Apache POI - the Java API

介绍

Home：http://poi.apache.org/

Doc：http://poi.apache.org/apidocs/index.html

Download：http://poi.apache.org/download.html

Example: http://poi.apache.org/spreadsheet/examples.html

Apache POI是Apache开源项目，用于操作Office系列产品的一套开源的API。

注意事项

1. 文档中使用的是目前最新的3.17版本，支持JDK9。
2. 文档中的范例使用到的是项目当前路径下的“res/temp/”，请自行创建。

导入POI jar包

方式一：使用官网下载地址将下载的依赖包导入即可。

方式二： 推荐使用maven自动下载添加依赖。http://mvnrepository.com/ 输入poi搜索即可。

```
<!-- https://mvnrepository.com/artifact/org.apache.poi/poi-ooxml -->
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-ooxml</artifactId>
    <version>3.17</version>
</dependency>
```

POI包简介

包名	说明
org.apache.poi.xssf.usermodel.HSSF	提供读写Microsoft Excel XLS格式档案的功能
org.apache.poi.xssf.usermodel.XSSF	提供读写Microsoft Excel OOXML XLSX格式档案的功能
org.apache.poi.xssf.usermodel.HWPF	提供读写Microsoft Word DOC格式档案的功能
org.apache.poi.xssf.usermodel.HSLF	提供读写Microsoft PowerPoint格式档案的功能
org.apache.poi.xssf.usermodel.HDGF	提供读Microsoft Visio格式档案的功能
org.apache.poi.xssf.usermodel.HPBF	提供读Microsoft Publisher格式档案的功能
org.apache.poi.xssf.usermodel.HSMF	提供读Microsoft Outlook格式档案的功能

org.apache.poi.hssf.usermodel.HSSF	提供读写Microsoft Excel格式档的功能

Workbook接口（工作簿）

工作簿(Workbook)的概念也就是我们所谓的一个excel文件，是操作excel的入口，包含两个重要类。

读写.xls格式的excel文件	org.apache.poi.xssf.usermodel.HSSFWorkbook.java
读写.xlsx格式的excel文件	org.apache.poi.xssf.usermodel.XSSFWorkbook.java

创建工作簿一般语法：

```
Workbook workBook = new XSSFWorkbook(): //xlsx格式
Workbook workBook = new HSSFWorkbook(): //xls格式
```

XSSFWorkbook 核心API讲解

类的构造函数

S.No.	构造函数和说明
1	XSSFWorkbook() 从头开始创建一个新的XSSFworkbook对象。
2	XSSFWorkbook(java.io.File file) 构造从给定文件中的XSSFWorkbook对象。
3	XSSFWorkbook(java.io.InputStream is) 构造一个XSSFWorkbook对象，通过缓冲整个输入流到内存中，然后为它打开一个OPCPackage对象。
4	XSSFWorkbook(java.lang.String path) 构建一个给定文件的完整路径的XSSFWorkbook对象。

类方法

S.No.	方法及描述
1	createSheet() 创建一个XSSFSheet本工作簿，将其添加到表，并返回高层表示。
2	createSheet(java.lang.String sheetname) 创建此工作簿的新表，并返回高层表示。
3	createFont() 创建一个新的字体，并将其添加到工作簿的字体表。
4	createCellStyle() 创建一个新的XSSFCellStyle并将其添加到工作簿的样式表。
5	createFont() 创建一个新的字体，并将其添加到工作簿的字体表。
6	setPrintArea(int sheetIndex, int startColumn, int endColumn, int startRow,int endRow) 设置一个给定的表按照指定参数的打印区域。
7	cloneSheet(int sheetNum) 克隆指定索引sheet页

对于此类的其余的方法，请参阅完整的API文档：

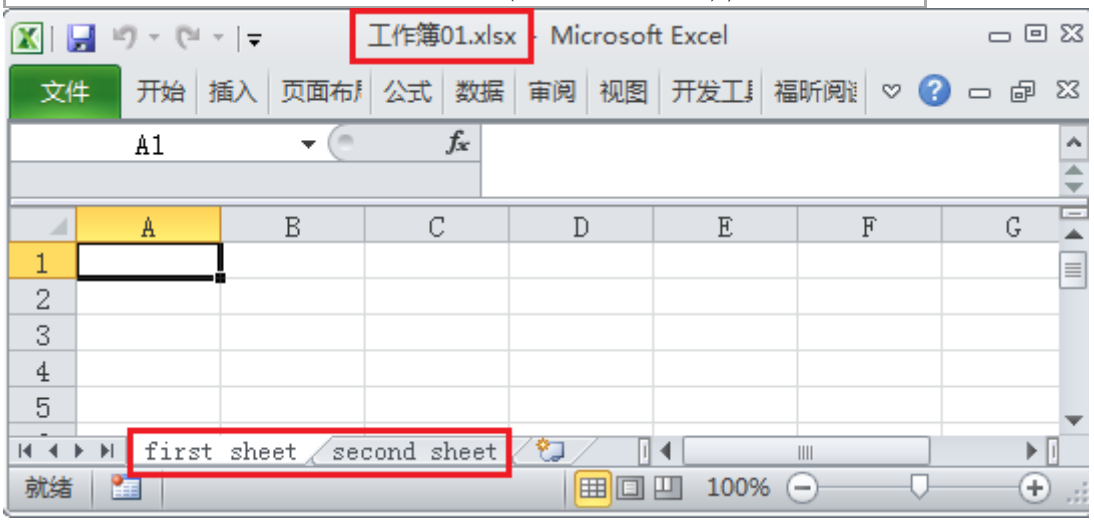
http://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFWorkbook.html

Sheet接口（工作表）

先有工作簿才有sheet页的概念，sheet页指的就是我们excel文件中的每一个表格，一个工作簿最少需要包含一个Sheet。

创建sheet页一般语法：

```
Sheet  sheet1  =  workBook.createSheet("first  sheet");
Sheet  sheet2  =  workBook.createSheet("second  sheet");
```



XSSFSheet 核心API

包：org.apache.poi.xssf.usermodel. XSSFSheet

类的构造函数

S.No.	构造函数及描述
1	XSSFSheet() 创造了新的XSSFSheet- 调用XSSFWorkbook从头开始创建一个表。
2	XSSFSheet(PackagePart part, PackageRelationship rel) 创建XSSFSheet表示给定包的一部分和关系。

类方法

S.No.	方法和描述
1	addMergedRegion(CellRangeAddress region) 添加单元的合并区域（因此这些单元格合并形成一个）。
2	autoSizeColumn(int column) 调整列宽，以适应的内容。
3	iterator() 此方法是用于rowIterator()的别名，以允许foreach循环
4	addHyperlink(XSSFHyperlink hyperlink) 注册超链接的集合中的超链接此工作表格上
5	getFirstRowNum() 获得第一行的序列号.排除前面的空行。

	getLastRowNum() 获得最后一行的序列号。包含空行。
	getPhysicalNumberOfRows() 获得实际存在的行的总数.排除存在的空行

对于此类的其余的方法，请参阅完整的API在:

https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFSheet.html

范例:创建一个工作簿，包含两个sheet。

```
        public static void main(String[] args) throws Exception
        {
                Workbook workBook = new XSSFWorkbook(); // 在内存中创建工作簿
                // Workbook workBook = new HSSFWorkbook(); //在内存中创建工作簿,xls格式
                Sheet sheet1 = workBook.createSheet("first sheet"); //在内存中创建sheet页
                Sheet sheet2 = workBook.createSheet("second sheet");
                FileOutputStream fileOutputStream = new FileOutputStream("res/temp/工作簿01.xlsx");
                workBook.write(fileOutputStream); // 将工作簿写入到具体的文件
                workBook.close();
                System.out.println("finish");
        }
```

范例：获取工作表中的行数

这里主要介绍3个方法：

getFirstRowNum()：获取Sheet页中的第一行的序列号，包含空行。例如在Sheet中第2行创建一行，第一行的行号结果就为2.默认索引从0开始计算。

getLastRowNum()：获取Sheet页中的最后行的序列号，包含空行。

getPhysicalNumberOfRows()：获取Sheet页中实际行数相加的结果。例如在第二行、第四行、第10行分别创建三行，那么得到的结果是3行。

```
        public static void main(String[] args) throws Exception
        {
                XSSFWorkbook workbook = new XSSFWorkbook();
                XSSFSheet sheet = workbook.createSheet("工作表行数获取");
                System.out.println("创建行之前的结果");
                System.out.println("第一行: " + sheet.getFirstRowNum()); // 第一行行号
                System.out.println("最后行: " + sheet.getLastRowNum()); // 最后行号
                System.out.println("实际行:" + sheet.getPhysicalNumberOfRows()); // 表中实际存在的行数
                sheet.createRow(2).createCell(0).setCellValue("A3");
                ; // 在第3行创建一行
                System.out.println("创建第2行后的状态");
                System.out.println("第一行: " +
```

```
sheet.getFirstRowNum()); // 第一行行号
                System.out.println("最后行：" +
sheet.getLastRowNum()); // 最后行号
                System.out.println("实际行:" +
sheet.getPhysicalNumberOfRows()); // 表中实际存在的行数
                sheet.createRow(4).createCell(0).setCellValue("A5");//
在第5行创建一行
                System.out.println("创建第4行后的状态");
                System.out.println("第一行：" +
sheet.getFirstRowNum()); // 第一行行号
                System.out.println("最后行：" +
sheet.getLastRowNum()); // 最后行号
                System.out.println("实际行:" +
sheet.getPhysicalNumberOfRows()); // 表中实际存在的行数
                workbook.write(new FileOutputStream("res/temp/工作簿
02.xlsx"));

                workbook.close();
        }
```

Row接口（行）

有了sheet页后就可以创行和列。Row就代表工作表中的某一行。

XSSFRow 核心API

类方法

S.No.	描述
1	createCell(int columnIndex) 创建新单元行并返回。
2	setHeight(short height) 设置短单位的高度。
3	getLastCellNum() 获取最后一列单元格的序号，也就是最大列数
4	getPhysicalNumberOfCells() 获取实际单元格列数，排除空白单元格

重点：row.getLastCellNum()这个方法一般用于获取表格的列数用于遍历单元格。

Cell接口（单元格）

单元格可以使用各种属性，例如空白，数字，日期，错误等单元格被添加到一个行之前应具有（基于0）自己的编号。

XSSFCell 核心API

字段摘要

下面列出的是一些XSSFCell类的字段以及它们的描述。

单元格类型	描述
CELL_TYPE_BLANK	代表空白单元格
CELL_TYPE_BOOLEAN	代表布尔单元（true或false）
CELL_TYPE_ERROR	表示在单元的误差值
CELL_TYPE_FORMULA	表示一个单元格公式的结果
CELL_TYPE_NUMERIC	表示对一个单元的数字数据
CELL_TYPE_STRING	表示对一个单元串（文本）

类方法

S.No.	描述
1	setCellStyle(CellStyle style) 为单元格设置样式。
2	setCellType(int cellType) 设置单元格的类型（数字，公式或字符串）。
3	setCellValue(boolean value) 设置单元格一个布尔值
4	setCellValue(java.util.Calendar value) 设置一个日期值的单元格。
5	setCellValue(double value) 设置为单元格的数值。
6	setCellValue(java.lang.String str) 设置为单元格的字符串值。
7	setHyperlink(Hyperlink hyperlink) 分配超链接到该单元格。

对于这个类的剩余方法和字段，请访问以下链接查看详细：

https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFCell.html

范例：创建单元格

```
        public static void main(String[] args) throws Exception
        {
                Workbook workBook = new XSSFWorkbook(); // 在内存中创建工作簿
                // Workbook workBook = new HSSFWorkbook(); //在内存中创建工作簿,xls格式
                Sheet sheet1 = workBook.createSheet("first sheet"); //在内存中创建sheet页
                Sheet sheet2 = workBook.createSheet("second sheet");
                // 创建一个单元格并在其中加入内容.先创建一行
                Row row1 = sheet1.createRow(0);
                row1.createCell(0).setCellValue("A1"); // 在第1行创建第1个单元格，并设置值为A1
```

```
第1个单元格，并设置值为A1
                row1.createCell(1).setCellValue("1.2"); // 在第1行创建
第2个单元格，并设置值为1.2
                row1.createCell(2).setCellValue("this is string"); //
在第1行创建第3个单元格，并设置值
                row1.createCell(3).setCellValue(true); // 在第1行创建
第4个单元格，并设置值为boolean值
                FileOutputStream fileOutputStream = new
FileOutputStream("res/temp/工作簿01.xlsx");
                workBook.write(fileOutputStream); // 将工作簿写入到具
体的文件
                workBook.close();
                System.out.println("finish");
        }
```

范例：创建时间类型的单元格

```
        public static void main(String[] args) throws Exception
        {
                Workbook workBook = new XSSFWorkbook(); // 在内存中创
建工作簿
                // Workbook workBook = new HSSFWorkbook(); //在内存中
创建工作簿,xls格式
                Sheet sheet1 = workBook.createSheet("first sheet"); //
在内存中创建sheet页
                Sheet sheet2 = workBook.createSheet("second sheet");
                Row row1 = sheet1.createRow(0);// 创建一行
                Cell cell1 = row1.createCell(0);        //在row1行创建
一个单元格
                cell1.setCellValue(new Date()); //直接使用是无法创建时
间格式的单元格
                System.out.println(new Date());

                Cell cell2 = row1.createCell(1);        //创建第二个单
元格，设置格式为时间
                CellStyle cellStyle = workBook.createCellStyle();
                CreationHelper creationHelper =
workBook.getCreationHelper();
                //设置单元格格式
        cellStyle.setDataFormat(creationHelper.createDataFormat().getF
ormat("m/d/yy h:mm"));
                cell2.setCellValue(new Date());
                cell2.setCellStyle(cellStyle);

                FileOutputStream fileOutputStream = new
FileOutputStream("res/temp/工作簿01.xlsx");
                workBook.write(fileOutputStream); // 将工作簿写入到具
体的文件
                workBook.close();
                System.out.println("finish");
        }
```

范例：遍历单元格行和列

Row对象定义了一个cellIterator()方法用来处理单元格的遍历（可以通过调用row.cellIterator() 来获得Iterator<Cell>对象），另外,Sheet 对象提供了一个rowIterator()方法对所有行进行遍历。除此之外，Sheet 和Row对象都实现了java.lang.Iterable 接口。

```
/**
 *
 * @Description: 表格的遍历
 * @author shan_dongdong
 * @date 2017年10月11日 下午2:28:34
 * @version V1.0.0
 */
public class Test05
{
	public static void main(String[] args) throws IOException
	{
		Workbook workbook = new XSSFWorkbook("res/temp/工作簿
01.xlsx");
		Sheet sheet1 = workbook.getSheetAt(0);
		// 读取每一行
		for (Iterator<Row> iter = sheet1.rowIterator();
iter.hasNext();)
		{
			Row row = iter.next();
			// 读取每一列
			for (Iterator<Cell> iterCell =
row.cellIterator(); iterCell.hasNext();)
			{
				Cell cell = iterCell.next();
				cell.setCellType(CellType.STRING); //
将单元格的值设置为字符串

System.out.print(cell.getStringCellValue() + "\t");
			}
			System.out.println();
		}
		System.out.println("第二种遍历方式-----------------
-->");
		for (Row row : sheet1)
		{
			for (Cell cell : row)
			{


				cell.setCellType(CellType.STRING); //
将单元格的值设置为字符串

System.out.print(cell.getStringCellValue() + "\t");
			}
			System.out.println();
		}
		workbook.close();
```

```
        }
}
```

范例：合并单元格

合并单元格是针对sheet页而言的。主要使用了了Sheet.addMergedRegion(new CellRangeAddress(1, 1, 1, 2));方法，进行行列的合并。

需要注意的是，当你合并单元格时，合并完后的单元格输入插入时，需要插入合并时的第一个单元格。例如：1,1,1,2。在第二行合并第二列和第三列,第二行BC两列合并在一起后，那么createCell应该是在第二行B列创建单元格，并写入值，而不是第二行C列写入值。

```
/**
 *
 * @Description: 单元格的合并
 * @author shan_dongdong
 * @date 2017年10月11日 下午5:09:10
 * @version V1.0.0
 */
public class Test08
{
        public static void main(String[] args) throws IOException
        {
                Workbook workbook = new XSSFWorkbook();
                Sheet sheet1 = workbook.createSheet("first sheet");
                // CellRangeAddress类构造方法接受4个参数。参数1：第一行。参数2：最后一行。参数3：第一列。参数4：最后一列
                sheet1.addMergedRegion(new CellRangeAddress(1, 1, 1, 2)); // 合并第二行BC列
                Row row1 = sheet1.createRow(1); // 创建第二行
                row1.setHeightInPoints(30);
                Cell cell1 = row1.createCell(1);
                cell1.setCellValue("测试单元格合并");
                workbook.write(new FileOutputStream("res/temp/工作簿01.xlsx"));
                workbook.close();
                System.out.println("finish");
        }
}
```

范例：在单元格内使用函数

```
package com.github.shandongdong.api;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
/**
 *
 * @Description: 向单元格内读写入excel公式.向C2单元格里写入A2+B2，C3单元格里写入AVERAGE(A3,B3)
 * @author shan_dongdong
 * @date 2017年10月18日 下午4:17:05
```

```
 * @version V1.0.0
 */
public class SetCellFormula
{
	public static void main(String[] args) throws Exception
	{
		// 将文件转换为流读取
		FileInputStream fileInputStream = new
FileInputStream("res/temp/工作簿01.xlsx");// 将excel文件转为输入流
		XSSFWorkbook workbook = new
XSSFWorkbook(fileInputStream); // XSSFWorkbook构造方法可以接受一个输入
流做参数
		// 向excel中写入公式
		XSSFSheet sheet = workbook.getSheetAt(0);
		XSSFRow row1 = sheet.createRow(1);
		row1.createCell(0).setCellValue("12");
		row1.createCell(1).setCellValue("34");
		XSSFCell cell1 = row1.createCell(2);
		cell1.setCellFormula("A2+B2"); // 写入函数
		row1 = sheet.createRow(2);
		row1.createCell(0).setCellValue(10); // 如果是求平均
数，两个数都需要使用整数类型，不能用字符串
		row1.createCell(1).setCellValue(5);
		// 上面两行的值如果都用字符串类型则，单元格写入后会出
现错误"#DIV/0!"
		row1.createCell(2).setCellFormula("AVERAGE(A3,B3)");
// 写入函数
		// 读取excel中的公式
		String formula = row1.getCell(2).getCellFormula();
		System.out.println("C3单元格公式是:=" + formula); //
实际在excel中需要写=
		// 保存并关闭流
		FileOutputStream out = new FileOutputStream("res/temp/
工作簿01.xlsx"); // 保存位置
		workbook.write(out); // 将工作簿内容写入到输出流
		workbook.close();
	}
}
```

范例：遍历单元格并将值写入另一张sheet表中

```
package com.github.shandongdong.report;
/**
 *
 * @Description: 通过WeTrack导出的Bug数据，自动统计
 * @author shan_dongdong
 * @date 2017年10月10日 上午10:37:04
 * @version V1.0.0
 */
public class ExcelBug
{
	public static void main(String[] args) throws Exception
	{
		InputStream inputStream = new FileInputStream(new
File("E:/POI/excel报表需求/123.xls"));
		Workbook workBook = new HSSFWorkbook(inputStream);
```

```
                Workbook workBook = new HSSFWorkbook(inputStream);
                // 读取数据写入工作表1
                Sheet sheet1 = workBook.getSheetAt(0);
                Sheet sheet3 = workBook.createSheet("拷贝的数据");
                int i = 0;

                for (Row row : sheet1)
                {
                        Row newRow = sheet3.createRow(i); // 在sheet页
3创建新行
                        int j = 0;
                        for (Cell cell : row)
                        {
                                Cell newCell = newRow.createCell(j);
// 在sheet页3创建新列

newCell.setCellValue(cell.getStringCellValue());
                                j++;
                        }
                        i++;    //行数递增
                }
                //第二种方式
                for (Iterator<Row> iter = sheet1.rowIterator();
iter.hasNext();)
                {
                        Row newRow = sheet3.createRow(i); // 在sheet页
3创建新行
                        Row row = iter.next();
                        int j = 0;
                        for (Iterator<Cell> cellIter =
row.cellIterator(); cellIter.hasNext();)
                        {
                                Cell cellValue = cellIter.next();
                                Cell newCell = newRow.createCell(j);
// 在sheet页3创建新列

newCell.setCellValue(cellValue.getStringCellValue());
                                j++;
                        }
                        i++;
                }

                FileOutputStream fileOut = new
FileOutputStream("E:/POI/excel报表需求/问题单导出结果2017_10_09-
-17_25_45_new2.xls");
                workBook.write(fileOut);
                workBook.close();
                }
}
```

注意：通过以上两种遍历方式如果遇到空白单元格那么就会存在问题。因为使用增强的for循环遍历时，当你调用iter.hasNext()方法时，会自动排除空白的单元格，这样我们就没法处理空白单

元格的情况。

例如，下面的表格。第一行读出的列数为7。第二行列数为5，第三行列数为4.

问题单号	简要描述	测试领域	问题大类	测试类型	测试分析	严重程度
1	2	3	4		6	7
1	2	3		5		7

所以一般情况下我们都不会使用以上两种遍历单元格方式操作单元格。会使用sheet.getLastRowNum() 获取最大行，如果有空行进行去除。使用sheet.getRow(0).getLastCellNum();获取第一行的最大列数，作为最大列。

范例：遍历处理空白单元格

```
/**
 *
 * @Description: 处理Excel中存在的空白单元格
 * @author shan_dongdong
 * @date 2017年10月10日 上午10:37:04
 * @version V1.0.0
 */
public class ExcelBug_V2
{
        public static void main(String[] args) throws Exception
        {
                InputStream inputStream = new FileInputStream(new
File("E:/POI/excel报表需求/问题单导出结果2017_10_09--17_25_45.xls"));
                Workbook workBook = new HSSFWorkbook(inputStream);
                // 读取数据写入工作表1
                Sheet sheet1 = workBook.getSheetAt(0);
                Sheet sheet3 = workBook.createSheet("拷贝的数据");
                int maxRow = sheet1.getLastRowNum(); // 最大行，包含空
行
                short maxCell = sheet1.getRow(0).getLastCellNum(); //
第一行最大列，包含空列
                for (int i = 0; i <= maxRow; i++)
                {
                        Row row = sheet1.getRow(i); // 读取每行

                        Row newRow = sheet3.createRow(i); // 在sheet页
3创建新行

                        for (int j = 0; j < maxCell; j++)
                        {
                                Cell cell = null;
                                try
                                {
                                        cell = row.getCell(j); // 读取
每列,如果第一单元格都为空则跳过此行
                                }
                                catch (java.lang.NullPointerException
e)
                                {
                                        break;
                                }
```

```
                                        // 单元格为空，不存在的情况。直接创建
设备为空白即可
                                        if (null == cell)
                                        {
                                                cell = row.createCell(j);
                                                cell.setCellValue("");
                                        }
                                        // 需要处理单元格为空或者空白的情况
                                        CellType cellType =
cell.getCellTypeEnum();
                                        System.out.println("cellType:" +
cellType);
                                        Cell newCell = newRow.createCell(j);
// 在sheet页3创建新列
                                        System.out.print("第" + j + "列单元格
的类型是：" + cellType.name() + ",值是:" + cell.getStringCellValue() +
"。\t");
                                        if (cellType == CellType._NONE)
                                        {
                                                newCell.setCellValue("NONE");
                                                System.out.println("----------
---------NONE");
                                        }
                                        else if (cellType == CellType.BLANK)
                                        {
                                                newCell.setCellValue("BLANK");
                                                System.out.println("----------
---------BLANK");
                                        }
                                        else if (cellType == CellType.BOOLEAN)
                                        {
                                                newCell.setCellValue("true");
                                                System.out.println("----------
---------BOOLEAN");
                                        }
                                        else if (cellType == CellType.ERROR)
                                        {
                                                newCell.setCellValue("error");

                                                System.out.println("----------
---------ERROR");
                                        }
                                        else if (cellType == CellType.FORMULA)
                                        {
                                                newCell.setCellValue("公式");
                                                System.out.println("----------
---------公式");
                                        }
                                        // 对于字符串的处理
                                        if (cellType.name().equals(""))
                                        {
                                                newCell.setCellValue("空字符
串");
                                        }
```

```
                                        else if (cellType == CellType.STRING)
                                        {

newCell.setCellValue(cell.getStringCellValue());
                                        }
                                        else
                                        {
                                                newCell.setCellValue("无效
值");
                                                System.out.println("----------
---------无效值");
                                        }
                                }
                                System.out.println();
                        }
                        FileOutputStream fileOut = new
FileOutputStream("E:/POI/excel报表需求/问题单导出结果2017_10_09-
-17_25_45_new2.xls");
                        workBook.write(fileOut);
                        workBook.close();
                }
}
```

CellStyle接口（单元格样式）

CellStyle提供了对单元格设置各种格式、颜色、对其方式等样式的设定接口。

XSSFCellStyle 核心API

在org.apache.poi.xssf.usermodel包的类。它将提供关于在电子表格的单元格中的内容的格式可能的信息。它也提供了用于修正该格式的选项。它实现了CellStyle接口。

字段摘要

下表列出了从CellStyle接口继承一些字段。

字段名称	字段描述
ALIGN_CENTER	中心对齐单元格内容
ALIGN_CENTER_SELECTION	中心选择水平对齐方式
ALIGN_FILL	单元格适应于内容的大小
ALIGN_JUSTIFY	适应单元格内容的宽度
ALIGN_LEFT	左对齐单元格内容
ALIGN_RIGHT	右对齐单元格内容
BORDER_DASH_DOT	使用破折号和点单元格样式
BORDER_DOTTED	用虚线边框的单元格样式
BORDER_DASHED	用虚线边框的单元格样式
BORDER_THICK	厚厚的边框单元格样式

BORDER_THIN	薄边框的单元格样式
VERTICAL_BOTTOM	对齐单元格内容的垂直下方
VERTICAL_CENTER	对齐单元格内容垂直居中
VERTICAL_JUSTIFY	对齐和垂直对齐的单元格内容
VERTICAL_TOP	顶部对齐为垂直对齐

类的构造函数

S.No.	构造函数及描述
1	XSSFCellStyle(int cellXfId, int cellStyleXfId, StylesTable stylesSource, ThemesTable theme) 创建一个单元格样式，从所提供的部分
2	XSSFCellStyle(StylesTable stylesSource) 创建一个空的单元样式

类方法

设置边框的类型为单元格的底部边界

S.No	方法及描述
1	setAlignment(short align) 设置单元格为水平对齐的类型
2	setBorderBottom(short border)
3	setBorderColor(XSSFCellBorder.BorderSide side, XSSFColor color) 选定的边框颜色
4	setBorderLeft(Short border) 设置边界的类型单元格的左边框
5	setBorderRight(short border) 设置边框的类型为单元格的右边界
6	setBorderTop(short border) 设置边界的类型的单元上边框
7	setFillBackgroundColor(XSSFColor color) 设置表示为XSSFColor值背景填充颜色。
8	setFillForegroundColor(XSSFColor color) 设置表示为XSSFColor的值前景填充颜色。
9	setFillPattern(short fp) 指定单元格的填充信息模式和纯色填充单元。
10	setFont(Font font) 设置此样式的字体。
11	setRotation(short rotation) 设置的旋转为在单元格中文本的程度。
12	setVerticalAlignment(short align) 设置单元类型为垂直取向。

对于这个类剩下的方法和字段，通过以下链接：

https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFCellStyle.html

范例：单元格的各种对齐方式

使用 CellStyle cellStyle = workBook.createCellStyle(); 为单元格设置格式。

```
        /**
         * 创建一个单元格并为其设定指定对齐方式
         *
         * @param workbook
         * @param row
         * @param cellColumn
```

```
         *            单元格的列坐标
         * @param value
         *            单元格的值
         * @param horizontalAlignment
         *            水平方向
         * @param verticalAlignment
         *            垂直方向
         */
        private static void createCell(Workbook workBook, Row row,
int cellColumn, String value,
                        HorizontalAlignment horizontalAlignment,
VerticalAlignment verticalAlignment)
        {
                Cell cell = row.createCell(cellColumn);
                cell.setCellValue(new XSSFRichTextString(value));
                CellStyle cellStyle = workBook.createCellStyle();
                cellStyle.setAlignment(horizontalAlignment);
                cellStyle.setVerticalAlignment(verticalAlignment);
                cell.setCellStyle(cellStyle); // 使样式生效
        }
```

范例：在单元格内使用换行

单元格内使用换行很简单，直接使用转义字符 \n 即可。

```
        public static void main(String[] args) throws Exception
        {
                InputStream inputStream = new
FileInputStream("res/temp/工作簿01.xlsx");
                Workbook workBook =
WorkbookFactory.create(inputStream);
                Sheet sheet1 = workBook.getSheetAt(0);

                Row row = sheet1.createRow(2);
                Cell cell = row.createCell(2);
                cell.setCellValue("使用\n进行创建单元格的换行");
                //使用换行必须先设置单元格样式 wrap=true
                CellStyle cellStyle = workBook.createCellStyle();
                cellStyle.setWrapText(true);
                //设置单元格的高度使能容纳两个文字
                row.setHeightInPoints(2 *
sheet1.getDefaultRowHeightInPoints());
                //自动调整第二列的列宽
                sheet1.autoSizeColumn(2);
                row.setRowStyle(cellStyle);

                workBook.write(new FileOutputStream("res/temp/工作簿
01.xlsx"));
                workBook.close();
        }
```

Color接口（颜色）

XSSFColor 核心API

这是在org.apache.poi.xssf.usermodel包的类。它是用来表示在电子表格中的颜色。它实现了颜色的接口。下面列出的是它的一些方法和构造函数。

类的构造函数

S.No.	Constructor and 描述
1	XSSFColor() 创建XSSFColor的新实例。
2	XSSFColor(byte[] rgb) 创建XSSFColor使用RGB的新实例。
3	XSSFColor(java.awt.Color clr) 创建XSSFColor使用Color类从AWT包的新实例。

类方法

S.No.	方法和描述
1	setAuto(boolean auto) 设置一个布尔值，表示ctColor是自动的，系统ctColor依赖。
2	setIndexed(int indexed) 设置索引ctColor值系统ctColor。

对于其余的方法，请访问以下链接：

https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFColor.html

范例：设置单元格背景色

设置单元格的颜色与格式相同，也是通过CellStyle接口来实现的。

```
CellStyle cellStyle = workBook.createCellStyle();
```

```
        public static void main(String[] args) throws Exception
        {
                Workbook workBook = new XSSFWorkbook(); // 在内存中创建工作簿
                Sheet sheet1 = workBook.createSheet("first sheet"); // 在内存中创建sheet页
                Row row = sheet1.createRow(0);// 创建一行
                row.setHeightInPoints(30);
                // 设置背景色为浅绿色
                CellStyle cellStyle = workBook.createCellStyle();

cellStyle.setFillBackgroundColor(IndexedColors.AQUA.getIndex()); // 设置颜色为浅绿色,默认只有选中才会出现效果
                cellStyle.setFillPattern(FillPatternType.SPARSE_DOTS);
                Cell cell = row.createCell(0);
                cell.setCellValue("SDD");
                cell.setCellStyle(cellStyle); // 使样式生效
```

```
                cell.setCellStyle(cellStyle); // 使样式生效
                // 橙色前景色。前景色：前景色是指正在使用的填充颜色，而非字体颜色
                cellStyle = workBook.createCellStyle();

cellStyle.setFillForegroundColor(IndexedColors.ORANGE.getIndex());

cellStyle.setFillPattern(FillPatternType.SOLID_FOREGROUND);
                cell = row.createCell(2);
                cell.setCellValue("X");
                cell.setCellStyle(cellStyle);// 使样式生效
                FileOutputStream fileOutputStream = new FileOutputStream("res/temp/工作簿01.xlsx");
                workBook.write(fileOutputStream); // 将工作簿写入到具体的文件
                workBook.close();
                System.out.println("finish");

        }
```

范例：自定义颜色

自定义颜色主要是使用RGB的值来自定义颜色。

```
cellStyle.setFillForegroundColor(new XSSFColor(new java.awt.Color(128, 128, 128)));
//自定义背景色
```

```
        public static void main(String[] args) throws Exception
        {
                XSSFWorkbook workbook = new XSSFWorkbook();
                XSSFSheet sheet1 = workbook.createSheet("first sheet");
                XSSFRow row = sheet1.createRow(0);
                row.setHeightInPoints(30);
                XSSFCell cell = row.createCell(0);
                cell.setCellValue("测试XSSF自定义颜色");
                XSSFCellStyle cellStyle = workbook.createCellStyle();
                cellStyle.setFillForegroundColor(new XSSFColor(new java.awt.Color(128, 128, 128))); // 自定义背景色

cellStyle.setFillPattern(FillPatternType.SOLID_FOREGROUND); // 填充单元格的模式
                cell.setCellStyle(cellStyle);
                workbook.write(new FileOutputStream("res/temp/工作簿01.xlsx"));
                workbook.close();
                System.out.println("finish");
        }
```

Font接口（字体）

XSSFFont 核心API

处理工作簿中不同的字体格式、大小等。

类的构造函数

S.No.	构造函数和描述
1	XSSFFont() 创建一个新的XSSFont实例。

类方法

S.No.	方法和描述
1	setBold(boolean bold) 设置“bold”属性的布尔值。
2	setColor(short color) 设置索引颜色的字体。
3	setColor(XSSFColor color) 设置为标准Alpha RGB颜色值的字体颜色。
4	setFontHeight(short height) 设置在点的字体高度。
5	setFontName(java.lang.String name) 设置字体的名称。
6	setItalic(boolean italic) 设置“italic”属性一个布尔值。

对于其余的方法，通过以下链接：

https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFFont.html

范例：字体的处理

字体需要自己创建Font，创建好后的字体需要通过CellStyle来设置使生效。

```
public static void main(String[] args) throws Exception
{
        Workbook workbook = new XSSFWorkbook();
        Sheet sheet1 = workbook.createSheet("first sheet");
        Row row1 = sheet1.createRow(1);
        row1.setHeightInPoints(30);
        // 创建字体
        Font font = workbook.createFont();
        font.setFontHeightInPoints((short) 24); // 设置字体大小为24，这里的24也就是我们在excel中下来选择的大小。工作簿字体最大正整数为32767。应当尽量避免在for循环中重复去设置字体
        font.setItalic(true); // 斜体
        font.setStrikeout(true); // 删除线
        font.setBold(true);     //加粗
        // 字体需要在样式载入中才能使用，因此我们需要先创建一
```

```
个样式，将字体应用于此样式
                CellStyle cellStyle = workbook.createCellStyle();
                cellStyle.setFont(font);
                // 创建单元格，并给单元格设置样式
                Cell cell1 = row1.createCell(1);
                cell1.setCellValue("测试字体样式");
                cell1.setCellStyle(cellStyle); // 应用样式
                workbook.write(new FileOutputStream("res/temp/工作簿
01.xlsx"));
                workbook.close();
                System.out.println("finish");
        }
```

Hyperlink接口（超链接）

XSSFHyperlink 核心API

字段

字段	描述
LINK_DOCUMENT	用于连接任何其他文件
LINK_EMAIL	用于链接的电子邮件
LINK_FILE	用于以任何格式链接任何其他文件
LINK_URL	用来连接一个网页URL

类方法

S.No.	方法及描述
1	setAddress(java.lang.String address) 超链接地址。

对于其余的方法，请访问以下链接：

https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFHyperlink.html

范例：超链接

超链接主要用于跳转，一般分为四种：

1. URL链接：点击打开浏览器跳转到指定网页。
2. 文件链接：点击打开指定目录下的具体文件。
3. Email链接：点击打开Outlook发送邮件。
4. 文档链接：点击跳转到工作簿的其他Sheet页。

```
        public static void main(String[] args) throws Exception
        {
                Workbook workBook = new XSSFWorkbook();
```

```
                Sheet sheet = workBook.createSheet("Hyperlinks");
                Cell cell = sheet.createRow(3).createCell(3);
                cell.setCellValue("URL Link");

                // 设置单元格格式为超链接，默认超链接为下划线、蓝色字
体
                CellStyle linkStyle = workBook.createCellStyle();
                Font linkFont = workBook.createFont();
                linkFont.setUnderline(Font.U_SINGLE); // 下划线
                linkFont.setColor(IndexedColors.BLUE.getIndex()); //
蓝色
                linkStyle.setFont(linkFont);
                // URL超链接
                CreationHelper creationHelper =
workBook.getCreationHelper();
                Hyperlink link =
creationHelper.createHyperlink(HyperlinkType.URL);
                link.setAddress("http://poi.apache.org/");
                cell.setHyperlink(link); // 设置单元格类型为URL
                cell.setCellStyle(linkStyle); // 设置单元格样式
                // 文件超链接，链接到当前目录下的文件
                cell = sheet.createRow(4).createCell(4);
                cell.setCellValue("File Link");
                link =
creationHelper.createHyperlink(HyperlinkType.FILE);
                link.setAddress("fileLink.xlsx"); // 链接到当前目录下
的fileLink.xlsx文件
                cell.setHyperlink(link);
                cell.setCellStyle(linkStyle);
                // e-mail链接
                cell = sheet.createRow(5).createCell(5);
                cell.setCellValue("Email  Link");
                link =
creationHelper.createHyperlink(HyperlinkType.EMAIL); // 注意：如果邮件
主题中存在空格，请保证是按照URL格式书写的
                link.setAddress("mailto:poi@apache.org?
subject=Hyperlinks");
                cell.setHyperlink(link);
                cell.setCellStyle(linkStyle);
                // 工作簿中的位置
                Sheet sheet2 = workBook.createSheet("Target Sheet");
// 创建另一个sheet页
                sheet2.createRow(0).createCell(0).setCellValue("Target
Cell");
                cell = sheet.createRow(6).createCell(6);
                cell.setCellValue("Worksheet Link");
                link =
creationHelper.createHyperlink(HyperlinkType.DOCUMENT);
                // 设置需要连接的地址
                link.setAddress("'Target Sheet'!A1");// 注意单引号内名
称必须与sheet名称完全一致，引号后面不允许有空格。!A表示单元格跳转到定
位到A1位置
                cell.setHyperlink(link);
                cell.setCellStyle(linkStyle);
                // 获取超链接
```

```
                if (null != link)
                        System.out.println(link.getAddress());
                // 写入文件，并关闭流

                workBook.write(new FileOutputStream("res/temp/工作簿
01.xlsx"));
                workBook.close();
        }
```

CreationHelper接口

XSSFCreationHelper 核心API

这是在org.apache.poi.xssf.usermodel包的类。它实现了CreationHelper接口。它被用作公式求值和设置超文本链接支持类。

类方法

S.No.	方法和描述
1	createFormulaEvaluator() 创建一个XSSFFormulaEvaluator例如，结果计算公式的单元格的对象。
2	createHyperlink(int type) Creates a new XSSFHyperlink.

对于其余的方法，请参考以下链接：

https://poi.apache.org/apidocs/org/apache/poi/xssf/usermodel/XSSFCreationHelper.html

PrintSetup接口（打印）

XSSFPrintSetup 核心API

这是在org.apache.poi.xsssf.usermodel包下的类。它实现了PrintSetup接口。它是用来设置打印页面大小，面积，选项和设置。

类方法

S.No.	方法及说明
1	setLandscape(boolean ls) 设置一个布尔值，允许或阻止横向打印。
2	setLeftToRight(boolean ltor) 设置是否向左走向右或自上而下的顺序，同时打印。
3	setPaperSize(short size) 设置纸张尺寸。

读取和重写工作簿

读取工作簿也是通过Workbook接口实现类来操纵。XSSFWorkbook(InputStream)构造方法参数接受一个输入流，可以将输入流读取到内存后进行操作。

范例：读取excel文件并另存文件

```
        public static void main(String[] args) throws Exception
        {
                FileInputStream fileInputStream = new
FileInputStream("res/temp/工作簿01.xlsx");// 将excel文件转为输入流
                Workbook workbook = new XSSFWorkbook(fileInputStream);
// XSSFWorkbook构造方法可以接受一个输入流做参数
                FileOutputStream out = new FileOutputStream("res/temp/
工作簿02.xlsx"); // 保存位置
                workbook.write(out); // 将工作簿内容写入到输出流
                workbook.close();
        }
```

范例：修改工作簿内的内容。只会重写修改的部分。

```
        public static void main(String[] args) throws Exception
        {
                InputStream inputStream = new
FileInputStream("res/temp/工作簿01.xlsx");
//              InputStream inputStream = new
FileInputStream("res/temp/工作簿01.xls");
                Workbook workBook =
WorkbookFactory.create(inputStream);
                Sheet sheet1 = workBook.getSheetAt(0);
                Row row = sheet1.getRow(0);
                Cell cell = row.createCell(0);
                cell.setCellType(CellType.STRING);
                cell.setCellValue("测试读取文件然后重新写入");

                workBook.write(new FileOutputStream("res/temp/工作簿
01.xlsx"));
                workBook.close();
        }
```

绘制图形

POI 支持调用MS-Office 绘图工具进行图形的绘制。一张sheet页上的图形是按照有层级的图形组和图形来安排的。最顶层的图形是patriarch，它在sheet页上是不可见的。在开始绘制图形之前，您需要在HSSFSheet 对象中调用createPatriarch方法。调用这个方法会清除在该sheet页中储存的所有的图形信息。默认情况下只要这个方法不被调用，POI不会将图形数据清除。 要创建一个图形，您需要做以下几步：

1. 创建一个patriarch 对象
2. 创建一个锚点以供图形在sheet页上定位
3. 调用patriarch对象创建一个图形
4. 设置图形类型（直线，椭圆，矩形等等）
5. 设置图形的其他样式细节（例如：线条粗细等等）

数据有效性校验

Alibaba easyExcel

https://github.com/alibaba/easyexcel

参考文档

https://wenku.baidu.com/view/0541b193daef5ef7ba0d3cb6.html

http://blog.csdn.net/hao_hl1314/article/details/58585234

https://wenku.baidu.com/view/9084ab0af78a6529647d5384.html

http://www.cnblogs.com/LiZhiW/p/4313789.html?utm_source=tuicool

http://blog.csdn.net/hehexiaoyou/article/category/854403