转载请注明链接：https://blog.csdn.net/feather_wch/article/details/81503652

GreenDao

GreenDao的作用

1、GreenDao的作用

> 1. GreenDAO 是一个将对象映射到 SQLite 数据库中的轻量且快速的 ORM解决方案。
>
> 
>

2、ORM是什么

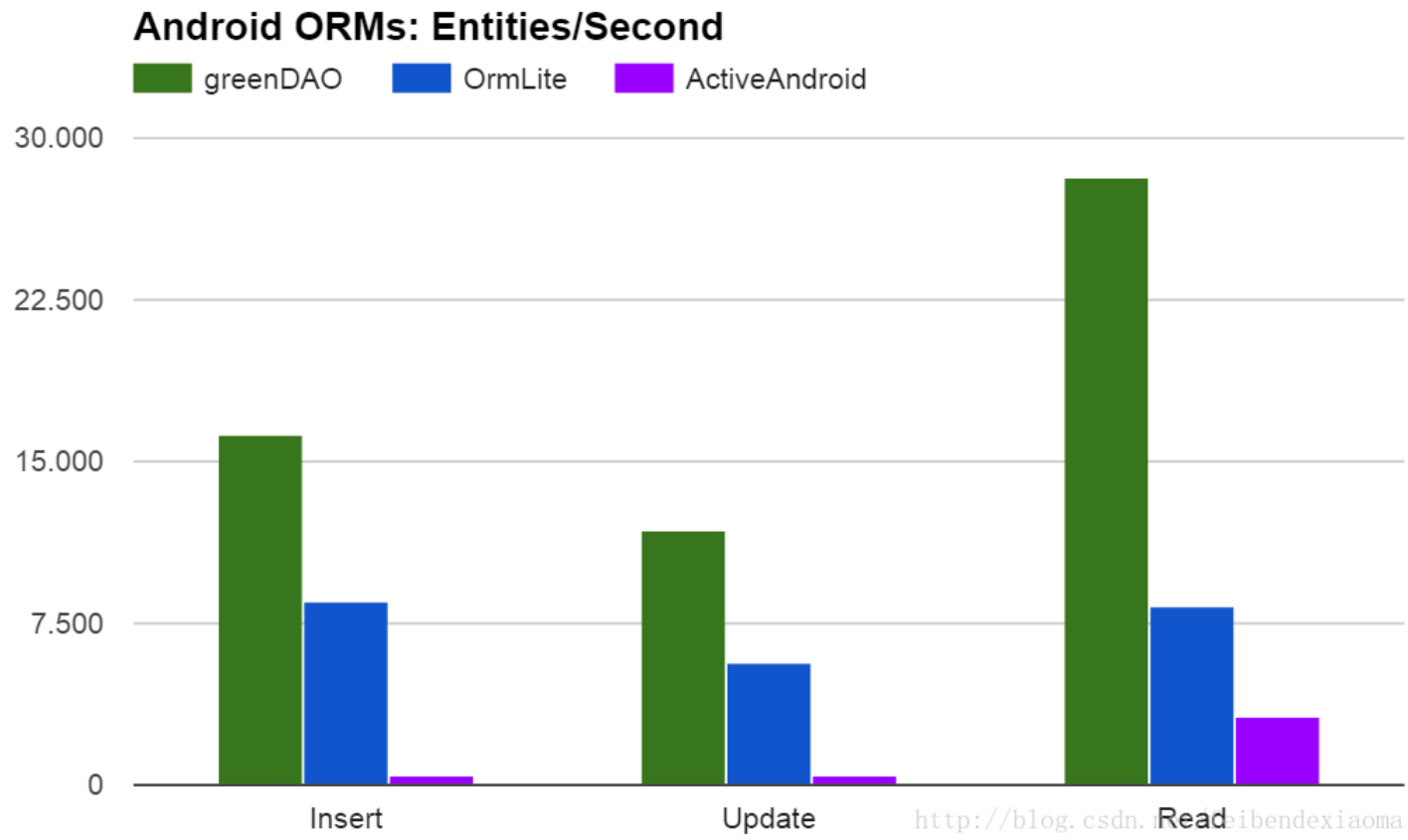
> 通过将Java对象映射到数据库表（称为ORM“对象/关系映射”）。

GreenDao的优缺点

3、GreenDao的优点

> 1. 目前为止性能最高、内存消耗最小
> 2. 使用人数众多，开发者长期维护。
> 3. 使用简单：简洁直观的API
> 4. 轻量：库<150K，它只是纯Java jar（没有CPU依赖的本机部分）
> 5. 快速：可能是由智能代码生成驱动的Android中最快的ORM
> 6. 安全和表达式的查询API：QueryBuilder使用属性常量来避免打字错误
> 7. 强大的连接：跨实体查询，甚至链接复杂关系
> 8. 灵活的属性类型：使用自定义类或枚举来表示实体中的数据
> 9. 支持数据库加密：支持SQLCipher加密数据库
> 10. 代码自动生成
> 11. 开源
> 12. 支持缓存

4、GreenDao的速度对比



GreenDao离线导入

libs中导入GreenDao函数库

1. libs 中的文件 拷入到 androidStudio的libs中 并 Add as Library；
2. 在modle的build.gradle中第一行 添加

```
apply plugin: 'org.greenrobot.greendao'
```

3. 在modle的build.gradle中的android{ }函数中添加

```
greendao {
        schemaVersion 1
        targetGenDir 'src/main/java'
        daoPackage '包名.greendao' //
    }
```

4. 最终实例

```
apply plugin: 'com.android.application'
apply plugin: 'org.greenrobot.greendao'
android {
    xxx
    greendao {
        schemaVersion 1
        targetGenDir 'src'
        daoPackage '包名.greendao'
    }
}

dependencies {
    xxx
    compile files('libs/greendao-3.2.2.jar')
    compile files('libs/greendao-api-3.2.2.jar')
}
```

导入GreenDao gradle插件的离线包

1. 将greenDaoPlugin文件复制到 项目的根目录中(最外层`build.gradle`的同级目录中)
2. 在项目根目录的build.gradle中添加 classpath

```
fileTree(includes:['*.jar'],dir:'greendaoPlugin')
```

3. 最终实例

```
 buildscript {
     repositories {
         xxx
     }
     dependencies {
         classpath 'com.android.tools.build:gradle:2.3.0'
         classpath fileTree(includes:['*.jar'],dir:'greenDaoPlugin')
     }
 }
 xxx


     // in the individual module build.gradle files
 }

}
```

GreenDao的使用

自定义GreenDaoManager

1、GreenDaoManager的作用

> 随时随地可以获取操作所需要的 `DaoSession` 且只创建一次

2、GreenDaoManager的作用

> 1-GreenDaoManager.java

```
public class GreenDaoManager {
    // 数据库名称
    private static final String DB_NAME="greendao";
    private static GreenDaoManager mInstance;
    // DaoMaster
    private DaoMaster daoMaster;
    // DaoSession
    private DaoSession daoSession;
    // 开启日志输出

    public static GreenDaoManager getInstance(){
        if(mInstance==null){
            synchronized (GreenDaoManager.class){
                if(mInstance==null){
                    mInstance =new GreenDaoManager();
                }
            }
        }
        return mInstance;
    }
    private GreenDaoManager(){
        if(mInstance==null){
            DaoMaster.DevOpenHelper devOpenHelper = new DaoMaster.DevOpenHelper(BaseApplicatior
            daoMaster = new DaoMaster(devOpenHelper.getWritableDatabase());
            daoSession = daoMaster.newSession();
            // 开启日志输出
            QueryBuilder.LOG_SQL = true;
            QueryBuilder.LOG_VALUES = true;
        }
    }
    public DaoSession getDaoSession(){
        return daoSession;
    }
    public DaoMaster getDaoMaster(){
        return daoMaster;
    }
}
```

> 2-BaseApplication(用于获得Context)

```
public class  BaseApplication extends  Application
{
    private static Context mContext;

    public static Context getContext(){
        return mContext;
    }

    @Override
    public void onCreate()
    {
        mContext = getApplicationContext();
        super.onCreate();
    }

}
```

实体类Bean

注解

@Entity注解

1. @Entity的作用
    1. @Entity注解标记了一个Java类作为greenDAO一个presistable实体。
    2. 简单理解为，他告诉GreenDao，要根据这个实体类去生成相应的Dao，方便我们去操作，
    3. 同样也相当于将我们的实体类和表做了关联。
2. @Entity配置信息如下：

```
@Entity(
    // 如果你有一个以上的模式，你可以告诉greendao实体属于哪个模式（选择任何字符串作为名称）。
    schema = "myschema",

    // 标志允许实体类可有更新，删除，刷新方法
    active = true,

    // 指定数据库中表的名称。默认情况下，该名称基于实体类名。
    nameInDb = "AWESOME_USERS",

    // 在这里定义多个列的索引
    indexes = {
            @Index(value = "name DESC", unique = true)
    },

    // 如果DAO创建数据库表(默认为true)，则设置标记去标识。如果有多个实体映射到一个表，或者在gre
    createInDb = false,

    // 是否应该生成所有的属性构造函数。一个无args构造函数总是需要的
    generateConstructors = true,

    // 是否生成属性的getter和setter
    generateGettersSetters = true
)
```

@Id注解

1. @Id注解的作用
    1. 选择long / Long属性作为实体ID。
    2. 在数据库方面，它是主要的关键参数autoincrement 是使ID值不断增加的标志（不重复使用旧值-自增长）。

@Property

1. @Property的作用
    1. 允许您定义属性映射到的非默认列名称。
    2. 如果缺少，greenDAO将以SQL-ish方式使用字段名称（大写字母，下划线而不是驼峰命名法，例如 customName将成为 CUSTOM_NAME）。注意：当前只能使用内联常量来指定列名。

@NotNull

1. @NotNull的作用
    1. 该属性在数据库端成为“NOT NULL”列。
    2. 通常使用@NotNull标记原始类型（long，int，short，byte）是有意义的，而具有包装类（Long，Integer，Short，Byte））的可空值。

@Transient

1. @Transient的作用
    1. 标记要从持久性排除的属性，使用这些临时状态等。或者，也可以使用来自Java 的transient关键字。

@Index

1. @Index的作用
    1. 为相应的数据库列创建数据库索引
    2. 名称：如果不喜欢greenDAO为索引生成的默认名称，则可以在此处指定。
    3. 唯一：向索引添加UNIQUE约束，强制所有值都是唯一的。
2. @Index的使用

```
@Entity
public class User {
    @Id private Long id;
    @Index(unique = true)
    private String name;
}
```

@Unique

1. @Unique的作用
    向数据库列添加了一个UNIQUE约束。请注意，SQLite还会隐式地为其创建索引。例子如下：
2. @Unique的使用

```
@Entity
public class User {
    @Id private Long id;
    @Unique private String name;
}
```

使用@Entity生成Bean

```
@Entity
public class VodInfoItemDBBean {
    @Id(autoincrement = true) private Long id;
    @Index(unique = true)
    private String programcode;
    private String programname;
    private String programtype;
    private String contentcode;
    private String seriesprogramcode;
    private String mediaservices;

    //XXX
    @Convert(columnType = String.class,converter = VideoInfoBeanConverter.class)
    private List<VideoInfoBean> videoInfo;

    public static class VideoInfoBeanConverter
            implements PropertyConverter<List<VideoInfoBean>, String>{

        @Override
        public List<VideoInfoBean> convertToEntityProperty(String s) {
            if(s == null){
                return null;
            }else{
                Gson gson = new Gson();
                Log.d("feather", "#convertToEntityProperty = " + s);
                List<String> jsonList = Arrays.asList(s.split("\\|"));
                List<VideoInfoBean> videoInfos = new ArrayList<>();
                for (String json : jsonList) {
                    VideoInfoBean infoBean = gson.fromJson(json, VideoInfoBean.class);
                    videoInfos.add(infoBean);
                }
                return videoInfos;
            }
        }

        @Override
        public String convertToDatabaseValue(List<VideoInfoBean> videoInfoBeen) {
            if (videoInfoBeen == null){
                return null;
            }else{
                Gson gson = new Gson();
                StringBuffer stringBuffer = new StringBuffer();
                for (VideoInfoBean infoBean : videoInfoBeen) {
                    String json = gson.toJson(infoBean);
                    stringBuffer.append(json);
                    stringBuffer.append("|");
                }
                String result = stringBuffer.toString();
                if(result.length() > 0){
                    Log.d("feather", "#convertToDatabaseValue = " + result.substring(0, result.
                    return result.substring(0, result.length() - 1);
                }else{
                    Log.d("feather", "#convertToDatabaseValue = " + result);
                    return result;
```

```
                }
            }
        }
    }


    public static class VideoInfoBean {
        /**
         * videocode : 00000050280000001985
         * mediaservice : 2
         * videotype : 28
         * definition : 2
         * videoelapsedtime :
         * encrypttype : 0
         */

        private String videocode;
        private String mediaservice;
        private String videotype;
        private String definition;
        private String videoelapsedtime;
        private String encrypttype;
    }
}
```

Bean的成员变量为List时该如何处理

1. Bean的成员变量为List时该如何处理
    1. 使用 `@Convert` 注解
    2. 自定义 `PropertyConverter` 进行转换，将 `List` 转换为可以存储的如 `String` 等内容。
2. 使用 `@Convert` 注解

```
@Convert(columnType = String.class,converter = VideoInfoBeanConverter.class)
private List<VideoInfoBean> videoInfo;
```

3. 自定义PropertyConverter

```
public static class VideoInfoBeanConverter
    implements PropertyConverter<List<VideoInfoBean>, String>{

  @Override
  public List<VideoInfoBean> convertToEntityProperty(String s) {
    if(s == null){
        return null;
    }else{
        Gson gson = new Gson();
        Log.d("feather", "#convertToEntityProperty = " + s);
        List<String> jsonList = Arrays.asList(s.split("\\|"));
        List<VideoInfoBean> videoInfos = new ArrayList<>();
        for (String json : jsonList) {
            VideoInfoBean infoBean = gson.fromJson(json, VideoInfoBean.class);
            videoInfos.add(infoBean);
        }
        return videoInfos;
    }
  }

  @Override
  public String convertToDatabaseValue(List<VideoInfoBean> videoInfoBeen) {
    if (videoInfoBeen == null){
        return null;
    }else{
        Gson gson = new Gson();
        StringBuffer stringBuffer = new StringBuffer();
        for (VideoInfoBean infoBean : videoInfoBeen) {
            String json = gson.toJson(infoBean);
            stringBuffer.append(json);
            stringBuffer.append("|");
        }
        String result = stringBuffer.toString();
        if(result.length() > 0){
            Log.d("feather", "#convertToDatabaseValue = " + result.substring(0, result.len
            return result.substring(0, result.length() - 1);
        }else{
            Log.d("feather", "#convertToDatabaseValue = " + result);
            return result;
        }
    }
  }
}
```

基本操作

1、获取操作的Dao(用于增删改查)

```
VodInfoItemDBBeanDao vodInfoDao = GreenDaoManager.getInstance().getDaoSession().getVodInfoIter
```

插入

```
// 1. 直接插入
vodInfoDao.insert(infoItemDBBean);
// 2. 不存在就插入，存在则替换
vodInfoDao.insertOrReplace(infoItemDBBean);
```

删除

```
vodInfoDao.delete(infoItemDBBean);
```

更新

```
vodInfoDao.update(infoItemDBBean);
```

查询

```
// 1、查询单个数据
VodInfoItemDBBean bean
        = vodInfoDao.queryBuilder()
        .where(VodInfoItemDBBeanDao.Properties.Programcode.eq(infoItemDBBean.getProgramcode()))
        .unique();

// 2、查询一组数据
List<VodInfoItemDBBean> tempQueryList
        = vodInfoDao.queryBuilder()
        .where(VodInfoItemDBBeanDao.Properties.Programcode.eq(infoItemDBBean.getProgramcode()))
        .list();
```

数据库升级

1、什么时候需要版本升级更新?

> 当数据库需要新增一个字段或者改变某些字段的属性时，就会需要更新。

2、数据更新的实现思路

> 1. 首先创建临时表（数据格式和原表一模一样）。
> 2. 把当前表的数据插入到临时表中去。
> 3. 删除掉原表，创建新表。
> 4. 把临时表数据插入到新表中去，然后删除临时表。

数据库升级的具体方法

3、数据库升级的实例

> 1-使用 升级辅助类-MigrationHelper

```
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.support.annotation.NonNull;
import android.text.TextUtils;
import android.util.Log;

import org.greenrobot.greendao.AbstractDao;
import org.greenrobot.greendao.database.Database;
import org.greenrobot.greendao.database.StandardDatabase;
import org.greenrobot.greendao.internal.DaoConfig;

import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class MigrationHelper {
    public static boolean DEBUG = false;
    private static String TAG = "MigrationHelper2";
    private static final String SQLITE_MASTER = "sqlite_master";
    private static final String SQLITE_TEMP_MASTER = "sqlite_temp_master";

    public static void migrate(SQLiteDatabase db, Class<? extends AbstractDao<?, ?>>... daoClas
        printLog("【The Old Database Version】" + db.getVersion());
        Database database = new StandardDatabase(db);
        migrate(database, daoClasses);
    }

    public static void migrate(Database database, Class<? extends AbstractDao<?, ?>>... daoClas
        printLog("【Generate temp table】start");
        generateTempTables(database, daoClasses);
        printLog("【Generate temp table】complete");

        dropAllTables(database, true, daoClasses);
        createAllTables(database, false, daoClasses);

        printLog("【Restore data】start");
        restoreData(database, daoClasses);
        printLog("【Restore data】complete");
    }

    private static void generateTempTables(Database db, Class<? extends AbstractDao<?, ?>>... c
        for (int i = 0; i < daoClasses.length; i++) {
            String tempTableName = null;

            DaoConfig daoConfig = new DaoConfig(db, daoClasses[i]);
            String tableName = daoConfig.tablename;
            if (!isTableExists(db, false, tableName)) {
                printLog("【New Table】" + tableName);
                continue;
            }
            try {
```

```
            tempTableName = daoConfig.tablename.concat("_TEMP");
            StringBuilder dropTableStringBuilder = new StringBuilder();
            dropTableStringBuilder.append("DROP TABLE IF EXISTS ").append(tempTableName).ap
            db.execSQL(dropTableStringBuilder.toString());

            StringBuilder insertTableStringBuilder = new StringBuilder();
            insertTableStringBuilder.append("CREATE TEMPORARY TABLE ").append(tempTableName
            insertTableStringBuilder.append(" AS SELECT * FROM ").append(tableName).append(
            db.execSQL(insertTableStringBuilder.toString());
            printLog("【Table】" + tableName +"\n --Columns-->"+getColumnsStr(daoConfig));
            printLog("【Generate temp table】" + tempTableName);
        } catch (SQLException e) {
            Log.e(TAG, "【Failed to generate temp table】" + tempTableName, e);
        }
    }
}

private static boolean isTableExists(Database db, boolean isTemp, String tableName) {
    if (db == null || TextUtils.isEmpty(tableName)) {
        return false;
    }
    String dbName = isTemp ? SQLITE_TEMP_MASTER : SQLITE_MASTER;
    String sql = "SELECT COUNT(*) FROM " + dbName + " WHERE type = ? AND name = ?";
    Cursor cursor=null;
    int count = 0;
    try {
        cursor = db.rawQuery(sql, new String[]{"table", tableName});
        if (cursor == null || !cursor.moveToFirst()) {
            return false;
        }
        count = cursor.getInt(0);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (cursor != null)
            cursor.close();
    }
    return count > 0;
}


private static String getColumnsStr(DaoConfig daoConfig) {
    if (daoConfig == null) {
        return "no columns";
    }
    StringBuilder builder = new StringBuilder();
    for (int i = 0; i < daoConfig.allColumns.length; i++) {
        builder.append(daoConfig.allColumns[i]);
        builder.append(",");
    }
    if (builder.length() > 0) {
        builder.deleteCharAt(builder.length() - 1);
    }
    return builder.toString();
}
```

```
private static void dropAllTables(Database db, boolean ifExists, @NonNull Class<? extends A
    reflectMethod(db, "dropTable", ifExists, daoClasses);
    printLog("【Drop all table】");
}

private static void createAllTables(Database db, boolean ifNotExists, @NonNull Class<? exte
    reflectMethod(db, "createTable", ifNotExists, daoClasses);
    printLog("【Create all table】");
}

/**
 * dao class already define the sql exec method, so just invoke it
 */
private static void reflectMethod(Database db, String methodName, boolean isExists, @NonNul
    if (daoClasses.length < 1) {
        return;
    }
    try {
        for (Class cls : daoClasses) {
            Method method = cls.getDeclaredMethod(methodName, Database.class, boolean.class
            method.invoke(null, db, isExists);
        }
    } catch (NoSuchMethodException e) {
        e.printStackTrace();
    } catch (InvocationTargetException e) {
        e.printStackTrace();
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    }
}

private static void restoreData(Database db, Class<? extends AbstractDao<?, ?>>... daoClass
    for (int i = 0; i < daoClasses.length; i++) {
        DaoConfig daoConfig = new DaoConfig(db, daoClasses[i]);
        String tableName = daoConfig.tablename;
        String tempTableName = daoConfig.tablename.concat("_TEMP");

        if (!isTableExists(db, true, tempTableName)) {
            continue;
        }

        try {
            // get all columns from tempTable, take careful to use the columns list
            List<String> columns = getColumns(db, tempTableName);
            ArrayList<String> properties = new ArrayList<>(columns.size());
            for (int j = 0; j < daoConfig.properties.length; j++) {
                String columnName = daoConfig.properties[j].columnName;
                if (columns.contains(columnName)) {
                    properties.add(columnName);
                }
            }
            if (properties.size() > 0) {
                final String columnSQL = TextUtils.join(",", properties);
```

```
                    StringBuilder insertTableStringBuilder = new StringBuilder();
                    insertTableStringBuilder.append("INSERT INTO ").append(tableName).append("
                    insertTableStringBuilder.append(columnSQL);
                    insertTableStringBuilder.append(") SELECT ");
                    insertTableStringBuilder.append(columnSQL);
                    insertTableStringBuilder.append(" FROM ").append(tempTableName).append(";")
                    db.execSQL(insertTableStringBuilder.toString());
                    printLog("【Restore data】 to " + tableName);
                }
                StringBuilder dropTableStringBuilder = new StringBuilder();
                dropTableStringBuilder.append("DROP TABLE ").append(tempTableName);
                db.execSQL(dropTableStringBuilder.toString());
                printLog("【Drop temp table】" + tempTableName);
            } catch (SQLException e) {
                Log.e(TAG, "【Failed to restore data from temp table 】" + tempTableName, e);
            }
        }
    }

    private static List<String> getColumns(Database db, String tableName) {
        List<String> columns = null;
        Cursor cursor = null;
        try {
            cursor = db.rawQuery("SELECT * FROM " + tableName + " limit 0", null);
            if (null != cursor && cursor.getColumnCount() > 0) {
                columns = Arrays.asList(cursor.getColumnNames());
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (cursor != null)
                cursor.close();
            if (null == columns)
                columns = new ArrayList<>();
        }
        return columns;
    }

    private static void printLog(String info){
        if(DEBUG){
            Log.d(TAG, info);
        }
    }

}
```

> 2-自定义类继承 `DaoMaster.OpenHelper` - `onUpgrade()` 会在数据库版本提升时调用。

```
/**
 *  1. 自定义  MySQLiteOpenHelper继承  DaoMaster.OpenHelper
 *  2. 重写更新数据库的方法
 *  3. 当app下的build.gradle  的schemaVersion数据库的版本号改变时，创建数据库会调用onUpgrade更细数
 */
public class MySQLiteOpenHelper extends DaoMaster.OpenHelper {
    /**
     * @param context 上下文
     * @param name    原来定义的数据库的名字    新旧数据库一致
     * @param factory 可以null
     */
    public MySQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factor
        super(context, name, factory);
    }

    /**
     * @param db
     * @param oldVersion
     * @param newVersion 更新数据库的时候自己调用
     */
    @Override
    public void onUpgrade(Database db, int oldVersion, int newVersion) {
        Log.d("flag", "onUpgrade oldVersion=" + oldVersion+" newVersion="+newVersion);
        /**
         *  将db传入，将目录下的所有的Dao.类传入
         */
        MigrationHelper.migrate(db, StudentDao.class, VodInfoItemDBBeanDao.class);
    }
}
```

> 3- 更改Bean 或者 创建Bean
> 4-在 build 中 make Project 后将生成的 Dao文件 添加到 MySQLiteOpenHelper的migrate(db, ..., NewBeanDao.class) 中
> 5-更改 app目录 下 build.gradle 的 数据库版本号

```
greendao {
    schemaVersion 2 //提升了版本号
    targetGenDir 'src'
    daoPackage 'com.zte.iptvclient.android.greendao' //
}
```

> 6-进行更新(需要新旧数据库的名称一致)

```
MySQLiteOpenHelper helper = new MySQLiteOpenHelper(this,"MyGreenDb.db",null);
DaoMaster daoMaster = new DaoMaster(helper.getWritableDatabase());
```