由于项目需要对大量Excel数据进行输入输出处理，在使用JXL，POI后发现很容易出现OOM，最后在网上找到阿里的开源项目EasyExcel能很快速的读取写入超大Excel文件。经过大量的调试优化，现通过JAVA生成104万行20列的数据并写入到Excel文件的Sheet中只需要70秒的时间。

以下为本工程代码：

如果是maven工程在pom.xml中加入以下内容：

   <dependency>

           <groupId>com.google.guava</groupId>

           <artifactId>guava</artifactId>

           <version>27.0-jre</version>

       </dependency>

       <dependency>

           <groupId>com.alibaba</groupId>

           <artifactId>easyexcel</artifactId>

           <version>1.1.2-beta5</version>

       </dependency>

工具包封装

package utils;

import java.io.ByteArrayOutputStream;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.util.List;

import java.util.Map;

import java.util.Map.Entry;

import com.alibaba.excel.EasyExcelFactory;

import com.alibaba.excel.ExcelReader;

import com.alibaba.excel.ExcelWriter;

import com.alibaba.excel.context.AnalysisContext;

import com.alibaba.excel.event.AnalysisEventListener;

import com.alibaba.excel.metadata.BaseRowModel;

import com.alibaba.excel.metadata.Sheet;

import com.google.common.base.Splitter;

import com.google.common.base.Strings;

import com.google.common.collect.Lists;

import com.google.common.collect.Maps;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

/\*\*

\* @author zhao.yingchao

\* @date 2019-02-27

\* @since v1.0.0

\*/

public class EasyExcelUtil {

private static final Logger LOGGER = LoggerFactory.getLogger(EasyExcelUtil.class);

/\*\*

\* 通过String类,读取工作表数据

\*

\* @param filePath 文件路径

\* @return 数据集

\*/

public static Map<String, List<List<String>>> readExcelByString(String filePath) {

return readExcelByString(filePath, null);

}

/\*\*

\* 通过String类,读取工作表数据

\*

\* @param filePath 文件路径

\* @param sheetName sheetName

\* @return 数据集合

\*/

public static Map<String, List<List<String>>> readExcelByString(String filePath, String sheetName) {

// 创建返回信息

Map<String, List<List<String>>> dataListMap;

// 解析监听器

StringExcelListener excelListener = new StringExcelListener();

InputStream inputStream = null;

try {

// 创建文件流

inputStream = new FileInputStream(filePath);

dataListMap = readExcelByStringFromInputStream(inputStream, sheetName);

} catch (Exception e) {

throw new EasyExcelException("readExcelByModel from filePath failed." + e, e);

} finally {

// 关闭文件流

try {

if (null != inputStream) {

inputStream.close();

}

} catch (IOException e) {

LOGGER.error("inputStream.close failed!", e);

}

}

return dataListMap;

}

/\*\*

\* 通过String类,读取工作表数据

\*

\* @param inputStream 文件流

\* @param sheetName sheetName

\* @return 数据集合

\*/

public static Map<String, List<List<String>>> readExcelByStringFromInputStream(InputStream inputStream,

String sheetName) {

// 创建返回信息

Map<String, List<List<String>>> dataListMap = Maps.newLinkedHashMap();

// 解析监听器

StringExcelListener excelListener = new StringExcelListener();

try {

// 创建文件流

ExcelReader excelReader = EasyExcelFactory.getReader(inputStream, excelListener);

// 得到所有工作表

List<Sheet> sheets = excelReader.getSheets();

// 取所有工作表数据

for (Sheet sheet : sheets) {

// 工作表名称

String currentSheetName = sheet.getSheetName();

// 没有指定工作表，或多个工作表

if (Strings.isNullOrEmpty(sheetName) || Splitter.on(',').trimResults().omitEmptyStrings().splitToList(

sheetName).contains(currentSheetName)) {

// 读取Excel数据

excelReader.read(sheet);

// 返回明细数据

List<List<String>> sheetDataInfo = Lists.newArrayList(excelListener.getDataList());

// 将工作表数据放入工作薄

dataListMap.put(currentSheetName, sheetDataInfo);

// 清除缓存数据

excelListener.clear();

}

}

} catch (Exception e) {

throw new EasyExcelException("readExcelByStringFromInputStream from inputStream failed." + e, e);

}

return dataListMap;

}

/\*\*

\* 通过Model类,读取工作表数据

\*

\* @param filePath 文件路径

\* @param clazz BaseRowModel

\* @return 数据集合

\*/

public static Map<String, List<? extends BaseRowModel>> readExcelByModel(String filePath,

Class<? extends BaseRowModel> clazz) {

return readExcelByModel(filePath, null, clazz);

}

/\*\*

\* 通过Model类,读取工作表数据

\*

\* @param file 文件

\* @param clazz BaseRowModel

\* @return 数据集合

\*/

public static Map<String, List<? extends BaseRowModel>> readExcelByModel(File file,

Class<? extends BaseRowModel> clazz) {

return readExcelByModel(file, null, clazz);

}

/\*\*

\* 通过Model类,读取工作表数据

\*

\* @param filePath 文件路径

\* @param sheetName sheetName

\* @param clazz BaseRowModel

\* @return 数据集合

\*/

public static List<? extends BaseRowModel> readExcelByModelSheetName(String filePath, String sheetName,

Class<? extends BaseRowModel> clazz) {

Map<String, List<? extends BaseRowModel>> dataListMap = readExcelByModel(filePath, sheetName, clazz);

return dataListMap.getOrDefault(sheetName, null);

}

/\*\*

\* 通过Model类,读取工作表数据

\*

\* @param file 文件

\* @param sheetName sheetName

\* @param clazz BaseRowModel

\* @return 数据集合

\*/

public static List<? extends BaseRowModel> readExcelByModelSheetName(File file, String sheetName,

Class<? extends BaseRowModel> clazz) {

Map<String, List<? extends BaseRowModel>> dataListMap = readExcelByModel(file, sheetName, clazz);

return dataListMap.getOrDefault(sheetName, null);

}

/\*\*

\* 通过Model类,读取工作表数据

\*

\* @param filePath 文件路径

\* @param sheetName sheetName

\* @param clazz BaseRowModel

\* @return 数据集合

\*/

public static Map<String, List<? extends BaseRowModel>> readExcelByModel(String filePath, String sheetName,

Class<? extends BaseRowModel> clazz) {

Map<String, List<? extends BaseRowModel>> dataListMap;

InputStream inputStream = null;

try {

// 创建文件流

inputStream = new FileInputStream(filePath);

dataListMap = readExcelByModelFromInputStream(inputStream, sheetName, clazz);

} catch (Exception e) {

throw new EasyExcelException("readExcelByModel from filePath failed." + e, e);

} finally {

// 关闭文件流

try {

if (null != inputStream) {

inputStream.close();

}

} catch (IOException e) {

LOGGER.error("inputStream.close failed!", e);

}

}

return dataListMap;

}

/\*\*

\* 通过Model类,读取工作表数据

\*

\* @param file 文件

\* @param sheetName sheetName

\* @param clazz BaseRowModel

\* @return 数据集合

\*/

public static Map<String, List<? extends BaseRowModel>> readExcelByModel(File file, String sheetName,

Class<? extends BaseRowModel> clazz) {

Map<String, List<? extends BaseRowModel>> dataListMap;

InputStream inputStream = null;

try {

// 创建文件流

inputStream = new FileInputStream(file);

dataListMap = readExcelByModelFromInputStream(inputStream, sheetName, clazz);

// 关闭文件流

inputStream.close();

} catch (Exception e) {

throw new EasyExcelException("readExcelByModel from File failed." + e, e);

} finally {

// 关闭文件流

try {

if (null != inputStream) {

inputStream.close();

}

} catch (IOException e) {

LOGGER.error("inputStream.close failed!", e);

}

}

return dataListMap;

}

/\*\*

\* 通过Model类,读取工作表数据

\*

\* @param inputStream 文件流

\* @param sheetName sheetName

\* @param clazz BaseRowModel

\* @return 数据集合

\*/

public static Map<String, List<? extends BaseRowModel>> readExcelByModelFromInputStream(InputStream inputStream,

String sheetName,

Class<?

extends BaseRowModel> clazz) {

// 解析每行结果在listener中处理

// 创建返回信息

Map<String, List<? extends BaseRowModel>> dataListMap = Maps.newLinkedHashMap();

// 解析监听器

ModelExcelListener excelListener = new ModelExcelListener();

try {

// 创建文件流

ExcelReader excelReader = EasyExcelFactory.getReader(inputStream, excelListener);

// 得到所有工作表

List<Sheet> sheets = excelReader.getSheets();

// 取所有工作表数据

for (Sheet sheet : sheets) {

// 工作表名称

String currentSheetName = sheet.getSheetName();

if (Strings.isNullOrEmpty(sheetName) || Splitter.on(',').trimResults().omitEmptyStrings().splitToList(

sheetName).contains(currentSheetName)) {

// 设置模板

sheet.setClazz(clazz);

// 读取Excel数据

excelReader.read(sheet);

// 返回明细数据

List<? extends BaseRowModel> sheetDataInfo = Lists.newArrayList(excelListener.getDataList());

// 将工作表数据放入工作薄

dataListMap.put(currentSheetName, sheetDataInfo);

// 清除缓存数据

excelListener.clear();

}

}

} catch (Exception e) {

throw new EasyExcelException("readExcelByModel from inputStream failed." + e, e);

}

return dataListMap;

}

/\*\*

\* 通过String类,将一个sheet写入到一个Excel

\*

\* @param filePath 文件路径

\* @param sheetName sheetName

\* @param dataList 数据集

\*/

public static void writeExcelByString(String filePath, String sheetName, List<List<String>> dataList) {

// 创建返回信息

Map<String, List<List<String>>> dataListMap = Maps.newLinkedHashMap();

// 将工作表放入到Excel中

dataListMap.put(sheetName, dataList);

// 输出Excel数据

writeExcelByString(filePath, dataListMap);

}

/\*\*

\* 通过String类,将多个sheet写入到一个Excel

\*

\* @param filePath 文件路径

\* @param dataListMap 数据集

\*/

public static void writeExcelByString(String filePath, Map<String, List<List<String>>> dataListMap) {

try {

// 工作表编号

int sheetNo = 1;

// 创建文件流

OutputStream out = new FileOutputStream(filePath);

ExcelWriter writer = EasyExcelFactory.getWriter(out);

// 循环写入每个工作表

for (Entry<String, List<List<String>>> entry : dataListMap.entrySet()) {

// 得到工作表名称

String sheetName = entry.getKey();

// 得到工作表数据

List<List<String>> dataList = entry.getValue();

// 设置工作表信息

Sheet sheet1 = new Sheet(sheetNo++, 1, null, sheetName, null);

// 设置开始行为-1

sheet1.setStartRow(-1);

// 设置自适应宽度

sheet1.setAutoWidth(Boolean.TRUE);

// 开始写数据

writer.write0(dataList, sheet1);

}

// 清空缓存

writer.finish();

// 关闭文件

out.close();

} catch (Exception e) {

throw new EasyExcelException("writeExcelByString failed." + e, e);

}

}

/\*\*

\* 通过Model类,将一个sheet写入到一个Excel

\*

\* @param filePath 文件路径

\* @param sheetName sheetName

\* @param dataList 数据集

\* @param clazz BaseRowModel

\*/

public static void writeExcelByModel(String filePath, String sheetName, List<? extends BaseRowModel> dataList,

Class<? extends BaseRowModel> clazz) {

// 创建返回信息

Map<String, List<? extends BaseRowModel>> dataListMap = Maps.newLinkedHashMap();

// 将工作表放入到Excel中

dataListMap.put(sheetName, dataList);

// 输出Excel数据

writeExcelByModel(filePath, dataListMap, clazz);

}

/\*\*

\* 通过String类,将多个sheet写入到一个Excel

\*

\* @param filePath 文件路径

\* @param dataListMap 数据集

\* @param clazz BaseRowModel

\*/

public static void writeExcelByModel(String filePath, Map<String, List<? extends BaseRowModel>> dataListMap,

Class<? extends BaseRowModel> clazz) {

try {

// 创建文件流

OutputStream out = new FileOutputStream(filePath);

// 写入文件

writeIntoOutputStream(dataListMap, clazz, out);

// 关闭文件

out.close();

} catch (Throwable e) {

throw new EasyExcelException("write to file failed." + e, e);

}

}

/\*\*

\* export to byte array.

\*

\* @param sheetName sheetName

\* @param dataList data

\* @param clazz BaseRowModel

\* @return return a byte array with data.

\*/

public static byte[] exportByteArray(String sheetName, List<? extends BaseRowModel> dataList,

Class<? extends BaseRowModel> clazz) {

// 创建返回信息

Map<String, List<? extends BaseRowModel>> dataListMap = Maps.newLinkedHashMap();

// 将工作表放入到Excel中

dataListMap.put(sheetName, dataList);

ByteArrayOutputStream out = new ByteArrayOutputStream();

writeIntoOutputStream(dataListMap, clazz, out);

return out.toByteArray();

}

/\*\*

\* export to byte array.

\*

\* @param dataListMap data

\* @param clazz BaseRowModel

\* @return return a byte array with data.

\*/

public static byte[] exportByteArray(Map<String, List<? extends BaseRowModel>> dataListMap,

Class<? extends BaseRowModel> clazz) {

ByteArrayOutputStream out = new ByteArrayOutputStream();

writeIntoOutputStream(dataListMap, clazz, out);

return out.toByteArray();

}

/\*\*

\* 数据写入输出流

\*

\* @param dataListMap 数据

\* @param clazz BaseRowModel

\* @param outputStream 输出流

\*/

public static void writeIntoOutputStream(Map<String, List<? extends BaseRowModel>> dataListMap,

Class<? extends BaseRowModel> clazz,

OutputStream outputStream) {

try {

// 工作表编号

int sheetNo = 1;

// 创建文件流

ExcelWriter writer = EasyExcelFactory.getWriter(outputStream);

// 循环写入每个工作表

for (Entry<String, List<? extends BaseRowModel>> entry : dataListMap.entrySet()) {

// 得到工作表名称

String sheetName = entry.getKey();

// 得到工作表数据

List<? extends BaseRowModel> dataList = entry.getValue();

// 设置工作表信息

Sheet sheet1 = new Sheet(sheetNo++, 1, clazz, sheetName, null);

// 设置自适应宽度

sheet1.setAutoWidth(Boolean.TRUE);

// 开始写数据

writer.write(dataList, sheet1);

}

// 清空缓存

writer.finish();

} catch (Throwable e) {

throw new EasyExcelException("write to OutputStream failed." + e, e);

}

}

/\*\*

\* String类,解析监听器

\*/

private static class StringExcelListener extends AnalysisEventListener<List<String>> {

/\*\*

\* 自定义用于暂时存储data 可以通过实例获取该值

\*/

private List<List<String>> dataList = Lists.newArrayList();

@Override

public void invoke(List<String> rowInfo, AnalysisContext context) {

// 数据存储到list，供批量处理，或后续自己业务逻辑处理。

dataList.add(rowInfo);

}

@Override

public void doAfterAllAnalysed(AnalysisContext context) {

//解析结束销毁不用的资源

}

private List<List<String>> getDataList() {

return dataList;

}

private void setDataList(List<List<String>> dataList) {

this.dataList = dataList;

}

private void clear() {

dataList.clear();

}

}

/\*\*

\* Model类,解析监听器

\*/

private static class ModelExcelListener extends AnalysisEventListener<BaseRowModel> {

/\*\*

\* 自定义用于暂时存储data 可以通过实例获取该值

\*/

private List<BaseRowModel> dataList = Lists.newArrayList();

@Override

public void invoke(BaseRowModel rowInfo, AnalysisContext context) {

dataList.add(rowInfo);

}

/\*\*

\* 解析结束销毁不用的资源

\*

\* @param context AnalysisContext

\*/

@Override

public void doAfterAllAnalysed(AnalysisContext context) {

//解析结束销毁不用的资源

}

/\*\*

\* 获取

\*

\* @return 返回sheet数据

\*/

private List<? extends BaseRowModel> getDataList() {

return dataList;

}

/\*\*

\* 设置sheet数据

\*

\* @param dataList 数据

\*/

private void setDataList(List<BaseRowModel> dataList) {

this.dataList = dataList;

}

/\*\*

\* 清空数据

\*/

private void clear() {

dataList.clear();

}

}

/\*\*

\* EasyExcelException

\*/

public static class EasyExcelException extends RuntimeException {

private static final long serialVersionUID = -5456062088984840434L;

public EasyExcelException() {

super();

}

public EasyExcelException(String message) {

super(message);

}

public EasyExcelException(String message, Throwable cause) {

super(message, cause);

}

public EasyExcelException(Throwable cause) {

super(cause);

}

}

}

样例：

/\*\*

\* Excel模型

\*

\* @author zhao.yingchao

\* @date 2019年02月27日

\* @since v1.0.0

\*/

@Data

public class Test extends BaseRowModel {

@ExcelProperty(value = "姓名", index = 0)

private String name;

@ExcelProperty(value = "年龄", index = 1)

private String age;

@ExcelProperty(value = "手机号", index = 2)

private String phoneNum;

}

使用：

/\*\*

\* @author zhao.yingchao

\* @date 2019-02-27

\* @since v1.0.0

\*/

public class EasyExcelUtilTest {

@Test

public void test\_parse\_excel() {

Resource resource = new ClassPathResource("file/list1W.xlsx");

try {

long startTime = MonitorUtil.startLogInfo("parse begin");

List<Test> list = (List<Test>)EasyExcelUtil.readExcelByModelSheetName(resource.getFile(),

"Sheet1",

Test.class);

MonitorUtil.endLogInfo(startTime, "parse end");

Assert.assertEquals(list.size(), 10000);

} catch (IOException e) {

e.printStackTrace();

}

}

}

————————————————

版权声明：本文为CSDN博主「ZhaoYingChao88」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/ZYC88888/article/details/87989649