

# Poi-tl Documentation

Sayi  
sayi@apache.org  
Version 1.12.2

## Table of Contents

- 1. Why poi-tl
- 2. Getting Started
  - 2.1. 前提
  - 2.2. Maven
  - 2.3. Gradle
  - 2.4. 2分钟入门
  - 2.5. Template: 模板
  - 2.6. Data-model: 数据
  - 2.7. Output: 输出
- 3. 标签
  - 3.1. 文本
  - 3.2. 图片
  - 3.3. 表格
  - 3.4. 列表
  - 3.5. 区块对
  - 3.6. 嵌套
- 4. 引用标签
  - 4.1. 图片
  - 4.2. 多系列图表
  - 4.3. 单系列图表
  - 4.4. 组合图表
- 5. 配置
  - 5.1. 前后缀
  - 5.2. 标签类型
  - 5.3. 标签格式
  - 5.4. Spring表达式
  - 5.5. 错误处理
  - 5.6. 模板生成模板
  - 5.7. 无模板创建文档
  - 5.8. 日志
- 6. 插件
  - 6.1. 默认插件
  - 6.2. 开发一个插件
  - 6.3. 使用插件
  - 6.4. 插件列表
  - 6.5. 表格行循环
  - 6.6. 表格列循环
  - 6.7. 动态表格
  - 6.8. 批注
  - 6.9. 插入附件
  - 6.10. 代码高亮
  - 6.11. Markdown
- 7. 示例
  - 7.1. 软件说明文档
  - 7.2. 付款通知书
  - 7.3. 野生动物现状

- 7.4. 证书奖状
- 7.5. 个人简历
- 7.6. Swagger文档
- 8. 源码
- 9. License
- 10. 版本
- 11. 打赏个小费
- 12. VIP专属服务
- 13. 常见问题

poi-tl (poi template language) 是Word模板引擎，使用模板和数据创建很棒的**Word**文档。

“ 在文档的任何地方做任何事情 (*Do Anything Anywhere*) 是`poi-tl`的星辰大海。

## 1. Why poi-tl

方案	移植性	功能性	易用性
Poi-tl	Java跨平台	Word模板引擎，基于Apache POI，提供更友好的API	低代码，准备文档模板和数据即可
Apache POI	Java跨平台	Apache项目，封装了常见的文档操作，也可以操作底层XML结构	文档不全，这里有一个教程： <a href="http://deepoove.com/poi-tl/apache-poi-guide.html">Apache POI Word快速入门</a> ( <a href="http://deepoove.com/poi-tl/apache-poi-guide.html">http://deepoove.com/poi-tl/apache-poi-guide.html</a> )
Freemarker	XML跨平台	仅支持文本，很大的局限性	不推荐，XML结构的代码几乎无法维护
OpenOffice	部署OpenOffice，移植性较差	-	需要了解OpenOffice的API
HTML浏览器导出	依赖浏览器的实现，移植性较差	HTML不能很好的兼容Word的格式，样式糟糕	-
Jacob、winlib	Windows平台	-	复杂，完全不推荐使用

**poi-tl**是一个基于Apache POI的Word模板引擎，也是一个免费开源的Java类库，你可以非常方便的加入到你的项目中，并且拥有着让人喜悦的特性。

Word模板引擎功能	描述
✓ 文本	将标签渲染为文本
✓ 图片	将标签渲染为图片
✓ 表格	将标签渲染为表格
✓ 列表	将标签渲染为列表
✓ 图表	条形图（3D条形图）、柱形图（3D柱形图）、面积图（3D面积图）、折线图（3D折线图）、雷达图、饼图（3D饼图）、散点图等图表渲染
✓ If Condition判断	根据条件隐藏或者显示某些文档内容（包括文本、段落、图片、表格、列表、图表等）
✓ Foreach Loop循环	根据集合循环某些文档内容（包括文本、段落、图片、表格、列表、图表等）
✓ Loop表格行	循环复制渲染表格的某一行
✓ Loop表格列	循环复制渲染表格的某一列

Word模板引擎功能	描述
✓ Loop有序列表	支持有序列表的循环，同时支持多级列表
✓ Highlight代码高亮	word中代码块高亮展示，支持26种语言和上百种着色样式
✓ Markdown	将Markdown渲染为word文档
✓ Word批注	完整的批注功能，创建批注、修改批注等
✓ Word附件	Word中插入附件
✓ SDT内容控件	内容控件内标签支持
✓ Textbox文本框	文本框内标签支持
✓ 图片替换	将原有图片替换成另一张图片
✓ 书签、锚点、超链接	支持设置书签，文档内锚点和超链接功能
✓ Expression Language	完全支持SpringEL表达式，可以扩展更多的表达式：OGNL, MVEL...
✓ 样式	模板即样式，同时代码也可以设置样式
✓ 模板嵌套	模板包含子模板，子模板再包含子模板
✓ 合并	Word合并Merge，也可以在指定位置进行合并
✓ 用户自定义函数(插件)	插件化设计，在文档任何位置执行函数

## 2. Getting Started

### 2.1. 前提

- JDK1.8+
- Apache POI5.2.2+

### 2.2. Maven

```
<dependency>
    <groupId>com.deeboove</groupId>
    <artifactId>poi-tl</artifactId>
    <version>1.12.2</version>
</dependency>
```

XML

### 2.3. Gradle

```
implementation 'com.deeboove:poi-tl:1.12.2'
```

GROOVY

### 2.4. 2分钟入门

新建Word文档template.docx，包含标签 {{title}}

*template.docx*

  {{title}}

代码示例

```
XWPFTemplate template = XWPFTemplate.compile("template.docx").render(
    new HashMap<String, Object>() {{
        put("title", "Hi, poi-tl Word模板引擎");
    }}); ① ②
template.writeAndClose(new FileOutputStream("output.docx")); ③
```

JAVA

- ① compile 编译模板
- ② render 渲染数据
- ③ write 输出到流

TDO模式：Template + data-model = output

*output.docx*

  Hi, poi-tl Word模板引擎

## 2.5. Template: 模板

模板是Docx格式的Word文档，你可以使用Microsoft office、WPS Office、Pages等任何你喜欢的软件制作模板，也可以使用Apache POI代码来生成模板。

所有的标签都是以 {{ 开头，以 }} 结尾，标签可以出现在任何位置，包括页眉，页脚，表格内部，文本框等，表格布局可以设计出很多优秀专业的文档，推荐使用表格布局。

poi-tl模板遵循“所见即所得”的设计，模板和标签的样式会被完全保留。

## 2.6. Data-model: 数据

数据类似于哈希或者字典，可以是Map结构（key是标签名称）：

```
Map<String, Object> data = new HashMap<>();
data.put("name", "Sayi");
data.put("start_time", "2019-08-04");
```

JAVA

可以是对象（属性名是标签名称）：

```
public class Data {
    private String name;
    private String startTime;
    private Author author;
}
```

JAVA

数据可以是树结构，每级之间用点来分隔开，比如 {{author.name}} 标签对应的数据是author对象的name属性值。



FreeMarker、Velocity文本模板中可以通过三个标签设置图片路径、宽和高： ，但是Word模板不是由简单的文本表示，所以在渲染图片、表格等元素时提供了数据模型，它们都实现了接口 `RenderData`，比如图片数据模型 `PictureRenderData` 包含图片路径、宽、高三个属性。

## 2.7. Output: 输出

以流的方式进行输出：

```
template.write(OutputStream stream);
```

JAVA

比如文件流：

```
template.write(new FileOutputStream("output.docx"));
```

JAVA

比如网络流：

```
response.setContentType("application/octet-stream");
response.setHeader("Content-disposition","attachment;filename="""+out_template.docx+"\"");
// HttpServletResponse response
OutputStream out = response.getOutputStream();
BufferedOutputStream bos = new BufferedOutputStream(out);
template.write(bos);
bos.flush();
out.flush();
```

最后不要忘记关闭这些流。

```
PoitlIOWutils.closeQuietlyMulti(template, bos, out);
```

JAVA

### 3. 标签

poi-tl是一种无逻辑「logic-less」的模板引擎，没有复杂的控制结构和变量赋值，只有标签。标签由前后两个大括号组成，{{title}}是标签，{{?title}}也是标签，title是这个标签的名称，问号标识了标签类型，接下来我们来看看有哪些默认标签类型（用户可以创建新的标签类型，这属于更高级的话题）。

### 3.1. 文本

{{var}}

## 数据模型:

- `String` : 文本
  - `TextRenderData` : 有样式的文本
  - `HyperlinkTextRenderData` : 超链接和锚点文本
  - `Object` : 调用 `toString()` 方法转化为文本

推荐使用工厂 `Texts` 构建文本模型。

## 代码示例

```
put("name", "Sayi");
put("author", Texts.of("Sayi").color("000000").create());
put("link", Texts.of("website").link("http://deepoove.com").create());
put("anchor", Texts.of("anchortxt").anchor("appendix1").create());
```

JAVA

所见即所得，标签的样式会应用到替换后的文本上，也可以通过代码设定文本的样式。

## TextRenderData的结构体

```
{  
  "text": "Sayi",  
  "style": {  
    "strike": false, ①  
    "bold": true, ②  
    "italic": false, ③  
    "color": "#00FF00", ④  
    "underLine": false, ⑤  
    "fontFamily": "微软雅黑", ⑥  
    "fontSize": 12, ⑦  
    "highlightColor": "green", ⑧  
    "vertAlign": "superscript", ⑨  
    "characterSpacing": 20 ⑩  
  }  
}
```

JSON

## 1 删 除 线

- ② 粗体
- ③ 斜体
- ④ 颜色
- ⑤ 下划线
- ⑥ 字体
- ⑦ 字号
- ⑧ 背景高亮色
- ⑨ 上标或者下标
- ⑩ 间距



文本换行使用 `\n` 字符。

### 3.2. 图片

图片标签以@开始: `{{@var}}`

数据模型:

- `String` : 图片url或者本地路径, 默认使用图片自身尺寸
- `ByteArrayPictureRenderData`
- `FilePictureRenderData`
- `UrlPictureRenderData`

推荐使用工厂 `Pictures` 构建图片模型。

## 代码示例

JAVA

```
// 指定图片路径
put("image", "logo.png");
// svg图片
put("svg", "https://img.shields.io/badge/jdk-1.6%2B-orange.svg");

// 图片文件
put("image1", Pictures.ofLocal("logo.png").size(120, 120).create());

// 图片流
put("streamImg", Pictures.ofStream(new FileInputStream("logo.jpeg"), PictureType.JPG)
    .size(100, 100).create());

// 网络图片(注意网络耗时对系统可能的性能影响)
put("urlImg", Pictures.ofUrl("http://deepoove.com/images/icecream.png")
    .size(100, 100).create());

// java图片，我们可以利用Java生成图表插入到word文档中
put("buffered", Pictures.ofBufferedImage(bufferImage, PictureType.PNG)
    .size(100, 100).create());
```

## FilePictureRenderData的结构体

JSON

```
{
  "pictureType": "PNG", ①
  "path": "logo.png", ②
  "pictureStyle": {
    "width": 100, ③
    "height": 100 ④
  },
  "altMeta": "图片不存在" ⑤
}
```

- ① 图片类型
- ② 图片路径
- ③ 宽度，单位是像素
- ④ 高度，单位是像素
- ⑤ 当无法获取图片时展示的文字

### 3.3. 表格

表格标签以#开始: {{#var}}

数据模型:

- TableRenderData

推荐使用工厂 `Tables`、`Rows` 和 `Cells` 构建表格模型。

*Example 1.* 基础表格示例

```
// 一个2行2列的表格
put("table0", Tables.of(new String[][] {
    new String[] { "00", "01" },
    new String[] { "10", "11" }
}).border(BorderStyle.DEFAULT).create());
```

JAVA

00	01
10	11

*Example 2.* 表格样式示例

```
// 第0行居中且背景为蓝色的表格
RowRenderData row0 = Rows.of("姓名", "学历").textColor("FFFFFF")
    .bgColor("4472C4").center().create();
RowRenderData row1 = Rows.create("李四", "博士");
put("table1", Tables.create(row0, row1));
```

JAVA

姓名	学历
李四	博士

*Example 3.* 表格合并示例

```
// 合并第1行所有单元格的表格
RowRenderData row0 = Rows.of("列0", "列1", "列2").center().bgColor("4472C4").create();
RowRenderData row1 = Rows.create("没有数据", null, null);
MergeCellRule rule = MergeCellRule.builder().map(Grid.of(1, 0), Grid.of(1, 2)).build();
put("table3", Tables.of(row0, row1).mergeRule(rule).create());
```

JAVA

列0	列1	列2
没有数据		

TableRenderData表格模型在单元格内可以展示文本和图片，同时也可以指定表格样式、行样式和单元格样式，而且在N行N列渲染完成后可以应用单元格合并规则 **MergeCellRule**，从而实现更复杂的表格。

## TableRenderData的结构体

```

{
    "rows": [ ①
        {
            "cells": [ ②
                {
                    "paragraphs": [ ③
                        {
                            "contents": [
                                {
                                    [TextRenderData] ④
                                },
                                {
                                    [PictureRenderData] ⑤
                                }
                            ],
                            "paragraphStyle": null ⑥
                        }
                    ],
                    "cellStyle": { ⑦
                        "backgroundColor": "000000",
                        "vertAlign": "CENTER"
                    }
                }
            ],
            "rowStyle": { ⑧
                "height": 2.0f
            }
        }
    ],
    "tableStyle": { ⑨
        ⑩
        "width": 14.63f,
        "colWidths": null
    },
    "mergeRule": {
        "mapping": {
            "0-0": "1-2"
        }
    }
}

```

- ① 行数据
- ② 单元格数据
- ③ 单元格内段落
- ④ 单元格内文本
- ⑤ 单元格内图片
- ⑥ 单元格内段落文本的样式：对齐
- ⑦ 单元格样式：垂直对齐方式，背景色
- ⑧ 行样式：行高(单位cm)
- ⑨ 表格样式：表格对齐、边框样式

- ⑩ 表格宽度(单位cm), 表格的最大宽度 = 页面宽度 - 页边距宽度 \* 2, 页面宽度为A4(20.99 \* 29.6, 页边距为3.18 \* 2.54)的文档最大表格宽度14.63cm。
- ⑪ 单元格合并规则, 比如第0行第0列至第1行第2列单元格合并



产品需求中表格的布局和样式可能很复杂, 可以尝试一些已有表格插件来解决, 参见更多插件列表。

我们也可以编写插件, 完全由自己生成整个表格, 前提是需要熟悉Apache POI XWPFTable相关API, 但是自由度最高: 参见 开发一个插件。

### 3.4. 列表

列表标签以\*开始: {{\*var}}

数据模型:

- `List<String>`
- `NumberingRenderData`

推荐使用工厂 `Numberings` 构建列表模型。

#### 代码示例

```
put("list", Numberings.create("Plug-in grammar",
    "Supports word text, pictures, table...",
    "Not just templates"));
```

JAVA

编号样式支持罗马字符、有序无序等, 可以通过 `Numberings.of(NumberingFormat)` 来指定。

```
DECIMAL //1. 2. 3.
DECIMAL_PARENTHESSES //1) 2) 3)
BULLET //● ● ●
LOWER_LETTER //a. b. c.
LOWER_ROMAN //i ii iii
UPPER_LETTER //A. B. C.
```

JAVA



`NumberingRenderData`可以创建多级列表, 但是推荐使用区块对: 区块对的循环功能可以很好的循环列表, 并且保持有序列表编号有序。

### 3.5. 区块对

区块对由前后两个标签组成, 开始标签以?标识, 结束标签以/标识: {{?sections}}{{/sections}}

区块对开始和结束标签中间可以包含多个图片、表格、段落、列表、图表等, 开始和结束标签可以跨多个段落, 也可以在同一个段落, 但是如果在表格中使用区块对, 开始和结束标签必须在同一个单元格内, 因为跨多个单元格的渲染行为是未知的。

区块对在处理一系列文档元素的时候非常有用，位于区块对中的文档元素可以被渲染零次，一次或N次，这取决于区块对的取值。

### False或空集合

隐藏区块中的所有文档元素

### 非False且不是集合

显示区块中的文档元素，渲染一次

### 非空集合

根据集合的大小，循环渲染区块中的文档元素



集合是根据值的类型是否实现了 `Iterable` 接口来判断。

### 3.5.1. False或空集合

如果区块对的值是 `null`、`false` 或者空的集合，位于区块中的所有文档元素将不会显示，这就等同于if语句的条件为 `false`。

#### *data-model*

```
{
  "announce": false
}
```

JSON

#### *template.docx*

Made it,Ma!{{?announce}}Top of the world!{{/announce}}

Made it,Ma!

{{?announce}}

Top of the world!

{{/announce}}

#### *output.docx*

Made it,Ma!

Made it,Ma!

### 3.5.2. 非False且不是集合

如果区块对的值不为 `null`、`false`，且不是集合，位于区块中的所有文档元素会被渲染一次，这就等同于if语句的条件为 `true`。

#### *data-model*

JSON

```
{
  "person": { "name": "Sayi" }
}
```

*template.docx*

{{?person}}

Hi {{name}}!

{{/person}}

*output.docx*

Hi Sayi!



区块对中标签的作用域会被限定在当前区块对内，当且仅当区块对的值是 `boolean` 类型且为 `true` 时，这些标签作用域才不会改变。

### 3.5.3. 非空集合

如果区块对的值是一个非空集合，区块中的文档元素会被迭代渲染一次或者N次，这取决于集合的大小，类似于 `foreach` 语法。

*data-model*

JSON

```
{
  "songs": [
    { "name": "Memories" },
    { "name": "Sugar" },
    { "name": "Last Dance" }
  ]
}
```

*template.docx*

{{?songs}}

{{name}}

{{/songs}}

*output.docx*

Memories

Sugar

Last Dance

## 循环内置变量

在循环中提供了一些内置变量，这些内置变量只能用于区块对中。

变量	类型	说明
_index	int	返回当前迭代从0开始的索引
_is_first	boolean	辨别循环项是否是当前迭代的第一项。
_is_last	boolean	辨别循环项是否是当前迭代的最后一项。
_has_next	boolean	辨别循环项是否有下一项。
_is_even_item	boolean	辨别循环项是否是当前迭代间隔1的奇数项。
_is_odd_item	boolean	辨别循环项是否是当前迭代间隔1的偶数项。
#this	object	引用当前对象，由于#和已有表格标签标识冲突，所以在文本标签中需要使用=号标识来输出文本。

示例数据:

```
{
  "produces": [
    "application/json",
    "application/xml"
  ]
}
```

JSON

template.docx(注意：如果标签内要使用运算符，需要开启Spring表达式):

```
{{?produces}}
{{_index + 1}}. {{=#this}}
{{/produces}}
```

output.docx:

```
1. application/json
2. application/xml
```

HTML

## 3.6. 嵌套

嵌套又称为导入、包含或者合并，以+标识: {{+var}}

数据模型:

- DocxRenderData

推荐使用工厂 `Includes` 构建嵌套模型。

## 代码示例

JAVA

```
class AddrModel {  
    private String addr;  
    public AddrModel(String addr) {  
        this.addr = addr;  
    }  
    // Getter/Setter  
}  
  
List<AddrModel> subData = new ArrayList<>();  
subData.add(new AddrModel("Hangzhou,China"));  
subData.add(new AddrModel("Shanghai,China"));  
put("nested", Includes.ofLocal("sub.docx").setRenderModel(subData).create()); ① ②
```

① 主模板包含嵌套标签{{+nested}}

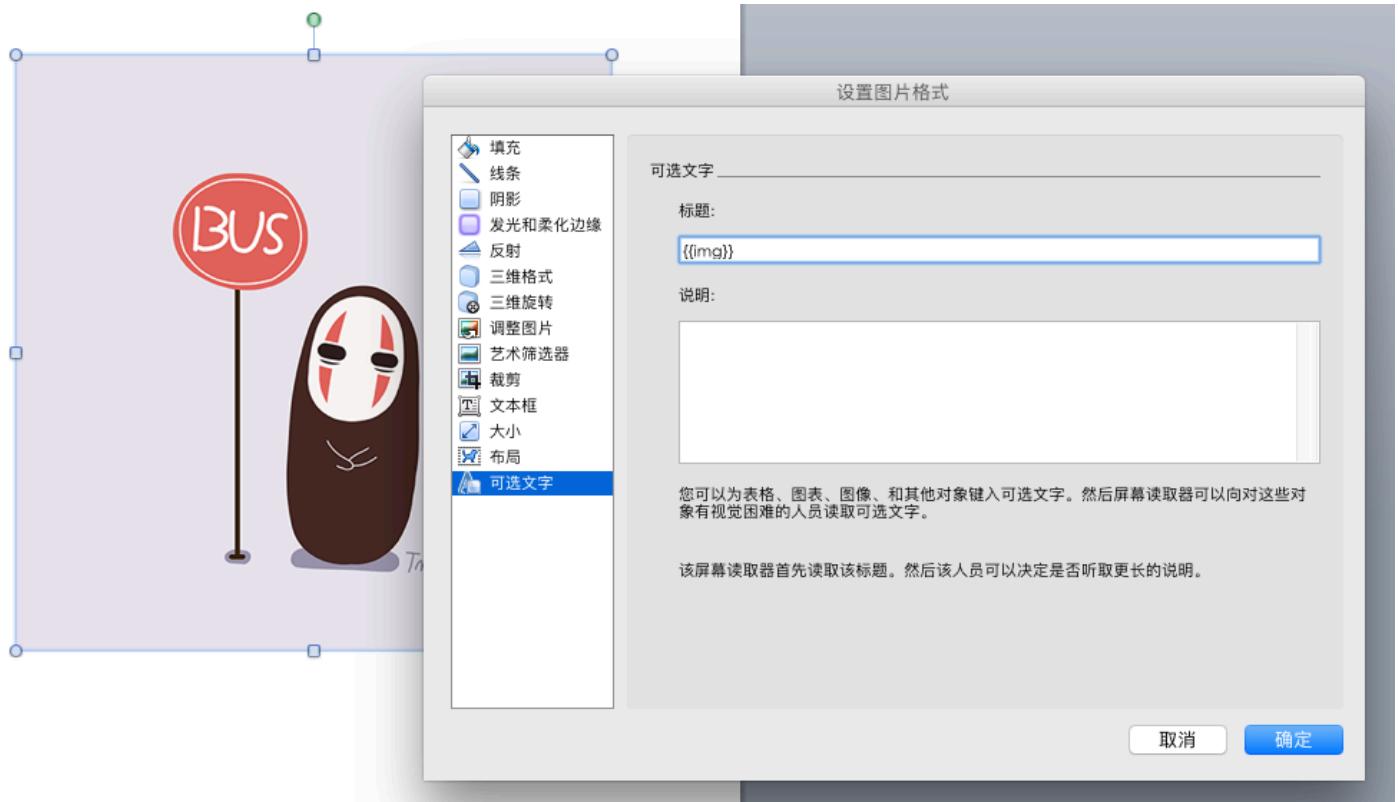
② sub.docx是一个包含了{{addr}}的子模板，使用subData集合渲染后合并到主模板

## 4. 引用标签

引用标签是一种特殊位置的特殊标签，提供了直接引用文档中的元素句柄的能力，这个重要的特性在我们只想改变文档中某个元素极小一部分样式和属性的时候特别有用，因为其余样式和属性都可以在模板中预置好，真正的所见即所得。

### 4.1. 图片

引用图片标签是一个文本：{{var}}，标签位置在：设置图片格式—可选文字—标题或者说明（新版本Microsoft Office标签位置在：编辑替换文字-替换文字）。



引用图片标签只会替换图片而不会改变图片尺寸和布局，数据模型和图片标签一致：`PictureRenderData`。

#### 代码示例

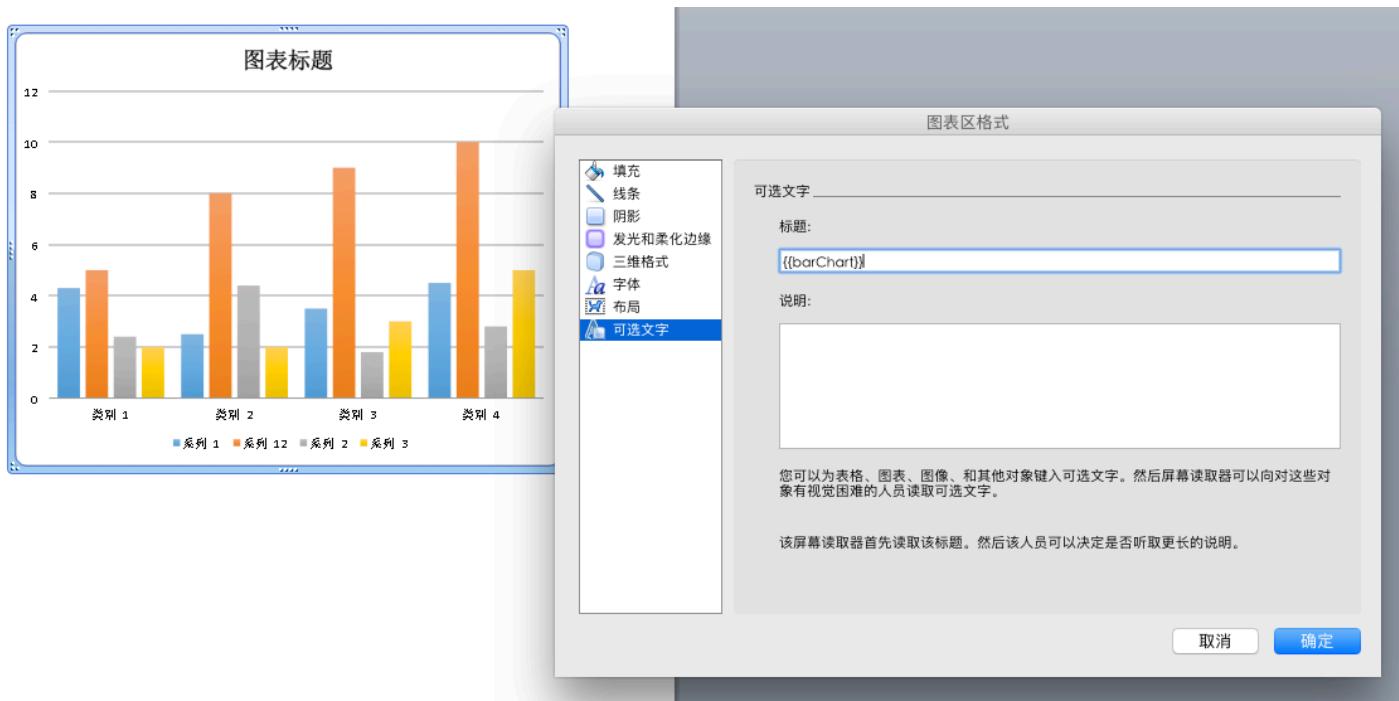
```
put("img", Pictures.ofLocal("sayi.png").create());
```

JAVA

### 4.2. 多系列图表

多系列图表指的是条形图（3D条形图）、柱形图（3D柱形图）、面积图（3D面积图）、折线图（3D折线图）、雷达图、散点图等。

多系列图表的标签是一个文本：{{var}}，标签位置在：图表区格式—可选文字—标题（新版本Microsoft Office标签位置在：编辑替换文字-替换文字）。



数据模型:

- `ChartMultiSeriesRenderData`

推荐使用工厂 `Charts` 构建图表模型。

### 代码示例

```
ChartMultiSeriesRenderData chart = Charts
        .ofMultiSeries("ChartTitle", new String[] { "中文", "English" })
        .addSeries("countries", new Double[] { 15.0, 6.0 })
        .addSeries("speakers", new Double[] { 223.0, 119.0 })
        .create();

put("barChart", chart);
```

JAVA

新的图表系列数据会完全替换原有图表数据，而原有图表的样式都会被保留。

## ChartMultiSeriesRenderData的结构体

```
{
  "chartTitle": "ChartTitle", ①
  "categories": [ ②
    "中文", "English"
  ],
  "seriesDatas": [ ③
    {
      "name": "countries", ④
      "values": [ ⑤
        15, 6
      ]
    },
    {
      "name": "speakers",
      "values": [
        223, 119
      ]
    }
  ]
}
```

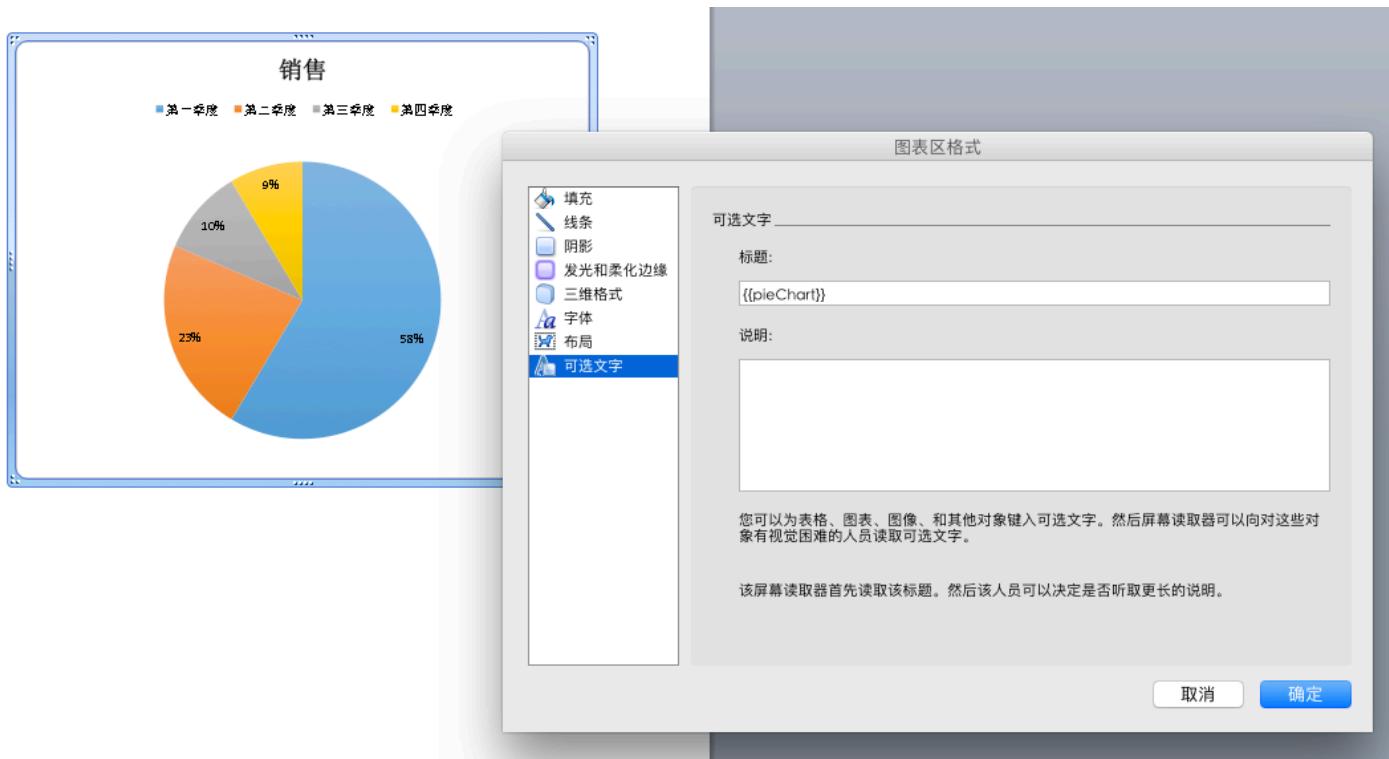
JSON

- ① 图表标题
- ② 种类
- ③ 所有系列
- ④ 当前系列名称
- ⑤ 当前系列对应每个种类的值

### 4.3. 单系列图表

单系列图表指的是饼图（3D饼图）、圆环图等。

单系列图表的标签是一个文本：{{var}}，标签位置在：图表区格式—可选文字—标题（新版本Microsoft Office标签位置在：编辑替换文字-替换文字）。



数据模型:

- `ChartSingleSeriesRenderData`

推荐使用工厂 `Charts` 构建图表模型。

### 代码示例

```
ChartSingleSeriesRenderData pie = Charts
    .ofSingleSeries("ChartTitle", new String[] { "美国", "中国" })
    .series("countries", new Integer[] { 9826675, 9596961 })
    .create();

put("pieChart", pie);
```

JAVA

### ChartSingleSeriesRenderData的结构体

```
{
  "chartTitle": "ChartTitle", ①
  "categories": [ ②
    "美国",
    "中国"
  ],
  "seriesData": { ③
    "name": "countries", ④
    "values": [ ⑤
      9826675,
      9596961
    ]
  }
}
```

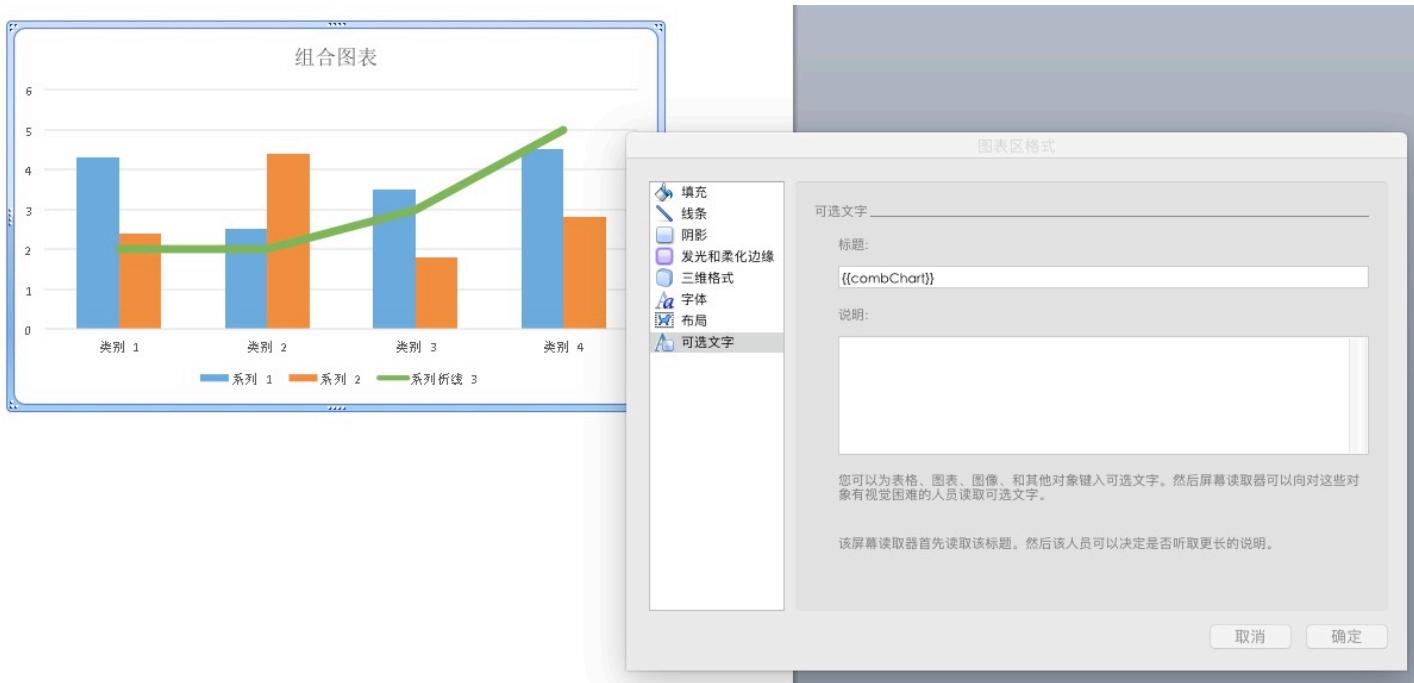
JSON

- ① 图表标题
- ② 种类
- ③ 单系列
- ④ 单系列名称
- ⑤ 单系列对应每个种类的值

#### 4.4. 组合图表

组合图表指的是由多系列图表（柱形图、折线图、面积图）组合而成的图表。

组合图表的标签是一个文本：{{var}}，标签位置在：图表区格式—可选文字—标题（新版本Microsoft Office标签位置在：编辑替换文字-替换文字）。



同多系列图表 `ChartMultiSeriesRenderData` 数据模型。

#### 代码示例

```
ChartSingleSeriesRenderData comb = Charts
    .ofComboSeries("MyChart", new String[] { "中文", "English" })
    .addBarSeries("countries", new Double[] { 15.0, 6.0 })
    .addBarSeries("speakers", new Double[] { 223.0, 119.0 })
    .addBarSeries("NewBar", new Double[] { 223.0, 119.0 })
    .addLineSeries("youngs", new Double[] { 323.0, 89.0 })
    .addLineSeries("NewLine", new Double[] { 123.0, 59.0 }).create();

put("combChart", comb);
```

JAVA

## ChartMultiSeriesRenderData的结构体

```
{  
    "chartTitle": "MyChart", ①  
    "categories": [ ②  
        "中文", "English"  
    ],  
    "seriesDatas": [ ③  
        {  
            "name": "countries", ④  
            "comboType": "BAR", ⑤  
            "values": [ ⑥  
                15, 6  
            ]  
        },  
        {  
            "name": "speakers",  
            "comboType": "LINE",  
            "values": [  
                223, 119  
            ]  
        }  
    ]  
}
```

JSON

- ① 图表标题
- ② 种类
- ③ 所有系列
- ④ 当前系列名称
- ⑤ 当前系列的图表类型comboType: 柱形图BAR、折线图LINE、面积图AREA
- ⑥ 当前系列对应每个种类的值

## 5. 配置

poi-tl提供了类 `Configure` 来配置常用的设置，使用方式如下：

```
ConfigureBuilder builder = Configure.builder();
XWPFTemplate.compile("template.docx", builder.build());
```

JAVA

### 5.1. 前后缀

我一直使用 `{}{}` 的方式来致敬Google CTemplate，如果你更偏爱freemarker `${}{}` 的方式：

```
builder.buildGramer("${", "}");
```

JAVA

### 5.2. 标签类型

默认的图片标签是以@开始，如果你希望使用%开始作为图片标签：

```
builder.addPlugin('%', new PictureRenderPolicy());
```

JAVA

如果你不是很喜欢默认的标签标识类型，你也可以自由更改：

```
builder.addPlugin('@', new TableRenderPolicy());
builder.addPlugin('#', new PictureRenderPolicy());
```

JAVA

这样`{@var}`就变成了表格标签，`{#var}`变成了图片标签，虽然不建议改变默认标签标识，但是从中可以看到poi-tl插件的灵活度，在插件章节中我们将会看到如何自定义自己的标签。

### 5.3. 标签格式

标签默认支持中文、字母、数字、下划线的组合，我们可以通过正则表达式来配置标签的规则，比如不允许中文：

```
builder.buildGrammerRegex("[\\w]+(\\.\\w+)*");
```

JAVA

比如允许除了标签前后缀外的任意字符：

```
builder.buildGrammerRegex(RegexUtils.createGeneral("{}{}"));
```

JAVA

### 5.4. Spring表达式

Spring Expression Language 是一个强大的表达式语言，支持在运行时查询和操作对象图，可作为独立组件使用，需要引入相应的依赖：

```
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-expression</artifactId>
<version>5.3.18</version>
</dependency>
```

XML

为了在模板标签中使用SpringEL表达式，需要将标签配置为SpringEL模式：

```
builder.useSpringEL();
```

JAVA

### 5.4.1. 基本使用

关于SpringEL的写法可以参见[官方文档](#)

(<https://docs.spring.io/spring-framework/docs/5.3.18/reference/html/core.html#expressions>)，下面给出一些典型的示例。

```
{{name}}
{{name.toUpperCase()}} ①
{{name == 'poi-tl'}} ②
{{empty?:'这个字段为空'}}
{{sex ? '男' : '女'}} ③
{{new java.text.SimpleDateFormat('yyyy-MM-dd HH:mm:ss').format(time)}} ④
{{price/10000 + '万元'}} ⑤
{{dogs[0].name}} ⑥
{{localDate.format(T(java.time.format.DateTimeFormatter).ofPattern('yyyy年MM月dd日'))}} ⑦
```

① 类方法调用，转大写

② 判断条件

③ 三目运算符

④ 类方法调用，时间格式化

⑤ 运算符

⑥ 数组列表使用下标访问

⑦ 使用静态类方法

### 5.4.2. SpringEL作为区块对的条件

Spring表达式与区块对结合可以实现更强大的功能，示例如下：

#### *data-model*

```
{
  "desc": "",
  "summary": "Find A Pet",
  "produces": [
    "application/xml"
  ]
}
```

JSON

#### *template.docx*

```
{?desc == null or desc == ""}{summary}{/}

{?produces == null or produces.size() == 0}无{/}
```

#### *output.docx*

## Find A Pet



使用SpringEL时区块对的结束标签可以是: {{/}}。

## 5.5. 错误处理

poi-tl支持在发生错误的时候定制引擎的行为。

### 5.5.1. 标签无法被计算

标签无法被计算的场景有几种，比如模板中引用了一个不存在的变量，或者级联的前置结果不是一个哈希，如  
{{author.name}} 中author的值为null，此时就无法计算name的值。

poi-tl可以在发生这种错误时对计算结果进行配置，默认会认为标签值为 null。当我们需要严格校验模板是否有人为失误时，可以抛出异常：

```
builder.useDefaultEL(true);
```

JAVA

注意的是，如果使用SpringEL表达式，可以通过参数来配置是否抛出异常：

```
builder.useSpringEL(true);
```

JAVA

### 5.5.2. 标签数据类型不合法

我们知道渲染图片、表格等标签时对数据模型是有要求的，如果数据不合法（为NULL或者是一个错误的数据类型），可以配置模板标签的渲染行为。

poi-tl默认的行为会清空标签，如果希望对标签不作任何处理：

```
builder.setValidErrorHandler(new DiscardHandler());
```

JAVA

如果希望执行严格的校验，直接抛出异常：

```
builder.setValidErrorHandler(new AbortHandler());
```

JAVA

## 5.6. 模板生成模板

模板引擎不仅仅可以生成文档，也可以生成新的模板，比如我们把原先的一个文本标签分成一个文本标签和一个表格标签：

```
Configure config = Configure.builder().bind("title", new DocumentRenderPolicy()).build();

Map<String, Object> data = new HashMap<>();

DocumentRenderData document = Documents.of()
    .addParagraph(Paragraphs.of("{{title}}").create())
    .addParagraph(Paragraphs.of("{{#table}}").create())
    .create();
data.put("title", document);
```

JAVA

## 5.7. 无模板创建文档

使用 `XWPFTemplate.create` 在无需模板的情况下创建文档，可以充分利用poi-tl友好的API来生成文档元素。

```
String text = "this a paragraph";
DocumentRenderData data = Documents.of().addParagraph(Paragraphs.of(text).create()).create();
XWPFTemplate template = XWPFTemplate.create(data);
```

JAVA

## 5.8. 日志

poi-tl使用slf4j作为日志门面，你可以自由选择日志实现，比如logback、log4j等。我们以logback为例：

首先在项目中添加logback依赖：

```
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-core</artifactId>
  <version>1.2.13</version>
</dependency>
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.2.13</version>
</dependency>
```

XML

然后配置logback.xml文件，可以配置日志级别和格式：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
    </encoder>
  </appender>

  <logger name="com.deepoove.poi" level="debug" additivity="false">
    <appender-ref ref="STDOUT" />
  </logger>
  <root level="info">
    <appender-ref ref="STDOUT" />
  </root>
</configuration>
```

XML

debug级别的日志会打印解析渲染过程中的信息，有利于程序调试，另外在模板引擎执行结束后会打印耗时信息：

Successfully Render the template file in 13 millis

## 6. 插件

插件，又称为自定义函数，它允许用户在模板标签位置处执行预先定义好的函数。由于插件机制的存在，我们几乎可以在模板的任何位置执行任何操作。

插件是poi-tl的核心，默认的标签和引用标签都是通过插件加载。

### 6.1. 默认插件

poi-tl默认提供了八个策略插件，用来处理文本、图片、列表、表格、文档嵌套、引用图片、引用多系列图表、引用单系列图表等：

- TextRenderPolicy
- PictureRenderPolicy
- NumberingRenderPolicy
- TableRenderPolicy
- DocxRenderPolicy
- MultiSeriesChartTemplateRenderPolicy
- SingleSeriesChartTemplateRenderPolicy
- DefaultPictureTemplateRenderPolicy

由于这八个插件如此通用，因此将这些插件注册为不同的标签类型，从而搭建了poi-tl的标签体系，也构筑了poi-tl高度自由的插件机制。

### 6.2. 开发一个插件

实现一个插件就是要告诉我们在模板的某个地方用某些数据做某些事情，我们可以通过实现 `RenderPolicy` 接口开发自己的插件：

```
public interface RenderPolicy {
    void render(ElementTemplate eleTemplate, Object data, XWPFTemplate template); ① ② ③
}
```

JAVA

- ① `ElementTemplate`是当前标签位置
- ② `data`是数据模型
- ③ `XWPFTemplate`代表整个模板

接下来我们写一个将标签替换为Hello, world的插件：

```
public class HelloWorldRenderPolicy implements RenderPolicy {  
  
    @Override  
    public void render(ElementTemplate eleTemplate, Object data, XWPFTemplate template) {  
        XWPFRun run = ((RunTemplate) eleTemplate).getRun(); ①  
        // String thing = String.valueOf(data);  
        String thing = "Hello, world";  
        run.setText(thing, 0); ②  
    }  
}
```

① XWPFRun是Apache POI的类，表示当前位置

② 渲染文本hello, world

poi-tl提供了抽象模板类 `AbstractRenderPolicy`，它定义了一些骨架步骤并且将数据模型的校验和渲染逻辑分开，使用泛型约束数据类型，让插件开发起来更简单，接下来我们再写一个更复杂的插件，在模板标签位置完完全全使用代码创建一个表格，这样我们就可以随心所欲的操作表格：

```
public class CustomTableRenderPolicy extends AbstractRenderPolicy<Object> {  
  
    @Override  
    protected void afterRender(RenderContext<Object> context) {  
        // 清空标签  
        clearPlaceholder(context, true);  
    }  
  
    @Override  
    public void doRender(RenderContext<Object> context) throws Exception {  
        XWPFRun run = context.getRun();  
        BodyContainer bodyContainer = BodyContainerFactory.getBodyContainer(run);  
        // 定义行列  
        int row = 10, col = 8;  
        // 插入表格  
        XWPFTable table = bodyContainer.insertNewTable(run, row, col);  
  
        // 表格宽度  
        TableTools.setWidth(table, UnitUtils.cm2Twips(14.63f) + "", null);  
        // 边框和样式  
        TableTools.borderTable(table, BorderStyle.DEFAULT);  
  
        // 1) 调用XWPFTable API操作表格  
        // 2) 调用TableRenderPolicy.Helper.renderRow方法快速方便的渲染一行数据  
        // 3) 调用TableTools类方法操作表格，比如合并单元格  
        // .....  
        TableTools.mergeCellsHorizontal(table, 0, 0, 7);  
        TableTools.mergeCellsVertically(table, 0, 1, 9);  
    }  
}
```

通过 `bodyContainer.insertNewTable` 在当前标签位置插入表格，使用XWPFTable API来操作表格。



随心所欲的意思是原则上Apache POI支持的操作，都可以在当前标签位置进行渲染，Apache POI不支持的操作也可以通过直接操纵底层XML来实现。

## 6.3. 使用插件

插件开发好后，为了让插件在某个标签处执行，我们需要将插件与标签绑定。

### 6.3.1. 将插件应用到标签

当我们有个模板标签为 `{{report}}`，默认是文本标签，如果希望在这个位置做些不一样或者更复杂的事情，我们可以将插件应用到这个模板标签：

```
ConfigureBuilder builder = Configure.builder();
builder.bind("report", new CustomTableRenderPolicy());
```

JAVA

此时，`{{report}}` 将不再是一个文本标签，而是一个自定义标签。

### 6.3.2. 将插件注册为新标签类型

当开发的插件具有一定的通用能力就可以将其注册为新的标签类型。比如增加%标识：`{{%var}}`，对应自定义的渲染策略 `HelloWorldRenderPolicy`：

```
builder.addPlugin('%', new HelloWorldRenderPolicy());
```

JAVA

此时，`{{%var}}` 将成为一种新的标签类型，它的执行函数是 `HelloWorldRenderPolicy`。

## 6.4. 插件列表

除了八个通用的策略插件外，还内置了一些非常有用的插件。

<code>ParagraphRenderPolicy</code>	渲染一个段落，可以包含不同样式文本，图片等	
<code>DocumentRenderPolicy</code>	渲染整个word文档	
<code>CommentRenderPolicy</code>	完整的批注功能	示例-批注
<code>AttachmentRenderPolicy</code>	插入附件功能	示例-插入附件
<code>LoopRowTableRenderPolicy</code>	循环表格行，下文会详细介绍	示例-表格行循环
<code>LoopColumnTableRenderPolicy</code>	循环表格列	示例-表格列循环
<code>DynamicTableRenderPolicy</code>	动态表格插件，允许直接操作表格对象	示例-动态表格
<code>BookmarkRenderPolicy</code>	书签和锚点	示例-Swagger文档
<code>AbstractChartTemplateRenderPolicy</code>	引用图表插件，允许直接操作图表对象	

TOCRenderPolicy	Beta实验功能：目录，打开文档时会提示更新域	
-----------------	-------------------------	--

同时有更多的独立插件可以使用（需要引入对应Maven依赖）：

HighlightRenderPolicy	Word支持代码高亮	示例-代码高亮
MarkdownRenderPolicy	使用Markdown来渲染word	示例-Markdown



如果你写了一个不错的插件，欢迎分享。

## 6.5. 表格行循环

`LoopRowTableRenderPolicy` 是一个特定场景的插件，根据集合数据循环表格行。

### *template.docx*

货物明细和人工费在同一个表格中，货物明细需要展示所有货物，人工费需要展示所有费用。`{{goods}}` 是个标准的标签，将 `{{goods}}` 置于循环行的上一行，循环行设置要循环的标签和内容，注意此时的标签应该使用 `[]`，以此来区别poi-tl的默认标签语法。同理，`{{labors}}` 也置于循环行的上一行。

货物明细						
{{goods}}数量	货物	说明	折扣	应纳税	单价	总计
[count]	名称: [name] 简介: [desc]	[@picture]	[discount]	[tax]	[price]	[count] *[price] =[totalPrice]
人工费						
{{labors}}种类				人数	单价	
[category]				[people]	[price]	[totalPrice]
{ {total} }						

### 代码示例

`{{goods}}` 和 `{{labors}}` 标签对应的数据分别是货物集合和人工费集合，如果集合为空则会删除循环行。

```

class Goods {
    private int count;
    private String name;
    private String desc;
    private int discount;
    private int tax;
    private int price;
    private int totalPrice;
    // getter setter
}

class Labor {
    private String category;
    private int people;
    private int price;
    private int totalPrice;
    // getter setter
}

List<Goods> goods = new ArrayList<>();
List<Labor> labors = new ArrayList<>();

```

接下来我们将插件应用到这两个标签。

```

LoopRowTableRenderPolicy policy = new LoopRowTableRenderPolicy();

Configure config = Configure.builder()
    .bind("goods", policy).bind("labors", policy).build(); ①

XWPFTemplate template = XWPFTemplate.compile(resource, config).render(
    new HashMap<String, Object>() {{
        put("goods", goods);
        put("labors", labors);
    }}
);

```

① 绑定插件

*output.docx*

最终生成的文档列出了所有货物和人工费。

货物明细						
数量	货物	说明	折扣	应纳税	单价	总计
4	名称: 墙纸 简介: 书房卧室		1500	27	400	4 *400 =1600
4	名称: 墙纸 简介: 书房卧室		1500	27	400	4 *400 =1600
4	名称: 墙纸 简介: 书房卧室		1500	27	400	4 *400 =1600
人工费						
种类			人数	单价		
<b>油漆工</b>			2	400	1600	
<b>油漆工</b>			2	400	1600	
<b>油漆工</b>			2	400	1600	
						1024

源码参见 [JUnit LoopRowTableRenderPolicyTest](#)

(<https://github.com/Sayi/poi-tl/tree/master/poi-tl/src/test/java/com/deepoove/poi/tl/plugin/HackLoopTableRenderPolicyTest.java>)

，如果希望模板标签和循环行在同一行而不是在上一行，可以使用 `new`

`LoopRowTableRenderPolicy(true)` 来构造插件。

## 6.6. 表格列循环

`LoopColumnTableRenderPolicy` 是一个特定场景的插件，根据集合数据循环表格列。要注意的是，由于文档宽度有限，因此模板列必须设置宽度，所有循环列将平分模板列的宽度。

*template.docx*

`LoopColumnTableRenderPolicy` 循环列的使用方式和插件 `LoopRowTableRenderPolicy` 是一样的，需要将占位标签放在循环列的前一列。

数量 <code>{goods}</code>	<code>[count]</code>
货物	<code>名称: [name]</code> <code>简介: [desc]</code>
说明	<code>[@picture]</code>
折扣	<code>[discount]</code>

代码示例

JAVA

```

LoopColumnTableRenderPolicy policy = new LoopColumnTableRenderPolicy();

Configure config = Configure.builder().bind("goods", policy).build();

XWPFTemplate template = XWPFTemplate.compile(resource, config).render(
    new HashMap<String, Object>() {{
        put("goods", goods);
    }}
);

```

*output.docx*

最终生成的文档列出了所有货物和人工费。

数量	4	4	4	4
货物	名称: 墙纸 简介: 书房卧室	名称: 墙纸 简介: 书房卧室	名称: 墙纸 简介: 书房卧室	名称: 墙纸 简介: 书房卧室
说明				
折扣	1500	1500	1500	1500

源码参见 [JUnit LoopColumnTableRenderPolicyTest](#)

(<https://github.com/Sayi/poi-tl/tree/master/poi-tl/src/test/java/com/deepoove/poi/tl/plugin/LoopColumnTableRenderPolicyTest.java>)

## 6.7. 动态表格

当需求中的表格更加复杂的时候，我们完全可以设计好那些固定的部分，将需要动态渲染的部分单元格交给自定义模板渲染策略。poi-tl提供了抽象表格策略 `DynamicTableRenderPolicy` 来实现这样的功能。

```

public abstract class DynamicTableRenderPolicy implements RenderPolicy {
    public abstract void render(XWPFTable table, Object data);
}

```

JAVA

*template.docx*

`{{detail_table}}`标签可以在表格内的任意单元格内，`DynamicTableRenderPolicy`会获取`XWPFTable`对象进而获得操作整个表格的能力。

{{detail_table}} 货物明细						
数量	货物	说明	折扣	应纳税	单价	总计
人工费						
种类			人数	单价		
{{total}}						

## 代码示例

首先新建渲染策略DetailTablePolicy，继承于抽象表格策略。

```
public class DetailTablePolicy extends DynamicTableRenderPolicy { JAVA

    // 货品填充数据所在行数
    int goodsStartRow = 2;
    // 人工费填充数据所在行数
    int laborsStartRow = 5;

    @Override
    public void render(XWPFTable table, Object data) throws Exception {
        if (null == data) return;
        DetailData detailData = (DetailData) data;

        // 人工费
        List<RowRenderData> labors = detailData.getLabors();
        if (null != labors) {
            table.removeRow(laborsStartRow);
            // 循环插入行
            for (int i = 0; i < labors.size(); i++) {
                XWPFTableRow insertNewTableRow = table.insertNewTableRow(laborsStartRow);
                for (int j = 0; j < 7; j++) insertNewTableRow.createCell();

                // 合并单元格
                TableTools.mergeCellsHorizontal(table, laborsStartRow, 0, 3);
                // 单行渲染
                TableRenderPolicy.Helper.renderRow(table.getRow(laborsStartRow), labors.get(i));
            }
        }

        // 货物
        List<RowRenderData> goods = detailData.getGoods();
        if (null != goods) {
            table.removeRow(goodsStartRow);
            for (int i = 0; i < goods.size(); i++) {
                XWPFTableRow insertNewTableRow = table.insertNewTableRow(goodsStartRow);
                for (int j = 0; j < 7; j++) insertNewTableRow.createCell();
                TableRenderPolicy.Helper.renderRow(table.getRow(goodsStartRow), goods.get(i));
            }
        }
    }
}
```

然后将模板标签{{detail\_table}}设置成此策略。

```
Configure config = Configure.builder().bind("detail_table", new
DetailTablePolicy()).build();
```

### output.docx

最终生成的文档列出了所有货物和人工费。

货物明细						
数量	货物	说明	折扣	应纳税	单价	总计
4	墙纸	书房+卧室	1500	/	400	1600
4	墙纸	书房+卧室	1500	/	400	1600
4	墙纸	书房+卧室	1500	/	400	1600
人工费						
种类			人数	单价		
油漆工			2	200	400	
油漆工			2	200	400	
油漆工			2	200	400	
油漆工			2	200	400	
						总共: 7200

源码参见 [JUnit PaymentExample](#)

(<https://github.com/Sayi/poi-tl/tree/master/poi-tl/src/test/java/com/deeppooove/poi/tl/example/PaymentExample.java>)

## 6.8. 批注

CommentRenderPolicy 是内置插件，提供了对批注完整功能的支持。

数据模型：

- CommentRenderData

### 代码示例

```
CommentRenderData comment = Comments.of("鹅")
    .signature("Sayi", "s", LocaleUtil.getLocaleCalendar())
    .comment("鹅，是一种动物")
    .create(); ①
Map<String, Object> data = new HashMap<>();
data.put("comment", comment);
Configure config = Configure.builder().bind("comment", new CommentRenderPolicy()).build();
②

XWPFTemplate.compile("comment_template.docx", config).render(data);
```

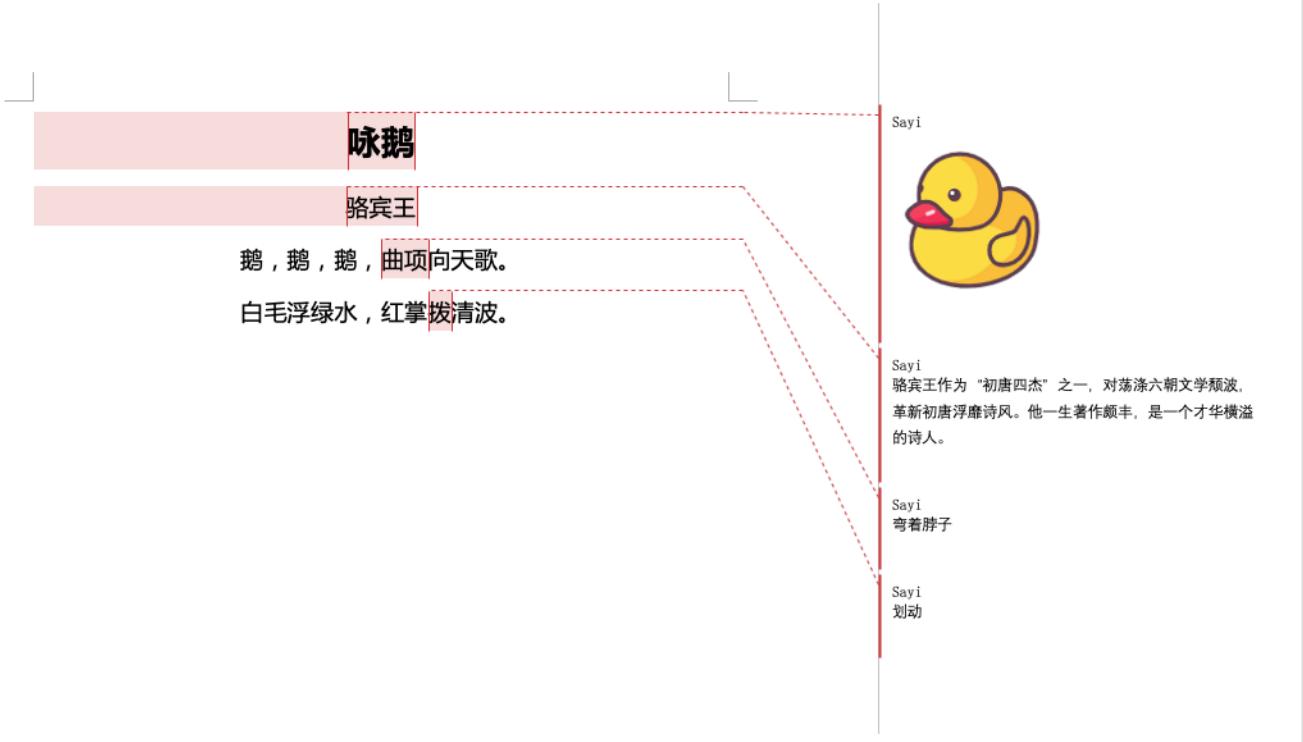
① 批注内容

② 将批注插件和comment标签绑定

### 6.8.1. 示例

*output.docx*

批注中支持添加文字、图片等文档内容。



源码参见 [JUnit CommentRenderPolicyTest](#)

(<https://github.com/Sayi/poi-tl/tree/master/poi-tl/src/test/java/com/deeboove/poi/tl/plugin/CommentRenderPolicyTest.java>)

。

### 6.9. 插入附件

`AttachmentRenderPolicy` 是内置插件，提供了插入附件功能的支持。

数据模型：

- `AttachmentRenderData`

#### 代码示例

JAVA

```
AttachmentRenderData attach = Attachments.ofLocal("attachment.xlsx",
AttachmentType.XLSX).create(); ①

Map<String, Object> data = new HashMap<>();
data.put("attachment", attach);
Configure config = Configure.builder().bind("attachment", new
AttachmentRenderPolicy()).build(); ②

XWPFTemplate.compile("attachment_template.docx", config).render(data);
```

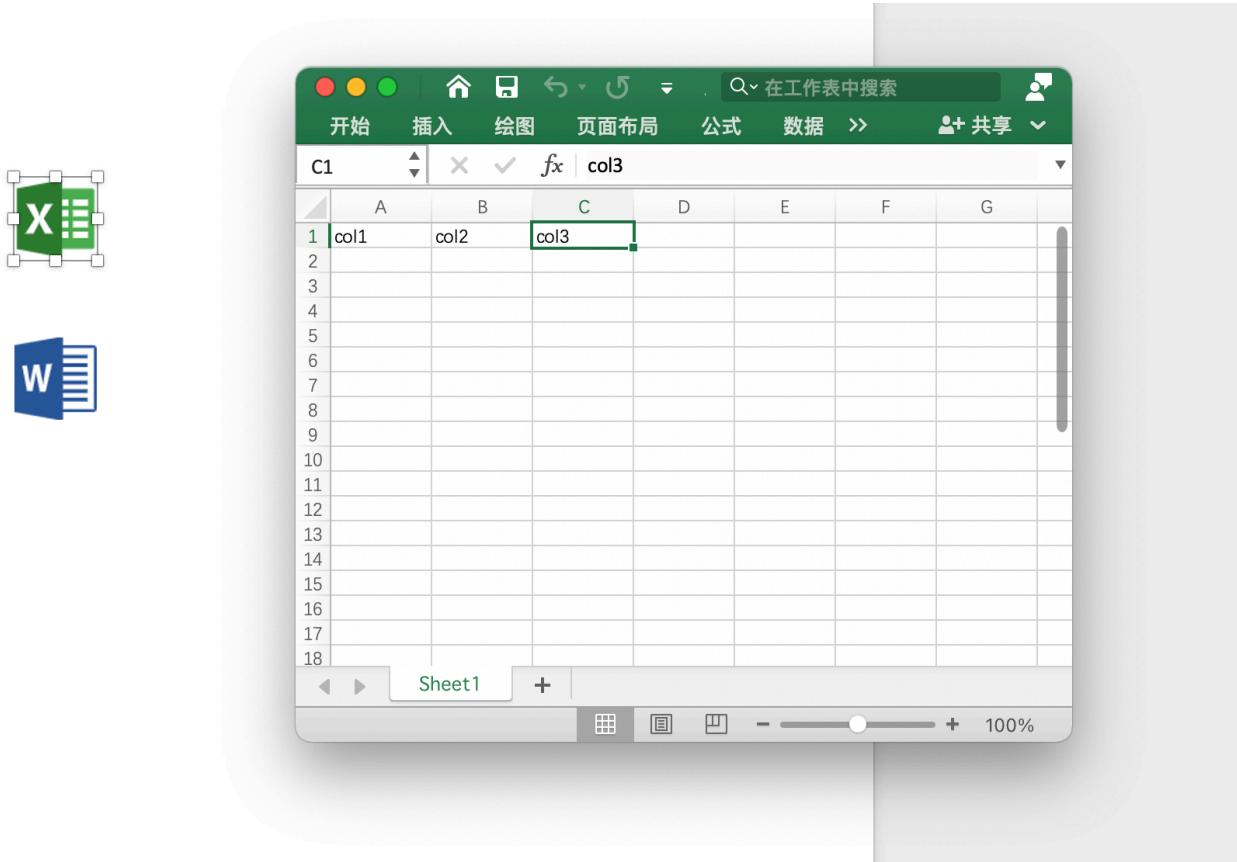
- ① 附件文档，Word或者Excel

## ② 绑定标签和附件插件

### 6.9.1. 示例

*output.docx*

文档中插入Excel，双击图标打开附件。



源码参见 [JUnit AttachmentRenderPolicyTest](#)

(<https://github.com/Sayi/poi-tl/tree/master/poi-tl/src/test/java/com/deeppoove/poi/tl/policy/AttachmentRenderTest.java>)

。

## 6.10. 代码高亮

`HighlightRenderPolicy` 插件对Word代码块进行高亮展示。

### 6.10.1. 引入依赖:

```
<dependency>
<groupId>com.deeppoove</groupId>
<artifactId>poi-tl-plugin-highlight</artifactId>
<version>1.0.0</version>
</dependency>
```

XML

### 6.10.2. 快速开始

数据模型:

- [HighlightRenderData](#)

## 代码示例

```
HighlightRenderData code = new HighlightRenderData();
code.setCode("/**\n"
+ " * @author John Smith <john.smith@example.com>\n"
+ "*/\n"
+ "package l2f.gameserver.model;\n"
+ "\n"
+ "public abstract strictfp class L2Char extends L2Object {\n"
+ "    public static final Short ERROR = 0x0001;\n"
+ "\n"
+ "    public void moveTo(int x, int y, int z) {\n"
+ "        _ai = null;\n"
+ "        log(\"Should not be called\");\n"
+ "        if (1 > 5) { // wtf!?\n"
+ "            return;\n"
+ "        }\n"
+ "    }\n"
+ "}");
code.setLanguage("java"); ①
code.setStyle(HighlightStyle.builder().withShowLine(true).withTheme("zenburn").build()); ②
Map<String, Object> data = new HashMap<>();
data.put("code", code);

Configure config = Configure.builder().bind("code", new HighlightRenderPolicy()).build();
③
XWPFTemplate.compile("highlight_template.docx", config).render(data);
```

- ① 代码语言
- ② 设置主题样式
- ③ 将代码高亮插件和code标签绑定

### 6.10.3. 示例

#### *output.docx*

示例展示了代码高亮插件支持20多种编程语言和几十种主题样式。

**Default style with background**

```

1 /**
2 * @author John Smith <john.smith@example.com>
3 */
4 package l2f.gameserver.model;
5
6 public abstract strictfp class L2Char extends L2Object {
7     public static final Short ERROR = 0x0001;
8
9     public void moveTo(int x, int y, int z) {
10         _ai = null;
11         log("Should not be called");
12         if (z > 0) { // wtf?
13             return;
14         }
15     }
16 }
```

**github**

```

1 /**
2 * @author John Smith <john.smith@example.com>
3 */
4 package l2f.gameserver.model;
5
6 public abstract strictfp class L2Char extends L2Object {
7     public static final Short ERROR = 0x0001;
8
9     public void moveTo(int x, int y, int z) {
10         _ai = null;
11         log("Should not be called");
12         if (z > 5) { // wtf?
13             return;
14         }
15     }
16 }
```

**1. idea**

```

1 /**
2 * @author John Smith <john.smith@example.com>
3 */
4 package l2f.gameserver.model;
```

```

5
6 public abstract strictfp class L2Char extends L2Object {
7     public static final Short ERROR = 0x0001;
8
9     public void moveTo(int x, int y, int z) {
10         _ai = null;
11         log("Should not be called");
12         if (z > 5) { // wtf?
13             return;
14         }
15     }
16 }
```

**2. zenburn**

```

1 /**
2 * @author John Smith <john.smith@example.com>
3 */
4 package l2f.gameserver.model;
5
6 public abstract strictfp class L2Char extends L2Object {
7     public static final Short ERROR = 0x0001;
8
9     public void moveTo(int x, int y, int z) {
10         _ai = null;
11         log("Should not be called");
12         if (z > 5) { // wtf?
13             return;
14         }
15     }
16 }
```

**3. androidstudio**

```

1 /**
2 * @author John Smith <john.smith@example.com>
3 */
4 package l2f.gameserver.model;
5
6 public abstract strictfp class L2Char extends L2Object {
7     public static final Short ERROR = 0x0001;
8
9     public void moveTo(int x, int y, int z) {
10         _ai = null;
11         log("Should not be called");
12     }
13 }
```

源码参见 [JUnit HighlightRenderPolicyTest](#)



(<https://github.com/Sayi/poi-tl/tree/master/poi-tl-plugin-highlight/src/test/java/com/deeppooove/poi/plugin/highlight/HighlightRenderPolicyTest.java>)

o

#### 6.10.4. 常用语言支持

- apache
- bash
- cpp
- cs
- css
- diff
- go
- groovy
- http
- ini
- java
- javascript
- json
- makefile

- markdown
- objectivec
- perl
- php
- python
- ruby
- scala
- shell
- sql
- xml
- yaml

#### 6.10.5. 常用主题样式

- github
- idea
- zenburn
- androidstudio
- solarized- light
- solarized- dark
- xcode
- vs
- agate
- darcula
- dark
- dracula
- foundation
- googlecode
- monokai
- mono- blue
- far
- gml

### 6.11. Markdown

`MarkdownRenderPolicy` 插件支持通过Markdown生成word文档。

#### 6.11.1. 引入依赖:

```
<dependency>
    <groupId>com.deepoove</groupId>
    <artifactId>poi-tl-plugin-markdown</artifactId>
    <version>1.0.3</version>
</dependency>
```

## 6.11.2. 快速开始

数据模型：

- MarkdownRenderData

### 代码示例

JAVA

```
MarkdownRenderData code = new MarkdownRenderData();
code.setMarkdown(new String(Files.readAllBytes(Paths.get("README.md"))));
code.setStyle(MarkdownStyle.newStyle()); ①

Map<String, Object> data = new HashMap<>();
data.put("md", code);

Configure config = Configure.builder().bind("md", new MarkdownRenderPolicy()).build(); ②
XWPFTemplate.compile("markdown_template.docx", config).render(data);
```

① 定制markdown转为word的样式

② 将Markdown插件和md标签绑定

## 6.11.3. 示例

### *output.docx*

通过Markdown插件将poi-tl根目录下的README.md内容转为word文档的结果示例：[markdown.docx](#)

1

## poi-tl(po-template-language)

[build passing](#) [jdk 1.6+](#) [jdk 1.8](#) [apache-poi 3.16+](#) [apache-poi 4.0.0](#)

[chat on gitter](#)

A better way to generate word(docx) with template , based on Apache POI - the Java API for Microsoft Documents.

### 1. What is poi-tl

FreeMarker or Velocity generates new html pages or configuration files based on text template and data. poi-tl is a Word template engine that generates new documents based on Word template and data.

The Word template has rich styles. Poi-tl will perfectly retain the styles in the template in the generated documents. You can also set styles for the tags. The styles of the tags will be applied to the replaced text, so you can focus on the template design.

poi-tl is a "logic-less" template engine. There is no complicated control structure and variable assignment, only tags, some tags can be replaced with text, pictures, tables, etc., some tags will hide certain some document content, while other tags will loop a series of document content.

"Powerful" constructs like variable assignment or conditional

2

statements make it easy to modify the look of an application within the template system exclusively... however, at the cost of separation, turning the templates themselves into part of the application logic.

[«Google CTemplate»](#)

poi-tl supports custom functions (plug-ins), functions can be executed anywhere in the Word template, do anything anywhere in the document is the goal of poi-tl.

### 2. Maven

```
<dependency>
<groupId>com.deepoove</groupId>
<artifactId>poi-tl</artifactId>
<version>1.10.0-beta</version>
</dependency>
```

### 3. Quick start

Start with a deadly simple example: replace `{{title}}` with "poi-tl template engine".

- 1) Create a new document `template.docx`, including the content `{{title}}`
- 2) TDO mode: Template + data-model = output

```
//The core API uses a minimalist design, only one line of code is required
XWPFTemplate.compile("template.docx").render(new HashMap<String, Object>(){
    put("title", "poi-tl template engine");
}).writeToFile("out_template.docx");
```

Open the `out_template.docx` document, everything is as you wish.

源码参见 [JUnit MarkdownRenderPolicyTest](#)

(<https://github.com/Sayi/poi-tl/tree/master/poi-tl-plugin-markdown/src/test/java/com/deopoove/poi/plugin/markdown/MarkdownRenderPolicyTest.java>)



## 7. 示例

接下来的示例采取三段式output+template+data-model来说明，首先直接展示生成后的文档，然后一览模板的样子，最后我们对数据模型作个介绍。

### 7.1. 软件说明文档

#### output.docx

需要生成这样的一份软件说明书：拥有封面和页眉，正文含有不同样式的文本，还有表格，列表和图片。[poi-tl.docx](#)

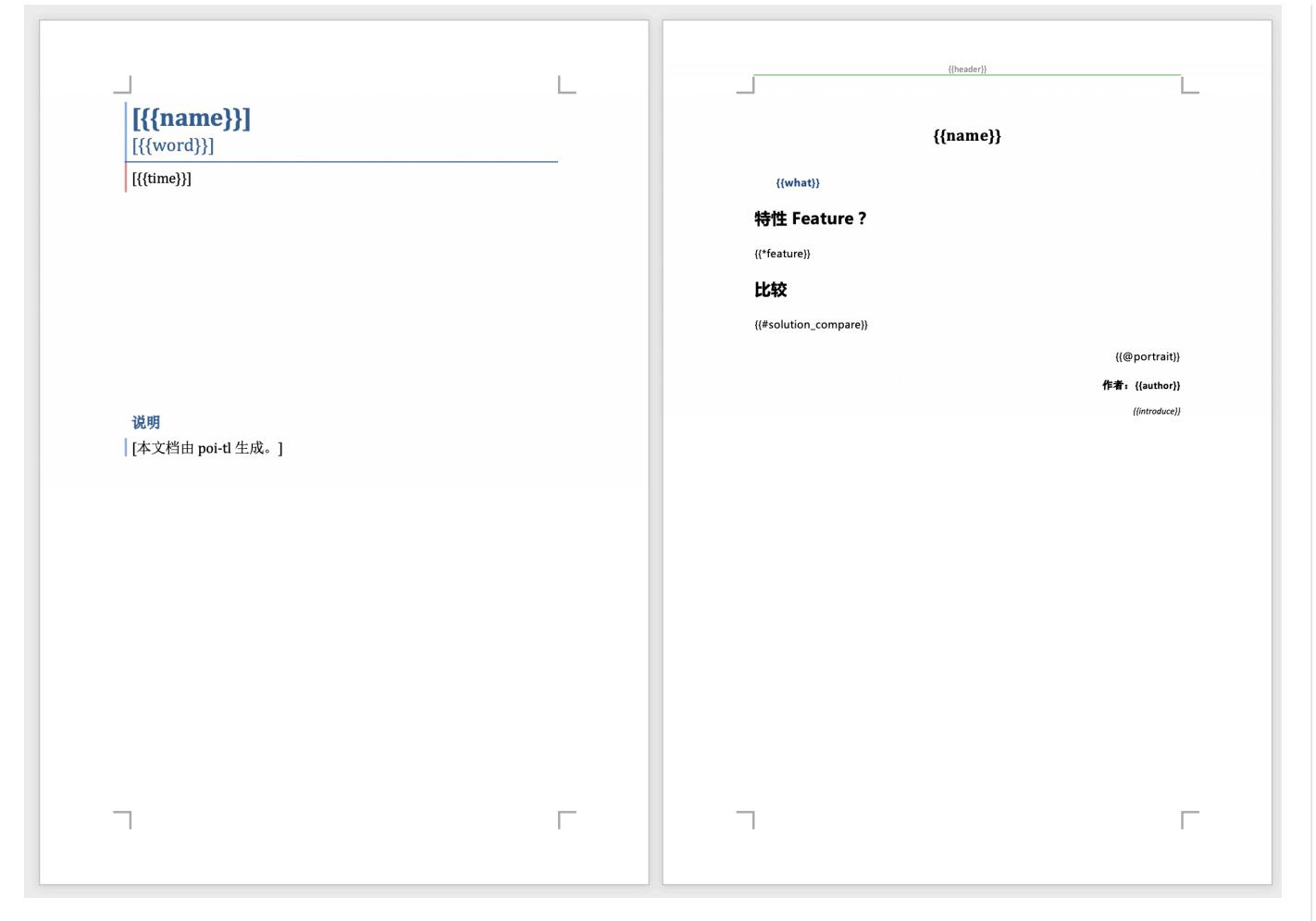
The screenshot shows two pages of a Microsoft Word document. The first page is a cover page with a blue header bar containing the text '[Poi-tl]' and '[模板引擎]'. Below the header is the date '[2020-08-31]'. The second page is the main content page. It features a header 'Deeply love what you love.' and a sub-header 'Poi-tl'. Below this is a descriptive text block about Java Word templating. Following the text are two sections: '特性 Feature ?' and '比较'. The '特性' section contains a bulleted list of features. The '比较' section is a table comparing Poi-tl with Apache Poi, Freemarker, OpenOffice, and Jacob. The table has three columns: 'Word 处理方案', '是否跨平台', and '易用性'. The table data is as follows:

Word 处理方案	是否跨平台	易用性
Poi-tl	纯 Java 组件，跨平台	简单；模板引擎功能，并对 POI 进行了一些封装
Apache Poi	纯 Java 组件，跨平台	简单，缺少一些功能的封装
Freemarker	XML 操作，跨平台	复杂，需要理解 XML 语法
OpenOffice	需要安装 OpenOffice 软件	复杂，需要了解 OpenOffice 的 API
Jacob、winlib	Windows 平台	复杂，不推荐使用

At the bottom right of the content page, there is a small green square icon with the number '7' and the text '作者: Sayi 帅一' followed by a URL 'http://www.deepoove.com'.

#### template.docx

使用poi-tl标签制作模板，可以看到标签可以拥有样式。



这个示例向我们展示了poi-tl最基本的能力，它在模板标签位置，插入基本的数据模型，所见即所得。



源码参见 [JUnit XWPFTemplateTest](#)

(<https://github.com/Sayi/poi-tl/tree/master/poi-tl/src/test/java/com/deeppoove/poi/tl/XWPFTemplateTest.java>)

## 7.2. 付款通知书

*output.docx*

需要生成这样的一份流行的通知书：大部分数据是由表格构成的，需要创建一个订单的表格(图中第一个表格)，还需要在一个已有表格中，填充货物明细和人工费数据(图中第二个表格)。下载最终生成的文件[payment.docx](#)

本文档由 POI-TL 生成

## 付款通知书

付款通知书编号: KB.6890451

客户 ID: ZHANG\_SAN\_091

抬头:

深圳 XX 家装有限公司

收货方:

丙丁

日期	订单编号	销售代表	离岸价	发货方式	条款	税号
2018-06-12	SN18090	李四	5000 元	快递	附录 A	T11090

货物明细						
数量	货物	说明	折扣	应纳税	单价	总计
4	墙纸	书房 + 卧室	1500	/	400	1600
4	墙纸	书房 + 卧室	1500	/	400	1600
4	墙纸	书房 + 卧室	1500	/	400	1600

人工费						
种类			人数	单价		
油漆工			2	200	400	
油漆工			2	200	400	
油漆工			2	200	400	
油漆工			2	200	400	
						总共: 7200

小计:	8000
税额:	600
运输:	120
其他:	250
未付余额:	6600

### template.docx

使用{{#order}}生成poi-tl提供的默认样式的表格，设置{{detail\_table}}为自定义模板渲染策略(继承抽象表格策略DynamicTableRenderPolicy)，自定义已有表格中部分单元格的渲染。

本文档由 POI-TL 生成

## 付款通知书

付款通知书编号: {{NO}}      抬头: {{taitou}}      收货方: {{consignee}}

客户 ID: {{ID}}

{{#order}}

{{detail_table}} 货物明细						
数量	货物	说明	折扣	应纳税	单价	总计

人工费

种类	人数	单价	

{{total}}

小计:	{{subtotal}}
税额:	{{tax}}
运输:	{{transform}}
其他:	{{other}}
未付余额:	{{unpay}}

这个示例向我们展示了poi-tl在表格操作上的一些思考。示例中货物明细和人工费的表格就是一个相当复杂的表格，货物明细是由7列组成，行数不定，人工费是由4列组成，行数不定。

这个示例主要用来展示DynamicTableRenderPolicy的用法，货物明细和人工费仅仅是循环渲染表格行，使用LoopRowTableRenderPolicy 插件会更方便。



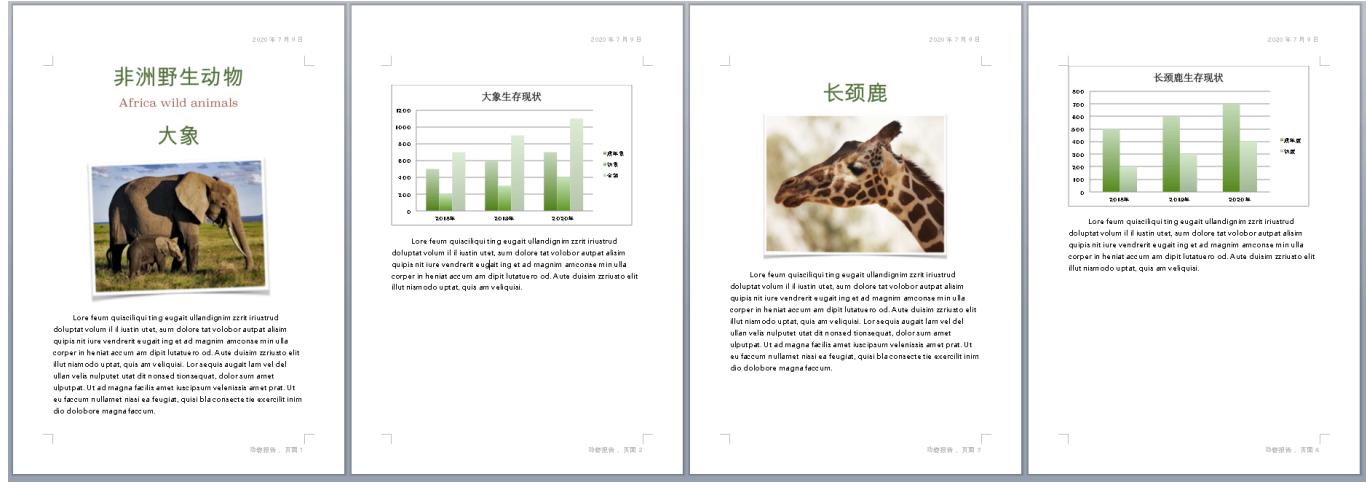
源码参见 [JUnit PaymentExample](#)

(<https://github.com/Sayi/poi-tl/tree/master/poi-tl/src/test/java/com/deeboove/poi/tl/example/PaymentExample.java>)

## 7.3. 野生动物现状

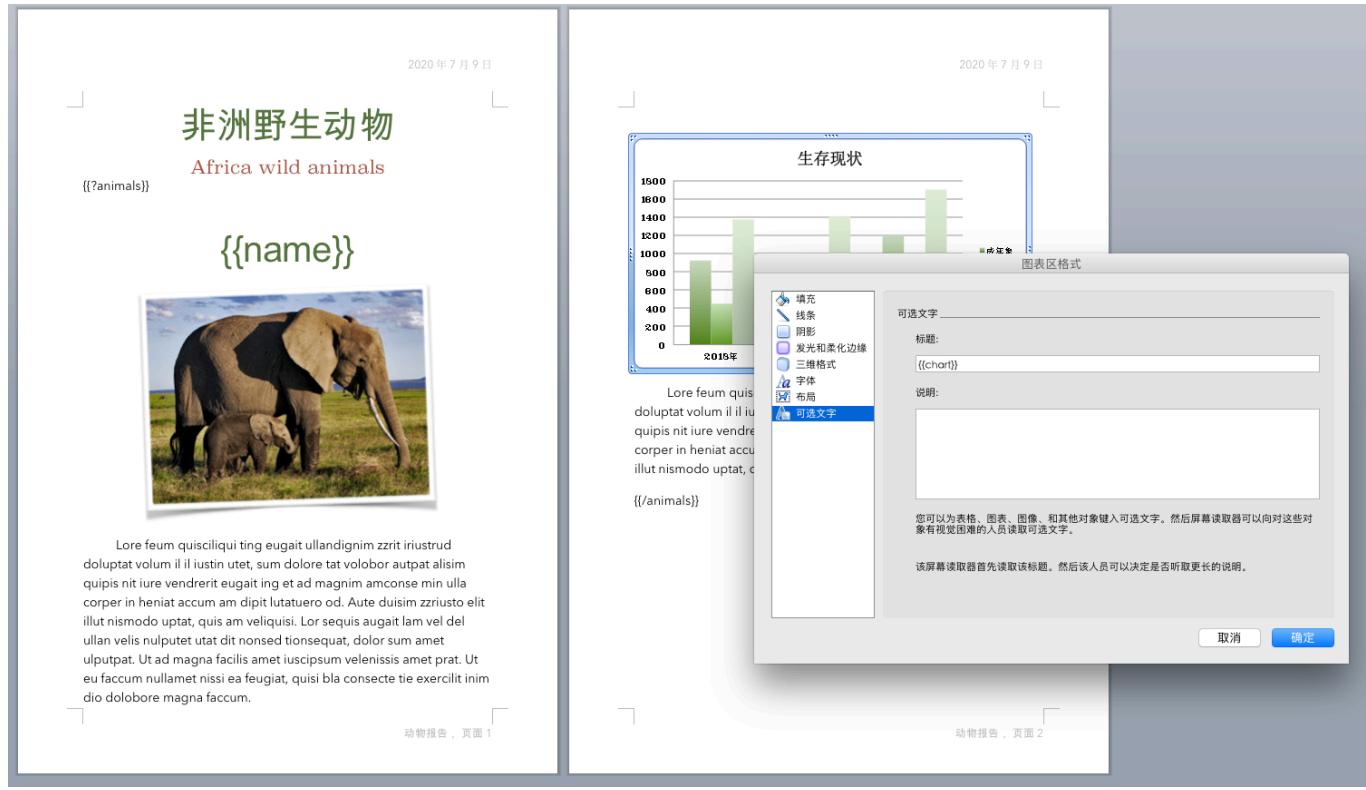
### *output.docx*

针对野生动物出具一份现状的调查报告，野生动物种类不确定，调查报告包含图片、文字和图表。下载最终生成的文件 [animal.docx](#)



### *template.docx*

不确定动物种类使用区块对{{?animals}}的循环功能实现，图片和图表如模板所示，使用引用标签，在可选文字标题位置输入标签。



这个示例展示了区块对的循环功能，以及如何在循环中使用引用图片和引用图表的功能。



源码参见 [JUnit AnimalExample](#)  
(<https://github.com/Sayi/poi-tl/tree/master/poi-tl/src/test/java/com/deepoove/poi/tl/example/AnimalExample.java>)

## 7.4. 证书奖状

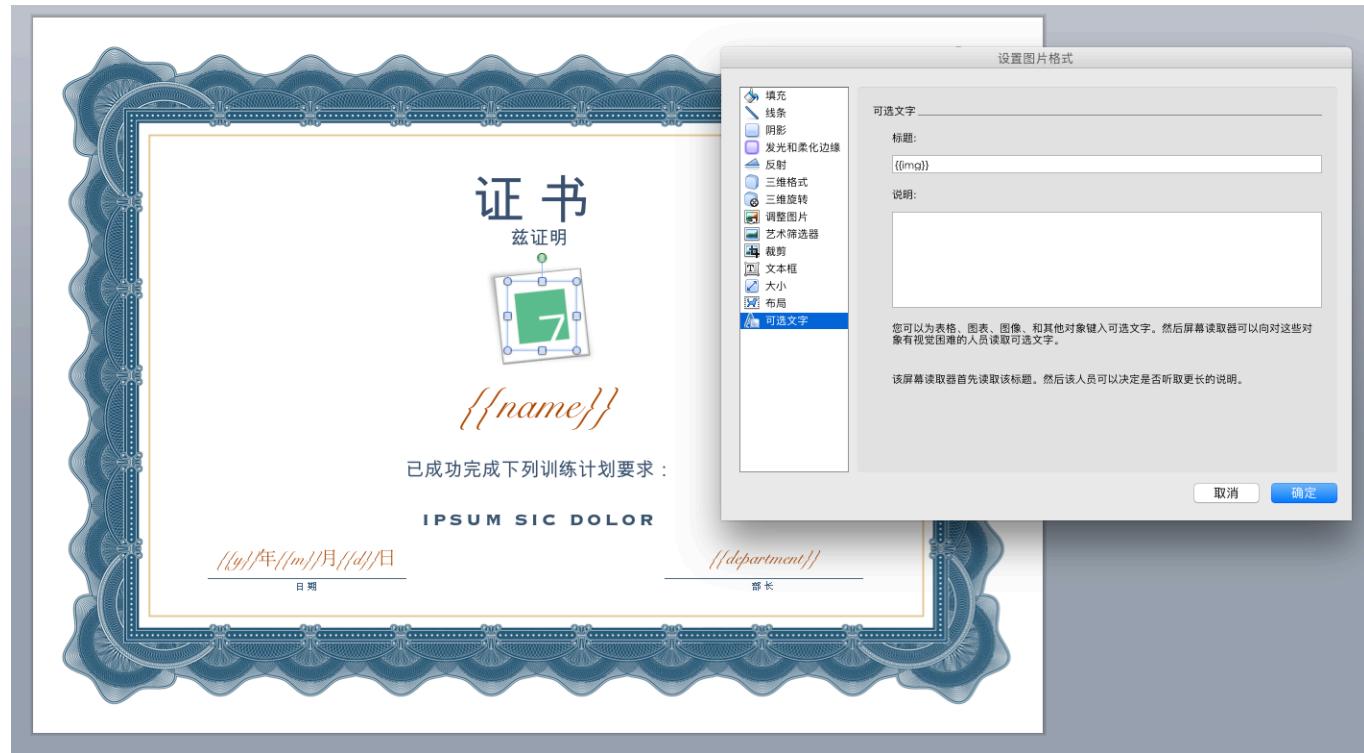
*output.docx*

颁发一张由特殊样式图片、姓名、日期构成的证书奖状。下载最终生成的文件[certificate.docx](#)



*template.docx*

图片格式和布局由模板指定，图片使用引用标签替换即可。



这个示例展示了引用图片和文本框的功能。



源码参见 [JUnit CertificateExample](#)  
[\(https://github.com/Sayi/poi-tl/tree/master/poi-tl/src/test/java/com/deeboove/poi/tl/example/CertificateExample.java\)](https://github.com/Sayi/poi-tl/tree/master/poi-tl/src/test/java/com/deeboove/poi/tl/example/CertificateExample.java)

## 7.5. 个人简历

### *output.docx*

需要生成这样的一份个人简历：左侧是个人的基本信息，技术栈是个典型的列表，右侧是个人的工作经历，数量不定。

下载最终生成的文件[resume.docx](#)

7  
卅一  
BUG 工程师

### 个人信息

电话: 18080809090 性别: 男  
籍贯: 杭州 生日: 2000.08.19  
邮箱: adasai90@gmail.com  
地址: 浙江省杭州市西湖区古荡 1 号  
英语: CET6 620

### 教育背景

美国斯坦福大学  
生命工程 学士 班级排名 1/36

### 技术栈

- SpringBoot、SpringCloud、Mybatis
- SpringBoot、SpringCloud、Mybatis
- SpringBoot、SpringCloud、Mybatis

### 兴趣爱好

音乐、画画、乒乓球、旅游、读书  
<https://github.com/Sayi>

### 工作经历

香港微软 Office 事业部 2001.07-2020.06  
BUG 工程师

- 负责生产 BUG, 然后修复 BUG, 同时有效实施招聘行为
- 负责生产 BUG, 然后修复 BUG, 同时有效实施招聘行为

自由职业 OpenSource 项目组 2015.07-2020.06  
研发工程师

- 开源项目的迭代和维护工作
- 持续集成、Swagger 文档等工具调研
- 开源项目的迭代和维护工作

香港微软 Office 事业部 2001.07-2020.06  
BUG 工程师

- 负责生产 BUG, 然后修复 BUG, 同时有效实施招聘行为
- 负责生产 BUG, 然后修复 BUG, 同时有效实施招聘行为

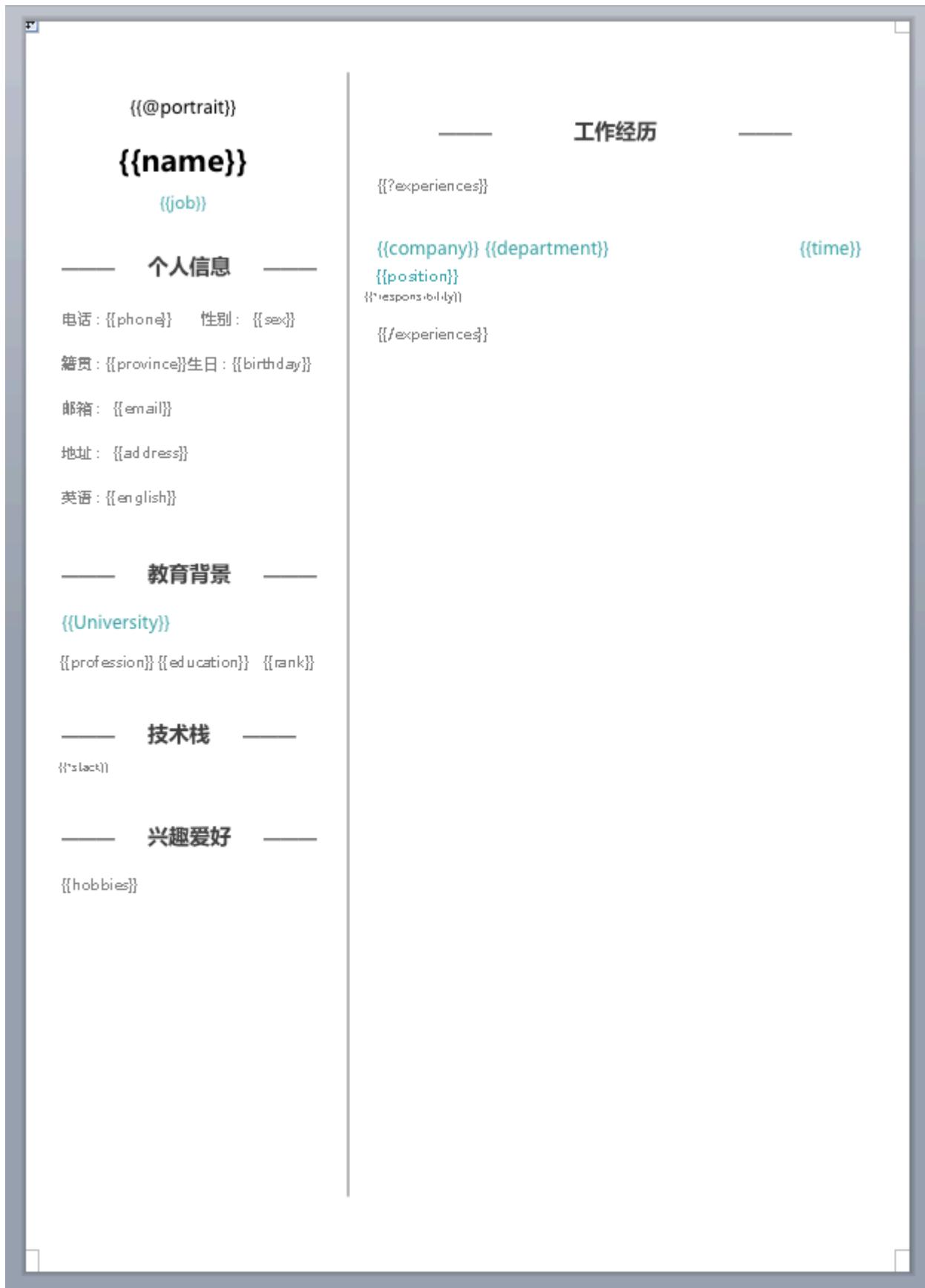
自由职业 OpenSource 项目组 2015.07-2020.06  
研发工程师

- 开源项目的迭代和维护工作
- 持续集成、Swagger 文档等工具调研
- 开源项目的迭代和维护工作

#### 7.5.1. 方案一：使用区块对标签

*template.docx*

工作经历是一个循环显示的内容，我们使用区块对标签{{?experiences}}{{/experiences}}。

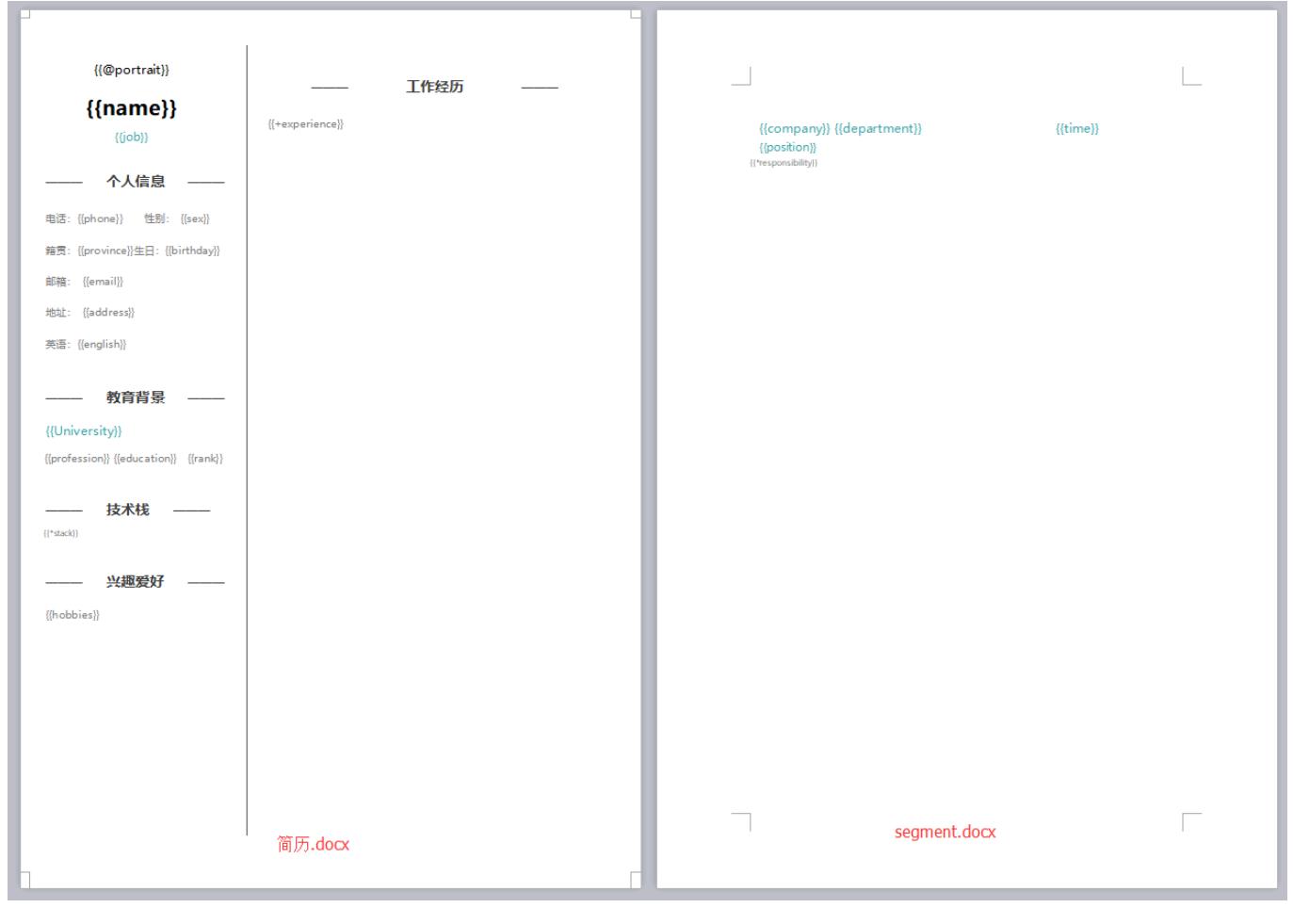


**i** 源码参见 [JUnit Iterable ResumeExample](#)  
(<https://github.com/Sayi/poi-tl/blob/master/poi-tl/src/test/java/com/deepoove/poi/tl/example/ResumeIterableExample.java>)

## 7.5.2. 方案二：使用嵌套标签

### *template.docx*

工作经历可以使用嵌套标签，我们制作两个模板，一套主模板简历.docx(下图左侧)，一套为文档模板segment.docx(下图右侧)。



看起来很复杂的简历，其实对于模版引擎来说，和普通的Word文档没有什么区别，我们只需要制作好一份简历，将需要替换的内容用模版标签代替。

因为模版即样式，模版引擎无需考虑样式，只关心数据，我们甚至可以制作10种不同样式的简历模板，用同一份数据去渲染。



源码参见 [JUnit ResumeExample](#)  
[\(https://github.com/Sayi/poi-tl/tree/master/poi-tl/src/test/java/com/deeppoove/poi/tl/example/ResumeExample.java\)](https://github.com/Sayi/poi-tl/tree/master/poi-tl/src/test/java/com/deeppoove/poi/tl/example/ResumeExample.java)

## 7.6. Swagger文档

### *output.docx*

这是一份非常专业的Swagger Word文档，样式优雅且有着清晰完整的文档结构，API列表需要循环展示，接口的请求参数需要循环展示，接口的返回值需要循环展示，数据类型支持锚点到具体的模型，模型支持代码块高亮展示。下载最终生成的文件[swagger.docx](#)

The screenshot shows the Swagger Petstore API documentation. On the left is a sidebar with a tree view of the API structure:

- Swagger Petstore
  - 1. 概览
  - 2. 接口列表
  - 3. 数据模型

The main content area displays the Petstore API documentation. It includes sections for:
 

- 1. 概览**: A brief introduction to the sample server.
- 1.1. 描述**: A detailed description of the API.
- 1.2. 版本**: The API version.
- 1.3. 联系人**: Contact information for the API team.
- 1.4. 标签**: Tags for the API.
- 2.1. pet**: An endpoint for finding a pet by ID.
- 2.1.1. Find pet by ID**: Detailed description of the find pet by ID operation.
- 2.1.2. Updates a pet in the store with form data**: Detailed description of the update pet operation.
- 2.1.3. Deletes a pet**: Detailed description of the delete pet operation.
- 2.1.4. uploads an image**: Detailed description of the upload file operation.
- 2.1.5. Updata an existing pet**: Detailed description of the update an existing pet operation.
- 2.1.6. Add a new pet to the store**: Detailed description of the add a new pet to the store operation.
- 2.1.7. Finds Pets by status**: Detailed description of the finds pets by status operation.
- 2.1.8. Finds Pets by tags**: Detailed description of the finds pets by tags operation.
- 2.2. store**: An endpoint for returning pet inventories by status.
- 2.2.1. Returns pet inventories by status**: Detailed description of the returns pet inventories by status operation.
- 2.2.2. Find purchase order by ID**: Detailed description of the finds purchase order by ID operation.
- 2.2.3. Delete purchase order by ID**: Detailed description of the deletes purchase order by ID operation.
- 2.2.4. Place an order for a pet**: Detailed description of the places an order for a pet operation.
- 2.3. user**: An endpoint for getting user information.
- 2.3.1. Get user by user name**: Detailed description of the gets user by user name operation.
- 2.3.2. Updated user**: Detailed description of the updated user operation.
- 2.3.3. Delete user**: Detailed description of the deletes user operation.
- 2.3.4. Logs user into the system**: Detailed description of the logs user into the system operation.
- 2.3.5. Logs out current logged in user session**: Detailed description of the logs out current logged in user session operation.
- 2.3.6. Create an event**: Detailed description of the creates an event operation.
- 2.3.7. Creates list of users with given input array**: Detailed description of the creates list of users with given input array operation.
- 2.3.8. Creates list of users with given input array**: Detailed description of the creates list of users with given input array operation.
- 3. 数据模型**: A section for data models.
- 3.1. Category**: A model for categories.
- 3.2. Pet**: A model for pets.
- 3.3. Tag**: A model for tags.
- 3.4. ApiResponse**: A model for API responses.
- 3.5. Order**: A model for orders.
- 3.6. User**: A model for users.
- 4. 关于文档**: A section about the documentation.

The screenshot shows the Swagger Petstore API documentation. On the left is a sidebar with a tree view of the API structure:

- Swagger Petstore
  - 3. 数据模型

The main content area displays the Petstore API documentation. It includes sections for:
 

- 3.1. CATEGORY**: A model for categories.
- 3.1.1. 属性**: Properties of the category model.

属性名	描述	类型
<b>id</b>		integer
<b>name</b>		string

- 3.1.2. 示例**: Examples of category objects.

```
{
  "id": 0,
  "name": "string"
}
```

- 3.2. PET**: A model for pets.
- 3.2.1. 属性**: Properties of the pet model.

属性名	描述	类型
<b>id</b>		integer
<b>category</b>		<a href="#">Category</a>
<b>name*</b>		string

## template.docx

使用区块对标签完成所有循环功能，可以完美的支持有序和多级列表；表格使用 `LoopRowTableRenderPolicy` 插件的约定，可以非常方便的完成参数、返回值等表格的渲染；使用 Spring 表达式来支持丰富的条件判断；代码块高亮使用 `HighlightRenderPolicy` 插件。

**({{INFO.TITLE}})**

1. 概览

1.1. 描述  
{{info.description}}

1.2. 版本  
{{info.version}}

1.3. 联系人  
{{info.contact.email}}

1.4. 许可证  
{{info.license.name}}

2. 接口列表

({?resources})

2.1. {{NAME}}  
{{description}}

({?endpoints})

2.1.1. {{SUMMARY}}  
{{httpMethod}} {{url}}

描述

({?description}}{{?description == null or description == ""}}{{summary}}{{/}}

---

**PRODUCES**

({?produces == null or produces.size() == 0})无{{/}}

- {{#produces}}

({{/produces}})

---

**CONSUMES**

({?consumes == null or consumes.size() == 0})无{{/}}

- {{#consumes}}

({{/consumes}})

---

**参数**

({parameters}) 参数类型	参数名	描述	数据类型
[in]	[name]{{?required}} <span style="color: red;">必填</span> {{/required}}	[description]	[?schema][{{#this}}][{/schema}]

返回值

({responses})	描述	数据类型
[code]	[description]	[?schema][{{#this}}][{/schema}]

```
[?headers]  
[name]([type]): [description]  
[/headers]
```

{{/endpoints}}

{{/resources}}

### 3. 数据模型

{?definitions}

#### 3.1. {{NAME}}

##### 3.1.1. 属性

属性名	描述	类型
[name][?required]*[/required]	[description]	[?schema][=#this]/

##### 3.1.2. 示例

{definitionCode}

{/definitions}

### 4. 关于文档

代码示例

```

SwaggerParser swaggerParser = new SwaggerParser();
Swagger swagger = swaggerParser.read("https://petstore.swagger.io/v2/swagger.json");
SwaggerView viewData = convert(swagger); ①

LoopRowTableRenderPolicy LoopRowTableRenderPolicy = new LoopRowTableRenderPolicy();
Configure config = Configure.builder()
    .bind("parameters", hackLoopTableRenderPolicy)
    .bind("responses", hackLoopTableRenderPolicy)
    .bind("properties", hackLoopTableRenderPolicy)
    .bind("definitionCode", new HighlightRenderPolicy())
    .useSpringEL()
    .build(); ②

XWPFTemplate template = XWPFTemplate.compile("swagger.docx", config).render(viewData); ③
template.writeToFile("out_example_swagger.docx");

```

① 解析Swagger.json

② 配置模板引擎

③ Swagger导出Word

没错，一切都是如此简洁：简洁的导出代码，简洁的Word模板，甚至生成的Swagger文档都看起来那么简洁，愿一切如你所愿。



源码参见 [JUnit SwaggerToWordExample](#)  
[\(https://github.com/Sayi/poi-tl/tree/master/poi-tl-plugin-highlight/src/test/java/com/deeppoove/poi/plugin/highlight/example/SwaggerToWordExample.java\)](https://github.com/Sayi/poi-tl/tree/master/poi-tl-plugin-highlight/src/test/java/com/deeppoove/poi/plugin/highlight/example/SwaggerToWordExample.java)

## 8. 源码

[GitHub](https://github.com/Sayi/poi-tl) (<https://github.com/Sayi/poi-tl>)

## 9. License

Apache License 2.0

## 10. 版本

Apache POI已经进入5.0.0+时代，如果你仍希望使用低版本的Apache POI，请查阅历史版本。

- 📖 当前版本 1.12.2 Documentation, Apache POI5.2.2+, JDK1.8+
- [1.11.x Documentation](http://deepoove.com/poi-tl/1.11.x/) (http://deepoove.com/poi-tl/1.11.x/), Apache POI5.1.0+, JDK1.8+
- [1.10.x Documentation](http://deepoove.com/poi-tl/1.10.x/) (http://deepoove.com/poi-tl/1.10.x/), Apache POI4.1.2, JDK1.8+
- [1.9.x Documentation](http://deepoove.com/poi-tl/1.9.x/) (http://deepoove.com/poi-tl/1.9.x/), Apache POI4.1.2, JDK1.8+
- [1.8.x Documentation](http://deepoove.com/poi-tl/1.8.x/) (http://deepoove.com/poi-tl/1.8.x/), Apache POI4.1.2, JDK1.8+
- [1.7.x Documentation](http://deepoove.com/poi-tl/1.7.x/) (http://deepoove.com/poi-tl/1.7.x/), Apache POI4.0.0+, JDK1.8+
- [1.6.x Documentation](http://deepoove.com/poi-tl/1.6.x/) (http://deepoove.com/poi-tl/1.6.x/), Apache POI4.0.0+, JDK1.8+
- [1.5.x Documentation](http://deepoove.com/poi-tl/1.5.x/) (http://deepoove.com/poi-tl/1.5.x/), Apache POI3.16+, JDK1.6+

## 11. 打赏个小费

poi-tl开源的初衷是希望让所有有需要的人享受Word模板引擎的功能，而且它可能是Java中最好的Word模板引擎。

如果你觉得它节省了你的时间，给你带来了方便和灵感，或者认同这个开源项目，可以为我的付出打赏点小费哦。



“ poi-tl 是给你的礼物！

— Sayi

## 12. VIP专属服务

你应该不需要VIP专属服务，除非：

- Word技术需要具备一定的专业背景，你需要和作者进行1V1的答疑
- 单纯为知识付费

可以赞赏99元、199元，加作者微信，交个朋友（请在赞赏留言中备注下你的微信号）：



## 13. 常见问题

1. 出现*NoSuchMethodError*、*ClassNotFoundException*、*NoClassDefFoundError*异常？

poi-tl依赖的apache-poi版本是5.2.2+，如果你的项目引用了低版本，请升级或删除。

2. 有没有*HTML*转*Word*的插件？

参考[poi-tl-ext](https://github.com/draco1023/poi-tl-ext) (<https://github.com/draco1023/poi-tl-ext>)。

3. 有没有公式的插件？

参考[poi-tl-ext](https://github.com/draco1023/poi-tl-ext) (<https://github.com/draco1023/poi-tl-ext>)。

4. 如何通过标签指定格式化函数？

Spring表达式，应有尽有。

5. 我不是很熟悉*Apache POI*，我该怎么编写插件？

编写插件还是需要熟悉下POI，你可以参考现有插件的源码，或者Google下Apache POI的用法，这里有一个入门教程：

[Apache POI Word快速入门](http://deepoove.com/poi-tl/apache-poi-guide.html) (<http://deepoove.com/poi-tl/apache-poi-guide.html>)

6. 我有一些问题要请教或者指导，该怎么办？

请参阅本文档的12章节 。