# EasyExcel

1. EasyExcel介绍

EasyExcel是一个基于Java的、快速、简洁、解决大文件内存溢出的Excel处理工具。他能让

你在不用考虑性能、内存的等因素的情况下，快速完成Excel的读、写等功能。EasyExcel

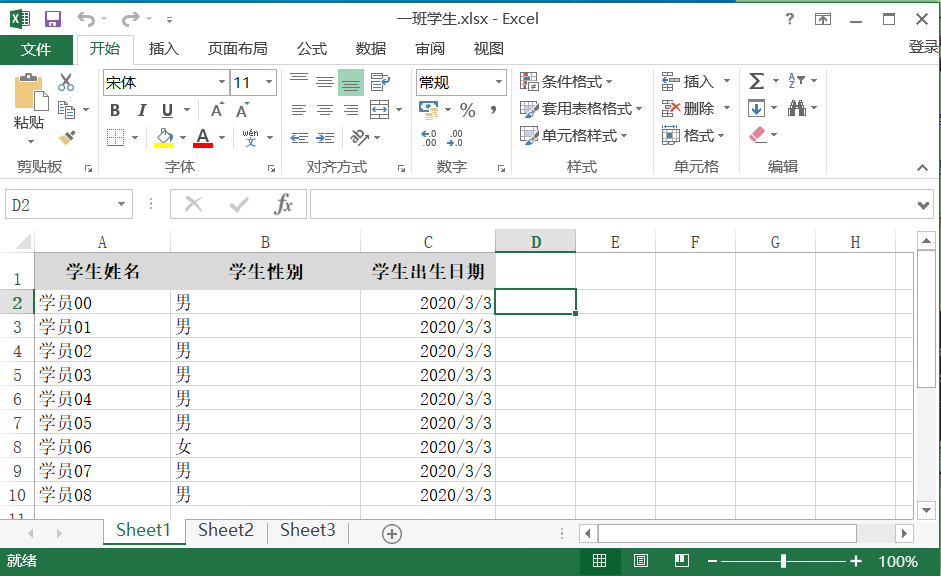
是Alibaba开源的操作excel的框架。

2. EasyExcel官网

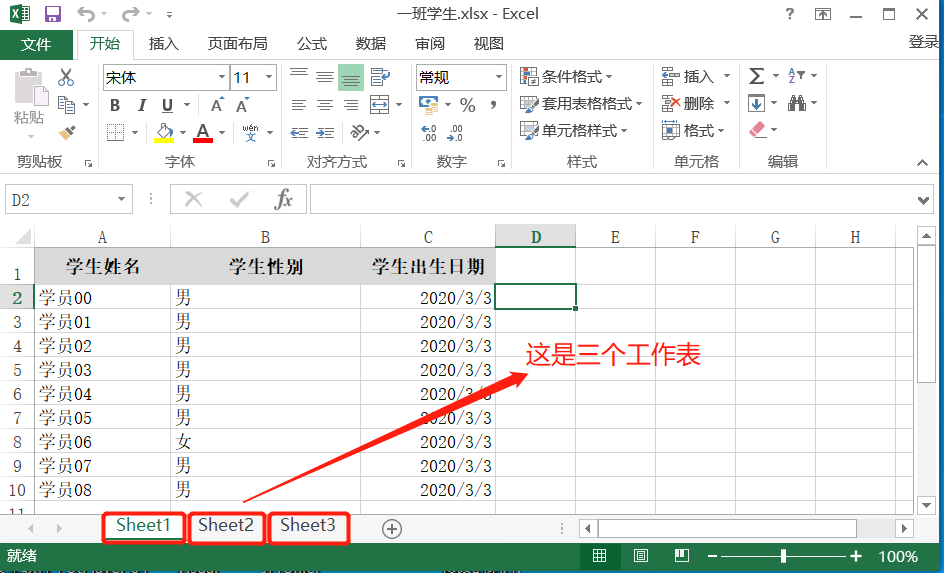
<https://easyexcel.opensource.alibaba.com/>

3.Excel文件结构

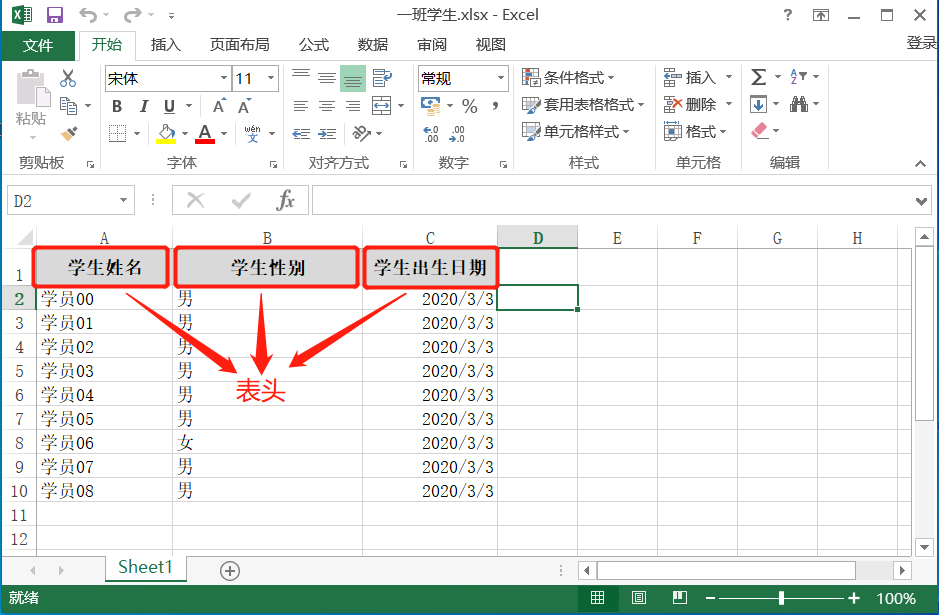
(1)一个Excel文件就是一个工作簿。图解：



(2)一个工作簿中可以有多个工作表(sheet)。图解：



(3)表头



4.使用EasyExcel读\*.xlsx格式的文件

(1)导入maven依赖

<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>easyexcel</artifactId>  
 <version>3.1.1</version>  
</dependency>

注意：这个依赖中包含有其他的jar包，比如：slf4j-api、lombok。

(2)创建一个实体类

@Data  
public class StudentForRead {  
 private String name;  
 private String gender;  
 private Date birthday;  
 private String id;  
}

(3)创建一个监听器类

public class StudentListener extends AnalysisEventListener<StudentForRead> {  
  
 */\*\*  
 \* 针对捕捉到异常的数据触发的函数，抛出异常时会停止读取  
 \** ***@param*** *exception  
 \** ***@param*** *context  
 \** ***@throws*** *Exception  
 \*/* @Override  
 public void onException(Exception exception, AnalysisContext context) throws Exception {  
  
 }  
  
 */\*\*  
 \* 每读一行内容，都会调用一次invoke方法，在invoke方法中可以操作读取到的内容  
 \** ***@param*** *studentForRead 每次读取到的数据封装的对象  
 \** ***@param*** *analysisContext  
 \*/* @Override  
 public void invoke(StudentForRead studentForRead, AnalysisContext analysisContext) {  
 System.*out*.println("读取的内容："+ studentForRead);  
 }  
  
 */\*\*  
 \* 读取第一行表头信息后调用的方法  
 \** ***@param*** *headMap  
 \** ***@param*** *context  
 \*/* @Override  
 public void invokeHeadMap(Map<Integer, String> headMap, AnalysisContext context) {  
 super.invokeHeadMap(headMap, context);  
 }  
  
 */\*\*  
 \* 读取额外信息：批注、超链接、合并单元格信息等  
 \** ***@param*** *extra  
 \** ***@param*** *context  
 \*/* @Override  
 public void extra(CellExtra extra, AnalysisContext context) {  
  
 }  
  
 */\*\*  
 \* 读取完整个工作簿后调用的方法  
 \** ***@param*** *analysisContext  
 \*/* @Override  
 public void doAfterAllAnalysed(AnalysisContext analysisContext) {  
 System.*out*.println("文件已读取完成！");  
 }  
  
 */\*\*  
 \* 返回值表示是否还有下一行数据  
 \** ***@param*** *context  
 \** ***@return*** *\*/* @Override  
 public boolean hasNext(AnalysisContext context) {  
 return true;  
 }  
  
}

(4)读取文件数据

@Test  
public void testRead(){  
 //获取一个工作簿对象，read还有更多重载方法  
 ExcelReaderBuilder workbook = EasyExcel.*read*("一班学生.xlsx", StudentForRead.class, new StudentListener());  
 //获取工作簿中的第一个工作表对象，sheet还有更多重载方法  
 ExcelReaderSheetBuilder sheet = workbook.sheet();  
 //读取工作表中的内容，每读一行，都会调用监听器的invoke方法  
 sheet.doRead();  
}

5.使用EasyExcel写\*.xlsx格式的文件

(1)导入maven依赖

<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>easyexcel</artifactId>  
 <version>3.1.1</version>  
</dependency>

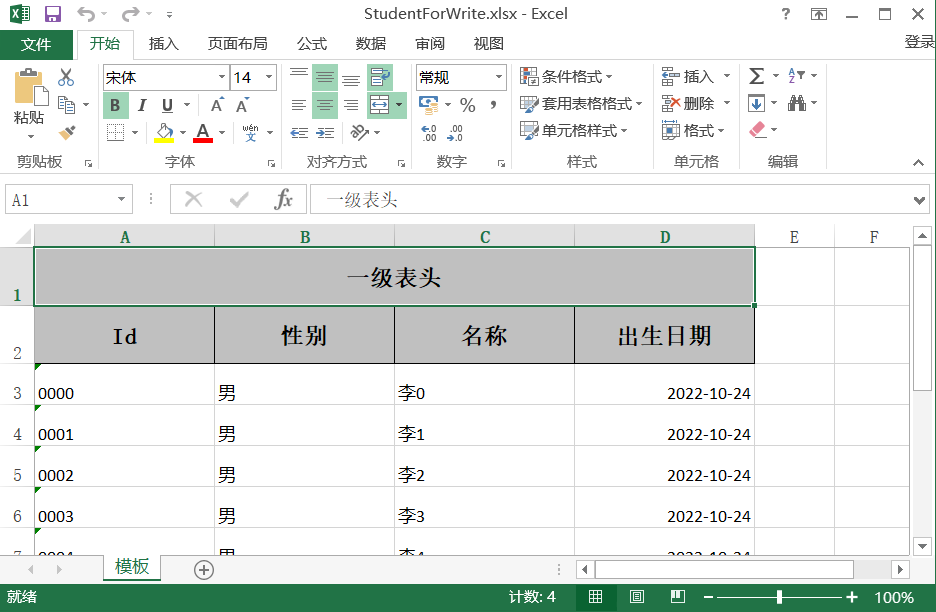
(2)创建一个实体类并使用相关注解

@Data  
@HeadRowHeight(35) //表头行高  
@ContentRowHeight(25) //内容行高  
@ColumnWidth(20) //列宽。也可以修饰属性，指定某列的宽度  
public class StudentForWrite {  
 */\*\*  
 \** ***@ExcelProperty注解用于设置标题和排序*** *\* value的值是一个字符串数组，其第一个值为一级表头，第二个值为二级表头，以此类推  
 \* order的值是int类型，为排序的值  
 \* index(未展示)的值是int类型，指定列的索引，跟order作用相似。  
 \* 不同之处是order没法设置空列，而指定index时，可以指定属性值  
 \* 在哪一列显示。比如：当index=0时，是在第一列展示；index=3时，  
 \* 是在第四列展示。  
 \*/* @ExcelProperty(value = {"一级表头","Id"},order = 1)  
 private String id;  
 @ExcelProperty(value = {"一级表头","名称"},order = 5)  
 private String name;  
 @ExcelProperty(value = {"一级表头","性别"},order = 2)  
 private String gender;  
 @ExcelProperty(value = {"一级表头","出生日期"},order = 4)  
 @DateTimeFormat("yyyy-MM-dd") //指定日期的格式  
 private Date birthday;  
  
 //忽略这个字段  
 @ExcelIgnore  
 private String ignore;  
}

(3)写入文件数据

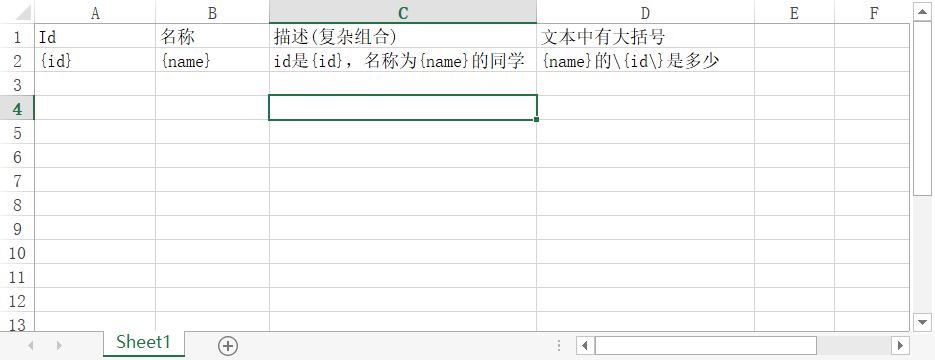
@Test  
public void testWrite(){  
 List<StudentForWrite> list = new ArrayList<>();  
 for(int i = 0;i < 10;i++){  
 StudentForWrite write = new StudentForWrite();  
 write.setId("000"+i);  
 write.setName("李"+i);  
 write.setGender("男");  
 write.setBirthday(new Date());  
 list.add(write);  
 }  
 Set<String> set = new HashSet<>();  
 set.add("id");  
 set.add("name");  
 String fileName = "StudentForWrite.xlsx";  
 //数据写入Excel文件，如果文件已存在，则先清空文件内容，再写入新数据  
 EasyExcel.*write*(fileName, StudentForWrite.class)  
 //表示execl文件只写入实体类中属性名和set集合中的值相同的列  
// .includeColumnFieldNames(set)  
 //(实体类不使用@ColumnWidth指定列宽时)自适应列宽，但不是特别精确  
// .registerWriteHandler(new LongestMatchColumnWidthStyleStrategy())  
 .sheet("模板").doWrite(list);  
}

(4)生成的文件内容展示



6.填充一组数据

(1)准备模板execl文件，预览图。



说明：{}用来包裹需要被替换的变量值，如果文本中本来就有{或者}，需要使用\来转义。

代码中用来填充数据的实体类对象的属性名或Map集合的key需要和{}中被包裹的变量

名一致。

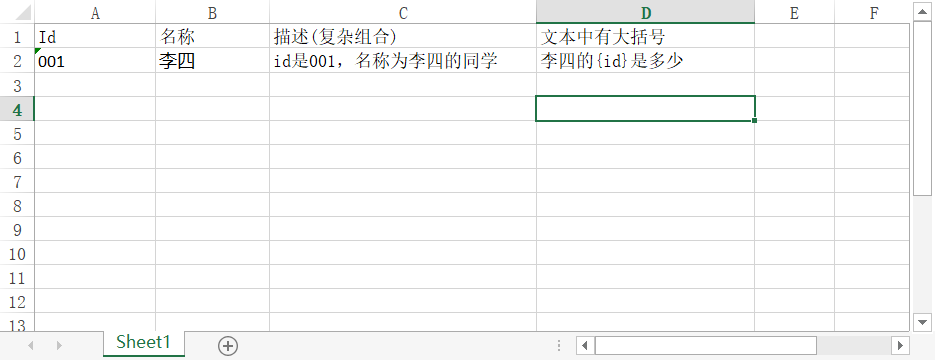
(2)填充数据的实体类(使用Map也可以)

@Data  
public class StudentForFill {  
 private String id;  
 private String name;  
}

(3)填充方法示例

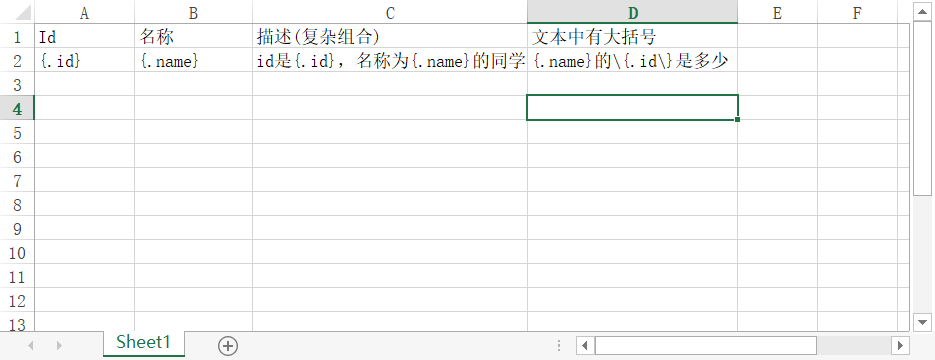
@Test  
public void testFillOneSet(){  
 //准备模板文件  
 String template = "填充一组数据\_模板.xlsx";  
 //准备数据  
 StudentForFill fill = new StudentForFill();  
 fill.setId("001");  
 fill.setName("李四");  
 //创建工作簿对象，withTemplate有更多重载方法  
 ExcelWriterBuilder excelWriterBuilder = EasyExcel.*write*("填充一组数据\_实现.xlsx", StudentForFill.class).withTemplate(template);  
 //创建工作表对象  
 ExcelWriterSheetBuilder sheet = excelWriterBuilder.sheet();  
 //填充数据  
 sheet.doFill(fill);  
}

(4)填充完成的预览图



7.填充多组数据

(1)准备模板execl文件，预览图。

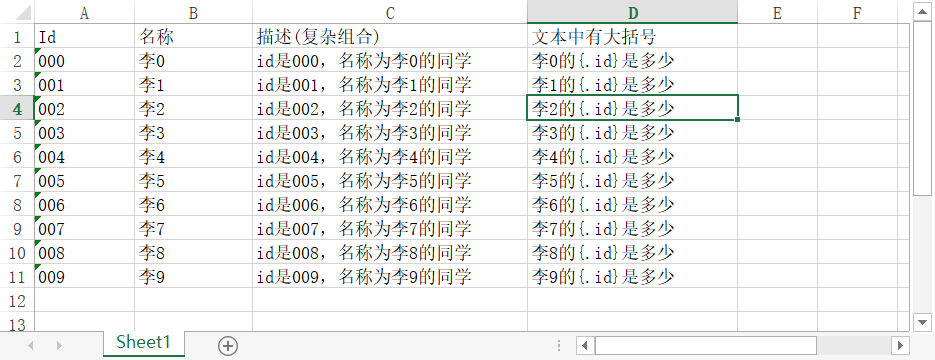


说明：与一组不同的是，在{}中加了.表示为多组数据填充。

(2)填充方法示例

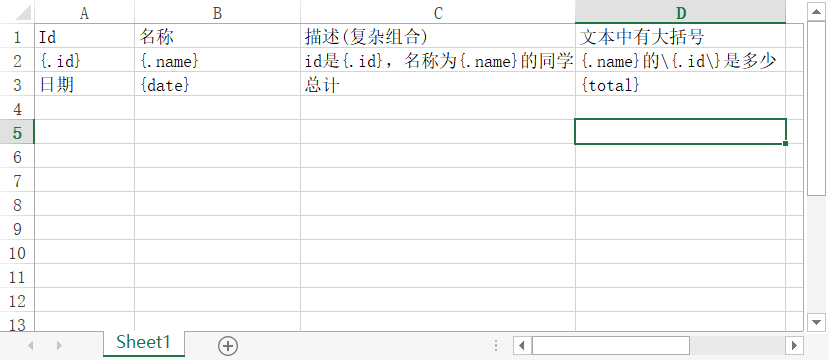
@Test  
public void testFillMultipleSet(){  
 //准备模板文件  
 String template = "填充多组数据\_模板.xlsx";  
 //准备数据  
 List<StudentForFill> list = new ArrayList<>();  
 for(int i = 0;i < 10;i++){  
 StudentForFill fill = new StudentForFill();  
 fill.setId("00"+i);  
 fill.setName("李"+i);  
 list.add(fill);  
 }  
 //创建工作簿对象，withTemplate有更多重载方法  
 ExcelWriterBuilder excelWriterBuilder = EasyExcel.*write*("填充多组数据\_实现.xlsx", StudentForFill.class).withTemplate(template);  
 //创建工作表对象  
 ExcelWriterSheetBuilder sheet = excelWriterBuilder.sheet();  
 //填充数据，doXxx系列方法会在读/写完成后自动关闭流  
 sheet.doFill(list);  
}

(3)填充完成的预览图



8.组合填充，既有一组，也有多组

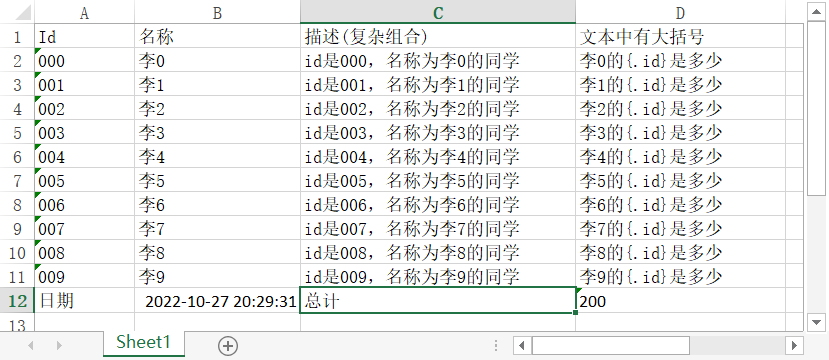
(1)准备模板execl文件，预览图。



(2)填充方法示例

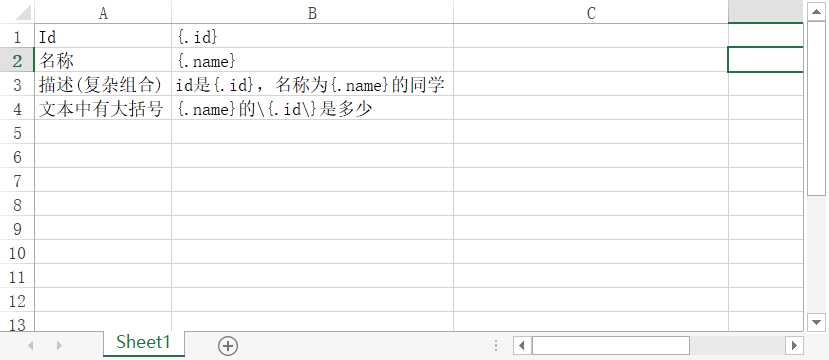
@Test  
public void testFillSet(){  
 //准备模板文件  
 String template = "组合填充数据\_模板.xlsx";  
 //准备数据  
 List<StudentForFill> list = new ArrayList<>();  
 for(int i = 0;i < 10;i++){  
 StudentForFill fill = new StudentForFill();  
 fill.setId("00"+i);  
 fill.setName("李"+i);  
 list.add(fill);  
 }  
 Map<String,Object> data = new HashMap<>();  
 data.put("date",new Date());  
 data.put("total","200");  
 //创建工作簿对象，withTemplate有更多重载方法  
 ExcelWriter workBook = EasyExcel.*write*("组合填充数据\_实现.xlsx", StudentForFill.class).  
 withTemplate(template).build();  
 //创建工作表对象  
 WriteSheet sheet = EasyExcel.*writerSheet*().build();  
 //设置换行，否则在模板文件中，如果多组数据在单组数据的上面，则多组数据会覆盖单组数据  
 FillConfig fillConfig = FillConfig.*builder*().forceNewRow(true).build();  
 //多组填充数据  
 workBook.fill(list,fillConfig,sheet);  
 //单组填充数据  
 workBook.fill(data,sheet);  
 //关闭流  
 workBook.finish();  
}

(3)填充完成的预览图



9.多组水平填充

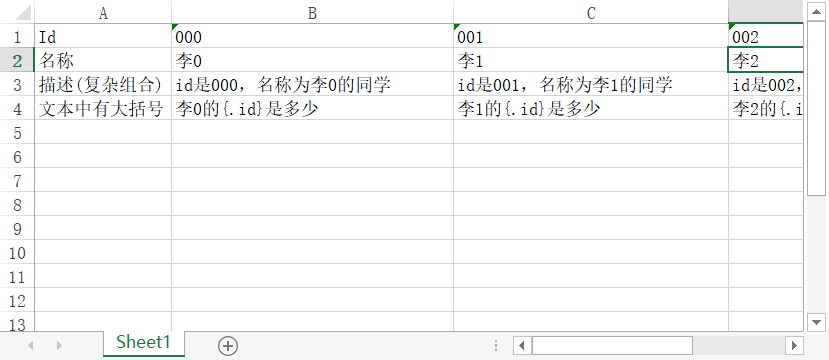
(1)准备模板execl文件，预览图。



(2)填充方法示例

@Test  
public void testFillMultipleSetToHorizontal(){  
 //准备模板文件  
 String template = "多组水平填充\_模板.xlsx";  
 //准备数据  
 List<StudentForFill> list = new ArrayList<>();  
 for(int i = 0;i < 10;i++){  
 StudentForFill fill = new StudentForFill();  
 fill.setId("00"+i);  
 fill.setName("李"+i);  
 list.add(fill);  
 }  
 //创建工作簿对象，withTemplate有更多重载方法  
 ExcelWriter workBook = EasyExcel.*write*("多组水平填充\_实现.xlsx", StudentForFill.class).  
 withTemplate(template).build();  
 //创建工作表对象  
 WriteSheet sheet = EasyExcel.*writerSheet*().build();  
 //设置水平填充  
 FillConfig fillConfig = FillConfig.*builder*().direction(WriteDirectionEnum.*HORIZONTAL*).build();  
 //多组填充数据  
 workBook.fill(list,fillConfig,sheet);  
 //关闭流  
 workBook.finish();  
}

(3)填充完成的预览图



10.补充说明

在EasyExcel官网上还有许多关于读/写的其他功能，包括读表头数据、读额外信息(批注、超

链接)、写数据时指定单元格样式、写图片等。详情可以去官网查看，这里不再一一测

试。