|  |
| --- |
| Android系统提供了四种存储数据方式。分别为：SharePreference、SQLite、Content Provider和File。但由于Android系统中，数据基本是私有的，都是存放于”data/data”程序包名目录下，所以要实现数据共享，正确方式是使用Content Provider。  1、SharedPreference  特点：保存少量数据，支持字符串、基础类型。其本质就是一个xml文件，数据以键值对的形式保存，常用于存储较简单的参数设置。默认存储路径：/data/data/<PackageName>/shared\_prefs。  操作模式：MODE\_PRIVATE（默认）：只有当前的应用程序才能对文件进行读写； Context.MODE\_WORLD\_READABLE:  指定该SharedPreferences数据能被其他应用程序读，但不能写；  Context.MODE\_WORLD\_WRITEABLE:  指定该SharedPreferences数据能被其他应用程序读，写。  A.获取数据  SharedPreferences read = getSharedPreferences("lock", MODE\_WORLD\_READABLE);  String value = read.getString("code", "");  B.存储数据  •调用SharedPreferences对象的edit()方法获取一个SharedPreferences.Editor对象  •向Editor对象中添加数据putBoolean、putString等  •调用commit()方法提交数据  C.commit与apply  commit 会在调用者线程同步执行写文件，返回写入结果;  apply 将写文件的操作异步执行，没有返回值。  可以根据具体情况选择性使用，推荐使用 apply。  2、File：  Context提供了两个方法来打开数据文件里的文件IO流 FileInputStream openFileInput(String name); FileOutputStream(String name , int mode),这两个方法第一个参数 用于指定文件名，第二个参数指定打开文件的模式。  MODE\_PRIVATE：为默认操作模式，代表该文件是私有数据，只能被应用本身访问，在该模式下，写入的内容会覆盖原文件的内容。  MODE\_APPEND：模式会检查文件是否存在，存在就往文件追加内容，否则就创建新文件。  MODE\_WORLD\_READABLE：表示当前文件可以被其他应用读取；  MODE\_WORLD\_WRITEABLE：表示当前文件可以被其他应用写入。  常用于存储大数量的数据，但是缺点是更新数据将是一件困难的事情。  默认存储路径：/data/data/<PackageName>/files。  3、SQLite：SQLite是一个轻量级的数据库，支持基本的SQL语法，是常被采用的一种数据存储方式。Android为此数据库提供了一个名为SQLiteDatabase的类，封装了一些操作数据库的api。  第三方框架：GreenDao 与 Realm。  SQLiteOpenHelper是SQLiteDatabase的一个帮助类，用来管理数据库的创建和版本的更新。一般是建立一个类继承它，并实现它的onCreate和onUpgrade方法。  onCreate(SQLiteDatabase db)；创建数据库时调用；  onUpgrade(SQLiteDatabase db,int oldVersion , int newVersion)； 版本更新时调用。根据不同的版本，不同的更新处理逻辑。注意：假如是版本比较多的情况，upGrade()方法体就会显得比较庞大。  事务（Transaction）是由一系列对系统中数据进行访问与更新的操作所组成的一个程序执行逻辑单元。  事务具有4个基本特征，分别是：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Duration），简称ACID。  4、ContentProvider  ContentProvider是安卓平台中，在不同应用程序之间实现数据共享的一种机制。一个应用程序如果需要让别的程序可以操作自己的数据，即可采用这种机制。并且此种方式忽略了底层的数据存储实现，ContentProvider提供了一种统一的通过Uri实现数据操作的方式。其步骤为：  a. 在当前应用程序中定义一个ContentProvider。  b. 在当前应用程序的AndroidManifest.xml中注册此ContentProvider  c. 其他应用程序通过ContentResolver和Uri来获取此ContentProvider的数据。  ContentResolver提供了诸如insert(), delete(),query()和update()之类的方法。用于实现对ContentProvider中数据的存取操作。  例如音频，视频，图片和通讯录，一般都可以采用此种方式进行存储。  每个ContentProvider都会对外提供一个公共的URI（包装成Uri对象），如果应用程序有数据需要共享时，就需要使用Content Provider为这些数据定义一个URI，然后其他的应用程序就通过ContentProvider传入  这个URI来对数据进行操作。  URI由3个部分组成:"content://"、数据的路径、标识ID(可选)。 |

本文介绍Android平台进行数据存储的五大方式,分别如下:

**[1 使用SharedPreferences存储数据](https://www.cnblogs.com/ITtangtang/p/3920916.html" \l "type1)**

[2 文件存储数据](https://www.cnblogs.com/ITtangtang/p/3920916.html" \l "type2)

**[3 SQLite数据库存储数据](https://www.cnblogs.com/ITtangtang/p/3920916.html" \l "type3)**

**[4 使用ContentProvider存储数据](https://www.cnblogs.com/ITtangtang/p/3920916.html" \l "type4)**

**5 网络存储**

# ****1、使用SharedPreferences存储数据****

适用范围**：**保存少量的数据，且这些数据的格式非常简单：字符串型、基本类型的值。比如应用程序的各种配置信息（如是否打开音效、是否使用震动效果、小游戏的玩家积分等），解锁口令密码等

Android 中的 SharedPreference 是轻量级的数据存储方式，能够保存简单的数据类型，比如 String、int、boolean 值等。其内部是以 XML 结构保存在 /data/data/包名/shared\_prefs 文件夹下，数据以键值对的形式保存。

应用首次使用 SP 的时候会从磁盘读取，之后缓存在内存中。

通过 getSharedPreferences 可以获取 SP 实例，从首次初始化到读到数据会存在延迟，因为读文件的操作阻塞调用的线程直到文件读取完毕，如果在主线程调用，可能会对 UI 流畅度造成影响。

commit 会在调用者线程同步执行写文件，返回写入结果；apply 将写文件的操作异步执行，没有返回值。可以根据具体情况选择性使用，推荐使用 apply。

核心原理**：**保存基于XML（Extensive Markup Language）文件存储的key-value键值对数据，通常用来存储一些简单的配置信息。通过DDMS的File Explorer面板，展开文件浏览树,很明显SharedPreferences数据总是存储在/data/data/<package name>/shared\_prefs目录下。SharedPreferences对象本身只能获取数据而不支持存储和修改,存储修改是通过SharedPreferences.edit()获取的内部接口Editor对象实现。 SharedPreferences本身是一个接口，程序无法直接创建SharedPreferences实例，只能通过Context提供的getSharedPreferences(String name, int mode)方法来获取SharedPreferences实例，该方法中name表示要操作的xml文件名，第二个参数具体如下：

  Context.MODE\_PRIVATE: 指定该SharedPreferences数据只能被本应用程序读、写。

  Context.MODE\_WORLD\_READABLE:  指定该SharedPreferences数据能被其他应用程序读，但不能写。

**Context.MODE\_WORLD\_WRITEABLE**:  指定该SharedPreferences数据能被其他应用程序读，写

Editor有如下主要重要方法：

SharedPreferences.Editor clear():清空SharedPreferences里所有数据。

  SharedPreferences.Editor putXxx(String key , xxx value): 向SharedPreferences存入指定key对应的数据，其中xxx 可以是boolean,float,int等各种基本类型据。

SharedPreferences.Editor remove(): 删除SharedPreferences中指定key对应的数据项。

boolean commit(): 当Editor编辑完成后，使用该方法提交修改

  实际案例：

这里只提供了两个按钮和一个输入文本框，布局简单，故在此不给出界面布局文件了,程序核心代码如下：

[IMG_257](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

class ViewOcl implements View.OnClickListener{

@Override

public void onClick(View v) {

switch(v.getId()){

case R.id.btnSet:

//步骤1：获取输入值

String code = txtCode.getText().toString().trim();

//步骤2-1：创建一个SharedPreferences.Editor接口对象，lock表示要写入的XML文件名，MODE\_WORLD\_WRITEABLE写操作

SharedPreferences.Editor editor = getSharedPreferences("lock", MODE\_WORLD\_WRITEABLE).edit();

//步骤2-2：将获取过来的值放入文件

editor.putString("code", code);

//步骤3：提交

editor.commit();

Toast.makeText(getApplicationContext(), "口令设置成功", Toast.LENGTH\_LONG).show();

break;

case R.id.btnGet:

//步骤1：创建一个SharedPreferences接口对象

SharedPreferences read = getSharedPreferences("lock", MODE\_WORLD\_READABLE);

//步骤2：获取文件中的值

String value = read.getString("code", "");

Toast.makeText(getApplicationContext(), "口令为："+value, Toast.LENGTH\_LONG).show();

break;

}

}

}

[IMG_258](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

读写其他应用的SharedPreferences: 步骤如下

1、在创建SharedPreferences时，指定MODE\_WORLD\_READABLE模式，表明该SharedPreferences数据可以被其他程序读取

2、创建其他应用程序对应的Context:

Context pvCount = createPackageContext("com.tony.app", Context.CONTEXT\_IGNORE\_SECURITY);这里的com.tony.app就是其他程序的包名

3、使用其他程序的Context获取对应的SharedPreferences

SharedPreferences read = pvCount**.**getSharedPreferences("lock", Context.MODE\_WORLD\_READABLE);

4、如果是写入数据，使用Editor接口即可，所有其他操作均和前面一致。

SharedPreferences对象与SQLite数据库相比，免去了创建数据库，创建表，写SQL语句等诸多操作，相对而言更加方便，简洁。但是SharedPreferences也有其自身缺陷，比如其职能存储boolean，int，float，long和String五种简单的数据类型，比如其无法进行条件查询等。所以不论SharedPreferences的数据存储操作是如何简单，它也只能是存储方式的一种补充，而无法完全替代如SQLite数据库这样的其他数据存储方式。

# ****2、文件存储数据****

核心原理: Context提供了两个方法来打开数据文件里的文件IO流 FileInputStream openFileInput(String name); FileOutputStream(String name , int mode),这两个方法第一个参数 用于指定文件名，第二个参数指定打开文件的模式。具体有以下值可选：

**MODE\_PRIVATE**：为默认操作模式，代表该文件是私有数据，只能被应用本身访问，在该模式下，写入的内容会覆盖原文件的内容，如果想把新写入的内容追加到原文件中。可以使用Context.MODE\_APPEND

MODE\_APPEND：模式会检查文件是否存在，存在就往文件追加内容，否则就创建新文件。

**MODE\_WORLD\_READABLE**：表示当前文件可以被其他应用读取；

MODE\_WORLD\_WRITEABLE：表示当前文件可以被其他应用写入。

除此之外，Context还提供了如下几个重要的方法：

**getDir(String name , int mode)**:在应用程序的数据文件夹下获取或者创建name对应的子目录。

**File getFilesDir()**:获取该应用程序的数据文件夹得绝对路径

**String[] fileList()**:返回该应用数据文件夹的全部文件

实际案例：界面沿用上图

             核心代码如下：

[IMG_259](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

public String read() {

try {

FileInputStream inStream = this.openFileInput("message.txt");

byte[] buffer = new byte[1024];

int hasRead = 0;

StringBuilder sb = new StringBuilder();

while ((hasRead = inStream.read(buffer)) != -1) {

sb.append(new String(buffer, 0, hasRead));

}

inStream.close();

return sb.toString();

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

public void write(String msg){

// 步骤1：获取输入值

if(msg == null) return;

try {

// 步骤2:创建一个FileOutputStream对象,MODE\_APPEND追加模式

FileOutputStream fos = openFileOutput("message.txt",

MODE\_APPEND);

// 步骤3：将获取过来的值放入文件 fos.write(msg.getBytes());

// 步骤4：关闭数据流 fos.close();

} catch (Exception e) {

e.printStackTrace();

}

}

[IMG_260](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

openFileOutput()方法的第一参数用于指定文件名称，不能包含路径分隔符“/” ，如果文件不存在，Android 会自动创建它。创建的文件保存在/data/data/<package name>/files目录，如： /data/data/cn.tony.app/files/message.txt，

下面讲解某些特殊文件读写需要注意的地方:

**读写sdcard上的文件**

其中读写步骤按如下进行:

1、调用Environment的getExternalStorageState()方法判断手机上是否插了sd卡,且应用程序具有读写SD卡的权限，如下代码将返回true

Environment.getExternalStorageState().equals(Environment.MEDIA\_MOUNTED)

2、调用Environment.getExternalStorageDirectory()方法来获取外部存储器，也就是SD卡的目录,或者使用"/mnt/sdcard/"目录

3、使用IO流操作SD卡上的文件

注意点：手机应该已插入SD卡，对于模拟器而言，可通过mksdcard命令来创建虚拟存储卡

必须在AndroidManifest.xml上配置读写SD卡的权限

<uses-permission android:name="android.permission.MOUNT\_UNMOUNT\_FILESYSTEMS"/>  
<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"/>

案例代码：

[IMG_261](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

// 文件写操作函数

private void write(String content) {

if (Environment.getExternalStorageState().equals(

Environment.MEDIA\_MOUNTED)) { // 如果sdcard存在

File file = new File(Environment.getExternalStorageDirectory()

.toString()

+ File.separator

+ DIR

+ File.separator

+ FILENAME); // 定义File类对象

if (!file.getParentFile().exists()) { // 父文件夹不存在

file.getParentFile().mkdirs(); // 创建文件夹 }

PrintStream out = null; // 打印流对象用于输出

try {

out = new PrintStream(new FileOutputStream(file, true)); // 追加文件 out.println(content);

} catch (Exception e) {

e.printStackTrace();

} finally {

if (out != null) {

out.close(); // 关闭打印流 }

}

} else { // SDCard不存在，使用Toast提示用户

Toast.makeText(this, "保存失败，SD卡不存在！", Toast.LENGTH\_LONG).show();

}

}

// 文件读操作函数

private String read() {

if (Environment.getExternalStorageState().equals(

Environment.MEDIA\_MOUNTED)) { // 如果sdcard存在

File file = new File(Environment.getExternalStorageDirectory()

.toString()

+ File.separator

+ DIR

+ File.separator

+ FILENAME); // 定义File类对象

if (!file.getParentFile().exists()) { // 父文件夹不存在

file.getParentFile().mkdirs(); // 创建文件夹 }

Scanner scan = null; // 扫描输入

StringBuilder sb = new StringBuilder();

try {

scan = new Scanner(new FileInputStream(file)); // 实例化Scanner

while (scan.hasNext()) { // 循环读取

sb.append(scan.next() + "\n"); // 设置文本 }

return sb.toString();

} catch (Exception e) {

e.printStackTrace();

} finally {

if (scan != null) {

scan.close(); // 关闭打印流 }

}

} else { // SDCard不存在，使用Toast提示用户

Toast.makeText(this, "读取失败，SD卡不存在！", Toast.LENGTH\_LONG).show();

}

return null;

}

[IMG_262](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

# ****3、SQLite存储数据****

GreenDao 与 Realm:

由于GreenDao在3.0.1后的使用极其方便，并且使用的习惯与拓展性、稳定性优于Realm，推荐使用GreenDao。

如果你的项目中对添加与查询操作要求极高的话，推荐使用Realm，但它目前相对不稳定，官方也在持续优化中，需留意官网并实时替换新版本。

SQLite是轻量级嵌入式数据库引擎，它支持 SQL 语言，并且只利用很少的内存就有很好的性能。现在的主流移动设备像Android、iPhone等都使用SQLite作为复杂数据的存储引擎，在我们为移动设备开发应用程序时，也许就要使用到SQLite来存储我们大量的数据，所以我们就需要掌握移动设备上的SQLite开发技巧。

SQLiteDatabase类为我们提供了很多种方法，上面的代码中基本上囊括了大部分的数据库操作；对于添加、更新和删除来说，我们都可以使用

1 db.executeSQL(String sql);

2 db.executeSQL(String sql, Object[] bindArgs);//sql语句中使用占位符，然后第二个参数是实际的参数集

除了统一的形式之外，他们还有各自的操作方法：

1 db.insert(String table, String nullColumnHack, ContentValues values);

2 db.update(String table, Contentvalues values, String whereClause, String whereArgs);

3 db.delete(String table, String whereClause, String whereArgs);

以上三个方法的第一个参数都是表示要操作的表名；insert中的第二个参数表示如果插入的数据每一列都为空的话，需要指定此行中某一列的名称，系统将此列设置为NULL，不至于出现错误；insert中的第三个参数是ContentValues类型的变量，是键值对组成的Map，key代表列名，value代表该列要插入的值；update的第二个参数也很类似，只不过它是更新该字段key为最新的value值，第三个参数whereClause表示WHERE表达式，比如“age > ? and age < ?”等，最后的whereArgs参数是占位符的实际参数值；delete方法的参数也是一样

下面给出demo

**数据的添加**

1.使用insert方法

[IMG_263](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

1 ContentValues cv = new ContentValues();//实例化一个ContentValues用来装载待插入的数据

2 cv.put("title","you are beautiful");//添加title

3 cv.put("weather","sun"); //添加weather

4 cv.put("context","xxxx"); //添加context

5 String publish = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")

6 .format(new Date());

7 cv.put("publish ",publish); //添加publish

8 db.insert("diary",null,cv);//执行插入操作

[IMG_264](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

2.使用execSQL方式来实现

String sql = "insert into user(username,password) values ('Jack Johnson','iLovePopMuisc');//插入操作的SQL语句

db.execSQL(sql);//执行SQL语句

**数据的删除**

同样有2种方式可以实现

String whereClause = "username=?";//删除的条件

String[] whereArgs = {"Jack Johnson"};//删除的条件参数

db.delete("user",whereClause,whereArgs);//执行删除

使用execSQL方式的实现

String sql = "delete from user where username='Jack Johnson'";//删除操作的SQL语句

db.execSQL(sql);//执行删除操作

**数据修改**

同上，仍是2种方式

ContentValues cv = new ContentValues();//实例化ContentValues

cv.put("password","iHatePopMusic");//添加要更改的字段及内容

String whereClause = "username=?";//修改条件

String[] whereArgs = {"Jack Johnson"};//修改条件的参数

db.update("user",cv,whereClause,whereArgs);//执行修改

使用execSQL方式的实现

String sql = "update user set password = 'iHatePopMusic' where username='Jack Johnson'";//修改的SQL语句

db.execSQL(sql);//执行修改

**数据查询**

下面来说说查询操作。查询操作相对于上面的几种操作要复杂些，因为我们经常要面对着各种各样的查询条件，所以系统也考虑到这种复杂性，为我们提供了较为丰富的查询形式：

[IMG_265](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

1 db.rawQuery(String sql, String[] selectionArgs);

2 db.query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy);

3 db.query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit);

4 db.query(String distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit);

[IMG_266](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

上面几种都是常用的查询方法，第一种最为简单，将所有的SQL语句都组织到一个字符串中，使用占位符代替实际参数，selectionArgs就是占位符实际参数集；

各参数说明：

* table：表名称
* colums：表示要查询的列所有名称集
* selection：表示WHERE之后的条件语句，可以使用占位符
* selectionArgs：条件语句的参数数组
* groupBy：指定分组的列名
* having：指定分组条件,配合groupBy使用
* orderBy：y指定排序的列名
* limit：指定分页参数
* distinct：指定“true”或“false”表示要不要过滤重复值
* Cursor：返回值，相当于结果集ResultSet

最后，他们同时返回一个Cursor对象，代表数据集的游标，有点类似于JavaSE中的ResultSet。下面是Cursor对象的常用方法：

[IMG_267](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

1 c.move(int offset); //以当前位置为参考,移动到指定行

2 c.moveToFirst(); //移动到第一行

3 c.moveToLast(); //移动到最后一行

4 c.moveToPosition(int position); //移动到指定行

5 c.moveToPrevious(); //移动到前一行

6 c.moveToNext(); //移动到下一行

7 c.isFirst(); //是否指向第一条

8 c.isLast(); //是否指向最后一条

9 c.isBeforeFirst(); //是否指向第一条之前

10 c.isAfterLast(); //是否指向最后一条之后

11 c.isNull(int columnIndex); //指定列是否为空(列基数为0)

12 c.isClosed(); //游标是否已关闭

13 c.getCount(); //总数据项数

14 c.getPosition(); //返回当前游标所指向的行数

15 c.getColumnIndex(String columnName);//返回某列名对应的列索引值

16 c.getString(int columnIndex); //返回当前行指定列的值

[IMG_268](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

实现代码

[IMG_269](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

String[] params = {12345,123456};  
Cursor cursor = db.query("user",columns,"ID=?",params,null,null,null);//查询并获得游标if(cursor.moveToFirst()){//判断游标是否为空

for(int i=0;i<cursor.getCount();i++){

cursor.move(i);//移动到指定记录

String username = cursor.getString(cursor.getColumnIndex("username");

String password = cursor.getString(cursor.getColumnIndex("password"));

}

}

[IMG_270](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

通过rawQuery实现的带参数查询

[IMG_271](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

Cursor result=db.rawQuery("SELECT ID, name, inventory FROM mytable");//Cursor c = db.rawQuery("s name, inventory FROM mytable where ID=?",new Stirng[]{"123456"}); result.moveToFirst();

while (!result.isAfterLast()) {

int id=result.getInt(0);

String name=result.getString(1);

int inventory=result.getInt(2);

// do something useful with these result.moveToNext();

}

result.close();

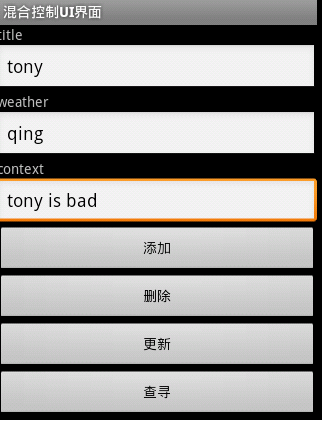
[IMG_272](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

在上面的代码示例中，已经用到了这几个常用方法中的一些，关于更多的信息，大家可以参考官方文档中的说明。

最后当我们完成了对数据库的操作后，记得调用SQLiteDatabase的close()方法释放数据库连接，否则容易出现SQLiteException。

上面就是SQLite的基本应用，但在实际开发中，为了能够更好的管理和维护数据库，我们会封装一个继承自SQLiteOpenHelper类的数据库操作类，然后以这个类为基础，再封装我们的业务逻辑方法。

这里直接使用案例讲解：下面是案例demo的界面



**SQLiteOpenHelper类介绍**

SQLiteOpenHelper是SQLiteDatabase的一个帮助类，用来管理数据库的创建和版本的更新。一般是建立一个类继承它，并实现它的onCreate和onUpgrade方法。

|  |  |
| --- | --- |
| **方法名** | **方法描述** |
| **SQLiteOpenHelper(Context context,String name,SQLiteDatabase.CursorFactory factory,int version)** | 构造方法，其中  context 程序上下文环境 即：XXXActivity.this;  name :数据库名字;  factory:游标工厂，默认为null,即为使用默认工厂;  version 数据库版本号 |
| **onCreate(SQLiteDatabase db)** | 创建数据库时调用 |
| **onUpgrade(SQLiteDatabase db,int oldVersion , int newVersion)** | 版本更新时调用 |
| **getReadableDatabase()** | 创建或打开一个只读数据库 |
| **getWritableDatabase()** | 创建或打开一个读写数据库 |

## 3.1 首先创建数据库类

[IMG_274](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

1 import android.content.Context;

2 import android.database.sqlite.SQLiteDatabase;

3 import android.database.sqlite.SQLiteDatabase.CursorFactory;

4 import android.database.sqlite.SQLiteOpenHelper;

5

6 public class SqliteDBHelper extends SQLiteOpenHelper {

7

8 // 步骤1：设置常数参量

9 private static final String DATABASE\_NAME = "diary\_db";

10 private static final int VERSION = 1;

11 private static final String TABLE\_NAME = "diary";

12

13 // 步骤2：重载构造方法

14 public SqliteDBHelper(Context context) {

15 super(context, DATABASE\_NAME, null, VERSION);

16 }

17

18 /\*

19 \* 参数介绍：context 程序上下文环境 即：XXXActivity.this

20 \* name 数据库名字

21 \* factory 接收数据，一般情况为null

22 \* version 数据库版本号

23 \*/

24 public SqliteDBHelper(Context context, String name, CursorFactory factory,

25 int version) {

26 super(context, name, factory, version);

27 }

28 //数据库第一次被创建时，onCreate()会被调用

29 @Override

30 public void onCreate(SQLiteDatabase db) {

31 // 步骤3：数据库表的创建

32 String strSQL = "create table "

33 + TABLE\_NAME

34 + "(tid integer primary key autoincrement,title varchar(20),weather varchar(10),context text,publish date)";

35 //步骤4：使用参数db,创建对象

36 db.execSQL(strSQL);

37 }

38 //数据库版本变化时，会调用onUpgrade()

39 @Override

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

41

42 }

43 }

[IMG_275](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

正如上面所述，数据库第一次创建时onCreate方法会被调用，我们可以执行创建表的语句，当系统发现版本变化之后，会调用onUpgrade方法，我们可以执行修改表结构等语句。

 我们需要一个Dao，来封装我们所有的业务方法，代码如下：

[IMG_276](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

1 import android.content.Context;

2 import android.database.Cursor;

3 import android.database.sqlite.SQLiteDatabase;

4

5 import com.chinasoft.dbhelper.SqliteDBHelper;

6

7 public class DiaryDao {

8

9 private SqliteDBHelper sqliteDBHelper;

10 private SQLiteDatabase db;

11

12 // 重写构造方法

13 public DiaryDao(Context context) {

14 this.sqliteDBHelper = new SqliteDBHelper(context);

15 db = sqliteDBHelper.getWritableDatabase();

16 }

17

18 // 读操作

19 public String execQuery(final String strSQL) {

20 try {

21 System.out.println("strSQL>" + strSQL);

22 // Cursor相当于JDBC中的ResultSet

23 Cursor cursor = db.rawQuery(strSQL, null);

24 // 始终让cursor指向数据库表的第1行记录

25 cursor.moveToFirst();

26 // 定义一个StringBuffer的对象，用于动态拼接字符串

27 StringBuffer sb = new StringBuffer();

28 // 循环游标，如果不是最后一项记录

29 while (!cursor.isAfterLast()) {

30 sb.append(cursor.getInt(0) + "/" + cursor.getString(1) + "/"

31 + cursor.getString(2) + "/" + cursor.getString(3) + "/"

32 + cursor.getString(4)+"#");

33 //cursor游标移动

34 cursor.moveToNext();

35 }

36 db.close();

37 return sb.deleteCharAt(sb.length()-1).toString();

38 } catch (RuntimeException e) {

39 e.printStackTrace();

40 return null;

41 }

42

43 }

44

45 // 写操作

46 public boolean execOther(final String strSQL) {

47

db.beginTransaction(); //开始事务

48 try {

49 System.out.println("strSQL" + strSQL);

50 db.execSQL(strSQL);

51 db.setTransactionSuccessful(); //设置事务成功完成

52 db.close();

53 return true;

54 } catch (RuntimeException e) {

55 e.printStackTrace();

56 return false;

57 }finally {

58 db.endTransaction(); //结束事务

59 }

60

61 }

62 }

[IMG_277](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

我们在Dao构造方法中实例化sqliteDBHelper并获取一个SQLiteDatabase对象，作为整个应用的数据库实例；在增删改信息时，我们采用了事务处理，确保数据完整性；最后要注意释放数据库资源db.close()，这一个步骤在我们整个应用关闭时执行，这个环节容易被忘记，所以朋友们要注意。

我们获取数据库实例时使用了getWritableDatabase()方法，也许朋友们会有疑问，在getWritableDatabase()和getReadableDatabase()中，你为什么选择前者作为整个应用的数据库实例呢？在这里我想和大家着重分析一下这一点。

我们来看一下SQLiteOpenHelper中的getReadableDatabase()方法：

[IMG_278](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

1 public synchronized SQLiteDatabase getReadableDatabase() {

2 if (mDatabase != null && mDatabase.isOpen()) {

3 // 如果发现mDatabase不为空并且已经打开则直接返回

4 return mDatabase;

5 }

6

7 if (mIsInitializing) {

8 // 如果正在初始化则抛出异常

9 throw new IllegalStateException("getReadableDatabase called recursively");

10 }

11

12 // 开始实例化数据库mDatabase

13

14 try {

15 // 注意这里是调用了getWritableDatabase()方法

16 return getWritableDatabase();

17 } catch (SQLiteException e) {

18 if (mName == null)

19 throw e; // Can't open a temp database read-only!

20 Log.e(TAG, "Couldn't open " + mName + " for writing (will try read-only):", e);

21 }

22

23 // 如果无法以可读写模式打开数据库 则以只读方式打开

24

25 SQLiteDatabase db = null;

26 try {

27 mIsInitializing = true;

28 String path = mContext.getDatabasePath(mName).getPath();// 获取数据库路径

29 // 以只读方式打开数据库

30 db = SQLiteDatabase.openDatabase(path, mFactory, SQLiteDatabase.OPEN\_READONLY);

31 if (db.getVersion() != mNewVersion) {

32 throw new SQLiteException("Can't upgrade read-only database from version " + db.getVersion() + " to "

33 + mNewVersion + ": " + path);

34 }

35

36 onOpen(db);

37 Log.w(TAG, "Opened " + mName + " in read-only mode");

38 mDatabase = db;// 为mDatabase指定新打开的数据库

39 return mDatabase;// 返回打开的数据库

40 } finally {

41 mIsInitializing = false;

42 if (db != null && db != mDatabase)

43 db.close();

44 }

45 }

[IMG_279](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

在getReadableDatabase()方法中，首先判断是否已存在数据库实例并且是打开状态，如果是，则直接返回该实例，否则试图获取一个可读写模式的数据库实例，如果遇到磁盘空间已满等情况获取失败的话，再以只读模式打开数据库，获取数据库实例并返回，然后为mDatabase赋值为最新打开的数据库实例。既然有可能调用到getWritableDatabase()方法，我们就要看一下了：

[IMG_280](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

public synchronized SQLiteDatabase getWritableDatabase() {

if (mDatabase != null && mDatabase.isOpen() && !mDatabase.isReadOnly()) {

// 如果mDatabase不为空已打开并且不是只读模式 则返回该实例

return mDatabase;

}

if (mIsInitializing) {

throw new IllegalStateException("getWritableDatabase called recursively");

}

// If we have a read-only database open, someone could be using it

// (though they shouldn't), which would cause a lock to be held on

// the file, and our attempts to open the database read-write would

// fail waiting for the file lock. To prevent that, we acquire the

// lock on the read-only database, which shuts out other users.

boolean success = false;

SQLiteDatabase db = null;

// 如果mDatabase不为空则加锁 阻止其他的操作

if (mDatabase != null)

mDatabase.lock();

try {

mIsInitializing = true;

if (mName == null) {

db = SQLiteDatabase.create(null);

} else {

// 打开或创建数据库

db = mContext.openOrCreateDatabase(mName, 0, mFactory);

}

// 获取数据库版本(如果刚创建的数据库,版本为0)

int version = db.getVersion();

// 比较版本(我们代码中的版本mNewVersion为1)

if (version != mNewVersion) {

db.beginTransaction();// 开始事务

try {

if (version == 0) {

// 执行我们的onCreate方法 onCreate(db);

} else {

// 如果我们应用升级了mNewVersion为2,而原版本为1则执行onUpgrade方法 onUpgrade(db, version, mNewVersion);

}

db.setVersion(mNewVersion);// 设置最新版本

db.setTransactionSuccessful();// 设置事务成功

} finally {

db.endTransaction();// 结束事务 }

}

onOpen(db);

success = true;

return db;// 返回可读写模式的数据库实例

} finally {

mIsInitializing = false;

if (success) {

// 打开成功

if (mDatabase != null) {

// 如果mDatabase有值则先关闭

try {

mDatabase.close();

} catch (Exception e) {

}

mDatabase.unlock();// 解锁 }

mDatabase = db;// 赋值给mDatabase

} else {

// 打开失败的情况：解锁、关闭

if (mDatabase != null)

mDatabase.unlock();

if (db != null)

db.close();

}

}

}

[IMG_281](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

大家可以看到，几个关键步骤是，首先判断mDatabase如果不为空已打开并不是只读模式则直接返回，否则如果mDatabase不为空则加锁，然后开始打开或创建数据库，比较版本，根据版本号来调用相应的方法，为数据库设置新版本号，最后释放旧的不为空的mDatabase并解锁，把新打开的数据库实例赋予mDatabase，并返回最新实例。

看完上面的过程之后，大家或许就清楚了许多，如果不是在遇到磁盘空间已满等情况，getReadableDatabase()一般都会返回和getWritableDatabase()一样的数据库实例，所以我们在DBManager构造方法中使用getWritableDatabase()获取整个应用所使用的数据库实例是可行的。当然如果你真的担心这种情况会发生，那么你可以先用getWritableDatabase()获取数据实例，如果遇到异常，再试图用getReadableDatabase()获取实例，当然这个时候你获取的实例只能读不能写了

最后，让我们看一下如何使用这些数据操作方法来显示数据,界面核心逻辑代码:

[IMG_282](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

public class SQLiteActivity extends Activity {

public DiaryDao diaryDao;

//因为getWritableDatabase内部调用了mContext.openOrCreateDatabase(mName, 0, mFactory);

//所以要确保context已初始化,我们可以把实例化Dao的步骤放在Activity的onCreate里 @Override

protected void onCreate(Bundle savedInstanceState) {

diaryDao = new DiaryDao(SQLiteActivity.this);

initDatabase();

}

class ViewOcl implements View.OnClickListener {

@Override

public void onClick(View v) {

String strSQL;

boolean flag;

String message;

switch (v.getId()) {

case R.id.btnAdd:

String title = txtTitle.getText().toString().trim();

String weather = txtWeather.getText().toString().trim();;

String context = txtContext.getText().toString().trim();;

String publish = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")

.format(new Date());

// 动态组件SQL语句

strSQL = "insert into diary values(null,'" + title + "','"

+ weather + "','" + context + "','" + publish + "')";

flag = diaryDao.execOther(strSQL);

//返回信息

message = flag?"添加成功":"添加失败";

Toast.makeText(getApplicationContext(), message, Toast.LENGTH\_LONG).show();

break;

case R.id.btnDelete:

strSQL = "delete from diary where tid = 1";

flag = diaryDao.execOther(strSQL);

//返回信息

message = flag?"删除成功":"删除失败";

Toast.makeText(getApplicationContext(), message, Toast.LENGTH\_LONG).show();

break;

case R.id.btnQuery:

strSQL = "select \* from diary order by publish desc";

String data = diaryDao.execQuery(strSQL);

Toast.makeText(getApplicationContext(), data, Toast.LENGTH\_LONG).show();

break;

case R.id.btnUpdate:

strSQL = "update diary set title = '测试标题1-1' where tid = 1";

flag = diaryDao.execOther(strSQL);

//返回信息

message = flag?"更新成功":"更新失败";

Toast.makeText(getApplicationContext(), message, Toast.LENGTH\_LONG).show();

break;

}

}

}

private void initDatabase() {

// 创建数据库对象

SqliteDBHelper sqliteDBHelper = new SqliteDBHelper(SQLiteActivity.this);

sqliteDBHelper.getWritableDatabase();

System.out.println("数据库创建成功");

}

}

[IMG_283](https://www.cnblogs.com/ITtangtang/p/javascript:void(0);)

**Android sqlite3数据库管理工具**

Android SDK的tools目录下提供了一个sqlite3.exe工具，这是一个简单的sqlite数据库管理工具。开发者可以方便的使用其对sqlite数据库进行命令行的操作。

程序运行生成的\*.db文件一般位于"/data/data/项目名(包括所处包名)/databases/\*.db"，因此要对数据库文件进行操作需要先找到数据库文件：

1、进入shell 命令

adb shell

2、找到数据库文件

#cd data/data

#ls --列出所有项目

#cd project\_name --进入所需项目名

#cd databases

#ls --列出现寸的数据库文件

3、进入数据库

#sqlite3 test\_db --进入所需数据库

会出现类似如下字样：

SQLite version 3.6.22

Enter ".help" for instructions

Enter SQL statements terminated with a ";"

sqlite>

至此，可对数据库进行sql操作。

4、sqlite常用命令

>.databases --产看当前数据库

>.tables --查看当前数据库中的表

>.help --sqlite3帮助

>.schema --各个表的生成语句

## 3.2数据可升级

数据库SQL增删改查语句：

sql语句不区分大小写

查找：

select column1,column2 from table\_name

或者

select \* from table\_name

插入：

insert into table\_name(column1,column2) values (value1,value2)

更新：

更新已有表的数据：

update table\_name set column1 = value1, column2 = value2

where +条件语句

向表中插入列：

alter table table\_name add column column\_name text(varchar(10)、char等)

删除：

delete from table\_name where +条件语句

数据库创建：

首先创建一个类去继承SQLiteOpenHelper，创建一个test.db的数据库

public class DBHelper extends SQLiteOpenHelper {

private static final String DB\_NAME = "test.db";

public static final int DB\_VERSION = 1;

public DBHelper(Context context) {

super(context, DB\_NAME, null, DB\_VERSION);

}

@Override

public void onCreate(SQLiteDatabase database) {

database.execSQL(DBDao.SQL\_CREATE\_TABLE);

}

@Override

public void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion) {

}

}

其中onCreate()是在数据库首次创建的时候调用，onUpgrade()是在数据库版本号DB\_VERSION升级的时候才会调用。

**一、软件v1.0**

安装v1.0,假设v1.0版本只有一个account表，这时走继承SQLiteOpenHelper的onCreate，不走onUpgrade。

1、v1.0(直接安装v1.0)

**二、软件v2.0**

有2种安装软件情况：

1、v1.0   -->  v2.0              不走onCreate，走onUpgrade

2、v2.0(直接安装v2.0)          走onCreate，不走onUpgrade

v1.0版本只有一个account表，软件版本升级到v2.0了，但是v2.0数据库需要新增一个member表，那怎么办呢？这里有2种情况了：

一种是安装了v1.0升级到v2.0，这时不会走继承SQLiteOpenHelper的onCreate，而是直接走onUpgrade，这时就要在onUpgrade添加member表的代码了，在onCreate加了也没用，因为这种情况都不走onCreate。

另一种情况就是用户从来没有安装过这个软件，直接安装v2.0，这时走继承SQLiteOpenHelper的onCreate，不走onUpgrade,所以要在onCreate添加member表的代码。这怎么办呢？这就要合理升级数据库版本了。

三、**软件v3.0**

假设v3.0又新增一个news表，这里有三种情况：

1、v1.0   -->  v3.0              不走onCreate，走onUpgrade

2、v2.0   -->  v3.0              不走onCreate，走onUpgrade

3、v3.0(直接安装v3.0)          走onCreate，不走onUpgrade

那数据库添加表语句在那里写呢？数据库有一个版本号用DATABASE\_VERSION表示

其实想一下，就知道不是onCreate写就是onUpgrade写，就是要兼容各种情况下安装app，都能把数据库表添加进去就好了。这里很巧妙：

1、v1.0     DATABASE\_VERSION=1000    onCreate      --添加--  account

2、v2.0     DATABASE\_VERSION=1001    onCreate      --添加--  account  （v1.0代码不变）  onUpgrade（DATABASE\_VERSION>1000）

                                                       onUpgrade   --添加--  member

3、v3.0     DATABASE\_VERSION=1002    onCreate      --添加--  account  （v1.0代码不变）  onUpgrade（DATABASE\_VERSION>1001）

                                                       onUpgrade   --添加--  member （v2.0代码不变）

                                                       onUpgrade   --添加--  news

这样就可以解决问题了，第一版本的都在onCreate，其他版本新增的在onUpgrade,而且在onCreate执行onUpgrade。做判断是否执行onUpgrade该怎么判断呢，所以有了数据库版本的概念了，DATABASE\_VERSION保存当前的数据库版本，只要当前的数据库版本比已经安装的数据库版本大时，就进入onUpgrade，这时还会把上一个数据库版本号（oldVersion）跟安装的数据库版本号（newVersion）做比较，不同的DATABASE\_VERSION添加自己所需要的表（跨版本升级数据库）。

下面做一个简单的实例：

（1）、V1.0  : DATABASE\_VERSION = 1000 添加一个favorite表

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | public class DBHelper extends SQLiteOpenHelper {        private static final String DATABASE\_NAME = "mall.db";      private static final int DATABASE\_VERSION = 1000;        private static DBHelper instance = null;          public DBHelper(Context context) {          super(context, DATABASE\_NAME, null, DATABASE\_VERSION);      }        public synchronized static DBHelper getInstance(Context context) {          if (instance == null) {              instance = new DBHelper(context);          }          return instance;      }        @Override      public void onCreate(SQLiteDatabase db) {          db.execSQL(SQL.CREATE\_TABLE\_FAVORITE);            // 若不是第一个版本安装，直接执行数据库升级          // 请不要修改FIRST\_DATABASE\_VERSION的值，其为第一个数据库版本大小          final int FIRST\_DATABASE\_VERSION = 1000;          onUpgrade(db, FIRST\_DATABASE\_VERSION, DATABASE\_VERSION);      }        @Override      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {          // 使用for实现跨版本升级数据库          for (int i = oldVersion; i < newVersion; i++) {              switch (i) {                default:                  break;              }          }      }  } |

其中SQL.java是建表语句

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public class SQL {      public static final String T\_FAVORITE = "favorite";          public static final String CREATE\_TABLE\_FAVORITE =              "CREATE TABLE IF NOT EXISTS " + T\_FAVORITE + "(" +                      "id VARCHAR PRIMARY KEY, " +                      "title VARCHAR, " +                      "url VARCHAR, " +                      "createDate VARCHAR " +                      ")";  } |

（2）、V2.0  : DATABASE\_VERSION = 1001 在favorite表添加1个deleted字段

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | public class DBHelper extends SQLiteOpenHelper {        private static final String DATABASE\_NAME = "mall.db";      private static final int DATABASE\_VERSION = 1001;        private static DBHelper instance = null;          public DBHelper(Context context) {          super(context, DATABASE\_NAME, null, DATABASE\_VERSION);      }        public synchronized static DBHelper getInstance(Context context) {          if (instance == null) {              instance = new DBHelper(context);          }          return instance;      }        @Override      public void onCreate(SQLiteDatabase db) {          db.execSQL(SQL.CREATE\_TABLE\_FAVORITE);            // 若不是第一个版本安装，直接执行数据库升级          // 请不要修改FIRST\_DATABASE\_VERSION的值，其为第一个数据库版本大小          final int FIRST\_DATABASE\_VERSION = 1000;          onUpgrade(db, FIRST\_DATABASE\_VERSION, DATABASE\_VERSION);      }        @Override      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {          // 使用for实现跨版本升级数据库          for (int i = oldVersion; i < newVersion; i++) {              switch (i) {              case 1000:                  upgradeToVersion1001(db);                  break;              default:                  break;              }          }      }        private void upgradeToVersion1001(SQLiteDatabase db){          // favorite表新增1个字段          String sql1 = "ALTER TABLE "+SQL.T\_FAVORITE+" ADD COLUMN deleted VARCHAR";          db.execSQL(sql1);      }  } |

（3）、V3.0  : DATABASE\_VERSION = 1002 在favorite表添加message和type字段

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60 | public class DBHelper extends SQLiteOpenHelper {        private static final String DATABASE\_NAME = "mall.db";      private static final int DATABASE\_VERSION = 1002;        private static DBHelper instance = null;          public DBHelper(Context context) {          super(context, DATABASE\_NAME, null, DATABASE\_VERSION);      }        public synchronized static DBHelper getInstance(Context context) {          if (instance == null) {              instance = new DBHelper(context);          }          return instance;      }        @Override      public void onCreate(SQLiteDatabase db) {          db.execSQL(SQL.CREATE\_TABLE\_FAVORITE);            // 若不是第一个版本安装，直接执行数据库升级          // 请不要修改FIRST\_DATABASE\_VERSION的值，其为第一个数据库版本大小          final int FIRST\_DATABASE\_VERSION = 1000;          onUpgrade(db, FIRST\_DATABASE\_VERSION, DATABASE\_VERSION);      }        @Override      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {          // 使用for实现跨版本升级数据库          for (int i = oldVersion; i < newVersion; i++) {              switch (i) {              case 1000:                  upgradeToVersion1001(db);                  break;              case 1001:                  upgradeToVersion1002(db);                  break;                default:                  break;              }          }      }        private void upgradeToVersion1001(SQLiteDatabase db){          // favorite表新增1个字段          String sql1 = "ALTER TABLE "+SQL.T\_FAVORITE+" ADD COLUMN deleted VARCHAR";          db.execSQL(sql1);      }      private void upgradeToVersion1002(SQLiteDatabase db){          // favorite表新增2个字段,添加新字段只能一个字段一个字段加，sqlite有限制不予许一条语句加多个字段          String sql1 = "ALTER TABLE "+SQL.T\_FAVORITE+" ADD COLUMN message VARCHAR";          String sql2 = "ALTER TABLE "+SQL.T\_FAVORITE+" ADD COLUMN type VARCHAR";          db.execSQL(sql1);          db.execSQL(sql2);      }  } |

就是这样，无论v1.0升级到v3.0，或者v2.0升级到3.0，还是v3.0直接安装，安装后的v3.0数据库结构都是一样的，理解透彻就是好啊，刚做sqlite数据库的肯定会遇到这些问题，所以在这里详细地写了一下，不过还是要注意一下，就是onUpgrade升级时候一定要写对，测试好，不然安装后的数据库都有问题就麻烦了。

## 3.3 比较复杂的升级

存在的问题：（1）假如是版本比较多的情况，upGrade()方法体就会显得比较庞大。

1. 一些表格的表结构的改变。

## 3.4 事务

事务（Transaction）是由一系列对系统中数据进行访问与更新的操作所组成的一个程序执行逻辑单元。

ACID

事务具有4个基本特征，分别是：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Duration），简称ACID。

隔离级别

ACID这4个特征中，最难理解的是隔离性。在标准SQL规范中，定义了4个事务隔离级别，不同的隔离级别对事务的处理不同。4个隔离级别分别是：读未提交（READ\_UNCOMMITTED）、读已提交（READ\_COMMITTED）、可重复读（REPEATABLE\_READ）、顺序读（SERIALIZABLE）。

事务并发引起的问题

数据库在不同的隔离性级别下并发访问可能会出现以下几种问题：脏读（Dirty Read）、不可重复读（Unrepeatable Read）、幻读（Phantom Read）。

（一）ACID

1. 原子性（Atomicity）

事务的原子性是指事务必须是一个原子的操作序列单元。事务中包含的各项操作在一次执行过程中，只允许出现两种状态之一。

全部执行成功

全部执行失败

任何一项操作都会导致整个事务的失败，同时其它已经被执行的操作都将被撤销并回滚，只有所有的操作全部成功，整个事务才算是成功完成。

2. 一致性（Consistency）

事务的一致性是指事务的执行不能破坏数据库数据的完整性和一致性，一个事务在执行之前和执行之后，数据库都必须处以一致性状态。

比如：如果从A账户转账到B账户，不可能因为A账户扣了钱，而B账户没有加钱。

3. 隔离性（Isolation）

事务的隔离性是指在并发环境中，并发的事务是互相隔离的，一个事务的执行不能被其它事务干扰。也就是说，不同的事务并发操作相同的数据时，每个事务都有各自完整的数据空间。

一个事务内部的操作及使用的数据对其它并发事务是隔离的，并发执行的各个事务是不能互相干扰的。

隔离性分4个级别，下面会介绍。

4. 持久性（Duration）

事务的持久性是指事务一旦提交后，数据库中的数据必须被永久的保存下来。即使服务器系统崩溃或服务器宕机等故障。只要数据库重新启动，那么一定能够将其恢复到事务成功结束后的状态。

事务隔离级别

1. 读未提及（READ\_UNCOMMITTED）

读未提及，该隔离级别允许脏读取，其隔离级别是最低的。换句话说，如果一个事务正在处理某一数据，并对其进行了更新，但同时尚未完成事务，因此还没有提交事务；而以此同时，允许另一个事务也能够访问该数据。

脏读示例：

在事务A和事务B同时执行时可能会出现如下场景：



余额应该为1500元才对。请看T5时间点，事务A此时查询的余额为0，这个数据就是脏数据，他是事务B造成的，很明显是事务没有进行隔离造成的。

2. 读已提交（READ\_COMMITTED）

读已提交是不同的时候执行的时候只能获取到已经提交的数据。

这样就不会出现上面的脏读的情况了。

不可重复读示例

可是解决了脏读问题，但是还是解决不了可重复读问题。



事务A其实除了查询两次以外，其它什么事情都没做，结果钱就从1000编程0了，这就是不可重复读的问题。

3. 可重复读（REPEATABLE\_READ）

可重复读就是保证在事务处理过程中，多次读取同一个数据时，该数据的值和事务开始时刻是一致的。因此该事务级别进制了不可重复读取和脏读，但是有可能出现幻读的数据。

幻读

幻读就是指同样的事务操作，在前后两个时间段内执行对同一个数据项的读取，可能出现不一致的结果。



银行工作人员在一个事务中多次统计总存款时看到结果不一样。如果要解决幻读，那只能使用顺序读了。

4. 顺序读（SERIALIZABLE）

顺序读是最严格的事务隔离级别。它要求所有的事务排队顺序执行，即事务只能一个接一个地处理，不能并发。

事务隔离级别对比



4种事务隔离级别从上往下，级别越高，并发性越差，安全性就越来越高。

一般数据默认级别是读以提交或可重复读。

## 3.5 一致性的解读

一致性是指数据处于一种语义上的有意义且正确的状态。一致性是对数据可见性的约束，保证在一个事务中的多次操作的数据中间状态对其他事务不可见的。因为这些中间状态，是一个过渡状态，与事务的开始状态和事务的结束状态是不一致的。

举个栗子，张三给李四转账100元。事务要做的是从张三账户上减掉100元，李四账户上加上100元。一致性的含义是其他事务要么看到张三还没有给李四转账的状态，要么张三已经成功转账给李四的状态，而对于张三少了100元，李四还没加上100元这个中间状态是不可见的。

那么反驳的声音来了：

要么转账操作全部成功，要么全部失败，这是原子性。从例子上看全部成功，那么一致性就是原子性的一部分咯，为什么还要单独说一致性和原子性？

你说的不对。在未提交读的隔离级别下是事务内部操作是可见的，这时候会出现脏读，明显违背了一致性，怎么解释？

好吧，你的疑问很有道理，也很充分，这正说明你对事务的ACID特性理解的很到位。不过，需要注意的是：

原子性和一致性的的侧重点不同：原子性关注状态，要么全部成功，要么全部失败，不存在部分成功的状态。而一致性关注数据的可见性，中间状态的数据对外部不可见，只有最初状态和最终状态的数据对外可见。

在未提交读的隔离级别下，会造成脏读，这就是因为一个事务读到了另一个事务操作内部的数据。ACID中是的一致性描述的是一个最理想的事务应该怎样的，是一个强一致性状态，如果要做到这点，需要使用排它锁把事务排成一队，即Serializable的隔离级别，这样性能就大大降低了。现实是骨感的，所以使用隔离性的不同隔离级别来破坏一致性，来获取更好的性能。

# 4、ContentProvider

ContentProvider是安卓平台中，在不同应用程序之间实现数据共享的一种机制。一个应用程序如果需要让别的程序可以操作自己的数据，即可采用这种机制。并且此种方式忽略了底层的数据存储实现，ContentProvider提供了一种统一的通过Uri实现数据操作的方式。其步骤为：

　　1. 在当前应用程序中定义一个ContentProvider。

　　2. 在当前应用程序的AndroidManifest.xml中注册此ContentProvider

　　3. 其他应用程序通过ContentResolver和Uri来获取此ContentProvider的数据。

ContentResolver提供了诸如insert(), delete(), query()和update()之类的方法。用于实现对ContentProvider中数据的存取操作。

　　Uri是一个通用资源标志符，将其分为A，B，C，D 4个部分：

　　A：无法改变的标准前缀，包括；"content://"、"tel://"等。当前缀是"content://"时，说明通过一个Content Provider控制这些数据

　　B：URI的标识，它通过authorities属性声明，用于定义了是哪个ContentProvider提供这些数据。对于第三方应用程序，为了保证URI标识的唯一性，它必须是一个完整的、小写的 类名。例如;"content://com.test.data.myprovider"

　 C：路径，可以近似的理解为需要操作的数据库中表的名字，如："content://hx.android.text.myprovider/name"中的name

　　D：如果URI中包含表示需要获取的记录的ID；则就返回该id对应的数据，如果没有ID，就表示返回全部；

下面进行案展示：演示一下如何在应用程序之间相互获取数据：

在应用程序A中，继承ContentProvider类，并重写其中方法：

|  |
| --- |
| public class MyProvider extends ContentProvider{  @Override  public int delete(Uri uri, String selection, String[] selectionArgs) {  // TODO Auto-generated method stub  return 0;  }  @Override  public String getType(Uri uri) {  // TODO Auto-generated method stub  return null;  }  @Override  public Uri insert(Uri uri, ContentValues values) {  return null;  }  //在Create中初始化一个数据库  @Override  public boolean onCreate() {  SQLiteDatabase db = this.getContext().openOrCreateDatabase("test\_db.db3", Context.MODE\_PRIVATE, null);  db.execSQL("create table tab(\_id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL)");  ContentValues values = new ContentValues();  values.put("name", "test");  db.insert("tab", "\_id", values);  db.close();  return true;  }  //实现query方法  @Override  public Cursor query(Uri uri, String[] projection, String selection,  String[] selectionArgs, String sortOrder) {  SQLiteDatabase db = this.getContext().openOrCreateDatabase("test\_db.db3", Context.MODE\_PRIVATE, null);  Cursor c = db.query("tab", null, null, null, null, null,null);  return c;  }  @Override  public int update(Uri uri, ContentValues values, String selection,  String[] selectionArgs) {  // TODO Auto-generated method stub  return 0;  }  } |

在其AndroidManifest.xml中声明此ContentProvider，其中authorities属性定义了此ContentProvider的Uri标识

<provider android:name=".MyProvider" android:authorities="com.test.MyProvider"/>

在应用程序B中，通过ContentResolver获取程序A的ContentProvider中的数据。

|  |
| --- |
| public class MainActivity extends Activity {  @Override  public void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.main);  //获取上下文  Context ctx = MainActivity.this;  //获取ContentResolver对象  ContentResolver resolver = ctx.getContentResolver();  //获取Uri对象  Uri uri = Uri.parse("content://com.test.MyProvider");  //获取数据  Cursor c = resolver.query(uri, null, null, null, null);  c.moveToFirst();  for(int i=0; i<c.getCount(); i++){  int index = c.getColumnIndexOrThrow("name");  String src = c.getString(index);  Log.d("", src);  c.moveToNext();  }  }  } |

应用程序B的运行结果如下，从此图可以发现我们在程序B中成功的获取到了程序A中的数据。