导读：Android面试中高频率出现的题都在这了。试题大部分从互联网收集，博主下了一番功夫进行梳理总结，难免有不足之处，还请见谅。这篇博客属于[Androi，你够了！！！](http://www.jianshu.com/p/9e5d3ad68607)专题中的一篇，其余文章会陆续发表，第一时间会发布在[本人Github](https://github.com/xinghongfei)上，敬请关注。这篇博客包括五个部分：热点，基础，进阶，性能优化，高级。后续会不断补充完善，希望能为小伙伴们找工作增加点自信😄。  
题库收集整理的周期比较长，很多原作者的链接没有了，非常抱歉。如果文章中用到了您的内容，请在下面留言，会把链接加进来，都是大牛，大家可以进来关注一下：  
[浪子潇涧](http://www.jianshu.com/users/83440ddd909c)的[面试总结](http://hujiaweibujidao.github.io/blog/2015/10/09/a-li-bao-bao-de-man-man-qiu-zhi-lu/)  
推荐：清华[浪子潇涧](http://www.jianshu.com/users/83440ddd909c)的面试总结－－[面试大礼包](https://github.com/hujiaweibujidao/AndroidInterviews)

**热点**

**如何保证Service不被杀死**

*Android 进程不死从3个层面入手：*

* A.提供进程优先级，降低进程被杀死的概率  
  方法一：监控手机锁屏解锁事件，在屏幕锁屏时启动1个像素的 Activity，在用户解锁时将 Activity 销毁掉。  
  方法二：启动前台service。  
  方法三：提升service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。
* B. 在进程被杀死后，进行拉活  
  方法一：注册高频率广播接收器，唤起进程。如网络变化，解锁屏幕，开机等  
  方法二：双进程相互唤起。  
  方法三：依靠系统唤起。  
  方法四：onDestroy方法里重启service：service +broadcast 方式，就是当service走ondestory的时候，发送一个自定义的广播，当收到广播的时候，重新启动service；
* C. 依靠第三方  
  根据终端不同，在小米手机（包括 MIUI）接入小米推送、华为手机接入华为推送；其他手机可以考虑接入腾讯信鸽或极光推送与小米推送做 A/B Test。

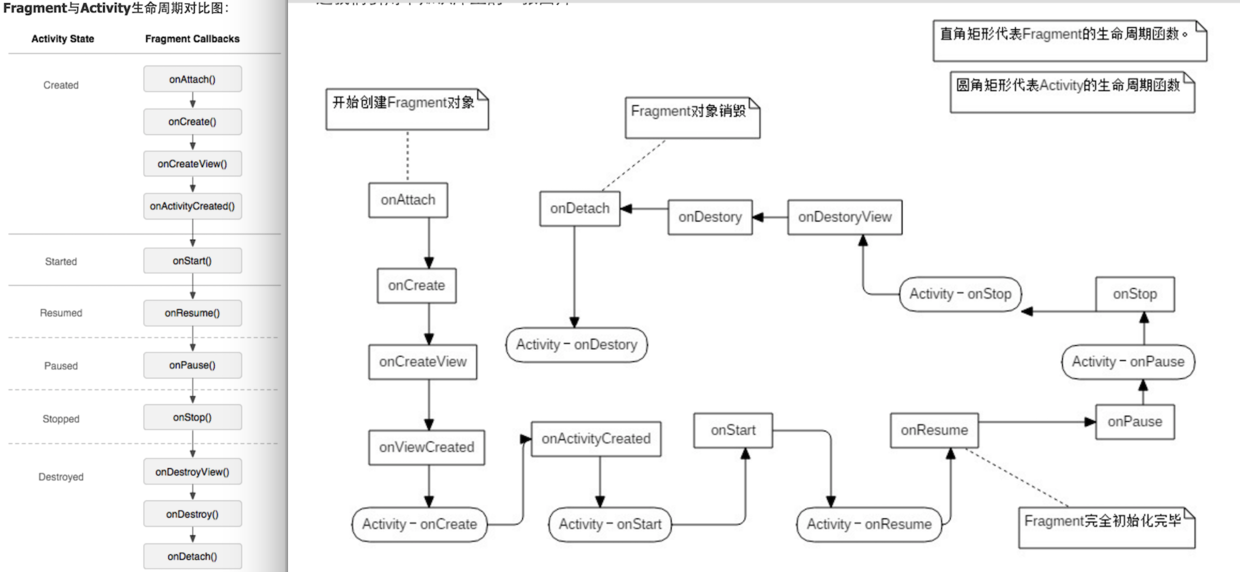
参考博客：[Android 进程保活招式大全](http://mp.weixin.qq.com/s?__biz=MzA3NTYzODYzMg==&mid=2653577617&idx=1&sn=623256a2ff94641036a6c9eea17baab8&scene=0#wechat_redirect)

**ButterKnife原理**  
ButterKnife对性能的影响很小，因为没有使用使用反射，而是使用的Annotation Processing Tool(APT),注解处理器，javac中用于编译时扫描和解析Java注解的工具。在编译阶段执行的，它的原理就是读入Java源代码，解析注解，然后生成新的Java代码。新生成的Java代码最后被编译成Java字节码，注解解析器不能改变读入的Java 类，比如不能加入或删除Java方法。

参考资料:  
[最新ButterKnife框架原理](http://www.trinea.cn/android/java-annotation-android-open-source-analysis/)  
[Java Annotation 及几个常用开源项目注解原理简析](http://www.tuicool.com/articles/QB73eyf)  
[ButterKnife 简单原理实现](http://www.itnose.net/detail/6291539.html)

**基础**

**Activity和Fragment生命周期有哪些？**



屏幕快照 2016-09-19 下午8.10.27.png

**横竖屏切换时候Activity的生命周期**

不设置Activity的android:configChanges时，切屏会重新回掉各个生命周期，切横屏时会执行一次，切竖屏时会执行两次  
设置Activity的android:configChanges=”orientation”时，切屏还是会调用各个生命周期，切换横竖屏只会执行一次  
设置Activity的android:configChanges=”orientation |keyboardHidden”时，切屏不会重新调用各个生命周期，只会执行onConfigurationChanged方法

**AsyncTask**  
[源码解析](http://blog.csdn.net/singwhatiwanna/article/details/17596225)  
*AsyncTask的缺陷和问题*  
关于线程池：asynctask对应的线程池ThreadPoolExecutor都是进程范围内共享的，都是static的，所以是asynctask控制着进程范围内所有的子类实例。由于这个限制的存在，当使用默认线程池时，如果线程数超过线程池的最大容量，线程池就会爆掉(3.0后默认串行执行，不会出现这个问题)。针对这种情况，可以尝试自定义线程池，配合asynctask使用。

关于默认线程池：核心线程池中最多有CPU\_COUNT+1个，最多有CPU\_COUNT\*2+1个，线程等待队列的最大等待数为128，但是可以自定义线程池。线程池是由AsyncTask来管理的，线程池允许tasks并行运行，需要注意的是并发情况下数据的一致性问题，新数据可能会被老数据覆盖掉，类似volatile变量。所以希望tasks能够串行运行的话，使用SERIAL\_EXECUTOR。

*AsyncTask在不同的SDK版本中的区别：*  
调用AsyncTask的excute方法不能立即执行程序的原因分析及改善方案  
通过查阅官方文档发现，AsyncTask首次引入时，异步任务是在一个独立的线程中顺序的执行，也就是说一次只能执行一个任务，不能并行的执行，从1.6开始，AsyncTask引入了线程池，支持同时执行5个异步任务，也就是说同时只能有5个线程运行，超过的线程只能等待，等待前面的线程某个执行完了才被调度和运行。换句话说，如果一个进程中的AsyncTask实例个数超过5个，那么假如前5个都运行很长时间的话，那么第6个只能等待机会了。这是AsyncTask的一个限制，而且对于2.3以前的版本无法解决。如果你的应用需要大量的后台线程去执行任务，那么你只能放弃使用AsyncTask，自己创建线程池来管理Thread。不得不说，虽然AsyncTask较Thread使用起来方便，但是它最多只能同时运行5个线程，这也大大局限了它的实力，你必须要小心设计你的应用，错开使用AsyncTask的时间，尽力做到分时，或者保证数量不会大于5个，否则就会遇到上次提到的问题。可能是Google意识到了AsyncTask的局限性了，从Android3.0开始对AsyncTask的API作出了一些调整：每次只启动一个线程执行一个任务，完成之后再执行第二个任务，也就是相当于只有一个后台线程在执行所提交的任务。

* 1.生命周期  
  很多开发者会认为一个在Activity中创建的AsyncTask会随着Activity的销毁而销毁。然而事实并非如此。AsyncTask会一直执行，直到doInBackground()方法执行完毕。然后，如果cancel(boolean)被调用,那么onCancelled(Result result)方法会被执行；否则，执行onPostExecute(Result result)方法。如果我们的Activity销毁之前，没有取消AsyncTask，这有可能让我们的AsyncTask崩溃(crash)。因为它想要处理的view已经不在了。所以，我们总是必须确保在销毁活动之前取消任务。总之，我们使用AsyncTask需要确保AsyncTask正确的取消。
* 2.内存泄漏  
  如果AsyncTask被声明为Activity的非静态的内部类，那么AsyncTask会保留一个对Activity的引用。如果Activity已经被销毁，AsyncTask的后台线程还在执行，它将继续在内存里保留这个引用，导致Activity无法被回收，引起内存泄漏。
* 3.结果丢失  
  屏幕旋转或Activity在后台被系统杀掉等情况会导致Activity的重新创建，之前运行的AsyncTask会持有一个之前Activity的引用，这个引用已经无效，这时调用onPostExecute()再去更新界面将不再生效。
* 4.并行还是串行  
  在Android1.6之前的版本，AsyncTask是串行的，在1.6至2.3的版本，改成了并行的。在2.3之后的版本又做了 修改，可以支持并行和串行，当想要串行执行时，直接执行execute()方法，如果需要执行executeOnExecutor(Executor)。

**Acitivty的任务栈**

使用android:launchMode="standard|singleInstance|single Task|singleTop"来控制Acivity任务栈。

任务栈是一种后进先出的结构。位于栈顶的Activity处于焦点状态,当按下back按钮的时候,栈内的Activity会一个一个的出栈,并且调用其onDestory()方法。如果栈内没有Activity,那么系统就会回收这个栈,每个APP默认只有一个栈,以APP的包名来命名.

* standard : 标准模式,每次启动Activity都会创建一个新的Activity实例,并且将其压入任务栈栈顶,而不管这个Activity是否已经存在。Activity的启动三回调(onCreate()->onStart()->onResume())都会执行。
* singleTop : 栈顶复用模式.这种模式下,如果新Activity已经位于任务栈的栈顶,那么此Activity不会被重新创建,所以它的启动三回调就不会执行,同时Activity的onNewIntent()方法会被回调.如果Activity已经存在但是不在栈顶,那么作用于standard模式一样.
* singleTask: 栈内复用模式.创建这样的Activity的时候,系统会先确认它所需任务栈已经创建,否则先创建任务栈.然后放入Activity,如果栈中已经有一个Activity实例,那么这个Activity就会被调到栈顶,onNewIntent(),并且singleTask会清理在当前Activity上面的所有Activity.(clear top)
* singleInstance : 加强版的singleTask模式,这种模式的Activity只能单独位于一个任务栈内,由于栈内复用的特性,后续请求均不会创建新的Activity,除非这个独特的任务栈被系统销毁了  
  Activity的堆栈管理以ActivityRecord为单位,所有的ActivityRecord都放在一个List里面.可以认为一个ActivityRecord就是一个Activity栈

**onSaveInstanceState() 与onRestoreIntanceState()**  
用户或者程序员主动去销毁一个Activity的时候不会掉用，其他情况都会调动，来保存界面信息。如代码中finish（）或用户按下back，不会掉用。

**android中进程的优先级？**

* 前台进程：即与用户正在交互的Activity或者Activity用到的Service等，如果系统内存不足时前台进程是最后被杀死的
* 可见进程：可以是处于暂停状态(onPause)的Activity或者绑定在其上的Service，即被用户可见，但由于失去了焦点而不能与用户交互
* 服务进程：其中运行着使用startService方法启动的Service，虽然不被用户可见，但是却是用户关心的，例如用户正在非音乐界面听的音乐或者正在非下载页面自己下载的文件等；当系统要空间运行前两者进程时才会被终止
* 后台进程：其中运行着执行onStop方法而停止的程序，但是却不是用户当前关心的，例如后台挂着的QQ，这样的进程系统一旦没了有内存就首先被杀死
* 空进程：不包含任何应用程序的程序组件的进程，这样的进程系统是一般不会让他存在的

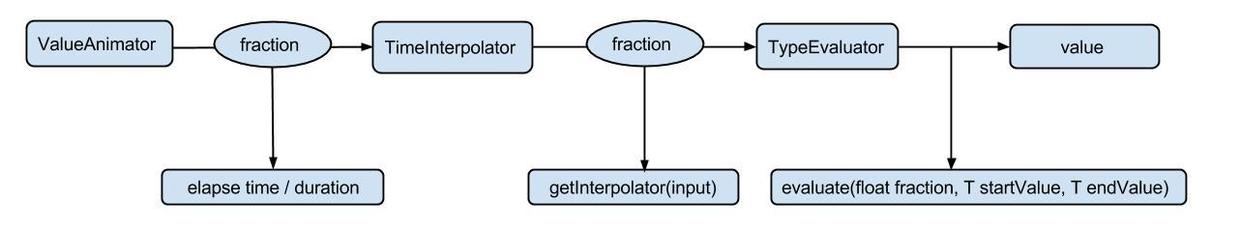
**Serializable和Parcelable**  
序列化，表示将一个对象转换成可存储或可传输的状态。序列化后的对象可以在网络上进行传输，也可以存储到本地。

* Serializable（Java自带）：  
  Serializable是序列化的意思，表示将一个对象转换成可存储或可传输的状态。序列化后的对象可以在网络上进行传输，也可以存储到本地。
* Parcelable（android 专用）：  
  除了Serializable之外，使用Parcelable也可以实现相同的效果，  
  不过不同于将对象进行序列化，Parcelable方式的实现原理是将一个完整的对象进行分解，  
  而分解后的每一部分都是Intent所支持的数据类型，这样也就实现传递对象的功能了。

**动画**

* tween 补间动画。通过指定View的初末状态和变化时间、方式，对View的内容完成一系列的图形变换来实现动画效果。 Alpha, Scale ,Translate, Rotate。
* frame 帧动画 AnimationDrawable 控制 animation-list xml布局
* PropertyAnimation 属性动画 3.0引入，属性动画核心思想是对值的变化。

**属性动画**[**详解**](http://a.codekk.com/detail/Android/lightSky/%E5%85%AC%E5%85%B1%E6%8A%80%E6%9C%AF%E7%82%B9%E4%B9%8B%20Android%20%E5%8A%A8%E7%94%BB%E5%9F%BA%E7%A1%80)  
Property Animation 动画有两个步聚：  
1.计算属性值  
2.为目标对象的属性设置属性值，即应用和刷新动画



valuecaculate.png

计算属性分为3个过程：

* 过程一：计算已完成动画分数 elapsed fraction 为了执行一个动画，你需要创建一个 ValueAnimator，并且指定目标对象属性的开始、结束值和持续时间。在调用 start 后的整个动画过程中， ValueAnimator 会根据已经完成的动画时间计算得到一个 0 到 1 之间的分数，代表该动画的已完成动画百分比。0 表示 0%，1 表示 100%。
* 过程二：计算插值（动画变化率）interpolated fraction 当 ValueAnimator 计算完已完成动画分数后，它会调用当前设置的 TimeInterpolator，去计算得到一个 interpolated（插值）分数，在计算过程中，已完成动画百分比会被加入到新的插值计算中。
* 过程三：计算属性值 当插值分数计算完成后，ValueAnimator 会根据插值分数调用合适的 TypeEvaluator 去计算运动中的属性值。  
  以上分析引入了两个概念：已完成动画分数（elapsed fraction）、插值分数( interpolated fraction )。

**Android的数据存储形式**

* SQLite：SQLite是一个轻量级的数据库，支持基本的SQL语法，是常被采用的一种数据存储方式。 Android为此数据库提供了一个名为SQLiteDatabase的类，封装了一些操作数据库的api
* SharedPreference： 以键值对的形势储存。其本质就是一个xml文件，常用于存储较简单的参数设置。
* File： 即常说的文件（I/O）存储方法，常用语存储大数量的数据，但是缺点是更新数据将是一件困难的事情。
* ContentProvider: Android系统中能实现所有应用程序共享的一种数据存储方式，由于数据通常在各应用间的是互相私密的，所以此存储方式较少使用，但是其又是必不可少的一种存储方式。例如音频，视频，图片和通讯录，一般都可以采用此种方式进行存储。每个Content Provider都会对外提供一个公共的URI（包装成Uri对象），如果应用程序有数据需要共享时，就需要使用Content Provider为这些数据定义一个URI，然后其他的应用程序就通过Content Provider传入这个URI来对数据进行操作。

**Context相关**

Activity和Service以及Application的Context是不一样的,Activity继承自ContextThemeWraper.其他的继承自ContextWrapper.  
每一个Activity和Service以及Application的Context都是一个新的ContextImpl对象  
getApplication()用来获取Application实例的，但是这个方法只有在Activity和Service中才能调用的到。那么也许在绝大多数情况下我们都是在Activity或者Service中使用Application的，但是如果在一些其它的场景，比如BroadcastReceiver中也想获得Application的实例，这时就可以借助getApplicationContext()方法，getApplicationContext()比getApplication()方法的作用域会更广一些，任何一个Context的实例，只要调用getApplicationContext()方法都可以拿到我们的Application对象。  
创建Toast和对话框不可以用Application 的context，只能用Activity的context。  
Context的数量等于Activity的个数 + Service的个数 + 1，这个1为Application

**Android各版本新特性**  
*Android5.0新特性*

* MaterialDesign设计风格
* 支持多种设备
* 支持64位ART虚拟机

*Android6.0新特性*

* 大量漂亮流畅的动画
* 支持快速充电的切换
* 支持文件夹拖拽应用
* 相机新增专业模式

**Android7.0新特性**

* 分屏多任务
* 增强的Java8语言模式
* 夜间模式

**Json**

* JSON的全程是JavaScript Object Notation，也就是JavaScript 对象表示法
* JSON是存储和交换文本信息的语法，类似XML，但是比XML更小、更快，更易解析
* JSON是轻量级的文本数据交换格式，独立于语言，具有自我描述性，更易理解

对象可以包含多个名称/值对，比如：

{"name":"zhangsan" , "age":25}

使用谷歌的GSON包进行解析  
在 Android Studio 里引入依赖：

compile 'com.google.code.gson:gson:2.7'

*值得注意的是实体类中变量名称必须和json中的值名相同。*  
json1的解析  
我们这里的实体类是Student.class

Gson gson = new Gson();

Student student = gson.fromJson(json1, Student.class);

json2的解析  
我们可以解析成int数组，也可以解析成Integer的List。  
解析成数组：

Gson gson = new Gson();

int[] ages = gson.fromJson(json2, int[].class);

解析成List：

Gson gson = new Gson();

List<Integer> ages = gson.fromJson(json2, new TypeToken<List<Integer>>(){}.getType);

json3的解析  
同样可以解析成List或者数组，我们就直接解析成List.

Gson gson = new Gson();

List<Student> students = gson.fromJson(json3, new TypeToke<List<Student>>(){}.getType);

**android中有哪几种解析xml的类,官方推荐哪种？以及它们的原理和区别**

* DOM解析

优点:  
1.XML树在内存中完整存储,因此可以直接修改其数据和结构.  
2.可以通过该解析器随时访问XML树中的任何一个节点.  
3.DOM解析器的API在使用上也相对比较简单.  
缺点:如果XML文档体积比较大时,将文档读入内存是非常消耗系统资源的.  
使用场景:DOM 是用与平台和语言无关的方式表示 XML 文档的官方 W3C 标准.DOM 是以层次结构组织的节点的集合.这个层次结构允许开发人员在树中寻找特定信息.分析该结构通常需要加载整个文档和构造层次结构,然后才能进行任何工作.DOM是基于对象层次结构的.

* SAX解析

优点:  
SAX 对内存的要求比较低,因为它让开发人员自己来决定所要处理的标签.特别是当开发人员只需要处理文档中所包含的部分数据时,SAX 这种扩展能力得到了更好的体现.  
缺点:  
用SAX方式进行XML解析时,需要顺序执行,所以很难访问到同一文档中的不同数据.此外,在基于该方式的解析编码过程也相对复杂.  
使用场景:  
对于含有数据量十分巨大,而又不用对文档的所有数据进行遍历或者分析的时候,使用该方法十分有效.该方法不用将整个文档读入内存,而只需读取到程序所需的文档标签处即可.

* Xmlpull解析

android SDK提供了xmlpull api,xmlpull和sax类似,是基于流（stream）操作文件,然后根据节点事件回调开发者编写的处理程序.因为是基于流的处理,因此xmlpull和sax都比较节约内存资源,不会象dom那样要把所有节点以对橡树的形式展现在内存中.xmlpull比sax更简明,而且不需要扫描完整个流.

**Jar和Aar的区别**

Jar包里面只有代码，aar里面不光有代码还包括代码还包括资源文件，比如 drawable 文件，xml 资源文件。对于一些不常变动的 Android Library，我们可以直接引用 aar，加快编译速度

**什么是三级缓存**  
（研究中）

**三级缓存原理**  
（研究中）

**Android为每个应用程序分配的内存大小是多少**  
android程序内存一般限制在16M，也有的是24M

**更新UI方式**  
Activity.runOnUiThread(Runnable)  
View.post(Runnable),View.postDelay(Runnable,long)  
Handler  
AsyncTask

**进阶**

**引起内存泄漏的情况**

* 对于使用了BraodcastReceiver，ContentObserver，File，游标 Cursor，Stream，Bitmap等资源的使用，应该在Activity销毁时及时关闭或者注销。
* 静态内部类持有外部成员变量（或context）:可以使用弱引用或使用ApplicationContext。
* 内部类持有外部类引用,异步任务中，持有外部成员变量。
* 集合中没用的对象没有及时remove。
* 不用的对象及时释放，如使用完Bitmap后掉用recycle（），再赋null。
* handler引起的内存泄漏，MessageQueue里的消息如果在activity销毁时没有处理完，就会引起内存的泄漏，可以使用弱引用解决。
* 设置过的监听不用时，及时移除。如在Destroy时及时remove。尤其以addListener开头的，在Destroy中都需要remove。
* activity泄漏可以使用LeakCanary。

[Android 内存溢出解决方案（OOM） 整理总结](http://mzh3344258.blog.51cto.com/1823534/804237)

* 在内存引用上做些处理，常用的有软引用、弱引用
* 在内存中加载图片时直接在内存中作处理，如：边界压缩
* 动态回收内存
* 优化Dalvik虚拟机的堆内存分配
* 自定义堆内存大小

**Activity/Window/View三者的差别,fragment的特点**

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。

在Activity中调用attach，创建了一个Window  
创建的window是其子类PhoneWindow，在attach中创建PhoneWindow  
在Activity中调用setContentView(R.layout.xxx)  
其中实际上是调用的getWindow().setContentView()  
调用PhoneWindow中的setContentView方法  
创建ParentView：  作为ViewGroup的子类，实际是创建的DecorView(作为FramLayout的子类）  
将指定的R.layout.xxx进行填充 通过布局填充器进行填充【其中的parent指的就是DecorView】  
调用到ViewGroup  
调用ViewGroup的removeAllView()，先将所有的view移除掉  
添加新的view：addView()

**Fragment 特点**  
Fragment可以作为Activity界面的一部分组成出现；  
可以在一个Activity中同时出现多个Fragment，并且一个Fragment也可以在多个Activity中使用；  
在Activity运行过程中，可以添加、移除或者替换Fragment；  
Fragment可以响应自己的输入事件，并且有自己的生命周期，它们的生命周期会受宿主Activity的生命周期影响。

**JVM 和Dalvik虚拟机的区别**

* JVM:  
  .java -> javac -> .class -> jar -> .jar  
  架构: 堆和栈的架构.
* DVM:  
  .java -> javac -> .class -> dx.bat -> .dex  
  架构: 寄存器(cpu上的一块高速缓存)

**怎么考虑数据传输的安全性**  
如果应用对传输的数据没有任何安全措施，攻击者设置的钓鱼网络中更改DNS服务器。这台服务器可以获取用户信息，或充当中间人与原服务器交换数据。在SSL/TLS通信中，客户端通过数字证书判断服务器是否可信，并采用证书的公钥与服务器进行加密通信。

**自定义View的相关方法**  
1.自定义属性  
2.onLayout(Viewgroup)  
3.onMesure  
4.onDraw  
5.交互:  
onIntercepterTouchEvent()  
onTouchEvent()

**事件传递机制** [详解](http://a.codekk.com/detail/Android/Trinea/%E5%85%AC%E5%85%B1%E6%8A%80%E6%9C%AF%E7%82%B9%E4%B9%8B%20View%20%E4%BA%8B%E4%BB%B6%E4%BC%A0%E9%80%92)  
当手指触摸到屏幕时，系统就会调用相应View的onTouchEvent，并传入一系列的action。

dispatchTouchEvent的执行顺序为：

首先触发ACTIVITY的dispatchTouchEvent,然后触发ACTIVITY的onInterceptTouchEvent.  
然后触发LAYOUT的dispatchTouchEvent，然后触发LAYOUT的onInterceptTouchEvent  
这就解释了重写ViewGroup时必须调用super.dispatchTouchEvent();

(1)dispatchTouchEvent:

此方法一般用于初步处理事件，因为动作是由此分发，所以通常会调用super.dispatchTouchEvent。这样就会继续调用onInterceptTouchEvent，再由onInterceptTouchEvent决定事件流向。

(2)onInterceptTouchEvent:

若返回值为true事件会传递到自己的onTouchEvent();若返回值为false传递到下一个View的dispatchTouchEvent();

(3)onTouchEvent():

若返回值为true，事件由自己消耗，后续动作让其处理；若返回值为false，自己不消耗事件了，向上返回让其他的父View的onTouchEvent接受处理

三大方法关系的伪代码：如果当前View拦截事件，就交给自己的onTouchEvent去处理，否则就丢给子View继续走相同的流程。

public boolean dispatchTouchEvent(MotionEvent ev)

{

boolean consume = false;

if(onInterceptTouchEvent(ev))

{

consume = onTouchEvent(ev);

}

else

{

consume = child.dispatchTouchEvent(ev);

}

return consume;

}

onTouchEvent的传递：

当有多个层级的View时，在父层级允许的情况下，这个action会一直传递直到遇到最深层的View。所以touch事件最先调用的是最底层View的onTouchEvent，如果View的onTouchEvent接收到某个touch action并做了相应处理，最后有两种返回方式return true和return false；return true会告诉系统当前的View需要处理这次的touch事件，以后的系统发出的ACTION\_MOVE,ACTION\_UP还是需要继续监听并接收的，并且这次的action已经被处理掉了，父层的View是不可能触发onTouchEvent的了。所以每一个action最多只能有一个onTouchEvent接口返回true。如果返回false，便会通知系统，当前View不关心这一次的touch事件，此时这个action会传向父级，调用父级View的onTouchEvent。但是这一次的touch事件之后发出任何action，该View都不在接受，onTouchEvent在这一次的touch事件中再也不会触发，也就是说一旦View返回false，那么之后的ACTION\_MOVE,ACTION\_UP等ACTION就不会在传入这个View,但是下一次touch事件的action还是会传进来的。

父层的onInterceptTouchEvent

前面说了底层的View能够接收到这次的事件有一个前提条件：在父层允许的情况下。假设不改变父层级的dispatch方法，在系统调用底层onTouchEvent之前会调用父View的onInterceptTouchEvent方法判断，父层View是否要截获本次touch事件之后的action。如果onInterceptTouchEvent返回了true，那么本次touch事件之后的所有action都不会向深层的View传递，统统都会传给父层View的onTouchEvent，就是说父层已经截获了这次touch事件，之后的action也不必询问onInterceptTouchEvent，在这次的touch事件之后发出的action时onInterceptTouchEvent不会再被调用，直到下一次touch事件的来临。如果onInterceptTouchEvent返回false，那么本次action将发送给更深层的View，并且之后的每一次action都会询问父层的onInterceptTouchEvent需不需要截获本次touch事件。只有ViewGroup才有onInterceptTouchEvent方法，因为一个普通的View肯定是位于最深层的View，touch能够传到这里已经是最后一站了，肯定会调用View的onTouchEvent()。

底层View的getParent().requestDisallowInterceptTouchEvent(true)

对于底层的View来说，有一种方法可以阻止父层的View获取touch事件，就是调用getParent().requestDisallowInterceptTouchEvent(true)方法。一旦底层View收到touch的action后调用这个方法那么父层View就不会再调用onInterceptTouchEvent了，也无法截获以后的action（如果父层ViewGroup和最底层View需要截获不同焦点，或不同手势的touch，不能使用这个写死）。

**ART和Dalvik区别**

art上应用启动快，运行快，但是耗费更多存储空间，安装时间长，总的来说ART的功效就是”空间换时间”。

ART: Ahead of Time  
Dalvik: Just in Time

什么是Dalvik：Dalvik是Google公司自己设计用于Android平台的Java虚拟机。Dalvik虚拟机是Google等厂商合作开发的Android移动设备平台的核心组成部分之一，它可以支持已转换为.dex(即Dalvik Executable)格式的Java应用程序的运行，.dex格式是专为Dalvik应用设计的一种压缩格式，适合内存和处理器速度有限的系统。Dalvik经过优化，允许在有限的内存中同时运行多个虚拟机的实例，并且每一个Dalvik应用作为独立的Linux进程执行。独立的进程可以防止在虚拟机崩溃的时候所有程序都被关闭。

什么是ART:Android操作系统已经成熟，Google的Android团队开始将注意力转向一些底层组件，其中之一是负责应用程序运行的Dalvik运行时。Google开发者已经花了两年时间开发更快执行效率更高更省电的替代ART运行时。ART代表Android Runtime,其处理应用程序执行的方式完全不同于Dalvik，Dalvik是依靠一个Just-In-Time(JIT)编译器去解释字节码。开发者编译后的应用代码需要通过一个解释器在用户的设备上运行，这一机制并不高效，但让应用能更容易在不同硬件和架构上运行。ART则完全改变了这套做法，在应用安装的时候就预编译字节码到机器语言，这一机制叫Ahead-Of-Time(AOT)编译。在移除解释代码这一过程后，应用程序执行将更有效率，启动更快。

ART优点：

系统性能的显著提升  
应用启动更快、运行更快、体验更流畅、触感反馈更及时。  
更长的电池续航能力  
支持更低的硬件

ART缺点：  
更大的存储空间占用，可能会增加10%-20%  
更长的应用安装时间

**Scroller原理**

Scroller执行流程里面的三个核心方法

mScroller.startScroll()  
mScroller.computeScrollOffset()  
view.computeScroll()  
1、在mScroller.startScroll()中为滑动做了一些初始化准备，比如：起始坐标，滑动的距离和方向以及持续时间(有默认值)，动画开始时间等。

2、mScroller.computeScrollOffset()方法主要是根据当前已经消逝的时间来计算当前的坐标点。因为在mScroller.startScroll()中设置了动画时间，那么在computeScrollOffset()方法中依据已经消逝的时间就很容易得到当前时刻应该所处的位置并将其保存在变量mCurrX和mCurrY中。除此之外该方法还可判断动画是否已经结束。

[**Android中Java和JavaScript交互**](http://www.imooc.com/article/1475)

webView.addJavaScriptInterface(new Object(){xxx}, "xxx");

1

答案：可以使用WebView控件执行JavaScript脚本，并且可以在JavaScript中执行Java代码。要想让WebView控件执行JavaScript，需要调用WebSettings.setJavaScriptEnabled方法，代码如下：

WebView webView = (WebView)findViewById(R.id.webview);

WebSettings webSettings = webView.getSettings();

//设置WebView支持JavaScript

webSettings.setJavaScriptEnabled(true);

webView.setWebChromeClient(new WebChromeClient());

JavaScript调用Java方法需要使用WebView.addJavascriptInterface方法设置JavaScript调用的Java方法，代码如下：

webView.addJavascriptInterface(new Object()

{

//JavaScript调用的方法

public String process(String value)

{

//处理代码

return result;

}

}, "demo"); //demo是Java对象映射到JavaScript中的对象名

可以使用下面的JavaScript代码调用process方法，代码如下：

<script language="javascript">

function search()

{

//调用searchWord方法

result.innerHTML = "<font color='red'>" + window.demo.process('data') + "</font>";

}

**SurfaceView和View的最本质的区别**

SurfaceView是在一个新起的单独线程中可以重新绘制画面，而view必须在UI的主线程中更新画面。

在UI的主线程中更新画面可能会引发问题，比如你更新的时间过长，那么你的主UI线程就会被你正在画的函数阻塞。那么将无法响应按键、触屏等消息。当使用SurfaceView由于是在新的线程中更新画面所以不会阻塞你的UI主线程。但这也带来了另外一个问题，就是事件同步。比如你触屏了一下，你需要SurfaceView中thread处理，一般就需要有一个event queue的设计来保存touchevent，这会稍稍复杂一点，因为涉及到线程安全。  
**常用的设计模式**  
单例，适配器，策略模式等常用的设计模式需要掌握

单例

public class Singleton{

private volatile static Singleton mSingleton;

private Singleton(){

}

public static Singleton getInstance(){

if(mSingleton == null){\\A

synchronized(Singleton.class){\\C

if(mSingleton == null)

mSingleton = new Singleton();\\B

}

}

return mSingleton;

}

}

**ANR排错**

1、ANR排错一般有三种情况

* KeyDispatchTimeout(5 seconds) –主要类型按键或触摸事件在特定时间内无响应
* BroadcastTimeout(10 secends) –BroadcastReceiver在特定时间内无法处理完成
* ServiceTimeout(20 secends) –小概率事件 Service在特定的时间内无法处理完成  
  *发生原因*  
  主线程被IO操作（从4.0之后网络IO不允许在主线程中）阻塞。  
  主线程中存在耗时的计算  
  主线程中错误的操作，比如Thread.wait或者Thread.sleep等 Android系统会监控程序的响应状况  
  使用AsyncTask处理耗时IO操作。

2、如何避免

UI线程尽量只做跟UI相关的工作  
耗时的操作(比如数据库操作，I/O,连接网络或者别的有可能阻塞UI线程的操作)把它放在单独的线程处理  
尽量用Handler来处理UIthread和别的thread之间的交互.  
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处理工作线程结果，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，建议使用IntentService处理。

3、ANR定位和修正  
如果开发机器上出现问题，我们可以通过查看/data/anr/traces.txt即可，最新的ANR信息在最开始部分。

**Android程序运行时权限与文件系统权限**  
1,Linux 文件系统权限。不同的用户对文件有不同的读写执行权限。在android系统中，system和应用程序是分开的，system里的数据是不可更改的。  
2，Android中有3种权限，进程权限UserID，签名，应用申明权限。每次安装时，系统根据包名为应用分配唯一的userID，不同的userID运行在不同的进程里，进程间的内存是独立的，不可以相互访问，除非通过特定的Binder机制。  
Android提供了如下的一种机制，可以使两个apk打破前面讲的这种壁垒。  
在AndroidManifest.xml中利用sharedUserId属性给不同的package分配相同的userID，通过这样做，两个package可以被当做同一个程序，系统会分配给两个程序相同的UserID。当然，基于安全考虑，两个package需要有相同的签名，否则没有验证也就没有意义了。

**如何让程序自动启动**  
定义一个Braodcastreceiver，action为BOOT——COMPLETE，接受到广播后启动程序。

[View绘制流程](http://a.codekk.com/detail/Android/lightSky/%E5%85%AC%E5%85%B1%E6%8A%80%E6%9C%AF%E7%82%B9%E4%B9%8B%20View%20%E7%BB%98%E5%88%B6%E6%B5%81%E7%A8%8B)

[**断电续传详解**](http://blog.csdn.net/wwj_748/article/details/20146869)

**Handler机制**  
andriod提供了Handler 和 Looper 来满足线程间的通信。Handler先进先出原则。Looper类用来管理特定线程内对象之间的消息交换(MessageExchange)。  
Looper: 一个线程可以产生一个Looper对象，由它来管理此线程里的MessageQueue(消息队列)。  
Handler: 你可以构造Handler对象来与Looper沟通，以便push新消息到MessageQueue里;或者接收Looper从Message Queue取出)所送来的消息。  
Message Queue(消息队列):用来存放线程放入的消息。  
线程：UIthread 通常就是main thread，而Android启动程序时会替它建立一个MessageQueue。

**ListView卡顿原因**  
Adapter的getView方法里面convertView没有使用setTag和getTag方式；  
在getView方法里面ViewHolder初始化后的赋值或者是多个控件的显示状态和背景的显示没有优化好，抑或是里面含有复杂的计算和耗时操作；  
在getView方法里面 inflate的row 嵌套太深（布局过于复杂）或者是布局里面有大图片或者背景所致；  
Adapter多余或者不合理的notifySetDataChanged；  
listview 被多层嵌套，多次的onMessure导致卡顿，如果多层嵌套无法避免，建议把listview的高和宽设置为fill\_parent. 如果是代码继承的listview，那么也请你别忘记为你的继承类添加上LayoutPrams，注意高和宽都是fill\_parent的；

**启动一个程序，可以主界面点击图标进入，也可以从一个程序中跳转过去，二者有什么区别？**  
是因为启动程序（主界面也是一个app），发现了在这个程序中存在一个设置为

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

的activity,  
所以这个launcher会把icon提出来，放在主界面上。当用户点击icon的时候，发出一个Intent：

Intent intent = mActivity.getPackageManager().getLaunchIntentForPackage(packageName);

mActivity.startActivity(intent);

跳过去可以跳到任意允许的页面，如一个程序可以下载，那么真正下载的页面可能不是首页（也有可能是首页），这时还是构造一个Intent，startActivity.  
这个intent中的action可能有多种view,download都有可能。系统会根据第三方程序向系统注册的功能，为你的Intent选择可以打开的程序或者页面。所以唯一的一点  
不同的是从icon的点击启动的intent的action是相对单一的，从程序中跳转或者启动可能样式更多一些。本质是相同的。

**AIDL的全称是什么?如何工作?能处理哪些类型的数据?**  
AIDL全称Android Interface Definition Language（Android接口描述语言）是一种接口描述语言; 编译器可以通过aidl文件生成一段代码，通过预先定义的接口达到两个进程内部通信进程跨界访问对象的目的.AIDL的IPC的机制和COM或CORBA类似, 是基于接口的，但它是轻量级的。它使用代理类在客户端和实现层间传递值. 如果要使用AIDL, 需要完成2件事情: 1. 引入AIDL的相关类.; 2. 调用aidl产生的class.  
理论上, 参数可以传递基本数据类型和String, 还有就是Bundle的派生类, 不过在Eclipse中,目前的ADT不支持Bundle做为参数,  
具体实现步骤如下:  
1、创建AIDL文件, 在这个文件里面定义接口, 该接口定义了可供客户端访问的方法和属性。  
2、编译AIDL文件, 用Ant的话, 可能需要手动, 使用Eclipse plugin的话,可以根据adil文件自动生产java文件并编译, 不需要人为介入.  
3、在Java文件中, 实现AIDL中定义的接口. 编译器会根据AIDL接口, 产生一个JAVA接口。这个接口有一个名为Stub的内部抽象类，它继承扩展了接口并实现了远程调用需要的几个方法。接下来就需要自己去实现自定义的几个接口了.  
4、向客户端提供接口ITaskBinder, 如果写的是service，扩展该Service并重载onBind ()方法来返回一个实现上述接口的类的实例。  
5、在服务器端回调客户端的函数. 前提是当客户端获取的IBinder接口的时候,要去注册回调函数, 只有这样, 服务器端才知道该调用那些函数  
AIDL语法很简单,可以用来声明一个带一个或多个方法的接口，也可以传递参数和返回值。 由于远程调用的需要, 这些参数和返回值并不是任何类型.下面是些AIDL支持的数据类型:

1. 不需要import声明的简单Java编程语言类型(int,boolean等)
2. String, CharSequence不需要特殊声明
3. List, Map和Parcelables类型, 这些类型内所包含的数据成员也只能是简单数据类型, String等其他比支持的类型.  
   (另外: 我没尝试Parcelables, 在Eclipse+ADT下编译不过, 或许以后会有所支持).  
   实现接口时有几个原则:  
   1.抛出的异常不要返回给调用者. 跨进程抛异常处理是不可取的.  
   2.IPC调用是同步的。如果你知道一个IPC服务需要超过几毫秒的时间才能完成地话，你应该避免在Activity的主线程中调用。也就是IPC调用会挂起应用程序导致界面失去响应. 这种情况应该考虑单起一个线程来处理.  
   3.不能在AIDL接口中声明静态属性。  
   IPC的调用步骤:
4. 声明一个接口类型的变量，该接口类型在.aidl文件中定义。
5. 实现ServiceConnection。
6. 调用ApplicationContext.bindService(),并在ServiceConnection实现中进行传递.
7. 在ServiceConnection.onServiceConnected()实现中，你会接收一个IBinder实例(被调用的Service). 调用  
   YourInterfaceName.Stub.asInterface((IBinder)service)将参数转换为YourInterface类型。
8. 调用接口中定义的方法。你总要检测到DeadObjectException异常，该异常在连接断开时被抛出。它只会被远程方法抛出。
9. 断开连接，调用接口实例中的ApplicationContext.unbindService()  
   参考：<http://buaadallas.blog.51cto.com/399160/372090>

aidl主要就是帮助我们完成了包装数据和解包的过程，并调用了transact过程，而用来传递的数据包我们就称为parcel

AIDL: xxx.aidl->xxx.java,注册service

用aidl定义需要被调用方法接口  
实现这些方法  
调用这些方法

**FC(Force Close)**

*什么时候会出现*

* Error
* OOM，内存溢出
* StackOverFlowError
* Runtime,比如说空指针异常

*解决的办法*

* 注意内存的使用和管理
* 使用Thread.UncaughtExceptionHandler接口

**性能优化**

**界面优化**  
太多重叠的背景(overdraw)

这个问题其实最容易解决，建议就是检查你在布局和代码中设置的背景，有些背景是隐藏在底下的，它永远不可能显示出来，这种没必要的背景一定要移除，因为它很可能会严重影响到app的性能。如果采用的是selector的背景，将normal状态的color设置为”@android:color/transparent”,也同样可以解决问题。

太多重叠的View

第一个建议是 ：使用ViewStub来加载一些不常用的布局，它是一个轻量级且默认是不可见的视图，可以动态的加载一个布局，只要你用到这个重叠着的View的时候才加载，推迟加载的时间。

第二个建议是：如果使用了类似Viewpager＋Fragment这样的组合或者有多个Fragment在一个界面上，需要控制Fragment的显示和隐藏，尽量使用动态的Inflation view，它的性能要比SetVisibility好。

复杂的Layout层级

这里的建议比较多一些，首先推荐使用Android提供的布局工具Hierarchy Viewer来检查和优化布局。第一个建议是：如果嵌套的线性布局加深了布局层次，可以使用相对布局来取代。第二个建议是：用标签来合并布局。第三个建议是：用标签来重用布局，抽取通用的布局可以让布局的逻辑更清晰明了。记住，这些建议的最终目的都是使得你的Layout在Hierarchy Viewer里变得宽而浅，而不是窄而深。

总结：可以考虑多使用merge和include，ViewStub。尽量使布局浅平，根布局尽量少使用RelactivityLayout,因为RelactivityLayout每次需要测量2次。

**内存优化**  
核心思想：减少内存使用，能不new的不new，能少分配的少分配。因为分配更多的内存就意味着发生更多的GC，每次触发GC都会占用CPU时间，影响性能。

* 集合优化：Android提供了一系列优化过后的数据集合工具类，如SparseArray、SparseBooleanArray、LongSparseArray，使用这些API可以让我们的程序更加高效。HashMap工具类会相对比较低效，因为它需要为每一个键值对都提供一个对象入口，而SparseArray就避免掉了基本数据类型转换成对象数据类型的时间。
* Bitmap优化：读取一个Bitmap图片的时候，千万不要去加载不需要的分辨率。可以压缩图片等操作。
* 尽量避免使用依赖注入框架。
* 避免创作不必要的对象：字符串拼接使用StringBuffer，StringBuilder。
* onDraw方法里面不要执行对象的创建.
* 重写onTrimMemory，根据传入的参数，进行内存释放。
* 使用static final 优化成员变量。

**移动端获取网络数据优化的几个点**

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

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。

减少请求数据的大小：对于post请求，body可以做gzip压缩的，header也可以做数据压缩(不过只支持http 2.0)。  
返回数据的body也可以做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）  
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）

**高级**

**Android系统启动过程，App启动过程**  
]从桌面点击到activity启动的过程

1.Launcher线程捕获onclick的点击事件，调用Launcher.startActivitySafely,进一步调用Launcher.startActivity，最后调用父类Activity的startActivity。

2.Activity和ActivityManagerService交互，引入Instrumentation，将启动请求交给Instrumentation，调用Instrumentation.execStartActivity。

3.调用ActivityManagerService的startActivity方法，这里做了进程切换（具体过程请查看源码）。

4.开启Activity，调用onCreate方法

**热布丁**  
原因：因为一个dvm中存储方法id用的是short类型，导致dex中方法不能超过65536个  
原理：将编译好的class文件拆分打包成两个dex，绕过dex方法数量的限制以及安装时的检查，在运行时再动态加载第二个dex文件中。使用Dexclassloader。

**动态加载(也叫插件化技术)**  
动态加载主要解决3个技术问题：  
1，使用ClassLoader加载类。  
2，资源访问。  
3，生命周期管理。

参考  
1、[Android热补丁动态修复技术系列](http://blog.csdn.net/u010386612/article/details/50885320)   
2、[Android 利用 APT 技术在编译期生成代码](http://brucezz.itscoder.com/articles/2016/08/06/use-apt-in-android/)  
3、[Android中的动态加载机制](http://blog.csdn.net/jiangwei0910410003/article/details/17679823)   
4、[掌阅 Android App 插件补丁实践 (ZeusPlugin)](http://blog.adisonhyh.com/2016/08/20/ZeusPlugin-%E6%8E%8C%E9%98%85Android-App%E6%8F%92%E4%BB%B6%E8%A1%A5%E4%B8%81%E5%AE%9E%E8%B7%B5/)  
5、[插件化开发从入门到深入](http://www.androidblog.cn/index.php/Index/detail/id/16)  
6、[Android Hotfix 新方案——Amigo 源码解读（from diycode）](http://www.diycode.cc/topics/280) 【更新于2016年9月2号】  
7、[插件化由理论到实践](http://kymjs.com/column/plugin.html)【更新于2016年9月3日】  
8[Android插件化原理解析](http://weishu.me/2016/01/28/understand-plugin-framework-overview/)  
9 [Android动态加载技术三个关键问题详解](http://www.infoq.com/cn/articles/android-dynamic-loading)  
10 [Android apk动态加载机制的研究](http://blog.csdn.net/singwhatiwanna/article/details/22597587)

**Binder机制**

跨进程间通信(IPC):四大组件之间通过Intent互相跳转，Android实现IPC的方式是binder机制。

[android中的跨进程通信的实现（一）——远程调用过程和aidl](http://blog.csdn.net/notice520/article/details/8135600)  
[Android中的Binder机制的简要理解](http://www.linuxidc.com/Linux/2012-07/66195.htm)  
[Android中的Binder机制的简要理解二](http://blog.csdn.net/sunxingzhesunjinbiao/article/details/42195043)

**Android系统的架构**



屏幕快照 2016-09-07 上午9.26.57.png

android的系统架构和其操作系统一样，采用了分层的架构。从架构图看，android分为四个层，从高层到低层分别是应用程序层、应用程序框架层、系统运行库层和linux核心层。  
　　**1.应用程序**  
　　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。  
　　**2.应用程序框架**  
　　开发人员也可以完全访问核心应用程序所使用的API框架。该应用程序的架构设计简化了组件的重用;任何一个应用程序都可以发布它的功能块并且任何其它的应用程序都可以使用其所发布的功能块(不过得遵循框架的安全性限制)。同样，该应用程序重用机制也使用户可以方便的替换程序组件。  
　　隐藏在每个应用后面的是一系列的服务和系统, 其中包括;  
　　 *丰富而又可扩展的视图(Views)，可以用来构建应用程序， 它包括列表(lists)，网格(grids)，文本框(text boxes)，按钮(buttons)， 甚至可嵌入的web浏览器。* 内容提供器(Content Providers)使得应用程序可以访问另一个应用程序的数据(如联系人数据库)， 或者共享它们自己的数据  
　　 *资源管理器(Resource Manager)提供 非代码资源的访问，如本地字符串，图形，和布局文件( layout files )。* 通知管理器 (Notification Manager) 使得应用程序可以在状态栏中显示自定义的提示信息。  
　　\* 活动管理器( Activity Manager) 用来管理应用程序生命周期并提供常用的导航回退功能。  
　　有关更多的细节和怎样从头写一个应用程序，请参考 如何编写一个 Android 应用程序.

**3.系统运行库**  
　　1)程序库  
　　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：  
　　系统 C 库 - 一个从 BSD 继承来的标准 C 系统函数库( libc )， 它是专门为基于 embedded linux 的设备定制的。  
　　 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 。  
　　 Surface Manager - 对显示子系统的管理，并且为多个应用程序提 供了2D和3D图层的无缝融合。  
　　 LibWebCore - 一个最新的web浏览器引擎用，支持Android浏览器和一个可嵌入的web视图。  
　　SGL- 底层的2D图形引擎  
　　 3D libraries - 基于OpenGL ES 1.0 APIs实现;该库可以使用硬件 3D加速(如果可用)或者使用高度优化的3D软加速。  
　　 FreeType -位图(bitmap)和矢量(vector)字体显示。  
　　 *SQLite - 一个对于所有应用程序可用，功能强劲的轻型关系型数据库引擎。  
　　2)Android 运行库  
　　Android 包括了一个核心库，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。  
　　Dalvik虚拟机依赖于linux内核的一些功能，比如线程机制和底层内存管理机制。  
　　\**4.Linux 内核\*\*  
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。

文／Maat红飞（简书作者）  
原文链接：http://www.jianshu.com/p/89f19d67b348  
著作权归作者所有，转载请联系作者获得授权，并标注“简书作者”。