**平台架构图：**

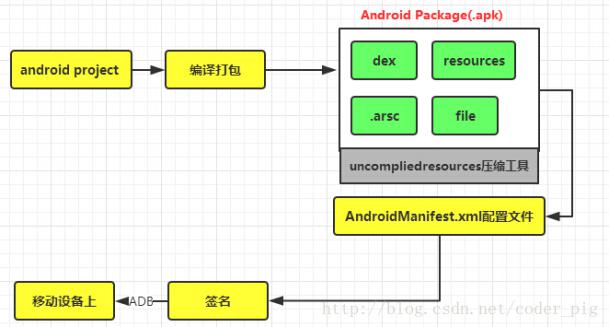


1. Dalvik： Android特有的虚拟机,和JVM不同,Dalvik虚拟机非常适合在移动终端上使用!
2. AVD： (android virtual machine):安卓虚拟设备,就是安卓的模拟器
3. ADT： (android development tools)安卓开发工具
4. SDK：(software development kit)软件开发工具包,就是安卓系统,平台架构等的工具集合,如adb.exe
5. DDMS：(dalvik debug monitor service)安卓调试工具
6. adb：安卓调试桥,在sdk的platform-tools目录下,功能很多,命令行必备
7. DX工具：将.class转换成.dex文件
8. AAPT：(android asset packing tool),安卓资源打包工具
9. R.java文件：由aapt工具根据App中的资源文件自动生成,可以理解为资源字典
10. AndroidManifest.xml：app包名 + 组件声明 + 程序兼容的最低版本 + 所需权限等程序的配置文件

**ADB命令行：**



**APP程序打包与安装的流程：**

**

**APP的安装过程：**



图片资源：

* drawable：存放各种位图文件，(.png，.jpg，.9png，.gif等)除此之外可能是一些其他的drawable类型的XML文件
* mipmap-hdpi：高分辨率，一般我们把图片丢这里
* mipmap-mdpi：中等分辨率，很少，除非兼容的的手机很旧
* mipmap-xhdpi：超高分辨率，手机屏幕材质越来越好，以后估计会慢慢往这里过渡
* mipmap-xxhdpi：超超高分辨率，这个在高端机上有所体现

MainActivity文件：

public class MainActivity extends AppCompatActivity {

@Override

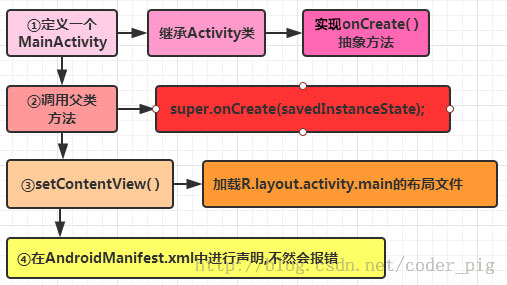
protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

}



布局文件：activity\_main.xml：

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

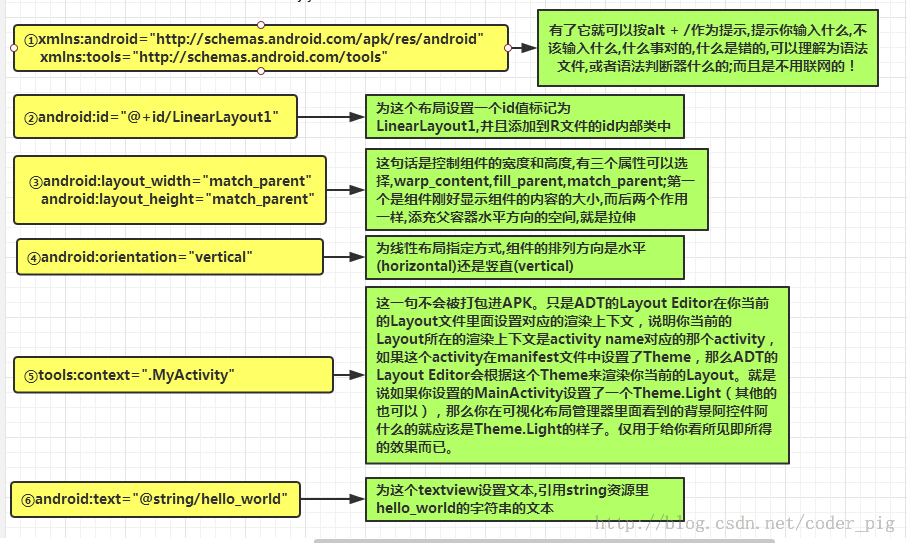
xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".MainActivity">

</RelativeLayout>



AndroidManifest.xml配置文件:

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="jay.com.example.firstapp" >

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:theme="@style/AppTheme" >

<activity

android:name=".MainActivity"

android:label="@string/app\_name" >

<intent-filter>

<action android:name="android.intent.action.MAIN" />

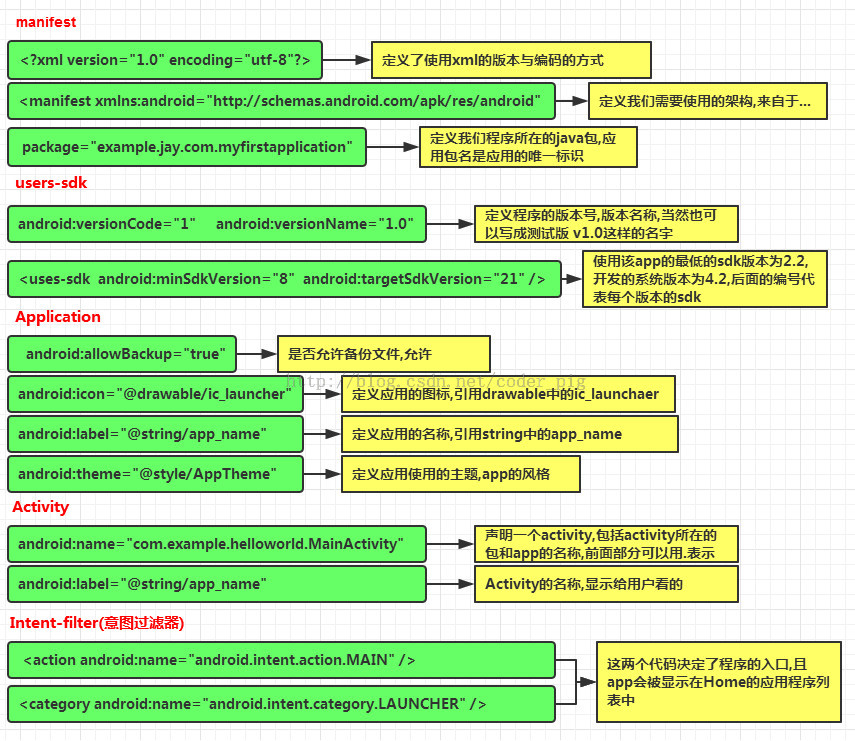
<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

</manifest>



APK签名有以下好处：

1. 应用程序升级：如果你希望用户无缝升级到新的版本，那么你必须用同一个证书进行签名。
2. 应用程序模块化：Android系统可以允许同一个证书签名的多个应用程序在一个进程里运行
3. 代码或者数据共享： Android提供了基于签名的权限机制，那么一个应用程序就可以为另一个以相同证书签名的应用程序公开自己的功能。

View与ViewGroup：



ImageView(图像视图)：

1、src属性和background属性的区别：

①background通常指的都是背景,而src指的是内容!!

②当使用src填入图片时,是按照图片大小直接填充,并不会进行拉伸

而使用background填入图片,则是会根据ImageView给定的宽度来进行拉伸

2、adjustViewBounds设置缩放是否保存原图长宽比

单独设置不起作用，需要配合maxWidth和maxHeight属性一起使用，需要adjustViewBounds为true才会生效

1. scaleType设置缩放类型

可以通过imageView.setScaleType(ImageView.ScaleType.CENTER);来设置 可选值如下：

fitXY:对图像的横向与纵向进行独立缩放,使得该图片完全适应ImageView,但是图片的横纵比可能会发生改变

fitStart:保持纵横比缩放图片,直到较长的边与Image的边相等,缩放完成后将图片放在ImageView的左上角

fitCenter:同上,缩放后放于中间;

fitEnd:同上,缩放后放于右下角;

center:保持原图的大小，显示在ImageView的中心。当原图的size大于ImageView的size，超过部分裁剪处理。

centerCrop:保持横纵比缩放图片,直到完全覆盖ImageView,可能会出现图片的显示不完全

centerInside:保持横纵比缩放图片,直到ImageView能够完全地显示图片

matrix:默认值，不改变原图的大小，从ImageView的左上角开始绘制原图，原图超过ImageView的部分作裁剪处理

ProgressBar(进度条)：

常用属性详解：

android:max：进度条的最大值

android:progress：进度条已完成进度值

android:progressDrawable：设置轨道对应的Drawable对象

android:indeterminate：如果设置成true，则进度条不精确显示进度

android:indeterminateDrawable：设置不显示进度的进度条的Drawable对象

android:indeterminateDuration：设置不精确显示进度的持续时间

android:secondaryProgress：二级进度条，类似于视频播放的一条是当前播放进度，一条是缓冲进度，前者通过progress属性进行设置！

SeekBar(拖动条)：

SeekBar基本用法：

android:max="100" //滑动条的最大值

android:progress="60" //滑动条的当前值

android:secondaryProgress="70" //二级滑动条的进度

android:thumb = "@mipmap/sb\_icon" //滑块的drawable

SeekBar.OnSeekBarChangeListener 我们只需重写三个对应的方法：

onProgressChanged：进度发生改变时会触发

onStartTrackingTouch：按住SeekBar时会触发

onStopTrackingTouch：放开SeekBar时触发

ScrollView(滚动条)：

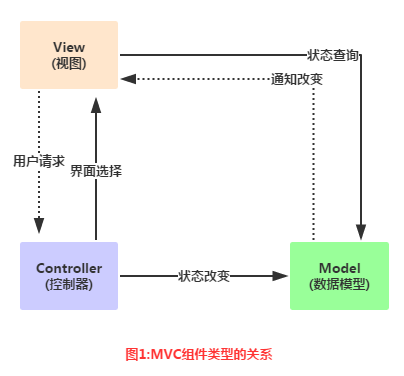
滚动到底部：

可以直接利用ScrollView给我们提供的:fullScroll()方法：

scrollView.fullScroll(ScrollView.FOCUS\_DOWN);滚动到底部

scrollView.fullScroll(ScrollView.FOCUS\_UP);滚动到顶部

MCV模式：

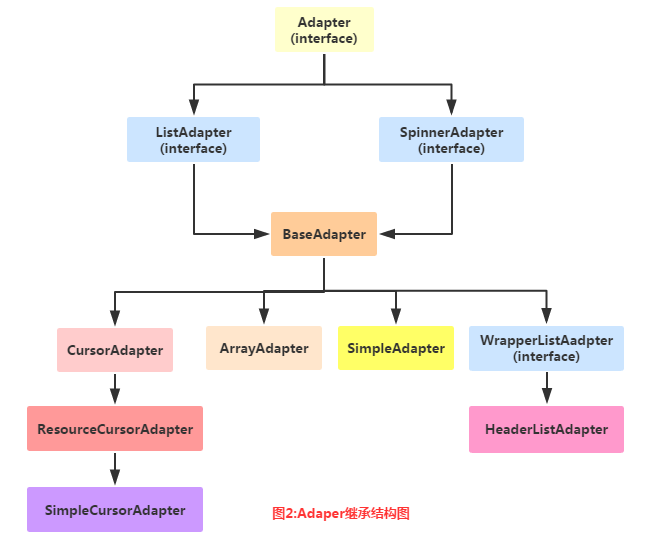


关系图解析：

Model：通常可以理解为数据,负责执行程序的核心运算与判断逻辑,,通过view获得用户 输入的数据,然后根据从数据库查询相关的信息,最后进行运算和判断,再将得到的结果交给view来显示

view:用户的操作接口,说白了就是GUI,应该使用哪种接口组件,组件间的排列位置与顺序都需要设计

Controller:控制器,作为model与view之间的枢纽,负责控制程序的执行流程以及对象之间的一个互动



BaseAdapter：抽象类，实际开发中我们会继承这个类并且重写相关方法，用得最多的一个Adapter！

ArrayAdapter：支持泛型操作，最简单的一个Adapter，只能展现一行文字~

SimpleAdapter：同样具有良好扩展性的一个Adapter，可以自定义多种效果！

SimpleCursorAdapter：用于显示简单文本类型的listView，一般在数据库那里会用到，不过有点过时， 不推荐使用！

ListView：

表头表尾分割线的设置：

listview作为一个列表控件，他和普通的列表一样，可以自己设置表头与表尾： 以及分割线，可供我们设置的属性如下：

footerDividersEnabled：是否在footerView(表尾)前绘制一个分隔条,默认为true

headerDividersEnabled:是否在headerView(表头)前绘制一个分隔条,默认为true

divider:设置分隔条,可以用颜色分割,也可以用drawable资源分割

dividerHeight:设置分隔条的高度

只能在Java中写代码 进行设置了，可供我们调用的方法如下：

addHeaderView(View v)：添加headView(表头),括号中的参数是一个View对象

addFooterView(View v)：添加footerView(表尾)，括号中的参数是一个View对象

addHeaderView(headView, null, false)：和前面的区别：设置Header是否可以被选中

addFooterView(View,view,false)：同上

Spinner(列表选项框)：

android:dropDownHorizontalOffset：设置列表框的水平偏移距离

android:dropDownVerticalOffset：设置列表框的水平竖直距离

android:dropDownSelector：列表框被选中时的背景

android:dropDownWidth：设置下拉列表框的宽度

android:gravity：设置里面组件的对其方式

android:popupBackground：设置列表框的背景

android:prompt：设置对话框模式的列表框的提示信息(标题)，只能够引用string.xml 中的资源id,而不能直接写字符串

android:spinnerMode：列表框的模式，有两个可选值： dialog：对话框风格的窗口 dropdown：下拉菜单风格的窗口(默认)

可选属性：android:entries：使用数组资源设置下拉列表框的列表项目

ViewPager：

ViewPager则有一个特定的Adapter—— PagerAdapter

FragmentPageAdapter和FragmentStatePagerAdapter区别：

FragmentPageAdapter：和PagerAdapter一样，只会缓存当前的Fragment以及左边一个，右边 一个，即总共会缓存3个Fragment而已，假如有1，2，3，4四个页面：

处于1页面：缓存1，2

处于2页面：缓存1，2，3

处于3页面：销毁1页面，缓存2，3，4

处于4页面：销毁2页面，缓存3，4

更多页面的情况，依次类推~

FragmentStatePagerAdapter：当Fragment对用户不 见得时，整个Fragment会被销毁， 只会保存Fragment的状态！而在页面需要重新显示的时候，会生成新的页面！

PagerAdapter的使用：

getCount():获得viewpager中有多少个view

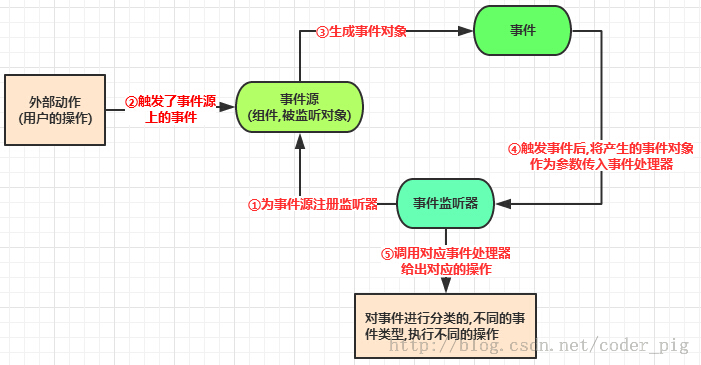
destroyItem():移除一个给定位置的页面。适配器有责任从容器中删除这个视图。 这是为了确保在finishUpdate(viewGroup)返回时视图能够被移除。

instantiateItem(): ①将给定位置的view添加到ViewGroup(容器)中,创建并显示出来 ②返回一个代表新增页面的Object(key),通常都是直接返回view本身就可以了,当然你也可以 自定义自己的key,但是key和每个view要一一对应的关系

isViewFromObject(): 判断instantiateItem(ViewGroup, int)函数所返回来的Key与一个页面视图是否是 代表的同一个视图(即它俩是否是对应的，对应的表示同一个View),通常我们直接写 return view == object!

基于监听事件的处理机制：

1. 基于监听的时间处理机制模型



事件监听机制中由事件源、事件、事件监听器三类对象组成

1:为某个事件源(组件)设置一个监听器,用于监听用户操作 2:用户的操作,触发了事件源的监听器 3:生成了对应的事件对象 4:将这个事件源对象作为参数传给事件监听器 5:事件监听器对事件对象进行判断,执行对应的事件处理器(对应事件的处理方法)

事件监听机制是一种委派式的事件处理机制,事件源(组件)事件处理委托给事件监听器 当事件源发生指定事件时,就通知指定事件监听器,执行相应的操作

基于回调的事件处理机制：

自定义view 注意:在xml布局中使用自定义的view时,需要使用"全限定类名"

常见View组件的回调方法：

android为GUI组件提供了一些事件处理的回调方法,以View为例,有以下几个方法

①在该组件上触发屏幕事件: boolean onTouchEvent(MotionEvent event);

②在该组件上按下某个按钮时: boolean onKeyDown(int keyCode,KeyEvent event);

③松开组件上的某个按钮时: boolean onKeyUp(int keyCode,KeyEvent event);

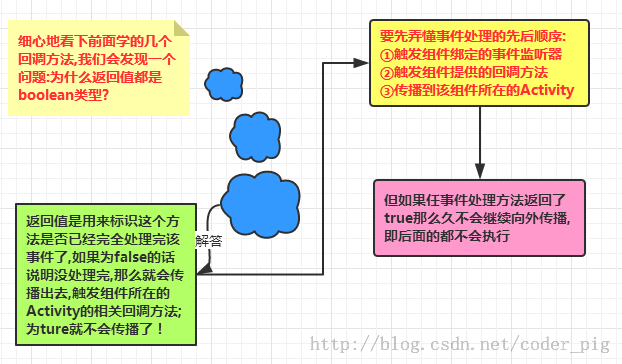
④长按组件某个按钮时: boolean onKeyLongPress(int keyCode,KeyEvent event);

⑤键盘快捷键事件发生: boolean onKeyShortcut(int keyCode,KeyEvent event);

⑥在组件上触发轨迹球屏事件: boolean onTrackballEvent(MotionEvent event);

⑦当组件的焦点发生改变,和前面的6个不同,这个方法只能够在View中重写哦！ protected void onFocusChanged(boolean gainFocus, int direction, Rect previously FocusedRect)

基于回调事件传播

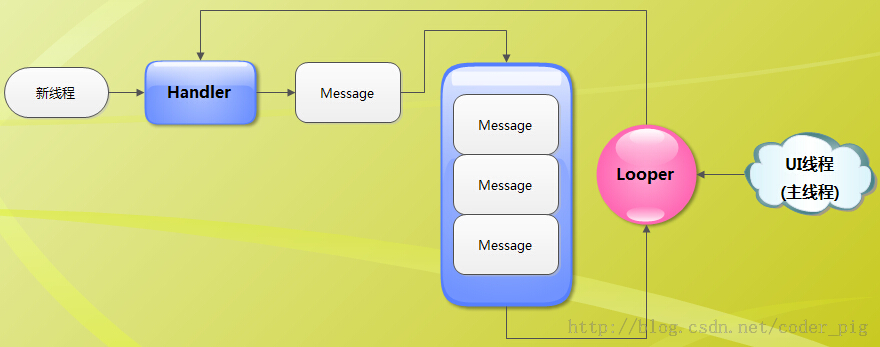


传播的顺序是: 监听器--->view组件的回调方法--->Activity的回调方法;

Handler消息传递机制：



Hanlder执行流程



UI线程:就是我们的主线程,系统在创建UI线程的时候会初始化一个Looper对象,同时也会创建一个与其关联的MessageQueue;

**Handler**:作用就是发送与处理信息,如果希望Handler正常工作,在当前线程中要有一个Looper对象

**Message**:Handler接收与处理的消息对象

**MessageQueue**:消息队列,先进先出管理Message,在初始化Looper对象时会创建一个与之关联的MessageQueue;

**Looper**:每个线程只能够有一个Looper,管理MessageQueue,不断地从中取出Message分发给对应的Handler处理！

Handler的相关方法:

void **handleMessage**(Message msg):处理消息的方法,通常是用于被重写!

**sendEmptyMessage**(int what):发送空消息

**sendEmptyMessageDelayed**(int what,long delayMillis):指定延时多少毫秒后发送空信息

**sendMessage**(Message msg):立即发送信息

**sendMessageDelayed**(Message msg):指定延时多少毫秒后发送信息

final boolean **hasMessage**(int what):检查消息队列中是否包含what属性为指定值的消息 如果是参数为(int what,Object object):除了判断what属性,还需要判断Object属性是否为指定对象的消息

1）Handler写在主线程中

在主线程中,因为系统已经初始化了一个Looper对象,所以我们直接创建Handler对象,就可以进行信息的发送与处理了！

2）Handler写在子线程中

如果是Handler写在了子线程中的话,我们就需要自己创建一个Looper对象了!创建的流程如下:

1 )直接调用Looper.prepare()方法即可为当前线程创建Looper对象,而它的构造器会创建配套的MessageQueue; 2 )创建Handler对象,重写handleMessage( )方法就可以处理来自于其他线程的信息了! 3 )调用Looper.loop()方法启动Looper

基于监听的onTouchListener:

OnTouchListener相关方法与属性:

onTouch(View v, MotionEvent event):这里面的参数依次是触发触摸事件的组件,触碰事件event 封装了触发事件的详细信息，同样包括事件的类型、触发时间等信息。比如event.getX(),event.getY()

可以对触摸的动作类型进行判断,使用event.getAction( )再进行判断;如:

event.getAction == MotionEvent.ACTION\_DOWN：按下事件

event.getAction == MotionEvent.ACTION\_MOVE:移动事件

event.getAction == MotionEvent.ACTION\_UP:弹起事件

基于回调的onTouchEvent

AsyncTask异步任务：

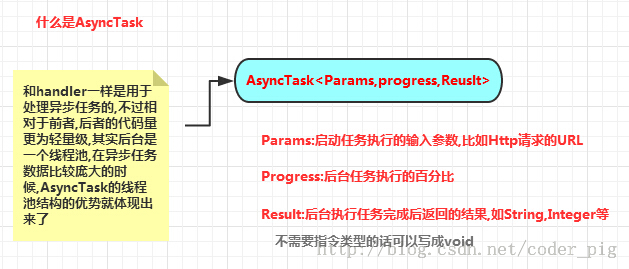
1、应用程序(Application)：为了完成特定任务，用某种语言编写的一组指令集合(一组静态代码)

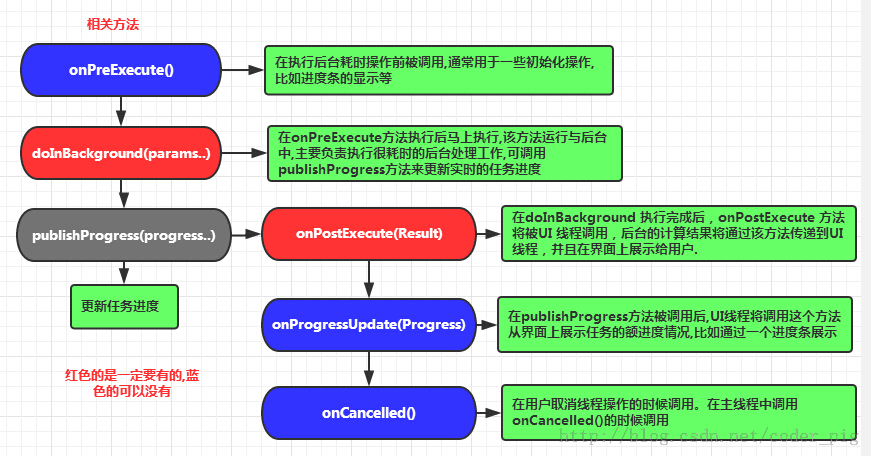
2、进程(Process) :运行中的程序，系统调度与资源分配的一个独立单位，操作系统会为每个进程分配 一段内存空间，程序的依次动态执行，经理代码加载 -> 执行 -> 执行完毕的完整过程！

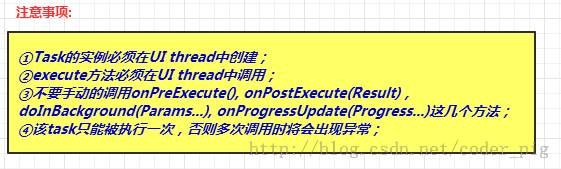
3、线程(Thread)：比进程更小的执行单元，每个进程可能有多条线程，线程需要放在一个进程中才能执行！ 线程是由程序负责管理的！！！而进程则是由系统进行调度的！！！

4、多线程概念(Multithreading)：并行地执行多条指令，将CPU的时间片按照调度算法，分配给各个线程，实际上是分时执行的，只是这个切换的时间很短，用户感觉是同时而已！

AsyncTask的基本结构：AsyncTask是一个抽象类





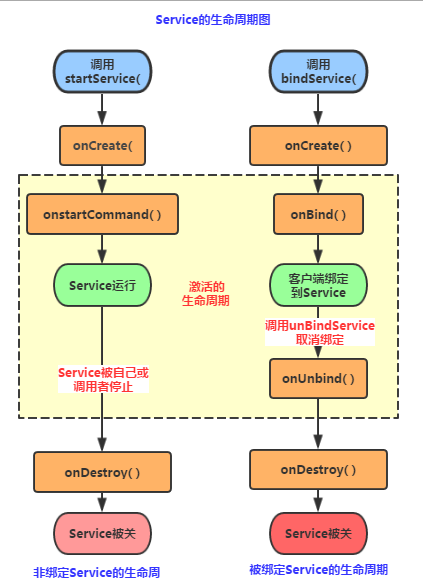


Service与Thread线程的区别：

Thread是线程，程序执行的最小单元，分配CPU的基本单位！

Service则是Android提供一个允许长时间留驻后台的一个组件，最常见的用法就是做轮询操作！或者想在后台做一些事情，比如后台下载更新！

**Service的生命周期图：**



Android中使用Service的方式有两种：

1）StartService()启动Service

2）BindService()启动Service

PS:还有一种，就是启动Service后，绑定Service！

**StartService启动Service**

①首次启动会创建一个Service实例,依次调用onCreate()和onStartCommand()方法,此时Service 进入运行状态,如果再次调用StartService启动Service,将不会再创建新的Service对象, 系统会直接复用前面创建的Service对象,调用它的onStartCommand()方法！

②但这样的Service与它的调用者无必然的联系,就是说当调用者结束了自己的生命周期, 但是只要不调用stopService,那么Service还是会继续运行的!

③无论启动了多少次Service,只需调用一次StopService即可停掉Service

**BindService启动Service**

①当首次使用bindService绑定一个Service时,系统会实例化一个Service实例,并调用其onCreate()和onBind()方法,然后调用者就可以通过IBinder和Service进行交互了,此后如果再次使用bindService绑定Service,系统不会创建新的Sevice实例,也不会再调用onBind()方法,只会直接把IBinder对象传递给其他后来增加的客户端!

②如果我们解除与服务的绑定,只需调用unbindService(),此时onUnbind和onDestory方法将会被调用!这是一个客户端的情况,假如是多个客户端绑定同一个Service的话,情况如下 当一个客户完成和service之间的互动后，它调用 unbindService() 方法来解除绑定。当所有的客户端都和service解除绑定后，系统会销毁service。（除非service也被startService()方法开启）

③bindService模式下的Service是与调用者相互关联的,在bindService后,一旦调用者销毁,那么Service也立即终止!

通过BindService调用Service时调用的Context的bindService的解析 bindService(Intent Service,ServiceConnection conn,int flags)

service:通过该intent指定要启动的Service

conn:ServiceConnection对象,用户监听访问者与Service间的连接情况, 连接成功回调该对象中的onServiceConnected(ComponentName,IBinder)方法; 如果Service所在的宿主由于异常终止或者其他原因终止,导致Service与访问者间断开 连接时调用onServiceDisconnected(CompanentName)方法,主动通过unBindService()方法断开并不会调用上述方法!

flags:指定绑定时是否自动创建Service(如果Service还未创建), 参数可以是0(不自动创建),BIND\_AUTO\_CREATE(自动创建)

java.lang.IllegalArgumentException: Service Intent must be explicit: Intent

原因是Android 5.0对Service Intent的调用策略发生改变了，必须通过显示Intent来启动Service

解决办法如下：

1、通过显示意图启动Service(直接用类名)；

Intent intent = new Intent(com.yulore.test.AppService.class);

context.startService(intent);

2、如果想继续使用隐式意图的话，加上包名信息即可；

Intent intent = new Intent();

intent.setAction("com.yulore.recognize.android");

intent.setPackage(context.getPackageName()); //兼容Android 5.0

context.startService(intent);

ServcieConnection对象监听访问者和Service间的链接情况，

1）连接成功：回调onServiceConnected()

2）异常终止或其他原因断开连接：回调onServiceDisconnected()方法

onServiceDisconnected方法中有一个IBinder对象，自定义Service类实现IBinder onBinder()方法，返回值回传给onServiceDisconnected，实现Service和IBinder通信

验证BinderService启动Service：

1. 自定义Service中继承Binder
2. 通过onBind()方法返回自定义的Binder对象
3. 在绑定Service类中定义ServiceConnection对象，重写两个方法

IntentService:

IntentService是继承于Service并处理异步请求的一个类,在IntentService中有一个工作线程来处理耗时操作,请求的Intent记录会加入队列

Activity与Service通信:

基本流程如下：

1.自定义Service中，自定义一个Binder类，然后将需要暴露的方法都写到该类中！

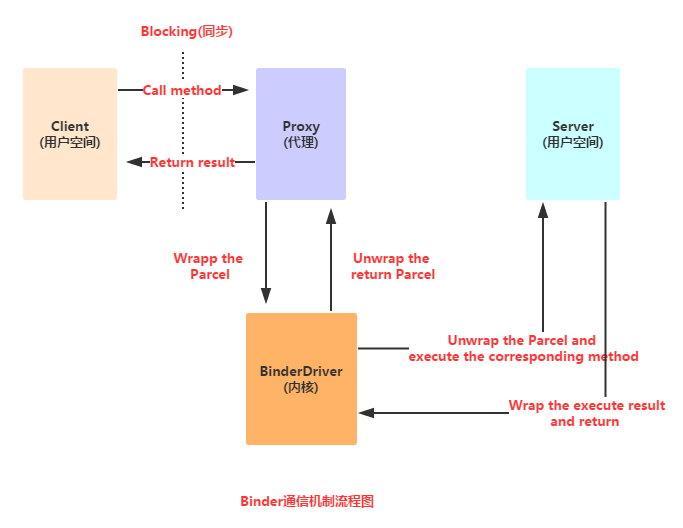
2.Service类中，实例化这个自定义Binder类，然后重写onBind()方法，将这个Binder对象返回！

3.Activity类中实例化一个ServiceConnection对象，重写onServiceConnected()方法，然后 获取Binder对象，然后调用相关方法

Binder机制:

IBinder是Android给我们提供的一个进程间通信的一个接口，而我们一般是不直接实现这个接口的， 而是通过继承Binder类来实现进程间通信！是Android中实现IPC(进程间通信)的一种方式！

Android中的Binder机制由一系列系统组件构成： Client、Server、Service Manager和Binder驱动程序



-> Client调用某个代理接口中的方法时，代理接口的方法会将Client传递的参数打包成Parcel对象；

-> 然后代理接口把该Parcel对象发送给内核中的Binder driver；

-> 然后Server会读取Binder Driver中的请求数据，假如是发送给自己的，解包Parcel对象，处理并将结果返回；

PS:代理接口中的定义的方法和Server中定义的方法是一一对应的，另外，整个调用过程是一个同步的，即Server在处理时，Client会被Block(锁)住! 而这里说的代理接口的定义就是等下要说的AIDL(Android接口描述语言)！

AIDL使用

AIDL注意事项：

接口名词需要与aidl文件名相同

接口和方法前面不要加访问权限修饰符：public ,private,protected等，也不能用static final!

AIDL默认支持的类型包括Java基本类型，String，List，Map，CharSequence，除此之外的其他类型都 需要import声明，对于使用自定义类型作为参数或者返回值，自定义类型需要实现Parcelable接口， 详情请看后面的传递复杂数据类型

自定义类型和AIDL生成的其它接口类型在aidl描述文件中，应该显式import，即便在该类和定义 的包在同一个包中

 Intent的基本使用：

四大组件间的 枢纽——Intent(意图)，Android通信的桥梁

startActivity(Intent)/startActivityForResult(Intent)：来启动一个Activity

startService(Intent)/bindService(Intent)：来启动一个Service

sendBroadcast：发送广播到指定BroadcastReceiver

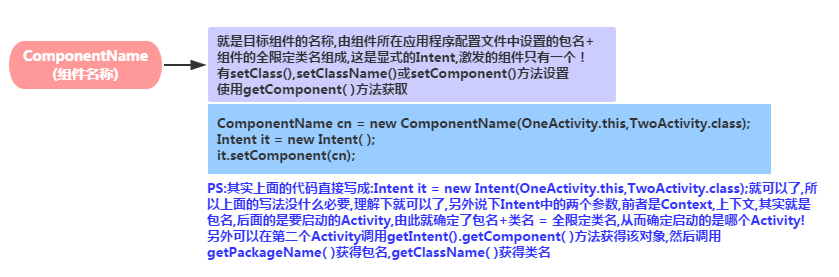
显式Intent与隐式Intent的区别：

1、显式Intent：通过组件名指定启动的目标组件,比如startActivity(new Intent(A.this,B.class)); 每次启动的组件只有一个~

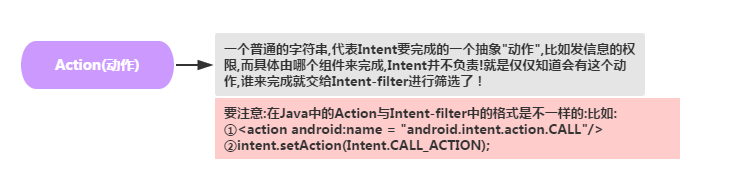
2、隐式Intent:不指定组件名,而指定Intent的Action,Data,或Category,当我们启动组件时, 会去匹配AndroidManifest.xml相关组件的Intent-filter,逐一匹配出满足属性的组件,当不止一个满足时, 会弹出一个让我们选择启动哪个的对话框~

Intent七个属性：

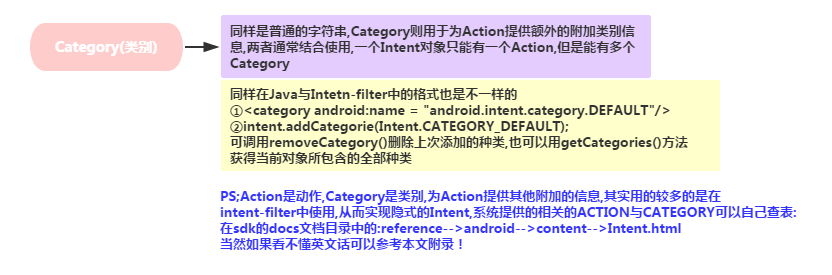
1. ComponentName(组件名称)



1. Action(动作)



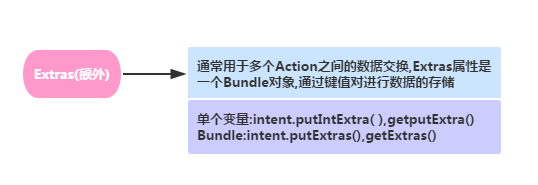
1. Category(类别)



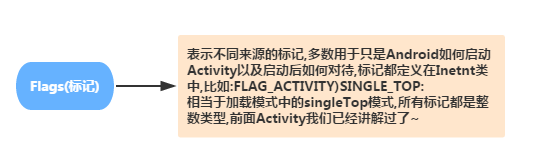
1. Data,Type



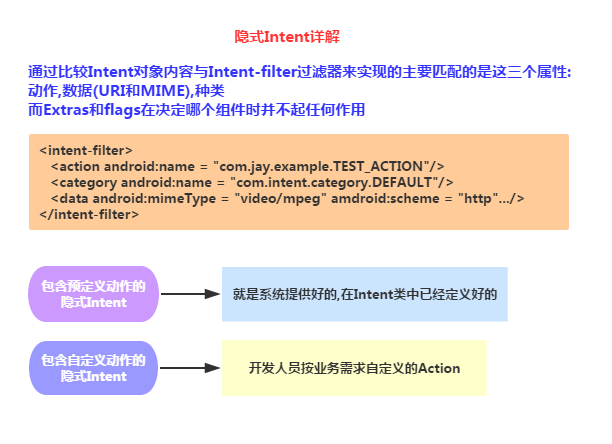
1. Extras(额外)



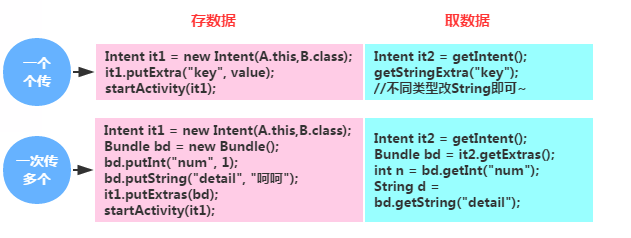
1. Flags(标识)



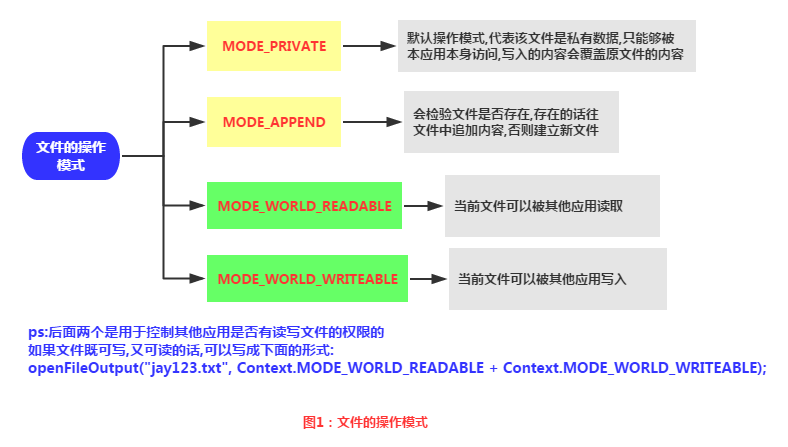
隐式Intent详解：



Intent传递简单数据：



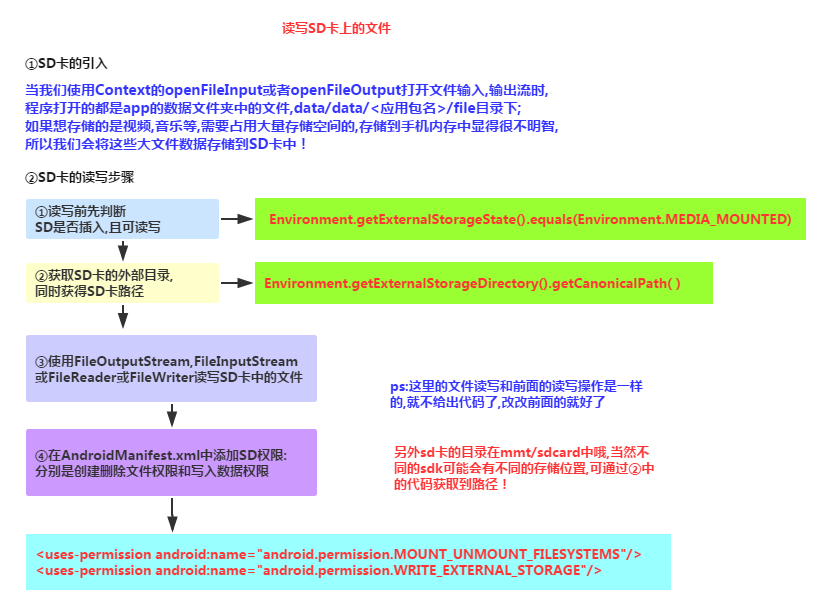
文件存储读写：  
1、安卓文件的操作模式



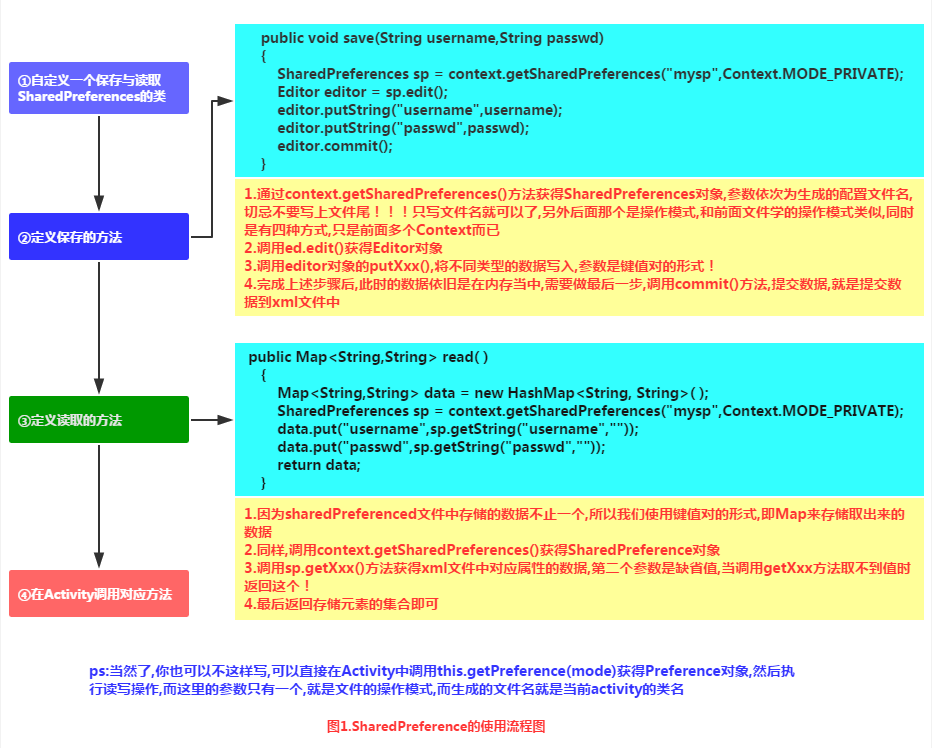
1. 文件的相关操作



1. 读取SD卡上的文件



SharedPreferences保存用户偏好参数：



SQLite：

SQLite是一个轻量级的关系型数据库，运算速度快，占用资源少，很适合在移动设备上使用， 不仅支持标准SQL语法，还遵循ACID(数据库事务)原则，无需账号，使用起来非常方便！

SQlite通过文件来保存数据库，一个文件就是一个数据库，数据库中又包含多个表格，表格里又有 多条记录，每个记录由多个字段构成，每个字段有对应的值，每个值我们可以指定类型，也可以不指定 类型(主键除外)

1. SQLiteOpenHelper：抽象类，能获取数据库实例或关闭数据库

Step 1：自定义一个类继承SQLiteOpenHelper类

Step 2：在该类的构造方法的super中设置好要创建的数据库名,版本号

Step 3：重写onCreate( )方法创建表结构

Step 4：重写onUpgrade( )方法定义版本号发生改变后执行的操作

1. SQLiteDatabase：数据库访问类，做一些增删改查操作
2. Cursor：游标，指向数据库中某一记录的指针

SQL语句操作数据库：

SQLiteDatabase相关方法：

execSQL(SQL, Object[]) 使用带占位符的SQL语句，执行修改数据库内容

rawQuery(SQL, Object[]) 使用带占位符的SQL查询语句

move(offset) 指定向上或向下移动的行数，正数向下，负数向上

moveToFirst() 指针移动到第一行，成功返回true

moveToLast() 指针移动到最后一行，成功返回true

moveToNext() 指针移动到下一行，成功返回true

moveToPrevious() 移动到上一条记录

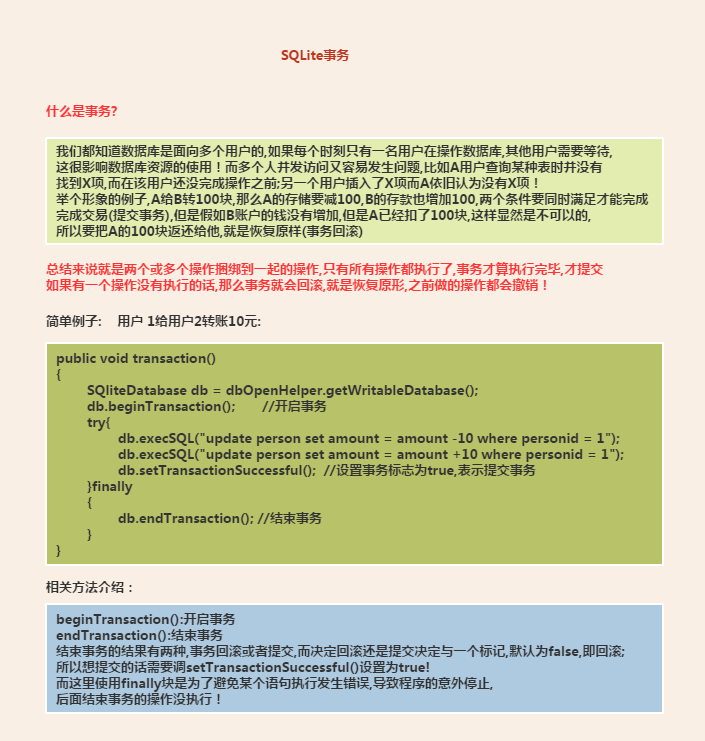
getCount() 获得总数据条数

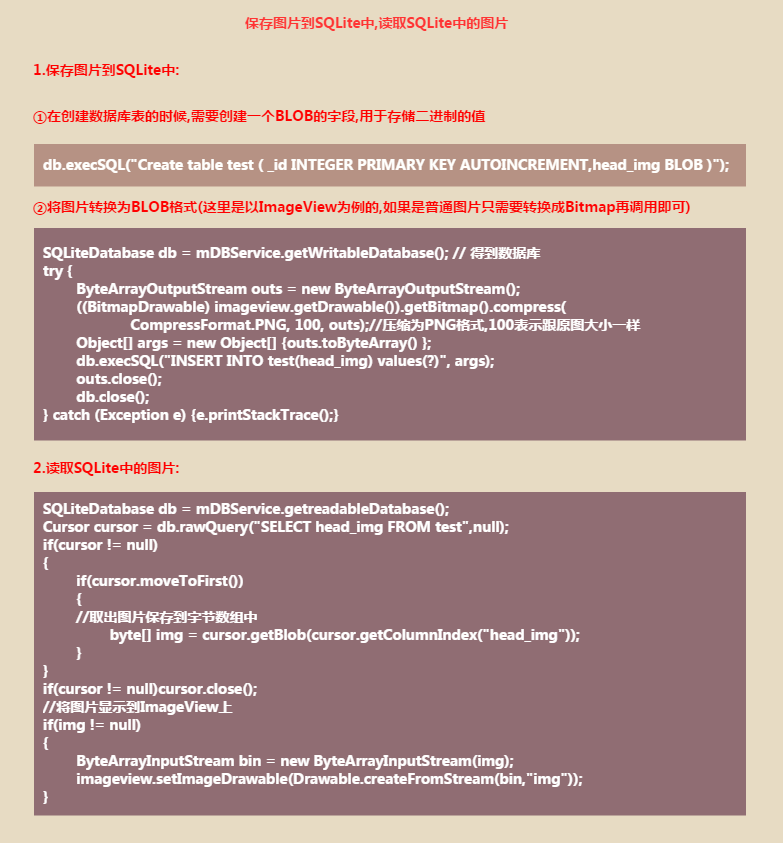
isFirst() 是否为第一条记录

isLast() 是否为最后一项

moveToPosition(int) 移动到指定行

SQLite事务





SoundPool播放音效：

SoundPool(int maxStreams, int streamType, int srcQuality) 参数依次是：

①指定支持多少个声音，SoundPool对象中允许同时存在的最大流的数量。

②指定声音类型，流类型可以分为STREAM\_VOICE\_CALL, STREAM\_SYSTEM, STREAM\_RING,STREAM\_MUSIC 和 STREAM\_ALARM四种类型。在AudioManager中定义。

③指定声音品质（采样率变换质量），一般直接设置为0！

API 21(Android 5.0)后用SoundPool.Builder，我们要实例化SoundPool只需调用：

SoundPool.Builder spb = new SoundPool.Builder();

spb.setMaxStreams(10);

spb.setAudioAttributes(null); //转换音频格式

SoundPool sp = spb.build(); //创建SoundPool对象

①加载声音资源：

load(Context context, int resId, int priority)

load(String path, int priority)

load(FileDescriptor fd, long offset, long length, int priority)

load(AssetFileDescriptor afd, int priority)

上述方法都会返回一个声音的ID，后面我们可以通过这个ID来播放指定的声音

context：上下文

resId：资源id

priority：建议设置为1，保持和未来的兼容性

path：文件路径

FileDescriptor：

AssetFileDescriptor：从asset目录读取某个资源文件，用法： AssetFileDescriptor descriptor = assetManager.openFd("biaobiao.mp3")；

②播放控制：

play(int soundID, float leftVolume, float rightVolume, int priority, int loop, float rate)

soundID：Load()返回的声音ID号

leftVolume：左声道音量设置

rightVolume：右声道音量设置

priority：指定播放声音的优先级，数值越高，优先级越大。

loop：指定是否循环：-1表示无限循环，0表示不循环，其他值表示要重复播放的次数

rate：指定播放速率：1.0的播放率可以使声音按照其原始频率，而2.0的播放速率，可以使声音按照其 原始频率的两倍播放。如果为0.5的播放率，则播放速率是原始频率的一半。播放速率的取值范围是0.5至2.0。

③资源释放：

可以调用release()方法释放所有SoundPool对象占据的内存和资源，当然也可以根据声音 ID来释放！

MediaPlayer：

1. 获得MediaPlayer实例，直接new或create方法创建

MediaPlayer mp = new MediaPlayer();

MediaPlayer mp = MediaPlayer.create(this, R.raw.test); //无需再调用setDataSource

1. 设置播放文件

//①raw下的资源：

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

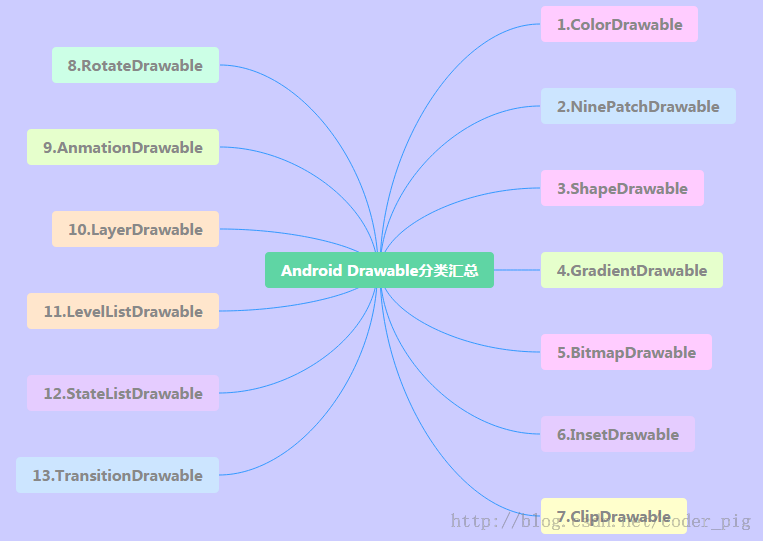
//②本地文件路径：

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

//③网络URL文件：

mp.setDataSource("http://www.xxx.com/music/test.mp3");

Drawable：



Drawable分为两种： 一种是我们普通的图片资源Android Studio中一般放到res/mipmap目录下，另一种是XML形式的Drawable资源，一般放到res/drawable目录下

Paint(画笔):

Paint paint = new Paint( );

setARGB(int a,int r,int g,int b): 设置绘制的颜色，a代表透明度，r，g，b代表颜色值。

setAlpha(int a): 设置绘制图形的透明度。

setColor(int color): 设置绘制的颜色，使用颜色值来表示，该颜色值包括透明度和RGB颜色。

setShadowLayer(float radius ,float dx,float dy,int color)：在图形下面设置阴影层，产生阴影效果， radius为阴影的角度，dx和dy为阴影在x轴和y轴上的距离，color为阴影的颜色

Canvas(画布):

clipRect(new Rect())

translate(float dx, float dy)： 平移，将画布的坐标原点向左右方向移动x，向上下方向移动y.canvas的默认位置是在（0,0）

scale(float sx, float sy)：扩大，x为水平方向的放大倍数，y为竖直方向的放大倍数

rotate(float degrees)：旋转，angle指旋转的角度，顺时针旋转

Path(路径)：

transform(Matrix matrix)：应用矩阵变换

transform(Matrix matrix, Path dst)：应用矩阵变换并将结果放到新的路径中，即第二个参数。

moveTo(float x, float y)：不会进行绘制，只用于移动移动画笔

lineTo(float x, float y)：用于直线绘制，默认从(0，0)开始绘制，用moveTo移动！ 比如 mPath.lineTo(300, 300); canvas.drawPath(mPath, mPaint);

MaskFilter：

BlurMaskFilter(模糊效果)：

BlurMaskFilter(10f,BlurMaskFilter.Blur.NORMAL);

第一个参数：指定模糊边缘的半径；

第二个参数：指定模糊的风格，可选值有：

BlurMaskFilter.Blur.NORMAL：内外模糊

BlurMaskFilter.Blur.OUTER：外部模糊

BlurMaskFilter.Blur.INNER：内部模糊

BlurMaskFilter.Blur.SOLID：内部加粗，外部模糊

补间动画：

AlphaAnimation:透明度效果，对应<alpha/>

fromAlpha :起始透明度

toAlpha:结束透明度

透明度的范围为：0-1，完全透明-完全不透明

ScaleAnimation：缩放效果，对应<scale/>

fromXScale/fromYScale：沿着X轴/Y轴缩放的起始比例

toXScale/toYScale：沿着X轴/Y轴缩放的结束比例

pivotX/pivotY：缩放的中轴点X/Y坐标，即距离自身左边缘的位置，比如50%就是以图像的 中心为中轴点

TranslateAnimation：位移渐变效果，对应<translate/>

fromXDelta/fromYDelta：动画起始位置的X/Y坐标

toXDelta/toYDelta：动画结束位置的X/Y坐标

RotateAnimation：旋转渐变效果，对应<rotate/>

fromDegrees/toDegrees：旋转的起始/结束角度

repeatCount：旋转的次数，默认值为0，代表一次，值为-1或者infinite时，表示动画永不停止

repeatMode：设置重复模式，默认restart，但只有当repeatCount大于0或者infinite或-1时 才有效.还可以设置成reverse，表示偶数次显示动画时会做方向相反的运动！

AnimationSet：组合渐变效果，对应<set/>

Interpolator接口:

用来控制动画的变化速度

写动画xml文件时会用到，属性是：android:interpolator， 而上面对应的值是：@android:anim/linear\_interpolator