**Android中高级题**

**1、Activity生命周期？**

onCreate() -> onStart() -> onResume() -> onPause() -> onStop() -> onDetroy()

<https://blog.csdn.net/geekerhw/article/details/48749935?ref=myread>

横竖屏下的生命周期：

<https://blog.csdn.net/hzw19920329/article/details/51345971>

Activity和Fragment之间错综复杂的生命周期关系

<https://blog.csdn.net/u012702547/article/details/50253955>

<https://blog.csdn.net/binbin_1989/article/details/64437995>

[Activity上有Dialog的时候按Home键时的生命周期](https://blog.csdn.net/xiyoucode/article/details/79595815)

<https://blog.csdn.net/hanhan1016/article/details/47977489>

两个Activity 之间跳转时必然会执行的是哪几个方法？

前台切换到后台，然后再回到前台，Activity生命周期回调方法。弹出Dialog，生命值周期回调方法。

A启动B，A到前台，B到后台

A.onPasue ->这里要注意B的启动模式 B.onCreate -> B.onStart -> B.OnResume -> B全部遮挡A 调用 A.onStop 否则不调用

如果A长时间在后台被杀死

B.onPause -> A.onCreate -> A.onStart -> A.onResume -> B.onStop

如果没有被杀死

B.onPause  -> A.onRestart -> A.onStart -> A.onResume

**2、Service生命周期？**

service 启动方式有两种，一种是通过startService()方式进行启动，另一种是通过bindService()方式进行启动。不同的启动方式他们的生命周期是不一样.

通过startService()这种方式启动的service，生命周期是这样：调用startService() --> onCreate()--> onStartConmon()--> onDestroy()。这种方式启动的话，需要注意一下几个问题，第一：当我们通过startService被调用以后，多次在调用startService(),onCreate()方法也只会被调用一次，而onStartConmon()会被多次调用当我们调用stopService()的时候，onDestroy()就会被调用，从而销毁服务。第二：当我们通过startService启动时候，通过intent传值，在onStartConmon()方法中获取值的时候，一定要先判断intent是否为null。

通过bindService()方式进行绑定，这种方式绑定service，生命周期走法：bindService-->onCreate()-->onBind()-->unBind()-->onDestroy()  bingservice 这种方式进行启动service好处是更加便利activity中操作service，比如加入service中有几个方法，a,b ，如果要在activity中调用，在需要在activity获取ServiceConnection对象，通过ServiceConnection来获取service中内部类的类对象，然后通过这个类对象就可以调用类中的方法，当然这个类需要继承Binder对象

<https://www.cnblogs.com/huihuizhang/p/7623760.html>

**3、Activity的启动过程（不要回答生命周期）**

app启动的过程有两种情况，第一种是从桌面launcher上点击相应的应用图标，第二种是在activity中通过调用startActivity来启动一个新的activity。

我们创建一个新的项目，默认的根activity都是MainActivity，而所有的activity都是保存在堆栈中的，我们启动一个新的activity就会放在上一个activity上面，而我们从桌面点击应用图标的时候，由于launcher本身也是一个应用，当我们点击图标的时候，系统就会调用startActivitySately(),一般情况下，我们所启动的activity的相关信息都会保存在intent中，比如action，category等等。我们在安装这个应用的时候，系统也会启动一个PackaManagerService的管理服务，这个管理服务会对AndroidManifest.xml文件进行解析，从而得到应用程序中的相关信息，比如service，activity，Broadcast等等，然后获得相关组件的信息。当我们点击应用图标的时候，就会调用startActivitySately()方法，而这个方法内部则是调用startActivty(),而startActivity()方法最终还是会调用startActivityForResult()这个方法。而在startActivityForResult()这个方法。因为startActivityForResult()方法是有返回结果的，所以系统就直接给一个-1，就表示不需要结果返回了。而startActivityForResult()这个方法实际是通过Instrumentation类中的execStartActivity()方法来启动activity，Instrumentation这个类主要作用就是监控程序和系统之间的交互。而在这个execStartActivity()方法中会获取ActivityManagerService的代理对象，通过这个代理对象进行启动activity。启动会就会调用一个checkStartActivityResult()方法，如果说没有在配置清单中配置有这个组件，就会在这个方法中抛出异常了。当然最后是调用的是Application.scheduleLaunchActivity()进行启动activity，而这个方法中通过获取得到一个ActivityClientRecord对象，而这个ActivityClientRecord通过handler来进行消息的发送，系统内部会将每一个activity组件使用ActivityClientRecord对象来进行描述，而ActivityClientRecord对象中保存有一个LoaderApk对象，通过这个对象调用handleLaunchActivity来启动activity组件，而页面的生命周期方法也就是在这个方法中进行调用。

**4、Broadcast注册方式与区别**

此处延伸：什么情况下用动态注册

Broadcast广播，注册方式主要有两种.

第一种是静态注册，也可成为常驻型广播，这种广播需要在Androidmanifest.xml中进行注册，这中方式注册的广播，不受页面生命周期的影响，即使退出了页面，也可以收到广播这种广播一般用于想开机自启动啊等等，由于这种注册的方式的广播是常驻型广播，所以会占用CPU的资源。

第二种是动态注册，而动态注册的话，是在代码中注册的，这种注册方式也叫非常驻型广播，收到生命周期的影响，退出页面后，就不会收到广播，我们通常运用在更新UI方面。这种注册方式优先级较高。最后需要解绑，否会会内存泄露

广播是分为有序广播和无序广播。

**5、HttpClient与HttpUrlConnection的区别**

此处延伸：Volley里用的哪种请求方式（2.3前HttpClient，2.3后HttpUrlConnection）

首先HttpClient和HttpUrlConnection 这两种方式都支持Https协议，都是以流的形式进行上传或者下载数据，也可以说是以流的形式进行数据的传输，还有ipv6,以及连接池等功能。HttpClient这个拥有非常多的API，所以如果想要进行扩展的话，并且不破坏它的兼容性的话，很难进行扩展，也就是这个原因，Google在Android6.0的时候，直接就弃用了这个HttpClient.

而HttpUrlConnection相对来说就是比较轻量级了，API比较少，容易扩展，并且能够满足Android大部分的数据传输。比较经典的一个框架volley，在2.3版本以前都是使用HttpClient,在2.3以后就使用了HttpUrlConnection。

<https://www.cnblogs.com/liushuibufu/p/4140914.html>

<https://www.cnblogs.com/liushuibufu/p/4140914.html>

**6、java虚拟机和Dalvik虚拟机的区别**

<https://blog.csdn.net/rzwinters/article/details/77098138>

<https://www.cnblogs.com/lxjshuju/p/7191910.html>

Java虚拟机：

1、java虚拟机基于栈。基于栈的机器必须使用指令来载入和操作栈上数据，所需指令更多。

2、java虚拟机运行的是java字节码。（java类会被编译成一个或多个字节码.class文件）

Dalvik虚拟机：

1、dalvik虚拟机是基于寄存器的

2、Dalvik运行的是自定义的.dex字节码格式。（java类被编译成.class文件后，会通过一个dx工具将所有的.class文件转换成一个.dex文件，然后dalvik虚拟机会从其中读取指令和数据

3、常量池已被修改为只使用32位的索引，以简化解释器。

4、一个应用，一个虚拟机实例，一个进程（所有android应用的线程都是对应一个linux线程，都运行在自己的沙盒中，不同的应用在不同的进程中运行。每个android dalvik应用程序都被赋予了一个独立的linux PID(app\_\*)）

**7、进程保活（不死进程）**

进程的优先级是什么？

<https://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B%E4%BC%98%E5%85%88%E7%BA%A7/9729904?fr=aladdin>

<https://blog.csdn.net/mignatian/article/details/79738176>

<https://www.jianshu.com/p/63aafe3c12af>

<https://blog.csdn.net/qq_34664695/article/details/79756573>

当前业界的Android进程保活手段主要分为\*\* 黑、白、灰 \*\*三种，其大致的实现思路如下：

黑色保活：不同的app进程，用广播相互唤醒（包括利用系统提供的广播进行唤醒）

白色保活：启动前台Service

灰色保活：利用系统的漏洞启动前台Service

黑色保活

所谓黑色保活，就是利用不同的app进程使用广播来进行相互唤醒。举个3个比较常见的场景：

场景1：开机，网络切换、拍照、拍视频时候，利用系统产生的广播唤醒app

场景2：接入第三方SDK也会唤醒相应的app进程，如微信sdk会唤醒微信，支付宝sdk会唤醒支付宝。由此发散开去，就会直接触发了下面的 场景3

场景3：假如你手机里装了支付宝、淘宝、天猫、UC等阿里系的app，那么你打开任意一个阿里系的app后，有可能就顺便把其他阿里系的app给唤醒了。（只是拿阿里打个比方，其实BAT系都差不多）

白色保活

白色保活手段非常简单，就是调用系统api启动一个前台的Service进程，这样会在系统的通知栏生成一个Notification，用来让用户知道有这样一个app在运行着，哪怕当前的app退到了后台。如下方的LBE和QQ音乐这样：

灰色保活

灰色保活，这种保活手段是应用范围最广泛。它是利用系统的漏洞来启动一个前台的Service进程，与普通的启动方式区别在于，它不会在系统通知栏处出现一个Notification，看起来就如同运行着一个后台Service进程一样。这样做带来的好处就是，用户无法察觉到你运行着一个前台进程（因为看不到Notification）,但你的进程优先级又是高于普通后台进程的。那么如何利用系统的漏洞呢，大致的实现思路和代码如下：

思路一：API < 18，启动前台Service时直接传入new Notification()；

思路二：API >= 18，同时启动两个id相同的前台Service，然后再将后启动的Service做stop处理

熟悉Android系统的童鞋都知道，系统出于体验和性能上的考虑，app在退到后台时系统并不会真正的kill掉这个进程，而是将其缓存起来。打开的应用越多，后台缓存的进程也越多。在系统内存不足的情况下，系统开始依据自身的一套进程回收机制来判断要kill掉哪些进程，以腾出内存来供给需要的app。这套杀进程回收内存的机制就叫 Low Memory Killer ，它是基于Linux内核的 OOM Killer（Out-Of-Memory killer）机制诞生。

进程的重要性，划分5级：

前台进程 (Foreground process)

可见进程 (Visible process)

服务进程 (Service process)

后台进程 (Background process)

空进程 (Empty process)

了解完 Low Memory Killer，再科普一下oom\_adj。什么是oom\_adj？它是linux内核分配给每个系统进程的一个值，代表进程的优先级，进程回收机制就是根据这个优先级来决定是否进行回收。对于oom\_adj的作用，你只需要记住以下几点即可：

进程的oom\_adj越大，表示此进程优先级越低，越容易被杀回收；越小，表示进程优先级越高，越不容易被杀回收

普通app进程的oom\_adj>=0,系统进程的oom\_adj才可能<0

有些手机厂商把这些知名的app放入了自己的白名单中，保证了进程不死来提高用户体验（如微信、QQ、陌陌都在小米的白名单中）。如果从白名单中移除，他们终究还是和普通app一样躲避不了被杀的命运，为了尽量避免被杀，还是老老实实去做好优化工作吧。

所以，进程保活的根本方案终究还是回到了性能优化上，进程永生不死终究是个彻头彻尾的伪命题！

**8、讲解一下Context**

Context是一个抽象基类。在翻译为上下文，也可以理解为环境，是提供一些程序的运行环境基础信息。Context下有两个子类，ContextWrapper是上下文功能的封装类，而ContextImpl则是上下文功能的实现类。而ContextWrapper又有三个直接的子类， ContextThemeWrapper、Service和Application。其中，ContextThemeWrapper是一个带主题的封装类，而它有一个直接子类就是Activity，所以Activity和Service以及Application的Context是不一样的，只有Activity需要主题，Service不需要主题。Context一共有三种类型，分别是Application、Activity和Service。这三个类虽然分别各种承担着不同的作用，但它们都属于Context的一种，而它们具体Context的功能则是由ContextImpl类去实现的，因此在绝大多数场景下，Activity、Service和Application这三种类型的Context都是可以通用的。不过有几种场景比较特殊，比如启动Activity，还有弹出Dialog。出于安全原因的考虑，Android是不允许Activity或Dialog凭空出现的，一个Activity的启动必须要建立在另一个Activity的基础之上，也就是以此形成的返回栈。而Dialog则必须在一个Activity上面弹出（除非是System Alert类型的Dialog），因此在这种场景下，我们只能使用Activity类型的Context，否则将会出错。

getApplicationContext()和getApplication()方法得到的对象都是同一个application对象，只是对象的类型不一样。

Context数量 = Activity数量 + Service数量 + 1 （1为Application）

**9、理解Activity，View,Window三者关系**

这个问题真的很不好回答。所以这里先来个算是比较恰当的比喻来形容下它们的关系吧。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。

1：Activity构造的时候会初始化一个Window，准确的说是PhoneWindow。

2：这个PhoneWindow有一个“ViewRoot”，这个“ViewRoot”是一个View或者说ViewGroup，是最初始的根视图。

3：“ViewRoot”通过addView方法来一个个的添加View。比如TextView，Button等

4：这些View的事件监听，是由WindowManagerService来接受消息，并且回调Activity函数。比如onClickListener，onKeyDown等。

**10、四种LaunchMode及其使用场景**

<https://blog.csdn.net/shinay/article/details/7898492/>

此处延伸：栈(First In Last Out)与队列(First In First Out)的区别

栈与队列的区别：

1. 队列先进先出，栈先进后出

2. 对插入和删除操作的"限定"。 栈是限定只能在表的一端进行插入和删除操作的线性表。 队列是限定只能在表的一端进行插入和在另一端进行删除操作的线性表。

3. 遍历数据速度不同

standard 模式

这是默认模式，每次激活Activity时都会创建Activity实例，并放入任务栈中。使用场景：大多数Activity。

singleTop 模式

如果在任务的栈顶正好存在该Activity的实例，就重用该实例( 会调用实例的 onNewIntent() )，否则就会创建新的实例并放入栈顶，即使栈中已经存在该Activity的实例，只要不在栈顶，都会创建新的实例。使用场景如新闻类或者阅读类App的内容页面。

singleTask 模式

如果在栈中已经有该Activity的实例，就重用该实例(会调用实例的 onNewIntent() )。重用时，会让该实例回到栈顶，因此在它上面的实例将会被移出栈。如果栈中不存在该实例，将会创建新的实例放入栈中。使用场景如浏览器的主界面。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。

singleInstance 模式

在一个新栈中创建该Activity的实例，并让多个应用共享该栈中的该Activity实例。一旦该模式的Activity实例已经存在于某个栈中，任何应用再激活该Activity时都会重用该栈中的实例( 会调用实例的 onNewIntent() )。其效果相当于多个应用共享一个应用，不管谁激活该 Activity 都会进入同一个应用中。使用场景如闹铃提醒，将闹铃提醒与闹铃设置分离。singleInstance不要用于中间页面，如果用于中间页面，跳转会有问题，比如：A -> B (singleInstance) -> C，完全退出后，在此启动，首先打开的是B。

**11、View的绘制流程**

自定义控件：

1、组合控件。这种自定义控件不需要我们自己绘制，而是使用原生控件组合成的新控件。如标题栏。

2、继承原有的控件。这种自定义控件在原生控件提供的方法外，可以自己添加一些方法。如制作圆角，圆形图片。

3、完全自定义控件：这个View上所展现的内容全部都是我们自己绘制出来的。比如说制作水波纹进度条。

View的绘制流程：OnMeasure()——>OnLayout()——>OnDraw()

第一步：OnMeasure()：测量视图大小。从顶层父View到子View递归调用measure方法，measure方法又回调OnMeasure。

第二步：OnLayout()：确定View位置，进行页面布局。从顶层父View向子View的递归调用view.layout方法的过程，即父View根据上一步measure子View所得到的布局大小和布局参数，将子View放在合适的位置上。

第三步：OnDraw()：绘制视图。ViewRoot创建一个Canvas对象，然后调用OnDraw()。六个步骤：①、绘制视图的背景；②、保存画布的图层（Layer）；③、绘制View的内容；④、绘制View子视图，如果没有就不用；

⑤、还原图层（Layer）；⑥、绘制滚动条。

**12、View，ViewGroup事件分发**

1. Touch事件分发中只有两个主角:ViewGroup和View。ViewGroup包含onInterceptTouchEvent、dispatchTouchEvent、onTouchEvent三个相关事件。View包含dispatchTouchEvent、onTouchEvent两个相关事件。其中ViewGroup又继承于View。

2.ViewGroup和View组成了一个树状结构，根节点为Activity内部包含的一个ViwGroup。

3.触摸事件由Action\_Down、Action\_Move、Aciton\_UP组成，其中一次完整的触摸事件中，Down和Up都只有一个，Move有若干个，可以为0个。

4.当Acitivty接收到Touch事件时，将遍历子View进行Down事件的分发。ViewGroup的遍历可以看成是递归的。分发的目的是为了找到真正要处理本次完整触摸事件的View，这个View会在onTouchuEvent结果返回true。

5.当某个子View返回true时，会中止Down事件的分发，同时在ViewGroup中记录该子View。接下去的Move和Up事件将由该子View直接进行处理。由于子View是保存在ViewGroup中的，多层ViewGroup的节点结构时，上级ViewGroup保存的会是真实处理事件的View所在的ViewGroup对象:如ViewGroup0-ViewGroup1-TextView的结构中，TextView返回了true，它将被保存在ViewGroup1中，而ViewGroup1也会返回true，被保存在ViewGroup0中。当Move和UP事件来时，会先从ViewGroup0传递至ViewGroup1，再由ViewGroup1传递至TextView。

6.当ViewGroup中所有子View都不捕获Down事件时，将触发ViewGroup自身的onTouch事件。触发的方式是调用super.dispatchTouchEvent函数，即父类View的dispatchTouchEvent方法。在所有子View都不处理的情况下，触发Acitivity的onTouchEvent方法。

7.onInterceptTouchEvent有两个作用：1.拦截Down事件的分发。2.中止Up和Move事件向目标View传递，使得目标View所在的ViewGroup捕获Up和Move事件。

**13、保存Activity状态**

onSaveInstanceState(Bundle)会在activity转入后台状态之前被调用，也就是onStop()方法之前，onPause方法之后被调用；

**14、Android中的几种动画**

帧动画：指通过指定每一帧的图片和播放时间，有序的进行播放而形成动画效果，比如想听的律动条。

补间动画：指通过指定View的初始状态、变化时间、方式，通过一系列的算法去进行图形变换，从而形成动画效果，主要有Alpha、Scale、Translate、Rotate四种效果。注意：只是在视图层实现了动画效果，并没有真正改变View的属性，比如滑动列表，改变标题栏的透明度。

属性动画：在Android3.0的时候才支持，通过不断的改变View的属性，不断的重绘而形成动画效果。相比于视图动画，View的属性是真正改变了。比如view的旋转，放大，缩小。

**15、Android中跨进程通讯的几种方式**

Android 跨进程通信，像intent，contentProvider,广播，service都可以跨进程通信。

intent：这种跨进程方式并不是访问内存的形式，它需要传递一个uri,比如说打电话。

contentProvider：这种形式，是使用数据共享的形式进行数据共享。

sevice：远程服务，aidl

广播

**16、AIDL理解**

此处延伸：简述Binder

AIDL: 每一个进程都有自己的Dalvik VM实例，都有自己的一块独立的内存，都在自己的内存上存储自己的数据，执行着自己的操作，都在自己的那片狭小的空间里过完自己的一生。而aidl就类似与两个进程之间的桥梁，使得两个进程之间可以进行数据的传输，跨进程通信有多种选择，比如 BroadcastReceiver , Messenger 等，但是 BroadcastReceiver 占用的系统资源比较多，如果是频繁的跨进程通信的话显然是不可取的；Messenger 进行跨进程通信时请求队列是同步进行的，无法并发执行。

Binde机制简单理解:

在Android系统的Binder机制中，是有Client,Service,ServiceManager,Binder驱动程序组成的，其中Client，service，Service Manager运行在用户空间，Binder驱动程序是运行在内核空间的。而Binder就是把这4种组件粘合在一块的粘合剂，其中核心的组件就是Binder驱动程序，Service Manager提供辅助管理的功能，而Client和Service正是在Binder驱动程序和Service Manager提供的基础设施上实现C/S 之间的通信。其中Binder驱动程序提供设备文件/dev/binder与用户控件进行交互，

Client、Service，Service Manager通过open和ioctl文件操作相应的方法与Binder驱动程序进行通信。而Client和Service之间的进程间通信是通过Binder驱动程序间接实现的。而Binder Manager是一个守护进程，用来管理Service，并向Client提供查询Service接口的能力。

**17、Handler的原理**

Android中主线程是不能进行耗时操作的，子线程是不能进行更新UI的。所以就有了handler，它的作用就是实现线程之间的通信。

handler整个流程中，主要有四个对象，handler，Message,MessageQueue,Looper。当应用创建的时候，就会在主线程中创建handler对象，

我们通过要传送的消息保存到Message中，handler通过调用sendMessage方法将Message发送到MessageQueue中，Looper对象就会不断的调用loop()方法

不断的从MessageQueue中取出Message交给handler进行处理。从而实现线程之间的通信。

**18、Binder机制原理**

在Android系统的Binder机制中，是有Client,Service,ServiceManager,Binder驱动程序组成的，其中Client，service，Service Manager运行在用户空间，Binder驱动程序是运行在内核空间的。而Binder就是把这4种组件粘合在一块的粘合剂，其中核心的组件就是Binder驱动程序，Service Manager提供辅助管理的功能，而Client和Service正是在Binder驱动程序和Service Manager提供的基础设施上实现C/S 之间的通信。其中Binder驱动程序提供设备文件/dev/binder与用户控件进行交互，Client、Service，Service Manager通过open和ioctl文件操作相应的方法与Binder驱动程序进行通信。而Client和Service之间的进程间通信是通过Binder驱动程序间接实现的。而Binder Manager是一个守护进程，用来管理Service，并向Client提供查询Service接口的能力。

**19、热修复的原理**

我们知道Java虚拟机 —— JVM 是加载类的class文件的，而Android虚拟机——Dalvik/ART VM 是加载类的dex文件，

而他们加载类的时候都需要ClassLoader,ClassLoader有一个子类BaseDexClassLoader，而BaseDexClassLoader下有一个

数组——DexPathList，是用来存放dex文件，当BaseDexClassLoader通过调用findClass方法时，实际上就是遍历数组，

找到相应的dex文件，找到，则直接将它return。而热修复的解决方法就是将新的dex添加到该集合中，并且是在旧的dex的前面，

所以就会优先被取出来并且return返回。

**20、Android内存泄露及管理**

（1）内存溢出（OOM）和内存泄露（对象无法被回收）的区别。

（2）引起内存泄露的原因

(3) 内存泄露检测工具 ------>LeakCanary

内存溢出 out of memory：是指程序在申请内存时，没有足够的内存空间供其使用，出现out of memory；比如申请了一个integer,但给它存了long才能存下的数，那就是内存溢出。内存溢出通俗的讲就是内存不够用。

内存泄露 memory leak：是指程序在申请内存后，无法释放已申请的内存空间，一次内存泄露危害可以忽略，但内存泄露堆积后果很严重，无论多少内存,迟早会被占光

内存泄露原因：

一、Handler 引起的内存泄漏。

解决：将Handler声明为静态内部类，就不会持有外部类SecondActivity的引用，其生命周期就和外部类无关，

如果Handler里面需要context的话，可以通过弱引用方式引用外部类

二、单例模式引起的内存泄漏。

解决：Context是ApplicationContext，由于ApplicationContext的生命周期是和app一致的，不会导致内存泄漏

三、非静态内部类创建静态实例引起的内存泄漏。

解决：把内部类修改为静态的就可以避免内存泄漏了

四、非静态匿名内部类引起的内存泄漏。

解决：将匿名内部类设置为静态的。

五、注册/反注册未成对使用引起的内存泄漏。

注册广播接受器、EventBus等，记得解绑。

六、资源对象没有关闭引起的内存泄漏。

在这些资源不使用的时候，记得调用相应的类似close（）、destroy（）、recycler（）、release（）等方法释放。

七、集合对象没有及时清理引起的内存泄漏。

通常会把一些对象装入到集合中，当不使用的时候一定要记得及时清理集合，让相关对象不再被引用。

**21、Fragment与Fragment、Activity通信的方式**

1.直接在一个Fragment中调用另外一个Fragment中的方法

2.使用接口回调

3.使用广播

4.Fragment直接调用Activity中的public方法

**22、Android UI适配**

字体使用sp,使用dp，多使用match\_parent，wrap\_content，weight

图片资源，不同图片的的分辨率，放在相应的文件夹下可使用百分比代替。

**23、app优化**

app优化:(工具：Hierarchy Viewer 分析布局  工具：TraceView 测试分析耗时的)

App启动优化

布局优化

响应优化

内存优化

电池使用优化

网络优化

App启动优化(针对冷启动)

App启动的方式有三种：

冷启动：App没有启动过或App进程被killed, 系统中不存在该App进程, 此时启动App即为冷启动。

热启动：热启动意味着你的App进程只是处于后台, 系统只是将其从后台带到前台, 展示给用户。

介于冷启动和热启动之间, 一般来说在以下两种情况下发生:

(1)用户back退出了App, 然后又启动. App进程可能还在运行, 但是activity需要重建。

(2)用户退出App后, 系统可能由于内存原因将App杀死, 进程和activity都需要重启, 但是可以在onCreate中将被动杀死锁保存的状态(saved instance state)恢复。

优化：

Application的onCreate（特别是第三方SDK初始化），首屏Activity的渲染都不要进行耗时操作，如果有，就可以放到子线程或者IntentService中

布局优化

尽量不要过于复杂的嵌套。可以使用<include>，<merge>，<ViewStub>

响应优化

Android系统每隔16ms会发出VSYNC信号重绘我们的界面(Activity)。

页面卡顿的原因：

(1)过于复杂的布局.

(2)UI线程的复杂运算

(3)频繁的GC,导致频繁GC有两个原因:1、内存抖动, 即大量的对象被创建又在短时间内马上被释放.2、瞬间产生大量的对象会严重占用内存区域。

内存优化：参考内存泄露和内存溢出部分

电池使用优化(使用工具：Batterystats & bugreport)

(1)优化网络请求

(2)定位中使用GPS, 请记得及时关闭

网络优化(网络连接对用户的影响:流量,电量,用户等待)可在Android studio下方logcat旁边那个工具Network Monitor检测

API设计：App与Server之间的API设计要考虑网络请求的频次, 资源的状态等. 以便App可以以较少的请求来完成业务需求和界面的展示.

Gzip压缩：使用Gzip来压缩request和response, 减少传输数据量, 从而减少流量消耗.

图片的Size：可以在获取图片时告知服务器需要的图片的宽高, 以便服务器给出合适的图片, 避免浪费.

网络缓存：适当的缓存, 既可以让我们的应用看起来更快, 也能避免一些不必要的流量消耗.

**24、图片优化**

(1)对图片本身进行操作。尽量不要使用setImageBitmap、setImageResource、BitmapFactory.decodeResource来设置一张大图，因为这些方法在完成decode后，

最终都是通过java层的createBitmap来完成的，需要消耗更多内存.

(2)图片进行缩放的比例，SDK中建议其值是2的指数值,值越大会导致图片不清晰。

(3)不用的图片记得调用图片的recycle()方法

**25、HybridApp WebView和JS交互**

Android与JS通过WebView互相调用方法，实际上是：

Android去调用JS的代码

1. 通过WebView的loadUrl(),使用该方法比较简洁，方便。但是效率比较低，获取返回值比较困难。

2. 通过WebView的evaluateJavascript(),该方法效率高，但是4.4以上的版本才支持，4.4以下版本不支持。所以建议两者混合使用。

JS去调用Android的代码

1. 通过WebView的addJavascriptInterface（）进行对象映射 ，该方法使用简单，仅将Android对象和JS对象映射即可，但是存在比较大的漏洞。

漏洞产生原因是：当JS拿到Android这个对象后，就可以调用这个Android对象中所有的方法，包括系统类（java.lang.Runtime 类），从而进行任意代码执行。没有

解决方式：

(1)Google 在Android 4.2 版本中规定对被调用的函数以 @JavascriptInterface进行注解从而避免漏洞攻击。

(2)在Android 4.2版本之前采用拦截prompt（）进行漏洞修复。

2. 通过 WebViewClient 的shouldOverrideUrlLoading ()方法回调拦截 url 。这种方式的优点：不存在方式1的漏洞；缺点：JS获取Android方法的返回值复杂。(ios主要用的是这个方式)

(1)Android通过 WebViewClient 的回调方法shouldOverrideUrlLoading ()拦截 url

(2)解析该 url 的协议

(3)如果检测到是预先约定好的协议，就调用相应方法

3. 通过 WebChromeClient 的onJsAlert()、onJsConfirm()、onJsPrompt（）方法回调拦截JS对话框alert()、confirm()、prompt（） 消息

这种方式的优点：不存在方式1的漏洞；缺点：JS获取Android方法的返回值复杂。

**26、JAVA GC原理**

垃圾收集算法的核心思想是：对虚拟机可用内存空间，即堆空间中的对象进行识别，如果对象正在被引用，那么称其为存活对象

，反之，如果对象不再被引用，则为垃圾对象，可以回收其占据的空间，用于再分配。垃圾收集算法的选择和垃圾收集系统参数的合理调节直接影响着系统性能。

**27、ANR**

ANR全名Application Not Responding, 也就是"应用无响应". 当操作在一段时间内系统无法处理时, 系统层面会弹出上图那样的ANR对话框.

产生原因：

(1)5s内无法响应用户输入事件(例如键盘输入, 触摸屏幕等).

(2)BroadcastReceiver在10s内无法结束

(3)Service 20s内无法结束（低概率）

解决方式：

(1)不要在主线程中做耗时的操作，而应放在子线程中来实现。如onCreate()和onResume()里尽可能少的去做创建操作。

(2)应用程序应该避免在BroadcastReceiver里做耗时的操作或计算。

(3)避免在Intent Receiver里启动一个Activity，因为它会创建一个新的画面，并从当前用户正在运行的程序上抢夺焦点。

(4)service是运行在主线程的，所以在service中做耗时操作，必须要放在子线程中。

**28、设计模式**

此处延伸：Double Check的写法被要求写出来。

单例模式：分为恶汉式和懒汉式

恶汉式：

*public class Singleton*

*{*

*private static Singleton instance = new Singleton();*

*public static Singleton getInstance()*

*{*

*return instance ;*

*}*

*}*

懒汉式：

public class Singleton02

{

    private static Singleton02 instance;

    public static Singleton02 getInstance()

    {

        if (instance == null)

        {

            synchronized (Singleton02.class)

            {

                if (instance == null)

                {

                    instance = new Singleton02();

                }

            }

        }

        return instance;

    }

}

2**9、RxJava**

<https://www.jianshu.com/p/0cd258eecf60>

<https://www.jianshu.com/p/88aacbed8aa5>

**30、MVP，MVC，MVVM**

此处延伸：手写mvp例子，与mvc之间的区别，mvp的优势

MVP模式，对应着Model--业务逻辑和实体模型,view--对应着activity，负责View的绘制以及与用户交互,Presenter--负责View和Model之间的交互,MVP模式是在MVC模式的基础上，将Model与View彻底分离使得项目的耦合性更低，在Mvc中项目中的activity对应着mvc中的C--Controllor,而项目中的逻辑处理都是在这个C中处理，同时View与Model之间的交互，也是也就是说，mvc中所有的逻辑交互和用户交互，都是放在Controllor中，也就是activity中。View和model是可以直接通信的。而MVP模式则是分离的更加彻底，分工更加明确Model--业务逻辑和实体模型，view--负责与用户交互，Presenter 负责完成View于Model间的交互，MVP和MVC最大的区别是MVC中是允许Model和View进行交互的，而MVP中很明显，Model与View之间的交互由Presenter完成。还有一点就是Presenter与View之间的交互是通过接口的

**31、手写算法（选择冒泡必须要会）**

<https://blog.csdn.net/cc1258000/article/details/79113211>

<https://www.cnblogs.com/chengxiao/p/6103002.html>

**32、JNI**

(1)安装和下载Cygwin，下载 Android NDK

(2)在ndk项目中JNI接口的设计

(3)使用C/C++实现本地方法

(4)JNI生成动态链接库.so文件

(5)将动态链接库复制到java工程，在java工程中调用，运行java工程即可

**33、RecyclerView和ListView的区别**

RecyclerView可以完成ListView,GridView的效果，还可以完成瀑布流的效果。同时还可以设置列表的滚动方向（垂直或者水平）；

RecyclerView中view的复用不需要开发者自己写代码，系统已经帮封装完成了。

RecyclerView可以进行局部刷新。

RecyclerView提供了API来实现item的动画效果。

在性能上：

如果需要频繁的刷新数据，需要添加动画，则RecyclerView有较大的优势。

如果只是作为列表展示，则两者区别并不是很大。

**34、Universal-ImageLoader，Picasso，Fresco，Glide对比**

Fresco 是 Facebook 推出的开源图片缓存工具，主要特点包括：两个内存缓存加上 Native 缓存构成了三级缓存，

优点：

1. 图片存储在安卓系统的匿名共享内存, 而不是虚拟机的堆内存中, 图片的中间缓冲数据也存放在本地堆内存, 所以, 应用程序有更多的内存使用, 不会因为图片加载而导致oom, 同时也减少垃圾回收器频繁调用回收 Bitmap 导致的界面卡顿, 性能更高。

2. 渐进式加载 JPEG 图片, 支持图片从模糊到清晰加载。

3. 图片可以以任意的中心点显示在 ImageView, 而不仅仅是图片的中心。

4. JPEG 图片改变大小也是在 native 进行的, 不是在虚拟机的堆内存, 同样减少 OOM。

5. 很好的支持 GIF 图片的显示。

缺点:

1. 框架较大, 影响 Apk 体积

2. 使用较繁琐

Universal-ImageLoader：（估计由于HttpClient被Google放弃，作者就放弃维护这个框架）

优点：

1.支持下载进度监听

2.可以在 View 滚动中暂停图片加载，通过 PauseOnScrollListener 接口可以在 View 滚动中暂停图片加载。

3.默认实现多种内存缓存算法 这几个图片缓存都可以配置缓存算法，不过 ImageLoader 默认实现了较多缓存算法，如 Size 最大先删除、使用最少先删除、最近最少使用、先进先删除、时间最长先删除等。

4.支持本地缓存文件名规则定义

Picasso 优点

1. 自带统计监控功能。支持图片缓存使用的监控，包括缓存命中率、已使用内存大小、节省的流量等。

2.支持优先级处理。每次任务调度前会选择优先级高的任务，比如 App 页面中 Banner 的优先级高于 Icon 时就很适用。

3.支持延迟到图片尺寸计算完成加载

4.支持飞行模式、并发线程数根据网络类型而变。 手机切换到飞行模式或网络类型变换时会自动调整线程池最大并发数，比如 wifi 最大并发为 4，4g 为 3，3g 为 2。  这里 Picasso 根据网络类型来决定最大并发数，而不是 CPU 核数。

5.“无”本地缓存。无”本地缓存，不是说没有本地缓存，而是 Picasso 自己没有实现，交给了 Square 的另外一个网络库 okhttp 去实现，这样的好处是可以通过请求 Response Header 中的 Cache-Control 及 Expired 控制图片的过期时间。

 Glide 优点

1. 不仅仅可以进行图片缓存还可以缓存媒体文件。Glide 不仅是一个图片缓存，它支持 Gif、WebP、缩略图。甚至是 Video，所以更该当做一个媒体缓存。

2. 支持优先级处理。

3. 与 Activity/Fragment 生命周期一致，支持 trimMemory。Glide 对每个 context 都保持一个 RequestManager，通过 FragmentTransaction 保持与 Activity/Fragment 生命周期一致，并且有对应的 trimMemory 接口实现可供调用。

4. 支持 okhttp、Volley。Glide 默认通过 UrlConnection 获取数据，可以配合 okhttp 或是 Volley 使用。实际 ImageLoader、Picasso 也都支持 okhttp、Volley。

5. 内存友好。Glide 的内存缓存有个 active 的设计，从内存缓存中取数据时，不像一般的实现用 get，而是用 remove，再将这个缓存数据放到一个 value 为软引用的 activeResources map 中，并计数引用数，在图片加载完成后进行判断，如果引用计数为空则回收掉。内存缓存更小图片，Glide 以 url、view\_width、view\_height、屏幕的分辨率等做为联合 key，将处理后的图片缓存在内存缓存中，而不是原始图片以节省大小与 Activity/Fragment 生命周期一致，支持 trimMemory。图片默认使用默认 RGB\_565 而不是 ARGB\_888，虽然清晰度差些，但图片更小，也可配置到 ARGB\_888。

6.Glide 可以通过 signature 或不使用本地缓存支持 url 过期

**42、Xutils, OKhttp, Volley, Retrofit对比**

Xutils这个框架非常全面，可以进行网络请求，可以进行图片加载处理，可以数据储存，还可以对view进行注解，使用这个框架非常方便，但是缺点也是非常明显的，使用这个项目，会导致项目对这个框架依赖非常的严重，一旦这个框架出现问题，那么对项目来说影响非常大的。、

OKhttp：Android开发中是可以直接使用现成的api进行网络请求的。就是使用HttpClient,HttpUrlConnection进行操作。okhttp针对Java和Android程序，封装的一个高性能的http请求库，支持同步，异步，而且okhttp又封装了线程池，封装了数据转换，封装了参数的使用，错误处理等。API使用起来更加的方便。但是我们在项目中使用的时候仍然需要自己在做一层封装，这样才能使用的更加的顺手。

Volley：Volley是Google官方出的一套小而巧的异步请求库，该框架封装的扩展性很强，支持HttpClient、HttpUrlConnection， 甚至支持OkHttp，而且Volley里面也封装了ImageLoader，所以如果你愿意你甚至不需要使用图片加载框架，不过这块功能没有一些专门的图片加载框架强大，对于简单的需求可以使用，稍复杂点的需求还是需要用到专门的图片加载框架。Volley也有缺陷，比如不支持post大数据，所以不适合上传文件。不过Volley设计的初衷本身也就是为频繁的、数据量小的网络请求而生。

Retrofit：Retrofit是Square公司出品的默认基于OkHttp封装的一套RESTful网络请求框架，RESTful是目前流行的一套api设计的风格， 并不是标准。Retrofit的封装可以说是很强大，里面涉及到一堆的设计模式,可以通过注解直接配置请求，可以使用不同的http客户端，虽然默认是用http ，可以使用不同Json Converter 来序列化数据，同时提供对RxJava的支持，使用Retrofit + OkHttp + RxJava + Dagger2 可以说是目前比较潮的一套框架，但是需要有比较高的门槛。

Volley VS OkHttp

Volley的优势在于封装的更好，而使用OkHttp你需要有足够的能力再进行一次封装。而OkHttp的优势在于性能更高，因为 OkHttp基于NIO和Okio ，所以性能上要比 Volley更快。IO 和 NIO这两个都是Java中的概念，如果我从硬盘读取数据，第一种方式就是程序一直等，数据读完后才能继续操作这种是最简单的也叫阻塞式IO,还有一种是你读你的,程序接着往下执行，等数据处理完你再来通知我，然后再处理回调。而第二种就是 NIO 的方式，非阻塞式， 所以NIO当然要比IO的性能要好了,而 Okio是 Square 公司基于IO和NIO基础上做的一个更简单、高效处理数据流的一个库。理论上如果Volley和OkHttp对比的话，更倾向于使用 Volley，因为Volley内部同样支持使用OkHttp,这点OkHttp的性能优势就没了，  而且 Volley 本身封装的也更易用，扩展性更好些。

OkHttp VS Retrofit

毫无疑问，Retrofit 默认是基于 OkHttp 而做的封装，这点来说没有可比性，肯定首选 Retrofit。

Volley VS Retrofit

这两个库都做了不错的封装，但Retrofit解耦的更彻底,尤其Retrofit2.0出来，Jake对之前1.0设计不合理的地方做了大量重构， 职责更细分，而且Retrofit默认使用OkHttp,性能上也要比Volley占优势，再有如果你的项目如果采用了RxJava ，那更该使用  Retrofit 。所以这两个库相比，Retrofit更有优势，在能掌握两个框架的前提下该优先使用 Retrofit。但是Retrofit门槛要比Volley稍高些，要理解他的原理，各种用法，想彻底搞明白还是需要花些功夫的，如果你对它一知半解，那还是建议在商业项目使用Volley吧。

**Java**

**1、线程中sleep和wait的区别**

(1)这两个方法来自不同的类，sleep是来自Thread，wait是来自Object；

(2)sleep方法没有释放锁，而wait方法释放了锁。

(3)wait,notify,notifyAll只能在同步控制方法或者同步控制块里面使用，而sleep可以在任何地方使用。

**2、Thread中的start()和run()方法有什么区别**

start()方法是用来启动新创建的线程，而start()内部调用了run()方法，这和直接调用run()方法是不一样的，如果直接调用run()方法，

则和普通的方法没有什么区别。

3**、关键字final和static是怎么使用的。**

final:

1、final变量即为常量，只能赋值一次。

2、final方法不能被子类重写。

3、final类不能被继承。

static：

1、static变量：对于静态变量在内存中只有一个拷贝（节省内存），JVM只为静态分配一次内存，

在加载类的过程中完成静态变量的内存分配，可用类名直接访问（方便），当然也可以通过对象来访问（但是这是不推荐的）。

2、static代码块

 static代码块是类加载时，初始化自动执行的。

3、static方法

static方法可以直接通过类名调用，任何的实例也都可以调用，因此static方法中不能用this和super关键字，

不能直接访问所属类的实例变量和实例方法(就是不带static的成员变量和成员成员方法)，只能访问所属类的静态成员变量和成员方法。

**4、String,StringBuffer,StringBuilder区别**

1、三者在执行速度上：StringBuilder > StringBuffer > String (由于String是常量，不可改变，拼接时会重新创建新的对象)。

2、StringBuffer是线程安全的，StringBuilder是线程不安全的。（由于StringBuffer有缓冲区）

**5、Java中重载和重写的区别：**

1、重载：一个类中可以有多个相同方法名的，但是参数类型和个数都不一样。这是重载。

2、重写：子类继承父类，则子类可以通过实现父类中的方法，从而新的方法把父类旧的方法覆盖。

**6、Http https区别**

此处延伸：https的实现原理

1、https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。

2、http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。

3、http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

4、http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。

https实现原理：

（1）客户使用https的URL访问Web服务器，要求与Web服务器建立SSL连接。

（2）Web服务器收到客户端请求后，会将网站的证书信息（证书中包含公钥）传送一份给客户端。

（3）客户端的浏览器与Web服务器开始协商SSL连接的安全等级，也就是信息加密的等级。

（4）客户端的浏览器根据双方同意的安全等级，建立会话密钥，然后利用网站的公钥将会话密钥加密，并传送给网站。

（5）Web服务器利用自己的私钥解密出会话密钥。

（6）Web服务器利用会话密钥加密与客户端之间的通信。

**7、Http位于TCP/IP模型中的第几层？为什么说Http是可靠的数据传输协议？**

tcp/ip的五层模型：

从下到上：物理层->数据链路层->网络层->传输层->应用层

其中tcp/ip位于模型中的网络层，处于同一层的还有ICMP（网络控制信息协议）。http位于模型中的应用层

由于tcp/ip是面向连接的可靠协议，而http是在传输层基于tcp/ip协议的，所以说http是可靠的数据传输协议。

**8、HTTP链接的特点**

HTTP连接最显著的特点是客户端发送的每次请求都需要服务器回送响应，在请求结束后，会主动释放连接。

从建立连接到关闭连接的过程称为“一次连接”。

**9、TCP和UDP的区别**

tcp是面向连接的，由于tcp连接需要三次握手，所以能够最低限度的降低风险，保证连接的可靠性。

udp 不是面向连接的，udp建立连接前不需要与对象建立连接，无论是发送还是接收，都没有发送确认信号。所以说udp是不可靠的。

由于udp不需要进行确认连接，使得UDP的开销更小，传输速率更高，所以实时行更好。

**10、Socket建立网络连接的步骤**

建立Socket连接至少需要一对套接字，其中一个运行与客户端--ClientSocket，一个运行于服务端--ServiceSocket

1、服务器监听：服务器端套接字并不定位具体的客户端套接字，而是处于等待连接的状态，实时监控网络状态，等待客户端的连接请求。

2、客户端请求：指客户端的套接字提出连接请求，要连接的目标是服务器端的套接字。注意：客户端的套接字必须描述他要连接的服务器的套接字，

指出服务器套接字的地址和端口号，然后就像服务器端套接字提出连接请求。

3、连接确认：当服务器端套接字监听到客户端套接字的连接请求时，就响应客户端套接字的请求，建立一个新的线程，把服务器端套接字的描述

发给客户端，一旦客户端确认了此描述，双方就正式建立连接。而服务端套接字则继续处于监听状态，继续接收其他客户端套接字的连接请求。

**11、Tcp／IP三次握手，四次挥手**

<https://www.cnblogs.com/zmlctt/p/3690998.html>

<https://www.cnblogs.com/lms0755/p/9053119.html>