# GreenDao简单使用

　　greenDAO是Android的对象/关系映射（ORM）工具。它为关系数据库SQLite提供了面向对象的接口。像greenDAO这样的ORM工具可以为您完成许多重复性任务，并为您的数据提供简单的界面。

# 使用GreenDao的优点

1 只需要定义数据模型，GreenDao框架将创建数据实例和DAO（数据访问对象)，能够节省部分代码

2使用GreenDao大多数实体可以以每秒几千个实体的速率进行插入，更新和加载

3.GreenDao支持加密数据库来保护敏感数据

4.微小的依赖库，GreenDao的关键依赖库大小不超过100kb

5.如果需要，实体可以被激活。而活动实体可以透明的解析关系，并且有更新/删除/刷新方法，以便访问持久性功能

6.GreenDao允许你将协议缓冲区对象直接保存到数据库中，如果你通过protobuf通话到你的服务器则不需要另一个映射。常规实体的所有持久性操作都可以用于protobuf对象。

7.自动生成代码，我们需要关注实体类以及Dao，因为GreenDao已经帮我们生成了。

8.开源

# GreenDao对外提供的核心类

1 DaoMaster

　　保存数据库对象 SQLiteDatabase 并管理特定模式的Dao类。它具有静态方法来创建表或将他们删除。其内部类OpenHelper和DevOpenHelper时SQLite数据库中创建模式的SQLiteOpenHelper实现

2 DaoSession

　　管理特定模式的所有可用Dao对象，可以使用其中一个getter方法获取，DaoSession还为实体提供了一些通用的持久性方法如插入、加载、更新、刷新、删除。最后Daosession对象也跟踪一个身份范围

3 Dao层

　　数据访问对象Dao持续存在并查询实体。对于每个实体，GreenDao生成一个Dao，它比DaoSesssion有更多的持久化方法，例如：count，loadAll，insertInTx

4. 实体

　　持久对象，通常实体时使用标准java属性如POJO或JavaBean来表示数据库的对象

# 关于注解的解释

Entity注释将Java类标记为greenDAO的可预设实体。即生成数据库中的一个表

Id注释选择long / Long属性作为实体ID。在数据库方面，它是主键。参数autoincrement是一个标志，用于使ID值不断增加（不重用旧值）。

@Property

设置一个非默认关系映射所对应的列名，默认是使用字段名,例如：@Property(nameInDb = “userName”)

@NotNull

设置数据库表当前列不能为空

@Transient

添加此标记后不会生成数据库表的列

@Unique

表名该属性在数据库中只能有唯一值

@ToMany

定义一对多个实体对象的关系

@ToOne

表示一对一关系

@OrderBy

更加某一字段排序 ，例如：@OrderBy(“date ASC”)

1.）实体@Entity注解

schema：告知GreenDao当前实体属于哪个schema

active：标记一个实体处于活动状态，活动实体有更新、删除和刷新方法

nameInDb：在数据中使用的别名，默认使用的是实体的类名

indexes：定义索引，可以跨越多个列

createInDb：标记创建数据库表\*\*

2.）基础属性注解

@Id :主键 Long型，可以通过@Id(autoincrement = true)设置自增长

@Property：设置一个非默认关系映射所对应的列名，默认是的使用字段名举例：@Property (nameInDb="name")

@NotNul：设置数据库表当前列不能为空

@Transient：添加次标记之后不会生成数据库表的列

3.)索引注解

@Index：使用@Index作为一个属性来创建一个索引，通过name设置索引别名，也可以通过unique给索引添加约束

@Unique：向数据库列添加了一个唯一的约束

4.）关系注解

@ToOne：定义与另一个实体（一个实体对象）的关系

@ToMany：定义与多个实体对象的关系

# 下面进入GreenDao的使用

## 1 配置环境，添加依赖

在工程目录下build.gradle下dependencies添加插件

复制代码

buildscript **{**

repositories **{** google()  
 jcenter()  
 **}** dependencies **{** classpath "com.android.tools.build:gradle:4.0.1"  
 classpath 'org.greenrobot:greendao-gradle-plugin:3.3.0' // 添加GreenDao插件  
 // NOTE: Do not place your application dependencies here; they belong  
 // in the individual module build.gradle files  
 **}  
}**

在app的build.gradle文件下进行如下配置

apply plugin: 'com.android.application'

apply plugin: 'org.greenrobot.greendao' // greendao  
  
android **{  
 ……** greendao **{** schemaVersion 1 // 指定数据库schema版本号，迁移等操作会用到;  
 daoPackage 'com.example.greendaodemo.dao' // 生成的DAO，DaoMaster和DaoSession的包名，默认是entity所在的包；  
 targetGenDir 'src/main/java' // 生成数据库文件的目录;  
 **}  
   
}**dependencies **{**  
 implementation 'org.greenrobot:greendao:3.2.2' // add library  
**}**

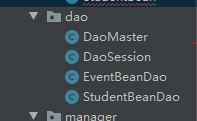
## 2. 新建实体类

用@Entity注解，实体类中的属性即为数据库中对应的字段，最后build项目机会生成相应的代码

@Entity

public class StudentBean {  
 private long id;  
 private String name;  
 private int age;  
 private String gender;  
}

build以后生成如下文件



## 3 GreenDao初始化

在Application中设置一个全局的会话

获取DaoSession

public class MyApplication extends Application {

private static Context *mContext*;  
 private DaoSession daoSession;  
   
 @Override  
 public void onCreate() {  
 super.onCreate();  
 *mContext* = this;  
// initGreenDao();  
 }  
   
 public static Context app() {  
 return *mContext*;  
 }  
   
 private void initGreenDao() {  
 //先通过DaoMaster的DevOpenHelper方法来创建一个数据库  
 DaoMaster.DevOpenHelper helper = new DaoMaster.DevOpenHelper(this,"student.db");  
 //获得一个db  
 SQLiteDatabase db = helper.getWritableDatabase();  
 //新建一个DaoMaster，获得master  
 DaoMaster daoMaster = new DaoMaster(db);  
 //通过master new一个Daosession  
 daoSession = daoMaster.newSession();  
 }  
   
 public DaoSession getDaoSession() {  
 return daoSession;  
 }  
}

## 实现

1.插入

insert(User entity)：插入一条记录, 当指定主键在表中存在时会发生异常，插入的id在数据已经存在会报错：

android.database.sqlite.SQLiteConstraintException: UNIQUE constraint

failed: STUDENT\_BEAN.\_id (code 1555 SQLITE\_CONSTRAINT\_PRIMARYKEY)

insertOrReplace(User entity)：当指定主键在表中存在时会覆盖数据,有该数据时则更新，推荐同步数据库时使用该方法

save(User entity):save 类似于insertOrReplace，区别在于save会判断传入对象的key，有key的对象执行更新，无key的执行插入。当对象有key但并不在数据库时会执行失败.适用于保存本地列表。

//save的源码

public void save(T entity) {

if (hasKey(entity)) {  
 update(entity);  
 } else {  
 insert(entity);  
 }  
}

public void inserOrReplace(StudentBean student){

daoSession.insertOrReplace(student);  
}

StudentBean student = new StudentBean();

student.setId(1);  
student.setName("李四");  
student.setAge(12);  
student.setGender("男");  
//insertData(student);  
inserOrReplace(student);

其他一些插入方法

insertInTx(T... entities)：使用事务在数据库中插入给定的实体。

insertInTx(Iterable<T> entities)：使用事务操作，将给定的实体集合插入数据库。  
insertInTx(Iterable<T> entities, boolean setPrimaryKey)：使用事务操作，将给定的实体集合插入数据库，并设置是否设定主键 。  
insertOrReplaceInTx(T... entities)：使用事务操作，将给定的实体插入数据库，若此实体类存在，则覆盖  
insertOrReplaceInTx(Iterable<T> entities)：使用事务操作，将给定的实体插入数据库，若此实体类存在，则覆盖 。  
insertOrReplaceInTx(Iterable<T> entities, boolean setPrimaryKey)：使用事务操作，将给定的实体插入数据库，若此实体类存在，则覆盖，并设置是否设定主键 。  
insertWithoutSettingPk(T entity)：将给定的实体插入数据库,但不设定主键。  
// 新增数据插入相关API  
save(T entity)：将给定的实体插入数据库  
saveInTx(Iterable<T> entities)：将给定的实体集合插入数据库  
saveInTx(T... entities)：使用事务操作，将给定的实体插入数据库

2.查询

List<StudentBean> students = studentDao.loadAll();

StudentBean students2 = studentDao.load(1L);  
StudentBean students3 = studentDao.loadByRowId(0L);

条件查询

//查询全部

List<User> list = mUserDao.queryBuilder().list();  
 //查询 name等于xyh8的数据  
 List<User> list= mUserDao.queryBuilder().where(UserDao.Properties.Name.eq("xyh8")).list();  
 //查询 name不等于xyh8的数据  
 List<User> list= mUserDao.queryBuilder().where(UserDao.Properties.Name.notEq("xyh8")).list();  
 //like 模糊查询  
//查询 name以xyh3开头的数据  
 List<User> list = mUserDao.queryBuilder().where(UserDao.Properties.Name.like("xyh3%")).list();  
 //between 区间查询 年龄在20到30之间  
 List<User> list = mUserDao.queryBuilder().where(UserDao.Properties.Age.between(20,30)).list();  
 //gt: greater than 半开区间查询，年龄大于18  
 List<User> list = mUserDao.queryBuilder().where(UserDao.Properties.Age.gt(18)).list();  
 //ge: greater equal 半封闭区间查询，年龄大于或者等于18  
 List<User> list = mUserDao.queryBuilder().where(UserDao.Properties.Age.ge(18)).list();  
 //lt: less than 半开区间查询，年龄小于18  
 List<User> list = mUserDao.queryBuilder().where(UserDao.Properties.Age.lt(18)).list();  
 //le: less equal 半封闭区间查询，年龄小于或者等于18  
 List<User> list = mUserDao.queryBuilder().where(UserDao.Properties.Age.le(18)).list();  
//排序  
 //名字以xyh8开头，年龄升序排序  
 List<User> list = mUserDao.queryBuilder()  
 .where(UserDao.Properties.Name.like("xyh8%"))  
 .orderAsc(UserDao.Properties.Age)  
 .list();  
  
 //名字以xyh8开头，年龄降序排序  
 List<User> list = mUserDao.queryBuilder()  
 .where(UserDao.Properties.Name.like("xyh8%"))  
 .orderDesc(UserDao.Properties.Age)  
 .list();

3 更新

update(T entity) ：更新给定的实体

updateInTx(Iterable<T> entities) ：使用事务操作，更新给定的实体  
updateInTx(T... entities)：使用事务操作，更新给定的实体  
studentDao.update(student);  
studentDao.updateInTx(student);

4 删除

//删除全部

mUserDao.deleteAll();  
delete(T entity)：从数据库中删除给定的实体  
deleteByKey(K key)：从数据库中删除给定Key所对应的实体  
deleteInTx(T... entities)：使用事务操作删除数据库中给定的实体  
deleteInTx(<T> entities)：使用事务操作删除数据库中给定实体集合中的实体  
deleteByKeyInTx(K... keys)：使用事务操作删除数据库中给定的所有key所对应的实体  
deleteByKeyInTx(Iterable<K> keys)：使用事务操作删除数据库中给定的所有key所对应的实体   
public void delete(StudentBean student)  
{  
 studentDao.delete(student);  
}

5 封装

public class DaoManager {

private Context mContext;  
  
 //创建数据库的名字  
 private static final String *DB\_NAME* = "MyGreenDaoDb.db";  
  
 //多线程中要被共享的使用volatile关键字修饰 GreenDao管理类  
 private volatile static DaoManager *mInstance*;  
  
 //它里边实际上是保存数据库的对象  
 private static DaoMaster *mDaoMaster*;  
  
 //创建数据库的工具  
 private static DaoMaster.DevOpenHelper *mHelper*;  
  
 //管理gen里生成的所有的Dao对象里边带有基本的增删改查的方法  
 private static DaoSession *mDaoSession*;  
  
  
 private DaoManager() {  
 mContext = MyApplication.*app*();  
 }  
  
 */\*\*  
 \* 单例模式获得操作数据库对象  
 \*  
 \** ***@return*** *\*/* public static DaoManager getInstance() {  
 if (*mInstance* == null) {  
 synchronized (DaoManager.class) {  
 if (*mInstance* == null) {  
 *mInstance* = new DaoManager();  
 }  
 }  
 }  
 return *mInstance*;  
 }  
  
 */\*\*  
 \* 判断是否有存在数据库，如果没有则创建  
 \*  
 \** ***@return*** *\*/* public DaoMaster getDaoMaster() {  
 if (*mDaoMaster* == null) {  
 *mHelper* = new MyDevOpenHelper(mContext, *DB\_NAME*);  
 *mDaoMaster* = new DaoMaster(*mHelper*.getWritableDatabase());  
 }  
 return *mDaoMaster*;  
 }  
  
 */\*\*  
 \* 完成对数据库的添加、删除、修改、查询操作，  
 \*  
 \** ***@return*** *\*/* public DaoSession getDaoSession() {  
 if (*mDaoSession* == null) {  
 if (*mDaoMaster* == null) {  
 *mDaoMaster* = getDaoMaster();  
 }  
 *mDaoSession* = *mDaoMaster*.newSession();  
 }  
 return *mDaoSession*;  
 }  
  
 */\*\*  
 \* 关闭所有的操作，数据库开启后，使用完毕要关闭  
 \*/* public void closeConnection() {  
 closeHelper();  
 closeDaoSession();  
 }  
  
 public void closeHelper() {  
 if (*mHelper* != null) {  
 *mHelper*.close();  
 *mHelper* = null;  
 }  
 }  
  
 public void closeDaoSession() {  
 if (*mDaoSession* != null) {  
 *mDaoSession*.clear();  
 *mDaoSession* = null;  
 }  
 }  
}

6 数据库升级

比如需要在实体类加一个字段 或者 改变字段属性等 就需要版本更新来保存以前的数据了；

思路：创建临时表–>删除原表–>创建新表–>复制临时表数据到新表并删除临时表；这样数据库表的更新就完成了

2.由于升级数据库需要在DevOpenHelper类的onUpgrade()方法里面继续，因此我们需要自定义一个类继承DevOpenHelper重写onUpgrade()方法

3 修改在项目根目录build.gradle文件中配置的数据库版本号(新版本号一定要比老版本大)

greendao {

schemaVersion 6  
 daoPackage 'com.example.greendaodemo1'  
 targetGenDir 'src/main/java'  
}

在StudentBean中新增一个字段我们新增的字段和修改的字段最好为String类型，避免字段不能为null的情况发生

# 目录1

## 目录2

### 目录3

#### 目录4

##### 目录5