# Android学习总结

[Android学习总结 1](#_Toc90393315)

[一.Activity 11](#_Toc90393316)

[1.Activity的启动流程 11](#_Toc90393317)

[2.onSaveInstanceState(),onRestoreInstanceState的调用时机 11](#_Toc90393318)

[3.activity的启动模式和使用场景 12](#_Toc90393319)

[4.Activity A跳转Activity B，再按返回键，生命周期执行的顺序 12](#_Toc90393320)

[5.横竖屏切换,按home键,按返回键,锁屏与解锁屏幕,跳转透明Activity界面,启动一个 Theme 为 Dialog 的 Activity，弹出Dialog时Activity的生命周期 13](#_Toc90393321)

[6.onStart 和 onResume、onPause 和 onStop 的区别 13](#_Toc90393322)

[7.Activity之间传递数据的方式Intent是否有大小限制，如果传递的数据量偏大，有哪些方案 13](#_Toc90393323)

[8.Activity的onNewIntent()方法什么时候会执行 13](#_Toc90393324)

[二.Service 13](#_Toc90393325)

[1.service 的生命周期，两种启动方式的区别 13](#_Toc90393326)

[2.Service启动流程 13](#_Toc90393327)

[3.Service与Activity怎么实现通信 13](#_Toc90393328)

[4.IntentService是什么,IntentService原理，应用场景及其与Service的区别 14](#_Toc90393329)

[5.Service 的 onStartCommand 方法有几种返回值?各代表什么意思? 14](#_Toc90393330)

[6.bindService和startService混合使用的生命周期以及怎么关闭 14](#_Toc90393331)

[3.BroadcastReceiver 14](#_Toc90393332)

[1.广播的分类和使用场景 14](#_Toc90393333)

[2.广播的两种注册方式的区别 14](#_Toc90393334)

[3.广播发送和接收的原理 14](#_Toc90393335)

[4.本地广播和全局广播的区别 14](#_Toc90393336)

[4.ContentProvider 15](#_Toc90393337)

[1.什么是ContentProvider及其使用 15](#_Toc90393338)

[2.ContentProvider,ContentResolver,ContentObserver之间的关系 15](#_Toc90393339)

[3.ContentProvider的实现原理 15](#_Toc90393340)

[4.ContentProvider的优点 15](#_Toc90393341)

[5.Uri 是什么 15](#_Toc90393342)

[五.Handler 15](#_Toc90393343)

[1.Handler的实现原理 15](#_Toc90393344)

[2.子线程中能不能直接new一个Handler,为什么主线程可以主线程的Looper第一次调用loop方法,什么时候,哪个类 16](#_Toc90393345)

[3.Handler导致的内存泄露原因及其解决方案 16](#_Toc90393346)

[4.一个线程可以有几个Handler,几个Looper,几个MessageQueue对象 16](#_Toc90393347)

[5.Message对象创建的方式有哪些 & 区别？Message.obtain()怎么维护消息池的 16](#_Toc90393348)

[6.Handler 有哪些发送消息的方法 16](#_Toc90393349)

[7.Handler的post与sendMessage的区别和应用场景 16](#_Toc90393350)

[8.handler postDealy后消息队列有什么变化，假设先 postDelay 10s, 再postDelay 1s, 怎么处理这2条消息 16](#_Toc90393351)

[9.MessageQueue是什么数据结构 17](#_Toc90393352)

[10.Handler怎么做到的一个线程对应一个Looper，如何保证只有一个MessageQueue 17](#_Toc90393353)

[11.ThreadLocal在Handler机制中的作用 17](#_Toc90393354)

[六.View绘制 17](#_Toc90393355)

[1.View绘制流程 17](#_Toc90393356)

[2.MeasureSpec是什么 17](#_Toc90393357)

[3.子View创建MeasureSpec创建规则是什么 17](#_Toc90393358)

[4.自定义Viewwrap\_content不起作用的原因 18](#_Toc90393359)

[5.在Activity中获取某个View的宽高有几种方法 18](#_Toc90393360)

[6.为什么onCreate获取不到View的宽高 18](#_Toc90393361)

[7.View#post与Handler#post的区别 18](#_Toc90393362)

[8.Android绘制和屏幕刷新机制原理 18](#_Toc90393363)

[9.Choreography原理 18](#_Toc90393364)

[10.什么是双缓冲 18](#_Toc90393365)

[七.View事件分发 19](#_Toc90393366)

[1.View事件分发机制 19](#_Toc90393367)

[2.view的onTouchEvent，OnClickListerner和OnTouchListener的onTouch方法 三者优先级 19](#_Toc90393368)

[3.onTouch 和onTouchEvent 的区别 19](#_Toc90393369)

[4.ACTION\_CANCEL什么时候触发 19](#_Toc90393370)

[5.事件是先到DecorView还是先到Window 19](#_Toc90393371)

[6.点击事件被拦截，但是想传到下面的View，如何操作 19](#_Toc90393372)

[7.如何解决View的事件冲突 19](#_Toc90393373)

[8.在 ViewGroup 中的 onTouchEvent 中消费 ACTION\_DOWN 事件，ACTION\_UP事件是怎么传递 20](#_Toc90393374)

[9.Activity ViewGroup和View都不消费ACTION\_DOWN,那么ACTION\_UP事件是怎么传递的 20](#_Toc90393375)

[10.同时对父 View 和子 View 设置点击方法，优先响应哪个 20](#_Toc90393376)

[11.requestDisallowInterceptTouchEvent的调用时机 20](#_Toc90393377)

[八.RecycleView 20](#_Toc90393378)

[1.RecyclerView的多级缓存机制,每一级缓存具体作用是什么,分别在什么场景下会用到哪些缓存 20](#_Toc90393379)

[2.RecyclerView的滑动回收复用机制 20](#_Toc90393380)

[3.RecyclerView的刷新回收复用机制 20](#_Toc90393381)

[4.RecyclerView 为什么要预布局 21](#_Toc90393382)

[5.ListView 与 RecyclerView区别 21](#_Toc90393383)

[6.RecyclerView性能优化 21](#_Toc90393384)

[九.Viewpager&Fragment 21](#_Toc90393385)

[1.Fragment的生命周期 & 结合Activity的生命周期 21](#_Toc90393386)

[2.Activity和Fragment的通信方式， Fragment之间如何进行通信 21](#_Toc90393387)

[3.为什么使用Fragment.setArguments(Bundle)传递参数 21](#_Toc90393388)

[4.FragmentPageAdapter和FragmentStatePageAdapter区别及使用场景 21](#_Toc90393389)

[5.Fragment懒加载 22](#_Toc90393390)

[6.ViewPager2与ViewPager区别 22](#_Toc90393391)

[7.Fragment嵌套问题 22](#_Toc90393392)

[十.WebView 22](#_Toc90393393)

[1.如何提高WebView加载速度 22](#_Toc90393394)

[2.WebView与 js的交互 22](#_Toc90393395)

[3.WebView的漏洞 22](#_Toc90393396)

[4.JsBridge原理 22](#_Toc90393397)

[十一.动画 23](#_Toc90393398)

[1.动画的类型 23](#_Toc90393399)

[2.补间动画和属性动画的区别 23](#_Toc90393400)

[3.ObjectAnimator，ValueAnimator及其区别 23](#_Toc90393401)

[4.TimeInterpolator插值器，自定义插值器 23](#_Toc90393402)

[5.TypeEvaluator估值器 23](#_Toc90393403)

[十二.Bitmap 23](#_Toc90393404)

[1.Bitmap 内存占用的计算 23](#_Toc90393405)

[2.getByteCount() & getAllocationByteCount()的区别 23](#_Toc90393406)

[3.Bitmap的压缩方式 24](#_Toc90393407)

[4.LruCache & DiskLruCache原理 24](#_Toc90393408)

[5.如何设计一个图片加载库 24](#_Toc90393409)

[6.有一张非常大的图片,如何去加载这张大图片 24](#_Toc90393410)

[7.如果把drawable-xxhdpi下的图片移动到drawable-xhdpi下，图片内存是如何变的。 24](#_Toc90393411)

[8.如果在hdpi、xxhdpi下放置了图片，加载的优先级。如果是400800，10801920，加载的优先级。 24](#_Toc90393412)

[十三.mvc&mvp&mvvm 25](#_Toc90393413)

[1.MVC及其优缺点 25](#_Toc90393414)

[2.MVP及其优缺点 25](#_Toc90393415)

[3.MVVM及其优缺点 25](#_Toc90393416)

[4.MVP如何管理Presenter的生命周期，何时取消网络请求 25](#_Toc90393417)

[十四.Binder 25](#_Toc90393418)

[1.Android中进程和线程的关系,区别 25](#_Toc90393419)

[2.为何需要进行IPC,多进程通信可能会出现什么问题 25](#_Toc90393420)

[3.Android中IPC方式有几种、各种方式优缺点 25](#_Toc90393421)

[4.为何新增Binder来作为主要的IPC方式 26](#_Toc90393422)

[5.什么是Binder 26](#_Toc90393423)

[6.Binder的原理 26](#_Toc90393424)

[7.Binder Driver 如何在内核空间中做到一次拷贝的？ 26](#_Toc90393425)

[7.使用Binder进行数据传输的具体过程 26](#_Toc90393426)

[8.Binder框架中ServiceManager的作用 26](#_Toc90393427)

[十五.内存泄漏&内存溢出 26](#_Toc90393428)

[1.什么是OOM & 什么是内存泄漏以及原因 26](#_Toc90393429)

[2.Thread是如何造成内存泄露的，如何解决？ 27](#_Toc90393430)

[3.Handler导致的内存泄露的原因以及如何解决 27](#_Toc90393431)

[4.如何加载Bitmap防止内存溢出 27](#_Toc90393432)

[5.MVP中如何处理Presenter层以防止内存泄漏的 27](#_Toc90393433)

[十六.性能优化 27](#_Toc90393434)

[1.内存优化 27](#_Toc90393435)

[2.启动优化 27](#_Toc90393436)

[3.布局加载和绘制优化 27](#_Toc90393437)

[4.卡顿优化 28](#_Toc90393438)

[5.网络优化 28](#_Toc90393439)

[十七.Window&WindowManager 28](#_Toc90393440)

[1.什么是Window 28](#_Toc90393441)

[2.什么是WindowManager 28](#_Toc90393442)

[3.什么是ViewRootImpl 28](#_Toc90393443)

[4.什么是DecorView 28](#_Toc90393444)

[5.Activity，View，Window三者之间的关系 28](#_Toc90393445)

[6.DecorView什么时候被WindowManager添加到Window中 29](#_Toc90393446)

[十八.WMS 29](#_Toc90393447)

[1.什么是WMS 29](#_Toc90393448)

[2.WMS是如何管理Window的 29](#_Toc90393449)

[3.IWindowSession是什么，WindowSession的创建过程是怎样的 29](#_Toc90393450)

[4.WindowToken是什么 29](#_Toc90393451)

[5.WindowState是什么 29](#_Toc90393452)

[6.Android窗口大概分为几种？分组原理是什么 29](#_Toc90393453)

[7.Dialog的Context只能是Activity的Context，不能是Application的Context 30](#_Toc90393454)

[8.App应用程序如何与SurfaceFlinger通信的View 的绘制是如何把数据传递给 SurfaceFlinger 的 30](#_Toc90393455)

[9.共享内存的具体实现是什么 30](#_Toc90393456)

[10.relayout是如何向SurfaceFlinger申请Surface 30](#_Toc90393457)

[11.什么是Surface 30](#_Toc90393458)

[十九.AMS 30](#_Toc90393459)

[1.ActivityManagerService是什么？什么时候初始化的？有什么作用？ 30](#_Toc90393460)

[2.ActivityThread是什么?ApplicationThread是什么?他们的区别 31](#_Toc90393461)

[3.Instrumentation是什么？和ActivityThread是什么关系？ 31](#_Toc90393462)

[4.ActivityManagerService和zygote进程通信是如何实现的 31](#_Toc90393463)

[5.ActivityRecord、TaskRecord、ActivityStack，ActivityStackSupervisor，ProcessRecord 31](#_Toc90393464)

[6.ActivityManager、ActivityManagerService、ActivityManagerNative、ActivityManagerProxy的关系 31](#_Toc90393465)

[7.手写实现简化版AMS 31](#_Toc90393466)

[二十.系统启动 31](#_Toc90393467)

[1.android系统启动流程 31](#_Toc90393468)

[2.SystemServer，ServiceManager，SystemServiceManager的关系 32](#_Toc90393469)

[3.孵化应用进程这种事为什么不交给SystemServer来做，而专门设计一个Zygote 32](#_Toc90393470)

[4.Zygote的IPC通信机制为什么使用socket而不采用binder 32](#_Toc90393471)

[二十一.App启动&打包&安装 32](#_Toc90393472)

[1.应用启动流程 32](#_Toc90393473)

[2.apk组成和Android的打包流程 32](#_Toc90393474)

[3.Android的签名机制，签名如何实现的,v2相比于v1签名机制的改变 32](#_Toc90393475)

[4.APK的安装流程 32](#_Toc90393476)

[二十二.序列化 33](#_Toc90393477)

[1.什么是序列化 33](#_Toc90393478)

[2.为什么需要使用序列化和反序列化 33](#_Toc90393479)

[3.序列化的有哪些好处 33](#_Toc90393480)

[4.Serializable 和 Parcelable 的区别 33](#_Toc90393481)

[5.什么是serialVersionUID 33](#_Toc90393482)

[6.为什么还要显示指定serialVersionUID的值? 33](#_Toc90393483)

[二十三.Art & Dalvik 及其区别 34](#_Toc90393484)

[1.Art & Dalvik 及其区别 34](#_Toc90393485)

[二十四.模块化&组件化 34](#_Toc90393486)

[1.什么是模块化 34](#_Toc90393487)

[2.什么是组件化 34](#_Toc90393488)

[3.组件化优点和方案 34](#_Toc90393489)

[4.组件独立调试 34](#_Toc90393490)

[5.组件间通信 34](#_Toc90393491)

[6.Aplication动态加载 34](#_Toc90393492)

[7.ARouter原理 35](#_Toc90393493)

[二十五.热修复&插件化 35](#_Toc90393494)

[1.插件化的定义 35](#_Toc90393495)

[2.插件化的优势 35](#_Toc90393496)

[3.插件化框架对比 35](#_Toc90393497)

[4.插件化流程 35](#_Toc90393498)

[5.插件化类加载原理 35](#_Toc90393499)

[6.插件化资源加载原理 35](#_Toc90393500)

[7.插件化Activity加载原理 36](#_Toc90393501)

[8.热修复和插件化区别 36](#_Toc90393502)

[9.热修复原理 36](#_Toc90393503)

[二十六.AOP 36](#_Toc90393504)

[1.AOP是什么 36](#_Toc90393505)

[2.AOP的优点 36](#_Toc90393506)

[3.AOP的实现方式,APT,AspectJ,ASM,epic,hook 36](#_Toc90393507)

[二十七.jectpack 37](#_Toc90393508)

[1.Navigation 37](#_Toc90393509)

[2.DataBinding 37](#_Toc90393510)

[3.Viewmodel 37](#_Toc90393511)

[4.livedata 37](#_Toc90393512)

[5.liferecycle 37](#_Toc90393513)

[二十八.开源框架 37](#_Toc90393514)

[1.Okhttp源码流程,线程池 37](#_Toc90393515)

[2.Okhttp拦截器,addInterceptor 和 addNetworkdInterceptor区别 37](#_Toc90393516)

[3.Okhttp责任链模式 38](#_Toc90393517)

[4.Okhttp缓存怎么处理 38](#_Toc90393518)

[5.Okhttp连接池和socket复用 38](#_Toc90393519)

[6.Glide怎么绑定生命周期 38](#_Toc90393520)

[7.Glide缓存机制,内存缓存，磁盘缓存 38](#_Toc90393521)

[8.Glide与Picasso的区别 38](#_Toc90393522)

[9.LruCache原理 38](#_Toc90393523)

[10.Retrofit源码流程,动态代理 39](#_Toc90393524)

## 一.Activity

#### 1.Activity的启动流程

#### 2.onSaveInstanceState(),onRestoreInstanceState的调用时机

**(1) onSaveInstanceState(Bundle outState)在什么时机会被调用呢？**

答案是当activity有可能被系统回收的情况下,而且是在onStop()之前。注意是有可能，如果是已经确定会被销毁，比如用户按下了返回键，或者调用了finish()方法销毁activity，则onSaveInstanceState不会被调用。

总结下，onSaveInstanceState(Bundle outState)会在以下情况被调用：

* 当用户按下HOME键时。
* 从最近应用中选择运行其他的程序时。
* 按下电源按键（关闭屏幕显示）时。
* 从当前activity启动一个新的activity时。
* 屏幕方向切换时(无论竖屏切横屏还是横屏切竖屏都会调用)。

在前4种情况下，当前activity的生命周期为：

onPause -> onSaveInstanceState -> onStop。

**(2) onRestoreInstanceState什么时机被调用？**

onRestoreInstanceState(Bundle savedInstanceState)只有在activity确实是被系统回收，重新创建activity的情况下才会被调用。

比如第5种情况屏幕方向切换时，activity生命周期如下：

onPause -> onSaveInstanceState -> onStop -> onDestroy -> onCreate -> onStart -> onRestoreInstanceState -> onResume

在这里onRestoreInstanceState被调用，是因为屏幕切换时原来的activity确实被系统回收了，又重新创建了一个新的activity。

而按HOME键返回桌面，又马上点击应用图标回到原来页面时，activity生命周期如下：

onPause -> onSaveInstanceState -> onStop -> onRestart -> onStart -> onResume

因为activity没有被系统回收，因此onRestoreInstanceState没有被调用。

如果onRestoreInstanceState被调用了，则页面必然被回收过，则onSaveInstanceState必然被调用过。

**(3) onCreate()里也有Bundle参数，可以用来恢复数据，它和onRestoreInstanceState有什么区别？**

因为onSaveInstanceState 不一定会被调用，所以onCreate()里的Bundle参数可能为空，如果使用onCreate()来恢复数据，一定要做非空判断。

而onRestoreInstanceState的Bundle参数一定不会是空值，因为它只有在上次activity被回收了才会调用。

而且onRestoreInstanceState是在onStart()之后被调用的。有时候我们需要onCreate()中做的一些初始化完成之后再恢复数据，用onRestoreInstanceState会比较方便。下面是官方文档对onRestoreInstanceState的说明：

注意这个说明的最后一句是什么意思？

to allow subclasses to decide whether to use your default implementation.

它是说，用onRestoreInstanceState方法恢复数据，你可以决定是否在方法里调用父类的onRestoreInstanceState方法，即是否调用super.onRestoreInstanceState(savedInstanceState);

而用onCreate()恢复数据，你必须调用super.onCreate(savedInstanceState);

#### activity的启动模式和使用场景

https://blog.csdn.net/black\_bird\_cn/article/details/79764794

**(1)基本描述**

standard：标准模式：如果在mainfest中不设置就默认standard；每次启动时都会创建一个新的实例

singleTop：栈顶复用模式：**使用场景：展示推送过来的消息**，如果栈中没有activity创建并入栈，如果栈中已经有Activity且此activity在栈顶直接复用，如果栈中有Activity但是并不在栈顶新建activity并入栈，与standard相比栈顶复用可以有效减少activity重复创建对资源的消耗，但是这要根据具体情况而定，不能一概而论；

singleTask：栈内单例模式，**使用场景：程序入口等启动页面，**栈内只会有一个activity实例，栈内已存activity实例，如果在栈顶则直接复用，不在栈顶，会把该activity上所有activity实例出栈以后再复用此activity

singleInstance :堆内单例：**使用场景：完全独立的，类似闹钟的提示**，整个手机操作系统里面只有一个实例存在就是内存单例；若该实例不存在,则要启动一个新activity实例,并且会存在于一个单独的任务栈中

##### **(2)Intent中标志位设置启动模式**

启动模式可以在AndroidMainfest的xml文件中进行配置，GoogleAndroid团队同时提供另种级别更高的设置方式，即通过Intent.setFlags(int flags)设置启动模式；

FLAG\_ACTIVITY\_CLEAR\_TOP : 等同于mainfest中配置的singleTask；

FLAG\_ACTIVITY\_SINGLE\_TOP: 同样等同于mainfest中配置的singleTop;

FLAG\_ACTIVITY\_EXCLUDE\_FROM\_RECENTS: 其对应在AndroidManifest中的属性为android:excludeFromRecents=“true”,当用户按了“最近任务列表”时候,该Task不会出现在最近任务列表中，可达到隐藏应用的目的。

FLAG\_ACTIVITY\_NO\_HISTORY: 对应在AndroidManifest中的属性为:android:noHistory=“true”，这个FLAG启动的Activity，一旦退出，它不会存在于栈中。

FLAG\_ACTIVITY\_NEW\_TASK: 这个属性需要在被start的目标Activity在AndroidManifest.xml文件配置taskAffinity的值【必须和startActivity发起者Activity的包名不一样，如果是跳转另一个App的话可以taskAffinity可以省略】，则会在新标记的Affinity所存在的taskAffinity中压入这个Activity。

##### **(二)、taskAffinity属性**

taskAffinity只有和SingleTask启动模式匹配使用时，启动的Activity才会运行在名字和taskAffinity相同的任务栈中

1.如果不指定taskAffinity属性，应用程序的所有Activity都存放于默认task（single instance启动的activity除外，因为single instance启动的activity独占一个task）

2.指定taskAffinity，只有当Activity设置FLAG\_ACTIVITY\_NEW\_TASK或者在AndroidMainfest.xml启动模式设置为singleTask才起作用，否则不起作用。

### **Activity四种启动模式常见使用场景**



#### 4.Activity A跳转Activity B，再按系统返回键，生命周期执行的顺序

（1）Activity A，Activity B都是启动模式都是标准模式（standard）

A onCreate-->A onStart()-->A onResume()-->A onpasue()-->B onCreate()-->B onStart()-->B onResume()-->A onStop()-->B onPause()-->A onRestart()-->A onstart()-->A onResume()--->B onStop()-->B onDestory()

#### 5.横竖屏切换,按home键,按返回键,锁屏与解锁屏幕,跳转透明Activity界面,启动一个 Theme 为 Dialog 的 Activity，弹出Dialog时Activity的生命周期

Android4.0以前配置横竖屏切换属性是：android:configChanges="keyboardHidden|orientation”

Android4.0以后配置横竖屏切换属性是：

android:configChanges="keyboardHidden|orientation|screenSize"

1、未配置android:configChanges="keyboardHidden|orientation|screenSize"切换横竖屏生命周期

onCreate()-->onStart()-->onResume()-->onPasue()-->切换横竖屏onStop()--onSaveInstanceState()-->onDestory()-->onCreate()-->onStart()--onRestoreInstanceState()-->onResume()

2、配置android:configChanges="keyboardHidden|orientation|screenSize"切换横竖屏生命周期

onCreate()-->onStart()-->onResume()-->切换竖屏onConfigurationChanged()切换横屏onConfigurationChanged()

熄屏 onCreate()-->onStart()-->onResume()熄屏-->onPasue()-->onStop()-->onSaveInstanceState()

解锁屏幕 解锁屏幕后如果页面不可见执行的生命周期(如锁屏后网易云音乐界面在前台，解锁后可见页面也是网易云音乐)onRestart（）-->onStart()

解锁屏幕后如果页面可见执行的生命周期（解锁后可见页面是网易云音乐，右滑关闭网易云页面，app页面可见执行的生命周期）onRestart（）-->onStart()-->onResume()

按home键 执行的生命周期 onPause()-->onstop()--->onSaveInstanceState()

从最近任务选择app执行的生命周期是：onRestart()-->onstart()-->onResume()

按返回键 onPasue()-->onStop()-->onDestory()

启动一个 Theme 为 Dialog 的 Activity，弹出Dialog时Activity的生命周期

onCreate()-->onStart()-->onResume()--->启动一个 Theme 为 Dialog 的 Activity -->onPasue()按返回键--->onResume()

#### 6.onStart 和 onResume、onPause 和 onStop 的区别

onStart

onCreate或者onRestart方法执行后会调用此方法，该方法表明Activity准备展示给用户了

This is a good place to begin drawing visual elements, running animations

这是开始绘制视觉元素、运行动画的好地方

可以在这个方法里调用finish，会跳过onResume, onPause直接调用onStop

Dispatch onStart() to all fragments.

onResume

在onRestoreInstanceState, onRestart, 或者onPause之后被调用，以便页面开始和用户进行交互

这时activity已经激活并且准备接受输入内容，此时activity在栈顶并且用户可见

Build.VERSION\_CODES.Q（Android 10）之前，这也是尝试打开独占访问设备或访问单例资源的好地方（如相机设备等），Android 10以后，Android具有多个任务同时工作的情况，所以多个Activity的onResume可能同时恢复工作，但是要判断谁在栈顶可以通过onTopResumedActivityChanged(boolean)来判断

将onResume()分派fragments， 注意：为了更好的与旧版本的平台进行交互操作，再此调用fragments attached 到activity不会恢复

onPause

当用户不在主动与activity交互时调用，但是仍然在屏幕可见，与之相对应的是重新开始

当在activity A页面启动activity B时，将在A上调用此回调，在A的onPause返回之前，不会创建B，因此请确保不要在此处执行任何冗长的操作

此回调主要用于保存activity 正在编辑的任何持久状态，向用户提供“原地编辑”模型，并确保在没有足够资源启动新活动而不首先终止此活动时不会丢失任何内容。这也是一个关闭一些耗时CPU较高的工作的好地方，以便尽可能快地切换到下一个activity

Build.VERSION\_CODES.Q（Android 10）之前，这也是关闭独占访问设备或释放单例资源的好地方（如相机设备等），Android 10以后，Android具有多个任务同时工作的情况，所以多个Activity的onResume可能同时恢复工作，但是要判断谁在栈顶可以通过onTopResumedActivityChanged(boolean)来判断

如果activity在栈顶，在接收到这个调用以后，你一般将会收到onstop调用（在下一个activity已经被恢复并且显示）。然而在某些情况下，会直接回调onResume，而不会进入stopped状态。在某些情况下，当处于多窗口模式时，活动也可以处于暂停状态，用户仍然可以看到

onStop

用户不在可见时调用，方法后续可能会执行onRestart, onDestroy或者什么也没有，具体取决于后面用户的activity。这里是一个停止刷新ui、运行动画和其他视觉效果的好地方

#### 7.Activity之间传递数据的方式Intent是否有大小限制，如果传递的数据量偏大，有哪些方案

#### 8.Activity的onNewIntent()方法什么时候会执行

## 二.Service

#### 1.service 的生命周期，两种启动方式的区别



如果组件通过调用 startService() 启动服务（这会引起对 onStartCommand() 的调用），则服务会一直运行，直到其使用 stopSelf() 自行停止运行，或由其他组件通过调用 stopService() 将其停止为止。如果服务已经被创建，多次调用startService启动服务，会重复调用onStartCommand()方法  
  
如果组件通过调用 bindService() 来创建服务，且未调用 onStartCommand()，则服务只会在该组件与其绑定时运行。当该服务与其所有组件取消绑定后，系统便会将其销毁。如果服务已经被创建，多次调用bindService绑定服务，并不会重复调用onBind()方法

#### 2.Service启动流程

#### 3.Service与Activity怎么实现通信

通过Binder或者广播通信

#### 4.IntentService是什么,IntentService原理，应用场景及其与Service的区别

 IntentService是 Service 的子类，其使用工作线程逐一处理所有启动请求。如果不要求服务同时处理多个请求，此类为最佳选择。实现 onHandleIntent()，该方法会接收每个启动请求的 Intent，以便执行后台工作  
  
IntentService原理   
原理是继承Service,并使用Handler开启一个子线程处理请求

#### 5.Service 的 onStartCommand 方法有几种返回值?各代表什么意思?

* [START\_NOT\_STICKY](https://developer.android.com/reference/android/app/Service?hl=zh-cn" \l "START_NOT_STICKY)

如果系统在 onStartCommand() 返回后终止服务，系统不会自动重启该服务。

* [START\_STICKY](https://developer.android.com/reference/android/app/Service?hl=zh-cn" \l "START_STICKY)

如果系统在 onStartCommand() 返回后终止服务，则其会重建服务并调用 onStartCommand()，但不保留 Intent

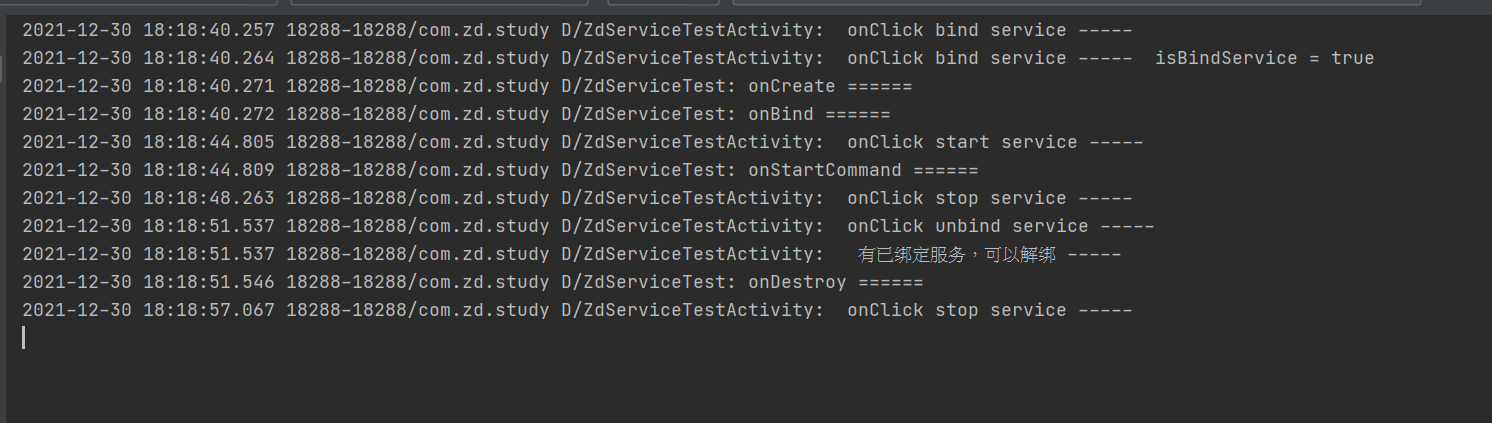
* [START\_REDELIVER\_INTENT](https://developer.android.com/reference/android/app/Service?hl=zh-cn" \l "START_REDELIVER_INTENT)

重传Intent。使用这个返回值时，如果在执行完onStartCommand后，服务被异常kill掉，系统会自动重启该服务，并将Intent的值传入。

#### 6.bindService和startService混合使用的生命周期以及怎么关闭

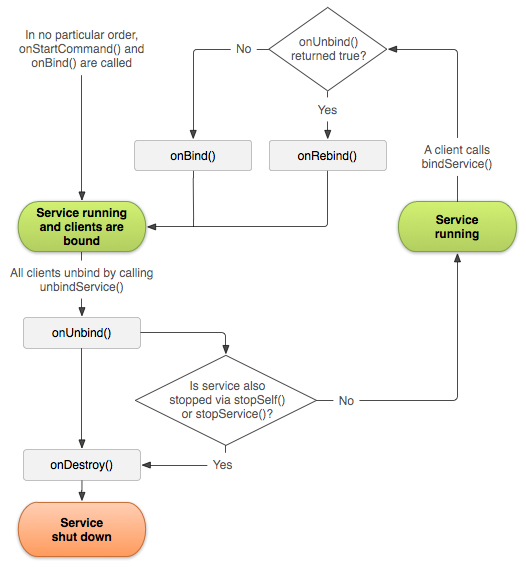


先启动服务再绑定服务，调用stop并不能销毁服务，只有调用unbindService，服务才会解绑并调用Serivce 的onDestory()方法



先绑定再启动服务服务，调用unbindService并不能销毁服务，只有调用stop才会停止服务并调用service的onDestory()方法

已启动并且还允许绑定的服务的生命周期，如下所示：



## 3.BroadcastReceiver

#### 1.广播的分类和使用场景

#### 2.广播的两种注册方式的区别

#### 3.广播发送和接收的原理

#### 4.本地广播和全局广播的区别

## 4.ContentProvider

#### 1.什么是ContentProvider及其使用

#### 2.ContentProvider,ContentResolver,ContentObserver之间的关系

#### 3.ContentProvider的实现原理

#### 4.ContentProvider的优点

#### 5.Uri 是什么

## 五.Handler

#### 1.Handler的实现原理

#### 2.子线程中能不能直接new一个Handler,为什么主线程可以主线程的Looper第一次调用loop方法,什么时候,哪个类

#### 3.Handler导致的内存泄露原因及其解决方案

#### 4.一个线程可以有几个Handler,几个Looper,几个MessageQueue对象

#### 5.Message对象创建的方式有哪些 & 区别？Message.obtain()怎么维护消息池的

#### 6.Handler 有哪些发送消息的方法

#### 7.Handler的post与sendMessage的区别和应用场景

#### 8.handler postDealy后消息队列有什么变化，假设先 postDelay 10s, 再postDelay 1s, 怎么处理这2条消息

#### 9.MessageQueue是什么数据结构

#### 10.Handler怎么做到的一个线程对应一个Looper，如何保证只有一个MessageQueue

#### 11.ThreadLocal在Handler机制中的作用

## 六.View绘制

#### 1.View绘制流程

#### 2.MeasureSpec是什么

#### 3.子View创建MeasureSpec创建规则是什么

#### 4.自定义Viewwrap\_content不起作用的原因

#### 5.在Activity中获取某个View的宽高有几种方法

#### 6.为什么onCreate获取不到View的宽高

#### 7.View#post与Handler#post的区别

#### 8.Android绘制和屏幕刷新机制原理

#### 9.Choreography原理

#### 10.什么是双缓冲

## 七.View事件分发

#### 1.View事件分发机制

#### 2.view的onTouchEvent，OnClickListerner和OnTouchListener的onTouch方法 三者优先级

#### 3.onTouch 和onTouchEvent 的区别

#### 4.ACTION\_CANCEL什么时候触发

#### 5.事件是先到DecorView还是先到Window

#### 6.点击事件被拦截，但是想传到下面的View，如何操作

#### 7.如何解决View的事件冲突

#### 8.在 ViewGroup 中的 onTouchEvent 中消费 ACTION\_DOWN 事件，ACTION\_UP事件是怎么传递

#### 9.Activity ViewGroup和View都不消费ACTION\_DOWN,那么ACTION\_UP事件是怎么传递的

#### 10.同时对父 View 和子 View 设置点击方法，优先响应哪个

#### 11.requestDisallowInterceptTouchEvent的调用时机

## 八.RecycleView

#### 1.RecyclerView的多级缓存机制,每一级缓存具体作用是什么,分别在什么场景下会用到哪些缓存

#### 2.RecyclerView的滑动回收复用机制

#### 3.RecyclerView的刷新回收复用机制

#### 4.RecyclerView 为什么要预布局

#### 5.ListView 与 RecyclerView区别

#### 6.RecyclerView性能优化

## 九.Viewpager&Fragment

#### 1.Fragment的生命周期 & 结合Activity的生命周期

#### 2.Activity和Fragment的通信方式， Fragment之间如何进行通信

#### 3.为什么使用Fragment.setArguments(Bundle)传递参数

#### 4.FragmentPageAdapter和FragmentStatePageAdapter区别及使用场景

#### 5.Fragment懒加载

#### 6.ViewPager2与ViewPager区别

#### 7.Fragment嵌套问题

## 十.WebView

#### 1.如何提高WebView加载速度

#### 2.WebView与 js的交互

#### 3.WebView的漏洞

#### 4.JsBridge原理

## 十一.动画

#### 1.动画的类型

#### 2.补间动画和属性动画的区别

#### 3.ObjectAnimator，ValueAnimator及其区别

#### 4.TimeInterpolator插值器，自定义插值器

#### 5.TypeEvaluator估值器

## 十二.Bitmap

#### 1.Bitmap 内存占用的计算

#### 2.getByteCount() & getAllocationByteCount()的区别

#### 3.Bitmap的压缩方式

#### 4.LruCache & DiskLruCache原理

#### 5.如何设计一个图片加载库

#### 6.有一张非常大的图片,如何去加载这张大图片

#### 7.如果把drawable-xxhdpi下的图片移动到drawable-xhdpi下，图片内存是如何变的。

#### 8.如果在hdpi、xxhdpi下放置了图片，加载的优先级。如果是400800，10801920，加载的优先级。

## 十三.mvc&mvp&mvvm

#### 1.MVC及其优缺点

#### 2.MVP及其优缺点

#### 3.MVVM及其优缺点

#### 4.MVP如何管理Presenter的生命周期，何时取消网络请求

## 十四.Binder

#### 1.Android中进程和线程的关系,区别

#### 2.为何需要进行IPC,多进程通信可能会出现什么问题

#### 3.Android中IPC方式有几种、各种方式优缺点

#### 4.为何新增Binder来作为主要的IPC方式

#### 5.什么是Binder

#### 6.Binder的原理

#### 7.Binder Driver 如何在内核空间中做到一次拷贝的？

#### 7.使用Binder进行数据传输的具体过程

#### 8.Binder框架中ServiceManager的作用

## 十五.内存泄漏&内存溢出

#### 1.什么是OOM & 什么是内存泄漏以及原因

#### 2.Thread是如何造成内存泄露的，如何解决？

#### 3.Handler导致的内存泄露的原因以及如何解决

#### 4.如何加载Bitmap防止内存溢出

#### 5.MVP中如何处理Presenter层以防止内存泄漏的

## 十六.性能优化

#### 1.内存优化

#### 2.启动优化

#### 3.布局加载和绘制优化

#### 4.卡顿优化

#### 5.网络优化

## 十七.Window&WindowManager

#### 1.什么是Window

#### 2.什么是WindowManager

#### 3.什么是ViewRootImpl

#### 4.什么是DecorView

#### 5.Activity，View，Window三者之间的关系

#### 6.DecorView什么时候被WindowManager添加到Window中

## 十八.WMS

#### 1.什么是WMS

#### 2.WMS是如何管理Window的

#### 3.IWindowSession是什么，WindowSession的创建过程是怎样的

#### 4.WindowToken是什么

#### 5.WindowState是什么

#### 6.Android窗口大概分为几种？分组原理是什么

#### 7.Dialog的Context只能是Activity的Context，不能是Application的Context

#### 8.App应用程序如何与SurfaceFlinger通信的View 的绘制是如何把数据传递给 SurfaceFlinger 的

#### 9.共享内存的具体实现是什么

#### 10.relayout是如何向SurfaceFlinger申请Surface

#### 11.什么是Surface

## 十九.AMS

#### 1.ActivityManagerService是什么？什么时候初始化的？有什么作用？

#### 2.ActivityThread是什么?ApplicationThread是什么?他们的区别

#### 3.Instrumentation是什么？和ActivityThread是什么关系？

#### 4.ActivityManagerService和zygote进程通信是如何实现的

#### 5.ActivityRecord、TaskRecord、ActivityStack，ActivityStackSupervisor，ProcessRecord

#### 6.ActivityManager、ActivityManagerService、ActivityManagerNative、ActivityManagerProxy的关系

#### 7.手写实现简化版AMS

## 二十.系统启动

#### 1.android系统启动流程

#### 2.SystemServer，ServiceManager，SystemServiceManager的关系

#### 3.孵化应用进程这种事为什么不交给SystemServer来做，而专门设计一个Zygote

#### 4.Zygote的IPC通信机制为什么使用socket而不采用binder

## 二十一.App启动&打包&安装

#### 1.应用启动流程

#### 2.apk组成和Android的打包流程

#### 3.Android的签名机制，签名如何实现的,v2相比于v1签名机制的改变

#### 4.APK的安装流程

## 二十二.序列化

#### 1.什么是序列化

#### 2.为什么需要使用序列化和反序列化

#### 3.序列化的有哪些好处

#### 4.Serializable 和 Parcelable 的区别

#### 5.什么是serialVersionUID

#### 6.为什么还要显示指定serialVersionUID的值?

## 二十三.Art & Dalvik 及其区别

#### 1.Art & Dalvik 及其区别

## 二十四.模块化&组件化

#### 1.什么是模块化

#### 2.什么是组件化

#### 3.组件化优点和方案

#### 4.组件独立调试

#### 5.组件间通信

#### 6.Aplication动态加载

#### 7.ARouter原理

## 二十五.热修复&插件化

#### 1.插件化的定义

#### 2.插件化的优势

#### 3.插件化框架对比

#### 4.插件化流程

#### 5.插件化类加载原理

#### 6.插件化资源加载原理

#### 7.插件化Activity加载原理

#### 8.热修复和插件化区别

#### 9.热修复原理

## 二十六.AOP

#### 1.AOP是什么

#### 2.AOP的优点

#### 3.AOP的实现方式,APT,AspectJ,ASM,epic,hook

## 二十七.jectpack

#### 1.Navigation

#### 2.DataBinding

#### 3.Viewmodel

#### 4.livedata

#### 5.liferecycle

## 二十八.开源框架

#### 1.Okhttp源码流程,线程池

#### 2.Okhttp拦截器,addInterceptor 和 addNetworkdInterceptor区别

#### 3.Okhttp责任链模式

#### 4.Okhttp缓存怎么处理

#### 5.Okhttp连接池和socket复用

#### 6.Glide怎么绑定生命周期

#### 7.Glide缓存机制,内存缓存，磁盘缓存

#### 8.Glide与Picasso的区别

#### 9.LruCache原理

#### 10.Retrofit源码流程,动态代理