使用嵌入式关系型SQLite数据库存储数据

除了可以使用文件或SharedPreferences存储数据，还可以选择使用SQLite数据库存储数据。

在Android平台上，集成了一个嵌入式关系型数据库—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把各种类型的数据保存到任何字段中，而不用关心字段声明的数据类型是什么。例如：可以在Integer类型的字段中存放字符串，或者在布尔型字段中存放浮点数，或者在字符型字段中存放日期型值。 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，如下面语句会忽略 name字段的类型信息：

CREATE TABLE person (personid integer primary key autoincrement, name varchar(20))

SQLite可以解析大部分标准SQL语句，如：

查询语句：select \* from 表名 where 条件子句 group by 分组字句 having ... order by 排序子句

如：select \* from person

select \* from person order by id desc

select name from person group by name having count(\*)>1

分页SQL与mysql类似，下面SQL语句获取5条记录，跳过前面3条记录

select \* from Account limit 5 offset 3 或者 select \* from Account limit 3,5

插入语句：insert into 表名(字段列表) values(值列表)。如： insert into person(name, age) values(‘传智’,3)

更新语句：update 表名 set 字段名=值 where 条件子句。如：update person set name=‘传智‘ where id=10

删除语句：delete from 表名 where 条件子句。如：delete from person where id=10

使用SQLiteOpenHelper对数据库进行版本管理

我们在编写数据库应用软件时，需要考虑这样的问题：因为我们开发的软件可能会安装在很多用户的手机上，如果应用使用到了SQLite数据库，我们必须在用户初次使用软件时创建出应用使用到的数据库表结构及添加一些初始化记录，另外在软件升级的时候，也需要对数据表结构进行更新。那么，我们如何才能实现在用户初次使用或升级软件时自动在用户的手机上创建出应用需要的数据库表呢？总不能让我们在每个需要安装此软件的手机上通过手工方式创建数据库表吧？因为这种需求是每个数据库应用都要面临的，所以在Android系统，为我们提供了一个名为SQLiteOpenHelper的抽象类，必须继承它才能使用，它是通过对数据库版本进行管理来实现前面提出的需求。

为了实现对数据库版本进行管理，SQLiteOpenHelper类提供了两个重要的方法，分别是onCreate(SQLiteDatabase db)和onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)，前者用于初次使用软件时生成数据库表，后者用于升级软件时更新数据库表结构。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，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，一般在软件升级时才需改变版本号，而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1，由于业务的变更，修改了数据库表结构，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(有同学问设置为3行不行？当然可以，如果你愿意，设置为100也行)，并且在onUpgrade()方法里面实现表结构的更新。当软件的版本升级次数比较多，这时在onUpgrade()方法里面可以根据原版号和目标版本号进行判断，然后作出相应的表结构及数据更新。

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。

public class DatabaseHelper extends SQLiteOpenHelper {

//类没有实例化,是不能用作父类构造器的参数,必须声明为静态

private static final String name = "itcast"; //数据库名称

private static final int version = 1; //数据库版本

public DatabaseHelper(Context context) {

//第三个参数CursorFactory指定在执行查询时获得一个游标实例的工厂类,设置为null,代表使用系统默认的工厂类

super(context, name, null, version);

}

@Override public void onCreate(SQLiteDatabase db) {

db.execSQL("CREATE TABLE IF NOT EXISTS person (personid integer primary key autoincrement, name varchar(20), age INTEGER)");

}

@Override public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

db.execSQL("DROP TABLE IF EXISTS person");

onCreate(db);

}

}

上面onUpgrade()方法在数据库版本每次发生变化时都会把用户手机上的数据库表删除，然后再重新创建。一般在实际项目中是不能这样做的，正确的做法是在更新数据库表结构时，还要考虑用户存放于数据库中的数据不会丢失。

使用SQLiteDatabase操作SQLite数据库

Android提供了一个名为SQLiteDatabase的类，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）。对SQLiteDatabase的学习，我们应该重点掌握execSQL()和rawQuery()方法。 execSQL()方法可以执行insert、delete、update和CREATE TABLE之类有更改行为的SQL语句； rawQuery()方法用于执行select语句。

execSQL()方法的使用例子：

SQLiteDatabase db = ....;

db.execSQL("insert into person(name, age) values('传智播客', 4)");

db.close();

执行上面SQL语句会往person表中添加进一条记录，在实际应用中， 语句中的“传智播客”这些参数值会由用户输入界面提供，如果把用户输入的内容原样组拼到上面的insert语句， 当用户输入的内容含有单引号时，组拼出来的SQL语句就会存在语法错误。要解决这个问题需要对单引号进行转义，也就是把单引号转换成两个单引号。有些时候用户往往还会输入像“ & ”这些特殊SQL符号，为保证组拼好的SQL语句语法正确，必须对SQL语句中的这些特殊SQL符号都进行转义，显然，对每条SQL语句都做这样的处理工作是比较烦琐的。 SQLiteDatabase类提供了一个重载后的execSQL(String sql, Object[] bindArgs)方法，使用这个方法可以解决前面提到的问题，因为这个方法支持使用占位符参数(?)。使用例子如下：

SQLiteDatabase db = ....;

db.execSQL("insert into person(name, age) values(?,?)", new Object[]{"传智播客", 4});

db.close();

execSQL(String sql, Object[] bindArgs)方法的第一个参数为SQL语句，第二个参数为SQL语句中占位符参数的值，参数值在数组中的顺序要和占位符的位置对应。

SQLiteDatabase的rawQuery() 用于执行select语句，使用例子如下：  
 SQLiteDatabase db = ....;

Cursor cursor = db.rawQuery(“select \* from person”, null);

while (cursor.moveToNext()) {

int personid = cursor.getInt(0); //获取第一列的值,第一列的索引从0开始

String name = cursor.getString(1);//获取第二列的值

int age = cursor.getInt(2);//获取第三列的值

}

cursor.close();

db.close();

rawQuery()方法的第一个参数为select语句；第二个参数为select语句中占位符参数的值，如果select语句没有使用占位符，该参数可以设置为null。带占位符参数的select语句使用例子如下：

Cursor cursor = db.rawQuery("select \* from person where name like ? and age=?", new String[]{"%传智%", "4"});

Cursor是结果集游标，用于对结果集进行随机访问，如果大家熟悉jdbc， 其实Cursor与JDBC中的ResultSet作用很相似。使用moveToNext()方法可以将游标从当前行移动到下一行，如果已经移过了结果集的最后一行，返回结果为false，否则为true。另外Cursor 还有常用的moveToPrevious()方法（用于将游标从当前行移动到上一行，如果已经移过了结果集的第一行，返回值为false，否则为true ）、moveToFirst()方法（用于将游标移动到结果集的第一行，如果结果集为空，返回值为false，否则为true ）和moveToLast()方法（用于将游标移动到结果集的最后一行，如果结果集为空，返回值为false，否则为true ） 。

除了前面给大家介绍的execSQL()和rawQuery()方法， SQLiteDatabase还专门提供了对应于添加、删除、更新、查询的操作方法： insert()、delete()、update()和query() 。这些方法实际上是给那些不太了解SQL语法的菜鸟使用的，对于熟悉SQL语法的程序员而言，直接使用execSQL()和rawQuery()方法执行SQL语句就能完成数据的添加、删除、更新、查询操作。

Insert()方法用于添加数据，各个字段的数据使用ContentValues进行存放。 ContentValues类似于MAP，相对于MAP，它提供了存取数据对应的put(String key, Xxx value)和getAsXxx(String key)方法， key为字段名称，value为字段值，Xxx指的是各种常用的数据类型，如：String、Integer等。

SQLiteDatabase db = databaseHelper.getWritableDatabase();

ContentValues values = new ContentValues();

values.put("name", "传智播客");

values.put("age", 4);

long rowid = db.insert(“person”, null, values);//返回新添记录的行号，与主键id无关

不管第三个参数是否包含数据，执行Insert()方法必然会添加一条记录，如果第三个参数为空，会添加一条除主键之外其他字段值为Null的记录。Insert()方法内部实际上通过构造insert SQL语句完成数据的添加，Insert()方法的第二个参数用于指定空值字段的名称，相信大家对该参数会感到疑惑，该参数的作用是什么？是这样的：如果第三个参数values 为Null或者元素个数为0， 由于Insert()方法要求必须添加一条除了主键之外其它字段为Null值的记录，为了满足SQL语法的需要， insert语句必须给定一个字段名，如：insert into person(name) values(NULL)，倘若不给定字段名 ， insert语句就成了这样： insert into person() 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 person(personid) 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。

delete()方法的使用：

SQLiteDatabase db = databaseHelper.getWritableDatabase();

db.delete("person", "personid<?", new String[]{"2"});

db.close();

上面代码用于从person表中删除personid小于2的记录。

update()方法的使用：

SQLiteDatabase db = databaseHelper.getWritableDatabase();

ContentValues values = new ContentValues();

values.put(“name”, “传智播客”);//key为字段名，value为值

db.update("person", values, "personid=?", new String[]{"1"});

db.close();

上面代码用于把person表中personid等于1的记录的name字段的值改为“传智播客”。

query()方法实际上是把select语句拆分成了若干个组成部分，然后作为方法的输入参数：

SQLiteDatabase db = databaseHelper.getWritableDatabase();

Cursor cursor = db.query("person", new String[]{"personid,name,age"}, "name like ?", new String[]{"%传智%"}, null, null, "personid desc", "1,2");

while (cursor.moveToNext()) {

int personid = cursor.getInt(0); //获取第一列的值,第一列的索引从0开始

String name = cursor.getString(1);//获取第二列的值

int age = cursor.getInt(2);//获取第三列的值

}

cursor.close();

db.close();

上面代码用于从person表中查找name字段含有“传智”的记录，匹配的记录按personid降序排序，对排序后的结果略过第一条记录，只获取2条记录。

query(table, columns, selection, selectionArgs, groupBy, having, orderBy, limit)方法各参数的含义：

**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。

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

**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“?”

**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。

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

**having**：相当于select语句having关键字后面的部分

**orderBy**：相当于select语句order by关键字后面的部分，如：personid desc, age asc;

**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。

使用SQLiteOpenHelper获取用于操作数据库的SQLiteDatabase实例

public class DatabaseHelper extends SQLiteOpenHelper {

private static final String name = "itcast"; //数据库名称

private static final int version = 1; //数据库版本

......略

}

public class HelloActivity extends Activity {

@Override public void onCreate(Bundle savedInstanceState) {

......

Button button =(Button) this.findViewById(R.id.button);

button.setOnClickListener(new View.OnClickListener(){

public void onClick(View v) {

DatabaseHelper databaseHelper = new DatabaseHelper(HelloActivity.this);

SQLiteDatabase db = databaseHelper.getWritableDatabase();

db.execSQL("insert into person(name, age) values(?,?)", new Object[]{"传智播客", 4});

db.close();

}});

}

}

第一次调用getWritableDatabase()或getReadableDatabase()方法后，SQLiteOpenHelper会缓存当前的SQLiteDatabase实例，SQLiteDatabase实例正常情况下会维持数据库的打开状态，所以在你不再需要SQLiteDatabase实例时，请及时调用close()方法释放资源。一旦SQLiteDatabase实例被缓存，多次调用getWritableDatabase()或getReadableDatabase()方法得到的都是同一实例。

使用事务操作SQLite数据库

使用SQLiteDatabase的beginTransaction()方法可以开启一个事务，程序执行到endTransaction() 方法时会检查事务的标志是否为成功，如果程序执行到endTransaction()之前调用了setTransactionSuccessful() 方法设置事务的标志为成功则提交事务，如果没有调用setTransactionSuccessful() 方法则回滚事务。使用例子如下：  
 SQLiteDatabase db = ....;

db.beginTransaction();//开始事务

try {

db.execSQL("insert into person(name, age) values(?,?)", new Object[]{"传智播客", 4});

db.execSQL("update person set name=? where personid=?", new Object[]{"传智", 1});

db.setTransactionSuccessful();//调用此方法会在执行到endTransaction() 时提交当前事务，如果不调用此方法会回滚事务

} finally {

db.endTransaction();//由事务的标志决定是提交事务，还是回滚事务

}

db.close();

上面两条SQL语句在同一个事务中执行。