EasyPoi教程 V1.0

文档

1. 前传

1.1 前言

easypoi功能如同名字easy,主打的功能就是容易,让一个没见接触过poi的人员
就可以方便的写出Excel导出,Excel模板导出,Excel导入,Word模板导出,通过简单的注解和模板语言(熟悉的表达式语法),完成以前复杂的写法

官网：http://www.afterturn.cn/
邮箱： qrb.jueyue@gmail.com
QQ群: 364192721

不如poi那么自定义,不如jxl那么多标签,但是我们就是写的少,写的少

EasyPoi的主要特点

1.设计精巧,使用简单
2.接口丰富,扩展简单
3.默认值多,write less do more
4.spring mvc支持,web导出可以简单明了

功能

Excel自适应xls和xlsx两种格式,word只支持docx模式

1.Excel导入
• 注解导入
• Map导入
• 大数据量导入sax模式
• 导入文件保存
• 文件校验
• 字段校验
2.Excel导出
• 注解导出
• 模板导出
• html导出
3.Excel转html
4.word导出
5.pdf导出

1.2 Easypoi介绍

Easypoi 为谁而开发

- 不太熟悉poi的
- 不想写太多重复太多的
- 只是简单的导入导出的
- 喜欢使用模板的

都可以使用easypoi

Easypoi的目标是什么

Easypoi的目标不是替代poi,而是让一个不懂导入导出的快速使用poi完成Excel和word的各种操作,而不是看很多api才可以完成这样工作

为什么会写Easypoi

以前的以前(岁月真TMD的快)我虽然写了不少代码但还是很少写poi,然后跳到一家公司之后就和业务人员聊上了,来这个需要个报表,这个报表样式是这样的,这个表头是这样的,就这样我写了大量的poi代码,每次都是大量的篇幅,copy to copy,无聊的一逼,然后加入了jeecg,jeecg中有一个小的工具类,虽然我也不知道是谁写的,然是可以用注解搞定最简单的导出,突然豁然开朗,我可以完善,让我从报表的苦海当中脱离出来,这样我花了一周的时间做了第一个版本支持导入导出放到了jeecg,发现还是不错的,慢慢的用的人越来越多,我就把这块独立出来了,再然后有人提出了模板,然后就加入了模板功能,提出了word的需求,加入了word的功能,后来工作忙了虽然没再参与jeecg,但还是一直维持这easypoi的更新,根据见识的增长也不断的重构这代码,直到现在

独特的功能

- 基于注解的导入导出,修改注解就可以修改Excel
- 支持常用的样式自定义
- 基于map可以灵活定义的表头字段
- 支持一堆多的导出,导入
- 支持模板的导出,一些常见的标签,自定义标签
- 支持HTML/Excel转换,如果模板还不能满足用户的变态需求,请用这个功能
- 支持word的导出,支持图片,Excel

小白如何开始

- 下载demo运行看看,基本上常见的用用法都在里面easypoi-test(https://gitee.com/lemur/easypoi-test)
- 查看几个*Util的用法,Easypoi的主要输出就是这个
- 看看注解的意思
- 看看模板的标签用法
- 可以出师了

1.3 使用

- 1.easypoi 父包--作用大家都懂得

- 2.easypoi-annotation 基础注解包,作用与实体对象上,拆分后方便maven多工程的依赖管理
- 3.easypoi-base 导入导出的工具包,可以完成Excel导出,导入,Word的导出,Excel的导出功能
- 4.easypoi-web 耦合了spring-mvc 基于AbstractView,极大的简化spring-mvc下的导出功能
- 5.sax 导入使用xercesImpl这个包(这个包可能造成奇怪的问题哈),word导出使用poi-scratchpad,都作为可选包了

如果不使用spring mvc的便捷福利,直接引入easypoi-base 就可以了,easypoi-annotation

如果使用maven,请使用如下坐标

```
<dependency>
        <groupId>cn.afterturn</groupId>
        <artifactId>easypoi-base</artifactId>
        <version>3.2.0</version>
    </dependency>
    <dependency>
        <groupId>cn.afterturn</groupId>
        <artifactId>easypoi-web</artifactId>
        <version>3.2.0</version>
    </dependency>
    <dependency>
        <groupId>cn.afterturn</groupId>
        <artifactId>easypoi-annotation</artifactId>
        <version>3.2.0</version>
    </dependency>
```

如果没有maven请直接下jar,在alimaven(http://maven.aliyun.com/nexus/#nexus-search;quick~easypoi)

1.4 测试项目

测试这个事情真不是个容易的事情

测试项目包括两块 Junit 的常见测试和spring 的view测试

1.spring view测试

运行application就可以了,访问界面,然后看到界面

图片地址：https://static.oschina.net/uploads/img/201710/10230428_pMr5.png

对应的代码在view下面

2.Junit的测试目录结构如下

- tohtml html预览测试
- view 导出的view测试
- cache 自定义缓存测试
- html html互转测试
- test

- excel
- read 读取Excel测试
- check 导入检查测试
- hanlder 导入数据处理
- img 含图片导入测试
- styler 导出样式自定义测试
- template 模板导出测试
- sum 导出含统计测试
- test 导出测试
- groupname groupname 属性测试
- img 导出图片测试
- pdf pdf测试
- word word导出测试
- util util 内部测试

目前的测试覆盖率

图片地址：https://static.oschina.net/uploads/img/201710/10230945_d2rq.png

1.5 可能存在的小坑

缓存问题

缓存问题好像是很多童鞋都遇到的问题，可能是我设计的逻辑问题，但是一直没有好的解决，

但是我也给了一个无奈的解决方案，大家可以自己实现接口IFileLoader

```
public interface IFileLoader {
    /**
     * 可以自定义KEY的作用
     * @param key
     * @return
     */
    public byte[] getFile(String key);

}
```

来自己获取自己的文件，用来解决文件获取不到的问题

设置提供了两种方案，都是POICacheManager 的静态方法

```
    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);
    }
}
```

第一个是全局替换，可以在项目启动的时候，设置下就可以了，第一个是当前线程有效，希望可以帮助大家解决问题，我再研究下更通用的文件获取，或者那位朋友提供下自己的通用方案。

2. Excel 注解版

2.1 Excel导入导出

Excel的导入导出是Easypoi的核心功能,前期基本也是围绕这个打造的,主要分为三种方式的处理,其中模板和Html目前只支持导出,因为支持Map.class其实导入应该是怎样都支持的

- 注解方式,注解变种方式
- 模板方式
- Html方式

下面分别就这三种方式进行讲解

2.2 注解

注解介绍

easypoi起因就是Excel的导入导出,最初的模板是实体和Excel的对应,model--row,filed--col 这样利用注解我们可以和容易做到excel到导入导出

经过一段时间发展,现在注解有5个类分别是

- @Excel 作用到filed上面,是对Excel一列的一个描述
- @ExcelCollection 表示一个集合,主要针对一对多的导出,比如一个老师对应多个科目,科目就可以用集合表示
- @ExcelEntity 表示一个继续深入导出的实体,但他没有太多的实际意义,只是告诉系统这个对象里面同样有导出的字段
- @ExcelIgnore 和名字一样表示这个字段被忽略跳过这个导导出
- @ExcelTarget 这个是作用于最外层的对象,描述这个对象的id,以便支持一个对象可以针对不同导出做出不同处理

注解中的ID的用法

这个ID算是一个比较独特的例子,比如

```
@ExcelTarget("teacherEntity")
public class TeacherEntity implements java.io.Serializable {
    /** name */
    @Excel(name = "主讲老师_teacherEntity,代课老师_absent", orderNum = "1",
mergeVertical = true,needMerge=true,isImportField = "true_major,true_absent")
    private String name;
```

这里的@ExcelTarget 表示使用teacherEntity这个对象是可以针对不同字段做不同处理

同样的ExcelEntity 和ExcelCollection 都支持这种方式

当导出这对象时,name这一列对应的是主讲老师,而不是代课老师还有很多字段都支持这种做法

@Excel

这个是必须使用的注解,如果需求简单只使用这一个注解也是可以的,涵盖了常用的Excel需求,需要大家熟悉这个功能,主要分为基础,图片处理,时间处理,合并处理几块,name_id是上面讲的id用法,这里就不累言了

属性	类型	默认值	功能
name	String	null	列名,支持name_id
needMerge	boolean	fasle	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)
orderNum	String	"0"	列的排序,支持name_id
replace	String[]	{}	值得替换 导出是{a_id,b_id} 导入反过来
savePath	String	"upload"	导入文件保存路径,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/

type	int	1	导出类型 1 是文本 2 是图片,3 是函数 ,10 是数字 默认是文本
width	double	10	列宽
height	double	10	**列高,后期打算统一使用 @ExcelTarget的 height,这个会被废弃,注意**
isStatistics	boolean	fasle	自动统计数据,在追加一行统计,把所有数据都和输出 ** 这个处理会吞没异常,请注意这一点**
isHyperlink	boolean	false	超链接,如果是需要实现接口返回对象
isImportField	boolean	true	校验字段,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,读取失败,支持 name_id
exportFormat	String	""	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
importFormat	String	""	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

format	String	""	时间格式,相当于同时设置了 exportFormat 和 importFormat
databaseFormat	String	"yyyyMMddHHmmss"	导出时间设置,如果字段是Date类型则不需要设置 数据库如果是string 类型,这个需要设置这个数据库格式,用以转换时间格式输出
numFormat	String	""	数字格式化,参数是Pattern,使用的对象是DecimalFormat
imageType	int	1	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
suffix	String	""	文字后缀,如% 90 变成90%
isWrap	boolean	true	是否换行 即支持\n
mergeRely	int[]	{}	合并单元格依赖关系,比如第二列合并是基于第一列 则{0}就可以了
mergeVertical	boolean	fasle	纵向合并内容相同的单元格
fixedIndex	int	-1	对应excel的列,忽略名字
isColumnHidden	boolean	false	导出隐藏列

@ExcelTarget

限定一个到处实体的注解,以及一些通用设置,作用于最外面的实体

属性	类型	默认值	功能
value	String	null	定义ID
height	double	10	设置行高
fontSize	short	11	设置文字大小

@ExcelEntity

标记是不是导出excel 标记为实体类,一遍是一个内部属性类,标记是否继续穿透,可以自定义内部id

属性	类型	默认值	功能
id	String	null	定义ID

@ExcelCollection

一对多的集合注解,用以标记集合是否被数据以及集合的整体排序

属性	类型	默认值	功能
id	String	null	定义ID
name	String	null	定义集合列名,支持nanm_id
orderNum	int	0	排序,支持name_id
type	Class<?>	ArrayList.class	导入时创建对象使用

@ExcelIgnore

忽略这个属性,多使用需循环引用中,无需多解释吧^^

2.3 注解导出,导入

2.3.1 对象定义

注解介绍了这么多,大家基本上也了解我们的注解是如何定义Excel的了吧,下面我们来跟着路飞实战吧

这天老师吧路飞叫到了办公室,让给给老师实现一个报表的需求,就是从教育平台把某个班级的人员导出来

需求是,导出我们班的所有学生的姓名,性别,出生日期,进校日期

正巧路飞刚看到Easypo,就打算用Easypoi来实现,实现方法如下:

首先定义一个我们导出的对象,**为了节省篇幅,统一忽略getter,setter**

```
public class StudentEntity implements java.io.Serializable {
   /**
    * id
    */
   private String        id;
   /**
    * 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_st", width = 20)
   private Date          birthday;

   @Excel(name = "进校日期", databaseFormat = "yyyyMMddHHmmss", format =
"yyyy-MM-dd")
   private Date registrationDate;

}
```

这里设置我们的4列分别是学生姓名,学生性别,出生日期,进校日期

其中学生姓名定义了我们的列的行高,学生性别因为我们基本上都是存在数据库都是数字所以我们转换下,两个日期我们都是进行了格式化输出了,这样我们就完成了业务对我们Excel的样式需求,后面只有把这个学生列表输出就可以了

生成Excel代码如下

```
Workbook workbook = ExcelExportUtil.exportExcel(new ExportParams("计算机一班学生
","学生"),
          StudentEntity .class, list);
```

这样我们就得到的一个java中的Excel,然后把这个输出就得到我们的Excel了

https://static.oschina.net/uploads/space/2017/0622/212811*uh7e*1157922.png

图片地址：https://static.oschina.net/uploads/space/2017/0622/212811_uh7e_1157922.png

2.3.2 集合定义

路飞很快的完成了老师的任务,花了也就是喝杯茶的时间,就交差了,但过了一会就又被老师叫去了,让他给出一个某个班级选择选择某些课的学生以及对应的老师

路飞又很快的想到了Easypoi,其中有一对多的导出,这不正是一对多的体现吗,然后他继续定义实体:

一个课程对应一个老师

一个课程对应N个学生

课程的实体

```
 @ExcelTarget("courseEntity")
 public class CourseEntity implements java.io.Serializable {
    /** 主键 */
    private String        id;
    /** 课程名称 */
    @Excel(name = "课程名称", orderNum = "1", width = 25)
    private String        name;
    /** 老师主键 */
    @ExcelEntity(id = "absent")
    private TeacherEntity mathTeacher;

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

教师的实体

```
@ExcelTarget("teacherEntity")
public class TeacherEntity implements java.io.Serializable {
    private String id;
    /** name */
    @Excel(name = "主讲老师_major,代课老师_absent", orderNum = "1", isImportField =
"true_major,true_absent")
    private String name;
```

这里在课程这个实体里面就完成了一堆多的导出,达到了我们基础需求,同时使用了orderNum对我们的列进行了排序,满足老师的需求,导出代码如下

```
 Workbook workbook = ExcelExportUtil.exportExcel(new ExportParams("2412312", "测
试", "测试"),
            CourseEntity.class, list);
```

这样我们就完成了老师的需求,效果如图2.3.2-1

但是课程名和代课老师没有合并,不太美观

路飞又果断给课程名称和代课老师加了needMerge = true的属性,就可以完成单元格的合并

```
    /** 课程名称 */
     @Excel(name = "课程名称", orderNum = "1", width = 25,needMerge = true)
     private String        name;

    //------------------------------
    /** name */
     @Excel(name = "主讲老师_major,代课老师_absent", orderNum = "1",needMerge =
true, isImportField = "true_major,true_absent")
```

效果如图2.3.2-2

到这里,路飞就完美的完成了老师的任务,快乐的去交差了

图片地址
：https://static.oschina.net/uploads/space/2017/0622/221100_MD8y_1157922.png

图片地址：https://static.oschina.net/uploads/space/2017/0622/222202_217m_1157922.png

2.3.3 图片的导出

在日常运作中不可避免的会遇到图片的导入导出,这里提供了两种类型的图片导出方式

```
@Excel(name = "公司LOGO", type = 2 ,width = 40 , height = 20,imageType = 1)
    private String companyLogo;
```

1. 表示type =2 该字段类型为图片,imageType=1 (默认可以不填),表示从file读取,字段类型是个字符串类型

可以用相对路径也可以用绝对路径,绝对路径优先依次获取

```
@Excel(name = "公司LOGO", type = 2 ,width = 40 , height = 20,imageType = 1)
    private byte[] companyLogo;
```

2.表示type =2 该字段类型为图片,imageType=2 ,表示从数据库或者已经读取完毕,字段类型是个字节数组

直接使用

同时,image 类型的cell最好设置好宽和高,**会百分百缩放到cell那么大,不是原尺寸,这里注意下**

效果如下

```
List<CompanyHasImgModel> list;

    @Before
```

```
    public void initData() {
        list = new ArrayList<CompanyHasImgModel>();
        list.add(new CompanyHasImgModel("百度", "imgs/company/baidu.png", "北京市海淀区西北旺东路10号院百度科技园1号楼"));
        list.add(new CompanyHasImgModel("阿里巴巴", "imgs/company/ali.png", "北京市海淀区西北旺东路10号院百度科技园1号楼"));
        list.add(new CompanyHasImgModel("Lemur", "imgs/company/lemur.png", "亚马逊热带雨林"));
        list.add(new CompanyHasImgModel("一众", "imgs/company/one.png", "山东济宁俺家"));


    }

    @Test
    public void exportCompanyImg() throws Exception {

        File savefile = new File("D:/excel/");
        if (!savefile.exists()) {
            savefile.mkdirs();
        }
        Workbook workbook = ExcelExportUtil.exportExcel(new ExportParams(), CompanyHasImgModel.class, list);
        FileOutputStream fos = new FileOutputStream("D:/excel/ExcelExportHasImgTest.exportCompanyImg.xls");
        workbook.write(fos);
        fos.close();
    }
```

运行效果

图片地址：https://static.oschina.net/uploads/space/2017/0825/144432_tkYG_1157922.png

2.3.4 Excel导入介绍

有导出就有导入,基于注解的导入导出,配置配置上是一样的,只是方式反过来而已,比如类型的替换 导出的时候是1替换成男,2替换成女,导入的时候则反过来,男变成1 ,女变成2,时间也是类似

导出的时候date被格式化成 2017-8-25 ,导入的时候2017-8-25被格式成date类型

下面说下导入的基本代码,注解啥的都是上面讲过了,这里就不累赘了

```
  @Test
    public void test2() {
        ImportParams params = new ImportParams();
```

```
        params.setTitleRows(1);
        params.setHeadRows(1);
        long start = new Date().getTime();
        List<MsgClient> list = ExcelImportUtil.importExcel(
          new File(PoiPublicUtil.getWebRootPath("import/ExcelExportMsgClient.xlsx")),
           MsgClient.class, params);
        System.out.println(new Date().getTime() - start);
        System.out.println(list.size());
        System.out.println(ReflectionToStringBuilder.toString(list.get(0)));
    }
```

基本是写法也很简单,ImportParams 参数介绍下

属性	类型	默认值	功能
titleRows	int	0	表格标题行数,默认0
headRows	int	1	表头行数,默认1
startRows	int	0	字段真正值和列标题之间的距离 默认0
keyIndex	int	0	***主键设置,如何这个cell没有值,就跳过或者认为这个是list的下面的值*** 这一列必须有值,不然认为这列为无效数据
startSheetIndex	int	0	开始读取的sheet位置,默认为0
sheetNum	int	1	上传表格需要读取的sheet 数量,默认为1
needSave	boolean	false	是否需要保存上传的Excel
needVerfiy	boolean	false	是否需要校验上传的Excel

saveUrl	String	"upload/excelUpload"	保存上传的Excel目录,默认是 如TestEntity这个类保存路径就是 upload/excelUpload/Test/yyyyMMddHHmss_***** 保存名称上传时间_五位随机数
verifyHanlder	IExcelVerifyHandler	null	校验处理接口,自定义校验
lastOfInvalidRow	int	0	最后的无效行数,不读的行数
readRows	int	0	手动控制读取的行数
importFields	String[]	null	导入时校验数据模板,是不是正确的Excel
keyMark	String	":"	Key-Value 读取标记,以这个为Key,后面一个Cell 为Value,多个改为ArrayList
readSingleCell	boolean	false	按照Key-Value 规则读取全局扫描Excel,但是跳过List读取范围提升性能 仅仅支持titleRows + headRows + startRows 以及lastOfInvalidRow

dataHanlder	IExcelDataHandler	null	数据处理接口,以此为主 ,replace,format都在这后面

2.3.5 Excel导入小功能

1. 读取指定的sheet

比如要读取上传得第二个sheet 那么需要把startSheetIndex = 1 就可以了

1. 读取几个sheet

比如读取前2个sheet,那么 sheetNum=2 就可以了

1. 读取第二个到第五个sheet

设置 startSheetIndex = 1 然后sheetNum = 4

1. 读取全部的sheet

sheetNum 设置大点就可以了

1. 保存Excel

设置 needVerfiy = true,默认保存的路径为upload/excelUpload/Test/yyyyMMddHHmss* *保存名称上传时间*五位随机数

如果自定义路径 修改下saveUrl 就可以了,同时saveUrl也是图片上传时候的保存的路径

1. 判断一个Excel是不是合法的Excel

importFields 设置下值,就是表示表头必须至少包含的字段,如果缺一个就是不合法的excel,不导入

2.3.6 图片的导入

有图片的导出就有图片的导入,导入的配置和导出是一样的,但是需要设置保存路径

1.设置保存路径saveUrl 默认为"upload/excelUpload"

可以手动修改 ImportParams 修改下就可以了

```
@Test
  public void test() {
     try {
        ImportParams params = new ImportParams();
        params.setNeedSave(true);
        List<CompanyHasImgModel> result = ExcelImportUtil.importExcel(
              new File(PoiPublicUtil.getWebRootPath("import/imgexcel.xls")),
              CompanyHasImgModel.class, params);
        for (int i = 0; i < result.size(); i++) {
           System.out.println(ReflectionToStringBuilder.toString(result.get(i)));
        }
        Assert.assertTrue(result.size() == 4);
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

导入日志

```
16:35:43.081 [main] DEBUG c.a.e.e.imports.ExcelImportServer - Excel import start ,class is class cn.afterturn.easypoi.test.entity.img.CompanyHasImgModel
16:35:43.323 [main] DEBUG c.a.e.e.imports.ExcelImportServer -  start to read excel by is ,startTime is 1503650143323
16:35:43.344 [main] DEBUG c.a.e.e.imports.ExcelImportServer -  end to read excel by is ,endTime is 1503650143344
16:35:43.429 [main] DEBUG c.a.e.e.imports.ExcelImportServer -  end to read excel list by pos ,endTime is 1503650143429
cn.afterturn.easypoi.test.entity.img.CompanyHasImgModel@1b083826[companyName=百度
,companyLogo=upload/CompanyHasImgModel/pic88273295062.PNG,companyAddr=北京市海淀区西北旺东路10号院百度科技园1号楼]
cn.afterturn.easypoi.test.entity.img.CompanyHasImgModel@105fece7[companyName=阿里巴巴
,companyLogo=upload/CompanyHasImgModel/pic22507938183.PNG,companyAddr=北京市海淀区西北旺东路10号院百度科技园1号楼]
cn.afterturn.easypoi.test.entity.img.CompanyHasImgModel@3ec300f1[companyName=Lemur,companyLogo=upload/CompanyHasImgModel/pic86390457892.PNG,companyAddr=亚马逊热带雨林]
cn.afterturn.easypoi.test.entity.img.CompanyHasImgModel@482cd91f[companyName=一众
,companyLogo=upload/CompanyHasImgModel/pic69566571093.PNG,companyAddr=山东济宁俺家]
```

图片地址：https://static.oschina.net/uploads/space/2017/0825/163643_LlQg_1157922.png

2.3.7 Excel多Sheet导出

目前单Sheet和单Class的方式比较多，对于多Sheet的方式还是一片空白，这里做一下说明：

导出基本采用ExportParams 这个对象，进行参数配置；

我们需要进行多Sheet导出，那么就需要定义一个基础配置对象

```
public class ExportView {

public ExportView(){
```

```
}


private ExportParams exportParams;
private List<?> dataList;
private Class<?> cls;

public ExportParams getExportParams() {
return exportParams;
}
public void setExportParams(ExportParams exportParams) {
this.exportParams = exportParams;
}

public Class<?> getCls() {
return cls;
}
public void setCls(Class<?> cls) {
this.cls = cls;
}
public List<?> getDataList() {
return dataList;
}
public void setDataList(List<?> dataList) {
this.dataList = dataList;
}


public ExportView(Builder builder) {
this.exportParams = builder.exportParams;
this.dataList = builder.dataList;
this.cls = builder.cls;
}

public static class Builder {
private ExportParams exportParams=null;
private List<?> dataList=null;
private Class<?> cls=null;

public Builder() {
```

```
}
public Builder exportParams(ExportParams exportParams) {
this.exportParams = exportParams;
return this;
}

public Builder dataList(List<?> dataList) {
this.dataList = dataList;
return this;
}
public Builder cls(Class<?> cls) {
this.cls = cls;
return this;
}

public ExportView create() {
return new ExportView(this);
}
}



}
```

对象主要有三个属性：

// 该注解配置的导出属性

1. ExportParams exportParams

// 对应注解 class 实例对象的数据集合

1. List<?> dataList

// 对应注解的 class

1. Class<?> cls

这里没有用泛型，因为多Sheet导出时，会引用到不同的注解对象；

定义基础配置的集合

```
public class ExportMoreView {
private List<ExportView> moreViewList=Lists.newArrayList();

public List<ExportView> getMoreViewList() {
return moreViewList;
}

public void setMoreViewList(List<ExportView> moreViewList) {
```

```
this.moreViewList = moreViewList;
}
}
```

最后在实现调用的方法中，对整个集合进行配置和解析

```
List<Map<String, Object>> exportParamList=Lists.newArrayList();
//该行主要用于获取业务数据，请根据具体的情况进行修改和调整
ExportMoreView
moreView=this.getBaseTransferService().mergeExportView(templateTypeCode);
        //迭代导出对象，将对应的配置信息写入到实际的配置中
for(ExportView view:moreView.getMoreViewList()){
Map<String, Object> valueMap=Maps.newHashMap();
valueMap.put(NormalExcelConstants.PARAMS,view.getExportParams());
valueMap.put(NormalExcelConstants.DATA_LIST,view.getDataList());
valueMap.put(NormalExcelConstants.CLASS,view.getCls());
exportParamList.add(valueMap);
}
     //实现导出配置
modelMap.put(NormalExcelConstants.FILE_NAME,new
DateTime().toString("yyyyMMddHHmmss"));
    //将转换完成的配置接入到导出中
modelMap.put(NormalExcelConstants.MAP_LIST,exportParamList);
return NormalExcelConstants.JEECG_EXCEL_VIEW;
```

如果不是采用的MVC的方式，请将转换的配置采用以下的方式实现：

图片地址：https://static.oschina.net/uploads/space/2017/1128/111045_9s7X_2343396.png

2.4 注解变种-更自由的导出

这天老师又把路飞喊道的办公室,要求路飞导出班级学生的整体信息

```
    @Excel(name = "学生姓名", height = 20, width = 30, isImportField = "true_st")
    private String        name;
    @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_st", width = 20)
    private Date          birthday;
    @Excel(name = "进校日期", databaseFormat = "yyyyMMddHHmmss", format =
```

```
"yyyy-MM-dd")
    private Date registrationDate;
```

路飞飞快的用到上面的学到的知识搞定了,这这时有一个老师把路飞叫去,说想要导出一个不要出生日期的Excel,感觉用户需求很无奈,路飞又造两个一个bean,把这个注解去掉了,来导出

```
    @Excel(name = "学生姓名", height = 20, width = 30, isImportField = "true_st")
    private String        name;
    @Excel(name = "学生性别", replace = { "男_1", "女_2" }, suffix = "生", isImportField =
"true_st")
    private int           sex;
    @Excel(name = "进校日期", databaseFormat = "yyyyMMddHHmmss", format =
"yyyy-MM-dd")
    private Date registrationDate;
```

虽然解决了老师的需求,但这个并不是一个完美的解决方案,下面介绍一个更自由的解决方案

注解的导出,规定我们必须把model写好,并且注解写好,每次导出的Excel都是固定的,无法动态控制导出的列,虽然可以通过id来处理一个案例,但是自由度远远不够,这里介绍个变种支持,基本支持注解所有的功能

基于List<ExcelExportEntity> 的导出,ExcelExportEntity是注解经过处理翻译成的实体类,两者几乎是一对的,所以如果我们要动态自定义导出列,我们只要动态拼装ExcelExportEntity就可以了

下面我们看下这个类

```
/**
     * 如果是MAP导出,这个是map的key
     */
    private Object                key;

    private double                width          = 10;

    private double                height         = 10;

    /**
     * 图片的类型,1是文件,2是数据库
     */
    private int                   exportImageType = 0;

    /**
     * 排序顺序
     */
    private int                   orderNum        = 0;

    /**
```

```
    * 是否支持换行
    */
    private boolean                isWrap;

    /**
    * 是否需要合并
    */
    private boolean                needMerge;
    /**
    * 单元格纵向合并
    */
    private boolean                mergeVertical;
    /**
    * 合并依赖
    */
    private int[]                  mergeRely;
    /**
    * 后缀
    */
    private String                 suffix;
    /**
    * 统计
    */
    private boolean                isStatistics;

    private String                 numFormat;

    private List<ExcelExportEntity> list;
```

基本上是和注解对应的, **List<ExcelExportEntity> list 这个是对应的一对多的导出,相当于集合,其他基本上都是和注解保持一致**

下面给出正常的demo

```
public void test() {
        try {
            List<ExcelExportEntity> entity = new ArrayList<ExcelExportEntity>();
//构造对象等同于@Excel
            ExcelExportEntity excelentity = new ExcelExportEntity("姓名", "name");
            excelentity.setNeedMerge(true);
            entity.add(excelentity);
            entity.add(new ExcelExportEntity("性别", "sex"));
            excelentity = new ExcelExportEntity(null, "students");
```

```
            List<ExcelExportEntity> temp = new ArrayList<ExcelExportEntity>();
            temp.add(new ExcelExportEntity("姓名", "name"));
            temp.add(new ExcelExportEntity("性别", "sex"));
//构造List等同于@ExcelCollection
            excelentity.setList(temp);
            entity.add(excelentity);
            List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();
//把我们构造好的bean对象放到params就可以了
            Workbook workbook = ExcelExportUtil.exportExcel(new ExportParams("测试",
"测试"), entity,
                list);
            FileOutputStream fos = new
FileOutputStream("D:/excel/ExcelExportForMap.tt.xls");
            workbook.write(fos);
            fos.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

路飞想到了这个方案,并且用上面做了测试可以完美解决所以他把之前的代码改为了(**代码有删减,基本上都是和注解对应的**)

```
List<ExcelExportEntity> beanList = new ArrayList<ExcelExportEntity>();
beanList .add(new ExcelExportEntity(new ExcelExportEntity("学生姓名", "name"));
beanList .add(new ExcelExportEntity("学生性别", "sex"));
beanList .add(new ExcelExportEntity("进校日期", "registrationDate"));
if(needBirthday()){
  beanList .add(new ExcelExportEntity("出生日期", "birthday"));
}
  Workbook workbook = ExcelExportUtil.exportExcel(new ExportParams("测试", "测试"),
beanList ,
                list);
```

用同一套代买完美了支持了老师的需求,心满意足的回宿舍了^^

2.5 Map导入,自由发挥

这天,老师把路飞叫到办公室,总是被叫,能者的悲哀啊,让他临时导入一批数据,到数据库,但是中间需要处理一些字段逻辑没办法直接导入到数据库,

这时路飞首先想到构造一个bean然后标记注解,导入处理对象,但是想想一次的对象太过于浪费,不如用map试试,获取map处理map也是一样的

导入的逻辑就变成了

```
        ImportParams params = new ImportParams();
        params.setDataHanlder(new MapImportHanlder());
        long start = new Date().getTime();
        List<Map<String, Object>> list = ExcelImportUtil.importExcel(
            new File(PoiPublicUtil.getWebRootPath("import/check.xls")), Map.class,
params);
```

导入后,处理每个map,然后入库完美的解决了老师的需求,简单更快捷,和bean导入基础没有区别,省去了bean的构造时间

PS:这个作者也只是在临时方案中或者一次性活当中使用,一般还是推荐注解这种方式,拥有更高的代码阅读性

!!!测试了时间的,最好导入使用文本格式,可以获取时间格式可能无法获取

2.6 Excel的样式自定义

"路飞,来办公室一趟",就这样路飞又被叫到了办公室,这次老师的需求是,想要一个漂亮点的Excel,希望路飞可以点缀下Excel,思来想去还是需要用poi的style来解决,但是如果每个都写style是不是太麻烦,而且Excel的styler数量是有限制的,这里就需要尽量复用已经创造的style,看看之前的Excel表格,大体上可以分为[标题,表头,表体],那可以说的就是创建一个接口每次调用这三个接口就可以了不说干就干

```
public interface IExcelExportStyler {
    /**
     * 列表头样式
     * @param headerColor
     * @return
     */
    public CellStyle getHeaderStyle(short headerColor);
    /**
     * 标题样式
     * @param color
     * @return
     */
    public CellStyle getTitleStyle(short color);
    /**
     * 获取样式方法
     * @param Parity
     * @param entity
     * @return
     */
    public CellStyle getStyles(boolean Parity, ExcelExportEntity entity);
}
```

实现类尽量复用已经创建的Styler,切记

这样路飞先造了一个带边框的styler ,**ExcelExportStylerBorderImpl**

效果如下

图片地址：https://static.oschina.net/uploads/img/201707/11203544_eOw4.png

然后路飞又手痒写了个带换行颜色的 **ExcelExportStylerColorImpl**

效果如下

图片地址：https://static.oschina.net/uploads/img/201707/11203718_miwu.png

客官看到这里应该就大体理解了我们的实现方法了吧,

最后路飞实现了一个复杂的按照老师要求的样式交差了

---

styler接口用法

上面两个表头和标题样式不用解释

后面这个是传入当前列的以及奇偶行,用户可以根据需求实现业务,包括去掉Excel的小箭头(也就是设置数字为数字格式的Cell),完成居中,字体等等各式各样的需求

但是这里无法实现特别没的Excel,如果有这种需求可以使用模板来实现,在Excel点点就可以完美实现

2.7 如何自定义数据处理

导入导出总有一些自定义格式转换,EasyPoi虽然定义了很多服务,但是也无法满足所有客户的需求,这个时候就需要咱们自己定义数据处理

EasyPoi提供了

```
/**
 * Excel 导入导出 数据处理接口
 *
 * @author JueYue
 *  2014年6月19日 下午11:59:45
 */
public interface IExcelDataHandler<T> {

    /**
     * 导出处理方法
     *
     * @param obj
     *            当前对象
     * @param name
     *            当前字段名称
     * @param value
     *            当前值
     * @return
```

```
     */
    public Object exportHandler(T obj, String name, Object value);

    /**
     * 获取需要处理的字段,导入和导出统一处理了, 减少书写的字段
     *
     * @return
     */
    public String[] getNeedHandlerFields();

    /**
     * 导入处理方法 当前对象,当前字段名称,当前值
     *
     * @param obj
     *            当前对象
     * @param name
     *            当前字段名称
     * @param value
     *            当前值
     * @return
     */
    public Object importHandler(T obj, String name, Object value);

    /**
     * 设置需要处理的属性列表
     * @param fields
     */
    public void setNeedHandlerFields(String[] fields);

    /**
     * 设置Map导入,自定义 put
     * @param map
     * @param originKey
     * @param value
     */
    public void setMapValue(Map<String, Object> map, String originKey, Object value);

    /**
     * 获取这个字段的 Hyperlink ,07版本需要,03版本不需要
     * @param creationHelper
     * @param obj
     * @param name
```

```
     * @param value
     * @return
     */
    public Hyperlink getHyperlink(CreationHelper creationHelper, T obj, String name,
Object value);

}
```

简单的使用方法如下

```
 CourseHandler hanlder = new CourseHandler();
        hanlder.setNeedHandlerFields(new String[] { "课程名称" });
        exportParams.setDataHandler(hanlder);
```

我们自己实现以下这个类,也可以继承ExcelDataHandlerDefaultImpl ,避免实现多余的接口

setNeedHandlerFields 这个是需要我们自己处理的字段,需要手动设置

让我们看一个demo

```
public class MapImportHandler extends ExcelDataHandlerDefaultImpl<Map<String,
Object>> {

    @Override
    public void setMapValue(Map<String, Object> map, String originKey, Object value)
{
        if (value instanceof Double) {
            map.put(getRealKey(originKey), PoiPublicUtil.doubleToString((Double) value));
        } else {
            map.put(getRealKey(originKey), value != null ? value.toString() : null);
        }
    }

    private String getRealKey(String originKey) {
        if (originKey.equals("交易账户")) {
            return "accountNo";
        }
        if (originKey.equals("姓名")) {
            return "name";
        }
        if (originKey.equals("客户类型")) {
            return "type";
        }
        return originKey;
    }
```

```
}
```

这里我们在map导入的时候把map的key给转了,从中文转为习惯的英文

2.8 Excel导入校验

校验,是一个不可或缺的功能,现在java校验主要是JSR 303 规范,实现方式主流的有两种

- Hibernate Validator
- Apache Commons Validator

这个EasyPoi没有限制,只要你防止一个实现丢到maven中就可以了,但是Hibernate Validator用的貌似多一些

之前的版本EasyPoi有定义自己的实现,但是后来抛弃了,没有必要造这种轮子,这个了功能已经够丰富了

使用

对象

EasyPoi的校验使用也很简单,对象上加上通用的校验规则或者这定义的这个看你用的哪个实现

然后params.setNeedVerfiy(true);配置下需要校验就可以了

看下具体的代码

```
 /**
     * Email校验
     */
    @Excel(name = "Email", width = 25)
    private String email;
    /**
     * 最大
     */
    @Excel(name = "Max")
    @Max(value = 15,message = "max 最大值不能超过15" ,groups =
{ViliGroupOne.class})
    private int    max;
    /**
     * 最小
     */
    @Excel(name = "Min")
    @Min(value = 3, groups = {ViliGroupTwo.class})
    private int    min;
    /**
     * 非空校验
     */
```

```
    @Excel(name = "NotNull")
    @NotNull
    private String notNull;
    /**
     * 正则校验
     */
    @Excel(name = "Regex")
    @Pattern(regexp = "[\u4E00-\u9FA5]*", message = "不是中文")
    private String regex;
```

这里的校验规则都是JSR 303 的,使用方式也是的,这里就不做解释了

然后使用方式是

```
@Test
    public void basetest() {
        try {
            ImportParams params = new ImportParams();
            params.setNeedVerfiy(true);
            params.setVerfiyGroup(new Class[]{ViliGroupOne.class});
            ExcelImportResult<ExcelVerifyEntity> result =
ExcelImportUtil.importExcelMore(
                new File(PoiPublicUtil.getWebRootPath("import/verfiy.xlsx")),
                ExcelVerifyEntity.class, params);
            FileOutputStream fos = new
FileOutputStream("D:/excel/ExcelVerifyTest.basetest.xlsx");
            result.getWorkbook().write(fos);
            fos.close();
            for (int i = 0; i < result.getList().size(); i++) {
                System.out.println(ReflectionToStringBuilder.toString(result.getList().get(i)));
            }
            Assert.assertTrue(result.getList().size() == 1);
            Assert.assertTrue(result.isVerfiyFail());
        } catch (Exception e) {
            LOGGER.error(e.getMessage(),e);
        }
    }
```

***ExcelImportResult ***

我们会返回一个ExcelImportResult 对象,比我们平时返回的list多了一些元素

```
 /**
     * 结果集
     */
```

```
    private List<T>   list;

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

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

一个是集合,是一个是是否有校验失败的数据,一个原本的文档,但是在文档后面追加了错误信息

注意,这里的list,有两种返回

- 一种是只返回正确的数据
- 一种是返回全部的数据,但是要求这个对象必须实现IExcelModel接口,如下

IExcelModel

```
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;
    }

}
```

***IExcelDataModel**

获取错误数据的行号

```
public interface IExcelDataModel {

    /**
     * 获取行号
```

```
     * @return
     */
    public int getRowNum();

    /**
     *  设置行号
     * @param rowNum
     */
    public void setRowNum(int rowNum);

}
```

需要对象实现这个接口

每行的错误数据也会填到这个错误信息中,方便用户后面自定义处理

看下代码

```
   @Test
    public void baseModetest() {
        try {
            ImportParams params = new ImportParams();
            params.setNeedVerfiy(true);
            ExcelImportResult<ExcelVerifyEntityOfMode> result =
ExcelImportUtil.importExcelMore(
                    new FileInputStream(new
File(PoiPublicUtil.getWebRootPath("import/verfiy.xlsx"))),
                ExcelVerifyEntityOfMode.class, params);
            FileOutputStream fos = new FileOutputStream("D:/excel/baseModetest.xlsx");
            result.getWorkbook().write(fos);
            fos.close();
            for (int i = 0; i < result.getList().size(); i++) {
                System.out.println(ReflectionToStringBuilder.toString(result.getList().get(i)));
            }
            Assert.assertTrue(result.getList().size() == 4);
        } catch (Exception e) {
            LOGGER.error(e.getMessage(),e);
        }
    }
```

IExcelVerifyHandler

加入上面的不满足你,你可以用接口实现自己的校验规则,比如唯一性校验,等等,需要返回错误信息和成功与否

```
public interface IExcelVerifyHandler<T> {
```

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

}
```

调用顺序是先通用的,再接口,到这里校验的就完整了,下面给大家看下错误的excel返回

图片地址：https://static.oschina.net/uploads/img/201709/18141356_UPlR.png

2.9 Excel 大批量读取

2.10 Excel大数据导出

大数据导出是当我们的导出数量在几万,到上百万的数据时,一次从数据库查询这么多数据加载到内存然后写入会对我们的内存和CPU都产生压力,这个时候需要我们像分页一样处理导出分段写入Excel缓解Excel的压力 EasyPoi提供的是两个方法 ***强制使用 xssf版本的Excel** *

```
  /**
     * @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 exportBigExcel(ExportParams entity, Class<?> pojoClass,
                                          Collection<?> dataSet) {
        ExcelBatchExportServer batachServer = ExcelBatchExportServer
            .getExcelBatchExportServer(entity, pojoClass);
        return batachServer.appendData(dataSet);
    }

    public static void closeExportBigExcel() {
        ExcelBatchExportServer batachServer =
ExcelBatchExportServer.getExcelBatchExportServer(null,
            null);
```

```
        batachServer.closeExportBigExcel();
    }
```

添加数据和关闭服务,关闭服务不是必须的,可以调也可以不掉

我们只需要for循环写入Excel就可以了

```
@Test
    public void bigDataExport() throws Exception {

        List<MsgClient> list = new ArrayList<MsgClient>();
        Workbook workbook = null;
        Date start = new Date();
        ExportParams params = new ExportParams("大数据测试", "测试");
        for (int i = 0; i < 1000000; i++) {  //一百万数据量
            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);
            if(list.size() == 10000){
                workbook = ExcelExportUtil.exportBigExcel(params, MsgClient.class, list);
                list.clear();
            }
        }
        ExcelExportUtil.closeExportBigExcel();
        System.out.println(new Date().getTime() - start.getTime());
        File savefile = new File("D:/excel/");
        if (!savefile.exists()) {
            savefile.mkdirs();
        }
        FileOutputStream fos = new
FileOutputStream("D:/excel/ExcelExportBigData.bigDataExport.xlsx");
        workbook.write(fos);
        fos.close();
    }
```

生成的Excel数据

图片地址：https://static.oschina.net/uploads/img/201709/07174436_1jDz.png

Cpu和内存

图片地址：https://static.oschina.net/uploads/img/201709/07174540_2ic5.png

多次测试用时统计,速度还是可以接受的,^^

数据量	用时	文件大小	列数
100W	16.4s	24.3MB	5
100W	15.9s	24.3MB	5
200W	29.5s	48.5MB	5
100W	30.8s	37.8MB	10
200W	58.7s	76.1MB	10

2.11 导入获取Key-Value

from 3.0.1

工作中是否会遇到导入读取一些特定的字段比如

图片地址：https://static.oschina.net/uploads/img/201709/11093955_h1Df.png

Excel 中的委托方,代理方,日期,单号,或者尾部的身份证号,电话等等,需要我们统一入库,这些字段没有具体位置,只能特定计算

这里给出了一个全新的解决办法 key-value 导入方法

key 是要导入的字段名称比如 委托方: 就认为是一个要导入的字段,后面的一个cell就是起对应的值

比如委托方：一众科技有限公司 这样导入进去就是 key委托方,value 一众科技有限公司

示例代码

```
@Test
    public void test() {
        try {
            ImportParams params = new ImportParams();
            params.setKeyMark("：");
            params.setReadSingleCell(true);
            params.setTitleRows(7);
            params.setLastOfInvalidRow(9);
            ExcelImportResult<Map> result = ExcelImportUtil.importExcelMore(
                    new File(PoiPublicUtil.getWebRootPath("import/业务委托单.xlsx")),
                    Map.class, params);
```

```
            for (int i = 0; i < result.getList().size(); i++) {
                System.out.println(result.getList().get(i));
            }
            Assert.assertTrue(result.getList().size() == 10);
            System.out.println(result.getMap());
        } catch (Exception e) {
            LOGGER.error(e.getMessage(),e);
        }
    }
```

需要设置两个或者一个值

params.setKeyMark("："); 判断一个cell是key的规则,可以自定义,默认就是 ":"

params.setReadSingleCell(true); 是否需要读取这种单独的sql

读取完毕后,通过result.getMap() 就可以拿到自己想要的值了比如上面的Excel读取到的map就是

{境内详细收货地址、联系人、电话：=1.3112345678E10, 委托方：=一众科技有限公司, 代理方：=上海一众金融信息服务有限公司, 委托单号：=XH-HZHY-20170504, 日期：=2017.5.4, 供应商交货方式：=, 合计：=, 境内交货方式：=, 指定收货人身份证号：=3.70827198801020 99E17}

这样就比较方便的处理较为复杂的Excel导入了

2.12 groupname和ExcelEntity的name属性

之前一直没想好，双号表头如何处理数据，直到前几天突然想到了groupname这个属性，下面先介绍下这两个属性解决的问题，也是之前很多朋友问到的问题

图片地址：https://static.oschina.net/uploads/img/201710/20112857_np8y.png

这种双行的表头，之前只有在集合的模式情况下才会支持，但是很多情况都不是集合模式，也只是一列数据，

• 简单的groupname

比如这里的时间算是两个时间的聚合，单也是对象当中的元素而已，我们要导出这样的数据现在只要设置下groupname就可以了

```
@Excel(name = "电话号码", groupName = "联系方式", orderNum = "1")
    private String clientPhone = null;
    // 客户姓名
    @Excel(name = "姓名")
    private String clientName = null;
    // 备注
    @Excel(name = "备注")
    private String remark = null;
    // 生日
    @Excel(name = "出生日期", format = "yyyy-MM-dd", width = 20, groupName = "时间", orderNum = "2")
```

```
    private Date birthday = null;
    // 创建人
    @Excel(name = "创建时间", groupName = "时间", orderNum = "3")
    private String createBy = null;
```

这样就会把两个groupname合并到一起展示，使用也比较简单

• ExcelEntity 一个对象在一起

假如我们需要一个对象属性统一在一起，name我们需要设置下这个对象的name属性，并且 show=true 这两个是 且的关系

比如

```
 @Excel(name = "电话号码", groupName = "联系方式", orderNum = "1")
    private String clientPhone = null;
    @Excel(name = "姓名")
    private String clientName = null;
    @ExcelEntity(name = "学生", show = true)
    private GnStudentEntity studentEntity;
```

学生对象的内部就是普通的注解

```
 @Excel(name = "学生姓名", height = 20, width = 30, orderNum = "2")
    private String name;

    @Excel(name = "学生性别", replace = {"男_1", "女_0"}, suffix = "生", orderNum = "3")
    private int sex;

    @Excel(name = "出生日期", format = "yyyy-MM-dd", width = 20, orderNum = "4")
    private Date birthday;

    @Excel(name = "进校日期", format = "yyyy-MM-dd", orderNum = "5")
    private Date registrationDate;
```

出来的效果如下

图片地址：https://static.oschina.net/uploads/img/201710/20113530_FjBG.png

使用起来还是很简单的，**导入的话同样设置就可以获取到了**

• 排序问题

导出时，表头双行显示,聚合,排序以最小的值参与总体排序再内部排序

导出排序跟定义了annotation的字段的顺序有关 可以使用a*id,b*id来确实是否使用

优先弱与 @ExcelEntity 的name和show属性

简单说就是先排外部顺序，再排内部顺序

3. Excel 模板版

3.1 模板 指令介绍

模板是处理复杂Excel的简单方法，复杂的Excel样式，可以用Excel直接编辑，完美的避开了代码编写样式的雷区，同时指令的支持，也提了模板的有效性

下面列举下EasyPoi支持的指令以及作用，**最主要的就是各种fe的用法**

- 空格分割
- 三目运算 {{test ? obj:obj2}}
- n: 表示 这个cell是数值类型 {{n:}}
- le: 代表长度{{le:()}} 在if/else 运用{{le:() > 8 ? obj1 : obj2}}
- fd: 格式化时间 {{fd:(obj;yyyy-MM-dd)}}
- fn: 格式化数字 {{fn:(obj;###.00)}}
- fe: 遍历数据,创建row
- !fe: 遍历数据不创建row
- $fe: 下移插入,把当前行,下面的行全部下移.size()行,然后插入
- #fe: 横向遍历
- v_fe: 横向遍历值
- !if: 删除当前列 {{!if:(test)}}
- 单引号表示常量值 '' 比如'1' 那么输出的就是 1
- &NULL& 空格
-]] 换行符 多行遍历导出
- sum : 统计数据

整体风格和el表达式类似，大家应该也比较熟悉

采用的写法是{()}代表表达式，然后根据表达式里面的数据取值

关于样式问题

easypoi不会改变excel原有的样式，如果是遍历，easypoi会根据模板的那一行样式进行复制

测试项目

在cn.afterturn.easypoi.test.excel.template 这个目录下面

https://gitee.com/lemur/easypoi-test/tree/master/src/test/java/cn/afterturn/easypoi/test/excel/template(https://gitee.com/lemur/easypoi-test/tree/master/src/test/java/cn/afterturn/easypoi/test/excel/template)

3.2 基本导出

看一个常见的到处模板--专项支出用款申请书

图片地址：https://static.oschina.net/uploads/img/201709/26145656_oqzr.png

这里面有正常的标签以及$fe遍历，$fe遍历应该是使用最广的遍历，用来解决遍历后下面还有数据的处理方式

我们要生成的是这个需要一些list集合和一些单纯的数据

fe的写法 fe标志 冒号 list数据 单个元素数据（默认t，可以不写） 第一个元素

{{$fe: maplist t t.id }}

看下数据代码，主要是构造数据TemplateExportParams是主要的参数数据

```
@Test
    public void fe_map() throws Exception {
        TemplateExportParams params = new TemplateExportParams(
                "WEB-INF/doc/专项支出用款申请书_map.xls");
        Map<String, Object> map = new HashMap<String, Object>();
        map.put("date", "2014-12-25");
        map.put("money", 2000000.00);
        map.put("upperMoney", "贰佰万");
        map.put("company", "执笔潜行科技有限公司");
        map.put("bureau", "财政局");
        map.put("person", "JueYue");
        map.put("phone", "1879740****");
        List<Map<String, String>> listMap = new ArrayList<Map<String, String>>();
        for (int i = 0; i < 4; i++) {
            Map<String, String> lm = new HashMap<String, String>();
            lm.put("id", i + 1 + "");
            lm.put("zijin", i * 10000 + "");
            lm.put("bianma", "A001");
            lm.put("mingcheng", "设计");
            lm.put("xiangmumingcheng", "EasyPoi " + i + "期");
            lm.put("quancheng", "开源项目");
            lm.put("sqje", i * 10000 + "");
            lm.put("hdje", i * 10000 + "");

            listMap.add(lm);
        }
        map.put("maplist", listMap);

        Workbook workbook = ExcelExportUtil.exportExcel(params, map);
        File savefile = new File("D:/excel/");
        if (!savefile.exists()) {
            savefile.mkdirs();
        }
        FileOutputStream fos = new FileOutputStream("D:/excel/专项支出用款申请书
_map.xls");
        workbook.write(fos);
        fos.close();
    }
```

看下输出的效果

图片地址：https://static.oschina.net/uploads/img/201709/26150826_DKZQ.png

3.3 模板当中使用注解

3.4 图片导出

模板图片导出，没有注解导出图片那么容易，但也不算复杂,构建一个ImageEntity

设置下高宽，地址或者byte[]及可以了

```
ImageEntity image = new ImageEntity();
image.setHeight(200);
image.setWidth(500);
image.setUrl("imgs/company/baidu.png");
```

具体的导出代码

```
 @Test
   public void one() throws Exception {
      TemplateExportParams params = new TemplateExportParams(
            "doc/exportTemp_image.xls", true);
      Map<String, Object> map = new HashMap<String, Object>();
      // sheet 2
      map.put("month", 10);
      Map<String, Object> temp;
      for (int i = 1; i < 8; i++) {
         temp = new HashMap<String, Object>();
         temp.put("per", i * 10);
         temp.put("mon", i * 1000);
         temp.put("summon", i * 10000);
         ImageEntity image = new ImageEntity();
         image.setHeight(200);
         image.setWidth(500);
         image.setUrl("imgs/company/baidu.png");
         temp.put("image", image);
         map.put("i" + i, temp);
      }
      Workbook book = ExcelExportUtil.exportExcel(params, map);
      File savefile = new File("D:/excel/");
      if (!savefile.exists()) {
         savefile.mkdirs();
      }
```

```
        FileOutputStream fos = new FileOutputStream("D:/excel/exportTemp_image.xls");
        book.write(fos);
        fos.close();

    }
```

4. Excel<->Html

4.1 Excel 的Html预览

Excel预览，这里支持了比较简单的预览，样式也都可以转换过去，支持03 和 更高版本
使用也是简单的很ExcelXorHtmlUtil.excelToHtml(params)，也支持图片的预览，demo如下

```
  /**
     * 07 版本EXCEL预览
     */
    @RequestMapping("07")
    public void toHtmlOf07Base(HttpServletResponse response) throws IOException,
InvalidFormatException {
        ExcelToHtmlParams params = new
ExcelToHtmlParams(WorkbookFactory.create(POICacheManager.getFile("exceltohtml/t
estExportTitleExcel.xlsx")));

response.getOutputStream().write(ExcelXorHtmlUtil.excelToHtml(params).getBytes());
    }
    /**
     * 03 版本EXCEL预览
     */
    @RequestMapping("03img")
    public void toHtmlOf03Img(HttpServletResponse response) throws IOException,
InvalidFormatException {
        ExcelToHtmlParams params = new
ExcelToHtmlParams(WorkbookFactory.create(POICacheManager.getFile("exceltohtml/e
xporttemp_img.xls")),true,"yes");

response.getOutputStream().write(ExcelXorHtmlUtil.excelToHtml(params).getBytes());
    }
```

返回一个string的html界面，输出到前台就可以了

4.2 html转Excel更神奇的导出

这个是一个MM提出的需求，需求原因是，她要导出一个比较复杂的Excel，无论用模板还是注解都比较难实现，所以她想到了这个方案，然后就实现了如下的方法，我的使用方法如下

自己搞个html，然后用模板引擎，beetl，freemark等生成html，然后调用easypoi提供的方法转换成Excel，因为html的标签以及规则大家比Excel要熟悉的多，更容易编写复杂的table，然后easypoi转换成Excel再导出，麻烦了点，但是可以处理一些特定的情况，也同样生成两个版本的Excel都支持

使用demo

```
    @Test
    public void htmlToExcelByStr() throws Exception {
        StringBuilder html = new StringBuilder();
        Scanner s = new Scanner(getClass().getResourceAsStream("/html/sample.html"),
"utf-8");
        while (s.hasNext()) {
            html.append(s.nextLine());
        }
        s.close();
        Workbook workbook = ExcelXorHtmlUtil.htmlToExcel(html.toString(),
ExcelType.XSSF);
        File savefile = new File("D:\\home\\lemur");
        if (!savefile.exists()) {
            savefile.mkdirs();
        }
        FileOutputStream fos = new
FileOutputStream("D:\\home\\lemur\\htmlToExcelByStr.xlsx");
        workbook.write(fos);
        fos.close();
        workbook = ExcelXorHtmlUtil.htmlToExcel(html.toString(), ExcelType.HSSF);
        fos = new FileOutputStream("D:\\home\\lemur\\htmlToExcelByStr.xls");
        workbook.write(fos);
        fos.close();
    }

    @Test
    public void htmlToExcelByIs() throws Exception {
        Workbook workbook =
ExcelXorHtmlUtil.htmlToExcel(getClass().getResourceAsStream("/html/sample.html"),
ExcelType.XSSF);
        File savefile = new File("D:\\home\\lemur");
        if (!savefile.exists()) {
            savefile.mkdirs();
        }
```

```
        FileOutputStream fos = new
FileOutputStream("D:\\home\\lemur\\htmlToExcelByIs.xlsx");
        workbook.write(fos);
        fos.close();
        workbook =
ExcelXorHtmlUtil.htmlToExcel(getClass().getResourceAsStream("/html/sample.html"),
ExcelType.HSSF);
        fos = new FileOutputStream("D:\\home\\lemur\\htmlToExcelByIs.xls");
        workbook.write(fos);
        fos.close();
    }
```

提供了流或者字符串的入参，内部都多了缓存，多次生成不会重复解析

5. Word

5.1 word模板导出

word模板和Excel模板用法基本一致，支持的标签也是一致的，仅仅支持07版本的word也是只能生成后缀是docx的文档，poi对doc支持不好，所以这里也就懒得支持了，支持表格和图片，具体demo如下

```
/**

     * 简单导出包含图片

     */
    @Test
    public void imageWordExport() {
        Map<String, Object> map = new HashMap<String, Object>();
        map.put("department", "Easypoi");
        map.put("person", "JueYue");
        map.put("time", format.format(new Date()));
        WordImageEntity image = new WordImageEntity();
        image.setHeight(200);
        image.setWidth(500);
        image.setUrl("cn/afterturn/easypoi/test/word/img/testCode.png");
        image.setType(WordImageEntity.URL);
        map.put("testCode", image);
        try {
            XWPFDocument doc = WordExportUtil.exportWord07(
                "cn/afterturn/easypoi/test/word/doc/Image.docx", map);
```

```
            FileOutputStream fos = new FileOutputStream("D:/excel/image.docx");
            doc.write(fos);
            fos.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**

     * 简单导出没有图片和Excel

     */
    @Test
    public void SimpleWordExport() {
        Map<String, Object> map = new HashMap<String, Object>();
        map.put("department", "Easypoi");
        map.put("person", "JueYue");
        map.put("time", format.format(new Date()));
        map.put("me","JueYue");
        map.put("date", "2015-01-03");
        try {
            XWPFDocument doc = WordExportUtil.exportWord07(
                "cn/afterturn/easypoi/test/word/doc/Simple.docx", map);
            FileOutputStream fos = new FileOutputStream("D:/excel/simple.docx");
            doc.write(fos);
            fos.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

6. PDF

7. Spring MVC

7.1 View 介绍

easypoi view 项目是为了更简单的方便搭建在导出时候的操作，利用spring mvc 的view 封装，更加符合spring mvc的风格

view下面包括多个 view的实现

- EasypoiBigExcelExportView 大数据量导出
- EasypoiMapExcelView　map 列表导出
- EasypoiPDFTemplateView pdf导出
- EasypoiSingleExcelView 注解导出
- EasypoiTemplateExcelView 模板导出
- EasypoiTemplateWordView word模板导出
- MapGraphExcelView 图表导出

view的是使用方法大同小异，都有一个对应的bean，里面保护指定的参数常量

同意用modelmap.put(‘常量参数名’，‘值’)就可以，最后返回这个view名字

注解目录扫描的时候加上

cn.afterturn.easypoi.view

就可以使用了

7.2 大数据导出View的用法

EasypoiBigExcelExportView 是针对大数据量导出特定的View,在跳转到这个View的时候不需要查询数据,而且这个View自己去查询数据,用户只要实现**IExcelExportServer**接口就可以了

对应的常量类**BigExcelConstants**

```
public interface IExcelExportServer {
    /**
     * 查询数据接口
     * @param obj    查询条件
     * @param page   当前页数
     * @return
     */
    public List<Object> selectListForExcelExport(Object obj, int page);

}
```

EasypoiBigExcelExportView 判断是否还有下一页的条件是,如果selectListForExcelExport 返回null就认为是最后一页了,如果返回有数据这page+1继续查询

在我们自己的controller中

```
 @RequestMapping("load")
    public void downloadByPoiBaseView(ModelMap map, HttpServletRequest request,
                                      HttpServletResponse response) {
        ExportParams params = new ExportParams("2412312", "测试", ExcelType.XSSF);
        params.setFreezeCol(2);
        map.put(BigExcelConstants.CLASS, MsgClient.class);
```

```
        map.put(BigExcelConstants.PARAMS, params);
        //就是我们的查询参数,会带到接口中,供接口查询使用
        map.put(BigExcelConstants.DATA_PARAMS, new HashMap<String,String>());
        map.put(BigExcelConstants.DATA_INTER,excelExportServer);
        PoiBaseView.render(map, request, response,
    BigExcelConstants.EASYPOI_BIG_EXCEL_VIEW);

    }
```

我们需要把参数条件封装成map或者其他类型,上面的obj可以把参数自己转回来 参数名字 **BigExcelConstants.DATA_PARAM**

然后把实现查询的接口注入进来就可以了

map.put(BigExcelConstants.DATA_INTER,excelExportServer);

后面就和其他View一样了

7.3 注解导出View用法

注解导出的View是这个**EasypoiSingleExcelView**,其实View大家可以忽略不看，主要用到的还是他对应的bean对象

NormalExcelConstants 注解到处还比较简单，大家只要把datalist，class和params 这几个参数put下就可以了。

具体的案例

```
@RequestMapping()
    public String download(ModelMap map) {
        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);
        }
        ExportParams params = new ExportParams("2412312", "测试", ExcelType.XSSF);
        params.setFreezeCol(2);
        map.put(NormalExcelConstants.DATA_LIST, list); // 数据集合
```

```
        map.put(NormalExcelConstants.CLASS, MsgClient.class);//导出实体
        map.put(NormalExcelConstants.PARAMS, params);//参数
        map.put(NormalExcelConstants.FILE_NAME, params);//文件名称
        return NormalExcelConstants.EASYPOI_EXCEL_VIEW;//View名称

    }
```

和非View导出基本一致，只是把调用方法封装了而已，其他参数还都是一样的，具体可以看下测试项目的

EasypoiSingleExcelViewTest(https://gitee.com/lemur/easypoi-test/blob/master/src/main/java/cn/afterturn/easypoi/view/EasypoiSingleExcelViewTest.java)

7.4 注解变种Map类型的导出View

作为动态注解存在的 List<ExcelExportEntity> ，也提供的单独的View方便大家使用，**EasypoiMapExcelView**

使用方法都是一样，直接看下例子吧

```
@RequestMapping()
    public String download(ModelMap modelMap) {
        List<ExcelExportEntity> entity = new ArrayList<ExcelExportEntity>();
        ExcelExportEntity excelentity = new ExcelExportEntity("姓名", "name");
        excelentity.setNeedMerge(true);
        entity.add(excelentity);
        entity.add(new ExcelExportEntity("性别", "sex"));
        excelentity = new ExcelExportEntity(null, "students");
        List<ExcelExportEntity> temp = new ArrayList<ExcelExportEntity>();
        temp.add(new ExcelExportEntity("姓名", "name"));
        temp.add(new ExcelExportEntity("性别", "sex"));
        excelentity.setList(temp);
        entity.add(excelentity);

        List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();
        Map<String, Object> map;
        for (int i = 0; i < 10; i++) {
            map = new HashMap<String, Object>();
            map.put("name", "1" + i);
            map.put("sex", "2" + i);

            List<Map<String, Object>> tempList = new ArrayList<Map<String,
Object>>();
            tempList.add(map);
```

```
            tempList.add(map);
            map.put("students", tempList);

            list.add(map);
        }

        ExportParams params = new ExportParams("2412312", "测试", ExcelType.XSSF);
        params.setFreezeCol(2);
        modelMap.put(MapExcelConstants.MAP_LIST, list); //数据集合
        modelMap.put(MapExcelConstants.ENTITY_LIST, entity); //注解集合
        modelMap.put(MapExcelConstants.PARAMS, params);//参数
        modelMap.put(MapExcelConstants.FILE_NAME, "EasypoiMapExcelViewTest");//文
件名称
        return MapExcelConstants.EASYPOI_MAP_EXCEL_VIEW;//View名称

    }
```

具体案例参考EasypoiMapExcelViewTest(https://gitee.com/lemur/easypoi-test/blob/master/src/main/java/cn/afterturn/easypoi/view/EasypoiMapExcelViewTest.java)

7.5Excel模板导出View

模板导出提供的EasypoiTemplateExcelView以及对应的bean **TemplateExcelConstants**

案例

```
@RequestMapping()
    public String download(ModelMap modelMap) {
        Map<String, Object> map = new HashMap<String, Object>();
        TemplateExportParams params = new TemplateExportParams(
            "doc/foreach.xlsx");
        List<TemplateExcelExportEntity> list = new
ArrayList<TemplateExcelExportEntity>();

        for (int i = 0; i < 4; i++) {
            TemplateExcelExportEntity entity = new TemplateExcelExportEntity();
            entity.setIndex(i + 1 + "");
            entity.setAccountType("开源项目");
            entity.setProjectName("EasyPoi " + i + "期");
            entity.setAmountApplied(i * 10000 + "");
            entity.setApprovedAmount((i + 1) * 10000 - 100 + "");
            list.add(entity);
        }
        map.put("entitylist", list);
```

```
        map.put("manmark", "1");
        map.put("letest", "12345678");
        map.put("fntest", "12345678.2341234");
        map.put("fdtest", null);
        List<Map<String, Object>> mapList = new ArrayList<Map<String, Object>>();
        for (int i = 0; i < 1; i++) {
            Map<String, Object> testMap = new HashMap<String, Object>();

            testMap.put("id", "xman");
            testMap.put("name", "小明" + i);
            testMap.put("sex", "1");
            mapList.add(testMap);
        }
        map.put("maplist", mapList);

        mapList = new ArrayList<Map<String, Object>>();
        for (int i = 0; i < 6; i++) {
            Map<String, Object> testMap = new HashMap<String, Object>();

            testMap.put("si", "xman");
            mapList.add(testMap);
        }
        map.put("sitest", mapList);
        modelMap.put(TemplateExcelConstants.FILE_NAME, "用户信息"); //文件名
        modelMap.put(TemplateExcelConstants.PARAMS, params);//参数
        modelMap.put(TemplateExcelConstants.MAP_DATA, map);//数据
        return TemplateExcelConstants.EASYPOI_TEMPLATE_EXCEL_VIEW;//view名称

    }
```

具体案例EasypoiTemplateExcelViewTest(https://gitee.com/lemur/easypoi-test/blob/master/src/main/java/cn/afterturn/easypoi/view/EasypoiTemplateExcelViewTest.java)

7.6 PoiBaseView.render view的补救

假如因为不可抗拒或者其他神奇的原因，view导出无法使用，作者遇到过好几次了，各种神奇原因都有，提供一个统一的封装，算是一个补救措施吧

上面的modelMap写法和设置参数还是一样，最后直接输出就可以了

PoiBaseView.render(modelMap, request, response,View名称);

看个简单demo

```
@RequestMapping("load")
    public void downloadByPoiBaseView(ModelMap map, HttpServletRequest request,
                                      HttpServletResponse response) {
        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);
        }
        ExportParams params = new ExportParams("2412312", "测试", ExcelType.XSSF);
        params.setFreezeCol(2);
        map.put(NormalExcelConstants.DATA_LIST, list);
        map.put(NormalExcelConstants.CLASS, MsgClient.class);
        map.put(NormalExcelConstants.PARAMS, params);
        PoiBaseView.render(map, request, response,
NormalExcelConstants.EASYPOI_EXCEL_VIEW);

    }
```