Android内存管理之道

内存泄漏：对象在内存heap堆中中分配的空间，当不再使用或没有引用指向的情况下，仍不能被GC正常回收的情况。多数出现在不合理的编码情况下，比如在Activity中注册了一个广播接收器，但是在页面关闭的时候进行unRegister，就会出现内存溢出的现象。通常情况下，大量的内存泄漏会造成OOM。

OOM：即OutOfMemoery，顾名思义就是指内存溢出了。内存溢出是指APP向系统申请超过最大阀值的内存请求，系统不会再分配多余的空间，就会造成OOM error。在我们Android平台下，多数情况是出现在图片不当处理加载的时候。

内存管理之道嘛，无非就是先理解并找出内存泄漏的原因，再基于这些反式去合理的编码，去防范进而避免内存开销过大的情形。学习如何合理的管理内存，最好先了解内存分配的机制和原理。只有深层次的理解了内部的原理，才能真正避免OOM的发生。但是本文就不介绍Jvm/Davilk内存分配的机制了，如有兴趣，请查看历史消息，以前做过题为《JVM运行时数据区域分析》的分享。

Android APP的所能申请的最大内存大小是多少，有人说是16MB，有人又说是24MB。这种事情，还是亲自用自己的手机测试下比较靠谱。测试方式也比较简单，Java中有个Runtime类，主要用作APP与运行环境交互，APP并不会为我们创建Runtime的实例，但是Java为我们提供了单例获取的方式Runtime.getRuntime()。通过maxMemory()方法获取系统可为APP分配的最大内存，totalMemory()获取APP当前所分配的内存heap空间大小。我手上有两部手机，一部Oppo find7，运行Color OS，实测最大内存分配为192MB；一部天语v9，运行小米系统，实测最大内存分配为100MB。这下看出点眉目了吧，由于Android是开源系统，不同的手机厂商其实是拥有修改这部分权限能力的，所以就造成了不同品牌和不同系统的手机，对于APP的内存支持也是不一样的，和IOS的恒久100MB是不同的。一般来说，手机内存的配置越高，厂商也会调大手机支持的内存最大阀值，尤其是现在旗舰机满天发布的情况下。但是开发者为了考虑开发出的APP的内存兼容性，无法保证APP运行在何种手机上，只能从编码角度来优化内存了。

下面我们逐条来分析Android内存优化的关键点。

1、万恶的static

static是个好东西，声明赋值调用就是那么的简单方便，但是伴随而来的还有性能问题。由于static声明变量的生命周期其实是和APP的生命周期一样的，有点类似与Application。如果大量的使用的话，就会占据内存空间不释放，积少成多也会造成内存的不断开销，直至挂掉。static的合理使用一般用来修饰基本数据类型或者轻量级对象，尽量避免修复集合或者大对象，常用作修饰全局配置项、工具类方法、内部类。

2、无关引用

很多情况下，我们需求用到传递引用，但是我们无法确保引用传递出去后能否及时的回收。比如比较有代表性的Context泄漏，很多情况下当Activity结束掉后，由于仍被其他的对象指向导致一直迟迟不能回收，这就造成了内存泄漏。这时可以考虑第三条建议。

3、善用SoftReference/WeakReference/LruCache

Java、Android中有没有这样一种机制呢，当内存吃紧或者GC扫过的情况下，就能及时把一些内存占用给释放掉，从而分配给需要分配的地方。答案是肯定的，java为我们提供了两个解决方案。如果对内存的开销比较关注的APP，可以考虑使用WeakReference，当GC回收扫过这块内存区域时就会回收；如果不是那么关注的话，可以使用SoftReference，它会在内存申请不足的情况下自动释放，同样也能解决OOM问题。同时Android自3.0以后也推出了LruCache类，使用LRU算法就释放内存，一样的能解决OOM，如果兼容3.0一下的版本，请导入v4包。关于第二条的无关引用的问题，我们传参可以考虑使用WeakReference包装一下。

4、谨慎handler

在处理异步操作的时候，handler + thread是个不错的选择。但是相信在使用handler的时候，大家都会遇到警告的情形，这个就是lint为开发者的提醒。handler运行于UI线程，不断处理来自MessageQueue的消息，如果handler还有消息需要处理但是Activity页面已经结束的情况下，Activity的引用其实并不会被回收，这就造成了内存泄漏。解决方案，一是在Activity的onDestroy方法中调用

handler.removeCallbacksAndMessages(null);取消所有的消息的处理，包括待处理的消息；二是声明handler的内部类为static。

5、Bitmap终极杀手

Bitmap的不当处理极可能造成OOM，绝大多数情况都是因这个原因出现的。Bitamp位图是Android中当之无愧的胖小子，所以在操作的时候当然是十分的小心了。由于Dalivk并不会主动的去回收，需要开发者在Bitmap不被使用的时候recycle掉。使用的过程中，及时释放是非常重要的。同时如果需求允许，也可以去BItmap进行一定的缩放，通过BitmapFactory.Options的inSampleSize属性进行控制。如果仅仅只想获得Bitmap的属性，其实并不需要根据BItmap的像素去分配内存，只需在解析读取Bmp的时候使用BitmapFactory.Options的inJustDecodeBounds属性。最后建议大家在加载网络图片的时候，使用软引用或者弱引用并进行本地缓存，推荐使用android-universal-imageloader或者xUtils，牛人出品，必属精品。前几天在讲《自定义控件（三） 继承控件》的时候，也整理一个，大家可以去Github下载看看。

6、Cursor及时关闭

在查询SQLite数据库时，会返回一个Cursor，当查询完毕后，及时关闭，这样就可以把查询的结果集及时给回收掉。

7、页面背景和图片加载

在布局和代码中设置背景和图片的时候，如果是纯色，尽量使用color；如果是规则图形，尽量使用shape画图；如果稍微复杂点，可以使用9patch图；如果不能使用9patch的情况下，针对几种主流分辨率的机型进行切图。

8、ListView和GridView的item缓存

对于移动设备，尤其硬件参差不齐的android生态，页面的绘制其实是很耗时的，findViewByld也是蛮慢的。所以不重用View，在有列表的时候就尤为显著了，经常会出现滑动很卡的现象。具体参照历史文章《说说ViewHolder的另一种写法》

9、BroadCastReceiver、Service

绑定广播和服务，一定要记得在不需要的时候给解绑。

10、I/O流

I/O流操作完毕，读写结束，记得关闭。

11、线程

线程不再需要继续执行的时候要记得及时关闭，开启线程数量不易过多，一般和自己机器内核数一样最好，推荐开启线程的时候，使用线程池。

12、String/StringBuffer

当有较多的字符创需要拼接的时候，推荐使用StringBuffer。