数据存储在开发中是使用最频繁的，在这里主要介绍 Android 平台中实现数据存储的 5 种方式，分别是：

1 使用 SharedPreferences 存储数据

2 文件存储数据

3 SQLite 数据库存储数据

4 使用 ContentProvider 存储数据

5 网络存储数据

下面将为大家一一详细介绍。

第一种： 使用 SharedPreferences 存储数据

SharedPreferences 是 Android 平台上一个轻量级的存储类，主要是保存一些常用的配置比如窗口状态，一般在 Activity 中 重载窗口状态 onSaveInstanceState 保存一般使用 SharedPreferences 完成，它提供了 Android 平台常规的 Long 长 整形、Int 整形、String 字符串型的保存。

它是什么样的处理方式呢? SharedPreferences 类似过去 Windows 系统上的 ini 配置文件，但是它分为多种权限，可以全局共享访问，android123 提示最终是以 xml 方式来保存，整体效率来看不是特别的高，对于常规的轻量级而言比 SQLite 要好不少，如果真的存储量不大可以考虑自己定义文件格式。xml 处理时 Dalvik 会通过自带底层的本地 XML Parser 解析，比如 XMLpull 方式，这样对于内存资源占用比较好。

它的本质是基于 XML 文件存储 key-value 键值对数据，通常用来存储一些简单的配置信息。

其存储位置在/data/data/<包名>/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 Editor 对象实现。

实现 SharedPreferences 存储的步骤如下：

一、根据 Context 获取 SharedPreferences 对象

二、利用 edit()方法获取 Editor 对象。

三、通过 Editor 对象存储 key-value 键值对数据。

四、通过 commit()方法提交数据。

下面是示例代码：

```
public class MainActivity extends Activity {
```

```
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //获取 SharedPreferences 对象
        Context ctx = MainActivity.this;
        SharedPreferences sp = ctx.getSharedPreferences("SP", MODE_PRIVATE);
        //存入数据
        Editor editor = sp.edit();
        editor.putString("STRING_KEY", "string");
        editor.putInt("INT_KEY", 0);
        editor.putBoolean("BOOLEAN_KEY", true);
        editor.commit();

        //返回 STRING_KEY 的值
        Log.d("SP", sp.getString("STRING_KEY", "none"));
        //如果 NOT_EXIST 不存在，则返回值为"none"
        Log.d("SP", sp.getString("NOT_EXIST", "none"));
    }
}
```

这段代码执行过后，即在/data/data/com.test/shared_prefs 目录下生成了一个 SP.xml 文件，一个应用可以创建多个这样的 xml 文件。

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。

第二种： 文件存储数据

关于文件存储，Activity 提供了 openFileOutput()方法可以用于把数据输出到文件中，具体的实现过程与在 J2SE 环境中保存数据到文件中是一样的。

文件可用来存放大量数据，如文本、图片、音频等。

默认位置：/data/data/<包>/files/***.***。

代码示例：

```
public void save()

{

      try {

            FileOutputStream outStream=this.openFileOutput("a.txt",Context.MODE_W
ORLD_READABLE);

            outStream.write(text.getText().toString().getBytes());

            outStream.close();

            Toast.makeText(MyActivity.this,"Saved",Toast.LENGTH_LONG).show();

        } catch (FileNotFoundException e) {

            return;

        }

        catch (IOException e){

            return ;

        }

}
```

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.itcast.action/files/itcast.txt ，通过点击 Eclipse 菜单“Window”-“Show View”-“Other”，在对话窗口中展开 android 文件夹，选择下面的 File Explorer 视图，然后在 File Explorer 视图中展开/data/data/<package name>/files 目录就可以看到该文件。

openFileOutput()方法的第二参数用于指定操作模式，有四种模式，分别为：

Context.MODE_PRIVATE = 0

Context.MODE_APPEND = 32768

Context.MODE_WORLD_READABLE = 1

Context.MODE_WORLD_WRITEABLE = 2

Context.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

Context.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。

Context.MODE_WORLD_READABLE 和 Context.MODE_WORLD_WRITEABLE 用来控制其他应用是否有权限读写该文件。

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。

如果希望文件被其他应用读和写，可以传入： openFileOutput("itcast.txt", Context.MODE_WORLD_READABLE + Context.MODE_WORLD_WRITEABLE); android 有一套自己的安全模型，当应用程序(.apk)在安装时系统就会分配给他一个 userid，当该应用要去访问其他资源比如文件的时候，就需要 userid 匹配。默认情况下，任何应用创建的文件，sharedpreferences，数据库都应该是私有的（位于/data/data/<package name>/files），其他程序无法访问。

除非在创建时指定了 Context.MODE_WORLD_READABLE 或者 Context.MODE_WORLD_WRITEABLE ，只有这样其他程序才能正确访问。

读取文件示例：

```
public void load()
{
    try {
        FileInputStream inStream=this.openFileInput("a.txt");
        ByteArrayOutputStream stream=new ByteArrayOutputStream();
        byte[] buffer=new byte[1024];
        int length=-1;

    while((length=inStream.read(buffer))!=-1)    {
            stream.write(buffer,0,length);
        }

       stream.close();
        inStream.close();
        text.setText(stream.toString());
        Toast.makeText(MyActivity.this,"Loaded",Toast.LENGTH_LONG).show();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    catch (IOException e){
        return ;
    }

}
```

对于私有文件只能被创建该文件的应用访问，如果希望文件能被其他应用读和写，可以在创建文件时，指定 Context.MODE_WORLD_READABLE 和 Context.MODE_WORLD_WRITEABLE 权限。

Activity 还提供了 getCacheDir()和 getFilesDir()方法： getCacheDir()方法用于获取/data/data/<package name>/cache 目录 getFilesDir()方法用于获取/data/data/<package name>/files 目录。

把文件存入 SDCard：

使用 Activity 的 openFileOutput()方法保存文件，文件是存放在手机空间上，一般手机的存储空间不是很大，存放些小文件还行，如果要存放像视频这样的大文件，是不可行的。对于像视频这样的大文件，我们可以把它存放在 SDCard。

SDCard 是干什么的？你可以把它看作是移动硬盘或 U 盘。 在模拟器中使用 SDCard，你需要先创建一张 SDCard 卡（当然不是真的 SDCard，只是镜像文件）。

创建 SDCard 可以在 Eclipse 创建模拟器时随同创建，也可以使用 DOS 命令进行创建，如下： 在 Dos 窗口中进入 android SDK 安装路径的 tools 目录，输入以下命令创建一张容量为 2G 的 SDCard，文件后缀可以随便取，建议使用.img： mksdcard 2048M D:\AndroidTool\sdcard.img 在程序中访问 SDCard，你需要申请访问 SDCard 的权限。

在 AndroidManifest.xml 中加入访问 SDCard 的权限如下:

```
<!-- 在 SDCard 中创建与删除文件权限 -->
 <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>

 <!-- 往 SDCard 写入数据权限 -->
 <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

要往 SDCard 存放文件，程序必须先判断手机是否装有 SDCard，并且可以进行读写。

注意：访问 SDCard 必须在 AndroidManifest.xml 中加入访问 SDCard 的权限。

```
if(Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)){

 File sdCardDir = Environment.getExternalStorageDirectory();//获取 SDCard 目录


 File saveFile = new File(sdCardDir, “a.txt”);
     FileOutputStream outStream = new FileOutputStream(saveFile);
```

```
        outStream.write("test".getBytes());
        outStream.close();
}
```

Environment.getExternalStorageState()方法用于获取 SDCard 的状态，如果手机装有 SDCard，并且可以进行读写，那么方法返回的状态等于 Environment.MEDIA_MOUNTED。

Environment.getExternalStorageDirectory()方法用于获取 SDCard 的目录，当然要获取 SDCard 的目录，你也可以这样写：

File sdCardDir = new File("/sdcard"); //获取 SDCard 目录

File saveFile = new File(sdCardDir, "itcast.txt");

//上面两句代码可以合成一句：

File saveFile = new File("/sdcard/a.txt");

FileOutputStream outStream = new FileOutputStream(saveFile);

outStream.write("test".getBytes());

outStream.close();

第三种： SQLite 数据库存储数据

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。此外它还是开源的，任何人都可以使用它。许多开源项目（(Mozilla, PHP, Python）都使用了 SQLite.SQLite 由以下几个组件组成：SQL 编译器、内核、后端以及附件。SQLite 通过利用虚拟机和虚拟数据库引擎（VDBE），使调试、修改和扩展 SQLite 的内核变得更加方便。

特点：

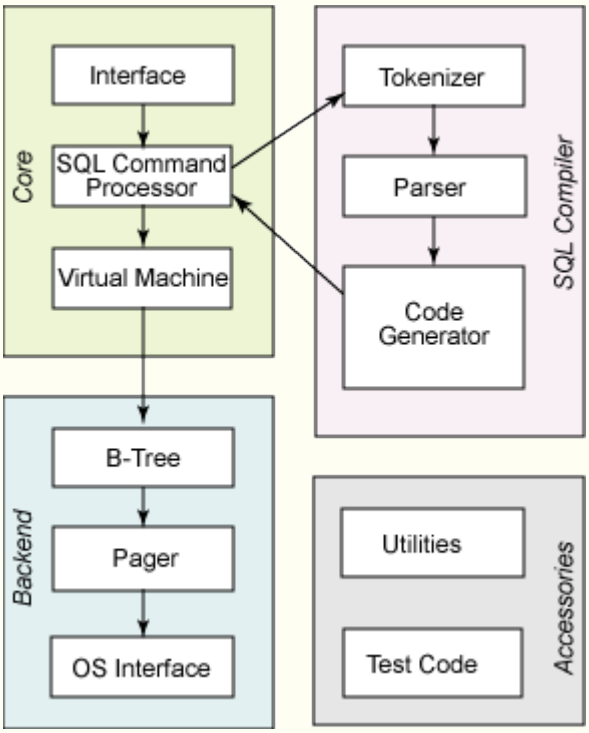
面向资源有限的设备，

没有服务器进程,

所有数据存放在同一文件中跨平台，

可自由复制。

SQLite 内部结构：



SQLite 基本上符合 SQL-92 标准，和其他的主要 SQL 数据库没什么区别。它的优点就是高效，Android 运行时环境包含了完整的 SQLite。

SQLite 和其他数据库最大的不同就是对数据类型的支持，创建一个表时，可以在 CREATE TABLE 语句中指定某列的数据类型，但是你可以把任何数据类型放入任何列中。当某个值插入数据库时，SQLite 将检查它的类型。如果该类型与关联的列不匹配，则 SQLite 会尝试将该值转换成该列的类型。如果不能转换，则该值将作为其本身具有的类型存储。比如可以把一个字符串（String）放入 INTEGER 列。SQLite 称这为“弱类型”（manifest typing.）。 此外，SQLite 不支持一些标准的 SQL 功能，特别是外键约束（FOREIGN KEY constrains），嵌套 transcaction 和 RIGHT OUTER JOIN 和 FULL OUTER JOIN, 还有一些 ALTER TABLE 功能。 除了上述功能外，SQLite 是一个完整的 SQL 系统，拥有完整的触发器，交易等等。

Android 集成了 SQLite 数据库 Android 在运行时（run-time）集成了 SQLite，所以每个 Android 应用程序都可以使用 SQLite 数据库。

对于熟悉 SQL 的开发人员来时，在 Android 开发中使用 SQLite 相当简单。但是，由于 JDBC 会消耗太多的系统资源，所以 JDBC 对于手机这种内存受限设备来说并不合适。因此，Android 提供了一些新的 API 来使用 SQLite 数据库，Android 开发中，程序员需要学使用这些 API。

数据库存储在 data/< 项目文件夹 >/databases/ 下。 Android 开发中使用 SQLite 数据库 Activites 可以通过 Content Provider 或者 Service 访问一个数据库。

下面会详细讲解如果创建数据库，添加数据和查询数据库。 创建数据库 Android 不自动提供数据库。在 Android 应用程序中使用 SQLite，必须自己创建数据库，然后创建表、索引，填充数据。

Android 提供了 SQLiteOpenHelper 帮助你创建一个数据库，你只要继承 SQLiteOpenHelper 类，就可以轻松的创建数据库。SQLiteOpenHelper 类根据开发应用程序的需要，封装了创建和更新数据库使用的逻辑。

SQLiteOpenHelper 的子类，至少需要实现三个方法：

1 构造函数，调用父类 SQLiteOpenHelper 的构造函数。这个方法需要四个参数：上下文环境（例如，一个 Activity），数据库名字，一个可选的游标工厂（通常是 Null），一个代表你正在使用的数据库模型版本的整数。

2 onCreate（）方法，它需要一个 SQLiteDatabase 对象作为参数，根据需要对这个对象填充表和初始化数据。

3 onUpgrage() 方法，它需要三个参数，一个 SQLiteDatabase 对象，一个旧的版本号和一个新的版本号，这样你就可以清楚如何把一个数据库从旧的模型转变到新的模型。

下面示例代码展示了如何继承 SQLiteOpenHelper 创建数据库：

```
public class DatabaseHelper extends SQLiteOpenHelper {

  DatabaseHelper(Context context, String name, CursorFactory cursorFactory, int v
ersion)
  {
    super(context, name, cursorFactory, version);
     }

     @Override
     public void onCreate(SQLiteDatabase db) {
          // TODO 创建数据库后，对数据库的操作
     }

     @Override
 public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
          // TODO 更改数据库版本的操作
     }

 @Override
 public void onOpen(SQLiteDatabase db) {
          super.onOpen(db);
          // TODO 每次成功打开数据库后首先被执行
```

```
    }
}
```

接下来讨论具体如何创建表、插入数据、删除表等等。调用 getReadableDatabase() 或 getWriteableDatabase() 方法，你可以得到 SQLiteDatabase 实例，具体调用那个方法，取决于你是否需要改变数据库的内容：

```
db=(new DatabaseHelper(getContext())).getWritableDatabase();
return (db == null) ? false : true;
```

上面这段代码会返回一个 SQLiteDatabase 类的实例，使用这个对象，你就可以查询或者修改数据库。当你完成了对数据库的操作（例如你的 Activity 已经关闭），需要调用 SQLiteDatabase 的 Close() 方法来释放掉数据库连接。 创建表和索引 为了创建表和索引，需要调用 SQLiteDatabase 的 execSQL() 方法来执行 DDL 语句。如果没有异常，这个方法没有返回值。

例如，你可以执行如下代码：

```
db.execSQL("CREATE TABLE mytable (_id INTEGER PRIMARY KEY AUTOINCREMENT, titl
e TEXT, value REAL);");
```

这条语句会创建一个名为 mytable 的表，表有一个列名为 _id，并且是主键，这列的值是会自动增长的整数（例如，当你插入一行时，SQLite 会给这列自动赋值），另外还有两列：title(字符) 和 value(浮点数)。 SQLite 会自动为主键列创建索引。 通常情况下，第一次创建数据库时创建了表和索引。

如果你不需要改变表的 schema，不需要删除表和索引 . 删除表和索引，需要使用 execSQL() 方法调用 DROP INDEX 和 DROP TABLE 语句。 给表添加数据 上面的代码，已经创建了数据库和表，现在需要给表添加数据。有两种方法可以给表添加数据。

像上面创建表一样，你可以使用 execSQL() 方法执行 INSERT, UPDATE, DELETE 等语句来更新表的数据。execSQL() 方法适用于所有不返回结果的 SQL 语句。

例如： db.execSQL("INSERT INTO widgets (name, inventory)"+ "VALUES ('Sprocket', 5)");

另一种方法是使用 SQLiteDatabase 对象的 insert(), update(), delete() 方法。这些方法把 SQL 语句的一部分作为参数。

示例如下：

ContentValues cv=new ContentValues();

cv.put(Constants.TITLE, "example title");

cv.put(Constants.VALUE, SensorManager.GRAVITY_DEATH_STAR_I);

db.insert("mytable", getNullColumnHack(), cv);

update（）方法有四个参数，分别是表名，表示列名和值的 ContentValues 对象，可选的 WHERE 条件和可选的填充 WHERE 语句的字符串，这些字符串会替换 WHERE 条件中的“？”标记。

update() 根据条件，更新指定列的值，所以用 execSQL() 方法可以达到同样的目的。 WHERE 条件和其参数和用过的其他 SQL APIs 类似。

例如：

String[] parms=new String[] {"this is a string"};

db.update("widgets", replacements, "name=?", parms);

delete() 方法的使用和 update() 类似，使用表名，可选的 WHERE 条件和相应的填充 WHERE 条件的字符串。 查询数据库 类似 INSERT, UPDATE, DELETE，有两种方法使用 SELECT 从 SQLite 数据库检索数据。

1．使用 rawQuery() 直接调用 SELECT 语句： 使用 query() 方法构建一个查询。

Raw Queries 正如 API 名字，rawQuery() 是最简单的解决方法。通过这个方法你就可以调用 SQL SELECT 语句。

例如： Cursor c=db.rawQuery("SELECT name FROM sqlite_master WHERE type='table' AND name='mytable'", null);

在上面例子中，我们查询 SQLite 系统表（sqlite_master）检查 table 表是否存在。返回值是一个 cursor 对象，这个对象的方法可以迭代查询结果。 如果查询是动态的，使用这个方法就会非常复杂。

例如，当你需要查询的列在程序编译的时候不能确定，这时候使用 query() 方法会方便很多。

Regular Queries query() 方法用 SELECT 语句段构建查询。SELECT 语句内容作为 query() 方法的参数，比如：要查询的表名，要获取的字段名，WHERE 条件，包含可选的位置参数，去替代 WHERE 条件中位置参数的值，GROUP BY 条件，HAVING 条件。除了表名，其他参数可以是 null。所以，以前的代码段可以可写成：

```
String[] columns={"ID", "inventory"};
```

```
String[] parms={"snicklefritz"};

Cursor result=db.query("widgets", columns, "name=?",parms, null, null, null);
```

使用游标

不管你如何执行查询，都会返回一个 Cursor，这是 Android 的 SQLite 数据库游标，

使用游标，你可以：

通过使用 getCount() 方法得到结果集中有多少记录；

通过 moveToFirst(), moveToNext(), 和 isAfterLast() 方法遍历所有记录；

通过 getColumnNames() 得到字段名；

通过 getColumnIndex() 转换成字段号；

通过 getString()，getInt() 等方法得到给定字段当前记录的值；

通过 requery() 方法重新执行查询得到游标；

通过 close() 方法释放游标资源；

例如，下面代码遍历 mytable 表：

```
Cursor result=db.rawQuery("SELECT ID, name, inventory FROM mytable");

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();
```

在 Android 中使用 SQLite 数据库管理工具 在其他数据库上作开发，一般都使用工具来检查和处理数据库的内容，而不是仅仅使用数据库的 API。

使用 Android 模拟器，有两种可供选择的方法来管理数据库。

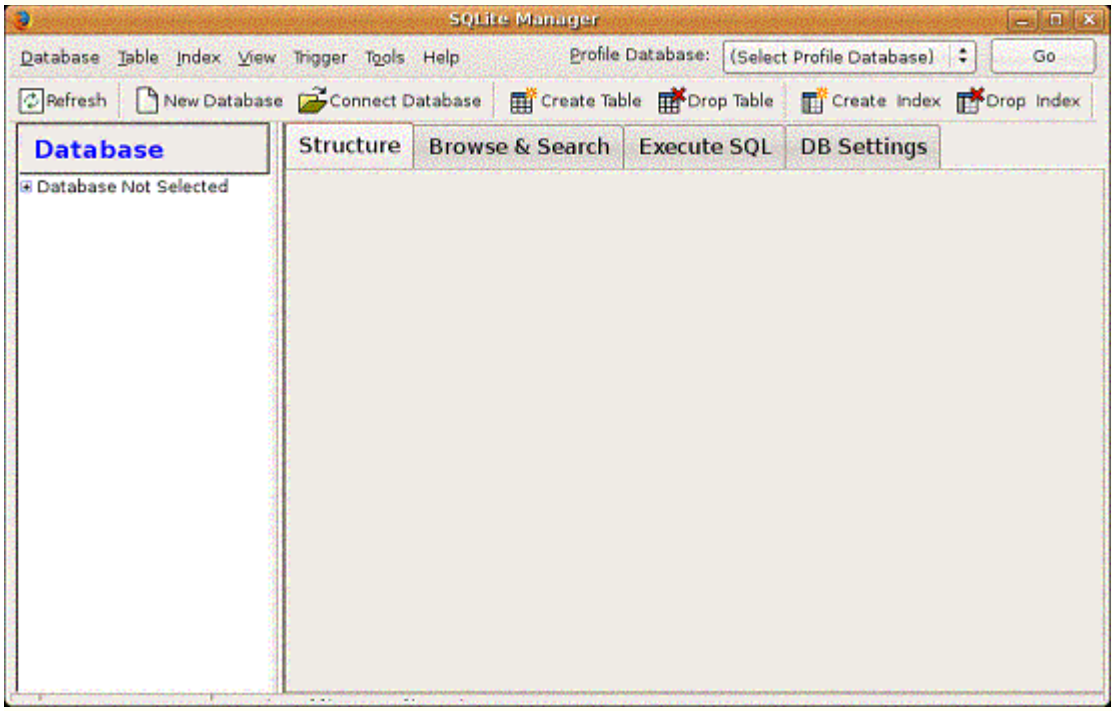
首先，模拟器绑定了 sqlite3 控制台程序，可以使用 adb shell 命令来调用他。只要你进入了模拟器的 shell，在数据库的路径执行 sqlite3 命令就可以了。

数据库文件一般存放在： /data/data/your.app.package/databases/your-db-name 如果你喜欢使用更友好的工具，你可以把数据库拷贝到你的开发机上，使用 SQLite-aware 客户端来操作它。这样的话，你在一个数据库的拷贝上操作，如果你想要你的修改能反映到设备上，你需要把数据库备份回去。

把数据库从设备上考出来，你可以使用 adb pull 命令（或者在 IDE 上做相应操作）。

存储一个修改过的数据库到设备上，使用 adb push 命令。 一个最方便的 SQLite 客户端是 FireFox SQLite Manager 扩展，它可以跨所有平台使用。

下图是 SQLite Manager 工具：



如果你想要开发 Android 应用程序，一定需要在 Android 上存储数据，使用 SQLite 数据库是一种非常好的选择。

第四种： 使用 ContentProvider 存储数据

Android 这个系统和其他的操作系统还不太一样，我们需要记住的是，数据在 Android 当中是私有的，当然这些数据包括文件数据和数据库数据以及一些其他类型的数据。那这个时候有读者就会提出问题，难道两个程序之间就没有办法对于数据进行交换？Android 这么优秀的系统不会让这种情况发生的。解决这个问题主要靠 ContentProvider。一个 Content Provider 类实现了一组标准的方法接口，从而能够让其他的应用保存或读取此 Content Provider 的各种数据类型。也就是说，一个程序可以通过实现一个 Content Provider 的抽象接口将自己的数据暴露出去。外界根本看不到，也不用看到这个应用暴露的数据在应用当中是如何存储的，或者是用数据库存储还是用文件存储，还是通过网上获得，这些一切都不重要，重要的是外界可以通过这一套标准及统一的接口和程序里的数据打交道，可以读取程序的数据，也可以删除程序的数据，当然，中间也会涉及一些权限的问题。

一个程序可以通过实现一个 ContentProvider 的抽象接口将自己的数据完全暴露出去，而且 ContentProviders 是以类似数据库中表的方式将数据暴露，也就是说 ContentProvider 就像一个“数据库”。那么外界获取其提供的数据，也就应该与从数据库中获取数据的操作基本一样，只不过是采用 URI 来表示外界需要访问的“数据库”。

Content Provider 提供了一种多应用间数据共享的方式，比如：联系人信息可以被多个应用程序访问。

Content Provider 是个实现了一组用于提供其他应用程序存取数据的标准方法的类。 应用程序可以在 Content Provider 中执行如下操作：查询数据 修改数据 添加数据 删除数据

标准的 Content Provider：Android 提供了一些已经在系统中实现的标准 Content Provider，比如联系人信息，图片库等等，你可以用这些 Content Provider 来访问设备上存储的联系人信息，图片等等。

查询记录：

在 Content Provider 中使用的查询字符串有别于标准的 SQL 查询。很多诸如 select, add, delete, modify 等操作我们都使用一种特殊的 URI 来进行，这种 URI 由 3 个部分组成， “content://”, 代表数据的路径，和一个可选的标识数据的 ID。

以下是一些示例 URI：

content://media/internal/images 这个 URI 将返回设备上存储的所有图片

content://contacts/people/ 这个 URI 将返回设备上的所有联系人信息

content://contacts/people/45 这个 URI 返回单个结果（联系人信息中 ID 为 45 的联系人记录）

尽管这种查询字符串格式很常见，但是它看起来还是有点令人迷惑。为此，Android 提供一系列的帮助类（在 android.provider 包下），里面包含了很多以类变量形式给出的查询字符串，这种方式更容易让我们理解一点，参见下例：

MediaStore.Images.Media.INTERNAL_CONTENT_URI Contacts.People.CONTENT_URI

因此，如上面 content://contacts/people/45 这个 URI 就可以写成如下形式：

Uri person = ContentUris.withAppendedId(People.CONTENT_URI, 45);

然后执行数据查询：Cursor cur = managedQuery(person, null, null, null);

这个查询返回一个包含所有数据字段的游标，我们可以通过迭代这个游标来获取所有的数据：

```
package com.wissen.testApp;
public class ContentProviderDemo extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
       displayRecords();
    }
```

```
    private void displayRecords() {
        //该数组中包含了所有要返回的字段
     String columns[] = new String[] { People.NAME, People.NUMBER };
       Uri mContacts = People.CONTENT_URI;
       Cursor cur = managedQuery(
           mContacts,
          columns,  // 要返回的数据字段
       null,           // WHERE 子句
       null,          // WHERE 子句的参数
       null          // Order-by 子句
     );
       if (cur.moveToFirst()) {
           String name = null;
           String phoneNo = null;
           do {
              // 获取字段的值
         name = cur.getString(cur.getColumnIndex(People.NAME));
              phoneNo = cur.getString(cur.getColumnIndex(People.NUMBER));
              Toast.makeText(this, name + " " + phoneNo, Toast.LENGTH_LONG).show();
          } while (cur.moveToNext());
       }
    }
}
```

上例示范了一个如何依次读取联系人信息表中的指定数据列 name 和 number。

修改记录:

我们可以使用 ContentResolver.update()方法来修改数据，我们来写一个修改数据的方法:

```
   private void updateRecord(int recNo, String name) {

        Uri uri = ContentUris.withAppendedId(People.CONTENT_URI, recNo);
         ContentValues values = new ContentValues();
         values.put(People.NAME, name);
         getContentResolver().update(uri, values, null, null);

   }
```

现在你可以调用上面的方法来更新指定记录： updateRecord(10, ”XYZ”); //更改第 10 条记录的 name 字段值为 “XYZ”

添加记录:

要增加记录，我们可以调用 ContentResolver.insert()方法，该方法接受一个要增加的记录的目标 URI，以及一个包含了新记录值的 Map 对象，调用后的返回值是新记录的 URI，包含记录号。

上面的例子中我们都是基于联系人信息簿这个标准的 Content Provider，现在我们继续来创建一个 insertRecord() 方法以对联系人信息簿中进行数据的添加：

```
private void insertRecords(String name, String phoneNo) {

   ContentValues values = new ContentValues();
    values.put(People.NAME, name);
    Uri uri = getContentResolver().insert(People.CONTENT_URI, values);
    Log.d(”ANDROID”, uri.toString());
    Uri numberUri = Uri.withAppendedPath(uri, People.Phones.CONTENT_DIRECTORY);
    values.clear();
    values.put(Contacts.Phones.TYPE, People.Phones.TYPE_MOBILE);
    values.put(People.NUMBER, phoneNo);
    getContentResolver().insert(numberUri, values);
}
```

这样我们就可以调用 insertRecords(name, phoneNo)的方式来向联系人信息簿中添加联系人姓名和电话号码。

删除记录:

Content Provider 中的 getContextResolver.delete()方法可以用来删除记录。

下面的记录用来删除设备上所有的联系人信息：

```
private void deleteRecords() {
Uri uri = People.CONTENT_URI;
getContentResolver().delete(uri, null, null);
    }
```

你也可以指定 WHERE 条件语句来删除特定的记录：

getContentResolver().delete(uri, "NAME=" + "'XYZ XYZ'", null);

这将会删除 name 为 'XYZ XYZ'的记录。

创建 Content Provider:

至此我们已经知道如何使用 Content Provider 了，现在让我们来看下如何自己创建一个 Content Provider。

要创建我们自己的 Content Provider 的话，我们需要遵循以下几步：

1. 创建一个继承了 ContentProvider 父类的类

2. 定义一个名为 CONTENT_URI，并且是 public static final 的 Uri 类型的类变量，你必须为其指定一个唯一的字符串值，最好的方案是以类的全名称，

如: public static final Uri CONTENT_URI = Uri.parse("content://com.google.android.MyContentProvider");

3. 创建你的数据存储系统。大多数 Content Provider 使用 Android 文件系统或 SQLite 数据库来保持数据，但是你也可以以任何你想要的方式来存储。

4. 定义你要返回给客户端的数据列名。如果你正在使用 Android 数据库，则数据列的使用方式就和你以往所熟悉的其他数据库一样。但是，你必须为其定义一个叫_id 的列，它用来表示每条记录的唯一性。

5. 如果你要存储字节型数据，比如位图文件等，那保存该数据的数据列其实是一个表示实际保存文件的 URI 字符串，客户端通过它来读取对应的文件数据，处理这种数据类型的 Content Provider 需要实现一个名为_data 的字段，_data 字段列出了该文件在 Android 文件系统上的精确路径。这个字段不仅是供客户端使用，而且也可以供 ContentResolver 使用。客户端可以调用 ContentResolver.openOutputStream()方法来处理该 URI 指向的文件资源，如果是 ContentResolver 本身的话，由于其持有的权限比客户端要高，所以它能直接访问该数据文件。

6. 声明 public static String 型的变量，用于指定要从游标处返回的数据列。

7. 查询返回一个 Cursor 类型的对象。所有执行写操作的方法如 insert(), update() 以及 delete()都将被监听。我们可以通过使用 ContentResover().notifyChange()方法来通知监听器关于数据更新的信息。

8. 在 AndroidMenifest.xml 中使用标签来设置 Content Provider。

9. 如果你要处理的数据类型是一种比较新的类型，你就必须先定义一个新的 MIME 类型，以供 ContentProvider.geType(url)来返回。

MIME 类型有两种形式:

一种是为指定的单个记录的，还有一种是为多条记录的。这里给出一种常用的格式: vnd.android.cursor.item/vnd.yourcompanyname.contenttype （单个记录的 MIME 类型） 比如, 一个请求列车信息的 URI 如 content://com.example.transportationprovider/trains/122 可能就会返回 typevnd.android.cursor.item/vnd.example.rail 这样一个 MIME 类型。

vnd.android.cursor.dir/vnd.yourcompanyname.contenttype （多个记录的 MIME 类型） 比如，一个请求所有列车信息的 URI 如 content://com.example.transportationprovider/trains 可能就会返回 vnd.android.cursor.dir/vnd.example.rail 这样一个 MIME 类型。

下列代码将创建一个 Content Provider，它仅仅是存储用户名称并显示所有的用户名称（使用 SQLLite 数据库存储这些数据）：

```
package com.wissen.testApp;
public class MyUsers {
    public static final String AUTHORITY   = "com.wissen.MyContentProvider";

    // BaseColumn 类中已经包含了 _id 字段
   public static final class User implements BaseColumns {
        public static final Uri CONTENT_URI   = Uri.parse("content://com.wissen.MyConte
ntProvider");
        // 表数据列
     public static final String  USER_NAME   = "USER_NAME";
    }

}
```

上面的类中定义了 Content Provider 的 CONTENT_URI，以及数据列。下面我们将定义基于上面的类来定义实际的 Content Provider 类：

```
package com.wissen.testApp.android;
public class MyContentProvider extends ContentProvider {
    private SQLiteDatabase     sqlDB;
    private DatabaseHelper    dbHelper;
    private static final String  DATABASE_NAME     = "Users.db";
    private static final int        DATABASE_VERSION         = 1;
    private static final String TABLE_NAME    = "User";
    private static final String TAG = "MyContentProvider";

    private static class DatabaseHelper extends SQLiteOpenHelper {
        DatabaseHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }

        @Override
```

```
        public void onCreate(SQLiteDatabase db) {
            //创建用于存储数据的表
        db.execSQL("Create table " + TABLE_NAME + "( _id INTEGER PRIMARY KEY AUTOINCRE
MENT, USER_NAME TEXT);");
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
            onCreate(db);
        }
    }

    @Override
    public int delete(Uri uri, String s, String[] as) {
        return 0;
    }

    @Override
    public String getType(Uri uri) {
        return null;
    }

    @Override
    public Uri insert(Uri uri, ContentValues contentvalues) {
        sqlDB = dbHelper.getWritableDatabase();
        long rowId = sqlDB.insert(TABLE_NAME, "", contentvalues);
        if (rowId > 0) {
            Uri rowUri = ContentUris.appendId(MyUsers.User.CONTENT_URI.buildUpon(), ro
wId).build();
            getContext().getContentResolver().notifyChange(rowUri, null);
            return rowUri;
        }
        throw new SQLException("Failed to insert row into " + uri);
    }

    @Override
    public boolean onCreate() {
        dbHelper = new DatabaseHelper(getContext());
        return (dbHelper == null) ? false : true;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection, String[] selec
```

```
tionArgs, String sortOrder) {
        SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
        SQLiteDatabase db = dbHelper.getReadableDatabase();
        qb.setTables(TABLE_NAME);
        Cursor c = qb.query(db, projection, selection, null, null, null, sortOrder);
        c.setNotificationUri(getContext().getContentResolver(), uri);
        return c;
    }

    @Override
    public int update(Uri uri, ContentValues contentvalues, String s, String[] as) {
        return 0;
    }
}
```

一个名为 MyContentProvider 的 Content Provider 创建完成了，它用于从 Sqlite 数据库中添加和读取记录。

Content Provider 的入口需要在 AndroidManifest.xml 中配置:

之后，让我们来使用这个定义好的 Content Provider:

```
package com.wissen.testApp;
public class MyContentDemo extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        insertRecord(”MyUser”);
        displayRecords();
    }

    private void insertRecord(String userName) {
        ContentValues values = new ContentValues();
        values.put(MyUsers.User.USER_NAME, userName);
        getContentResolver().insert(MyUsers.User.CONTENT_URI, values);
    }

    private void displayRecords() {
        String columns[] = new String[] { MyUsers.User._ID, MyUsers.User.USER_NAME };
        Uri myUri = MyUsers.User.CONTENT_URI;
```

```
        Cursor cur = managedQuery(myUri, columns,null, null, null );
        if (cur.moveToFirst()) {
            String id = null;
            String userName = null;
            do {
                id = cur.getString(cur.getColumnIndex(MyUsers.User._ID));
                userName = cur.getString(cur.getColumnIndex(MyUsers.User.USER_NAME));
                Toast.makeText(this, id + " " + userName, Toast.LENGTH_LONG).show();
           } while (cur.moveToNext());
       }
    }
}
```

上面的类将先向数据库中添加一条用户数据，然后显示数据库中所有的用户数据。

第五种： 网络存储数据

前面介绍的几种存储都是将数据存储在本地设备上，除此之外，还有一种存储（获取）数据的方式，通过网络来实现数据的存储和获取。

我们可以调用 WebService 返回的数据或是解析 HTTP 协议实现网络数据交互。

具体需要熟悉 java.net.*，Android.net.*这两个包的内容，在这就不赘述了，请大家参阅相关文档。

下面是一个通过地区名称查询该地区的天气预报，以 POST 发送的方式发送请求到 webservicex.net 站点，访问 WebService.webservicex.net 站点上提供查询天气预报的服务。

代码如下：

```
package com.android.weather;

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

import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
```

```
import org.apache.http.protocol.HTTP;
import org.apache.http.util.EntityUtils;

import android.app.Activity;
import android.os.Bundle;

public class MyAndroidWeatherActivity extends Activity {
    //定义需要获取的内容来源地址
    private static final String SERVER_URL =
        "http://www.webservicex.net/WeatherForecast.asmx/GetWeatherByPlaceName";


    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        HttpPost request = new HttpPost(SERVER_URL); //根据内容来源地址创建一个 Http 请求
        // 添加一个变量
        List<NameValuePair> params = new ArrayList<NameValuePair>();
        // 设置一个地区名称
        params.add(new BasicNameValuePair("PlaceName", "NewYork"));  //添加必须的参数


        try {
            //设置参数的编码
            request.setEntity(new UrlEncodedFormEntity(params, HTTP.UTF_8));
            //发送请求并获取反馈
            HttpResponse httpResponse = new DefaultHttpClient().execute(request);

            // 解析返回的内容
            if(httpResponse.getStatusLine().getStatusCode() != 404){
               String result = EntityUtils.toString(httpResponse.getEntity());
               System.out.println(result);
            }
        } catch (Exception e) {
            e.printStackTrace();
       }
    }
}
```

别忘记了在配置文件中设置访问网络权限：

```
<uses-permission android:name="android.permission.INTERNET" />
```