Android 常见原理性面试专题

1.Handler **机制和底层实现**

2.Handler、Thread **和** HandlerThread **的差别**

1）Handler 线程的消息通讯的桥梁，主要用来发送消息及处理消息。

2）Thread 普通线程，如果需要有自己的消息队列，需要调用 Looper.prepare()创建Looper 实例，调用 loop()去循环消息。

3）HandlerThread 是一个带有 Looper 的线程，在 HandleThread 的run()方法中调用了 Looper.prepare()创建了 Looper 实例，并调用 Looper.loop()开启了 Loop 循环，循环从消息队列中获取消息并交由 Handler 处理。利用该线程的 Looper 创建Handler 实例，此 Handler 的handleMessage()方法是运行在子线程中的。即 Handler 利用哪个线程的 Looper 创建的实例，它就和相应的线程绑定到一起，处理该线程上的消息，它的 handleMessage()方法就是在那个线程中运行的，无参构造默认是主线程。HandlerThread 提供了 quit()/quitSafely()方法退出 HandlerThread 的消息循环，它们分别调用 Looper 的quit 和quitSafely 方法，quit 会将消息队列中的所有消息移除，而 quitSafely 会将消息队列所有延迟消息移除，非延迟消息派发出去让 Handler 去处理。

HandlerThread 适合处理本地 IO 读写操作（读写数据库或文件），因为本地 IO 操作耗时不长，对于单线程+异步队列不会产生较大阻塞，而网络操作相对比较耗时，容易阻塞后面的请求，因此 HandlerThread 不适合加入网络操作。

--handler 发消息给子线程，looper 怎么启动?

--关于 Handler，在任何地方 new Handler 都是什么线程下?

--ThreadLocal 原理，实现及如何保证 Local 属性?

--请解释下在单线程模型中 Message、Handler、Message Queue、Looper 之间的关系

--请描述一下 View 事件传递分发机制

--Touch 事件传递流程

--事件分发中的 onTouch 和 onTouchEvent 有什么区别，又该如何使用？

--View 和 ViewGroup 分别有哪些事件分发相关的回调方法

--View 刷新机制

--View 绘制流程

--自定义控件原理

--自定义 View 如何提供获取 View 属性的接口？

--Android 代码中实现 WAP 方式联网

--AsyncTask 机制

--AsyncTask 原理及不足

--如何取消 AsyncTask？

--为什么不能在子线程更新 UI？

--ANR 产生的原因是什么？

--ANR 定位和修正

--oom 是什么？

（oom(Out Of Memory)内存溢出）

--什么情况导致 oom？

--有什么解决方法可以避免 OOM？

--Oom 是否可以 try catch？为什么？

（可以，当）

--内存泄漏是什么？

内存泄露就是指该被 GC 垃圾回收的，但被一个生命周期比它长的对象仍然在引用它，导致无法回收，造成内存泄露，过多的内存泄露会导致 OOM。

--什么情况导致内存泄漏？

1）非静态内部类、匿名内部类：非静态内部类、匿名内部类 都会持有外部类的一个引用，如果有一个静态变量引用了非静态内部类或者匿名内部类，导致非静态内部类或者匿名内部类的生命周期比外部类（Activity）长，就会导致外部类在该被回收的时候，无法被回收掉，引起内存泄露，除非外部类被卸载。

解决办法：将非静态内部类、匿名内部类 改成静态内部类，或者直接抽离成一个外部类。如果在静态内部类中，需要引用外部类对象，那么可以将这个引用封装在一个WeakReference 中。

2）静态的View：当一个Activity 经常启动，但是对应的View 读取非常耗时，我们可以通过静态View 变量来保持对该Activity 的rootView 引用。这样就可以不用每次启动Activity 都去读取并渲染View 了。但View attach 到我们的Window 上，就会持有一个Context(即Activity)的引用。而我们的View 有事一个静态变量，所以导致Activity 不被回收。

解决办法： 在使用静态 View 时，需要确保在资源回收时，将静态 View detach 掉。

3）Handler：在Activity 中定义Handler 对象，那么Handler 持有Activty 的引用。而每个Message 对象是持有Handler 的引用的（Message 对象的target 属性持有Handler 引用）， 从而导致Message 间接引用到了Activity。如果在Activty destroy 之后，消息队列中还有Message 对象，Activty 是不会被回收的。

解决办法：将Handler 放入单独的类或者将 Handler 放入到静态内部类中（静态内部类不会持有外部类的引用）。如果想要在 handler 内部去调用所在的外部类 Activity，可以在 handler 内部使用弱引用的方式指向所在 Activity，在onDestory 时，调用相应的方法移除回调和删除消息。

4）监听器（各种需要注册的 Listener，Watcher 等）：当我们需要使用系统服务时，比如执行某些后台任务、为硬件访问提供接口等等系统服务。我们需要把自己注册到服务的监听器中。然而，这会让服务持有 activity 的引用，如果程序员忘记在 activity 销毁时取消注册，那就会导致 activity 泄漏了。

解决办法：在 onDestory 中移除注册

5）. 资源对象没关闭造成内存泄漏：当我们打开资源时，一般都会使用缓存。比如读写文件资源、打开数据库资源、使用 Bitmap 资源等等。当我们不再使用时，应该关闭它们，使得缓存内存区域及时回收。

解决办法：使用 try finally 结合，在 try 块中打开资源，在 finally 中关闭资源

6）. 属性动画：在使用 ValueAnimator 或者 ObjectAnimator 时，如果没有及时做 cancel 取消动画，就可能造成内存泄露。因为在 cancel 方法里，最后调用了 endAnimation(); ，在endAnimation 里，有个 AnimationHandler 的单例，会持有属性动画对象的引用。

解决办法：在 onDestory 中调用动画的 cancel 方法

7）. RxJava：在使用 RxJava 时，如果在发布了一个订阅后，由于没有及时取消，导致Activity/Fragment 无法销毁，导致的内存泄露。

解决办法：使用 RxLifeCycle

--内存泄漏和内存溢出区别?

（1）内存泄漏

1）内存泄漏：指程序中已动态分配的堆内存由于某种原因未释放或无法释放，造成系统内存的浪费，导致程序运行速度减慢甚至系统奔溃等严重后果。

2）一次内存泄漏似乎不会有大的影响，但内存泄漏后堆积的结果就是内存溢出。

3）内存泄漏具有隐蔽性，积累性的特征，比其他内存非法访问错误更难检测。这是因为内存泄漏产生的原因是内存块未被释放，属于遗漏型缺陷而不是过错型缺陷。此外，内存泄漏不会直接产生可观察的错误，而是逐渐积累，降低系统的整体性性能。

4）如何有效的进行内存分配和释放，防止内存泄漏，是软件开发人员的关键问题，比如一

个服务器应用软件要长时间服务多个客户端，若存在内存泄漏，则会逐渐堆积，导致一系列严重后果。

(2)内存溢出

指程序在申请内存时，没有足够的内存供申请者使用，或者说，给了你一块存储 int 类型数据的存储空间，但是你却存储 long 类型的数据，就会导致内存不够用，报错 OOM，即出现内存溢出的错误。

--LruCache 默认缓存大小

（4MB）

--ContentProvider 的权限管理(解答：读写分离，权限控制-精确到表级，URL 控制)

--如何通过广播拦截和 abort 一条短信？

--广播是否可以请求网络？

--广播引起 anr 的时间限制是多少？

--计算一个 view 的嵌套层级

--Activity 栈

--Android 线程有没有上限？

--线程池有没有上限？

--ListView 重用的是什么？

--Android 为什么引入 Parcelable？

--有没有尝试简化 Parcelable 的使用？