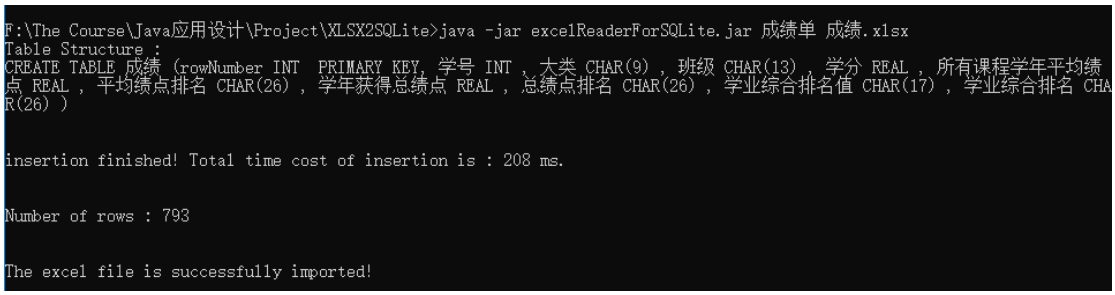
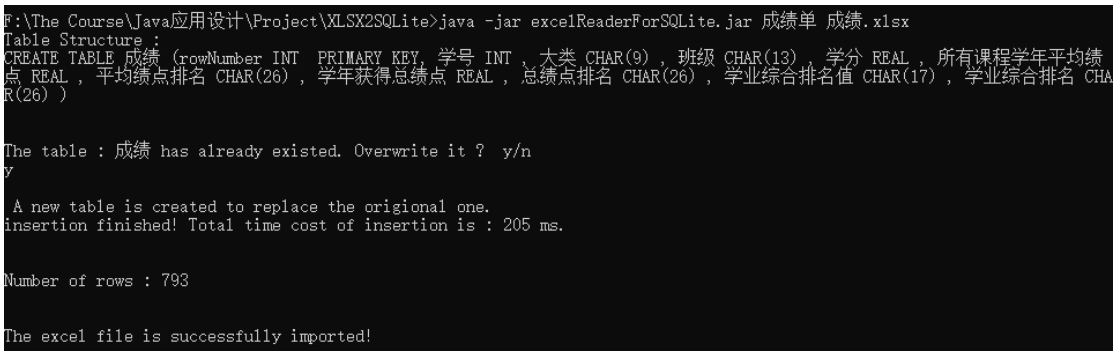
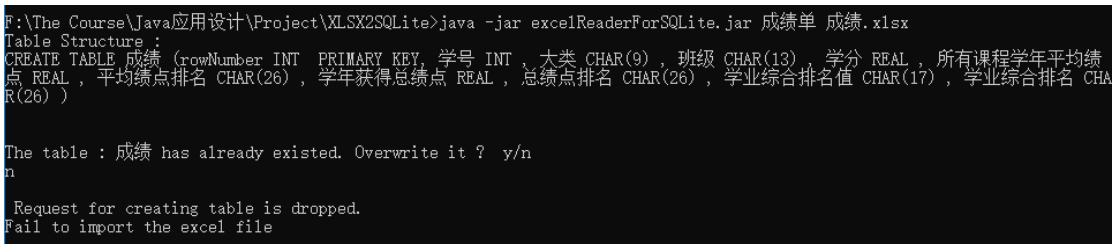
XLSX2SQLite







Author : 漆翔宇　计算机科学与技术　3170104557
Email : 3170104557@zju.edu.cn
Tel : 17342017090

Chapter1 项目结构概述

整个程序由三个类组成：excelReaderForSQLite, SQLiteManager, excelReader。

1. SQLiteManager

用于管理 jdbc 接口的类。创建表单，添加表项等和 SQLite 数据库进行交流的工作全部都封装在了这个类中。

2. excelReader

用于读取 excel 文件的类。 主要负责解析 excel 文件，把文件内容读取到内存中，封装成 Sheet, Cell 这样的 POI 标准对象，并处理读取时发生的异常事件。最后返回我们需要的目标页面。

3. excelReaderForSQLite

程序的主类，通过调用 excelReader 对象获取我们需要的 Sheet 对象，解析生成 SQL 命令，通过 SQLiteManager 对象向 SQLite 发送命令创建目标数据库，表单，添加表项。

Chapter2 excelReader.class

```
excelReader.class 的组成：
class excelReader
{
      private Workbook excelFile;
      excelReader(String fileName);
      public boolean successfullyRead();
      public Sheet getSheet(String sheetName);
}
```

1. POI

excel 文件的解析是通过引入外部库 POI 完成的。lib 目录下 poi-4.0.1.jar，poi-ooxml-4.0.1.jar，poi-ooxml-schemas-4.0.1.jar，commons-compress-1.18.jar，commons-collections4-4.2.jar 这五个.jar 文件都是我们读取 excel 文件时依赖的外部库文件。

2. 用到的 POI 类

Workbook：用于存储读入的 excel 文件对象。

HSSFWorkbook：Workbook 的子类，构造函数接受一个 InputStream 对象。是用于解析.xsl 格式文件时需要用到的类。

XSSFWorkbook：Workbook 的子类，构造函数接受一个 InputStream 对象。是用于解析.xslx 格式文件时需要用到的类。

Sheet：通过 Workbook 的对象就可以获得，是 excel 文件的一个子页的对象。管理 excel 一个具体页的数据。

Row：对应于 excel 文件中的一行。

Cell：对应于 excel 文件中的一个单元格。

CellType：POI 中单元格是被解析了数据类型的，包括 STRING，NUMERIC ,BOOLEAN ,FORMULA 等。这些类型由 CellType 对象封装表示。用于实验要求我们为添加到 SQLite 中的数据做 INT , REAL , CHAR 类型的区分，所以我们还需要获得这个对象，进一步分类成我们的目标类型对象。

3. 用到的 POI 对象方法

Workbook :

```
getNumberOfSheets();
//返回 excel 对象的 sheet 的页数
getSheetAt(int);
//所有 Sheet 对象都从 0 开始编号，用这个方法可以获取指定编号的 Sheet 对象
```

Sheet :

```
getSheetName();
//返回一个 String 类型字符串，对应于 Sheet 的名称
getLastRowNum();
//Sheet 的行也是从 0 开始编号的，这个方法返回最后一行的编号，也就间接获得了行数
getRow(int);
```

//传入一个整数，获取整数对应行号的 Row 对象

Row ：

getPhysicalNumberOfCells();

//返回一个整数，表明 Row 对象这一行有多少个单元格

getCell(int);

//传入一个整数，返回对应的单元格对象 Cell

Cell :

getCellType();

//返回一个 CellType 对象,表明类型

getNumericCellValue();

getBooleanCellValue();

getStringCellValue();

getCellFormula();

//对于不同类型的单元格用不用的方法获取单元格内容

4. 构造器

excelReader 提供了一个接受一个字符串的构造器。这个字符串是目标 excel 文件，后缀不可省略。构造器会检查文件名后缀，如果是.xls 文件则用 HSSFWorkbook 对象读入文件输入流，如果是.xlsx 文件则用 XSSFWorkbook 对象读入。读入的对象会赋值给 excelFile 成员变量。

如果读入失败或者文件名不规范，excelFile 成员会被赋值为 null。

5. successfullyRead

一个返回 boolean 类型的方法。判断 excelFile 成员是否为 null。如果为 null 会返回 false，反之返回 true。主要是用来指示我们的 reader 对象是否成功读入文件。如果没有，就要及时的终止程序并抛出异常信息。

6. getSheet

接受一个字符串参数，表明目标页的名称。getSheet 方法会从 Workbook 对象中搜索每一个 sheet，看名称是否与目标页名称匹配，然后返回匹配的 Sheet 对象。如果匹配失败或者发生任何问题，就会返回 null。

Chapter3 SQLiteManager.class

```
class SQLiteManager
{
    private static String sqliteJdbc="org.sqlite.JDBC";
    private String dataBaseName;
    private Connection connector;
    private Statement stmt;
    private PreparedStatement pstmt;
    public static long totTime=0;
    public SQLiteManager(String dataBaseName);
    public int startInsert();
    public int insertItem(String sql);
    public int endInsert();
    public int createTable(String sql);
}
```

1. JDBC

lib 目录下的 sqlite-jdbc-3.23.1.jar 文件是 SQLiie 的 JDBC 外部库文件。我们使用 JDBC 连接 SQLite 引擎，通过这个软件层从 java 程序发送 SQL 命令到 SQLite 引擎，并从 SQLite 接收返回的信息。

2. 使用到的 sql 类和方法

Java 提供了统一的 JDBC 接口，我们统一使用 java.sql 下的类来连接 JDBC.

Connection :

用于建立和数据库的连接。在调用 DriverManager.getConnection(dataBaseName)方法后，会返回一个 Connection 对象。借助这个 Connection 对象，我们和 SQLite 数据库进行“沟通“。

createStatement();

//返回一个 Statement 对象.我们用这个对象来将 SQL 命令传递给数据库引擎执行

setAutoCommit(boolean);

/*设置是否自动提交 SQL 命令，默认为 true，设置为 false 的时候我们可以一次把多个命令一起提交*/

commit();//非自动提交模式下,把囤积起来的命令一起提交

close();//断开 Connection 对象和数据库的连接

getMetaData();//返回一个 DataBaseMeta 对象，获取数据库的元信息

Statement :

用于向 SQLite 引擎发送 SQL 命令。

executeUpdate(String)//传入一个 SQL 命令字符串，执行 SQL 命令

close()//断开 Statement 对象和数据库的连接

ResultSet :

用于接收从数据库传回得到信息集。

next();//返回一个 boolean 对象，表示是否还有下一个元素

DataBaseMeta :

存储数据库的元信息
getTables(String,String,String,String);
//从元信息中搜索符合要求的表，并以 ResultSet 的形式返回

3. 构造器

接受一个字符串作为数据库的名称。

```
this.dataBaseName="jdbc:sqlite:./SQLite dataBase/"+dataBaseName;
if(dataBaseName.indexOf(".db")==-1)this.dataBaseName+=".db";
Class.forName(sqliteJdbc);
```

处理传入的数据库名称，前面加上前缀 jdbc:sqltie: 。并且加上一个路径/SQLite dataBase，这样数据库位置就被限定在了我们项目的 SQLite dataBase 目录下。自动创建的数据库都会自动放在目录下。检测是否有.db 后缀，如果没有，添加上。

最后调用 Class.forName(sqliteJdbc)拉起我们的 SQLite JDBC 引擎。

4. 创建表单

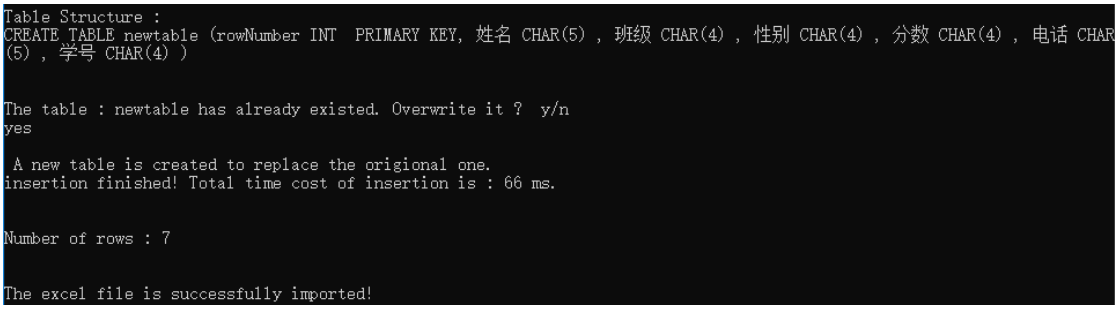
```
public int createTable(String sql,String tableName)
```

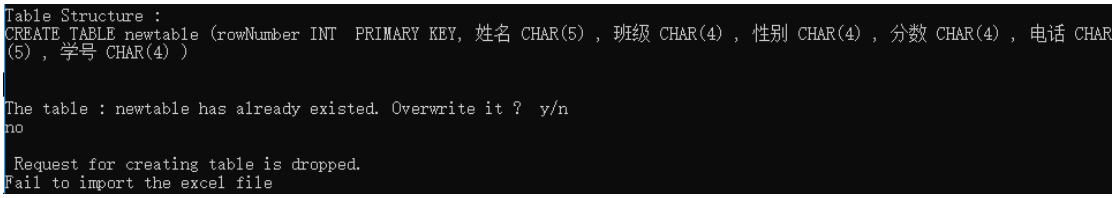
接受一个创建表单的命令字符串和一个指示表名的字符串。创建相应的 Connect 对象和目标数据库连接，然后创建 Statement 对象。

由于 SQLite 不允许同样名称的表单被重复创建，所以在执行 SQL 命令建立同名表单的时候，实际上会抛出异常。我们在执行创建表单的 SQL 命令前检查一下是否已经有同名的表单存在了。

```
ResultSet rs = connector.getMetaData().getTables(null, null, tableName,
null);
if (rs.next())
{
     rs.close();
     System.out.println("The table : "+tableName +
     " has already existed. Overwrite it ?   y/n ");
     Scanner in = new Scanner(System.in);
     String confirm=in.nextLine().toLowerCase();
    ............
}else{.........}
```

在数据库的元信息中查找名为 tableName 的表，如果 rs.next()==true 说明存在同名表单。此时程序抛出一个确认信息，调用 Scanner.nextLine()方法阻塞程序，等待用户确认是否覆盖原来的同名表。用户输入 y 或者 yes 后执行 SQL 命令 DROP TABLE tableName，删除原来的表单，再执行 sql 字符串中的新建表单命令。用户输入 n 后则放弃执行 sql 命令，返回 0，调用这个方法的父程序段得到返回值 0 后，就会继续返回错误代码，程序会终止，并告知导入失败。





当创建表单的命令被正常执行后，程序返回 1。

5. 添加表项

```
public int startInsert()
public int insertItem(String sql)
public int endInsert()
```

最开始，我的设计中没有 startInser 和 endInsert 方法。只有单独的 insertItem 方法，每一次都和数据库连接一次，发送一次命令，再断开。结果发现这样的操作非常的慢。平均每添加一项需要耗时 100ms。添加一个 140 项的表单，需要长达 14s 左右的时间。由于数据库要下学期才学，现在对数据库的了解还不太多，所以也不太清楚原因。

一开始我猜测是连接数据库这一步比较耗时，所以添加了 startInsert 和 endInser 方法，在第一条添加命令之前调用 startInsert 方法，事先建立好连接，准备好 Statement 方法，然后每次调用 insertItem 方法只执行一次 SQL 命令就好了。结果依然非常缓慢。

就在我感到无从下手的时候，由于寝室要断电了，我把项目从台式机移到了笔记本上。结果再笔记本上执行同样的程序，只花了 2s 钟，快了七倍。我的台式机 CPU 和笔记本 CPU 都是 3.7GHZ，如果瓶颈在 CPU 上，不应该有这么大的性能差距。我注意到我把项目移植到笔记本时，是放在了桌面上，也就是说我正在 C 盘上运行程序，而这里和之前台式机上工作环境最大的区别是，我们正在 SSD 上工作。这让我想到很有可能是磁盘带来的性能瓶颈。

通过查阅文档，我发现，SQLite JDBC 默认为自动提交模式。也就是说每一次执行了 stmt.execute(sql)时，都会把数据库内容从磁盘中读到内存中，在内存中把添加的表项放进去后又立刻写回磁盘。。相当于每次都读写了一次磁盘。。一次来回时间大概为 100ms，这是非常低效的。

解决方法是在执行 startInsert 的时候，调用 connector.setAutoCommit(false)禁用自动提交，这样我们在执行 stmt.execute(sql)时，就不会立刻写回磁盘了。直到我们完成了所有的表项写入操作，调用 endInsert 方法的时候，执行 connector.commit()方法，完成提交。这个时候才会被写回磁盘。

经过测试，写入 140 条总耗时约为 100ms，大概是之前写入 1 条的总耗时。由此可见，读写数据库的效率瓶颈严重依赖于读写磁盘的次数。

Chapter4 excelReaderForSQLite.class

```
public class excelReaderForSQLite
{
    private String dataBaseName;
    private String excelFileName;
    private String sheetName;
    private String tableName;
    private String [] type;
    private Sheet sourceSheet;
    private excelReader myExcel;
    private SQLiteManager mySQLite;
    private excelReaderForSQLite(String[] args)
    private String analyzeCell(Cell x)
    private int importTable()
    public int startImport()
    public static excelReaderForSQLite
    createExcelToSQLiteImporter(String[] args);
    public static void main(String[] args);
}
```

1. main

```
public static void main(String[] args)
{
    excelReaderForSQLite Importer=createExcelToSQLiteImporter(args);
    if(Importer==null)System.exit(0);
    int status =Importer.startImport();
    if(status==1)
    System.out.println("The excel file is successfully imported!");
    else
    System.out.println("Fail to import the excel file");
    return;
}
```

根据命令行构造 excelReaderForSQLite 对象，调用对象的 startImport 方法开始导入。检测导入函数的返回，判断是否成功。

2. createExcelToSQLiteImport

把 main 方法接受到的命令行参数传入，解析命令行参数。对非法参数进行反馈，对于合法的命令行参数返回 excelReaderForSQLite 对象

3. 构造器

对参数做解析，构造对象。并构造自己的 excelReader 成员和 SQLiteManager 成员。在

构造函数中即完成了数据库连接的初始化，解析和读入目标 excel 文件。

4. startImport

检查 excel 文件是否正确读入了，失败就直接返回。

调用自己的 getTableInfor 方法，解析读入 excel 文件的指定页，对页表的表项和类型做分析，以 SQL 格式返回表项信息。

调用 SQLiteManager 对象的 createTable 方法，创建与页表表项性质相符的页表。

调用自己的 importTable 方法，开始导入表项。

5. getTableInfor

调用 excelReader 对象的 getSheet 方法获取目标页。如果获取失败，则返回错误代码。

解析获得的目标页 Sheet 对象，对第一行的内容进行分析，得到各表项的名称。然后扫描每列，分析每列的每个元素，得到这个表项的类型。

返回 SQL 格式的表项串。其中附加了一个行号作为 PK。

值得一提的是解析表项类型的时候，需要获取每个 Cell 的内容。POI 没有提供把内容全部转换为字符串返回的接口。所以需要调用一个 analyzeCell 方法，这个方法负责获得表项的 POI 下格式，然后不同的格式用不同的接口获得自己的内容，最后再手动转换为字符串返回。然后我们再扫描这个字符串，把它划分为 CHAR,INT,REAL 三种类型。。

同时，值得一提的是，POI 中，对于格子为空的单元格（如下图所示），你获得它的 Cell 的时候，它返回的是一个 null，如果这个时候你再用这个引用调用方法的话，就会抛出 nullPointer 异常。并且如果中间有一整行都是空的，那么获得的 Row 对象也会是空的。所以我们要判定这种特殊情况，对于为 null 的引用，我们手动的把这里的字符串设置为“NULL“，这样添加进数据库时，这些空格位置都被 NULL 这个字符串给替代了。

	A	B	C	D	E	F	G	H
1	姓名	班级	性别	分数	电话	学号		
2	AFDSA	232	MALE	35			24	
3	FSDF	323		YY	15	45	4545	0
4	xx							1
5								
6								
7								
8					dfsaf			
9								

6. analyzeCell

```
private String analyzeCell(Cell x)
{
    if(x==null)return "NULL";
    CellType tempCellType=x.getCellType();
    String temp;
    if(tempCellType==CellType.STRING)
    temp = x.getStringCellValue();
    else if(tempCellType==CellType.NUMERIC)
    {
```

```
            double y=x.getNumericCellValue();
            if(Math.floor(y)==y)temp=""+(int)Math.floor(y);
            else temp = ""+y;
        }
        else if(tempCellType==CellType.BOOLEAN)
        temp = ""+x.getBooleanCellValue();
        else if(tempCellType== CellType.FORMULA)
        temp = ""+x.getCellFormula();
        else temp="NULL";
        return temp;
}
```

分析类型，用对应的方法获得内容，并统一以字符串格式返回。
判断空引用，直接返回"NULL"字符串。

7. importTable

把 Sheet 对象中的内容逐行读取出来，整理成 SQL 命令形式，统一调用 SQLiteManager 对象的 insertItem()方法执行添加表项的命令。

第一次添加前，调用 startInsert，添加完毕后调用 endInsert。

Chapter5 实验结果

```
F:\The Course\Java应用设计\Project\XLSX2SQLite>java -jar excelReaderForSQLite.jar 成绩单 成绩.xlsx
Table Structure :
CREATE TABLE 成绩 (rowNumber INT  PRIMARY KEY, 学号 INT , 大类 CHAR(9) , 班级 CHAR(13) , 学分 REAL , 所有课程学年平均绩
点 REAL , 平均绩点排名 CHAR(26) , 学年获得总绩点 REAL , 总绩点排名 CHAR(26) , 学业综合排名值 CHAR(17) , 学业综合排名 CHA
R(26) )


insertion finished! Total time cost of insertion is : 208 ms.


Number of rows : 793


The excel file is successfully imported!
```

```
F:\The Course\Java应用设计\Project\XLSX2SQLite>java -jar excelReaderForSQLite.jar 成绩单 成绩.xlsx
Table Structure :
CREATE TABLE 成绩 (rowNumber INT  PRIMARY KEY, 学号 INT , 大类 CHAR(9) , 班级 CHAR(13) , 学分 REAL , 所有课程学年平均绩
点 REAL , 平均绩点排名 CHAR(26) , 学年获得总绩点 REAL , 总绩点排名 CHAR(26) , 学业综合排名值 CHAR(17) , 学业综合排名 CHA
R(26) )


The table : 成绩 has already existed. Overwrite it ?  y/n
y

 A new table is created to replace the origional one.
insertion finished! Total time cost of insertion is : 205 ms.


Number of rows : 793


The excel file is successfully imported!
```

```
F:\The Course\Java应用设计\Project\XLSX2SQLite>java -jar excelReaderForSQLite.jar 成绩单 成绩.xlsx
Table Structure :
CREATE TABLE 成绩 (rowNumber INT  PRIMARY KEY, 学号 INT , 大类 CHAR(9) , 班级 CHAR(13) , 学分 REAL , 所有课程学年平均绩
点 REAL , 平均绩点排名 CHAR(26) , 学年获得总绩点 REAL , 总绩点排名 CHAR(26) , 学业综合排名值 CHAR(17) , 学业综合排名 CHA
R(26) )


The table : 成绩 has already existed. Overwrite it ?  y/n
n

 Request for creating table is dropped.
Fail to import the excel file
```