Android Framework Foundation

# 顶层设计

## 一个应用内多进程的划分

如果一个进程占用内存超过了这个内存限制，就会报OOM的问题，很多涉及到大图片的频繁操作或者需要读取一大段数据在内存中使用时，很容易报OOM的问题。为了彻底地解决应用内存的问题，Android引入了多进程的概念，它允许在同一个应用内，为了分担主进程的压力，将占用内存的某些页面单独开一个进程，比如Flash、视频播放页面，频繁绘制的页面等。

1. 进程特点：

* Android应用的进程都是从一个叫做Zygote的进程fork出来的。Zygote进程在系统启动，并载入通用的framework的代码与资源之后开始启动。为了启动一个新的程序进程，系统会fork Zygote进程生成一个新的进程，然后在新的进程中加载并运行应用<http://blog.csdn.net/ls5718/article/details/51858976>程序的代码。这就使得大多数的RAM pages被用来分配给framework的代码，同时促使RAM资源能够在应用的所有进程之间进行共享。
* [参考链接1](http://blog.csdn.net/whb20081815/article/details/70243105)。[参考链接2](https://developer.android.com/topic/performance/memory.html)。[参考网址](https://developer.android.com/guide/components/processes-and-threads.html)。

1. 进程是系统资源和分配的基本单位，而线程是调度的基本单位。
2. 每个进程都有自己独立的资源和内存空间
3. 其它进程不能任意访问当前进程的内存和资源
4. 系统给每个进程分配的内存会有限制
5. 进程间如何通信。

在android SDK中提供了4种用于跨进程通讯的方式。这4种方式正好对应于android系统中4种应用程序组件：Activity、Content Provider、Broadcast和Service。其中Activity可以跨进程调用其他应用程序的Activity；Content Provider可以跨进程访问其他应用程序中的数据（以Cursor对象形式返回），当然，也可以对其他应用程序的数据进行增、删、改操 作；Broadcast可以向android系统中所有应用程序发送广播，而需要跨进程通讯的应用程序可以监听这些广播；Service和Content Provider类似，也可以访问其他应用程序中的数据，但不同的是，Content Provider返回的是Cursor对象，而Service返回的是Java对象，这种可以跨进程通讯的服务叫AIDL服务。

* 1. 使用Messenger
  2. AIDL
  3. ...

1. 优先级：
   1. Foreground Process（焦点所在进程）
   2. Visible Process
   3. Service Process
   4. Background Process
   5. Empty Process
2. 注意事项
   1. 什么情况下，Service需要运行在Application相同的进程？什么情况下，需要放在一个进程里面？
   2. 后台周期性运行的服务（比如IntentService, Service, JobService, JobScheduler等）需要运行在一个单独的进程吗？

## 关于Application

1. Android程序的入口是ActivityThread.main(String[] args)。
2. onCreate在主线程执行吗？执行时机是什么？
3. 四大组件的onCreate优先于Application执行。
4. 多进程和Application的关系。有哪些注意事项；Application里面哪些需要做进程区分的？进程之间共享数据是怎么样的？[多次初始化](http://blog.sina.com.cn/s/blog_c7ef5e8c0102wtns.html)。[参考链接1](http://blog.csdn.net/wei1583812/article/details/53395234).[消除卡顿](http://blog.csdn.net/axi295309066/article/details/72675365)。
5. [参考网址1](http://blog.csdn.net/Simon_Crystin/article/details/70315106).[参考网址2](http://blog.csdn.net/alwayspursuing/article/details/54965582).[参考网址3](https://www.jianshu.com/p/91f60bd0d1f9).

## 从点击App图标说起

1. 首先是系统加载程序。[参考连接1](https://www.jianshu.com/p/9ecea420eb52)。
2. 然后运行ActivityThread.main(String[] args)。

## Binder通信机制

1. 工作在Linux层面。属于驱动，运行在内核态。
2. 分为三部分：服务端接口、Binder驱动、客户端接口。

# 四大组件

## Activity

1. Android主题

## BrocastReceiver

1. 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，如果需要完成一项比较耗时的工作 , 应该通过发送 Intent 给 Service, 由Service 来完成 . 这里不能使用子线程来解决 , 因为 BroadcastReceiver 的生命周期很短 , 子线程可能还没有结束BroadcastReceiver 就先结束了 .BroadcastReceiver 一旦结束 , 此时 BroadcastReceiver 的所在进程很容易在系统需要内存时被优先杀死 , 因为它属于空进程 ( 没有任何活动组件的进程 ). 如果它的宿主进程被杀死 , 那么正在工作的子线程也会被杀死 . 所以采用子线程来解决是不可靠的。
2. 。。。

## ContentProvider

## Service

1. BindService

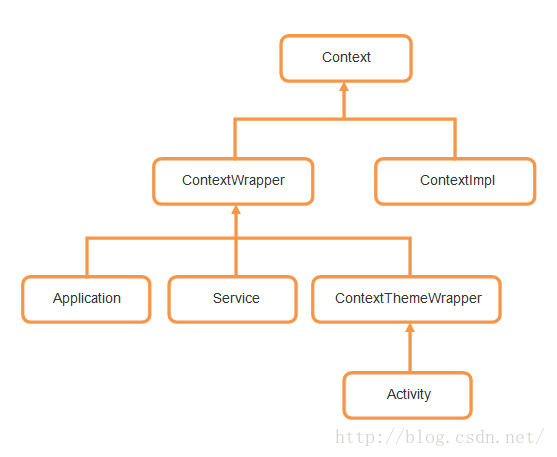
服务是如何绑定的？不同进程直接如何保证绑定的服务的方法能够执行？是Android的什么机制保证了这一行为？

1. RegisterService
2. IntentService

# Context解析

Abstract class Context是维系Android程序中各组件正常工作的一个核心“功能类”。本身没有任何数据成员，只声明了一些方法。

## 类组织结构



1. 子类
2. ContextWrapper extends Context

作为Context的代理类，Context mBase是其唯一的数据成员。其中protected void attachBaseContext(Context base)方法提供了设置mBase的入口。

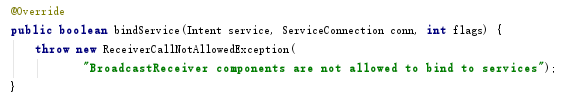
1. ContextImpl extends Context

实现了Context所有方法。

1. ReceiverRestrictedContext extends ContextWrapper
2. 静态注册的BroadcastReceiver的onReceive(Context context, Intent intent)函数conext对象类型是ReceiverRestrictedContext。
3. 不允许通过registerReceiver(BroadcastReceiver receiver, IntentFilter filter)注册新的BroadcastReceiver。但是可以传入receiver = nul。

例如Intent intent = registerReceiver(null, new IntentFilter(Intent.ACTION\_BATTERY\_CHANGED));查询电量的时候可以通过上面这种方式返回一个intent，从这个intent中能够拿到我们想要要的信息，而不是注册一个BroadcastReceiver不停回调onReceiver方法。

1. 屏蔽了bindService，为什么？



1. Activity extends ContextThemeWrapper extends ContextWrapper
2. Service extends ContextWrapper
3. Application extends ContextWrapper

## Context提供的方法

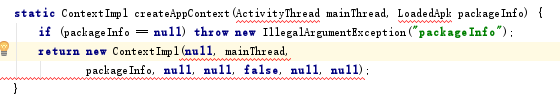
1. startActivity
2. startService
3. sendBroadCast
4. registerReceiver
5. stopService
6. unbindService
7. ...

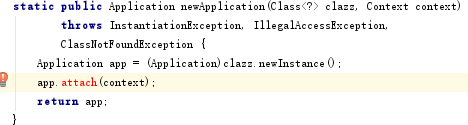
## 作用域

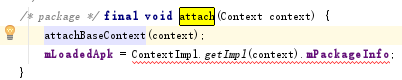
## 关键代码一览

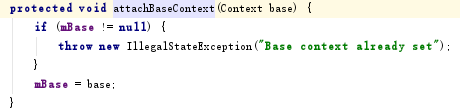
1. Application及其BaseContext的创建过程：

（LoadedApk.java里面的makeApplication函数。）

（ContextIml.java里面的createAppContext函数。）

（Instrumentation.java里面的newApplication函数。）

 （Application的attach函数。）



## 注意事项

1. 静态对象持有对Context对象的引用而会造成的内存泄漏。
2. 当Application的Context能搞定的情况下，并且生命周期长的对象，优先使用Application的Context。
3. 不要让生命周期长于Activity的对象持有到Activity的引用。
4. Receiver不是一个Contex，为什么不能在Receiver里面的intent里面执行startActivity.
5. onReceive(Context context, Intent intent)返回的Context指的是此BroadcastReceiver对象的运行环境，因此对于注册方式不一样的BroadcastReceiver，其回调函数onReceiver里面的Context是不一样的：静态注册的BroadcastReceiver，返回的是ReceiverRestrictedContext；应用内注册的，返回是Application 对象；Activity里面返回的是Activity 对象；Service里面注册的，返回的是Service 对象；

## TL.NR

1. 既然Context只有一个实现类，并且Activity,Service以及Application都是继承自ComtextWrapper,即都是Context的代理类，且其实现都是ContextImpl。
2. 三个的Context有何差别呢？在Activity里面如下图所示：



如上图所示：

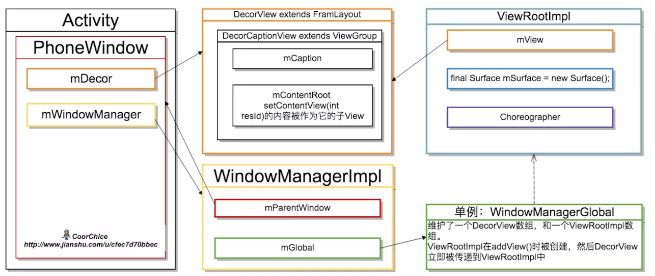
1. getApplication()，getApplicationContext()，getApplication.getApplicationContext()是用一个地址，即指向同一Application对象，因为Application间接继承自Context；
2. getBaseContext是获取此Activity中的ContextImpl对象；
3. getApplication().getBaseContext是获取的Application里面的ComtextImpl对象。

# 编译链接与加载

1. Multidex，[相关链接1](http://blog.csdn.net/u014247915/article/details/52209835) [链接2](https://www.jianshu.com/p/79a14d340cb0) [链接3](https://www.jianshu.com/p/dd90d7e7c691)，[参考链接4](https://www.jianshu.com/p/72d311d72fcb)，
2. 类加载器
   1. Java 源程序（.java 文件）在经过 Java 编译器编译之后就被转换成 Java 字节代码（.class 文件）。类加载器负责读取 Java 字节代码，并转换成 java.lang.Class类的一个实例。每个这样的实例用来表示一个 Java 类。通过此实例的 newInstance()方法就可以创建出该类的一个对象。实际的情况可能更加复杂，比如 Java 字节代码可能是通过工具动态生成的，也可能是通过网络下载的。
   2. Java.lang.ClassLoader类的基本职责就是根据一个指定的类的名称，找到或者生成其对应的字节代码，然后从这些字节代码中定义出一个 Java 类，即 java.lang.Class类的一个实例。除此之外，ClassLoader还负责加载 Java 应用所需的资源，如图像文件和配置文件等。不过本文只讨论其加载类的功能。

# 视图框架

基本框架如下图所示：[参考链接1](https://juejin.im/entry/58f5b6a18d6d81006493bb7e)，



# 运行效率与优化

## 用户交互流畅性

* + - 1. 数据库操作的优化。应用起的的时候调用openDataBase会导致应用启动卡顿。
      2. 内存优化。
      3. APP为什么总是卡顿？[参考链接1](https://juejin.im/entry/58f5b6a18d6d81006493bb7e)，[参考链接2](https://mp.weixin.qq.com/s?__biz=MzA3NjA3NTI5Mg==&mid=2656330117&idx=1&sn=a304224af107ab97a6dfc8e04e153bef&chksm=84c619f6b3b190e08796ba9448d8e23e92b9977e1c12845457b27b3cb6e824d455ab7759e400&scene=21" \l "wechat_redirect)，[参考网址3](https://www.jianshu.com/p/5ad013eb5364?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation).[参考网址4](https://www.jianshu.com/p/dad51f6c9c4d?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation).[参考网址5](https://mp.weixin.qq.com/s?__biz=MzA3NjA3NTI5Mg==&mid=2656330117&idx=1&sn=a304224af107ab97a6dfc8e04e153bef&chksm=84c619f6b3b190e08796ba9448d8e23e92b9977e1c12845457b27b3cb6e824d455ab7759e400&scene=21" \l "wechat_redirect).

## 启动优化

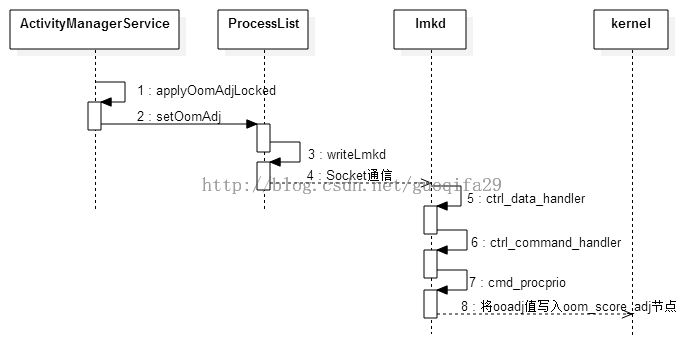
* + - 1. 啊手动阀

## 内存优化

Android进程回收主要涉及到两个组件：ActivityManagerService(AMS)和lowmemoryKiller。进程对用户越不重要（Oomadj值就越大），占用内存越大，进程就越容易被干掉。

详细地说，当app状态发生改变时，比如退到后台时，AMS会对app的进程计算出一个值，即Oomadj（ams#computeOomAdjLocked），然后把这个值传给linux内核，lowmemorykiller就可以拿到这个值了，lowmemorykiller则就有了所有app进程的Oomadj值，即进程对用户的重要程度。当手机内存不足时，lowmemorykiller就有了足够的信息决定干掉哪个进程了。

* AMS计算出一个危险的Oomadj值会调用onTrimMemory通知app，此时app应该把不重要的内存释放掉，只要比友商app占用的内存小被 lowmemorykiller干掉的概率就小。
* 在收到onTrimMemory的时候，可以干掉其他应用，以保护自己。



* + - 1. 如何正确加载图片：LRU、弱引用
      2. [参考网址1](https://www.jianshu.com/p/75e36c1428e2).[参考网址2](https://mp.weixin.qq.com/s/AjtzDxwJzyqC95FXgDPS1g)[.参考网址3](https://mp.weixin.qq.com/s?__biz=MzA3NTYzODYzMg==&mid=2653579440&idx=1&sn=1cb59ac4de5a356eda53907b51b17cad&scene=21" \l "wechat_redirect).[参考网址4](http://www.heqiangfly.com/2017/06/15/android-knowledge-point-drawable-cache/).[参考网址5](http://blog.csdn.net/heqiangflytosky/article/details/73278146).[参考网址6](https://www.jianshu.com/p/6e3ba6b71313).
      3. [参考网址1](https://www.jianshu.com/p/b46870bcafc2).
      4. 不要用枚举； float 类型的数据存取速度是 int 类型的一半，尽量优先采用 int 类型。

## 数据存取优化

* + - 1. 数据库查询优化
         1. 安抚应尽量避免在 where 子句中使用!=或<>操作符。否则将引擎放弃使用索引而进行全表扫描。
         2. 应尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描。
         3. 不论什么地方都不要使用 select \* from t ，用详细的字段列表取代“\*”，不要返回用不到的不论什么字段。
         4. 操作大数据量时开启事务对 SQLite 进行优化。
         5. 频繁的数据库查询操作，并且所查询的表通常为空，会不会影响到效率？
      2. ShparePreference优化
      3. Serializable 和 Parcelable 的区别

两者最大的区别在于存储媒介的不同，Serializable 使用 I/O 读写存储在硬盘上，而 Parcelable 是直接在内存中读写。

很明显，内存的读写速度通常大于 IO 读写，所以在 Android中传递数据优先选择Parcelable。Serializable 会使用反射，序列化和反序列化过程需要大量 I/O 操作， Parcelable 自已实现封送和解封（marshalled &unmarshalled）操作不需要用反射，数据也存放在 Native 内存中，效率要快很多。

* + - 1. 两个Activity 之间传递对象还需要注意对象的大小，使用的 Binder 的缓冲区是有大小限制的（有些手机是 2 M），而一个进程默认有 16 个 Binder 线程，所以一个线程能占用的缓冲区就更小了（ 有人以前做过测试，大约一个线程可以占用 128 KB）。
      2. 啊手动阀

## 绘制优化

* + - 1. 啊手动阀
      2. 阿斯顿f

## 电量优化

# Others

## HandlerThread

## 在UI线程新建一个HandlerMessage有何意义？

## 判断是否在主线程。

## Instrumentation

## ANR

1. 当前发生ANR的应用进程被第一个添加进firstPids集合中，所以会第一个向traces文件中写入信息。反过来说，traces文件中出现的第一个进程正常情况下就是发生ANR的那个进程。不过有时候会很不凑巧，发生ANR的进程还没有来得及输出trace信息，就由于某种原因退出了，所以偶尔会遇到traces文件中找不到发生ANR的进程信息的情况。
2. 获取anr信息：setprop dalvik.vm.stack-trace-file /tmp/stack-traces.txt；adb dump /data/anr/traces.txt。
3. 每次发生ANR时都会删除旧的traces文件，重新创建新文件。也就是说Android只保留最后一次发生ANR时的traces信息。
4. [参考网址](http://rayleeya.iteye.com/blog/1955657)。

# Adb Shell

## Adb shell

1. Adb shell dump meminfo “package name”
2. Adb shell screenrecord /sdcard/aaaa.mp4 > adb pull /sdcard/aaa.mp4 录屏
3. Adb shell screencap /sdcard/aaa.png > adb pull /sdcard/aaa.png
4. Adb shell dumpsys activity activities 获取当前activity栈
5. Adb shell pm clear “com.lm.powersecurity” 清楚应用数据
6. Adb shell pm list packages 列出手机应用

## 调试。[参考链接1](http://www.freebuf.com/articles/terminal/114869.html)，

# 适配与兼容

## V4与V7、V13详解

如果在低版本Android平台上开发一个应用程序，而应用程序又想使用高版本才拥有的功能，就需要使用Support库。[参考网址](https://developer.android.com/topic/libraries/support-library/index.html)，[参考网址2](https://www.jianshu.com/p/d36a1e5bf246)，[参考网址3](https://www.jianshu.com/p/9c3c9cccc8df)，

1. V13只在开发平板机上使用。

# Android代码框架与设计模式

## 享元模式

1. Handler和Looper的实现。
2. 啊手动阀

# 交互优化

## MD风格

MD风格即。有如下特征：

1. 啊打发