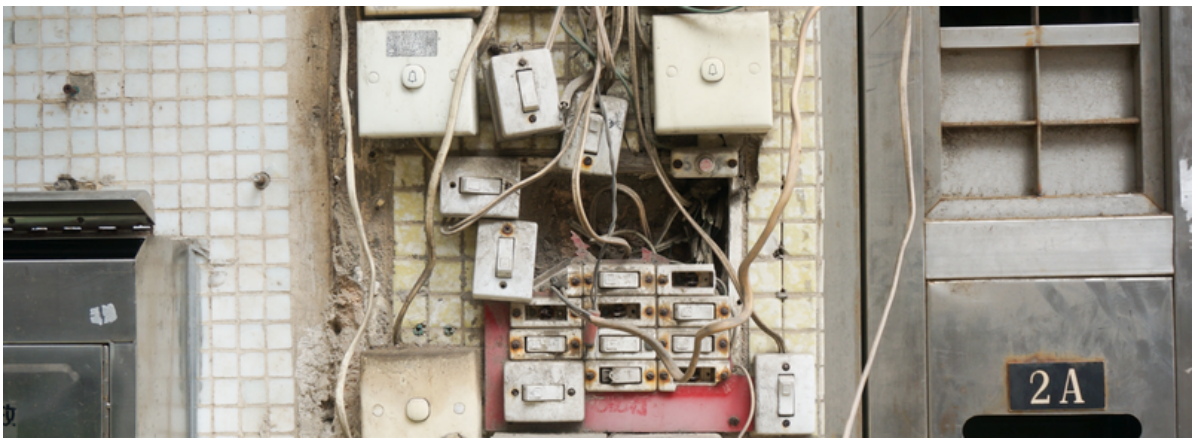
理解Room的数据迁移

泡在网上的日子 / 文 发表于2017-07-28 12:39 第4570次阅读 迁移 (/tags.php?/迁移/),SQLite (/tags.php?/SQLite/),持久化 (/tags.php?/持久化/),Room (/tags.php?/Room/)

(http://www.jiathis.com/share)

编辑推荐：稀土掘金 (https://juejin.im/)，这是一个针对技术开发者的一个应用，你可以在掘金上获取最新最优质的技术干货，不仅仅是Android知识、前端、后端以至于产品和设计都有涉猎，想成为全栈工程师的朋友不要错过！

Understanding migrations with Room (https://medium.com/google-developers/understanding-migrations-with-room-f01e04b07929)



使用SQLite API执行数据库迁移总有一种是在拆弹的感觉－仿佛一不小心就会让app在用户手中爆炸。如果你使用Room (https://developer.android.com/topic/libraries/architecture/room.html)来处理数据库的操作，那么迁移就非常简单了。

使用Room的时候，如果你改变了数据库的schema但是没有更新version，app将会crash。而如果你更新了version但是没有提供迁移，数据库的表就会drop掉，用户将丢失数据。

数据库迁移背后的原理

SQLite API 做了什么

SQLite数据库处理schema的改变是在database version的帮助下完成的。更准确的说，每当你添加，删除，或者修改表导致schema变化的时候，你都必须增加数据库的版本号并更新SQLiteOpenHelper.onUpgrade方法。当你从旧版本到新版本的时候，是它告诉SQLite该做什么。

它也是app开始和数据库工作是所触发的第一个调用。SQLite将首先处理版本的升级，然后才打开数据库。

Room做了什么

Room以 Migration (https://developer.android.com/reference/android/arch/persistence/room/migration/Migration.html) 类的形式提供可一个简化SQLite迁移的抽象层。Migration提供了从一个版本到另一个版本迁移的时候应该执行的操作。Room使用它自己实现的SQLiteOpenHelper，并在onUpgrade方法中触发你定义的迁移步骤。

这里是第一次获取数据库时将发生的事情：

1. Room数据库被创建
2. SQLiteOpenHelper.onUpgrade 方法被调用，然后Room触发迁移
3. 数据库被打开

如果你增加了数据库版本但是没有提供迁移，那么你的app可能会崩溃，数据可能会丢失，具体清空见下面的讨论。

identity hash字符串在migration内部扮演者重要的角色，它用来对数据库版本进行唯一标识。当前版本的identity hash被保存在一个由Room管理的配置表中。因此如果你在数据库中看到一个room_master_table表不要感到奇怪。

让我们以一个简单的user表为例，它有两个字段：

- ID, int, 同时也是 primary key
- user name, String

users表是版本为1的数据库的一部分，是用SQLiteDatabase API实现的。

假设你的用户已经在使用这个版本，现在你想开始使用Room，我们看看以下几种场景下Room是如何处理的。

Migrate SQLite API code to Room

在另一篇文章 (http://www.jcodecraeer.com/a/anzhuokaifa/androidkaifa/2017/0726/8249.html)中我们看到了如如何把你的app迁移到Room。那我们就在此基础上更详细的介绍数据迁移的细节。假设User entity 类 (https://developer.android.com/topic/libraries/architecture/room.html#entities) 和 UserDao都已经创建好了，重点放在继承了RoomDatabase (https://developer.android.com/reference/android/arch/persistence/room/RoomDatabase.html)的UsersDatabase类上面。

```
1.  @Database(entities = {User.class}, version = 1)
2.  public abstract class UsersDatabase extends RoomDatabase
```

场景 1: 保持 database 版本不变—app crashe

如果我们保持数据库版本的不变然后运行app的话，Room会做这些事情：

第一步：尝试打开database

- 通过比较当前版本和存到room_master_table中的identity hash来识别身份。但是，因为room_master_table中没有identity hash，app将会抛出IllegalStateException❌。

```
java.lang.IllegalStateException: Room cannot verify the data integrity.
Looks like you've changed schema but forgot to update the version number.
You can simply fix this by increasing the version number.
```

如果你修改了数据库的schema但是没有更新版本号，Room总是会抛出IllegalStateException。

> 译者注：为什么说这里schema发生了变化呢？因为Room增加了一个room_master_table表。所以从传统的SQLite API转为Room一定会发生schema的改变。

那我们就根据错误提示增加版本号。

```
@Database(entities = {User.class}, version = 2)
public abstract class UsersDatabase extends RoomDatabase
```

场景 2: 增加版本, 但不提供mogration—app crashes

现在再次运行app，Room将做如下事情：

第一步:尝试从 version1（安装到设备上的）更新到version 2

- 因为没有提供migration，app将crash，抛出IllegalStateException❌。

```
java.lang.IllegalStateException: A migration from 1 to 2 is necessary.
Please provide a Migration in the builder or call fallbackToDestructiveMigration in the builder in which
```

如果没有提供Migration，Room将抛出IllegalStateException。

场景 3: 增加版本，启用 fallback to destructive migration—数据库被清空

如果你不想提供migration，而且希望更新版本之后清空数据库，调用database builder的fallbackToDestructiveMigration。

```
database = Room.databaseBuilder(context.getApplicationContext(),
                        UsersDatabase.class, "Sample.db")
                .fallbackToDestructiveMigration()
                .build();
```

再次运行app，Room将做如下事情：

第一步 : 尝试从 version1（安装到设备上的）更新到version 2

- 因为没有migration，而且调用了fallbackToDestructiveMigration，所有表被丢弃，同时 identity_hash 被插入。

第二步：尝试打开数据库

- 现在当前版本的Identity hash 和保存在数据库room_master_table表中的就是相同的了。✅

现在我们的app不会崩溃了，但是我们丢失了所有数据，所以做之前要考虑清楚。

场景 4: 版本增加， 提供了migration—数据可以保存下来

为了保存用户的数据，我们需要实现一个migration。因为schema并没有发生变化（这里是指原有的表没有发生变换，其实严格说来是变化了的，因为增加了room_master_table表），所以我们只需提供一个空的migration。

```
1.  @Database(entities = {User.class}, version = 2)
2.  public abstract class UsersDatabase extends RoomDatabase {
3.  …
4.  static final Migration MIGRATION_1_2 = new Migration(1, 2) {
5.      @Override
6.      public void migrate(SupportSQLiteDatabase database) {
7.          // Since we didn't alter the table, there's nothing else to do here.
8.      }
9.  };
10. …
11. database =  Room.databaseBuilder(context.getApplicationContext(),
12.          UsersDatabase.class, "Sample.db")
13.          .addMigrations(MIGRATION_1_2)
14.          .build();
```

当运行app的时候，Room做如下事情：

第一步：尝试从version 1更新到version 2

- 触发定义的空migration✅
- 更新room_master_table表中的identity hash✅

第二步：尝试打开数据库

- 现在当前版本的Identity hash 和保存在数据库room_master_table表中的就是相同的了✅。

那么现在app可以打开了，同时用户数据也迁移了过来。

schema简单变化的迁移

让我们修改User类，向users表中添加一个新的字段：last_update。在UsersDatabase类中我们需要做如下工作：

1.把版本号增加到 3

```
1.  @Database(entities = {User.class}, version = 3)
2.  public abstract class UsersDatabase extends RoomDatabase
```

2. 添加一个version 2到 version 3的Migration

```
1.  static final Migration MIGRATION_2_3 = new Migration(2, 3) {
2.      @Override
3.      public void migrate(SupportSQLiteDatabase database) {
4.          database.execSQL("ALTER TABLE users "
5.                  + " ADD COLUMN last_update INTEGER");
6.      }
7.  };
```

3. 把migration 添加到 Room database builder:

```
database = Room.databaseBuilder(context.getApplicationContext(),
        UsersDatabase.class, "Sample.db")
        .addMigrations(MIGRATION_1_2, MIGRATION_2_3)
        .build();
```

当运行app的时候，下面的步骤被执行：

第一步:尝试从version 2更新到version 3

- 触发migration并修改表，保持用户的数据✅。
- 更新room_master_table中的identity hash ✅

第二步：尝试打开数据库

- 现在当前版本的Identity hash 和保存在数据库room_master_table表中的就是相同的了✅。

schema 复杂变化的迁移

SQLite的ALTER TABLE命令非常局限 (https://sqlite.org/lang_altertable.html)，只支持重命名表以及添加新的字段。比如，把user的id从int类型改成String需要经过如下几步才能完成：

- 创建一个新的临时表，
- 把users表中的数据拷贝到临时表中，
- 丢弃users表
- 把临时表重命名为users

使用Room，Migration的实现是这样的：

```
static final Migration MIGRATION_3_4 = new Migration(3, 4) {
    @Override
    public void migrate(SupportSQLiteDatabase database) {
        // Create the new table
        database.execSQL(
                "CREATE TABLE users_new (userid TEXT, username TEXT, last_update INTEGER, PRIMARY KEY(use
// Copy the data
        database.execSQL(
                "INSERT INTO users_new (userid, username, last_update) SELECT userid, username, last_upda
// Remove the old table
        database.execSQL("DROP TABLE users");
// Change the table name to the correct one
        database.execSQL("ALTER TABLE users_new RENAME TO users");
    }
};
```

多版本迁移

要是你的用户有一个运行版本号为1的app，想升级到版本4呢？目前位置我们定义了这些migrations：version 1 到 2, version 2 到 3, version 3 到 4, 所以Room 会一个接一个的触发所有 migration。

Room可以处理大于1的版本增量：我们可以一次性定义一个从1到4的migration，让迁移的速度更快。

```
static final Migration MIGRATION_1_4 = new Migration(1, 4) {
    @Override
    public void migrate(SupportSQLiteDatabase database) {
        // Create the new table
        database.execSQL(
                "CREATE TABLE users_new (userid TEXT, username TEXT, last_update INTEGER, PRIMARY KEY(use

        // Copy the data
        database.execSQL(
                "INSERT INTO users_new (userid, username, last_update) SELECT userid, username, last_upda
// Remove the old table
        database.execSQL("DROP TABLE users");
// Change the table name to the correct one
        database.execSQL("ALTER TABLE users_new RENAME TO users");
    }
};
```

然后，我们只需把它添加到migration列表中：

```
database = Room.databaseBuilder(context.getApplicationContext(),
        UsersDatabase.class, "Sample.db")
        .addMigrations(MIGRATION_1_2, MIGRATION_2_3, MIGRATION_3_4, MIGRATION_1_4)
        .build();
```

> Note that the queries you write in the Migration.migrate implementation are not compiled at run time, unlike the queries from your DAOs. Make sure that you're implementing tests for your migrations (https://medium.com/google-developers/testing-room-migrations-be93cdb0d975).

Show me the code

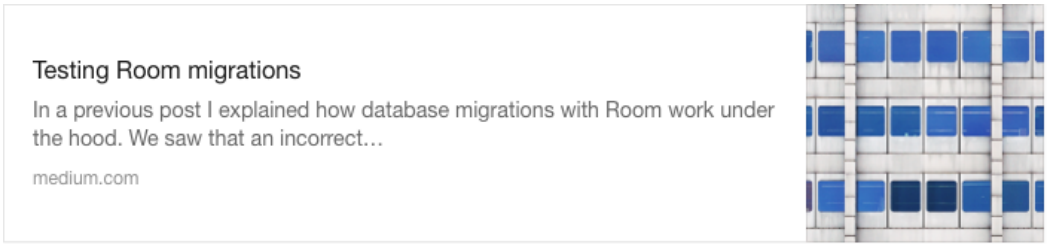
你可以在 这个 sample app (https://github.com/googlesamples/android-architecture-components)中找到实现的代码。为了遍于比较，每个database版本都实现了自己的flavor：

1. sqlite (https://github.com/googlesamples/android-architecture-components/tree/master/PersistenceMigrationsSample/app/src/sqlite/java/com/example/android/persistence/migrations)—使用 SQLiteOpenHelper 和 传统的 SQLite 接口.
2. room (https://github.com/googlesamples/android-architecture-components/tree/master/PersistenceMigrationsSample/app/src/room/java/com/example/android/persistence/migrations)—用Room来实现，并提供到版本2的迁移
3. room2 (https://github.com/googlesamples/android-architecture-components/tree/master/PersistenceMigrationsSample/app/src/room2/java/com/example/android/persistence/migrations)—把DB更新到新的schema, 版本为 3
4. room3 (https://github.com/googlesamples/android-architecture-components/tree/master/PersistenceMigrationsSample/app/src/room3/java/com/example/android/persistence/migrations)—更新到版本4，提供 version 2 to 3, version 3 to 4 and version 1 to 4的迁移路径。

总结

你的schema变化了吗？只需增加数据库版本并写一个Migration (https://developer.android.com/reference/android/arch/persistence/room/migration/Migration.html)就可以了。这样就可以确保app不会崩溃并且用户数据也不会丢失了。

但是如何测试迁移是否正确呢？在这里我们详细讨论了testing migrations (https://developer.android.com/topic/libraries/architecture/room.html#db-migration-testing)，并且这篇文章也涵盖了一些不同的场景：



(https://medium.com/google-developers/testing-room-migrations-be93cdb0d975)

收藏(3) 赞(5) 踩(0)

他们收藏了这篇文章

(/member/index.php?uid=sunnygeek) (/member/index.php?uid=秋意原) (/member/index.php?uid=niexueliang)

相关文章

Room使用七步曲 (/a/anzhuokaifa/androidkaifa/2017/0726/8249.html)	2017-07-28
手动迁移ADT 的ANT结构工程至Gradle (/a/anzhuokaifa/Android_Studio/2015/0127/2368.html)	2015-01-27
Android Studio导入Project的方法 (/a/anzhuokaifa/Android_Studio/2015/0128/2370.html)	2015-01-28
在Room中使用RxJava (/a/anzhuokaifa/androidkaifa/2017/0726/8268.html)	2017-07-26
Room Persistence Library（官网文档翻译） (/a/anzhuokaifa/androidkaifa/2017/0525/7971.html)	2017-08-09
Room，Realm,，ObjectBox 你选择哪个？ (/a/anzhuokaifa/androidkaifa/2017/0926/8551.html)	2017-09-29
Room使用七步曲 (/a/anzhuokaifa/androidkaifa/2017/0726/8249.html)	2017-07-28
MatrixCursor：可以实例化的Cursor以及其应用场景 (/a/anzhuokaifa/androidkaifa/2014/1024/1838.html)	2014-10-24
GreenDao官方文档翻译 (/a/anzhuokaifa/androidkaifa/2014/1127/2069.html)	2015-04-22
Android上令人愉快的持久化 (/a/anzhuokaifa/androidkaifa/2016/0506/4213.html)	2016-05-06

上一篇：**在Room中使用RxJava (/a/anzhuokaifa/androidkaifa/2017/0726/8268.html)**
更少的重复代码，编译时检查的SQL查询，除此之外还有异步功能和可观察的查询－听起来是不是很牛？有了 Room ， 这些都成为可能。异步查询返回 LiveData 或者RxJava的 Maybe , Single 或者 Flowable 。它们都是可观察的查询，可以让你在在数据变更的时候自动获

下一篇：**官方ORM框架Room (/a/anzhuokaifa/androidkaifa/2017/0728/8279.html)**

发表评论

推荐文章

SQLBrite:一个响应式的数据查询框架 (/a/anzhuokaifa/androidkaifa/2015/0306/2552.html)

NumberProgressBar：一个简约性感的数字ProgressBar (/a/anzhuokaifa/androidkaifa/2014/0813/1645.html)

Android插件化原理解析——概要 (/a/anzhuokaifa/androidkaifa/2016/0227/4005.html)

App开发架构指南（谷歌官方文档译文） (/a/anzhuokaifa/androidkaifa/2017/0523/7963.html)

终于等到你Depth-LIB-Android (/a/anzhuokaifa/androidkaifa/2016/0429/4200.html)

将Eclipse代码导入到Android Studio的两种方式 (/a/anzhuokaifa/androidkaifa/2015/0104/2259.html)

赞助商