EasyExcel

> EasyExcel相比于EasyPoi性能好一点，特别是在大数据量时候，EasyExcel的性能相较于更好一些。对于普通低数据量的导入导出EasyExcel占用资源的数量相对少一些。

1. 初体验

1.1 依赖

```
<!--目前最新的版本-->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>easyexcel</artifactId>
    <version>3.0.5</version>
</dependency>
```

1.2 注解

@ExcelProperty

- 作用： 用于指定实体类属性和表格列名称的映射关系
- 举个栗子：

```
@ExcelProperty("编号")
```

- 注解源码

```
Author: jipengfei
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
@Inherited
public @interface ExcelProperty {

    /** The name of the sheet header. ...*/
    String[] value() default {""};    用于设置表头信息（列名称）

    Index of column Read or write it on the index of column, If it's equal to -1, it's sorted by Java class.
    priority: index > order > default sort
    Returns: Index of column
    int index() default -1;        索引，用于读写排序，优先级：索引>排序>默认，

    Defines the sort order for an column. priority: index > order > default sort
    Returns: Order of column
    int order() default Integer.MAX_VALUE;    用于对于列进行排序，值越小对应列越靠前

    /** Force the current field to use this converter. ...*/
    Class<? extends Converter<?>> converter() default AutoConverter.class;    用于强迫当前属性使用转换器

    default @see com.alibaba.excel.util.TypeUtil if default is not meet you can set format
    Deprecated please use com.alibaba.excel.annotation.format.DateTimeFormat
    Returns:     Format string
    @Deprecated
    String format() default "";    用于时间、日期的格式化
}
```

@ColumnWidth

- 作用： 用于指定某个属性对应的表格列的宽度。
- 举个栗子：

```
    @ExcelProperty(value = "编号")
     @ColumnWidth(20)
     private Long id;
```

@HeadRowHeight

- 作用： 用于设置头部行高 **用于类上**
- 举个例子：

```
@HeadRowHeight()
@ContentRowHeight()
public class User implements Serializable {
    ...
}
```

@ContentRowHeight

- 作用： 用于设置表格内容每行的行高 **用于类上**

@ExcelIgnore

- 作用： 只作为一个属性标记，用于标记该属性不被导出到excel。

@DateTimeFormat

- 用于格式化日期。**注意： 此处是excel包下的**

@NumberFormat

- 标注在成员变量上，数字转换，代码中用String类型的成员变量去接收excel数字格式的数据会调用这个注解。里边的value参照java.text.DecimalFormat

@ExcelIgnoreUnannotated

- 标注在类上。

> 不标注该注解时，默认类中所有的成员变量都会参与读写，无论是否在成员变量上加了`@ExcelProperty`的注解。
> 标注该注解后，类中的成员变量如果没有标注`@ExcelProperty`注解将不会参与读写。

1.3 写操作

3.1 简单写

- 官方demo的

```
   //第一种写法
    String filePath = "C:\\Users\\ArchieSean\\Desktop\\a.xlsx";
    //doWrite会默认自动关流,这种写法导出没有表格标题
    EasyExcel.write(filePath, User.class)
        //设置写入第一个工作簿，并且工作簿命名为 ‘用户信息’
        .sheet(0, "用户信息")
        //执行写入操作，入参支持集合、lambda表达式
        .doWrite(users);
```

```
 //第二种写法
String filePath = "C:\\Users\\ArchieSean\\Desktop\\b.xlsx";
        ExcelWriter excelWriter = EasyExcel.write(filePath, User.class).build();
        //构建工作簿对象
        WriteSheet writeSheet = EasyExcel.writerSheet("模板").build();
        excelWriter.write(users, writeSheet);
        //关闭资源
        excelWriter.finish();
```

3.2 指定写

```
  EasyExcel.write(filePath, User.class)
                .includeColumnFiledNames()  //指定导出哪些属性
                .excludeColumnFiledNames()  //指定不导出哪些属性
```

3.3 分页写

```
String filePath = "C:\\Users\\ArchieSean\\Desktop\\b.xlsx";
        //构建工作簿对象，一个文件只需要构建一个
        ExcelWriter excelWriter = EasyExcel.write(filePath, User.class).build();
        //构建一个sheet，默认为第一个sheet
        WriteSheet writeSheet = EasyExcel.writerSheet("模板").build();
        //循环调用，并控制数量
         excelWriter.write(users, writeSheet);

        //手动关流
          excelWriter.finish();
```

3.4 图片导出

```
//第一种解决方案：【常用】
    //1.将图片对应的属性更改为URL
    //2.通过new的方式将string转为URL，直接写出即可
//图片的所有解决方案：
  private File file;
    private InputStream inputStream;
//imageData.setInputStream(FileUtils.openInputStream(new File(imagePath));

    /**
     * 如果string类型 必须指定转换器，string默认转换成string
     */
    @ExcelProperty(converter = StringImageConverter.class)
    private String string;  //直接写出
```

```
    private byte[] byteArray;  //FileUtils.readFileToByteArray(new
File(imagePath))
    /**
     * 根据url导出
     *
     * @since 2.1.1
     */
    private URL url;  //new URL(filePath)
```

3.5 自定义样式

样式策略: WriteCellSty类

```
// 头的策略
    WriteCellStyle headWriteCellStyle = new WriteCellStyle();
// 内容的策略
    WriteCellStyle contentWriteCellStyle = new WriteCellStyle();
```

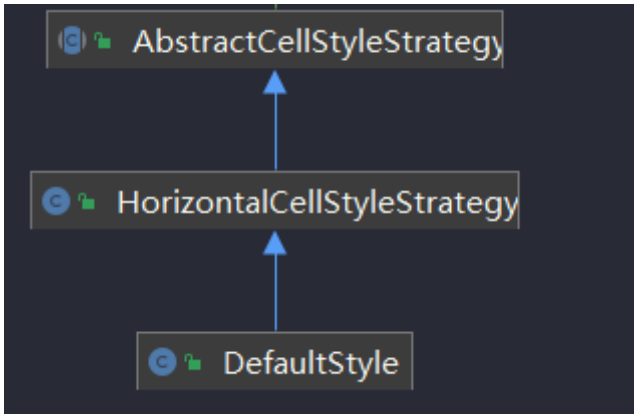
- WriteCellSty源码

```
    //用于格式化头部日期
    private DataFormatData dataFormatData;
    //设置字体相关样式（字体名称、大小、斜体、删除线、颜色、设置普通/上标/下标、下划线类型、字
符集、加粗）
    private WriteFont writeFont;
    //使用隐藏
    private Boolean hidden;
    //使用锁定
    private Boolean locked;
    //是否打开样式的“引用前缀、或者123前缀”
    private Boolean quotePrefix;
   //单元格水平对齐方式（普通/左对齐/居中/右对齐/填充/栅格/中心选择/分散）
    private HorizontalAlignment horizontalAlignment;
    //是否换行
    private Boolean wrapped;
    //设置单元格的垂直对齐方式（顶对齐/水平对齐/底部对齐/栅格/分散）
    private VerticalAlignment verticalAlignment;
    //设置单元格文本的旋转度数
    private Short rotation;
    //设置单元格中文本缩进的空格数
    private Short indent;
    //左边框类型(无边框/线/点线/双实线)
    private BorderStyle borderLeft;
    //右边框线
    private BorderStyle borderRight;
    //顶部边框线
    private BorderStyle borderTop;
    //底部边框线
    private BorderStyle borderBottom;
    //左边框的颜色 -IndexedColors
    private Short leftBorderColor;
    //右边框颜色
    private Short rightBorderColor;
    //顶部边框颜色
    private Short topBorderColor;
```

```
    //底部边框颜色
    private Short bottomBorderColor;
    //设置单元格填充图案类型
    private FillPatternType fillPatternType;
    //设置后景色填充单元格
    private Short fillBackgroundColor;
    //设置前景色填充单元格
    private Short fillForegroundColor;
    //设置单元格大小自动适应文本
    private Boolean shrinkToFit;
```

样式设置：HorizontalCellStyleStrategy

- HorizontalCellStyleStrategy的家族



- 基本使用
- 1. 源码：

```
   public HorizontalCellStyleStrategy(WriteCellStyle headWriteCellStyle,
WriteCellStyle contentWriteCellStyle) {
        this.headWriteCellStyle = headWriteCellStyle;
        if (contentWriteCellStyle != null) {
            this.contentWriteCellStyleList =
ListUtils.newArrayList(contentWriteCellStyle);
        }
    }
```

- 2. 使用：

```
 HorizontalCellStyleStrategy strategy = new HorizontalCellStyleStrategy(头部样式,
内容样式);
```

- **注意： 在pojo类上添加的列宽、行高会样式继承**

3.6 模板写出

- 将模板放在项目当中，写出时设置模板，通过withTemplate()方法设置模板。
- 举个例子

```
/**
 * 根据模板写入
 * <p>1. 创建excel对应的实体对象 参照{@link IndexData}
 * <p>2. 使用{@link ExcelProperty}注解指定写入的列
 * <p>3. 使用withTemplate 写取模板
```

```
     * <p>4. 直接写即可
     */
    @Test
    public void templateWrite() {
        //模板文件路径
        String templateFileName = TestFileUtil.getPath() + "demo" + File.separator +
    "demo.xlsx";
        //写出文件路径
        String fileName = TestFileUtil.getPath() + "templateWrite" +
    System.currentTimeMillis() + ".xlsx";
        // 这里 需要指定写用哪个class去写，然后写到第一个sheet，名字为模板 然后文件流会自动关闭
        EasyExcel.write(fileName,
    DemoData.class).withTemplate(templateFileName).sheet().doWrite(data());
    }
```

3.7 合并单元格

- 使用的类：LoopMergeStrategy
- 源码：

```
/**
 * 用于给某行某列进行横向扩充
 */
public class LoopMergeStrategy implements RowWriteHandler {
    /**
     * 迭代行
     */
    private int eachRow;
    /**
     * 扩展列
     */
    private int columnExtend;
    /**
     * 当前列
     */
    private int columnIndex;
}
```

- 使用：

```
//每两行的第一列，默认扩展1
LoopMergeStrategy loopMergeStrategy = new LoopMergeStrategy(2, 0);

//使用上边的合并策略
EasyExcel.write(fileName,
DemoData.class).registerWriteHandler(loopMergeStrategy).sheet("模板")
         .doWrite(data());
```

1.4 读操作

4.1 简单读

```
   String filePath = "C:\\Users\\ArchieSean\\Desktop\\b.xlsx";
        /*
         * 构建一个读的工作簿对象
         * @param pathName 要读的文件的路径
         * @param head 文件中每一行数据要存储到的实体的类型class
         * @param readListener  读监听器，每读一行内容，都会调用一次对象的invoke，在
invoke可以操作使用读取到的数据
         * @return Excel reader builder.
         */
        EasyExcel.read(filePath, User.class, new AnalysisEventListener<User>() {
            @Override
            public void invoke(User data, AnalysisContext context) {
                System.out.println(data.toString());
            }
            @Override
            public void doAfterAllAnalysed(AnalysisContext context) {
                System.out.println("=======解析完成===");
            }
        }).sheet(0).doRead();
```

4.2 读多个sheet

- 一次读全部sheet

```
 // 这里需要注意 DemoDataListener的doAfterAllAnalysed 会在每个sheet读取完毕后调用一次。
然后所有sheet都会往同一个DemoDataListener里面写
    EasyExcel.read(fileName, DemoData.class, new
DemoDataListener()).doReadAll();
```

- 读部分sheet

```
 ExcelReader excelReader = EasyExcel.read(fileName).build();
    // 这里为了简单 所以注册了 同样的head 和Listener 自己使用功能必须不同的Listener
    ReadSheet readSheet1 =
        EasyExcel.readSheet(0).head(DemoData.class).registerReadListener(new
DemoDataListener()).build();
    ReadSheet readSheet2 =
        EasyExcel.readSheet(1).head(DemoData.class).registerReadListener(new
DemoDataListener()).build();
    // 这里注意 一定要把sheet1 sheet2 一起传进去，不然有个问题就是03版的excel 会读取多次，
浪费性能
    excelReader.read(readSheet1, readSheet2);
    // 这里千万别忘记关闭，读的时候会创建临时文件，到时磁盘会崩的
    excelReader.finish();
```

4.3 日期、数字或者自定义格式转换

- 转日期

```
  @DateTimeFormat("yyyy年MM月dd日HH时mm分ss秒")
```

- 转数字

```
//数字转百分比
@NumberFormat("#.##%")
```