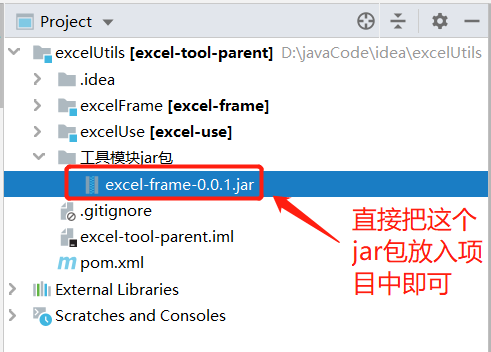
# 项目简介及使用教程

1. 下载地址：

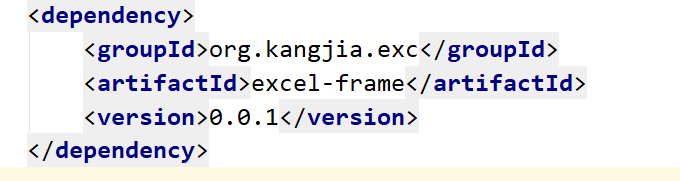
https://gitee.com/renjiale/excel-utils

二、导入方式：

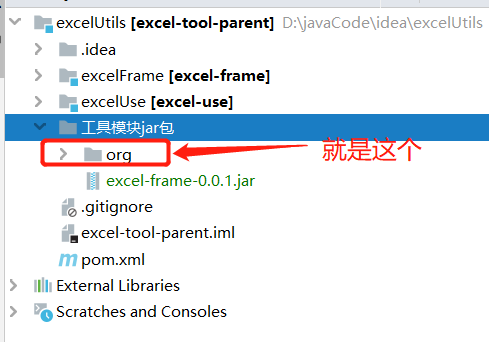
<1.> 直接导入jar包



<2.> maven依赖导入



哦！记得把这个复制到你的maven仓库中哦。因为这个项目没上传到网上，只能 作为本地的自定义jar包使用。

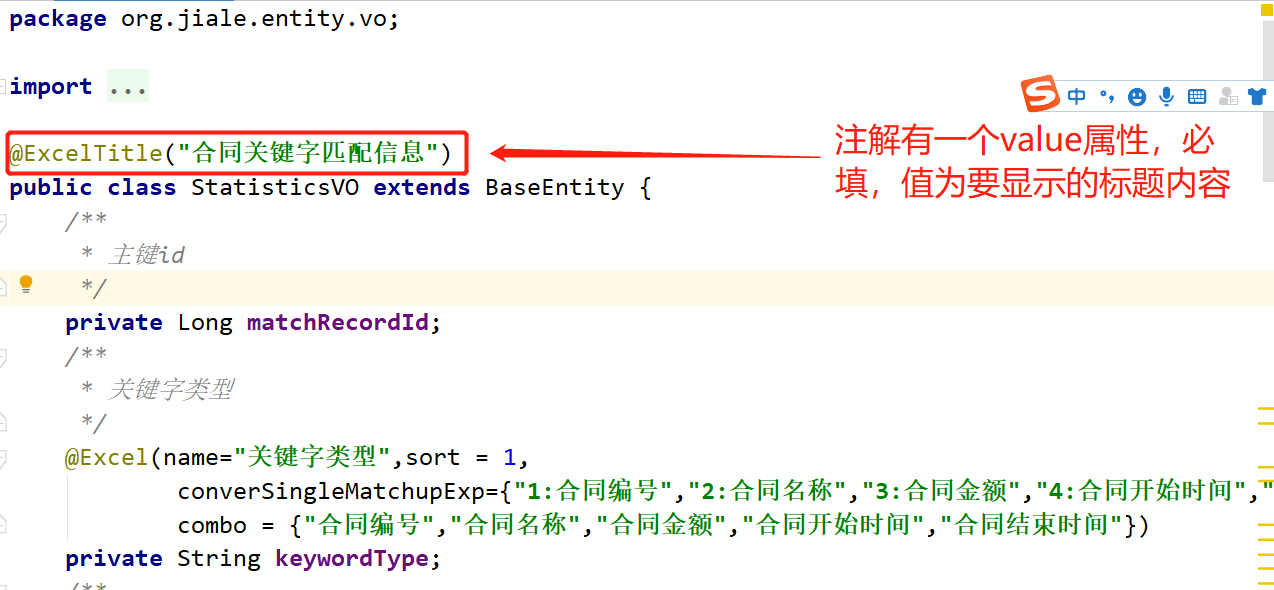


1. 使用方式：

* 导出

导出时需要使用会用到两个注解@Excel和@ExcelTitle

@ExcelTitle注解只可以加在类上，它的作用是添加总标题，如





如果导出的文件中不需要加这个总标题，也可以不用此注解。

@Excel注解只可以加在属性上，给我们需要导出的属性加上这个注解就可以，此注解里的参数有：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 默认值 | 描述 |
| value | String | 属性名 | 导出到Excel中的列名，不填写取类中的属性名作为列名 |
| name | String | 属性名 | 导出到Excel中的列名，不填写取类中的属性名作为列名 |
| sort | int | Integer.MAX\_VALUE | 导出到Excel中的排序，建议赋值，不赋值默认按属性在类中的顺序排序 |
| dateFormat | String |  | 日期格式，只能是Date类型属性上使用，如：yyyy-MM-dd HH:mm:ss |
| converSingleMatchupExp | String[] |  | 对应代理表达式，有时候我们喜欢用不同的数字（key）代替不同的状态（value），这个参数可以配置它们的对应关系。如：{“0:男”,”1:女”,”2:未知”} |
| cellType | ColumnType | ColumnType.STRING | 导出类型，值为枚举型。可选值为：ColumnType.STRING，ColumnType.NUMERIC  注：选择数字类型时加前后缀无效 |
| combo | String[] |  | 设置该列的固定值，有时候我们只是导出一份模板文件，单元格内只能填写固定的值 |
| titleHeight | short | 25 | 导出时在Excel中每个标题单元格的高度，单位为字符 |
| height | short | 20 | 导出时在Excel中每个数据单元格的高度，单位为字符 |
| width | double | 16 | 导出时在Excel中每个单元格的宽度，单位为字符 |
| prefix | String |  | 文字前缀，加前缀后此单元格格式变为字符串型 |
| suffix | String |  | 文字后缀，加后缀后此单元格格式变为字符串型 |
| defaultValue | String |  | 导出时此值为空时，此列会自动赋值默认值 |
| isExportDate | boolean | true | 是否导出数据，有时候我们需要此列保持为空，希望用户填写此项，就需要赋值false |
| hasChildren | boolean | false | 是否拥有子级，这个参数对我们生成复杂标题非常重要，当我们属性是自定义的类，而此类中也有属性加了@Excel注解，就需要赋值true |
| isStatistics | boolean | false | 是否自动统计数据，在最后追加一行统计数据。注：只能number类型才能统计 |

在我们的导出的接口中先查出要导出的数据，然后调用ExcelUtils.exportExcel(),此方 法的参数有：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 是否必须 | 描述 |
| clazz | Class<T> | true | 导出的数据类型的类对象 |
| list | List<T> | false | 导出的数据，为空时表示导出模板 |
| sheetName | String | true | sheetName |
| fileName | String | false | 文件名称，为空时文件名称和sheetName一样 |
| response | HttpServletResponse | true | http响应 |
| customize | Customize | false | 自定义表格样式，这是一个接口，如果我们觉得框架中提供的默认样式不喜欢，想要修改导出文件中表格的样式，可以继承CustomizeStyles类，重接其三个方法（createTitleStyles:定义标题样式，createDateStyles:定义数据样式，createTotalStyles:定义统计行样式，）。或者继承TitleStyles类，重写其方法定义标题样式；或者继承DateStyles类，重写其方法定义数据样式；或者继承TotalStyles类，重写其方法定义统计行样式 |

* 导入

导入时要调用ExcelUtils.importExcel()，此方法的参数有：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 是否必须 | 描述 |
| is | InputStream | true | 导入文件的输入流 |
| sheetName | String | false | sheetName |
| clazz | Class<T> | true | 数据承载类的类对象 |
| titleRowNum | Integer | true | 标题结尾行号 |
| cacheSize | Integer | false | 缓存集合大小。导入数据时，当文件内数据过多时全部封装再保存数据库时太耗资源，所以设置缓存大小，当封装足够量的数据时保存一批。值为空时框架默认为100条数据 |
| handler | ExcelImportHandler<T> | true | 读取数据的钩子接口，当框架读取到每个单元格、每行、每次缓存集合放满都会回调该接口的对应方法 |

ExcelImportHandler接口的方法有：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 方法名 | 返回值 | 方法描述 | 参数 | 参数类型 | 参数描述 |
| eachColumnCallBack | boolean | 处理每个单元格的回调，可以在这里对每一列数据进行处理，比如数据校验、数据转换、对象赋值 | sheet | Sheet | 当前sheet对象 |
| rowNum | int | 当前行索引 |
| row | Row | 当前行对象 |
| column | int | 当前列索引 |
| cell | Cell | 当前单元格对象 |
| vo | 泛型对象 | 承接数据的对象 |
| value | Object | 当前单元格值 |
| eachRowCallBack | 泛型对象 | 处理每行数据的回调，在这里可以赋值其他导入数据之外的字段 | sheet | Sheet | 当前sheet对象 |
| rowNum | int | 当前行索引 |
| row | Row | 当前行对象 |
| vo | 泛型对象 | 承接数据的对象 |
| batchSaveCallBack | boolean | 批量保存的回调，数据缓存区放满或数据加载完毕执行此方法 | cacheList | List<T> | 数据缓存集合 |

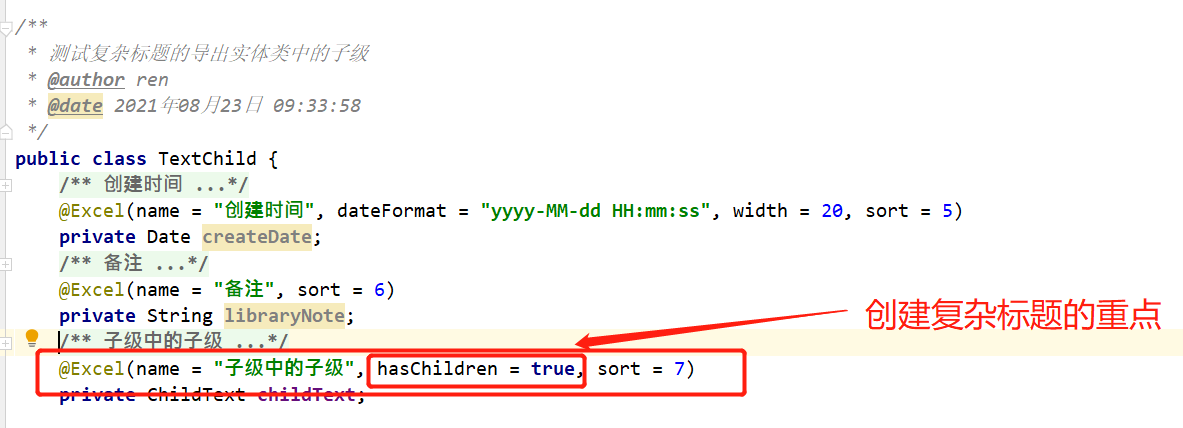
导入方法的返回值为ImportResult对象，具体为：

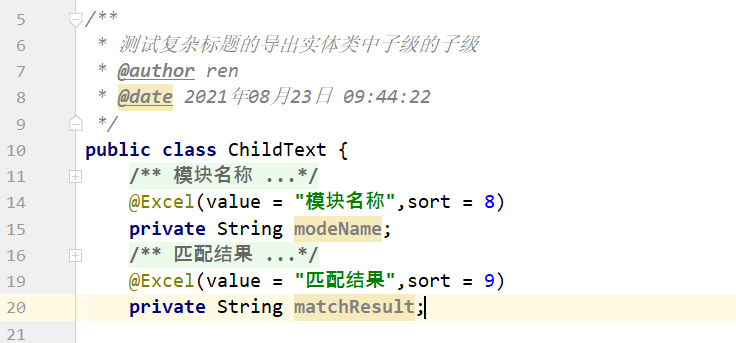
|  |  |  |
| --- | --- | --- |
| 属性 | 数据类型 | 描述 |
| code | ImportResult.Type | 状态码。ImportResult.Type.SUCCESS:全部成功，ImportResult.Type.ERROR:全部错误，ImportResult.Type.PART\_PASS :部分成功 |
| errorCount | Integer | 发生错误的次数 |
| errorRecord | List<String> | 发生错误的描述记录 |
| errorRow | List<Integer> | 发生错误的数据在Excel文件中的行号 |
| successRow | List<Integer> | 成功保存的数据在Excel文件中的行号 |
| elapsedTime | Long | 导入数据的总耗时，单位：毫秒 |

1. 演示案例：

* 导出







这里注意两点，hasCildren=true的属性不能为null，子级内的导出属性顺序要连贯。

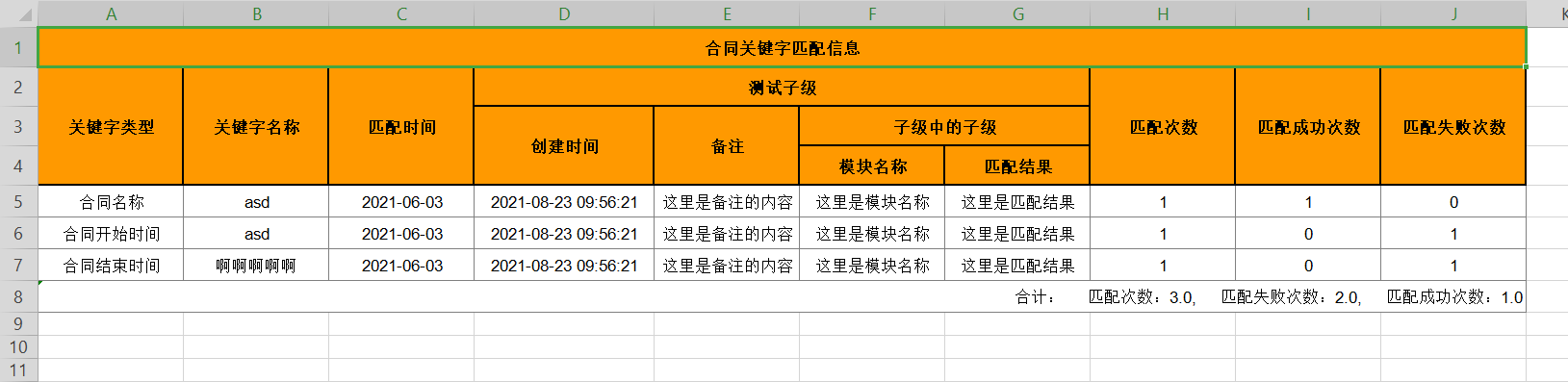
/\*\*  
 \* 导出合同关键字匹配统计列表  
 \* @param response  
 \* @param vo  
 \* @return  
 \*/  
 @GetMapping("/export")  
 @ResponseBody  
 public void export(HttpServletResponse response, StatisticsVO vo) throws Exception {  
 //查询数据

List<StatisticsVO> list = costKeywordMatchRecordService.

statisticsCostKeywordMatchRecord(vo);  
 ExcelUtils.exportExcel(StatisticsVO.class, list, "测试", "匹配统计.xlsx",

response, new CustomizeStyles() {  
 @Override  
 protected CellStyle createTitleStyles(Workbook wb) {  
 CellStyle style = wb.createCellStyle();  
 //设置单元格水平对齐方式  
 style.setAlignment(HorizontalAlignment.CENTER);  
 //设置单元格垂直对齐方式  
 style.setVerticalAlignment(VerticalAlignment.CENTER);  
 //设置右边框  
 style.setBorderRight(BorderStyle.MEDIUM);  
 //设置右边框颜色  
 style.setRightBorderColor(IndexedColors.BLACK.getIndex());  
 //设置左边框  
 style.setBorderLeft(BorderStyle.MEDIUM);  
 //设置左边框颜色  
 style.setLeftBorderColor(IndexedColors.BLACK.getIndex());  
 //设置上边框  
 style.setBorderTop(BorderStyle.MEDIUM);  
 //设置上边框颜色  
 style.setTopBorderColor(IndexedColors.BLACK.getIndex());  
 //设置下边框  
 style.setBorderBottom(BorderStyle.MEDIUM);  
 //设置下边框颜色  
 style.setBottomBorderColor(IndexedColors.BLACK.getIndex());  
 //设置前景颜色  
 style.setFillForegroundColor(IndexedColors.LIGHT\_ORANGE.getIndex());  
 //设置前景的风格样式  
 style.setFillPattern(FillPatternType.SOLID\_FOREGROUND);  
 //设置字体样式  
 Font headerFont = wb.createFont();  
 headerFont.setFontName("Arial");  
 headerFont.setFontHeightInPoints((short) 10);  
 headerFont.setBold(true);  
 headerFont.setColor(IndexedColors.BLACK.getIndex());  
 style.setFont(headerFont);  
 return style;  
 }  
 @Override  
 protected CellStyle createDataStyles(Workbook wb) {  
 CellStyle style = wb.createCellStyle();  
 //设置单元格水平对齐方式  
 style.setAlignment(HorizontalAlignment.CENTER);  
 //设置单元格垂直对齐方式  
 style.setVerticalAlignment(VerticalAlignment.CENTER);  
 //设置右边框  
 style.setBorderRight(BorderStyle.THIN);  
 //设置右边框颜色  
 style.setRightBorderColor(IndexedColors.GREY\_50\_PERCENT.getIndex());  
 //设置左边框  
 style.setBorderLeft(BorderStyle.THIN);  
 //设置左边框颜色  
 style.setLeftBorderColor(IndexedColors.GREY\_50\_PERCENT.getIndex());  
 //设置上边框  
 style.setBorderTop(BorderStyle.THIN);  
 //设置上边框颜色  
 style.setTopBorderColor(IndexedColors.GREY\_50\_PERCENT.getIndex());  
 //设置下边框  
 style.setBorderBottom(BorderStyle.THIN);  
 //设置下边框颜色  
 style.setBottomBorderColor(IndexedColors.GREY\_50\_PERCENT.getIndex());  
 //设置字体样式  
 Font dataFont = wb.createFont();  
 dataFont.setFontName("Arial");  
 dataFont.setFontHeightInPoints((short) 10);  
 style.setFont(dataFont);  
 return style;  
 }  
 @Override  
 protected CellStyle createTotalStyles(Workbook wb) {  
 CellStyle style = wb.createCellStyle();  
 //设置单元格水平对齐方式  
 style.setAlignment(HorizontalAlignment.RIGHT);  
 //设置单元格垂直对齐方式  
 style.setVerticalAlignment(VerticalAlignment.CENTER);  
 //设置右边框  
 style.setBorderRight(BorderStyle.THIN);  
 //设置右边框颜色  
 style.setRightBorderColor(IndexedColors.GREY\_50\_PERCENT.getIndex());  
 //设置左边框  
 style.setBorderLeft(BorderStyle.THIN);  
 //设置左边框颜色  
 style.setLeftBorderColor(IndexedColors.GREY\_50\_PERCENT.getIndex());  
 //设置上边框  
 style.setBorderTop(BorderStyle.THIN);  
 //设置上边框颜色  
 style.setTopBorderColor(IndexedColors.GREY\_50\_PERCENT.getIndex());  
 //设置下边框  
 style.setBorderBottom(BorderStyle.THIN);  
 //设置下边框颜色  
 style.setBottomBorderColor(IndexedColors.GREY\_50\_PERCENT.getIndex());  
 //设置字体样式  
 Font totalFont = wb.createFont();  
 totalFont.setFontName("Arial");  
 totalFont.setFontHeightInPoints((short) 10);  
 style.setFont(totalFont);  
 return style;  
 }  
 });  
 }

导出结果：



* 导入

public class Test {  
 public static void main(String[] args) throws IOException,InvalidFormatException {  
 File f = new File("C:\\Users\\Administrator\\Desktop\\匹配库.xlsx");  
 InputStream is = new FileInputStream(f);  
 ImportResult result =ExcelUtils.importExcel(is,CostKeywordMatchLibrary.class, 1,

new ExcelImportHandler<CostKeywordMatchLibrary>() {  
  @Override  
 public boolean eachColumnCallBack(Sheet sheet, int rowNum, Row row, int column,

Cell cell, CostKeywordMatchLibrary vo, Object value) {  
 System.out.println("执行每列的回调，当前是第" + (rowNum + 1) + "行" +

(column + 1) + "列，得到值为："+value);  
 switch (column){  
 case 0:  
 String type = (String)value;  
 switch (type){  
 case "合同编号": type = "1";break;  
 case "合同名称": type = "2";break;  
 case "合同金额": type = "3";break;  
 case "合同开始时间": type = "4";break;  
 case "合同结束时间": type = "5";break;  
 default: return false;  
 }  
 vo.setKeywordType(type);break;  
 case 1:vo.setKeywordCode((String)value);break;  
 case 2:vo.setKeywordName((String)value);break;  
 case 3:  
 String enabledState = (String)value;  
 switch (enabledState){  
 case "未启用": enabledState = "0";break;  
 case "已启用": enabledState = "1";break;  
 default: return false;  
 }  
 vo.setEnabledState(enabledState);break;  
 case 4:vo.setUpdateUser((String)value);break;  
 default:return false;  
 }  
 return true;  
 }  
 @Override  
 public CostKeywordMatchLibrary eachRowCallBack(Sheet sheet, int rowNum, Row row, CostKeywordMatchLibrary vo) {  
 vo.setCreateDate(new Date());  
 vo.setCreateUser("窝嫩叠");  
 vo.setUpdateDate(new Date());  
 vo.setOrgCode("组织机构代码");  
 vo.setModeName("模块名称");  
 vo.setModeId("模块id");  
 vo.setLibraryNote("这里是备注");  
 System.out.println("执行每行的回调，当前是第" + (rowNum + 1) + "行，处理对象："

+ vo.toString());  
 return vo;  
 }  
 @Override  
 public boolean batchSaveCallBack(List<CostKeywordMatchLibrary> cacheList) {  
 System.out.println("执行用户保存的回调，在这里模拟保存数据，集合中的数据为：");  
 for (CostKeywordMatchLibrary library : cacheList) {  
 System.out.println(library.toString());  
 }  
 return true;  
 }  
 });  
 System.out.println(result.toString());  
 }  
}

