

## 常见原因

### 1.集合类

集合类如果仅仅有添加元素的方法，而没有相应的删除机制，导致内存被占用。如果这个集合类是全局性的变量(比如类中的静态属性，全局性的 map 等即有静态引用或 final 一直指向它)，那么没有相应的删除机制，很可能导致集合所占用的内存只增不减。

### 2.单例模式

不正确使用单例模式是引起内存泄露的一个常见问题，单例对象在被初始化后将在 JVM 的整个生命周期中存在(以静态变量的方式)，如果单例对象持有外部对象的引用，那么这个外部对象将不能被 JVM 正常回收，导致内存泄露

### 3.Android 组件或特殊集合对象的使用

BraodcastReceiver, ContentObserver, FileObserver, Cursor, Callback等在 Activity onDestroy 或者某类生命周期结束之后一定要 unregister 或者 close 掉，否则这个 Activity 类会被 system 强引用，不会被内存回收。

不要直接对 Activity 进行直接引用作为成员变量，如果不得不这么做，请用 private WeakReference mActivity 来做，相同的，对于Service 等其他有自己声明周期的对象来说，直接引用都需要谨慎考虑是否会存在内存泄露的可能。

### 4. Handler

要知道，只要 Handler 发送的 Message 尚未被处理，则该 Message 及发送它的 Handler 对象将被线程 MessageQueue 一直持有。由于 Handler 属于 TLS(Thread Local Storage) 变量, 生命周期和 Activity 是不一致的。因此这种实现方式一般很难保证跟 View 或者 Activity 的生命周期保持一致，故很容易导致无法正确释放。如上所述，Handler 的使用要尤为小心，否则将很容易导致内存泄露的发生。

### 5. Thread 内存泄露

线程也是造成内存泄露的一个重要的源头。线程产生内存泄露的主要原因在于线程生命周期的不可控。比如线程是 Activity 的内部类，则线程对象中保存了 Activity 的一个引用，当线程的 run 函数耗时较长没有结束时，线程对象是不会被销毁的，因此它所引用的老的 Activity 也不会被销毁，因此就出现了内存泄露的问题。

### 6.一些不良代码造成的内存压力

有些代码并不造成内存泄露，但是它们，或是对没使用的内存没进行有效及时的释放，或是没有有效的利用已有的对象而是频繁的申请新内存。

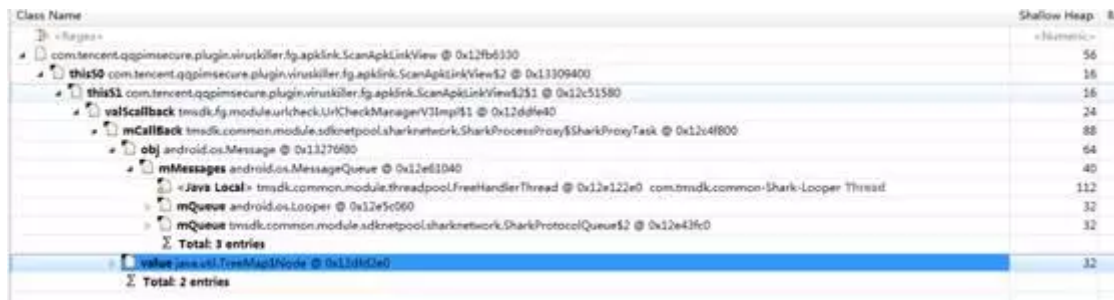
#### 6.1 Bitmap 没调用 recycle().

Bitmap 对象在不使用时,我们应该先调用 recycle() 释放内存，然后才它设置为 null. 因为加载 Bitmap 对象的内存空间，一部分是 java 的，一部分 C 的（因为 Bitmap 分配的底层是通过 JNI 调用的）。而这个 recycle() 就是针对 C 部分的内存释放。

6.2 构造 Adapter 时，没有使用缓存的 convertView。

以业务测试过程中常见的部分内存泄露实例来说明：

### 1. callback 只有 add 操作，没有注销 remove.



Class Name	Shallow Heap
com.tencent.qqpinsecure.plugin.viruskiller.fg.apklink.ScanApkLinkView @ 0x12fb6130	56
this\$0 com.tencent.qqpinsecure.plugin.viruskiller.fg.apklink.ScanApkLinkView\$2 @ 0x13309400	16
this\$1 com.tencent.qqpinsecure.plugin.viruskiller.fg.apklink.ScanApkLinkView\$2\$1 @ 0x12c51580	16
val\$callback tmsdk.fg.module.urlcheck.UrlCheckManagerV3Imp\$1 @ 0x12dd9e40	24
mCallback tmsdk.common.module.adknetpool.sharknetwork.SharkProcessProxy\$SharkProxyTask @ 0x12c4f800	88
obj android.os.Message @ 0x12176800	64
mMessages android.os.MessageQueue @ 0x12e61040	40
<Java Local> tmsdk.common.module.threadpool.FreeHandlerThread @ 0x12e122e0 com.tmsdk.common-Shark-Looper Thread	112
mQueue android.os.Looper @ 0x12e5c060	32
mQueue tmsdk.common.module.adknetpool.sharknetwork.SharkProtocolQueue\$2 @ 0x12e43fc0	32
Total: 3 entries	
val\$ java.util.TreeMap\$Node @ 0x131d12e0	32
Total: 2 entries	

从引用关系可以看到当前 view 被 callback 引用，而 callback 被外部对象 sharkprotocolQueue 持有引用而导致泄漏。

### 2. 发送延时消息时，如果该消息未处理，在退出页面后会导致该页面无法回收。

Android 应用启动的时候会创建 UI 主线程的 Looper 对象，它存在于整个应用的生命周期，用于处理消息队列里的 Message。而这些 Message 会引用发送该消息的 Handler 对象。

那么问题来了，如果这些 Handler 是 Activity 的内部类，那么当这些 Handler 的消息未处理完或者消息本身是延时消息的话，就会导致 Activity 退出后，从 Activity 到 Handler 到 Message 到 Looper 的引用链条一直存在，从而导致 Activity 的泄露！

### 3. 异步线程未完成前退出 Activity 等组件，可能会导致界面资源无法释放。

这种情况是典型的线程对象导致的内存泄露。原因也很简单，线程 Thread 对象的 run 任务未执行完之前，对象本身是不会释放的。因此 Activity 等组件对象内的线程对象成员如果有耗时任务（一般也都是耗时任务），就会导致一直持有组件本身的引用内存泄露！

本文部分内容和经验摘自网络，结合本次内存泄露的排查总结予以归纳。

## 优秀实践

9. 对 Activity 等组件的引用应该控制在 Activity 的生命周期之内；如果不能就考虑使用 `getApplicationContext` 或者 `getApplication`，以避免 Activity 被外部长生命周期的对象引用而泄露。

10. 在代码复审的时候关注长生命周期对象：全局性的集合、单例模式的使用、类的 static 变量等等。

11. 尽量不要在静态变量或者静态内部类中使用非静态外部成员变量（包括 context），即使要使用，也要考虑适时把外部成员变量置空；也可以在内部类中使用弱引用来引用外部类的变量。

12. Handler 的持有的引用对象最好使用弱引用，资源释放时也可以清空 Handler 里面的消息。比如在 Activity `onStop` 或者 `onDestroy` 的时候，取消掉该 Handler 对

象的 Message和 Runnable.

13. removeCallbacks(Runnable r) 或 removeMessages(int what), 或 removeCallbacksAndMessages (null) 等。

14. 线程 Runnable 执行耗时操作, 注意在页面返回时及时取消或者把 Runnable 写成静态类。

a) 如果线程类是内部类, 改为静态内部类。

b) 线程内如果需要引用外部类对象如 context, 需要使用弱引用。

21. 在 Java 的实现过程中, 也要考虑其对象释放, 最好的方法是在不使用某对象时, 显式地将此对象赋空, 如清空对图片等资源有直接引用或者间接引用的数组 (使用 array.clear(); array = null) , 最好遵循谁创建谁释放的原则。

小编有话说

看了技术分享文章觉得意犹未尽? 来Bugly社区和我们的技术咖聊一聊, 我们的网址是:

<http://bugly.qq.com/bbs/> (请复制此链接到浏览器打开), 或者直接点击文末的阅读原文来与我们进行互动。

**本文系腾讯Bugly独家内容, 转载请在文章开头显眼处注明注明作者和出处 “腾讯Bugly(<http://bugly.qq.com>)”**

腾讯Bugly 最专业的质量跟踪平台

精神哥、小萝莉, 为您定期分享应用崩溃解决方案



▲长按二维码可识别关注