

POI 实战 (v1.0)

--- author: Vintage Yu
---QQ: 1093782707
---e-mail: yuzhenling@gmail.com
---2011-08-03

目录

1. POI 入门	4
1.1 Excel 基本知识	4
1.2 POI 基本类	5
1.3 POI 简单读取 Excel 数据	5
1.4 POI 简单写出 Excel	9
2. 复杂读取	16
2.1 单元格各类型数据读取	16
2.1.1 基本类型	16
2.1.2 日期类型	18
2.2 自定义类型	21
3. 复杂写入	22
3.1 复杂写入	22
3.2 多层公式	27
4. 常用操作	30
4.1 注释	30
4.1.1 单表注释	30
4.1.2 多表注释	32
4.1.3 空单元格注释	34
4.2 单元格合并与数据读取	34
4.3 窗口冻结	37
4.4 下拉列表	38
5. POI 样式	39
5.1 POI 样式相关类	39
5.2 单元格边框样式	40
5.3 单元格背景色	41
5.4 单元格字体格式	41
5.5 单元格对齐方式	42
5.6 单元格数字格式化	45
5.7 单元格宽度与高度	46
5.8 合并单元格样式	47
5.9 Excel 样式实例	48
6. 总结	52
7. 附录	53

前言

由于最近开发涉及到比较多的 Excel 文件的处理，所以也有机会接触到了 POI，同时也接触过些 JXL，在本人使用喜爱方面还是偏向 POI 的，所以写的这个文档。

此篇 POI 讲解相对基础，都是平时我们在开发中用到的功能。主要包括 Excel 的读取、写入，各种数据格式处理、单元格合并、注释、下拉列表及单元格的边框、背景色、宽高度调整等。其中也有开发中经常会出现的小问题注意。以此供自己与大家参考。Ps：小女水平有限，如有错误欢迎纠正，如有功能补充欢迎来电谢谢。

1.POI 入门

1.1 Excel 基本知识

首先简单介绍一下 Excel 在开发中必备的基本知识。

如图 1 中，此 Excel 数据为 sheet1 表中 4 行 4 列数据。POI 读取 Excel 数据方式可以说是按行按单元格读取。

Excel 与 POI 数据对应关系为：

表关系

Excel	POI
Sheet1 表	0 表
Sheet2 表	1 表
Sheet3 表	2 表
依次类推...	

行关系

Excel	POI
1 行	0 行
2 行	1 行
3 行	2 行
4 行	3 行
依次类推...	

列关系

Excel	POI
A 列	0 列
B 列	1 列
C 列	2 列
D 列	3 列
依次类推...	

	A	B	C	D	E
1	姓名	学号	籍贯	学院	
2	张三	0001	天津	信息学院	
3	李四	0002	浙江	物流学院	
4	王五	0003	台湾	化学院	
5					
6					
7					

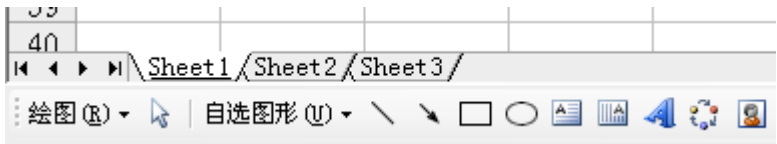


图 1

1.2 POI 基本类

POI 在读取 Excel 时,需要先初始化整个 Excel 然后再去获得 Sheet,根据 Sheet 获得 Row, 再根据 Row 获得 Cell 这样一个顺序。

Excel 与 POI 类对应关系

Excel	POI	用途
整个 Excel	org.apache.poi.hssf.usermodel.HSSFWorkbook	创建或装载整个 Excel 文件
Sheet	org.apache.poi.hssf.usermodel.HSSFSheet	创建或装载 Excel 中的某个 sheet
行	org.apache.poi.hssf.usermodel.HSSFRow	创建或装载 Excel 中某行
单元格	org.apache.poi.hssf.usermodel.HSSFCell	创建或装载 Excel 中某个单元格

1.3 POI 简单读取 Excel 数据

代码部分:

```
package com.vintage.testpoi;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;

import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;

/**
 * POI入门：简单读取excel数据
 * @author VintageYu
 */
```

```
*/
public class ReadExcelTest {

    public static void read(InputStream inputStream) throws IOException{
        //初始整个Excel
        HSSFWorkbook workbook = new HSSFWorkbook(inputStream);
        //获取第一个Sheet表
        HSSFSheet sheet = workbook.getSheetAt(0); //或者 HSSFSheet sheet =
workbook.getSheet("Sheet1");
        //获取第一行
        HSSFRow row0 = sheet.getRow(0);
        //获取第一行的第一个单元格
        HSSFCell cell = row0.getCell(0);
        //打印
        System.out.println(cell.getRichStringCellValue().getString());

    }

    public static void main(String[] args) {
        InputStream inputStream = null;
        try {
            //读取文件流
            inputStream = new FileInputStream(new File("E:\\read1.xls"));
            read(inputStream);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally{
            try {
                if(inputStream != null){
                    inputStream.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

输入结果见图 2



图 2

为了进一步贴近实际，将上面部分代码进行如下修改

```
//获取第一个Sheet表
HSSFSSheet sheet = workbook.getSheetAt(0); //或者 HSSFSSheet sheet =
workbook.getSheet("Sheet1");
//获取第一行
HSSFRow row0 = sheet.getRow(0);
//获取第一行的第一个单元格
HSSFCell cell = row0.getCell(0);
//打印
System.out.println(cell.getRichStringCellValue().getString());
```

将这部分代码改为

```
//循环workbook中所有sheet
for(int sheetIndex = 0; sheetIndex <
workbook.getNumberOfSheets(); sheetIndex++){
    HSSFSSheet sheet = workbook.getSheetAt(sheetIndex);
    System.out.println("sheet序号:"+sheetIndex+", sheet名称:
"+workbook.getSheetName(sheetIndex));
    //循环该sheet中的有数据的每一行
    for(int rowIndex = 0; rowIndex <= sheet.getLastRowNum();
rowIndex++){
        HSSFRow row = sheet.getRow(rowIndex);
        if(row == null){
            continue;
        }
        //循环该行的每一个单元格
        for(int cellnum = 0; cellnum < row.getLastCellNum();
cellnum++){
            HSSFCell cell = row.getCell(cellnum);
            System.out.println("第"+rowIndex+"行    第
"+cellnum+"列    内容为:
"+cell.getRichStringCellValue().getString());
        }
    }

    System.out.println("-----
-----");
}
```

打印结果如图 3



图 3

注:

workbook.getNumberOfSheets() = 3 --- 测试用 Excel(图 1)有 3 个 sheet,得到的就是 **sheet 的个数**

sheet.getLastRowNum() = 3 --- 测试用 Excel (图 1) 中 Sheet1 表有 4 行数据,所以得到的数据是 POI 中行序列的 **3**

row.getLastCellNum() = 4 --- 测试用 Excel (图 1) 中 Sheet1 表有 4 列数据,所以得到的数据是 POI 中列数

1.4 POI 简单写出 Excel

代码部分:

```
package com.vintage.testpoi;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;

import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFRichTextString;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
```

```
/**
 * POI入门：简单写出excel数据
 * @author VintageYu
 *
 */
public class WriteExcelTest {

    public static void write(OutputStream outputStream) throws
IOException{
        //初始一个workbook
        HSSFWorkbook workbook = new HSSFWorkbook();
        //创建一个表
        HSSFSheet sheet = workbook.createSheet("firstSheet");
        //创建行
        HSSFRow row = sheet.createRow(0);
        //创建单元格
        HSSFCell cell = row.createCell(0);
        cell.setCellValue(new HSSFRichTextString("hello POI"));
        workbook.write(outputStream);
    }

    public static void main(String[] args) {
        OutputStream outputStream = null;
        try {
            outputStream = new FileOutputStream(new
File("E:\\helloPOI.xls"));
            write(outputStream);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally{
            if(outputStream != null){
                try {
                    outputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

```
}
```

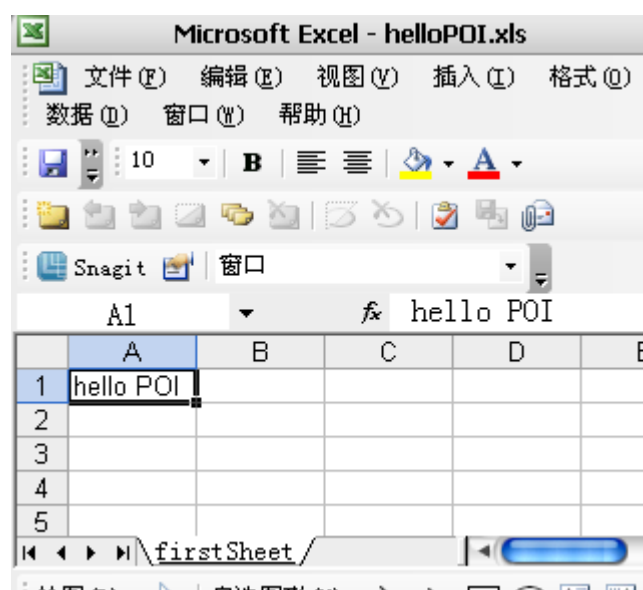


图 4

以下贴近实际进一步修改：

从数据库中读出相关信息并写入到 Excel 中

数据库表如图：

	ID	NAME	NO	NATIVE	EDU
1	1	张三	0001	天津	信息学院
2	2	李四	0002	浙江	物流学院
3	3	王五	0003	台湾	化学院

图 5

主要类：

```
package com.vintage.testpoi;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.util.List;

import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
```

```
/**
 * POI入门：简单写出excel数据
 * @author VintageYu
 *
 */
public class WriteExcelTest {

    public static void write(OutputStream outputStream) throws
IOException{
        //初始一个workbook
        HSSFWorkbook workbook = new HSSFWorkbook();
        List<Student> list = Conn.getData();
        //循环创建多个sheet
        for(int sheetIndex = 0; sheetIndex < 3; sheetIndex++){
            HSSFSheet sheet = workbook.createSheet("sheet"+sheetIndex);
            //创建多行
            for(int rowIndex = 0; rowIndex < list.size(); rowIndex++){
                HSSFRow row = sheet.createRow(rowIndex);
                Student student = list.get(rowIndex);
                //创建多列
                for(int cellnum = 0; cellnum < 4; cellnum++){
                    HSSFCell cell = row.createCell(cellnum);
                    switch (cellnum) {
                        case 0:
                            cell.setCellValue(student.getName());
                            break;
                        case 1:
                            cell.setCellValue(student.getNo());
                            break;
                        case 2:
                            cell.setCellValue(student.getNativePlace());
                            break;
                        case 3:
                            cell.setCellValue(student.getEdu());
                            break;
                    }
                }
            }
        }
        workbook.write(outputStream);
    }
}
```

```
public static void main(String[] args) {
    OutputStream outputStream = null;
    try {
        outputStream = new FileOutputStream(new
File("E:\\helloPOI.xls"));
        write(outputStream);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } finally{
        if(outputStream != null){
            try {
                outputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

依赖类:

```
public class Conn {

    public static Connection getConn(){
        Connection cn = null;
        try {
            Class.forName( "oracle.jdbc.driver.OracleDriver" );
            cn =
DriverManager.getConnection( "jdbc:oracle:thin:@****:1521:****", "****",
"****");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return cn;
    }

    public static List<Student> getData(){
        List<Student> list = new ArrayList<Student>();
```

```
Connection conn = getConn();
String sql = "SELECT * FROM tpoi_vintage t";
try {
    conn.setAutoCommit(false);
    PreparedStatement ps = conn.prepareStatement(sql);
    ResultSet rs = ps.executeQuery();
    while(rs.next()){
        Integer id = rs.getInt("id");
        String name = rs.getString("name");
        String no = rs.getString("no");
        String nativePlace = rs.getString("native");
        String edu = rs.getString("edu");
        Integer year = rs.getInt("YEAR");
        Integer math = rs.getInt("MATH");
        Integer chinese = rs.getInt("CHINESE");
        Integer english = rs.getInt("ENGLISH");
        Integer science = rs.getInt("SCIENCE");
        Integer isCity = rs.getInt("ISCITY");
        Date schoolDate = rs.getDate("SCHOOLDATE");
        Date birth = rs.getDate("BIRTH");

        Student student = new Student();
        student.setId(id);
        student.setName(name);
        student.setNo(no);
        student.setNativePlace(nativePlace);
        student.setEdu(edu);
        student.setYear(year);
        student.setMath(math);
        student.setChinese(chinese);
        student.setEnglish(english);
        student.setScience(science);
        if(isCity == 0){
            student.setCity(false);
        }else if(isCity == 1){
            student.setCity(true);
        }
        student.setSchoolDate(schoolDate);
        student.setBirth(birth);

        list.add(student);
    }
    conn.commit();
}
```

```
    } catch (SQLException e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    }  
  
    return list;  
}  
  
public class Student {  
    private Integer id;  
    private String name;  
    private String no;  
    private String nativePlace;  
    private String edu;  
    以下省略。。。  
}
```

写出 Excel 预览:

	A	B	C	D
1	张三	0001	天津	信息学院
2	李四	0002	浙江	物流学院
3	王五	0003	台湾	化学院
4				
5				

sheet0 sheet1 sheet2

	A	B	C	D
1	张三	0001	天津	信息学院
2	李四	0002	浙江	物流学院
3	王五	0003	台湾	化学院
4				
5				

sheet0 sheet1 sheet2

	A	B	C	D
1	张三	0001	天津	信息学院
2	李四	0002	浙江	物流学院
3	王五	0003	台湾	化学院
4				
5				

sheet1 sheet2

图 6

2. 复杂读取

2.1 单元格各类型数据读取

2.1.1 基本类型

在实际工作中，我们处理的 Excel 数据都不止限于字符型数据，更多的是数字、日期、甚至公式等。

下面是单元格类型说明：

类型	
CELL_TYPE_BLANK	空值（cell 不为空）
CELL_TYPE_BOOLEAN	布尔
CELL_TYPE_ERROR	错误
CELL_TYPE_FORMULA	公式
CELL_TYPE_STRING	字符串
CELL_TYPE_NUMERIC	数值

以上单元格的类型，可以通过 `getCellType()` 方法获得，返回值为 `int`。

下面读取一个多类型数据 Excel 文件：

图 7 中，数据文件格式包括字符、数字、公式、布尔。



	A	B	C	D	E	F	G	H	I	J	K
1	姓名	学号	籍贯	学院	入学年份	数学	语文	英语	理综	总成绩	是否城镇户口
2	张三	0001	天津	信息学院	2008	140	130	110	300	680	TRUE
3	李四	0002	浙江	物流学院	2008	141	131	111	301	684	FALSE
4	王五	0003	台湾	化学院	2008	142	132	112	302	688	FALSE
5											

图 7

代码片段：

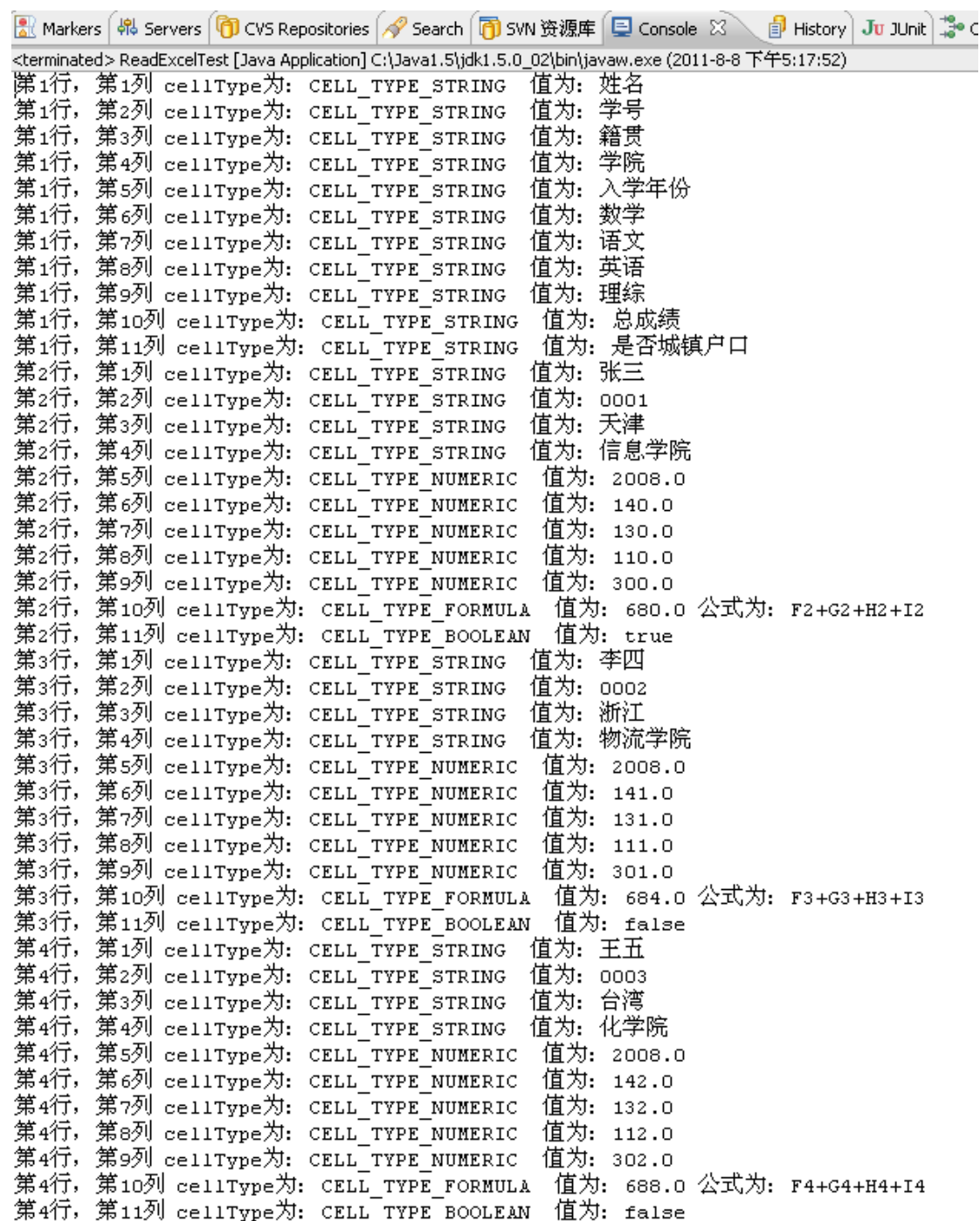
```
public static void read(InputStream inputStream) throws IOException{
    HSSFWorkbook workbook = new HSSFWorkbook(inputStream);
    //循环workbook中所有sheet
    for(int sheetIndex = 0; sheetIndex <
workbook.getNumberOfSheets(); sheetIndex++){
        HSSFSheet sheet = workbook.getSheetAt(sheetIndex);
        //循环该sheet中的有数据的每一行
        for(int rowIndex = 0; rowIndex <= sheet.getLastRowNum();
```



```
rowIndex++){
    HSSFRow row = sheet.getRow(rowIndex);
    if(row == null){
        continue;
    }
    //循环该行的每一个单元格
    for(int cellnum = 0; cellnum < row.getLastCellNum();
cellnum++){
        HSSFCell cell = row.getCell(cellnum);
        getCellValue(cell, rowIndex, cellnum);
    }
}

public static void getCellValue(HSSFCell cell, int rowIndex, int
cellnum){
    if(cell == null){
        return;
    }else if(cell.getCellType() == HSSFCell.CELL_TYPE_BLANK){
        System.out.println("第"+(rowIndex+1)+"行, 第"+(cellnum+1)+"
列 cellType为: CELL_TYPE_BLANK");
    }else if(cell.getCellType() == HSSFCell.CELL_TYPE_STRING){
        System.out.println("第"+(rowIndex+1)+"行, 第"+(cellnum+1)+"
列 cellType为: CELL_TYPE_STRING 值为:
"+cell.getRichStringCellValue().getString());
    }else if(cell.getCellType() == HSSFCell.CELL_TYPE_NUMERIC){
        System.out.println("第"+(rowIndex+1)+"行, 第"+(cellnum+1)+"
列 cellType为: CELL_TYPE_NUMERIC 值为: "+cell.getNumericCellValue());
    }else if(cell.getCellType() == HSSFCell.CELL_TYPE_BOOLEAN){
        System.out.println("第"+(rowIndex+1)+"行, 第"+(cellnum+1)+"
列 cellType为: CELL_TYPE_BOOLEAN 值为: "+cell.getBooleanCellValue());
    }else if(cell.getCellType() == HSSFCell.CELL_TYPE_FORMULA){
        System.out.println("第"+(rowIndex+1)+"行, 第"+(cellnum+1)+"
列 cellType为:CELL_TYPE_FORMULA 值为:"+cell.getNumericCellValue()+" 公
式为: "+cell.getCellFormula());
    }
    return;
}
```

打印预览:



```

<terminated> ReadExcelTest [Java Application] C:\Java1.5\jdk1.5.0_02\bin\javaw.exe (2011-8-8 下午5:17:52)
第1行, 第1列 cellType为: CELL_TYPE_STRING 值为: 姓名
第1行, 第2列 cellType为: CELL_TYPE_STRING 值为: 学号
第1行, 第3列 cellType为: CELL_TYPE_STRING 值为: 籍贯
第1行, 第4列 cellType为: CELL_TYPE_STRING 值为: 学院
第1行, 第5列 cellType为: CELL_TYPE_STRING 值为: 入学年份
第1行, 第6列 cellType为: CELL_TYPE_STRING 值为: 数学
第1行, 第7列 cellType为: CELL_TYPE_STRING 值为: 语文
第1行, 第8列 cellType为: CELL_TYPE_STRING 值为: 英语
第1行, 第9列 cellType为: CELL_TYPE_STRING 值为: 理综
第1行, 第10列 cellType为: CELL_TYPE_STRING 值为: 总成绩
第1行, 第11列 cellType为: CELL_TYPE_STRING 值为: 是否城镇户口
第2行, 第1列 cellType为: CELL_TYPE_STRING 值为: 张三
第2行, 第2列 cellType为: CELL_TYPE_STRING 值为: 0001
第2行, 第3列 cellType为: CELL_TYPE_STRING 值为: 天津
第2行, 第4列 cellType为: CELL_TYPE_STRING 值为: 信息学院
第2行, 第5列 cellType为: CELL_TYPE_NUMERIC 值为: 2008.0
第2行, 第6列 cellType为: CELL_TYPE_NUMERIC 值为: 140.0
第2行, 第7列 cellType为: CELL_TYPE_NUMERIC 值为: 130.0
第2行, 第8列 cellType为: CELL_TYPE_NUMERIC 值为: 110.0
第2行, 第9列 cellType为: CELL_TYPE_NUMERIC 值为: 300.0
第2行, 第10列 cellType为: CELL_TYPE_FORMULA 值为: 680.0 公式为: F2+G2+H2+I2
第2行, 第11列 cellType为: CELL_TYPE_BOOLEAN 值为: true
第3行, 第1列 cellType为: CELL_TYPE_STRING 值为: 李四
第3行, 第2列 cellType为: CELL_TYPE_STRING 值为: 0002
第3行, 第3列 cellType为: CELL_TYPE_STRING 值为: 浙江
第3行, 第4列 cellType为: CELL_TYPE_STRING 值为: 物流学院
第3行, 第5列 cellType为: CELL_TYPE_NUMERIC 值为: 2008.0
第3行, 第6列 cellType为: CELL_TYPE_NUMERIC 值为: 141.0
第3行, 第7列 cellType为: CELL_TYPE_NUMERIC 值为: 131.0
第3行, 第8列 cellType为: CELL_TYPE_NUMERIC 值为: 111.0
第3行, 第9列 cellType为: CELL_TYPE_NUMERIC 值为: 301.0
第3行, 第10列 cellType为: CELL_TYPE_FORMULA 值为: 684.0 公式为: F3+G3+H3+I3
第3行, 第11列 cellType为: CELL_TYPE_BOOLEAN 值为: false
第4行, 第1列 cellType为: CELL_TYPE_STRING 值为: 王五
第4行, 第2列 cellType为: CELL_TYPE_STRING 值为: 0003
第4行, 第3列 cellType为: CELL_TYPE_STRING 值为: 台湾
第4行, 第4列 cellType为: CELL_TYPE_STRING 值为: 化学院
第4行, 第5列 cellType为: CELL_TYPE_NUMERIC 值为: 2008.0
第4行, 第6列 cellType为: CELL_TYPE_NUMERIC 值为: 142.0
第4行, 第7列 cellType为: CELL_TYPE_NUMERIC 值为: 132.0
第4行, 第8列 cellType为: CELL_TYPE_NUMERIC 值为: 112.0
第4行, 第9列 cellType为: CELL_TYPE_NUMERIC 值为: 302.0
第4行, 第10列 cellType为: CELL_TYPE_FORMULA 值为: 688.0 公式为: F4+G4+H4+I4
第4行, 第11列 cellType为: CELL_TYPE_BOOLEAN 值为: false

```

图 8

注：公式格式的单元格值为数字，所以取值的时候是通过 `cell.getNumericCellValue()` 方法，而取单元格的公式需要通过 `cell.getCellFormula()` 方法。

2.1.2 日期类型

Excel 中的 Date 类型以 Double 型数字存储的，表示当前时间与 1900 年 1 月

1 日相隔的天数。所以在 Excel 中如果单元格格式为 NUMERIC 类型还需要进一步判断是否为日期类型。在读取日期单元格时需要调用 HSSFDateUtil 的 isCellDateFormatted 方法，来判断该 Cell 的数据格式是否是 Date 类型，然后通过 HSSFCell 的 getDateCellValue 方法获取 Date。

如图 9，在之前 Excel 数据后新加了 Date 格式的数据

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	姓名	学号	籍贯	学院	入学年份	数学	语文	英语	理综	总成绩	是否城镇户口	入学年份	出生日期
2	张三	0001	天津	信息学院	2008	140	130	110	300	680	TRUE	2008-9-1	1990-2-16
3	李四	0002	浙江	物流学院	2008	141	131	111	301	684	FALSE	2008-9-1	1990-8-5
4	王五	0003	台湾	化学院	2008	142	132	112	302	688	FALSE	2008-9-1	1990-12-20

图 9

代码片段：

```
public static void getCellValue(HSSFCell cell, int rowIndex, int
cellnum) {
    if (cell == null) {
        return;
    } else if (cell.getCellType() == HSSFCell.CELL_TYPE_BLANK) {
        System.out.println("第" + (rowIndex + 1) + "行, 第" + (cellnum + 1) + "
列 cellType为: CELL_TYPE_BLANK");
    } else if (cell.getCellType() == HSSFCell.CELL_TYPE_STRING) {
        System.out.println("第" + (rowIndex + 1) + "行, 第" + (cellnum + 1) + "
列 cellType为: CELL_TYPE_STRING 值为:
" + cell.getRichStringCellValue().getString());
    } else if (cell.getCellType() == HSSFCell.CELL_TYPE_NUMERIC) {
        if (HSSFDateUtil.isCellDateFormatted(cell)) {
            Date date = cell.getDateCellValue();
            SimpleDateFormat dateFormat = new
SimpleDateFormat("yyyy-MM-dd");
            System.out.println("第" + (rowIndex + 1) + "行, 第
" + (cellnum + 1) + "列 cellType为: Date 值为: " + dateFormat.format(date));
        } else {
            System.out.println("第" + (rowIndex + 1) + "行, 第" + (cellnum + 1) + "
列 cellType为: CELL_TYPE_NUMERIC 值为: " + cell.getNumericCellValue());
        }
    } else if (cell.getCellType() == HSSFCell.CELL_TYPE_BOOLEAN) {
        System.out.println("第" + (rowIndex + 1) + "行, 第" + (cellnum + 1) + "
列 cellType为: CELL_TYPE_BOOLEAN 值为: " + cell.getBooleanCellValue());
    } else if (cell.getCellType() == HSSFCell.CELL_TYPE_FORMULA) {
        System.out.println("第" + (rowIndex + 1) + "行, 第" + (cellnum + 1) + "
列 cellType为: CELL_TYPE_FORMULA 值为: " + cell.getNumericCellValue() + " 公
式为: " + cell.getCellFormula());
    }
    return;
}
```

打印预览:

```

<terminated> ReadExcelTest [Java Application] C:\Java1.5\jdk1.5.0_02\bin\javaw.exe (2011-8-8 下午5:57:26)
第1行, 第6列 cellType为: CELL_TYPE_STRING 值为: 数学
第1行, 第7列 cellType为: CELL_TYPE_STRING 值为: 语文
第1行, 第8列 cellType为: CELL_TYPE_STRING 值为: 英语
第1行, 第9列 cellType为: CELL_TYPE_STRING 值为: 理综
第1行, 第10列 cellType为: CELL_TYPE_STRING 值为: 总成绩
第1行, 第11列 cellType为: CELL_TYPE_STRING 值为: 是否城镇户口
第1行, 第12列 cellType为: CELL_TYPE_STRING 值为: 入学年份
第1行, 第13列 cellType为: CELL_TYPE_STRING 值为: 出生日期
第2行, 第1列 cellType为: CELL_TYPE_STRING 值为: 张三
第2行, 第2列 cellType为: CELL_TYPE_STRING 值为: 0001
第2行, 第3列 cellType为: CELL_TYPE_STRING 值为: 天津
第2行, 第4列 cellType为: CELL_TYPE_STRING 值为: 信息学院
第2行, 第5列 cellType为: CELL_TYPE_NUMERIC 值为: 2008.0
第2行, 第6列 cellType为: CELL_TYPE_NUMERIC 值为: 140.0
第2行, 第7列 cellType为: CELL_TYPE_NUMERIC 值为: 130.0
第2行, 第8列 cellType为: CELL_TYPE_NUMERIC 值为: 110.0
第2行, 第9列 cellType为: CELL_TYPE_NUMERIC 值为: 300.0
第2行, 第10列 cellType为: CELL_TYPE_FORMULA 值为: 680.0 公式为: F2+G2+H2+I2
第2行, 第11列 cellType为: CELL_TYPE_BOOLEAN 值为: true
第2行, 第12列 cellType为: Date 值为: 2008-09-01
第2行, 第13列 cellType为: Date 值为: 1990-02-16
第3行, 第1列 cellType为: CELL_TYPE_STRING 值为: 李四
第3行, 第2列 cellType为: CELL_TYPE_STRING 值为: 0002
第3行, 第3列 cellType为: CELL_TYPE_STRING 值为: 浙江
第3行, 第4列 cellType为: CELL_TYPE_STRING 值为: 物流学院
第3行, 第5列 cellType为: CELL_TYPE_NUMERIC 值为: 2008.0
第3行, 第6列 cellType为: CELL_TYPE_NUMERIC 值为: 141.0
第3行, 第7列 cellType为: CELL_TYPE_NUMERIC 值为: 131.0
第3行, 第8列 cellType为: CELL_TYPE_NUMERIC 值为: 111.0
第3行, 第9列 cellType为: CELL_TYPE_NUMERIC 值为: 301.0
第3行, 第10列 cellType为: CELL_TYPE_FORMULA 值为: 684.0 公式为: F3+G3+H3+I3
第3行, 第11列 cellType为: CELL_TYPE_BOOLEAN 值为: false
第3行, 第12列 cellType为: Date 值为: 2008-09-01
第3行, 第13列 cellType为: Date 值为: 1990-08-05
第4行, 第1列 cellType为: CELL_TYPE_STRING 值为: 王五
第4行, 第2列 cellType为: CELL_TYPE_STRING 值为: 0003
第4行, 第3列 cellType为: CELL_TYPE_STRING 值为: 台湾
第4行, 第4列 cellType为: CELL_TYPE_STRING 值为: 化学院
第4行, 第5列 cellType为: CELL_TYPE_NUMERIC 值为: 2008.0
第4行, 第6列 cellType为: CELL_TYPE_NUMERIC 值为: 142.0
第4行, 第7列 cellType为: CELL_TYPE_NUMERIC 值为: 132.0
第4行, 第8列 cellType为: CELL_TYPE_NUMERIC 值为: 112.0
第4行, 第9列 cellType为: CELL_TYPE_NUMERIC 值为: 302.0
第4行, 第10列 cellType为: CELL_TYPE_FORMULA 值为: 688.0 公式为: F4+G4+H4+I4
第4行, 第11列 cellType为: CELL_TYPE_BOOLEAN 值为: false
第4行, 第12列 cellType为: Date 值为: 2008-09-01
第4行, 第13列 cellType为: Date 值为: 1990-12-20

```

图 10

2.2 自定义类型

在 Excel 中有许多种数据格式，并且支持用户自定义格式。如图 11。可以通过 HSSFDataFormat 类进行操作。getBuiltinFormat(short index)方法根据编号返回内置数据类型，getBuiltinFormat(java.lang.String format)方法根据数据类型返回其编号，getBuiltinFormats()返回整个内置的数据格式列表。

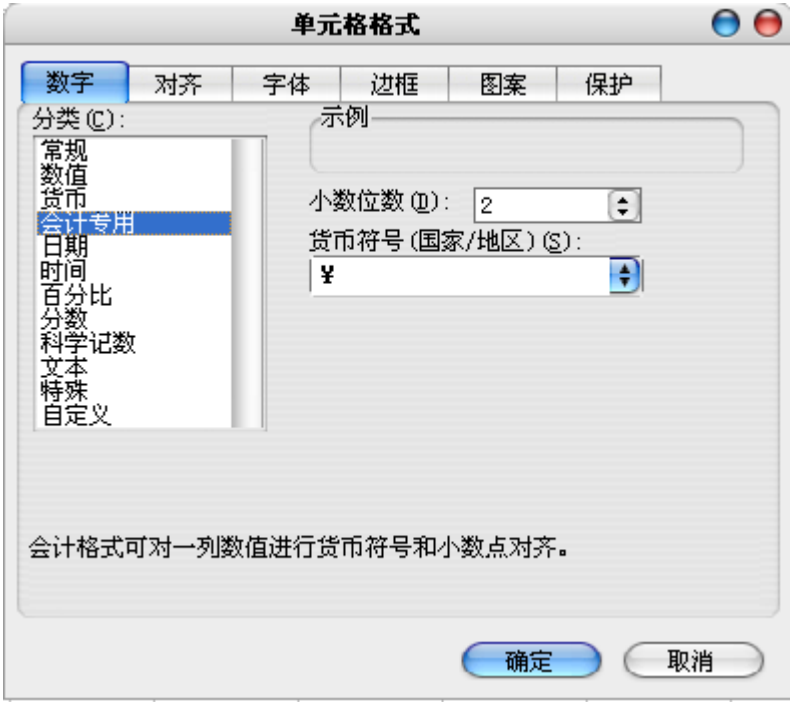


图 11

HSSFDataFormat 的数据格式

内置数据类型	编号
"General"	0
"0"	1
"0.00"	2
"#,##0"	3
"#,##0.00"	4
"(\$#,##0);(\$#,##0)"	5
"(\$#,##0);[Red](\$#,##0)"	6
"(\$#,##0.00);(\$#,##0.00)"	7
"(\$#,##0.00);[Red](\$#,##0.00)"	8
"0%"	9
"0.00%"	0xa
"0.00E+00"	0xb
"# ?/?"	0xc
"# ??/??"	0xd
"m/d/yy"	0xe

"d-mmm-yy"	0xf
"d-mmm"	0x10
"mmm-yy"	0x11
"h:mm AM/PM"	0x12
"h:mm:ss AM/PM"	0x13
"h:mm"	0x14
"h:mm:ss"	0x15
"m/d/yy h:mm"	0x16
保留为过国际化用	0x17 - 0x24
"(##,##0_);(##,##0)"	0x25
"(##,##0_);[Red](##,##0)"	0x26
"(##,##0.00_);(##,##0.00)"	0x27
"(##,##0.00_);[Red](##,##0.00)"	0x28
"_(\$*##,##0_);_(\$*(##,##0);_(\$*\"-\"_);_(@_)"	0x29
"_(\$*##,##0.00_);_(\$*(##,##0.00);_(\$*\"-\"??_);_(@_)"	0x2a
"_(\$*##,##0.00_);_(\$*(##,##0.00);_(\$*\"-\"??_);_(@_)"	0x2b
"_(\$*##,##0.00_);_(\$*(##,##0.00);_(\$*\"-\"??_);_(@_)"	0x2c
"mm:ss"	0x2d
"[h]:mm:ss"	0x2e
"mm:ss.0"	0x2f
"##0.0E+0"	0x30
"@" - This is text format	0x31

3. 复杂写入

3.1 复杂写入

POI 复杂写出主要是同复杂读取一样，在实际开发中，从数据库中读取字符、数字、日期各种类型数据，并通过相应的计算公式写出到 Excel 中。

这次写出的 Excel，不重新创建，而采用 Excel 模板（图 12）形式，将数据从数据库（图 13）中读出，并通过公式方式计算每个学生的总成绩、平均分，最后写入模板。

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	姓名	学号	籍贯	学院	入学年份	数学	语文	英语	理综	总成绩	平均分	是否城镇	入学年份	出生日期
2														
3														
4														
5														

图 12

	ID	NAME	NO	NATIVE	EDU	YEAR	MATH	CHINESE	ENGLISH	SCIENCE	ISCITY	SCHOOLDATE	BIRTH
▶ 1	1	张三	0001	天津	信息学院	2008	140	130	110	280	1	2008-9-1	1990-2-16
2	2	李四	0002	浙江	物流学院	2008	141	131	111	281	0	2008-9-1	1990-8-5
3	3	王五	0003	台湾	化学院	2008	142	132	112	282	0	2008-9-1	1990-12-20

图 13

代码片段:

```

public static HSSFWorkbook write(InputStream inputStream) throws
IOException, ClassNotFoundException{
    Class clazz = Class.forName(Student.class.getName());
    Field[] fields = clazz.getDeclaredFields();

    //初始一个workbook
    HSSFWorkbook workbook = new HSSFWorkbook(inputStream);
    List<Student> list = Conn.getData();
    //创建一个sheet
    HSSFSheet sheet = workbook.getSheetAt(0);
    //创建多行,从列名下一行开始创建
    for(int rowIndex = 1; rowIndex <= list.size(); rowIndex++){
        HSSFRow row = sheet.getRow(rowIndex);
        if(row == null){
            row = sheet.createRow(rowIndex);
        }
        Student student = list.get(rowIndex-1);
        //创建多列(不包括ID列)
        for(int cellnum = 0; cellnum < fields.length-1; cellnum++){
            HSSFCell cell = row.getCell(cellnum);
            if(cell == null){
                cell = row.createCell(cellnum);
            }
            switch (cellnum) {
                case 0:
                    cell.setCellValue(new
HSSFRichTextString(student.getName()));
                    break;
                case 1:
                    cell.setCellValue(new
HSSFRichTextString(student.getNo()));
                    break;
                case 2:
                    cell.setCellValue(new
HSSFRichTextString(student.getNativePlace()));
                    break;
                case 3:
                    cell.setCellValue(new
HSSFRichTextString(student.getEdu()));

```

```
        break;
    case 4:
        cell.setCellValue(student.getYear());
        break;
    case 5:
        cell.setCellValue(student.getMath());
        break;
    case 6:
        cell.setCellValue(student.getChinese());
        break;
    case 7:
        cell.setCellValue(student.getEnglish());
        break;
    case 8:
        cell.setCellValue(student.getScience());
        break;
    case 9:

        cell.setCellValue("F"+(rowIndex+1)+"G"+(rowIndex+1)+"H"+(rowIndex+1)+"I"+(rowIndex+1)); //写入总成绩公式
        break;
    case 10:

        cell.setCellValue("AVERAGE(F"+(rowIndex+1)+":I"+(rowIndex+1)+")"); //写入平均成绩公式
        break;
    case 11:
        cell.setCellValue(student.isCity());
        break;
    case 12:
        cell.setCellValue(new
HSSFRichTextString(formatDate(student.getSchoolDate())));
        break;
    case 13:
        cell.setCellValue(new
HSSFRichTextString(formatDate(student.getBirth())));
        break;
    }
}

return workbook;
}

public static String formatDate(Date date){
```



```
SimpleDateFormat dateFormat = new
SimpleDateFormat("yyyy-MM-dd");
    return dateFormat.format(date);
}
```

Main 方法修改部分:

```
public static void main(String[] args) {
    OutputStream outputStream = null;
    try {
        //读取模板
        InputStream inputStream = new FileInputStream(new
File("E:\\helloPOI.xls"));
        HSSFWorkbook workbook = write(inputStream);
        outputStream = new FileOutputStream(new
File("E:\\helloPOI.xls"));
        workbook.write(outputStream);
    }
    略.....
}
```

依赖类修改部分:

```
public static List<Student> getData() {
    List<Student> list = new ArrayList<Student>();
    Connection conn = getConn();
    String sql = "SELECT * FROM tpoi_vintage t";
    try {
        conn.setAutoCommit(false);
        PreparedStatement ps = conn.prepareStatement(sql);
        ResultSet rs = ps.executeQuery();
        while(rs.next()) {
            Integer id = rs.getInt("id");
            String name = rs.getString("name");
            String no = rs.getString("no");
            String nativePlace = rs.getString("native");
            String edu = rs.getString("edu");
            Integer year = rs.getInt("YEAR");
            Integer math = rs.getInt("MATH");
            Integer chinese = rs.getInt("CHINESE");
            Integer english = rs.getInt("ENGLISH");
            Integer science = rs.getInt("SCIENCE");
            Integer isCity = rs.getInt("ISCITY");
            Date schoolDate = rs.getDate("SCHOOLDATE");
            Date birth = rs.getDate("BIRTH");

            Student student = new Student();
        }
    }
}
```

```
        student.setId(id);
        student.setName(name);
        student.setNo(no);
        student.setNativePlace(nativePlace);
        student.setEdu(edu);
        student.setYear(year);
        student.setMath(math);
        student.setChinese(chinese);
        student.setEnglish(english);
        student.setScience(science);
        if(isCity == 0){
            student.setCity(false);
        }else if(isCity == 1){
            student.setCity(true);
        }
        student.setSchoolDate(schoolDate);
        student.setBirth(birth);

        list.add(student);

    }
    conn.commit();
```

```
public class Student {
    private Integer id;
    private String name;
    private String no;
    private String nativePlace;
    private String edu;
    private Integer year;
    private Integer math;
    private Integer chinese;
    private Integer english;
    private Integer science;
    private Integer scores;
    private Integer ave;
    private boolean isCity;
    private Date schoolDate;
    private Date birth;
```

略... .

输出结果图 14:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	姓名	学号	籍贯	学院	入学年份	数学	语文	英语	理综	总成绩	平均分	是否城镇	入学年份	出生日期
2	张三	0001	天津	信息学院	2008	140	130	110	280	660	165	TRUE	2008-09-01	1990-02-16
3	李四	0002	浙江	物流学院	2008	141	131	111	281	664	166	FALSE	2008-09-01	1990-08-05
4	王五	0003	台湾	化学院	2008	142	132	112	282	668	167	FALSE	2008-09-01	1990-12-20

年份	数学	语文	英语	理综	总成绩	平均分
2008	140	130	110	280	=F2+G2+H2+I2	660
2008	141	131	111	281		664

年份	数学	语文	英语	理综	总成绩	平均分	是否城镇
2008	140	130	110	280	660	165	TRUE
2008	141	131	111	281	664	=AVERAGE(F3:I3)	
2008	142	132	112	282	668	AVERAGE(number1, [n	

图 14

3.2 多层公式

多层公式引用是实际开发中常见的情况，例如有 A 列、B 列、C 列数据，A 列为初始就有数值，B 列为公式依赖于 A 列，C 列为公式依赖于 A 列和 B 列，这样的情况，以及各种统计计算。在此多层公式引用情况下，有可能打开 Excel 后，公式列获取焦点后再手动移开，公式才生效，所以需要设置 `sheet.setForceFormulaRecalculation(true)`。

现在模拟下多层公式引用，增加一行平均分，通过“总成绩/4”计算，增加一行总计信息。

代码片段：

```
public static HSSFWorkbook write(InputStream inputStream) throws
IOException, ClassNotFoundException{
    Class clazz = Class.forName(Student.class.getName());
    Field[] fields = clazz.getDeclaredFields();

    //初始一个workbook
    HSSFWorkbook workbook = new HSSFWorkbook(inputStream);
    List<Student> list = Conn.getData();
    //创建一个sheet
    HSSFSheet sheet = workbook.getSheetAt(0);
    sheet.setForceFormulaRecalculation(true);
    int currentRowIndex = 0;
    //创建多行,从列名下一行开始创建
    for(int rowIndex = 1; rowIndex <= list.size(); rowIndex++){
        currentRowIndex = rowIndex;
        HSSFRow row = sheet.getRow(rowIndex);
        if(row == null){
            row = sheet.createRow(rowIndex);
        }
    }
}
```

```
    }
    Student student = list.get(rowIndex-1);
    //创建多列 (不包括ID列)
    for(int cellnum = 0; cellnum < fields.length; cellnum++){
        HSSFCell cell = row.getCell(cellnum);
        if(cell == null){
            cell = row.createCell(cellnum);
        }
        switch (cellnum) {
            case 0:
                cell.setCellValue(new
HSSFRichTextString(student.getName()));
                break;
            case 1:
                cell.setCellValue(new
HSSFRichTextString(student.getNo()));
                break;
            case 2:
                cell.setCellValue(new
HSSFRichTextString(student.getNativePlace()));
                break;
            case 3:
                cell.setCellValue(new
HSSFRichTextString(student.getEdu()));
                break;
            case 4:
                cell.setCellValue(student.getYear());
                break;
            case 5:
                cell.setCellValue(student.getMath());
                break;
            case 6:
                cell.setCellValue(student.getChinese());
                break;
            case 7:
                cell.setCellValue(student.getEnglish());
                break;
            case 8:
                cell.setCellValue(student.getScience());
                break;
            case 9:

                cell.setCellFormula("F"+(rowIndex+1)+"G"+(rowIndex+1)+"H"+(rowI
ndex+1)+"I"+(rowIndex+1));
```

```
                break;
            case 10:

                cell.setCellFormula("AVERAGE(F"+(rowIndex+1)+":I"+(rowIndex+1)+")");

                break;
            case 11:
                cell.setCellFormula("J"+(rowIndex+1)+"/4");
                break;
            case 12:
                cell.setCellValue(student.isCity());
                break;
            case 13:
                cell.setCellValue(new
HSSFRichTextString(formatDate(student.getSchoolDate())));
                break;
            case 14:
                cell.setCellValue(new
HSSFRichTextString(formatDate(student.getBirth())));
                break;
        }
    }

    currentRowIndex++;
    HSSFRow row = sheet.createRow(currentRowIndex);
    HSSFCell cell0 = row.createCell(0);
    HSSFCell cell5 = row.createCell(5);
    HSSFCell cell6 = row.createCell(6);
    HSSFCell cell7 = row.createCell(7);
    HSSFCell cell8 = row.createCell(8);
    HSSFCell cell9 = row.createCell(9);
    HSSFCell cell10 = row.createCell(10);
    HSSFCell cell11 = row.createCell(11);
    cell0.setCellValue(new HSSFRichTextString("总计"));
    cell5.setCellFormula("SUM(F1:F"+currentRowIndex+")");
    cell6.setCellFormula("SUM(G1:G"+currentRowIndex+")");
    cell7.setCellFormula("SUM(H1:H"+currentRowIndex+")");
    cell8.setCellFormula("SUM(I1:I"+currentRowIndex+")");
    cell9.setCellFormula("SUM(J1:J"+currentRowIndex+")");
    cell10.setCellFormula("AVERAGE(K1:K"+currentRowIndex+")");
    cell11.setCellFormula("AVERAGE(L1:L"+currentRowIndex+")");

    return workbook;
}
```

输出结果：

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	姓名	学号	籍贯	学院	入学年份	数学	语文	英语	理综	总成绩	平均分	平均分 (总/4)	是否城镇户口	入学年份	出生日期
2	张三	0001	天津	信息学院	2008	140	130	110	280	660	165	165	TRUE	2008-09-01	1990-02-16
3	李四	0002	浙江	物流学院	2008	141	131	111	281	664	166	166	FALSE	2008-09-01	1990-08-05
4	王五	0003	台湾	化学院	2008	142	132	112	282	668	167	167	FALSE	2008-09-01	1990-12-20
5	总计					423	393	333	843	1992	166	166			

I	J	K	L
理综	总成绩	平均分	平均分 (总/4)
280	660	165	=J2/4
281	664	166	
282	668	167	
843	=SUM(J1:J4)		

J	K	L
总成绩	平均分	平均分 (总/4)
660	165	
664	166	
668	167	
1992	=AVERAGE(K1:K4)	

图 15

4.常用操作

4.1 注释

4.1.1 单表注释

POI 支持 Excel 单元格加注释功能。创建注释需要通过 sheet 表去创建类 `org.apache.poi.hssf.usermodel.HSSFPatriarch`，`HSSFPatriarch` 实例再通过 `HSSFClientAnchor` 去创建 `HSSFComment`。

`HSSFClientAnchor` 类需要设置大小与位置、注释内容、作者等，以下为 `HSSFComment` 参数说明：

参数	说明
dx1	第 1 个单元格中 x 轴的偏移量
dy1	第 1 个单元格中 y 轴的偏移量
dx2	第 2 个单元格中 x 轴的偏移量
dy2	第 2 个单元格中 y 轴的偏移量
col1	第 1 个单元格的列号
row1	第 1 个单元格的行号
col2	第 2 个单元格的列号
row2	第 2 个单元格的行号

下面在之前输出的 Excel 上创建注释，以 D 列、F 列、K 列、M 列、O 列的各列序号加入注释。

代码片段：

```
public static HSSFWorkbook write(InputStream inputStream) throws
IOException, ClassNotFoundException{
    //初始一个workbook
    HSSFWorkbook workbook = new HSSFWorkbook(inputStream);
    List<Student> list = Conn.getData();
    //创建一个sheet
    HSSFSheet sheet = workbook.getSheetAt(0);
```

```
//创建本sheet的HSSF Patriarch
HSSF Patriarch patriarch = sheet.createDrawingPatriarch();
for(int rowIndex = 1; rowIndex <= sheet.getLastRowNum();
rowIndex++){
    HSSFRow row = sheet.getRow(rowIndex);
    for(int cellnum = 0; cellnum < row.getLastCellNum();
cellnum++){
        HSSFCell cell = row.getCell(cellnum);
        if(cell == null){
            continue;
        }
        switch (cellnum) {
            case 3:
                addComment(patriarch, cell, cellnum);
                break;
            case 5:
                addComment(patriarch, cell, cellnum);
                break;
            case 10:
                addComment(patriarch, cell, cellnum);
                break;
            case 12:
                addComment(patriarch, cell, cellnum);
                break;
            case 14:
                addComment(patriarch, cell, cellnum);
                break;
        }
    }
}

return workbook;
}

/**
 * 设置单元格注释
 * @param patriarch
 * @param cell
 * @param cellnum
 */
public static void addComment(HSSFPatriarch patriarch, HSSFCell cell,
int cellnum){
    HSSFComment comment = patriarch.createComment(new
```

```

HSSFClientAnchor(0, 0, 0, 0, (short)1, 2, (short)4, 4));
    comment.setString(new
HSSFRichTextString(String.valueOf(cellnum)));
    cell.setCellComment(comment);
}

```

输入结果:

R1:O	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	姓名	学号	籍贯	学院	入学年份	数学	语文	英语	理综	总成绩	平均分	平均分 (总/4)	是否城镇	入学年份	出生日期
2	张三	0001	天津	信息学院	2008	140	130	110	280	660	165	165	TRUE	2008-09-(1990-02-16	
3	李四	0002	浙江	物流学院	2008	141	131	111	281	664	166	166	FALSE	2008-09-(1990-08-05	
4	王五	0003	台湾	化学院	2008	142	132	112	282	668	167	167	FALSE	2008-09-(1990-12-20	
5	总计					423	393	333	843	1992	166	166			
6															

C	D	E	F	G
贯	学院	入学年份	数学	语文
津	信息学院	2008	140	130
江	物流学院	2008	141	131
湾	化学院	2008	142	132
			423	393

K	L
平均分	平均分 (总/4)
0	165
4	166
8	167
2	166

O	P
是否城镇	入学年份
TRUE	2008-09-(1990-02-16
FALSE	2008-09-(1990-08-05
FALSE	2008-09-(1990-12-20

图 16

4.1.2 多表注释

在此多表注释开发中需要注意：一个 sheet 表应共用一个 HSSFPatriarch 实例，通过 HSSFPatriarch 可以创建出 sheet 上的注释，如果一个 Excel 表中有多个 sheet，那么根据每个 sheet 去分别实例化该 sheet 上的 HSSFPatriarch，然后分别去创建各个 sheet 上的注释。

将上面的 excel 数据再复制出 2 个 sheet，然后按照上面的方式加注释。

代码片段：

```

public static HSSFWorkbook write(InputStream inputStream) throws
IOException, ClassNotFoundException{
    //初始一个workbook
    HSSFWorkbook workbook = new HSSFWorkbook(inputStream);
    List<Student> list = Conn.getData();
    //循环创建sheet
    for(int sheetIndex = 0; sheetIndex <
workbook.getNumberOfSheets(); sheetIndex++){
        HSSFSheet sheet = workbook.getSheetAt(sheetIndex);
        //创建本个sheet的HSSFPatriarch

```



```
HSSFPatriarch patriarch = sheet.createDrawingPatriarch();
for(int rowIndex = 1; rowIndex <= sheet.getLastRowNum();
rowIndex++){
    HSSFRow row = sheet.getRow(rowIndex);
    for(int cellnum = 0; cellnum < row.getLastCellNum();
cellnum++){
        HSSFCell cell = row.getCell(cellnum);
        if(cell == null){
            continue;
        }
        switch (cellnum) {
        case 3:
            addComment(patriarch, cell, cellnum);
            break;
        case 5:
            addComment(patriarch, cell, cellnum);
            break;
        case 10:
            addComment(patriarch, cell, cellnum);
            break;
        case 12:
            addComment(patriarch, cell, cellnum);
            break;
        case 14:
            addComment(patriarch, cell, cellnum);
            break;
        }
    }
}

return workbook;
}
```

输出结果如下:

The screenshot shows an Excel spreadsheet with two sheets, Sheet1 and Sheet2. Sheet1 contains student data with comments added to cells D11, F11, I11, K11, and M11. Sheet2 contains the same data with comments added to cells F12, I12, and M12. The comments are visible as small text boxes next to the cells.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	姓名	学号	籍贯	学院	入学年份	数学	语文	英语	理综	总成绩	平均分	平均分 (总/4)	是否城镇	入学年份	出生日期
2	张三	0001	天津	信息学院	2008	140	130	110	280	660	165	165	TRUE	2008-09	1990-02-16
3	李四	0002	浙江	物流学院	2008	141	131	111	281	664	166	166	FALSE	2008-09	1990-08-05
4	王五	0003	台湾	化学院	2008	142	132	112	282	668	167	167	FALSE	2008-09	1990-12-20
5	总计					423	393	333	843	1992	166	166			

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	姓名	学号	籍贯	学院	入学年份	数学	语文	英语	理综	总成绩	平均分	平均分 (总/4)	是否城镇	入学年份	出生日期
2	张三	0001	天津	信息学院	2008	140	130	110	280	660	165	165	TRUE	2008-09	1990-02-16
3	李四	0002	浙江	物流学院	2008	141	131	111	281	664	166	166	FALSE	2008-09	1990-08-05
4	王五	0003	台湾	化学院	2008	142	132	112	282	668	167	167	FALSE	2008-09	1990-12-20
5	总计					423	393	333	843	1992	166	166			

图 17

4.1.3 空单元格注释

空单元格加注释，需要先创建一个单元格实例，可以设置成 CELL_TYPE_BLANK 或者不设置，然后再加上注释。

代码片段：

```
public static HSSFWorkbook write(InputStream inputStream) throws
IOException, ClassNotFoundException{
    //初始一个workbook
    HSSFWorkbook workbook = new HSSFWorkbook(inputStream);
    //获取第一张sheet
    HSSFSheet sheet = workbook.getSheetAt(0);
    //创建此表的HSSFPatriarch实例
    HSSFPatriarch patriarch = sheet.createDrawingPatriarch();
    //创建一行
    HSSFRow row = sheet.createRow(0);
    //创建一个空单元格
    HSSFCell cell = row.createCell(0,HSSFCell.CELL_TYPE_BLANK);
    //添加注释
    addComment(patriarch, cell, 0);

    return workbook;
}
```

输出结果：

	A	B	C	D
1		0		
2				
3				
4				
5				
6				

图 18

4.2 单元格合并与数据读取

POI 中支持单元格合并，主要类为 org.apache.poi.hssf.util.Region, 通过 new Region(rowFrom, colFrom, rowTo, colTo)设置合并的行列，四个参数说明如

下:

参数	说明
rowFrom	合并单元格的起始行 (POI 中 row 的标号)
colFrom	合并单元格的起始列 (POI 中 row 的标号)
rowTo	合并单元格的结束行 (POI 中 column 的标号)
colTo	合并单元格的结束列 (POI 中 column 的标号)

如图 19, 写一个 sheet 的 title 是合并的。步骤是: 先创建每个合并单元格的第一个单元格, 将数据写入到单元格, 再进行单元格合并。

	A	B	C	D	E	F	G	H	I	J	K
1				2010年度休假数据					2011年度休假数据		
2	工号	姓名	部门	2010法定总假(天)	2010弹性总假(天)	2010病假总假(天)	2010补充总假(天)	2011法定总假(天)	2011弹性总假(天)	2011病假总假(天)	2011补充总假(天)
3											
4											

图 19

代码片段:

```
@SuppressWarnings("deprecation")

public static HSSFWorkbook write(InputStream inputStream) throws
IOException, ClassNotFoundException{
    //初始一个workbook
    HSSFWorkbook workbook = new HSSFWorkbook(inputStream);
    //获取第一张sheet
    HSSFSheet sheet = workbook.createSheet("合并单元格");
    //创建合并单元格的第一个单元格数据
    HSSFRow row = sheet.createRow(0);
    HSSFCell c0 = row.createCell(0);
    c0.setCellValue(new HSSFRichTextString("工号"));
    HSSFCell c1 = row.createCell(1);
    c1.setCellValue(new HSSFRichTextString("姓名"));
    HSSFCell c2 = row.createCell(2);
    c2.setCellValue(new HSSFRichTextString("部门"));
    HSSFCell c3 = row.createCell(3);
    c3.setCellValue(new HSSFRichTextString("2010年度休假数据"));
    HSSFCell c4 = row.createCell(7);
    c4.setCellValue(new HSSFRichTextString("2011年度休假数据"));

    HSSFRow row1 = sheet.createRow(1);
    HSSFCell c5 = row1.createCell(3);
```

```
c5.setCellValue(new HSSFRichTextString("2010法定总假(天)"));
HSSFCell c6 = row1.createCell(4);
c6.setCellValue(new HSSFRichTextString("2010弹性总假(天)"));
HSSFCell c7 = row1.createCell(5);
c7.setCellValue(new HSSFRichTextString("2010病假总假(天)"));
HSSFCell c8 = row1.createCell(6);
c8.setCellValue(new HSSFRichTextString("2010补充总假(天)"));

HSSFCell c9 = row1.createCell(7);
c9.setCellValue(new HSSFRichTextString("2011法定总假(天)"));
HSSFCell c10 = row1.createCell(8);
c10.setCellValue(new HSSFRichTextString("2011弹性总假(天)"));
HSSFCell c11 = row1.createCell(9);
c11.setCellValue(new HSSFRichTextString("2011病假总假(天)"));
HSSFCell c12 = row1.createCell(10);
c12.setCellValue(new HSSFRichTextString("2011补充总假(天)"));

//设置合并单元格的区域
Region region1 = new Region(0, (short)0, 1, (short)0);
Region region2 = new Region(0, (short)1, 1, (short)1);
Region region3 = new Region(0, (short)2, 1, (short)2);
Region region4 = new Region(0, (short)3, 0, (short)6);
Region region5 = new Region(0, (short)7, 0, (short)10);

sheet.addMergedRegion(region1);
sheet.addMergedRegion(region2);
sheet.addMergedRegion(region3);
sheet.addMergedRegion(region4);
sheet.addMergedRegion(region5);

return workbook;
}
```

合并后的单元格数据读取主要是读取合并前的第一个单元格的内容即可(默认合并单元格保留数据为第一个单元格的数据), 所以我们在将以上合并后的单元格进行读取操作。

代码片段:

```
public static HSSFWorkbook write(InputStream inputStream) throws
IOException, ClassNotFoundException{
    //初始一个workbook
    HSSFWorkbook workbook = new HSSFWorkbook(inputStream);
    //获取第一张sheet
```

```
HSSFSheet sheet = workbook.getSheet("合并单元格");  
//循环所有的合并单元格  
for(int numMR = 0; numMR < sheet.getNumMergedRegions(); numMR++) {  
    //获取合并单元格  
    Region region = sheet.getMergedRegionAt(numMR);  
    //获取合并单元格的第一个单元格  
    HSSFCell cell =  
sheet.getRow(region.getRowFrom()).getCell(region.getColumnFrom());  
    System.out.println("第"+(numMR+1)+"个合并单元格值为：  
"+cell.getRichStringCellValue().toString());  
}  
return workbook;  
}
```

输出结果：

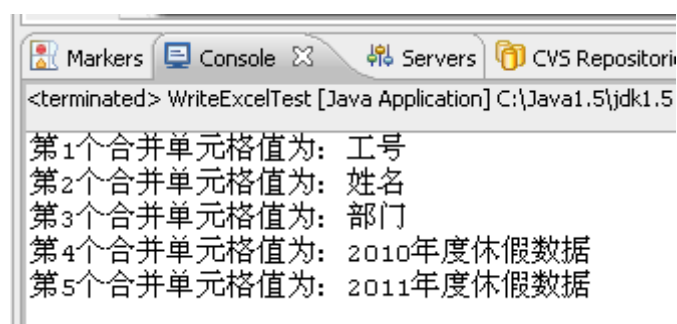


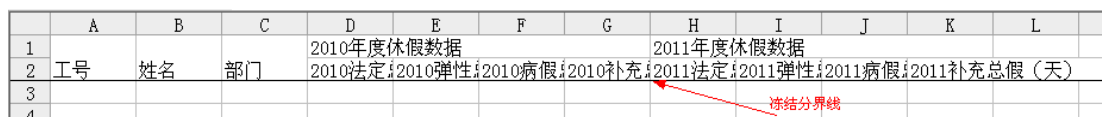
图 20

4.3 窗口冻结

在很多 Excel 中，由于数据过多需要对相关行、相关列进行冻结设置。POI 同样支持行列冻结，行列冻结主要由拆分行列后冻结而成。通过 sheet 的 createFreezePane(colSplit, rowSplit, leftmostColumn, topRow) 方法。参数说明如下：

参数	说明
colSplit	拆分单元格的列（Excel 中 row 的标号）
rowSplit	拆分单元格的行（Excel 中 row 的标号）
leftmostColumn	右边区域可见的左边列数(Excel 中 column 的标号)
topRow	下面区域可见的首行（Excel 中 column 的标号）

现在我们需要冻结标题行，即第 2 行以上冻结，在上面代码中 return 前 加上 sheet.createFreezePane(0, 2, 0, 2);



	A	B	C	D	E	F	G	H	I	J	K	L
1				2010年度休假数据				2011年度休假数据				
2	工号	姓名	部门	2010法定	2010弹性	2010病假	2010补充	2011法定	2011弹性	2011病假	2011补充总假(天)	
3												
4												

图 21

再冻结最左边的三列，那么修改为 `sheet.createFreezePane(3, 2, 3, 2);`

	A	B	C	D	E	F	G	H	I	J	K	L	M
1				2010年度休假数据				2011年度休假数据					
2	工号	姓名	部门	2010法定	2010弹性	2010病假	2010补充	2011法定	2011弹性	2011病假	2011补充总假(天)		
3													
4													

图 22

4.4 下拉列表

POI 对下拉列表稍优于 JXL, jxl 纯 java 下 build 下拉列表会出现下拉列表数量限度（之前我曾经碰到过此类问题）。两个下拉列表原理：首先创建一列隐藏数据，然后将此列中的数据 build 到下拉列表中展示。

主要类说明：

类名	说明
CellRangeAddressList	New CellRangeAddress(firstRow,lastRow, firstCol, lastCol)四个参数表示设置成下拉列表的单元格范围
DVConstraint	生成下拉列表的内容
HSSFDataValidation	将下拉内容与下拉框进行绑定

下面将 Excel 第一列中的 1-10 行设置成下拉列表。

代码片段：

```
public static HSSFWorkbook write(InputStream inputStream) throws
IOException, ClassNotFoundException{
    //初始一个workbook
    HSSFWorkbook workbook = new HSSFWorkbook(inputStream);
    //创建一个sheet
    HSSFSheet sheet = workbook.getSheetAt(0);
    //准备下拉列表数据
    String[] strs = new
String[]{"aa", "bb", "cc", "dd", "ee", "ff", "gg", "hh", "ii"};
    //设置第一列的1-10行为下拉列表
    CellRangeAddressList regions = new CellRangeAddressList(0, 9, 0,
0);
    //创建下拉列表数据
    DVConstraint constraint =
DVConstraint.createExplicitListConstraint(strs);
    //绑定
```

```
        HSSFDataValidation dataValidation = new
HSSFDataValidation(regions, constraint);
        sheet.addValidationData(dataValidation);

        return workbook;
    }
```

输出结果：

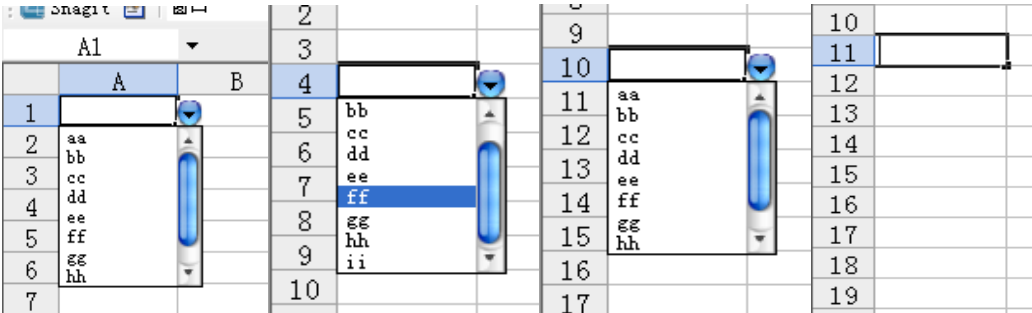


图 23

注：如果想设置成整列都是下拉列表即将参数 lastRow 设置为 65535。目前下拉列表这块好像只支持到 128 个数量。

5.POI 样式

在 Excel 应用中，会需要用到各种样式，包括单元格背景色、边框、高度、宽度、内容相对位置、字体格式、字体大小、字体颜色等等。POI 提供了一系列的样式，能满足我们一般开发中的需求。

5.1 POI 样式相关类

POI 设置 Excel 样式主要通过以下几个相关类：

参数	说明
HSSFCellStyle	POI 样式类
HSSFFont	字体样式类
HSSFColor	颜色类
HSSFBorderFormatting	边框样式类

HSSFCellStyle 是最基本的样式类。HSSFCellStyle 可以直接对上下左右四个边框、内容相对位置、单元格背景色、单元格填充方式、数字格式等进行设置。HSSFFont 是字体样式类，需 set 到 HSSFCellStyle 生效。HSSFFont 可以设置字体样式、字体大小、字体颜色等。HSSFColor 是颜色类，该类里面有大部分基本颜色属性，提供基本颜色的直接调用。

5.2 单元格边框样式

单元格边框分为上、下、左、右四部分，可以设置其边框的宽度、样式及颜色。边框默认为无边框，若加上边框默认为黑色。

现在为一行 10 个单元格全部加上边框（细），并设置颜色为红色。

代码片段：

```
public static HSSFWorkbook write(InputStream inputStream) throws
IOException, ClassNotFoundException{
    //初始一个workbook
    HSSFWorkbook workbook = new HSSFWorkbook(inputStream);
    //获取样式
    HSSFCellStyle style = createCellStyle(workbook);
    //创建一个sheet
    HSSFSheet sheet = workbook.getSheetAt(0);
    HSSFRow row = sheet.createRow(1);
    for(int cellnum = 0; cellnum < 10; cellnum++){
        HSSFCell cell = row.createCell(cellnum);
        cell.setCellStyle(style);
    }
    return workbook;
}

/**
 * 设置单元格的边框（细）且为红色
 * @param workbook
 * @param cellnum
 * @return
 */
public static HSSFCellStyle createCellStyle(HSSFWorkbook workbook){
    HSSFCellStyle style = workbook.createCellStyle();
    //设置上下左右四个边框宽度
    style.setBorderTop(HSSFBorderFormatting.BORDER_THIN);
    style.setBorderBottom(HSSFBorderFormatting.BORDER_THIN);
    style.setBorderLeft(HSSFBorderFormatting.BORDER_THIN);
    style.setBorderRight(HSSFBorderFormatting.BORDER_THIN);
    //设置上下左右四个边框颜色
    style.setTopBorderColor(HSSFColor.RED.index);
    style.setBottomBorderColor(HSSFColor.RED.index);
    style.setLeftBorderColor(HSSFColor.RED.index);
    style.setRightBorderColor(HSSFColor.RED.index);

    return style;
}
```


输出结果：

	A	B	C	D	E	F	G	H	I	J
1										
2										
3										

图 24

5.3 单元格背景色

单元格背景色填充涉及到填充颜色和填充方式，现以最常用的填充方式填充天蓝色背景。

代码片段：

在上面代码中加入：

```
//设置单元格背景色
style.setFillForegroundColor(HSSFColor.SKY_BLUE.index);
style.setFillPattern(HSSFCellStyle.SOLID_FOREGROUND);
```

输出结果：

	A	B	C	D	E	F	G	H	I	J
1										
2										
3										

图 25

注：单元格背景色暂不做详解，在附录中我会列出颜色作为开发参考。

5.4 单元格字体格式

字体格式包括书写体、颜色、大小、加粗、斜体、下划线、删除线等。

以下为 POI 中字体默认属性：

字体属性	Default
书写体	宋体
颜色	黑色
大小	12 有待考证
加粗	无
斜体	无
下划线	无
删除线	无

现将字体设置为幼圆、9px、颜色黄色、加粗、斜体、下划线、删除线作为示例。

代码片段：

在上面代码中加入：

设置数据到单元格代码片段（略）

```
//设置字体格式
```

```
HSSFFont font = workbook.createFont();
font.setFontName("幼圆");
font.setFontHeightInPoints((short)9);
font.setColor(HSSFColor.YELLOW.index);
font.setBoldweight(font.BOLDWEIGHT_BOLD);
font.setItalic(true);
font.setStrikeout(true);
font.setUnderline((byte)1);
//将字体格式设置到HSSFCellStyle上
style.setFont(font);
```

输出结果:

	A	B	C	D	E	F	G	H	I	J
1										
2	#50	#51	#52	#53	#54	#55	#56	#57	#58	#59
3										

图 26

5.5 单元格对齐方式

单元格的对齐方式是针对单元格中的内容，单元格中的内容可以靠左、靠右、靠上、靠下、以及垂直居中、水平居中等等。

针对以上各种对齐方式，将目前 6.4 中输出结果 Excel 基础上进行微调后操作，将目前基本对齐方式进行展示。

代码片段:

```
public static HSSFWorkbook write(InputStream inputStream) throws
IOException, ClassNotFoundException{
    //初始一个workbook
    HSSFWorkbook workbook = new HSSFWorkbook(inputStream);
    //创建一个sheet
    HSSFSheet sheet = workbook.getSheetAt(0);
    HSSFRow row = sheet.getRow(1);
    for(int cellnum = 0; cellnum < 13; cellnum++){
        HSSFCell cell = row.getCell(cellnum);
        //获取样式
        HSSFCellStyle style = createCellStyle(workbook);
        cell.setCellStyle(style);
    }
    return workbook;
}

/**
 * 设置单元格的边框（细）且为红色
 * @param workbook
 * @param cellnum
```

```
* @return
*/
public static HSSFCellStyle createCellStyle(HSSFWorkbook workbook) {
    HSSFCellStyle style = workbook.createCellStyle();
    //设置上下左右四个边框宽度
    style.setBorderTop(HSSFBorderFormatting.BORDER_THIN);
    style.setBorderBottom(HSSFBorderFormatting.BORDER_THIN);
    style.setBorderLeft(HSSFBorderFormatting.BORDER_THIN);
    style.setBorderRight(HSSFBorderFormatting.BORDER_THIN);
    //设置上下左右四个边框颜色
    style.setTopBorderColor(HSSFColor.RED.index);
    style.setBottomBorderColor(HSSFColor.RED.index);
    style.setLeftBorderColor(HSSFColor.RED.index);
    style.setRightBorderColor(HSSFColor.RED.index);
    //设置单元格背景色
    style.setFillForegroundColor(HSSFColor.SKY_BLUE.index);
    style.setFillPattern(HSSFCellStyle.SOLID_FOREGROUND);
    //设置字体格式
    HSSFFont font = workbook.createFont();
    font.setFontName("幼圆");
    font.setFontHeightInPoints((short)9);
    font.setColor(HSSFColor.YELLOW.index);
    font.setBoldweight(font.BOLDWEIGHT_BOLD);
    font.setItalic(true);
    font.setStrikeout(true);
    font.setUnderline((byte)1);
    //将字体格式设置到HSSFCellStyle上
    style.setFont(font);
    //设置单元格居中方式
    setCellAlign(style);
    return style;
}

private static void setCellAlign(HSSFCellStyle style) {
    switch (num) {
        case 0:
            //此单元格格式暂不动，默认进行对照
            num++;
            break;
        case 1:
            style.setAlignment(HSSFCellStyle.ALIGN_LEFT); //靠左
            num++;
            break;
        case 2:
```

```
style.setAlignment(HSSFCellStyle.ALIGN_RIGHT); //靠右
num++;
break;
case 3:
style.setAlignment(HSSFCellStyle.ALIGN_CENTER); //水平居中
num++;
break;
case 4:
style.setVerticalAlignment(HSSFCellStyle.VERTICAL_TOP); //垂直靠上
num++;
break;
case 5:
style.setVerticalAlignment(HSSFCellStyle.VERTICAL_BOTTOM); //垂直靠下
num++;
break;
case 6:
style.setVerticalAlignment(HSSFCellStyle.VERTICAL_CENTER); //垂直居中
num++;
break;
case 7:
style.setVerticalAlignment(HSSFCellStyle.VERTICAL_JUSTIFY); //垂直平铺
num++;
break;
case 8:
style.setAlignment(HSSFCellStyle.ALIGN_CENTER_SELECTION); //跨列居中
num++;
break;
case 9:
style.setAlignment(HSSFCellStyle.ALIGN_FILL); //填充
num++;
break;
case 10:
style.setAlignment(HSSFCellStyle.ALIGN_GENERAL); //普通默认
num++;
break;
case 11:
```

```

        style.setAlignment(HSSFCellStyle.ALIGN_JUSTIFY); //两端对齐
        num++;
        break;
    case 12:

        style.setVerticalAlignment(HSSFCellStyle.VERTICAL_CENTER); //水平居中且垂直居中
        style.setAlignment(HSSFCellStyle.ALIGN_CENTER);
        num++;
        break;
    }
}

```

输出结果:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1					#54			#53					
2	#50	#51	#52	#53		#55			#58	#59	#59	#59	#59
3													

图 27

5.6 单元格数字格式化

实际应用开发中，涉及到大量的数据，而各行、各列数据的格式不尽相同，有要求整数型，有要求精确到小数点两位等。解决此类问题一种方式是我们用 java 将数据进行 format 后输出，但此种情况导致的问题是输出数据格式为 String 类型。在这里 POI 为我们提供了单元格数据格式化的方法。

现将 Excel 中创建 4 个单元格依上面的样式，并设置每个单元格的值都是 double count = 666.1415926; 然后进行下面的 format。

代码片段:

```

    private static void setCellFormat(HSSFWorkbook workbook,
HSSFCellStyle style){
        HSSFFormat format = workbook.createDataFormat();
        switch (num) {
            case 0:
                //此单元格格式暂不动，默认进行对照
                num++;
                break;
            case 1:
                style.setDataFormat(format.getFormat("##.00"));
                num++;
                break;
            case 2:

```

```
        style.setDataFormat(format.getFormat("##.000"));
        num++;
        break;
    case 3:
        style.setDataFormat(format.getFormat("###"));
        num++;
        break;
    }
}
```

输出结果：

	A	B	C	D
1				
2	666.1415926	666.14	666.142	666
3				

图 28

注：POI 中的数字格式化支持四舍五入。

5.7 单元格宽度与高度

Excel 默认的单元格宽度与高度有限，如果我们写入的内容比较长或者需要换行等都涉及到调整宽度与高度。所以需要进行单元格的宽度与高度设置。

这个设置很简单，通过 `HSSFSheet` 设置整列的宽度，通过 `HSSFRow` 设置正行的高度。POI 中宽度、高度设置需要换算，换算单位如下：

	单位	用法 (10px)
高度	15.625	15.625*10
宽度	35.7	35.7*10

现在做一个示范，将 Excel 的第一、二行设置为高度为 40px（两种方法），A 列宽度为 80px。

代码片段：

```
public static HSSFWorkbook write(InputStream inputStream) throws
IOException, ClassNotFoundException{
    //初始一个workbook
    HSSFWorkbook workbook = new HSSFWorkbook(inputStream);
    //创建一个sheet
    HSSFSheet sheet = workbook.getSheetAt(0);
    HSSFRow row0 = sheet.createRow(0);
    HSSFRow row1 = sheet.createRow(1);
    //设置行高40px
    row0.setHeight((short) (15.625*40));
    row1.setHeightInPoints((float) 40);
}
```

```
//设置列宽100px
sheet.setColumnWidth(0, (int) 35.7*100);
return workbook;
}
```

输出结果：

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2												
3												

图 29

5.8 合并单元格样式

单元格合并后依然是 N 个单元格，所以目前在设置合并后的单元格样式时需要分解每一个单元格去设置。

如下图，我们将标题行合并单元格加边框、背景色等样式。

	A	B	C	D	E	F	G	H	I	J	K	L
1				2010年度休假数据				2011年度休假数据				
2	工号	姓名	部门	2010法定	2010弹性	2010病假	2010补充	2011法定	2011弹性	2011病假	2011补充	2011补充总假(天)
3												

图 30

代码片段：

```
public static HSSFWorkbook write(InputStream inputStream) throws
IOException, ClassNotFoundException{
    //初始一个workbook
    HSSFWorkbook workbook = new HSSFWorkbook(inputStream);
    //获取第一张sheet
    HSSFSheet sheet = workbook.getSheet("合并单元格");
    //准备样式
    Map<String,HSSFCellStyle> styleMap = createCellStyle(workbook);
    //循环所有的合并单元格
    for(int numMR = 0; numMR < sheet.getNumMergedRegions(); numMR++){
        //获取合并单元格
        Region region = sheet.getMergedRegionAt(numMR);
        //获取合并单元格每一个单元格
        for(int rownum = region.getRowFrom(); rownum <=
region.getRowTo(); rownum++){
            HSSFRow row = sheet.getRow(rownum);
            for(int cellnum = region.getColumnFrom(); cellnum <=
region.getColumnTo(); cellnum++){
                HSSFCell cell = row.getCell(cellnum);
                if(cell == null){
```

```

        cell = row.createCell(cellnum);
    }
    cell.setCellStyle(styleMap.get("titleStyle"));
}

}

return workbook;
}

```

输出结果:

	A	B	C	D	E	F	G	H	I	J	K
1	工号	姓名	部门	2010年度休假数据				2011年度休假数据			
2				2010法定	2010弹性	2010病假	2010补充	2011法定	2011弹性	2011病假	2011补充
3											
4											

图 31

5.9 Excel 样式实例

根据前面 7 小节的内容，我们将 Excel 的样式进行规范下的使用与展示。

将下图中的 Excel

- 1) 标题行（第一、二行）填充为背景天蓝色，标题的字体设置为红色、加粗、12px，标题单元格设置为细边框，垂直且水平居中。
- 2) 内容数据设置为黑色、9px，内容单元格为细边框
- 3) 总计行填充为背景黄色，字体 9px、黑色加粗，单元格为细边框

	A	B	C	D	E	F	G	H	I	J	K
1	部门名称	员工工号	中文名	去年递延法定	去年递延病假	去年递延补充	去年合计	今年法定年假	今年法定病假	今年补充年假	本年合计
2											
3	研发部	888	张三	1	0	3	4	5	5	5	15
4	研发部	888	张三	1	0	3	4	5	5	5	15
5	研发部	888	张三	1	0	3	4	5	5	5	15
6	研发部	888	张三	1	0	3	4	5	5	5	15
7	研发部	888	张三	1	0	3	4	5	5	5	15
8	总计			5	0	15	20	25	25	25	75
9											

图 32

代码片段:

```

public static HSSFWorkbook write(InputStream inputStream) throws
IOException, ClassNotFoundException{
    //初始一个workbook
    HSSFWorkbook workbook = new HSSFWorkbook(inputStream);
    //获取第一张sheet
    HSSFSheet sheet = workbook.getSheet("Sheet1");
    //准备样式
    Map<String,HSSFCellStyle> styleMap = createCellStyle(workbook);
    //设置标题行样式
    for(int numMR = 0; numMR < sheet.getNumMergedRegions(); numMR++){
        //获取合并单元格
    }
}

```



```
Region region = sheet.getMergedRegionAt(numMR);
//获取合并单元格每一个单元格
for(int rownum = region.getRowFrom(); rownum <=
region.getRowTo(); rownum++){
    HSSFRow row = sheet.getRow(rownum);
    for(int cellnum = region.getColumnFrom(); cellnum <=
region.getColumnTo(); cellnum++){
        HSSFCell cell = row.getCell(cellnum);
        if(cell == null){
            cell = row.createCell(cellnum);
        }
        cell.setCellStyle(styleMap.get("titleStyle"));
    }
}
//设置内容样式
for(int cRowNum = 2; cRowNum < sheet.getLastRowNum(); cRowNum++){
    HSSFRow cRow = sheet.getRow(cRowNum);
    for(int cCellnum = 0; cCellnum < cRow.getLastCellNum();
cCellnum++){
        HSSFCell cell = cRow.getCell(cCellnum);
        cell.setCellStyle(styleMap.get("contentStyle"));
    }
}
//设置总计样式
HSSFRow cRow = sheet.getRow(sheet.getLastRowNum());
for(int cCellnum = 0; cCellnum < cRow.getLastCellNum();
cCellnum++){
    HSSFCell cell = cRow.getCell(cCellnum);
    if(cell == null){
        cell = cRow.createCell(cCellnum);
    }
    cell.setCellStyle(styleMap.get("totalStyle"));
}
return workbook;
}

public static Map<String, HSSFCellStyle> createCellStyle(HSSFWorkbook
workbook){
    Map<String, HSSFCellStyle> styleMap = new HashMap<String,
HSSFCellStyle>();
    //标题格式
    HSSFCellStyle titleStyle = workbook.createCellStyle();
    titleStyle.setAlignment(HSSFCellStyle.ALIGN_CENTER);
```

```
titleStyle.setVerticalAlignment(HSSFCellStyle.VERTICAL_CENTER);
    setCellStyle(titleStyle);
    setBoldCellFontStyle(workbook, titleStyle, (short)10,
HSSFColor.RED.index);
    setBackgroundStyle(titleStyle, HSSFColor.SKY_BLUE.index);
    styleMap.put("titleStyle", titleStyle);
    //内容样式
    HSSFCellStyle contentStyle = workbook.createCellStyle();
    setCellStyle(contentStyle);
    setSimpleCellFontStyle(workbook, contentStyle, (short)9,
HSSFColor.BLACK.index);
    styleMap.put("contentStyle", contentStyle);
    //总计样式
    HSSFCellStyle totalStyle = workbook.createCellStyle();
    setCellStyle(totalStyle);
    setBoldCellFontStyle(workbook, totalStyle, (short)9,
HSSFColor.BLACK.index);
    setBackgroundStyle(totalStyle, HSSFColor.YELLOW.index);
    styleMap.put("totalStyle", totalStyle);
    return styleMap;
}

private static HSSFCellStyle setCellStyle(HSSFCellStyle
cellStyle){
    cellStyle.setBorderLeft(HSSFCellStyle.BORDER_THIN);
    cellStyle.setBorderRight(HSSFCellStyle.BORDER_THIN);
    cellStyle.setBorderTop(HSSFCellStyle.BORDER_THIN);
    cellStyle.setBorderBottom(HSSFCellStyle.BORDER_THIN);
    return cellStyle;
}

private static HSSFCellStyle setSimpleCellFontStyle(HSSFWorkbook
workbook,HSSFCellStyle cellStyle, short size, short color){
    HSSFFont font = workbook.createFont();
    font.setFontHeightInPoints(size);
    font.setColor(color);
    cellStyle.setFont(font);
    return cellStyle;
}

private static HSSFCellStyle setBoldCellFontStyle(HSSFWorkbook
workbook,HSSFCellStyle cellStyle, short size, short color){
    HSSFFont font = workbook.createFont();
```

```

        font.setBoldweight(font.BOLDWEIGHT_BOLD);
        font.setFontHeightInPoints(size);
        font.setColor(color);
        cellStyle.setFont(font);
        return cellStyle;
    }

    private static HSSFCellStyle setBackgroundStyle(HSSFCellStyle
cellStyle, short color){
        cellStyle.setFillForegroundColor(color);
        cellStyle.setFillPattern(HSSFCellStyle.SOLID_FOREGROUND);
        return cellStyle;
    }

```

输出结果:

	A	B	C	D	E	F	G	H	I	J	K
1	部门名称	员工工号	中文名	去年递延法	去年递延病	去年递延补	去年合计	今年法定年龄	今年法定病	今年补充年	本年合计
2											
3	研发部	888	张三	1	0	3	4	5	5	5	15
4	研发部	888	张三	1	0	3	4	5	5	5	15
5	研发部	888	张三	1	0	3	4	5	5	5	15
6	研发部	888	张三	1	0	3	4	5	5	5	15
7	研发部	888	张三	1	0	3	4	5	5	5	15
8	总计			5	0	15	20	25	25	25	75
9											

图 33

6.总结

以上内容均是基础，在开发中需要大家灵活运用。在这里我想说下，在开发中如果可以将一些固定的内容（比如标题、样式、冻结窗口、合并单元格等）固定下来，那我建议大家还是尽量做到用模板，可以减少由变更导致修改代码的问题。

7.附录

HSSFColor 类:

	A	B	C	D	E
1		LIGHT_CORNFLOWER_BLUE:31	ROYAL_BLUE:30	CORAL:29	ORCHID:28
2	MAROON:25	LEMON_CHIFFON:26	CORNFLOWER_BLUE:24	WHITE:9	LAVENDER:46
3	PALE_BLUE:44	LIGHT_TURQUOISE:41	LIGHT_GREEN:42	LIGHT_YELLOW:43	TAN:47
4	ROSE:45	GREY_25_PERCENT:22	PLUM:61	SKY_BLUE:40	TURQUOISE:15
5	BRIGHT_GREEN:11	YELLOW:13	GOLD:51	PINK:14	GREY_40_PERCENT:55
6	VIOLET:20	LIGHT_BLUE:48	AQUA:49	SEA_GREEN:57	LIME:50
7	LIGHT_ORANGE:52	RED:10	GREY_50_PERCENT:23	BLUE_GREY:54	BLUE:12
8	TEAL:21	GREEN:17	DARK_YELLOW:19	ORANGE:53	DARK_RED:16
9	GREY_80_PERCENT:63	INDIGO:62	DARK_BLUE:18	DARK_TEAL:56	DARK_GREEN:45
10	OLIVE_GREEN:59	BROWN:60			
11					

生成代码:

```
public class TestHSSFColor {
    @SuppressWarnings({ "rawtypes", "unused" })
    public static List<String> SysoColor() throws
        ClassNotFoundException, SecurityException, NoSuchFieldException,
        IllegalArgumentException, IllegalAccessException{
        List<String> list = new ArrayList<String>();
        Class clazz = Class.forName(HSSFColor.class.getName());
        Class[] Classes = clazz.getDeclaredClasses();
        Method[] methods = clazz.getDeclaredMethods();
        for(int i = 0; i < Classes.length; i++){
            Class colorClazz = Classes[i];
            Field field = colorClazz.getDeclaredField("index");
            Object property = field.get(colorClazz);
            list.add(colorClazz.getSimpleName()+":"+property);
        }
        return list;
    }

    public static HSSFWorkbook write(){
        HSSFWorkbook workbook = null;
        try {
            int count = 0;
            List<String> list = SysoColor();
            workbook = new HSSFWorkbook();
            HSSFSheet sheet = workbook.createSheet("HSSFColor");
            double size = list.size();
            double rowlength = size/5.0;
            for(int rownum = 0; rownum < Math.ceil(rowlength); rownum++){
                HSSFRow row = sheet.createRow(rownum);
                for(int cellnum = 0; cellnum < 5; cellnum++){
                    HSSFCell cell = row.createCell(cellnum);
```

```
        if(count >= list.size()){
            break;
        }
        String str = list.get(count);
        count++;
        String indexStr = str.substring(str.indexOf(":")+1,
str.length());

        cell.setCellValue(new HSSFRichTextString(str));
        HSSFCellStyle cellStyle = workbook.createCellStyle();

        cellStyle.setFillForegroundColor(Short.valueOf(indexStr));

        cellStyle.setFillPattern(HSSFCellStyle.SOLID_FOREGROUND);
        cell.setCellStyle(cellStyle);
    }
}
} catch (ClassNotFoundException e) {
    e.printStackTrace();
} catch (SecurityException e) {
    e.printStackTrace();
} catch (IllegalArgumentException e) {
    e.printStackTrace();
} catch (NoSuchFieldException e) {
    e.printStackTrace();
} catch (IllegalAccessException e) {
    e.printStackTrace();
}
}
return workbook;
}

public static void main(String[] args) {
    OutputStream outputStream = null;
    try {
        HSSFWorkbook workbook = TestHSSFCOLOR.write();
        outputStream = new FileOutputStream(new
File("E:\\helloPOI1.xls"));
        workbook.write(outputStream);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } finally{
        if(outputStream != null){
```

```
        try {  
            outputStream.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}  
}
```