JEPLDroid

(JEPLayer for Android)

Reference Manual

v1.2

Doc. version 1.0

Apr. 9, 2015

TABLE OF CONTENTS

1. INTRODUCTION

1.1 WHAT IS JEPLDROID (OR JEPLAYER FOR ANDROID)

JEPLDroid is the port of JEPLayer[1] to Android environment, basically is a feature reduced version of JEPLayer removing stuff not compatible or hard to port to Android environment like JTA APIs.

In simple words: JEPLDroid = JEPLayer – JTA

APIs related to JTA have been removed, if your JEPLayer based code is not using JTA porting to JEPLDroid is straightforward.

Furthermore, because JEPLDroid is a subset of JEPLayer sharing identical source code, version numbers are the same.

The main objective of JEPLDroid is to leverage JDBC in Android environment, it has two uses:

1) JEPLDroid is JDBC driver agnostic: if your JDBC driver works in Android, JEPLDroid can work the same as JEPLayer in Android environment.
2) JEPLDroid used to leverage the JDBC API provided by SQLDroid[2] the famous JDBC driver for the built-in SQLite database engine included in any Android based device.

You easily realize the second option is the most interesting and useful use because rarely you are going to directly connect an Android device to a remote database and most of Android applications need some kind of local persistence and built-in SQLite is very common for this task.

1.2 WHY JEPLAYER AND BY EXTENSION JDBC IN ANDROID

JDBC with no doubt is mainstream and is the basic and low level piece for RDBMS persistence in Java. "Every" Java developer knows the most common JDBC methods in spite of high level ORMs usually hide the underlying JDBC calls.

In spite of JDBC is the low level layer for persistence it has become a bit complex and big, this complexity and size is not a problem in desktop and server programming but it can be overwhelming in power and memory limited devices like Android phones and tablets and database engines like SQLite. This is why SQLite API[3] in Android is specific, small and pragmatic[4].

In spite of Android was born for very limited devices, today this is not longer true, current phones have processors with more than 1 core, speeds higher than 1 GHz and memory climbing to GB levels, these days complete or semi-complete built-in support of JDBC for SQLite would not be a problem.

Anyway SQLDroid offers a basic and pragmatic JDBC support of SQLite based on the public Android Java API (SQLiteDataBase), and SQLDroid implementation shows how much Android SQLite API is not very different to JDBC (for instance `Cursor` regarding to `ResultSet`).

[1] http://code.google.com/p/jeplayer/

[2] http://code.google.com/p/sqldroid/

[3] http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html

[4] In spite of Android includes a internal and mysterious JDBC "driver" for SQLite, its use is discouraged

JEPLDroid automatically detects when SQLDroid JDBC driver is being used and makes the best to make JEPLayer workable in this very limited environment, furthermore JEPLDroid leverage SQLDroid providing indirect absolute positioning extremely important to request data in pages of big tables.

JEPLayer removes most of tedious tasks typical of database programming, like transaction demarcation and connection management.

2. CONSIDERATIONS

2.1 DOCUMENT SCOPE

This manual only explains the specifics of JEPLDroid, the JEPLayer port of Android. For a complete documentation of JEPLayer, download JEPLayer distribution[5] which includes extensive documentation.

2.2 DOCUMENT CONVENTIONS

A Verdana font is used to describe.

`A Courier New font is used for Java source code.`

2.3 LICENSE

JEPLDroid is open source and Apache License Version 2 licensed[6].

2.4 COPYRIGHTS

Jose María Arranz Santamaría is the author and intellectual property owner of JEPLDroid source code, documentation and examples.

2.5 REQUIRED DEVELOPER TECHNICAL SKILLS

Java 1.6, JDBC and JEPLayer knowledge are required.

[5] http://code.google.com/p/jeplayer/downloads/list

[6] http://www.apache.org/licenses/LICENSE-2.0.html

3. REQUIREMENTS AND INSTALLATION

3.1 TECHNICAL REQUIREMENTS, LIMITATIONS AND DEPENDENCIES

JEPLDroid has been compiled and tested on AndroidStudio v1.1, Oracle's Java Standard Edition (Java SE) SDK 1.7 and the Android SDK provided by Google. Source and binaries configuration is set to 1.6.

JEPLDroid is provided as an AAR (.aar) file ready to be included in your `libs` folder, however JEPLDroid source code compiles in an Android application with `minSdkVersion="15"`.

3.2 JEPLDROID DISTRIBUTION

JEPLDroid is distributed on three forms:

3.2.1 Binaries and manual

`JEPLDroid_dist_X.zip` contains the required binaries to include in Android applications using JEPLDroid, where X is the version.

Decompress this distribution ZIP file. The content of this file is two folders:

docs/ => Contains this manual and javadoc archives

lib/ => Contains the `jepldroid-version.aar` with the compiled Java classes. Just add this file to your `libs` folder of your Android application.

3.2.2 JCenter and Maven Central

Gradle:

```
dependencies {
    ...
    compile 'org.innowhere:jepldroid:(version)'
}
```

3.2.3 Source code

Development is done in GitHub. Releases are marked with a tag.

3.3 PROVIDE A DATASOURCE FOR SQLITE

JEPLayer needs a `DataSource` to access to your database, this also applies to JEPLDroid and SQLite. In Android a `DataSource` is not really important because the typical Android application just need one connection (`DataSource` is typically used as a `Connection` pool). This is why SQLDroid just provide a conventional JDBC driver but not a `DataSource`, do not worry JEPLDroid source distribution provides a simple `DataSource` with name `example.loadmanually.SimpleDataSource` located in `test` folder, this class also needs the class `SimpleConnectionWrapper` in the same package.

The following code creates a SQLite `DataSource` for built-in SQLite based on SQLDroid JDBC driver valid for an Android application with package id `com.innowhere.jepldroidtest` (you can obtain this package programmatically)[7]:

```
String jdbcDriver = "org.sqldroid.SQLDroidDriver";
String url =
         "jdbc:sqlite://data/data/com.innowhere.jepldroidtest/test.db";
String userName = "myLogin";
String password = "myPW";
int poolSize = 1;

DataSource ds =
    new SimpleDataSource(jdbcDriver,url,userName,password,poolSize);
```

JEPLDroid automatically detects when you are using the JDBC SQLDroid driver and perform some quirks to effectively run in JEPLDroid, in fact as explained latter JEPLDroid slightly improves SQLDroid under the hood.

[7] http://stackoverflow.com/questions/6589797/how-to-get-package-name-from-anywhere

4. LIMITATIONS AND IMPROVEMENTS OF JEPLDROID USING SQLDROID/SQLITE

4.1 LIMITATIONS

4.1.1 Affected rows are ever zero

The method `JEPLDALQuery.executeUpdate()` ever returns zero, this limitation is imposed by SQLite Java API, there is no return of affected rows when executing persistent actions.

There are some JEPLayer methods related with the result of `JEPLDALQuery.executeUpdate()` like `JEPLDALQuery.setStrictMinRows(int)` and `JEPLDALQuery.setStrictMaxRows(int)` these restrictions are ignored when SQLDroid JDBC driver is detected.

4.1.2 The method `getGeneratedKey` is not a single SQL sentence

This is a limitation of SQLite in any platform, SQLite JDBC drivers do not support `PreparedStatement` objects created passing the parameter `Statement.RETURN_GENERATED_KEYS`. The typical solution is executing the sentence `SELECT LAST_INSERT_ROWID()` immediately after the `INSERT` operation. This is how `JEPLDALQuery.getGeneratedKey(Class<U>)` is implemented under the hood.

4.2 IMPROVEMENTS

Because JEPLDroid is a layer on top of JDBC it can provide some improvements over SQLDroid:

4.2.1 Do not care about the type of parameters

SQLDroid does not implement `PreparedStatement.setObject(int column,Object param)` this is not a problem for calling methods like `JEPLDALQuery.addParameters(Object...)`

4.2.2 `getGeneratedKey` works as usual

As explained before `JEPLDALQuery.getGeneratedKey(Class<U>)` is a replacement of the non-implemented `PreparedStatement.getGeneratedKeys()`.

4.2.3 Use of the SQLite absolute positioning

SQLDroid does not support `ResultSet.absolute(int)`, this method is absolutely necessary for effective pagination of results in tables and SQL sentences with many rows. JEPLDroid automatically uses absolute positioning of SQLite.

By this way Java sentences like these

```
public List<Contact> selectJEPLDAOQueryRange(int from,int to)
{
    return dao.createJEPLDAOQuery("SELECT * FROM CONTACT")
            .setFirstResult(from)
            .setMaxResults(to - from)
            .getResultList();
}
```

can be used to get paginated sets of rows without loading more rows than requested.