POI 报表

第一章　POI 简介

--Jakata Poi HSSF：纯 java 的 Excel 解决方案

在我们实际的开发中，表现层的解决方案虽然有多样，但是 IE 浏览器已成为最多人使用的浏览器，因为大家都用 Windows。在企业办公系统中，常常有客户这样子要求：你要把我们的报表直接用 Excel 打开(电信系统、银行系统)。或者是：我们已经习惯用 Excel 打印。

Apache 的 Jakata 项目的 POI 子项目，目前比较成熟的是 HSSF 接口，处理 MSExcel 对象。它不象我们仅仅是用 csv 生成的没有格式的可以由 Excel 转换的东西，而是真正的 Excel 对象，你可以控制一些属性如 sheet,cell 等等。

首先，理解一下一个Excel的文件的组织形式，一个Excel文件对应于一个workbook(HSSFWorkbook)，一个workbook可以有多个sheet（HSSFSheet）组成，一个sheet是由多个row（HSSFRow）组成，一个row是由多个cell（HSSFCell）组成。

POI 可以到 www.apache.org 下载到。实际运行时，需要有 poi 包就可以了。HSSF 提供给用户使用的对象在 rg.apache.poi.hssf.usermodel 包中,主要部分包括 Excel 对象，样式和格式，还有辅助操作。有以下几种对象：

HSSFWorkbook	excel 的文档对象
HSSFSheet	excel 的表单
HSSFRow	excel 的行
HSSFCell	excel 的格子单元
HSSFFont	excel 字体
HSSFDataFormat	日期格式
在 poi1.7 中才有以下 2 项：	
HSSFHeader	sheet 头
HSSFFooter	sheet 尾（只有打印的时候才能看到效果）
和这个样式	
HSSFCellStyle	cell 样式
辅助操作包括	
HSSFDateUtil	日期
HSSFPrintSetup	打印
HSSFErrorConstants	错误信息表

以下可能需要使用到如下的类

```
import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFCellStyle;
import org.apache.poi.hssf.usermodel.HSSFDataFormat;
import org.apache.poi.hssf.usermodel.HSSFFont;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
```

```
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.hssf.util.HSSFColor;
```

先看 poi 的 examples 包中提供的最简单的例子，建立一个空 xls 文件。

```
import java.io.FileOutputStream;
import java.io.IOException;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
public class ExcelSample1 {

    public static void main(String[] args) throws IOException {
        //创建一个 excel 文件
        HSSFWorkbook wb= new HSSFWorkbook();
        FileOutputStream fileOut= new FileOutputStream("c:\\workbook.xls");
        // FileOutputStream fileOut= new FileOutputStream("c:/workbook.xls");
        wb.write(fileOut);
        fileOut.close();
    }
}
```

通过这个例子，我们在 c 盘下建立的是一个空白的 xls 文件（不是空文件）。在此基础上，我们可以进一步看其它的例子。

```
import org.apache.poi.hssf.usermodel.*;
import java.io.FileOutputStream;
import java.io.IOException;
public class CreateCells
{
    public static void main(String[] args) throws IOException
    {
      HSSFWorkbook wb = new HSSFWorkbook();       //建立新 HSSFWorkbook 对象
      HSSFSheet sheet = wb.createSheet("new sheet");   //建立新的 sheet 对象
      HSSFRow row = sheet.createRow((short)0);
      //在 sheet 里创建一行，参数为行号（第一行，此处可想象成数组）
      HSSFCell cell = row.createCell((short)0);
      //在 row 里建立新 cell（单元格），参数为列号（第一列）
      cell.setCellvalue(1);                              //设置 cell 的整数类型的值
      row.createCell((short)1).setCellvalue(1.2);        //设置 cell 浮点类型的值
      row.createCell((short)2).setCellvalue("test");     //设置 cell 字符类型的值
      row.createCell((short)3).setCellvalue(true);       //设置 cell 布尔类型的值
      HSSFCellStyle cellStyle = wb.createCellStyle(); //建立新的 cell 样式
      cellStyle.setDataFormat(HSSFDataFormat. getBuiltinFormat("m/d/yy h:mm"));
      //设置 cell 样式为定制的日期格式
      HSSFCell dCell =row.createCell((short)4);
      dCell.setCellvalue(new Date());                    //设置 cell 为日期类型的值
      dCell.setCellStyle(cellStyle);                     //设置该 cell 日期的显示格式
      HSSFCell csCell =row.createCell((short)5);
```

```
        csCell.setEncoding(HSSFCell.ENCODING_UTF_16);
        //设置 cell 编码解决中文高位字节截断
        csCell.setCellvalue("中文测试_Chinese Words Test");  //设置中西文结合字符串
        row.createCell((short)6).setCellType(HSSFCell.CELL_TYPE_ERROR);
        //建立错误 cell
        FileOutputStream fileOut = new FileOutputStream("workbook.xls");
        wb.write(fileOut);
        fileOut.close();
        }
}
```

通过这个例子，我们可以清楚的看到 xls 文件从大到小包括了 HSSFWorkbook HSSFSheet HSSFRow HSSFCell 这样几个对象。我们可以在 cell 中设置各种类型的值。

尤其要注意的是如果你想正确的显示非欧美的字符时，尤其象中日韩这样的语言，必须设置编码为 16 位的即是 HSSFCell.ENCODING_UTF_16，才能保证字符的高 8 位不被截断而引起编码失真形成乱码。

其他测试可以通过参考 examples 包中的测试例子掌握 poi 的详细用法，包括字体的设置，cell 大小和低纹的设置等。需要注意的是 POI 是一个仍然在完善中的公开代码的项目，所以有些功能正在不断的扩充。

感觉上面的操作比较的繁琐，然后就自己写了一个方法。这个方法不需要事先创建 row 和 cell，直接进行 cteateCell 就可以了，在程序中会自动进行判断，如果不存在的话会创建。

```
    private static void  cteateCell(HSSFWorkbook wb,HSSFRow row,short col,short align,String val){
        HSSFCell cell = row.createCell(col);
        cell.setEncoding(HSSFCell.ENCODING_UTF_16);
        cell.setCellValue(val);
        HSSFCellStyle cellstyle = wb.createCellStyle();
        cellstyle.setAlignment(align);
        cell.setCellStyle(cellstyle);
    }
```

对里面的几个参数的说明：

short col　　应该是你的 cell 单元格的位置也就是列号；

short align　　应该是你的对齐方式；

String val　　应该是你单元格里面要添加的值；

具体的调用如下：

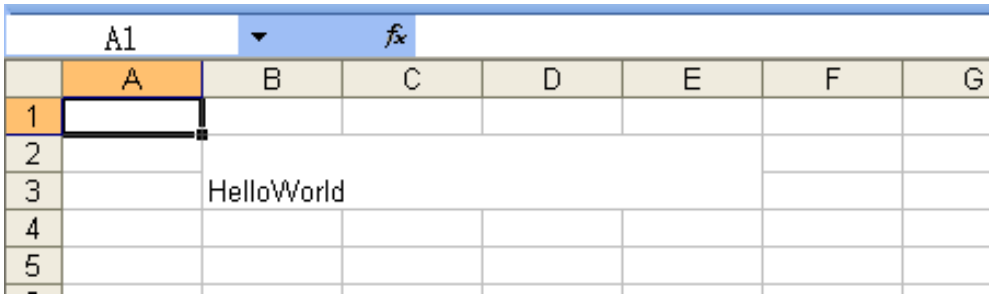
```
HSSFRow row = sheet.createRow((short)1);
cteateCell(wb,row,(short)0,HSSFCellStyle.ALIGN_CENTER_SELECTION,"SampleID");
```

在上边的例子里我们看到了要设置一个单元格里面信息的格式（例如，要将信息居中）设置的操作如下：

```
    HSSFCellStyle cellstyle = wb.createCellStyle();
    cellstyle.setAlignment(HSSFCellStyle.ALIGN_CENTER_SELECTION);
     cell.setCellStyle(cellstyle);
```

还有我们我们经常会用到的合并单元格，在这里我们也有这样的操作，代码如下：

```
sheet.addMergedRegion(new Region(1,(short)1,2,(short)4));
```



这里面我们还要介绍一个经常会遇到的问题，就是怎么来冻结一个窗口。poi 也为我们集成了这样的事情了。代码如下：

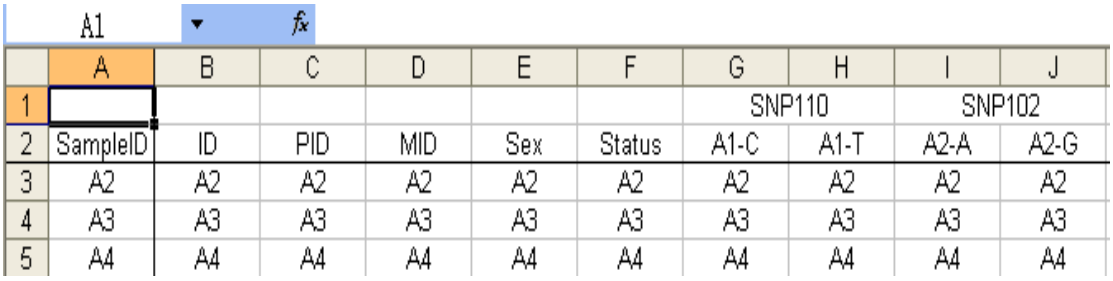
```
sheet.createFreezePane(1,2);
```

λ　在这里我们需要注意的是

一、该方法是在一个具体的 sheet 里面来进行操作。

二、方法 createFreezepane;有 2 个参数。前一个参数代表列；后一个参数代表行。

上边的代码对应的 excel 文件如下：



我么在画面上看到了明显的两条黑线，这就是冻结的窗口。

然后我们来看一个完整的 STRUTS 的小例子，在这个例子里面我们要做的事情是要模拟移动公司的网上营业厅里面的一个功能，我们要把一个客户当月的通话记录和各种信息查询出来，并且生成一张 excel 报表。首先，我们来看一下网上效果的截图。

13104459518 详单查询结果 (200604)

- 通话时长单位：秒
- 费用单位：元

被叫
主叫

生成Excel文件 共 380 条记录， 19 页/ 20 条 第 1 页 上一页 下一页

	主/被叫	详单年月	本方所在地	对方号码	对方归属地	通话日期	通话时间	通话时长	通话费	长途费	客服费	合计
380	主叫	200604	吉林.长春	130[redacted]	吉林.长春	20060417	17:53:38	142	0.00	0.00	0.00	0.00
379	主叫	200604	吉林.长春	1300[redacted]		20060417	17:21:45	1	0.10	0.00	0.00	0.10
378	主叫	200604	吉林.长春	130[redacted]		20060417	17:18:33	1	0.10	0.00	0.00	0.10
377	主叫	200604	吉林.长春	130[redacted]	吉林.长春	20060417	16:34:53	60	0.00	0.00	0.00	0.00
376	主叫	200604	吉林.长春	13[redacted]		20060417	12:37:35	1	0.10	0.00	0.00	0.10
375	主叫	200604	吉林.长春	130[redacted]		20060417	12:33:14	1	0.10	0.00	0.00	0.10
374	主叫	200604	吉林.长春	1300[redacted]		20060417	12:09:19	1	0.10	0.00	0.00	0.10
373	主叫	200604	吉林.长春	131[redacted]	吉林.长春	20060417	08:44:10	20	0.00	0.00	0.00	0.00

然后就是我们具体的代码实现了。

struts-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN" "http://struts.apache.org/dtds/struts-config_1_2.dtd">

<struts-config>
    <action-mappings>
    <action
        path="/search"
        type="action.SearchAction">
            <forward name="success" path="/detial.jsp"/>
    </action>
    <action
        path="/down"
        type="action.DownAction">
            <forward name="display" path="/down.jsp" />
    </action>
    </action-mappings>
    <message-resources parameter="ApplicationResources" />
</struts-config>
```

index.jsp

```
<%@ page contentType="text/html; charset=gb2312" language="java"%>
```

```
<html>
  <head>
    <title>欢迎进入 POI-Excel 文件报表系统</title>
  </head>
  <body>
<table align="center">
    <tr>
        <td align="center">欢迎进入 POI-Excel 文件报表系统</td>
    </tr>
        <tr>
        <td align="center">
          <a href="<%=request.getContextPath()%>/search.do">进入查询页面</a>
        </td>
    </tr>
</table>
</body>
</html>
```

连接数据库的 SQLBean,这个 bean 和我们之前在分页里面用到的 bean 是一样的。

```
package bean;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
public class SQLBean
{
    String url="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=poi_mvc";
    Connection con=null;
    Statement sta=null;
    public SQLBean()
    {
        try
        {
            Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
            con=DriverManager.getConnection(url,"sa","");
            sta=con.createStatement();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
    public ResultSet select(String selects) throws Exception
        {
            return sta.executeQuery(selects);
```

```
        }
}
```

为了体现 MVC 模式，我们在这里面引入了一个存储数据的 Bean。EntityBean.java

```
package bean;

public class EntityBean {
    private String sjhm;    private String hjlx;    private String dfhm;
    private String qssj;    private String thsc;    private String thdd;
    private String ctlx;    private String jbhf;    private String chf;
    private String zhf;     private String zjls;

    public EntityBean(String sjhm,String hjlx,String dfhm,String qssj,
                      String thsc,String thdd,String ctlx,String jbhf,
                      String chf,String zhf){
        this.sjhm=sjhm;         this.hjlx=hjlx;     this.dfhm=dfhm;
        this.qssj=qssj;     this.thsc=thsc;     this.thdd=thdd;
        this.ctlx=ctlx;     this.jbhf=jbhf;     this.chf=chf;
        this.zhf=zhf;   }
    public String getChf() {
        return chf;
    }
    public void setChf(String chf) {
        this.chf = chf;
    }
    public String getCtlx() {
        return ctlx;
    }
    public void setCtlx(String ctlx) {
        this.ctlx = ctlx;
    }
    public String getDfhm() {
        return dfhm;
    }
    public void setDfhm(String dfhm) {
        this.dfhm = dfhm;
    }
    public String getHjlx() {
        return hjlx;
    }
    public void setHjlx(String hjlx) {
        this.hjlx = hjlx;
    }
    public String getJbhf() {
        return jbhf;
```

```
    }
    public void setJbhf(String jbhf) {
        this.jbhf = jbhf;
    }
    public String getQssj() {
        return qssj;
    }
    public void setQssj(String qssj) {
        this.qssj = qssj;
    }
    public String getSjhm() {
        return sjhm;
    }
    public void setSjhm(String sjhm) {
        this.sjhm = sjhm;
    }
    public String getThdd() {
        return thdd;
    }
    public void setThdd(String thdd) {
        this.thdd = thdd;
    }
    public String getThsc() {
        return thsc;
    }
    public void setThsc(String thsc) {
        this.thsc = thsc;
    }
    public String getZhf() {
        return zhf;
    }
    public void setZhf(String zhf) {
        this.zhf = zhf;
    }
    public String getZjls() {
        return zjls;
    }
    public void setZjls(String zjls) {
        this.zjls = zjls;
    }
}
```

然后让我们来看看 SearchAction,他的主要功能把数据库里的数据提取出来，然后封装到 EntityBean 里面，在转发到一个新的页面。

```
package action;
```

```
import java.sql.ResultSet;
import java.util.ArrayList;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import bean.EntityBean;
import bean.SQLBean;

public class SearchAction extends Action {
    public Log log = LogFactory.getLog("Poi_mvc");
    public ActionForward execute(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response) {
        ArrayList list = new ArrayList();
        SQLBean sq = new SQLBean();
        String sql = "select * from detial";
        try{
            ResultSet res = sq.select(sql);
            while(res.next()){
                String sjhm = res.getString("sjhm");
                String hjlx = res.getString("hjlx");
                String dfhm = res.getString("dfhm");
                String qssj = res.getString("qssj");
                String thsc = res.getString("thsc");
                String thdd = res.getString("thdd");
                String ctlx = res.getString("ctlx");
                String jbhf = res.getString("jbhf");
                String chf = res.getString("chf");
                String zhf = res.getString("zhf");
                log.info("开始封装数据：手机号码="+sjhm+"呼叫类型"
                        +hjlx+"对方号码"+dfhm+"起始时间"+qssj+"通话时间"
                    +thsc+"通话地点"+thdd+"长途类型"+ctlx+"基本话费"
                    +jbhf+"常话费"+chf+"总话费"+zhf+"总纪录"+i);
        EntityBean eb = new EntityBean(sjhm,hjlx,dfhm,qssj,thsc,thdd,ctlx,jbhf,chf,zhf);
                list.add(eb);
                }
```

```
                res.close();
            }
            catch(Exception e){
                e.printStackTrace();
            }
            request.setAttribute("result",list);
            return mapping.findForward("success");
        }
}
```

显示页面 detail.jsp

```
<%@ page contenttype="text/html; charset=gb2312" language="java"%>

<%@ taglib uri="http://jakarta.apache.org/struts/tags-bean"      prefix="bean"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html"      prefix="html"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-logic"     prefix="logic"%>
<html>
<head>
<title>显示查询结果</title>
</head>
<body>
<div align="center">
<table border="1" align="center">
    <tr>
        <td colspan="10" align="center"><html:link page="/down.do">点击生成 excel 文件
</html:link></td>
    </tr>

    <tr>
        <td colspan="10" align="center"> 总 纪 录 数 ： <bean:size id="zj"
name="result"/><bean:write name="zj" /></td>
    </tr>
    <tr>
        <td>
        <p align="center">手机号码</p>
        </td>
        <td>
        <p align="center">呼叫类型</p>
        </td>
        <td>
        <p align="center">对方号码</p>
        </td>
        <td>
        <p align="center">起始时间</p>
        </td>
```

```
        <td>
        <p align="center">通话时间</p>
        </td>
        <td>
        <p align="center">通话地点</p>
        </td>
        <td>
        <p align="center">长途类型</p>
        </td>
        <td>
        <p align="center">基本话费</p>
        </td>
        <td>
        <p align="center">长话费</p>
        </td>
        <td>
        <p align="center">总话费</p>
        </td>
    </tr>

    <logic:iterate id="res" name="result">
        <tr>
            <td>
            <div align="center"><bean:write name="res" property="sjhm" /></div>
            </td>
            <td>
            <div align="center"><bean:write name="res" property="hjlx" /></div>
            </td>
            <td>
            <div align="center"><bean:write name="res" property="dfhm" /></div>
            </td>
            <td>
            <div align="center"><bean:write name="res" property="qssj" /></div>
            </td>
            <td>
            <div align="center"><bean:write name="res" property="thsc" /></div>
            </td>
            <td>
            <div align="center"><bean:write name="res" property="thdd" /></div>
            </td>
            <td>
            <div align="center"><bean:write name="res" property="ctlx" /></div>
            </td>
            <td>
```

```
                    <div align="center"><bean:write name="res" property="jbhf" /></div>
                    </td>
                    <td>
                    <div align="center"><bean:write name="res" property="chf" /></div>
                    </td>
                    <td>
                    <div align="center"><bean:write name="res" property="zhf" /></div>
                    </td>
                </tr>
        </logic:iterate>
        <tr>
            <td colspan="10">总纪录数：<bean:write name="zj" /></td>
        </tr>
</table>
</div>
</body>
</html>
```

模拟效果图：

点击生成excel文件									
总纪录数：18									
手机号码	呼叫类型	对方号码	起始时间	通话时间	通话地点	长途类型	基本话费	长话费	总话费
13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
总纪录数：18									

当我们点击生成 excel 文件这个超链接的时候，会找到这个 DownAction

```
package action;

import java.io.OutputStream;
import java.sql.ResultSet;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import bean.ExcelBean;
import bean.SQLBean;

public class DownAction extends Action {
	public ActionForward execute(ActionMapping mapping, ActionForm form,
			HttpServletRequest request, HttpServletResponse response)
			throws Exception {
		SQLBean sq = new SQLBean();
		String sql = "select * from detial";
		try {
			String fname = "detial";// Excel 文件名
			OutputStream os = response.getOutputStream();// 取得输出流
			response.reset();// 清空输出流
			response.setHeader("Content-disposition", "attachment; filename="
					+ fname + ".xls");
		// 设定输出文件头,该方法有两个参数，分别表示应答头的名字和值。

			response.setContentType("application/msexcel");
		// 定义输出类型

			ResultSet res = sq.select(sql);
			ExcelBean eb = new ExcelBean();
			eb.createFixationSheet(res, os);// 调用生成 excel 文件 bean
			res.close();
		} catch (Exception e) {
			System.out.println(e);
		}

		return mapping.findForward("display");
	}

}
```

在这个 action 里面我们调用了 ExcelBean

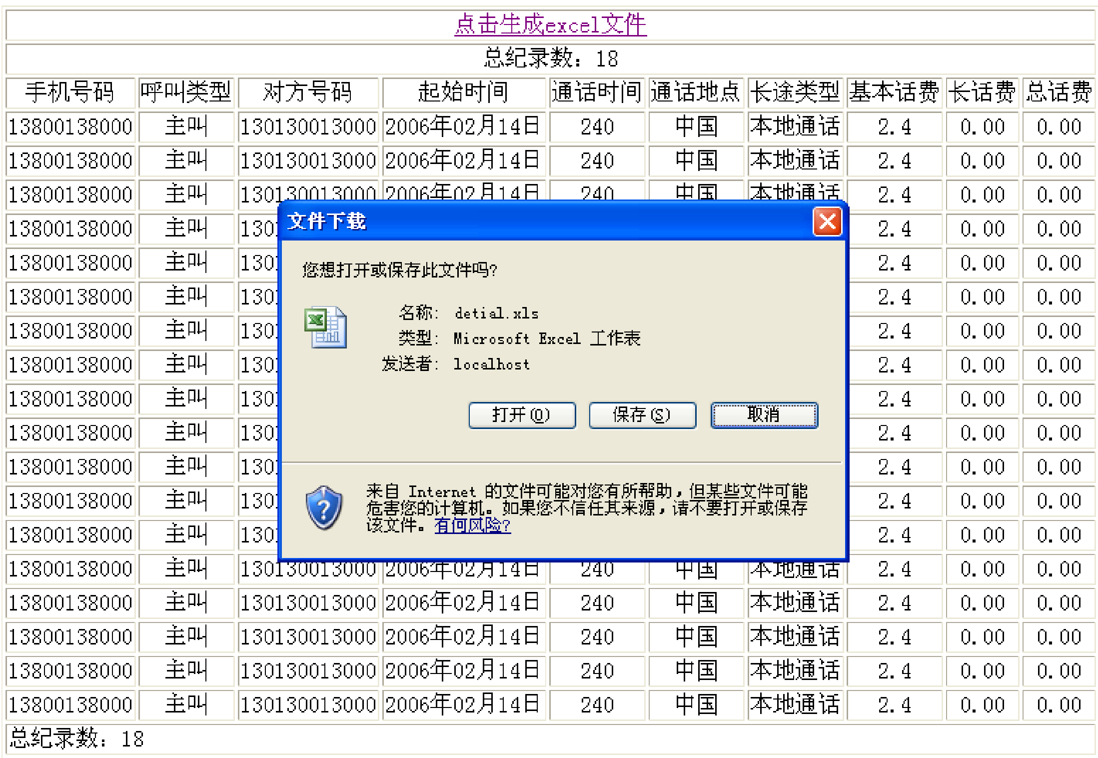
```
package bean;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.sql.ResultSet;
```

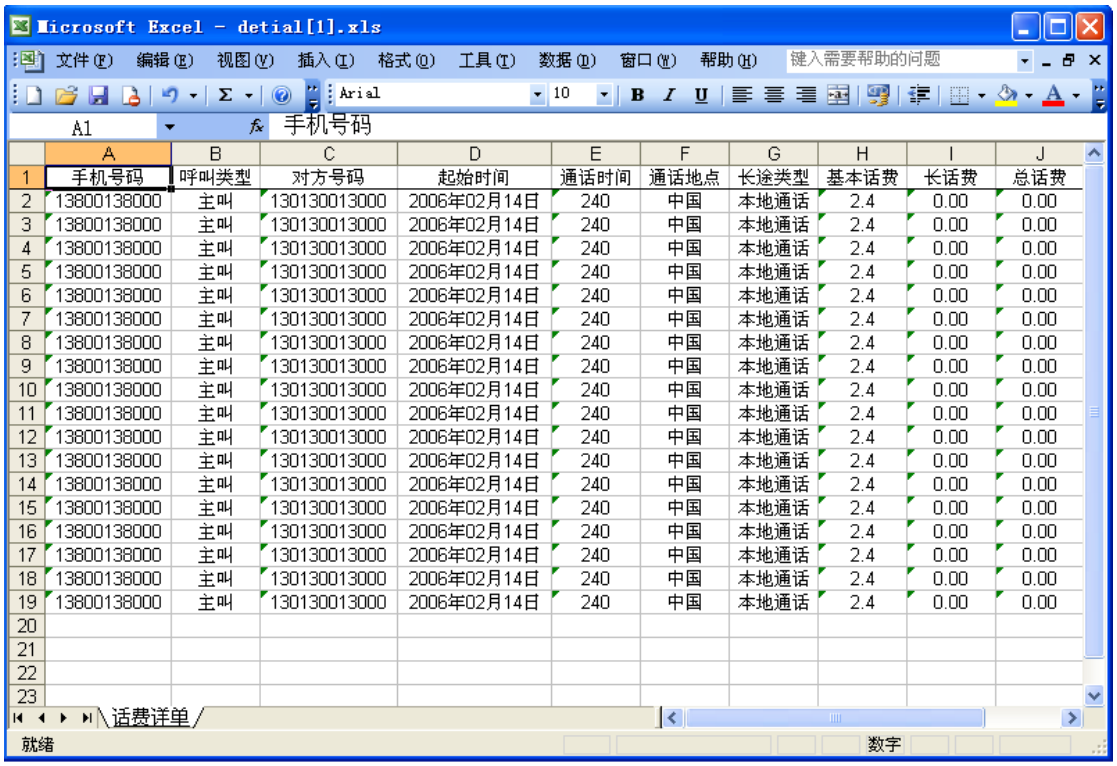
```
import java.sql.SQLException;
import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFCellStyle;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.hssf.util.Region;
public class ExcelBean {
	private HSSFWorkbook wb=null;
	public ExcelBean()
	{
		wb=new HSSFWorkbook();
	}
	public void createFixationSheet(ResultSet res,OutputStream os) throws IOException
	{
		HSSFSheet sheet=wb.createSheet("new sheet");
		wb.setSheetName(0,"话费详单",HSSFWorkbook.ENCODING_UTF_16);
		HSSFRow row=sheet.createRow((short)0);
		sheet.createFreezePane(0,1);
		cteateCell(wb,row,(short)0,"手机号码");
		cteateCell(wb,row,(short)1,"呼叫类型");
		cteateCell(wb,row,(short)2,"对方号码");
		cteateCell(wb,row,(short)3,"起始时间");
		cteateCell(wb,row,(short)4,"通话时间");
		cteateCell(wb,row,(short)5,"通话地点");
		cteateCell(wb,row,(short)6,"长途类型");
		cteateCell(wb,row,(short)7,"基本话费");
		cteateCell(wb,row,(short)8,"长话费");
		cteateCell(wb,row,(short)9,"总话费");
		int ii=0;
		try
		{
			int i=0;
			ii=res.getMetaData().getColumnCount();
			while(res.next())
			{
				i++;
				HSSFRow row2=sheet.createRow((short)i);
				for(int j=0;j<ii;j++)
				{
					String ss="";
					if(res.getString(j+1)==null)
						ss="空    null";
					else
```

```
                    ss=res.getString(j+1);
                    cteateCell(wb,row2,(short)j,ss);
                }
            }
        } catch(SQLException e)
        {
            e.printStackTrace();
        }
        wb.write(os);
      os.flush();
      os.close();
    }
    private void cteateCell(HSSFWorkbook wb,HSSFRow row,short col,String val)
    {
        HSSFCell cell=row.createCell(col);
        cell.setEncoding(HSSFCell.ENCODING_UTF_16);
        cell.setCellValue(val);
        HSSFCellStyle cellstyle=wb.createCellStyle();
        cellstyle.setAlignment(HSSFCellStyle.ALIGN_CENTER_SELECTION);
        cell.setCellStyle(cellstyle);
    }
}
```

我们看一下点击了，生成 excel 文件 的效果图



看看我们生成的 excel 文件



	A	B	C	D	E	F	G	H	I	J
1	手机号码	呼叫类型	对方号码	起始时间	通话时间	通话地点	长途类型	基本话费	长话费	总话费
2	13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
3	13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
4	13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
5	13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
6	13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
7	13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
8	13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
9	13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
10	13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
11	13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
12	13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
13	13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
14	13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
15	13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
16	13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
17	13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
18	13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
19	13800138000	主叫	130130013000	2006年02月14日	240	中国	本地通话	2.4	0.00	0.00
20										
21										
22										
23										

数据库表结构如下

```
create database poi_mvc
use poi_mvc
create table detial(
sjhm varchar(20),    --手机号码
hjlx varchar(20),    --呼叫类型
dfhm varchar(20),    --对方号码
qssj varchar(20),    --起始时间
thsc varchar(20),    --通话时长
thdd varchar(20),    --通话地点
ctlx varchar(20),    --长途类型
jbhf varchar(20),    --基本话费
chf varchar(20),     --长话费
zhf varchar(20),     --总话费
)
```

Apache :POI-HSSF

- 字体大小:
- 小
- 中
- 大

Jakarta_POI 使用Java读写Excel(97-2002)文件，可以满足大部分的需要。
因为刚好有一个项目使用到了这个工具，花了点时间顺便翻译了一下POI本身带的一个Guide.有一些节减和修改，希望给使用这个项目的人一些入门帮助。
POI 下面有几个自项目:HSSF用来实现Excel 的读写.以下是HSSF的主页
http://jakarta.apache.org/poi/hssf/index.html

下面的介绍是基于以下地址的翻译：
http://jakarta.apache.org/poi/hssf/quick-guide.html

目前的版本为 1.51 应该是很长时间之内的一个稳定版，但HSSF提供的Sample不是基于 1.51 所写，所以使用的时候需要适当的注意.
其实POI下面的几个子项目侧重不同读写 Word 的HDF正在开发当中.

XML下的FOP(http://xml.apache.org/fop/index.html)
可以输出pdf文件，也是比较好的一个工具

目录:
创建一个workbook
创建一个sheet
创建cells
创建日期cells
设定单元格格式

说明：
以下可能需要使用到如下的类

```
import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFCellStyle;
import org.apache.poi.hssf.usermodel.HSSFDataFormat;
import org.apache.poi.hssf.usermodel.HSSFFont;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.hssf.util.HSSFColor;
```

创建workbook

```
HSSFWorkbook wb = new HSSFWorkbook();
//使用默认的构造方法创建workbook
FileOutputStream fileOut = new FileOutputStream("workbook.xls");
//指定文件名
wb.write(fileOut);
//输出到文件
fileOut.close();
```

创建一个sheet

```
HSSFWorkbook wb = new HSSFWorkbook();
HSSFSheet sheet1 = wb.createSheet("new sheet");
//workbook创建sheet
HSSFSheet sheet2 = wb.createSheet("second sheet");
//workbook创建另外的sheet
FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

创建cells

```
HSSFWorkbook wb = new HSSFWorkbook();
```

```
HSSFSheet sheet = wb.createSheet("new sheet");
//注意以下的代码很多方法的参数是short 而不是int 所以需要做一次类型转换
HSSFRow row = sheet.createRow((short)0);
//sheet 创建一行
HSSFCell cell = row.createCell((short)0);
//行创建一个单元格
cell.setCellValue(1);
//设定单元格的值
//值的类型参数有多中double ,String ,boolean,
row.createCell((short)1).setCellValue(1.2);
row.createCell((short)2).setCellValue("This is a string");
row.createCell((short)3).setCellValue(true);

// Write the output to a file
FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

创建日期cells

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

HSSFRow row = sheet.createRow((short)0);

HSSFCell cell = row.createCell((short)0);
//设定值为日期
cell.setCellValue(new Date());

HSSFCellStyle cellStyle = wb.createCellStyle();
//指定日期显示格式
cellStyle.setDataFormat(HSSFDataFormat.getFormat("m/d/yy h:mm"));
cell = row.createCell((short)1);
cell.setCellValue(new Date());
//设定单元格日期显示格式
cell.setCellStyle(cellStyle);

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

设定单元格格式

单元格格式的设定有很多形式包括单元格的对齐方式，内容的字体设置，单元格的背景色等，因为形式比较多，只举一些例子.以下的例子在POI1.5 中可能会有所改变具体查看API.

```
.....
// Aqua background
HSSFCellStyle style = wb.createCellStyle();
//创建一个样式
style.setFillBackgroundColor(HSSFCellStyle.AQUA);
//设定此样式的的背景颜色填充
style.setFillPattern(HSSFCellStyle.BIG_SPOTS);

//样式的填充类型。
//有多种式样如:
//HSSFCellStyle.BIG_SPOTS
//HSSFCellStyle.FINE_DOTS
//HSSFCellStyle.SPARSE_DOTS等
style.setAlignment(HSSFCellStyle.ALIGN_CENTER );
//居中对齐
style.setFillBackgroundColor(HSSFColor.GREEN.index);
//设定单元个背景颜色
style.setFillForegroundColor(HSSFColor.RED.index);
//设置单元格显示颜色
HSSFCell cell = row.createCell((short) 1);
cell.setCellValue("X");
cell.setCellStyle(style);
```

使用 poi 的 hssf 生成一个 excel 文件以后
有一个主类 Workbook(相当于一个 excel 文件)的方法
Workbook.write(OutputStream)可以写到 response.getOutputStream()里面
如果事先设置 response 的 contentType 为 excel 和下载的附件名称就可下载 excel

```
        HSSFWorkbook book = _proxy.expertExcel(_formBean,_login);

        if(book!=null)
        {
                response.setContentType ( "application/ms-excel" ) ;
                response.setHeader ( "Content-Disposition" ,
                                                "attachment;filename
="+new String("导出 Excel.xls".getBytes(),"iso-8859-1")) ;
                book.write(response.getOutputStream());
        }
```

其中 expertExcel 无非是从数据库或者其他地方获取数据创建 excel 即可.

POI 读取 EXCEL 教程

一、Excel基础

二、HSSF概况

三、通过usermodel读取文件

四、通过usermodel写入文件

五、通过eventusermodel读取文件

六、HSSF电子表格结构

七、通过HPSF读取文档属性

八、文档摘要信息

九、附录

正文：

在上一篇文章中，我们介绍了 POI项目的基本概念，了解了如何用 POI来读写OLE 2 复合文档结构，并给出了两个简单的例子：用 POI来读写Excel文件的Workbook流。本文继续前文的话题，阐述如何用 POI来读取/写入完整的Excel文件。

约定：POI项目 2.0 版现在已经接近正式发行阶段，开发进度迅速，不断有新的功能集成到原有的系统，同时也有对原有系统的修改。为了保证本文的及时性，本文将按照最近的 1.9 开发版说明。虽然编译最近的发行版源代码也能正常运行，但现在的代码和 2.0 的发行版会有一些出入。

一、Excel基础

Microsoft Excel 97 文件格式也被称为BIFF8，最近版本的Excel只对该格式作了少量的改动。增加对新格式的支持除了增加项目的复杂性之外，唯一的效果也许只是不得不使每个用户升级代码，没有什么实际的好处。因此，在下文说明中，凡是提到Excel 97 格式的地方其实都是指Excel从 97 到XP的格式。

二、HSSF概况

POI项目实现的Excel 97 文件格式称为HSSF——也许你已经猜到，HSSF是Horrible SpreadSheet Format的缩写，也即“讨厌的电子表格格式”（微软使某些原本简单的事情过分复杂，同时又过分简单地处理了某些原本需要灵活性的事情，让人不胜佩服！）也许HSSF的名字有点滑稽，就本质而言它是一个非常严肃、正规的API。通过HSSF，你可以用纯Java代码来读取、写入、修改Excel文件。

前面一篇文章提到了 POIFS，那么HSSF和 POIFS又有什么关系呢？就象其他 POI的API一样，HSSF建立在 POIFS的基础上，因此在HSSF内的有些代码和前文的某些代码很相似。不过，当我们编写基于HSSF API的代码时，一般不需要了解 POIFS API的细节。

HSSF为读取操作提供了两类API：usermodel和eventusermodel，即“用户模型”和“事件-用户模型”。前者很好理解，后者比较抽象，但操作效率要高得多。usermodel主要有org.apache.poi.hssf.usermodel和 org.apache.poi.hssf.eventusermodel 包实现（在HSSF的早期版本中，org.apache.poi.hssf.eventusermodel属于eventmodel包）。

usermodel包把Excel文件映射成我们熟悉的结构，诸如Workbook、Sheet、Row、Cell等，它把整个结构以一组对象的形式保存在内存之中。eventusermodel要求用户熟悉文件格式的底层结构，它的操作风格类似于XML的SAX API和AWT的事件模型（这就是eventusermodel名称的起源），要掌握窍门才能用好。另外，eventusermodel的API只提供读取文件的功能，也就是说不能用这个API来修改文件。

三、通过usermodel读取文件

用HSSF的usermodel读取文件很简单。首先创建一个InputStream，然后创建一个HSSFWorkbook：

```
InputStream myxls = new FileInputStream("workbook.xls"));
HSSFWorkbook wb      = new HSSFWorkbook(myxls);
```

有了HSSFWorkbook实例，接下来就可以提取工作表、工作表的行和列，例如：

```
HSSFSheet sheet = wb.getSheetAt(0);       // 第一个工作表
HSSFRow row     = sheet.getRow(2);        // 第三行
HSSFCell cell   = row.getCell((short)3);  // 第四个单元格
```

上面这段代码提取出第一个工作表第三行第四单元格。利用单元格对象可以获得它的值，提取单元格的值时请注意它的类型：

```
if (cell.getCellType() == HSSFCell.CELL_TYPE_STRING) {
   ("单元格是字符串，值是： " + cell.getStringCellValue());
} else if (cell.getCellType() == HSSFCell.CELL_TYPE_NUMERIC) {
   ("单元格是数字，值是： " + cell.getCellValue());
} else () {
   ("单元格的值不是字符串或数值。");
}
```

如果搞错了数据类型，程序将遇到异常。特别地，用HSSF处理日期数据要小心。Excel内部以数值的形式保存日期数据，区别日期数据的唯一办法是通过单元格的格式（如果你曾经在Excel中设置过日期格式，应该明白这是什么意思）。

因此，对于包含日期数据的单元格，cell.getCellType()将返回HSSFCell.CELL_TYPE_NUMERIC，不过利用工具函数HSSFDateUtil.isCellDateFormatted(cell)可以判断出单元格的值是否为日期。isCellDateFormatted函数通过比较单元格的日期和Excel的内置日期格式得出结论——可以想象，按照这种判断方法，很多时候isCellDateFormatted函数会返回否定的结论，存在一定的误判可能。

本文附录包含了一个在Servlet环境中利用HSSF创建和返回Excel工作簿的实例。

四、通过usermodel写入文件

写入XLS文件比读取XLS文件还要简单。创建一个HSSFWorkbook实例，然后在适当的时候创建一个把文件写入磁盘的OutputStream，但延迟到处理结束时创建OutputStream也可以：

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

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

创建工作表及其内容必须从相应的父对象出发，例如：

```
HSSFSheet sheet = wb.createSheet();
HSSFRow row     = sheet.createRow((short)0);
HSSFCell cell   = row.createCell((short)0);
cell.setCellValue(1);
row.createCell((short)1).setCellValue(1.2);
row.createCell((short)2).setCellValue("一个字符串");
row.createCell((short)3).setCellValue(true);
```

如果要设置单元格的样式，首先要创建一个样式对象，然后把它指定给一个单元格——或者把它指定给多个具有相同样式的单元格，例如，如果Excel表格中有一个摘要行，摘要行的数据必须是粗体、斜体，你可以创建一个summaryRowStyle样式对象，然后把这个样式指定给所有摘要行上的单元格。

注意，CellFormat和CellStyle对象是工作簿对象的成员，单元格对象只是引用它们。

```
...
HSSFCellStyle style = workbook.createCellStyle();
style.setDataFormat
    (HSSFDataFormat.getBuiltinFormat("($#,##0_);[Red]($#,##0)"));
style.setFillBackgroundColor(HSSFColor.AQUA.index);
style.setFillPattern(HSSFCellStyle.BIG_SPOTS);
...
someCell.setCellStyle(style);
someOtherCell.setCellStyle(style);
```

版本较新的HSSF允许使用数量有限的Excel公式。这一功能目前还是“Beta级质量”，正式使用之前务必仔细测试。指定公式的方式类如：someCell.setCellFormula(SUM(A1:A2:);。

当前，公式中已经可以调用所有内建的函数或操作符，但逻辑操作符和函数（例如IF函数）除外，这部分功能目前还在开发之中。

五、通过eventusermodel读取文件

通过eventusermodel读取文件要比使用usermodel复杂得多，但效率也要高不少，因为它要求应用程序一边读取数据，一边处理数据。eventusermodel实际上模拟了DOM环境下SAX处理XML文档的办法，应

用程序首先要注册期望处理的数据，eventusermodel将在遇到匹配的数据结构时回调应用程序注册的方法。使用eventusermodel最大的困难在于你必须熟悉Excel工作簿的内部结构。

在HSSF中，低层次的二进制结构称为记录（Record）。记录有不同的类型，每一种类型由org.apache.poi.hssf.record包中的一个Java类描述。例如，BOFRecord记录表示Workbook或Sheet区域的开始，RowRecord表示有一个行存在并保存其样式信息。所有具有CellValueRecordInterface接口的记录表示Excel的单元格，包括NumericRecord、LabelSSTRecord和FormulaRecord（还有其他一些，其中部分已被弃置不用，部分用于优化处理，但一般而言，HSSF可以转换它们）。

下面是一个注册事件处理句柄的例子：

```
private EventRecordFactory factory = new EventRecordFactory();
factory.registerListener(new ERFListener() {
    public boolean processRecord(Record rec) {
        (got BOF Record);
        return true;
    }
}, new short[] {BOFRecord.sid});
factory.processRecords(someInputStream);
```

六、HSSF电子表格结构

如前所述，HSSF建立在 POIFS的基础上。具体地说，Excel 97+文件是OLE 2 复合文档（ OLE 2 Compound Document），底层的OLE 2 复合文档保存了一个总是命名为Workbook（Excel 95 除外，HSSF不支持Excel 95）的流。然而，宏和图片并不保存在Workbook流，它们有自己独立的流，有时甚至会放到OLE 2 CDF文件之内的另一个目录。理想情况下，宏也应该被保留，不过目前 POI项目中还没有合适的API来处理宏。

每一个流之内是一组记录，一个记录其实就是一个字节数组，可分为记录头、记录体两部分。记录头指明了记录的类型（也即ID）以及后继数据的长度，记录体被分割成多个字段（Field），字段包含数值数据（包括对其他记录的引用）、字符数据或标记。

下图概要说明了Excel工作簿的顶级结构：

```
Bla.xls {
    OLE2CDF headers
    "Workbook" stream {
        Workbook {
            Static String Table Record..
            Sheet names… and pointers
        }
        Sheet {
```

```
            ROW
            ROW
            …
            NUMBER RECORD (cell)
            LABELSST Record (cell)
            …
        }
        Sheet
    }
}
… images, macros, etc.
Document Summary
Summary
```

七、通过HPSF读取文档属性

在Microsoft Word、Excel、PowerPoint等软件中，用户可以通过“文件”→“属性”菜单给文档添加附加信息，包括文档的标题、主题、摘要、类别、关键词等，同时应用软件本身还会加入最后访问的用户、最后访问和修改/打印的日期时间等信息。

文档的属性和正文是分开保存的。如前所述，OLE 2 CDF文件内部就象是一个容器，里面包含许多类似目录和文件的结构，而 POIFS就是用来访问其中的文件的工具。这些文件也称为流，文档的属性就保存在 POIFS文件系统中专用的流里面。以一个Word文档为例：虽然在资源管理器中你只看到一个叫做MyFile.doc的文档，其实在这个文档的内部，又包含了一个WordDocument、一个SummaryInformation和一个DocumentSummaryInformation文档；通常还会有其他的文档，这里暂且不管。

你能够猜出这些文档（流）分别包含什么内容吗？不错，WordDocument包含了你在Word里面编辑的文本，文档的属性保存在SummaryInformation和DocumentSummaryInformation流里面。也许将所有属性保存在单个文档里面看起来太简单了，所以Microsoft决心要使用两个流，为了使事情更复杂一点，这两个流的名字前面还加上了八进制的\005 字符——这是一个不可打印的字符，因此前面就把它省略了。

Microsoft定义的标准属性有一个好处，它们并不在乎主文档到底是什么类型——不管是Word文档、Excel工作簿还是PowerPoint幻灯。只要你知道如何读取Excel文档的属性，就知道了如何读取其他文档的属性。

读取文档属性其实并不复杂，因为Java程序可以利用 POI项目的HPSF包。HPSF是 Horrible Property Set Format的缩写，译成中文就是“讨厌的属性集格式”。HPSF包是 POI项目实现的读取属性工具，目前还不支持属性写入。

对于读取Microsoft定义的标准属性，通过HPSF提供的API可以很方便地办到；但如果要读取任意属性集就要用到更一般化的API，可以想象它要比读取标准属性的API复杂不少。本文只介绍读取标准属性的简单API，因为对大多数应用程序来说这已经完全足够了。

下面就是一个读取OLE 2 CDF文档的标题（title）属性的Java程序：

```
import java.io.*;
import org.apache.poi.hpsf.*;
import org.apache.poi.poifs.eventfilesystem.*;

/**
* 读取OLE 2 文档标题的示例程序，
* 在命令行参数中指定文档的文件名字。
*/

public class ReadTitle
{
    public static void main(String[] args) throws IOException
    {
        final String filename = args[0];
        POIFSReader r          = new POIFSReader();
        r.registerListener(new MyPOIFSReaderListener(),
            "\005SummaryInformation");
        r.read(new FileInputStream(filename));
    }

    static class MyPOIFSReaderListener
            implements POIFSReaderListener
    {
        public void processPOIFSReaderEvent(POIFSReaderEvent event)
        {
            SummaryInformation si = null;
            try
            {
                si = (SummaryInformation)
                PropertySetFactory.create(event.getStream());
            }
            catch (Exception ex)
            {
                throw new RuntimeException
                    ("属性集流\"" + event.getPath() +
                        event.getName() + "\": " + ex);
            }

            final String title = si.getTitle();

            if (title != null)
```

```
            System.out.println("标题: \"" + title + "\"");
        else
            System.out.println("该文档没有标题.");
        }
    }
}
```

main()方法利用 POIFS的事件系统从命令行指定的OLE 2 文档读取名为\005SummaryInformation的流，当 POIFSReader 遇到这个流时，它把控制传递给 MyPOIFSReaderListener 的processPOIFSReaderEvent()方法。
processPOIFSReaderEvent() 到底有什么用呢？它通过参数获得一个输入流，该输入流包含了文档标题等属性。为了访问文档的属性，我们从输入流创建一个PropertySet实例，如下所示：

```
si = (SummaryInformation) PropertySetFactory.create(event.getStream());
```

这个语句其实包含三个步骤的操作：

◆ event.getStream()从 POIFSReader传入的 POIFSReaderEvent获得输入流。

◆ 以刚才获得的输入流为参数，调用PropertySetFactory的静态方法create()。正如其名字所暗示的，PropertySetFactory是一个工厂类，它有一台“机器”能够把一个输入流转换成一个PropertySet实例，这台机器就是create()方法。

◆ 把create()方法返回的PropertySet定型（cast）成为SummaryInformation。PropertySet提供了按照一般办法读取属性集的各种机制，SummaryInformation是PropertySet的子类，即SummaryInformation类在PropertySet类的基础上增加了操作Microsoft标准属性的便捷方法。

在这个处理过程中，可能引起错误的因素很多，因此我们把这部分内容放入了一个try块——不过这个示例程序只按照最简单的方式处理了异常，在实际应用中，最好能够对可能出现的不同异常类型分别处理。除了一般的I/O异常之外，还有可能遇到HPSF特有的异常，例如，如果输入流不包含属性集或属性集非法，就会抛出NoPropertySetStreamException异常。

有一种错误不太常见，但也不是绝无可能——\005SummaryInformation包含一个合法的属性集，但不是摘要信息属性集。如果出现这种情况，则定型成SummaryInformation操作会失败，引发ClassCastException异常。

获得SummaryInformation实例之后，剩下的事情就很简单了，只要调用getTitle()方法，然后输出结果。

除了getTitle()之外，SummaryInformation还包含其他一些便捷方法，例如getApplicationName()、getAuthor()、getCharCount()、和getCreateDateTime()等。HPSF的JavaDoc文档详细说明了所有这些方

法。

八、文档摘要信息

遗憾的是，并非所有的属性都保存在摘要信息属性集之中。许多（但不是全部）OLE 2 文件还有另一个属性集，称为“文档摘要信息”，对应的流是\005DocumentSummaryInformation。这个属性集保存的属性包括文档的类别、PowerPoint幻灯的多媒体剪辑数量，等等。

要访问文档摘要信息属性集，程序的处理过程也和上例相似，只是注册的目标应该改成\005DocumentSummaryInformation流——有时，你可能想要同时注册到摘要信息和文档摘要信息这两个流。其余的处理方式和前面的例子差不多，你应该把包含文档摘要信息的流传递给PropertySetFactory.create()，但这次工厂方法将返回一个DocumentSummaryInformation对象（而不是前面例子中的SummaryInformation对象）。如果同时注册到了两个流，注意检查返回值的具体类型，或者使用Java的instanceof操作符，或者使用专用的isSummaryInformation()和isDocumentSummaryInformation()方法。记住，create()方法返回的总是一个PropertySet对象，因此你总是可以对create()返回对象调用isSummaryInformation()和isDocumentSummaryInformation()方法，PropertySet类之所以要提供这两个方法，是因为属性集可能是自定义的。

如果你想要处理自定义的属性集，或者要从标准的属性集读取用户定义的属性，必须使用一个更一般化的API，前面已经提到，这个API要复杂得多，本文不再讨论，请参见HPSF的HOW-TO文档和 POI的文档。

结束语：本文探讨了HSSF的应用以及如何输出到Excel文件，另外还涉及了HPSF以及如何读取属性集文档摘要信息。POI是一个功能非常强大的项目，许多主题本文尚未涉及，例如如何用HSSF Serializer将XML文档转换成Excel格式等，这一切就有待你自己去研究了。

参考：

Jakarta POI项目主页

Jakarta POI 源代码

九、附录

实例：利用Servlet创建和返回一个工作簿。

```
package org.apache.poi.hssf.usermodel.examples;

import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.poi.hssf.usermodel.*;
```

```
public class HSSFCreate extends HttpServlet {
    public void init(ServletConfig config)
        throws ServletException {
        super.init(config);
    }

    public void destroy() {
    }

    /** 处理HTTP GET 和POST请求
     * @param request：请求
     * @param response：应答
     */
    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("application/vnd.ms-excel");
        HSSFWorkbook wb = new HSSFWorkbook();
        HSSFSheet sheet = wb.createSheet("new sheet");

        // 创建一个新的行，添加几个单元格。
        // 行号从 0 开始计算
        HSSFRow row     = sheet.createRow((short)0);
        // 创建一个单元格，设置单元格的值
        HSSFCell cell    = row.createCell((short)0);
        cell.setCellValue(1);

        row.createCell((short)1).setCellValue(1.2);
        row.createCell((short)2).setCellValue("一个字符串值");
        row.createCell((short)3).setCellValue(true);
        // 写入输出结果
        OutputStream out = response.getOutputStream();
        wb.write(out);
        out.close();
    }

    /** 处理HTTP GET请求
     * @param request：请求
     * @param response：应答
     */
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
            throws ServletException, IOException {
```

```
        processRequest(request, response);
    }

    /** 处理HTTP POST请求
     * @param request：请求
     * @param response：应答
     */
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /** 返回关于Servlet的简单说明
     */
    public String getServletInfo() {
      return "示例：在Servlet中用HSSF创建Excel工作簿";
    }
}
```

POI HSSF 操作MS Excel简述 (OMIS二期设计阶段寻找Excel导入导出实现方法）
POI HSSF 操作MS Excel简述

POI HSSF是一个专门操作EXCEL的java包，可通过纯java操作xls文件。
POI HSSF的类文件都放在在org.apache.poi.hssf包下，通过此包中的类就可实现用java操作Excel文件了。
下面是用 POI HSSF操作Excel文件的方法简述：

一， 建立Excel工作薄

```
HSSFWorkbook wb = new HSSFWorkbook();
```

二， 建立Excel工作表，每个工作表对应的是Excel界面左下角的一个标签sheet1，sheet2 …

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

三， 在工作表中建立单元格

```
//首先，建立行对像，行号作为参数传给createRow方法，第一行由 0 开始计算。
HSSFRow row = sheet.createRow((short)0);

//建单元格
HSSFCell cell = row.createCell((short)0);
```

```
//给单元格赋值
cell.setCellValue(1);

//也可同一行内完成建立单元格和赋值
row.createCell((short)1).setCellValue(1.2);
row.createCell((short)2).setCellValue("This is a string");
row.createCell((short)3).setCellValue(true);

//数据格式可通过创建单元格值时默认如上面所视
//也可以创建单元格后调用setCellType指定
cell.setCellType(CELL_TYPE_NUMERIC);
```

四， 向单元格插入日期值

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

// 可通过Sheet.setSheetName(sheetindex,"SheetName",encoding)设定工作表名

// 创建新行并向其加入单元格，行号由 0 开始。
HSSFRow row = sheet.createRow((short)0);

// 创建一个单元格并向其输入一日期值，但这第一个单元格并非是日期格式。
HSSFCell cell = row.createCell((short)0);
cell.setCellValue(new Date());

// 我们将这第二个单元格改成日期格式，这需要从工作薄创建一个新的单元格格式，这可// 以只影响当前建立的一个单元格。
HSSFCellStyle cellStyle = wb.createCellStyle();
cellStyle.setDataFormat(HSSFDataFormat.getBuiltinFormat("m/d/yy h:mm"));
cell = row.createCell((short)1);
cell.setCellValue(new Date());
cell.setCellStyle(cellStyle);
```

五， 各种单元格样式

```
HSSFCellStyle cellStyle = wb.createCellStyle();
//对齐
cellStyle.setAlignment(HSSFCellStyle.ALIGN_CENTER);

//带边框
cellStyle.setBorderBottom(HSSFCellStyle.BORDER_THIN);

//颜色与填充样式
cellStyle.setFillBackgroundColor(HSSFColor.AQUA.index);
```

```
cellStyle.setFillPattern(HSSFCellStyle.BIG_SPOTS);
cellStyle.setFillForegroundColor(HSSFColor.ORANGE.index);
cellStyle.setFillPattern(HSSFCellStyle.SOLID_FOREGROUND);
```

六， 行高,列宽。

```
HSSFWorkbook wb = new HSSFWorkbook();
HSSFSheet sheet = wb.createSheet("new sheet");
HSSFRow row = sheet.createRow((short)0);

//2 是行高值
row.setRowHeight(2);

//3 是列号，4 是列宽值
sheet.setColumnWidth(3, 4);
```

七，例程

首先调用一个方法将Oracle数据库中的数据取出，放到List实例中，这里我调用了srrd项目中ProductData类的listProductQuery()取得一个List实例。List中的对象是一系列名为ProductQuery实体类的实例。然后读List，将ProductQuery实例中的数据取出放到HSSFCell单元格中。最后将HSSFWorkbook中的数据输出到输出流，完成数据导出。

```
//建工作薄
HSSFWorkbook wb = new HSSFWorkbook();
//建名为example的工作表
HSSFSheet sheet = wb.createSheet("example");
//给工作表前 8 列定义列宽
sheet.setColumnWidth((short)0,(short)2500);
sheet.setColumnWidth((short)1,(short)6000);
sheet.setColumnWidth((short)2,(short)3500);
sheet.setColumnWidth((short)3,(short)9000);
sheet.setColumnWidth((short)4,(short)8000);
sheet.setColumnWidth((short)5,(short)8000);
sheet.setColumnWidth((short)6,(short)20000);
sheet.setColumnWidth((short)7,(short)8000);
//在表中建行
HSSFRow row = sheet.createRow(0);
//建立单元格
HSSFCell cell[] = new HSSFCell[8];
for (short i = 0; i < 8; i++) {
cell = row.createCell(i);
//将单元格定义成UTF_16 编码，这样才能使输出数据不会乱码
cell.setEncoding(HSSFCell.ENCODING_UTF_16);
```

```
}
//写单元格标题
cell[0].setCellValue("登记ID");
cell[1].setCellValue("登记号");
cell[2].setCellValue("所在地市ID");
cell[3].setCellValue("产品中文名");
cell[4].setCellValue("产品英文名");
cell[5].setCellValue("产品服务对象");
cell[6].setCellValue("产品功能描述");
cell[7].setCellValue("产品类别");
//查询数据库，取得数据列表的List实例
List list = new ArrayList();
ProductDataManager mgr = new ProductDataManager();
try {
list = mgr.listProductQuery("","", "", "", "", "1999-2-1", "2004-2-1");
} catch (SrrdException e) {
e.printStackTrace();
}
//从List中取出数据放入工作表中
if (list != null && list.size() > 0) {
for (int i = 0; i < list.size() - 1; i++) {
ProductQuery query = (ProductQuery) list.get(i);
HSSFRow datarow = sheet.createRow(i + 1);
HSSFCell data[] = new HSSFCell[8];
for (short j = 0; j < 8; j++) {
data[j] = datarow.createCell(j);
//将单元格定义成UTF_16 编码，这样才能使输出数据不会乱码
data[j].setEncoding(HSSFCell.ENCODING_UTF_16);
}
data[0].setCellValue(query.getCertId());
data[1].setCellValue(query.getCertNum());
data[2].setCellValue(query.getCityCode());
data[3].setCellValue(query.getSoftWareCname());
data[4].setCellValue(query.getSoftWareEname());
data[5].setCellValue(query.getSoftwareFor());
data[6].setCellValue(query.getSoftwareFuncDesc());
data[7].setCellValue(query.getSoftwareType());
}
}
//将工作薄输出到输出流
ServletOutputStream sos=response.getOutputStream();
wb.write(sos);
sos.close();
```

```
//也可输出成xls文件
File file = new File("workbook.xls");
try {
FileOutputStream fileOut = new FileOutputStream(file);
wb.write(fileOut);
fileOut.close();
} catch (IOException e) {
e.printStackTrace();
}
```