

# 尚马教育 JAVA 高级课程

## Excel 与 Word 操作

文档编号：C11

创建日期：2017-07-07

最后修改日期：2019-10-14

版本号：V3.0

电子版文件名：尚马教育-第三阶段-13.excel 与 word 操作.docx

文档修改记录：

更新日期	更新作者	更新说明	版本号
2017-07-30	张元林	初始版本	V1.0
2018-08-01	王绍成	Mybatis 版本更新	V2.0
2019-08-09	冯勇涛	课件格式以及课程深度加深	V3.0
2021-02-26	冯勇涛	添加 maven 依赖	V3.5

## 目录

1. 常用场景 .....	2
2. Excel 操作 .....	4
2.1. 组件介绍 .....	4
2.1.1. JXL .....	4
2.1.2. POI .....	5
2.2. Excel 认识 .....	6
2.3. 读 Excel .....	7
2.3.1. 基本思路 .....	7
2.3.2. 代码演示 .....	7
2.4. 写 Excel .....	9
2.4.1. 代码思路 .....	9
2.4.2. Workbook 对象 .....	9
2.4.3. 代码演示 .....	9
3. Word 操作 .....	11
3.1. 写 word .....	11
3.1.1. 基本思路 .....	11
3.1.2. 编辑 docx 模板 .....	12
3.1.3. 模板另存为 xml .....	12
3.1.4. 代码演示 .....	13

## 1. 常用场景

Office 软件的读写操作时未来开发软件或者办公系统会经常用到的技术。
<b>excel 读写场景:</b>
<p>由于公司办公系统或者政府的政务系统等，面向的人群是使用 office 软件非常熟悉的人群，对于他们来讲，程序员辛辛苦苦开发出来的系统，看似功能强大，在批量录入数据操作习惯性方面，远没有日常使用的 office 来的熟悉，更希望在 Excel 中把的数据编辑好，然后导入 Excel。</p> <p>所以，经常会遇见这种情况：为使用者开发好了很好很强大的功能，但是他会提出一个 Excel 导入数据和导出数据的功能。</p>
<b>Word 使用场景:</b>
<p>政务系统、教育系统经常用到的技术</p> <p>公司项目合同导出，用于打印签字。</p> <p>教育系统问卷导出，用于打印答题。</p> <p>其他需要打印的场景，word 的导出相对于 Excel，使用的概率更少。</p> <p>这里介绍几种 Word 的导出方式</p> <p>没有介绍 Word 的导入方式，实际上不会用 word 来进行导入，因为数据格式十分不可控。</p>

## 2. Excel 操作

### 2.1. 组件介绍

Java 对 Excel 的支持有两个类库：POI 和 JXL，主流使用 POI。

本课程学习 POI 库。

### 2.1.1. JXL

- 下载地址:

<https://sourceforge.net/projects/jxl/>

- 优点:

- 对中文支持非常好, 操作简单, 方法看名知意。
- 是纯 javaAPI, 在跨平台上表现的非常完美, 代码可以再 windows 或者 Linux 上运行而无需重新编写
- 支持 Excel 95-2000 的所有版本
- 生成 Excel 2000 标准格式
- 支持字体、数字、日期操作
- 能够修饰单元格属性
- 支持图像和图表,但是这套 API 对图形和图表的支持很有限, 而且仅识别 PNG 格式

- 缺点:

效率低, 图片支持不完善, 对格式的支持不如 POI 强大。

只能读取 xls 文件, 不能读取 **xlsx 文件**

### 2.1.2. POI

- 下载地址:

<https://poi.apache.org/download.html>

- 优点:

- 支持公式，宏，能保持 Excel 里原有的宏，但不能用它写新的宏
- 能够修饰单元格属性
- 支持字体、数字、日期操作
- 在一些业务场景中代码相对复杂，但是 API 丰富，支持多种模式的读写
- 支持比较新版本的 excel.

- 缺点:

- 不支持跨平台（主要就是 Java 语言）
- 读写的时候比较占内存。
- 支持大数量大文件的读写操作

总体来说，对于简单的单表 excel 导入导出的需求，建议使用 JXL。数据量稍微小点，占用内存少，速度快。

对于报表类的，涉及月份数据量，多表数据聚合在一起建议使用 POI。

## 2.2. Excel 认识

学习之前，对 Excel 的一些知识做一下说明

- 工作簿      Workbook
- 工作表      Sheet
- 单元格      Cell

一个 Excel 是由一个工作簿组成

一个工作簿由多个工作表组成

一个工作表是由若干行组成

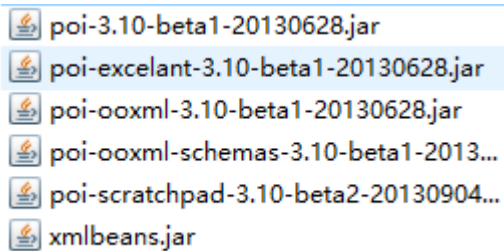
一个工作表/行也是由若干个单元格组成

## 2.3. 添加 poi 依赖

Maven 依赖

```
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi-ooxml</artifactId>
  <version>3.17</version>
</dependency>
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi-scratchpad</artifactId>
  <version>3.17</version>
</dependency>
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi-excelant</artifactId>
  <version>3.17</version>
</dependency>
```

添加 jar 包



poi-3.10-beta1-20130628.jar  
poi-excelant-3.10-beta1-20130628.jar  
poi-ooxml-3.10-beta1-20130628.jar  
poi-ooxml-schemas-3.10-beta1-2013...  
poi-scratchpad-3.10-beta2-20130904...  
xmlbeans.jar

## 2.4. 读 Excel

### 2.4.1. 基本思路

- 获取工作簿
- 获取工作表
- 获取总行数
- 根据第一行获取总列数
- 根据行数和列数，循环获取每一个单元格
- 判断单元格数据类型，输出数据

### 2.4.2. 代码演示

```
public static void readExcel(){  
    try {  
        String excelUrl = "D:\\abc.xlsx";//文件地址  
        File xlsFile = new File(excelUrl);//创建 file 对象  
        //获得工作簿 org.apache.poi.ss.usermodel.Workbook  
        Workbook workbook = WorkbookFactory.create(xlsFile);  
        // 获得工作表个数  
        int sheetCount = workbook.getNumberOfSheets();  
        // 遍历工作表  
        for (int i = 0; i < sheetCount; i++) {
```

```
//获得工作表 org.apache.poi.ss.usermodel.Sheet
Sheet sheet = workbook.getSheetAt(i);

// 获得行数
int rows = sheet.getLastRowNum() + 1;

if(rows <= 1){ continue; }

// 获得列数，先获得一行，在得到该行列数
Row tmp = sheet.getRow(0);

if (tmp == null) { continue; }

//第一行的列数 认为第一行的列数代表整个工作表的列数
int cols = tmp.getPhysicalNumberOfCells();

for (int row = 0; row < rows; row++) { // 读取数据

    Row r = sheet.getRow(row); //行

    for (int col = 0; col < cols; col++) {

        Cell cell = r.getCell(col); //单元格

        int type = cell.getCellType(); //单元格中的数据类型

        String str = "";

        switch (type){

            case Cell.CELL_TYPE_STRING://字符串型 返回值 String

                str = cell.getStringCellValue();

                break;

            case Cell.CELL_TYPE_NUMERIC://数字型返回值 double

                str = String.valueOf(cell.getNumericCellValue());

                break;

        }

        System.out.print(str + "\t");

    }

    System.out.println();

}

}
```



```
    } catch (IOException e) {  
        e.printStackTrace();  
    } catch (InvalidFormatException e) {  
        e.printStackTrace();  
    }  
}
```

## 2.5. 写 Excel

### 2.5.1. 代码思路

- 创建输入文件对象 `FileOutputStream`
- 创建工作簿 `Workbook`
- 创建工作表 `Sheet`
- 创建行 `Row`
- 创建单元格 `Cell`
- 写入数据

### 2.5.2. Workbook 对象

该对象有三个实现类：

- `HSSFWorkbook` 是操作 Excel2003 以前（包括 2003）的版本，扩展名是.xls。该对象生成 excel 有所限制，导出的行数至多为 65535 行，超出 65536 条后系统就会报错。
- `XSSFWorkbook` 是操作 Excel2007 后的版本，扩展名是.xlsx。
- 突破 `HSSFWorkbook` 的 65535 行局限，最多可以导出 104 万行，仍然存在问题---OOM 内存溢出，原因是创建的 `workbook sheet row cell` 对象是在内存的，并

没有持久化。

- **SXSSFWorkbook** 是操作 Excel2007 后的版本，扩展名是.xlsx。
- 对于大型 excel 文件的创建，要确保不会内存溢出，只有使用 **SXSSFWorkbook**，该对象用硬盘空间换内存，避免内存溢出。

### 2. 5. 3. 代码演示

```
public static void writeExcel(){
    try {
        String excelUrl = "D:\\javasm.xlsx";//导出地址
        File excelFile = new File(excelUrl);//创建 File 对象
        FileOutputStream xlsStream = new FileOutputStream(excelFile);
        // 创建工作簿 org.apache.poi.hssf.usermodel.HSSFWorkbook
        HSSFWorkbook workbook = new HSSFWorkbook();
        // 创建工作表 org.apache.poi.hssf.usermodel.HSSFSheet
        HSSFSheet sheet = workbook.createSheet("工作表名称");
        //循环向工作表(Sheet)中添加单元格(Cell)
        String temple = "第%s 行%s 列";
        //假设添加 10 行 8 列的数据
        for (int row = 0; row < 10; row++) {
            //创建一行 org.apache.poi.hssf.usermodel.HSSFRow
            HSSFRow rows = sheet.createRow(row);
            for (int col = 0; col < 8; col++) {
                //org.apache.poi.ss.usermodel.Cell
                Cell cell = rows.createCell(col);//单元格
                // 向工作表中添加数据
                cell.setCellValue(String.format(temple,row,col));
            }
        }
        workbook.write(xlsStream);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

## 3. Word 操作

### 3.1. 写 word

关于 word 的导出，方法非常多，本课程只学习最常用方法：基于 freeMarker 模板引擎进行 word 导出。

Freemarker 不是专门为导出 word 而设计的技术。

FreeMarker 是一款模板引擎，多用于生成 HTML 页面，源代码等，是一种基于模板改变数据生成文本的工具。

FreeMarker 是开源的技术，有自己独特的语法结构

在线文档：<http://freemarker.foofun.cn/ref.html>

文档中有较为详细的教程语法介绍

#### 3.1.1. 添加 freemarker 依赖

```
<!-- https://mvnrepository.com/artifact/org.freemarker/freemarker -->
<dependency>
    <groupId>org.freemarker</groupId>
    <artifactId>freemarker</artifactId>
    <version>2.3.23</version>
</dependency>
```

#### 3.1.2. 基本思路

- 先编辑模板，这是最重要的，编辑 docx，另存为 xml
- 模板中可以有 list，判断，map 等
- Java 中拼装数据，存入 map
- Map 中的 key，是 xml 模板中需要读取的变量
- 模板路径和名称设置正确

- 设置好编码
- 写入

### 3.1.3. 编辑 docx 模板

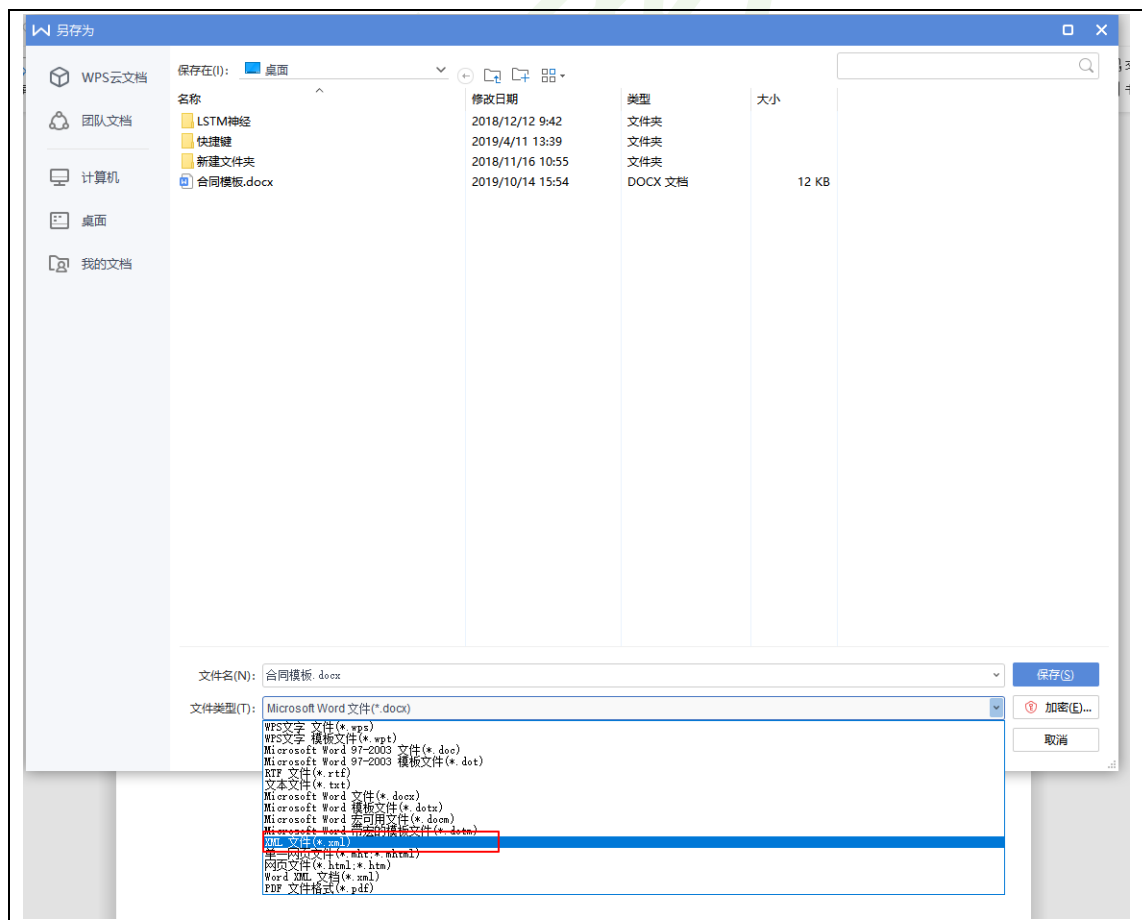
## 租房合同

甲方：房东姓名  
 乙方：\${userName}

身份证：\${userCode} 价格 \${housePrice}.....

日期：\${houseDate}

### 3.1.4. 模板另存为 xml



### 3.1.5. 代码演示

```
public void writeWord(){
    try {
        Map<String,Object> map = new HashMap<>();
        map.put("userName","fyt");
        map.put("userCode","41012345646645821222");
        map.put("housePrice","$2000");
        map.put("houseDate","2019-10-14");
        Writer out = new OutputStreamWriter(new FileOutputStream("F:\\test10.docx"),
"UTF-8");

        //freemarker.template.Configuration
        Configuration configuration = new Configuration();
        configuration.setDefaultEncoding("UTF-8");//设置编码
        //类加载器，设置 xml 模板文件的路径
        configuration.setClassForTemplateLoading(this.getClass(),
"/com/javasm/word/xml");

        //获取模板对象 传入模板名称
        Template t = configuration.getTemplate("test.xml");
        t.process(map, out);//模板中传入数据
        out.close();
    } catch (IOException | TemplateException e) {
        e.printStackTrace();
    }
}
```