# 应用程序内存优化

对于开发一个应用程序来说，在前期完成主要功能之后，后期有一项非常重要的工作，那就是优化应用程序的内存。而内存优化的方向主要是两个，一个是内存溢出，另外一个就是内存泄露。

内存溢出和内存泄露是两个不同的概念。内存溢出指的是内存不够用了，内存的使用超过的系统规定的最大值。而内存泄露指的是由于程序的逻辑错误，导致一些没有用的对象无法被垃圾回收器回收而一直占用着内存。所以内存泄露的堆积可能会造成内存溢出。而内存溢出不一定都是由内存泄露引起的。这两个概念要搞清楚。

## 内存溢出：

在安卓的应用程序中，内存溢出主要提要在几个方面

**1、ListView的显示**

我们在使用ListView的时候，都会给他设置Adapter，如果在Adapter中的getView方法 中，我们没有复用convertView，就会造成在滑动ListView的时候，会为每一个item都 生成一个View对象，而不管这个item之前是否已经生成过View对象。如果来回滑动 的次数太多的话，就会造成View生成的数量太多，最终会造成内存溢出。

如果ListView中的item是包含图片的，那么，如果在快速滑动的过程中，我们就去为 item加载图片，此时非常容易造成内存溢出。因为，在快速滑动的过程中，垃圾回收 器还来不及回收内存，而新的item又需要新的内存来显示图片。所以，在这种情况下， 一般的做法是监听ListView的滚动状态，当ListView的滚动状态为空闲的情况下，里面 的每一个Item才去加载图片。

**2、加载图片相关**

a、加载多张图片

对于加载多张图片，我们一般会使用三级缓存来实现图片的加载。内存缓存、本地 缓存、网络缓存。缓存的目的是为了下一次加载速度更快。所以在内存中保存着一 定数量的图片是有助于下一次图片显示的速度。但是，内存中不能保存太多的图片 对象，所以我们使用LRUCache来保存内存中的图片，并且控制在一定的数量之内。

b、加载单张图片

这种情况是针对于某一张图片特别大的情况。如果一张图片非常非常大，如果50M， 那么，我只要一去加载它，那么我的程序肯定就会挂，根本还没使用到三级缓存应 用程序就受不了了。所以，对于大图片的显示需要特殊处理。因为图片虽然特别大， 但是这个图片所需要显示的控件有可能是很小的。我们可以先把图片的宽和高得 到，再得到这张图片所需要显示的控件的宽高，就可以得到图片和控件的缩放比例 了。最后，根据缩放比例，设置图片的采样率，来减小单张图片的内存占用。

1. BitmapFactory.Options options = new BitmapFactory.Options();
2. //只得到图片的大小，不去加载图片的内容
3. options.inJustDecodeBounds = true;
4. Bitmap boundBitmap = BitmapFactory.decodeFile(filePath,options);
5. int imageWidth = boundBitmap.getWidth();
6. int imageHeight = boundBitmap.getHeight();
7. //计算inSampleSize的值 此时可以根据控件大小和图片大小来得到缩放比例 这里先写成2
8. options.inSampleSize = 2;//图片宽高都为原来的二分之一，即图片为原来的四分之一
9. options.inJustDecodeBounds = false;
10. Bitmap bitmap = BitmapFactory.decodeFile(filePath,options);

## 内存泄露：

内存泄露是因为有些对象已经没有用了，但是却无法被垃圾回收器回收内存，导致这一块内存泄露了。随着这类问题的堆积，泄露的内存越来越大，可使用的内存越来越小，直到内存溢出。

而垃圾回收器无法回收某一些对象，是因为，这些对象有人引用它了。所以内存泄露的本质就是，生命周期短的对象一直被生命周期长的对象引用，导致生命周期短的对象无法被垃圾回收器回收。

在安卓的应用程序当中，内存泄露主要体现在几个方面

**1、注册了之后忘了反注册**

BraodcastReceiver，ContentObserver，FileObserver在Activity onDestory或者某类声明 周期结束之后一定要unregister掉，否则这个Activity会被system强引用，不会被内存 回收。对于观察者模式的写法，在注册某一些观察者的时候，要既得在适当的时候，把 对应的观察者从观察者列表中移除。否则存储观察者的集合列表会不断的扩大。

**2、资源对象没关闭**

　 资源性对象比如Cursor，File文件等，在内部的实现机制中都用了一些缓冲，所以在不 使用的时候这些资源对象的时候应该及时关闭它们，这样，他们的缓冲区域才能被既时 候回收。它们的缓冲不仅存在于Java虚拟机内，还存在于Java虚拟机外。如果我们仅 仅是把它的引用设置为null,而不关闭它们，往往会造成内存泄露。我们需要在这些对象 不使用的情况下，立即调用close方法，然后再设置为null。

**3、activity被集合引用导致activity不能释放**

一般我们在应用程序中，做一个退出应用程序的功能。当点击退出按钮的时候，此时应 该把这个应用中所有activity都finish掉，这样就可以退出这个应用了。所以，我们需 要在每一个activity中的onCreate方法里，将当前启动的activity保存在某一个集合列 表里面。这个集合列表可以存在于Application中的一个ArrayList。此时，如果在activity 中的onDestory没有将当前的activity对象从集合列表中移除的话，那么就会造成activity 虽然退出了，但是这个activity的对象依然无法被系统回收，因为有一个集合一直引用 着这个activity对象。所以一定要记得在onDestory方法中把当前的activity对象从集合 列表中移除。

**4、Bitmap对象不在使用时调用recycle()释放内存**

Bitmap非常的耗内存。 多使用几个Bitmap很可能一下子就会超过Java堆的限制。因此，在 用完Bitmap时，要 及时的recycle掉。recycle并不能立即将Bitmap的内存释放，但是会在垃 圾回收器下一个工作的时机将这个bitmap回收。

## 解决方案

无论是内存泄露还是内存溢出，最终的后果基本上是一致的，那就是造成应用程序强行关闭。在应用程序的功能开发完之后，怎么样才能确定应用程序有没有内存的问题呢？又怎样来确定到底是哪一块代码出的问题呢？接下来我们就来说说关于内存问题的解决方案。

**1、使用monkey工具**

因为有些内存问题藏的比较深，要长期使用才能出现异常。所以可以使用monKey工具 来对我们的应用程序进行压力测试。

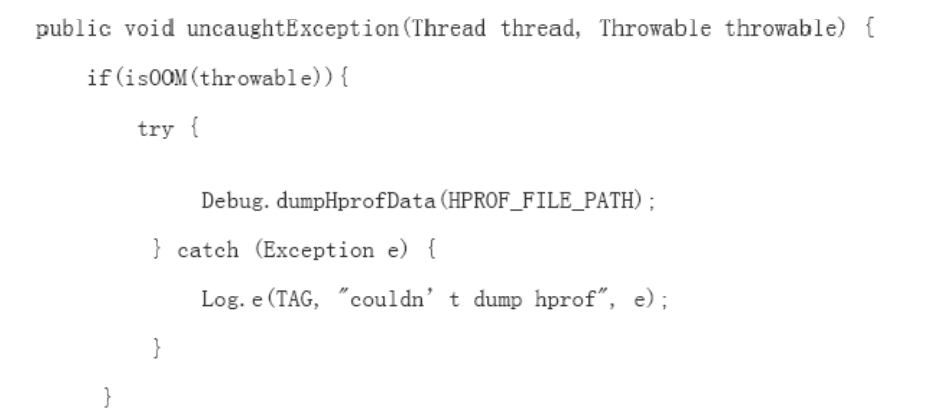
模拟200000次用户操作的参考命令： adb shell monkey -s 23 -p cn.itcast.XXX(所测试的 包) --ignore-crashes --ignore-timeouts -v -v -v 200000 > D:\文件名.log

回车之后，我们所需要测试的应用程序就启动了，并且还有不断触摸事件被促发，程序 生成的log会保存在D目录下。

**2、捕获OOM异常**

自定义Application并让它实现UncaughtExceptionHandler 接口，在onCreate方法中让 自己成为系统的默认异常处理机制。Thread.setDefaultUncaughtExceptionHandler(this);

这样子设置之后，如果应用程序出现异常，就会调用Application中的uncaughtException 方法。我们就在这个方法中判断异常是否为OOM异常。如果是OOM异常，就将内存 快照导到sd卡中去。



这样子之后，我们就可以在sd卡上找到发生异常时的内存快照

**3、分析内存快照**

使用MAT工具对内存快照文件进行分析。

在使用MAT工具打开内存快照文件的时候，需要先将文件进行转换一下。因为MAT只 能识别JAVA的内存快照，而我们安卓的内存快照和JAVA的内存快照有点不太一样，所 以需要进行转化。

例如：



接下来就是使用MAT工具打开转化过后的内存快照文件了。通过MAT工具可以看出这 个内存快照有几个对象，对象之间的引用关系是怎样的。这样对我们定位内存问题非常 有帮助。

一般在应用程序开发的后期会使用这三个步骤来增强应用程序的健壮性。