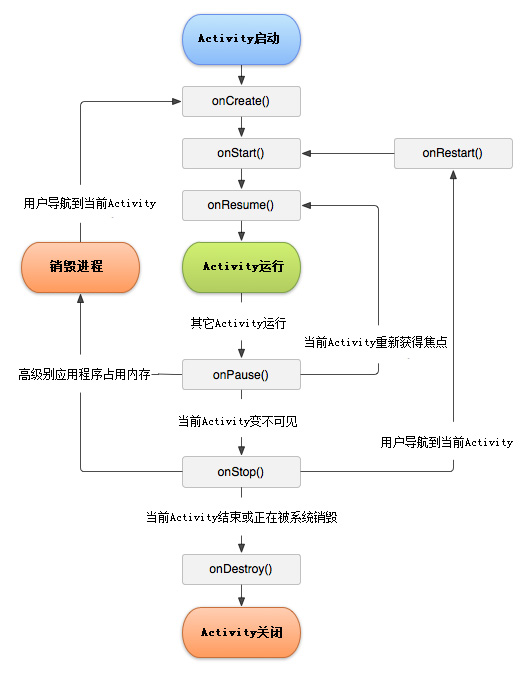
基础

谈谈Activity的生命周期



介绍不同场景下Activity生命周期的变化过程

* **启动Activity**： onCreate()—>onStart()—>onResume()，Activity进入运行状态。
* **Activity退居后台**： 当前Activity转到新的Activity界面或按Home键回到主屏： onPause()—>onStop()，进入停滞状态。
* **Activity返回前台**： onRestart()—>onStart()—>onResume()，再次回到运行状态。
* **Activity退居后台，且系统内存不足**， 系统会杀死这个后台状态的Activity，若再次回到这个Activity,则会走onCreate()–>onStart()—>onResume()
* **锁定屏与解锁屏幕** 只会调用onPause()，而不会调用onStop方法，开屏后则调用onResume()

Activity销毁但Task如果没有销毁掉，当Activity重启时这个AsyncTask该如何解决？

还是屏幕旋转这个例子，在重建Activity的时候，会回调   
Activity.onRetainNonConfigurationInstance()重新传递一个新的对象给AsyncTask，完成引用的更新

Asynctask为什么要设置为只能够一次任务

考虑到线程安全问题

若Activity已经销毁,此时AsynTask执行完并返回结果,会报异常么?

当一个App旋转时，整个Activity会被销毁和重建。当Activity重启时，AsyncTask中对该Activity的引用是无效的，因此onPostExecute()就不会起作用，若AsynTask正在执行，折会报 view not attached to window manager 异常

同样也是生命周期的问题，在 Activity 的onDestory()方法中调用Asyntask.cancal方法，让二者的生命周期同步

内存不足时,系统会杀死后台的Activity,如果需要进行一些临时状态的保存,在哪个方法进行

Activity的 onSaveInstanceState() 和 onRestoreInstanceState()并不是生命周期方法,不同于 onCreate()、onPause()等生命周期方法，它们并不一定会被触发。当应用遇到意外情况（如：内存不足、用户直接按Home键）由系统销毁一个Activity，onSaveInstanceState() 会被调用。但是当用户主动去销毁一个Activity时，例如在应用中按返回键，onSaveInstanceState()就不会被调用。除非该activity是被用户主动销毁的，通常onSaveInstanceState()只适合用于保存一些临时性的状态，而onPause()适合用于数据的持久化保存。

介绍Activity 四中launchMode:

* standard
* singleTop
* singleTask
* singleInstance

我们可以在AndroidManifest.xml配置的android:launchMode属性为以上四种之一。   
1. standard standard模式是默认的启动模式，不用为配置android:launchMode属性即可，当然也可以指定值为standard。standard启动模式，不管有没有已存在的实例，都生成新的实例。   
2. singleTop 我们在上面的基础上为指定属性android:launchMode=”singleTop”，系统就会按照singleTop启动模式处理跳转行为。跳转时系统会先在栈结构中寻找是否有一个Activity实例正位于栈顶，如果有则不再生成新的，而是直接使用。如果系统发现存在有Activity实例,但不是位于栈顶，重新生成一个实例。 这就是singleTop启动模式，如果发现有对应的Activity实例正位于栈顶，则重复利用，不再生成新的实例。   
3. singleTask 如果发现有对应的Activity实例，则使此Activity实例之上的其他Activity实例统统出栈，使此Activity实例成为栈顶对象，显示到幕前。   
4. singleInstance 这种启动模式比较特殊，因为它会启用一个新的栈结构，将Acitvity放置于这个新的栈结构中，并保证不再有其他Activity实例进入。

LaunchMode使用场景

1. singleTop适合接收通知启动的内容显示页面。   
   例如，某个新闻客户端的新闻内容页面，如果收到10个新闻推送，每次都打开一个新闻内容页面是很烦人的。
2. singleTask适合作为程序入口点。   
   例如浏览器的主界面。不管从多少个应用启动浏览器，只会启动主界面一次，其余情况都会走onNewIntent，并且会清空主界面上面的其他页面。
3. singleInstance应用场景：   
   闹铃的响铃界面。 你以前设置了一个闹铃：上午6点。在上午5点58分，你启动了闹铃设置界面，并按 Home 键回桌面；在上午5点59分时，你在微信和朋友聊天；在6点时，闹铃响了，并且弹出了一个对话框形式的 Activity(名为 AlarmAlertActivity) 提示你到6点了(这个 Activity 就是以 SingleInstance 加载模式打开的)，你按返回键，回到的是微信的聊天界面，这是因为 AlarmAlertActivity 所在的 Task 的栈只有他一个元素， 因此退出之后这个 Task 的栈空了。如果是以 SingleTask 打开 AlarmAlertActivity，那么当闹铃响了的时候，按返回键应该进入闹铃设置界面。

如何把一个应用设置为系统应用

1. Android设置是Debug版本,且root,直接将该apk用adb工具push到system/app或system/priv-app
2. 如果是非root设备,需要编译后烧写镜像
3. 有些权限(如WRITE\_SECURE\_SETTINGS)不开放给第三方应用,只能在对应设备源码总编译然后作为系统app使用

Activity,Window,View三者的联系和区别?

Activity像一个工匠（控制单元），Window像窗户（承载模型），View像窗花（显示视图） LayoutInflater像剪刀，Xml配置像窗花图纸。

Activity启动Service的两种方式

1. startService:生命周期和调用者不同.启动后若调用者未调用stopService而直接退出,Service仍会运行
2. bindService:生命周期与调用者绑定,调用者一旦退出,Service就会调用unBind->onDestory

Android两个应用能在同一个任务栈吗？

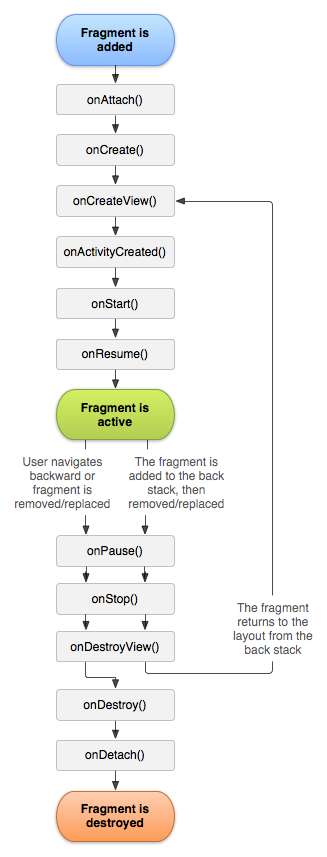
栈一般以包名命名，两个应用的签名和udid要相同

Fragment是什么?你曾经遇到哪些有关Fragment的问题?

Fragment可以作为Activity界面的一部分组成出现.一个Activity中可以同时出现多个Fragment,并一个Fragment也可以在多个Activity中使用.

在Activity中可以添加,删除,替换Fragment.Fragment可以响应自己的输入时间,并且有自己的生命周期,但其生命周期收Activity影响.

Fragment生命周期



如何实现Activity窗口快速变暗

\* @param from\>=0&&from\<=1.0f

\* @param to\>=0&&to\<=1.0f

\*

\* \*/

private void dimBackground(final float from, final float to) {

final Window window = getWindow();

ValueAnimator valueAnimator = ValueAnimator.ofFloat(from, to);

valueAnimator.setDuration(500);

valueAnimator.addUpdateListener(new AnimatorUpdateListener() {

@Override

public void onAnimationUpdate(ValueAnimator animation) {

WindowManager.LayoutParams params = window.getAttributes();

params.alpha = (Float) animation.getAnimatedValue();

window.setAttributes(params);

}

});

valueAnimator.start();

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19

Fragment重叠问题

是否使用过本地广播,和全局广播有什么区别?

本地广播在本应用范围内传播,不用担心隐私数据泄露,不用担心别的应用伪造广播.相比全局广播,本地广播更高效.

注册广播的几种方法?

1.静态注册:在清单文件中注册， 常见的有监听设备启动，常驻注册不会随程序生命周期改变   
2.动态注册:在代码中注册，随着程序的结束，也就停止接受广播了

补充一点：有些广播只能通过动态方式注册，比如时间变化事件、屏幕亮灭事件、电量变更事件，因为这些事件触发频率通常很高，如果允许后台监听，会导致进程频繁创建和销毁，从而影响系统整体性能

为什么Android引入广播机制?

ａ:从MVC的角度考虑(应用程序内) 其实回答这个问题的时候还可以这样问，android为什么要有那4大组件，现在的[**移动开发**](http://lib.csdn.net/base/android)模型基本上也是照搬的web那一套MVC[**架构**](http://lib.csdn.net/base/architecture)，只不过是改了点嫁妆而已。android的四大组件本质上就是为了实现移动或者说[**嵌入式**](http://lib.csdn.net/base/embeddeddevelopment)设备上的MVC架构，它们之间有时候是一种相互依存的关系，有时候又是一种补充关系，引入广播机制可以方便几大组件的信息和数据交互。   
b：程序间互通消息(例如在自己的应用程序内监听系统来电)   
c：效率上(参考UDP的广播协议在局域网的方便性)   
d：设计模式上(反转控制的一种应用，类似监听者模式)

BroadCastReceiver的安全性问题

具体参见:[[http://blog.csdn.net/t12x3456/article/details/9256609]](http://blog.csdn.net/t12x3456/article/details/9256609%5d)

了解IntentServices吗?

IntentService是Service的子类，是一个异步的，会自动停止的服务，很好解决了传统的Service中处理完耗时操作忘记停止并销毁Service的问题

生成一个默认的且与线程相互独立的工作线程执行所有发送到onStartCommand()方法的Intent,可以在onHandleIntent()中处理.

串行队列,每次只运行一个任务,不存在线程安全问题,所有任务执行完后自动停止服务,不需要自己手动调用stopSelf()来停止.

Service的onCreate运行在哪个线程中?

UI线程

提升Service进程优先级

在AndroidManifest.xml文件中对于intent-filter可以通过android:priority = “1000”这个属性设置最高优先级，1000是最高值，如果数字越小则优先级越低，同时适用于广播。

介绍Android下的数据存储方式

1. SharedPreference
2. 内部存储
3. 外部存储
4. SQLite
5. 网络存储

ContentProvider是如何实现数据共享

当一个应用程序需要把自己的数据暴露给其他程序使用时，该就用程序就可通过提供ContentProvider来实现；其他应用程序就可通过ContentResolver来操作ContentProvider暴露的数据。 一旦某个应用程序通过ContentProvider暴露了自己的数据操作接口，那么不管该应用程序是否启动，其他应用程序都可以通过该接口来操作该应用程序的内部数据，包括增加数据、删除数据、修改数据、查询数据等。

ContentProvider以某种Uri的形式对外提供数据，允许其他应用访问或修改数据；其他应用程序使用ContentResolver根据Uri去访问操作指定数据。 步骤：   
1. 定义自己的ContentProvider类，该类需要继承Android提供的ContentProvider基类。   
2. 在AndroidManifest.xml文件中注册个ContentProvider，注册ContenProvider时需要为它绑定一个URL。 例： android:authorities=”com.myit.providers.MyProvider” /> 说明：authorities就相当于为该ContentProvider指定URL。 注册后，其他应用程序就可以通过该Uri来访问MyProvider所暴露的数据了。   
3. 接下来，使用ContentResolver操作数据，Context提供了如下方法来获取ContentResolver对象。 一般来说，ContentProvider是单例模式，当多个应用程序通过ContentResolver来操作 ContentProvider提供的数据时，ContentResolver调用的数据操作将会委托给同一个ContentProvider处理。 使用ContentResolver操作数据只需两步： 1、调用Activity的ContentResolver获取ContentResolver对象。 2、根据需要调用ContentResolver的insert()、delete()、update()和query（）方法操作数据即可

ContentProvider和sql的区别

ContentProvider的主要还是用于数据共享，其可以对Sqlite，SharePreferences，File等进行数据操作用来共享数据。而sql的可以理解为[**数据库**](http://lib.csdn.net/base/mysql)的一门语言，可以使用它完成CRUD等一系列的操作

数据存储相关

* **文件存储**： 通过[**Java**](http://lib.csdn.net/base/javase).io.FileInputStream和java.io.FileOutputStream这两个类来实现对文件的读写，java.io.File类则用来构造一个具体指向某个文件或者文件夹的对象。
* **SharedPreferences**： SharedPreferences是一种轻量级的数据存储机制，他将一些简单的数据类型的数据，包括boolean类型，int类型，float类型，long类型以及String类型的数据，以键值对的形式存储在应用程序的私有Preferences目录（/data/data/<包名>/shared\_prefs/）中，这种Preferences机制广泛应用于存储应用程序中的配置信息。
* **SQLite数据库**： 当应用程序需要处理的数据量比较大时，为了更加合理地存储、管理、查询数据，我们往往使用关系数据库来存储数据。Android系统的很多用户数据，如联系人信息，通话记录，短信息等，都是存储在SQLite数据库当中的，所以利用操作SQLite数据库的API可以同样方便的访问和修改这些数据。
* **ContentProvider**: 主要用于在不同的应用程序之间实现数据共享的功能，不同于sharepreference和文件存储中的两种全局可读写操作模式，内容提供其可以选择只对哪一部分数据进行共享，从而保证我们程序中的隐私数据不会有泄漏的风险

如何导入外部数据库?

如何将打开res aw目录中的数据库文件?

在Android中不能直接打开res aw目录中的数据库文件，而需要在程序第一次启动时将该文件复制到手机内存或SD卡的某个目录中，然后再打开该数据库文件。复制的基本方法是使用getResources().openRawResource方法获得res aw目录中资源的 InputStream对象，然后将该InputStream对象中的数据写入其他的目录中相应文件中。在Android SDK中可以使用SQLiteDatabase.openOrCreateDatabase方法来打开任意目录中的SQLite数据库文件。

一条最长的短信息约占多少byte?

中文70(包括标点)，英文160，160个字节。

Context与ApplicationContext的区别,

Application的Context是一个全局静态变量，SDK的说明是只有当你引用这个context的生命周期超过了当前activity的生命周期，而和整个应用的生命周期挂钩时，才去使用这个application的context。   
在android中context可以作很多操作，但是最主要的功能是加载和访问资源。在android中有两种context，一种是 application context，一种是activity context，通常我们在各种类和方法间传递的是activity context。

什么是aar?aar是jar有什么区别?

“aar”包是 Android 的类库项目的二进制发行包。   
文件扩展名是.aar，maven 项目类型应该也是aar，但文件本身是带有以下各项的 zip 文件：   
/AndroidManifest.xml (mandatory)   
/classes.jar (mandatory)   
/res/ (mandatory)   
/R.txt (mandatory)   
/assets/ (optional)   
/libs/\*.jar (optional)   
/jni//\*.so (optional)   
/proguard.txt (optional)   
/lint.jar (optional)   
这些条目是直接位于 zip 文件根目录的。 其中R.txt 文件是aapt带参数–output-text-symbols的输出结果。   
jar打包不能包含资源文件，比如一些drawable文件、xml资源文件之类的,aar可以。

SQLite支持事务吗?添加删除如何提高性能?

SQLite作为轻量级的数据库，比[**MySQL**](http://lib.csdn.net/base/mysql)还小，但支持SQL语句查询，提高性能可以考虑通过原始经过优化的SQL查询语句方式处理

SQLite优化

参考:[[http://www.cnblogs.com/devinzhang/archive/2012/01/16/2323949.html]](http://www.cnblogs.com/devinzhang/archive/2012/01/16/2323949.html%5d)

如何将SQLite数据库(dictionary.db文件)与apk文件一起发布?

可以将dictionary.db文件复制到Eclipse Android工程中的res aw目录中。所有在res aw目录中的文件不会被压缩，这样可以直接提取该目录中的文件。可以将dictionary.db文件复制到res aw目录中

Webview中的漏洞

[[http://jiajixin.cn/2014/09/16/webview-js-safety/]](http://jiajixin.cn/2014/09/16/webview-js-safety/%5d)

Service和Activity通信

1. 通过Binder
2. 通过broadcast

如何保证Service在后台不被kill

1. **Service设置成START\_STICKY** kill 后会被重启（等待5秒左右），重传Intent，保持与重启前一样
2. **通过 startForeground将进程设置为前台进程**， 做前台服务，优先级和前台应用一个级别​，除非在系统内存非常缺，否则此进程不会被 kill
3. **双进程Service**： 让2个进程互相保护\*\*，其中一个Service被清理后，另外没被清理的进程可以立即重启进程
4. **QQ黑科技**: 在应用退到后台后，另起一个只有 1 像素的页面停留在桌面上，让自己保持前台状态，保护自己不被后台清理工具杀死
5. **在已经root的设备下，修改相应的权限文件,将App伪装成系统级的应用** Android4.0系列的一个漏洞，已经确认可行
6. **用C编写守护进程(即子进程)**: Android系统中当前进程(Process)fork出来的子进程，被系统认为是两个不同的进程。当父进程被杀死的时候，子进程仍然可以存活，并不受影响。**鉴于目前提到的在Android->- Service层做双守护都会失败**，我们可以fork出c进程，多进程守护。死循环在那检查是否还存在，具体的思路如下（Android5.0以上的版本不可行）
7. 用C编写守护进程(即子进程)，守护进程做的事情就是循环检查目标进程是否存在，不存在则启动它。
8. 在NDK环境中将1中编写的C代码编译打包成可执行文件(BUILD\_EXECUTABLE)。主进程启动时将守护进程放入私有目录下，赋予可执行权限，启动它即可。
9. **联系厂商，加入白名单**

谈谈你对Android中Context的理解

参考:[[http://blog.csdn.net/qinjuning/article/details/7310620]](http://blog.csdn.net/qinjuning/article/details/7310620%5d)

RemoteView的应用

widget和Notification中

Android中如何获得手机的唯一标示.

1 首先尝试读取IMEI、Mac地址、CPU号等物理信息（有不少工具可以修改IMEI）；   
2 如果均失败，可以自己生成UUID然后保存到文件（文件也可能被篡改或删除）

参考:[[http://blog.csdn.net/xushuaic/article/details/25077179]](http://blog.csdn.net/xushuaic/article/details/25077179%5d)

Android应用中验证码登录都有哪些实现方案

验证码应该只有两种获取方式： 从服务器端获取图片， 通过短信服务，将验证码发送给客户端这两种

为什么要设计Bundle而不是直接使用Map?

Map里实现了Serializable接口，而在Bundle实现了Parcelable的接口   
Bundle 父类 BaseBundle内部确实有个 ArrayMap

Android中XML解析方式的比较急优缺点

DOM,SAX,Pull解析。   
SAX解析器的优点是解析速度快，占用内存少；   
DOM在内存中以树形结构存放，因此检索和更新效率会更高。但是对于特别大的文档，解析和加载整个文档将会很耗资源，不适合移动端；   
PULL解析器的运行方式和SAX类似，都是基于事件的模式，PULL解析器小巧轻便，解析速度快，简单易用，非常适合在Android移动设备中使用，Android系统内部在解析各种XML时也是用PULL解析器。

布局相关

LinearLayout和RelativeLayout性能对比

1 RelativeLayout会让子View调用2次onMeasure，LinearLayout 在有weight时，也会调用子View2次onMeasure   
2 RelativeLayout的子View如果高度和RelativeLayout不同，则会引发效率问题，当子View很复杂时，这个问题会更加严重。如果可以，尽量使用padding代替margin。   
3 在不影响层级深度的情况下,使用LinearLayout和FrameLayout而不是RelativeLayout。   
最后再思考一下文章开头那个矛盾的问题，为什么Google给开发者默认新建了个RelativeLayout，而自己却在DecorView中用了个LinearLayout。因为DecorView的层级深度是已知而且固定的，上面一个标题栏，下面一个内容栏。采用RelativeLayout并不会降低层级深度，所以此时在根节点上用LinearLayout是效率最高的。而之所以给开发者默认新建了个RelativeLayout是希望开发者能采用尽量少的View层级来表达布局以实现性能最优，因为复杂的View嵌套对性能的影响会更大一些。

屏幕适配相关

dp, dip, dpi, px, sp是什么意思以及他们的换算公式？layout-sw400dp, layout-h400dp分别代表什么意思

布局优化

* 避免OverDraw过渡绘制
* 优化布局层级
* 避免嵌套过多无用布局
* 当我们在画布局的时候，如果能实现相同的功能，优先考虑相对布局，然后在考虑别的布局，不要用绝对布局。
* 使用<include />标签把复杂的界面需要抽取出来
* 使用<merge />标签，因为它在优化UI结构时起到很重要的作用。目的是通过删减多余或者额外的层级，从而优化整个Android Layout的结构。核心功能就是减少冗余的层次从而达到优化UI的目的！
* ViewStub 是一个隐藏的，不占用内存空间的视图对象，它可以在运行时延迟加载布局资源文件。

mipmap文件夹和drawable文件夹的区别

它只是用来放启动图标的,好处就是，你只用放一个mipmap图标，它就会给你各种版本（比如平板，手机）的apk自动生成相应分辨率的图标，以节约空间。

ListView卡顿的原因以及优化策略

* **重用converView**： 通过复用converview来减少不必要的view的创建，另外Infalte操作会把xml文件实例化成相应的View实例，属于IO操作，是耗时操作。
* **减少findViewById()操作**： 将xml文件中的元素封装成viewholder静态类，通过converview的setTag和getTag方法将view与相应的holder对象绑定在一起，避免不必要的findviewbyid操作
* **避免在 getView 方法中做耗时的操作**: 例如加载本地 Image 需要载入内存以及解析 Bitmap ，都是比较耗时的操作，如果用户快速滑动listview，会因为getview逻辑过于复杂耗时而造成滑动卡顿现象。用户滑动时候不要加载图片，待滑动完成再加载，可以使用这个第三方库[glide](https://github.com/bumptech/glide" \t "_blank)
* **Item的布局层次结构尽量简单，避免布局太深或者不必要的重绘**
* **尽量能保证 Adapter 的 hasStableIds() 返回 true** 这样在 notifyDataSetChanged() 的时候，如果item内容并没有变化，ListView 将不会重新绘制这个 View，达到优化的目的
* **在一些场景中，ScollView内会包含多个ListView，可以把listview的高度写死固定下来。** 由于ScollView在快速滑动过程中需要大量计算每一个listview的高度，阻塞了UI线程导致卡顿现象出现，如果我们每一个item的高度都是均匀的，可以通过计算把listview的高度确定下来，避免卡顿现象出现
* **使用 RecycleView 代替listview**： 每个item内容的变动，listview都需要去调用notifyDataSetChanged来更新全部的item，太浪费性能了。RecycleView可以实现当个item的局部刷新，并且引入了增加和删除的动态效果，在性能上和定制上都有很大的改善
* **ListView 中元素避免半透明**： 半透明绘制需要大量乘法计算，在滑动时不停重绘会造成大量的计算，在比较差的机子上会比较卡。 在设计上能不半透明就不不半透明。实在要弄就把在滑动的时候把半透明设置成不透明，滑动完再重新设置成半透明。
* **尽量开启硬件加速**： 硬件加速提升巨大，避免使用一些不支持的函数导致含泪关闭某个地方的硬件加速。当然这一条不只是对 ListView。

如何实现一个局部更新的ListView

如何实现ListView多种布局

ViewHolder为什么要被声明成静态内部类

这个是考静态内部类和非静态内部类的主要区别之一。非静态内部类会隐式持有外部类的引用，就像大家经常将自定义的adapter在Activity类里，然后在adapter类里面是可以随意调用外部activity的方法的。当你将内部类定义为static时，你就调用不了外部类的实例方法了，因为这时候静态内部类是不持有外部类的引用的。声明ViewHolder静态内部类，可以将ViewHolder和外部类解引用。大家会说一般ViewHolder都很简单，不定义为static也没事吧。确实如此，但是如果你将它定义为static的，说明你懂这些含义。万一有一天你在这个ViewHolder加入一些复杂逻辑，做了一些耗时工作，那么如果ViewHolder是非静态内部类的话，就很容易出现内存泄露。如果是静态的话，你就不能直接引用外部类，迫使你关注如何避免相互引用。 所以将 ViewHolder内部类 定义为静态的，是一种好习惯

进程,线程

有哪些进程通信的方式?

1. Intent
2. Binder（AIDL
3. Messenger
4. BroadcastReceiver

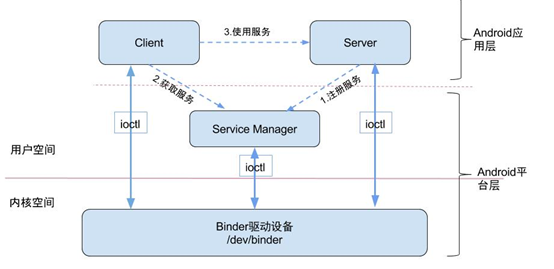
AIDL是什么?

AIDL全程是Android Interface Definition Language,用于生成两个进程之间通信(IPC)的代码.

AIDL 体现了哪些设计思想

代理

Binder机制

很多人吧Binder的原理解释的很复杂,让人看着就头大,但是熟悉[**Linux**](http://lib.csdn.net/base/linux)开发的小伙伴可以一下想到Binder驱动本质就是文件,实现采用了代理而已.具体看下图:   


简单的说说Handler机制

* Handler通过调用sendmessage方法把消息放在消息队列MessageQueue中，Looper负责把消息从消息队列中取出来，重新再交给Handler进行处理，三者形成一个循环
* 通过构建一个消息队列，把所有的Message进行统一的管理，当Message不用了，并不作为垃圾回收，而是放入消息队列中，供下次handler创建消息时候使用，提高了消息对象的复用，减少系统垃圾回收的次数
* 每一个线程，都会单独对应的一个looper，这个looper通过ThreadLocal来创建，保证每个线程只创建一个looper，looper初始化后就会调用looper.loop创建一个MessageQueue，这个方法在UI线程初始化的时候就会完成，我们不需要手动创建

动画相关

Android中的动画有哪些?

* **逐帧动画(Drawable Animation)**： 加载一系列Drawable资源来创建动画，简单来说就是播放一系列的图片来实现动画效果，可以自定义每张图片的持续时间
* **补间动画(Tween Animation)**： Tween可以对View对象实现一系列简单的动画效果，比如位移，缩放，旋转，透明度等等。但是它并不会改变View属性的值，只是改变了View的绘制的位置，比如，一个按钮在动画过后，不在原来的位置，但是触发点击事件的仍然是原来的坐标。
* **属性动画(Property Animation)**： 动画的对象除了传统的View对象，还可以是Object对象，动画结束后，Object对象的属性值被实实在在的改变了

Android动画原理

Animation框架定义了透明度，旋转，缩放和位移几种常见的动画，而且控制的是整个View，实现原理是每次绘制视图时View所在的ViewGroup中的drawChild函数获取该View的Animation的Transformation值，然后调用canvas.concat(transformToApply.getMatrix())，通过矩阵运算完成动画帧，如果动画没有完成，继续调用invalidate()函数，启动下次绘制来驱动动画，动画过程中的帧之间间隙时间是绘制函数所消耗的时间，可能会导致动画消耗比较多的CPU资源，最重要的是，动画改变的只是显示，并不能相应事件

Android属性动画特性

如果你的需求中只需要对View进行移动、缩放、旋转和淡入淡出操作，那么补间动画确实已经足够健全了。但是很显然，这些功能是不足以覆盖所有的场景的，一旦我们的需求超出了移动、缩放、旋转和淡入淡出这四种对View的操作，那么补间动画就不能再帮我们忙了，也就是说它在功能和可扩展方面都有相当大的局限性，那么下面我们就来看看补间动画所不能胜任的场景。   
注意上面我在介绍补间动画的时候都有使用“对View进行操作”这样的描述，没错，补间动画是只能够作用在View上的。也就是说，我们可以对一个Button、TextView、甚至是LinearLayout、或者其它任何继承自View的组件进行动画操作，但是如果我们想要对一个非View的对象进行动画操作，抱歉，补间动画就帮不上忙了。可能有的朋友会感到不能理解，我怎么会需要对一个非View的对象进行动画操作呢？这里我举一个简单的例子，比如说我们有一个自定义的View，在这个View当中有一个Point对象用于管理坐标，然后在onDraw()方法当中就是根据这个Point对象的坐标值来进行绘制的。也就是说，如果我们可以对Point对象进行动画操作，那么整个自定义View的动画效果就有了。显然，补间动画是不具备这个功能的，这是它的第一个缺陷。   
然后补间动画还有一个缺陷，就是它只能够实现移动、缩放、旋转和淡入淡出这四种动画操作，那如果我们希望可以对View的背景色进行动态地改变呢？很遗憾，我们只能靠自己去实现了。说白了，之前的补间动画机制就是使用硬编码的方式来完成的，功能限定死就是这些，基本上没有任何扩展性可言。   
最后，补间动画还有一个致命的缺陷，就是它只是改变了View的显示效果而已，而不会真正去改变View的属性。什么意思呢？比如说，现在屏幕的左上角有一个按钮，然后我们通过补间动画将它移动到了屏幕的右下角，现在你可以去尝试点击一下这个按钮，点击事件是绝对不会触发的，因为实际上这个按钮还是停留在屏幕的左上角，只不过补间动画将这个按钮绘制到了屏幕的右下角而已。

View绘制相关

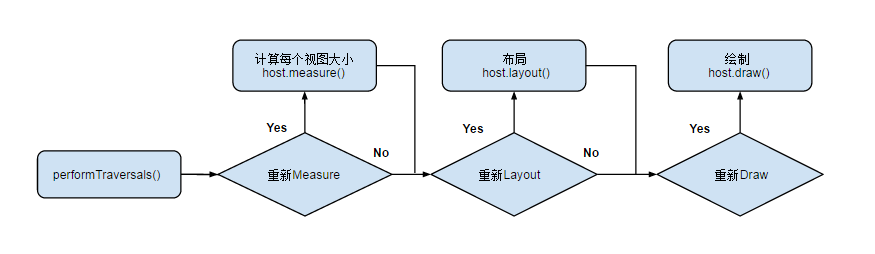
SurfaceView和View的区别

SurfaceView中采用了双缓存技术，在单独的线程中更新界面   
View在UI线程中更新界面

介绍下自定义view的基本流程

1. 明确需求，确定你想实现的效果
2. 确定是使用组合控件的形式还是全新自定义的形式，组合控件即使用多个系统控件来合成一个新控件，你比如titilebar，这种形式相对简单，参考
3. 如果是完全自定义一个view的话，你首先需要考虑继承哪个类，是View呢，还是ImageView等子类。
4. 根据需要去复写View#onDraw、View#onMeasure、View#onLayout方法
5. 根据需要去复写dispatchTouchEvent、onTouchEvent方法
6. 根据需要为你的自定义view提供自定义属性，即编写attr.xml,然后在代码中通过TypedArray等类获取到自定义属性值 7.需要处理滑动冲突、像素转换等问题

谈谈View的绘制流程

   
measure()方法，layout()，draw()三个方法主要存放了一些标识符，来判断每个View是否需要再重新测量，布局或者绘制，主要的绘制过程还是在onMeasure，onLayout，onDraw这个三个方法中

**1.onMesarue()**为整个View树计算实际的大小，即设置实际的高(对应属性:mMeasuredHeight)和宽(对应属性: mMeasureWidth)，每个View的控件的实际宽高都是由父视图和本身视图决定的。

**2.onLayout()** 为将整个根据子视图的大小以及布局参数将View树放到合适的位置上。

**3. onDraw()**开始绘制图像，绘制的流程如下

1. 首先绘制该View的背景
2. 调用onDraw()方法绘制视图本身 (每个View都需要重载该方法，ViewGroup不需要实现该方法)
3. 如果该View是ViewGroup，调用dispatchDraw ()方法绘制子视图
4. 绘制滚动条

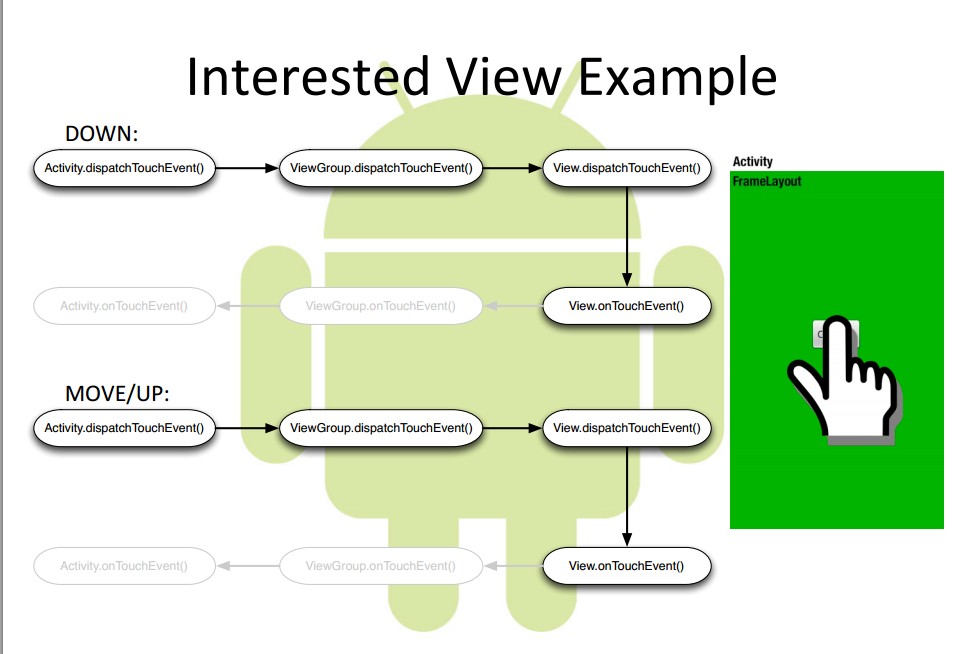
自定义View执行invalidate()方法,为什么有时候不会回调onDraw()

1. 自定义一个view时，重写onDraw。调用view.invalidate(),会触发onDraw和computeScroll()。前提是该view被附加在当前窗口.   
   view.postInvalidate(); //是在非UI线程上调用的
2. 自定义一个ViewGroup，重写onDraw。onDraw可能不会被调用，原因是需要先设置一个背景(颜色或图)。表示这个group有东西需要绘制了，才会触发draw，之后是onDraw。因此，一般直接重写dispatchDraw来绘制viewGroup.自定义一个ViewGroup,dispatchDraw会调用drawChild.

如何实现一个字体的描边与阴影效果

事件传递机制

谈谈touch事件的传递流程

   
1. 所有Touch事件都被封装成了MotionEvent对象，包括Touch的位置、时间、历史记录以及第几个手指(多指触摸)等。   
2. 事件类型分为ACTION\_DOWN, ACTION\_UP, ACTION\_MOVE, ACTION\_POINTER\_DOWN, ACTION\_POINTER\_UP, ACTION\_CANCEL，每个事件都是以ACTION\_DOWN开始ACTION\_UP结束。   
3. 对事件的处理包括三类，分别为传递——dispatchTouchEvent()函数、拦截——onInterceptTouchEvent()函数、消费——onTouchEvent()函数和OnTouchListener()

简单来说:

1. 事件从Activity.dispatchTouchEvent()开始传递，只要没有被停止或拦截，从最上层的View(ViewGroup)开始一直往下(子View)传递。子View可以通过onTouchEvent()对事件进行处理。
2. 事件由父View(ViewGroup)传递给子View，ViewGroup可以通过onInterceptTouchEvent()对事件做拦截，停止其往下传递。
3. 如果事件从上往下传递过程中一直没有被停止，且最底层子View没有消费事件，事件会反向往上传递，这时父View(ViewGroup)可以进行消费，如果还是没有被消费的话，最后会到Activity的onTouchEvent()函数。
4. 如果View没有对ACTION\_DOWN进行消费，之后的其他事件不会传递过来。
5. OnTouchListener优先于onTouchEvent()对事件进行消费。   
   上面的消费即表示相应函数返回值为true。

View中setOnTouchListener中的onTouch,onTouchEvent,onClick的执行顺序

onTouch优于onTouchEvent,onTouchEvent优于onClick

Android下滑冲突的常见解决思路

相关的滑动组件 重写onInterceptTouchEvent，然后判断根据xy值，来决定是否要拦截当前操作

高效使用Bitmap

谈谈你对Bitmap的理解,以及什么时候该bitmap.recycle()

Bitmap是android中经常使用的一个类，它代表了一个图片资源。 Bitmap消耗内存很严重，如果不注意优化代码，经常会出现OOM问题，优化方式通常有这么几种： 1. 使用缓存； 2. 压缩图片； 3. 及时回收；

至于什么时候需要手动调用recycle，这就看具体场景了，原则是当我们不再使用Bitmap时，需要回收。另外，我们需要注意，2.3之前Bitmap对象与像素数据是分开存放的，Bitmap对象存在[**Java**](http://lib.csdn.net/base/java)Heap中而像素数据存放在Native Memory中，这时很有必要调用recycle回收内存。但是2.3之后，Bitmap对象和像素数据都是存在Heap中，GC可以回收其内存。

反射相关

什么时候会用到反射?

JAVA反射机制是在#运行时#，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法；这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制。 Java反射机制主要提供了以下功能： a)在运行时判断任意一个对象所属的类； b)在运行时构造任意一个类的对象； c)在运行时判断任意一个类所具有的成员变量和方法； d)在运行时调用任意一个对象的方法；生成动态代理。

你曾经利用反射做过什么?

JNI系列

NDK是什么?

NDK是一些列工具的集合，   
NDK提供了一系列的工具，帮助开发者迅速的开发C/C++的动态库，并能自动将so和java 应用打成apk包。   
NDK集成了交叉编译器，并提供了相应的mk文件和隔离cpu、平台等的差异，开发人员只需简单的修改mk文件就可以创建出so

Android ndk主要在哪些场景下使用?有啥坑?

1. 加密
2. 音视频解码
3. 图像操作
4. 安全相关,比如hookt
5. 增量更新
6. 游戏开发

NDK开发需要注意什么?

[[https://developer.android.com/training/articles/perf-jni.html]](https://developer.android.com/training/articles/perf-jni.html%5d)

使用JNI的流程?

1. JAVA中声明native 方法如private native String printJNI(String inputStr);
2. 使用javah工具生成.h头文件这时候头文件中就会自动生成对应的函数JNIEXPORT jstring JNICALL Java\_com\_wenming\_HelloWorld\_printJNI
3. 实现JNI原生函数源文件，新建HelloWorld.c文件，对刚才自动生成的函数进行具体的逻辑书写，例如返回一个java叫做HelloWorld的字符串等
4. 编译生成动态链接so文件\*\*
5. Java中调用Sysytem.load方法把刚才的so库加载进来，就可以调用native方法了

如何通过JNI传递String对象

Java的String和C++的string是不能对等起来的，所以当我们拿到.h文件下面的jstring对象，会做一次转换我们把jstring转换为C下面的char\*类型， 获取值   
  
constchar\* str;   
str = env->GetStringUTFChars(prompt,false);   
  
赋予值   
  
char\* tmpstr ="return string succeeded";   
jstring rtstr = env->NewStringUTF(tmpstr);

网络优化

移动端获取数据优化的几个点

1. 连接复用 :   
   节省连接建立时间，如开启 keep-alive。   
   对于 Android 来说默认情况下 HttpURLConnection 和 HttpClient 都开启了 keep-alive。只是 2.2 之前 HttpURLConnection 存在影响连接池的 Bug，具体可见：[Android HttpURLConnection 及 HttpClient 选择](http://www.trinea.cn/android/android-http-api-compare/)
2. 请求合并:   
   即将多个请求合并为一个进行请求，比较常见的就是网页中的 CSS Image Sprites。如果某个页面内请求过多，也可以考虑做一定的请求合并。
3. 减少请求数据的大小：   
   对于post请求，body可以做gzip压缩的，header也可以作数据压缩(不过只支持http 2.0)。
4. 返回的数据的body也可以作gzip压缩，body数据体积可以缩小到原来的30%左右。(也可以考虑压缩返回的json数据的key数据的体积，尤其是针对返回数据格式变化不大的情况，支付宝聊天返回的数据用到了)
5. 根据用户的当前的网络质量来判断下载什么质量的图片(电商用的比较多)。

如何设计一个良好的网络层?

[[http://blog.csdn.net/column/details/simple-net.html]](http://blog.csdn.net/column/details/simple-net.html%5d)

如何防止重复发送网络请求

点击activity上的一个按钮，发送网络请求，在网络比较慢的情况下，用户可能会继续去点击按钮，这个时候，发送其他无谓的请求，不知道大家是怎么处理这类问题来拦截？   
HTTP header中加入max-age，这样某个固定的时间内都将返回empty body，当然这个方法是死的，把时间完全限制了，这个方法回掉也会同样要执行多次。   
还有个晕招，就是直接设置按钮的clickable为false，或者使用progressbar，类似于楼主的方法，比如点赞的场景。   
使用Map的话，在回掉的时候，还是需要回收HashMap的，维护Map还不如只维护一个boolean呢。   
Volley中如果开了缓存的话, 相同的请求同时只会有一个去真正的请求, 后续都走缓存, 虽然不会请求多次, 但是回调是会执行多次的, 和这个需求不match

如何实现wap联网

参考:[[http://blog.csdn.net/asce1885/article/details/7844159]](http://blog.csdn.net/asce1885/article/details/7844159%5d)

测试与调试

如何调试Android应用程序

Debug   
Log

Android中常用的测试工具?

内存分析:mat,ddms,leakcanary   
静态分析:find bugs   
压力[**测试**](http://lib.csdn.net/base/softwaretest):monkey   
自动化测试:UIAutomatorMonkeyRunner,Rubotium

内存泄漏/内存溢出相关

内存泄漏问题

1. 资源对象没有关闭造成,如查询数据库没有关闭游标
2. 构造Adapter时,没有使用缓存ConvertView
3. Bitmap对象在不使用时调用recycle()释放内存
4. context逃逸问题
5. 注册没有取消,如动态注册广播在Activity销毁前没有unregisterReceiver
6. 集合对象未清理,如无用时没有释放对象的引用
7. 在Activity中使用非静态的内部类，并开启一个长时间运行的线程，因为内部类持有Activity的引用，会导致Activity本来可以被gc时却长期得不到回收

哪些情况下发生OOM

* **类的静态变量持有**[**大数据**](http://lib.csdn.net/base/hadoop)**对象** 静态变量长期维持到大数据对象的引用，阻止垃圾回收。
* **非静态内部类存在静态实例** 非静态内部类会维持一个到外部类实例的引用，如果非静态内部类的实例是静态的，就会间接长期维持着外部类的引用，阻止被回收掉。
* **资源对象未关闭** 资源性对象比如（Cursor，File文件等）往往都用了一些缓冲，我们在不使用的时候，应该及时关闭它们， 以便它们的缓冲及时回收内存。它们的缓冲不仅存在于java虚拟机内，还存在于java虚拟机外。 如果我们仅仅是把它的引用设置为null,而不关闭它们，往往会造成内存泄露。 **解决办法**： 比如SQLiteCursor（在析构函数finalize（）,如果我们没有关闭它，它自己会调close()关闭）， 如果我们没有关闭它，系统在回收它时也会关闭它，但是这样的效率太低了。 因此对于资源性对象在不使用的时候，应该调用它的close()函数，将其关闭掉，然后才置为null. 在我们的程序退出时一定要确保我们的资源性对象已经关闭。 程序中经常会进行查询数据库的操作，但是经常会有使用完毕Cursor后没有关闭的情况。如果我们的查询结果集比较小， 对内存的消耗不容易被发现，只有在常时间大量操作的情况下才会复现内存问题，这样就会给以后的测试和问题排查带来困难和风险，记得try catch后，在finally方法中关闭连接
* **Handler内存泄漏** Handler作为内部类存在于Activity中，但是Handler生命周期与Activity生命周期往往并不是相同的，比如当Handler对象有Message在排队，则无法释放，进而导致本该释放的Acitivity也没有办法进行回收。 **解决办法**：
  + 声明handler为static类，这样内部类就不再持有外部类的引用了，就不会阻塞Activity的释放
  + 如果内部类实在需要用到外部类的对象，可在其内部声明一个弱引用引用外部类

**一些不良代码习惯** 有些代码并不造成内存泄露，但是他们的资源没有得到重用，频繁的申请内存和销毁内存，消耗CPU资源的同时，也引起内存抖动 **解决方案** 如果需要频繁的申请内存对象和和释放对象，可以考虑使用对象池来增加对象的复用。 例如ListView便是采用这种思想，通过复用converview来避免频繁的GC

如何排查OOM

参见:<http://www.jcodecraeer.com/a/anzhuokaifa/androidkaifa/2015/0920/3478.html>

如何避免OOM

**1. 使用更加轻量的**[**数据结构**](http://lib.csdn.net/base/datastructure) 例如，我们可以考虑使用ArrayMap/SparseArray而不是HashMap等传统数据结构。通常的HashMap的实现方式更加消耗内存，因为它需要一个额外的实例对象来记录Mapping操作。另外，SparseArray更加高效，在于他们避免了对key与value的自动装箱（autoboxing），并且避免了装箱后的解箱。

**2. 避免在Android里面使用Enum** Android官方培训课程提到过“Enums often require more than twice as much memory as static constants. You should strictly avoid using enums on Android.”，具体原理请参考《Android性能优化典范（三）》，所以请避免在Android里面使用到枚举。

**3. 减小Bitmap对象的内存占用** Bitmap是一个极容易消耗内存的大胖子，减小创建出来的Bitmap的内存占用可谓是重中之重，，通常来说有以下2个措施： **inSampleSize**：缩放比例，在把图片载入内存之前，我们需要先计算出一个合适的缩放比例，避免不必要的大图载入。 **decode format**：解码格式，选择ARGB\_6666/RBG\_545/ARGB\_4444/ALPHA\_6，存在很大差异

**4.Bitmap对象的复用** 缩小Bitmap的同时，也需要提高BitMap对象的复用率，避免频繁创建BitMap对象，复用的方法有以下2个措施 **LRUCache** : “最近最少使用[**算法**](http://lib.csdn.net/base/datastructure)”在Android中有极其普遍的应用。ListView与GridView等显示大量图片的控件里，就是使用LRU的机制来缓存处理好的Bitmap，把近期最少使用的数据从缓存中移除，保留使用最频繁的数据， **inBitMap高级特性**:利用inBitmap的高级特性提高Android系统在Bitmap分配与释放执行效率。使用inBitmap属性可以告知Bitmap解码器去尝试使用已经存在的内存区域，新解码的Bitmap会尝试去使用之前那张Bitmap在Heap中所占据的pixel data内存区域，而不是去问内存重新申请一块区域来存放Bitmap。利用这种特性，即使是上千张的图片，也只会仅仅只需要占用屏幕所能够显示的图片数量的内存大小

**4. 使用更小的图片** 在涉及给到资源图片时，我们需要特别留意这张图片是否存在可以压缩的空间，是否可以使用更小的图片。尽量使用更小的图片不仅可以减少内存的使用，还能避免出现大量的InflationException。假设有一张很大的图片被XML文件直接引用，很有可能在初始化视图时会因为内存不足而发生InflationException，这个问题的根本原因其实是发生了OOM。

**5.StringBuilder** 在有些时候，代码中会需要使用到大量的字符串拼接的操作，这种时候有必要考虑使用StringBuilder来替代频繁的“+”。

**4.避免在onDraw方法里面执行对象的创建** 类似onDraw等频繁调用的方法，一定需要注意避免在这里做创建对象的操作，因为他会迅速增加内存的使用，而且很容易引起频繁的gc，甚至是内存抖动。

**5. 避免对象的内存泄露** android中内存泄漏的场景以及解决办法，参考上一问

ANR错误

什么是ANR

ANR全称Application Not Responding，意思就是程序未响应。如果一个应用无法响应用户的输入，系统就会弹出一个ANR对话框，用户可以自行选择继续等待亦或者是停止当前程序。一旦出现下面两种情况，则弹出ANR对话框   
1. 应用在5秒内未响应用户的输入事件（如按键或者触摸）   
2. BroadcastReceiver未在10秒内完成相关的处理

Service在特定的时间内无法处理完成   
超时的原因一般有两种：   
(1)当前的事件没有机会得到处理（UI线程正在处理前一个事件没有及时完成或者looper被某种原因阻塞住）   
(2)当前的事件正在处理，但没有及时完成   
UI线程尽量只做跟UI相关的工作，耗时的工作（数据库操作，I/O，连接网络或者其他可能阻碍UI线程的操作）放入单独的线程处理，尽量用Handler来处理UI thread和thread之间的交互。   
UI线程主要包括如下：   
• Activity:onCreate(), onResume(), onDestroy(), onKeyDown(), onClick()   
• AsyncTask: onPreExecute(), onProgressUpdate(), onPostExecute(), onCancel()   
• Mainthread handler: handleMessage(), post(runnable r)

如何定位ANR错误

开发机器上,查看/data/anr/traces.text.最新的ANR信息在最开始部分.

如何避免ANR

避免ANR最核心的一点就是在主线程减少耗时操作.通常需要从以下几个方案下手:

* 使用子线程处理耗时IO操作。
* 降低子线程优先级使用Thread或者HandlerThread时，调用Process.setThreadPriority(Process.THREAD\_PRIORITY\_BACKGROUND)设置优先级，否则仍然会降低程序响应，因为默认Thread的优先级和主线程相同。
* 使用Handler处理子线程结果，而不是使用Thread.wait()或者Thread.sleep()来阻塞主线程。
* Activity的onCreate和onResume回调中尽量避免耗时的代码
* BroadcastReceiver中onReceive代码也要尽量减少耗时操作建议使用IntentService处理。IntentService是一个异步的，会自动停止的服务，很好解决了传统的Service中处理完耗时操作忘记停止并销毁Service的问题

安全相关

本地存储的数据怎么加密好?比如:shared\_prefs,sqlite数据,用户名,密码等.如果是aes加密,怎么保存key?

密码不要存在本地，一般保存token。最基本的要代码混淆，可以的话加壳，防止反编译

系统相关

Android各版本API区别

参考:[[http://blog.csdn.net/lijun952048910/article/details/7980562]](http://blog.csdn.net/lijun952048910/article/details/7980562%5d)

什么是Dalvik虚拟机

Dalvik虚拟机是Android平台的核心。它可以支持.dex格式的程序的运行，.dex格式是专为Dalvik设计的一种压缩格式，可以减少整体文件尺寸，提高I/O操作的速度，适合内存和处理器速度有限的系统

Dalvik虚拟机和JVM有什么区别

* Dalvik 基于寄存器，而 JVM 基于栈。基于寄存器的虚拟机对于更大的程序来说，在它们编译的时候，花费的时间更短。
* Dalvik执行.dex格式的字节码，而JVM执行.class格式的字节码

Android为每个应用程序分配的内存大小是多少

一般是16m或者24m,但是可以通过android:largeHeap申请更多内存,具体参考:   
[[https://liuzhichao.com/2016/use-android\_largeHeap.html]](https://liuzhichao.com/2016/use-android_largeHeap.html%5d)   
[[http://www.cnblogs.com/mythou/p/3203536.html]](http://www.cnblogs.com/mythou/p/3203536.html%5d)

如何解决方法数65k问题?

使用Android Studio 的gradle 可以构建MutilDex

Android系统启动流程分析

1. 打开adb shell 然后执行ps命令，可以看到首先执行的是init方法！找到init.c这个文件.
2. 然后走init里面的main方法，在这main方法里面执行mkdir进行创建很多的文件夹，和挂载一些目录
3. 然后回去初始化init.rc这个配置文件！在这个配置文件里面回去启动孵化器这个服务，这个服务会去启动app\_process这个文件夹，这个文件夹里面有个app\_main.cpp这个文件！
4. 然后在app\_main.cpp这个c文件里面在main方法里面它会去启动安卓的虚拟机，然后安卓虚拟机会去启动os.zygoteinit这个服务！
5. zygoteinit这是个java代码写的，然后我们找到了main方法，在这个方法里面我们看到他首先设置虚拟机的最小堆内存为5兆，然后走到preloadclasses（）这个方法来加载安卓系统所有的2000多个类通过类加载器加载进来,比如activity,contentx,http,…（其实没有必要一下子全部加载下来，我们可以等用到的时候在加载也可以！）
6. 然后又走preloadresources（）这个方法来预加载安卓中定义好的资源比如颜色，图片，系统的id等等。。。都加载了！（其实这也是没必要的！ ）
7. 然后又走startSystemServer(),这个方法来加载系统的服务！他会先使用natvieJNI去调用C去初始化界面和声音的服务，这就是我们为什么先听到声音和界面的原因！
8. 最后等服务加载完成后也就启动起来了!

总结 linux启动->init进程启动(加载init.rc配置)->zygote启动->systemServer启动,systemServer会通过init1和init2启动navite世界和java世界