**Android学习小结**

|  |  |  |  |
| --- | --- | --- | --- |
| 编制： | 张 亮 | 编制日期： | 2017/12/5 |

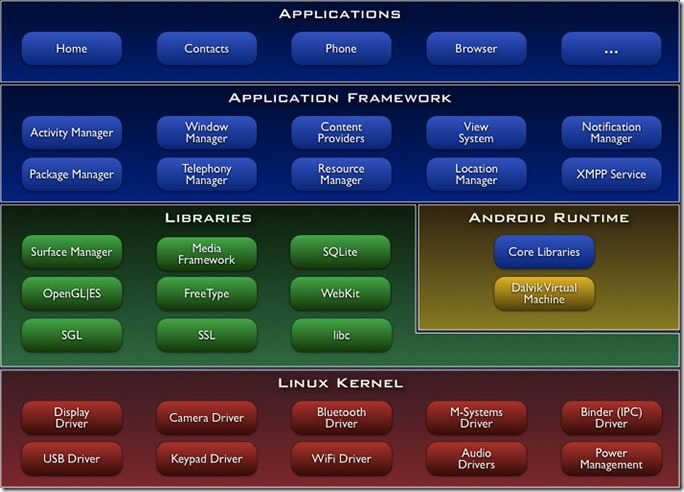
1. **搭建开发环境**
   1. 配置Java环境：

在macOS安装Java相对来说比较简单，只需要在Oracle网站上下载Mac安装包，然后双击PKG文件进行安装即可。

* 1. Android Studio安装

直接在[Android Developer官网](https://developer.android.com/studio/index.html)下载后安装，安装成功后进入到设置向导页经过简单设置并下载依赖的组件新建项目就能正常使用。

1. **Android基础知识**
   1. Android系统架构



如图自底往上分别为：

* 第一层:LINUX KERNEL，Android的核心系统服务依赖于Linux 2.6 内核，如安全性，内存管理，进程管理， 网络协议栈和驱动模型。 Linux 内核也同时作为硬件和软件栈之间的抽象层。其外还对其做了部分修改，主要涉及两部分修改：Binder (IPC)、电源管理
* 第二层：Libraris & Android Runtime，1）Libraris：Android 包含一些C/C++库，这些库能被Android系统中不同的组件使用。它们通过 Android 应用程序框架为开发者提供服务。以下是一些核心库：➀Bionic系统C库；➁媒体库- 基于 PacketVideo OpenCORE；该库支持多种常用的音频、视频格式回放和录制，同时支持静态图像文件。编码格式包括MPEG4, H.264, MP3, AAC, AMR, JPG, PNG；➂Surface Manager；➃Webkit,LibWebCore；➄SGL；➅3D libraries；➆FreeType；➇SQLite；➈ HAL Layer-硬件抽象层:framework层和操作系统之间的纽带，驱动层是没有业务逻辑的,基本都是IO操作，比如启动摄像头,驱动层是没有的,但是在Framework层就会有，这样就需要个纽带。一层封装,有点稍微点的逻辑性，是和硬件打交道。2）Android Runtime：该核心库提供了JAVA编程语言核心库的大多数功能。每一个Android应用程序都在它自己的进程中运 行，都拥有一个独立的Dalvik虚拟 机实例。Dalvik被设计成一个设备可以同时高效地运行多个虚拟系统。 Dalvik虚拟机执行（.dex）的Dalvik可执行文件，该格式文件针对小内存使用做了 优化。同时虚拟机是基于寄存器的，所有的类都经由JAVA编译器编译，然后通过SDK中 的 "dx" 工具转化成.dex格式由虚拟机执行。
* 第三层：Application Frameworks，提供调用系统功能的API；这层提供的API也是java调用，很多的Manager.比如:内容提供者,资源管理器,通知管理器,活动管理器,包管理器等,此层次通过JNI来调C++的这一层。
* 第四层：Applications，Android会同一系列核心应用程序包一起发布，该应用程序包包括email客户端，SMS短消息程序，日历，地图，浏览器，联系人管理程序等。所有的应用程序都是使用JAVA语言编写的。通常开发人员就处在这一层。
  1. Android系统特性
* Android在独立的进程里运动每一个应用程序，每个进程都有自己的虚拟机。这提供了一个受保护的内存环境。而且,通过将应用程序隔离到独立进程中，系统可以控制哪个应用程序具有较高级别优先级。
* 每一个进程都有自己的虚拟机，所以每个应用程序相对独立。
* Android操作系统是基于Linux的多用户操作系统,每个应用程序都是使用不同的用户。(启动和安装时，和pid[进程id]一样，是个临时用户，消亡时就消除了。)这样各个应用程序之间的安全性屏蔽的比较好。
* 默认情况，一个应用一个userid,系统设定的，不需要我们手动去设定。针对这样的应用程序,该用户有权限进行所有操作。
* 默认情况下，应用程序都有自己的Linux进程，系统会决定何时关闭它，即当资源不够时。
* 可以给两个应用程序共享同一个userid,让彼此之间共享资源，应用程序可以请求访问系统资源，比如通讯录等，但需要安装时候被授权。
* 虚拟机只执行.dex的可执行文件。当Java程序通过编译,最后还需要通过SDK中的dx工具转化为.dex格式才能正常在虚拟机上执行。
* Art模式：程序在安装时需要预编译读取,将代码转换为机器码，

a) 好处：程序运行时，无需时时转换,运行速度快

b) 坏处：安装时间过长，由于转换机器码，占用存储略高

* 1. Android系统启动
* 守护进程(Zygote)->创建新的vm->加载app，并且管理app。(这里守护进程加载时用到PackageManagerService来实例化Activity)
* .java->.class->dx->.dex->(打包签名)->apk
* Ru的过程:Run->编译dex->打包apk->选择target设备->上传apk->安装apk->启动主activity
* 每个apk安装时，就会有个系统的哈希表(常驻内存，管理当前已安装的应用程序，以及应用程序安装后的地址)。当前LauncherActivity就可以读出来表。并且显示出来,当点击一个应用程序时，就是调用了startActivity()，其实就是两个activity的切换。启动了一个进程的Activity，那那个进程也就随即被启动了，那到底要start哪个Activity呢，在加载的时候就已经查到了这个intent-filter中有launcher的category的那个Activity,Launcher直接扫描intent-filter。
* 在系统启动的时候,PackageManagerService(系统级服务)会读取Manifest文件的信息。而且是读取所安装的所有应用程序的Manifest的信息，并且会建立信息库，在系统级共享内存里面。供查询(可以看作可供查询的数据库表)。
* 守护进程加载时使用到PackageManagerService(帮Activity实例化)->Manifest->主Activity->Layout
* 启动时：PackageManager(依赖于PackageManagerService)会去找主Activity，如果没进程，Zegote就会实例化一个DavlikVM虚拟机进程去加载。
* 运行过程：当安装了这个应用程序.开始部署后，Android系统就会读取这个文件中的功能清单文件AndroidMainfest(假如有个注册表对各个组件登记)，就知道应用中有哪些功能组件，就开始登记这些组件，如：

组件 Action Category

HelloActivity action.Main category.LAUNCHER

然后用户点击应用程序,操作系统将点击行为封装成一个Intent(包含两个内容，一个是action一个是Category)，发送给(传递给)操作系统的intent处理器，intent处理器在注册表里的这个应用里面查找与这个应用匹配的组件。就知道了这个Activity了，假设没有启动，就先实例化HelloActivity(通过反射)，之后调用onCreate(初始化工作，只会执行一次)。

* 1. 程序资源访问
     1. 一般资源访问
* res文件夹下不支持嵌套文件夹,即:不能在layout文件夹下再建文件夹
* 获取资源的方式：
  + 获取xml:getXML();
  + 获取raw:getResources().openResource(); or getRawResource();
  + 获取string.xml:getResources().getStringArray();or getText()
  + 获取color.xml:getResources().getColor();
  + 获取drawable.xml:getResource().getDrawable();
  + 获取dimens.xml:getDimension(resId);
    1. res/raw文件夹访问

InputStream is = getResources().openRawResource(R.raw.filename);

* + 1. Assets文件夹访问
* AssetManager am = getAssets()

InputStream is = am.open(“fileName”);

* 主要存放一些不常用的原生资源文件.
* 保存应用的资源文件，不会频繁的被访问(不经常被用户修改的)，例如声音、音频、视频
  + 1. res/raw和assets异同
       1. 相同点

两者目录下的文件在打包后会原封不动的保存在apk包中,不会被编译成二进制.

* + - 1. 不同点
* res/raw中的文件会被映射到R.java文件中，访问的时候直接使用资源ID即R.id.filename；assets文件夹下的文件不会被映射到R.java中，访问的时候需要AssetManager类。
* res/raw不可以有目录结构,而assets则可以有目录结构，也就是assets目录下可以再建立文件夹
* res是直接访问，assets是通过路径访问的
* res不用的不会打包进应用,assets会全部打包进去。res中的都会产生ID。一般大文件放在assets里面。比如放一些Mp3之类的。
  1. Android Mainfest文件
* 可以自定义权限：<permission /> 在activity中设置android:permission
* Manifest如何解析?->何时:在安装的时候
  + 应用程序在安装的时候,manifest信息会放到一张系统索引表里面。
  + 系统启动的时候,PackageManagerService，找到配置为mainActivity
  + 应用程序启动时候，Launcher进行系统级判断，比如最小SDK等。
  + 四大组件在需要的时候初始化
* 内部配置:
  + uses-featrue：比如设置name为摄像头的，为true，就是说,在market中，为用户过滤应用时，显示有摄像头的程序。
  + <activity> android:screenoretation//默认的屏幕显示:横屏或者竖屏。
  + activity-alias必须要放在<activity >之后，这个是activity的别名。
  + “android:versionCode”指定应用的版本号。如果应用需要不断升级，在升级的时候应该修改该值。
  + <manifest android:installLocation=“auto/preferExternal” –指定apk安装的位置。
  1. Android进程等级
* 进程的销毁是系统来控制的，不需要自己来控制。
* 分五种进程等级：
* Active Process(前台进程)：不会被自动干掉。(消耗资源最高，优先级别最高，前台进程是不会因为内存少而被杀掉，如果要杀死，要得到用户的允许) 一个进程如果满足下列任何条件被认为是前台进程：➀正运行着一个正在与用户交互的活动（Activity对象的onResume()方法已经被调用）；➁这个进程拥有一个绑定到正在与用户交互的activity上的Service；➂这个进程拥有一个前台运行的Service—service调用了方法startForeground()；➃这个进程拥有一个正在执行其任何一个生命周期回调方法（onCreate(),onStart(或onDestroy())的Service；➄这个进程拥有正在执行其onReceive()方法的BroadcastReceiver；
* Visible Process(可见进程)：一个进程不拥有运行于前台的组件，但是依然能影响用户所见。一个可见的进程是极其重要的，通常不会被终止，除非内存不够，需要释放内存以便前台进程运行。满足下列条件时，进程即为可见：➀这个进程拥有一个不在前台但仍可见的Activity(它的onPause()方法被调用)；➁这个进程拥有一个绑定在前台（或者可见）Activity的服务。
* Service Porcess(服务进程)：一个进程不在上述两种之内，但它运行着一个被startService()所启动的service。尽管一个服务进程不直接影响用户所见，但是它们通常做一些用户关心的事情（比如播放音乐或下载数据），所以除非系统没有足够的空间运行前台进程和可见进程时才会终止一个服务进程。
* Background Process（后台进程）：不可见状态的Activity进程，又没有任何相关的Service运行的(Service的onDestory()被调用)，系统有个进程列，,一般情况采取后进先出的原则来释放进程资源。比如按了home键，不可见，不消耗资源。
* Empty Process(空进程)：没有运行任何Components的进程,保留这个进程主要是为了缓存的需要；等待垃圾的回收。
  1. Android日志
* 日志级别：VERBOSE、DEBUG、INFO、WARN、ERROR
* 根据规范:建议Log.v和Log.d在开发中用，其他的在发布时
* 实现是用的native方法.是用的操作系统底层的方法, 设备驱动Logger(软驱动)
* 在data/log中(开发模式下的机器)
* 何时会产生:
  + 程序异常退出.uncasused exception
  + 强制关闭了.(Force Closed)
  + 程序无响应.ANR(界面操作(按钮点击)等待响应时间超过5秒;(onHandleMessage回调函数中执行超过10秒,接受到消息onReceiver()方法处理超过10秒)，IO操作和复杂的运算最好用异步.)

1. **Android四大基本组件**
   1. Activity
      1. Activity基础

* Activity是业务类。是承载应用程序的界面以及业务行为的基础。包括UI，Service…类似于JavaBean。
* Activity的事件是Window来触发的。
* Activity显示：启动，然后Window加载layout生成控件树。
* 窗体Window加载元素view在上面,窗体本身不可见,只是承载元素view的容器，这样就构成了setContentView。
* 事件是控件分发给view的，然后分发给window，然后window来处理。
* Activity常用到的事件：
  + onKeyDown(int keyCode, KeyEvent event) 按键按下事件；
  + onTouchEvent(MotionEvent event) 点击屏幕事件；
  + onKeyUp(int keyCode, KeyEvent event) 按键松开事件；
    1. Activity深入

理解：Activity(脸面)中this.setContentView(View)(实际是Window类来执行的)。设置的并不是Activity和Layout的关系,而是设置的view(五官,可以是任意多个)的关系。

* + 1. Activity生命周期



Activity整个生命周期的4种状态、7个重要方法和3个嵌套循环。

* 四种状态
  + 活动（Activity/Running）状态：当Activity运行在屏幕前台(处于当前任务活动栈的最上面),此时它获取了焦点能响应用户的操作,属于运行状态，同一个时刻只会有一个Activity处于活动(Activity)或运行(Running)状态。
  + 暂停(Paused)状态：当Activity失去焦点但仍对用户可见(如在它之上有另一个透明的Activity或Toast、AlertDialog等弹出窗口时)它处于暂停状态。暂停的Activity仍然是存活状态(它保留着所有的状态和成员信息并保持和窗口管理器的连接),但是当系统内存极小时可以被系统杀掉。
  + 停止(Stopped)状态：完全被另一个Activity遮挡时处于停止状态,它仍然保留着所有的状态和成员信息。只是对用户不可见,当其他地方需要内存时它往往被系统杀掉。
  + 非活动（Dead）状态：Activity 尚未被启动、已经被手动终止，或已经被系统回收时处于非活动的状态，要手动终止Activity，可以在程序中调用"finish"方法。如果是（按根据内存不足时的回收规则）被系统回收，可能是因为内存不足了内存不足时，Dalvak 虚拟机会根据其内存回收规则来回收内存：➀先回收与其他Activity 或Service/Intent Receiver 无关的进程(即优先回收独立的Activity)因此建议,我们的一些(耗时)后台操作，最好是作成Service的形式；➁不可见(处于Stopped状态的)Activity；➂ Service进程(除非真的没有内存可用时会被销毁)；➃非活动的可见的(Paused状态的)Activity；➄当前正在运行（Active/Running状态的）Activity。
* 7个重要方法：当Activity从一种状态进入另一状态时系统会自动调用下面相应的方法来通知用户这种变化。
  + onCreate(Bundle savedInstanceState)：当Activity从一种状态进入另一状态时系统会自动调用下面相应的方法来通知用户这种变化。当Activity第一次被实例化的时候系统会调用,整个生命周期只调用1次这个方法，通常用于初始化设置
  + onStart()：当Activity可见未获得用户焦点不能交互时系统会调用；
  + onRestart()：当Activity已经停止然后重新被启动时系统会调用；
  + onResume()：当Activity可见且获得用户焦点能交互时系统会调用；
  + onPause()：当系统启动另外一个新的Activity时,在新Activity启动之前被系统调用保存现有的Activity中的持久数据、停止动画等,这个实现方法必须非常快。当系统而不是用户自己出于回收内存时，关闭了activity 之后。用户会期望当他再次回到这个activity 的时候，它仍保持着上次离开时的样子。此时用到了onSaveInstanceState()，方法onSaveInstanceState()用来保存Activity被杀之前的状态,在onPause()之前被触发,当系统为了节省内存销毁了Activity(用户本不想销毁)时就需要重写这个方法了,当此Activity再次被实例化时会通过onCreate(Bundle savedInstanceState)将已经保存的临时状态数据传入因为onSaveInstanceState()方法不总是被调用,触发条件为(按下HOME键,按下电源按键关闭屏幕,横竖屏切换情况下),你应该仅重写onSaveInstanceState()来记录activity的临时状态，而不是持久的数据。应该使用onPause()来存储持久数据。
  + onStop()：当Activity被新的Activity完全覆盖不可见时被系统调用；
  + onDestroy()：当Activity(用户调用finish()或系统由于内存不足)被系统销毁杀掉时系统调用,（整个生命周期只调用1次）用来释放onCreate ()方法中创建的资源,如结束线程等；
* 3个嵌套循环
  + Activity完整的生命周期:从第一次调用onCreate()开始直到调用onDestroy()结束；
  + Activity的可视生命周期:从调用onStart()到相应的调用onStop()在这两个方法之间,可以保持显示Activity所需要的资源。如在onStart()中注册一个广播接收者监听影响你的UI的改变,在onStop() 中注销。
  + Activity的前台生命周期:从调用onResume()到相应的调用onPause()。
  1. BroadcastReceiver

你的应用可以使用它对外部事件进行过滤只对感兴趣的外部事件(如当电话呼入时，或者数据网络可用时)进行接收并做出响应。广播接收器没有用户界面。然而，它们可以启动一个activity或serice 来响应它们收到的信息，或者用NotificationManager 来通知用户。通知可以用很多种方式来吸引用户的注意力──闪动背灯、震动、播放声音等。一般来说是在状态栏上放一个持久的图标，用户可以打开它并获取消息。

* + 1. 广播类型：
* 普通广播，通过Context.sendBroadcast(Intent myIntent)发送的；
* 有序广播，通过Context.sendOrderedBroadcast(intent, receiverPermission)发送的，该方法第2个参数决定该广播的级别，级别数值是在 -1000 到 1000 之间 , 值越大 , 发送的优先级越高；广播接收者接收广播时的级别级别（可通过intentfilter中的priority进行设置设为2147483647时优先级最高），同级别接收的先后是随机的， 再到级别低的收到广播，高级别的或同级别先接收到广播的可以通过abortBroadcast()方法截断广播使其他的接收者无法收到该广播，还有其他构造函数。
* 异步广播，通过Context.sendStickyBroadcast(Intent myIntent)发送的，还有sendStickyOrderedBroadcast(intent, resultReceiver, scheduler,  initialCode, initialData, initialExtras)方法，该方法具有有序广播的特性也有异步广播的特性；发送异步广播要： <uses-permission android:name="android.permission.BROADCAST\_STICKY" />权限，接收并处理完Intent后，广播依然存在，直到你调用removeStickyBroadcast(intent)主动把它去掉
  + 1. 监听广播Intent步骤：
* 写一个继承BroadCastReceiver的类,重写onReceive()方法,广播接收器仅在它执行这个方法时处于活跃状态。当onReceive()返回后，它即为失活状态,注意:为了保证用户交互过程的流畅,一些费时的操作要放到线程里,如类名SMSBroadcastReceiver。
* 注册该广播接收者,注册有两种方法程序动态注册和AndroidManifest文件中进行静态注册（可理解为系统中注册）。
  + 1. 生命周期

生命周期只有十秒左右，如果在 onReceive() 内做超过十秒内的事情，就会报ANR(Application No Response) 程序无响应的错误信息。它的生命周期为从回调onReceive()方法开始到该方法返回结果后结束。

* 1. Service
     1. 基本概念

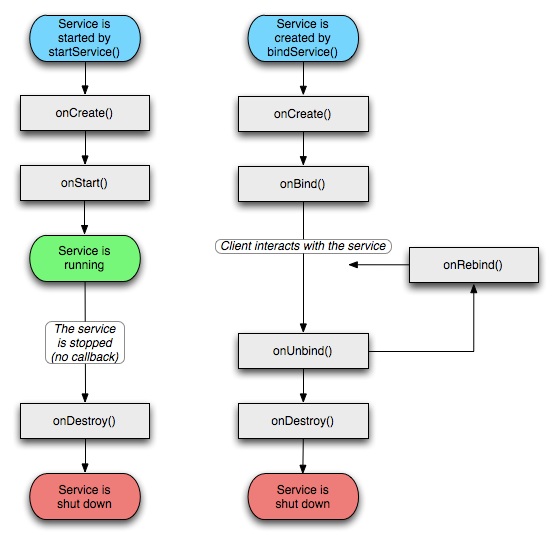
一个Service 是一段长生命周期的，没有用户界面的程序，可以用来开发如监控类程序。比较好的一个例子就是一个正在从播放列表中播放歌曲的媒体播放器。在一个媒体播放器的应用中，应该会有多个activity，让使用者可以选择歌曲并播放歌曲。然而，音乐重放这个功能并没有对应的activity，因为使用者当然会认为在导航到其它屏幕时音乐应该还在播放的。在这个例子中，媒体播放器这个activity 会使用Context.startService()来启动一个service，从而可以在后台保持音乐的播放。同时，系统也将保持这个service 一直执行，直到这个service 运行结束。另外，我们还可以通过使用Context.bindService()方法，连接到一个service 上（如果这个service 还没有运行将启动它）。当连接到一个service 之后，我们还可以service 提供的接口与它进行通讯。拿媒体播放器这个例子来说，我们还可以进行暂停、重播等操作。

* + 1. Service使用步骤如下：
* 继承service类
* AndroidManifast.xml配置清单文件中<application>节点里对服务进行配置：<service name=".SMSService"/>
  + 1. 启动Service

服务不能自己运行,需要通过Contex.startService()或Contex.bindService()启动服务。通过startService()方法启动的服务于调用者没有关系,即使调用者关闭了,服务仍然运行想停止服务要调用Context.stopService(),此时系统会调用onDestory(),使用此方法启动时,服务首次启动系统先调用服务的onCreate()-->onStart(),如果服务已经启动再次调用只会触发onStart()方法。

使用bindService()启动的服务与调用者绑定,只要调用者关闭服务就终止,使用此方法启动时,服务首次启动系统先调用服务的onCreate()-->onBind(),如果服务已经启动再次调用不会再触发这2个方法,调用者退出时系统会调用服务的onUnbind()-->onDestory(),想主动解除绑定可使用Contex.unbindService(),系统依次调用onUnbind()-->onDestory();

* + 1. 生命周期



* Service完整的生命周期:从调用onCreate()开始直到调用onDestroy()结束；
* Service有两种使用方法：
  + 以调用Context.startService()启动，而以调用Context.stopService()结束
  + 以调用Context.bindService()方法建立，以调用Context.unbindService()关闭；
    1. service和thread的选取
* 逻辑上认为一个后台操作，跟主线程交互密切，逻辑关系密切，那就用多线程；
* 逻辑上能分的开，那就用service
  1. Content Provider
     1. 概述

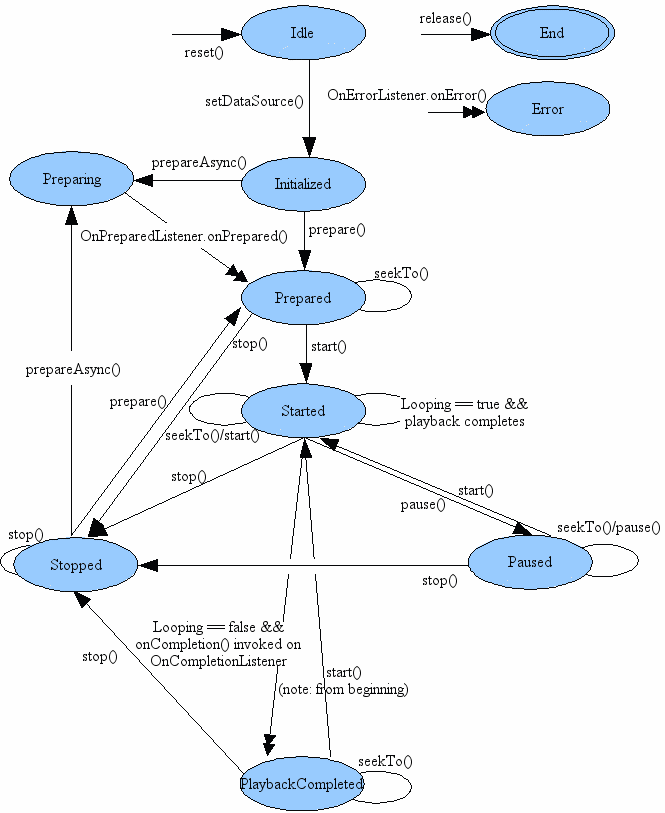
android平台提供了Content Provider使一个应用程序的指定数据集提供给其他应用程序。这些数据可以存储在文件系统中、在一个SQLite数据库、或以任何其他合理的方式,他应用可以通过ContentResolver类(见ContentProviderAccessApp例子)从该内容提供者中获取或存入数据.(相当于在应用外包了一层壳),只有需要在多个应用程序间共享数据是才需要内容提供者。例如，通讯录数据被多个应用程序使用，且必须存储在一个内容提供者中。

* + 1. 使用Content Provider对外共享数据的步骤
* 继承ContentProvider类并根据需求重写以下方法：
  + query()：从您的提供程序检索数据。使用参数选择要查询的表、要返回的行和列以及结果的排序顺序。 将数据作为 Cursor 对象返回
  + insert()：在您的提供程序中插入一个新行。使用参数选择目标表并获取要使用的列值。 返回新插入行的内容 URI。
  + update()：更新您提供程序中的现有行。使用参数选择要更新的表和行，并获取更新后的列值。 返回已更新的行数。
  + delete()：从您的提供程序中删除行。使用参数选择要删除的表和行。 返回已删除的行数。
  + getType()：返回内容 URI 对应的 MIME 类型。实现内容提供程序 MIME 类型部分对此方法做了更详尽的描述。
  + onCreate()：初始化您的提供程序。Android 系统会在创建您的提供程序后立即调用此方法。 请注意，ContentResolver 对象尝试访问您的提供程序时，系统才会创建它。
* 这些方法中的Uri参数,得到后需要进行解析然后做对应处理,Uri表示要操作的数据,包含两部分信息:➀ 需要操作的contentprovider；➁对contentprovider中的什么数据进行操作,一个Uri格式:结构头://authorities(域名)/路径(要操作的数据,根据业务而定)
* 在AndroidManifest.xml中使用<provider>对ContentProvider进行配置注册(内容提供者注册它自己就像网站注册域名),ContentProvider采用authoritie(原意授权,可理解为域名)作为唯一标识,方便其他应用能找到

1. **一个播放器小应用**
   1. 概述

播放器主要实现支持两种方式打开，一种基于界面选择本地音频播放，另一种基于命令行方式本地播放。界面播放采用Android MediaPlayer实现播放基本接口，命令行播放采用BroadcastReceiver传输路径与控制命令。

* 1. MediaPlayer



Android下对于音频、视频的支持均需要使用到MediaPlayer，它主要用来控制Android下播放文件或流的类。MediaPlayer处于Android多媒体包下"android.media.MediaPlayer"，仅有一个无参的构造函数，虽然仅为我们提供了一个无参的构造函数，为了方便我们初始化，还为我们提供了几个静态的create()方法用于完成MediaPlayer初始化的工作。基本使用方法如下：

* 获得MediaPlayer实例：
  + 可以使用直接new的方式：

MediaPlayer mp = new MediaPlayer();

* + 也可以使用create的方式，如：

MediaPlayer mp = MediaPlayer.create(this, R.raw.test);//这时就不用调用setDataSource了

* 设置要播放的文件：

MediaPlayer要播放的文件主要包括3个来源：

* + 用户在应用中事先自带的resource资源，如

MediaPlayer.create(this, R.raw.test);

* + 存储在SD卡或其他文件路径下的媒体文件，如

mp.setDataSource("/sdcard/test.mp3");

* + 网络上的媒体文件，如

mp.setDataSource("http://www.citynorth.cn/music/confucius.mp3");

* 对播放器的主要控制方法：

Android通过控制播放器的状态的方式来控制媒体文件的播放，其中：

* + prepare()和prepareAsync() 提供了同步和异步两种方式设置播放器进入prepare状态，需要注意的是，如果MediaPlayer实例是由create方法创建的，那么第一次启动播放前不需要再调用prepare（）了，因为create方法里已经调用过了。
  + start()是真正启动文件播放的方法，pause()和stop()比较简单，起到暂停和停止播放的作用。
  + seekTo()是定位方法，可以让播放器从指定的位置开始播放，需要注意的是该方法是个异步方法，也就是说该方法返回时并不意味着定位完成，尤其是播放的网络文件，真正定位完成时会触发OnSeekComplete.onSeekComplete()，如果需要是可以调用setOnSeekCompleteListener(OnSeekCompleteListener)设置监听器来处理的。
  + release()可以释放播放器占用的资源，一旦确定不再使用播放器时应当尽早调用它释放资源。
  + reset()可以使播放器从Error状态中恢复过来，重新会到Idle状态。
* 设置播放器的监听器：

MediaPlayer提供了一些设置不同监听器的方法来更好地对播放器的工作状态进行监听，以期及时处理各种情况，

如： setOnCompletionListener(MediaPlayer.OnCompletionListener listener)、

setOnErrorListener(MediaPlayer.OnErrorListener listener)等,设置播放器时需要考虑到播放器可能出现的情况设置好监听和处理逻辑，以保持播放器的健壮性。