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/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;
        }
}
```

29、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