[Android入门——数据存储之SQLite详解与简单应用](http://blog.csdn.net/crazymo_/article/details/48846737)

# 引言

很多时候我们的APP都不能独立于数据之外，常常对数据进行各种操作。**[Android](http://lib.csdn.net/base/android" \o "Android知识库" \t "http://blog.csdn.net/CrazyMo_/article/details/_blank)**为数据存储提供了多种方式，主要有五种：**文件、SharedPreferences、SQLite、网络、内容提供者**。这一篇先总结下关于如何使用SQLite**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/CrazyMo_/article/details/_blank)**。

## 一SQLite 数据库概述

在Android平台上，集成了一个**[嵌入式](http://lib.csdn.net/base/embeddeddevelopment" \o "嵌入式开发知识库" \t "http://blog.csdn.net/CrazyMo_/article/details/_blank)**关系型数据库—SQLite，SQLite3支持 NULL、INTEGER、REAL（浮点数字）、TEXT(字符串文本)和BLOB(二进制对象)数据类型，虽然它支持的类型虽然只有五种，但实际上sqlite3也接受varchar(n)、char(n)、decimal(p,s) 等数据类型，只不过在运算或保存时会转成对应的五种数据类型。 SQLite最大的特点是你可以**保存任何类型的数据到任何字段中，无论这列声明的数据类型是什么，也可以不必在创建表的时候指定字段的长度**，但是作为**主键的字段必须是Integer类型**的。例如：可以在Integer字段中存放字符串，或者在布尔型字段中存放浮点数等。 但有一种情况例外：**定义为INTEGER PRIMARY KEY的字段只能存储64位整数， 当向这种字段中保存除整数以外的数据时，将会产生错误**。 另外， SQLite 在解析CREATE TABLE 语句时，会忽略 CREATE TABLE 语句中跟在字段名后面的数据类型信息，又如下面语句会忽略 name字段的类型信息：   
**CREATE TABLE payout (\_id integer primary key autoincrement, name varchar(20))**其他相关知识欲了解更多，请参见[SQlite官网](http://www.sqlite.org/lang_corefunc.html)

## 二SQLite 数据库支持的SQL语法

SQLite数据库和其他关系型数据一样，也一样支持数据的查询、插入、更新、删除、视图、索引、触发器、连接、聚合函数等常见操作。SQL语法形式和MSSQL、**[Oracle](http://lib.csdn.net/base/oracle" \o "Oracle知识库" \t "http://blog.csdn.net/CrazyMo_/article/details/_blank)**大同小异。其中中括号代表可选项,非必填。

### 1 创建数据表和修改表结构[default 默认值]

#### 1.1 创建数据表，并给字段赋默认值

create if not exists table 表名(字段列表 [default 默认值]);

#### 1.2 创建数据表，并给字段设置check约束

create table if not exists 表名(字段列表 [check(布尔表达式)]);

#### 1.3 创建数据表，并给字段设置unique约束[主键约束 primary key autoincrement]

create table if not exists 表名(字段列表 [unique]);

#### 1.4 修改表名

alter table 旧表名 rename to 新表名;

#### 1.5 向表中添加新列

alter table 表名 add column 新字段名 类型;

### 2 Select查询语法

select \* from 表名 [where 条件子句 group by 分组字句 having ... order by 排序子句]

//如：select name from payout group by name having count(\*)>1

### 3 Insert插入语法

insert into 表名(字段列表) values(值列表)

//如： insert into payout(name, fee) values('旅游',2000)

### 4 Update更新语法

update 表名 set 字段名=值 [where 条件子句]

//如：update payout set fee=2800 where \_id=10

### 5 Delete 删除语法

delete from 表名 [where 条件子句]

//如：delete from payout where id=10

### 6 视图语法

#### 6.1 创建简单视图

create view if not exists 视图名 as sql查询语句结果集合

/\*CREATE VIEW if not exists user\_count AS SELECT count(user\_id) FROM \*user;

\*应用：SELECT \* FROM user\_count

\*/

#### 6.2 创建临时视图

create temp view 视图名 as sql查询语句结果集合

#### 6.3删除视图user\_count

drop view if exists 视图名

//如：drop view if exists user\_count;

### 7 索引语法

#### 7.1 创建索引基于数据表的字段

create index 索引名 on 数据表(字段);

#### 7.2 创建索引基于数据表的字段1、字段2，并指定个字段的排序规则

create index 索引名 on 数据表(字段名1 asc, 字段2 desc);

#### 7.3 创建唯一索引

create unique index 索引名 on 数据表(字段名1 asc, 字段2 desc);

#### 7.4 删除索引

drop index if exists 索引名;

### 8 连接语法

#### 8.1 内连接Inner join 取交集

select xxx from 表名1 inner join 表名2 on 表名1.列名 = 表名2.列名

#### 8.2 外连接Outer join

select xxx from 表名1 left outer join 表名2 on 表名1.列名 = 表名2.列名

## 三 Android中使用SQLite所涉及到对象

Android中集成了SQlite自然也就封装了许多操作数据库的类，掌握以下几个类就能在Android应用中熟练使用SQlite。

### 1 ****SQLiteOpenHelper****

在用户初次使用APP时，需要创建应用使用到的数据库表结构及添加一些初始化记录，当然也能在APP升级的时候，也需要对数据表结构进行更新。在Android系统，封装了一个名为**SQLiteOpenHelper**的抽象类，该类用于对数据库版本进行管理，里面封装了一些方法用户创建数据库、数据表、获取数据库的实例等。

#### 1.1 获取SQLite 数据库实例****getWritableDatabase****()或者****getReadableDatabase****()方法

当调用SQLiteOpenHelper的getWritableDatabase()或者getReadableDatabase()方法获取用于操作数据库的SQLiteDatabase实例的时候，如果数据库不存在，Android系统会自动生成一个数据库，接着调用onCreate()方法，需要注意的是，getWritableDatabase()和getReadableDatabase()方法都可以获取一个用于操作数据库的SQLiteDatabase实例。但getWritableDatabase() 方法以读写方式打开数据库，一旦数据库的**磁盘空间满**了，数据库就**只能读而不能写**，倘若使用的是getWritableDatabase() 方法就会出错。getReadableDatabase()方法**先以读写方式打开数据库，如果数据库的磁盘空间满了，就会打开失败，当打开失败后会继续尝试以只读方式打开数据库。**

#### 1.2 ****OnCreate****和****onUpdate****方法

**onCreate()方法在初次生成数据库时才会被调用**，在onCreate()方法里可以生成数据库表结构及添加一些应用使用到的初始化数据。onUpgrade()方法在数据库的版本发生变化时会被调用，数据库的版本是由程序员控制的，假设数据库现在的版本（是由创建数据库时第三个参数值决定的）是1，由于业务的需要，修改了数据库表的结构，这时候就需要升级软件，升级软件时希望更新用户手机里的数据库表结构，为了实现这一目的，可以把原来的数据库版本设置为2(当然3…任意整数，只要不为0即可)

### 2 ****SQLiteDatabase****

该类封装了一些操作数据库的API，使用该类可以完成对数据进行添加(Create)、查询(Retrieve)、更新(Update)和删除(Delete)操作（这些操作简称为CRUD）。SQLiteOpenHelper的作用之一就是为了得到这个实例，SQLiteDatabase类也支持(占位符？)。

#### 2.1 ****execSQL****()方法可以执行insert、delete、update和CREATE TABLE之类有更改行为的SQL语句，但无返回值。

private final String createStudTb="Create Table Student(\_id integer primary key autoincrement,name varchar(16),sex varchar(2))";

//创建数据表Student

arg0.execSQL(createStudTb);

#### 2.2 ****rawQuery****()方法可以执行select语句，****返回游标对象Cursor****

Cursor cursor = db.rawQuery("select \* from Student where name like ?", new String[]{"%cm%"});

#### 2.3 SQLiteDatabase中专门处理简单插入操作****public long insert (String table, String nullColumnHack, ContentValues values)****

@param table：数据表名

@param nullColumnHack：(SQL doesn't allow inserting a completely empty row, so if initialValues is empty this column will explicitly be assigned a NULL value

)如果第三个参数values 为Null或者元素个数为0， Insert()方法必然要添加一条除了主键之外其它字段为Null值的记录，为了满足这条insert语句的语法， insert语句必须给定一个字段名，如：insert into payout(name) values(NULL)，倘若不给定字段名 ， insert语句就成了这样： insert into payout() values()，显然这不满足标准SQL的语法。对于字段名，建议使用主键之外的字段，如果使用了INTEGER类型的主键字段，执行类似insert into payout(\_id) values(NULL)的insert语句后，该主键字段值也不会为NULL。如果第三个参数values 不为Null并且元素的个数大于0 ，可以把第二个参数设置为null。

@param values：要保存的键值对对象

@return：返回新插入记录的rowid，插入失败返回-1

#### 2.4 SQLiteDatabase中专门处理简单删除操作****public int delete(String table,String whereClause,String[] whereArgs)****

@param whereClause：Where子句，不能包含where关键句@param whereArgs： 替换占位符的实参@return 返回删除的行数

#### 2.5 SQLiteDatabase中专门处理简单的更新操作****int update(String table, ContentValues values, String whereClause, String[] whereArgs)****

#### 2.6 SQLiteDatabase中专门处理简单的查询操作****Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)****

@param table：表名。相当于select语句from关键字后面的部分。如果是多表联合查询，可以用逗号将两个表名分开。

@param columns：要查询出来的列名。相当于select语句select关键字后面的部分。

@param selection：查询条件子句，相当于select语句where关键字后面的部分，在条件子句允许使用占位符“?”

@param selectionArgs：对应于selection语句中占位符的值，值在数组中的位置与占位符在语句中的位置必须一致，否则就会有异常。

@param groupBy：相当于select语句group by关键字后面的部分

@param having：相当于select语句having关键字后面的部分

@param orderBy：相当于select语句order by关键字后面的部分，如：\_id desc, age asc;

@param limit：指定偏移量和获取的记录数，相当于select语句limit关键字后面的部分。

### 3 ****Cursor****游标对象

查询结果的返回集合一般都是以Cursor形式放回的，和其他数据库的游标特点大同小异。

#### 3.1 ****getColumnCount ()**** 返回所有列的总数

#### 3.2 ****getColumnIndex (String columnName)**** 返回指定列的索引，如果不存在返回-1

#### 3.3 ****getCount ()**** 返回Cursor 中的行数

#### 3.4 ****moveToFirst ()**** 移动光标到第一行，如cur.moveToFirst()==false则Cursor为空。

#### 3.5 ****moveToLast ()**** 移动光标到最后一行

#### 3.6 ****moveToNext ()**** 移动光标到下一行

#### 3.7 ****close ()**** 关闭游标，释放资源

#### 3.8 ****getXxx(int columnIndex)****根据列的索引值获取对应的值，其中Xxx代表String、Int、Double、Short等。

### 4 ****ContentValues**** 键值对对象

ContentValues 和 HashTable 类似都是一种存储的机制，但是两者最大的区别就在于：ContentValues 只能存储基本类型的数据，像string、int之类的，不能存储对象这种东西，而HashTable却可以存储对象。操作方法和HasMap大同小异。

public void saveByInsert(Student stu){

SQLiteDatabase db=dbhelper.getWritableDatabase();

ContentValues values=new ContentValues();

values.put("name", stu.getName());

values.put("sex",stu.getSex());

db.insert("Student", null, values);

db.close();

}

## 四 Android中使用SQLite

### 1 继承SQLiteOpenHelper，实现构造函数和onCreate方法。根据实际情况完成相关的业务逻辑

package cmo.learn.service;

import android.content.Context;import android.database.sqlite.SQLiteDatabase;import android.database.sqlite.SQLiteOpenHelper;

public class DBHelper extends SQLiteOpenHelper {

private static final String dbname="STUDENT.db";

private static final int version=1;

private static DBHelper dbHelper;

private final String createStudTb="Create Table Student(\_id integer primary key autoincrement,name varchar(16),sex varchar(2))";//可以不指定字段的类型、长度，因为int类型也可以保存Char类型的创建学生表

/\*\*

\* SQLiteOpenHelper(context, name, factory, version)

\* @param context 上下文对象

\* @param name 数据库名称

\* @param factory 游标工厂对象，为null则取系统默认的游标工厂对象

\* @param version 数据库版本号取值为任意非0整数

\*/

public DBHelper(Context context) {

//super(context, dbname, null, version);//创建DB成功之后会保存在/<package name>/databases/初始版本号为1

super(context, dbname, null, 2);

}

public static DBHelper getInstance(Context context) {

if (dbHelper == null) { //单例模式

dbHelper = new DBHelper(context.getApplicationContext());

}

return dbHelper;

}

@Override

public void onCreate(SQLiteDatabase arg0) {

//当数据库第一次创建时触发,所以可以在这可以创建数据库表和完成一些初始化操作，SQLiteDatabase封装了针对SQLite的所有操作，数据库操作实例类

arg0.execSQL(createStudTb);

}

@Override

public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2) {

//当version值与初始值不同时触发

arg0.execSQL("Alter Table Student add amount integer");

}

}

### 2 通过我们自定实现的SQLiteOpenHelper子类得到SQLiteDatabase 实例，再调用SQLiteDatabase 实例的方法完成数据的操作

package cmo.learn.service;

import java.util.ArrayList;import java.util.List;

import android.content.ContentValues;import android.content.Context;import android.database.Cursor;import android.database.sqlite.SQLiteDatabase;import cmo.learn.domain.Student;

public class StudentDBOperate {

private DBHelper dbhelper;

//要操作数据库操作实例首先得得到数据库操作实例

public StudentDBOperate(Context context){

this.dbhelper=new DBHelper(context);

}

public void save(Student stu){

SQLiteDatabase db=dbhelper.getWritableDatabase();

db.execSQL("Insert into Student(name,sex) Values(?,?)",

new Object[] { stu.getName(),stu.getSex() });

db.close();

}

public void saveByInsert(Student stu){

SQLiteDatabase db=dbhelper.getWritableDatabase();

ContentValues values=new ContentValues();

values.put("name", stu.getName());

values.put("sex",stu.getSex());

db.insert("Student", null, values);

db.close();

}

public void delete2(Integer id){

SQLiteDatabase db=dbhelper.getWritableDatabase();

db.delete("Student", "\_id=?", new String[]{id.toString()});

}

public void delete(Integer id){

SQLiteDatabase db=dbhelper.getWritableDatabase();

db.execSQL("Delete from Student where \_id=?",

new String[] { String.valueOf(id) });

db.close();

}

public void update2(Student stu){

SQLiteDatabase db=dbhelper.getWritableDatabase();

ContentValues values=new ContentValues();

values.put("name", stu.getName());

values.put("sex",stu.getSex());

db.update("Student", values, "\_id=?", new String[]{String.valueOf(stu.get\_id())});

}

public void update(Student stu){

SQLiteDatabase db=dbhelper.getWritableDatabase();

db.execSQL("Update Student Set name=?,sex=? where \_id=?",

new String[] {stu.getName(),stu.getSex(),String.valueOf(stu.get\_id()) });

db.close();

}

public Student query(Integer \_id){

SQLiteDatabase db=dbhelper.getReadableDatabase();

Cursor stuimf=db.rawQuery("Select \_id,name,sex from Student where \_id=?",

new String[] {String.valueOf(\_id)});

if(stuimf!=null){

if(stuimf.moveToFirst()){

int id=stuimf.getInt(stuimf.getColumnIndex("\_id"));

String name=stuimf.getString(stuimf.getColumnIndex("name"));

String sex=stuimf.getString(stuimf.getColumnIndex("sex"));

return new Student(id,name,sex);

}

}

db.close();

return null;

}

public Student query2(Integer \_id){

SQLiteDatabase db=dbhelper.getReadableDatabase();

Cursor stuimf= db.query("Student", new String[]{"\_id","name","sex"}, "\_id=?",new String[]{\_id.toString()}, null, null, null);

if(stuimf!=null){

if(stuimf.moveToFirst()){

int id=stuimf.getInt(stuimf.getColumnIndex("\_id"));

String name=stuimf.getString(stuimf.getColumnIndex("name"));

String sex=stuimf.getString(stuimf.getColumnIndex("sex"));

return new Student(id,name,sex);

}

}

return null;

}

/\*\*

\* 分页查询

\* @param offset 跳过前面多少记录

\* @param maxRecord 每页获取多少记录

\* @return

\*/

public List<Student> getScrollData(int offset,int maxRecord){

SQLiteDatabase db=dbhelper.getReadableDatabase();

List<Student> stulist = new ArrayList<Student>();

Cursor stuimf=db.rawQuery("Select \_id,name,sex from Student order by \_id asc limit ?,?",

new String[] {String.valueOf(offset),String.valueOf(maxRecord)});

if(stuimf!=null){

while(stuimf.moveToNext()){

int id=stuimf.getInt(stuimf.getColumnIndex("\_id"));

String name=stuimf.getString(stuimf.getColumnIndex("name"));

String sex=stuimf.getString(stuimf.getColumnIndex("sex"));

stulist.add(new Student(id,name,sex));

}

return stulist;

}

db.close();

return null;

}

public long getCount(){

long result=0;

SQLiteDatabase db=dbhelper.getReadableDatabase();

Cursor stuimf=db.rawQuery("Select count(\*) from Student", null);

if(stuimf!=null){

//因为只要创建了表，count(\*)最小值为1，即还未插入数据时

stuimf.moveToFirst();

result=stuimf.getLong(0);

}

return result;

}

}

### 3 对数据进行操作测试（这里为了方便使用单元测试）

package cmo.learn.service;

import java.util.ArrayList;import java.util.List;

import android.content.ContentValues;import android.content.Context;import android.database.Cursor;import android.database.sqlite.SQLiteDatabase;import cmo.learn.domain.Student;

public class StudentDBOperate {

private DBHelper dbhelper;

//要操作数据库操作实例首先得得到数据库操作实例

public StudentDBOperate(Context context){

this.dbhelper=new DBHelper(context);

}

public void save(Student stu){

SQLiteDatabase db=dbhelper.getWritableDatabase();

db.execSQL("Insert into Student(name,sex) Values(?,?)",

new Object[] { stu.getName(),stu.getSex() });

db.close();

}

public void saveByInsert(Student stu){

SQLiteDatabase db=dbhelper.getWritableDatabase();

ContentValues values=new ContentValues();

values.put("name", stu.getName());

values.put("sex",stu.getSex());

db.insert("Student", null, values);

db.close();

}

public void delete2(Integer id){

SQLiteDatabase db=dbhelper.getWritableDatabase();

db.delete("Student", "\_id=?", new String[]{id.toString()});

}

public void delete(Integer id){

SQLiteDatabase db=dbhelper.getWritableDatabase();

db.execSQL("Delete from Student where \_id=?",

new String[] { String.valueOf(id) });

db.close();

}

public void update2(Student stu){

SQLiteDatabase db=dbhelper.getWritableDatabase();

ContentValues values=new ContentValues();

values.put("name", stu.getName());

values.put("sex",stu.getSex());

db.update("Student", values, "\_id=?", new String[]{String.valueOf(stu.get\_id())});

}

public void update(Student stu){

SQLiteDatabase db=dbhelper.getWritableDatabase();

db.execSQL("Update Student Set name=?,sex=? where \_id=?",

new String[] {stu.getName(),stu.getSex(),String.valueOf(stu.get\_id()) });

db.close();

}

public Student query(Integer \_id){

SQLiteDatabase db=dbhelper.getReadableDatabase();

Cursor stuimf=db.rawQuery("Select \_id,name,sex from Student where \_id=?",

new String[] {String.valueOf(\_id)});

if(stuimf!=null){

if(stuimf.moveToFirst()){

int id=stuimf.getInt(stuimf.getColumnIndex("\_id"));

String name=stuimf.getString(stuimf.getColumnIndex("name"));

String sex=stuimf.getString(stuimf.getColumnIndex("sex"));

return new Student(id,name,sex);

}

}

db.close();

return null;

}

public Student query2(Integer \_id){

SQLiteDatabase db=dbhelper.getReadableDatabase();

Cursor stuimf= db.query("Student", new String[]{"\_id","name","sex"}, "\_id=?",new String[]{\_id.toString()}, null, null, null);

if(stuimf!=null){

if(stuimf.moveToFirst()){

int id=stuimf.getInt(stuimf.getColumnIndex("\_id"));

String name=stuimf.getString(stuimf.getColumnIndex("name"));

String sex=stuimf.getString(stuimf.getColumnIndex("sex"));

return new Student(id,name,sex);

}

}

return null;

}

/\*\*

\* 分页查询

\* @param offset 跳过前面多少记录

\* @param maxRecord 每页获取多少记录

\* @return

\*/

public List<Student> getScrollData(int offset,int maxRecord){

SQLiteDatabase db=dbhelper.getReadableDatabase();

List<Student> stulist = new ArrayList<Student>();

Cursor stuimf=db.rawQuery("Select \_id,name,sex from Student order by \_id asc limit ?,?",

new String[] {String.valueOf(offset),String.valueOf(maxRecord)});

if(stuimf!=null){

while(stuimf.moveToNext()){

int id=stuimf.getInt(stuimf.getColumnIndex("\_id"));

String name=stuimf.getString(stuimf.getColumnIndex("name"));

String sex=stuimf.getString(stuimf.getColumnIndex("sex"));

stulist.add(new Student(id,name,sex));

}

return stulist;

}

db.close();

return null;

}

public long getCount(){

long result=0;

SQLiteDatabase db=dbhelper.getReadableDatabase();

Cursor stuimf=db.rawQuery("Select count(\*) from Student", null);

if(stuimf!=null){

//因为只要创建了表，count(\*)最小值为1，即还未插入数据时

stuimf.moveToFirst();

result=stuimf.getLong(0);

}

return result;

}

}

## 小结

至此，完成了SQLite的基本应用的所有工作。我们再来回顾下几个对象的主要作用，**SQLiteOpenHelper**用于创建数据库、数据库结构和得到真正操作数据库的实例**SQLiteDatabase**；**SQLiteDatabase**直接完成数据的各种操作；**Cursor**对象用于把查询集合封装起来；而**ContentValus**也是封装数据，把数据列和对应的值封装成为键值对。