jxl 操作 excel 文件的方法

2010-11-4 杭州三部：王飞

最后的“进阶操作”确实很有用处，笔者感觉文档写到这里以后把以前很多不明白的东西搞清楚了，豁然开朗啊。建议仔细看看“数据格式”、“方法字典”和“其他讲解”三章，含金量很高。
文中附带代码，可读性很高，具有代表性，可以直接拿来用，不过其中一些适配字段要自己填。

使用 Windows 操作系统的朋友对 Excel（电子表格）一定不会陌生，但是要使用 Java 语言来操纵 Excel 文件并不是一件容易的事。在 Web 应用日益盛行的今天，通过 Web 来操作 Excel 文件的需求越来越强烈，目前较为流行的操作是在 JSP 或 Servlet 中创建一个 CSV （comma separated values）文件，并将这个文件以 MIME，text/csv 类型返回给浏览器，接着浏览器调用 Excel 并且显示 CSV 文件。这样只是说可以访问到 Excel 文件，但是还不能真正的操纵 Excel 文件，本文将向大家介绍一个开放源码项目，Java Excel API，使用它大家就可以方便地操纵 Excel 文件了。

Java Excel API 简介

Java Excel 是一开放源码项目，通过它 Java 开发人员可以读取 Excel 文件的内容、创建新的 Excel 文件、更新已经存在的 Excel 文件。使用该 API 非 Windows 操作系统也可以通过纯 Java 应用来处理 Excel 数据表。因为是使用 Java 编写的，所以我们在 Web 应用中可以通过 JSP、Servlet 来调用 API 实现对 Excel 数据表的访问。

jxl 是一个韩国人写的 java 操作 excel 的工具, 在开源世界中，有两套比较有影响的 API 可供使用，一个是 POI，一个是 jExcelAPI。其中功能相对 POI 弱一点。但 jExcelAPI 对中文支持非常好，API 是纯 Java 的， 并不依赖 Windows 系统，即使运行在 Linux 下，它同样能够正确的处理 Excel 文件。 另外需要说明的是，这套 API 对图形和图表的支持很有限，而且仅仅识别 PNG 格式。

中国的同志们加油了，啥时候咱也搞个通用的，适应 EXCEL 的不行，适应 WPS 的总行吧。

读写操作

JExcelApi 可以从文件或者输入流进行读取操作。
基本的步骤：

由文件或者输入流创建一个 workbook；

由 workbook 的 getSheet 方法创建一个 Sheet（2 种方法，下标和名字，下标从 0 开始）；

由 Sheet 的 getCell （x，y）方法得到某个 Cell，Cell 对象可以读取它的类型（getType ）、内容（getContents ）等。

下面详细解说（入门级）：

1，创建文件

拟生成一个名为“测试数据.xls”的 Excel 文件，其中第一个工作表被命名为“第一页”，大致效果如下（CreateXLS.java）：

```
//生成 Excel 的类
import java.io.*;
import jxl.*;
import jxl.write.*;

public class CreateXLS {
    public static void main(String args[]) {
        try {
            // 打开文件
            WritableWorkbook book = Workbook.createWorkbook(new File("测试.xls"));
            // 生成名为“第一页”的工作表，参数 0 表示这是第一页
            WritableSheet sheet = book.createSheet("第一页", 0);
            // 在 Label 对象的构造子中指名单元格位置是第一列第一行(0,0)
            // 以及单元格内容为 test
            Label label = new Label(0, 0, "test");
            // 将定义好的单元格添加到工作表中
            sheet.addCell(label);
            /*
             * 生成一个保存数字的单元格 必须使用 Number 的完整包路径，否则有语法歧义 单元格位置是第二列，第一行，值为 789.123
             */
            jxl.write.Number number = new jxl.write.Number(1, 0, 789.123);
            sheet.addCell(number);
            // 写入数据并关闭文件
            book.write();
            book.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

编译执行后，会在当前位置产生一个 Excel 文件。里面有一个叫“第一页”的 sheet，sheet 里第一列第一行、第一列第二行均有值。

2，读取文件

以刚才我们创建的 Excel 文件为例，做一个简单的读取操作，程序代码如下：

```
//读取 Excel 的类
import java.io.*;
import jxl.*;

public class ReadXLS {
    public static void main(String args[]) {
        try {
            Workbook book = Workbook.getWorkbook(new File("测试.xls"));
            // 获得第一个工作表对象
            Sheet sheet = book.getSheet(0);
            // 得到第一列第一行的单元格
            Cell cell1 = sheet.getCell(0, 0);
            String result = cell1.getContents();
            System.out.println(result);
            book.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

程序执行结果：test

如果仅仅是取得 Cell 的值，我们可以方便地通过 getContents()方法，它可以将任何类型的 Cell 值都作为一个字符串返回。
如果有需要知道 Cell 内容的确切类型，API 也提供了一系列的方法。

```
String strc00 = null;
double strc10 = 0.00;
Date strc11 = null;

Cell c00 = rs.getCell(0, 0);
Cell c10 = rs.getCell(1, 0);
Cell c11 = rs.getCell(1, 1);

if(c00.getType() == CellType.LABEL)
{
    LabelCell labelc00 = (LabelCell)c00;
    strc00 = labelc00.getString();
}
if(c10.getType() == CellType.NUMBER)
{
```

```
    NmberCell numc10 = (NumberCell)c10;
    strc10 = numc10.getValue();
}
if(c11.getType() == CellType.DATE)
{
    DateCell datec11 = (DateCell)c11;
    strc11 = datec11.getDate();
}

System.out.println("Cell(0, 0)" + " value : " + strc00 + "; type : " + c00.getType());
System.out.println("Cell(1, 0)" + " value : " + strc10 + "; type : " + c10.getType());
System.out.println("Cell(1, 1)" + " value : " + strc11 + "; type : " + c11.getType());
```

在得到 Cell 对象后，通过 getType()方法可以获得该单元格的类型，然后与 API 提供的基本类型相匹配，强制转换成相应的类型，最后调用相应的取 值方法 getXXX()，就可以得到确定类型的值。API 提供了以下基本类型，与 Excel 的数据格式相对应。

jxl.CellType	
BOOLEAN	CellType
BOOLEAN_FORMULA	CellType
DATE	CellType
EMPTY	CellType
ERROR	CellType
FORMULA_ERROR	CellType
LABEL	CellType
NUMBER	CellType
NUMBER_FORMULA	CellType
STRING_FORMULA	CellType

3，修改文件

利用 jExcelAPI 可以修改已有的 Excel 文件，修改 Excel 文件的时候，除了打开文件的方式不同之外，其他操作和创建 Excel 是一样的。下面的例子是在我们已经生成的 Excel 文件中添加一个工作表：

```
//修改 Excel 的类，添加一个工作表
import java.io.*;
import jxl.*;
import jxl.write.*;

public class UpdateXLS {
    public static void main(String args[]) {
        try {
            // Excel 获得文件
            Workbook wb = Workbook.getWorkbook(new File("测试.xls"));
            // 打开一个文件的副本，并且指定数据写回到原文件
```

```
                WritableWorkbook book = Workbook.createWorkbook(new File("测试.xls"),
                        wb);
                // 添加一个工作表
                WritableSheet sheet = book.createSheet("第二页", 1);
                sheet.addCell(new Label(0, 0, "第二页的测试数据"));
                book.write();
                book.close();
            } catch (Exception e) {
                System.out.println(e);
            }
        }
}
```

高级操作（进阶）

下面是一些进阶操作，我们常常卡壳也是在这里，往往要翻看大量代码才能获得正确的操作方法，而其实我们需要的知识一份比较清晰的说明文档。

1，数据格式

在 Excel 中不涉及复杂的数据类型，能够比较好的处理字串、数字和日期已经能够满足一般的应用

（1） 格式化

字串格式化

字符串的格式化涉及到的是字体、粗细、字号等元素，这些功能主要由 WritableFont 和 WritableCellFormat 类来负责。假设我们在生成一个含有字串的单元格时，使用如下语句，为方便叙述，我们为每一行命令加了编号：

```
//设置字体格式为 excel 支持的格式
WritableFont font1 = new WritableFont(WritableFont.TIMES,16,WritableFont.BOLD);
①
WritableCellFormat format1=new WritableCellFormat(font1); ②
Label label=new Label(0,0,”data 4 test”,format1); ③
```

其中①指定了字串格式：字体为 TIMES，字号 16，加粗显示。WritableFont 有非常丰富的构造子，供不同情况下使用，jExcelAPI 的 java-doc 中有详细列表，这里不再列出。

②处代码使用了 WritableCellFormat 类，这个类非常重要，通过它可以指定单元格的各种属性，后面的单元格格式化中会有更多描述。

③处使用了 Label 类的构造子，指定了字串被赋予那种格式。

在 WritableCellFormat 类中，还有一个很重要的方法是指定数据的对齐方式，比如针对我们上面的实例，可以指定：

```
//把水平对齐方式指定为居中
format1.setAlignment(jxl.format.Alignment.CENTRE);
```

```
//把垂直对齐方式指定为居中
format1.setVerticalAlignment(jxl.format.VerticalAlignment.CENTRE);
//设置自动换行
format1.setWrap(true);
```

数字格式化（下面的 NUMBER 类型是 jxl.write.Number，搞错了会出问题滴）

数字的格式化主要是用来说明这个数字是整形、浮点型或者是小数点后保留几位小数等，用到的类除了最基本的那两个类，还有 NumberFormats 。

```
WritableCellFormat integerFormat = new WritableCellFormat
(NumberFormats.INTEGER);
Number number2 = new Number(0, 4, 3.141519, integerFormat);
sheet.addCell(number2);
WritableCellFormat floatFormat = new WritableCellFormat (NumberFormats.FLOAT);
Number number3 = new Number(1, 4, 3.141519, floatFormat);
sheet.addCell(number3);
```

这样，A5 和 B5 的内容就分别为 3 和 3.14 。

使用 NumberFormats 定义新的格式：

```
NumberFormat fivedps = new NumberFormat("#.#####");
WritableCellFormat fivedpsFormat = new WritableCellFormat(fivedps);
Number number4 = new Number(2, 4, 3.141519, fivedpsFormat);
sheet.addCell(number4);
```

当然，数字的格式也可以有基本的字体、字号等。如：

```
WritableCellFormat fivedpsFontFormat = new WritableCellFormat (times16font,
fivedps);
Number number5 = new Number(3, 4, 3.141519, fivedpsFontFormat);
sheet.addCell(number5);
```

日期格式化

日期的格式和数字的格式做法类似，主要用到的类除了基本的那两个类以外还有 java.text.SimpleDateFormat。

```
Date now = Calendar.getInstance().getTime();
DateFormat customDateFormat = new DateFormat ("dd MMM yyyy hh:mm:ss");
WritableCellFormat dateFormat = new WritableCellFormat (customDateFormat);
DateTime dateCell = new DateTime(0, 6, now, dateFormat);
sheet.addCell(dateCell);
```

（2） 单元格操作

Excel 中很重要的一部分是对单元格的操作，比如行高、列宽、单元格合并等，所幸 jExcelAPI 提供了这些支持。这些操作相对比较简单，下面只介绍一下相关的 API。

需要注意的是，只有放在第一个位置的将会在合并后的单元格中存在，其他单元格的数据将被丢弃；即使合并后，也不能使用其他位置的单元格进行操作

1、 合并单元格

```
WritableSheet.mergeCells(int m,int n,int p,int q);
```

作用是从(m,n)到(p,q)的单元格全部合并，比如：

```
WritableSheet sheet=book.createSheet("第一页",0);
//合并第一列第一行（0，0）到第六列第一行（5,0）的所有单元格
sheet.mergeCells(0,0,5,0);
```

合并既可以是横向的，也可以是纵向的。合并后的单元格不能再次进行合并，否则会触发异常。

当试图合并一个已经合并的单元格的时候，如果数量少的话，只会在面板上打出几个 warning，数量大的话，整个文件将无法输出。

2、 行高和列宽

行高

```
WritableSheet.setRowView(int i,int height);
```

作用是指定第 **i+1** 行的高度，比如：

```
//将第一行的高度设为200
sheet.setRowView(0,200);
```

列宽

```
WritableSheet.setColumnView(int i,int width);
```

作用是指定第 **i+1** 列的宽度，比如：

```
sheet.setColumnView(0,30);
```

（3） 操作图片

```
import java.io.*;
import jxl.*;
import jxl.write.*;

public class imageXLS {
    public static void write() throws Exception {
        WritableWorkbook wwb = Workbook.createWorkbook(new File("c:/1.xls"));
        WritableSheet ws = wwb.createSheet("Test Sheet 1", 0);
        File file = new File(
                "C:\\jbproject\\PVS\\WebRoot\\weekhit\\1109496996281.png");
        WritableImage image = new WritableImage(1, 4, 6, 18, file);
        ws.addImage(image);
        wwb.write();
        wwb.close();
    }
}
```

很简单和插入单元格的方式一样，不过就是参数多了些，WritableImage 这个类继承了 Draw，上面只是他构造方法的一种，最后一个参数不用了说了，前面四个参数的类型都是 double，依次是 x, y, width, height,注意，这里的宽和高可不是图片的宽和高，而是图片所要占的单位格的个数，因为继承的 Draw 所以他的类型必须是 double）。

（4） 电子表格的拷贝

简单，就一个命令，没啥好说的。

```
Workbook workbook = Workbook.getWorkbook(new File("myfile.xls"));

WritableWorkbook copy = Workbook.createWorkbook(new File("output.xls"),
workbook);
```

（5） 读操作，进阶

读的时候是这样的一个思路，先用一个输入流(InputStream)得到 Excel 文件，然后用 jxl 中的 Workbook 得到工作薄，用 Sheet 从工作薄中得到工作表，用 Cell 得到工作表中得某个单元格。

InputStream→Workbook→Sheet→Cell,就得到了 excel 文件中的单元格

```
import java.io.*;
import jxl.*;
import jxl.write.*;
/**
 * @author kfzx-wangfei02
 *
 */
public class readmoreXLS {
    public static void main(String args[]) {
        try {
            String path = "c:\\excel.xls";// Excel文件URL
            InputStream is = new FileInputStream(path);// 写入到FileInputStream
            jxl.Workbook wb = Workbook.getWorkbook(is); // 得到工作薄
            jxl.Sheet st = wb.getSheet(0);// 得到工作薄中的第一个工作表
            Cell cell = st.getCell(0, 0);// 得到工作表的第一个单元格,即A1
            String content = cell.getContents();// getContents()将Cell中的字符
转为字符串
            wb.close();// 关闭工作薄
            is.close();// 关闭输入流
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

典型的从 InputStream 开始读取，一直到 Cell 获取数据的示例。

进阶：

我们可以通过 Sheet 的 getCell(x,y)方法得到任意一个单元格，x,y 和 excel 中的坐标对应。

例如 A1 对应(0,0)，A2 对应(0,1)，D3 对应(3,2)。Excel 中坐标从 A1 开始，jxl 中全部是从 0 开始。

还可以通过 Sheet 的 getRows()，getColumns()方法得到行数列数，并用于循

环控制,输出一个 sheet 中的所有内容。

（6） 写操作，进阶

往 Excel 中写入内容主要是用 jxl.write 包中的类.

思路是这样的：

OutputStream←WritableWorkbook←WritableSheet←Label

这里面 Label 代表的是写入 Sheet 的 Cell 位置及内容。

```
import java.io.*;
import jxl.*;
import jxl.write.*;

public class writemoreXLS {
    public static void main(String args[]) {
        try {

            OutputStream os = new FileOutputStream("c:\\test.xls");// 输出的
Excel文件URL
            WritableWorkbook wwb = Workbook.createWorkbook(os);// 创建可写工作
薄
            WritableSheet ws = wwb.createSheet("sheet1", 0);// 创建可写工作表
            Label labelCF = new Label(0, 0, "hello");// 创建写入位置和内容
            ws.addCell(labelCF);// 将Label写入sheet中

            WritableFont wf = new WritableFont(WritableFont.TIMES, 12,
                    WritableFont.BOLD, false);// 设置写入字体
            WritableCellFormat wcfF = new WritableCellFormat(wf);// 设置
CellFormat
            Label labelCF1 = new Label(0, 0, "hello", wcfF);// 创建写入位置,内
容和格式
            ws.addCell(labelCF1);// 将Label写入sheet中

            // 现在可以写了
            wwb.write();
            // 写完后关闭
            wwb.close();
            // 输出流也关闭吧
            os.close();

        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Label 的构造函数 Label(int x, int y,String aString)，x、y 的意思同读的时候的 x、y，aString 是写入的内容。

Label 的另一构造函数 Label(int c, int r, String cont, CellFormat st)可以对写入内容进行格式化,设置字体及其它的属性。

（7） 程序示例

```
import java.io.*;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.LinkedHashMap;

import com.icbc.cte.base.CTEObjectNotFoundException;
import com.icbc.cte.base.Context;
import com.icbc.ctp.auth.base.BaseDAO;

import jxl.*;
import jxl.write.*;

/**
 * @author kfzx-wangfei02
 *
 */
public class XLS extends BaseDAO {

    /**
     * 构造函数
     */
    public XLS() throws CTEObjectNotFoundException {
        super(Context.getRoot());
    }

    public static void main(String[] args) {
        ResultSet rs = null;
        Connection conn = null;
        PreparedStatement pstmt = null;
        LinkedHashMap lhmap = new LinkedHashMap();

        try {
            conn = getConnection();
            String sql = "select * from tablename";
            pstmt = conn.prepareStatement(sql);
            rs = pstmt.executeQuery(sql);

            // 新建Excel文件
```

```
            String filePath = request.getRealPath("aaa.xls");

            File myFilePath = new File(filePath);
            if (!myFilePath.exists())
                myFilePath.createNewFile();

            FileWriter resultFile = new FileWriter(myFilePath);
            PrintWriter myFile = new PrintWriter(resultFile);
            resultFile.close();
            // 用JXL向新建的文件中添加内容
            OutputStream outf = new FileOutputStream(filePath);
            jxl.write.WritableWorkbook wwb = Workbook.createWorkbook(outf);
            jxl.write.WritableSheet ws = wwb.createSheet("sheettest", 0);

            int i = 0;
            int j = 0;
            for (int k = 0; k < rs.getMetaData().getColumnCount(); k++) {
                ws.addCell(new Label(k, 0, rs.getMetaData()
                        .getColumnName(k + 1)));
            }
            while (rs.next()) {
                System.out.println(rs.getMetaData().getColumnCount());
                for (int k = 0; k < rs.getMetaData().getColumnCount(); k++) {
                    ws.addCell(new Label(k, j + i + 1, rs.getString(k + 1)));
                }
                i++;
            }
            wwb.write();
            wwb.close();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            rs.close();
            conn.close();
        }
        response.sendRedirect("aaa.xls");

    }

}
```

这个要自己钻研，我还是不废话了，查数据库，然后导出到 EXCEL 中大概就是这么个流程，内部格式之类的，自己添上就好了。

（8）

2，方法字典

Workbook 类提供的方法

1. int **getNumberOfSheets**()

   获得工作薄（Workbook）中工作表（Sheet）的个数，示例：

```
jxl.Workbook rwb = jxl.Workbook.getWorkbook(new File(sourcefile));
int sheets = rwb.getNumberOfSheets();
```

2. **Sheet[] getSheets**()

   返回工作薄（Workbook）中工作表（Sheet）对象数组，示例：

```
jxl.Workbook rwb = jxl.Workbook.getWorkbook(new File(sourcefile));
Sheet[] sheets = rwb.getSheets();
```

3. **String getVersion()**

   返回正在使用的 API 的版本号，好像是没什么太大的作用。

```
jxl.Workbook rwb = jxl.Workbook.getWorkbook(new File(sourcefile));
String apiVersion = rwb.getVersion();
```

Sheet 接口提供的方法

1. **String** getName()

   获取 Sheet 的名称，示例：

```
jxl.Workbook rwb = jxl.Workbook.getWorkbook(new File(sourcefile));
jxl.Sheet rs = rwb.getSheet(0);
String sheetName = rs.getName();
```

2. int **getColumns**()

   获取 Sheet 表中所包含的总列数，示例：

```
jxl.Workbook rwb = jxl.Workbook.getWorkbook(new File(sourcefile));
jxl.Sheet rs = rwb.getSheet(0);
int rsColumns = rs.getColumns();
```

3. Cell[] **getColumn**(int column)　　-------这个和上面的不同，少了个 s，不要看错了

   获取某一列的所有单元格，返回的是单元格对象数组，示例：

```
jxl.Workbook rwb = jxl.Workbook.getWorkbook(new File(sourcefile));
jxl.Sheet rs = rwb.getSheet(0);
Cell[] cell = rs.getColumn(0);
```

4. int **getRows**()

   获取 Sheet 表中所包含的总行数，示例：

```
jxl.Workbook rwb = jxl.Workbook.getWorkbook(new File(sourcefile));
jxl.Sheet rs = rwb.getSheet(0);
int rsRows = rs.getRows();
```

5. **Cell[]** getRow(int row)

   获取某一行的所有单元格，返回的是单元格对象数组，示例子：

```
jxl.Workbook rwb = jxl.Workbook.getWorkbook(new File(sourcefile));
jxl.Sheet rs = rwb.getSheet(0);
Cell[] cell = rs.getRow(0);
```

6. Cell getCell(int column, int row)

   获取指定单元格的对象引用，需要注意的是它的两个参数，第一个是列数，第二个是行数，这与通常的行、列组合（第一个是行数，第二个是列数）有些不同。

```
jxl.Workbook rwb = jxl.Workbook.getWorkbook(new File(sourcefile));
jxl.Sheet rs = rwb.getSheet(0);
Cell cell = rs.getCell(0, 0);
```

3，其他讲解

API 提供了两种方式来处理可写入的输出流，一种是直接生成本地文件，如果文件名不带全路径的话，缺省的文件会定位在当前目录，如果文件名带有全路径的话，则生成的 Excel 文件则会定位在相应的目录；另外一种是将 Excel 对象直接写入到输出流，例如：用户通过浏览器来访问 Web 服务器，如果 HTTP 头设置正确的话，浏览器自动调用客户端的 Excel 应用程序，来显示动态生成的 Excel 电子表格。

写入：

```
import java.io.*;
import jxl.*;
import jxl.write.*;

public class template {
    public static void main(String[] args) {
        try {
            // 构建Workbook对象, 只读Workbook对象
            // Method 1：创建可写入的Excel工作薄
            jxl.write.WritableWorkbook wwb = Workbook.createWorkbook(new File(
                    targetfile));

            // Method 2：将WritableWorkbook直接写入到输出流
            /*
             * OutputStream os = new FileOutputStream(targetfile);
             * jxl.write.WritableWorkbook wwb = Workbook.createWorkbook(os);
             */

            // 创建Excel工作表
```

```
            jxl.write.WritableSheet ws = wwb.createSheet("Test Sheet 1", 0);

            // 现在要做的只是实例化API所提供的Excel基本数据类型，并将它们添加到工作表中就可以了

            // 1.添加Label对象
            jxl.write.Label labelC = new jxl.write.Label(0, 0,
                    "This is a Label cell");
            ws.addCell(labelC);

            // 添加带有字型Formatting的对象
            jxl.write.WritableFont wf = new jxl.write.WritableFont(
                    WritableFont.TIMES, 18, WritableFont.BOLD, true);
            jxl.write.WritableCellFormat wcfF = new jxl.write.WritableCellFormat(
                    wf);
            jxl.write.Label labelCF = new jxl.write.Label(1, 0,
                    "This is a Label Cell", wcfF);
            ws.addCell(labelCF);

            // 添加带有字体颜色Formatting的对象
            jxl.write.WritableFont wfc = new jxl.write.WritableFont(
                    WritableFont.ARIAL, 10, WritableFont.NO_BOLD, false,
                    Underlinestyle.NO_UNDERLINE, jxl.format.Colour.RED);
            jxl.write.WritableCellFormat wcfFC = new jxl.write.WritableCellFormat(
                    wfc);
            jxl.write.Label labelCFC = new jxl.write.Label(1, 0,
                    "This is a Label Cell", wcfFC);
            ws.addCell(labelCF);

            // 2.添加Number对象
            jxl.write.Number labelN = new jxl.write.Number(0, 1, 3.1415926);
            ws.addCell(labelN);

            // 添加带有formatting的Number对象
            jxl.write.NumberFormat nf = new jxl.write.NumberFormat("#.##");
            jxl.write.WritableCellFormat wcfN = new jxl.write.WritableCellFormat(
                    nf);
            jxl.write.Number labelNF = new jxl.write.Number(1, 1, 3.1415926,
                    wcfN);
            ws.addCell(labelNF);

            // 3.添加Boolean对象
            jxl.write.Boolean labelB = new jxl.write.Boolean(0, 2, false);
            ws.addCell(labelB);
```

```
            // 4.添加DateTime对象
            jxl.write.DateTime labelDT = new jxl.write.DateTime(0, 3,
                    new java.util.Date());
            ws.addCell(labelDT);

            // 添加带有formatting的DateFormat对象
            jxl.write.DateFormat df = new jxl.write.DateFormat(
                    "dd MM yyyy hh:mm:ss");
            jxl.write.WritableCellFormat wcfDF = new jxl.write.WritableCellFormat(
                    df);
            jxl.write.DateTime labelDTF = new jxl.write.DateTime(1, 3,
                    new java.util.Date(), wcfDF);
            ws.addCell(labelDTF);

            // 写入Exel工作表
            wwb.write();

            // 关闭Excel工作薄对象
            wwb.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

这与读取 Excel 文件的操作有少少不同，在关闭 Excel 对象之前，你必须要先调用 write() 方法，因为先前的操作都是存储在缓存中的，所以要通过该方法将操作的内容保存在文件中。如果你先关闭了 Excel 对象，那么只能得到一张空的工作薄了。

与读取不同，写是分数据类型的，具体有 Label ， Number 和 Date。

下面的例子在上文曾出现过，重温一下。

```
WritableWorkbook workbook = Workbook.createWorkbook(new File("output.xls"));
WritableSheet sheet = workbook.createSheet("First Sheet", 0);

Label label = new Label(0, 2, "A label record");
sheet.addCell(label);

Number number = new Number(3, 4, 3.1459);
sheet.addCell(number);

Date now = Calendar.getInstance().getTime();
DateTime dateCell = new DateTime(0, 6, now);
sheet.addCell(dateCell);
```

```
workbook.write();
workbook.close();
```

修改：

```
//创建只读的Excel工作薄的对象
jxl.Workbook rw = jxl.Workbook.getWorkbook(new File(sourcefile));

//创建可写入的Excel工作薄对象
jxl.write.WritableWorkbook wwb = Workbook.createWorkbook(new File(targetfile), rw);

//读取第一张工作表
jxl.write.WritableSheet ws = wwb.getSheet(0);

//获得第一个单元格对象
jxl.write.WritableCell wc = ws.getWritableCell(0, 0);

//判断单元格的类型, 做出相应的转化
if(wc.getType() == CellType.LABEL)
{
Label l = (Label)wc;
l.setString("The value has been modified.");
}

//写入Excel对象
wwb.write();

//关闭可写入的Excel对象
wwb.close();

//关闭只读的Excel对象
rw.close();
```

之所以使用这种方式构建 Excel 对象，完全是因为效率的原因，因为上面的示例才是 API 的主要应用。为了提高性能，在读取工作表时，与数据相关的一些输出信息，所有的格式信息，如：字体、颜色等等，是不被处理的，因为我们的目的是获得行数据的值，既使没有了修饰，也不会对行数据的值产生什么影响。唯一的不利之处就是，在内存中会同时保存两个同样的工作表，这样当工作表体积比较大时，会占用相当大的内存，但现在好像内存的大小并不是什么关键因素了。

一旦获得了可写入的工作表对象，我们就可以对单元格对象进行更新的操作了，在这里我们不必调用 API 提供的 add()方法，因为单元格已经于工作表当中，所以我们只需要调用相应的 setXXX()方法，就可以完成更新的操作了。

尽单元格原有的格式化修饰是不能去掉的，我们还是可以将新的单元格修饰加上去，以使单元格的内容以不同的形式表现。

新生成的工作表对象是可写入的，我们除了更新原有的单元格外，还可以添加新的单元格到工作表中，这与示例 2 的操作是完全一样的。

最后，不要忘记调用 write()方法，将更新的内容写入到文件中，然后关闭工作薄对象，这里有两个工作薄对象要关闭，一个是只读的，另外一个是可写入的。

小结

本文只是对 Java Excel API 中常用的方法作了介绍，要想更详尽地了解 API，请大家参考 API 文档，或源代码。Java Excel API 是一个开放源码项目，请大家关注它的最新进展，有兴趣的朋友也可以申请加入这个项目，或者是提出宝贵的意见。

参考资料

Java Excel API 文档

http://www.andykhan.com/jexcelapi/

附件：JXL.jar 拷贝就能用

Jxl.jar