**我的Android心得**

本文主要讲述几年来开发Android的心得体会，

对很多基础组件、工具等的理解。

Activity的生命周期 4

Fragment 6

启动activity的几种方法 11

自定义控件 13

动画 17

Android API及优化技术 22

多线程 22

socket通信原理 22

数据结构和算法 22

网络异步应用开发模式 22

MAVEN 22

Android Studio获取SHA1 22

触控 23

桌面 23

文件存储数据及读取 23

相机（照片+拍照） 23

感应器（重力+光感） 23

网页开发（适配） 23

proguard 24

ADB操作命令 25

类的加载过程 26

数据存储区域（内存分析） 26

JNI 26

为ViewPager添加点击事件 26

怎样做出扩展性好的插件 26

android5.0源码分析 26

设计模式 27

框架选择（比较） 27

2D/3D（小游戏开发+游戏引擎） 27

将图片做成字体加载 27

客户端与服务器的RESTful接口 27

在客户端上加广告（如何产生利润） 27

Android Studio使用技巧 28

Android开发文档阅读笔记 28

Style的使用（自定义控件的不同style显示） 28

Android测试 28

源代码管理（git以及CVS） 28

适配横竖屏 28

数据安全 29

国际化 29

# Android系统架构

Android是一个移动设备的开发平台，其软件层次结构包括操作系统、中间件和应用程序。根据android的软件框图，其软件层次结构自下而上分为以下4层。

操作系统层（OS）

各种库（Libraryies）和android运行环境（Runtime）

应用程序框架（Application Framework）

应用程序（Application）



1. 操作系统层（OS）——最底层

因为Android源于[Linux](http://www.2cto.com/os/linux/)，使用了[Linux](http://www.2cto.com/os/linux/)内核，所以Android使用Linux 2.6作为操作系统。Linux 2.6是一种标准的技术，Linux也是一个开放的操作系统。Android对操作系统的使用包括核心和驱动程序两部分，Android的Linux核心为标准的Linux 2.6内核，Android更多的是需要一些与移动设备相关的驱动程序。主要的驱动如下所示。

    显示驱动（Display Driver）：常用基于Linux的帧缓冲（Frame Buffer）驱动。

    Flash内存驱动（Flash Memory Driver）：是基于MTD的Flash驱动程序。

    相机驱动（Camera Driver）：常用基于Linux的v4l驱动。

    音频驱动（Audio Driver）：常用基于ALSA（Advanced Linux Sound Architecture，高级Linux声音体系）的驱动。

    WiFi驱动（Camera Driver）：基于IEEE 802.11标准的驱动程序。

    键盘驱动（KeyBoard Driver）：作为输入设备的键盘驱动。

    蓝牙驱动（Bluetooth Driver）：基于IEEE 802.15.1标准的无线传输技术。

    Binder IPC驱动：Android中一个特殊的驱动程序，具有单独的设备节点，提供进程间通信的功能。

    Power Management（能源管理）：管理电池电量等信息。

2. 各种库（Libraries）和Android 运行环境（RunTime）——中间层

本层次对应一般嵌入式系统，相当于中间件层次。Android的本层次分成两个部分，一个是各种库，另一个是Android 运行环境。本层的内容大多是使用C和C++实现的。其中包含的各种库如下所示。

    C库：C语言的标准库，也是系统中一个最为底层的库，C库是通过Linux的系统调用来实现的。

    多媒体框架（Media Frameword）：这部分内容是Android多媒体的核心部分，基于PacketVideo（即PV）的OpenCORE。从功能上本库一共分为两部分，一部分是音频、视频的回放（PlayBack），另一部分则是音视频的记录（Recorder）。

    SGL：2D图像引擎。

    SSL：即Secure Socket Layer，位于TCP/IP协议与各种应用层协议之间，为数据通信提供安全支持。

    OpenGL ES 1.0：提供了对3D的支持。

    界面管理工具（Surface Management）：提供了管理显示子系统等功能。

    SQLite：一个通用的嵌入式数据库。

    WebKit：网络[浏览器](http://www.2cto.com/os/liulanqi/)的核心。

    FreeType：位图和矢量字体的功能。

Android的各种库一般是以系统中间件的形式提供的，它们都有的一个显著特点是与移动设备的平台的应用密切相关。

Android运行环境主要指的是虚拟机技术——Dalvik。Dalvik虚拟机和一般Java虚拟机（Java VM）不同，它执行的不是Java标准的字节码（Bytecode），而是Dalvik可执行格式（.dex）中的执行文件。在执行的过程中，每一个应用程序即一个进程（Linux的一个Process）。二者最大的区别在于，Java VM是基于栈的虚拟机（Stack-based），而Dalvik是基于寄存器的虚拟机（Register-based）。显然，后者最大的好处在于可以根据硬件实现更大的优化，这更适合移动设备的特点。

3. 应用程序（Application）

Android的应用程序主要是用户界面（User Interface）方面的，通常用Java语言编写，其中还可以包含各种资源文件（放置在res目录中）、Java程序及相关资源经过编译后，将生成一个APK包。Android本身提供了主屏幕（Home）、联系人（Contact）、电话（Phone）以及[浏览器](http://www.2cto.com/os/liulanqi/)（Brower）等众多的核心应用。同时应用程序的开发者还可以使用应用程序框架层的API实现自己的程序。这也是Android开源的巨大潜力的体现。

4. 应用程序框架（Application Framework）

Android的应用程序框架为应用程序层的开发者提供了API，它实际上是一个应用程序的框架。由于上层的应用程序是以Java构建的，因此本层次提供的首先包含了UI程序中所需要的各种控件，例如：Views（视图组件），其中又包括了List（列表）、Grid（栅格）、Text Box（文本框）、Button（按钮）等，甚至一个嵌入式的Web浏览器。

一个基本的Android应用程序可以利用应用程序框架中的以下5个部分。

    Activity（活动）

    Broadcast Intent Receiver（广播意图接收者）

    Service（服务）

    Content Provider（内容提供者）

    Intent and Intent Filter（意图和意图过滤器）

本书的目的是讲解Android多媒体应用的开发知识，讲解的是应用程序（Application）方面的知识。这些知识都是用Java开发的，当然也还需要掌握一些其他层的相关知识，例如底层的内核和驱动等知识。

# Activity的生命周期

学习Android最开始一定是理解activity的生命周期，下图不知看过多少遍了，谈谈我对它的理解。

什么是activity？activity就是一个容器，一切创建实例、初始化、销毁实例等过程都是容器来调用的。

1. 启动activity：onCreate() 🡪 onStart() 🡪 onResume(), activity进入运行状态。
2. 当前activity被其他activity覆盖或被锁屏：onPause(),暂停当前activity的执行。
3. activity由覆盖状态回到前台或解锁屏： onResume()，再次进入运行状态。
4. 当前activity转到新activity界面或按home键回到主屏：onPause() 🡪 onStop(),进入停滞状态。
5. 后退回到此activity：onRestart() 🡪 onStart() 🡪 onResume()，进入运行状态。
6. 当activity处于被覆盖或后台不可见状态，系统内存不足杀死当前activity，而后用户退回当前activity：onCreate() 🡪 onStart() 🡪 onResume(),进入运行状态。
7. 用户退出当前activity：onPause() 🡪 onStop() 🡪 onDestroy(),结束当前activity。

下面看一下具体哪些操作应该放入这几个方法里：

1. onCreate():这里应该做所有静态的初始化，如create views, bind data to lists, etc. 同时它也提供了一个保存之前冻结状态的Bundle。
2. onPause()：当用户要离开此activity时，保存任何的改变、停止动画等。此方法里不能执行耗时任务，因为下一个activity会等待此方法的返回。
3. onWindowFocusChanged():在activity窗口获得或失去焦点时被调用，例如创建时首次呈现在用户面前；当前activity被其他activity覆盖，当前activity转到其他activity或按home键回到主屏；用户退出当前activity。几种情况会调用。
4. onSaveInstanceState():

（1）、在activity被覆盖或退居后台之后，系统资源不足将其杀死；在用户改变屏幕方向时；在当前activity跳转到其他activity或按home键回到主屏；此方法会被调用；

（2）调用此方法可以保存一些临时数据和当前窗口各个View的状态。使用putXXX()方法将数据存储在Bundle里。

（3）它的调用顺序在onPause()之前。

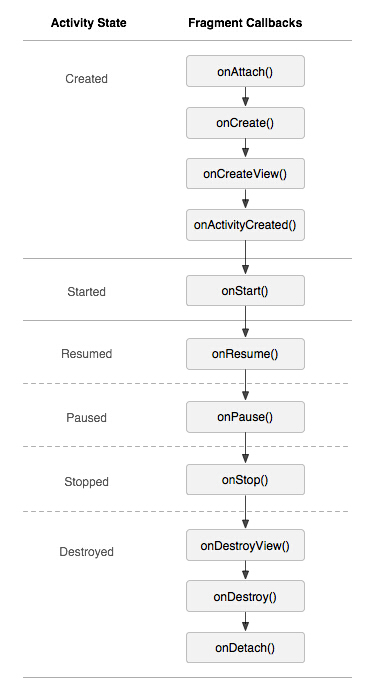
（4）注意！报错永久数据最好在onPause()方法中而不是在此方法里，因为此方法不在activity的生命中期里，有时不会被调用。

1. onRestoreInstanceState():
2. 在activity被覆盖或退居后台之后，系统资源不足将其杀死后用户又回到此activity；在改变屏幕方向时，重建过程中，此方法会被调用。
3. 可以在此方法中恢复一些临时数据，使用getXXX()方法获得状态参数。

# Fragment

想必大家都用过fragment了，自然就要对比一下fragment的生命周期和activity的生命周期的不同与关联。

Fragment必须依存于Activity，因此activity的生命周期会直接影响到fragment的生命周期。



可以看到Fragment比Activity多了几个额外的生命周期回调函数：

onAttach(Activity); **//当Activity与Fragment发生关联时调用**

onCreateView(LayoutInflater,ViewGroup,Bundle); **//创建该Fragment的视图**

onActivityCreate(bundle); **//当Activity的onCreate()；方法返回时调用**

onDestoryView();**//与onCreateView相对应，当改Fragment被移除时调用**

onDetach();　　**//与onAttach()相对应，当Fragment与Activity的关联被取消时调用**

注意：除了onCreateView，其他的所有方法如果你重写了，必须调用父类对于该方法的实现。

1. 静态调用Fragment

在activity的布局文件中，把fragment当成普通控件调用。

1. 继承Fragment，重写onCreateView决定Fragment布局。

View view = inflater.inflate(R.layout.title\_fragment, container, false);

1. 在activity中声明此Fragment，就和普通的View一样。
2. 动态调用Fragment
3. 通常都会在activity的xml布局文件中声明一个FrameLayout
4. 在Activity代码中，

FragmentManager manager = getFragmentManager();

// 开启事务

FragmentTransaction transaction = manager.beginTransaction();

mWeiXinFragment = new ContentFragment();

transaction.replace(R.id.id\_content, mWeiXinFragment);

transaction.commit();

1. Fragment的管理

需要使用FragmentManager，通过getFragmentManager()或getSupportFragmentManager()获得，常用方法有：

manager.findFragmentById();//主要用在静态添加fragment的布局中

manager.findFragmentByTag();//用在动态添加的fragment中

manager.getFragments();

FragmentTransaction，一般有几个方法：

//将一个fragment实例添加到Activity的最上层

add(**int** containerViewId, Fragment fragment, String tag);

//将一个fragment实例从Activity的fragment队列中删除

remove(Fragment fragment);

//替换containerViewId中的fragment实例，注意，它首先把containerViewId中所有fragment删除，然后再add进去当前的fragment

replace(**int** containerViewId, Fragment fragment);

还有hide()、show()、detach()、attach()这些方法。

1. FragmentTransaction事务的回滚使用：

如果想将上一次commit的操作返回就需要回滚功能。

在transaction.commit()之前，使用addToBackStack()将其添加到回退栈中。

transaction.addToBackStack(String tag);

在需要回退时，使用popBackStack()将最上层的操作弹出回退栈。

manager.popBackStack();

当栈中有多层时，可以根据id或TAG标识来指定弹出到操作所在层。

void popBackStack(int id, int flags);

void popBackStack(String name, int flags);

其中参数int id是当提交变更时transaction.commit()的返回值。

**private** **int** addFragment(Fragment fragment,String stackName){

FragmentManager manager = getSupportFragmentManager();

FragmentTransaction transaction = manager.beginTransaction();

transaction.add(R.id.fragment\_container,fragment);

transaction.addToBackStack(stackName);

**return** transaction.commit();

}

1. Fragment间参数的传递
2. 同一个container中不同fragment间的参数传递。这种情况一般发生在fragment跳转，上一个fragment将参数传递给下一个fragment。

在Fragment2中，新建一个函数newInstance(String text)来接受传过来的参数，将参数通过setArguments设置到其中：

**public** **static** Fragment2 newInstance(String text) {

    Fragment2 fragment = **new** Fragment2();

    Bundle args = **new** Bundle();

    args.putString("param", text);

    fragment.setArguments(args);

**return** fragment;

}

然后在Fragment2的OnCreateView的时候再从arguments中获取参数：

**public** View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

    View view =  inflater.inflate(R.layout.fragment2, container, **false**);

**if** (getArguments() != **null**) {

        String mParam1 = getArguments().getString("param");

        TextView tv =  (TextView)view.findViewById(R.id.textview);

        tv.setText(mParam1);

    }

**return** view;

}

1. 同一个activity中，不同container间fragment的参数传递。

在各自的fragment中可以进行view的各种操作，它们之间的交互必然是通过activity。

首先，在fragment1中定义接口及变量，接口是给activity用的，采用强制转换的方式，在fragment与activity相关联时，进行强制转换，防止用户忘记对接口变量赋值。如果activity没有实现接口就会报错：

**public** **void** onAttach(Activity activity) {

**super**.onAttach(activity);

**try** {

        mSelectInterface = (titleSelectInterface) activity;

    } **catch** (Exception e) {

**throw** **new** ClassCastException(activity.toString() + "must implement OnArticleSelectedListener");

    }

}

调用接口变量：mSelectInterface.onTitleSelect(str);

在activity中实现接口：

@Override

**public** **void** onTitleSelect(String title) {

        FragmentManager manager = getSupportFragmentManager();

        Fragment2 fragment2 = (Fragment2)manager.findFragmentById(R.id.fragment2);

        fragment2.setText(title);

}

# 启动activity的几种方法

1. 首先，讲讲activity的四种启动模式

standard、singleTop、singleTask、singleInstance

可以根据实际的需求为activity设置对应的启动模式，从而避免创建大量重复的activity等问题。

设置启动模式，只需在AndroidManifest.xml中<activity>标签设置Android:launchMode属性即可。

1. standard模式，默认模式，可不用写配置。这个模式下，都会默认创建一个新的实例。也就是可以有多个相同的实例，也允许多个相同activity叠加。
2. singleTop模式，可以有多个实例，但不允许多个相同activity叠加。如果activity在栈顶，启动相同的activity，不会创建新的实例，而会调用其onNewIntent方法。
3. singleTask模式，只有一个实例，在同一个应用程序中启动它时，若activity不存在，则会在当前task创建一个新实例，若存在，则会把task中在其之上的其他activity destroy掉并调用它的onNewIntent方法。
4. singleInstance模式，只有一个实例，并且这个实例独立运行在一个task中，这个task只有这个实例，不允许有别的activity存在。
5. activity的几种启动方法

显示启动

通过包名启动

1. ComponentName对象方式（可启动其他应用的activity）

ComponentName cn = new ComponentName(“com.example”,

“com.example.ui.TestActivity”);

Intent intent = new Intent();

intent.setComponent(cn);

startActivity(intent);

1. 初始化Intent时指定包名

Intent intent = new Intent(“android.intent.action.MAIN”);

Intent.setClassName(“com.example”, “com.example.ui.TestActivity”);

startActivity(intent);

隐式启动

通过指定action或action和data属性启动activity，如果有多个activity符合启动条件，系统会让用户选择启动哪个应用。

1. 只为被启动activity指定action属性。

Intent intent = new Intent(“com.example. action.DEMO”);

startActivity(intent);

1. 为被启动activity指定action和data属性。

Intent intent = new Intent(“com.example. action.DEMO”, Uri.parse(“info://”));

startActivity(intent);

启动自己应用的activity

前面两种不管是本应用的activity还是另外程序的activity都可以得以解决，但启动自己应用的activity就没必要麻烦了。

Intent intent = new Intent(ActivityA.this, ActivityB.class);

startActivty(intent);

有时需要在一个activity结束时返回一个结果，如在activity中用户选中联系人列表的某一项，当activity结束时返回所选的内容。

startActivtyForResult(intent, CODE);

这里的CODE会在onActivityResult(int, int, Intent);方法返回。

通过包名启动apk

Intent intent = getPackageManager().

getLaunchIntentForPachage(“com.example.activityb”);

if(intent != null) {

startActivity(intent);

}

# 自定义控件

自定义view的步骤：

1. 自定义view的属性

在attrs.xml里面定义属性和声明样式：

**<declare-styleable** name="CustomTitleView"**>**

**<attr** name="titleText" format="string" **/>**

**<attr** name="titleTextColor" format="color" **/>**

**<attr** name="titleTextSize" format="dimension" **/>**

**</declare-styleable>**

format属性的取值类型有：string、color、demension、integer、enum、reference、float、boolean、fraction、flag。

然后在布局文件中声明自定义的view：

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

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

    xmlns:custom="http://schemas.android.com/apk/res/com.example.customview01"

    android:layout\_width="match\_parent"

    android:layout\_height="match\_parent" >

    <com.example.customview01.view.CustomTitleView

        android:layout\_width="200dp"

        android:layout\_height="100dp"

        custom:titleText="3712"

        custom:titleTextColor="#ff0000"

        custom:titleTextSize="40sp" />

</RelativeLayout>

1. 在view的构造方法中获得我们自定义的属性

**public** CustomTitleView(Context context, AttributeSet attrs, **int** defStyle)

    {

**super**(context, attrs, defStyle);

        /\*\*

         \* 获得我们所定义的自定义样式属性

         \*/

        TypedArray a = context.getTheme().obtainStyledAttributes(attrs, R.styleable.CustomTitleView, defStyle, 0);

**int** n = a.getIndexCount();

**for** (**int** i = 0; i < n; i++)

        {

**int** attr = a.getIndex(i);

**switch** (attr)

            {

**case** R.styleable.CustomTitleView\_titleText:

                mTitleText = a.getString(attr);

**break**;

**case** R.styleable.CustomTitleView\_titleTextColor:

                // 默认颜色设置为黑色

                mTitleTextColor = a.getColor(attr, Color.BLACK);

**break**;

**case** R.styleable.CustomTitleView\_titleTextSize:

                // 默认设置为16sp，TypeValue也可以把sp转化为px

                mTitleTextSize = a.getDimensionPixelSize(attr, (**int**) TypedValue.applyDimension(

                        TypedValue.COMPLEX\_UNIT\_SP, 16, getResources().getDisplayMetrics()));

**break**;

            }

        }

// 一定要回收，否则会内存泄漏

        a.recycle();

        /\*\*

         \* 获得绘制文本的宽和高

         \*/

        mPaint = **new** Paint();

        mPaint.setTextSize(mTitleTextSize);

        // mPaint.setColor(mTitleTextColor);

        mBound = **new** Rect();

        mPaint.getTextBounds(mTitleText, 0, mTitleText.length(), mBound);

    }

1. 重写onMessure方法（不一定必须）

重写之前先了解MessureSpec的specMode，一共有三种类型：

EXACTLY：一般是设置了明确的值或者是MATCH\_PARENT

AT\_MOST：表示子布局限制在一个最大值内，一般为WRAP\_CONTENT

UNSPECIFIED：表示子布局想要多大就多大，很少使用。

@Override

**protected** **void** onMeasure(**int** widthMeasureSpec, **int** heightMeasureSpec) {

**int** widthMode = MeasureSpec.getMode(widthMeasureSpec);

**int** widthSize = MeasureSpec.getSize(widthMeasureSpec);

**int** heightMode = MeasureSpec.getMode(heightMeasureSpec);

**int** heightSize = MeasureSpec.getSize(heightMeasureSpec);

**int** width;

**int** height ;

**if** (widthMode == MeasureSpec.EXACTLY) {

        width = widthSize;

    } **else** {

        mPaint.setTextSize(mTitleTextSize);

        mPaint.getTextBounds(mTitle, 0, mTitle.length(), mBounds);

**float** textWidth = mBounds.width();

**int** desired = (**int**) (getPaddingLeft() + textWidth + getPaddingRight());

        width = desired;

    }

**if** (heightMode == MeasureSpec.EXACTLY) {

        height = heightSize;

    } **else** {

        mPaint.setTextSize(mTitleTextSize);

        mPaint.getTextBounds(mTitle, 0, mTitle.length(), mBounds);

**float** textHeight = mBounds.height();

**int** desired = (**int**) (getPaddingTop() + textHeight + getPaddingBottom());

        height = desired;

    }

    setMeasuredDimension(width, height);

}

1. 重写onDraw方法

    @Override

**protected** **void** onDraw(Canvas canvas)

    {

        mPaint.setColor(Color.YELLOW);

        canvas.drawRect(0, 0, getMeasuredWidth(), getMeasuredHeight(), mPaint);

        mPaint.setColor(mTitleTextColor);

        canvas.drawText(mTitleText, getWidth() / 2 - mBound.width() / 2, getHeight() / 2 + mBound.height() / 2, mPaint);

    }

# 动画

Android的animation由四种类型组成：alpha、scale、translate、rotate。

1. scale标签-调节尺寸

scale标签是缩放动画，可以实现动态调控件尺寸的效果，有下面几个属性：

android:fromXScale    起始的X方向上相对自身的缩放比例，浮点值，比如1.0代表自身无变化，0.5代表起始时缩小一倍，2.0代表放大一倍；

android:toXScale        结尾的X方向上相对自身的缩放比例，浮点值；

android:fromYScale    起始的Y方向上相对自身的缩放比例，浮点值，

android:toYScale        结尾的Y方向上相对自身的缩放比例，浮点值；

android:pivotX            缩放起点X轴坐标，可以是数值、百分数、百分数p 三种样式，比如 50、50%、50%p，当为数值时，表示在当前View的左上角，即原点处加上50px，做为起始缩放点；如果是50%，表示在当前控件的左上角加上自己宽度的50%做为起始点；如果是50%p，那么就是表示在当前的左上角加上父控件宽度的50%做为起始点x轴坐标。

android:pivotY           缩放起点Y轴坐标，取值及意义跟android:pivotX一样。

从Animation类继承的属性，以scale为例

android:duration        动画持续时间，以毫秒为单位

android:fillAfter          如果设置为true，控件动画结束时，将保持动画最后时的状态

android:fillBefore       如果设置为true,控件动画结束时，还原到开始动画前的状态

android:fillEnabled    与android:fillBefore 效果相同，都是在动画结束时，将控件还原到初始化状态

android:repeatCount 重复次数

android:repeatMode 重复类型，有reverse和restart两个值，reverse表示倒序回放，restart表示重新放一遍，必须与repeatCount一起使用才能看到效果。因为这里的意义是重复的类型，即回放时的动作。

android:interpolator  设定插值器，其实就是指定的动作效果，比如弹跳效果等。

2、alpha标签-调节透明度

android:fromAlpha   动画开始的透明度，从0.0 --1.0 ，0.0表示全透明，1.0表示完全不透明

android:toAlpha       动画结束时的透明度，也是从0.0 --1.0 ，0.0表示全透明，1.0表示完全不透明

3、rotate标签-旋转

android:fromDegrees     开始旋转的角度位置，正值代表顺时针方向度数，负值代码逆时针方向度数

android:toDegrees         结束时旋转到的角度位置，正值代表顺时针方向度数，负值代码逆时针方向度数

android:pivotX               缩放起点X轴坐标，可以是数值、百分数、百分数p 三种样式，比如 50、50%、50%p，具体意义已在scale标签中讲述，这里就不再重讲

android:pivotY               缩放起点Y轴坐标，可以是数值、百分数、百分数p 三种样式，比如 50、50%、50%p

4、translate标签-平移

android:fromXDelta     起始点X轴坐标，可以是数值、百分数、百分数p 三种样式，比如 50、50%、50%p，具体意义已在scale标签中讲述，这里就不再重讲

android:fromYDelta    起始点Y轴从标，可以是数值、百分数、百分数p 三种样式；

android:toXDelta         结束点X轴坐标

android:toYDelta        结束点Y轴坐标

1. set标签-定义动作集合

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

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

    android:duration="3000"

    android:fillAfter="true"**>**

**<alpha**

    android:fromAlpha="0.0"

    android:toAlpha="1.0"**/>**

**<scale**

    android:fromXScale="0.0"

    android:toXScale="1.4"

    android:fromYScale="0.0"

    android:toYScale="1.4"

    android:pivotX="50%"

    android:pivotY="50%"**/>**

**<rotate**

    android:fromDegrees="0"

    android:toDegrees="720"

    android:pivotX="50%"

android:pivotY="50%"**/>**

**</set>**

6、如何将动画xml文件应用于控件中

Animation scaleAnimation = AnimationUtils.loadAnimation(this, R.anim.scaleanim);

tv.startAnimation(scaleAnimation);

7、interpolator插值

interpolator属性是Animation类的一个xml属性，指定动画如何变化。

AccelerateDecelerateInterpolator   在动画开始与结束的地方速率改变比较慢，在中间的时候加速

AccelerateInterpolator                     在动画开始的地方速率改变比较慢，然后开始加速

AnticipateInterpolator                      开始的时候向后然后向前甩

AnticipateOvershootInterpolator     开始的时候向后然后向前甩一定值后返回最后的值

BounceInterpolator                          动画结束的时候弹起

CycleInterpolator                             动画循环播放特定的次数，速率改变沿着正弦曲线

DecelerateInterpolator                    在动画开始的地方快然后慢

LinearInterpolator                            以常量速率改变

OvershootInterpolator                      向前甩一定值后再回到原来位置

android:interpolator="@android:anim/accelerate\_decelerate\_interpolator"

8、在代码中动态生成动画及插值器

scale —— ScaleAnimation

alpha —— AlphaAnimation

rotate —— RotateAnimation

translate —— TranslateAnimation

set —— AnimationSet

ScaleAnimation有下面几个构造函数：

ScaleAnimation(Context context, AttributeSet attrs)  从XML文件加载动画，基本用不到

ScaleAnimation(float fromX, float toX, float fromY, float toY)

ScaleAnimation(float fromX, float toX, float fromY, float toY, float pivotX, float pivotY)

ScaleAnimation(float fromX, float toX, float fromY, float toY, int pivotXType, float pivotXValue, int pivotYType, float pivotYValue)

在标签属性android:pivotX中有三种取值，数，百分数，百分数p；体现在构造函数中，就是最后一个构造函数的pivotXType，它的取值有三个，Animation.ABSOLUTE、Animation.RELATIVE\_TO\_SELF

和Animation.RELATIVE\_TO\_PARENT。

scaleAnim = **new** ScaleAnimation(0.0f,1.4f,0.0f,1.4f,Animation.RELATIVE\_TO\_SELF,0.5f,Animation.RELATIVE\_TO\_SELF,0.5f);

scaleAnim.setDuration(700);

AnimationSet它自己没有XML属性，它的构造函数：

AnimationSet(Context context, AttributeSet attrs)  基本不用

AnimationSet(boolean shareInterpolator)  shareInterpolator取值true或false，取true时，指在AnimationSet中定义一个插值器（interpolater），它下面的所有动画共同。如果设为false，则表示它下面的动画自己定义各自的插值器。

增加动画的函数：

public void addAnimation (Animation a)

# Android API及优化技术

# 多线程

# socket通信原理

# 数据结构和算法

# 网络异步应用开发模式

# MAVEN

# ANT

# Android Studio获取SHA1

打开android studio的terminal，输入

keytool –v –list –keystore keystore.jks

一定要加上-v参数，否则只能看到SHA1，没有MD5，keystore.jks，需要替换为你自己的秘钥所在路径和文件名

# 触控

# 桌面

# 文件存储数据及读取

# 相机（照片+拍照）

# 感应器（重力+光感）

# 网页开发（适配）

# proguard

proguard是Android Development Kit提供的一个工具，其主要功能是可以移除Java二进制代码中没有作用的程序代码，以达到缩小软件所占的空间。同时，它也会产生有效的记录文件，告诉我们这个程序内有哪些函数是完全没有被使用到且可以被省略的。它还有另一个好处是它会将函数名称进行替换，让这个函数更难被反编译。

虽然Eclipse和ADT会生成这个配置文件，但默认它是不会启动的。必须在default.properties中加入一行指定proguard参数的指定，如：

target=Google Inc.:Google APIs:8

proguard.config=./proguard.cfg

proguard会按照这个配置的信息来修改Java所产生的二进制jar文件。因此，知道proguard到底发生了什么事其实是很重要的。

首先是指定会执行优化的次数。

-optmizationpasses 5

这个代表proguard会执行优化的次数的上限，它会一次又一次地执行优化，看看有什么部分是多余且可删除的，如果你的应用程序经过优化后，和上次并没有不同，那proguard会在这个次数之前就停止，ADT默认设置5次。

-dontusemixedcaseclassnames

-dontskipnonpubliclibraryclasses

因为proguard默认会使用大小写来重新修改类的名称，但是在android的框架中会造成错误，所以必须要指定“dontusemixedcaseclassnames”，而“dontskipnonpubliclibraryclasses”是proguard的默认值，不需要再去修改它。

此外，在底下会看到很多的-keep的选项，如下是Android Development kit帮助设置的proguard默认参数：

-keep public class \* extends android.app.Activity

-keep public class \* extends android.app.Application

-keep public class \* extends android.app.Service

-keep public class \* extends android.content.BroadcaseReceiver

-keep public class \* extends android.content.ContentProvider

-keep public class com.android.vending.licensing.ILicensingService

proguard是如何帮助我们找到这些没有用到的程序代码的呢？

首先，输出签章过的APK文件，输出后可以看到这个项目的目录下多了个proguard目录，其中有4个文件：dump.txt、mapping.txt、seeds.txt、usage.txt。usage.txt列出了所检查出来不需要存在的程序代码。dump.txt下可以看到这个apk文件的结构。seed.txt文件会累出哪些是被-keep文件所保护不被修改的类。这个文件会列出所有被保留类名称的文件，因此可以用它来检查想要保留的文件是否正确被保留。mapping.txt会看到类和变量等被替换的对应关系。

# ADB操作命令

adb的全称是android调试桥接器（android debug bridge），就是协助用户调试android手机或模拟器的。

1. 列出链接的装置

adb devices

1. 指定装置并安装应用程序

adb –s [装置序号] [其他指令]

安装apk文件：adb install [apk的路径]

假设要安装的程序是HelloWorld.apk，而要安装的庄子需要时900123，则指令如下：

adb –s 900123 install HelloWorld.apk

1. 启动和终止adb服务

adb kill-server

adb start-server

1. 打印调试信息

adb -s [装置序号] logcat

1. 过滤调试信息

adb logcat [<标签>:<重要性>]

举例而言，假设发现这台装置的电源管理好像有问题，需要查看跟电源管理服务（PowerManagerService）相关的信息：

adb logcat PowerManagerService:I \*:S

其中I为Info和更严重的信息，\*:S表示对于其他的信息都保持Silent

# 类的加载过程

# 数据存储区域（内存分析）

# JNI

# 为ViewPager添加点击事件

# 怎样做出扩展性好的插件

# android5.0源码分析

# 设计模式

# 框架选择（比较）

# 2D/3D（小游戏开发+游戏引擎）

# 将图片做成字体加载

# 客户端与服务器的RESTful接口

# 在客户端上加广告（如何产生利润）

# Android Studio使用技巧

# Android开发文档阅读笔记

# Style的使用（自定义控件的不同style显示）

# Android测试

# 源代码管理（git以及CVS）

# 适配横竖屏

# 数据安全

# 国际化

举例来说，如果应用程序同时具有中文、英文和通用语系的字符串资源，这些资源的优先级会依照当时系统的语系决定。要为特定的语系制作字符串资源是很简单的，只要另外选定一个XML文件放到指定目录即可。Android是以下列这个规则来定义不同语系的目录：

*语系-r 区域码*

如英文：values-en-rUS

# AIDL

AIDL(Android Interface Defination Language)，它允许你定义客户端与服务端达成一致的程序接口使用进程间通信相互交流。在android上面，一个进程不能正常访问另一个进程的内存。所以说，他们需要分解他们的对象为操作系统可以理解的基本单位，然后为你把这些对象按次序跨越进程边界书写这些代码是单调冗长的，所以android使用AIDL为你处理这个问题。

注意：使用AIDL只有在你允许来自不同应用的客户端跨进程通信访问你的service，并且想要在你的service中处理多线程的时候才是必要的。如果你不需要执行不同应用之间的IPC并发，你应该通过实现Binder建立你的接口，或者如果你想执行IPC，但是不需要处理多线程，那么使用Messenger实现你的接口。

定义一个AIDL接口

你必须在一个.aidl文件中使用Java变成语言语法定义你的AIDL接口，然后在提供service的应用中和任何绑定到这个service的应用中的源代码中保存它。

当你编译包含.aidl文件的应用时，android SDK工具基于这个.aidl文件生成一个IBinder接口，并且把它保存到项目的gen目录下，service必须恰当的实现这个IBinder接口，之后客户端应用可以绑定到这个服务上，然后从IBinder调用方法来执行IPC。

使用AIDL建立一个邻接的service需要遵循下面的步骤

1. 1.建立.aidl文件 这个文件使用方法签名定义了语言接口
2. 2.实现这个接口 Android SDk工具基于你的.aidl文件使用java语言生成一个接口 这个接口有一个内部抽象类，叫做Stub，它是继承Binder并且实现你AIDL接口的 你必须继承这个Stub类并且实现这些方法

3.暴露这个接口给客户端 实现一个service并且覆盖onBind()方法返回你的Stub实现类

警告：在你第一次发布AIDL之后的其中任何的改变必须保持向后兼容来避免破坏其他应用程序使用你的service 也就是说，因为你的.aidl文件必须被复制到其他应用程序中来让他们访问你service的接口，你必须维护原始接口的支持。

**1.建立.aidl文件**

AIDL使用一个简单的语法让你声明一个带有一个或者多个带有参数和返回值方法的接口 参数和返回值可以是任何类型，甚至是AIDL生成的接口

你必须使用java语言构建.aidl文件 每一个.aidl文件必须定义一个简单的接口并且要求只有接口声明和方法签名

默认的，AIDL支持下面数据类型：

1. ava语言中的所有基本数据类型（比如int、long、char、boolean等等）
2. [String](http://www.android-doc.com/reference/java/lang/String.html)
3. [CharSequence](http://www.android-doc.com/reference/java/lang/CharSequence.html)
4. [List](http://www.android-doc.com/reference/java/util/List.html) List中的所有元素必须是AIDL支持的类型之一，或者是一个其他AIDL生成的接口，或者是你定义的parcelable List可以使用范型（例如，List） 接收端的实际类经常是一个ArrayList，尽管方法是使用List接口生成的
5. [Map](http://www.android-doc.com/reference/java/util/Map.html) Map中的所有元素必须是AIDL支持的类型之一，或者是一个其他AIDL生成的接口，或者是你定义的parcelable 范型map是不被支持的（比如这种形式Map） 接收端的实际类经常是一个HashMap，尽管方法是使用Map接口生成的

对于上述类型之外的类型，你必须声明 import ，即使在同一个包内。

当定义你的service接口的时候，注意：

* 方法可以接收0或多个参数，并且有返回值或者返回void
* 所有非基本数据类型要求要求一个定向的tag来指定数据是去往哪个方向的 无论是输入、输出，还是输入输出（参加下面的例子） 基本数据类型是默认支持的，并且不能是其他的。 警告：你应该限制方向于真正需要的地方，因为排列整理参数的开销是很昂贵的。
* .aidl文件中的所有的代码注释都在生成的IBinder接口中（除了在import和包声明之前的注释）

只支持方法，你不可以在AIDL暴露静态域

**2.实现接口**

当你编译你的应用时，Android SDK工具生成一个.java接口文件用你的.aidl文件命名 生成的接口包含一个名字为Stub的子类（比如YourInterface.Stub），这是一个它父类的抽象实现，并且声明了.aidl中所有的方法

注意：Stub也定义了一些辅助的方法，最显著的就是asInterface()，它是用来接收一个IBinder（通常IBinder传递给客户端的onServiceConnected()回调方法）并且返回一个Stub接口的实例 更多细节参考Calling an IPC Method章节。

为了实现来自.aidl文件生成的接口，需要继承Binder接口（例如YourInterface.Stub）并且实现从.aidl文件中继承的方法。

当实现你的AIDL接口的时候有很多规则需要注意

1. 调用不保证在主线程中执行，所以你需要一开始就考虑多线程并且适当的build你的service为线程安全的
2. 默认的，RPC调用是同步的。 如果你知道service需要花费一些时间来完成请求，你就不应该从activity的主线程中调用它，因为它可能使得应用没有响应（Android也许会显示一个ANR的对话框），通常你应该在客户端中一个单独的线程调用它
3. 抛出的异常不会返回给调用者

**3.暴露接口给客户端**

一旦你为service实现了接口，你需要把它暴露给客户端，这样他们才能绑定到上面 为了给你的service暴露接口，继承Service并且实现onBind()方法返回一个你实现生成的Stub类（像我们在上一结讨论的那样）

**跨进程传递对象**

如果你想通过IPC接口把一个类从一个进程传递到另一个进程中，那么是可以的。 然而，你必须保证为你的类而写的代码也是对IPC通道另一端是可用的，并且你的类必须支持Parcelable接口 支持Parcelable接口是很重要的，因为它允许Android系统把对象分解为可以被组织跨进程传输基本单元

为了建立一个支持Parcelable协议的类，你必须遵守下面的规则：

1. 要实现Parcelable接口
2. 实现writeToParcel，它是用来把对象的当前状态写入到一个Parcel对象中的。
3. 在你的类中添加一个叫CREATOR的静态域，它要实现Parcelable.Creator接口
4. 最后，建立一个.aidl文件声明你的parcelable类（如下面的Rect.aidl所示） 如果你使用一个定制的构建过程，不要构建.aidl文件。与C语言中的头文件类似，.aidl文件不会被编译

AIDL使用代码中的这些域和方法封装传送和解读你的对象

**调用一个IPC方法**

下面是调用步骤，调用者必须调用一个AIDL定义的远程接口

1. 在项目中的src目录下面导入.aidl文件
2. 声明一个IBinder接口（基于AIDL生成的）的实例
3. 实现ServiceConnection
4. 调用Context.bindService()，传递到你的ServiceConnection实现中。
5. 在你的onServiceConnected()实现中，你会收到一个IBinder实例（称为服务端） 调用YourInterfaceName.Stub.asInterface((IBinder)service)把返回值映射到YourInterface类型上面
6. 调用你接口中定义的方法 你应该捕获当连接损坏时抛出的DeadObjectException异常，这是远程方法唯一会抛出的异常
7. 使用你接口的实例调用Context.unbindService()来断开连接

调用IPC服务端的一些注释：

* 对象跨进程时是引用计数的

你可以传递一个匿名对象作为方法的参数

源代码路径：

http://www.android-doc.com/resources/samples/ApiDemos/src/com/example/android/apis/app/index.html

# 手机如何录制屏幕及转GIF

有时候我们需要录制Android手机的屏幕，比如写了一个Demo应用，需要发布到博客和微博上。

对于Android4.4的上的手机，系统自带了一个命令screenrecord，我们可以很方便的使用。

**录制命令**

adb shell screenrecord /sdcard/test.mp4

视频保存目录可以自己指定，如上面的/sdcard/test.mp4，命令执行后会一直录制180s，按下ctrl+c可以提前结束录制

**设定视频分辨率**

对于高分辨率的手机，录制的视频很大，我们分享又不需要这么大的

我们可以设置录制的视频分辨率

adb shell screenrecord --size 300\*480 /sdcard/test.mp4

**设定视频比特率**

默认比特率是4M/s，为了分享方便，我们可以调低比特率为3M

adb shell screenrecord --bit-rate 3000000 /sdcard/test.mp4

**获取视频文件**

使用adb pull 即可把手机SD卡中视频获取到本地

adb pull /sdcard/test.avi .

**转GIF文件**

在Windows下有个不错的软件 [***Free Video to GIF Converter***](http://www.video-gif-converter.com/index.html) 可以把mp4转换成GIF。

转换时还可以删除不需要的帧，这点真得很不错。

Mac上可以使用 [***gifrocket***](http://www.gifrocket.com/) 进行转换。

还有一些在线的 [***转换工具***](http://ezgif.com/video-to-gif) 可以使用，但是都会打上水印。