError.append("请填写是或者否,");

}

} else {

tempError.append("是否自动采集不能为空,");

}

//检定周期(device\_verification\_cycle)

String verificationCycle = deviceInfoDTO.getVerificationCycle();

if (StringUtils.isNotEmpty(verificationCycle)) {

List<SysDictVO> collect1 = sysDictVO.stream().filter(f -> null != f.getCategoryKey() && "device\_verification\_cycle".equals(f.getCategoryKey()))

.filter(f -> null != f.getLabel() && f.getLabel().equals(verificationCycle)).collect(Collectors.toList());

if (collect1.isEmpty()) {

tempError.append("没有找到该检定周期的值,");

} else {

String val = collect1.get(0).getVal();

deviceInfo.setVerificationCycle(val);

}

} else {

tempError.append("检定周期不能为空,");

}

//出厂日期

String exWorksTime = deviceInfoDTO.getExworksTimeTran();

if (StringUtils.isNotEmpty(exWorksTime)) {

Date date = DateUtil.strToDate("yyyy-MM-dd", exWorksTime.replace("/", "-"));

deviceInfo.setExworksTime(date);

} else {

tempError.append("出厂日期不能为空,");

}

//设备编号(交通行业)

String trafficDeviceNumber = deviceInfoDTO.getTrafficDeviceNumber();

deviceInfo.setTrafficDeviceNumber(trafficDeviceNumber);

//设备名称(交通行业)

String trafficDeviceName = deviceInfoDTO.getTrafficDeviceName();

deviceInfo.setTrafficDeviceName(trafficDeviceName);

//在账状态(device\_on\_account\_status)

String onAccountStatusDisplay = deviceInfoDTO.getOnAccountStatusDisplay();

if (StringUtils.isNotEmpty(onAccountStatusDisplay)) {

List<SysDictVO> collect1 = sysDictVO.stream().filter(f -> null != f.getCategoryKey() && "device\_on\_account\_status".equals(f.getCategoryKey()))

.filter(f -> null != f.getLabel() && f.getLabel().equals(onAccountStatusDisplay)).collect(Collectors.toList());

if (collect1.isEmpty()) {

tempError.append("没有找到该在账状态的值,");

} else {

String val = collect1.get(0).getVal();

deviceAssetInfoRecord.setOnAccountStatus(val);

}

} else {

tempError.append("在账状态不能为空,");

}

//资产编号

String assetCode = deviceInfoDTO.getAssetCode();

if (StringUtils.isNotEmpty(assetCode)) {

deviceAssetInfoRecord.setAssetCode(assetCode);

} else {

tempError.append("资产编号不能为空,");

}

//判断资产编号在excel表格中出现的次数 如果出现超过1次 则资产编号重复 提示用户

List<String> collect1 = deviceInfoDTOList.stream().map(DeviceInfoDTO::getAssetCode)

.filter(Objects::nonNull).collect(Collectors.toList());

int frequency1 = Collections.frequency(collect1, assetCode);

if (frequency1 > 1) {

tempError.append("导入模板中资产编号").append(assetCode).append("重复了");

}

//登记日期

String registerDate = deviceInfoDTO.getRegisterDateTran();

if (StringUtils.isNotEmpty(registerDate)) {

Date date = DateUtil.strToDate("yyyy-MM-dd", registerDate.replace("/", "-"));

deviceAssetInfoRecord.setRegisterDate(date);

} else {

tempError.append("登记日期不能为空,");

}

//设备价格

BigDecimal devicePrice = deviceInfoDTO.getDevicePrice();

if (null != devicePrice) {

deviceAssetInfoRecord.setDevicePrice(devicePrice);

} else {

tempError.append("设备价格不能为空,");

}

deviceAssetInfoRecordLis.add(deviceAssetInfoRecord);

//备注

String remarks = deviceInfoDTO.getRemarks();

deviceInfo.setRemarks(remarks);

// 如果有错误信息

if (StringUtils.isNotEmpty(tempError.toString())) {

tempError.insert(0, "第" + (i + 1) + "行：");

tempError.deleteCharAt(tempError.length() - 1);

tempError.append(";");

error.append(tempError);

}

deviceInfoList.add(deviceInfo);

}

// 如果有错误信息

if (StringUtils.isNotEmpty(error.toString())) {

error.deleteCharAt(error.length() - 1);

return error.toString();

}

return null;

}

/\*\*

\* 导出设备信息

\*

\* @param deviceInfoSelectDTO

\* @return void

\* @author wanggang

\* @date 2023/5/17 14:10

\*/

@Override

public void exportDeviceInfo(DeviceInfoSelectDTO deviceInfoSelectDTO, HttpServletResponse response) {

// 查询资源数据

Page<DeviceInfoVO> deviceInfoPage = this.selectDevicePage(deviceInfoSelectDTO);

String fileName = getFileName();

try {

EasyExcelUtil.writeExcel(response, DeviceInfoVO.class, deviceInfoPage.getRecords(), fileName);

} catch (IOException | IllegalAccessException | NoSuchFieldException e) {

throw new ServiceException(e);

}

}

private String getFileName() {

String start = "设备信息表";

String date = DateUtil.dateToStr("yyyyMMddHHmmss", new Date());

String suffix = ".xlsx";

return start + date + suffix;

}

}

/\*\*

\* @author Administrator

\* @description 针对表【device\_place(场所信息)】的数据库操作Service实现

\* @createDate 2023-05-23 09:23:24

\*/

@Service

@Slf4j

public class DevicePlaceServiceImpl extends ServiceImpl<DevicePlaceMapper, DevicePlace>

implements DevicePlaceService {

@Resource

private ISysOrgService sysOrgService;

@Resource

private DeviceRoomService deviceRoomService;

@Resource

private DeviceShelvesInfoService deviceShelvesInfoService;

@Override

public Page<DevicePlaceVO> findByPage(CommonSelectEntity dto, boolean isPage) {

Page<DevicePlaceVO> page = new Page<>(dto.getCurrent(), dto.getPageSize());

dto.dealTime();

if (isPage) {

baseMapper.findPlaceByPage(page, dto);

} else {

page.setRecords(baseMapper.findPlaceByPage(dto));

}

return page;

}

@Override

public DevicePlaceVO selectById(String id) {

if (StringUtils.isEmpty(id)) return null;

DevicePlace devicePlace = baseMapper.selectById(id);

return BeanUtil.copyProperties(devicePlace, DevicePlaceVO.class);

}

@Override

public int logicDeleteByIds(String[] ids, String tenantId) {

if (StringUtils.isEmpty(tenantId)) {

tenantId = TenantContextHolder.getContext().getTenantContextInfo().getTenantId();

}

return baseMapper.logicDelete(ids, tenantId);

}

private void insertBatch(List<SysOrg> subOrgList, SysUser sysUser, char code) {

List<DevicePlace> data = new ArrayList<>();

for (SysOrg s : subOrgList) {

DevicePlace devicePlace = new DevicePlace();

devicePlace.setUpdateTime(new Date());

devicePlace.setUpdateBy(sysUser.getId());

devicePlace.setUpdateName(sysUser.getUsername());

devicePlace.setOrgId(s.getId());

devicePlace.setPlaceName(s.getOrgName());

devicePlace.setPlaceCode(String.valueOf(code++));

if (code > 'Z') {

throw new ServiceException("请联系系统管理员解决组织编号不够使用的问题！");

}

data.add(devicePlace);

}

this.saveBatch(data);

}

@Override

@Transactional(rollbackFor = Exception.class)

public void add() {

// 1、查询子部门信息

List<SysOrg> orgByParentId = sysOrgService.findOrgByParentId("0");

SysUser sysUser = TenantContextHolder.getContext().getTenantContextInfo().getSysUser();

String tenantId = sysUser.getTenantId();

if (orgByParentId.isEmpty()) {

log.info(tenantId + "组织没有部门(组织信息)");

return;

}

List<SysOrg> subOrgList = sysOrgService.findOrgByParentId(orgByParentId.get(0).getId());

if (subOrgList.isEmpty()) {

log.info(orgByParentId.get(0).getId() + "-" + orgByParentId.get(0).getOrgName() + "组织没有子部门(组织信息)");

return;

}

// 2、查询场所信息

CommonSelectEntity entity = new CommonSelectEntity();

entity.setTenantId(tenantId);

List<DevicePlaceVO> place = baseMapper.findPlaceByPage(entity);

// 场所信息为空，全部插入

if (place.isEmpty()) {

char code = 'A';

this.insertBatch(subOrgList, sysUser, code);

} else {

Map<String, DevicePlaceVO> placeVOMap = place.stream().collect(Collectors.toMap(DevicePlaceVO::getOrgId, s -> s));

// 查询组织下面的最大编码

char maxCode = baseMapper.findmMaxCode(tenantId);

maxCode += 1;

List<SysOrg> newAddOrg = new ArrayList<>();

for (SysOrg org : subOrgList) {

if (placeVOMap.containsKey(org.getId())) {

DevicePlaceVO vo = placeVOMap.get(org.getId());

DevicePlace devicePlace = BeanUtil.copyProperties(vo, DevicePlace.class);

devicePlace.setPlaceName(org.getOrgName());

baseMapper.updateById(devicePlace);

} else {

newAddOrg.add(org);

}

}

this.insertBatch(newAddOrg, sysUser, maxCode);

}

}

@Override

public boolean update(DevicePlaceDTO update) {

if (update == null) return true;

DevicePlace devicePlace = BeanUtil.copyProperties(update, DevicePlace.class);

if (update.getId() == null) {

throw new ParameterException("请求的主键没有传入");

}

DevicePlace old = baseMapper.selectById(update.getId());

if (old == null) {

throw new ParameterException("请求更新的设备分类数据不存在，不能更新");

}

// 查询剩余编码

List<Character> surplusChar = this.selectSurplusPlaceCode();

if (StringUtils.isNotEmpty(update.getPlaceCode())) {

char placeCode = update.getPlaceCode().charAt(0);

char oldPlaceCode = old.getPlaceCode().charAt(0);

if (oldPlaceCode != placeCode) {

// 查询旧的编码是否绑定了存放室，如果绑定则不允许修改

List<DeviceRoomVO> deviceRoomVOS = deviceRoomService.selectByPlaceId(old.getId());

if (CollUtil.isNotEmpty(deviceRoomVOS)) {

throw new ParameterException("修改的【" + update.getPlaceName() + "】场所已经绑定存放室，请不要修改编码");

}

if (!surplusChar.contains(placeCode)) {

throw new ParameterException("修改【" + update.getPlaceName() + "】时，您提交的编码【" + placeCode + "】已经在使用，请不要重复提交");

}

}

}

return baseMapper.updateById(devicePlace) == 1;

}

@Override

public List<DevicePlaceTreeVO> treePlaceAndRoom() {

List<DevicePlaceTreeVO> result = new ArrayList<>();

CommonSelectEntity dto = new CommonSelectEntity();

SysUser sysUser = TenantContextHolder.getContext().getTenantContextInfo().getSysUser();

dto.setTenantId(sysUser.getTenantId());

List<DevicePlaceVO> placeByPage = baseMapper.findPlaceByPage(dto);

if (placeByPage.isEmpty()) {

return result;

}

placeByPage.forEach(p -> {

DevicePlaceTreeVO vo = BeanUtil.copyProperties(p, DevicePlaceTreeVO.class);

result.add(vo);

// 查询房间信息

List<DeviceRoomVO> deviceRoomVOS = deviceRoomService.selectByPlaceId(p.getId());

vo.setDeviceRoomVOS(deviceRoomVOS);

});

return result;

}

@Override

public List<Map<String, Object>> treePlaceAndRoomAndShelves() {

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

// 查询所有存放室

CommonSelectEntity dto = new CommonSelectEntity();

SysUser sysUser = TenantContextHolder.getContext().getTenantContextInfo().getSysUser();

dto.setTenantId(sysUser.getTenantId());

List<DevicePlaceVO> placeByPage = baseMapper.findPlaceByPage(dto);

for (DevicePlaceVO first : placeByPage) {

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

firstMap.put("place\_code", first.getPlaceCode());

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

this.dealMap(result, firstMap, first.getPlaceName(), first.getId(), subList);

// 查询房间信息

List<DeviceRoomVO> deviceRoomVOS = deviceRoomService.selectByPlaceId(first.getId());

for (DeviceRoomVO deviceRoomVO : deviceRoomVOS) {

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

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

secondMap.put("roomCode", deviceRoomVO.getRoomCode());

this.dealMap(subList, secondMap, deviceRoomVO.getRoomCode() + ":" + deviceRoomVO.getRoomName(), deviceRoomVO.getId(), thirdList);

// 查询所有货架

List<DeviceShelvesInfo> deviceShelvesInfos = deviceShelvesInfoService.selectByRoomId(deviceRoomVO.getId());

Set<String> shelvesCode = deviceShelvesInfos.stream().map(DeviceShelvesInfo::getShelvesCode).collect(Collectors.toSet());

// 获取所有的货架

for (String code : shelvesCode) {

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

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

this.dealMap(thirdList, thirdMap, code + ":" + deviceShelvesInfoService.selectShelvesNameByCode(code),

code, fourthList);

// 获取这个货架的层号

List<String> levelData = deviceShelvesInfos.stream()

.filter(d -> Objects.equals(d.getShelvesCode(), code))

.map(DeviceShelvesInfo::getLayerCode)

.distinct().sorted()

.toList();

for (String level : levelData) {

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

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

this.dealMap(fourthList, fourMap, "第" + level + "层", level + System.nanoTime(), fifthList);

List<DeviceShelvesInfo> levelShelves = deviceShelvesInfos.stream()

.filter(d -> Objects.equals(d.getShelvesCode(), code) && d.getLayerCode().equals(level))

.sorted(Comparator.comparing(DeviceShelvesInfo::getPositionCode))

.toList();

for (DeviceShelvesInfo levelShelf : levelShelves) {

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

fifthMap.put("roomName", levelShelf.getRoomName());

fifthMap.put("shelvesId", levelShelf.getId());

this.dealMap(fifthList, fifthMap, "第" + levelShelf.getPositionCode() + "号", levelShelf.getStorageCode(), Collections.emptyList());

}

}

}

}

}

return result;

}

private void dealMap(List<Map<String, Object>> result, Map<String, Object> tempMap, String title, String value, List<Map<String, Object>> subList) {

result.add(tempMap);

tempMap.put(SystemConstant.TREE\_TITLE, title);

tempMap.put(SystemConstant.TREE\_VALUE, value);

tempMap.put(SystemConstant.TREE\_CHILDREN, subList);

}

@Override

public List<Character> selectSurplusPlaceCode() {

List<Character> characters = baseMapper.selectPlaceCode(TenantContextHolder.getContext().getTenantContextInfo().getTenantId());

List<Character> surplusChar = new ArrayList<>();

for (char i = 'A'; i <= 'Z'; i++) {

if (!characters.contains(i)) {

surplusChar.add(i);

}

}

return surplusChar;

}

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_repair\_maintenance\_record(设备保养维修记录表)】的数据库操作Service实现

\* @createDate 2023-05-10 10:26:14

\*/

@Service

public class DeviceRepairMaintenanceRecordServiceImpl extends ServiceImpl<DeviceRepairMaintenanceRecordMapper, DeviceRepairMaintenanceRecord>

implements IDeviceRepairMaintenanceRecordService {

@Resource

private SysUserMapper sysUserMapper;

@Resource

private SysDictMapper sysDictMapper;

@Resource

private FileAttachMapper fileAttachMapper;

/\*\*

\* 分页查询设备保养维修记录

\*

\* @param deviceRepairMaintenanceRecordSelectDTO

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceRepairMaintenanceRecordVO>

\* @author wanggang

\* @date 2023/5/23 15:27

\*/

@Override

public Page<DeviceRepairMaintenanceRecordVO> selectDeviceRepairMaintenanceRecordPage(DeviceRepairMaintenanceRecordSelectDTO deviceRepairMaintenanceRecordSelectDTO) {

if (null == deviceRepairMaintenanceRecordSelectDTO) {

throw new ParameterException("查询参数缺失");

}

Page<DeviceRepairMaintenanceRecordVO> page = new Page<>(deviceRepairMaintenanceRecordSelectDTO.getCurrent(), deviceRepairMaintenanceRecordSelectDTO.getPageSize());

Page<DeviceRepairMaintenanceRecordVO> deviceRepairMaintenanceRecordPage = baseMapper.selectDeviceRepairMaintenanceRecordPage(page, deviceRepairMaintenanceRecordSelectDTO);

List<DeviceRepairMaintenanceRecordVO> records = deviceRepairMaintenanceRecordPage.getRecords();

//查询数据字典

//发出/接收状态 device\_situation

List<SysDictVO> deviceStatus = sysDictMapper.selectDict("device\_situation");

for (DeviceRepairMaintenanceRecordVO record : records) {

String runUseStatus = record.getRunUseStatus();

Integer calibrationResult = record.getCalibrationResult();

String textByValue = DeviceResultEnum.getTextByValue(calibrationResult);

record.setCalibrationResultDisplay(textByValue);

List<SysDictVO> sysDictVOList = deviceStatus.stream().filter(f -> null != f.getVal() && f.getVal().equals(runUseStatus)).collect(Collectors.toList());

if (CollUtil.isNotEmpty(sysDictVOList)) {

record.setRunUseStatusDisplay(sysDictVOList.get(0).getLabel());

}

String involvedStaff = record.getInvolvedStaff();

List<String> involvedStaffIdList = Arrays.asList(involvedStaff.split("、"));

List<String> collect = involvedStaffIdList.stream().distinct().collect(Collectors.toList());

//查询对应的参与人员名称

List<SysUserVO> sysUserVO = sysUserMapper.selectNameByUserId(collect);

List<String> nameList = sysUserVO.stream().map(SysUserVO::getEmpName).filter(Objects::nonNull).collect(Collectors.toList());

String involvedStaffName = nameList.stream().map(String::valueOf).collect(Collectors.joining("、"));

record.setInvolvedStaffName(involvedStaffName);

//处理文件信息

String fileId = record.getFileId();

if (StringUtils.isNotEmpty(fileId)) {

List<String> fileIdList = Arrays.asList(fileId.split("、"));

List<String> collect1 = fileIdList.stream().distinct().collect(Collectors.toList());

//查询对应文件信息

List<FileAttachVO> fileAttachVOList = fileAttachMapper.selectFileByFileId(collect1);

record.setFileAttachVOList(fileAttachVOList);

String uuId = fileAttachVOList.stream().map(FileAttachVO::getUuid)

.filter(Objects::nonNull).collect(Collectors.joining("、"));

String fileName = fileAttachVOList.stream().map(FileAttachVO::getFilename)

.filter(Objects::nonNull).collect(Collectors.joining("、"));

List<Date> collect2 = fileAttachVOList.stream().map(FileAttachVO::getCreateTime)

.filter(Objects::nonNull).collect(Collectors.toList());

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

for (Date date : collect2) {

String dateString = DateUtil.dateToStr("yyyy-MM-dd HH:mm:ss", date);

dateList.add(dateString);

}

String uploadTime = String.join("、", dateList);

record.setFileIdList(collect1);

record.setFilePath(uuId);

record.setFileName(fileName);

record.setUploadTime(uploadTime);

}

}

return deviceRepairMaintenanceRecordPage;

}

/\*\*

\* 根据设备维修保养记录ID查询修改回显数据

\*

\* @param id

\* @return com.cstc.jcgl.domain.device.vo.DeviceRepairMaintenanceRecordVO

\* @author wanggang

\* @date 2023/5/23 14:21

\*/

@Override

public DeviceRepairMaintenanceRecordVO selectDeviceRepairMaintenanceRecordById(String id) {

if (StringUtils.isEmpty(id)) {

throw new ParameterException("查询参数缺失");

}

//根据设备修保养记录ID查询修改回显数据

DeviceRepairMaintenanceRecordVO deviceRepairMaintenanceRecordVO = baseMapper.selectDeviceRepairMaintenanceRecordById(id);

if (null != deviceRepairMaintenanceRecordVO) {

String involvedStaff = deviceRepairMaintenanceRecordVO.getInvolvedStaff();

if (StringUtils.isNotEmpty(involvedStaff)) {

List<String> involvedStaffIdList = Arrays.asList(involvedStaff.split("、"));

List<String> collect = involvedStaffIdList.stream().distinct().collect(Collectors.toList());

//查询对应的参与人员名称

List<SysUserVO> sysUserVO = sysUserMapper.selectNameByUserId(collect);

String involvedStaffName = sysUserVO.stream().map(SysUserVO::getEmpName).filter(Objects::nonNull)

.collect(Collectors.joining("、"));

deviceRepairMaintenanceRecordVO.setInvolvedStaffName(involvedStaffName);

deviceRepairMaintenanceRecordVO.setInvolvedStaffList(sysUserVO);

}

String fileId = deviceRepairMaintenanceRecordVO.getFileId();

if (StringUtils.isNotEmpty(fileId)) {

List<String> fileIdList = Arrays.asList(fileId.split("、"));

List<String> collect1 = fileIdList.stream().distinct().collect(Collectors.toList());

//查询文件对应信息

List<FileAttachVO> fileAttachVOList = fileAttachMapper.selectFileByFileId(collect1);

deviceRepairMaintenanceRecordVO.setFileAttachVOList(fileAttachVOList);

}

}

return deviceRepairMaintenanceRecordVO;

}

/\*\*

\* 新增设备保养维修记录

\*

\* @param deviceRepairMaintenanceRecordDTO

\* @return void

\* @author wanggang

\* @date 2023/5/10 11:30

\*/

@Override

public Boolean addDeviceRepairMaintenanceRecord(DeviceRepairMaintenanceRecordDTO deviceRepairMaintenanceRecordDTO) {

if (null == deviceRepairMaintenanceRecordDTO) {

throw new ParameterException("新增参数缺失");

}

DeviceRepairMaintenanceRecord deviceRepairMaintenanceRecord = new DeviceRepairMaintenanceRecord();

List<String> fileIdList = deviceRepairMaintenanceRecordDTO.getFileIdList();

if (CollUtil.isNotEmpty(fileIdList)) {

String fileId = String.join("、", fileIdList);

deviceRepairMaintenanceRecordDTO.setFileId(fileId);

}

//取出参与人员的id处理成字符串

List<String> involvedStaffList = deviceRepairMaintenanceRecordDTO.getInvolvedStaffList();

if (CollUtil.isNotEmpty(involvedStaffList)) {

String involvedStaff = involvedStaffList.stream().map(String::valueOf).collect(Collectors.joining("、"));

deviceRepairMaintenanceRecordDTO.setInvolvedStaff(involvedStaff);

}

JcglBeanUtil.copyProperties(deviceRepairMaintenanceRecordDTO, deviceRepairMaintenanceRecord, true);

return this.save(deviceRepairMaintenanceRecord);

}

/\*\*

\* 修改设备保养维修记录

\*

\* @param deviceRepairMaintenanceRecordDTO

\* @return void

\* @author wanggang

\* @date 2023/5/10 11:30

\*/

@Override

public Boolean updateDeviceRepairMaintenanceRecord(DeviceRepairMaintenanceRecordDTO deviceRepairMaintenanceRecordDTO) {

if (null == deviceRepairMaintenanceRecordDTO || StringUtils.isEmpty(deviceRepairMaintenanceRecordDTO.getId())) {

throw new ParameterException("修改参数缺失");

}

DeviceRepairMaintenanceRecord deviceRepairMaintenanceRecord = new DeviceRepairMaintenanceRecord();

List<String> fileIdList = deviceRepairMaintenanceRecordDTO.getFileIdList();

if (CollUtil.isNotEmpty(fileIdList)) {

String fileId = String.join("、", fileIdList);

deviceRepairMaintenanceRecordDTO.setFileId(fileId);

}

//取出参与人员的id处理成字符串

List<String> involvedStaffList = deviceRepairMaintenanceRecordDTO.getInvolvedStaffList();

if (CollUtil.isNotEmpty(involvedStaffList)) {

String involvedStaff = involvedStaffList.stream().map(String::valueOf).collect(Collectors.joining("、"));

deviceRepairMaintenanceRecordDTO.setInvolvedStaff(involvedStaff);

}

JcglBeanUtil.copyProperties(deviceRepairMaintenanceRecordDTO, deviceRepairMaintenanceRecord, true);

return this.updateById(deviceRepairMaintenanceRecord);

}

/\*\*

\* 逻辑删除设备保养维修记录

\*

\* @param id

\* @return void

\* @author wanggang

\* @date 2023/5/10 11:30

\*/

@Override

public Boolean deleteDeviceRepairMaintenanceRecord(String id) {

if (StringUtils.isEmpty(id)) {

throw new ParameterException("逻辑删除Id不能为空!");

}

return this.removeById(id);

}

}

/\*\*

\* @author Administrator

\* @description 针对表【device\_room】的数据库操作Service实现

\* @createDate 2023-05-23 09:23:35

\*/

@Service

public class DeviceRoomServiceImpl extends ServiceImpl<DeviceRoomMapper, DeviceRoom>

implements DeviceRoomService {

@Resource

private DevicePlaceService devicePlaceService;

@Resource

private RedisUtil redisUtil;

@Resource

private DeviceShelvesInfoService shelvesInfoService;

@Override

public Page<DeviceRoomVO> findByPage(CommonSelectEntity dto, boolean isPage) {

Page<DeviceRoomVO> page = new Page<>(dto.getCurrent(), dto.getPageSize());

dto.dealTime();

if (isPage) {

baseMapper.findPlaceByPage(page, dto);

} else {

page.setRecords(baseMapper.findPlaceByPage(dto));

}

return page;

}

@Override

public DeviceRoomVO selectById(String id) {

if (StringUtils.isEmpty(id)) return null;

DeviceRoom deviceRoom = baseMapper.selectById(id);

DevicePlaceVO vo = devicePlaceService.selectById(deviceRoom.getPlaceId());

if (vo == null) {

throw new ServiceException("【" + deviceRoom.getPlaceName() + "】场所不存在");

}

DeviceRoomVO deviceRoomVO = BeanUtil.copyProperties(deviceRoom, DeviceRoomVO.class);

deviceRoomVO.setPlaceCode(vo.getPlaceCode());

return deviceRoomVO;

}

@Override

public List<DeviceRoomVO> selectByPlaceId(String placeId) {

return baseMapper.selectByPlaceId(TenantContextHolder.getContext().getTenantContextInfo().getTenantId(), placeId);

}

@Override

public int logicDeleteByIds(String[] ids, String tenantId) {

if (StringUtils.isEmpty(tenantId)) {

tenantId = TenantContextHolder.getContext().getTenantContextInfo().getTenantId();

}

// 判断房间是否有货架

LambdaQueryWrapper<DeviceShelvesInfo> in = new QueryWrapper<DeviceShelvesInfo>().lambda()

.eq(true, DeviceShelvesInfo::getTenantId, tenantId)

.eq(true, DeviceShelvesInfo::getDelFlag, 1)

.in(true, DeviceShelvesInfo::getRoomId, (Object) ids);

List<DeviceShelvesInfo> list = shelvesInfoService.list(in);

if (CollectionUtils.isNotEmpty(list)) {

throw new ServiceException("您选择删除的存放室有货架，不允许删除");

}

return baseMapper.logicDelete(ids, tenantId);

}

private void checkParameter(DeviceRoomDTO add) {

if (StringUtils.isEmpty(add.getPlaceId())) {

throw new ServiceException("场所id没有填写");

}

if (StringUtils.isEmpty(add.getRoomName())) {

throw new ServiceException("存放室名称有填写");

}

}

@Override

public boolean add(DeviceRoomDTO add) {

SysUser sysUser = TenantContextHolder.getContext().getTenantContextInfo().getSysUser();

String tenantId = sysUser.getTenantId();

DeviceRoom room = BeanUtil.copyProperties(add, DeviceRoom.class);

this.checkParameter(add);

// 检测存放室信息

DevicePlaceVO vo = devicePlaceService.selectById(add.getPlaceId());

if (vo == null) {

throw new ServiceException("【" + add.getPlaceId() + "】场所不存在");

}

// 设置场所编码

String roomCodeKey = "sys-roomcode";

String key = roomCodeKey + tenantId;

if (Boolean.FALSE.equals(redisUtil.hasKey(key))) {

Long maxCode = baseMapper.findMaxCode(tenantId);

if (maxCode == null) {

maxCode = 0L;

}

redisUtil.setCacheObject(key, maxCode, 20, TimeUnit.MINUTES);

}

Long increment = redisUtil.increment(key);

room.setRoomCode(vo.getPlaceCode() + (increment < 10 ? "0" + increment : increment));

room.setPlaceName(vo.getPlaceName());

room.setUpdateBy(sysUser.getId());

room.setUpdateName(sysUser.getEmpName());

room.setUpdateTime(new Date());

return baseMapper.insert(room) == 1;

}

@Override

public boolean update(DeviceRoomDTO update) {

if (update == null) return true;

DeviceRoom room = BeanUtil.copyProperties(update, DeviceRoom.class);

if (update.getId() == null) {

throw new ParameterException("请求的主键没有传入");

}

if (baseMapper.selectById(update.getId()) == null) {

throw new ParameterException("请求更新的设备分类数据不存在，不能更新");

}

return baseMapper.updateById(room) == 1;

}

}

/\*\*

\* @author Administrator

\* @description 针对表【device\_shelves\_info(设备存放货架信息表)】的数据库操作Service实现

\* @createDate 2023-05-16 11:26:31

\*/

@Service

@EnableAspectJAutoProxy(exposeProxy = true)

public class DeviceShelvesInfoServiceImpl extends ServiceImpl<DeviceShelvesInfoMapper, DeviceShelvesInfo> implements DeviceShelvesInfoService {

@Resource

private DevicePlaceService devicePlaceService;

@Resource

private DeviceRoomService deviceRoomService;

@Resource

private RedisUtil redisUtil;

@Override

public Page<DeviceShelvesInfoVO> findByPage(DeviceShelvesInfoDTO dto, boolean isPage) {

Page<DeviceShelvesInfoVO> page = new Page<>(dto.getCurrent(), dto.getPageSize());

// 时间处理

if (dto.getStartTime() != null) {

dto.setStartTimeString(dto.getStartTime().format(DateTimeFormatter.ofPattern(DateTimeConstant.DATE)) + DateTimeConstant.DAY\_START\_TIME);

}

if (dto.getEndTime() != null) {

dto.setEndTimeString(dto.getEndTime().format(DateTimeFormatter.ofPattern(DateTimeConstant.DATE)) + DateTimeConstant.DAY\_END\_TIME);

}

// 是否关闭分页查询的总数统计

if (isPage) {

baseMapper.findByPage(page, dto);

} else {

page.setRecords(baseMapper.findByPage(dto));

}

return page;

}

@Override

public DeviceShelvesInfoVO selectById(String id) {

if (StringUtils.isEmpty(id)) return null;

DeviceShelvesInfo deviceShelvesInfo = baseMapper.selectById(id);

return BeanUtil.copyProperties(deviceShelvesInfo, DeviceShelvesInfoVO.class);

}

@Override

public List<DeviceShelvesInfo> selectByRoomId(String roomId) {

if (StringUtils.isEmpty(roomId)) return Collections.emptyList();

LambdaQueryWrapper<DeviceShelvesInfo> params = new QueryWrapper<DeviceShelvesInfo>().lambda()

.eq(true, DeviceShelvesInfo::getDelFlag, 1)

.eq(true, DeviceShelvesInfo::getTenantId, TenantContextHolder.getContext().getTenantContextInfo().getTenantId())

.eq(true, DeviceShelvesInfo::getRoomId, roomId);

return baseMapper.selectList(params);

}

@Override

public int logicDeleteByIds(String[] ids, String tenantId) {

if (StringUtils.isEmpty(tenantId)) {

tenantId = TenantContextHolder.getContext().getTenantContextInfo().getTenantId();

}

return baseMapper.logicDelete(ids, tenantId);

}

private void checkShelvesInfo(DeviceShelvesAddInfoDTO entity) {

if (StringUtils.isEmpty(entity.getPlaceId())) {

throw new ServiceException("场所不能为空");

}

if (StringUtils.isEmpty(entity.getRoomId())) {

throw new ServiceException("存放室不能为空");

}

if (StringUtils.isEmpty(entity.getShelvesName())) {

throw new ServiceException("货架名称不能为空");

}

if (entity.getLayerStart() == null || entity.getLayerEnd() == null || entity.getPositionStart() == null || entity.getPositionEnd() == null) {

throw new ServiceException("货架的编号范围参数不能为空");

}

if (entity.getLayerStart() > entity.getLayerEnd() || entity.getPositionStart() > entity.getPositionEnd()) {

throw new ServiceException("层号或者序号开始位置不能大于结束位置");

}

}

@Override

public boolean add(DeviceShelvesAddInfoDTO add) {

String tenantId = TenantContextHolder.getContext().getTenantContextInfo().getTenantId();

// 检查参数是否正确

if (add == null) return true;

this.checkShelvesInfo(add);

DevicePlaceVO vo = devicePlaceService.selectById(add.getPlaceId());

if (vo == null) {

throw new ServiceException("【" + add.getPlaceId() + "】场所不存在，请联系管理员！");

}

DeviceRoomVO deviceRoomVO = deviceRoomService.selectById(add.getRoomId());

if (deviceRoomVO == null) {

throw new ServiceException("【" + add.getRoomId() + "】存放室不存在，请联系管理员！");

}

// 设置场所编码

String roomCodeKey = "sys-shelvescode";

String key = roomCodeKey + tenantId;

if (Boolean.FALSE.equals(redisUtil.hasKey(key))) {

Long maxCode = baseMapper.findMaxCode(tenantId);

if (maxCode == null) {

maxCode = 0L;

}

redisUtil.setCacheObject(key, maxCode, 20, TimeUnit.MINUTES);

}

Long code = redisUtil.increment(key);

String shelfCode = code < 10 ? "0" + code : String.valueOf(code);

// 查询所有存放编号

Set<String> storageCodes = baseMapper.selectStorageCodeByTenantId(add.getShelvesCode(), tenantId);

List<DeviceShelvesInfo> infos = new ArrayList<>();

for (int layer = add.getLayerStart(); layer <= add.getLayerEnd(); layer++) {

for (int p = add.getPositionStart(); p <= add.getPositionEnd(); p++) {

DeviceShelvesInfo shelvesInfo = new DeviceShelvesInfo();

shelvesInfo.setRoomId(add.getRoomId());

shelvesInfo.setRoomName(deviceRoomVO.getRoomName());

shelvesInfo.setRoomCode(deviceRoomVO.getRoomCode());

shelvesInfo.setPlaceId(add.getPlaceId());

shelvesInfo.setPlaceName(vo.getPlaceName());

shelvesInfo.setPlaceCode(vo.getPlaceCode());

shelvesInfo.setStorageCode(deviceRoomVO.getRoomCode() + "-" + shelfCode + "-" + layer + "-" + p);

// 如果存放编号位置已经存在则不处理

if (storageCodes.contains(shelvesInfo.getStorageCode())) {

continue;

}

shelvesInfo.setShelvesName(add.getShelvesName());

shelvesInfo.setShelvesCode(shelfCode);

shelvesInfo.setLayerCode(String.valueOf(layer));

shelvesInfo.setPositionCode(String.valueOf(p));

infos.add(shelvesInfo);

}

}

DeviceShelvesInfoService proxy = (DeviceShelvesInfoService) AopContext.currentProxy();

return proxy.saveBatch(infos);

}

@Override

public boolean update(DeviceShelvesInfoUpdateDTO update) {

if (update == null) return true;

DeviceShelvesInfo entity = BeanUtil.copyProperties(update, DeviceShelvesInfo.class);

if (entity.getId() == null) {

throw new ParameterException("请求的主键没有传入");

}

if (baseMapper.selectById(entity.getId()) == null) {

throw new ParameterException("请求更新的设备分类数据不存在，不能更新");

}

if (StringUtils.isBlank(update.getPositionDesc())) {

throw new ParameterException("位置描述不能为空");

}

return this.updateById(entity);

}

@Override

public String selectShelvesNameByCode(String shelvesCode) {

return baseMapper.selectShelvesNameByCode(TenantContextHolder.getContext().getTenantContextInfo().getTenantId(), shelvesCode);

}

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_verification\_record(设备检定记录表)】的数据库操作Service实现

\* @createDate 2023-05-10 14:09:52

\*/

@Service

public class DeviceVerificationRecordServiceImpl extends ServiceImpl<DeviceVerificationRecordMapper, DeviceVerificationRecord>

implements IDeviceVerificationRecordService {

@Resource

private SysDictMapper sysDictMapper;

@Resource

private FileAttachMapper fileAttachMapper;

/\*\*

\* 分页查询设备检定记录

\*

\* @param deviceVerificationRecordSelectDTO

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceVerificationRecordVO>

\* @author wanggang

\* @date 2023/5/10 15:27

\*/

@Override

public Page<DeviceVerificationRecordVO> selectDeviceVerificationRecordPage(DeviceVerificationRecordSelectDTO deviceVerificationRecordSelectDTO) {

if (null == deviceVerificationRecordSelectDTO) {

throw new ParameterException("查询参数缺失");

}

Page<DeviceVerificationRecordVO> page = new Page<>(deviceVerificationRecordSelectDTO.getCurrent(), deviceVerificationRecordSelectDTO.getPageSize());

Page<DeviceVerificationRecordVO> deviceVerificationRecordPage = baseMapper.selectDeviceVerificationRecordPage(page, deviceVerificationRecordSelectDTO);

List<DeviceVerificationRecordVO> records = deviceVerificationRecordPage.getRecords();

//查询数据字典

//证书类型 device\_certificate\_type

List<SysDictVO> deviceCertificateType = sysDictMapper.selectDict("device\_certificate\_type");

//检定方式

List<SysDictVO> deviceVerificationStyleList = sysDictMapper.selectDict("device\_verification\_style");

for (DeviceVerificationRecordVO record : records) {

String certificateClassification = record.getCertificateClassification();

Integer verificationResult = record.getVerificationResult();

String textByValue = DeviceResultEnum.getTextByValue(verificationResult);

record.setVerificationResultDisplay(textByValue);

String verificationStyle = record.getVerificationStyle();

List<SysDictVO> collect = deviceCertificateType.stream().filter(f -> null != f.getVal() && f.getVal().equals(certificateClassification)).collect(Collectors.toList());

if (CollUtil.isNotEmpty(collect)) {

record.setCertificateClassificationDisplay(collect.get(0).getLabel());

}

List<SysDictVO> collect3 = deviceVerificationStyleList.stream().filter(f -> null != f.getVal() && f.getVal().equals(verificationStyle)).collect(Collectors.toList());

if (CollUtil.isNotEmpty(collect3)) {

record.setVerificationStyleDisplay(collect3.get(0).getLabel());

}

//处理文件信息

String fileId = record.getFileId();

if (StringUtils.isNotEmpty(fileId)) {

List<String> fileIdList = Arrays.asList(fileId.split("、"));

List<String> collect1 = fileIdList.stream().distinct().collect(Collectors.toList());

//查询对应文件信息

List<FileAttachVO> fileAttachVOList = fileAttachMapper.selectFileByFileId(collect1);

record.setFileAttachVOList(fileAttachVOList);

String uuId = fileAttachVOList.stream().map(FileAttachVO::getUuid)

.filter(Objects::nonNull).collect(Collectors.joining("、"));

String fileName = fileAttachVOList.stream().map(FileAttachVO::getFilename)

.filter(Objects::nonNull).collect(Collectors.joining("、"));

List<Date> collect2 = fileAttachVOList.stream().map(FileAttachVO::getCreateTime)

.filter(Objects::nonNull).collect(Collectors.toList());

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

for (Date date : collect2) {

String dateString = DateUtil.dateToStr("yyyy-MM-dd HH:mm:ss", date);

dateList.add(dateString);

}

String uploadTime = String.join("、", dateList);

record.setFileIdList(collect1);

record.setFilePath(uuId);

record.setFileName(fileName);

record.setUploadTime(uploadTime);

}

}

return deviceVerificationRecordPage;

}

/\*\*

\* 根据设备检定记录ID查询修改回显数据

\*

\* @param id

\* @return com.cstc.jcgl.domain.device.vo.DeviceVerificationRecordVO

\* @author wanggang

\* @date 2023/5/18 14:21

\*/

@Override

public DeviceVerificationRecordVO selectDeviceVerificationRecordById(String id) {

if (StringUtils.isEmpty(id)) {

throw new ParameterException("查询参数缺失");

}

//根据设备信息ID查询设备信息表修改回显数据

DeviceVerificationRecordVO deviceVerificationRecordVO = baseMapper.selectDeviceVerificationRecordById(id);

if (null != deviceVerificationRecordVO) {

String fileId = deviceVerificationRecordVO.getFileId();

if (StringUtils.isNotEmpty(fileId)) {

List<String> fileIdList = Arrays.asList(fileId.split("、"));

List<String> collect = fileIdList.stream().distinct().collect(Collectors.toList());

//查询对应文件信息

List<FileAttachVO> fileAttachVOList = fileAttachMapper.selectFileByFileId(collect);

deviceVerificationRecordVO.setFileAttachVOList(fileAttachVOList);

}

}

return deviceVerificationRecordVO;

}

/\*\*

\* 新增设备检定记录

\*

\* @param deviceVerificationRecordDTO

\* @return void

\* @author wanggang

\* @date 2023/5/10 14:30

\*/

@Override

public Boolean addDeviceVerificationRecord(DeviceVerificationRecordDTO deviceVerificationRecordDTO) {

if (null == deviceVerificationRecordDTO) {

throw new ParameterException("新增参数缺失");

}

DeviceVerificationRecord deviceVerificationRecord = new DeviceVerificationRecord();

List<String> fileIdList = deviceVerificationRecordDTO.getFileIdList();

if (CollUtil.isNotEmpty(fileIdList)) {

String fileId = String.join("、", fileIdList);

deviceVerificationRecordDTO.setFileId(fileId);

}

JcglBeanUtil.copyProperties(deviceVerificationRecordDTO, deviceVerificationRecord, true);

return this.save(deviceVerificationRecord);

}

/\*\*

\* 修改设备检定记录

\*

\* @param deviceVerificationRecordDTO

\* @return void

\* @author wanggang

\* @date 2023/5/10 14:30

\*/

@Override

public Boolean updateDeviceVerificationRecord(DeviceVerificationRecordDTO deviceVerificationRecordDTO) {

if (null == deviceVerificationRecordDTO || StringUtils.isEmpty(deviceVerificationRecordDTO.getId())) {

throw new ParameterException("修改参数缺失");

}

DeviceVerificationRecord deviceVerificationRecord = new DeviceVerificationRecord();

List<String> fileIdList = deviceVerificationRecordDTO.getFileIdList();

if (CollUtil.isNotEmpty(fileIdList)) {

String fileId = String.join("、", fileIdList);

deviceVerificationRecordDTO.setFileId(fileId);

}

JcglBeanUtil.copyProperties(deviceVerificationRecordDTO, deviceVerificationRecord, true);

return this.updateById(deviceVerificationRecord);

}

/\*\*

\* 批量逻辑删除设备检定记录

\*

\* @param id

\* @return void

\* @author wanggang

\* @date 2023/5/10 14:30

\*/

@Override

public Boolean deleteDeviceVerificationRecord(String id) {

if (StringUtils.isEmpty(id)) {

throw new ParameterException("逻辑删除Id不能为空!");

}

return this.removeById(id);

}

}

/\*\*

\* @author wanggang

\* @description 针对表【device\_working\_record(设备运行记录表)】的数据库操作Service实现

\* @createDate 2023-05-09 11:37:02

\*/

@Service

public class DeviceWorkingRecordServiceImpl extends ServiceImpl<DeviceWorkingRecordMapper, DeviceWorkingRecord>

implements IDeviceWorkingRecordService {

@Resource

private SysDictMapper sysDictMapper;

/\*\*

\* 分页查询设备运行记录

\*

\* @param deviceWorkingRecordSelectDTO

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.device.vo.DeviceWorkingRecordVO>

\* @author wanggang

\* @date 2023/5/9 15:27

\*/

@Override

public Page<DeviceWorkingRecordVO> selectDeviceWorkingRecordPage(DeviceWorkingRecordSelectDTO deviceWorkingRecordSelectDTO) {

if (null == deviceWorkingRecordSelectDTO) {

throw new ParameterException("查询参数缺失");

}

Page<DeviceWorkingRecordVO> page = new Page<>(deviceWorkingRecordSelectDTO.getCurrent(), deviceWorkingRecordSelectDTO.getPageSize());

Page<DeviceWorkingRecordVO> deviceBorrowRecordVO = baseMapper.selectDeviceWorkingRecordPage(page, deviceWorkingRecordSelectDTO);

List<DeviceWorkingRecordVO> records = deviceBorrowRecordVO.getRecords();

//查询数据字典

//使用前/使用后状态 device\_situation

List<SysDictVO> deviceStatus = sysDictMapper.selectDict("device\_situation");

for (DeviceWorkingRecordVO record : records) {

String useFrontStatus = record.getUseFrontStatus();

String useEndStatus = record.getUseEndStatus();

List<SysDictVO> collect = deviceStatus.stream().filter(f -> null != f.getVal() && f.getVal().equals(useFrontStatus)).collect(Collectors.toList());

if (CollUtil.isNotEmpty(collect)) {

record.setUseFrontStatusDisplay(collect.get(0).getLabel());

}

List<SysDictVO> collect1 = deviceStatus.stream().filter(f -> null != f.getVal() && f.getVal().equals(useEndStatus)).collect(Collectors.toList());

if (CollUtil.isNotEmpty(collect1)) {

record.setUseEndStatusDisplay(collect1.get(0).getLabel());

}

}

return deviceBorrowRecordVO;

}

/\*\*

\* 根据设备运行记录ID查询修改回显数据

\*

\* @param id

\* @return com.cstc.jcgl.domain.device.vo.DeviceBorrowRecordVO

\* @author wanggang

\* @date 2023/5/24 17:21

\*/

@Override

public DeviceWorkingRecordVO selectDeviceWorkingRecordById(String id) {

if (StringUtils.isEmpty(id)) {

throw new ParameterException("查询参数缺失");

}

//根据设备运行记录ID查询修改回显数据

return baseMapper.selectDeviceWorkingRecordById(id);

}

/\*\*

\* 新增设备运行记录

\*

\* @param deviceWorkingRecordDTO

\* @return void

\* @author wanggang

\* @date 2023/5/9 11:30

\*/

@Override

public Boolean addDeviceWorkingRecord(DeviceWorkingRecordDTO deviceWorkingRecordDTO) {

if (null == deviceWorkingRecordDTO) {

throw new ParameterException("新增参数缺失");

}

DeviceWorkingRecord deviceWorkingRecord = new DeviceWorkingRecord();

JcglBeanUtil.copyProperties(deviceWorkingRecordDTO, deviceWorkingRecord, true);

return this.save(deviceWorkingRecord);

}

/\*\*

\* 修改设备运行记录

\*

\* @param deviceWorkingRecordDTO

\* @return void

\* @author wanggang

\* @date 2023/5/9 11:30

\*/

@Override

public Boolean updateDeviceWorkingRecord(DeviceWorkingRecordDTO deviceWorkingRecordDTO) {

if (null == deviceWorkingRecordDTO || StringUtils.isEmpty(deviceWorkingRecordDTO.getId())) {

throw new ParameterException("修改参数缺失");

}

DeviceWorkingRecord deviceWorkingRecord = new DeviceWorkingRecord();

JcglBeanUtil.copyProperties(deviceWorkingRecordDTO, deviceWorkingRecord, true);

return this.updateById(deviceWorkingRecord);

}

/\*\*

\* 逻辑删除设备运行记录

\*

\* @param id

\* @return void

\* @author wanggang

\* @date 2023/5/9 11:30

\*/

@Override

public Boolean deleteDeviceWorkingRecord(String id) {

if (StringUtils.isEmpty(id)) {

throw new ParameterException("逻辑删除Id不能为空!");

}

return this.removeById(id);

}

}

/\*\*

\* @author Administrator

\* @description 针对表【supplier\_contacts\_info("厂商联系人信息表)】的数据库操作Service实现

\* @createDate 2023-05-08 13:53:52

\*/

@Service

public class SupplierContactsInfoServiceImpl extends ServiceImpl<SupplierContactsInfoMapper, SupplierContactsInfo>

implements SupplierContactsInfoService {

@Override

public Page<SupplierContactsInfoVO> findByPage(SupplierContactsInfoSelectDTO dto, boolean isPage) {

Page<SupplierContactsInfoVO> page = new Page<>(dto.getCurrent(), dto.getPageSize());

// 是否关闭分页查询的总数统计

page.setSearchCount(isPage);

// 时间处理

if (dto.getStartTime() != null) {

dto.setStartTimeString(dto.getStartTime().format(DateTimeFormatter.ofPattern(DateTimeConstant.DATE))

+ DateTimeConstant.DAY\_START\_TIME);

}

if (dto.getEndTime() != null) {

dto.setEndTimeString(dto.getEndTime().format(DateTimeFormatter.ofPattern(DateTimeConstant.DATE))

+ DateTimeConstant.DAY\_END\_TIME);

}

if (isPage) {

baseMapper.findByPage(page, dto);

} else {

page.setRecords(baseMapper.findByPage(dto));

}

return page;

}

@Override

public SupplierContactsInfoVO selectById(String id) {

if (StringUtils.isEmpty(id)) return null;

SupplierContactsInfo supplierContactsInfo = baseMapper.selectById(id);

SupplierContactsInfoVO supplierContactsInfoVO = BeanUtil.copyProperties(supplierContactsInfo, SupplierContactsInfoVO.class);

supplierContactsInfoVO.setContactName(supplierContactsInfo.getName());

return supplierContactsInfoVO;

}

@Override

public List<SupplierContactsInfoDTO> selectListBySupplierId(String supplierId) {

LambdaQueryWrapper<SupplierContactsInfo> parameter = new QueryWrapper<SupplierContactsInfo>().lambda()

.eq(StringUtils.isNotBlank(supplierId), SupplierContactsInfo::getSupplierId, supplierId)

.eq(true, SupplierContactsInfo::getDelFlag, 1);

List<SupplierContactsInfo> supplierContactsInfos = baseMapper.selectList(parameter);

return supplierContactsInfos.stream().map(s -> BeanUtil.copyProperties(s, SupplierContactsInfoDTO.class)).toList();

}

@Override

@TenantIgnore

public int logicDeleteByIds(String[] ids, String[] supplierIds, String tenantId) {

if (StringUtils.isEmpty(tenantId)) {

tenantId = TenantContextHolder.getContext().getTenantContextInfo().getTenantId();

}

return baseMapper.logicDelete(ids, supplierIds, tenantId);

}

private void checkSupplierInfo(SupplierContactsInfo info) {

if (StringUtils.isEmpty(info.getName())) {

throw new ServiceException("联系人姓名不能为空");

}

if (StringUtils.isEmpty(info.getGender())) {

throw new ServiceException("联系人性别不能为空");

}

if (StringUtils.isEmpty(info.getPosition())) {

throw new ServiceException("联系人职位不能为空");

}

if (StringUtils.isEmpty(info.getTelephone())) {

throw new ServiceException("联系人电话不能为空");

}

if (StringUtils.isEmpty(info.getSupplierId())) {

throw new ServiceException("厂商id不能为空");

}

Set<String> names = baseMapper.selectContactsName(info.getSupplierId(), TenantContextHolder.getContext().getTenantContextInfo().getTenantId());

if (names.contains(info.getName())) {

throw new ServiceException("该联系人已经存在");

}

}

@Override

public boolean addSupplierContactsInfo(SupplierContactsInfoDTO add) {

if (add == null) return true;

SupplierContactsInfo supplierContactsInfo = add.transferEntity();

// 检查联系人是否存在

this.checkSupplierInfo(supplierContactsInfo);

return this.save(supplierContactsInfo);

}

@Override

public boolean updateSupplierContactsInfo(SupplierContactsInfoDTO update) {

if (update == null) return true;

SupplierContactsInfo supplierContactsInfo = update.transferEntity();

if (update.getId() == null) {

throw new ParameterException("请求的主键没有传入");

}

if (baseMapper.selectById(update.getId()) == null) {

throw new ParameterException("请求更新的设备分类数据不存在，不能更新");

}

return this.updateById(supplierContactsInfo);

}

@Override

@Transactional(rollbackFor = Exception.class)

public boolean batchInserData(List<SupplierContactsInfo> data) {

data.forEach(this::checkSupplierInfo);

return this.saveBatch(data);

}

}

/\*\*

\* @author Administrator

\* @description 针对表【supplier\_info(厂商信息表)】的数据库操作Service实现

\* @createDate 2023-05-08 13:46:24

\*/

@Service

public class SupplierInfoServiceImpl extends ServiceImpl<SupplierInfoMapper, SupplierInfo>

implements SupplierInfoService {

@Resource

private SupplierContactsInfoService contactInfoService;

@Resource

private ISysDictCategoryService sysDictCategoryService;

private void dealTime(SupplierInfoSelectDTO dto) {

if (dto.getStartTime() != null) {

dto.setStartTimeString(dto.getStartTime().format(DateTimeFormatter.ofPattern(DateTimeConstant.DATE))

+ DateTimeConstant.DAY\_START\_TIME);

}

if (dto.getEndTime() != null) {

dto.setEndTimeString(dto.getEndTime().format(DateTimeFormatter.ofPattern(DateTimeConstant.DATE))

+ DateTimeConstant.DAY\_END\_TIME);

}

}

@Override

@TenantIgnore

public Page<SupplierInfoVO> findByPage(SupplierInfoSelectDTO dto, boolean isPage) {

Page<SupplierInfoVO> page = new Page<>(dto.getCurrent(), dto.getPageSize());

// 时间处理

this.dealTime(dto);

if (isPage) {

baseMapper.findByPage(page, dto);

} else {

page.setRecords(baseMapper.findByPage(dto));

}

return page;

}

@Override

public Page<SupplierInfoVO> findSupplierByPage(SupplierInfoSelectDTO dto, boolean isPage) {

Page<SupplierInfoVO> page = new Page<>(dto.getCurrent(), dto.getPageSize());

// 时间处理

this.dealTime(dto);

if (isPage) {

baseMapper.findSupplierByPage(page, dto);

} else {

page.setRecords(baseMapper.findSupplierByPage(dto));

}

return page;

}

@Override

@Transactional(rollbackFor = Exception.class)

public int logicDelete(String[] ids) {

// 逻辑删除所有相关的联系人信息

String tenantId = TenantContextHolder.getContext().getTenantContextInfo().getSysUser().getTenantId();

contactInfoService.logicDeleteByIds(null, ids, tenantId);

return baseMapper.logicDelete(ids, tenantId);

}

private void checkSupplierInfo(SupplierInfo info) {

if (StringUtils.isEmpty(info.getName())) {

throw new ServiceException("厂商名称不能为空");

}

if (info.getIndustry() == null) {

throw new ServiceException("行业不能为空");

}

if (info.getType() == null) {

throw new ServiceException("厂商类型不能为空");

}

if (StringUtils.isEmpty(info.getContactTelephone())) {

throw new ServiceException("座机不能为空");

}

// 判断厂商名是否重复

Set<String> names = baseMapper.selectSupplierName(TenantContextHolder.getContext().getTenantContextInfo()

.getSysUser().getTenantId());

if (names.contains(info.getName())) {

throw new ServiceException("【" + info.getName() + "】厂商已经存在");

}

}

@Override

@Transactional(rollbackFor = Exception.class)

public boolean addSupplierInfo(SupplierInfoDTO add) {

if (add == null) return true;

SupplierInfo supplierInfo = add.transferEntity();

boolean result = this.save(supplierInfo);

// 保存相关的 联系人数据

if (CollectionUtils.isNotEmpty(add.getContactList())) {

List<SupplierContactsInfo> contactList = add.getContactList().stream().map(c -> {

SupplierContactsInfo supplierContactsInfo = c.transferEntity();

supplierContactsInfo.setSupplierId(supplierInfo.getId());

return supplierContactsInfo;

}).toList();

contactInfoService.batchInserData(contactList);

}

return result;

}

@Override

public boolean updateSupplierInfo(SupplierInfoUpdateDTO update) {

if (update == null) return true;

SupplierInfo supplierInfo = update.transferEntity();

if (update.getId() == null) {

throw new ParameterException("请求的主键没有传入");

}

if (baseMapper.selectById(update.getId()) == null) {

throw new ParameterException("请求更新的厂商数据不存在，不能更新");

}

return this.updateById(supplierInfo);

}

private void dealWithDictType(List<SupplierInfoVO> list, boolean order) {

// 获取对应的字典

Map<String, String> gender = sysDictCategoryService.selectDictBycategoryKey(null, "gender", order);

Map<String, String> supplierType = sysDictCategoryService.selectDictBycategoryKey(null, "supplier\_type", order);

Map<String, String> supplierIndustry = sysDictCategoryService.selectDictBycategoryKey(null, "supplier\_industry", order);

if (org.springframework.util.CollectionUtils.isEmpty(gender)) {

throw new ServiceException("性别字典没有数据，请联系管理员添加");

}

if (org.springframework.util.CollectionUtils.isEmpty(supplierType)) {

throw new ServiceException("厂商类型字典没有数据，请联系管理员添加");

}

if (org.springframework.util.CollectionUtils.isEmpty(supplierIndustry)) {

throw new ServiceException("厂商行业字典没有数据，请联系管理员添加");

}

list.forEach(s -> {

s.setGender(gender.get(String.valueOf(s.getGender())));

s.setIndustry(supplierIndustry.get(String.valueOf(s.getIndustry())));

s.setType(supplierType.get(String.valueOf(s.getType())));

});

}

// 由于厂商数据没有全局唯一标识字段，所以只能一行一行的插入数据

@Override

@Transactional(rollbackFor = Exception.class)

public void importData(List<SupplierInfoVO> data, Boolean isOverride, Boolean isFirstDelete) {

if (CollUtil.isEmpty(data)) {

return;

}

// 查询租户下面的所有数据

List<SupplierInfo> oldSupplierInfos = baseMapper.selectList(new QueryWrapper<SupplierInfo>().lambda()

.eq(true, SupplierInfo::getTenantId, TenantContextHolder.getContext().getTenantContextInfo()

.getSysUser().getTenantId())

.eq(true, SupplierInfo::getDelFlag, 1)

);

List<String> nameCollect = oldSupplierInfos.stream().map(SupplierInfo::getName).toList();

// 处理枚举类数据

this.dealWithDictType(data, false);

// 覆盖时 执行 下面的操作，一条条的插入，判断

if (Boolean.TRUE.equals(isOverride)) {

this.checkData(data, nameCollect, isOverride);

this.updateSupplierAndContactsData(isFirstDelete, isOverride, data, nameCollect, oldSupplierInfos);

} else {

// 1、提前判断是否删除历史数据

// 2、检查是 厂商名 否重复

// 3、全部插入数据

this.insertSupplierAndContactsData(isFirstDelete, isOverride, data, nameCollect);

}

}

// 检查 厂商名 在 租户范围内是否唯一

private void checkData(List<SupplierInfoVO> data, List<String> nameCollect, Boolean isOverride) {

// 获取 厂商的名称 集合

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

// 判断excel表中是否存在重复的租户号

for (SupplierInfoVO temp : data) {

// 不覆盖时检测是否重名

if (nameCollect.contains(temp.getSupplierName()) && Boolean.FALSE.equals(isOverride)) {

throw new ServiceException("【" + temp.getSupplierName() + "】厂商信息已经导入");

}

if (excelCollect.contains(temp.getSupplierName())) {

throw new ServiceException("【" + temp.getSupplierName() + "】厂商信息在表中重复");

} else {

excelCollect.add(temp.getSupplierName());

}

}

}

// 更新历史数据 和 插入新的数据

private void updateSupplierAndContactsData(Boolean isFirstDelete, Boolean isOverride, List<SupplierInfoVO> data,

List<String> nameCollect, List<SupplierInfo> oldSupplierInfos) {

if (CollectionUtils.isEmpty(nameCollect)) {

this.insertSupplierAndContactsData(isFirstDelete, isOverride, data, nameCollect);

} else {

data.forEach(d -> {

// 厂商名称存在则 设置id

SupplierInfo info = BeanUtil.copyProperties(d, SupplierInfo.class);

info.setName(d.getSupplierName());

if (nameCollect.contains(d.getSupplierName())) {

info.setId(oldSupplierInfos.get(nameCollect.indexOf(d.getSupplierName())).getId());

}

// 获取当前类的代理对象

this.saveOrUpdate(info);

// 联系人姓名是否存在

SupplierContactsInfo supplierContactsInfo = BeanUtil.copyProperties(d, SupplierContactsInfo.class);

supplierContactsInfo.setName(d.getContactName());

supplierContactsInfo.setSupplierId(info.getId());

List<SupplierContactsInfoDTO> contactList = contactInfoService.selectListBySupplierId(info.getId());

List<String> contactNameCollect = contactList.stream().map(SupplierContactsInfoDTO::getName).toList();

if (contactNameCollect.contains(d.getContactName())) {

supplierContactsInfo.setId(contactList.get(contactNameCollect.indexOf(d.getContactName())).getId());

}

contactInfoService.saveOrUpdate(supplierContactsInfo);

});

}

}

// 批量插入数据

private void insertSupplierAndContactsData(Boolean isFirstDelete, Boolean isOverride, List<SupplierInfoVO> data, List<String> nameCollect) {

if (Boolean.TRUE.equals(isFirstDelete)) this.logicDelete(null);

this.checkData(data, nameCollect, isOverride);

// 2、批量导入厂商数据

List<SupplierInfo> supplierInfos = data.stream().map(d -> {

SupplierInfo info = BeanUtil.copyProperties(d, SupplierInfo.class);

info.setName(d.getSupplierName());

this.checkSupplierInfo(info);

return info;

}).toList();

this.saveBatch(supplierInfos);

// 创建对应的联系人

List<SupplierContactsInfo> concurrencyInfos = new ArrayList<>(data.size());

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

SupplierInfo supplierInfo = supplierInfos.get(i);

SupplierInfoVO supplierInfoVO = data.get(i);

SupplierContactsInfo supplierContactsInfo = BeanUtil.copyProperties(supplierInfoVO, SupplierContactsInfo.class);

supplierContactsInfo.setName(supplierInfoVO.getContactName());

supplierContactsInfo.setSupplierId(supplierInfo.getId());

concurrencyInfos.add(supplierContactsInfo);

}

contactInfoService.saveBatch(concurrencyInfos);

}

@Override

public void exportData(SupplierInfoSelectDTO dto, HttpServletResponse response) {

Page<SupplierInfoVO> page = this.findByPage(dto, false);

List<SupplierInfoVO> records = page.getRecords();

this.dealWithDictType(records, true);

try {

EasyExcelUtil.writeExcel(response, SupplierInfoVO.class, records, "厂商和联系人信息" + System.currentTimeMillis() + ".xlsx");

} catch (IOException | IllegalAccessException | NoSuchFieldException e) {

throw new ServiceException(e);

}

}

}

/\*\*

\* 修改流程定义中的部分内容

\*

\* @author wanggang

\* @date 2023/4/26 11:12

\*/

public interface IFlowableDynamicBpmnService {

void updateUserTaskCandidateGroups(String id, List<String> candidateGroups);

}

/\*\*

\* 查看历史流程信息服务

\*

\* @author wanggang

\* @date 2023/5/4 11:29

\*/

public interface IFlowableHistoryService {

/\*\*

\* 查询所有历史任务信息

\*

\* @param

\* @return java.util.List<com.jcgl.flowable.workflow.domain.dto.HistoricTaskInstanceDTO>

\* @author wanggang

\* @date 2023/5/4 13:50

\*/

List<HistoricTaskInstanceDTO> selectHistoryTask();

/\*\*

\* 查询已完成历史任务信息

\*

\* @param

\* @return java.util.List<com.jcgl.flowable.workflow.domain.dto.HistoricTaskInstanceDTO>

\* @author wanggang

\* @date 2023/5/4 13:50

\*/

List<HistoricTaskInstanceDTO> selectHistoryTaskFinish();

}

/\*\*

\* 初始化用户信息

\*

\* @author wanggang

\* @date 2023/4/14 16:39

\*/

public interface IFlowableIdentityService {

/\*\*

\* 新增组

\*

\* @param groupDTO

\* @return void

\* @author wanggang

\* @date 2023/4/14 17:10

\*/

void addGroup(GroupDTO groupDTO);

/\*\*

\* 添加用户

\*

\* @param userDTO

\* @return void

\* @author wanggang

\* @date 2023/4/17 10:32

\*/

void addUser(UserDTO userDTO);

/\*\*

\* 根据组Id或者组Name或用户名查询组

\*

\* @param param

\* @return java.util.List<org.flowable.idm.api.Group>

\* @author wanggang

\* @date 2023/4/14 17:10

\*/

List<GroupDTO> selectGroup(String param);

/\*\*

\* 根据组Id或者组Name或用户名查询组

\*

\* @param param

\* @return java.util.List<org.flowable.idm.api.User>

\* @author wanggang

\* @date 2023/4/18 17:10

\*/

List<UserDTO> selectUser(String param);

/\*\*

\* 添加用户和组的关联

\*

\* @param groupAndUserDTOList

\* @return void

\* @author wanggang

\* @date 2023/4/17 10:32

\*/

void addUserAndGroupRelationBatch(List<GroupAndUserDTO> groupAndUserDTOList);

void addUserAndGroupRelation(GroupAndUserDTO groupAndUserDTO);

/\*\*

\* 修改组名信息

\*

\* @param groupDTO

\* @return void

\* @author wanggang

\* @date 2023/4/19 15:58

\*/

void updateGroup(GroupDTO groupDTO);

/\*\*

\* 删除组

\*

\* @param groupId

\* @return void

\* @author wanggang

\* @date 2023/4/19 15:58

\*/

void deleteGroup(String groupId);

/\*\*

\* 修改除密码外的其他信息

\*

\* @param userDTO

\* @return void

\* @author wanggang

\* @date 2023/4/19 16:41

\*/

void updateUser(UserDTO userDTO);

/\*\*

\* 删除用户

\*

\* @param userId

\* @return void

\* @author wanggang

\* @date 2023/4/19 16:46

\*/

void deleteUser(String userId);

/\*\*

\* 删除用户和组的关联关系

\*

\* @param groupAndUserDTO

\* @return void

\* @author wanggang

\* @date 2023/4/19 17:01

\*/

void deleteUserAndGroupRelation(GroupAndUserDTO groupAndUserDTO);

}

/\*\*

\* 流程定义和部署

\*

\* @author wanggang

\* @date 2023/4/25 10:18

\*/

public interface IFlowableRepositoryService {

/\*\*

\* 查看流程图或xml

\*

\* @param flowId 流程Id

\* @param resourceName 资源名

\* @return void

\* @author wanggang

\* @date 2023/4/25 10:23

\*/

void loadFlowXml(HttpServletResponse httpServletResponse, String flowId, String resourceName);

/\*\*

\* 查询所有流程定义信息

\*

\* @param name,key

\* @return java.util.List<com.jcgl.flowable.workflow.domain.dto.ProcessDefinitionDTO>

\* @author wanggang

\* @date 2023/4/25 17:25

\*/

List<ProcessDefinitionDTO> selectProcessDefinition(String name, String key);

/\*\*

\* 下载资源文件

\*

\* @param flowId, resourceName,request

\* @return void

\* @author wanggang

\* @date 2023/4/26 14:49

\*/

void downloadFlowXml(String flowId, String resourceName, HttpServletRequest request, HttpServletResponse response);

}

/\*\*

\* 流程任务服务

\*

\* @author wanggang

\* @date 2023/4/14 9:21

\*/

public interface IFlowableTaskService {

/\*\*

\* 创建启动流程

\*

\* @param

\* @return java.lang.String

\* @author wanggang

\* @date 2023/4/14 10:19

\*/

ProcessDTO createFlowableEngine();

ProcessDTO createFlowableEngineInit(List<InitDealUser> initDealUserList);

/\*\*

\* 获取流程图并显示当前路径

\*

\* @param httpServletResponse, processId

\* @return void

\* @author wanggang

\* @date 2023/4/18 11:09

\*/

void genProcessDiagram(HttpServletResponse httpServletResponse, String processId) throws Exception;

/\*\*

\* 根据taskId获取流程图并显示当前路径

\*

\* @param httpServletResponse, processId

\* @return void

\* @author wanggang

\* @date 2023/4/18 11:09

\*/

void genProcessDiagramByTaskId(HttpServletResponse httpServletResponse, String taskId) throws Exception;

/\*\*

\* 根据用户名查询自己的任务

\*

\* @param userId

\* @return org.flowable.task.api.Task

\* @author wanggang

\* @date 2023/4/19 10:27

\*/

List<TaskDTO> getFlowableMyTaskByUserId(String userId);

/\*\*

\* 根据组名查询任务

\*

\* @param param

\* @return org.flowable.task.api.Task

\* @author wanggang

\* @date 2023/4/14 10:27

\*/

List<TaskDTO> getFlowableTask(String param);

/\*\*

\* 领取任务

\*

\* @param userId

\* @return void

\* @author wanggang

\* @date 2023/4/19 10:59

\*/

void receiveTask(String userId);

/\*\*

\* 完成任务

\*

\* @param userId, approved

\* @return void

\* @author wanggang

\* @date 2023/4/19 11:19

\*/

void finishTask(String userId, Boolean approved, String remark);

IPage<FlowBaseInfo> findPageFlowBaseInfo(Integer current , Integer pageSize, SysUser sysUser, List<SysFlowRole> mainFlowRoles, List<SysUserOrg> slaveFlowRoles, String searchKey, String processKey, String processState, String startTime, String endTime);

/\*\*

\* 审核通过流程

\*

\* @param completeTaskDTO

\* @param mainFlowRoles

\* @param slaveOrgs

\* @param allFlowRoleCodes

\*/

void completeTask(CompleteTaskDTO completeTaskDTO, List<SysFlowRole> mainFlowRoles, List<SysUserOrg> slaveOrgs, Set<String> allFlowRoleCodes);

/\*\*

\* 判断当当前流程能否修改

\* @param processInstanceId

\* @return

\*/

boolean canUpdate(String processInstanceId, List<SysFlowRole> mainFlowRoles, List<SysUserOrg> slaveOrgs, Set<String> allFlowRoleCodes);

/\*\*

\* 是否能审核

\* @param processId

\* @param mainFlowRoles

\* @param slaveFlowRoles

\* @param allFlowRoleCodes

\* @return

\*/

boolean canApproval(String processId, List<SysFlowRole> mainFlowRoles, List<SysUserOrg> slaveFlowRoles, Set<String> allFlowRoleCodes);

void rejectTask(RejectTaskDTO rejectTaskDTO);

List<FlowHistVO> findFlowHistVOs(String processId);

}

/\*\*

\* @author yb

\* @description 针对表【flow\_base\_info(流程基础流水信息表)】的数据库操作Service

\* @createDate 2023-05-31 16:36:30

\*/

public interface IFlowBaseInfoService extends IService<FlowBaseInfo> {

/\*\*

\* 分页查询已经办理的流程

\*

\* @param tPage

\* @param sysUserId

\* @param processState

\* @param processKey

\* @param searchKey

\* @return

\*/

IPage<FlowBaseInfo> findCompletedProcessPage(Page<FlowBaseInfo> tPage, String sysUserId, String processState, String processKey, String searchKey,String startTime, String endTime);

/\*\*

\* 分页查询待办流程

\* @param flowBaseInfoPage

\* @param flowOrgRoleVOS

\* @param sysUserId

\* @param searchKey

\* @param processKey

\* @param processState

\* @param startTime

\* @param endTime

\* @return

\*/

IPage<FlowBaseInfo> findMyTaskPage(Page<FlowBaseInfo> flowBaseInfoPage, List<FlowOrgRoleVO> flowOrgRoleVOS, String sysUserId, String searchKey, String processKey, String processState, String startTime, String endTime);

/\*\*

\* 查询我的全部待办流程

\* @param flowOrgRoleVOS

\* @param sysUserId

\* @return

\*/

List<FlowBaseInfo> findMyTaskList(List<FlowOrgRoleVO> flowOrgRoleVOS, String sysUserId);

}

/\*\*

\* @author wanggang

\* @description 针对表【flow\_device\_purchase\_record(采购申请流水记录)】的数据库操作Service

\* @createDate 2023-06-12 15:55:34

\*/

public interface IFlowDevicePurchaseRecordService extends IService<FlowDevicePurchaseRecord> {

/\*\*

\* 新增采购申请流水记

\*

\* @param flowDevicePurchaseRecordDTO

\* @return String

\* @author wanggang

\* @date 2023/6/12 16:30

\*/

String addFlowDevicePurchaseRecord(FlowDevicePurchaseRecordDTO flowDevicePurchaseRecordDTO);

/\*\*

\* 修改采购申请流水记录

\*

\* @param flowDevicePurchaseRecordDTO

\* @return Boolean

\* @author wanggang

\* @date 2023/6/13 9:30

\*/

Boolean updateFlowDevicePurchaseRecord(FlowDevicePurchaseRecordDTO flowDevicePurchaseRecordDTO);

/\*\*

\* 暂存采购申请流水记录

\*

\* @param flowDeviceVerificationRecordStorageDTO

\* @return Boolean

\* @author wanggang

\* @date 2023/6/13 9:30

\*/

Boolean storageFlowDevicePurchaseRecord(FlowDevicePurchaseRecordStorageDTO flowDeviceVerificationRecordStorageDTO);

}

/\*\*

\* @author wanggang

\* @description 针对表【flow\_device\_purchase(采购申请明细记录)】的数据库操作Service

\* @createDate 2023-06-12 15:55:33

\*/

public interface IFlowDevicePurchaseService extends IService<FlowDevicePurchase> {

/\*\*

\* 批量新增采购申请明细记录

\*

\* @param flowDevicePurchaseDTOList

\* @return Boolean

\* @author wanggang

\* @date 2023/6/13 10:30

\*/

Boolean addFlowDevicePurchaseBatch(List<FlowDevicePurchaseDTO> flowDevicePurchaseDTOList);

/\*\*

\* 批量修改采购申请明细记录

\*

\* @param flowDevicePurchaseDTOList

\* @return Boolean

\* @author wanggang

\* @date 2023/6/2 16:30

\*/

Boolean updateFlowDevicePurchaseBatch(List<FlowDevicePurchaseDTO> flowDevicePurchaseDTOList);

}

/\*\*

\* @author wanggang

\* @description 针对表【flow\_device\_verification\_record(检定校准申请流水记录)】的数据库操作Service

\* @createDate 2023-06-05 17:00:34

\*/

public interface IFlowDeviceVerificationRecordService extends IService<FlowDeviceVerificationRecord> {

/\*\*

\* 新增检定校准申请流水记录

\*

\* @param flowDeviceVerificationRecordDTO 新增检定校准申请流水记录

\* @return String

\* @author wanggang

\* @date 2023/6/6 9:30

\*/

String addFlowDeviceVerificationRecord(FlowDeviceVerificationRecordDTO flowDeviceVerificationRecordDTO);

/\*\*

\* 修改检定校准申请流水记录

\*

\* @param flowDeviceVerificationRecordDTO 修改检定校准申请流水记录

\* @return Boolean

\* @author wanggang

\* @date 2023/6/6 9:30

\*/

Boolean updateFlowDeviceVerificationRecord(FlowDeviceVerificationRecordDTO flowDeviceVerificationRecordDTO);

}

/\*\*

\* @author wanggang

\* @description 针对表【flow\_device\_verification】的数据库操作Service

\* @createDate 2023-06-01 16:10:49

\*/

public interface IFlowDeviceVerificationService extends IService<FlowDeviceVerification> {

/\*\*

\* 分页查询检定校准申请流水记录

\*

\* @param flowDeviceVerificationRecordSelectDTO

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.flow.vo.FlowDeviceVerificationVO>

\* @author wanggang

\* @date 2023/6/7 15:27

\*/

Page<FlowDeviceVerificationVO> selectFlowDeviceVerificationRecordPage(FlowDeviceVerificationRecordSelectDTO flowDeviceVerificationRecordSelectDTO);

/\*\*

\* 根据流程记录ID查询所有的明细数据

\*

\* @param processRecordId

\* @return java.util.List<com.cstc.jcgl.domain.flow.vo.FlowDeviceVerificationVO>

\* @author wanggang

\* @date 2023/6/13 15:47

\*/

List<FlowDeviceVerificationVO> selectByProcessRecordId(String processRecordId);

/\*\*

\* 批量新增检定校准申请明细

\*

\* @param flowDeviceVerificationDTOList

\* @return Boolean

\* @author wanggang

\* @date 2023/5/22 16:30

\*/

Boolean addFlowDeviceVerificationBatch(List<FlowDeviceVerificationDTO> flowDeviceVerificationDTOList);

/\*\*

\* 批量修改检定校准申请明细

\*

\* @param flowDeviceVerificationDTOList

\* @return Boolean

\* @author wanggang

\* @date 2023/5/22 16:30

\*/

Boolean updateFlowDeviceVerificationBatch(List<FlowDeviceVerificationDTO> flowDeviceVerificationDTOList);

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/6/1 10:53

\*/

@Component

@Data

@AllArgsConstructor

public class IFlowFactory {

RuntimeService runtimeService;

TaskService taskService;

HistoryService historyService;

}

/\*\*

\* 修改流程定义中的部分内容

\*

\* @author wanggang

\* @date 2023/4/26 11:12

\*/

@Service

@TenantIgnore

public class FlowableDynamicBpmnServiceImpl implements IFlowableDynamicBpmnService {

@Autowired

DynamicBpmnService dynamicBpmnService;

@Override

public void updateUserTaskCandidateGroups(String id, List<String> candidateGroups) {

ObjectNode jsonNodes = generateUserTaskCandidateGroups(id, candidateGroups);

dynamicBpmnService.changeUserTaskCandidateGroups(id, candidateGroups, jsonNodes);

dynamicBpmnService.saveProcessDefinitionInfo(id, jsonNodes);

}

private ObjectNode generateUserTaskCandidateGroups(String id, List<String> candidateGroups) {

return dynamicBpmnService.changeUserTaskCandidateGroups(id, candidateGroups);

}

}

/\*\*

\* 查看历史流程信息服务

\*

\* @author wanggang

\* @date 2023/5/4 11:28

\*/

@Service

@TenantIgnore

public class FlowableHistoryServiceImpl implements IFlowableHistoryService {

@Autowired

HistoryService historyService;

/\*\*

\* 查询所有历史任务信息

\*

\* @param

\* @return java.util.List<com.jcgl.flowable.workflow.domain.dto.HistoricTaskInstanceDTO>

\* @author wanggang

\* @date 2023/5/4 13:50

\*/

@Override

public List<HistoricTaskInstanceDTO> selectHistoryTask() {

List<HistoricTaskInstance> list = historyService.createHistoricTaskInstanceQuery().list();

List<HistoricTaskInstanceDTO> historicTaskInstanceDTOList = new ArrayList<>();

for (HistoricTaskInstance historicTaskInstance : list) {

HistoricTaskInstanceDTO historicTaskInstanceDTO = new HistoricTaskInstanceDTO();

JcglBeanUtil.copyProperties(historicTaskInstance, historicTaskInstanceDTO, true);

historicTaskInstanceDTOList.add(historicTaskInstanceDTO);

}

return historicTaskInstanceDTOList;

}

/\*\*

\* 查询已完成历史任务信息

\*

\* @param

\* @return java.util.List<com.jcgl.flowable.workflow.domain.dto.HistoricTaskInstanceDTO>

\* @author wanggang

\* @date 2023/5/4 13:50

\*/

@Override

public List<HistoricTaskInstanceDTO> selectHistoryTaskFinish() {

List<HistoricTaskInstance> list = historyService.createHistoricTaskInstanceQuery().finished().list();

List<HistoricTaskInstanceDTO> historicTaskInstanceDTOList = new ArrayList<>();

for (HistoricTaskInstance historicTaskInstance : list) {

HistoricTaskInstanceDTO historicTaskInstanceDTO = new HistoricTaskInstanceDTO();

JcglBeanUtil.copyProperties(historicTaskInstance, historicTaskInstanceDTO, true);

historicTaskInstanceDTOList.add(historicTaskInstanceDTO);

}

return historicTaskInstanceDTOList;

}

}

/\*\*

\* 初始化用户信息

\*

\* @author wanggang

\* @date 2023/4/14 16:39

\*/

@Service

@TenantIgnore

public class FlowableIdentityServiceImpl implements IFlowableIdentityService {

@Autowired

IdentityService identityService;

/\*\*

\* 新增组

\*

\* @param groupDTO

\* @return void

\* @author wanggang

\* @date 2023/4/14 17:10

\*/

@Override

public void addGroup(GroupDTO groupDTO) {

GroupEntityImpl groupEntity = new GroupEntityImpl();

JcglBeanUtil.copyProperties(groupDTO, groupEntity, true);

identityService.saveGroup(groupEntity);

}

/\*\*

\* 根据组Id或者组Name或用户名查询组

\*

\* @param param

\* @return com.jcgl.flowable.workflow.util.JcglReturn

\* @author wanggang

\* @date 2023/4/14 17:10

\*/

@Override

public List<GroupDTO> selectGroup(String param) {

if (StringUtils.isEmpty(param)) {

List<Group> list = identityService.createGroupQuery().list();

return list.stream().map(info -> {

GroupDTO groupDTO = new GroupDTO();

JcglBeanUtil.copyProperties(info, groupDTO, true);

return groupDTO;

}).collect(Collectors.toList());

}

//根据 组Id 查询组信息

List<Group> groupById = identityService.createGroupQuery().groupId(param).list();

if (groupById.isEmpty()) {

//根据 组Name 查询组信息

List<Group> groupByName = identityService.createGroupQuery().groupNameLike("%" + param + "%").list();

if (groupByName.isEmpty()) {

//根据用户名查询组信息（组里包含该用户）

List<Group> groupByUserName = identityService.createGroupQuery().groupMember(param).list();

if (groupByUserName.isEmpty()) {

throw new ServiceException("没有查询到对应组!");

} else {

return groupByUserName.stream().map(info -> {

GroupDTO groupDTO = new GroupDTO();

JcglBeanUtil.copyProperties(info, groupDTO, true);

return groupDTO;

}).collect(Collectors.toList());

}

} else {

return groupByName.stream().map(info -> {

GroupDTO groupDTO = new GroupDTO();

JcglBeanUtil.copyProperties(info, groupDTO, true);

return groupDTO;

}).collect(Collectors.toList());

}

}

return groupById.stream().map(info -> {

GroupDTO groupDTO = new GroupDTO();

JcglBeanUtil.copyProperties(info, groupDTO, true);

return groupDTO;

}).collect(Collectors.toList());

}

/\*\*

\* 修改组名信息

\*

\* @param groupDTO

\* @return void

\* @author wanggang

\* @date 2023/4/19 15:58

\*/

@Override

public void updateGroup(GroupDTO groupDTO) {

String groupId = groupDTO.getId();

String name = groupDTO.getName();

Group group = identityService.createGroupQuery().groupId(groupId).singleResult();

group.setName(name);

identityService.saveGroup(group);

}

/\*\*

\* 删除组

\*

\* @param groupId

\* @return void

\* @author wanggang

\* @date 2023/4/19 15:58

\*/

@Override

public void deleteGroup(String groupId) {

identityService.deleteGroup(groupId);

}

/\*\*

\* 添加用户

\*

\* @param userDTO

\* @return void

\* @author wanggang

\* @date 2023/4/17 10:32

\*/

@Override

public void addUser(UserDTO userDTO) {

UserEntityImpl userEntityImpl = new UserEntityImpl();

JcglBeanUtil.copyProperties(userDTO, userEntityImpl, true);

identityService.saveUser(userEntityImpl);

}

/\*\*

\* 修改除密码外的其他信息

\*

\* @param userDTO

\* @return void

\* @author wanggang

\* @date 2023/4/19 16:41

\*/

@Override

public void updateUser(UserDTO userDTO) {

UserEntityImpl userEntityImpl = new UserEntityImpl();

JcglBeanUtil.copyProperties(userDTO, userEntityImpl, true);

identityService.saveUser(userEntityImpl);

}

/\*\*

\* 删除用户

\*

\* @param userId

\* @return void

\* @author wanggang

\* @date 2023/4/19 16:46

\*/

@Override

public void deleteUser(String userId) {

identityService.deleteUser(userId);

}

/\*\*

\* 根据组Id或者组Name或用户名查询用户

\*

\* @param param

\* @return java.util.List<org.flowable.idm.api.User>

\* @author wanggang

\* @date 2023/4/14 17:10

\*/

@Override

public List<UserDTO> selectUser(String param) {

if (StringUtils.isEmpty(param)) {

List<User> list = identityService.createUserQuery().list();

return list.stream().map(info -> {

UserDTO userDTO = new UserDTO();

JcglBeanUtil.copyProperties(info, userDTO, true);

return userDTO;

}).collect(Collectors.toList());

}

//根据 用户Id 查询用户信息

List<User> userById = identityService.createUserQuery().userId(param).list();

if (userById.isEmpty()) {

//根据 显示名模糊查询 用户信息

List<User> userByName = identityService.createUserQuery().userDisplayNameLike("%" + param + "%").list();

if (userByName.isEmpty()) {

throw new ServiceException("没有查询到对应用户!");

} else {

return userByName.stream().map(info -> {

UserDTO userDTO = new UserDTO();

JcglBeanUtil.copyProperties(info, userDTO, true);

return userDTO;

}).collect(Collectors.toList());

}

}

return userById.stream().map(info -> {

UserDTO userDTO = new UserDTO();

JcglBeanUtil.copyProperties(info, userDTO, true);

return userDTO;

}).collect(Collectors.toList());

}

/\*\*

\* 添加用户和组的关联

\*

\* @param groupAndUserDTOList

\* @return void

\* @author wanggang

\* @date 2023/4/17 10:32

\*/

@Override

public void addUserAndGroupRelationBatch(List<GroupAndUserDTO> groupAndUserDTOList) {

for (GroupAndUserDTO groupAndUserDTO : groupAndUserDTOList) {

addUserAndGroupRelation(groupAndUserDTO);

}

}

@Override

public void addUserAndGroupRelation(GroupAndUserDTO groupAndUserDTO) {

String userId = groupAndUserDTO.getUserId();

String groupId = groupAndUserDTO.getGroupId();

if (StringUtils.isEmpty(userId)) {

throw new ParameterException("用户id不能为空!");

}

if (StringUtils.isEmpty(groupId)) {

throw new ParameterException("组id不能为空!");

}

User user = identityService.createUserQuery().userId(userId).singleResult();

if (null == user) {

throw new ServiceException("用户id为:" + userId + "的用户不存在!");

}

Group group = identityService.createGroupQuery().groupId(groupId).singleResult();

if (null == group) {

throw new ServiceException("组id为:" + groupId + "的组不存在!");

}

identityService.createMembership(userId, groupId);

}

/\*\*

\* 删除用户和组的关联关系

\*

\* @param groupAndUserDTO

\* @return void

\* @author wanggang

\* @date 2023/4/19 17:01

\*/

@Override

public void deleteUserAndGroupRelation(GroupAndUserDTO groupAndUserDTO) {

String userId = groupAndUserDTO.getUserId();

String groupId = groupAndUserDTO.getGroupId();

identityService.deleteMembership(userId, groupId);

}

}

/\*\*

\* 流程定义和部署

\*

\* @author wanggang

\* @date 2023/4/25 10:18

\*/

@Service

@TenantIgnore

public class FlowableRepositoryServiceImpl implements IFlowableRepositoryService {

@Autowired

RepositoryService repositoryService;

/\*\*

\* 查看流程图或xml

\*

\* @param flowId, resourceName

\* @return void

\* @author wanggang

\* @date 2023/4/25 10:23

\*/

@Override

public void loadFlowXml(HttpServletResponse httpServletResponse, String flowId, String resourceName) {

//根据ID和名称获取到流

InputStream resourceAsStream = repositoryService.getResourceAsStream(flowId, resourceName);

if (resourceName.contains(".png")) {

httpServletResponse.setContentType("image/png");

}

try {

byte[] buf = new byte[1024];

int length;

OutputStream out = httpServletResponse.getOutputStream();

while ((length = resourceAsStream.read(buf)) != -1) {

out.write(buf, 0, length);

}

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* 下载资源文件

\*

\* @param flowId, resourceName

\* @return void

\* @author wanggang

\* @date 2023/4/26 14:49

\*/

@Override

public void downloadFlowXml(String flowId, String resourceName, HttpServletRequest request, HttpServletResponse response) {

InputStream resourceAsStream = repositoryService.getResourceAsStream(flowId, resourceName);

//构造输出流

FileOutputStream fileOutputStream = null;

try {

File file = new File(resourceName);

fileOutputStream = new FileOutputStream(file);

IOUtils.copy(resourceAsStream, fileOutputStream);

} catch (IOException e) {

e.printStackTrace();

} finally {

// 关闭流

try {

if (null != fileOutputStream) {

fileOutputStream.close();

}

resourceAsStream.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

/\*\*

\* 查询所有流程定义信息

\*

\* @param name,key

\* @return java.util.List<com.jcgl.flowable.workflow.domain.dto.ProcessDefinitionDTO>

\* @author wanggang

\* @date 2023/4/25 17:25

\*/

@Override

public List<ProcessDefinitionDTO> selectProcessDefinition(String name, String key) {

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

//根据名称和key查询

if (StringUtils.isNotEmpty(name) && StringUtils.isNotEmpty(key)) {

list = repositoryService.createProcessDefinitionQuery().processDefinitionName(name)

.processDefinitionKey(key).list();

}

//根据名称查询

if (StringUtils.isNotEmpty(name) && StringUtils.isEmpty(key)) {

list = repositoryService.createProcessDefinitionQuery().processDefinitionName(name).list();

}

//根据key查询

if (StringUtils.isNotEmpty(key) && StringUtils.isEmpty(name)) {

list = repositoryService.createProcessDefinitionQuery().processDefinitionKey(key).list();

}

//全查询

if (StringUtils.isEmpty(name) && StringUtils.isEmpty(key)) {

list = repositoryService.createProcessDefinitionQuery().list();

}

List<ProcessDefinitionDTO> processDefinitionDTOList = new ArrayList<>();

for (ProcessDefinition processDefinition : list) {

ProcessDefinitionDTO processDefinitionDTO = new ProcessDefinitionDTO();

JcglBeanUtil.copyProperties(processDefinition, processDefinitionDTO, true);

processDefinitionDTOList.add(processDefinitionDTO);

}

return processDefinitionDTOList;

}

}

/\*\*

\* 流程任务服务

\*

\* @author wanggang

\* @date 2023/4/14 9:21

\*/

@Service

public class FlowableTaskServiceImpl implements IFlowableTaskService {

@Autowired

RuntimeService runtimeService;

@Autowired

RepositoryService repositoryService;

@Autowired

TaskService taskService;

@Autowired

ProcessEngine processEngine;

@Autowired

IdentityService identityService;

@Autowired

IFlowBaseInfoService flowBaseInfoService;

@Autowired

IFlowFactory flowFactory;

@Autowired

ISysOrgService sysOrgService;

@Autowired

ISysFlowRoleService sysFlowRoleService;

@Autowired

ISysUserService sysUserService;

/\*\*

\* 创建启动流程

\*

\* @param

\* @return java.lang.String

\* @author wanggang

\* @date 2023/4/14 10:19

\*/

@Override

public ProcessDTO createFlowableEngine() {

//startProcessInstanceByKey的参数是我们绘制流程图时自己所填写的key

// 启动流程实例并获得流程Id

ProcessInstance processInstance = runtimeService.startProcessInstanceByKey("flowableDemo");

if (null == processInstance) {

throw new ServiceException("该流程不存在!");

}

ProcessDTO processDTO = new ProcessDTO();

String id = processInstance.getId();

processDTO.setProcessId(id);

return processDTO;

}

/\*\*

\* 创建启动流程并初始化处理组

\*

\* @param

\* @return java.lang.String

\* @author wanggang

\* @date 2023/4/14 10:19

\*/

@Override

public ProcessDTO createFlowableEngineInit(List<InitDealUser> initDealUserList) {

if (initDealUserList.size() < 3) {

throw new ParameterException("请填写3组候选用户组信息!");

}

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

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

InitDealUser initDealUser = initDealUserList.get(i);

//表达式中的index从流程的顺序开始 从1开始依次往后

String index = i + 1 + "";

String group = initDealUser.getGroup();

if (StringUtils.isNotEmpty(group)) {

map.put("group" + index, group);

}

}

//startProcessInstanceByKey的参数是我们绘制流程图时自己所填写的key

// 启动流程实例并获得流程Id

ProcessInstance processInstance = runtimeService.startProcessInstanceByKey("flowableDemo", map);

if (null == processInstance) {

throw new ServiceException("该流程不存在!");

}

ProcessDTO processDTO = new ProcessDTO();

String id = processInstance.getId();

processDTO.setProcessId(id);

return processDTO;

}

/\*\*

\* 获取流程图并显示当前路径

\*

\* @param httpServletResponse, processId

\* @return void

\* @author wanggang

\* @date 2023/4/18 11:09

\*/

@Override

public void genProcessDiagram(HttpServletResponse httpServletResponse, String processId) throws Exception {

// if (StringUtils.isEmpty(processId)) {

// throw new ParameterException("流程Id为空!");

// }

// ProcessInstance pi = runtimeService.createProcessInstanceQuery().processInstanceId(processId).singleResult();

//

// //流程走完的不显示图

// if (pi == null) {

// throw new ServiceException("该流程已结束或该流程不存在!");

// }

// Task task = taskService.createTaskQuery().processInstanceId(pi.getId()).singleResult();

// //使用流程实例ID，查询正在执行的执行对象表，返回流程实例对象

// String instanceId = task.getProcessInstanceId();

// List<Execution> executions = runtimeService

// .createExecutionQuery()

// .processInstanceId(instanceId)

// .list();

//

// //得到正在执行的Activity的Id

// List<String> activityIds = new ArrayList<>();

// List<String> flows = new ArrayList<>();

// for (Execution exe : executions) {

// List<String> ids = runtimeService.getActiveActivityIds(exe.getId());

// activityIds.addAll(ids);

// }

//

// //获取流程图

// BpmnModel bpmnModel = repositoryService.getBpmnModel(pi.getProcessDefinitionId());

// ProcessEngineConfiguration engineConfiguration = processEngine.getProcessEngineConfiguration();

// ProcessDiagramGenerator diagramGenerator = engineConfiguration.getProcessDiagramGenerator();

byte[] buf = new byte[1024];

int length;

httpServletResponse.setContentType("image/png");

try (InputStream in = genProcessDiagram(processId, null);

OutputStream out = httpServletResponse.getOutputStream()) {

while ((length = in.read(buf)) != -1) {

out.write(buf, 0, length);

}

}

}

/\*\*

\* 根据taskId获取流程图并显示当前路径

\*

\* @param httpServletResponse, processId

\* @return void

\* @author wanggang

\* @date 2023/4/18 11:09

\*/

@Override

public void genProcessDiagramByTaskId(HttpServletResponse httpServletResponse, String taskId) throws Exception {

if (StringUtils.isEmpty(taskId)) {

throw new ParameterException("任务Id为空!");

}

// 根据任务 ID 获取流程实例 ID

Task task = taskService.createTaskQuery().taskId(taskId).singleResult();

if (null == task) {

throw new ServiceException("该任务不存在!");

}

String processInstanceId = task.getProcessInstanceId();

ProcessInstance pi = runtimeService.createProcessInstanceQuery().processInstanceId(processInstanceId).singleResult();

//流程走完的不显示图

if (pi == null) {

throw new ServiceException("该流程已结束或该流程不存在!");

}

//使用流程实例ID，查询正在执行的执行对象表，返回流程实例对象

String instanceId = task.getProcessInstanceId();

List<Execution> executions = runtimeService

.createExecutionQuery()

.processInstanceId(instanceId)

.list();

//得到正在执行的Activity的Id

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

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

for (Execution exe : executions) {

List<String> ids = runtimeService.getActiveActivityIds(exe.getId());

activityIds.addAll(ids);

}

//获取流程图

BpmnModel bpmnModel = repositoryService.getBpmnModel(pi.getProcessDefinitionId());

ProcessEngineConfiguration engineConfiguration = processEngine.getProcessEngineConfiguration();

ProcessDiagramGenerator diagramGenerator = engineConfiguration.getProcessDiagramGenerator();

byte[] buf = new byte[1024];

int length;

httpServletResponse.setContentType("image/png");

try (InputStream in = diagramGenerator.generateDiagram(bpmnModel, "png", activityIds, flows,

engineConfiguration.getActivityFontName(), engineConfiguration.getLabelFontName(),

engineConfiguration.getAnnotationFontName(), engineConfiguration.getClassLoader(), 1.0, true);

OutputStream out = httpServletResponse.getOutputStream()) {

while ((length = in.read(buf)) != -1) {

out.write(buf, 0, length);

}

}

}

/\*\*

\* 根据用户名查询自己的任务

\*

\* @param userId

\* @return org.flowable.task.api.Task

\* @author wanggang

\* @date 2023/4/19 10:27

\*/

@Override

public List<TaskDTO> getFlowableMyTaskByUserId(String userId) {

if (StringUtils.isEmpty(userId)) {

throw new ParameterException("用户名为空!");

}

// 获取任务

List<Task> taskList = taskService.createTaskQuery().taskAssigneeLike("%" + userId + "%").list();

List<TaskDTO> taskDTOList = new ArrayList<>();

if (taskList.isEmpty()) {

throw new ServiceException("用户名为:" + userId + "的用户没有任务可处理");

}

for (Task task : taskList) {

TaskDTO taskDTO = new TaskDTO();

String name = task.getName();

String id = task.getId();

String assignee = task.getAssignee();

taskDTO.setId(id);

taskDTO.setName(name);

taskDTO.setAssignee(assignee);

taskDTOList.add(taskDTO);

}

return taskDTOList;

}

/\*\*

\* 根据组名或用户名查询任务

\*

\* @param param

\* @return org.flowable.task.api.Task

\* @author wanggang

\* @date 2023/4/14 10:27

\*/

@Override

public List<TaskDTO> getFlowableTask(String param) {

List<Task> taskList;

if (StringUtils.isEmpty(param)) {

taskList = taskService.createTaskQuery().list();

} else {

// 查询任务

taskList = taskService.createTaskQuery().taskCandidateGroup(param).list();

if (taskList.isEmpty()) {

taskList = taskService.createTaskQuery().taskCandidateUser(param).list();

}

}

List<TaskDTO> taskDTOList = new ArrayList<>();

if (taskList.isEmpty()) {

throw new ServiceException("没有查询到任务");

}

for (Task task : taskList) {

TaskDTO taskDTO = new TaskDTO();

String name = task.getName();

String id = task.getId();

String assignee = task.getAssignee();

taskDTO.setId(id);

taskDTO.setName(name);

taskDTO.setAssignee(assignee);

taskDTOList.add(taskDTO);

}

return taskDTOList;

}

/\*\*

\* 领取任务或指定某个用户完成这个任务

\*

\* @param userId

\* @return void

\* @author wanggang

\* @date 2023/4/19 10:59

\*/

@Override

public void receiveTask(String userId) {

if (StringUtils.isEmpty(userId)) {

throw new ParameterException("用户Id为空!");

}

//验证该用户是否是候选组用户

//根据用户名查询组信息（组里包含该用户）

Group group = identityService.createGroupQuery().groupMember(userId).singleResult();

if (null == group) {

throw new ServiceException("该用户不属于任何组!");

}

//获取组id

String groupId = group.getId();

//查询该组的任务

List<Task> taskList = taskService.createTaskQuery().taskCandidateGroup(groupId).list();

if (taskList.isEmpty()) {

throw new ServiceException("该用户所在的组没有可处理的任务!");

}

for (Task task : taskList) {

String taskId = task.getId();

//领取任务

taskService.claim(taskId, userId);

}

//通过第三方工具 通知该用户处理这个任务——待实现

}

/\*\*

\* 完成任务

\*

\* @param userId, approved

\* @return void

\* @author wanggang

\* @date 2023/4/19 11:19

\*/

@Override

public void finishTask(String userId, Boolean approved, String remark) {

if (StringUtils.isEmpty(userId)) {

throw new ParameterException("用户名为空!");

}

// 验证用户是否有可处理的任务

List<Task> taskList = taskService.createTaskQuery().taskAssigneeLike("%" + userId + "%").list();

if (taskList.isEmpty()) {

throw new ServiceException("用户名为:" + userId + "的用户没有可处理的任务");

}

for (Task task : taskList) {

//获取流程Id

String processInstanceId = task.getProcessInstanceId();

//获取任务Id

String taskId = task.getId();

Task singleResult = taskService.createTaskQuery().processInstanceId(processInstanceId).singleResult();

//验证流程是否存在

if (null == singleResult) {

throw new ServiceException("流程不存在");

}

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

map.put("approved", approved);

//添加备注

taskService.addComment(taskId, processInstanceId, remark);

// 完成任务

taskService.complete(taskId, map);

}

}

@Override

public IPage<FlowBaseInfo> findPageFlowBaseInfo(Integer current, Integer pageSize, SysUser sysUser, List<SysFlowRole> mainFlowRoles, List<SysUserOrg> slaveFlowRoles, String searchKey, String processKey, String processState, String startTime, String endTime) {

//

List<FlowOrgRoleVO> flowOrgRoleVOS = getAllFlowOrgRoleVOs(sysUser, mainFlowRoles, slaveFlowRoles);

if (flowOrgRoleVOS == null || flowOrgRoleVOS.isEmpty()) {

return new Page<>();

}

return flowBaseInfoService.findMyTaskPage(new Page<>(current, pageSize), flowOrgRoleVOS, sysUser.getId(),searchKey,processKey,processState,startTime,endTime);

}

@Override

public void completeTask(CompleteTaskDTO completeTaskDTO, List<SysFlowRole> mainFlowRoles, List<SysUserOrg> slaveOrgs, Set<String> allFlowRoleCodes) {

FlowBaseInfo flowBaseInfo = flowBaseInfoService.getOne(new QueryWrapper<FlowBaseInfo>().lambda().eq(FlowBaseInfo::getProcessId, completeTaskDTO.getProcessId()));

if (flowBaseInfo == null) {

throw new ProcessException("不存在的流程");

}

//

SysUser sysUser = TenantContextHolder.getContext().getTenantContextInfo().getSysUser();

Task task = flowFactory.getTaskService().createTaskQuery()

.processInstanceId(completeTaskDTO.getProcessId())

.singleResult();

List<FlowOrgRoleVO> flowOrgRoleVOS = getAllFlowOrgRoleVOs(sysUser, mainFlowRoles, slaveOrgs);

List<FlowBaseInfo> flowBaseInfos = flowBaseInfoService.findMyTaskList(flowOrgRoleVOS, sysUser.getId());

if (flowBaseInfos == null || flowBaseInfos.isEmpty()) {

throw new ProcessException("您不是流程审核人");

}

if (!flowBaseInfos.stream().map(FlowBaseInfo::getProcessId).collect(Collectors.toSet()).contains(completeTaskDTO.getProcessId())) {

throw new ProcessException("您不是流程审核人");

}

// isMyProcessTask(mainFlowRoles, slaveOrgs, allFlowRoleCodes, flowBaseInfo, sysUser, task);

Authentication.setAuthenticatedUserId(sysUser.getId());

flowFactory.getTaskService().claim(task.getId(), sysUser.getId());

//添加审核意见

flowFactory.getTaskService().addComment(task.getId(), completeTaskDTO.getProcessId(), completeTaskDTO.getComment());

//

flowFactory.getTaskService().complete(task.getId());

}

@Override

public boolean canUpdate(String processInstanceId, List<SysFlowRole> mainFlowRoles, List<SysUserOrg> slaveOrgs, Set<String> allFlowRoleCodes) {

FlowBaseInfo flowBaseInfo = flowBaseInfoService.getOne(new QueryWrapper<FlowBaseInfo>().lambda().eq(FlowBaseInfo::getProcessId, processInstanceId));

SysUser sysUser = TenantContextHolder.getContext().getTenantContextInfo().getSysUser();

//判断是否我发起流程

if (!flowBaseInfo.getCreateBy().equals(sysUser.getId())) {

return false;

}

List<FlowOrgRoleVO> flowOrgRoleVOS = getAllFlowOrgRoleVOs(sysUser, mainFlowRoles, slaveOrgs);

List<FlowBaseInfo> flowBaseInfos = flowBaseInfoService.findMyTaskList(flowOrgRoleVOS, sysUser.getId());

if (flowBaseInfos == null || flowBaseInfos.isEmpty()) {

return false;

}

if (!flowBaseInfos.stream().map(FlowBaseInfo::getProcessId).collect(Collectors.toSet()).contains(processInstanceId)) {

return false;

}

return true;

}

@Override

public boolean canApproval(String processId, List<SysFlowRole> mainFlowRoles, List<SysUserOrg> slaveFlowRoles, Set<String> allFlowRoleCodes) {

FlowBaseInfo flowBaseInfo = flowBaseInfoService.getOne(new QueryWrapper<FlowBaseInfo>().lambda().eq(FlowBaseInfo::getProcessId, processId));

if (flowBaseInfo == null) {

throw new ProcessException("不存在的流程");

}

//

SysUser sysUser = TenantContextHolder.getContext().getTenantContextInfo().getSysUser();

List<FlowOrgRoleVO> flowOrgRoleVOS = getAllFlowOrgRoleVOs(sysUser, mainFlowRoles, slaveFlowRoles);

List<FlowBaseInfo> flowBaseInfos = flowBaseInfoService.findMyTaskList(flowOrgRoleVOS, sysUser.getId());

if (flowBaseInfos == null || flowBaseInfos.isEmpty()) {

return false;

}

if (!flowBaseInfos.stream().map(FlowBaseInfo::getProcessId).collect(Collectors.toSet()).contains(processId)) {

return false;

}

return true;

}

@Override

@Transactional

public void rejectTask(RejectTaskDTO rejectTaskDTO) {

SysUser sysUser = TenantContextHolder.getContext().getTenantContextInfo().getSysUser();

Authentication.setAuthenticatedUserId(sysUser.getId());

Task task = flowFactory.getTaskService().createTaskQuery().processInstanceId(rejectTaskDTO.getProcessId()).singleResult();

flowFactory.getTaskService().claim(task.getId(), sysUser.getId());

flowFactory.getTaskService().addComment(task.getId(), rejectTaskDTO.getProcessId(), rejectTaskDTO.getComment());

flowFactory.getTaskService().setVariableLocal(task.getId(), "reject", true);

String firstTargetId = flowFactory.getRuntimeService().createActivityInstanceQuery()

.activityType(BpmnXMLConstants.ELEMENT\_TASK\_USER).processInstanceId(rejectTaskDTO.getProcessId()).list().get(0).getActivityId();

flowFactory.getRuntimeService().createChangeActivityStateBuilder().processInstanceId(rejectTaskDTO.getProcessId())

.moveActivityIdTo(task.getTaskDefinitionKey(), firstTargetId).changeState();

FlowBaseInfo flowBaseInfo = flowBaseInfoService.getOne(new QueryWrapper<FlowBaseInfo>().lambda().eq(FlowBaseInfo::getProcessId, rejectTaskDTO.getProcessId()));

flowBaseInfo.setState(ProcessState.REJECT);

flowBaseInfoService.updateById(flowBaseInfo);

}

@Override

public List<FlowHistVO> findFlowHistVOs(String processId) {

List<FlowHistVO> flowHistVOS = new ArrayList<>();

HistoricProcessInstance historicProcessInstance = flowFactory.getHistoryService().createHistoricProcessInstanceQuery().processInstanceId(processId).singleResult();

List<HistoricActivityInstance> list = flowFactory.getHistoryService().createHistoricActivityInstanceQuery()

.processInstanceId(processId)

.activityType(BpmnXMLConstants.ELEMENT\_TASK\_USER)

.orderByHistoricActivityInstanceStartTime()

.asc()

.list();

FlowBaseInfo flowBaseInfo = flowBaseInfoService.getOne(new QueryWrapper<FlowBaseInfo>().lambda().eq(FlowBaseInfo::getProcessId, processId));

if (flowBaseInfo == null) {

return new ArrayList<>();

}

SysUserVO sysUserStart = sysUserService.findUserVOByUserId(historicProcessInstance.getStartUserId());

FlowHistVO flowHistStartVO = new FlowHistVO("开始流程", null, historicProcessInstance.getStartTime(), sysUserStart == null ? "" : sysUserStart.getEmpName(), null, null, null);

flowHistVOS.add(flowHistStartVO);

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

for (HistoricActivityInstance hai : list) {

Task task = flowFactory.getTaskService().createTaskQuery().taskId(hai.getTaskId()).includeIdentityLinks().active().singleResult();

FlowHistVO flowHistVO = new FlowHistVO();

if (task != null) {

//激活状态的任务，查询待办人,(待审核中的任务)

flowHistVO.setTaskState(FlowTaskState.NOT\_APPROVE);

if (!Strings.isNullOrEmpty(hai.getAssignee())){

SysUserVO sysUserAssignee = sysUserService.findUserVOByUserId(hai.getAssignee());

flowHistVO.setCandidateUsers(sysUserAssignee == null ? "" : sysUserAssignee.getEmpName());

}else {

if (task.getIdentityLinks() != null && !task.getIdentityLinks().isEmpty()) {

List<String> groupIds = task.getIdentityLinks().stream().map(IdentityLinkInfo::getGroupId).toList();

//获取候选用户角色

String flowRoleCode = groupIds.get(0);

//查询当前流程角色下的候选人

List<SysUser> sysUserCandidateList = sysUserService.findUserEmpNamesByFlowCode(flowRoleCode, flowBaseInfo.getSysOrgId());

if (sysUserCandidateList != null && !sysUserCandidateList.isEmpty()) {

flowHistVO.setCandidateUsers(Joiner.on(",").join(sysUserCandidateList.stream().map(SysUser::getEmpName).collect(Collectors.toList())));

}

}

}

} else {

HistoricVariableInstance variableInstanceReject = flowFactory.getHistoryService().createHistoricVariableInstanceQuery().taskId(hai.getTaskId())

.variableName("reject").singleResult();

if (variableInstanceReject != null && Boolean.TRUE.equals(variableInstanceReject.getValue())) {

flowHistVO.setTaskState(FlowTaskState.REJECT);

} else {

flowHistVO.setTaskState(FlowTaskState.APPROVED);

}

//已办理任务设置办理人

SysUserVO sysUserAssignee = sysUserService.findUserVOByUserId(hai.getAssignee());

flowHistVO.setAssigneeEmpName(sysUserAssignee == null ? "" : sysUserAssignee.getEmpName());

}

flowHistVO.setActivityName(hai.getActivityName());

flowHistVO.setStartTime(hai.getStartTime());

flowHistVO.setEndTime(hai.getEndTime());

List<Comment> comments = flowFactory.getTaskService().getTaskComments(hai.getTaskId());

String commentMsg = "";

if (comments != null && !comments.isEmpty()) {

commentMsg = Joiner.on(",").join(comments.stream().map(Comment::getFullMessage).collect(Collectors.toList()));

}

flowHistVO.setComment(commentMsg);

flowHistVOS.add(flowHistVO);

}

}

if (historicProcessInstance.getEndTime() != null) {

FlowHistVO flowHistEndVO = new FlowHistVO("结束流程", null, historicProcessInstance.getEndTime(), null, null, null, null);

flowHistVOS.add(flowHistEndVO);

}

return flowHistVOS;

}

/\*\*

\* 判断是否我审核的流程

\*

\* @param mainFlowRoles

\* @param slaveOrgs

\* @param allFlowRoleCodes

\* @param flowBaseInfo

\* @param sysUser

\* @param task

\*/

private void isMyProcessTask(List<SysFlowRole> mainFlowRoles, List<SysUserOrg> slaveOrgs, Set<String> allFlowRoleCodes, FlowBaseInfo flowBaseInfo, SysUser sysUser, Task task) {

if (task == null) {

throw new ProcessException("当前流程中不存在您的任务，无法审核");

}

List<IdentityLink> identityLinks = flowFactory.getTaskService().getIdentityLinksForTask(task.getId());

Set<String> groupIds = identityLinks.stream().map(IdentityLinkInfo::getGroupId).collect(Collectors.toSet());

if (groupIds.isEmpty()) {

throw new ProcessException("当前任务没有设置候选组");

}

AtomicReference<Boolean> isMyTask = new AtomicReference<>(false);

groupIds.forEach(groupId -> {

SysFlowRole flowRole = sysFlowRoleService.getOne(new QueryWrapper<SysFlowRole>().lambda().eq(SysFlowRole::getCode, groupId));

if ("1".equals(flowRole.getType())) {

List<String> upDownOrgIds = sysOrgService.selectUpDownOrgIds(flowBaseInfo.getSysOrgId());

//判断主组织

List<String> mainFlowRoleCodes = mainFlowRoles.stream().map(SysFlowRole::getCode).toList();

if (mainFlowRoleCodes.contains(groupId) && upDownOrgIds.contains(sysUser.getSysOrgId())) {

//有交集说明是自己的任务

isMyTask.set(true);

}

//判断兼职组织

slaveOrgs.forEach(sysUserOrg -> {

List<String> slaveUpDownOrgIds = sysOrgService.selectUpDownOrgIds(flowBaseInfo.getSysOrgId());

if (sysUserOrg.getSysFlowRoles() != null && !sysUserOrg.getSysFlowRoles().isEmpty()) {

List<String> salveFlowRoleCodes = sysUserOrg.getSysFlowRoles().stream().map(SysFlowRole::getCode).toList();

if (salveFlowRoleCodes.contains(groupId) && slaveUpDownOrgIds.contains(sysUserOrg.getSysOrgId())) {

//有交集说明是自己的任务

isMyTask.set(true);

}

}

});

} else if ("2".equals(flowRole.getType())) {

//如果是全局角色判断这个groupId是否包含在当前有的角色里面，有就是自己的任务

if (allFlowRoleCodes.contains(groupId)) {

isMyTask.set(true);

}

}

});

if (Boolean.FALSE.equals(isMyTask.get())) {

throw new ServiceException("您不是当前任务的审核候选人，请查看任务是否已被他人审核");

}

}

@NotNull

private Consumer<SysFlowRole> getFlowRoleConsumer(SysUser sysUser, String sysOrgId, Set<String> processIds) {

return sysFlowRole -> {

List<Task> mainOrgTasks = flowFactory.getTaskService().createTaskQuery()

.active()

.taskCandidateGroup(sysFlowRole.getCode())

.list();

//

Set<String> mainOrgTaskProcessIds = mainOrgTasks.stream().map(TaskInfo::getProcessInstanceId).collect(Collectors.toSet());

//流程类型1 上下级 2全局

if ("1".equals(sysFlowRole.getType())) {

List<String> upDownOrgIds = sysOrgService.selectUpDownOrgIds(sysOrgId);

Set<String> upDownProcessIds = flowBaseInfoService.list(new QueryWrapper<FlowBaseInfo>().lambda()

.in(!mainOrgTaskProcessIds.isEmpty(), FlowBaseInfo::getProcessId, mainOrgTaskProcessIds)

.in(upDownOrgIds != null && !upDownOrgIds.isEmpty(), FlowBaseInfo::getSysOrgId, upDownOrgIds)

.select(FlowBaseInfo::getProcessId)

).stream().map(FlowBaseInfo::getProcessId).collect(Collectors.toSet());

processIds.addAll(upDownProcessIds);

} else if ("2".equals(sysFlowRole.getType())) {

processIds.addAll(mainOrgTaskProcessIds);

}

};

}

List<FlowOrgRoleVO> getAllFlowOrgRoleVOs(SysUser sysUser, List<SysFlowRole> mainFlowRoles, List<SysUserOrg> slaveFlowRoles) {

List<FlowOrgRoleVO> flowOrgRoleVOS = new ArrayList<>();

//主组织

if (mainFlowRoles != null && !mainFlowRoles.isEmpty()) {

FlowOrgRoleVO flowOrgRoleVO = new FlowOrgRoleVO();

flowOrgRoleVO.setUpDownOrgIds(sysOrgService.selectUpDownOrgIds(sysUser.getSysOrgId()));

List<FlowOrgRoleVO.FlowRoleCode> flowRoleCodes = new ArrayList<>();

mainFlowRoles.forEach(mflowOrgRoleVO -> {

FlowOrgRoleVO.FlowRoleCode flowRoleCode = new FlowOrgRoleVO.FlowRoleCode(mflowOrgRoleVO.getCode(), mflowOrgRoleVO.getType());

flowRoleCodes.add(flowRoleCode);

});

flowOrgRoleVO.setRoleCodes(flowRoleCodes);

flowOrgRoleVOS.add(flowOrgRoleVO);

}

//兼职组织

if (slaveFlowRoles != null && !slaveFlowRoles.isEmpty()) {

slaveFlowRoles.forEach(sysUserOrg -> {

FlowOrgRoleVO flowOrgRoleVO = new FlowOrgRoleVO();

flowOrgRoleVO.setUpDownOrgIds(sysOrgService.selectUpDownOrgIds(sysUserOrg.getSysOrgId()));

List<FlowOrgRoleVO.FlowRoleCode> flowRoleCodes = new ArrayList<>();

if (sysUserOrg.getSysFlowRoles() != null && !sysUserOrg.getSysFlowRoles().isEmpty()) {

sysUserOrg.getSysFlowRoles().forEach(mflowOrgRoleVO -> {

FlowOrgRoleVO.FlowRoleCode flowRoleCode = new FlowOrgRoleVO.FlowRoleCode(mflowOrgRoleVO.getCode(), mflowOrgRoleVO.getType());

flowRoleCodes.add(flowRoleCode);

});

}

flowOrgRoleVO.setRoleCodes(flowRoleCodes);

flowOrgRoleVOS.add(flowOrgRoleVO);

});

}

return flowOrgRoleVOS;

}

public InputStream genProcessDiagram(String PROC\_INST\_ID\_, String type) {

/\* 获得活动的节点 \*/

String processDefinitionId = "";

ProcessInstance pi = runtimeService.createProcessInstanceQuery().processInstanceId(PROC\_INST\_ID\_).singleResult();

if (pi == null) { // 如果流程已经结束，则得到结束节点

HistoricProcessInstance pihis = flowFactory.getHistoryService().createHistoricProcessInstanceQuery().processInstanceId(PROC\_INST\_ID\_).singleResult();

processDefinitionId = pihis.getProcessDefinitionId();

} else { // 如果流程没有结束，则取当前活动节点

/\* 根据流程实例ID获得当前处于活动状态的ActivityId合集 \*/

processDefinitionId = pi.getProcessDefinitionId();

}

/\* 获得活动的节点对象 \*/

List<HistoricActivityInstance> highLightedActivitList = flowFactory.getHistoryService().createHistoricActivityInstanceQuery()

.finished()

.processInstanceId(PROC\_INST\_ID\_).orderByHistoricActivityInstanceStartTime().asc().list();

List<String> highLightedActivitis = new ArrayList<>(); //节点对象ID

for (HistoricActivityInstance tempActivity : highLightedActivitList) {

String activityId = tempActivity.getActivityId();

highLightedActivitis.add(activityId);

}

BpmnModel bpmnModel = repositoryService.getBpmnModel(processDefinitionId); //获取流程图

ProcessEngineConfiguration engconf = processEngine.getProcessEngineConfiguration();

/\* 获得活动的连线对象 \*/

List<String> flows = new ArrayList<>(); //连线ID

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

List<ActivityInstance> highLightedFlowInstances = runtimeService.createActivityInstanceQuery().activityType(BpmnXMLConstants.ELEMENT\_SEQUENCE\_FLOW).processInstanceId(PROC\_INST\_ID\_).list();

for (ActivityInstance ai : highLightedFlowInstances) {

flows.add(ai.getActivityId());

}

} else {

List<HistoricActivityInstance> highLightedFlowInstances = flowFactory.getHistoryService().createHistoricActivityInstanceQuery().activityType(BpmnXMLConstants.ELEMENT\_SEQUENCE\_FLOW).processInstanceId(PROC\_INST\_ID\_).list();

for (HistoricActivityInstance ha : highLightedFlowInstances) {

flows.add(ha.getActivityId());

}

}

ProcessDiagramGenerator diagramGenerator = engconf.getProcessDiagramGenerator();

InputStream in = diagramGenerator.generateDiagram(bpmnModel, "png", highLightedActivitis, flows, engconf.getActivityFontName(), engconf.getLabelFontName(), engconf.getAnnotationFontName(), engconf.getClassLoader(), 1.0, true);

return in;

}

}

/\*\*

\* @author wanggang

\* @description 针对表【flow\_device\_purchase\_record(采购申请流水记录)】的数据库操作Service实现

\* @createDate 2023-06-12 15:55:34

\*/

@Service

public class FlowDevicePurchaseRecordServiceImpl extends ServiceImpl<FlowDevicePurchaseRecordMapper, FlowDevicePurchaseRecord>

implements IFlowDevicePurchaseRecordService {

@Resource

IFlowDevicePurchaseService iFlowDevicePurchaseService;

/\*\*

\* 新增采购申请流水记

\*

\* @param flowDevicePurchaseRecordDTO

\* @return String

\* @author wanggang

\* @date 2023/6/12 16:30

\*/

@Override

public String addFlowDevicePurchaseRecord(FlowDevicePurchaseRecordDTO flowDevicePurchaseRecordDTO) {

if (null == flowDevicePurchaseRecordDTO) {

throw new ParameterException("新增参数缺失");

}

FlowDevicePurchaseRecord flowDevicePurchaseRecord = new FlowDevicePurchaseRecord();

JcglBeanUtil.copyProperties(flowDevicePurchaseRecordDTO, flowDevicePurchaseRecord, true);

this.save(flowDevicePurchaseRecord);

return flowDevicePurchaseRecord.getId();

}

/\*\*

\* 修改采购申请流水记录

\*

\* @param flowDevicePurchaseRecordDTO

\* @return Boolean

\* @author wanggang

\* @date 2023/6/13 9:30

\*/

@Override

public Boolean updateFlowDevicePurchaseRecord(FlowDevicePurchaseRecordDTO flowDevicePurchaseRecordDTO) {

if (null == flowDevicePurchaseRecordDTO || StringUtils.isEmpty(flowDevicePurchaseRecordDTO.getId())) {

throw new ParameterException("修改参数缺失");

}

FlowDevicePurchaseRecord flowDevicePurchaseRecord = new FlowDevicePurchaseRecord();

JcglBeanUtil.copyProperties(flowDevicePurchaseRecordDTO, flowDevicePurchaseRecord, true);

return this.updateById(flowDevicePurchaseRecord);

}

/\*\*

\* 暂存采购申请流水记录

\*

\* @param flowDeviceVerificationRecordStorageDTO

\* @return Boolean

\* @author wanggang

\* @date 2023/6/13 9:30

\*/

@Override

@Transactional(rollbackFor = Exception.class)

public Boolean storageFlowDevicePurchaseRecord(FlowDevicePurchaseRecordStorageDTO flowDeviceVerificationRecordStorageDTO) {

if (null == flowDeviceVerificationRecordStorageDTO) {

throw new ParameterException("新增参数缺失");

}

//设置未申请的状态

flowDeviceVerificationRecordStorageDTO.setApplyStatus(ApplyState.NOT\_APPLY);

List<FlowDevicePurchaseDTO> dtoList = flowDeviceVerificationRecordStorageDTO.getDtoList();

FlowDevicePurchaseRecordDTO flowDevicePurchaseRecordDTO = new FlowDevicePurchaseRecordDTO();

JcglBeanUtil.copyProperties(flowDeviceVerificationRecordStorageDTO, flowDevicePurchaseRecordDTO, true);

String id = this.addFlowDevicePurchaseRecord(flowDevicePurchaseRecordDTO);

dtoList.forEach(f -> f.setProcessRecordId(id));

return iFlowDevicePurchaseService.addFlowDevicePurchaseBatch(dtoList);

}

}

/\*\*

\* @author wanggang

\* @description 针对表【flow\_device\_purchase(采购申请明细记录)】的数据库操作Service实现

\* @createDate 2023-06-12 15:55:33

\*/

@Service

public class FlowDevicePurchaseServiceImpl extends ServiceImpl<FlowDevicePurchaseMapper, FlowDevicePurchase>

implements IFlowDevicePurchaseService {

/\*\*

\* 批量新增采购申请明细记录

\*

\* @param flowDevicePurchaseDTOList

\* @return Boolean

\* @author wanggang

\* @date 2023/6/13 10:30

\*/

@Override

public Boolean addFlowDevicePurchaseBatch(List<FlowDevicePurchaseDTO> flowDevicePurchaseDTOList) {

if (CollUtil.isEmpty(flowDevicePurchaseDTOList)) {

throw new ParameterException("新增参数缺失");

}

List<FlowDevicePurchase> flowDevicePurchaseList = flowDevicePurchaseDTOList.stream().map(info -> {

FlowDevicePurchase flowDevicePurchase = new FlowDevicePurchase();

JcglBeanUtil.copyProperties(info, flowDevicePurchase, true);

return flowDevicePurchase;

}).collect(Collectors.toList());

return this.saveBatch(flowDevicePurchaseList);

}

/\*\*

\* 批量修改采购申请明细记录

\*

\* @param flowDevicePurchaseDTOList

\* @return Boolean

\* @author wanggang

\* @date 2023/6/2 16:30

\*/

@Override

public Boolean updateFlowDevicePurchaseBatch(List<FlowDevicePurchaseDTO> flowDevicePurchaseDTOList) {

if (CollUtil.isEmpty(flowDevicePurchaseDTOList)) {

throw new ParameterException("新增参数缺失");

}

List<FlowDevicePurchase> flowDevicePurchaseList = flowDevicePurchaseDTOList.stream().map(info -> {

FlowDevicePurchase flowDevicePurchase = new FlowDevicePurchase();

JcglBeanUtil.copyProperties(info, flowDevicePurchase, true);

return flowDevicePurchase;

}).collect(Collectors.toList());

return this.updateBatchById(flowDevicePurchaseList);

}

}

/\*\*

\* @author wanggang

\* @description 针对表【flow\_device\_verification\_record(检定校准申请流水记录)】的数据库操作Service实现

\* @createDate 2023-06-05 17:00:34

\*/

@Service

public class FlowDeviceVerificationRecordServiceImpl extends ServiceImpl<FlowDeviceVerificationRecordMapper, FlowDeviceVerificationRecord>

implements IFlowDeviceVerificationRecordService {

/\*\*

\* 新增检定校准申请流水记录

\*

\* @param flowDeviceVerificationRecordDTO

\* @return String

\* @author wanggang

\* @date 2023/6/6 9:30

\*/

@Override

public String addFlowDeviceVerificationRecord(FlowDeviceVerificationRecordDTO flowDeviceVerificationRecordDTO) {

if (null == flowDeviceVerificationRecordDTO) {

throw new ParameterException("新增参数缺失");

}

FlowDeviceVerificationRecord flowDeviceVerificationRecord = new FlowDeviceVerificationRecord();

JcglBeanUtil.copyProperties(flowDeviceVerificationRecordDTO, flowDeviceVerificationRecord, true);

//生成流水号

LocalDateTime now = LocalDateTime.now();

String date = now.format(DateTimeFormatter.ofPattern("yyyyMMddHHmmss"));

String serialNumber = "JJ" + date;

flowDeviceVerificationRecord.setSerialNumber(serialNumber);

this.save(flowDeviceVerificationRecord);

return flowDeviceVerificationRecord.getId();

}

/\*\*

\* 修改检定校准申请流水记录

\*

\* @param flowDeviceVerificationRecordDTO

\* @return Boolean

\* @author wanggang

\* @date 2023/6/6 9:30

\*/

@Override

public Boolean updateFlowDeviceVerificationRecord(FlowDeviceVerificationRecordDTO flowDeviceVerificationRecordDTO) {

if (null == flowDeviceVerificationRecordDTO || StringUtils.isEmpty(flowDeviceVerificationRecordDTO.getId())) {

throw new ParameterException("修改参数缺失");

}

FlowDeviceVerificationRecord flowDeviceVerificationRecord = new FlowDeviceVerificationRecord();

JcglBeanUtil.copyProperties(flowDeviceVerificationRecordDTO, flowDeviceVerificationRecord, true);

return this.updateById(flowDeviceVerificationRecord);

}

}

/\*\*

\* @author wanggang

\* @description 针对表【flow\_device\_verification检定校准申请表】的数据库操作Service实现

\* @createDate 2023-06-01 16:10:49

\*/

@Service

public class FlowDeviceVerificationServiceImpl extends ServiceImpl<FlowDeviceVerificationMapper, FlowDeviceVerification>

implements IFlowDeviceVerificationService {

@Resource

SysDictMapper sysDictMapper;

@Resource

FlowDeviceVerificationMapper flowDeviceVerificationMapper;

/\*\*

\* 分页查询检定校准申请流水记录

\*

\* @param flowDeviceVerificationRecordSelectDTO

\* @return com.baomidou.mybatisplus.extension.plugins.pagination.Page<com.cstc.jcgl.domain.flow.vo.FlowDeviceVerificationVO>

\* @author wanggang

\* @date 2023/6/7 15:27

\*/

@Override

public Page<FlowDeviceVerificationVO> selectFlowDeviceVerificationRecordPage(FlowDeviceVerificationRecordSelectDTO flowDeviceVerificationRecordSelectDTO) {

if (null == flowDeviceVerificationRecordSelectDTO) {

throw new ParameterException("查询参数缺失");

}

Page<FlowDeviceVerificationVO> page = new Page<>(flowDeviceVerificationRecordSelectDTO.getCurrent(), flowDeviceVerificationRecordSelectDTO.getPageSize());

Page<FlowDeviceVerificationVO> flowDeviceVerificationPage = baseMapper.selectFlowDeviceVerificationPage(page, flowDeviceVerificationRecordSelectDTO);

List<FlowDeviceVerificationVO> records = flowDeviceVerificationPage.getRecords();

//查询数据字典

//检定方式 device\_verification\_style

List<SysDictVO> deviceVerificationStyleList = sysDictMapper.selectDict("device\_verification\_style");

for (FlowDeviceVerificationVO record : records) {

String verificationStyle = record.getVerificationStyle();

List<SysDictVO> collect = deviceVerificationStyleList.stream().filter(f -> null != f.getVal() && f.getVal().equals(verificationStyle)).collect(Collectors.toList());

if (CollUtil.isNotEmpty(collect)) {

record.setVerificationStyleDisplay(collect.get(0).getLabel());

}

}

return flowDeviceVerificationPage;

}

/\*\*

\* 根据流程记录ID查询所有的明细数据

\*

\* @param processRecordId

\* @return java.util.List<com.cstc.jcgl.domain.flow.vo.FlowDeviceVerificationVO>

\* @author wanggang

\* @date 2023/6/13 15:47

\*/

@Override

public List<FlowDeviceVerificationVO> selectByProcessRecordId(String processRecordId) {

if (StringUtils.isEmpty(processRecordId)) {

throw new ParameterException("查询参数缺失!");

}

return flowDeviceVerificationMapper.selectByProcessRecordId(processRecordId);

}

/\*\*

\* 批量新增检定校准申请记录

\*

\* @param flowDeviceVerificationDTOList

\* @return Boolean

\* @author wanggang

\* @date 2023/6/2 16:30

\*/

@Override

public Boolean addFlowDeviceVerificationBatch(List<FlowDeviceVerificationDTO> flowDeviceVerificationDTOList) {

if (CollUtil.isEmpty(flowDeviceVerificationDTOList)) {

throw new ParameterException("新增参数缺失");

}

List<FlowDeviceVerification> flowDeviceVerificationList = flowDeviceVerificationDTOList.stream().map(info -> {

FlowDeviceVerification flowDeviceVerification = new FlowDeviceVerification();

JcglBeanUtil.copyProperties(info, flowDeviceVerification, true);

return flowDeviceVerification;

}).collect(Collectors.toList());

return this.saveBatch(flowDeviceVerificationList);

}

/\*\*

\* 批量修改检定校准申请记录

\*

\* @param flowDeviceVerificationDTOList

\* @return Boolean

\* @author wanggang

\* @date 2023/6/2 16:30

\*/

@Override

public Boolean updateFlowDeviceVerificationBatch(List<FlowDeviceVerificationDTO> flowDeviceVerificationDTOList) {

if (CollUtil.isEmpty(flowDeviceVerificationDTOList)) {

throw new ParameterException("新增参数缺失");

}

List<FlowDeviceVerification> flowDeviceVerificationList = flowDeviceVerificationDTOList.stream().map(info -> {

FlowDeviceVerification flowDeviceVerification = new FlowDeviceVerification();

JcglBeanUtil.copyProperties(info, flowDeviceVerification, true);

return flowDeviceVerification;

}).collect(Collectors.toList());

return this.updateBatchById(flowDeviceVerificationList);

}

}

/\*\*

\* @author yb

\* @description 针对表【flow\_base\_info(流程基础流水信息表)】的数据库操作Service实现

\* @createDate 2023-05-31 16:36:30

\*/

@Service

public class FlowBaseInfoServiceImpl extends ServiceImpl<FlowBaseInfoMapper, FlowBaseInfo>

implements IFlowBaseInfoService {

@Override

public IPage<FlowBaseInfo> findCompletedProcessPage(Page<FlowBaseInfo> page, String sysUserId, String processState, String processKey, String searchKey,String startTime, String endTime) {

page.setRecords(baseMapper.findCompletedProcessPage(page,sysUserId,processState,processKey,searchKey,startTime,endTime));

return page;

}

@Override

public IPage<FlowBaseInfo> findMyTaskPage(Page<FlowBaseInfo> page, List<FlowOrgRoleVO> flowOrgRoleVOS, String sysUserId, String searchKey, String processKey, String processState, String startTime, String endTime) {

page.setRecords(baseMapper.findMyTaskPage(page,flowOrgRoleVOS,sysUserId,searchKey,processKey,processState,startTime,endTime));

return page;

}

@Override

public List<FlowBaseInfo> findMyTaskList(List<FlowOrgRoleVO> flowOrgRoleVOS, String sysUserId) {

return baseMapper.findMyTaskPage(null,flowOrgRoleVOS,sysUserId, null, null, null, null, null);

}

}

/\*\*

\* @author yb

\* @description 针对表【file\_list】的数据库操作Service

\* @createDate 2023-03-27 15:55:10

\*/

public interface IFileAttachService extends IService<FileAttach> {

FileAttach findAttachByUuid(String fileName);

}

/\*\*

\* @author yb

\* @description 针对表【sys\_dict\_category】的数据库操作Service

\* @createDate 2023-05-15 15:51:38

\*/

public interface ISysDictCategoryService extends IService<SysDictCategory> {

/\*\*

\* 获取字典的键值对，

\*

\* @param tenantId 租户id ，默认是当前租户id

\* @param categoryKey 字典类型，必传

\* @param order 键值对映射顺序，必传。等于true: 返回的是 val:lable （例如：1:男） 键值对。false则相反

\* @return 返回 {@link Map} key和value都是 {@link String}

\*/

Map<String, String> selectDictBycategoryKey(String tenantId, String categoryKey, boolean order);

/\*\*

\* <li>如果，当前用户是系统租户直接返回</li>

\* <li>已经存在的数据类目的字典的值恢复成初始化———<b>适用于当前组织的字典参数初始化</b></li>

\*/

void dictDataReset(String tenantId);

/\*\*

\* <p>初始化当前租户字典数据 或者<b>适用于系统升级</p>

\* <li>如果，当前用户是系统租户直接返回</li>

\* <li>已经存在的类目字典不处理，添加新的字典</li>

\*/

void initDictData(String tenantId);

}

/\*\*

\* @author yb

\* @description 针对表【sys\_dict】的数据库操作Service

\* @createDate 2023-02-15 09:17:11

\*/

public interface ISysDictService extends IService<SysDict> {

/\*\*

\* 获取字典的键值对

\*

\* @param type 字典类型

\* @param order 等于true: 返回的是 val:lable （例如：1:男） 键值对。false则相反

\* @return Map

\*/

Map<String, String> selectDictByType(String type, boolean order);

/\*\*

\* 查询数据字典

\* @param tenantId

\* @return

\*/

List<SysDict> findListAll(String tenantId);

}

/\*\*

\* @author yb

\* @description 针对表【sys\_flow\_role(系统流程角色)】的数据库操作Service

\* @createDate 2023-05-30 13:39:26

\*/

public interface ISysFlowRoleService extends IService<SysFlowRole> {

List<SysUser> findSysUserMainByCodeOrg(String flowRoleCode,List<String> upDownOrgIds);

}

/\*\*

\* @author yb

\* @description 针对表【sys\_log】的数据库操作Service

\* @createDate 2023-02-10 10:17:11

\*/

public interface ISysLogService extends IService<SysLog> {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_menu(菜单)】的数据库操作Service

\* @createDate 2023-02-10 10:17:11

\*/

public interface ISysMenuService extends IService<SysMenu> {

/\*\*

\* 树形结构菜单查询

\*

\* @return

\*/

List<SysMenu> findMenuListTree(String title, String menuSys);

/\*\*

\* 获取自己的菜单

\*

\* @param userId

\* @param menuSys

\* @return

\*/

List<SysMenu> findMyMenuListTree(String userId, String menuSys);

}

/\*\*

\* @author yb

\* @description 针对表【sys\_org】的数据库操作Service

\* @createDate 2023-02-22 16:47:13

\*/

public interface ISysOrgService extends IService<SysOrg> {

List<SysOrg> findTreeByOrgId(String sysOrgId);

Set<String> findAuthIdsByOrgId(String sysOrgId);

SupOrgListVO findSupOrgListVOByOrgId(String sysOrgId);

/\*\*

\* 查询指定部门下面的子部门

\*

\* @param parentId 当前部门id

\* @return {@link List}

\*/

List<SysOrg> findOrgByParentId(@Nullable String parentId);

List<String> selectUpDownOrgIds(String sysOrgId);

List<SysOrg> findTreeByOrg(SysOrg sysOrg);

}

/\*\*

\* @author yb

\* @description 针对表【sys\_post】的数据库操作Service

\* @createDate 2023-02-23 17:08:24

\*/

public interface ISysPostService extends IService<SysPost> {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_role\_menu】的数据库操作Service

\* @createDate 2023-02-10 10:17:11

\*/

public interface ISysRoleMenuService extends IService<SysRoleMenu> {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_role(角色)】的数据库操作Service

\* @createDate 2023-02-10 10:17:11

\*/

public interface ISysRoleService extends IService<SysRole> {

List<String> findUsernameByRoleId(String roleId);

}

/\*\*

\* @author yb

\* @description 针对表【sys\_tenant\_pkg\_menu(套餐菜单)】的数据库操作Service

\* @createDate 2023-05-09 17:17:23

\*/

public interface ISysTenantPkgMenuService extends IService<SysTenantPkgMenu> {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_tenant\_pkg(租户套餐表)】的数据库操作Service

\* @createDate 2023-05-09 17:17:23

\*/

public interface ISysTenantPkgService extends IService<SysTenantPkg> {

Page<SysTenantPkgVO> findPage(Page<SysTenantPkgVO> sysTenantPkgVOPage, String pkgName);

void updatePkg(SysTenantPkgDTO sysTenantPkgDTO);

void savePkg(SysTenantPkgDTO sysTenantPkgDTO);

void removePkg(String id);

}

/\*\*

\* @author yb

\* @description 针对表【sys\_tenant(租户信息表)】的数据库操作Service

\* @createDate 2023-05-09 17:17:23

\*/

public interface ISysTenantService extends IService<SysTenant> {

Page<SysTenantVO> findPage(Page<SysTenantVO> sysTenantVOPage, String tenantName);

/\*\*

\* 新增租户

\* @param sysTenantAddDTO

\*/

void addSysTenant(SysTenantAddDTO sysTenantAddDTO);

void updateTenant(SysTenant sysTenant);

void removeTenant(String tenantId);

}

/\*\*

\* @author yb

\* @description 针对表【sys\_user\_flow\_role(用户流程角色)】的数据库操作Service

\* @createDate 2023-05-30 13:39:43

\*/

public interface ISysUserFlowRoleService extends IService<SysUserFlowRole> {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_user\_org(用户组织)】的数据库操作Service

\* @createDate 2023-05-31 10:45:14

\*/

public interface ISysUserOrgService extends IService<SysUserOrg> {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_user\_role】的数据库操作Service

\* @createDate 2023-02-10 10:17:11

\*/

public interface ISysUserRoleService extends IService<SysUserRole> {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_user】的数据库操作Service

\* @createDate 2023-02-08 16:09:11

\*/

public interface ISysUserService extends IService<SysUser> {

/\*\*

\* @param username

\* @param password

\* @return

\*/

SysUser login(String username, String password);

Page<SysUserVO> findUserPage(Page<SysUserVO> objectPage, String sysOrgId, String empName, String mobile);

/\*\*

\* 设置账号启用停用

\*

\* @param sysUserId

\* @param enabled

\* @return

\*/

int updateEnabled(String sysUserId, int enabled);

int updateEmpStatus(String sysUserId, int status);

SysUserVO findUserVOByUserId(String id);

/\*\*

\* 查询用户权限列表

\*

\* @param username

\* @return

\*/

List<String> findMenuCodesByUsername(String username);

/\*\*

\* 查询用户角色列表

\*

\* @param username

\* @return

\*/

List<String> findRoleCodeByUsername(String username);

List<String> findAuthSysOrgIds(String sysOrgId);

/\*\*

\* 查询用户租户Id

\* @param userId

\* @return

\*/

String findTenantId(String userId);

SysUser findSysUserByUsername(String username);

long findCountByTenantId(String id);

/\*\*

\*

\* @param tenantId

\* @param enabled 1 启用-0停用

\* @return

\*/

long setStatusByTenantId(String tenantId, String enabled);

List<SysFlowRole> findFlowRoleCodes(String sysUserId);

/\*\*

\* 根据流程角色查询候选人

\*

\* @param flowRoleCode

\* @param sysOrgId

\* @return

\*/

List<SysUser> findUserEmpNamesByFlowCode(String flowRoleCode, String sysOrgId);

/\*\*

\* <p>根据手机号码查找用户信息</p>

\* <p>注意： 这里查询的是账号启用的用户信息，员工状态没有限定</p>

\*

\* @param phoneNumber 手机号码

\* @return {@link SysUser}

\*/

SysUser findSysUserByPhoneNumber(String phoneNumber);

List<SysRole> findRolesByUserId(String sysUserId);

}

/\*\*

\* @author yb

\* @description 针对表【file\_list】的数据库操作Service实现

\* @createDate 2023-03-27 15:55:10

\*/

@Service

public class FileAttachServiceImpl extends ServiceImpl<FileAttachMapper, FileAttach>

implements IFileAttachService {

@Override

public FileAttach findAttachByUuid(String uuid) {

return baseMapper.selectOne(new QueryWrapper<FileAttach>().lambda().eq(FileAttach::getUuid, uuid));

}

}

/\*\*

\* @author yb

\* @description 针对表【sys\_dict\_category】的数据库操作Service实现

\* @createDate 2023-05-15 15:51:38

\*/

@Service

public class SysDictCategoryServiceImpl extends ServiceImpl<SysDictCategoryMapper, SysDictCategory>

implements ISysDictCategoryService {

@Resource

private ISysDictService sysDictService;

@Override

public Map<String, String> selectDictBycategoryKey(String tenantId, String categoryKey, boolean order) {

if (StringUtils.isBlank(tenantId)) {

tenantId = TenantContextHolder.getContext().getTenantContextInfo().getTenantId();

}

if (StringUtils.isBlank(categoryKey)) {

throw new IllegalArgumentException("键值key必传");

}

// 查询租户下面的键值对

List<SysDictCategory> sysDictCategories = baseMapper.selectList(new QueryWrapper<SysDictCategory>().lambda()

.eq(true, SysDictCategory::getCategoryKey, categoryKey)

.eq(true, SysDictCategory::getTenantId, tenantId)

);

List<String> list = sysDictCategories.stream().map(SysDictCategory::getId).toList();

if (list.isEmpty()) {

return Collections.emptyMap();

}

LambdaQueryWrapper<SysDict> parameter = new QueryWrapper<SysDict>().lambda()

.in(true, SysDict::getSysDictCategoryId, list);

List<SysDict> sysDicts = sysDictService.list(parameter);

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

if (order) {

sysDicts.forEach(d -> map.put(d.getVal(), d.getLabel()));

} else {

sysDicts.forEach(d -> map.put(d.getLabel(), d.getVal()));

}

return map;

}

@Override

public void dictDataReset(String tenantId) {

// 当前用户就是系统租户，不用执行任何操作

if (SystemConstant.SYSTEM\_TENANT\_ID.equals(tenantId)) return;

// 系统

LambdaQueryWrapper<SysDictCategory> queryWrapper = new QueryWrapper<SysDictCategory>().lambda()

.eq(SysDictCategory::getTenantId, SystemConstant.SYSTEM\_TENANT\_ID);

// 查询系统的租户字典数据

List<SysDictCategory> sysDictCategories = baseMapper.selectList(queryWrapper);

// 查询当前用户的租户字典数据

LambdaQueryWrapper<SysDictCategory> otherWrapper = new QueryWrapper<SysDictCategory>().lambda()

.eq(SysDictCategory::getTenantId, tenantId);

List<SysDictCategory> currentDictCategories = baseMapper.selectList(otherWrapper);

// 如果不存在相同的数据则

if (!currentDictCategories.isEmpty()) {

List<String> categoryIds = sysDictCategories.stream().map(SysDictCategory::getId).toList();

// 查询类目下的字典

LambdaQueryWrapper<SysDict> parameter = new QueryWrapper<SysDict>().lambda()

.in(true, SysDict::getSysDictCategoryId, categoryIds);

List<SysDict> sysDicts = sysDictService.list(parameter);

List<String> currentCategoryIds = currentDictCategories.stream().map(SysDictCategory::getId).toList();

// 查询类目下的字典

LambdaQueryWrapper<SysDict> currentParameter = new QueryWrapper<SysDict>().lambda()

.in(true, SysDict::getSysDictCategoryId, currentCategoryIds);

List<SysDict> currentDicts = sysDictService.list(currentParameter);

if (CollUtil.isEmpty(sysDicts)) return;

Map<String, SysDict> systemMap = sysDicts.stream().collect(Collectors.toMap(s -> s.getType() + s.getVal(), s -> s));

List<SysDict> updateDictList = new ArrayList<>();

currentDicts.forEach(c -> {

if (systemMap.containsKey(c.getType() + c.getVal())) {

c.setLabel(systemMap.get(c.getType() + c.getVal()).getLabel());

updateDictList.add(c);

}

});

sysDictService.updateBatchById(updateDictList);

}

}

@Override

@Transactional(rollbackFor = Exception.class)

public void initDictData(String tenantId) {

// 当前用户就是系统租户，不用执行任何操作

if (SystemConstant.SYSTEM\_TENANT\_ID.equals(tenantId)) return;

// 系统

LambdaQueryWrapper<SysDictCategory> queryWrapper = new QueryWrapper<SysDictCategory>().lambda()

.eq(SysDictCategory::getTenantId, SystemConstant.SYSTEM\_TENANT\_ID);

// 查询系统的租户字典数据

List<SysDictCategory> sysDictCategories = baseMapper.selectList(queryWrapper);

// 查询当前用户的租户字典数据

LambdaQueryWrapper<SysDictCategory> otherWrapper = new QueryWrapper<SysDictCategory>().lambda()

.eq(SysDictCategory::getTenantId, tenantId);

List<SysDictCategory> currentDictCategories = baseMapper.selectList(otherWrapper);

// 过滤掉已经存在的类目

if (!currentDictCategories.isEmpty()) {

throw new ServiceException("当前组织已经初始化，不允许再次执行！");

// Set<String> temp = currentDictCategories.stream().map(SysDictCategory::getCategoryKey).collect(Collectors.toSet());

// sysDictCategories = sysDictCategories.stream().filter(s -> !temp.contains(s.getCategoryKey())).toList();

}

// 系统类目都已经添加则不用执行下面的操作

if(sysDictCategories.isEmpty()){

return;

}

// 把当前字典类别的数据和相关的 字典数据复制一遍

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

sysDictCategories.forEach(s -> {

categoryIds.add(s.getId());

s.setTenantId(tenantId);

s.setId(null);

});

// 批量插入字典类别数据

this.saveBatch(sysDictCategories);

Map<String, String> newsDictCategories = sysDictCategories.stream()

.collect(Collectors.toMap(SysDictCategory::getCategoryKey, SysDictCategory::getId));

LambdaQueryWrapper<SysDict> parameter = new QueryWrapper<SysDict>().lambda()

.in(true, SysDict::getSysDictCategoryId, categoryIds);

List<SysDict> sysDicts = sysDictService.list(parameter);

if (CollUtil.isNotEmpty(sysDicts)) {

sysDicts.forEach(sysDict -> {

sysDict.setId(null);

sysDict.setSysDictCategoryId(newsDictCategories.get(sysDict.getType()));

});

sysDictService.saveBatch(sysDicts);

}

}

}

/\*\*

\* @author yb

\* @description 针对表【sys\_dict】的数据库操作Service实现

\* @createDate 2023-02-15 09:17:11

\*/

@Service

public class SysDictServiceImpl extends ServiceImpl<SysDictMapper, SysDict>

implements ISysDictService {

@Override

public Map<String, String> selectDictByType(String type, boolean order) {

if (StringUtils.isEmpty(type)) {

return Collections.emptyMap();

}

LambdaQueryWrapper<SysDict> parameter = new QueryWrapper<SysDict>().lambda()

.like(!Strings.isNullOrEmpty(type), SysDict::getType, type);

List<SysDict> sysDicts = baseMapper.selectList(parameter);

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

if (order) {

sysDicts.forEach(d -> map.put(d.getVal(), d.getLabel()));

} else {

sysDicts.forEach(d -> map.put(d.getLabel(), d.getVal()));

}

return map;

}

@Override

public List<SysDict> findListAll(String tenantId) {

return baseMapper.findListAll(tenantId);

}

}

/\*\*

\* @author yb

\* @description 针对表【sys\_flow\_role(系统流程角色)】的数据库操作Service实现

\* @createDate 2023-05-30 13:39:26

\*/

@Service

public class SysFlowRoleServiceImpl extends ServiceImpl<SysFlowRoleMapper, SysFlowRole>

implements ISysFlowRoleService {

@Override

public List<SysUser> findSysUserMainByCodeOrg(String flowRoleCode, List<String> upDownOrgIds) {

return baseMapper.findSysUserMainByCodeOrg(flowRoleCode,upDownOrgIds);

}

}

/\*\*

\* @author yb

\* @description 针对表【sys\_log】的数据库操作Service实现

\* @createDate 2023-02-10 10:17:11

\*/

@Service

public class SysLogServiceImpl extends ServiceImpl<SysLogMapper, SysLog>

implements ISysLogService {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_menu(菜单)】的数据库操作Service实现

\* @createDate 2023-02-10 10:17:11

\*/

@Service

public class SysMenuServiceImpl extends ServiceImpl<SysMenuMapper, SysMenu>

implements ISysMenuService {

@Override

public List<SysMenu> findMenuListTree(String title, String menuSys) {

//查询顶级菜单

return findChildrenByMenuId("0", null, title,menuSys);

}

@Override

public List<SysMenu> findMyMenuListTree(String userId, String menuSys) {

//查询用户对应的菜单Id

List<String> menuIds = baseMapper.findAuthMenuIdsByUserId(userId);

if ("1628267730502344706".equals(userId)) {

menuIds = baseMapper.selectList(null).stream().map(SysMenu::getId).collect(Collectors.toList());

}

if (menuIds == null || menuIds.isEmpty()) {

return new ArrayList<>();

}

return findChildrenByMenuId("0", menuIds, null, menuSys);

}

private List<SysMenu> findChildrenByMenuId(String menuId, List<String> authMenuIds, String title, String menuSys) {

List<SysMenu> sysMenus = baseMapper.selectList(new QueryWrapper<SysMenu>().lambda()

.like(!Strings.isNullOrEmpty(title), SysMenu::getTitle, title)

.eq(SysMenu::getParentId, menuId)

.eq(SysMenu::getMenuSys,menuSys)

.in(authMenuIds != null && !authMenuIds.isEmpty(), SysMenu::getId, authMenuIds)

.orderByAsc(SysMenu::getSort)

);

if (sysMenus != null && !sysMenus.isEmpty()) {

sysMenus.forEach(sysMenu -> {

List<SysMenu> sysMenuChildren = findChildrenByMenuId(sysMenu.getId(), authMenuIds, title, menuSys);

if (sysMenuChildren != null && !sysMenuChildren.isEmpty()) {

sysMenu.setChildren(sysMenuChildren);

}

});

}

return sysMenus;

}

}

/\*\*

\* @author yb

\* @description 针对表【sys\_org】的数据库操作Service实现

\* @createDate 2023-02-22 16:47:13

\*/

@Service

public class SysOrgServiceImpl extends ServiceImpl<SysOrgMapper, SysOrg>

implements ISysOrgService {

@Override

public List<SysOrg> findTreeByOrgId(String sysOrgId) {

return findChildrenByOrgId(sysOrgId);

}

@Override

public Set<String> findAuthIdsByOrgId(String sysOrgId) {

return findCOrgIdByPId(sysOrgId, new HashSet<>());

}

@Override

public SupOrgListVO findSupOrgListVOByOrgId(String sysOrgId) {

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

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

findSupAncestors(sysOrgId, ancestors, pNames);

SupOrgListVO supOrgListVO = new SupOrgListVO();

Collections.reverse(ancestors);

Collections.reverse(pNames);

supOrgListVO.setAncestors(ancestors);

supOrgListVO.setPOrgNames(pNames);

return supOrgListVO;

}

@Override

public List<SysOrg> findOrgByParentId(@Nullable String parentId) {

if (StringUtils.isEmpty(parentId)) return Collections.emptyList();

String tenantId = TenantContextHolder.getContext().getTenantContextInfo().getTenantId();

LambdaQueryWrapper<SysOrg> queryWrapper = new QueryWrapper<SysOrg>().lambda()

.eq(true, SysOrg::getTenantId, tenantId)

.eq(true, SysOrg::getParentId, parentId);

return baseMapper.selectList(queryWrapper);

}

@Override

public List<String> selectUpDownOrgIds(String sysOrgId) {

List<String> upOrgIds = Splitter.on(",").splitToList(baseMapper.selectById(sysOrgId).getAncestors());

//查询下级

List<String> downOrgIds = baseMapper.selectList(new QueryWrapper<SysOrg>().lambda().select(SysOrg::getId).like(SysOrg::getAncestors, sysOrgId)).stream().map(SysOrg::getId).toList();

return CollUtil.unionDistinct(upOrgIds, downOrgIds).stream().toList();

}

@Override

public List<SysOrg> findTreeByOrg(SysOrg sysOrg) {

return findChildrenByOrg(sysOrg);

}

private void findSupAncestors(String sysOrgId, List<String> ancestors, List<String> pNames) {

SysOrg sysOrg = baseMapper.selectById(sysOrgId);

if (sysOrg != null) {

if (!"-1".equals(sysOrg.getParentId())) {

ancestors.add(sysOrg.getId());

pNames.add(sysOrg.getOrgName());

findSupAncestors(sysOrg.getParentId(), ancestors, pNames);

}

}

}

//根据orgId查询下级id

private Set<String> findCOrgIdByPId(String sysOrgId, Set<String> lastOrgIds) {

List<String> orgIds = baseMapper.selectOrgIdByPId(sysOrgId);

if (!orgIds.isEmpty()) {

lastOrgIds.addAll(orgIds);

orgIds.forEach(orgId -> {

Set<String> cOrgIds = findCOrgIdByPId(orgId, lastOrgIds);

if (!cOrgIds.isEmpty()) {

lastOrgIds.addAll(cOrgIds);

}

});

}

return lastOrgIds;

}

private List<SysOrg> findChildrenByOrgId(String sysOrgId) {

List<SysOrg> sysOrgs = baseMapper.selectList(new QueryWrapper<SysOrg>().lambda()

.eq(SysOrg::getParentId, sysOrgId)

.orderByAsc(SysOrg::getSort)

);

if (sysOrgs != null && !sysOrgs.isEmpty()) {

sysOrgs.forEach(sysOrg -> {

List<SysOrg> sysOrgChildren = findChildrenByOrgId(sysOrg.getId());

if (sysOrgChildren != null && !sysOrgChildren.isEmpty()) {

sysOrg.setChildren(sysOrgChildren);

}

});

}

return sysOrgs;

}

private List<SysOrg> findChildrenByOrg(SysOrg psysOrg) {

List<SysOrg> sysOrgs = baseMapper.selectList(new QueryWrapper<SysOrg>().lambda()

.eq(SysOrg::getParentId, psysOrg.getParentId())

.like(SysOrg::getAncestors, psysOrg.getId())

.orderByAsc(SysOrg::getSort)

);

if (sysOrgs != null && !sysOrgs.isEmpty()) {

sysOrgs.forEach(sysOrg -> {

List<SysOrg> sysOrgChildren = findChildrenByOrgId(sysOrg.getId());

if (sysOrgChildren != null && !sysOrgChildren.isEmpty()) {

sysOrg.setChildren(sysOrgChildren);

}

});

}

return sysOrgs;

}

}

/\*\*

\* @author yb

\* @description 针对表【sys\_post】的数据库操作Service实现

\* @createDate 2023-02-23 17:08:24

\*/

@Service

public class SysPostServiceImpl extends ServiceImpl<SysPostMapper, SysPost>

implements ISysPostService {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_role\_menu】的数据库操作Service实现

\* @createDate 2023-02-10 10:17:11

\*/

@Service

public class SysRoleMenuServiceImpl extends ServiceImpl<SysRoleMenuMapper, SysRoleMenu>

implements ISysRoleMenuService {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_role(角色)】的数据库操作Service实现

\* @createDate 2023-02-10 10:17:11

\*/

@Service

public class SysRoleServiceImpl extends ServiceImpl<SysRoleMapper, SysRole>

implements ISysRoleService {

@Override

public List<String> findUsernameByRoleId(String roleId) {

return baseMapper.findUsernameByRoleId(roleId);

}

}

/\*\*

\* @author yb

\* @description 针对表【sys\_tenant\_pkg\_menu(套餐菜单)】的数据库操作Service实现

\* @createDate 2023-05-09 17:17:23

\*/

@Service

public class SysTenantPkgMenuServiceImpl extends ServiceImpl<SysTenantPkgMenuMapper, SysTenantPkgMenu>

implements ISysTenantPkgMenuService {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_tenant\_pkg(租户套餐表)】的数据库操作Service实现

\* @createDate 2023-05-09 17:17:23

\*/

@Service

@TenantIgnore

public class SysTenantPkgServiceImpl extends ServiceImpl<SysTenantPkgMapper, SysTenantPkg>

implements ISysTenantPkgService {

@Autowired

ISysTenantPkgMenuService sysTenantPkgMenuService;

@Autowired

ISysTenantService sysTenantService;

@Autowired

ISysRoleService sysRoleService;

@Autowired

ISysRoleMenuService sysRoleMenuService;

@Autowired

ISysUserService sysUserService;

@Override

public Page<SysTenantPkgVO> findPage(Page<SysTenantPkgVO> sysTenantPkgVOPage, String pkgName) {

List<SysTenantPkgVO> sysTenantPkgVOS = baseMapper.findPage(sysTenantPkgVOPage, pkgName);

sysTenantPkgVOS.forEach(sysTenantPkgVO -> {

List<String> sysMenuIdsPc = baseMapper.findMenuIdsBySys(sysTenantPkgVO.getId(), "pc");

sysTenantPkgVO.setSysMenuIdsPc(sysMenuIdsPc);

List<String> sysMenuIdsApp = baseMapper.findMenuIdsBySys(sysTenantPkgVO.getId(), "app");

sysTenantPkgVO.setSysMenuIdsApp(sysMenuIdsApp);

});

sysTenantPkgVOPage.setRecords(sysTenantPkgVOS);

return sysTenantPkgVOPage;

}

@Override

public void updatePkg(SysTenantPkgDTO sysTenantPkgDTO) {

this.updateById(sysTenantPkgDTO);

//删除之前的套餐菜单

sysTenantPkgMenuService.remove(new QueryWrapper<SysTenantPkgMenu>().lambda().eq(SysTenantPkgMenu::getSysTenantPkgId, sysTenantPkgDTO.getId()));

savePkgMenus(sysTenantPkgDTO);

//将关联的套餐对应的租户角色为'admin'的角色编码菜单权限修改为当前的

//查询关联的租户集合

List<SysTenant> sysTenants = sysTenantService.list(new QueryWrapper<SysTenant>().lambda().eq(SysTenant::getPackageId, sysTenantPkgDTO.getId()));

List<String> sysMenuIds = CollUtil.unionAll(sysTenantPkgDTO.getSysMenuIdsPc(), sysTenantPkgDTO.getSysMenuIdsApp());

if (sysTenants != null && !sysTenants.isEmpty()) {

sysTenants.forEach(sysTenant -> {

//删除减掉的菜单(没在新修改的菜单列表中的菜单)

//删除 admin 原有的roleMenu关联

sysRoleMenuService.remove(new QueryWrapper<SysRoleMenu>().lambda()

.eq(SysRoleMenu::getTenantId, sysTenant.getId())

.notIn(SysRoleMenu::getSysMenuId, sysMenuIds)

);

//将对应的角色

SysRole sysRole = sysRoleService.getOne(new QueryWrapper<SysRole>()

.lambda().eq(SysRole::getCode, "admin")

.eq(SysRole::getTenantId, sysTenant.getId())

);

if (sysRole != null) {

//删除 admin 角色重新授权admin角色对应菜单

sysRoleMenuService.remove(new QueryWrapper<SysRoleMenu>().lambda()

.eq(SysRoleMenu::getTenantId, sysTenant.getId())

.eq(SysRoleMenu::getSysRoleId, sysRole.getId())

);

//现在 admin 角色菜单

List<SysRoleMenu> sysRoleMenus = new ArrayList<>();

if (sysMenuIds != null && !sysMenuIds.isEmpty()) {

sysMenuIds.forEach(menuId -> {

SysRoleMenu sysRoleMenu = new SysRoleMenu();

sysRoleMenu.setTenantId(sysTenant.getId());

sysRoleMenu.setSysRoleId(sysRole.getId());

sysRoleMenu.setSysMenuId(menuId);

sysRoleMenus.add(sysRoleMenu);

});

sysRoleMenuService.saveBatch(sysRoleMenus);

//退出该租户全部用户信息

List<SysUser> sysUserList = sysUserService.list(new QueryWrapper<SysUser>().lambda().eq(SysUser::getTenantId, sysTenant.getId()));

if (sysUserList != null && !sysUserList.isEmpty()) {

sysUserList.forEach(sysUser -> {

StpUtil.logout(sysUser.getUsername());

});

}

}

}

});

}

}

@Override

public void savePkg(SysTenantPkgDTO sysTenantPkgDTO) {

this.save(sysTenantPkgDTO);

savePkgMenus(sysTenantPkgDTO);

}

@Override

public void removePkg(String id) {

//判断当前套餐下是否有租户，如果有租户就不能删除

long count = sysTenantService.count(new QueryWrapper<SysTenant>().lambda().eq(SysTenant::getPackageId, id));

if (count > 0) {

throw new ServiceException("当前套餐已经关联租户无法删除!");

}

this.removeById(id);

sysTenantPkgMenuService.remove(new QueryWrapper<SysTenantPkgMenu>().lambda().eq(SysTenantPkgMenu::getSysTenantPkgId, id));

}

private void savePkgMenus(SysTenantPkgDTO sysTenantPkgDTO) {

List<SysTenantPkgMenu> sysTenantPkgMenus = new ArrayList<>();

List<String> sysMenuIds = CollUtil.unionAll(sysTenantPkgDTO.getSysMenuIdsPc(), sysTenantPkgDTO.getSysMenuIdsApp());

sysMenuIds.forEach(sysMenuId -> {

SysTenantPkgMenu sysTenantPkgMenu = new SysTenantPkgMenu();

sysTenantPkgMenu.setSysTenantPkgId(sysTenantPkgDTO.getId());

sysTenantPkgMenu.setSysMenuId(sysMenuId);

sysTenantPkgMenus.add(sysTenantPkgMenu);

});

if (!sysTenantPkgMenus.isEmpty()) {

sysTenantPkgMenuService.saveBatch(sysTenantPkgMenus);

}

}

}

/\*\*

\* @author yb

\* @description 针对表【sys\_tenant(租户信息表)】的数据库操作Service实现

\* @createDate 2023-05-09 17:17:23

\*/

@Service

@TenantIgnore

public class SysTenantServiceImpl extends ServiceImpl<SysTenantMapper, SysTenant>

implements ISysTenantService {

@Autowired

ISysTenantPkgService sysTenantPkgService;

@Autowired

ISysTenantPkgMenuService sysTenantPkgMenuService;

@Autowired

ISysUserService sysUserService;

@Autowired

ISysOrgService sysOrgService;

@Autowired

ISysRoleService sysRoleService;

@Autowired

ISysRoleMenuService sysRoleMenuService;

@Autowired

ISysUserRoleService sysUserRoleService;

@Autowired

ISysDictCategoryService sysDictCategoryService;

@Override

public Page<SysTenantVO> findPage(Page<SysTenantVO> sysTenantVOPage, String tenantName) {

sysTenantVOPage.setRecords(baseMapper.findPage(sysTenantVOPage, tenantName));

return sysTenantVOPage;

}

@Override

@Transactional(rollbackFor = Exception.class)

public void addSysTenant(SysTenantAddDTO sysTenantAddDTO) {

SysTenant sysTenant = new SysTenant();

BeanUtils.copyProperties(sysTenantAddDTO, sysTenant);

//判断套餐是否存在

if (baseMapper.selectOne(new QueryWrapper<SysTenant>().lambda().eq(SysTenant::getTenantName, sysTenantAddDTO.getTenantName())) != null) {

throw new JcglException("租户名称不能重复");

}

//判断套餐是否存在

SysTenantPkg sysTenantPkg = sysTenantPkgService.getById(sysTenant.getPackageId());

if (sysTenantPkg == null) {

throw new JcglException("不存在的套餐");

}

//判断用户名是否重复

long userCount = sysUserService.count(new QueryWrapper<SysUser>().lambda()

.eq(SysUser::getUsername, sysTenantAddDTO.getUsername()));

if (userCount > 0) {

throw new JcglException("用户名已经存在");

}

baseMapper.insert(sysTenant);

//创建一个部门

SysOrg sysOrg = new SysOrg();

sysOrg.setOrgName(sysTenant.getTenantName());

sysOrg.setOrgType(1);

sysOrg.setParentId("0");

sysOrg.setFullOrgName(sysTenant.getTenantName());

sysOrg.setTenantId(sysTenant.getId());

sysOrgService.save(sysOrg);

sysOrg.setAncestors(sysOrg.getId());

sysOrgService.updateById(sysOrg);

//创建一个用户

SysUser sysUser = new SysUser();

sysUser.setUsername(sysTenantAddDTO.getUsername());

sysUser.setPassword(BCrypt.hashpw(sysTenantAddDTO.getPassword()));

if ("2".equals(sysTenant.getStatus())) {

sysUser.setEnabled("1");

} else {

sysUser.setEnabled("0");

}

sysUser.setEmpStatus("1");

sysUser.setEmpName("系统管理员");

sysUser.setGender("1");

sysUser.setSysOrgId(sysOrg.getId());

sysUser.setTenantId(sysTenant.getId());

sysUser.setMobile(sysTenantAddDTO.getMobile());

sysUserService.save(sysUser);

//创建一个默认的"系统管理员角色"

SysRole sysRole = new SysRole();

sysRole.setCode("admin");

sysRole.setRoleName("系统管理员");

sysRole.setTenantId(sysTenant.getId());

sysRoleService.save(sysRole);

//给这个角色分配套餐里面的菜单ID

//1查询套餐menuid

List<SysTenantPkgMenu> sysTenantPkgMenuList = sysTenantPkgMenuService.list(new QueryWrapper<SysTenantPkgMenu>().lambda().eq(SysTenantPkgMenu::getSysTenantPkgId, sysTenantPkg.getId()));

List<SysRoleMenu> sysRoleMenus = new ArrayList<>();

sysTenantPkgMenuList.forEach(sysTenantPkgMenu -> {

SysRoleMenu sysRoleMenu = new SysRoleMenu();

sysRoleMenu.setSysMenuId(sysTenantPkgMenu.getSysMenuId());

sysRoleMenu.setSysRoleId(sysRole.getId());

sysRoleMenu.setTenantId(sysTenant.getId());

sysRoleMenus.add(sysRoleMenu);

});

sysRoleMenuService.saveBatch(sysRoleMenus);

//给创建的这个用户分配 系统管理员角色

SysUserRole sysUserRole = new SysUserRole();

sysUserRole.setSysUserId(sysUser.getId());

sysUserRole.setSysRoleId(sysRole.getId());

sysUserRole.setTenantId(sysTenant.getId());

sysUserRoleService.save(sysUserRole);

//初始化数据字典

sysDictCategoryService.initDictData(sysTenant.getId());

}

@Override

public void updateTenant(SysTenant sysTenantUpdate) {

SysTenant sysTenantNow = getById(sysTenantUpdate.getId());

this.updateById(sysTenantUpdate);

//当租户状态不是激活状态‘2’得情况下将全部账户设置有停用

if (!"2".equals(sysTenantUpdate.getStatus())) {

sysUserService.setStatusByTenantId(sysTenantUpdate.getId(), "0");

//将当前租户下用户全部强制退出

List<SysUser> sysUsers = sysUserService.list(new QueryWrapper<SysUser>().lambda().eq(SysUser::getTenantId, sysTenantUpdate.getId()));

if (sysUsers != null && !sysUsers.isEmpty()) {

sysUsers.forEach(sysUser -> {

StpUtil.logout(sysUser.getUsername());

});

}

} else {

sysUserService.setStatusByTenantId(sysTenantUpdate.getId(), "1");

}

if (!sysTenantNow.getPackageId().equals(sysTenantUpdate.getPackageId())) {

//将关联的套餐对应的租户角色为'admin'的角色编码菜单权限修改为当前的

//查询关联的租户集合

List<SysTenantPkgMenu> sysTenantPkgMenus = sysTenantPkgMenuService.list(new QueryWrapper<SysTenantPkgMenu>().lambda().eq(SysTenantPkgMenu::getSysTenantPkgId, sysTenantUpdate.getPackageId()));

List<String> sysMenuIds = sysTenantPkgMenus.stream().map(SysTenantPkgMenu::getSysMenuId).collect(Collectors.toList());

//删除减掉的菜单(没在新修改的菜单列表中的菜单)

//删除 admin 原有的roleMenu关联

sysRoleMenuService.remove(new QueryWrapper<SysRoleMenu>().lambda()

.eq(SysRoleMenu::getTenantId, sysTenantUpdate.getId())

.notIn(SysRoleMenu::getSysMenuId, sysMenuIds)

);

//将对应的角色

SysRole sysRole = sysRoleService.getOne(new QueryWrapper<SysRole>()

.lambda().eq(SysRole::getCode, "admin")

.eq(SysRole::getTenantId, sysTenantUpdate.getId())

);

if (sysRole != null) {

//删除 admin 角色重新授权admin角色对应菜单

sysRoleMenuService.remove(new QueryWrapper<SysRoleMenu>().lambda()

.eq(SysRoleMenu::getTenantId, sysTenantUpdate.getId())

.eq(SysRoleMenu::getSysRoleId, sysRole.getId())

);

//现在 admin 角色菜单

List<SysRoleMenu> sysRoleMenus = new ArrayList<>();

if (!sysMenuIds.isEmpty()) {

sysMenuIds.forEach(menuId -> {

SysRoleMenu sysRoleMenu = new SysRoleMenu();

sysRoleMenu.setTenantId(sysTenantUpdate.getId());

sysRoleMenu.setSysRoleId(sysRole.getId());

sysRoleMenu.setSysMenuId(menuId);

sysRoleMenus.add(sysRoleMenu);

});

sysRoleMenuService.saveBatch(sysRoleMenus);

}

//退出该租户全部用户信息

List<SysUser> sysUserList= sysUserService.list(new QueryWrapper<SysUser>().lambda().eq(SysUser::getTenantId,sysTenantUpdate.getId()));

if (sysUserList!=null&&!sysUserList.isEmpty()){

sysUserList.forEach(sysUser -> {

StpUtil.logout(sysUser.getUsername());

});

}

}

}

}

@Override

public void removeTenant(String tenantId) {

//租户管理了账号时不能删除

long count = sysUserService.findCountByTenantId(tenantId);

if (count>0){

throw new ServiceException("当前租户已有账号无法删除!");

}

this.removeById(tenantId);

}

}

/\*\*

\* @author yb

\* @description 针对表【sys\_user\_flow\_role(用户流程角色)】的数据库操作Service实现

\* @createDate 2023-05-30 13:39:43

\*/

@Service

public class SysUserFlowRoleServiceImpl extends ServiceImpl<SysUserFlowRoleMapper, SysUserFlowRole>

implements ISysUserFlowRoleService {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_user\_org(用户组织)】的数据库操作Service实现

\* @createDate 2023-05-31 10:45:14

\*/

@Service

public class SysUserOrgServiceImpl extends ServiceImpl<SysUserOrgMapper, SysUserOrg>

implements ISysUserOrgService {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_user\_role】的数据库操作Service实现

\* @createDate 2023-02-10 10:17:11

\*/

@Service

public class SysUserRoleServiceImpl extends ServiceImpl<SysUserRoleMapper, SysUserRole>

implements ISysUserRoleService {

}

/\*\*

\* @author yb

\* @description 针对表【sys\_user(系统用户)】的数据库操作Service实现

\* @createDate 2023-02-10 10:17:11

\*/

@Service

public class SysUserServiceImpl extends ServiceImpl<SysUserMapper, SysUser>

implements ISysUserService {

@Autowired

ISysFlowRoleService sysFlowRoleService;

@Autowired

ISysOrgService sysOrgService;

@Override

public SysUser login(String username, String password) {

SysUser sysUser = baseMapper.selectOne(new QueryWrapper<SysUser>().lambda()

.eq(SysUser::getEnabled, "1")

.eq(SysUser::getEmpStatus, "1")

.eq(SysUser::getUsername, username));

if (sysUser == null) {

return null;

}

return BCrypt.checkpw(password, sysUser.getPassword()) ? sysUser : null;

}

@Override

public Page<SysUserVO> findUserPage(Page<SysUserVO> sysUserVOPage, String sysOrgId, String empName, String mobile) {

return baseMapper.findUserPage(sysUserVOPage, sysOrgId, empName, mobile);

}

@Override

public int updateEnabled(String sysUserId, int enabled) {

return baseMapper.updateEnabled(sysUserId, enabled);

}

@Override

public int updateEmpStatus(String sysUserId, int status) {

return baseMapper.updateEmpStatus(sysUserId, status);

}

@Override

public SysUserVO findUserVOByUserId(String id) {

return baseMapper.findUserVOByUserId(id);

}

@Override

@TenantIgnore

public List<String> findMenuCodesByUsername(String username) {

return baseMapper.findMenuCodesByUsername(username);

}

@Override

@TenantIgnore

public List<String> findRoleCodeByUsername(String username) {

return baseMapper.findRoleCodeByUsername(username);

}

@Override

public List<String> findAuthSysOrgIds(String sysOrgId) {

if ("0".equals(sysOrgId)) {

sysOrgId = null;

}

return baseMapper.findAuthSysOrgIds(sysOrgId);

}

@Override

@TenantIgnore

public String findTenantId(String userId) {

return baseMapper.findTenantId(userId);

}

@Override

@TenantIgnore

public SysUser findSysUserByUsername(String username) {

return baseMapper.selectOne(new QueryWrapper<SysUser>().lambda().eq(SysUser::getUsername, username));

}

@Override

public long findCountByTenantId(String id) {

return baseMapper.selectCount(new QueryWrapper<SysUser>().lambda().eq(SysUser::getTenantId, id));

}

@Override

public long setStatusByTenantId(String tenantId, String enabled) {

return baseMapper.setStatusByTenantId(tenantId, enabled);

}

@Override

@TenantIgnore

public List<SysFlowRole> findFlowRoleCodes(String sysUserId) {

return baseMapper.findFlowRoleCodes(sysUserId);

}

@Override

public List<SysUser> findUserEmpNamesByFlowCode(String flowRoleCode, String sysOrgId) {

List<SysUser> sysUserList = new ArrayList<>();

//先判断流程是上下级还是全局

SysFlowRole sysFlowRole = sysFlowRoleService.getOne(new QueryWrapper<SysFlowRole>().lambda().eq(SysFlowRole::getCode, flowRoleCode));

if (sysFlowRole != null) {

//流程上下级别

if ("1".equals(sysFlowRole.getType())) {

List<String> upDownOrgIds = sysOrgService.selectUpDownOrgIds(sysOrgId);

List<SysUser> sysUserListUpDown = sysFlowRoleService.findSysUserMainByCodeOrg(flowRoleCode, upDownOrgIds);

sysUserList.addAll(sysUserListUpDown);

} else {

//全局

List<SysUser> sysUserAll = sysFlowRoleService.findSysUserMainByCodeOrg(flowRoleCode, null);

sysUserList.addAll(sysUserAll);

}

}

return sysUserList;

}

@Override

public SysUser findSysUserByPhoneNumber(String phoneNumber) {

return baseMapper.selectOne(new QueryWrapper<SysUser>().lambda()

.eq(SysUser::getMobile, phoneNumber)

.eq(SysUser::getEnabled, 1));

}

@Override

public List<SysRole> findRolesByUserId(String sysUserId) {

return baseMapper.findRolesByUserId(sysUserId);

}

}

/\*\*

\* 时间相关工具

\*

\* @author wanggang

\* @date 2023/4/4 11:22

\*/

public class DateUtil {

private static final String FORMAT = "yyyy-MM-dd HH:mm:ss";

private static Date simpleFormatToDate(SimpleDateFormat simpleDateFormat, String strTime) {

Date dateTime = null;

try {

dateTime = simpleDateFormat.parse(strTime);

} catch (ParseException e) {

e.printStackTrace();

}

return dateTime;

}

/\*

将时间字符按指定格式转换成时间类型

\*/

public static Date strToDate(String format, String strTime) {

SimpleDateFormat simpleDateFormat = new SimpleDateFormat(format);

return simpleFormatToDate(simpleDateFormat, strTime);

}

/\*

将时间字符按指定格式转换成时间类型

\*/

public static Date strToDate(String format, Locale locale, String strTime) {

SimpleDateFormat simpleDateFormat = new SimpleDateFormat(format, locale);

return simpleFormatToDate(simpleDateFormat, strTime);

}

/\*

将时间按指定格式转换字符串

\*/

public static String dateToStr(String format, Date date) {

if (date == null)

return "";

SimpleDateFormat simpleDateFormat = new SimpleDateFormat(format);

return simpleDateFormat.format(date);

}

/\*\*

\* @Description: 获得日期一天最开始的时间

\* @Param:

\* @return:

\* @Author: Mr.Wang

\* @Date: 2019/7/30

\*/

public static Date getStartYearTime(Date date) {

Calendar calendar = Calendar.getInstance();

calendar.setTime(date);

String start = calendar.get(Calendar.YEAR) + "-01-01 00:00:00";

return strToDate(FORMAT, start);

}

/\*\*

\* @Description: 获得日期一天最开始的时间

\* @Param:

\* @return:

\* @Author: Mr.Wang

\* @Date: 2019/7/30

\*/

public static Date getStartDayTime(Date date) {

Calendar calendar = Calendar.getInstance();

calendar.setTime(date);

String start = calendar.get(Calendar.YEAR) + "-" + (calendar.get(Calendar.MONTH) + 1) + "-" + calendar.get(Calendar.DATE) + " 00:00:00";

return strToDate(FORMAT, start);

}

/\*\*

\* @Description: 获得一天中结束的时间

\* @Param:

\* @return:

\* @Author: Mr.Wang

\* @Date: 2019/7/30

\*/

public static Date getEndDayTime(Date date) {

Calendar calendar = Calendar.getInstance();

calendar.setTime(date);

String end = calendar.get(Calendar.YEAR) + "-" + (calendar.get(Calendar.MONTH) + 1) + "-" + calendar.get(Calendar.DATE) + " 23:59:59";

return strToDate(FORMAT, end);

}

/\*\*

\* @Description: 获得该时间的小时的开始

\* @Param:

\* @return:

\* @Author: Mr.Wang

\* @Date: 2019/7/30

\*/

public static Date getStartHourTime(Date date) {

Calendar calendar = Calendar.getInstance();

calendar.setTime(date);

String end = calendar.get(Calendar.YEAR) + "-" + (calendar.get(Calendar.MONTH) + 1) + "-" + calendar.get(Calendar.DATE) + " " + calendar.get(Calendar.HOUR\_OF\_DAY) + ":00:00";

return strToDate(FORMAT, end);

}

/\*\*

\* @Description: 获得该时间的小时的结束

\* @Param:

\* @return:

\* @Author: Mr.Wang

\* @Date: 2019/7/30

\*/

public static Date getEndHourTime(Date date) {

Calendar calendar = Calendar.getInstance();

calendar.setTime(date);

String end = calendar.get(Calendar.YEAR) + "-" + (calendar.get(Calendar.MONTH) + 1) + "-" + calendar.get(Calendar.DATE) + " " + calendar.get(Calendar.HOUR\_OF\_DAY) + ":59:59";

return strToDate(FORMAT, end);

}

/\*\*

\* 返回某个日期下几天的日期

\*/

public static Date getNextDay(Date date, int i) {

Calendar cal = new GregorianCalendar();

cal.setTime(date);

cal.set(Calendar.DATE, cal.get(Calendar.DATE) + i);

return cal.getTime();

}

/\*\*

\* 返回某个日期前几天的日期

\*/

public static Date getFrontDay(Date date, int i) {

Calendar cal = new GregorianCalendar();

cal.setTime(date);

cal.set(Calendar.DATE, cal.get(Calendar.DATE) - i);

return cal.getTime();

}

/\*\*

\* @Description: 查看两个时间区间是否有交集

\* @Param: time1 时间区间一开始时间

\* @Param: time2 时间区间一结束时间

\* @Param: time3 时间区间二开始时间

\* @Param: time4 时间区间二结束时间

\* @Return: boolean

\* @Author: CWR

\* @Date: 2020/2/28 18:27

\*/

public static boolean hasCross(Date time1, Date time2, Date time3, Date time4) {

try {

return (time3.getTime() <= time1.getTime() && time4.getTime() >= time1.getTime())

|| (time4.getTime() >= time1.getTime() && time3.getTime() <= time2.getTime());

} catch (Exception e) {

return false;

}

}

/\*\*

\* @Description: 解析日期(yyyy - MM - dd HH : mm : ss)

\* @Param: date 日期

\* @Return: java.lang.String

\* @Author: CWR

\* @Date: 2020/2/28 18:01

\*/

public static Date parse(String dateStr) {

try {

return new SimpleDateFormat(FORMAT).parse(dateStr);

} catch (ParseException e) {

return new Date();

}

}

// 获取本月的开始时间

public static Date getBeginDayOfMonth() {

Calendar calendar = Calendar.getInstance();

calendar.set(getNowYear(), getNowMonth() - 1, 1);

return getStartDayTime(calendar.getTime());

}

// 获取本月的结束时间

public static Date getEndDayOfMonth() {

Calendar calendar = Calendar.getInstance();

calendar.set(getNowYear(), getNowMonth() - 1, 1);

int day = calendar.getActualMaximum(5);

calendar.set(getNowYear(), getNowMonth() - 1, day);

return getEndDayTime(calendar.getTime());

}

// 获取今年是哪一年

public static Integer getNowYear() {

Date date = new Date();

GregorianCalendar gc = (GregorianCalendar) Calendar.getInstance();

gc.setTime(date);

return Integer.valueOf(gc.get(1));

}

// 获取本月是哪一月

public static int getNowMonth() {

Date date = new Date();

GregorianCalendar gc = (GregorianCalendar) Calendar.getInstance();

gc.setTime(date);

return gc.get(2) + 1;

}

// 获取本天是哪一天

public static int getNowDay() {

Date date = new Date();

GregorianCalendar gc = (GregorianCalendar) Calendar.getInstance();

gc.setTime(date);

return gc.get(5);

}

// 获取本年的开始时间

public static Date getBeginDayOfYear() {

Calendar cal = Calendar.getInstance();

cal.set(Calendar.YEAR, getNowYear());

cal.set(Calendar.MONTH, Calendar.JANUARY);

cal.set(Calendar.DATE, 1);

return getDayStartTime(cal.getTime());

}

// 获取本年的结束时间

public static Date getEndDayOfYear() {

Calendar cal = Calendar.getInstance();

cal.set(Calendar.YEAR, getNowYear());

cal.set(Calendar.MONTH, Calendar.DECEMBER);

cal.set(Calendar.DATE, 31);

return getDayEndTime(cal.getTime());

}

// 获取某一年的开始时间

public static Date getBeginDayOfYear(Integer year) {

Calendar cal = Calendar.getInstance();

cal.set(Calendar.YEAR, year);

cal.set(Calendar.MONTH, Calendar.JANUARY);

cal.set(Calendar.DATE, 1);

return getDayStartTime(cal.getTime());

}

// 获取某一年的结束时间

public static Date getEndDayOfYear(Integer year) {

Calendar cal = Calendar.getInstance();

cal.set(Calendar.YEAR, year);

cal.set(Calendar.MONTH, Calendar.DECEMBER);

cal.set(Calendar.DATE, 31);

return getDayEndTime(cal.getTime());

}

// 获取某个日期的开始时间

public static Timestamp getDayStartTime(Date d) {

Calendar calendar = Calendar.getInstance();

if (null != d)

calendar.setTime(d);

calendar.set(calendar.get(Calendar.YEAR), calendar.get(Calendar.MONTH),

calendar.get(Calendar.DAY\_OF\_MONTH), 0, 0, 0);

calendar.set(Calendar.MILLISECOND, 0);

return new Timestamp(calendar.getTimeInMillis());

}

// 获取某个日期的结束时间

public static Timestamp getDayEndTime(Date d) {

Calendar calendar = Calendar.getInstance();

if (null != d)

calendar.setTime(d);

calendar.set(calendar.get(Calendar.YEAR), calendar.get(Calendar.MONTH),

calendar.get(Calendar.DAY\_OF\_MONTH), 23, 59, 59);

calendar.set(Calendar.MILLISECOND, 999);

return new Timestamp(calendar.getTimeInMillis());

}

// 获取当年每月的开始时间

public static Date getBeginDayOfMonth(int year, int month) {

Calendar calendar = Calendar.getInstance();

calendar.set(year, month - 1, 1);

return getDayStartTime(calendar.getTime());

}

// 获取当年每月的结束时间

public static Date getEndDayOfMonth(int year, int month) {

Calendar calendar = Calendar.getInstance();

int day = calendar.getActualMaximum(5);

calendar.set(year, month - 1, day);

return getDayEndTime(calendar.getTime());

}

// 获取具体某一天的开始时间

public static Date getBeginDayOfDay(int year, int month, int day) {

Calendar calendar = Calendar.getInstance();

calendar.set(year, month - 1, day);

return getDayStartTime(calendar.getTime());

}

// 获取当年每月的结束时间

public static Date getEndDayOfDay(int year, int month, int day) {

Calendar calendar = Calendar.getInstance();

calendar.set(year, month - 1, day);

return getDayEndTime(calendar.getTime());

}

/\*\*

\* @param nowDate, sqlDate

\* @return: int

\* @author: wg

\* @date: 2022/2/11

\* description:计算两个时间的时间差 (小时)

\*/

public static BigDecimal different(Date nowDate, Date startDate) {

DateFormat df = new SimpleDateFormat(FORMAT);

String newDate = df.format(nowDate);

String startDate1 = df.format(startDate);

try {

nowDate = df.parse(newDate);

startDate = df.parse(startDate1);

} catch (ParseException e) {

e.printStackTrace();

}

long time = nowDate.getTime();

long time2 = startDate.getTime();

//转换为小数

BigDecimal endTime = new BigDecimal(time);

BigDecimal startTime = new BigDecimal(time2);

//24\*3600\*1000

BigDecimal divide = new BigDecimal(86400000);

return (endTime.subtract(startTime)).divide(divide, 1, RoundingMode.HALF\_UP);

}

/\*\*

\* @param endDate, sqlDate

\* @return: int

\* @author: wg

\* @date: 2022/2/11

\* description:计算两个时间的时间差(秒)

\*/

public static long differentSecond(Date endDate, Date startDate) {

DateFormat df = new SimpleDateFormat(FORMAT);

String newDate = df.format(endDate);

String startDate1 = df.format(startDate);

try {

endDate = df.parse(newDate);

startDate = df.parse(startDate1);

} catch (ParseException e) {

e.printStackTrace();

}

long time = endDate.getTime();

long time2 = startDate.getTime();

//1000

long result = time - time2;

return result / 1000;

}

/\*\*

\* @param date

\* @return: long

\* @author: wg

\* @date: 2022/3/16

\* description:转换成秒

\*/

public static long transformSeconds(Date date) {

DateFormat df = new SimpleDateFormat(FORMAT);

String startDate1 = df.format(date);

try {

date = df.parse(startDate1);

} catch (ParseException e) {

e.printStackTrace();

}

long time = date.getTime();

return time / 1000;

}

private DateUtil() {

}

}

/\*\*

\* ClassName：EasyExcel工具类

\* Description：excel导入导出工具， 注意：枚举类需要自己转换成自己需要的数据

\*/

@Slf4j

public class EasyExcelUtil {

private EasyExcelUtil() {

}

public static final String EXCEL\_SUFFIX = ".xlsx";

public static final String WRITE\_SHEET\_NAME = "sheet1";

// 读取文件解析监听类

public static class ExcelListener<T> extends AnalysisEventListener<T> {

private List<T> datas = new ArrayList<>();

@Override

public void invoke(T t, AnalysisContext analysisContext) {

datas.add(t);

}

@Override

public void doAfterAllAnalysed(AnalysisContext analysisContext) {

log.info("读取数据条数：{}条！", datas.size());

}

public List<T> getDatas() {

return this.datas;

}

}

/\*\*

\* 读取Excel文件返回数据集合，不包含表头，默认读取第一个sheet数据

\*

\* @param inputStream 输入流

\* @param tClass 数据映射类

\* @param excelListener 读取监听类

\* @return List 结果集

\*/

public static <T> List<T> readExcel(InputStream inputStream, Class<T> tClass, ExcelListener<T> excelListener) {

if (ObjectUtil.isNull(inputStream) || ObjectUtil.isNull(tClass) || ObjectUtil.isNull(excelListener)) {

return Collections.emptyList();

}

ExcelReaderBuilder read = EasyExcelFactory.read(inputStream, tClass, excelListener);

read.sheet().doRead();

return excelListener.getDatas();

}

/\*\*

\* 读取Excel文件返回数据集合，不包含表头，读取第x个sheet数据，不设置sheet就读取全部

\*

\* @param inputStream 输入流

\* @param sheetNo sheet编号，默认给0

\* @param tClass 数据映射类

\* @param excelListener 读取监听类

\* @return List 结果集

\*/

public static <T> List<T> readExcel(InputStream inputStream, Integer sheetNo, Class<T> tClass, ExcelListener<T> excelListener) {

if (ObjectUtil.isNull(inputStream) || ObjectUtil.isNull(tClass) || ObjectUtil.isNull(excelListener)) {

return Collections.emptyList();

}

if (sheetNo == null) {

sheetNo = 0;

}

ExcelReaderBuilder read = EasyExcelFactory.read(inputStream, tClass, excelListener);

if (ObjectUtil.isNotNull(sheetNo)) {

read.sheet(sheetNo).doRead();

} else {

ExcelReader excelReader = read.build();

excelReader.readAll();

excelReader.finish();

}

return excelListener.getDatas();

}

/\*\*

\* 不带模板输出数据到Excel，数据传输类属性用 @ExcelProperty("") 标注

\*

\* @param response 响应对象

\* @param tClass 输出格式

\* @param datas 输出的数据

\*/

public static <T> void writeExcel(HttpServletResponse response, Class<T> tClass, List<T> datas, String fileName) throws IOException, IllegalAccessException, NoSuchFieldException {

TimeInterval timer = DateUtil.timer();

if (ObjectUtil.isNull(response) || ObjectUtil.isNull(tClass)) {

return;

}

// 文件名称后缀检测

if (CharSequenceUtil.isBlank(fileName)) {

fileName = "excel.xlsx";

} else {

if (!(fileName.endsWith(EXCEL\_SUFFIX) || fileName.endsWith(".xls"))) {

fileName = fileName + EXCEL\_SUFFIX;

}

}

// 这里注意 有同学反应使用swagger 会导致各种问题，请直接用浏览器或者用postman

response.setContentType("application/vnd.openxmlformats-officedocument.spreadsheetml.sheet");

response.setCharacterEncoding("utf-8");

// 这里URLEncoder.encode可以防止中文乱码 当然和easyexcel没有关系

fileName = URLEncoder.encode(fileName, StandardCharsets.UTF\_8).replaceAll("\\+", "%20");

response.setHeader("Content-disposition", "attachment;filename\*=" + fileName);

EasyExcel.write(response.getOutputStream(), tClass)

.registerWriteHandler(new LongestMatchColumnWidthStyleStrategy())

.sheet(WRITE\_SHEET\_NAME).doWrite(datas);

log.info("导出excel数据：{}条，耗时：{}秒！", datas.size(), timer.intervalSecond());

}

/\*\*

\* 读取excel中单个sheet，返回所有数据

\*

\* @param sheet 名称, tClass 转换的资源类型, excelFile 导入的excel文件

\* @return java.util.List<T>

\* @author wanggang

\* @date 2023/5/16 9:55

\*/

public static <T> List<T> readExcelData(String sheet, Class<T> tClass, MultipartFile excelFile) {

String fileName = excelFile.getOriginalFilename();

ExcelTypeEnum excelType = null;

if (StringUtils.isNotEmpty(fileName)) {

if (fileName.endsWith(".xls")) {

excelType = ExcelTypeEnum.XLS;

} else if (fileName.endsWith(".xlsx")) {

excelType = ExcelTypeEnum.XLSX;

} else {

throw new ServiceException("文件类型不正确!");

}

}

ExcelRowListener<T> rowListener = new ExcelRowListener<>();

try {

ExcelReaderBuilder builder = EasyExcelFactory.read(excelFile.getInputStream(), tClass, rowListener)

.excelType(excelType);

if (sheet != null) {

builder.sheet(sheet).doRead();

} else {

builder.sheet().doRead();

}

} catch (IOException e) {

throw new ServiceException("无法读取excel文件!");

}

return rowListener.getRows();

}

/\*\*

\* 读取第一个sheet

\* @author wanggang

\* @date 2023/5/16 10:20

\* @param tClass, excelFile

\* @return java.util.List<T>

\*/

public static <T> List<T> readExcelData(Class<T> tClass, MultipartFile excelFile) {

return readExcelData(null, tClass, excelFile);

}

}

/\*\*

\* <h2>Entry</h2>

\* <p>请添加描述</p>

\*

\* @author Daizc

\* @date 2019/12/12

\*/

@Data

public class Entry<K, V> {

private K key;

private V value;

public Entry() {

}

public Entry(Map.Entry<K, V> entry) {

this.key = entry.getKey();

this.value = entry.getValue();

}

public Entry(K key, V value) {

this.key = key;

this.value = value;

}

}

/\*\*

\* Excel文档行处理监听器

\* @author wanggang

\* @date 2023/5/16 10:01

\*

\*/

public class ExcelRowListener<T> extends AnalysisEventListener<T> {

List<T> rows;

public List<T> getRows() {

return rows;

}

public void setRows(List<T> rows) {

this.rows = rows;

}

public ExcelRowListener(){

this.rows=new ArrayList<>();

}

@Override

public void invoke(T t, AnalysisContext analysisContext) {

rows.add(t);

}

@Override

public void doAfterAllAnalysed(AnalysisContext analysisContext) {

}

}

/\*\*

\* 实体类转换工具类

\*

\* @author wanggang

\* @date 2023/4/4 11:19

\*/

public class JcglBeanUtil {

public static void copyProperties(Object source, Object target, @Nullable boolean ignoreNullValue, @Nullable Class<?> editable,

@Nullable String... ignoreProperties) throws BeansException {

Assert.notNull(source, "Source must not be null");

Assert.notNull(target, "Target must not be null");

Class<?> actualEditable = target.getClass();

if (editable != null) {

if (!editable.isInstance(target)) {

throw new IllegalArgumentException("Target class [" + target.getClass().getName() +

"] not assignable to Editable class [" + editable.getName() + "]");

}

actualEditable = editable;

}

PropertyDescriptor[] targetPds = BeanUtils.getPropertyDescriptors(actualEditable);

List<String> ignoreList = (ignoreProperties != null ? Arrays.asList(ignoreProperties) : null);

for (PropertyDescriptor targetPd : targetPds) {

Method writeMethod = targetPd.getWriteMethod();

if (writeMethod != null && (ignoreList == null || !ignoreList.contains(targetPd.getName()))) {

PropertyDescriptor sourcePd = BeanUtils.getPropertyDescriptor(source.getClass(), targetPd.getName());

if (sourcePd != null) {

Method readMethod = sourcePd.getReadMethod();

if (readMethod != null &&

ClassUtils.isAssignable(writeMethod.getParameterTypes()[0], readMethod.getReturnType())) {

try {

if (!Modifier.isPublic(readMethod.getDeclaringClass().getModifiers())) {

readMethod.setAccessible(true);

}

Object value = readMethod.invoke(source);

if (!ignoreNullValue || value != null) {

if (!Modifier.isPublic(writeMethod.getDeclaringClass().getModifiers())) {

writeMethod.setAccessible(true);

}

writeMethod.invoke(target, value);

}

} catch (Throwable ex) {

throw new FatalBeanException(

"Could not copy property '" + targetPd.getName() + "' from source to target", ex);

}

}

}

}

}

}

public static void copyProperties(Object source, Object target, boolean ignoreNullValue) {

copyProperties(source, target, ignoreNullValue, null, (String[]) null);

}

public static Map<String, Object> getProperties(Object target, boolean ignoreNullValue) {

Assert.notNull(target, "Target must not be null");

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

Class<?> targetClass = target.getClass();

PropertyDescriptor[] targetPds = BeanUtils.getPropertyDescriptors(targetClass);

for (PropertyDescriptor targetPd : targetPds) {

if (targetPd.getWriteMethod() == null) continue;

Method readMethod = targetPd.getReadMethod();

if (!Modifier.isPublic(readMethod.getDeclaringClass().getModifiers())) {

readMethod.setAccessible(true);

}

Object value = null;

try {

value = readMethod.invoke(target);

} catch (Exception e) {

e.printStackTrace();

}

if (!ignoreNullValue || value != null) {

result.put(targetPd.getName(), value);

}

}

return result;

}

}

/\*\*

\* 根据调用的方法不同，将日志输出到对应文件，info\debug\error\warn

\*

\* @author wanggang

\* @date 2023/4/17 10:03

\*/

public class JcglLog {

public static final String blank = " ";

public static final String authorization = "Authorization";

private static final Logger LOG = LoggerFactory.getLogger(JcglLog.class);

public static void info(String msg) {

// 特别注意 不要使用 `+` 去拼接信息, 这样在控制输出级别之后, 有些日志操作不会打印, 但是它去操作了新字符串创建, 使用 `{}` 去代替 `+`

LOG.info("{}", msg);

}

public static void error(String msg) {

LOG.error("{}", msg);

}

public static void error(Exception ex) {

LOG.error(ex.getMessage(), ex);

}

public static void error(String msg, Exception ex) {

LOG.error(msg, ex);

}

public static void error(Throwable ex) {

LOG.error(ex.getMessage(), ex);

}

public static void warn(String msg) {

LOG.warn("{}", msg);

}

public static void debug(String msg) {

LOG.debug("{}", msg);

}

public static void beforeReturn(String postMsg) {

ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();

if (attributes != null) {

HttpServletRequest request = attributes.getRequest();

StringBuilder urlSb = new StringBuilder();

urlSb.append(" beforeReturn ");

urlSb.append(blank + authorization + blank).append(request.getHeader(authorization));

urlSb.append(" IP ").append(request.getRemoteAddr());

urlSb.append(" Port ").append(request.getRemotePort());

urlSb.append(" URL ").append(request.getRequestURL());// 记录下请求内容

urlSb.append(" Method ").append(request.getMethod());

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

while (enu.hasMoreElements()) {

String name = enu.nextElement();

urlSb.append(String.format(" name:{},value:{}", name, request.getParameter(name)));

}

debug(urlSb.toString());

if (StringUtils.isNotEmpty(postMsg)) debug(postMsg);

}

}

public static void afterReturn(Object returnMsg) {

ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();

if (attributes != null) {

HttpServletRequest request = attributes.getRequest();

StringBuilder urlSb = new StringBuilder();

urlSb.append(" afterReturn ");

urlSb.append(blank + authorization + blank).append(request.getHeader(authorization));

urlSb.append(" IP ").append(request.getRemoteAddr());

urlSb.append(" Port ").append(request.getRemotePort());

urlSb.append(" URL ").append(request.getRequestURL());// 记录下请求内容

urlSb.append(" Method ").append(request.getMethod());

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

while (enu.hasMoreElements()) {

String name = enu.nextElement();

urlSb.append(String.format(" name:{},value:{}", name, request.getParameter(name)));

}

debug(urlSb.toString());

if (returnMsg instanceof String) {

if (returnMsg != null) debug(returnMsg.toString());

} else {

if (returnMsg != null) debug(JSONObject.toJSONString(returnMsg));

}

}

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/3/6 15:07

\*/

@Component

public class MinioUtil {

final

MinioConfig prop;

final

MinioClient minioClient;

@Autowired

IFileAttachService fileAttachService;

public MinioUtil(MinioConfig prop, MinioClient minioClient) {

this.prop = prop;

this.minioClient = minioClient;

if (!bucketExists(prop.getBucketName())) {

makeBucket(prop.getBucketName());

}

}

/\*\*

\* 查看存储bucket是否存在

\*

\* @return boolean

\*/

public Boolean bucketExists(String bucketName) {

Boolean found;

try {

found = minioClient.bucketExists(BucketExistsArgs.builder().bucket(bucketName).build());

} catch (Exception e) {

e.printStackTrace();

return false;

}

return found;

}

/\*\*

\* 创建存储bucket

\*

\* @return Boolean

\*/

public Boolean makeBucket(String bucketName) {

try {

minioClient.makeBucket(MakeBucketArgs.builder()

.bucket(bucketName)

.build());

} catch (Exception e) {

e.printStackTrace();

return false;

}

return true;

}

/\*\*

\* 删除存储bucket

\*

\* @return Boolean

\*/

public Boolean removeBucket(String bucketName) {

try {

minioClient.removeBucket(RemoveBucketArgs.builder()

.bucket(bucketName)

.build());

} catch (Exception e) {

e.printStackTrace();

return false;

}

return true;

}

/\*\*

\* 获取全部bucket

\*/

public List<Bucket> getAllBuckets() {

try {

List<Bucket> buckets = minioClient.listBuckets();

return buckets;

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

/\*\*

\* 文件上传

\*

\* @param file 文件

\* @return Boolean

\*/

public String upload(MultipartFile file) {

String originalFilename = file.getOriginalFilename();

if (Strings.isNullOrEmpty(originalFilename)) {

throw new RuntimeException();

}

String fileName = UUID.fastUUID().toString(true) + originalFilename.substring(originalFilename.lastIndexOf("."));

String objectName = DateUtil.format(new Date(), "yyyyMMdd") + "/" + fileName;

InputStream is = null;

try {

is = file.getInputStream();

PutObjectArgs objectArgs = PutObjectArgs.builder().bucket(prop.getBucketName()).object(objectName)

.stream(is, file.getSize(), -1).contentType(file.getContentType()).build();

//文件名称相同会覆盖

minioClient.putObject(objectArgs);

FileAttach fileList = new FileAttach();

InputStream md5Stream = file.getInputStream();

fileList.setMd5(DigestUtils.md5Hex(md5Stream));

md5Stream.close();

fileList.setUuid(objectName);

fileList.setContentType(file.getContentType());

fileList.setFilename(file.getOriginalFilename());

fileList.setCreateTime(new Date());

fileAttachService.save(fileList);

} catch (Exception e) {

e.printStackTrace();

return null;

} finally {

try {

assert is != null;

is.close();

} catch (IOException e) {

throw new RuntimeException(e);

}

}

return objectName;

}

/\*\*

\* 文件上传 返回文件对象

\*

\* @param file 文件

\* @return Boolean

\*/

public FileAttach uploadFile(MultipartFile file) {

String originalFilename = file.getOriginalFilename();

if (Strings.isNullOrEmpty(originalFilename)) {

throw new RuntimeException();

}

String fileName = UUID.fastUUID().toString(true) + originalFilename.substring(originalFilename.lastIndexOf("."));

String objectName = DateUtil.format(new Date(), "yyyyMMdd") + "/" + fileName;

InputStream is = null;

FileAttach fileList = new FileAttach();

try {

is = file.getInputStream();

PutObjectArgs objectArgs = PutObjectArgs.builder().bucket(prop.getBucketName()).object(objectName)

.stream(is, file.getSize(), -1).contentType(file.getContentType()).build();

//文件名称相同会覆盖

minioClient.putObject(objectArgs);

InputStream md5Stream = file.getInputStream();

fileList.setMd5(DigestUtils.md5Hex(md5Stream));

md5Stream.close();

fileList.setUuid(objectName);

fileList.setContentType(file.getContentType());

fileList.setFilename(file.getOriginalFilename());

fileList.setCreateTime(new Date());

fileAttachService.save(fileList);

} catch (Exception e) {

e.printStackTrace();

return null;

} finally {

try {

assert is != null;

is.close();

} catch (IOException e) {

throw new RuntimeException(e);

}

}

return fileList;

}

/\*\*

\* 预览

\*

\* @param fileName

\* @return

\*/

public String preview(String fileName) {

// 查看文件地址

GetPresignedObjectUrlArgs build = GetPresignedObjectUrlArgs.builder().bucket(prop.getBucketName()).object(fileName).method(Method.GET).build();

try {

return minioClient.getPresignedObjectUrl(build);

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

/\*\*

\* 文件下载

\*

\* @param fileName 文件名称

\* @param res response

\* @return Boolean

\*/

public void download(String fileName, HttpServletResponse res) {

GetObjectArgs objectArgs = GetObjectArgs.builder().bucket(prop.getBucketName())

.object(fileName).build();

try (GetObjectResponse response = minioClient.getObject(objectArgs)) {

byte[] buf = new byte[1024];

int len;

try (FastByteArrayOutputStream os = new FastByteArrayOutputStream()) {

while ((len = response.read(buf)) != -1) {

os.write(buf, 0, len);

}

os.flush();

byte[] bytes = os.toByteArray();

res.setCharacterEncoding("utf-8");

// 设置强制下载不打开

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

FileAttach fileList = fileAttachService.getOne(new QueryWrapper<FileAttach>().lambda().eq(FileAttach::getUuid, fileName));

if (fileList != null) {

fileName = URLEncoder.encode(fileList.getFilename(), StandardCharsets.UTF\_8);

res.setContentType(fileList.getContentType());

}

res.addHeader("Content-Disposition", "attachment;filename=" + fileName);

try (ServletOutputStream stream = res.getOutputStream()) {

stream.write(bytes);

stream.flush();

}

}

} catch (Exception e) {

e.printStackTrace();

}

}

public InputStream getIs(String fileName) {

GetObjectArgs objectArgs = GetObjectArgs.builder().bucket(prop.getBucketName())

.object(fileName).build();

try (GetObjectResponse response = minioClient.getObject(objectArgs)) {

byte[] buf = new byte[1024];

int len;

try (FastByteArrayOutputStream os = new FastByteArrayOutputStream()) {

while ((len = response.read(buf)) != -1) {

os.write(buf, 0, len);

}

os.flush();

byte[] bytes = os.toByteArray();

return new ByteArrayInputStream(bytes);

}

} catch (Exception e) {

e.printStackTrace();

return null;

}

}

/\*\*

\* 查看文件对象

\*

\* @return 存储bucket内文件对象信息

\*/

public List<Item> listObjects() {

Iterable<Result<Item>> results = minioClient.listObjects(

ListObjectsArgs.builder().bucket(prop.getBucketName()).build());

List<Item> items = new ArrayList<>();

try {

for (Result<Item> result : results) {

items.add(result.get());

}

} catch (Exception e) {

e.printStackTrace();

return null;

}

return items;

}

/\*\*

\* 删除

\*

\* @param fileName

\* @return

\* @throws Exception

\*/

public boolean remove(String fileName) {

try {

minioClient.removeObject(RemoveObjectArgs.builder().bucket(prop.getBucketName()).object(fileName).build());

} catch (Exception e) {

return false;

}

return true;

}

}

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/3/6 10:28

\*/

@Component

public class RedisIndexIdUtil {

@Autowired

StringRedisTemplate stringRedisTemplate;

/\*\*

\* @return

\*/

public String egGenIndexId() {

String yyyyMMdd = DateUtil.format(new Date(), "yyyyMMdd");

String key = "eg\_genIndexId:" + yyyyMMdd;

Long id = stringRedisTemplate.opsForValue().increment(key);

stringRedisTemplate.expire(key, 1, TimeUnit.DAYS);

return yyyyMMdd + String.format("%06d", id);

}

public String sensorGenIndexId() {

String yyyyMMdd = DateUtil.format(new Date(), "yyyyMMdd");

String key = "sensor\_genIndexId:" + yyyyMMdd;

Long id = stringRedisTemplate.opsForValue().increment(key);

stringRedisTemplate.expire(key, 1, TimeUnit.DAYS);

return yyyyMMdd + String.format("%02d", id);

}

}

@Component

public class RedisUtil {

@Resource

private RedisTemplate<String, Object> redisTemplate;

/\*\*

\* 缓存基本的对象，Integer、String、实体类等

\*

\* @param key 缓存的键值

\* @param value 缓存的值

\*/

public <T> void setCacheObject(final String key, final T value) {

redisTemplate.opsForValue().set(key, value);

}

/\*\*

\* 缓存基本的对象，Integer、String、实体类等

\*

\* @param key 缓存的键值

\* @param value 缓存的值

\* @param timeout 时间

\* @param timeUnit 时间颗粒度

\*/

public <T> void setCacheObject(final String key, final T value, final Integer timeout, final TimeUnit timeUnit) {

redisTemplate.opsForValue().set(key, value, timeout, timeUnit);

}

/\*\*

\* 设置有效时间

\*

\* @param key Redis键

\* @param timeout 超时时间

\* @return true=设置成功；false=设置失败

\*/

public boolean expire(final String key, final long timeout) {

return expire(key, timeout, TimeUnit.SECONDS);

}

/\*\*

\* 设置有效时间

\*

\* @param key Redis键

\* @param timeout 超时时间

\* @param unit 时间单位

\* @return true=设置成功；false=设置失败

\*/

public Boolean expire(final String key, final long timeout, final TimeUnit unit) {

return redisTemplate.expire(key, timeout, unit);

}

// 根据指定的key 实现自增1

public Long increment(String key) {

ValueOperations<String, Object> valueOperations = redisTemplate.opsForValue();

return valueOperations.increment(key, 1L);

}

/\*\*

\* 获取有效时间

\*

\* @param key Redis键

\* @return 有效时间 {@link Long}

\*/

public Long getExpire(final String key) {

return redisTemplate.getExpire(key);

}

/\*\*

\* 判断 key是否存在

\*

\* @param key 键

\* @return true 存在 false不存在

\*/

public Boolean hasKey(String key) {

return redisTemplate.hasKey(key);

}

/\*\*

\* 获得缓存的基本对象。

\*

\* @param key 缓存键值

\* @return 缓存键值对应的数据

\*/

public Object getCacheObject(final String key) {

ValueOperations<String, Object> operation = redisTemplate.opsForValue();

return operation.get(key);

}

/\*\*

\* 删除单个对象

\*

\* @param key 键

\* @return {@link Boolean}

\*/

public Boolean deleteObject(final String key) {

return redisTemplate.delete(key);

}

/\*\*

\* 删除集合对象

\*

\* @param collection 多个对象

\* @return {@link Boolean}

\*/

public Boolean deleteObject(final Collection<String> collection) {

Long delete = redisTemplate.delete(collection);

return delete != null && delete > 0;

}

/\*\*

\* 缓存List数据

\*

\* @param key 缓存的键值

\* @param dataList 待缓存的List数据

\* @return 缓存的对象

\*/

public <T> long setCacheList(final String key, final List<T> dataList) {

Long count = redisTemplate.opsForList().rightPushAll(key, dataList);

return count == null ? 0 : count;

}

/\*\*

\* 获得缓存的list对象

\*

\* @param key 缓存的键值

\* @return 缓存键值对应的数据

\*/

public List<Object> getCacheList(final String key) {

return redisTemplate.opsForList().range(key, 0, -1);

}

/\*\*

\* 缓存Set

\*

\* @param key 缓存键值

\* @param dataSet 缓存的数据

\* @return 缓存数据的对象

\*/

public BoundSetOperations<String, Object> setCacheSet(final String key, final Set<Object> dataSet) {

BoundSetOperations<String, Object> setOperation = redisTemplate.boundSetOps(key);

for (Object o : dataSet) {

setOperation.add(o);

}

return setOperation;

}

/\*\*

\* 获得缓存的set

\*

\* @param key 键

\* @return {@link Set<Object>}

\*/

public Set<Object> getCacheSet(final String key) {

return redisTemplate.opsForSet().members(key);

}

/\*\*

\* 缓存Map

\*

\* @param key 键

\* @param dataMap 键值对

\*/

public void setCacheMap(final String key, final Map<String, Object> dataMap) {

if (dataMap != null) {

redisTemplate.opsForHash().putAll(key, dataMap);

}

}

/\*\*

\* 获得缓存的Map

\*

\* @param key 键

\* @return {@link Map}

\*/

public Map<Object, Object> getCacheMap(final String key) {

return redisTemplate.opsForHash().entries(key);

}

/\*\*

\* 往Hash中存入数据

\*

\* @param key Redis键

\* @param hKey Hash键

\* @param value 值

\*/

public <T> void setCacheMapValue(final String key, final String hKey, final T value) {

redisTemplate.opsForHash().put(key, hKey, value);

}

/\*\*

\* 获取Hash中的数据

\*

\* @param key Redis键

\* @param hKey Hash键

\* @return Hash中的对象

\*/

public <T> T getCacheMapValue(final String key, final String hKey) {

HashOperations<String, String, T> opsForHash = redisTemplate.opsForHash();

return opsForHash.get(key, hKey);

}

/\*\*

\* 获取多个Hash中的数据

\*

\* @param key Redis键

\* @param hKeys Hash键集合

\* @return Hash对象集合

\*/

public List<Object> getMultiCacheMapValue(final String key, final Collection<Object> hKeys) {

return redisTemplate.opsForHash().multiGet(key, hKeys);

}

/\*\*

\* 删除Hash中的某条数据

\*

\* @param key Redis键

\* @param hKey Hash键

\* @return 是否成功

\*/

public boolean deleteCacheMapValue(final String key, final String hKey) {

return redisTemplate.opsForHash().delete(key, hKey) > 0;

}

/\*\*

\* 获得缓存的基本对象列表

\*

\* @param pattern 字符串前缀

\* @return 对象列表

\*/

public Collection<String> keys(final String pattern) {

return redisTemplate.keys(pattern);

}

}

public class RestTemplateUtil {

private RestTemplateUtil() {

}

/\*\*

\* get请求

\*

\* @param requestUrl url,参数拼接再url上

\* @param requestEntity requestEntity

\* @param responseType 响应体

\* @return 返回泛型

\*/

public static <T> T getRequest(String requestUrl, HttpEntity<?> requestEntity, Class<T> responseType) {

RestTemplate restTemplate = new RestTemplate();

ResponseEntity<T> responseEntity = restTemplate.exchange(requestUrl, HttpMethod.GET, requestEntity, responseType);

return responseEntity.getBody();

}

/\*\*

\* post请求

\*

\* @param requestUrl 请求url

\* @param requestBody 请求的参数

\* @param responseType 响应体

\* @return 泛型

\*/

public static <T> T postRequest(String requestUrl, Map<String, Object> requestBody, Class<T> responseType) {

RestTemplate restTemplate = new RestTemplate();

HttpHeaders headers = new HttpHeaders();

headers.setContentType(MediaType.APPLICATION\_JSON);

HttpEntity<Map<String, Object>> requestEntity = new HttpEntity<>(requestBody, headers);

ResponseEntity<T> responseEntity = restTemplate.exchange(requestUrl, HttpMethod.POST, requestEntity, responseType);

if (responseEntity.getStatusCode() != HttpStatus.OK) {

throw new ServiceException(requestUrl + "接口的post请求失败");

}

return responseEntity.getBody();

}

}

/\*\*

\* <h2>StreamUtil</h2>

\* <p>函数式接口功能补充</p>

\*

\* @author Daizc

\* @date 2019/12/11

\*/

public class StreamUtil {

/\*\*

\* <h3>distinctByKey</h3>

\* <p>可用于在filter中根据属性过滤</p>

\*

\* @param function 接受一个 接受 类型A 返回 类型B 的函数式接口

\* @return Predicate 返回一个 接受 类型A 返回 Boolean型 的函数式接口

\*/

public static <T> Predicate<T> distinctByKey(Function<? super T, ?> function) {

Map<Object, Boolean> filterMap = new ConcurrentHashMap<>();

return t -> filterMap.putIfAbsent(function.apply(t), Boolean.TRUE) == null;

}

// public static void sortByKey(){

//

//

// return new Comparator() {

// @Override

// public int compare(Object o1, Object o2) {

// return 0;

// }

//

// @Override

// public boolean equals(Object obj) {

// return false;

// }

// };

// }

/\*\*

\* <h3>safeGet</h3>

\* <p>自动处理空指针异常</p>

\*

\* @param supplier 不接受参数 直接返回 类型R 的函数式接口

\* @return R 函数式接口的处理结果

\*/

public static <R> R safeGet(Supplier<R> supplier) {

R get = null;

try {

get = supplier.get();

} catch (NullPointerException ignored) {

}

return get;

}

public static <T, U extends Comparable<? super U>> Comparator<T> comparingEx(

Function<? super T, ? extends U> keyExtractor) {

Objects.requireNonNull(keyExtractor);

return (Comparator<T> & Serializable)

(c1, c2) -> {

U c1result;

U c2result;

try {

c1result = keyExtractor.apply(c1);

} catch (NullPointerException npe) {

return Integer.MAX\_VALUE;

}

try {

c2result = keyExtractor.apply(c2);

} catch (NullPointerException npe) {

return Integer.MIN\_VALUE;

}

return c1result.compareTo(c2result);

};

}

/\*\*

\* <h3>entry</h3>

\* <p>创建一个<b>Entry<KEY,VALUE>实体</b>的函数</p>

\* <p>可用于在map中将实体转为Entry</p>

\*

\* @param keyFunction 接受一个 接受 类型A 返回 类型B 的函数式接口

\* @param valueFunction 接受一个 接受 类型A 返回 类型B 的函数式接口

\* @return Function<T, Entry < KEY, VALUE>>

\*/

public static <KEY, VALUE, T> Function<T, Entry<KEY, VALUE>> entry(Function<T, KEY> keyFunction, Function<T, VALUE> valueFunction) {

return t -> new Entry<>(keyFunction.apply(t), valueFunction.apply(t));

}

}

@Component

public class WechatUtil {

private static final Logger log = LoggerFactory.getLogger(WechatUtil.class);

@Resource

private RedisUtil redisUtil;

@Value("${wechat.base-url}")

private String baseUrl;

@Value("${wechat.quota}")

private String quota;

@Value("${wechat.login-api}")

private String loginApi;

@Value("${wechat.token-api}")

private String tokenApi;

@Value("${wechat.get-phone-api}")

private String getPhoneApi;

@Value("${wechat.app-id}")

private String appId;

@Value("${wechat.app-secret}")

private String appSecret;

private static final String ERR\_CODE\_STRING = "errcode";

private static final String ERR\_MSG\_STRING = "errmsg";

/\*\*

\* <p>获取接口访问的access\_token</p>

\* <p>查看缓存是否存在，如果存在则返回数据；如果过期则重新请求token并缓存返回token</p>

\* <h2>说明：</h2>

\* <li>获取token接口返回的数据类型不一样，错误的时候返回的 格式是{@link BaseResponse}对象，正确时，返回的格式不包含{@link BaseResponse} 属性</li>

\*

\* @return access\_token

\*/

private String getAccessToken() {

String grantType = "client\_credential";

String url = this.baseUrl + this.tokenApi + "?grant\_type=%s&appid=%s&secret=%s";

String accessTokenString = "access\_token";

String tokenKey = SystemConstant.REDIS\_KEY\_PREFIX + accessTokenString;

if (Boolean.TRUE.equals(redisUtil.hasKey(tokenKey))) {

return (String) redisUtil.getCacheObject(tokenKey);

} else {

String requestUrl = String.format(url, grantType, this.appId, this.appSecret);

log.info("请求获取微信的token接口：{}" , requestUrl);

Map body = RestTemplateUtil.getRequest(requestUrl, null, Map.class);

// 返回有内容且没有错误码

if (body != null) {

// 请求成功

if (!body.containsKey(ERR\_CODE\_STRING)) {

// 默认设置过期时间是 7000

redisUtil.setCacheObject(tokenKey, (String) body.get(accessTokenString), 7000, TimeUnit.SECONDS);

return (String) body.get(accessTokenString);

} else {

int errCode = Integer.parseInt(String.valueOf(body.get(ERR\_CODE\_STRING)));

switch (errCode) {

case 40013 ->

throw new ServiceException("不合法的 AppID ，请开发者检查 AppID 的正确性，避免异常字符，注意大小写");

case 40125 ->

throw new ServiceException("不合法的 appsecret ，请开发者检查 appsecret 的正确性，避免异常字符，注意大小写");

case -1 -> throw new ServiceException("系统繁忙，此时请开发者稍候再试");

default -> throw new ServiceException(String.valueOf(body.get(ERR\_MSG\_STRING)));

}

}

} else {

throw new ServiceException("微信服务端没有响应，请稍后或者联系微信客服！");

}

}

}

/\*\*

\* 重置API调用次数

\*/

public void clearQuota() {

String url = this.baseUrl + this.quota + "?access\_token=" + this.getAccessToken();

log.info("调用接口次数清零：{}" , url);

Map<String, Object> requestBody = new LinkedHashMap<>();

requestBody.put("appid" , this.appId);

BaseResponse body = RestTemplateUtil.postRequest(url, requestBody, BaseResponse.class);

if (body != null && body.getErrcode() == 0) {

log.info("重置API调用次数成功");

} else {

log.error("重置API调用次数失败，{}" , body == null ? "" : body.toString());

}

}

/\*\*

\* <p>获取登录信息</p>

\* <p>注意： 登录失败信息，由您来定义给用户展示</p>

\*

\* @param jsCode 登录时获取的 code，可通过wx.login获取

\*/

public LoginResponse getLoginInfo(String jsCode) {

String grantType = "authorization\_code";

String url = String.format("%s%s?appid=%s&secret=%s&js\_code=%s&grant\_type=%s" , this.baseUrl, this.loginApi, this.appId, this.appSecret, jsCode, grantType);

log.info("登录微信：{}" , url);

return RestTemplateUtil.getRequest(url, null, LoginResponse.class);

}

/\*\*

\* <p>获取手机号</p>

\* <p>注意： 这里没有对返回的数据做处理，请自行判断</p>

\* <p>注意： 请自行判断access\_token 缓存失效的问题（这个问题是【人为】操作导致的token缓存不是最新的），需要您检测出错误码为40001，手动刷新缓存。

\* 可以参考 {@link LoginController} 的 wechatLogin 方法

\* </p>

\*

\* @param code <a href="https://developers.weixin.qq.com/miniprogram/dev/framework/open-ability/getPhoneNumber.html">手机号获取凭证</a>

\* @return {@link PhoneInfoResponse}

\*/

public PhoneInfoResponse getPhoneNumber(String code) {

String url = this.baseUrl + this.getPhoneApi + "?access\_token=" + this.getAccessToken();

Map<String, Object> requestBody = new LinkedHashMap<>();

requestBody.put("code" , code);

return RestTemplateUtil.postRequest(url, requestBody, PhoneInfoResponse.class);

}

}