Use Maven to build the project and introduce poi related jar packages.

```
<groupId>org.apache.poi</groupId>
            <artifactId>poi</artifactId>
            <version>3.13</version>
        </dependency>

        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi-ooxml</artifactId>
            <version>3.13</version>
        </dependency>
```

The code is as follows

- **Tools WordTemplate**

```
/**
 * @Title: WordTemplate2.java
 * @Package: com.highdata.templateTools
 * @Description: TODO
 * @author: Juveniless
 * @date: 2017年11月27日 下午3:23:13
 */
package com.hidata.tool;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import org.apache.poi.xwpf.usermodel.BodyElementType;
import org.apache.poi.xwpf.usermodel.IBodyElement;
import org.apache.poi.xwpf.usermodel.PositionInParagraph;
import org.apache.poi.xwpf.usermodel.XWPFDocument;
import org.apache.poi.xwpf.usermodel.XWPFParagraph;
import org.apache.poi.xwpf.usermodel.XWPFRun;
import org.apache.poi.xwpf.usermodel.XWPFTable;
import org.apache.poi.xwpf.usermodel.XWPFTableCell;
import org.apache.poi.xwpf.usermodel.XWPFTableRow;

/**
 *
 * 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
 *
 * @ClassName: WordTemplate
 * @Description: TODO(!!!使用word2013 docx文件)
 * @author Juveniless
 * @date: 2017年11月27日 下午3:25:56
 * <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
 * 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
 * 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
 * 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。
 *
 */

public class WordTemplate {

    private XWPFDocument document;

    public XWPFDocument getDocument() {
        return document;
    }

    public void setDocument(XWPFDocument document) {
        this.document = document;
    }

    /**
     * 初始化模板内容
     *
     * @author Juveniless
     * @date 2017年11月27日 下午3:59:22
     * @param inputStream
     *            模板的读取流(docx文件)
     * @throws IOException
     *
     */
    public WordTemplate(InputStream inputStream) throws IOException {
        document = new XWPFDocument(inputStream);
```

```
    }

    /**
     * 将处理后的内容写入到输出流中
     *
     * @param outputStream
     * @throws IOException
     */
    public void write(OutputStream outputStream) throws IOException {
        document.write(outputStream);
    }




    /**
     * 根据dataMap对word文件中的标签进行替换; <br><br>
     * ！！！！***需要注意dataMap的数据格式***！！！！ <br><br>
     * 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,
     * 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>
     * 对于需要循环生成的表格数据------key自定义，value为 --ArrayList&lt;Map&lt;String, String>>
     * @author Juveniless
     * @date 2017年11月27日 下午3:29:27
     * @param dataMap
     *
     */
    public void replaceDocument(Map<String, Object> dataMap) {

        if (!dataMap.containsKey("parametersMap")) {
            System.out.println("数据源错误--数据源(parametersMap)缺失");
            return;
        }
        @SuppressWarnings("unchecked")
        Map<String, Object> parametersMap = (Map<String, Object>) dataMap
                .get("parametersMap");

        List<IBodyElement> bodyElements = document.getBodyElements();// 所有对象（段落+表格）
        int templateBodySize = bodyElements.size();// 标记模板文件（段落+表格）总个数

        int curT = 0;// 当前操作表格对象的索引
        int curP = 0;// 当前操作段落对象的索引
        for (int a = 0; a < templateBodySize; a++) {
            IBodyElement body = bodyElements.get(a);
            if (BodyElementType.TABLE.equals(body.getElementType())) {// 处理表格
                XWPFTable table = body.getBody().getTableArray(curT);

                List<XWPFTable> tables = body.getBody().getTables();
                table = tables.get(curT);
                if (table != null) {

                    // 处理表格
                    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
                    String tableText = table.getText();// 表格中的所有文本

                    if (tableText.indexOf("##{foreach") > -1) {
                        // 查找到##{foreach标签，该表格需要处理循环
                        if (tableCells.size() != 2
                                || tableCells.get(0).getText().indexOf("##{foreach") < 0
                                || tableCells.get(0).getText().trim().length() == 0) {
                            System.out
                                    .println("文档中第"
                                            + (curT + 1)
                                            + "个表格模板错误,模板表格第一行需要设置2个单元格，"
                                            + "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。");
                            return;
                        }

                        String tableType = tableCells.get(0).getText();
                        String dataSource = tableCells.get(1).getText();
                        System.out.println("读取到数据源："+dataSource);
                        if (!dataMap.containsKey(dataSource)) {
                            System.out.println("文档中第" + (curT + 1) + "个表格模板数据源缺失");
                            return;
                        }
                        @SuppressWarnings("unchecked")
                        List<Map<String, Object>> tableDataList = (List<Map<String, Object>>) dataMap
                                .get(dataSource);
                        if ("##{foreachTable}##".equals(tableType)) {
                            // System.out.println("循环生成表格");
                            addTableInDocFooter(table, tableDataList, parametersMap, 1);

                        } else if ("##{foreachTableRow}##".equals(tableType)) {
                            // System.out.println("循环生成表格内部的行");
                            addTableInDocFooter(table, tableDataList, parametersMap, 2);
                        }

                    } else if (tableText.indexOf("{") > -1) {
                        // 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
```

```
                        addTableInDocFooter(table, null, parametersMap, 3);
                    } else {
                        // 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。
                        addTableInDocFooter(table, null, null, 0);
                    }
                    curT++;

                }
            } else if (BodyElementType.PARAGRAPH.equals(body.getElementType())) {// 处理段落
                // System.out.println("获取到段落");
                XWPFParagraph ph = body.getBody().getParagraphArray(curP);
                if (ph != null) {
                    // htmlText = htmlText+readParagraphX(ph);
                    addParagraphInDocFooter(ph, null, parametersMap, 0);

                    curP++;
                }
            }

        }
        // 处理完毕模板，删除文本中的模板内容
        for (int a = 0; a < templateBodySize; a++) {
            document.removeBodyElement(0);
        }

    }









    /**
     * 根据 模板表格 和 数据list 在word文档末尾生成表格
     * @author Juveniless
     * @date 2017年12月6日 上午10:12:05
     * @param templateTable 模板表格
     * @param list   循环数据集
     * @param parametersMap  不循环数据集
     * @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)
     *
     */
    public void addTableInDocFooter(XWPFTable templateTable, List<Map<String, Object>> list,
            Map<String, Object> parametersMap, int flag) {

        if (flag == 1) {// 表格整体循环
            for (Map<String, Object> map : list) {
                List<XWPFTableRow> templateTableRows = templateTable.getRows();// 获取模板表格所有行
                XWPFTable newCreateTable = document.createTable();// 创建新表格,默认一行一列
                for (int i = 1; i < templateTableRows.size(); i++) {
                    XWPFTableRow newCreateRow = newCreateTable.createRow();
                    CopyTableRow(newCreateRow, templateTableRows.get(i));// 复制模板行文本和样式到新行
                }
                newCreateTable.removeRow(0);// 移除多出来的第一行
                document.createParagraph();// 添加回车换行
                replaceTable(newCreateTable, map);//替换标签
            }

        } else if (flag == 2) {// 表格表格内部行循环
            XWPFTable newCreateTable = document.createTable();// 创建新表格,默认一行一列
            List<XWPFTableRow> TempTableRows = templateTable.getRows();// 获取模板表格所有行
            int tagRowsIndex = 0;// 标签行indexs
            for (int i = 0, size = TempTableRows.size(); i < size; i++) {
                String rowText = TempTableRows.get(i).getCell(0).getText();// 获取到表格行的第一个单元格
                if (rowText.indexOf("##{foreachRows}##") > -1) {
                    tagRowsIndex = i;
                    break;
                }
            }

            /* 复制模板行和标签行之前的行 */
            for (int i = 1; i < tagRowsIndex; i++) {
                XWPFTableRow newCreateRow = newCreateTable.createRow();
                CopyTableRow(newCreateRow, TempTableRows.get(i));// 复制行
                replaceTableRow(newCreateRow, parametersMap);// 处理不循环标签的替换
            }

            /* 循环生成模板行 */
            XWPFTableRow tempRow = TempTableRows.get(tagRowsIndex + 1);// 获取到模板行
            for (int i = 0; i < list.size(); i++) {
                XWPFTableRow newCreateRow = newCreateTable.createRow();
                CopyTableRow(newCreateRow, tempRow);// 复制模板行
                replaceTableRow(newCreateRow, list.get(i));// 处理标签替换
            }

            /* 复制模板行和标签行之后的行 */
            for (int i = tagRowsIndex + 2; i < TempTableRows.size(); i++) {
```

```
                XWPFTableRow newCreateRow = newCreateTable.createRow();
                CopyTableRow(newCreateRow, TempTableRows.get(i));// 复制行
                replaceTableRow(newCreateRow, parametersMap);// 处理不循环标签的替换
            }
            newCreateTable.removeRow(0);// 移除多出来的第一行
            document.createParagraph();// 添加回车换行

        } else if (flag == 3) {
            //表格不循环仅简单替换标签
            List<XWPFTableRow> templateTableRows = templateTable.getRows();// 获取模板表格所有行
            XWPFTable newCreateTable = document.createTable();// 创建新表格,默认一行一列
            for (int i = 0; i < templateTableRows.size(); i++) {
                XWPFTableRow newCreateRow = newCreateTable.createRow();
                CopyTableRow(newCreateRow, templateTableRows.get(i));// 复制模板行文本和样式到新行
            }
            newCreateTable.removeRow(0);// 移除多出来的第一行
            document.createParagraph();// 添加回车换行
            replaceTable(newCreateTable, parametersMap);

        } else if (flag == 0) {
            List<XWPFTableRow> templateTableRows = templateTable.getRows();// 获取模板表格所有行
            XWPFTable newCreateTable = document.createTable();// 创建新表格,默认一行一列
            for (int i = 0; i < templateTableRows.size(); i++) {
                XWPFTableRow newCreateRow = newCreateTable.createRow();
                CopyTableRow(newCreateRow, templateTableRows.get(i));// 复制模板行文本和样式到新行
            }
            newCreateTable.removeRow(0);// 移除多出来的第一行
            document.createParagraph();// 添加回车换行
        }

    }




    /**
     * 根据 模板段落 和 数据 在文档末尾生成段落
     *
     * @author Juveniless
     * @date 2017年11月27日 上午11:49:42
     * @param templateParagraph
     *            模板段落
     * @param list
     *            循环数据集
     * @param parametersMap
     *            不循环数据集
     * @param flag
     *            (0为不循环替换，1为循环替换)
     *
     */
    public void addParagraphInDocFooter(XWPFParagraph templateParagraph,
            List<Map<String, String>> list, Map<String, Object> parametersMap, int flag) {

        if (flag == 0) {
            XWPFParagraph createParagraph = document.createParagraph();
            // 设置段落样式
            createParagraph.getCTP().setPPr(templateParagraph.getCTP().getPPr());
            // 移除原始内容
            for (int pos = 0; pos < createParagraph.getRuns().size(); pos++) {
                createParagraph.removeRun(pos);
            }
            // 添加Run标签
            for (XWPFRun s : templateParagraph.getRuns()) {
                XWPFRun targetrun = createParagraph.createRun();
                CopyRun(targetrun, s);
            }

            replaceParagraph(createParagraph, parametersMap);

        } else if (flag == 1) {
            // 暂无实现
        }

    }




    /**
     * 根据map替换段落元素内的{**}标签
     * @author Juveniless
     * @date 2017年12月4日 下午3:09:00
     * @param xWPFParagraph
     * @param parametersMap
     *
     */
    public void replaceParagraph(XWPFParagraph xWPFParagraph, Map<String, Object> parametersMap) {
```

```
        List<XWPFRun> runs = xWPFParagraph.getRuns();
        String xWPFParagraphText = xWPFParagraph.getText();
        String regEx = "\\{.+?\\}";
        Pattern pattern = Pattern.compile(regEx);
        Matcher matcher = pattern.matcher(xWPFParagraphText);//正则匹配字符串{****}

        if (matcher.find()) {
            // 查找到有标签才执行替换
            int beginRunIndex = xWPFParagraph.searchText("{", new PositionInParagraph()).getBeginRun();// 标签开始run位置
            int endRunIndex = xWPFParagraph.searchText("}", new PositionInParagraph()).getEndRun();// 结束标签
            StringBuffer key = new StringBuffer();

            if (beginRunIndex == endRunIndex) {
                // {**}在一个run标签内
                XWPFRun beginRun = runs.get(beginRunIndex);
                String beginRunText = beginRun.text();

                int beginIndex = beginRunText.indexOf("{");
                int endIndex = beginRunText.indexOf("}");
                int length = beginRunText.length();

                if (beginIndex == 0 && endIndex == length - 1) {
                    // 该run标签只有{**}
                    XWPFRun insertNewRun = xWPFParagraph.insertNewRun(beginRunIndex);
                    insertNewRun.getCTR().setRPr(beginRun.getCTR().getRPr());
                    // 设置文本
                    key.append(beginRunText.substring(1, endIndex));
                    insertNewRun.setText(getValueBykey(key.toString(),parametersMap));
                    xWPFParagraph.removeRun(beginRunIndex + 1);
                } else {
                    // 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
                    XWPFRun insertNewRun = xWPFParagraph.insertNewRun(beginRunIndex);
                    insertNewRun.getCTR().setRPr(beginRun.getCTR().getRPr());
                    // 设置文本
                    key.append(beginRunText.substring(beginRunText.indexOf("{")+1, beginRunText.indexOf("}")));
                    String textString=beginRunText.substring(0, beginIndex) + getValueBykey(key.toString(),parametersMap)
                            + beginRunText.substring(endIndex + 1);
                    insertNewRun.setText(textString);
                    xWPFParagraph.removeRun(beginRunIndex + 1);
                }

            }else {
                // {**}被分成多个run

                //先处理起始run标签,取得第一个{key}值
                XWPFRun beginRun = runs.get(beginRunIndex);
                String beginRunText = beginRun.text();
                int beginIndex = beginRunText.indexOf("{");
                if (beginRunText.length()>1  ) {
                    key.append(beginRunText.substring(beginIndex+1));
                }
                ArrayList<Integer> removeRunList = new ArrayList<>();//需要移除的run
                //处理中间的run
                for (int i = beginRunIndex + 1; i < endRunIndex; i++) {
                    XWPFRun run = runs.get(i);
                    String runText = run.text();
                    key.append(runText);
                    removeRunList.add(i);
                }

                // 获取endRun中的key值
                XWPFRun endRun = runs.get(endRunIndex);
                String endRunText = endRun.text();
                int endIndex = endRunText.indexOf("}");
                //run中**}或者**}**
                if (endRunText.length()>1 && endIndex!=0) {
                    key.append(endRunText.substring(0,endIndex));
                }



                //*********************************************************************
                //取得key值后替换标签

                //先处理开始标签
                if (beginRunText.length()==2 ) {
                    // run标签内文本{
                    XWPFRun insertNewRun = xWPFParagraph.insertNewRun(beginRunIndex);
                    insertNewRun.getCTR().setRPr(beginRun.getCTR().getRPr());
                    // 设置文本
                    insertNewRun.setText(getValueBykey(key.toString(),parametersMap));
                    xWPFParagraph.removeRun(beginRunIndex + 1);//移除原始的run
                }else {
                    // 该run标签为**{**或者 {** ，替换key后，还需要加上原始key前的文本
                    XWPFRun insertNewRun = xWPFParagraph.insertNewRun(beginRunIndex);
                    insertNewRun.getCTR().setRPr(beginRun.getCTR().getRPr());
                    // 设置文本
                    String textString=beginRunText.substring(0,beginRunText.indexOf("{"))+getValueBykey(key.toString(),parametersMap);
                    insertNewRun.setText(textString);
```

```
                    xWPFParagraph.removeRun(beginRunIndex + 1);//移除原始的run
                }

                //处理结束标签
                if (endRunText.length()==1 ) {
                    // run标签内文本只有}
                    XWPFRun insertNewRun = xWPFParagraph.insertNewRun(endRunIndex);
                    insertNewRun.getCTR().setRPr(endRun.getCTR().getRPr());
                    // 设置文本
                    insertNewRun.setText("");
                    xWPFParagraph.removeRun(endRunIndex + 1);//移除原始的run

                }else {
                    // 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
                    XWPFRun insertNewRun = xWPFParagraph.insertNewRun(endRunIndex);
                    insertNewRun.getCTR().setRPr(endRun.getCTR().getRPr());
                    // 设置文本
                    String textString=endRunText.substring(endRunText.indexOf("}")+1);
                    insertNewRun.setText(textString);
                    xWPFParagraph.removeRun(endRunIndex + 1);//移除原始的run
                }

                //处理中间的run标签
                for (int i = 0; i < removeRunList.size(); i++) {
                    XWPFRun xWPFRun = runs.get(removeRunList.get(i));//原始run
                    XWPFRun insertNewRun = xWPFParagraph.insertNewRun(removeRunList.get(i));
                    insertNewRun.getCTR().setRPr(xWPFRun.getCTR().getRPr());
                    insertNewRun.setText("");
                    xWPFParagraph.removeRun(removeRunList.get(i) + 1);//移除原始的run
                }

            }// 处理${**}被分成多个run

            replaceParagraph( xWPFParagraph, parametersMap);

        }//if 有标签

    }









    /**
     * 复制表格行XWPFTableRow格式
     *
     * @param target
     *            待修改格式的XWPFTableRow
     * @param source
     *            模板XWPFTableRow
     */
    private void CopyTableRow(XWPFTableRow target, XWPFTableRow source) {

        int tempRowCellsize = source.getTableCells().size();// 模板行的列数
        for (int i = 0; i < tempRowCellsize - 1; i++) {
            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
        }
        // 复制样式
        target.getCtRow().setTrPr(source.getCtRow().getTrPr());
        // 复制单元格
        for (int i = 0; i < target.getTableCells().size(); i++) {
            copyTableCell(target.getCell(i), source.getCell(i));
        }
    }









    /**
     * 复制单元格XWPFTableCell格式
     *
     * @author Juveniless
     * @date 2017年11月27日 下午3:41:02
     * @param newTableCell
     *            新创建的的单元格
     * @param templateTableCell
     *            模板单元格
     *
     */
    private void copyTableCell(XWPFTableCell newTableCell, XWPFTableCell templateTableCell) {
        // 列属性
        newTableCell.getCTTc().setTcPr(templateTableCell.getCTTc().getTcPr());
        // 删除目标 targetCell 所有文本段落
        for (int pos = 0; pos < newTableCell.getParagraphs().size(); pos++) {
            newTableCell.removeParagraph(pos);
```

```
        }
        // 添加新文本段落
        for (XWPFParagraph sp : templateTableCell.getParagraphs()) {
            XWPFParagraph targetP = newTableCell.addParagraph();
            copyParagraph(targetP, sp);
        }
    }

    /**
     * 复制文本段落XWPFParagraph格式
     *
     * @author Juveniless
     * @date 2017年11月27日 下午3:43:08
     * @param newParagraph
     *            新创建的的段落
     * @param templateParagraph
     *            模板段落
     *
     */
    private void copyParagraph(XWPFParagraph newParagraph, XWPFParagraph templateParagraph) {
        // 设置段落样式
        newParagraph.getCTP().setPPr(templateParagraph.getCTP().getPPr());
        // 添加Run标签
        for (int pos = 0; pos < newParagraph.getRuns().size(); pos++) {
            newParagraph.removeRun(pos);

        }
        for (XWPFRun s : templateParagraph.getRuns()) {
            XWPFRun targetrun = newParagraph.createRun();
            CopyRun(targetrun, s);
        }

    }

    /**
     * 复制文本节点run
     * @author Juveniless
     * @date 2017年11月27日 下午3:47:17
     * @param newRun
     *            新创建的的文本节点
     * @param templateRun
     *            模板文本节点
     *
     */
    private void CopyRun(XWPFRun newRun, XWPFRun templateRun) {
        newRun.getCTR().setRPr(templateRun.getCTR().getRPr());
        // 设置文本
        newRun.setText(templateRun.text());


    }




    /**
     * 根据参数parametersMap对表格的一行进行标签的替换
     *
     * @author Juveniless
     * @date 2017年11月23日 下午2:09:24
     * @param tableRow
     *            表格行
     * @param parametersMap
     *            参数map
     *
     */
    public void replaceTableRow(XWPFTableRow tableRow, Map<String, Object> parametersMap) {

        List<XWPFTableCell> tableCells = tableRow.getTableCells();
        for (XWPFTableCell xWPFTableCell : tableCells) {
            List<XWPFParagraph> paragraphs = xWPFTableCell.getParagraphs();
            for (XWPFParagraph xwpfParagraph : paragraphs) {

                replaceParagraph(xwpfParagraph, parametersMap);
            }
        }

    }




    /**
     * 根据map替换表格中的{key}标签
     * @author Juveniless
     * @date 2017年12月4日 下午2:47:36
     * @param xwpfTable
     * @param parametersMap
```

```
     *
     */
    public void replaceTable(XWPFTable xwpfTable,Map<String, Object> parametersMap){
        List<XWPFTableRow> rows = xwpfTable.getRows();
        for (XWPFTableRow xWPFTableRow : rows ) {
            List<XWPFTableCell> tableCells = xWPFTableRow.getTableCells();
            for (XWPFTableCell xWPFTableCell : tableCells ) {
                List<XWPFParagraph> paragraphs2 = xWPFTableCell.getParagraphs();
                for (XWPFParagraph xWPFParagraph : paragraphs2) {
                    replaceParagraph(xWPFParagraph, parametersMap);
                }
            }
        }

    }



    private String getValueBykey(String key, Map<String, Object> map) {
        String returnValue="";
        if (key != null) {
            try {
                returnValue=map.get(key)!=null ? map.get(key).toString() : "";
            } catch (Exception e) {
                // TODO: handle exception
                System.out.println("key:"+key+"***"+e);
                returnValue="";
            }

        }
        return returnValue;
    }









}
```

- **use**

```
package com.hidata.tool;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Test {

    public static void main(String[] args) throws IOException {

        Map<String, Object> wordDataMap = new HashMap<String, Object>();// 存储报表全部数据
        Map<String, Object> parametersMap = new HashMap<String, Object>();// 存储报表中不循环的数据



        List<Map<String, Object>> table1 = new ArrayList<Map<String, Object>>();
        Map<String, Object> map1=new HashMap<>();
        map1.put("name", "张三");
        map1.put("age", "23");
        map1.put("email", "12121@qq.com");

        Map<String, Object> map2=new HashMap<>();
        map2.put("name", "李四");
        map2.put("age", "45");
        map2.put("email", "45445@qq.com");

        Map<String, Object> map3=new HashMap<>();
        map3.put("name", "Tom");
        map3.put("age", "34");
        map3.put("email", "6767@qq.com");

        table1.add(map1);
        table1.add(map2);
        table1.add(map3);

```

```
        List<Map<String, Object>> table2 = new ArrayList<Map<String, Object>>();
        Map<String, Object> map4=new HashMap<>();
        map4.put("name", "tom");
        map4.put("number", "sd1234");
        map4.put("address", "上海");

        Map<String, Object> map5=new HashMap<>();
        map5.put("name", "seven");
        map5.put("number", "sd15678");
        map5.put("address", "北京");

        Map<String, Object> map6=new HashMap<>();
        map6.put("name", "lisa");
        map6.put("number", "sd9078");
        map6.put("address", "广州");

        table2.add(map4);
        table2.add(map5);
        table2.add(map6);



        parametersMap.put("userName", "JUVENILESS");
        parametersMap.put("time", "2018-03-24");
        parametersMap.put("sum", "3");


        wordDataMap.put("table1", table1);
        wordDataMap.put("table2", table2);
        wordDataMap.put("parametersMap", parametersMap);
        File file = new File("D:\\Workspaces\\Eclipse 2017\\wordTemplate\\doc\\模板.docx");//改成你本地文件所在目录


        // 读取word模板
        FileInputStream fileInputStream = new FileInputStream(file);
        WordTemplate template = new WordTemplate(fileInputStream);

        // 替换数据
        template.replaceDocument(wordDataMap);


        //生成文件
        File outputFile=new File("D:\\Workspaces\\Eclipse 2017\\wordTemplate\\doc\\输出.docx");//改成你本地文件所在目录
        FileOutputStream fos  = new FileOutputStream(outputFile);
        template.getDocument().write(fos);

    }

}
```