# **HSSF XSSF SXSSF**

# 1、POI操作EXCEL工作簿的三种类型（都是实现Workbook接口）：

* HSSFWorkbook
* XSSFWorkbook
* SXSSFWorkbook

# 2、一般使用流程

1.创建工作簿Workbook  
2.创建Sheet  
3.创建行Row  
4.创建单元格Cell

# 3、HSSFWorkbook

HSSFWorkbook是操作Excel2003以前（包括2003）的版本，扩展名为.xls，所以每个Sheet局限就是导出的行数至多为65535行，一般不会发生内存不足的情况（OOM）。

# 4、XSSFWorkbook

这种形式的出现是由于HSSFWorkbook的局限性而产生的，因为其所导出的行数比较少，并且只针对Excel2003以前（包括2003）的版本的版本，所以 XSSFWookbook应运而生，其对应的是EXCEL2007以后的版本(1048576行，16384列)扩展名.xlsx，每个Sheet最多可以导出104万行，不过这样就伴随着一个OOM内存溢出的问题，原因是你所创建的sheet row cell 等此时是存在内存中的，随着数据量增大 ，内存的需求量也就增大，那么很大可能就是要OOM了。

# 5、对于不同版本的EXCEL文档要使用不同的工具类，如果使用错了程序会出现异常。

# 6、SXSSFWorkbook

从POI 3.8版本开始，提供了一种基于XSSFWorkbook的低内存占用的工作簿SXSSFWorkbook。

#### SXSSF (Streaming Usermodel API)

SXSSF (package: org.apache.poi.xssf.streaming) is an API-compatible streaming extension of XSSF to be used when very large spreadsheets have to be produced, and heap space is limited. SXSSF achieves its low memory footprint by limiting access to the rows that are within a sliding window, while XSSF gives access to all rows in the document. Older rows that are no longer in the window become inaccessible, as they are written to the disk.  
You can specify the window size at workbook construction time via new SXSSFWorkbook(int windowSize) or you can set it per-sheet via SXSSFSheet#setRandomAccessWindowSize(int windowSize)  
When a new row is created via createRow() and the total number of unflushed records would exceed the specified window size, then the row with the lowest index value is flushed and cannot be accessed via getRow() anymore.  
The default window size is 100 and defined by SXSSFWorkbook.DEFAULT\_WINDOW\_SIZE.  
A windowSize of -1 indicates unlimited access. In this case all records that have not been flushed by a call to flushRows() are available for random access.  
Note that SXSSF allocates temporary files that you **must** always clean up explicitly, by calling the dispose method.  
SXSSFWorkbook defaults to using inline strings instead of a shared strings table. This is very efficient, since no document content needs to be kept in memory, but is also known to produce documents that are incompatible with some clients. With shared strings enabled all unique strings in the document has to be kept in memory. Depending on your document content this could use a lot more resources than with shared strings disabled.  
Please note that there are still things that still may consume a large amount of memory based on which features you are using, e.g. merged regions, hyperlinks, comments, ... are still only stored in memory and thus may require a lot of memory if used extensively.  
Carefully review your memory budget and compatibility needs before deciding whether to enable shared strings or not.

引用官方的介绍，简单概括就是：  
SXSSF是对XSSF的一种流式扩展，**特点是采用了滑动窗口的机制，低内存占用**，主要用于数据量非常大的电子表格而虚拟机堆有限的情况。

原理是利用了滑动窗口机制。  
SXSSFWorkbook.DEFAULT\_WINDOW\_SIZE默认值是100，表示在内存中最多存在100个Row对象，当写第101个Row对象的时候就会把第1个Row对象以XML格式写入C:\Users\wange\AppData\Local\Temp路径下的临时文件中，后面的以此类推，始终保持内存中最多存在100个Row对象。

SXSSFWorkbook默认使用内联字符串而不是[共享字符串表](https://technet.microsoft.com/zh-cn/library/gg278314" \t "/Users/JustWyx/Documents\\x/_blank)(SharedStringsTable)。启用共享字符串时，文档中的所有唯一字符串都必须保存在内存中，因此会占用更多的内存。

/\*\*

\* workbook - 模板工作簿

\* rowAccessWindowSize - 保存在内存中，直到刷新的行数。

\* compressTmpFiles - 是否对临时文件使用gzip压缩，临时文件可能占用过大的情况

\* useSharedStringsTable - 是否使用共享字符串表

\*/SXSSFWorkbook(XSSFWorkbook workbook, int rowAccessWindowSize, boolean compressTmpFiles, boolean useSharedStringsTable)

与XSSF的对比，在一个时间点上，只可以访问一定数量的Row；不再支持Sheet.clone()；不再支持公式的求值。但是除了滑动窗口，其余的EXCLE操作仍然使用的是XSSF的API。  
另外官方提示导出EXCEL后应该调用wb.dispose()来删除之前保存的临时文件。

//The example below writes a sheet with a window of 100 rows. When the row count reaches 101, the row with rownum=0 is flushed to disk and removed from memory, when rownum reaches 102 then the row with rownum=1 is flushed, etc.

package cn.nubia;

import org.junit.Assert;import org.apache.poi.ss.usermodel.Cell;import org.apache.poi.ss.usermodel.Row;import org.apache.poi.ss.usermodel.Sheet;import org.apache.poi.ss.util.CellReference;import org.apache.poi.xssf.streaming.SXSSFWorkbook;

import java.io.FileOutputStream;import java.io.IOException;

public class App {

public static void main( String[] args ) throws IOException {

SXSSFWorkbook wb = new SXSSFWorkbook(100); // keep 100 rows in memory, exceeding rows will be flushed to disk

Sheet sh = wb.createSheet();

for(int rownum = 0; rownum < 1000; rownum++){

Row row = sh.createRow(rownum);

for(int cellnum = 0; cellnum < 10; cellnum++){

Cell cell = row.createCell(cellnum);

String address = new CellReference(cell).formatAsString();

cell.setCellValue(address);

}

}

// Rows with rownum < 900 are flushed and not accessible

for(int rownum = 0; rownum < 900; rownum++){

Assert.assertNull(sh.getRow(rownum));

}

// ther last 100 rows are still in memory

for(int rownum = 900; rownum < 1000; rownum++){

Assert.assertNotNull(sh.getRow(rownum));

}

FileOutputStream out = new FileOutputStream("C:\\Users\\wange\\Desktop\\wtf.xlsx");

wb.write(out);

out.close();

// dispose of temporary files backing this workbook on disk

wb.dispose();

}}

wb.write(out)通过源码了解到过程是  
1、将wb的所有sheet调用flushRows()移出内存，写入临时.xml文件中  
2、生成了一个临时.xlsx文件将wb的一些模板数据写入这个临时文件  
3、将这个临时.xlsx文件转成ZipFile，遍历所有ZipEntry来获取Sheet，如果没有Sheet则直接复制流。  
4、如果能够获取到Sheet的则是那些临时.xml文件，在对这些文件进行解析并追踪写入导出文件中。  
（这边可能是涉及到了一些EXCEL文件格式的原理，就不深入研究了）

SXSSFWorkbook wb = new SXSSFWorkbook(-1)  
初始化设置为-1的时候我们可以自己定义写临时文件规则，比如每读1000行记录flush到临时一次，可以大大减少磁盘IO次数。

# 7、SXSSFWorkbook提供了一种低内存占用的EXCEL导出方法，但是没有提供读取文件流的方法。因此读入大数据量的时候还是只能使用XSSFWorkbook来读取。

使用SAX模型来解析EXCEL不像DOM模型一下把所有文件内容加载进内存，它逐行扫描文档，一边扫描，一边解析。所以那些只需要单遍读取内容的应用程序就可以从SAX解析中受益，这对大型文档的解析是个巨大优势。

OPCPackage pkg = OPCPackage.open(filename);

XSSFReader xssfReader = new XSSFReader(pkg);