*package com.maxnerva.cloudmes.tpm.util*;  
  
  
*import cn.hutool.core.bean.BeanUtil*;  
*import org.apache.commons.lang.StringUtils*;  
*import org.apache.commons.lang.reflect.FieldUtils*;  
  
*import java.util.ArrayList*;  
*import java.util.List*;  
  
*/\*\*  
 \** ***@author*** *MFQ  
 \** ***@date*** *2023/9/22 上午 09:46  
 \*/  
  
public class TreeUtil* {  
 */\* 把列表转换为树结构  
 \*  
 \* @param originalList 原始list数据  
 \* @param idFieldName 作为唯一标示的字段名称  
 \* @param pidFieldName 父节点标识字段名  
 \* @param childrenFieldName 子节点（列表）标识字段名  
 \* @return 树结构列表  
 \*/  
 public static* <*T*> *List*<*T*> list2TreeList(*List*<*T*> *originalList*, *String idFieldName*, *String pidFieldName*,  
 *String childrenFieldName*) {  
 *// 获取根节点，即找出父节点为空的对象  
 List*<*T*> rootNodeList = *new* ArrayList<>();  
 *for* (*T* t : *originalList*) {  
 *String* parentId = *null*;  
  
 parentId = *BeanUtil*.*getProperty*(t, *pidFieldName*);  
  
 *if* (*StringUtils*.*isBlank*(parentId)) {  
 rootNodeList.add(0, t);  
 }  
 }  
  
 *// 将根节点从原始list移除，减少下次处理数据  
 originalList*.removeAll(rootNodeList);  
  
 *// 递归封装树  
 try* {  
 *packTree*(rootNodeList, *originalList*, *idFieldName*, *pidFieldName*, *childrenFieldName*);  
 } *catch* (*Exception e*) {  
 *e*.printStackTrace();  
 }  
  
 *return* rootNodeList;  
 }  
  
 */\*\*  
 \* 封装树（向下递归）  
 \*  
 \** ***@param* parentNodeList** *要封装为树的父节点对象集合  
 \** ***@param* originalList** *原始list数据  
 \** ***@param* keyName** *作为唯一标示的字段名称  
 \** ***@param* pidFieldName** *父节点标识字段名  
 \** ***@param* childrenFieldName** *子节点（列表）标识字段名  
 \*/  
 private static* <*T*> *void* packTree(*List*<*T*> *parentNodeList*, *List*<*T*> *originalList*, *String keyName*,  
 *String pidFieldName*, *String childrenFieldName*) *throws Exception* {  
 *for* (*T* parentNode : *parentNodeList*) {  
 *// 找到当前父节点的子节点列表  
 List*<*T*> children = *packChildren*(parentNode, *originalList*, *keyName*, *pidFieldName*, *childrenFieldName*);  
 *if* (children.isEmpty()) {  
 *continue*;  
 }  
  
 *// 将当前父节点的子节点从原始list移除，减少下次处理数据  
 originalList*.removeAll(children);  
  
 *// 开始下次递归  
 packTree*(children, *originalList*, *keyName*, *pidFieldName*, *childrenFieldName*);  
 }  
 }  
  
 */\*\*  
 \* 封装子对象  
 \*  
 \** ***@param* parentNode** *父节点对象  
 \** ***@param* originalList** *原始list数据  
 \** ***@param* keyName** *作为唯一标示的字段名称  
 \** ***@param* pidFieldName** *父节点标识字段名  
 \** ***@param* childrenFieldName** *子节点（列表）标识字段名  
 \*/  
 private static* <*T*> *List*<*T*> packChildren(*T parentNode*, *List*<*T*> *originalList*, *String keyName*, *String pidFieldName*,  
 *String childrenFieldName*) *throws Exception* {  
 *// 找到当前父节点下的子节点列表  
 List*<*T*> childNodeList = *new* ArrayList<>();  
 *String* parentId = *BeanUtil*.*getProperty*(*parentNode*, *keyName*);  
 *for* (*T* t : *originalList*) {  
 *String* childNodeParentId = *BeanUtil*.*getProperty*(t, *pidFieldName*);  
 *if* (parentId.equals(childNodeParentId)) {  
 childNodeList.add(t);  
 }  
 }  
  
 *// 将当前父节点下的子节点列表写入到当前父节点下（给子节点列表字段赋值）  
 if* (!childNodeList.isEmpty()) {  
 *FieldUtils*.*writeDeclaredField*(*parentNode*, *childrenFieldName*, childNodeList, *true*);  
 }  
  
 *return* childNodeList;  
 }  
}

創建Excel

*package com.maxnerva.cloudmes.tpm.util*;  
  
*import cn.hutool.core.util.ObjectUtil*;  
*import com.maxnerva.cloudmes.common.exception.CloudmesException*;  
*import com.maxnerva.cloudmes.common.utils.MessageUtils*;  
*import com.maxnerva.cloudmes.tpm.annotation.ExcelColAnnotation*;  
*import com.maxnerva.cloudmes.tpm.annotation.ExcelTitleAnnotation*;  
*import com.maxnerva.cloudmes.tpm.enums.SpmMouldResultCode*;  
*import org.apache.commons.logging.Log*;  
*import org.apache.commons.logging.*LogFactory;  
*import org.apache.poi.hssf.usermodel.HSSFDateUtil*;  
*import org.apache.poi.hssf.usermodel.HSSFWorkbook*;  
*import org.apache.poi.hssf.util.HSSFColor*;  
*import org.apache.poi.ss.usermodel.*\*;  
*import org.apache.poi.ss.util.CellRangeAddress*;  
*import org.apache.poi.xssf.usermodel.XSSFWorkbook*;  
*import org.springframework.util.*StringUtils;  
*import org.springframework.web.multipart.MultipartFile*;  
  
*import java.io.File*;  
*import java.io.FileOutputStream*;  
*import java.io.IOException*;  
*import java.io.*InputStream;  
*import java.lang.annotation.Annotation*;  
*import java.lang.reflect.Field*;  
*import java.lang.reflect.InvocationTargetException*;  
*import java.lang.reflect.Method*;  
*import java.text.SimpleDateFormat*;  
*import java.util.*\*;  
  
*public class ExcelUtils*<*T*> {  
  
 *private static final Log* log = LogFactory.*getLog*(*ExcelUtils*.*class*);  
   
 */\*\*  
 \* 行高  
 \*/  
 private int* rowHeight = 400;  
 */\*\*  
 \* 列宽  
 \*/  
 private int* colWidth = 8500;  
 */\*\*  
 \* 起始位置  
 \*/  
 private int* rowIndex = 0;  
 */\*\*  
 \* 默认标题  
 \*/  
 private String* title = "defaultExcel";  
   
 *private Workbook* workbook;  
 *private Sheet* sheet;  
 */\*\*  
 \* 公共列样式  
 \*/  
 private CellStyle* cellStyle;  
 */\*\*  
 \* 操作的实体类  
 \*/  
 private T* obj;  
 */\*\*  
 \* 列属性集合  
 \*/  
 private List*<*Map*<*String*,*Object*>> colList = *new* ArrayList<>();  
 */\*\*  
 \* 列样式集合  
 \*/  
 private List*<*CellStyle*> styleList = *new* ArrayList<>();  
   
 *private ClassUtils* util = *new* ClassUtils();  
   
 *public* ExcelUtils(*T obj* ) {  
 *this*.obj = *obj*;  
 initWorkbook();  
 }  
   
 *public* ExcelUtils(*T obj*, *int rowHeight*, *int colWidth*, *int rowIndex*, *String title*) {  
 *this*(*obj*);  
 *this*.rowHeight = *rowHeight*;  
 *this*.colWidth = *colWidth*;  
 *this*.rowIndex = *rowIndex*;  
 *this*.title = *title*;  
 }  
  
 */\*\*  
 \* 默认文档设置文档  
 \*/  
 private void* initWorkbook() {  
 *if* ( *ObjectUtil*.*isEmpty*(*this*.obj) ) {  
 *throw new* CloudmesException(*MessageUtils*.*get*(*SpmMouldResultCode*.EXPORT\_ENTITY\_NOT\_SPECIFIED.getLocalCode()));  
 }  
 util.parseBookAnnotation();  
 workbook = *new* XSSFWorkbook();  
 sheet = workbook.createSheet(*this*.title); *// 创建工作页* sheet.setDefaultColumnWidth(colWidth); *// 设置默认列宽* cellStyle = createCellStyle();  
 cellStyle.setFont(createFont(*null*, (*short*)0, (*short*)0));  
 titleSetting();  
 }  
   
 */\*\*  
 \* 标题、列名相关设置  
 \*/  
 private void* titleSetting() {  
 *Row* topRow = createRow((*short*)600);  
 mergedRegion(0, 0, 0, *this*.colList.size()-1); *// 合并标题行  
 Cell* cell = createCell(topRow,0,cellStyle);  
 cell.setCellValue(*this*.title);  
 *Row* textRow = createRow((*short*)0);  
 *for* ( *int* i = 0; i < colList.size(); i++ ) {  
 *Map*<*String*,*Object*> fieldMap = colList.get(i);   
 sheet.setColumnWidth(i, *Integer*.*parseInt*(fieldMap.get("width").toString()));  
 *Cell* cell1 = createCell(textRow,i,cellStyle);  
 cell1.setCellValue(fieldMap.get("text").toString());  
 addColStlye(*null*, (*short*)0, *Short*.*parseShort*(fieldMap.get("color").toString()));  
 }  
 }  
   
 */\*\*  
 \* 创建字体对象  
 \** ***@param* fontName** *\* 字体库名称  
 \** ***@param* fontSize** *\* 字体大小-传0默认14  
 \** ***@param* color** *\* 字体颜色参考{****@link*** *HSSFColor.HSSFColorPredefined}  
 \** ***@return*** *\*/  
 private Font* createFont(*String fontName*, *short fontSize*,*short color*) {  
 *Font* font = *this*.workbook.createFont();  
 *if* ( StringUtils.*isEmpty*(*fontName*) ) {  
 *fontName* = "宋体";  
 }  
 *if* ( *fontSize* == 0 ) {  
 *fontSize* = (*short*)14;  
 }  
 *if* ( *color* == 0 ) {  
 *color* = *HSSFColor*.*HSSFColorPredefined*.BLACK.getIndex();  
 }  
 font.setFontName(*fontName*); *//设置为宋体字* font.setFontHeightInPoints(*fontSize*); *//设置字体大小* font.setColor(*color*);  
 *return* font;  
 }  
   
   
   
 */\*\*  
 \* 创建列样式  
 \** ***@param* alignments** *\** ***@return*** *\*/  
 private CellStyle* createCellStyle(*short* ...*alignments*) {  
 *CellStyle* style = *this*.workbook.createCellStyle();  
 *short* alignmentNum = *HorizontalAlignment*.CENTER.getCode(),verticalNum = *VerticalAlignment*.CENTER.getCode();  
 *if* ( *alignments*.length > 0 ) {  
 alignmentNum = *alignments*[0];  
 *if* ( *alignments*.length > 1 ) {  
 verticalNum = *alignments*[1];  
 }  
 }  
 *//水平居中* style.setAlignment(*HorizontalAlignment*.*forInt*(alignmentNum));  
 *//垂直居中* style.setVerticalAlignment(*VerticalAlignment*.*forInt*(verticalNum));  
 *return* style;  
 }  
   
 */\*\*  
 \* 合并行、列  
 \** ***@param* firstRow** *\** ***@param* lastRow** *\** ***@param* firstCol** *\** ***@param* lastCol** *\*/  
 private void* mergedRegion(*int firstRow*, *int lastRow*, *int firstCol*, *int lastCol*) {  
 *CellRangeAddress* region = *new* CellRangeAddress(*firstRow*,*lastRow*,*firstCol*,*lastCol*); *// 合并行* sheet.addMergedRegion(region);  
 }  
   
 */\*\*  
 \* 创建单元格  
 \** ***@param* row** *\** ***@param* index** *\** ***@param* style** *\** ***@return*** *\*/  
 private Cell* createCell ( *Row row*,*int index*,*CellStyle style* ) {  
 *Cell* cell = *row*.createCell(*index*);  
 *if* ( *ObjectUtil*.*isNotEmpty*(*style*) ) {  
 cell.setCellStyle(*style*);  
 }  
 *return* cell;  
 }  
   
 */\*\*  
 \* 创建行  
 \** ***@return*** *\*/  
 private Row* createRow ( *short rowHeight* ) {  
 *Row* row = sheet.createRow(*this*.rowIndex);  
 *if* ( *rowHeight* == 0 ) {  
 *rowHeight* = (*short*)*this*.rowHeight;  
 }  
 row.setHeight((*short*) (2\**rowHeight*));  
 *this*.rowIndex ++;  
 *return* row;  
 }  
   
 */\*\*  
 \* 创建每一列的样式  
 \** ***@param* fontName** *\** ***@param* fontSize** *\** ***@param* color** *\** ***@param* alignments** *\*/  
 private void* addColStlye(*String fontName*, *short fontSize*,*short color*,*short* ...*alignments*) {  
 *Font* ft = createFont(*fontName*, *fontSize*, *color*);  
 *CellStyle* style = createCellStyle(*alignments*);  
 style.setFont(ft);  
 styleList.add(style);  
 }  
   
 */\*\*  
 \* 生成Excel表  
 \** ***@param* list** *\** ***@throws*** *SecurityException   
 \** ***@throws*** *NoSuchMethodException   
 \** ***@throws*** *InvocationTargetException   
 \** ***@throws*** *IllegalArgumentException   
 \** ***@throws*** *IllegalAccessException   
 \*/  
 public void* createExcel(*List*<*T*> *list*,*String parentPath*) *throws IOException*, *NoSuchMethodException*, *SecurityException*, *IllegalAccessException*, *IllegalArgumentException*, *InvocationTargetException* {  
 *for* ( *T* temp : *list* ) {  
 *Row* row = createRow((*short*)0);  
 *for* (*int* i = 0; i < colList.size(); i ++ ) {  
 *Map*<*String*,*Object*> cl = colList.get(i);  
 *Cell* cell = createCell(row,i,styleList.get(i));  
 *String* methodName = cl.get("methodName").toString();  
 *Object* o = util.valueToGet(temp, methodName);  
 *String* value = *ObjectUtil*.*isEmpty*(o) ? "" : o.toString();  
 cell.setCellValue(value);  
 }  
 }  
 *File* file = *new* File(*parentPath*);  
 *if* (!file.exists()){  
 file.createNewFile();  
 }  
 *FileOutputStream* outputStream = *new* FileOutputStream(file);  
 workbook.write(outputStream);  
 outputStream.close();  
 }  
  
 */\*\*  
 \* 生成Excel表  
 \** ***@param* list** *\** ***@throws*** *SecurityException  
 \** ***@throws*** *NoSuchMethodException  
 \** ***@throws*** *InvocationTargetException  
 \** ***@throws*** *IllegalArgumentException  
 \** ***@throws*** *IllegalAccessException  
 \*/  
 public Workbook* returnWorkBook(*List*<*T*> *list*) *throws NoSuchMethodException*, *SecurityException*, *IllegalAccessException*, *IllegalArgumentException*, *InvocationTargetException* {  
 *for* ( *T* temp : *list* ) {  
 *Row* row = createRow((*short*)0);  
 *for* (*int* i = 0; i < colList.size(); i ++ ) {  
 *Map*<*String*,*Object*> cl = colList.get(i);  
 *Cell* cell = createCell(row,i,styleList.get(i));  
 *String* methodName = cl.get("methodName").toString();  
 *Object* o = util.valueToGet(temp, methodName);  
 *String* value = *ObjectUtil*.*isEmpty*(o) ? "" : o.toString();  
 cell.setCellValue(value);  
 }  
 }  
 *return this*.workbook;  
 }  
  
 *class ClassUtils*{  
  
 *private* ClassUtils() {}  
 */\*\*  
 \* 反射获取value  
 \** ***@param* object** *\** ***@return*** *\** ***@throws*** *SecurityException   
 \** ***@throws*** *NoSuchMethodException   
 \** ***@throws*** *InvocationTargetException   
 \** ***@throws*** *IllegalArgumentException   
 \** ***@throws*** *IllegalAccessException   
 \*/  
 private Object* valueToGet( *Object object*, *String methodName* ) *throws NoSuchMethodException*, *SecurityException*, *IllegalAccessException*, *IllegalArgumentException*, *InvocationTargetException* {  
 *Method* method = *object*.getClass().getDeclaredMethod(*methodName*);  
 *return* method.invoke(*object*);  
 }  
   
 */\*\*  
 \* 首字母大写  
 \** ***@param* fieldName** *\** ***@return*** *\*/  
 public String* convertMethodName( *String fieldName* ) {  
 *String* newField = *fieldName*.substring(1, *fieldName*.length());  
 *return fieldName*.substring(0,1).toUpperCase()+newField;  
 }  
   
 */\*\*  
 \* 通过注解获取导出sheet相关注解属性  
 \*/  
 public void* parseBookAnnotation( ) {  
 *Annotation*[] ans = *ExcelUtils*.*this*.obj.getClass().getAnnotations();  
 *for* ( *Annotation* temp : ans ) {  
 *if* ( temp *instanceof ExcelTitleAnnotation*) {  
 *ExcelUtils*.*this*.title = ((*ExcelTitleAnnotation*) temp).title();  
 parseFielAnnotation( );  
 }  
 }  
 }  
   
 */\*\*  
 \* 通过注解获取列相关注解属性  
 \*/  
 public void* parseFielAnnotation( ) {  
 *Field* [] fiels = *ExcelUtils*.*this*.obj.getClass().getDeclaredFields();  
 *for* ( *Field* temp : fiels ) {  
 *Annotation*[] ans = temp.getAnnotations();  
 *for* ( *Annotation* tempAn : ans ) {  
 *if* ( tempAn *instanceof ExcelColAnnotation*) {  
 *ExcelColAnnotation* col = ((*ExcelColAnnotation*) tempAn);  
 *int* ignore = col.ignore();  
 *if* ( ignore == 0 ) {  
 *Map*<*String*,*Object*> fieldMap = *new* HashMap<>();  
 fieldMap.put("width", col.colWidth());  
 fieldMap.put("color", col.color());  
 fieldMap.put("text", col.text());  
 fieldMap.put("methodName", "get"+convertMethodName(temp.getName()));  
 fieldMap.put("sort",col.sort());  
 *ExcelUtils*.*this*.colList.add(fieldMap);  
 }  
 }  
 }  
 }  
 *Collections*.*sort*(colList, *new Comparator*<*Map*<*String*, *Object*>>() {  
 *@Override  
 public int* compare(*Map*<*String*, *Object*> *o1*, *Map*<*String*, *Object*> *o2*) {  
 *return o1*.get("sort").toString().compareTo(*o2*.get("sort").toString());  
 }  
 });  
 }  
 }  
  
 *public static Workbook* getWorkbook(InputStream *in*, *MultipartFile file*) {  
 *Workbook* wb = *null*;  
 *try* {  
 *if* (*file*.getOriginalFilename().endsWith("xls")) {  
 wb = *new* HSSFWorkbook(*in*);  
 } *else if* (*file*.getOriginalFilename().endsWith("xlsx")) {  
 wb = *new* XSSFWorkbook(*in*);  
 }  
 }*catch* (*Exception e*){  
 log.error(*e*.getMessage());  
 }  
 *return* wb;  
 }  
  
 *public static String* checkCellValue(*Cell cell*){  
 *SimpleDateFormat* sdf = *new* SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  
 *String* value = "";  
 *if* (*ObjectUtil*.*isEmpty*(*cell*)){  
 *return* value;  
 }  
 *CellType* cellType = *cell*.getCellTypeEnum();  
 *switch* (cellType){  
 *case* \_NONE:  
 *case* BLANK:  
 value = "";  
 *break*;  
 *case* STRING:  
 value = *cell*.getStringCellValue();  
 *break*;  
 *case* BOOLEAN:  
 value = *cell*.getBooleanCellValue()+"";  
 *break*;  
 *case* NUMERIC:  
 *boolean* isDate = *HSSFDateUtil*.*isCellDateFormatted*(*cell*);  
 *if* (isDate){  
 value = sdf.format(*cell*.getDateCellValue());  
 }*else* {  
 value = *String*.*valueOf*((*int*) *cell*.getNumericCellValue());  
 }  
 *break*;  
 }  
 *return* value;  
 }  
}

使用:

*Workbook* workbook = excelUtils.returnWorkBook(dtos);  
*//输出文件  
response*.setHeader("content-Type", "application/vnd.ms-excel;charset=utf-8");  
*response*.setContentType("application/vnd.ms-excel;charset=utf-8");  
*response*.setHeader("Content-Disposition",  
 "attachment;filename=" + *URLEncoder*.*encode*("出库明细报表.xlsx", "UTF-8"));  
*response*.flushBuffer();  
workbook.write(*response*.getOutputStream());  
workbook.close();