GreenDao

HOLLE_karry 关注
0.492 2020.04.02 11:31:51 字数 1,609 阅读 4,207

1.SQL语句分类

> DDL数据定义语言
> DML数据操作语言
> DCL数据控制语言
> DQL数据查询语言

2.SQL语句

> ①创建数据库：create database 库名；
> 查看所有数据库：show databases;
> 使用数据库：use 库名；
> 删除数据库：drop database 库名；

HOLLE_karry 关注
总资产17 (约1.26元)

推荐阅读

②查看该数据库中的所有表：show tables;

创建表：create table 表名（列名 类型，列名 类型）；

查看表结构：desc 表名；

删除表：drop table 表名；

③插入数据：insert into 表名（列名）values（值）；

修改数据：update 表名 set 列名=值 where=条件；

删除数据：delete from 表名 where=条件；

查看数据：select * from 表名；

④修改表

添加列：alter table 表名 add (列名 类型);

修改列类型：alter table 表名 modify 列名 类型；

修改列名：alter table 表名 change 旧列名 新列名 类型；

删除列：alter table 表名 drop 列名；

修改表名称：alter table 旧表名 rename to 新表名；

条件查询：

```
1.查询指定列
    select sid,sname from stu;
2.查询性别为女，并且年龄50的记录
    select * from stu where gender='female' and age=50;   --and
3.查询学号为S_1001，或者姓名为liSi的记录
    select * from stu where sid='S_1001' or sname='liSi';  --or
4.查询学号为S_1001，S_1002，S_1003的记录
    select * from stu where sid in ('S_1001','S_1002','S_1003');  -- in()
```

```
5.查询学号不是S_1001，S_1002，S_1003的记录
    select * from stu where sid not in ('S_1001','S_1002','S_1003');   -- not
6.查询年龄为null的记录
    select * from stu where age is null;
7.查询年龄在20到40之间的学生记录          区间范围：between and
    select * from stu where age between 20 and 40;
    select * from stu where age >= 20 and age<= 40;
8.查询性别非男的学生记录
    select * from stu where  gender != 'male' or gender is null;
9.查询姓名不为null的学生记录
    select * from stu where sname is not null;
```

模糊查询：

使用 like 关键字，其中“”*匹配任何一个字符，3个*“__”代表3个任意字符， “%”匹配0-n个任何字符

```
  1、查询名称由5个字母组成的学生记录
      select * from stu where sname like '_____';
  2.查询名称中第五个字母是‘u’的学生记录
     select * from stu where sname like '____u';
 3、查询名称以‘z’开头的学生记录
     select * from stu where sname like 'z%';  --中间带z的   %z%     以u结尾的 %u
```

排序：

```
select * from 表名 order by age； 默认升序排序
select * from 表名 order by age asc; 按照年龄升序排序
select * from 表名 order by age desc； 按照年龄 降序排序
select * from 表名 order by 字段名  如果是 字符串，按照首字母排序
```

聚合函数：

```
    SELECT COUNT(*) FROM stu;        求表中数据的行数
    SELECT MAX(age) FROM stu;        求年龄中的最大值
    SELECT MIN(age) FROM stu;        求年龄中的最小值
    SELECT SUM(age) FROM stu;        求所有年龄之和
    SELECT AVG(age) FROM stu;        求年龄的平局值
    SELECT COUNT(age) FROM stu;      求有年龄的人数
    SELECT AVG(age) FROM stu WHERE age>50;   求大于50岁的人的平均年龄
```

分组查询：

```
    Where 是分组前筛选
    HAVING 是分组后筛选
    1.group by 子句
    格式：group by 字段名
    功能：按照指定列进行分组查询、针对于重复性的条件
    1.2 HAVING子句
    Where  是分组前筛选
    HAVING 是分组后筛选
    Where是对分组前的记录进行筛选，如果某行记录没有满足where子句的条件，那么这行记录不会参加分组：而h
```

约束：

```
    约束插入数据的时候一些特殊规定
        insert into stu (name, sex) values ('tuantuan', 'nan');
    1.主键约束:
        创建一个表是,必须有一列(一个字段)要求不能为空,并且所有内容不能重复
        主键一般为int类型， 名字叫做id
```

```
        create table stus (id int primary key, name text);
        insert into stus values(1, 'zhangsan');
        select * from stus;
    2.自增,往往与主键一起使用
        create table stus (id int primary key auto_increment, name text);
    3.not null 非空
    4.唯一    UNIQUE 输入的内容不能重复,多行数据中,这一列数据不能重复
```

3.GreenDao的概述和特点

•概述：

将面向对象编程语言里的对象与数据库关联起来的一种技术，而greenDao就是实现这种技术之一，所以说greenDao其实就是一种将java object 与SQLite Database关联起来的桥梁

•特点：

最高效而且还在迭代的关系型数据库

> 存取速度快： 每秒中可以操作数千个实体
>
> 支持数据库加密：支持android原生的数据库SQLite，也支持SQLCipher
>
> 轻量级：greenDao的代码库仅仅100k大小
>
> 激活实体：处于激活状态下的实体可以有更多操作方法
>
> 支持缓存：将使用过的实体存在缓存中，下次使用时可以直接从缓存中取
>
> 代码自动生成：greenDao 会根据modle类自动生成实体类(entities)和Dao对象，并且Dao对象是根据entities类量身定做的并且一 一对应

4.GreenDao配置依赖

•官网

•配置

⑴工程配置:添加插件

```
buildscript {
    repositories {
        google()
        jcenter()
        mavenCentral()//添加的代码
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.4.1'
        classpath 'org.greenrobot:greendao-gradle-plugin:3.2.2'//生产代码插件
    }
}
```

⑵项目配置:添加插件

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

⑶项目配置:添加依赖

```
implementation 'org.greenrobot:greendao:3.2.2'
```

⑷初始化GreenDao配置

```
greendao{
        schemaVersion 1//版本号
        daoPackage 'com.example.greendao.dao'//数据库全路径
        targetGenDir 'src/main/java'//存放位置
 }
```

5.GreenDao的使用步骤

> ①配置文件中的设置
>
> ②设置Green中的DaoMaster/DaoSession/实体类Dao生成路径
>
> ③设置实体类
>
> ④文件生成 `Build-->ReBuild Project`
>
> ⑤在Application类中完成内容配置(或者使用工具类)
>
> Application有自己的生命周期，OnCreate方法必须首先被调用
>
> ⑴本类对象的获取
>
> ⑵DaoMaster、DaoSession对象的获取
>
> ⑶提供方法，获取DaoSession对象
>
> 需要注意：必须在AndroidManifest.xml文件完成配置
>
> <application
>
> android:name=".App">
>
> </application>

⑥获取实体类Dao对象

XXXDao xxxdao = App.getInstance().getDaoSession().getXXXDao();

xxxdao.增删改查();

6.GreenDao的注解

```
@Entity
定义实体
@Query
用于拼接Url路径后面的查询参数，一个@Query相当于拼接一个参数，多个参数中间用，隔开。
@QueryMap
主要的效果等同于多个@Query参数拼接,参数以map的形式传入,以Map的键值对为参数传入，只用于GET请求(@Quer
@Field
用法类似于@Query 但只用于POST请求
@FieldMap
和QueryMap用法一样以Map形式传入键值对拼接请求参数
只用于POST请求,必须添加@FormUrlEncoded要不然会报异常(@Field, @FieldMap)
@FormUrlEncoded
表示这个请求为表单请求，如果在使用POST请求时不添加这行会报异常Java.lang.IllegalArgumentException
@Path
用于替换Url路径中的变量字符。可以使用在Url里添加{page}  然后在参数里@Path("page") 类型 参数名 声明
@Url
是动态的Url请求数据的注解
@Multipart
标记这个请求用于文件传输,可以用@Part("字段名") Multipart.Part 参数名 来传入文件参数,也可以用 Requ
@Id
注解选择 long / Long 属性作为实体ID。在数据库术语中，它是主键。参数自动增量，是使ID值不断增加（不会这
@Property
让你定义一个非默认的列名，其属性映射到。如果不存在，greenDAO将在SQL杂交方式使用字段名（大写，下划线，
@Transient
表明这个字段不会被写入数据库，只是作为一个普通的java类字段，用来临时存储数据的，不会被持久化
@NotNull
```

```
不为null
@Unique
唯一约束
@nameInDb
在数据库中的名字，如不写则为实体中类名
@indexes
索引
@schema
指定架构名称为实体
@active
无论是更新生成都刷新
@ToMany
一对多
@OrderBy
排序
@ToOne
一对一
@generated
由greendao产生的构造函数或方法
```

7.GreenDao对外提供的核心类

> ①DaoMaster保存数据库对象并管理特定模式的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是在SQLite数据库中创建模式的SQLiteOpenHelper实现。
> ②DaoSession管理特定模式的所有可用Dao对象，您可以使用其中一个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，刷新和删除。最后，DaoSession对象也跟踪一个身份范围。
> ③BeanDao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DaoSession有更多的持久化方法，例如：count,loadAll和insertInTx。
>
> ④实体
>
> 持久对象，通常实体是使用标准Java属性(如POJO或JavaBean)来表示数据库的对象。
>
> 1、DevOpenHelper：创建SQLite数据库的SQLiteOpenHelper的具体实现。
>
> 2、DaoMaster：GreenDao的顶级对象，作为数据库对象、用于创建表和删除表。
>
> 3、DaoSession：管理所有的Dao对象，Dao对象中存在着增删改查等API。

8.⑴使用核心步骤

```
//1.创建数据库
//数据库的名字，高版本手机必须以db结尾
DaoMaster.DevOpenHelper devOpenHelper = new DaoMaster.DevOpenHelper(MyApp.getMyApp(),
//2.获取读写对象
DaoMaster daoMaster = new DaoMaster(devOpenHelper.getWritableDatabase());
//3.获取管理器类
DaoSession daoSession = daoMaster.newSession();
//4.获取表对象
studentDao = daoSession.getStudentDao();
daoSession.clear(); // 清除整个session，没有缓存对象返回
```

⑵完整步骤（APP）

```
import android.app.Application;
import android.database.sqlite.SQLiteDatabase;
import com.example.greendao.db.DaoMaster;
import com.example.greendao.db.DaoSession;

```

```java
public class BaseApp extends Application {
    private static BaseApp sInstance;
    private DaoMaster.DevOpenHelper mHelper;
    private DaoMaster mDaoMaster;
    private DaoSession mDaoSession;
    @Override
    public void onCreate() {
        super.onCreate();
        //本类对象
        sInstance=this;
        //创建数据库以及创建数据表
        setDatabase();
    }
    private void setDatabase() {
         //1.创建数据库
        mHelper = new DaoMaster.DevOpenHelper(this, "mydb", null);
        //2.获取读写对象
        SQLiteDatabase db = mHelper.getWritableDatabase();
        db.disableWriteAheadLogging();
        //3.获取管理器类
        mDaoMaster = new DaoMaster(db);
        //4.获取表对象
        mDaoSession=mDaoMaster.newSession();
    }
    public static BaseApp getInstance(){
        return sInstance;
    }
    public DaoSession getDaoSession(){
        return mDaoSession;
    }
}
```

⑶插入数据

```
private void insert() {
        //插入一条数据,相同主键的数据只能插入一次
        //一般不会使用insert,一般使用insertInTx
        beanDao.insert(new Bean(...));
        //插入或替换多条数据
        //插入或者替换,如果没有,插入,如果有,替换
        beanDao.insertOrReplaceInTx(beans);
        ArrayList<Bean> list = new ArrayList<>();
        for (int i = 0; i < 20; i++) {
            list.add(new Bean(null, "张三" + i, "" + i));
        }
        //插入多条数据
        beanDao.insertInTx(list);
    }
```

(4)删除数据

```
private void delete() {
        //根据key删除数据
        beanDao.deleteByKey(6L);
        //删除某条数据
        beanDao.delete(bean);
        //删除所有数据
        beanDao.deleteAll();
        //删除多条数据
        beanDao.deleteInTx();
    }
```

(5)修改数据

```
private void update() {
        //修改指定的数据
        Bean bean = new Bean(3L, "李四", "66");
        beanDao.update(bean);
        //修改多条数据
        beanDao.updateInTx();
}
```

⑹查询数据

> 常用操作符：
> •eq()：==
> •noteq()：!=
> •gt()： >
> •lt()：<
> •ge：>=
> •le:<=
> •like()：包含
> •between：俩者之间
> •in：在某个值内
> •notIn：不在某个值内

```
private void query() {
        //查询所有数据
```

```
        List<Bean> list = beanDao.queryBuilder().list();
        tv.setText(list.toString());
        //模糊查询
        List<Bean> list = beanDao.queryBuilder().where(BeanDao.Properties.Name.like("%
        tv.setText(list.toString());
        //条件查询
        //精确查询
        Bean bean = beanDao.queryBuilder().where(BeanDao.Properties.Id.eq(12)).unique(
        tv.setText(bean.toString());
        //区间查询
        //gt 大于 大于5
        List<Bean> list = beanDao.queryBuilder().where(BeanDao.Properties.Id.gt(10)).l
    }
```

9.数据库的升级

⑴背景

> 在版本迭代时，我们经常需要对数据库进行升级，而GreenDAO默认的DaoMaster.DevOpenHelper在进行数据升级时，会把旧表删除，然后创建新表，并没有迁移旧数据到新表中，从而造成数据丢失。

⑵原理

> ①创建一张新表
>
> ②旧表数据拷贝一份到新表

> ③删除旧表

❤注意：

> •新增的字段或修改的字段，其变量类型应使用基础数据类型的包装类
> •升级后，新增的字段和修改的字段，都会被赋予null值
> •数据库版本号只能增，不能减
> •表一旦创建成功，不能再添加字段
> •实体类如果要序列化 加 public static final long serialVersionUID = 1L;

⑶步骤：

第一步：在Bean类中添加字段，重新Build—>Make Project进行生成

第二步：复制MigrationHelper到项目，

https://github.com/yuweiguocn/GreenDaoUpgradeHelper中的MigrationHelper，它主要是通过创建一个临时表，将旧表的数据迁移到新表中

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

import androidx.annotation.NonNull;

import org.greenrobot.greendao.AbstractDao;
```

```
10  import org.greenrobot.greendao.database.Database;
11  import org.greenrobot.greendao.database.StandardDatabase;
12  import org.greenrobot.greendao.internal.DaoConfig;
13
14  import java.lang.ref.WeakReference;
15  import java.lang.reflect.InvocationTargetException;
16  import java.lang.reflect.Method;
17  import java.util.ArrayList;
18  import java.util.Arrays;
19  import java.util.List;
20
21  public class MigrationHelper {
22      public static boolean DEBUG = false;
23      private static String TAG = "MigrationHelper";
24      private static final String SQLITE_MASTER = "sqlite_master";
25      private static final String SQLITE_TEMP_MASTER = "sqlite_temp_master";
26
27      private static WeakReference<ReCreateAllTableListener> weakListener;
28
29      public interface ReCreateAllTableListener{
30          void onCreateAllTables(Database db, boolean ifNotExists);
31          void onDropAllTables(Database db, boolean ifExists);
32      }
33
34      public static void migrate(SQLiteDatabase db, Class<? extends AbstractDao<?, ?>>..
35          printLog("【The Old Database Version】" + db.getVersion());
36          Database database = new StandardDatabase(db);
37          migrate(database, daoClasses);
38      }
39
40      public static void migrate(SQLiteDatabase db, ReCreateAllTableListener listener, C
41          weakListener = new WeakReference<>(listener);
42          migrate(db, daoClasses);
43      }
44
45      public static void migrate(Database database, ReCreateAllTableListener listener, C
46          weakListener = new WeakReference<>(listener);
```

```
47          migrate(database, daoClasses);
48      }
49
50      public static void migrate(Database database, Class<? extends AbstractDao<?, ?>>..
51          printLog("【Generate temp table】start");
52          generateTempTables(database, daoClasses);
53          printLog("【Generate temp table】complete");
54
55          ReCreateAllTableListener listener = null;
56          if (weakListener != null) {
57              listener = weakListener.get();
58          }
59
60          if (listener != null) {
61              listener.onDropAllTables(database, true);
62              printLog("【Drop all table by listener】");
63              listener.onCreateAllTables(database, false);
64              printLog("【Create all table by listener】");
65          } else {
66              dropAllTables(database, true, daoClasses);
67              createAllTables(database, false, daoClasses);
68          }
69          printLog("【Restore data】start");
70          restoreData(database, daoClasses);
71          printLog("【Restore data】complete");
72      }
73
74      private static void generateTempTables(Database db, Class<? extends AbstractDao<?,
75          for (int i = 0; i < daoClasses.length; i++) {
76              String tempTableName = null;
77
78              DaoConfig daoConfig = new DaoConfig(db, daoClasses[i]);
79              String tableName = daoConfig.tablename;
80              if (!isTableExists(db, false, tableName)) {
81                  printLog("【New Table】" + tableName);
82                  continue;
83              }
```

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

                StringBuilder insertTableStringBuilder = new StringBuilder();
                insertTableStringBuilder.append("CREATE TEMPORARY TABLE ").append(temp
                insertTableStringBuilder.append(" AS SELECT * FROM `").append(tableNam
                db.execSQL(insertTableStringBuilder.toString());
                printLog("【Table】" + tableName +"\n ---Columns-->"+getColumnsStr(daoC
                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();
```

```
121         }
122         return count > 0;
123     }
124 
125 
126     private static String getColumnsStr(DaoConfig daoConfig) {
127         if (daoConfig == null) {
128             return "no columns";
129         }
130         StringBuilder builder = new StringBuilder();
131         for (int i = 0; i < daoConfig.allColumns.length; i++) {
132             builder.append(daoConfig.allColumns[i]);
133             builder.append(",");
134         }
135         if (builder.length() > 0) {
136             builder.deleteCharAt(builder.length() - 1);
137         }
138         return builder.toString();
139     }
140 
141 
142     private static void dropAllTables(Database db, boolean ifExists, @NonNull Class<? 
143         reflectMethod(db, "dropTable", ifExists, daoClasses);
144         printLog("【Drop all table by reflect】");
145     }
146 
147     private static void createAllTables(Database db, boolean ifNotExists, @NonNull Cla
148         reflectMethod(db, "createTable", ifNotExists, daoClasses);
149         printLog("【Create all table by reflect】");
150     }
151 
152     /**
153      * dao class already define the sql exec method, so just invoke it
154      */
155     private static void reflectMethod(Database db, String methodName, boolean isExists
156         if (daoClasses.length < 1) {
157             return;
```

```
        }
        try {
            for (Class cls : daoClasses) {
                Method method = cls.getDeclaredMethod(methodName, Database.class, bool
                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<?, ?>>...
        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 lis
                List<TableInfo> newTableInfos = TableInfo.getTableInfo(db, tableName);
                List<TableInfo> tempTableInfos = TableInfo.getTableInfo(db, tempTableN
                ArrayList<String> selectColumns = new ArrayList<>(newTableInfos.size()
                ArrayList<String> intoColumns = new ArrayList<>(newTableInfos.size());
                for (TableInfo tableInfo : tempTableInfos) {
                    if (newTableInfos.contains(tableInfo)) {
                        String column = `` + tableInfo.name + ``;
                        intoColumns.add(column);
                        selectColumns.add(column);
                    }
```

```
                }
                // NOT NULL columns list
                for (TableInfo tableInfo : newTableInfos) {
                    if (tableInfo.notnull && !tempTableInfos.contains(tableInfo)) {
                        String column = '`' + tableInfo.name + '`';
                        intoColumns.add(column);

                        String value;
                        if (tableInfo.dfltValue != null) {
                            value = "'" + tableInfo.dfltValue + "' AS ";
                        } else {
                            value = "'' AS ";
                        }
                        selectColumns.add(value + column);
                    }
                }

                if (intoColumns.size() != 0) {
                    StringBuilder insertTableStringBuilder = new StringBuilder();
                    insertTableStringBuilder.append("REPLACE INTO `").append(tableName
                    insertTableStringBuilder.append(TextUtils.join(",", intoColumns));
                    insertTableStringBuilder.append(") SELECT ");
                    insertTableStringBuilder.append(TextUtils.join(",", selectColumns)
                    insertTableStringBuilder.append(" FROM ").append(tempTableName).ap
                    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 】" + tempTableNam
            }
        }
    }

```

```
232     private static List<String> getColumns(Database db, String tableName) {
233         List<String> columns = null;
234         Cursor cursor = null;
235         try {
236             cursor = db.rawQuery("SELECT * FROM " + tableName + " limit 0", null);
237             if (null != cursor && cursor.getColumnCount() > 0) {
238                 columns = Arrays.asList(cursor.getColumnNames());
239             }
240         } catch (Exception e) {
241             e.printStackTrace();
242         } finally {
243             if (cursor != null)
244                 cursor.close();
245             if (null == columns)
246                 columns = new ArrayList<>();
247         }
248         return columns;
249     }
250 
251     private static void printLog(String info){
252         if(DEBUG){
253             Log.d(TAG, info);
254         }
255     }
256 
257     private static class TableInfo {
258         int cid;
259         String name;
260         String type;
261         boolean notnull;
262         String dfltValue;
263         boolean pk;
264 
265         @Override
266         public boolean equals(Object o) {
267             return this == o
268                     || o != null
```

```
                && getClass() == o.getClass()
                && name.equals(((TableInfo) o).name);
    }

    @Override
    public String toString() {
        return "TableInfo{" +
                "cid=" + cid +
                ", name='" + name + '\'' +
                ", type='" + type + '\'' +
                ", notnull=" + notnull +
                ", dfltValue='" + dfltValue + '\'' +
                ", pk=" + pk +
                '}';
    }

    private static List<TableInfo> getTableInfo(Database db, String tableName) {
        String sql = "PRAGMA table_info(`" + tableName + "`)";
        printLog(sql);
        Cursor cursor = db.rawQuery(sql, null);
        if (cursor == null)
            return new ArrayList<>();

        TableInfo tableInfo;
        List<TableInfo> tableInfos = new ArrayList<>();
        while (cursor.moveToNext()) {
            tableInfo = new TableInfo();
            tableInfo.cid = cursor.getInt(0);
            tableInfo.name = cursor.getString(1);
            tableInfo.type = cursor.getString(2);
            tableInfo.notnull = cursor.getInt(3) == 1;
            tableInfo.dfltValue = cursor.getString(4);
            tableInfo.pk = cursor.getInt(5) == 1;
            tableInfos.add(tableInfo);
            // printLog(tableName + ":" + tableInfo);
        }
        cursor.close();
```

```
            return tableInfos;
        }
    }
}
```

第三步：新建一个类MyOpenHelper，继承DaoMaster.DevOpenHelper，重写onUpgrade(Database db, int oldVersion, int newVersion)方法，在该方法中使用MigrationHelper进行数据库升级以及数据迁移。

```
public class MyOpenHelper extends DaoMaster.OpenHelper {
     public MyOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory fa
        super(context, name, factory);
    }

    @Override
    public void onUpgrade(Database db, int oldVersion, int newVersion) {
        //把需要管理的数据库表DAO作为最后一个参数传入到方法中
        MigrationHelper.migrate(db, new MigrationHelper.ReCreateAllTableListener() {

            @Override
            public void onCreateAllTables(Database db, boolean ifNotExists) {
                DaoMaster.createAllTables(db, ifNotExists);
            }

            @Override
            public void onDropAllTables(Database db, boolean ifExists) {
                DaoMaster.dropAllTables(db, ifExists);
            }
        },  BeanDao.class);
    }
}

```

然后在BaseApp中，使用MyOpenHelper替代DaoMaster.DevOpenHelper来进行创建数据库等操作

```
mHelper= new MyOpenHelper(BaseApp.getInstance(),"mydb", null);//建库
```

第四步：

修改Module下build.gradle中数据库的版本号schemaVersion ，递增加1即可，最后运行app

```
greendao{
        schemaVersion 2
        daoPackage 'com.example.greendao.db'
        targetGenDir 'src/main/java'
    }
```

HOLLE_karry
总资产17 (约1.26元)　共写了8.2W字　获得228个赞　共28个粉丝

关注

推荐阅读

更多精彩内容 >

第三方数据库GreenDao总结

一.GreenDao的概述以及特点： 基于对象关系的映射方式来操作数据库的框架，提供一个接口通过操作对象的方式操作...

清阳_　阅读 321　评论 0　赞 1

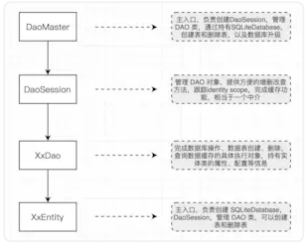
GreenDao学习笔记

greenDAO 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。它...

蕉下孤客　阅读 13,884　评论 18　赞 105

数据库学习之 greenDAO 源码分析

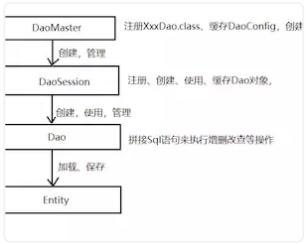
概述 greenDAO 主要使用起来方便的地方，就是使用 @Entity 注解实体类后，只需要build工程，Da...



Kip_Salens　阅读 166　评论 1　赞 1

从使用到源码—GreenDao（理解核心类篇）

引言 在上一篇 “从使用到源码—GreenDao（基本使用篇） ”中，已经从环境配置开始介绍了GreenDao的基...




horseLai 阅读 461 评论 0 赞 2

一篇技术好文之Android数据库 GreenDao的使用完全解析

之前在开发过程中，数据库基本上会使用Litepal或者SQlite自己写，最近换新环境，公司原先使用的数据库就是G...


aserbao 阅读 60,965 评论 41 赞 193