POI3.5 HSSF 和 XSSF Excel 操作快速入门

岑坚（高凯）翻译

2010-08-08

写在前面

想尽快的使用 HSSF 和 XSSF 对电子表格进行操作吗？这个指南正是您所需要的。现在稳定的 POI 的版本为 3.6。但最近在查阅 POI 的资料时发现，虽然资料很多，但是大都局限于 2.x 的版本，3.x 中文资料比较少，查阅 apache 网站的时候发现了这份文档，看着不错，就翻译了一下，希望能够对大家有所帮助。由于时间仓促难免有翻译不当之处，还望不吝赐教，费神指正，感激不尽！

岑坚（高凯）
cenjian@taobao.com
2010-08-08

目录

添加 POI 支持

鉴于现在大部分应用是以 Maven 作为构建，可在 pom.xml 中加入 POI 的 maven 依赖,但是 maven 现在只支持到 3.1 版本，对 3.5 版本还不提供支持。

```
<dependency>
   <groupId>poi</groupId>
   <artifactId>poi</artifactId>
   <version>3.1-FINAL</version>
</dependency>
```

因此可以直接在 Apache 官网下载 POI3.5 版本，作为 reference library 加入工程中

创建新工作簿

```
Workbook wb = new HSSFWorkbook();
FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();

Workbook wb = new XSSFWorkbook();
FileOutputStream fileOut = new
FileOutputStream("workbook.xlsx");
wb.write(fileOut);
fileOut.close();
```

创建新 sheet 页

```
Workbook wb = new HSSFWorkbook();
//Workbook wb = new XSSFWorkbook();
Sheet sheet1 = wb.createSheet("new sheet");
Sheet sheet2 = wb.createSheet("second sheet");
FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

创建单元格

```
Workbook wb = new HSSFWorkbook();
//Workbook wb = new XSSFWorkbook();
CreationHelper createHelper = wb.getCreationHelper();
Sheet sheet = wb.createSheet("new sheet");

//创建一列，在其中加入多个单元格，列索引号从 0 开始，单元格的索引号也是从 0
//开始
```

```
Row row = sheet.createRow((short)0);[1]
//创建一个单元格，并在其中加入内容.
Cell cell = row.createCell(0);
cell.setCellValue(1);

// 上面的多行代码可以采用下面的一行代码的方式完成
row.createCell(1).setCellValue(1.2);
row.createCell(2).setCellValue(
    createHelper.createRichTextString("This is a string"));
row.createCell(3).setCellValue(true);

//将输出流写入一个文件
FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

新建一个时间格式的单元格

```
//Workbook wb = new XSSFWorkbook();
CreationHelper createHelper = wb.getCreationHelper();
Sheet sheet = wb.createSheet("new sheet");

//创建一列，在其中加入多个单元格，列索引号从 0 开始，单元格的索引号也是从 0
//开始
Row row = sheet.createRow(0);

//创建一个单元格，并在其中加入内容.  第一个单元格的内容不设置为日期时间格式
Cell cell = row.createCell(0);
cell.setCellValue(new Date());

//我们将第二个单元格设置成日期（和时间）格式。对这个文件来说创建一个新的单
//元格格式非常重要，除非您能够不用内置的格式来渲染所有的单元格
CellStyle cellStyle = wb.createCellStyle();
cellStyle.setDataFormat(
createHelper.createDataFormat().getFormat("m/d/yy h:mm"));
cell = row.createCell(1);
cell.setCellValue(new Date());
cell.setCellStyle(cellStyle);

//也可以用 java.util.Calendar 来设置单元格格式
cell = row.createCell(2);
cell.setCellValue(Calendar.getInstance());
```

[1] 很多方法的参数是 short 而不是 int 所以需要做一次类型转换 --译者注

```
    cell.setCellStyle(cellStyle);

    // 将输出流写入一个文件
    FileOutputStream fileOut = new FileOutputStream("workbook.xls");
    wb.write(fileOut);
    fileOut.close();
```

处理不同内容格式的单元格

```
    Workbook wb = new HSSFWorkbook();
    Sheet sheet = wb.createSheet("new sheet");
    Row row = sheet.createRow((short)2);
    row.createCell(0).setCellValue(1.1);
    row.createCell(1).setCellValue(new Date());
    row.createCell(2).setCellValue(Calendar.getInstance());
    row.createCell(3).setCellValue("a string");
    row.createCell(4).setCellValue(true);
    row.createCell(5).setCellType(HSSFCell.CELL_TYPE_ERROR);

    //将输出流写入一个文件
    FileOutputStream fileOut = new FileOutputStream("workbook.xls");
    wb.write(fileOut);
    fileOut.close();
```

遍历列和单元格

有些时候您仅希望遍历一个 sheet 页中所有的列或是一列中所有的单元格。这可以用一个简单的 for 循环来实现。

幸运的是，这非常简单。Row 对象定义了一个 CellIterator 内部类用来处理单元格的遍历（可以通过调用 row.cellIterator() 来获得 Iterator 对象），另外,Sheet 对象提供了一个 rowIterator()方法对所有列进行遍历

除此之外，Sheet 和 Row 对象都实现了 java.lang.Iterable 接口，因此如果您用的是 Java 1.5 及以上版本，您可以简单的调用内置的“foreach”来实现。请看：

```
        Sheet sheet = wb.getSheetAt(0);
        for (Iterator<Row> rit = sheet.rowIterator(); rit.hasNext(); ) {
                Row row = rit.next();
                for (Iterator<Cell> cit = row.cellIterator(); cit.hasNext(); ) {
                        Cell cell = cit.next();
                        // Do something here
                }
```

```
        }
```

使用 Java1.5 foreach 循环遍历列和单元格

有时候您仅希望遍历一个 sheet 页中所有的列或是一列中所有的单元格。如果您使用的是 java1.5 或是以上版本，那么这将十分的方便，因为它支持新的 foreach 循环。

幸运的是，这非常的简单。Sheet 和 Row 对象都实现了 java.lang.Iterable 接口支持 foreach 循环。对 Row 对象来说它支持通过调用 CellIterator 内部类用来处理单元格的遍历，对于 Sheet 对象来说它提供了一个 rowIterator()方法对所有列进行遍历。

```
        Sheet sheet = wb.getSheetAt(0);
        for (Row row : sheet) {
                for (Cell cell : row) {
                        // Do something here
                }
        }
```

获得单元格内的内容

要想获得单元格内的内容，您首先要要知道单元格内容的格式（比如您想获得从字符格式的单元格内获得一个数字，您将得到的是 NumberFormatException 错误）。因此，您将希望能够切换到该单元格格式，然后为获得该单元格内容调用合适的 getter 方法。

在以下的代码中我们将对一个 sheet 页中的所有单元格进行遍历，并打印出单元格的引用和内容。

```
// import org.apache.poi.ss.usermodel.*;

Sheet sheet1 = wb.getSheetAt(0);
for (Row row : sheet1) {
        for (Cell cell : row) {
                CellReference          cellRef          =          new
CellReference(row.getRowNum(), cell.getCellNum());
                System.out.print(cellRef.formatAsString());
                System.out.print(" - ");

                switch(cell.getCellType()) {
     case Cell.CELL_TYPE_STRING:

System.out.println(cell.getRichStringCellValue().getString());
       break;
     case Cell.CELL_TYPE_NUMERIC:
       if(DateUtil.isCellDateFormatted(cell)) {
         System.out.println(cell.getDateCellValue());
       } else {
         System.out.println(cell.getNumericCellValue());
```

```
        }
        break;
      case Cell.CELL_TYPE_BOOLEAN:
        System.out.println(cell.getBooleanCellValue());
        break;
      case Cell.CELL_TYPE_FORMULA:
        System.out.println(cell.getCellFormula());
        break;
      default:
        System.out.println();
                }
        }
}
```

文本提取

对于大多数的文本提取需求，标准的 ExcelExtractor 类应该能够满足您所有的需求。

```
        InputStream inp = new FileInputStream("workbook.xls");
        HSSFWorkbook wb = new HSSFWorkbook(new POIFSFileSystem(inp));
        ExcelExtractor extractor = new ExcelExtractor(wb);

        extractor.setFormulasNotResults(true);
        extractor.setIncludeSheetNames(false);
        String text = extractor.getText();
```

对于特殊的文本提取，比如将 xls 文件内容写入 csv 文件，可以参考：*/src/examples/src/org/apache/poi/hssf/eventusermodel/examples/XLS2CSVmra.java* 文件

单元格的各种对齐方式

```
    public static void main(String[] args)  throws Exception {
        Workbook wb = new XSSFWorkbook(); //或者是 new HSSFWorkbook();

        Sheet sheet = wb.createSheet();
        Row row = sheet.createRow((short) 2);
        row.setHeightInPoints(30);

        createCell(wb, row, (short) 0, XSSFCellStyle.ALIGN_CENTER,
XSSFCellStyle.VERTICAL_BOTTOM);
        createCell(wb,          row,          (short)          1,
XSSFCellStyle.ALIGN_CENTER_SELECTION,
XSSFCellStyle.VERTICAL_BOTTOM);
        createCell(wb, row, (short) 2, XSSFCellStyle.ALIGN_FILL,
XSSFCellStyle.VERTICAL_CENTER);
```

```
        createCell(wb, row, (short) 3, XSSFCellStyle.ALIGN_GENERAL,
XSSFCellStyle.VERTICAL_CENTER);
        createCell(wb, row, (short) 4, XSSFCellStyle.ALIGN_JUSTIFY,
XSSFCellStyle.VERTICAL_JUSTIFY);
        createCell(wb, row, (short) 5, XSSFCellStyle.ALIGN_LEFT,
XSSFCellStyle.VERTICAL_TOP);
        createCell(wb, row, (short) 6, XSSFCellStyle.ALIGN_RIGHT,
XSSFCellStyle.VERTICAL_TOP);

        //将输出流写入一个文件
        FileOutputStream           fileOut           =           new
FileOutputStream("xssf-align.xlsx");
        wb.write(fileOut);
        fileOut.close();

    }

    /**
     *创建一个单元格并为其设定指定的对齐方式.
     *
     * @param wb      工作簿
     * @param row     单元格所在的列
     * @param column 单元格所在的行索引号
     * @param halign 单元格的水平对齐方式.
     */
    private static void createCell(Workbook wb, Row row, short column,
short halign, short valign) {
        Cell cell = row.createCell(column);
        cell.setCellValue(new XSSFRichTextString("Align It"));
        CellStyle cellStyle = wb.createCellStyle();
        cellStyle.setAlignment(halign);
        cellStyle.setVerticalAlignment(valign);
        cell.setCellStyle(cellStyle);
    }
```

如何处理边框

```
    Workbook wb = new HSSFWorkbook();
    Sheet sheet = wb.createSheet("new sheet");

   //创建一列，在其中加入多个单元格，列索引号从 0 开始，单元格的索引号也是从 0
    //开始.
    Row row = sheet.createRow(1);
```

```
//创建一个单元格，并在其中加入内容.
Cell cell = row.createCell(1);
cell.setCellValue(4);

//设置单元格边框为四周环绕.
CellStyle style = wb.createCellStyle();
style.setBorderBottom(CellStyle.BORDER_THIN);
style.setBottomBorderColor(IndexedColors.BLACK.getIndex());
style.setBorderLeft(CellStyle.BORDER_THIN);
style.setLeftBorderColor(IndexedColors.GREEN.getIndex());
style.setBorderRight(CellStyle.BORDER_THIN);
style.setRightBorderColor(IndexedColors.BLUE.getIndex());
style.setBorderTop(CellStyle.BORDER_MEDIUM_DASHED);
style.setTopBorderColor(IndexedColors.BLACK.getIndex());
cell.setCellStyle(style);

//将输出流写入一个文件
FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

填充色和颜色操作

```
Workbook wb = new XSSFWorkbook();
Sheet sheet = wb.createSheet("new sheet");

//创建一列，在其中加入多个单元格，列索引号从 0 开始，单元格的索引号也是从 0
//开始
Row row = sheet.createRow((short) 1);

// 浅绿色背景色
CellStyle style = wb.createCellStyle();
style.setFillBackgroundColor(IndexedColors.AQUA.getIndex());
style.setFillPattern(CellStyle.BIG_SPOTS);
Cell cell = row.createCell((short) 1);
cell.setCellValue("X");
cell.setCellStyle(style);

//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
style = wb.createCellStyle();
style.setFillForegroundColor(IndexedColors.ORANGE.getIndex());
style.setFillPattern(CellStyle.SOLID_FOREGROUND);
cell = row.createCell((short) 2);
cell.setCellValue("X");
```

```
cell.setCellStyle(style);

//将输出流写入一个文件
FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

合并单元格

```
Workbook wb = new HSSFWorkbook();
Sheet sheet = wb.createSheet("new sheet");

Row row = sheet.createRow((short) 1);
Cell cell = row.createCell((short) 1);
cell.setCellValue("This is a test of merging");

sheet.addMergedRegion(new CellRangeAddress(
        1, //first row (0-based)
        1, //last row  (0-based)
        1, //first column (0-based)
        2  //last column  (0-based)
));

//将输出流写入一个文件
FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

字体的处理

```
Workbook wb = new HSSFWorkbook();
Sheet sheet = wb.createSheet("new sheet");

//创建一列，在其中加入多个单元格，列索引号从 0 开始，单元格的索引号也是从 0
//开始
Row row = sheet.createRow(1);

// 创建一个字体并修改它的样式.
Font font = wb.createFont();
font.setFontHeightInPoints((short)24);
font.setFontName("Courier New");
font.setItalic(true);
font.setStrikeout(true);
```

```
//字体在样式（style）中载入才能使用，因此创建一个 style 来使用该字体
CellStyle style = wb.createCellStyle();
style.setFont(font);

//创建一个单元格，并在其中加入内容.
Cell cell = row.createCell(1);
cell.setCellValue("This is a test of fonts");
cell.setCellStyle(style);

//将输出流写入一个文件
FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

注意：

一个工作簿中最大的自定义字体数目为 32767（short 类型最大正数）。您应当复用这些字体，而不是为每一个单元格新建一个字体。比如：

错误的用法：

```
for (int i = 0; i < 10000; i++) {
    Row row = sheet.createRow(i);
    Cell cell = row.createCell((short) 0);

    CellStyle style = workbook.createCellStyle();
    Font font = workbook.createFont();
    font.setBoldweight(Font.BOLDWEIGHT_BOLD);
    style.setFont(font);
    cell.setCellStyle(style);
}
```

正确的用法：

```
CellStyle style = workbook.createCellStyle();
Font font = workbook.createFont();
font.setBoldweight(Font.BOLDWEIGHT_BOLD);
style.setFont(font);
for (int i = 0; i < 10000; i++) {
    Row row = sheet.createRow(i);
    Cell cell = row.createCell((short) 0);
    cell.setCellStyle(style);
}
```

自定义颜色

HSSF:

```
    HSSFWorkbook wb = new HSSFWorkbook();
    HSSFSheet sheet = wb.createSheet();
    HSSFRow row = sheet.createRow((short) 0);
    HSSFCell cell = row.createCell((short) 0);
    cell.setCellValue("Default Palette");

    //就像上面一个例子一样从标准调色板上获得几种颜色
    //我们将在石灰色的背景上应用红色字体

    HSSFCellStyle style = wb.createCellStyle();
    style.setFillForegroundColor(HSSFColor.LIME.index);
    style.setFillPattern(HSSFCellStyle.SOLID_FOREGROUND);

    HSSFFont font = wb.createFont();
    font.setColor(HSSFColor.RED.index);
    style.setFont(font);

    cell.setCellStyle(style);

    //替换保存为默认调色板
    FileOutputStream out = new
FileOutputStream("default_palette.xls");
    wb.write(out);
    out.close();

    //现在然我们把橙色和石灰色的组合替换成更有吸引力的组合
    // 还是比较喜欢 freebsd.org 提供的色彩方案

    cell.setCellValue("Modified Palette");

    //为该工作簿新建一个调色板
    HSSFPalette palette = wb.getCustomPalette();

    //替换标准红色为 freebsd.org 上的红色
    palette.setColorAtIndex(HSSFColor.RED.index,
            (byte) 153,  //RGB red (0-255)
            (byte) 0,    //RGB green
            (byte) 0     //RGB blue
    );
    //替换石灰色为 freebsd.org 上的金色
```

```
    palette.setColorAtIndex(HSSFColor.LIME.index, (byte) 255, (byte)
204, (byte) 102);

    //将其保存为“修改过的调色板”
    // 注意，不论我们以前在哪里用到了红色和石灰色，都会奇迹般的换上现在的颜色
    out = new FileOutputStream("modified_palette.xls");
    wb.write(out);
    out.close();
```

XSSF:

```
    XSSFWorkbook wb = new XSSFWorkbook();
    XSSFSheet sheet = wb.createSheet();
    XSSFRow row = sheet.createRow(0);
    XSSFCell cell = row.createCell( 0);
    cell.setCellValue("custom XSSF colors");

    XSSFCellStyle style1 = wb.createCellStyle();
    style1.setFillForegroundColor(new                XSSFColor(new
java.awt.Color(128, 0, 128)));
    style1.setFillPattern(CellStyle.SOLID_FOREGROUND);
```

读取和重写工作簿

```
    InputStream inp = new FileInputStream("workbook.xls");
    //InputStream inp = new FileInputStream("workbook.xlsx");

    Workbook wb = WorkbookFactory.create(inp);
    Sheet sheet = wb.getSheetAt(0);
    Row row = sheet.getRow(2);
    Cell cell = row.getCell(3);
    if (cell == null)
        cell = row.createCell(3);
    cell.setCellType(Cell.CELL_TYPE_STRING);
    cell.setCellValue("a test");

    //将输出流写入一个文件
    FileOutputStream fileOut = new FileOutputStream("workbook.xls");
    wb.write(fileOut);
    fileOut.close();
```

在单元格中使用换行

```
    Workbook wb = new XSSFWorkbook();   //or new HSSFWorkbook();
```

```
    Sheet sheet = wb.createSheet();

    Row row = sheet.createRow(2);
    Cell cell = row.createCell(2);
    cell.setCellValue("Use \n with word wrap on to create a new line");

    //为了能够使用换行，您需要设置单元格的样式 wrap=true
    CellStyle cs = wb.createCellStyle();
    cs.setWrapText(true);
    cell.setCellStyle(cs);

    //增加单元格的高度已能够容纳两行文字

row.setHeightInPoints((2*sheet.getDefaultRowHeightInPoints()));

    // 调整列宽以使用内容长度
    sheet.autoSizeColumn((short)2);

    FileOutputStream fileOut = new
FileOutputStream("ooxml-newlines.xlsx");
    wb.write(fileOut);
    fileOut.close();
```

创建用户自定义数据格式

```
    Workbook wb = new HSSFWorkbook();
    Sheet sheet = wb.createSheet("format sheet");
    CellStyle style;
    DataFormat format = wb.createDataFormat();
    Row row;
    Cell cell;
    short rowNum = 0;
    short colNum = 0;

    row = sheet.createRow(rowNum++);
    cell = row.createCell(colNum);
    cell.setCellValue(11111.25);
    style = wb.createCellStyle();
    style.setDataFormat(format.getFormat("0.0"));
    cell.setCellStyle(style);

    row = sheet.createRow(rowNum++);
    cell = row.createCell(colNum);
    cell.setCellValue(11111.25);
```

```
style = wb.createCellStyle();
style.setDataFormat(format.getFormat("#,##0.0000"));
cell.setCellStyle(style);

FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

Sheet 页自适应页面大小

```
Workbook wb = new HSSFWorkbook();
Sheet sheet = wb.createSheet("format sheet");
PrintSetup ps = sheet.getPrintSetup();

sheet.setAutobreaks(true);

ps.setFitHeight((short)1);
ps.setFitWidth((short)1);


//为电子表格创建多行多列.
    。。。。。。。
FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

设定打印区域

```
Workbook wb = new HSSFWorkbook();
Sheet sheet = wb.createSheet("Sheet1");
//为第一个 sheet 页设定打印区域
wb.setPrintArea(0, "$A$1:$C$2");

//也可以这样实现:
wb.setPrintArea(
       0, //sheet 页标
       0, //开始列标
       1, //结束列标
       0, //开始行标
       0  //结束行标
);

FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
```

```
    fileOut.close();
```

设置脚注页码

```
    HSSFWorkbook wb = new HSSFWorkbook();
    HSSFSheet sheet = wb.createSheet("format sheet");
    HSSFFooter footer = sheet.getFooter()

    footer.setRight( "Page " + HSSFFooter.page() + " of " +
HSSFFooter.numPages() );


    //为电子表格创建多行多列
    。。。。。。

    FileOutputStream fileOut = new FileOutputStream("workbook.xls");
    wb.write(fileOut);
    fileOut.close();
```

使用便捷函数

便捷函数存在于 contrib[2]中，并提供许多实用功能如不用明确的新建一个样式就能够为合并区域设置边框和改变样式。

```
    HSSFWorkbook wb = new HSSFWorkbook();
    HSSFSheet sheet1 = wb.createSheet( "new sheet" );

    //创建一个合并区域
    HSSFRow row = sheet1.createRow( (short) 1 );
    HSSFRow row2 = sheet1.createRow( (short) 2 );
    HSSFCell cell = row.createCell( (short) 1 );
    cell.setCellValue( "This is a test of merging" );
    Region region = new Region( 1, (short) 1, 4, (short) 4 );
    sheet1.addMergedRegion( region );

    // 设置边框和边框颜色
    final           short           borderMediumDashed           =
HSSFCellStyle.BORDER_MEDIUM_DASHED;
    HSSFRegionUtil.setBorderBottom( borderMediumDashed,
       region, sheet1, wb );
    HSSFRegionUtil.setBorderTop( borderMediumDashed,
       region, sheet1, wb );
```

[2]Contrib：外围爱好者根据需要自行编译并贡献的软件 —译者注

```
    HSSFRegionUtil.setBorderLeft( borderMediumDashed,
        region, sheet1, wb );
    HSSFRegionUtil.setBorderRight( borderMediumDashed,
        region, sheet1, wb );
    HSSFRegionUtil.setBottomBorderColor(HSSFColor.AQUA.index,
region, sheet1, wb);
    HSSFRegionUtil.setTopBorderColor(HSSFColor.AQUA.index,  region,
sheet1, wb);
    HSSFRegionUtil.setLeftBorderColor(HSSFColor.AQUA.index, region,
sheet1, wb);
    HSSFRegionUtil.setRightBorderColor(HSSFColor.AQUA.index, region,
sheet1, wb);

    //展示 HSSFCellUtil 的一些用法
    HSSFCellStyle style = wb.createCellStyle();
    style.setIndention((short)4);
    HSSFCellUtil.createCell(row, 8, "This is the value of the cell",
style);
    HSSFCell cell2 = HSSFCellUtil.createCell( row2, 8, "This is the
value of the cell");
    HSSFCellUtil.setAlignment(cell2,                              wb,
HSSFCellStyle.ALIGN_CENTER);

    //输出工作簿
    FileOutputStream fileOut = new FileOutputStream( "workbook.xls" );
    wb.write( fileOut );
    fileOut.close();
```

上下移动一行

```
        Workbook wb = new HSSFWorkbook();
        Sheet sheet = wb.createSheet("row sheet");

      //为电子表格创建多行多列
        ……
      // 将第 6 到 第 11 行移至顶端 第 0 到 第 5 行
        sheet.shiftRows(5, 10, -5);
```

将 sheet 页设定为默认选中

```
    Workbook wb = new HSSFWorkbook();
    Sheet sheet = wb.createSheet("row sheet");
    sheet.setSelected(true);
```

设置 sheet 页放大倍率

放大倍率是用分数来表示的，比如，如果放大倍率是 75%，那么可以用 3/4 来表示

```
Workbook wb = new HSSFWorkbook();
Sheet sheet1 = wb.createSheet("new sheet");
sheet1.setZoom(3,4);   // 75%放大倍率
```

拆分和冻结窗格

您可以创建两种窗格：冻结窗格和拆分窗格

冻结窗格是由行和列来分隔的。您可以采用以下的方式来创建冻结窗格：

```
sheet1.createFreezePane( 3, 2, 3, 2 );
```
[3]

The first two parameters are the columns and rows you wish to split by. The second two parameters indicate the cells that are visible in the bottom right quadrant.

前两个参数是您想分隔开的列数和行数，后两个参数表示在右下部分中可见的单元格。其中第三个参数是右边区域可见的左边列数，第四个参数是下面区域可见的首行。

拆分窗格看起来和冻结窗格很不一样。拆分窗格被分成了 4 个单独的工作区域。拆分是像素级别的，用户可以通过改变拆分点来调整拆分的效果。

拆分窗格可以通过调用以下方法实现：

```
sheet2.createSplitPane( 2000, 2000, 0, 0, Sheet.PANE_LOWER_LEFT );
```

第一个参数是拆分点在 X 轴上的坐标。单位是 1/20 个点。在这里可以把一个点看成一个像素。第二个参数是拆分点在 Y 轴上的坐标，单位同样也是 1/20 个点。第三个参数是右边区域可见的左边列数，第四个参数是下面区域可见的首行

最后一个参数指定了该拆分点落在了哪个窗格中。一共有四种选项 Sheet.PANE_LOWER_LEFT, PANE_LOWER_RIGHT, PANE_UPPER_RIGHT 或者 PANE_UPPER_LEFT.

```
Workbook wb = new HSSFWorkbook();
Sheet sheet1 = wb.createSheet("new sheet");
Sheet sheet2 = wb.createSheet("second sheet");
Sheet sheet3 = wb.createSheet("third sheet");
Sheet sheet4 = wb.createSheet("fourth sheet");

// 冻结第一行
sheet1.createFreezePane( 0, 1, 0, 1 );
```

[3] createFreezePane 方法有两种 **createFreezePane(int colSplit, int rowSplit, int leftmostColumn, int topRow) 和 createFreezePane(int colSplit, int rowSplit)** 在冻结整行整列时第二种方法更为简便 –译者注

```
// 冻结第一列
sheet2.createFreezePane( 1, 0, 1, 0 );
//冻结列和行（这里可以忽略右下方的滚动条的位置）
sheet3.createFreezePane( 2, 2 );
//拆分窗口并且使左下方有焦点
sheet4.createSplitPane( 2000, 2000, 0, 0, Sheet.PANE_LOWER_LEFT );

FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

重复列和行

在打印时，可以通过 HSSFWorkbook 类的 setRepeatingRowsAndColumns()方法设置重复的列和行。

这个方法包含五个参数。第一个参数是 sheet 页的索引（0 代表第一个 sheet 页）。第二和第三个参数指定了重复列的范围，如果不想重复列的话，可以用-1 代替。第四和第五个参数指定了重复行的范围，同样如果不想重复行可以用-1 替代起止索引号。

```
Workbook wb = new HSSFWorkbook();
Sheet sheet1 = wb.createSheet("new sheet");
Sheet sheet2 = wb.createSheet("second sheet");

//设置第一张 sheet 页中的第一到第三列重复
wb.setRepeatingRowsAndColumns(0,0,2,-1,-1);
// 为第二张 sheet 页设置重复列和行
wb.setRepeatingRowsAndColumns(1,4,5,1,2);

FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

页眉和页脚

这是一个页眉的例子，但页脚也同样适用

```
Workbook wb = new HSSFWorkbook();
Sheet sheet = wb.createSheet("new sheet");

Header header = sheet.getHeader();
header.setCenter("Center Header");
header.setLeft("Left Header");
header.setRight(HSSFHeader.font("Stencil-Normal", "Italic") +
```

```
                HSSFHeader.fontSize((short)  16)  +  "Right  w/
Stencil-Normal Italic font and size 16");

    FileOutputStream fileOut = new FileOutputStream("workbook.xls");
    wb.write(fileOut);
    fileOut.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）

```
    HSSFPatriarch patriarch = sheet.createDrawingPatriarch();

    a = new HSSFClientAnchor( 0, 0, 1023, 255, (short) 1, 0, (short)
1, 0 );
    HSSFSimpleShape shape1 = patriarch.createSimpleShape(a1);
    shape1.setShapeType(HSSFSimpleShape.OBJECT_TYPE_LINE);
```

文本框的创建方式如下：

```
    HSSFTextbox textbox1 = patriarch.createTextbox(
            new HSSFClientAnchor(0,0,0,0,(short)1,1,(short)2,2));
    textbox1.setString(new HSSFRichTextString("This is a test") );
```

在同一文本框中设置不同的字体样式也是可以实现的

```
    HSSFFont font = wb.createFont();
    font.setItalic(true);
    font.setUnderline(HSSFFont.U_DOUBLE);
    HSSFRichTextString string = new HSSFRichTextString("Woo!!!");
```

```
string.applyFont(2,5,font);
textbox.setString(string );
```

Just as can be done manually using Excel, it is possible to group shapes together. This is done by calling `createGroup()` and then creating the shapes using those groups.

如同 Excel 中可以手动设置组一样，调用 `createGroup（）`方法，用新建的组创建图形也可以实现。组中也可以嵌套其他组。

注意

任何一个组中必须至少包含两个不同图形或是其他的子组。

创建图片组可以参考下面的例子：

```
// 创建一个图片组.
HSSFShapeGroup group = patriarch.createGroup(
new HSSFClientAnchor(0,0,900,200,(short)2,2,(short)2,2));

//在图片组中创建多条直线.
HSSFSimpleShape shape1 = group.createShape(new
HSSFChildAnchor(3,3,500,500));
shape1.setShapeType(HSSFSimpleShape.OBJECT_TYPE_LINE);
( (HSSFChildAnchor)
shape1.getAnchor() ).setAnchor((short)3,3,500,500);
HSSFSimpleShape shape2 = group.createShape(new
HSSFChildAnchor((short)1,200,400,600));
shape2.setShapeType(HSSFSimpleShape.OBJECT_TYPE_LINE);
```

如果您足够细心，您将会发下，在图形组中添加的图形使用的是一个新类型的锚点对象 `HSSFChildAnchor`。这是因为新创建的组会有自己的坐标空间。在 `POI` 中，这个 坐标空间默认为(0,0,1023,255)，但是您也可以根据自己的意愿进行修改。

```
myGroup.setCoordinates(10,10,20,20); // top-left, bottom-right
```

如果您在一个图片组中加入了另外一个图片组，同样该子图片组也有自己的坐标空间。

设置图形样式

默认的图片演示看起来有些平常。为图形设置不同的样式也可以通过 POI 来实现。不过现在仅能支持以下几种图形样式改变：

- 变换填充色

- 去除填充色
- 改变线条粗细
- 改变线条样式 比如：虚线，点线等等
- 改变线条颜色

以下是如何实现

```
HSSFSimpleShape s = patriarch.createSimpleShape(a);
s.setShapeType(HSSFSimpleShape.OBJECT_TYPE_OVAL);
s.setLineStyleColor(10,10,10);
s.setFillColor(90,10,200);
s.setLineWidth(HSSFShape.LINEWIDTH_ONE_PT * 3);
s.setLineStyle(HSSFShape.LINESTYLE_DOTSYS);
```

图形和 Graphics2d 类

虽然推荐使用本地 POI 命名来创建图形，但有时我们也希望调用标准 AIP 获得和外部库的兼容性。处于这目的我们对 Graphics 和 Graphics2d 类进行了扩充。

注意：

在开始使用之前，您需要知道 Graphics2d 对 MS-office 绘图命令的支持不是很好，虽然 Graphics 类比起来对 MS-office 绘图命令更加兼容，但仍然差强人意。

所有的绘图指令都封装在 HSSFShapeGroup 类中。以下是它们的用法：

```
    a = new HSSFClientAnchor( 0, 0, 1023, 255, (short) 1, 0, (short) 1, 0 );
    group = patriarch.createGroup( a );
    group.setCoordinates( 0, 0, 80 * 4 , 12 * 23  );
    float verticalPointsPerPixel = a.getAnchorHeightInPoints(sheet) / (float)Math.abs(group.getY2() - group.getY1());
    g = new EscherGraphics( group, wb, Color.black, verticalPointsPerPixel );
    g2d = new EscherGraphics2d( g );
    drawChemicalStructure( g2d );
```

首先我们做的是为我们将要进行的绘制创建一个组并设置它的坐标。接下来，我们计算出一个合理的 fontSizeMultipler 创建一个 EscherGraphics 对象。因为我们想得到的是 Graphics2d 对象，所以我们我们新建一个 EscherGraphics2d 对象并把它传入一个创建好的 Graphics 对象中。最后我们会通过一系列流程将它绘制成一个 EscherGraphics2d 对象。

在垂直方向上每个点的像素值还需要再深入说明一下。在将 Graphics 对象调用转化成

Escher 对象调用中遇到的一个困难是，在 Excel 中没有明确的对绝对位置的像素定义。在 Excel 中，单元格的宽度是用“字符”宽度，高度是用“点”来度量的。不幸的是，在 Excel 中并没有明确的定义被用来度量的字符的类型。估计是由于 Excel 想在不同的平台上或是在同一平台上使用不同的字体造成的。

因为这些限制，我们必须要实现一个标准 verticalPointsPerPixel 用来度量每个垂直方向点所占的像素。当您调用像是 drawString（）这样的方法时，所用到的字体的 verticalPointsPerPixel 度量值必须明确。

计算 verticalPointsPerPixel 值的公式如下：

```
multipler = groupHeightInPoints / heightOfGroup
```

图形组的高度可以通过简单的计算不同图形边界盒的 Y 轴坐标来得到。 获得图新组的高度也可以简单的调用以下这个方法：

```
HSSFClientAnchor.getAnchorHeightInPoints().
```

许多 Graphic 类支持的方法还没有实现，现在已知实现的方法是：

- fillRect()
- fillOval()
- drawString()
- drawOval()
- drawLine()
- clearRect()

现在还不支持的方法会调用 POI 日志机制返回并记录日志信息（默认为 disabled）

提纲

提纲对将不同部分的信息聚合在一起非常重要。它可以用 POI 提供的 API 简单的这样实现：

```
Workbook wb = new HSSFWorkbook();
Sheet sheet1 = wb.createSheet("new sheet");

sheet1.groupRow( 5, 14 );
sheet1.groupRow( 7, 14 );
sheet1.groupRow( 16, 19 );

sheet1.groupColumn( (short)4, (short)7 );
sheet1.groupColumn( (short)9, (short)12 );
sheet1.groupColumn( (short)10, (short)11 );

FileOutputStream fileOut = new FileOutputStream(filename);
```

```
    wb.write(fileOut);
    fileOut.close();
```

折叠（或展开）一个提纲，可以用以下的方法：

```
   sheet1.setRowGroupCollapsed( 7, true );
    sheet1.setColumnGroupCollapsed( (short)4, true );
```

被选中的行或列必须包含一个已经创建好的组，可以是组内的任意位置。

图像处理

图像是绘图支持的一部分。提价一张图片可以用最高级别的 Drawing 对象调用 createPicture()。现在支持的图片格式有：

- PNG
- JPG
- DIB

应当值得注意的是任何已经存在的绘图可能会因为您添加一张图片到 sheet 页上而被清除

插入图片

```
    //创建一个新的工作簿
    Workbook wb = new XSSFWorkbook(); //or new HSSFWorkbook();

    //为工作簿添加图片.
    InputStream is = new FileInputStream("image1.jpeg");
    byte[] bytes = IOUtils.toByteArray(is);
    int pictureIdx = wb.addPicture(bytes,
Workbook.PICTURE_TYPE_JPEG);
    is.close();

   CreationHelper helper = wb.getCreationHelper();

    //创建 sheet 页
    Sheet sheet = wb.createSheet();

    // 创建顶层的 Drawing 对象，它是所有图形和图像的载体
    Drawing drawing = sheet.createDrawingPatriarch();

    //添加一个图片图形
    ClientAnchor anchor = helper.createClientAnchor();
```

```
//设置图形左上角的位置
//然后调用 Picture 的 resize() 方法，自动关联到新坐标
anchor.setCol1(3);
anchor.setRow1(2);
Picture pict = drawing.createPicture(anchor, pictureIdx);

//自动关联到新坐标
pict.resize();

//保存到工作簿
String file = "picture.xls";
if(wb instanceof XSSFWorkbook) file += "x";
FileOutputStream fileOut = new FileOutputStream(file);
wb.write(fileOut);
fileOut.close();
```

注意

Picture.resize() 现仅支持 JPEG 和 PNG 格式的图片，其他格式暂不支持

从工作簿中读取图片

```
List lst = workbook.getAllPictures();
for (Iterator it = lst.iterator(); it.hasNext(); ) {
    PictureData pict = (PictureData)it.next();
    String ext = pict.suggestFileExtension();
    byte[] data = pict.getData();
    if (ext.equals("jpeg")){
      FileOutputStream out = new FileOutputStream("pict.jpg");
      out.write(data);
      out.close();
    }
}
```

关联范围和关联单元格

关联范围是用一个名字代表一组单元格的方式。关联单元格是关联范围的下一级一组单元格中的一个。您可以用他们的关联范围创建或是得到对应的单元格。当要处理关联范围时可以用 org.apache.poi.hssf.util.CellReference 和 org.apache.poi.hssf.util.AreaReference 两个类（除了包名不一样外，在 XSSF 和 HSSF 中同样有这两个类）

创建管理范围/关联单元格

```
// 准备数据
String sname = "TestSheet", cname = "TestName", cvalue = "TestVal";
Workbook wb = new HSSFWorkbook();
Sheet sheet = wb.createSheet(sname);
sheet.createRow(0).createCell((short) 0).setCellValue(cvalue);

//1.用空间引用为一个单元格创建关联范围
Name namedCell = wb.createName();
namedCell.setNameName(cname);
String reference = sname+"!A1:A1"; // 空间引用
namedCell.setRefersToFormula(reference);

// 2.用单元格引用为一个单元格创建关联范围
Name namedCel2 = wb.createName();
namedCel2.setNameName(cname);
String reference = sname+"!A1"; //单元格引用
namedCel2.setRefersToFormula(reference);

// 3. 用空间引用为一组单元格创建关联范围
Name namedCel3 = wb.createName();
namedCel3.setNameName(cname);
String reference = sname+"!A1:C5"; // 空间引用
namedCel3.setRefersToFormula(reference);

// 4. 创建关联方程
Name namedCel4 = wb.createName();
namedCel4.setNameName("my_sum");
namedCel4.setRefersToFormula("SUM(sname+!$I$2:$I$6)");
```

从关联范围/关联单元格读取数据

```
// 准备数据
String cname = "TestName";
Workbook wb = getMyWorkbook(); // 获得工作簿

//获得关联范围
int namedCellIdx = wb.getNameIndex(cellName);
Name aNamedCell = wb.getNameAt(namedCellIdx);

//在关联范围中获得单元格并检测它的内容
   AreaReference aref = new
AreaReference(aNamedCell.getRefersToFormula());
CellReference[] crefs = aref.getAllReferencedCells();
for (int i=0; i<crefs.length; i++) {
```

```
        Sheet s = wb.getSheet(crefs[i].getSheetName());
        Row r = sheet.getRow(crefs[i].getRow());
        Cell c = r.getCell(crefs[i].getCol());
        //根据单元格数据格式取出单元格内容。。。。。。.
    }
```

从不连续的关联范围中读取数据

```
    // 准备数据
    String cname = "TestName";
    Workbook wb = getMyWorkbook(); // retrieve workbook

    // 获得关联范围
    // Will be something like "$C$10,$D$12:$D$14";
    int namedCellIdx = wb.getNameIndex(cellName);
    Name aNamedCell = wb.getNameAt(namedCellIdx);

    //在关联范围中获得单元格并检测它的内容
    //将会返回 C10 单元格和 D12 到 D14 单元格的空间引用
    AreaReference[] arefs =
AreaReference.generateContiguous(aNamedCell.getRefersToFormula())
;
    for (int i=0; i<arefs.length; i++) {
        // Only get the corners of the Area
        // (use arefs[i].getAllReferencedCells() to get all cells)
        CellReference[] crefs = arefs[i].getCells();
        for (int j=0; j<crefs.length; j++) {
            // Check it turns into real stuff
            Sheet s = wb.getSheet(crefs[j].getSheetName());
            Row r = s.getRow(crefs[j].getRow());
            Cell c = r.getCell(crefs[j].getCol());
            // 对此单元格进行相应的处理。。。。。
        }
    }
```

注意：

当单元格被删除的时候，Excel 并不会删除和它关联的关联范围。因此，在工作簿中会存在关联范围指向不存在的单元格的情况。在创建空间引用时应该先验证两者之间的关联引用是否正确。

```
    if(name.isDeleted()){
      //关联范围指向一个不存在的单元格
    } else {
      AreaReference ref = new
AreaReference(name.getRefersToFormula());
    }
```

为单元格添加注释

注释是不同于单元格的内容是附属于或者从属与单元格的富文本。注释的内容是独立于单元格独自存储的，并且是在绘制对象中展示的（比如一个文本框），因此它既独立于单元格同时又与单元格紧密联系。

添加注释

```
    Workbook wb = new XSSFWorkbook(); //or new HSSFWorkbook();

    CreationHelper factory = wb.getCreationHelper();

    Sheet sheet = wb.createSheet();

    Cell cell = sheet.createRow(3).createCell(5);
    cell.setCellValue("F4");

    Drawing drawing = sheet.createDrawingPatriarch();

    ClientAnchor anchor = factory.createClientAnchor();
    Comment comment = drawing.createCellComment(anchor);
    RichTextString   str   =   factory.createRichTextString("Hello,
World!");
    comment.setString(str);
    comment.setAuthor("Apache POI");
    //将注释注册到单元格
    cell.setCellComment(comment);

    String fname = "comment-xssf.xls";
    if(wb instanceof XSSFWorkbook) fname += "x";
    FileOutputStream out = new FileOutputStream(fname);
    wb.write(out);
    out.close();
```

读取单元格的注释

```
    Cell cell = sheet.get(3).getColumn((short)1);
    Comment comment = cell.getCellComment();
    if (comment != null) {
      RichTextString str = comment.getString();
      String author = comment.getAuthor();
    }
    //  另外您也可以用 sheet.getCellComment(row, column)来获取内容
    comment = sheet.getCellComment(3, 1);
```

根据内容调整单元格的宽度

```
Sheet sheet = workbook.getSheetAt(0);
    sheet.autoSizeColumn((short)0); //调整第一列的宽度
    sheet.autoSizeColumn((short)1); //调整第二列的宽度
```

注意：
因为 HSSFSheet.autoSizeColumn 使用 Java2D 类来计算列的宽度，因此在图形环境（graphical environment）不可用的情况下，调用会报错。如果图形环境不可用的情况下，您必须告诉 java 您使用的是 headless 模式，请设置以下系统属性：
java.awt.headless=true .

超级链接

获取超级链接

```
    Sheet sheet = workbook.getSheetAt(0);

    Cell cell = sheet.getRow(0).getCell((short)0);
    Hyperlink link = cell.getHyperlink();
    if(link != null){
        System.out.println(link.getAddress());
    }
```

创建超级链接

```
    Workbook wb = new XSSFWorkbook(); //or new HSSFWorkbook();
    CreationHelper createHelper = wb.getCreationHelper();

    //设置单元格格式为超级链接
    //默认情况下超级链接为下划线、蓝色字体
    CellStyle hlink_style = wb.createCellStyle();
    Font hlink_font = wb.createFont();
    hlink_font.setUnderline(Font.U_SINGLE);
    hlink_font.setColor(IndexedColors.BLUE.getIndex());
    hlink_style.setFont(hlink_font);

    Cell cell;
    Sheet sheet = wb.createSheet("Hyperlinks");
    //URL
    cell = sheet.createRow(0).createCell((short)0);
    cell.setCellValue("URL Link");

    Hyperlink                          link                          =
```

```
createHelper.createHyperlink(Hyperlink.LINK_URL);
    link.setAddress("http://poi.apache.org/");
    cell.setHyperlink(link);
    cell.setCellStyle(hlink_style);

    //关联到当前目录的文件
    cell = sheet.createRow(1).createCell((short)0);
    cell.setCellValue("File Link");
    link = createHelper.createHyperlink(Hyperlink.LINK_FILE);
    link.setAddress("link1.xls");
    cell.setHyperlink(link);
    cell.setCellStyle(hlink_style);

    //e-mail 关联
    cell = sheet.createRow(2).createCell((short)0);
    cell.setCellValue("Email Link");
    link = createHelper.createHyperlink(Hyperlink.LINK_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(hlink_style);

    //关联到工作簿中的位置

    //创建一个目标 sheet 页和目标单元格
    Sheet sheet2 = wb.createSheet("Target Sheet");
    sheet2.createRow(0).createCell((short)0).setCellValue("Target
Cell");

    cell = sheet.createRow(3).createCell((short)0);
    cell.setCellValue("Worksheet Link");
    Hyperlink                          link2                          =
createHelper.createHyperlink(Hyperlink.LINK_DOCUMENT);
    link2.setAddress("'Target Sheet'!A1");
    cell.setHyperlink(link2);
    cell.setCellStyle(hlink_style);

    FileOutputStream out = new FileOutputStream("hyperinks.xlsx");
    wb.write(out);
    out.close();
```

数据验证

注意：

最新的 3.5 版本，XSSF 数据流不支持数据验证而且 HSSF 数据流也不支持从 sheet 页中恢复数据的验证

检测用户输入是否与预定义的值相悖

以下的代码中，见限制用户在 A1 单元格中输入的数值仅为 10,20,30 中的一个

```
 HSSFWorkbook workbook = new HSSFWorkbook();
 HSSFSheet sheet = workbook.createSheet("Data Validation");
 CellRangeAddressList addressList = new CellRangeAddressList(
     0, 0, 0, 0);
 DVConstraint                    dvConstraint                    =
DVConstraint.createExplicitListConstraint(
     new String[]{"10", "20", "30"});
 HSSFDataValidation dataValidation = new HSSFDataValidation
     (addressList, dvConstraint);
 datavalidation.setSuppressDropDownArrow(true);
 sheet.addValidationData(dataValidation);
```

下拉列表

以下的代码将完成相同的功能，但是提供一个下拉列表供用户选择

```
 HSSFWorkbook workbook = new HSSFWorkbook();
 HSSFSheet sheet = workbook.createSheet("Data Validation");
 CellRangeAddressList addressList = new CellRangeAddressList(
     0, 0, 0, 0);
 DVConstraint                    dvConstraint                    =
DVConstraint.createExplicitListConstraint(
     new String[]{"10", "20", "30"});
 HSSFDataValidation dataValidation = new HSSFDataValidation
     (addressList, dvConstraint);
 datavalidation.setSuppressDropDownArrow(false);
 sheet.addValidationData(dataValidation);
```

错误提示信息

当用户输入非法值是弹出一个消息窗（Message box）告知用户

```
dataValidation.setErrorStyle(HSSFDataValidation.ErrorStyle.STOP);
 dataValidation.createErrorBox("Box Title", "Message Text");
```

替换“Box Title”为您想在消息框标题栏所显示的标题名称，替换“Message Text”为您的错误提示信息

提示框（Prompt）

当需要验证的单元格被选中时弹出提示框（Prompt）提醒用户

```
dataValidation.createPromptBox("Title", "Message Text");
dataValidation.setShowPromptBox(true);
```

createPromptBox() 中的第一个参数是提示框的标题，第二个参数是提示信息。DVConstraint 对象的 createExplicitListConstraint() 传入的参数可以是表示 Integer，floating point, date 或是文本的 String 数组。

高级数据校验

对输入的数据进行校验比如输入一个在 10 到 100 之间整型的值可以用

DVConstraint.createNumericConstraint(int, int, String, String)这个工厂方法

```
dvConstraint = DVConstraint.createNumericConstraint(
  DVConstraint.ValidationType.INTEGER,
  DVConstraint.OperatorType.BETWEEN, "10", "100");
```

如果查阅一下 javaDoc 可以看到其他的校验方法和可以支持的类型。请记住不是所有的类型都支持校验 。通过两个 String 类型传入的参数可以是公式。“=”是一个公式的标志。

```
dvConstraint = DVConstraint.createNumericConstraint(
  DVConstraint.ValidationType.INTEGER,
  DVConstraint.OperatorType.BETWEEN, "=SUM(A1:A3)", "100");
```

如果 createNumericConstraint()方法已经被调用，那将不能够创建一个下拉列表。同时 setSuppressDropDownArrow(false)的调用也将被忽略。 时间和日期的约束可以通过调用 createDateConstraint(int, String, String, String) 或是 createTimeConstraint(int, String, String)来实现。两者的调用与上面的方法类似，javaDoc 中有很详细的解释。

用单元格中数据创建数据校验值

特定的单元格的内容可以为数据校验提供校验值。DVConstraint.createFormulaListConstraint(String)方法对这种校验方式提供支持。指定从连续的单元格取得的内容作为校验值可以采用以下两种方法中的一种：

```
dvConstraint = DVConstraint.createFormulaListConstraint("$A$1:$A$3");
```

或者

```
HSSFNamedRange namedRange = workbook.createName();
namedRange.setNameName("list1");
namedRange.setRefersToFormula("$A$1:$A$3");
dvConstraint = DVConstraint.createFormulaListConstraint("list1");
```

在上面两个例子中，用户将可以从下拉列表中选择从 A1A2A3 单元格中取到的值。

所有的数据不一定都是校验值。然而从不同的 sheet 页中取得数据的话，这个 sheet 页在创建的时候必须别赋予一个 sheet 名，并且这个名字应该在方程式中使用。那么，假设存在一个名字是“Data Sheet”的 sheet 页，那可以这样操作：

```
HSSFNamedRange namedRange = workbook.createName();
namedRange.setNameName("list1");
namedRange.setRefersToFormula("'Data Sheet'!$A$1:$A$3");
dvConstraint = DVConstraint.createFormulaListConstraint("list1");
```

这样也是可以的:

```
dvConstraint = DVConstraint.createFormulaListConstraint("'Data Sheet'!$A$1:$A$3");
```

但这样是不行的:

```
HSSFNamedRange namedRange = workbook.createName();
namedRange.setNameName("list1");
namedRange.setRefersToFormula("'Sheet1'!$A$1:$A$3");
dvConstraint = DVConstraint.createFormulaListConstraint("list1");
```

这样也是不行的：

```
dvConstraint = DVConstraint.createFormulaListConstraint("'Sheet1'!$A$1:$A$3");
```

嵌入其他资源对象

可以对嵌入的 Excel、word、PowerPoint 文档或者其他类型的嵌入对象进行更加进行的操作。

HSSF:

```
POIFSFileSystem fs = new POIFSFileSystem(new FileInputStream("excel_with_embeded.xls"));
HSSFWorkbook workbook = new HSSFWorkbook(fs);
for (HSSFObjectData obj : workbook.getAllEmbeddedObjects()) {
    //该对象的 OLE2 类名
```

```
     String oleName = obj.getOLE2ClassName();
     if (oleName.equals("Worksheet")) {
        DirectoryNode dn = (DirectoryNode) obj.getDirectory();
        HSSFWorkbook embeddedWorkbook = new HSSFWorkbook(dn, fs,
false);
        //System.out.println(entry.getName()    +    ":    "    +
embeddedWorkbook.getNumberOfSheets());
     } else if (oleName.equals("Document")) {
        DirectoryNode dn = (DirectoryNode) obj.getDirectory();
        HWPFDocument embeddedWordDocument = new HWPFDocument(dn,
fs);
        //System.out.println(entry.getName()    +    ":    "    +
embeddedWordDocument.getRange().text());
     }  else if (oleName.equals("Presentation")) {
        DirectoryNode dn = (DirectoryNode) obj.getDirectory();
        SlideShow embeddedPowerPointDocument = new SlideShow(new
HSLFSlideShow(dn, fs));
        //System.out.println(entry.getName()    +    ":    "    +
embeddedPowerPointDocument.getSlides().length);
     } else {
        if(obj.hasDirectoryEntry()){
      // 文件夹实体类（DirectoryEntry）是一个文档节点（ DocumentNode）
      // 检查它的实体，获得它的内容
           DirectoryNode dn = (DirectoryNode) obj.getDirectory();
           for    (Iterator    entries    =    dn.getEntries();
entries.hasNext();) {
              Entry entry = (Entry) entries.next();
              //System.out.println(oleName      +      "."      +
entry.getName());
           }
        } else {
           // 如果没有文件夹实体类（DirectoryEntry）
           //从 HSSFObjectData 对象中恢复嵌入文档的数据
           byte[] objectData = obj.getObjectData();
        }
     }
  }
```

XSSF:

```
  XSSFWorkbook              workbook              =              new
XSSFWorkbook("excel_with_embeded.xlsx");
  for (PackagePart pPart : workbook.getAllEmbedds()) {
     String contentType = pPart.getContentType();
```

```
    //Excel 工作簿 二进制或 OpenXML 格式
    if (contentType.equals("application/vnd.ms-excel")) {
        HSSFWorkbook embeddedWorkbook = new
HSSFWorkbook(pPart.getInputStream());
    }
    //Excel 工作簿 OpenXML 格式
    else if
(contentType.equals("application/vnd.openxmlformats-officedocumen
t.spreadsheetml.sheet")) {
        OPCPackage docPackage =
OPCPackage.open(pPart.getInputStream());
        XSSFWorkbook embeddedWorkbook = new
XSSFWorkbook(docPackage);
    }
    // word 文档- 二进制 (OLE2CDF) 文件格式
    else if (contentType.equals("application/msword")) {
        HWPFDocument document = new
HWPFDocument(pPart.getInputStream());
    }
    // word 文档 - OpenXML 文件格式
    else if
(contentType.equals("application/vnd.openxmlformats-officedocumen
t.wordprocessingml.document")) {
        OPCPackage docPackage =
OPCPackage.open(pPart.getInputStream());
        XWPFDocument document = new XWPFDocument(docPackage);
    }
    //PPT 文档 –二进制文件格式
    else if (contentType.equals("application/vnd.ms-powerpoint"))
{
        HSLFSlideShow slideShow = new
HSLFSlideShow(pPart.getInputStream());
    }
    // PPT 文档 - OpenXML 文件格式
    else if
(contentType.equals("application/vnd.openxmlformats-officedocumen
t.presentationml.presentation")) {
        OPCPackage docPackage =
OPCPackage.open(pPart.getInputStream());
        XSLFSlideShow slideShow = new XSLFSlideShow(docPackage);
    }
    //其他的嵌入文件格式.
    else {
        System.out.println("Unknown Embedded Document: " +
```

```
contentType);
        InputStream inputStream = pPart.getInputStream();
    }
  }
```