

Use [Maven](#) to build the project and introduce poi related jar packages.

```
1 <groupId>org.apache.poi</groupId>
2     <artifactId>poi</artifactId>
3     <version>3.13</version>
4 </dependency>
5
6 <dependency>
7     <groupId>org.apache.poi</groupId>
8     <artifactId>poi-ooxml</artifactId>
9     <version>3.13</version>
10 </dependency>
```

The code is as follows

- Tools WordTemplate

```
1 /**
2  * @Title: WordTemplate2.java
3  * @Package: com.hidata.templateTools
4  * @Description: TODO
5  * @author: Juvenileless
6  * @date: 2017年11月27日 下午3:23:13
7 */
8 package com.hidata.tool;
9
10 import java.io.IOException;
11 import java.io.InputStream;
12 import java.io.OutputStream;
13 import java.util.ArrayList;
14 import java.util.List;
15 import java.util.Map;
16 import java.util.regex.Matcher;
17 import java.util.regex.Pattern;
18
19 import org.apache.poi.xwpf.usermodel.BodyElementType;
20 import org.apache.poi.xwpf.usermodel.IBodyElement;
21 import org.apache.poi.xwpf.usermodel.PositionInParagraph;
22 import org.apache.poi.xwpf.usermodel.XWPFDocument;
23 import org.apache.poi.xwpf.usermodel.XWPFFParagraph;
24 import org.apache.poi.xwpf.usermodel.XWPFRun;
25 import org.apache.poi.xwpf.usermodel.XWPFTable;
26 import org.apache.poi.xwpf.usermodel.XWPFTableCell;
27 import org.apache.poi.xwpf.usermodel.XWPFTableRow;
28
29 /**
30 *
31 * 对docx文件中的文本及表格中的内容进行替换 --模板仅支持对 {key} 标签的替换
32 *
33 * @ClassName: WordTemplate
34 * @Description: TODO(!!!使用word2013 docx文件)
35 * @author Juvenileless
36 * @date: 2017年11月27日 下午3:25:56
37 * @br>(1)word模板注意页边距的问题，存在问题：比如页边距默认为3cm，画表格时，仍然可以通过
38 * 拖拽，把表格边框拖动到看起来就像页边距只有1cm的样子，但是实际上此时页边距还是3cm，生成的
39 * word报表的页边距还是会按照3cm来生成。解决办法，在word文件里，设置好页边距，如果需要表格
40 * 两边页边距很窄，需要在word里设置页边距窄一点，而不是直接拖动表格边框来实现。
41 *
42 */
43
44 public class WordTemplate {
45
46     private XWPFDocument document;
47
48     public XWPFDocument getDocument() {
49         return document;
50     }
51
52     public void setDocument(XWPFDocument document) {
53         this.document = document;
54     }
55
56 /**
57 * 初始化模板内容
58 *
59 * @author Juvenileless
60 * @date 2017年11月27日 下午3:59:22
61 * @param inputStream
62 *          模板的读取流(docx文件)
63 * @throws IOException
64 *
65 */
66     public WordTemplate(InputStream inputStream) throws IOException {
67         document = new XWPFDocument(inputStream);
68     }
69 }
```

```

69     }
70
71 /**
72 * 将处理后的内容写入到输出流中
73 *
74 * @param outputStream
75 * @throws IOException
76 */
77 public void write(OutputStream outputStream) throws IOException {
78     document.write(outputStream);
79 }
80
81
82
83
84
85 /**
86 * 根据dataMap对word文件中的标签进行替换; <br><br>
87 * ! ! ! ! ***需要注意dataMap的数据格式***! ! ! ! <br><br>
88 * 对于需要替换的普通标签数据标签 (不需要循环) ----必须在dataMap中存储一个key为parametersMap的map,
89 * 来存储这些不需要循环生成的数据, 比如: 表头信息, 日期, 制表人等。 <br><br>
90 * 对于需要循环生成的表格数据-----key自定义, value为 --ArrayList<Map<String, String>>
91 * @author JuvenileSS
92 * @date 2017年11月27日 下午3:29:27
93 * @param dataMap
94 *
95 */
96 public void replaceDocument(Map<String, Object> dataMap) {
97
98     if (!dataMap.containsKey("parametersMap")) {
99         System.out.println("数据源错误--数据源(parametersMap)缺失");
100        return;
101    }
102    @SuppressWarnings("unchecked")
103    Map<String, Object> parametersMap = (Map<String, Object>) dataMap
104        .get("parametersMap");
105
106    List<IBodyElement> bodyElements = document.getBodyElements(); // 所有对象 (段落+表格)
107    int templateBodySize = bodyElements.size(); // 标记模板文件 (段落+表格) 总个数
108
109    int curT = 0; // 当前操作表格对象的索引
110    int curP = 0; // 当前操作段落对象的索引
111    for (int a = 0; a < templateBodySize; a++) {
112        IBodyElement body = bodyElements.get(a);
113        if (BodyElementType.TABLE.equals(body.getElementType())) { // 处理表格
114            XWPFTable table = body.getBody().getTableArray(curT);
115
116            List<XWPFTable> tables = body.getBody().getTables();
117            table = tables.get(curT);
118            if (table != null) {
119
120                // 处理表格
121                List<XWPFTableCell> tableCells = table.getRows().get(0).getTableCells(); // 获取到模板表格第一行, 用来判断表格类型
122                String tableText = table.getText(); // 表格中的所有文本
123
124                if (tableText.indexOf("#{foreach}") > -1) {
125                    // 查找到#{foreach}标签, 该表格需要处理循环
126                    if (tableCells.size() != 2
127                        || tableCells.get(0).getText().indexOf("#{foreach}") < 0
128                        || tableCells.get(0).getText().trim().length() == 0) {
129                        System.out
130                            .println("文档中第"
131                                + (curT + 1)
132                                + "个表格模板错误, 模板表格第一行需要设置2个单元格, "
133                                + "第一个单元格存储表格类型(##foreachTable## 或者 ##foreachTableRow##), 第二个单元格定义数据源。");
134
135                    }
136
137                    String tableType = tableCells.get(0).getText();
138                    String dataSource = tableCells.get(1).getText();
139                    System.out.println("读取到数据源: " + dataSource);
140                    if (!dataMap.containsKey(dataSource)) {
141                        System.out.println("文档中第" + (curT + 1) + "个表格模板数据源缺失");
142                        return;
143                    }
144                    @SuppressWarnings("unchecked")
145                    List<Map<String, Object>> tableDataList = (List<Map<String, Object>>) dataMap
146                        .get(dataSource);
147                    if ("##foreachTable##".equals(tableType)) {
148                        // System.out.println("循环生成表格");
149                        addTableInDocFooter(table, tableDataList, parametersMap, 1);
150
151                    } else if ("##foreachTableRow##".equals(tableType)) {
152                        // System.out.println("循环生成表格内部的行");
153                        addTableInDocFooter(table, tableDataList, parametersMap, 2);
154                    }
155
156                } else if (tableText.indexOf("{") > -1) {
157                    // 没有查找到#{foreach}标签, 查找到了普通替换数据的{}标签, 该表格只需要简单替换

```

```

158         addTableInDocFooter(table, null, parametersMap, 3);
159     } else {
160         // 没有查找到任何标签, 该表格是一个静态表格, 仅需要复制一个即可,
161         addTableInDocFooter(table, null, null, 0);
162     }
163     curT++;
164
165 }
166 } else if (BodyElementType.PARAGRAPH.equals(body.getElementType())) { // 处理段落
167     // System.out.println("获取到段落");
168     XWPFPParagraph ph = body.getBody().getParagraphArray(curP);
169     if (ph != null) {
170         // htmlText = htmlText+readParagraphX(ph);
171         addParagraphInDocFooter(ph, null, parametersMap, 0);
172
173         curP++;
174     }
175 }
176
177 }
178 // 处理完毕模板, 删除文本中的模板内容
179 for (int a = 0; a < templateBodySize; a++) {
180     document.removeBodyElement(a);
181 }
182
183 }
184
185
186
187
188
189
190
191
192 /**
193 * 根据 模板表格 和 数据list 在word文档末尾生成表格
194 * @author JuvenileSS
195 * @date 2017年12月6日 上午10:12:05
196 * @param templateTable 模板表格
197 * @param list 循环数据集
198 * @param parametersMap 不循环数据集
199 * @param flag (0为静态表格, 1为表格整体循环, 2为表格内部行循环, 3为表格不循环仅简单替换标签即可)
200 *
201 */
202 public void addTableInDocFooter(XWPFTable templateTable, List<Map<String, Object>> list,
203     Map<String, Object> parametersMap, int flag) {
204
205     if (flag == 1) { // 表格整体循环
206         for (Map<String, Object> map : list) {
207             List<XWPFTableRow> templateTableRows = templateTable.getRows(); // 获取模板表格所有行
208             XWPFTable newCreateTable = document.createTable(); // 创建新表格, 默认一行一列
209             for (int i = 1; i < templateTableRows.size(); i++) {
210                 XWPFTableRow newCreateRow = newCreateTable.createRow();
211                 CopyTableRow(newCreateRow, templateTableRows.get(i)); // 复制模板行文本和样式到新行
212             }
213             newCreateTable.removeRow(0); // 移除多出来的第一行
214             document.createParagraph(); // 添加回车换行
215             replaceTable(newCreateTable, map); // 替换标签
216         }
217     }
218
219     } else if (flag == 2) { // 表格表格内部行循环
220         XWPFTable newCreateTable = document.createTable(); // 创建新表格, 默认一行一列
221         List<XWPFTableRow> TempTableRows = templateTable.getRows(); // 获取模板表格所有行
222         int tagRowIndex = 0; // 标签行index
223         for (int i = 0, size = TempTableRows.size(); i < size; i++) {
224             String rowText = TempTableRows.get(i).getCell(0).getText(); // 获取到表格行的第一个单元格
225             if (rowText.indexOf("#{foreachRows}#") > -1) {
226                 tagRowIndex = i;
227                 break;
228             }
229         }
230
231         /* 复制模板行和标签行之前的行 */
232         for (int i = 1; i < tagRowIndex; i++) {
233             XWPFTableRow newCreateRow = newCreateTable.createRow();
234             CopyTableRow(newCreateRow, TempTableRows.get(i)); // 复制行
235             replaceTableRow(newCreateRow, parametersMap); // 处理不循环标签的替换
236         }
237
238         /* 循环生成模板行 */
239         XWPFTableRow tempRow = TempTableRows.get(tagRowIndex + 1); // 获取到模板行
240         for (int i = 0; i < list.size(); i++) {
241             XWPFTableRow newCreateRow = newCreateTable.createRow();
242             CopyTableRow(newCreateRow, tempRow); // 复制模板行
243             replaceTableRow(newCreateRow, list.get(i)); // 处理标签替换
244         }
245
246         /* 复制模板行和标签行之后的行 */
247         for (int i = tagRowIndex + 2; i < TempTableRows.size(); i++) {

```

```

248         XWPFTableRow newCreateRow = newCreateTable.createRow();
249         CopyTableRow(newCreateRow, TempTableRows.get(i)); // 复制行
250         replaceTableRow(newCreateRow, parametersMap); // 处理不循环标签的替换
251     }
252     newCreateTable.removeRow(0); // 移除多出来的第一行
253     document.createParagraph(); // 添加回车换行
254
255 } else if (flag == 3) {
256     //表格不循环仅简单替换标签
257     List<XWPFTableRow> templateTableRows = templateTable.getRows(); // 获取模板表格所有行
258     XWPFTable newCreateTable = document.createTable(); // 创建新表格,默认一行一列
259     for (int i = 0; i < templateTableRows.size(); i++) {
260         XWPFTableRow newCreateRow = newCreateTable.createRow();
261         CopyTableRow(newCreateRow, templateTableRows.get(i)); // 复制模板行文本和样式到新行
262     }
263     newCreateTable.removeRow(0); // 移除多出来的第一行
264     document.createParagraph(); // 添加回车换行
265     replaceTable(newCreateTable, parametersMap);
266
267 } else if (flag == 0) {
268     List<XWPFTableRow> templateTableRows = templateTable.getRows(); // 获取模板表格所有行
269     XWPFTable newCreateTable = document.createTable(); // 创建新表格,默认一行一列
270     for (int i = 0; i < templateTableRows.size(); i++) {
271         XWPFTableRow newCreateRow = newCreateTable.createRow();
272         CopyTableRow(newCreateRow, templateTableRows.get(i)); // 复制模板行文本和样式到新行
273     }
274     newCreateTable.removeRow(0); // 移除多出来的第一行
275     document.createParagraph(); // 添加回车换行
276 }
277
278 /**
279 * 根据 模板段落 和 数据 在文档末尾生成段落
280 *
281 * @author Juvenileless
282 * @date 2017年11月27日 上午11:49:42
283 * @param templateParagraph
284 * 模板段落
285 * @param list
286 * 循环数据集
287 * @param parametersMap
288 * 不循环数据集
289 * @param flag
290 * (0为不循环替换, 1为循环替换)
291 *
292 */
293 public void addParagraphInDocFooter(XWPFParagraph templateParagraph,
294     List<Map<String, String>> list, Map<String, Object> parametersMap, int flag) {
295
296     if (flag == 0) {
297         XWPFParagraph createParagraph = document.createParagraph();
298         // 设置段落样式
299         createParagraph.getCTP().setPPr(templateParagraph.getCTP().getPPr());
300         // 移除原始内容
301         for (int pos = 0; pos < createParagraph.getRuns().size(); pos++) {
302             createParagraph.removeRun(pos);
303         }
304         // 添加run标签
305         for (XWPFRun s : templateParagraph.getRuns()) {
306             XWPFRun targetrun = createParagraph.createRun();
307             CopyRun(targetrun, s);
308         }
309     }
310     replaceParagraph(createParagraph, parametersMap);
311
312 } else if (flag == 1) {
313     // 暂无实现
314 }
315
316 }
317
318 /**
319 * 根据map替换段落元素内的(**)标签
320 * @author Juvenileless
321 * @date 2017年12月4日 下午3:09:00
322 * @param xWPFParagraph
323 * @param parametersMap
324 *
325 */
326 public void replaceParagraph(XWPFParagraph xWPFParagraph, Map<String, Object> parametersMap) {
327
328 }
```

```

337 List<XWPFRun> runs = xWPFParagraph.getRuns();
338 String xWPFParagraphText = xWPFParagraph.getText();
339 String regEx = "\\{(.+?\\}\\}";
340 Pattern pattern = Pattern.compile(regEx);
341 Matcher matcher = pattern.matcher(xWPFParagraphText);//正则匹配字符串{***}
342
343 if (matcher.find()) {
344     // 查找到有标签才执行替换
345     int beginRunIndex = xWPFParagraph.searchText("{", new PositionInParagraph()).getBeginRun();// 标签开始run位置
346     int endRunIndex = xWPFParagraph.searchText("}", new PositionInParagraph()).getEndRun();// 结束标签
347     StringBuffer key = new StringBuffer();
348
349     if (beginRunIndex == endRunIndex) {
350         // {**}在一个run标签内
351         XWPFRun beginRun = runs.get(beginRunIndex);
352         String beginRunText = beginRun.text();
353
354         int beginIndex = beginRunText.indexOf("{");
355         int endIndex = beginRunText.indexOf("}");
356         int length = beginRunText.length();
357
358         if (beginIndex == 0 && endIndex == length - 1) {
359             // 该run标签只有**
360             XWPFRun insertNewRun = xWPFParagraph.insertNewRun(beginRunIndex);
361             insertNewRun.getCTR().setRPr(beginRun.getCTR().getRPr());
362             // 设置文本
363             key.append(beginRunText.substring(1, endIndex));
364             insertNewRun.setText(getValueByKey(key.toString(), parametersMap));
365             xWPFParagraph.removeRun(beginRunIndex + 1);
366         } else {
367             // 该run标签为**{**} 或者 **{**} 或者 **}**，替换key后，还需要加上原始key前后的文本
368             XWPFRun insertNewRun = xWPFParagraph.insertNewRun(beginRunIndex);
369             insertNewRun.getCTR().setRPr(beginRun.getCTR().getRPr());
370             // 设置文本
371             key.append(beginRunText.substring(beginRunText.indexOf("{") + 1, beginRunText.indexOf("}")));
372             String textString=beginRunText.substring(0, beginIndex) + getValueByKey(key.toString(), parametersMap)
373             + beginRunText.substring(endIndex + 1);
374             insertNewRun.setText(textString);
375             xWPFParagraph.removeRun(beginRunIndex + 1);
376         }
377     } else {
378         // {**}被分成多个run
379
380         //先处理起始run标签, 取得第一个{key}值
381         XWPFRun beginRun = runs.get(beginRunIndex);
382         String beginRunText = beginRun.text();
383         int beginIndex = beginRunText.indexOf("{");
384         if (beginRunText.length()>1 ) {
385             key.append(beginRunText.substring(beginIndex+1));
386         }
387         ArrayList<Integer> removeRunList = new ArrayList<>();//需要移除的run
388         //处理中间的run
389         for (int i = beginRunIndex + 1; i < endRunIndex; i++) {
390             XWPFRun run = runs.get(i);
391             String runText = run.text();
392             key.append(runText);
393             removeRunList.add(i);
394         }
395
396         // 获取endRun中的key值
397         XWPFRun endRun = runs.get(endRunIndex);
398         String endRunText = endRun.text();
399         int endIndex = endRunText.indexOf("}");
400         //run中**}或者**}**
401         if (endRunText.length()>1 && endIndex!=0) {
402             key.append(endRunText.substring(0,endIndex));
403         }
404
405
406
407         //*****取得key值后替换标签
408
409         //先处理开始标签
410         if (beginRunText.length()==2 ) {
411             // run标签内文本(
412             XWPFRun insertNewRun = xWPFParagraph.insertNewRun(beginRunIndex);
413             insertNewRun.getCTR().setRPr(beginRun.getCTR().getRPr());
414             // 设置文本
415             insertNewRun.setText(getValueByKey(key.toString(), parametersMap));
416             xWPFParagraph.removeRun(beginRunIndex + 1);//移除原始的run
417         }else {
418             // 该run标签为**{**或者 (**，替换key后，还需要加上原始key前的文本
419             XWPFRun insertNewRun = xWPFParagraph.insertNewRun(beginRunIndex);
420             insertNewRun.getCTR().setRPr(beginRun.getCTR().getRPr());
421             // 设置文本
422             String textString=beginRunText.substring(0,beginRunText.indexOf("{"))+getValueByKey(key.toString(), parametersMap);
423             insertNewRun.setText(textString);
424         }
425     }
426 }

```

```

426         xWPFPParagraph.removeRun(beginRunIndex + 1); // 移除原始的run
427     }
428
429     // 处理结束标签
430     if (endRunText.length() == 1) {
431         // run标签内文本只有
432         XWPFRun insertNewRun = xWPFPParagraph.insertNewRun(endRunIndex);
433         insertNewRun.getCTR().setRPr(endRun.getCTR().getRPr());
434         // 设置文本
435         insertNewRun.setText("");
436         xWPFPParagraph.removeRun(endRunIndex + 1); // 移除原始的run
437
438     } else {
439         // 该run标签为**或者 }** 或者 **), 替换key后, 还需要加上原始key后的文本
440         XWPFRun insertNewRun = xWPFPParagraph.insertNewRun(endRunIndex);
441         insertNewRun.getCTR().setRPr(endRun.getCTR().getRPr());
442         // 设置文本
443         String textString = endRunText.substring(endRunText.indexOf("}") + 1);
444         insertNewRun.setText(textString);
445         xWPFPParagraph.removeRun(endRunIndex + 1); // 移除原始的run
446     }
447
448     // 处理中间的run标签
449     for (int i = 0; i < removeRunList.size(); i++) {
450         XWPFRun xWPFRun = runs.get(removeRunList.get(i)); // 原始run
451         XWPFRun insertNewRun = xWPFPParagraph.insertNewRun(removeRunList.get(i));
452         insertNewRun.getCTR().setRPr(xWPFRun.getCTR().getRPr());
453         insertNewRun.setText("");
454         xWPFPParagraph.removeRun(removeRunList.get(i) + 1); // 移除原始的run
455     }
456
457     } // 处理{**}被分成多个run
458
459     replaceParagraph(xWPFPParagraph, parametersMap);
460
461 } // if 有标签
462
463 }
464
465
466
467
468
469
470
471 /**
472 * 复制表格行XWPFTableRow格式
473 *
474 * @param target
475 *          待修改格式的XWPFTableRow
476 * @param source
477 *          模板XWPFTableRow
478 */
479 private void CopyTableRow(XWPFTableRow target, XWPFTableRow source) {
480
481     int tempRowCellsize = source.getTableCells().size(); // 模板行的列数
482     for (int i = 0; i < tempRowCellsize - 1; i++) {
483         target.addNewTableCell(); // 为新添加的行添加与模板表格对应行行相同个数的单元格
484     }
485     // 复制样式
486     target.getCtRow().setTrPr(source.getCtRow().getTrPr());
487     // 复制单元格
488     for (int i = 0; i < target.getTableCells().size(); i++) {
489         copyTableCell(target.getCell(i), source.getCell(i));
490     }
491 }
492
493
494
495
496
497
498 /**
499 * 复制单元格XWPFTableCell格式
500 *
501 * @author Juvenile
502 * @date 2017年11月27日 下午3:41:02
503 * @param newTableCell
504 *          新创建的的单元格
505 * @param templateTableCell
506 *          模板单元格
507 *
508 */
509 private void copyTableCell(XWPFTableCell newTableCell, XWPFTableCell templateTableCell) {
510     // 属性
511     newTableCell.getCTC().setTcPr(templateTableCell.getCTC().getTcPr());
512     // 删除目标 targetCell 所有文本段落
513     for (int pos = 0; pos < newTableCell.getParagraphs().size(); pos++) {
514         newTableCell.removeParagraph(pos);

```

```

515     }
516     // 添加新文本段落
517     for (XWPFParagraph sp : templateTableCell.getParagraphs()) {
518         XWPFParagraph targetP = newTableCell.addParagraph();
519         copyParagraph(targetP, sp);
520     }
521 }
522 /**
523 * 复制文本段落XWPFParagraph格式
524 *
525 * @author JuvenileSS
526 * @date 2017年11月27日 下午3:43:08
527 * @param newParagraph
528 *          新创建的的段落
529 * @param templateParagraph
530 *          模板段落
531 *
532 */
533 /**
534 private void copyParagraph(XWPFParagraph newParagraph, XWPFParagraph templateParagraph) {
535     // 设置段落样式
536     newParagraph.getCTP().setPPr(templateParagraph.getCTP().getPPr());
537     // 添加Run标签
538     for (int pos = 0; pos < newParagraph.getRuns().size(); pos++) {
539         newParagraph.removeRun(pos);
540     }
541     for (XWPFRun s : templateParagraph.getRuns()) {
542         XWPFRun targetrun = newParagraph.createRun();
543         CopyRun(targetrun, s);
544     }
545 }
546 /**
547 * 复制文本节点run
548 * @author JuvenileSS
549 * @date 2017年11月27日 下午3:47:17
550 * @param newRun
551 *          新创建的的文本节点
552 * @param templateRun
553 *          模板文本节点
554 *
555 */
556 private void CopyRun(XWPFRun newRun, XWPFRun templateRun) {
557     newRun.getCTR().setRPr(templateRun.getCTR().getRPr());
558     // 设置文本
559     newRun.setText(templateRun.text());
560 }
561 /**
562 * 根据参数parametersMap对表格的一行进行标签的替换
563 *
564 * @author JuvenileSS
565 * @date 2017年11月23日 下午2:09:24
566 * @param tableRow
567 *          表格行
568 * @param parametersMap
569 *          参数map
570 *
571 */
572 public void replaceTableRow(XWPFTableRow tableRow, Map<String, Object> parametersMap) {
573     List<XWPFTableCell> tableCells = tableRow.getTableCells();
574     for (XWPFTableCell xwpfTableCell : tableCells) {
575         List<XWPFParagraph> paragraphs = xwpfTableCell.getParagraphs();
576         for (XWPFParagraph xwpfParagraph : paragraphs) {
577             replaceParagraph(xwpfParagraph, parametersMap);
578         }
579     }
580 }
581 /**
582 * 根据map替换表格中的{key}标签
583 * @author JuvenileSS
584 * @date 2017年12月4日 下午2:47:36
585 * @param xwpfTable
586 * @param parametersMap
587 */
588 
```

```

604 *
605 */
606 public void replaceTable(XWPFTable xwpfTable, Map<String, Object> parametersMap){
607     List<XWPFTableRow> rows = xwpfTable.getRows();
608     for (XWPFTableRow xWPFTableRow : rows) {
609         List<XWPFTableCell> tableCells = xWPFTableRow.getTableCells();
610         for (XWPFTableCell xWPFTableCell : tableCells) {
611             List<XWPFFParagraph> paragraphs2 = xWPFTableCell.getParagraphs();
612             for (XWPFFParagraph xWPFFParagraph : paragraphs2) {
613                 replaceParagraph(xWPFFParagraph, parametersMap);
614             }
615         }
616     }
617 }
618 }
619
620
621
622 private String getValueBykey(String key, Map<String, Object> map) {
623     String returnValue="";
624     if (key != null) {
625         try {
626             returnValue=map.get(key)!=null ? map.get(key).toString() : "";
627         } catch (Exception e) {
628             // TODO: handle exception
629             System.out.println("key:"+key+"***"+e);
630             returnValue="";
631         }
632     }
633     return returnValue;
634 }
635
636
637
638
639
640
641
642
643
}

```

- use

```

1 package com.hidata.tool;
2
3 import java.io.File;
4 import java.io.FileInputStream;
5 import java.io.FileOutputStream;
6 import java.io.IOException;
7 import java.util.ArrayList;
8 import java.util.HashMap;
9 import java.util.List;
10 import java.util.Map;
11
12 public class Test {
13
14     public static void main(String[] args) throws IOException {
15
16         Map<String, Object> wordDataMap = new HashMap<String, Object>();// 存储报表全部数据
17         Map<String, Object> parametersMap = new HashMap<String, Object>();// 存储报表中不循环的数据
18
19
20
21         List<Map<String, Object>> table1 = new ArrayList<Map<String, Object>>();
22         Map<String, Object> map1=new HashMap<String, Object>();
23         map1.put("name", "张三");
24         map1.put("age", "23");
25         map1.put("email", "12121@qq.com");
26
27         Map<String, Object> map2=new HashMap<String, Object>();
28         map2.put("name", "李四");
29         map2.put("age", "45");
30         map2.put("email", "45445@qq.com");
31
32         Map<String, Object> map3=new HashMap<String, Object>();
33         map3.put("name", "Tom");
34         map3.put("age", "34");
35         map3.put("email", "6767@qq.com");
36
37         table1.add(map1);
38         table1.add(map2);
39         table1.add(map3);
40
41

```

```
42
43
44
45 List<Map<String, Object>> table2 = new ArrayList<Map<String, Object>>();
46 Map<String, Object> map4=new HashMap<>();
47 map4.put("name", "tom");
48 map4.put("number", "sd1234");
49 map4.put("address", "上海");
50
51 Map<String, Object> map5=new HashMap<>();
52 map5.put("name", "seven");
53 map5.put("number", "sd15678");
54 map5.put("address", "北京");
55
56 Map<String, Object> map6=new HashMap<>();
57 map6.put("name", "lisa");
58 map6.put("number", "sd9078");
59 map6.put("address", "广州");
60
61 table2.add(map4);
62 table2.add(map5);
63 table2.add(map6);
64
65
66
67 parametersMap.put("userName", "JUVENILESS");
68 parametersMap.put("time", "2018-03-24");
69 parametersMap.put("sum", "3");
70
71
72 wordDataMap.put("table1", table1);
73 wordDataMap.put("table2", table2);
74 wordDataMap.put("parametersMap", parametersMap);
75 File file = new File("D:\\Workspaces\\Eclipse 2017\\wordTemplate\\doc\\模板.docx");//改成你本地文件所在目录
76
77
78 // 读取word模板
79 FileInputStream fileInputStream = new FileInputStream(file);
80 WordTemplate template = new WordTemplate(fileInputStream);
81
82 // 替换数据
83 template.replaceDocument(wordDataMap);
84
85
86 //生成文件
87 File outputFile=new File("D:\\Workspaces\\Eclipse 2017\\wordTemplate\\doc\\输出.docx");//改成你本地文件所在目录
88 FileOutputStream fos = new FileOutputStream(outputFile);
89 template.getDocument().write(fos);
90
91 }
92
93 }
```

 Juvenileless focus on