**Java操作Excel**

1. **添加依赖**

<!-- easypoi 核心依赖包 -->

<dependency>

<groupId>cn.afterturn</groupId>

<artifactId>easypoi-base</artifactId>

<version>3.1.0</version>

</dependency>

<dependency>

<groupId>cn.afterturn</groupId>

<artifactId>easypoi-web</artifactId>

<version>3.1.0</version>

</dependency>

<dependency>

<groupId>cn.afterturn</groupId>

<artifactId>easypoi-annotation</artifactId>

<version>3.1.0</version>

</dependency>

<!-- lombok 的依赖 可以省去对象的getter、setter方法-->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.16.20</version>

<scope>provided</scope>

</dependency>

1. **导出**

1）对象定义：

StudentModel类包括学生姓名，学生性别，出生日期以及进校日期：

package com.lamarsan.excel\_demo.model;

import cn.afterturn.easypoi.excel.annotation.Excel;

import cn.afterturn.easypoi.excel.annotation.ExcelTarget;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

import javax.validation.constraints.NotBlank;

import java.io.Serializable;

import java.util.Date;

/\*\*

\* className: StudentModel

\* description: TODO

\*

\* @author hasee

\* @version 1.0

\* @date 2019/7/11 17:43

\*/

@Data

@AllArgsConstructor

@NoArgsConstructor

@ExcelTarget("studentEntity")

public class StudentModel implements Serializable {

/\*\*

\* id

\*/

private String id;

/\*\*

\* 学生姓名

\*/

@Excel(name = "学生姓名", height = 20, width = 30, isImportField = "true\_st")

private String name;

/\*\*

\* 学生性别

\*/

@Excel(name = "学生性别", replace = {"男\_1", "女\_2"}, suffix = "生", isImportField = "true\_st")

private int sex;

@Excel(name = "出生日期", databaseFormat = "yyyyMMddHHmmss", format = "yyyy-MM-dd", isImportField = "true\_st", width = 20)

private Date birthday;

@Excel(name = "进校日期", databaseFormat = "yyyyMMddHHmmss", format = "yyyy-MM-dd")

private Date registrationDate;

}

2）导出。使用构造函数构造相应的几个数据后，编写导出语句。

Workbook workbook = ExcelExportUtil.exportExcel(new ExportParams("2412312", "测试"),CourseModel.class, courseModels);

1. 调用接口。下载后的excel表格如下。



图1 导出结果图

1. **导入**

1）导入。编写导入语句。

@PostMapping(value = "/excelImport")

public Object importExcel(@RequestParam("file") MultipartFile file) {

//接收导入数组

List<StudentModel> studentModels = null;

try {

studentModels = ExcelImportUtil.importExcel(file.getInputStream(), StudentModel.class, new ImportParams());

} catch (Exception e) {

e.printStackTrace();

}

return studentModels;

}

1. 调用接口。使用postman调用接口localhost:8080/excelReader/excelImpor

t，并去掉表格中的表格头，在body的formdata中选择file，上传刚刚导出的文件，key填写file，得到返回结果如下。



图2 导入结果图

1. **合并单元格导出**

1）对象定义：

课程实体类CourseModel：

package com.lamarsan.excel\_demo.model;

import cn.afterturn.easypoi.excel.annotation.Excel;

import cn.afterturn.easypoi.excel.annotation.ExcelCollection;

import cn.afterturn.easypoi.excel.annotation.ExcelEntity;

import cn.afterturn.easypoi.excel.annotation.ExcelTarget;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

import java.io.Serializable;

import java.util.List;

/\*\*

\* className: CourseModel

\* description: TODO

\*

\* @author hasee

\* @version 1.0

\* @date 2019/7/11 17:53

\*/

@Data

@ExcelTarget("courseEntity")

@NoArgsConstructor

@AllArgsConstructor

public class CourseModel implements Serializable {

/\*\*

\* 主键

\*/

private String id;

/\*\*

\* 课程名称

\*/

@Excel(name = "课程名称", orderNum = "1", width = 25)

private String name;

/\*\*

\* 老师主键

\*/

@ExcelEntity(id = "absent")

private TeacherModel mathTeacher;

@ExcelCollection(name = "学生", orderNum = "4")

private List<StudentModel> students;

}

教师实体类TeacherModel：

package com.lamarsan.excel\_demo.model;

import cn.afterturn.easypoi.excel.annotation.Excel;

import cn.afterturn.easypoi.excel.annotation.ExcelTarget;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

import java.io.Serializable;

/\*\*

\* className: TeacherModel

\* description: TODO

\*

\* @author hasee

\* @version 1.0

\* @date 2019/7/11 17:55

\*/

@Data

@ExcelTarget("teacherEntity")

@AllArgsConstructor

@NoArgsConstructor

public class TeacherModel implements Serializable {

private String id;

/\*\* name \*/

@Excel(name = "主讲老师\_major,代课老师\_absent", orderNum = "1", isImportField = "true\_major,true\_absent")

private String name;

}

2）导出：

使用构造函数构造相应的几个数据后，编写导出语句。

Workbook workbook = ExcelExportUtil.exportExcel(new ExportParams("2412312", "测试"),CourseModel.class, courseModels);

1. 导出结果：

导出结果如图所示：



图3 课程导出结果图

1. 优化与合并：

可以发现，空着并不是很好看，可以合并单元格，进行美化操作。

@Excel(name = "课程名称", orderNum = "1",needMerge = true, width = 25)

private String name;

@Excel(name = "主讲老师\_major,代课老师\_absent",needMerge = true, orderNum = "1", isImportField = "true\_major,true\_absent")

private String name;

1. 结果：



图4 课程导出结果图

1. **多sheet导出**
2. 定义基础配置对象。

导出基本采用ExportParams 这个对象，进行参数配置；

我们需要进行多Sheet导出，那么就需要定义一个基础配置对象。

package com.lamarsan.excel\_demo.common;

import cn.afterturn.easypoi.excel.entity.ExportParams;

import java.util.List;

/\*\*

\* className: ExportView

\* description: TODO

\*

\* @author hasee

\* @version 1.0

\* @date 2019/7/11 18:56

\*/

public class ExportView {

public ExportView() {

}

private ExportParams exportParams;

private List<?> dataList;

private Class<?> cls;

public ExportParams getExportParams() {

return exportParams;

}

public void setExportParams(ExportParams exportParams) {

this.exportParams = exportParams;

}

public Class<?> getCls() {

return cls;

}

public void setCls(Class<?> cls) {

this.cls = cls;

}

public List<?> getDataList() {

return dataList;

}

public void setDataList(List<?> dataList) {

this.dataList = dataList;

}

public ExportView(Builder builder) {

this.exportParams = builder.exportParams;

this.dataList = builder.dataList;

this.cls = builder.cls;

}

public static class Builder {

private ExportParams exportParams = null;

private List<?> dataList = null;

private Class<?> cls = null;

public Builder() {

}

public Builder exportParams(ExportParams exportParams) {

this.exportParams = exportParams;

return this;

}

public Builder dataList(List<?> dataList) {

this.dataList = dataList;

return this;

}

public Builder cls(Class<?> cls) {

this.cls = cls;

return this;

}

public ExportView create() {

return new ExportView(this);

}

}

}

1. 最后在实现调用的方法中，对整个集合进行配置和解析。

List<Map<String, Object>> exportParamList = Lists.newArrayList();

ExportView studentView = new ExportView(new ExportParams("学生表","表1",XSSF), studentModelList, StudentModel.class);

ExportView courseView = new ExportView(new ExportParams("课程表","表2",XSSF), courseModelList, CourseModel.class);

List<ExportView> exportViews = new ArrayList<>();

//导入studentlist

exportViews.add(studentView);

//导入courselist

exportViews.add(courseView);

for (ExportView view : exportViews) {

Map<String, Object> valueMap = Maps.newHashMap();

valueMap.put("title", view.getExportParams());

valueMap.put("data", view.getDataList());

valueMap.put("entity", view.getCls());

exportParamList.add(valueMap);

}

// 执行方法

Workbook workBook = ExcelExportUtil.exportExcel(exportParamList, XSSF);

ExcelUtil.downloadExcel(response, workBook, "计算机二班选课情况");

1. 导出结果：



图5 sheet1结果图

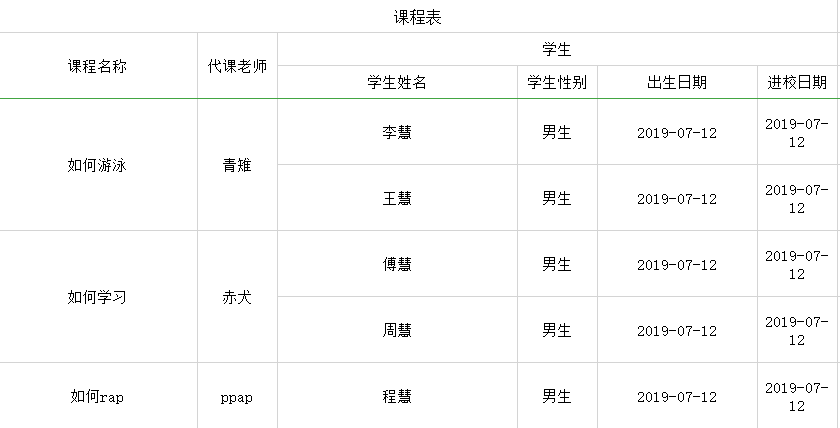


图6 sheet2结果图

1. **导出图片到Excel**
2. 说明：

从本小节开始，将脱离easyExcel工具，转而使用POI工具。

1. 传入图片IO流，如下所示：

// 先把读进来的图片放到一个ByteArrayOutputStream中，以便产生ByteArray

ByteArrayOutputStream byteArrayOut = new ByteArrayOutputStream();

ByteArrayOutputStream byteArrayOut2 = new ByteArrayOutputStream();

ByteArrayOutputStream byteArrayOut3 = new ByteArrayOutputStream();

//将图片读到BufferedImage

bufferImg = ImageIO.read(new File("D:\\图片\\140.png"));

bufferImg2 = ImageIO.read(new File("D:\\图片\\137.png"));

bufferImg3 = ImageIO.read(new File("D:\\图片\\139.png"));

// 将图片写入流中

ImageIO.write(bufferImg, "png", byteArrayOut);

ImageIO.write(bufferImg2, "png", byteArrayOut2);

ImageIO.write(bufferImg3, "png", byteArrayOut3);

1. 利用Drawing将图片写入EXCEL，如下所示：

Drawing patriarch = sheet.createDrawingPatriarch();

/\*\*

\* 该构造函数有8个参数

\* 前四个参数是控制图片在单元格的位置，分别是图片距离单元格left，top，right，bottom的像素距离

\* 后四个参数，前两个表示图片左上角所在的cellNum和 rowNum，后两个参数对应的表示图片右下角所在的cellNum和 rowNum，

\* excel中的cellNum和rowNum的index都是从0开始的

\*

\*/

//图片一导出到单元格B2中

HSSFClientAnchor anchor = new HSSFClientAnchor(0, 0, 0, 0,(short) 7, 4, (short) 8, 5);

HSSFClientAnchor anchor2 = new HSSFClientAnchor(0, 0, 0, 0,(short) 7, 5, (short) 8, 6);

HSSFClientAnchor anchor3 = new HSSFClientAnchor(0, 0, 0, 0,(short) 7, 6, (short) 8, 7);

// 插入图片

patriarch.createPicture(anchor, workBook.addPicture(byteArrayOut.toByteArray(), HSSFWorkbook.PICTURE\_TYPE\_JPEG));

patriarch.createPicture(anchor2, workBook.addPicture(byteArrayOut2.toByteArray(), HSSFWorkbook.PICTURE\_TYPE\_JPEG));

patriarch.createPicture(anchor3, workBook.addPicture(byteArrayOut3.toByteArray(), HSSFWorkbook.PICTURE\_TYPE\_JPEG));

1. 最终效果如下图所示：



图7 导入结果图

1. **多EXCEL打包**
2. 编写文件压缩工具类如下，编写srcFiles与zipFile的信息：

package com.lamarsan.excel\_demo.utils;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.util.zip.ZipEntry;

import java.util.zip.ZipOutputStream;

/\*\*

\* className: ZipMultiFile

\* description: TODO

\*

\* @author hasee

\* @version 1.0

\* @date 2019/7/12 17:46

\*/

public class ZipMultiFile {

public static void main(String[] args) {

File[] srcFiles = { new File("D:\\记事本\\公司\\下载2.xls"), new File("D:\\记事本\\公司\\标书递交\_投标文件递交记录表（非物资）.xls") };

File zipFile = new File("D:\\记事本\\公司\\ZipFile.zip");

// 调用压缩方法

zipFiles(srcFiles, zipFile);

}

public static void zipFiles(File[] srcFiles, File zipFile) {

// 判断压缩后的文件存在不，不存在则创建

if (!zipFile.exists()) {

try {

zipFile.createNewFile();

} catch (IOException e) {

e.printStackTrace();

}

}

// 创建 FileOutputStream 对象

FileOutputStream fileOutputStream = null;

// 创建 ZipOutputStream

ZipOutputStream zipOutputStream = null;

// 创建 FileInputStream 对象

FileInputStream fileInputStream = null;

try {

// 实例化 FileOutputStream 对象

fileOutputStream = new FileOutputStream(zipFile);

// 实例化 ZipOutputStream 对象

zipOutputStream = new ZipOutputStream(fileOutputStream);

// 创建 ZipEntry 对象

ZipEntry zipEntry = null;

// 遍历源文件数组

for (int i = 0; i < srcFiles.length; i++) {

// 将源文件数组中的当前文件读入 FileInputStream 流中

fileInputStream = new FileInputStream(srcFiles[i]);

// 实例化 ZipEntry 对象，源文件数组中的当前文件

zipEntry = new ZipEntry(srcFiles[i].getName());

zipOutputStream.putNextEntry(zipEntry);

// 该变量记录每次真正读的字节个数

int len;

// 定义每次读取的字节数组

byte[] buffer = new byte[1024];

while ((len = fileInputStream.read(buffer)) > 0) {

zipOutputStream.write(buffer, 0, len);

}

}

zipOutputStream.closeEntry();

zipOutputStream.close();

fileInputStream.close();

fileOutputStream.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

1. 压缩结果如图所示：



图8 压缩结果图

1. **Zip包解压，获取excel文件并进行分析**

1）编写接口如下，主要思路是解压zip，获取每个文件的io流，然后创建工作簿类获取到每个类的cell信息：

@PostMapping(value = "/importZip")  
private Object importZip(@RequestParam("file") MultipartFile zipFile) {  
 //获得文件名  
 String fileName = zipFile.getOriginalFilename();  
 //检查文件  
 if ("".equals(fileName)) {  
 System.out.println("文件为空");  
 }  
 List<List<PoiModel>> poiModelLists = new ArrayList<>();  
 try {  
 //再本地创建一个文件，读取此文件 防止浏览器读取的文件被损坏  
 File localFile = new File("D:\\记事本\\公司\\fyJyqdYhqdxxZip.zip");  
 FileOutputStream ftpOutstream = new FileOutputStream(localFile);  
 byte[] appByte = zipFile.getBytes();  
 ftpOutstream.write(appByte);  
 ftpOutstream.flush();  
 ftpOutstream.close();//创建完毕后删除  
  
 File file = new File("D:\\记事本\\公司\\fyJyqdYhqdxxZip.zip");  
 //不解压直接读取,加上UTF-8解决乱码问题,file转ZipInputStream  
 ZipInputStream in = new ZipInputStream(new FileInputStream(file), Charset.forName("GBK"));  
 //不解压直接读取,加上UTF-8解决乱码问题,ZipInputStream转BufferedReader  
 BufferedReader br = new BufferedReader(new InputStreamReader(in, "gbk"));  
 //把InputStream转成ByteArrayOutputStream 多次使用  
 ByteArrayOutputStream baos = new ByteArrayOutputStream();  
 ZipEntry ze;  
 while ((ze = in.getNextEntry()) != null) {  
 if (ze.isDirectory()) {  
 //如果是目录，不处理  
 continue;  
 }  
 try {  
 String zipFileName = ze.getName();  
 //不是我们指定的文件不导入，XXXXX.市场化清单.xls  
 //if (zipFileName != null && zipFileName.indexOf(".") != -1  
 // && zipFileName.equals(zipFileName.substring(0, zipFileName.indexOf(".xls")) + "市场化清单.xls")) {  
 // continue;  
 //}  
 byte[] buffer = new byte[1024];  
 int len;  
 while ((len = in.read(buffer)) > -1) {  
 baos.write(buffer, 0, len);  
 }  
 baos.flush();  
  
 InputStream stream = new ByteArrayInputStream(baos.toByteArray());  
 //获取Excel对象  
 HSSFWorkbook wb = new HSSFWorkbook(stream);  
 int sheets = wb.getNumberOfSheets();  
 for (int i = 0; i < sheets; i++) {  
 HSSFSheet sheet = wb.getSheetAt(i);  
 // 获取多少行  
 List<PoiModel> poiModels = new ArrayList<>();  
 int rows = sheet.getPhysicalNumberOfRows();  
 for (int j = 0; j < rows; j++) {  
 //获取Row对象  
 HSSFRow row = sheet.getRow(j);  
 //获取Cell对象的值并输出  
 PoiModel poiModel = new PoiModel(row.getCell(0).toString(), row.getCell(1).toString(), row.getCell(2).toString(), row.getCell(3).toString(), row.getCell(4).toString(), row.getCell(5).toString(), row.getCell(6).toString(), row.getCell(7).toString(), row.getCell(8).toString(), row.getCell(9).toString(), row.getCell(10).toString());  
 System.out.println(row.getCell(0) + " " + row.getCell(1));  
 poiModels.add(poiModel);  
 }  
 poiModelLists.add(poiModels);  
 }  
 baos.reset();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 br.close();  
 in.close();  
 baos.close();  
 //处理完毕删除  
 localFile.delete();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return poiModelLists;  
}

2）返回结果如下：

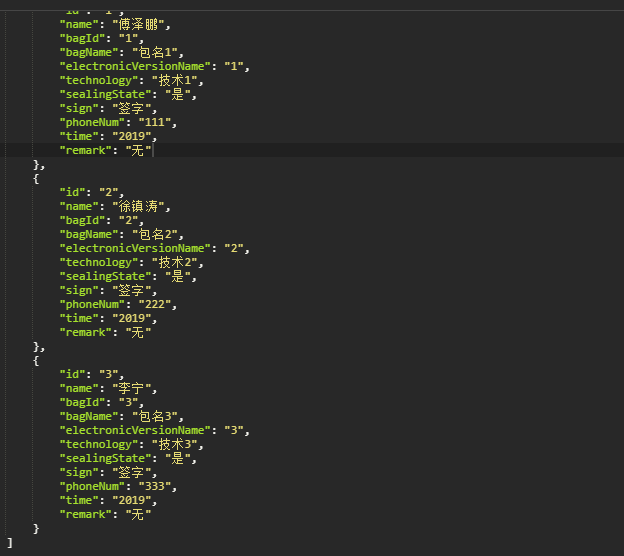


图9 解析结果图