**NiceXSSFWorkbook（XLSX）**

**使用说明**

1. **概要**

NiceXSSFWorkbook是针对POI中XSSFWorkbook功能的进一步封装和完善，以便更好的帮助java开发者操作excel。它继承类XSSFWorkbook中的所有的功能，并扩展了一些功能（包括excel表格的插入行、删除行、插入列、删除列等等功能）。XSSFTemplate可以通过getXSSFWorkbook()方法来获取这个对象，以便进行操作。

1. **功能**

**2.0. 通用参数注解**

sheetIndex：工作表位置（从0开始）

rowIndex：单元格所在的行（从0开始）

columnIndex：单元格所在的列（从0开始）

* 1. **创建**

NiceXSSFWorkbook workbook = new NiceXSSFWorkbook*()*;

打开一个空白的工作簿，工作簿中会默认创建一个Sheet0工作表。

File file = new File*(*Demo2.class.getClassLoader*()*.getResource*(*"template.xlsx"*)*.getPath*())*;  
InputStream is = new FileInputStream*(*file*)*;  
NiceXSSFWorkbook workbook = new NiceXSSFWorkbook*(*is*)*;

通过流的形式打开一个工作簿。

File file = new File*(*Demo3.class.getClassLoader*()*.getResource*(*"template.xlsx"*)*.getPath*())*;  
NiceXSSFWorkbook workbook = NiceXSSFWorkbook.*compile(*file*)*;

通过文件的形式打开一个工作簿

String path = Demo4.class.getClassLoader*()*.getResource*(*"template.xlsx"*)*.getPath*()*;  
NiceXSSFWorkbook workbook = NiceXSSFWorkbook.*compile(*path*)*;

通过文件路径的形式打开一个工作簿

* 1. **输出**

FileOutputStream fileOutputStream = new FileOutputStream*(*"C:\\Users\\Administrator\\Desktop\\excel\\create\\demo1.xlsx"*)*;  
workbook.writeToOutputStream*(*fileOutputStream*)*;

将工作簿输出到任意流中，如输出到文件流FileOutputStream生成新文档，输出到网络流ServletOutputStream供浏览器下载

NiceXSSFWorkbook workbook = new NiceXSSFWorkbook*(*is*)*;  
workbook.writeToFile*(*"C:\\Users\\Administrator\\Desktop\\excel\\create\\demo2.xlsx"*)*;

将工作簿保存到文件中

* 1. **获取单元格数据**

public String getStringCellValue*(*int sheetIndex, int rowIndex, int columnIndex*)*

获取字符串类型数据。

public Double getDoubleCellValue*(*int sheetIndex, int rowIndex, int columnIndex*)*

获取双精度浮点型数据。如果单元格为空，则返回null。

public Float getFloatCellValue*(*int sheetIndex, int rowIndex, int columnIndex*)*

获取单精度浮点型数据。如果单元格为空，则返回null。

public Long getLongCellValue*(*int sheetIndex, int rowIndex, int columnIndex*)*

获取长整型数据。如果单元格为空，则返回null。

public Integer getIntegerCellValue*(*int sheetIndex, int rowIndex, int columnIndex*)*

获取整型数据。如果单元格为空，则返回null。

public Date getDateCellValue*(*int sheetIndex, int rowIndex, int columnIndex*)*

获取日期类型数据，如果单元格为空，则返回null。

* 1. **写入单元格数据**

public void setCellValue*(*int sheetIndex, int rowIndex, int columnIndex, String value*)*

设置单元格的值，value表示写入的值

public void setCellValue*(*int sheetIndex, int rowIndex, int columnIndex, String value, Style style*)*

设置单元格的值，value表示写入的值，style表示单元格的样式（Style的构建请参考poi-et.docx文档中的TextRenderData模块下Style的构建方式）。

public void setCellValue*(*int sheetIndex, int rowIndex, int columnIndex, TextRenderData textRenderData*)*

设置单元格的值，TextRenderData表示文本渲染数据对象，构建方式请参考poi-et.docx文档中的TextRenderData。

public void setCellValue*(*XSSFCell cell, String value*)*

设置单元格的值，其中XSSFCell表示单元格对象，可以通过getCell（详见2.99.中的getCell）方法获取，value表示待写入的值。

public void setCellValue*(*XSSFCell cell, String value, Style style*)*

设置单元格的值，其中XSSFCell表示单元格对象，可以通过getCell（详见2.99.中的getCell）方法获取，value表示待写入的值，style表示单元格的样式（Style的构建请参考poi-et.docx文档中的TextRenderData模块下Style的构建方式）。

public void setCellValue*(*XSSFCell cell, TextRenderData textRenderData*)*

设置单元格的值。XSSFCell表示单元格对象，可以通过getCell（详见2.99.中的getCell）方法获取；TextRenderData表示文本渲染数据对象，构建方式请参考poi-et.docx文档中的TextRenderData。

public void setCellValue*(*int sheetIndex, int rowIndex, int columnIndex, PictureRenderData pictureRenderData*)*

向单元格所在的位置插入图片，PictureRenderData表示图片渲染数据，构建方式请参考poi-et.docx文档中的PictureRenderData。

public void setCellValue*(*XSSFCell cell, PictureRenderData pictureRenderData*)*

项单元格所在的位置插入图片。XSSFCell表示单元格对象，可以通过getCell（详见2.99.中的getCell）方法获取；PictureRenderData表示图片渲染数据，构建方式请参考poi-et.docx文档中的PictureRenderData。

* 1. **单元格合并与拆分**

isMergedRegion*(*int sheetIndex, int rowIndex, int columnIndex*)*

判断单元格是否为合并单元格（True：是；false：否）。判断依据为当前单元格是否为其中某个合并单元格的组成部分。

public boolean isMergedRegion*(*XSSFCell cell*)*

判断单元格是否为合并单元格（True：是；false：否），其中cell表示单元格对象。判断依据为当前单元格是否为其中某个合并单元格的组成部分。

public boolean isMergedRegionBegin*(*int sheetIndex, int rowIndex, int columnIndex*)*

判断单元格是否为合并单元格起点（True：是；false：否）

public boolean isMergedRegionBegin*(*XSSFCell cell*)*

判断单元格是否为合并单元格起点（True：是；false：否）

public CellRangeAddress getCellRangeAddress*(*int sheetIndex, int rowIndex, int columnIndex*)*

获取合并单元格，如果不存在合并单元格，则返回null，否则返回CellRangeAddress对象（POI中的合并单元格对象,使用方式请参考POI），其中是否存在单元格的判断依据参考isMergedRegion

public CellRangeAddress getCellRangeAddress*(*XSSFCell cell*)*

获取合并单元格，cell为单元格对象，如果不存在合并单元格，则返回null，否则返回CellRangeAddress对象（POI中的合并单元格对象,使用方式请参考POI），其中是否存在单元格的判断依据参考isMergedRegion。

public void updateCellRangeAddress*()*

更新合并单元格（用于单元格的合并和拆分之后的更新）

public void addMergedRegion*(*int sheetIndex, int firstRowIndex, int lastRowIndex, int firstColumnIndex, int lastColumnIndex*)*

合并单元格，其中firstRowIndex表示合并开始行位置，lastRowIndex表示合并结束行位置，firstColumnIndex表示合并开始列位置，lastColumnIndex表示合并结束列位置。

public void addMergedRegion*(*int sheetIndex, int firstRowIndex, int lastRowIndex, int firstColumnIndex, int lastColumnIndex, boolean isUpdate*)*

合并单元格，其中firstRowIndex表示合并开始行位置；lastRowIndex表示合并结束行位置；firstColumnIndex表示合并开始列位置；lastColumnIndex表示合并结束列位置；isUpdate表示是否更新合并单元格的数据，用于大批量的合并单元格，且保证单元格不会重叠的情况，可以选择false，最后统一调用updateMergedRegion方法进行更新，否则请选择true。

public void removeMergedRegion*(*XSSFCell cell*)*

拆分单元格，cell表示单元格对象。

public void removeMergedRegion*(*XSSFCell cell, boolean isUpdate*)*

拆分单元格，cell表示单元格对象，isUpdate表示是否更新合并单元格的数据，用于大批量的拆分单元格，如果选择false，则最后一定要调用updateMergedRegion方法进行更新，否则请选择true。

* 1. **行操作**

public void insertRowsBefore*(*int sheetIndex, int insertRowIndex, int insertNum*)*

在第insertRowIndex行之前插入insertNum行。

public void insertRowsAfter*(*int sheetIndex, int insertRowIndex, int insertNum*)*

在第insertRowIndex行之前插入insertNum行。

public void copyRow*(*int sheetIndex, int targetRowIndex, int sourceRowIndex*)*

将sourceRowIndex行复制到targetRowIndex行。复制行的数据和样式以及合并单元格（合并单元格只有复制的源行（sourceRowIndex）为起点的才会进行复制）

public void copyRow*(*XSSFRow targetRow, XSSFRow sourceRow*)*

将sourceRow行复制到targetRow行。其中sourceRow为源行，targetRow为目标行。复制行的数据和样式以及合并单元格（合并单元格只有复制的源行（sourceRowIndex）为起点的才会进行复制）

public void removeRow*(*int sheetIndex, int rowIndex*)*

移除rowIndex行。移除行的时候，如果存在已待移除行为起点的合并单元格，则这些单元格全部拆分，且涉及到的单元格样式全部清除。如果存在跨带移除行的合并单元格，则这些合并单元格合并行全部-1。

* 1. **列操作**

public void insertColumnsBefore*(*int sheetIndex, int columnIndex, int insertNum*)*

在columnIndex列之前插入insertNum列

public void insertColumnsAfter*(*int sheetIndex, int columnIndex, int insertNum*)*

在columnIndex列之后插入insertNum列

public void copyColumn*(*int sheetIndex, int targetColumnIndex, int sourceColumnIndex*)*

复制列，在同一个工作表内进行列的复制

public void copyColumn*(*int targetSheetIndex, int targetColumnIndex, int sourceSheetIndex, int sourceColumnIndex*)*

复制列（targetSheetIndex：目标工作表位置，targetColumnIndex：目标列位置；sourceSheetIndex：源工作表位置，sourceColumnIndex：源列位置）。

public void removeColumn*(*int sheetIndex, int columnIndex*)*

移除sheetIndex工作表columnIndex列

**2.8. 单元格操作**

public XSSFCell getCell*(*int sheetIndex, int rowIndex, int columnIndex*)*

获取poi中的XSSFCell对象

public void copyCell*(*int sheetIndex, int targetRowIndex, int targetColumnIndex, int sourceRowIndex, int sourceColumnIndex*)*

同一个工作表内复制单元格对象（包括内容，样式，合并单元格）

public void copyCell*(*int targetSheetIndex, int targetRowIndex, int targetColumnIndex, int sourceSheetIndex, int sourceRowIndex, int sourceColumnIndex*)*

复制单元格对象（包括内容，样式，合并单元格）

public void copyCell*(*XSSFCell targetCell, XSSFCell sourceCell*)*

复制单元格对象（包括内容，样式，合并单元格）