

简要教程

目录

简要教程.....	1
前面的话.....	1
第一章 Excel 篇	2
Excel 的注解	2
Excel 导出	6
Excel 模板导出	11
Excel 导入	15
第二章 Word 篇.....	20
Word 模板导出.....	20
第三章 PDF 导出	21
第四章 HTML 导出	22
HTML 导出	22
第五章 Excel 图表	23
第六章 缓存以及工具类.....	24

修改记录

日期	版本	作者	修订类型	描述
2016-1-20	1. 0	JueYue	新建	
2016-1-22	1.1	JueYue	修改	导入校验,文件合法性校验

























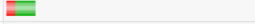



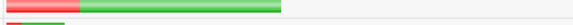





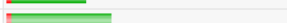



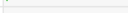


前面的话

EasyPoi 的编写其实是一次意外,之前我不太愿意写导入导出,因为代码号复杂,每次一个 Excel 都要写几百行,仅有少量的复用,一次需要写许多的导入导出,又没有人手,正好看到了 Jeecg 对应 Poi 的一个封装,但是他的封装比较简单,就自己在他思想的基础上开始构建现在的项目了.

EasyPoi 秉承思想就是尽量简单,入库少,可以很快把简单的工作干了这是第一个版本的功能,后来慢慢用的人多了,也就功能慢慢丰富了,现在包含了 Excel 的导入导出,Word 的导出,PDF 的导出,Excel Charts 的导出,Html 的导出 5 个功能模块,重点还是 Excel,毕竟 Excel 是最常用的.

EasyPoi 已经走过了 3 年,未来也会继续走,保持现有简单的功能继续前进.

程序员你懂得,懒得测试,目前测试覆盖率,基本功能我都写了,没有的就遇到了再说吧

>  org.jeecgframework.poi.excel.entity.result		26.5 %
>  org.jeecgframework.poi.excel.entity.sax		29.3 %
>  org.jeecgframework.poi.exception.excel		37.3 %
>  org.jeecgframework.poi.word.parse.excel		39.6 %
>  org.jeecgframework.poi.cache		40.2 %
>  org.jeecgframework.poi.word		41.2 %
>  org.jeecgframework.poi.handler.impl		45.5 %
>  org.jeecgframework.poi.word.entity		47.7 %
>  org.jeecgframework.poi.excel.export.stylar		50.9 %
>  org.jeecgframework.poi.excel		55.0 %
>  org.jeecgframework.poi.excel.imports.sax.parse		55.7 %
>  org.jeecgframework.poi.excel.html		59.8 %
>  org.jeecgframework.poi.excel.imports.base		62.0 %
>  org.jeecgframework.poi.excel.imports		63.6 %
>  org.jeecgframework.poi.excel.entity.params		67.3 %
>  org.jeecgframework.poi.excel.entity		67.9 %
>  org.jeecgframework.poi.excel.html.helper		68.8 %
>  org.jeecgframework.poi.cache.manager		69.9 %
>  org.jeecgframework.poi.util		70.7 %
>  org.jeecgframework.poi.excel.export.base		73.0 %
>  org.jeecgframework.poi.excel.imports.sax		73.5 %
>  org.jeecgframework.poi.exception.excel.enums		78.7 %
>  org.jeecgframework.poi.excel.entity.enums		84.4 %
>  org.jeecgframework.poi.excel.export.template		90.1 %
>  org.jeecgframework.poi.word.parse		93.4 %
>  org.jeecgframework.poi.excel.export		95.0 %
>  org.jeecgframework.poi.excel.entity.vo		100.0 %

第一章 Excel 篇

Excel 的注解

本来注解是整个模块的精髓,不过随着模板的使用,他的重要性也就不如以前了,不过还是最简单的使用方式.注解都是在 `easypoi-annotation` 这个 `jar`,拆分开的目的就是为了让多模块的 `maven` 项目少引用,因为 `easypoi-annotation` 是没有任何依赖的,只是单纯的注解

@Excel Excel 导出基本注释

这个注解是整个的基石,简单项目只靠这一个注解就可以完成所有功能

限定符和类型	可选元素和说明
<code>java.lang.String</code>	databaseFormat 导出时间设置,如果字段是 <code>Date</code> 类型则不需要设置 数据库如果是 <code>string</code> 类型,这个需要设置这个数据库格式
<code>java.lang.String</code>	exportFormat 导出的时间格式,以这个是否为空来判断是否需要格式化日期
<code>java.lang.String</code>	format 时间格式,相当于同时设置了 <code>exportFormat</code> 和 <code>importFormat</code>
<code>double</code>	height

	导出时在 excel 中每个列的高度 单位为字符，一个汉字=2 个字符
int	imageType 导出类型 1 从 file 读取 2 是从数据库中读取 默认是文件 同样导入也是一样的
java.lang.String	importFormat 导入的时间格式,以这个是否为空来判断是否需要格式化日期
boolean	isHyperlink 这个是不是超链接,如果是需要实现接口返回对象
java.lang.String	isImportField 导入时会校验这个字段,看看这个字段是不是导入的 Excel 中有,如果没有说明是错误的 Excel 本意是想用 true 的,想想还是 false 比较好 可以使用 a_id,b_id 来确实是否使用
boolean	isStatistics 是否自动统计数据,如果是统计,true 的话在最后追加一行统计,把所有数据都和 这个处理会吞没异常,请注意这一点
boolean	isWrap 是否换行 即支持\n
int[]	mergeRely 合并单元格依赖关系,比如第二列合并是基于第一列 则{1}就可以了
boolean	mergeVertical 纵向合并内容相同的单元格
boolean	needMerge 是否需要纵向合并单元格(用于含有 list 中,单个的单元格,合并 list 创建的多个 row)
java.lang.String	orderNum 展示到第几个可以使用 a_id,b_id 来确定不同排序
java.lang.String[]	replace 值得替换 导出是{a_id,b_id} 导入反过来,所以只用写一个
java.lang.String	savePath 导入路径,如果是图片可以填写,默认是 upload/className/IconEntity 这个类对应的就是 upload/Icon/
java.lang.String	suffix 文字后缀,如% 90 变成 90%
int	type

	导出类型 1 是文本 2 是图片,3 是函数 默认是文本
double	width 导出时在 excel 中每个列的宽 单位为字符, 一个汉字=2 个字符 如 以列名列内容中较合适的长度 例如姓名列 6 【姓名一般三个字】 性别列 4【男女占 1, 但是列标题两个汉字】 限制 1-255

这当中需要说明的是:

mergeVertical 和 mergeRely 这两者是一起存在的功能,就是纵向合并单元和, mergeRely 是判断和前面单元格的依赖,可用于多种集合,不太好导出情况,来合并处理

DEMO:

最简单的导出

```

/** 姓名 */
@Excel(name = "姓名")
private java.lang.String name;
/** 性别 */
@Excel(name = "性别")
private java.lang.String sex;
/** 资金 */
@Excel(name = "资金")
private java.lang.String money;
/** 身份证号 */
@Excel(name = "身份证号")
private java.lang.String card;
/** 住址 */
@Excel(name = "住址")
private java.lang.String address;

```

复杂点的导出:

```

/**
 * 学生姓名
 */
@Excel(name = "学生姓名", height = 20, width = 30, isImportField = "true_st")
private String name;
/**
 * 学生性别
 */
@Excel(name = "学生性别", replace = { "男_1", "女_2" }, suffix = "生", isImportField = "true_st")
private int sex;

@Excel(name = "出生日期", databaseFormat = "yyyyMMddHHmmss", format = "yyyy-MM-dd", isImportField = "true")
private Date birthday;

@Excel(name = "进校日期", databaseFormat = "yyyyMMddHHmmss", format = "yyyy-MM-dd", isImportField = "true")
private java.sql.Time registrationDate;

```

@ExcelTarget 标记导入 ID

就一个功能标记 ID,用以区别多个导出对象,便于一个对象用于多个导出实例

```
java.lang.String value
    定义 excel 导出 ID 来限定导出字段
```

DEMO

```
@ExcelTarget("courseEntity")
public class CourseEntityNoAnn implements java
    /** 主键 */
```

使用方法

```
@ExcelCollection(id = "st", name = "学生_courseEntity", orderNum = "4")
private List<StudentEntity> students;
```

@ExcelEntity 用来标示导出对象

因为对象不是基础属性,需要继续判断,所以加入这个属性,这个属性可以无限迭代,
ID 和 ExcelTarget 的 ID 功能一致

```
java.lang.String id
    定义 excel 导出 ID 来限定导出字段,处理一个类对应多个不同名称的情况

java.lang.String name
    导出时,对应数据库的字段 主要是用户区分每个字段, 不能有 annotation 重名的 导出时的列名 导出排序跟定义了 annotation 的字的段的顺序有关 可以使用 a_id,b_id 来确实是否使用
```

@ExcelCollection 标示集合

集合只能一层,如果 name 有值,表头就会有两行,有过没值就会只有一行

```
java.lang.String id
    定义 excel 导出 ID 来限定导出字段,处理一个类对应多个不同名称的情况

java.lang.String orderNum
    展示到第几个同样可以使用 a_id,b_id

java.lang.Class<?> type
```

创建时创建的类型 默认值是 arrayList

DEMO

```
@ExcelCollection(id = "st", name = "学生_courseEntity", orderNum = "4")
private List<StudentEntity> students;
```

测试					
课程名称	老师姓名	学生			
		学生姓名	学生性别	出生日期	进校日期
小白课程		撒旦法司法局	男生	2016-01-20	
		撒旦法司法局	男生	2016-01-20	

```
@ExcelCollection(id = "st", name = "|", orderNum = "4")
private List<StudentEntity> students;
```

2	测试					
3	课程名称	老师姓名	学生姓名	学生性别	出生日期	进校日期
4	小白课程		撒旦法司法局	男生	2016-01-20	
5			撒旦法司法局	男生	2016-01-20	

@ExcelIgnore 忽略字段

标记为 excel 创建实体忽略,放置死循环的造成

注解基本上就到这里来完了,主要还是各个注解直接的搭配使用,核心还是在于@Excel,主要的功能也是在这个地方,请大家开发之前先看看这个吧

Excel 导出

导出统一入口 ExcelExportUtil

excel, 导入导出, 模板导出的基础参数, 数据处理的接口

```
private IExcelDataHandler dataHandler
```

数据处理接口,以此为主,replace,format 都在这后面

默认实现 ExcelDataHandlerDefaultImpl,用户自定义的实现可以继承这个

```
>
; public class CourseHanlder extends ExcelDataHandlerDefaultImpl<CourseEntity> {
7
; @Override
; public Object exportHandler(CourseEntity obj, String name, Object value) {
;     if (name.equals("课程名称")) {
;         return String.valueOf(value) + "课程";
;     }
;     return super.exportHandler(obj, name, value);
; }
;
; @Override
; public Object importHandler(CourseEntity obj, String name, Object value) {
;     return super.importHandler(obj, name, value);
; }
; }
; }
```

当然你可以可以用 spring 来管理这个对象,都是一样的

ExportParams 导出参数

注解导出和 map 导出的参数设置

```
private boolean
```

addIndex

是否添加序列

```
private short
```

color

表头颜色

```
private java.lang.String[]
```

exclusions

过滤的属性

```
private int
```

freezeCol

冰冻列

```
private short
```

headerColor

属性说明行的颜色 例

如:HSSFColor.SKY_BLUE.index 默认

```
private java.lang.String
```

indexName

是否添加需要需要

```
private boolean
```

isAppendGraph

是否追加图形

```
private boolean
```

isCreateHeadRows

是否创建表头

private boolean	isDynamicData 是否动态获取数据
private java.lang.String	secondTitle 第二行名称
private short	secondTitleHeight 表格名称
private java.lang.String	sheetName sheetName
private java.lang.Class<?>	style Excel 导出 style
private java.lang.String	title 表格名称
private short	titleHeight 表格名称
private ExcelType	type Excel 导出版本

对象注解导出

注解导出 第一步先给对象加上注解,这个我就复述了,之后查出来对象列表,这个大家也都懂

```
/**
 * @param entity
 *         表格标题属性
 * @param pojoClass
 *         Excel对象Class
 * @param dataSet
 *         Excel对象数据List
 */
public static Workbook exportExcel(ExportParams entity, Class<?> pojoClass,
                                   Collection<?> dataSet) {
    Workbook workbook;
    if (ExcelType.HSSF.equals(entity.getType())) {
        workbook = new HSSFWorkbook();
    } else if (dataSet.size() < 1000) {
        workbook = new XSSFWorkbook();
    } else {
        workbook = new SXSSFWorkbook();
    }
    new ExcelExportServer().createSheet(workbook, entity, pojoClass, dataSet);
    return workbook;
}
```

就一个方法,大数据量推荐使用 2007 版本,效率高,生成文件小,

Class 填的就是 Collection 中的对象

DEMO

```
List<MsgClient> list = new ArrayList<MsgClient>();
for (int i = 0; i < 100; i++) {
    MsgClient client = new MsgClient();
    client.setBirthday(new Date());
    client.setClientName("小明" + i);
    client.setClientPhone("18797" + i);
    client.setCreateBy("JueYue");
    client.setId("1" + i);
    client.setRemark("测试" + i);
    MsgClientGroup group = new MsgClientGroup();
    group.setGroupName("测试" + i);
    client.setGroup(group);
    list.add(client);
}
Date start = new Date();
ExportParams params = new ExportParams("2412312", "测试", ExcelType.XSSF);
params.setFreezeCol(2);
Workbook workbook = ExcelExportUtil.exportExcel(params, MsgClient.class, list);
```

Map 的导出

有时候我们希望导出一个 map 集合或者我们导出的列表是不固定的,就需要这个导出了

```
/**
 * 根据Map创建对应的Excel
 * @param entity
 *      表格标题属性
 * @param entityList
 *      Map对象列表
 * @param dataSet
 *      Excel对象数据List
 */
public static Workbook exportExcel(ExportParams entity, List<ExcelExportEntity> entityList,
    Collection<? extends Map<?, ?>> dataSet) {
    Workbook workbook;
    if (ExcelType.HSSF.equals(entity.getType())) {
        workbook = new HSSFWorkbook();
    } else if (dataSet.size() < 1000) {
        workbook = new XSSFWorkbook();
    } else {
        workbook = new SXSSFWorkbook();
    }
    new ExcelExportServer().createSheetForMap(workbook, entity, entityList, dataSet);
    return workbook;
}
```

和上面方法的区别就是上面的 Classes 变成了 EntityList 其实 Classes 最好也会翻译成 EntityList, 只是自动帮你翻译了,这个方法就是把控制器还给你了,但是这个就比较麻烦,提供 4 个构造器 name 就是列的名称,key 对应就是 map 中的 key 或者属性的 name,便于 map 的快速导出 这个也是基本上支持注解中的所有功能

```
ExcelExportEntity()
```

```
ExcelExportEntity(java.lang.String name)
```

```
ExcelExportEntity(java.lang.String name,  
java.lang.Object key)
```

```
ExcelExportEntity(java.lang.String name,  
java.lang.Object key, int width)
```

DEMO

```
List<ExcelExportEntity> entity = new ArrayList<ExcelExportEntity>();  
ExcelExportEntity excelentity = new ExcelExportEntity("部门", "depart");  
excelentity.setMergeVertical(true);  
entity.add(excelentity);  
excelentity = new ExcelExportEntity("姓名", "name");  
excelentity.setMergeVertical(true);  
excelentity.setMergeRely(new int[]{0});  
entity.add(excelentity);  
excelentity = new ExcelExportEntity("电话", "phone");  
excelentity.setMergeVertical(true);  
excelentity.setMergeRely(new int[] { 1 });  
entity.add(excelentity);
```

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

```
Workbook workbook = ExcelExportUtil.exportExcel(new ExportParams("员工通讯录", "通讯录"),  
entity, list);
```

一次导出多个对象

其实就是对第一个方法的一个封装,没有什么需要介绍的

```
/**  
 * 一个excel 创建多个sheet  
 *  
 * @param list  
 *      多个Map key title 对应表格Title key entity 对应表格对应实体 key data  
 *      Collection 数据  
 * @return  
 */  
public static Workbook exportExcel(List<Map<String, Object>> list, String type) {  
    Workbook workbook;  
    if (ExcelType.HSSF.equals(type)) {  
        workbook = new HSSFWorkbook();  
    } else {  
        workbook = new XSSFWorkbook();  
    }  
    for (Map<String, Object> map : list) {  
        ExcelExportServer server = new ExcelExportServer();  
        server.createSheet(workbook, (ExportParams) map.get("title"),  
            (Class<?>) map.get("entity"), (Collection<?>) map.get("data"));  
    }  
    return workbook;  
}
```

定义样式

自定义 Cell 样式接口,方便大家自己处理 Excel 的样式

```
1
2 /**
3  * Excel导出样式接口
4  * @author JueYue
5  * 2015年1月9日 下午5:32:30
6  */
7 public interface IExcelExportStyler {
8
9     * 列表头样式
10    public CellStyle getHeaderStyle(short headerColor);
11
12
13     * 标题样式
14    public CellStyle getTitleStyle(short color);
15
16
17     * 获取样式方法
18    public CellStyle getStyles(boolean noneStyler, ExcelExportEntity entity);
19 }
```

提供了三个实现,

ExcelExportStylerBorderImpl 有边框样式

ExcelExportStylerColorImpl 有边框间隔行样式

ExcelExportStylerDefaultImpl 默认的风格

样式使用方法 ExportParams 中调用 setStyler 就可以了,必须实现有参构造器

```
public ExcelExportStylerBorderImpl(Workbook workbook) {
    super.createStyles(workbook);
}
```

Excel 模板导出

为了更好的导出 Excel,以及 Excel 的样式定义,复杂的表头等功能对已代码设置的很复杂的情况下使用模板确实是更好的办法,下面这样的样式我们自己实现要实现好久,但是模板就比较容易实现了,Excel 分分钟的事情,模板秉承的是继承你的 Cell 样式,尽量不破坏你的东西

10份部门工资汇总					
类 别		人数	金额(元)	本年累计金额 (元)	备注
后勤部门		10	1000	10000	
技术部	企业部	20	2000	20000	
	移动部	30	3000	30000	
	GIS部	40	4000	40000	
市场部	大客户部	50	5000	50000	
	个人市场部	60	6000	60000	
行政部		70	7000.00	70000	
合 计		0	0	0	

模板支持的语法

- 1) 空格分割
- 2) 三目运算 `{{test ? obj:obj2}}`
- 3) `n`: 表示 这个 cell 是数值类型 `{{n:}}`
 - 用来处理数字问题,防止上面有小叹号
- 4) `le`: 代表长度`{{le:()}}` 在 if/else 运用`{{le:() > 8 ? obj1 : obj2}}`
- 5) `fd`: 格式化时间 `{{fd:(obj;yyyy-MM-dd)}}`
 - 使用 java 的时间格式化,进行格式化处理
- 6) `fn`: 格式化数字 `{{fn:(obj;###.00)}}`
 - 处理小数点问题
- 7) 单引号表示常量值 " 比如'1' 那么输出的就是 1
- 8) `fe`: 遍历数据,创建 row
- 9) `!fe`: 遍历数据不创建 row
- 10) `$fe`: 下移插入,把当前行,下面的行全部下移.size()行,然后插入
- 11) `#fe`: 行遍历
 - 其他都是列遍历,这个是横向遍历
- 12) `!if`: 删除当前列 `{{!if:(test)}}`
- 13) `&NULL&` 控制
- 14) `]]` 换行符
 - 解决多行遍历的问题,详情看下嘛的遍历标签

遍历标签的使用

语法就上面这些,前面 7 个都没啥需要讲的都是我们一样的语法,说下下面几个

8,9,10 都是一样的就是 for 循环创建表格,不同的是 fe 是全部都创建自己的表格,!fe 是先使用现有的表格,不够了再创建表格,\$fe 是把这样之下的先往下移动 x 行,在便利,说下 3 个的场景
第一个不说了,是不是都要创建
第二个是:比如我们就创建 4 行或者<4 行,4 行之后还有别的数据,那么我们就使用!fe 来迭代

ID	姓名	性别	高度
{{!fe: maplist t.id	t.name	t.sex == 1 ? 男 : 女}	测试1
			测试2
			测试3
			测试4
插入下移tn	姓名	性别	

生成后不会影响其他数据

1	不创建Row的Foreach, 空白的cell必须满足collect.size			
2	ID	姓名	性别	高度
3	xman	小明0	女	测试1
4				测试2
5				测试3
6				测试4
7	插入下移ID	姓名	性别	

第三个是:当遍历行下面仍有数据时,不影响下面的数据,进行插入操作

专项支出用款申请书									
财政代码：									
申请单位：（公章）					申请日期：{{date}}			单位：元	
序号	资金性质	预算科目		项目名称	收款人			申请金额	核定金额
		编码	名称		全称	银行账号	开户银行		
{{ \$fe: ma	t.accountType	t.bianma	t.mingcheng	t.xiangmumingcheng	t.quancheng	t.yinhang	t.kaihu	t.sqje	t.hdje}}
合					计：{{money}}				
核定金额合计（大写）：{{upperMoney}}									
申请支付单位：{{company}}						××局：{{bureau}}			
单位负责人：（印章）						财务负责人：（印章）			
								年	月 日
注：本表一式三份。						联系人：{{person}}		联系电话：{{phone}}	

输出结果

专项支出用款申请书									
财政代码:									
申请单位: (公章)				申请日期: 2014-12-25				单位: 元	
序号	资金性质	预算科目		项目名称	收款人			申请金额	核定金额
		编码	名称		全称	银行账号	开户银行		
1		A001	设计	EasyPoi 0期	开源项目			0	0
2		A001	设计	EasyPoi 1期	开源项目			10000	10000
3		A001	设计	EasyPoi 2期	开源项目			20000	20000
4		A001	设计	EasyPoi 3期	开源项目			30000	30000
合 计: 200000.0									
核定金额合计(大写): 贰佰万									
申请支付单位: 杭州银行科技有限公司					xx局: 财政局				
单位负责人: (印章)					财务负责人: (印章)				
					年 月 日				
注: 本表一式三份。					联系人: JueYue		联系电话: 1879740****		

多行遍历

针对多行遍历问题,给出了]] 作为换行符用来标记 list 还没执行完的问题,多行遍历解决了部分 Excel 处理难的问题

班级成绩					
班号	名称	科目	0-60	60-80	80-100
{!fe: list t.id	t.name	' 语文'	&NULL&	t.a1.sixty	t.a1.eighty]]
		' 数学'	t.a2.zero	t.a2.sixty	t.a2.eighty]]
		' 英语'	t.a3.zero	t.a3.sixty	t.a3.eighty]]
		' 小计'	t.sum1		t.sum2}}

生产效果

	A	B	C	D	E	F	G	H
1	班级成绩							
2	班号	名称	科目	0-60	60-80	80-100		
3	0801010	大学0班	语文		89	52		
4			数学	29	2	52		
5			英语	28	76	20		
6			小计	300		400		
7	0801011	大学1班	语文		70	59		
8			数学	39	53	83		
9			英语	5	55	93		
0			小计	301		401		
1	0801012	大学2班	语文		8	26		
2			数学	72	59	89		
3			英语	29	24	6		
4			小计	302		402		

对应测试类 TemplateForEachTest2

横向遍历

	A	B	C	D	E
1	一级分类	二级分类	{!#fe: collist t.name}}		
2			#fe: collist t.	t.cw	t.tj}}
3			fe: collist t.	t.cwmk	t.tjmk}}

效果

	A	B	C	D	E	F	G	H
1	一级分类	二级分类	小明挑战				小红挑战	
2			正确	错误	统计	正确	错误	统计
3	运动	跑步	1	2	3	4	2	6
4		跳高	1	2	3	4	2	6
5		文化	1	2	3	4	2	6

用来解决横向纵向双向遍历统计的问题

对应测试类 ExcelExportTemplateColFeTest

SUM 的设计思路

因为模板中的 SUM 具有位置和属性的双重不确定性,所以针对 SUM 的这种运算,打算使用记忆的方法来预先记录 SUM 的位置以及需要 SUM 的对象,等待遍历处理的时候后置处理

了,也防止遍历后,再扫描整个界面来处理问题

SUM 的位置可以在 FE 的上面也可以在 FE 的下面,也可以直接在 FE 中来直接 SUM 比如在 FE 后

商户号	商户	交易额	省	市
{{ \$fe: maplist t.merchant_no	merchantname	t.money	t.prov	t.city}}
汇总		{{sum(t.money)}}		

在 FE 中

1	商户号	商户	交易额	省	市
2	{{ \$fe: maplist t.merchant_no	t.merchantname	sum(t.money)	t.prov	t.city}}
3					

在 FE 前

A	B	C	D	E
		交易总额: {{sum(t.money)}}		
商户号	商户	交易额	省	市
{{ \$fe: maplist t.merchant_no	t.merchantname	t.money	t.prov	t.city}}

为了防止效率的浪费,回预先记录这个 SUM 的位置,用了有效处理整个 SUM 后期处理寻址的问题,在使用中 SUM 只能 SUM list 遍历中的数据,不能 SUM 其他的数据,位置可以自己定义,没有要求

导出操作

提供两个方法,一个是都是模板,第二个是第一个封装,就是多个 sheet 的模板导出

```
* 导出文件通过模板解析只有模板,没有集合
public static Workbook exportExcel(TemplateExportParams params, Map<String, Object> map) {
    return new ExcelExportOfTemplateUtil().createExcelByTemplate(params, null, null, map);
}

* 导出文件通过模板解析只有模板,没有集合
public static Workbook exportExcel(Map<Integer, Map<String, Object>> map,
    TemplateExportParams params) {
    return new ExcelExportOfTemplateUtil().createExcelByTemplate(params, map);
}
```

我的使用方式,如果是样式复杂就是模板,样式简单就是代码导出的

Excel 导入

注解导入

基于 Excel 注解导出,只要反过来看代码就可以了,基本上不用讲什么,导出支持的功能,基本上导入同样支持

```

    * Excel 导入 数据源IO流,不返回校验结果 导入 字段类型 Integer,Long,Double,Date,String,Boolean
    public static <T> List<T> importExcel(InputStream inputstream, Class<?> pojoClass,
        ImportParams params) throws Exception {
        return new ExcelImportServer().importExcelByIs(inputstream, pojoClass, params).getList();
    }

```

Map 导入

传入对象传入 map 就可以,返回 Map<String,Object> 这个 Object 请大家注意了不确定是什么不过都是基本类型

```

    * Excel 导入 数据源IO流,不返回校验结果 导入 字段类型 Integer,Long,Double,Date,String,Boolean
    public static <T> List<T> importExcel(InputStream inputstream, Class<?> pojoClass,
        ImportParams params) throws Exception {
        return new ExcelImportServer().importExcelByIs(inputstream, pojoClass, params).getList();
    }

```

和上面的是同一个方法

Sax 导入

支持的功能不如上面的丰富,不在会吃图片,紧支持基本类型,
对于大数据量导入请使用 Sax,避免造成内存溢出

```

/**
 * Excel 通过SAX解析方法,适合大数据导入,不支持图片
 * 导入 数据源IO流,不返回校验结果 导入 字段类型 Integer,Long,Double,Date,String,Boolean
 *
 * @param inputstream
 * @param pojoClass
 * @param params
 * @return
 */
public static <T> List<T> importExcelBySax(InputStream inputstream, Class<?> pojoClass,
    ImportParams params) {
    return new SaxReadExcel().readExcel(inputstream, pojoClass, params, null, null);
}

```

Excel 数据校验

校验是集成的 JSR303,可选 hibernate 或者 Apache 的实现,这个我就不讲了,请大家自行百度,功能都是一样的,需要开启设置在 ImportParams 中

这是一个事前校验,一般是数据的合法性,针对数据唯一性什么的,可以使用接口校验

```

/**
 * 是否需要校验上传的Excel,默认为false
 */
private boolean needVerfiy = false;

```

同时提供了导入校验的接口,是整个对象一起校验的,返回校验结果就可以,比较简单


```

/**
 * 导入校验接口
 *
 * @author JueYue
 * 2014年6月23日 下午11:08:21
 */
public interface IExcelVerifyHandler<T> {

    /**
     * 导入校验方法
     *
     * @param obj
     *      当前对象
     * @return
     */
    public ExcelVerifyHanlderResult verifyHandler(T obj);

}

```

比如

```

public class ExcelVerifyHandlerImpl implements IExcelVerifyHandler<ExcelVerifyEntity> {

    @Override
    public ExcelVerifyHanlderResult verifyHandler(ExcelVerifyEntity obj) {
        StringBuilder builder = new StringBuilder();
        if (StringUtils.isEmpty(obj.getEmail())) {
            builder.append("Email must null;");
        }
        if (obj.getMax() > 15) {
            builder.append("max must lt 15;");
        }
        return new ExcelVerifyHanlderResult(false, builder.toString());
    }

}

```

下面说下错误信息的处理

错误信息默认会追加到这一行最后创建一个 Cell 中去,就是你可以返回这个 Excel 给用户,再让他改,

	A	B	C	D	E	F	G	H	I	J	K	L
Email		Max	Min	NotNull	Regex							
qrb.juevue@gmail.com		12	7	juevue	绝月	123	不是中文,may not be null,max 最大值不能超过15,must be greater than or equal to 3					
qrb.juevue@gmail.com	21312313	12312	0	-9 大苏打	萨达	2313	must be greater than or equal to 3,max 最大值不能超过15					
	21312313	13	0				may not be null,must be greater than or equal to 3,不是中文					

同时,我们也希望存放到数据库中这是我们需要我们的 Entity 实现一个接口

```


/**
 * Excel导入校验类
 * @author JueYue
 * 2015年2月24日 下午2:21:07
 */
public class ExcelVerifyEntityOfMode extends ExcelVerifyEntity implements IExcelModel {

    private String errorMsg;

    @Override
    public String getErrorMsg() {
        return errorMsg;
    }

    @Override
    public void setErrorMsg(String errorMsg) {
        this.errorMsg = errorMsg;
    }
}

```



接口

这样我们会把错误信息追加到你的对象中用于保存到数据库中
同时,会返回校验 Result 对象,用于判断是不是有错误,集合,已经 Excel

```

public class ExcelImportResult<T> {

```

```

/**
 * 结果集
 */
private List<T> list;

/**
 * 是否存在校验失败
 */
private boolean verfiyFail;

/**
 * 数据源
 */
private Workbook workbook;

```

文件合法性校验

文件合法性校验,是校验我们的文件是不是我们给对方的模板,是否包含全了我们要求的列

```

public static <T> List<T> importExcel(File file, Class<?> pojoClass, ImportParams params) {
    FileInputStream in = null;
    try {
        in = new FileInputStream(file);
        return new ExcelImportServer().importExcelByIs(in, pojoClass, params).getList();
    } catch (ExcelImportException e) {
        throw new ExcelImportException(e.getType(), e);
    } catch (Exception e) {
        LOGGER.error(e.getMessage(), e);
        throw new ExcelImportException(e.getMessage(), e);
    } finally {
        IOUtils.closeQuietly(in);
    }
}

```

这个校验是中断校验,发现校验失败后会爆出异常,异常枚举值

IS_NOT_A_VALID_TEMPLATE ("不是合法的Excel模板") ,

校验方法有两种,1.注解校验,2 数组校验,推荐后者

1. 注解校验

在注解中加入

```

/**
 * 导入时会校验这个字段,看看这个字段是不是导入的Excel中有,如果没有说明是错误的Excel
 * 本意是想用true的,想想还是false比较好
 * 可以使用a_id,b_id来确实是否使用
 * @return
 */
public String isImportField() default "false";

```

DEMO:

```

@Excel(name = "姓名", isImportField = "true")
private String name;
@Excel(name = "性别", isImportField = "true")
private String sex;
@Excel(name = "年纪", isImportField = "true")
private String age;
@Excel(name = "爱好", isImportField = "true")
private String hobby;

```

2. 数据值校验,

就是在导入参数中动态设置需要校验的对象比如:

```

params.setImportFields(new String[] { "姓名", "性别", "年纪", "爱好" });

```

如果有集合也可以校验

```

params.setImportFields(new String[] { "课程名称", "老师姓名", "学生_学生姓名", "学生_学生性别", "学生_出生日期", "学生_进校日期" });

```

同样抛出校验异常信息,终止导入流程

第二章 Word 篇

Word 模板导出

因为 POI 的 Word 2003 支持的很不好,我也就不进行支持了,现在只支持 2007 版本支持值替换,图片替换,table 等功能

```
/**
 * 解析Word2007版本
 *
 * @param url
 *      模板地址
 * @param map
 *      解析数据源
 * @return
 */
public static XWPFDocument exportWord07(String url, Map<String, Object> map) throws Exception {
    return new ParseWord07().parseWord(url, map);
}
```

基本功能是基于 Excel 模板导出功能的 clone,实现了 Excel 模板中的大部分功能,语法也都是保持一致,目前不能嵌套 table 请注意

语法啥的就不讲了,说下不同的地方

提供一个专用的图片替换类,判断这个对象会把这个地方替换成图片

```
25 public class WordImageEntity {
26
27     public static String URL = "url";
28     public static String Data = "data";
29     /**
30      * 图片输入方式
31      */
32     private String type = URL;
33     /**
34      * 图片宽度
35      */
36     private int width;
37     // 图片高度
38     private int height;
39     // 图片地址
40     private String url;
41     // 图片信息
42     private byte[] data;
43 }
```

第三章 PDF 导出

Pdf 导出

同样是基于 Excel 模块的功能扩展,目前只支持 Excel 注解导出和 map 导出,还没支持模板

提供了样式接口 IPdfExportStyler,用于创建 Cell 时候的样式设置

```
public interface IPdfExportStyler {  
  
    /**  
     * 获取文档格式  
     * @return  
     */  
    public Document getDocument();  
  
    /**  
     * 设置Cell的样式  
     * @param entity  
     * @param text  
     */  
    public void setCellStyler(PdfPCell iCell, ExcelExportEntity entity, String text);  
  
    /**  
     * 获取字体  
     * @param entity  
     * @param text  
     */  
    public Font getFont(ExcelExportEntity entity, String text);  
}
```

方法也和 Excel;类似,但是要提供流

```
* 根据注解导出数据  
public static Document exportPdf(PdfExportParams entity, Class<?> pojoClass,  
    Collection<?> dataSet, OutputStream outStream) {  
    return new PdfExportServer(outStream, entity).createPdf(entity, pojoClass, dataSet);  
}  
  
* 根据Map创建对应的PDF  
public static Document exportPdf(PdfExportParams entity, List<ExcelExportEntity> entityList,  
    Collection<? extends Map<?, ?>> dataSet,  
    OutputStream outStream) {  
  
    return new PdfExportServer(outStream, entity).createPdfByExportEntity(entity, entityList,  
        dataSet);  
}
```

第四章 HTML 导出

HTML 导出

目前只支持 Excel 转为 HTML,效果还算凑合,给用户看看可以,企业级还是需要专业的解决方案

专项支出用款申请书										
财政代码：										
申请单位：（公章）				申请日期：{{date}}						单位：元
序号	资金性质	预算科目		项目名称	收款人			申请金额	核定金额	
		编码	名称		全称	银行账号	开户银行			
合 计：{{money}}										
核定金额合计（大写）：{{upperMoney}}										
申请支付单位：{{company}}						××局：{{bureau}}				
单位负责人：（印章）				财务负责人：（印章）						
								年 月 日		
						联系人：{{person}}	联系电话：	{{phone}}		
注：本表一式三份。										

导出返回两种界面,支持图片转换
toTableHtml 只返回 table 对象
toAllHtml 返回包含 html 的对象

导出代码,算是一个单独的工具类

```
public class ExcelToHtmlUtil {  
  
    private ExcelToHtmlUtil() {  
    }  
  
    * 转换为Table  
    public static String toTableHtml(Workbook wb) {  
        return HtmlCache.getHtml(new ExcelToHtmlParams(wb, false, 0, null));  
    }  
  
    * 转换为Table,显示图片  
    public static String toTableHtml(Workbook wb, String path) {  
        return HtmlCache.getHtml(new ExcelToHtmlParams(wb, false, 0, path));  
    }  
}
```

第五章 Excel 图表

第六章 Spring web 集成

Spring mvc 的配置

使用 spring mvc 需要在 spring-mvc.xml 或者其他的配置文件中进行配置,主要是默认视图级别设置低点

```
<!-- 默认的视图解析器 在上边的解析错误时使用 (默认使用html)- -->
<bean id="defaultViewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver"
      p:order="3">
    <property name="viewClass"
      value="org.springframework.web.servlet.view.JstlView" />
    <property name="contentType" value="text/html" />
    <property name="prefix" value="/webpage/" />
    <property name="suffix" value=".jsp" />
</bean>
```

Bean 视图设置级别高一些,然后把我们的视图配置上,就完成了

```
<!-- Bean解析器,级别高于默认解析器,寻找bean对象进行二次处理 -->
<bean id="beanNameViewResolver"
      class="org.springframework.web.servlet.view.BeanNameViewResolver" p:order="0">
</bean>
<!-- Excel 处理 根据用户输入进行对象处理 -->
<bean id="jeecgExcelView" class="org.jeecgframework.poi.excel.view.JeecgSingleExcelView" />
<bean id="jeecgTemplateExcelView" class="org.jeecgframework.poi.excel.view.JeecgTemplateExcelView" />
<bean id="jeecgTemplateWordView" class="org.jeecgframework.poi.excel.view.JeecgTemplateWordView" />
<bean id="jeecgMapExcelView" class="org.jeecgframework.poi.excel.view.JeecgMapExcelView" />
```

2.0.8 版本后加上了@Controller 里面只要在

```
<context:component-scan base-
package="org.jeecgframework.poi.excel.view">
```

加入就可以了,但是解析器还是要配置的
这样就算集成完成了,下面我们介绍各种 VIEW

注解 view JeecgSingleExcelView

第七章 缓存以及工具类

缓存

缓存是基于 guava 来实现的
提供了 excel 模板读取缓存,Html 缓存,word 模板读取缓存,图片读取缓存

讲下自定义文件读取
IFileLoader 提供文件读取接口,自己实现这个接口,比如

```
4 public class FileLoadeImpl implements IFileLoader {
5
6     private static final Logger LOGGER = LoggerFactory.getLogger(FileLoadeImpl.class);
7
8     public byte[] getFile(String url) {
9         FileInputStream fileis = null;
10        ByteArrayOutputStream baos = null;
11        try {
12            //先用绝对路径查询,再查询相对路径
13            try {
14                fileis = new FileInputStream(url);
15            } catch (FileNotFoundException e) {
16                String path = PoiPublicUtil.getWebRootPath(url);
17                fileis = new FileInputStream(path);
18            }
19            baos = new ByteArrayOutputStream();
20            byte[] buffer = new byte[1024];
21            int len;
22            while ((len = fileis.read(buffer)) > -1) {
23                baos.write(buffer, 0, len);
24            }
25            baos.flush();
26            return baos.toByteArray();
27        }
28    }
29 }
```

然后根据自己的需要可以把自己实现的类设置为全局的类,或者线程内使用的类
POICacheManager 提供对应的方法

```
public static void setFileLoder(IFileLoader fileLoder) {
    POICacheManager.fileLoder = fileLoder;
}

/**
 * 一次线程有效
 * @param fileLoder
 */
public static void setFileLoderOnce(IFileLoader fileLoder) {
    if (fileLoder != null)
        LOCAL_FILELOADER.set(fileLoder);
}
```

对象	缓存时间	最大缓存数量
Excel,Word	1 小时	50
Html	7 天	200 个
Image	1 天	200 个

每个 cache 都提供了 set 方法,以便大家设置自己的实现

工具类

PoiReflectorUtil 反射工具类,从 mybatis 上 copy 下来的

PoiMergeCellUtil 纵向合并工具类 合并内容相同的单元格

PoiCellUtil 读取单元格值,判断合并情况