**POI和easyExcel讲解(一个小时!)**

**常见使用场景：**

1、将用户信息导出为excel表格



1. 将Excel表中的信息录入到网站数据库(习题上传)，大大减轻网站录入量



开发中经常会设计到excel的处理，如导出Excel，导入Excel到数据库中!操作Excel目前比较流行的就是Apache POI和阿里巴巴的easyExcel !

**POI**

Apache POl是Apache软件基金会的开放源码函式库，POI提供API给Java程序对Microsoft Ofice格式档案读和写的功能。

基本功能

HSSF一提供读写Microsoft Exce格式档案的功能。 03版本，存储上限是65535

XSSF一提供读写Microsoft Excel OOXML各式档案的功能。07版本，存储无上限

HWPF 一提供读写Microso Wword各式档案的功能。

HSLF一提供读写Microsof PowerPoint格式档案的功能。

HDGF一提供读写Microsoft visio格式档案的功能。

缺点：量比较大的时候会报OOM异常。

**EasyExcel**

JAVA解析Excel工具EasyExcel

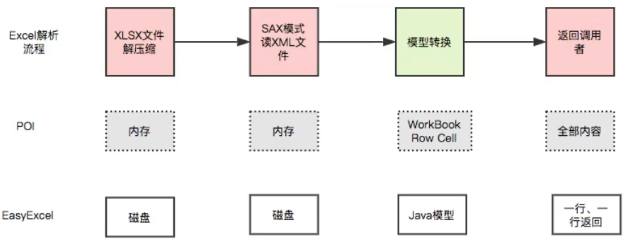
Java解析、生成Excel比较有名的框架有Apache poi、jxl。但他们都存在一个严重的问题就是非常的耗内存，poi有一套SAX模式的API可以一定程度的解决一些内存溢出的问题，但POlI还是有一些缺陷，比如07版Excel解压缩以及解压后存储都是在内存中完成的，内存消耗依然很大。easyexcel重写了poi对07版Excel的解析，能够原本一个3M的excel用POl sax依然需要100M左右内存降低到几M，并且再大的excel不会出现内存溢出，03版依赖POlI的sax模式。在上层做了模型转换的封装，让使用者更加简单方便。

EasyExcel是阿里巴巴开源的一个excel处理框架，**以使用简单、节省内存著称.**

EasyExcel能大大减少占用内存的主要原因是在解析Excel时没有将文件数据一次性全部加载到内存中，而是从磁盘上一行行读取数据，逐个解析。

内存问题:POI = 100w先加载到内存OOM，在写入文件EasyExcel=每次读取一行

下图是EasyExcel和POI在解析Excel时的对比图。

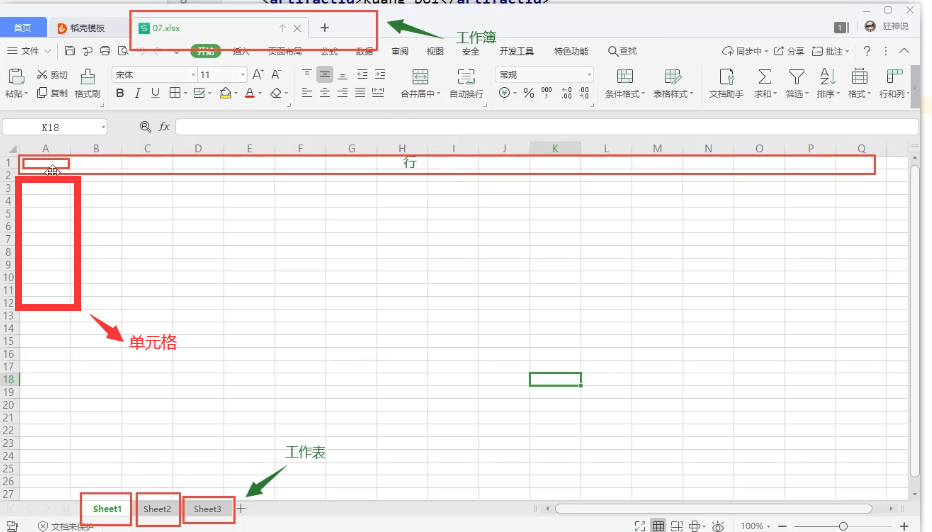


**POI使用**

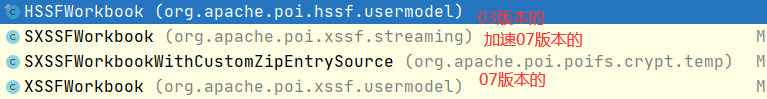
导入依赖：

<!-- 03版本 -->  
<dependency>  
 <groupId>org.apache.poi</groupId>  
 <artifactId>poi</artifactId>  
 <version>4.1.2</version>  
</dependency>  
  
<!-- 07版本 -->  
<dependency>  
 <groupId>org.apache.poi</groupId>  
 <artifactId>poi-ooxml</artifactId>  
 <version>4.1.2</version>  
</dependency>

对Excel有个初步了解：因为需要将其拆分成对象



创建对象是Workbook，往下实现类有以下几类：



具体代码实现：

package com.zsp;  
  
import org.apache.poi.hssf.usermodel.HSSFWorkbook;  
import org.apache.poi.ss.usermodel.Cell;  
import org.apache.poi.ss.usermodel.Row;  
import org.apache.poi.ss.usermodel.Sheet;  
import org.apache.poi.ss.usermodel.Workbook;  
import org.junit.Test;  
  
import java.io.FileNotFoundException;  
import java.io.FileOutputStream;  
  
public class ExcelWriteTest {  
 String PATH="D:\\java项目\\POI\\study-poi";  
 @Test  
 public void testWrite() throws Exception {  
// 1、创建一个工作簿  
 Workbook workbook= new HSSFWorkbook();  
// 2、创造一个工作表  
 Sheet sheet = workbook.createSheet("首个Excel文件");  
// 3、创造一行  
 Row row1 = sheet.createRow(0);  
// 4、在一行里创造一个单元格  
 Cell cell00 = row1.createCell(0);  
// 5、往格子里塞数据  
 cell00.setCellValue("今日学习时间：");  
 Cell cell01 = row1.createCell(1);  
 cell01.setCellValue("2小时");  
// 创建一个流用于创建Excel表  
 FileOutputStream fileOutputStream = new FileOutputStream(PATH + "今日学习时间03.xls");  
// 使用API将文件写入  
 workbook.write(fileOutputStream);  
 fileOutputStream.close();  
 System.out.println("表生成完毕了");  
 }  
}

批量处理：

大文件写HSSF 03

缺点︰最多只能处理65536行，否则会抛出异常

java.1ang.111ega1ArgumentException: Invalid row number (65536) outside allowable range (0..65535)

优点∶过程中写入缓存，不操作磁盘，最后一次性写入磁盘，速度快

public void writeLots03() throws Exception {  
 long start = System.currentTimeMillis();  
 Workbook workbook =new HSSFWorkbook();  
 Sheet sheet = workbook.createSheet();  
 for (int rowNum = 0; rowNum <65536 ; rowNum++) {  
 Row row = sheet.createRow(rowNum);  
 for (int cellNum = 0; cellNum <10 ; cellNum++) {  
 Cell cell = row.createCell(cellNum);  
 cell.setCellValue(cellNum);  
 }  
 }  
 FileOutputStream fileOutputStream = new FileOutputStream(PATH + "03批量写入.xls");  
 workbook.write(fileOutputStream);  
 fileOutputStream.close();  
 long end = System.currentTimeMillis();  
 System.out.println("花费时间为"+((double)(end-start)/1000)+"秒");  
}

大文件写XSSF 07

缺点∶写数据时速度非常慢，非常耗内存，也会发生内存溢出，如100万条优点∶可以写较大的数据量，如20万条

public void writeLots07() throws Exception {  
 long start = System.currentTimeMillis();  
 Workbook workbook =new XSSFWorkbook();  
 Sheet sheet = workbook.createSheet();  
 for (int rowNum = 0; rowNum <100000 ; rowNum++) {  
 Row row = sheet.createRow(rowNum);  
 for (int cellNum = 0; cellNum <10 ; cellNum++) {  
 Cell cell = row.createCell(cellNum);  
 cell.setCellValue(cellNum);  
 }  
 }  
 FileOutputStream fileOutputStream = new FileOutputStream(PATH + "07批量写入.xls");  
 workbook.write(fileOutputStream);  
 fileOutputStream.close();  
 long end = System.currentTimeMillis();  
 System.out.println("花费时间为"+((double)(end-start)/1000)+"秒");  
}

大文件写SXSSF 07加速版 （注意：这里需要关闭临时文件）

优点∶可以写非常大的数据量，如100万条甚至更多条，写数据速度快，占用更少的内存

注意∶

过程中会产生临时文件，需要清理临时文件

默认由100条记录被保存在内存中，如果超过这数量，则最前面的数据被写入临时文件如果想自定义内存中数据的数量，可以使用new SXSSFWorkbook(数量)

SXSSFWorkbook-来至官方的解释∶实现"BigGridDem"策略的流式XSSFWorkbook版本。这允许写入非常大的文件而不会耗尽内存，因为任何时候只有可配置的行部分被保存在内存中。

请注意，仍然可能会消耗大量内存，这些内存基于您正在使用的功能，例如合并区域，注和..…….当然只存储在内存中，因此如果广泛使用，可能需要大量内存。

public void writeLots07S() throws Exception {  
 long start = System.currentTimeMillis();  
 Workbook workbook =new SXSSFWorkbook();  
 Sheet sheet = workbook.createSheet();  
 for (int rowNum = 0; rowNum <100000 ; rowNum++) {  
 Row row = sheet.createRow(rowNum);  
 for (int cellNum = 0; cellNum <10 ; cellNum++) {  
 Cell cell = row.createCell(cellNum);  
 cell.setCellValue(cellNum);  
 }  
 }  
 FileOutputStream fileOutputStream = new FileOutputStream(PATH + "07S批量写入.xls");  
 workbook.write(fileOutputStream);  
 fileOutputStream.close();  
 ((SXSSFWorkbook)workbook).dispose(); //关闭临时文件  
 long end = System.currentTimeMillis();  
 System.out.println("花费时间为"+((double)(end-start)/1000)+"秒");  
}

读取文件：注意不论是什么版本都需要区分对象类型

03版本：

String PATH="D:\\java项目\\POI\\";  
@Test  
public void testRead03() throws Exception {  
 FileInputStream fileInputStream = new FileInputStream(PATH + "今日学习时间03.xls");  
 Workbook workbook=new HSSFWorkbook(fileInputStream);  
 Sheet sheetAt = workbook.getSheetAt(0);  
 Row row0 = sheetAt.getRow(0);  
 Cell cell00 = row0.getCell(0);  
 Cell cell01 = row0.getCell(1);  
 System.out.println(cell00.getStringCellValue());  
 System.out.println(cell01.getStringCellValue());  
 fileInputStream.close();}

07版本：注意不论是什么版本都需要区分对象类型

public void testRead07() throws Exception {  
 FileInputStream fileInputStream = new FileInputStream(PATH + "今日学习时间07.xlsx");  
 Workbook workbook=new XSSFWorkbook(fileInputStream);  
 Sheet sheetAt = workbook.getSheetAt(0);  
 Row row0 = sheetAt.getRow(0);  
 Cell cell00 = row0.getCell(0);  
 Cell cell01 = row0.getCell(1);  
 System.out.println(cell00.getStringCellValue());  
 System.out.println(cell01.getNumericCellValue()); //注意这里需要区分类型  
 fileInputStream.close();  
}

工具类：直接遍历整个列表

package com.zsp;  
  
import org.apache.poi.ss.usermodel.\*;  
import org.apache.poi.xssf.streaming.SXSSFWorkbook;  
import org.apache.poi.xssf.usermodel.XSSFWorkbook;  
import org.joda.time.DateTime;  
  
import java.io.FileInputStream;  
import java.io.FileOutputStream;  
import java.util.Date;  
  
import static org.apache.poi.ss.usermodel.CellType.STRING;  
  
public class POIUtils {  
 public static void POIUtils(String PATH) throws Exception {  
  
 FileInputStream fileInputStream = new FileInputStream(PATH);  
 Workbook workbook = new XSSFWorkbook(fileInputStream);  
 Sheet sheetAt = workbook.getSheetAt(0);  
 int physicalNumberOfRows = sheetAt.getPhysicalNumberOfRows();  
 for (int rowNum = 0; rowNum < physicalNumberOfRows; rowNum++) {  
 Row row = sheetAt.getRow(rowNum);  
 int physicalNumberOfCells = row.getPhysicalNumberOfCells();  
 for (int cellNum = 0; cellNum < physicalNumberOfCells; cellNum++) {  
 Cell cell = row.getCell(cellNum);  
 if (cell != null) {  
 CellType cellType = cell.getCellType();  
 String cellValue = "";  
 switch (cellType) {  
 case NUMERIC:  
 if (DateUtil.isCellDateFormatted(cell)) {  
 Date date = cell.getDateCellValue();  
 cellValue = new DateTime(date).toString("yyyy-MM-dd");  
 } else {  
 cellType = STRING;  
 cellValue = cell.toString();  
 }  
 System.out.print(cellValue + " ");  
 break;  
 case STRING:  
 cellValue = cell.getStringCellValue();  
 System.out.print(cellValue + " ");  
 break;  
 case FORMULA:  
 break;  
 case BLANK:  
 System.out.print("输出为空~");  
 break;  
 case BOOLEAN:  
 cellValue = String.valueOf(cell.getBooleanCellValue());  
 System.out.print(cellValue + " ");  
 break;  
 case ERROR:  
 break;  
 }  
 }  
 }  
 System.out.println();  
 }  
 fileInputStream.close();  
 }  
}

获取公式并且输出值：

public void testFormula() throws Exception {  
 FileInputStream fileInputStream=new FileInputStream(PATH+"公式学习.xls");  
 //确定要判断的单元格  
 Workbook workbook =new HSSFWorkbook(fileInputStream);  
 Sheet sheetAt = workbook.getSheetAt(0);  
 Row row = sheetAt.getRow(0);  
 Cell cell = row.getCell(4);  
 //判断类型  
 CellType cellType = cell.getCellType();  
 //获取计算公式  
 FormulaEvaluator formulaEvaluator = new HSSFFormulaEvaluator((HSSFWorkbook) workbook);  
 switch (cellType){  
 case FORMULA:  
 String cellFormula = cell.getCellFormula();  
 System.out.println(cellFormula);  
 CellValue evaluate = formulaEvaluator.evaluate(cell);  
 String value = evaluate.formatAsString();  
 System.out.println(value);  
 break;  
 }  
 fileInputStream.close();  
 }  
}

简化成工具类：

public class POIUtils {  
 public static void POIUtils(String PATH) throws Exception {  
  
 FileInputStream fileInputStream = new FileInputStream(PATH);  
 Workbook workbook = new XSSFWorkbook(fileInputStream);  
 Sheet sheetAt = workbook.getSheetAt(0);  
 int physicalNumberOfRows = sheetAt.getPhysicalNumberOfRows();  
 for (int rowNum = 0; rowNum < physicalNumberOfRows; rowNum++) {  
 Row row = sheetAt.getRow(rowNum);  
 int physicalNumberOfCells = row.getPhysicalNumberOfCells();  
 for (int cellNum = 0; cellNum < physicalNumberOfCells; cellNum++) {  
 Cell cell = row.getCell(cellNum);  
 if (cell != null) {  
 CellType cellType = cell.getCellType();  
 String cellValue = "";  
 switch (cellType) {  
 case NUMERIC:  
 if (DateUtil.isCellDateFormatted(cell)) {  
 Date date = cell.getDateCellValue();  
 cellValue = new DateTime(date).toString("yyyy-MM-dd");  
 } else {  
 cellType = STRING;  
 cellValue = cell.toString();  
 }  
 System.out.print(cellValue + " ");  
 break;  
 case STRING:  
 cellValue = cell.getStringCellValue();  
 System.out.print(cellValue + " ");  
 break;  
 case FORMULA:  
 break;  
 case BLANK:  
 System.out.print("输出为空~");  
 break;  
 case BOOLEAN:  
 cellValue = String.valueOf(cell.getBooleanCellValue());  
 System.out.print(cellValue + " ");  
 break;  
 case ERROR:  
 break;  
 }  
 }  
 }  
 System.out.println();  
 }  
 fileInputStream.close();  
 }  
}

**EasyExcel**

JAVA解析Excel工具EasyExcel

Java解析、生成Excel比较有名的框架有Apache poi、jyxl。但他们都存在一个严重的问题就是非常的耗内存，poi有一套SAX模式的API可以一定程度的解决一些内存溢出的问题，但POl还是有一些缺陷，比如07版Excel解压缩以及解压后存储都是在内存中完成的，内存消耗依然很大。easyexcel重写了poi对O7版Excel的解析，能够原本一个3M的excel用POl sax依然需要100M左右内存降低到几M，并且再大的excel不会出现内存溢出，03版依赖POlI的sax模式。在上层做了模型转换的封装，让使用者更加简单方便。

EasyExcel是阿里巴巴开源的一个excel处理框架，以使用简单、节省内存著称。

EasyExcel能大大减少占用内存的主要原因是在解析Excel时没有将文件数据一次性全部加载到内存中，而是从磁盘上一行行读取数据，逐个解析。

导入依赖：

<!-- easyexcel依赖-->  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>easyexcel</artifactId>  
 <version>2.2.6</version>  
 </dependency>  
<!-- 日志依赖-->  
 <dependency>  
 <groupId>log4j</groupId>  
 <artifactId>log4j</artifactId>  
 <version>1.2.17</version>  
 </dependency>  
<!-- fastjson-->  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>fastjson</artifactId>  
 <version>1.2.62</version>  
 </dependency>

代码如下：

工具类ExcelUtils ：

public class ExcelUtils {  
  
 public static String getExcel(String fileName,List<User> list) throws Exception {  
 try {  
 EasyExcel.write(fileName, User.class).sheet("学生信息表").doWrite(list);  
 return "已经保存到:"+fileName;  
 } catch (Exception e) {  
 throw new Exception("导出失败");  
 }  
 }  
  
 public static String getMysql(String fileName, UserService userService){  
  
 EasyExcel.read(fileName,User.class,new DemoDataListener(userService)).sheet().doRead();  
 return "Excel的表单内容存入数据库成功";  
 }  
}

监听器：（读取excel后可以将其存储到数据库上，在工具类里）

public class DemoDataListener extends AnalysisEventListener<User> {  
 private static final Logger LOGGER = LoggerFactory.getLogger(DemoDataListener.class);  
 /\*\*  
 \* 每隔5条存储数据库，实际使用中可以3000条，然后清理list ，方便内存回收  
 \*/  
 private UserService userService;  
 private static final int BATCH\_COUNT = 5;  
 List<User> list = new ArrayList<User>();  
 public DemoDataListener(){}  
 public DemoDataListener(UserService userService) {  
 this.userService = userService;  
 }  
 public void invoke(User data, AnalysisContext context) {  
 LOGGER.info("解析到一条数据:{}", JSON.toJSONString(data));  
 list.add(data);  
 // 达到BATCH\_COUNT了，需要去存储一次数据库，防止数据几万条数据在内存，容易OOM  
 if (list.size() >= BATCH\_COUNT) {  
 saveData();  
 // 存储完成清理 list  
 list.clear();  
 }  
 }  
 /\*\*  
 \* 所有数据解析完成了 都会来调用  
 \*  
 \* @param context  
 \*/  
 @Override  
 public void doAfterAllAnalysed(AnalysisContext context) {  
 // 这里也要保存数据，确保最后遗留的数据也存储到数据库  
 saveData();  
 LOGGER.info("所有数据解析完成！");  
 }  
 /\*\*  
 \* 加上存储数据库  
 \*/  
 private void saveData() {  
 LOGGER.info("{}条数据，开始存储数据库！", list.size());  
  
 for (User user : list) {  
 userService.save(user);  
  
 }  
 LOGGER.info("存储数据库成功！");  
 }  
}

Service层写法：

/\*\*  
 \* <p>  
 \* 服务实现类  
 \* </p>  
 \*  
 \* @author zsp  
 \* @since 2021-04-08  
 \*/  
@Service  
public class UserServiceImpl extends ServiceImpl<UserMapper, User> implements UserService {  
 private String fileName="D:\\java项目\\POI\\学生信息表03.xls";  
 public String getExcel() throws Exception {  
 try {  
 return ExcelUtils.getExcel(fileName,list());  
 } catch (Exception e) {  
 throw new Exception("导出失败Excel");  
 }  
 }  
 public String getMysql(String fileName,UserService userService) {  
 ExcelUtils.getMysql(fileName,userService);  
 return "将"+fileName+"导入到数据库成功";  
 }  
}

Controller层写法：

/\*\*  
 \* <p>  
 \* 前端控制器  
 \* </p>  
 \*  
 \* @author zsp  
 \* @since 2021-04-08  
 \*/  
@RestController  
@RequestMapping("/user")  
public class UserController {  
 @Autowired  
 UserService userService;  
  
 @RequestMapping("/list")  
 public List getList() {  
 List<User> list = userService.list();  
 return list;  
 }  
  
 @GetMapping("/getExcel")  
 public String getExcel() {  
 try {  
 return userService.getExcel();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return "生成excel出现错误";  
 }  
 @GetMapping("/getMysql")  
 public String getMysql() {  
 String fileName="D:\\java项目\\POI\\学生信息表03.xls";  
 try {  
 return userService.getMysql(fileName,userService);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return "生成excel出现错误";  
 }  
  
}