OrmLite是一个轻量级的ORM框架，面向JAVA语言。也是时下流行的Android的ORM框架之一。在Android中使用Sqlite数据，如果又不想写SQL，OrmLite或许是个不错的选择。

使用OrmLite，首先要在gradle配置依赖 compile 'com.j256.ormlite:ormlite-android:4.48'

也可以去ormlite官网下载查看文档 <http://ormlite.com/>

1.表创建

然后要要创建一个实体类，对应表结构。OrmLite提供了两个注解，@DatabaseField 代表表列名，@DatabaseTable 表名 tableName值为数据库中表的真实名称。下列的User类必须有一个无参数的构造函数。

需要指定一个字段为唯一标志，必须为int， Integer ，long， Long， Uuid类型

*数据库中的记录通过定义为唯一的特殊字段成为唯一标识。记录不是必须有唯一标识字段当时很多DAO操作（更新、删除、刷新）都需要一个唯一标识字段。这个标识要么用户提供要么数据库自动生成。标识字段有表中唯一的值并且如果你用DAO根据id查询、删除、刷新或者更新指定行的时候他们必须存在。为了配置一个成员变量作为标识成员，你应该使用下面三个设置之一（而且必须使用一个）：@DatabaseField: id, generatedId, generatedIdSequence 。*

@DatabaseField(id = true)指定哪个字段为主键

@DatabaseField(generatedId = true)自动增加的ID

@DatabaseField(generatedIdSequence = true) 设置序列名来匹配已经存在的schema，你可以使用generatedIdSequence指定序列名的值。

这样才可以使用ID来做删除，修改，查询等操作。否则调用相关方法抛出

Cannot query-for-id with class xxx.xxx.xxx.User because it doesn't have an id field相关异常。

1. @DatabaseTable(tableName = "t\_user")
2. public class User {
4. @DatabaseField(generatedId =true)
5. private int id;
7. @DatabaseField
8. private String name;
10. public User(int id, String name) {
11. this.name = name;
12. this.id = id;
13. }
15. public User() {
16. }
18. public String getName() {
19. return name;
20. }
22. public void setName(String name) {
23. this.name = name;
24. }
26. public int getId() {
27. return id;
28. }
30. public void setId(int id) {
31. this.id = id;
32. }
33. }

Android中使用Sqlite需要继承自SQLiteOpenHelper,要使用ormlite需要继承OrmLiteSqliteOpenHelper,来实现一些创建数据库，创建表，更新表的操作。TableUtils是个工具类，主要提供表的创建，表的移除等操作。

1. public class DbHelper extends OrmLiteSqliteOpenHelper {
3. private DbHelper(Context context) {
4. *//参数:上下文,数据库名称,cursor factory,数据库版本.*
5. super(context, "test.db", null, 1);
6. }
8. *//第一次操作数据库时候,被调用*
9. @Override
10. public void onCreate(SQLiteDatabase sqLiteDatabase, ConnectionSource connectionSource) {
11. try {
12. TableUtils.createTable(connectionSource, User.class);
13. } catch (SQLException e) {
14. e.printStackTrace();
15. }
16. }
18. *//当数据库版本升级的时候,被调用*
19. @Override
20. public void onUpgrade(SQLiteDatabase sqLiteDatabase, ConnectionSource connectionSource, int i, int i1) {
21. try {
22. *//这里只是粗暴的移除了旧表,新建新表,这样会导致数据丢失,现实一般不这么做*
23. TableUtils.dropTable(connectionSource, User.class, true);
24. onCreate(sqLiteDatabase, connectionSource);
25. } catch (SQLException e) {Orm
26. e.printStackTrace();
27. }
28. }
30. *//实现一个单例返回DbHelper实例*
31. private static DbHelper helper;
33. public static DbHelper getHelper(Context context) {
34. if (helper == null) {
35. helper = new DbHelper(context);
36. }
37. return helper;
38. }
39. }

2.表的增删改查

首先通过上述的单例类获取一个OrmLiteSqliteOpenHelper的实例，该类中有个getDao(Class<T> clazz)方法可以获取到对应实体的dao对象,参数clazz为表对应的实体的class对象，例如User.class 。

通过getDao返回一个Dao<T, ID>的实例，dao中有增删改查的方法。

从helper获取一个dao的实例

1. private Dao<User, Integer> userDao;
3. public UserDao() {
4. init();
5. }
7. private void init() {
8. DbHelper dbHelper = DbHelper.getHelper(ContextProvider.getApplicationContext());
9. try {
10. userDao = dbHelper.getDao(User.class);
11. } catch (SQLException e) {
12. e.printStackTrace();
13. }
14. }

在表中添加一条记录

public int create(T data) throws SQLException;

在表中添加一条记录，如果表不存在这条数据,根据设置的主键来判断是否存在

public T createIfNotExists(T data) throws SQLException;

在表中添加一条记录，如果存在则更新主键对应的一条记录，

public CreateOrUpdateStatus createOrUpdate(T data) throws SQLException;

1. User user = new User();
2. userDao.create(user);
3. userDao.createOrUpdate(user);
4. userDao.createIfNotExists(user);

删除

根据传入实体删除

public int delete(T data) throws SQLException;

根据ID删除

public int deleteById(ID id) throws SQLException;

根据集合删除

public int delete(Collection<T> datas) throws SQLException;

根据id集合删除

public int deleteIds(Collection<ID> ids) throws SQLException;

1. userDao.deleteIds(ids);
2. userDao.deleteById(id);
3. userDao.delete(user);
4. userDao.delete(list);

更新

//根据传入的实体更新数据，ID为唯一标志

public int update(T data) throws SQLException;

//更新ID,其他值不变

public int updateId(T data, ID newId) throws SQLException;

1. mDao.update(new User(1, "update"));
2. *//更新id指定行的数据*
4. mDao.updateId(new User(mId, mName), 10000);
5. *//把当前的行id更新为10000*

查询

根据唯一标志id检索一条记录，如果id为

public T queryForId(ID id) throws SQLException;

查询匹配到的所有行中的第一个

public T queryForFirst(PreparedQuery<T> preparedQuery) throws SQLException;

返回表中所有条目，可导致大量数据导入内存，应该使用iterator方法来代替此方法

public List<T> queryForAll() throws SQLException;

查询指定字段value等于查询值的行： where fieldName ＝ value

public List<T> queryForEq(String fieldName, Object value) throws SQLException;

匹配传入实体（字段不能为默认值，null，false，0，0.0等）的每个字段的值，每个条件进行and操作，返回的结果，可能导致SQL quote escaping

public List<T> queryForMatching(T matchObj) throws SQLException;

同上述方法，不会导致SQL quote escaping

public List<T> queryForMatchingArgs(T matchObj) throws SQLException;

根据传入的字段与value值的map匹配查询

public List<T> queryForFieldValues(Map<String, Object> fieldValues) throws SQLException;

 根据传入的字段与value值的map匹配查询

public List<T> queryForFieldValuesArgs(Map<String, Object> fieldValues) throws SQLException;

查询与传入实体id相等的数据行

public T queryForSameId(T data) throws SQLException;

1. mDao.queryForAll();
2. mDao.queryForId(mId);