性能优化之内存优化

# 1.基础知识

**堆栈特点对比：**

**栈**：当定义一个变量时，Java就在栈中为这个变量分配内存空间，当该变量退出该作用域后，Java会自动释放掉为该变量所分配的内存空间，该内存空间可以立即被另作他用。

**堆**：当堆中的new产生数组和对象超出其作用域后，它们不会被释放，只有在没有引用变量指向它们的时候才变成垃圾，不能再被使用。即使这样，所占内存也不会立即释放，而是等待被垃圾回收器收走。这也是Java比较占内存的原因。

**栈**：存取速度比堆要快，仅次于寄存器。但缺点是，存在栈中的数据大小与生存期必须是确定的，缺乏灵活性。

**堆**：堆是一个运行时数据区，可以动态地分配内存大小，因此存取速度较慢。也正因为这个特点，堆的生存期不必事先告诉编译器，而且Java的垃圾收集器会自动收走这些不再使用的数据。

**栈**：栈中的数据可以共享， 它是由编译器完成的，有利于节省空间。

**OOM：**

**内存泄露可以引发很多的问题：**

1.程序卡顿，响应速度慢（内存占用高时JVM虚拟机会频繁触发GC，这个可以说一下Devik虚拟机和Art虚拟机）

2.莫名消失（当你的程序所占内存越大，它在后台的时候就越可能被干掉。反之内存占用越小，在后台存在的时间就越长）

3.直接崩溃（OutOfMemoryError）

**ANDROID内存面临的问题：**

1.有限的堆内存，原始只有16M

2.内存大小消耗等根据设备，操作系统等级，屏幕尺寸的不同而不同

3.程序不能直接控制

4.支持后台多任务处理（multitasking）

5.运行在虚拟机之上

# 2.内存优化

**5R：**

本文主要通过如下的5R方法来对ANDROID内存进行优化：

**1.Reckon（计算）**

首先需要知道你的app所消耗内存的情况，知己知彼才能百战不殆

**2.Reduce（减少）**

消耗更少的资源

**3.Reuse（重用）**

当第一次使用完以后，尽量给其他的使用

**5.Recycle（回收）**

返回资源给生产流

**4.Review（检查）**

回顾检查你的程序，看看设计或代码有什么不合理的地方。

## 2.1 Reduce ：

Reduce的意思就是减少，直接减少内存的使用是最有效的优化方式。

**Bitmap：**

Bitmap是内存消耗大户，绝大多数的OOM崩溃都是在操作Bitmap时产生的，下面来看看如何几个处理图片的方法：

**优化1 使用缩略图：**

我们需要根据需求去加载图片的大小。

例如在列表中仅用于预览时加载缩略图（thumbnails ）。

只有当用户点击具体条目想看详细信息的时候，这时另启动一个fragment／activity／对话框等等，去显示整个图片

**图片大小：**

直接使用ImageView显示bitmap会占用较多资源，特别是图片较大的时候，可能导致崩溃。   
使用**BitmapFactory.Options**设置inSampleSize, 这样做可以减少对系统资源的要求。   
属性值inSampleSize表示缩略图大小为原始图片大小的几分之一，即如果这个值为2，则取出的缩略图的宽和高都是原始图片的1/2，图片大小就为原始大小的1/4。

**优化二：图片像素：**

Android中图片有四种属性，分别是：  
**ALPHA\_8：**每个像素占用1byte内存   
**ARGB\_4444：**每个像素占用2byte内存   
**ARGB\_8888：**每个像素占用4byte内存 （默认）  
**RGB\_565：**每个像素占用2byte内存

Android默认的颜色模式为ARGB\_8888，这个颜色模式色彩最细腻，显示质量最高。但同样的，占用的内存也最大。 所以在对图片效果不是特别高的情况下使用RGB\_565（565没有透明度属性）

**优化三：图片回收：**

使用Bitmap过后，就需要及时的调用**Bitmap.recycle()**方法来释放Bitmap占用的内存空间，而不要等Android系统来进行释放。

注意：对于SoftReference(软引用)或者WeakReference(弱引用)的Bitmap缓存方案，现在已经不推荐使用了。自Android2.3版本(API Level 9)开始，垃圾回收器更着重于对软/弱引用的回收，所以下面的内容可以选择忽略。

引申使用过的Glide的好处

1.默认的Bitmap格式是RGB\_565

2.加载的图片是控件的大小，在保证清晰度的情况下，尽量减少图片的内存消耗

3.对象池(Bitmap内存缓存)：这样会导致内存使用增加，Glide可以设置最大缓存，如果超过就释放不常用的

Glide原理的核心是为bitmap维护一个对象池。对象池的主要目的是通过减少大对象内存的分配以重用来提高性能。对象池的概念参见对象池的使用生命周期绑定：

图片的加载任务会与activity或者Fragment的生命周期绑定，当界面执行onStop的使用自动暂定，而当执行onStart的时候又会自动重新开启，同样的，动态Gif图的加载也是如此，以用来节省电量，同时Glide会对网络状态做监听，当网络状态发生改变时，会重启失败的任务，以减少任务因网络连接问题而失败的概率。预览图的使用

为加快加载速度，提高体验，优先加载预览图AbsListView内图片的预加载：

补充内容：Picasso会不管imageview大小是什么，总是直接缓存整张图片，而Glide就不一样了，它会为每个不同尺寸的Imageview缓存一张图片，也就是说不管你的这张图片有没有加载过，只要imageview的尺寸不一样，那么Glide就会重新加载一次，这时候，它会在加载的imageview之前从网络上重新下载，然后再缓存。Picasso不能加载gif

**内存缓存（LruCache）：**

以牺牲宝贵的应用内存为代价，内存缓存提供了快速的Bitmap访问方式。系统提供的LruCache类是非常适合用作缓存Bitmap任务的，它将最近被引用到的对象存储在一个强引用的LinkedHashMap中，并且在缓存超过了指定大小之后将最近不常使用的对象释放掉。

注意：以前有一个非常流行的内存缓存实现是SoftReference(软引用)或者WeakReference(弱引用)的Bitmap缓存方案，然而现在已经不推荐使用了。自Android2.3版本(API Level 9)开始，垃圾回收器更着重于对软/弱引用的回收，这使得上述的方案相当无效。

**硬盘缓存（DiskLruCache）：**

一个内存缓存对加速访问最近浏览过的Bitmap非常有帮助，但是你不能局限于内存中的可用图片。GridView这样有着更大的数据集的组件可以很轻易消耗掉内存缓存。你的应用有可能在执行其他任务(如打电话)的时候被打断，并且在后台的任务有可能被杀死或者缓存被释放。一旦用户重新聚焦(resume)到你的应用，你得再次处理每一张图片。

在这种情况下，硬盘缓存可以用来存储Bitmap并在图片被内存缓存释放后减小图片加载的时间(次数)。当然，从硬盘加载图片比内存要慢，并且应该在后台线程进行，因为硬盘读取的时间是不可预知的。

注意：如果访问图片的次数非常频繁，那么ContentProvider可能更适合用来存储缓存图片，例如Image Gallery这样的应用程序。

**池（PooL）**

**对象池：**

对象池使用的基本思路是：将用过的对象保存起来，等下一次需要这种对象的时候，再拿出来重复使用，从而在一定程度上减少频繁创建对象所造成的开销。 并非所有对象都适合拿来池化――因为维护对象池也要造成一定开销。对生成时开销不大的对象进行池化，反而可能会出现“维护对象池的开销”大于“生成新对象的开销”，从而使性能降低的情况。但是对于生成时开销可观的对象，池化技术就是提高性能的有效策略了。

**线程池：**

线程池的基本思想还是一种对象池的思想，开辟一块内存空间，里面存放了众多(未死亡)的线程，池中线程执行调度由池管理器来处理。当有线程任务时，从池中取一个，执行完成后线程对象归池，这样可以避免反复创建线程对象所带来的性能开销，节省了系统的资源。

比如：一个应用要和网络打交道，有很多步骤需要访问网络，为了不阻塞主线程，每个步骤都创建个线程，在线程中和网络交互，用线程池就变的简单，线程池是对线程的一种封装，让线程用起来更加简便，只需要创一个线程池，把这些步骤像任务一样放进线程池，在程序销毁时只要调用线程池的销毁函数即可。

**Recycle（回收）:**

Recycle（回收），回收可以说是在内存使用中最重要的部分。因为内存空间有限，无论你如何优化，如何节省内存总有用完的时候。而回收的意义就在于去清理和释放那些已经闲置，废弃不再使用的内存资源和内存空间。

垃圾回收机制（GC）

Thread（线程）回收：线程中涉及的任何东西GC都不能回收（Anything reachable by a thread cannot be GC'd ），所以线程很容易造成内存泄露。而且因为线程执行需要一定的时间，可能在执行的过程中，view被回收，也会导致空指针的问题；相互持有对象，可能还会导致无法被释放，例如Mvp模式中，vp相互持有对象，在onDestroy及时释放对象

Receiver（接收器）回收；Stream/File（流/文件）及时关闭流

参考网站

ANDROID内存优化(大汇总——上)：https://blog.csdn.net/a396901990/article/details/37914465

<https://www.cnblogs.com/yezhennan/p/5442557.html>

启动的优化：https://blog.csdn.net/qian520ao/article/details/81908505