# 前言

我们要说点什么：

1.我们应该怎么加载一个大图片，从而避免OOM（我喜欢stackoverflow.com的名字）

2.我们是否应该鄙视：setbackgroundresource等直接通过一个id构建bitmap/drawable的方法

3.一些参考资料

# 正文

## 加载那个大图片

你的图片有多大，在我看来，一张40K+的图片，已经被称为大图片了

当你的程序，通过view.setbackgroundxxx来频繁的（100ms一张图片，或者，更短的ms）更换图片时（或者，你采用surfaceview的style），上述的40K+已经非常的大了

你应该选择什么API来显示你的图片：

bitmapdrawable，而不是setbackgroundresource(int id)

为什么：通过一个id来获取一个bitmap/drawable时，android系统会对该资源进行缓存，从而下次访问迅速响应，但是，它只缓存，不释放，再多的memory也不够用，进而OOM

bitmapdrawable的创建过程很节省时间吗？：

我曾经做过一个实验，在无bitmapfactory.options的时候，反复的加载一个40K+的图片1000次，算的其平均加载一张图片的时间为：68ms

当你选择设置了options.inSampleSize = 2时（width和height均为原来的1/2），你会得到平均30ms左右的加载时间

当然，我不建议使用inSampleSize，因为它使得图片的尺寸进行了缩小，如果你的view是wrap\_content，那么你要进行额外的处理

一个最佳实践：

InputStream is = this.getResources().openRawResource(your\_drawable\_id);  
BitmapFactory.Options opt = new BitmapFactory.Options();  
opt.inPreferredConfig = Bitmap.Config.RGB\_565;  
opt.inPurgeable = true;  
opt.inInputShareable = true;  
Bitmap bitmap = BitmapFactory.decodeStream(is, null, opt);  
findViewById(your\_view\_id).setBackgroundDrawable(new BitmapDrawable(getResources(), bitmap));

通过上述方法，每次加载40K+的图片，平均只需要26ms，且没有改变图像的尺寸（别被骗了，你的图片规格有可能降低了：RGB\_565，但是，还好）

### 看看decodestream

使用

view.setBackgroundResource,imageview.setImageResource,bitmapFactory.decodeResource这样的方法来设置一张大图片的时候，

这些函数在完成decode后，最终都是通过java层的createBitmap来完成的，需要消耗更多内存。

decodeStream最大的秘密在于其直接调用JNI>>nativeDecodeAsset()来完成decode，无需再使用java层的createBitmap，从而节省了java层的空间。

如果在读取时加上图片的Config参数，可以跟有效减少加载的内存，从而跟有效阻止抛out of Memory异常

注意：decodeStream直接拿的图片来读取字节码了，不会根据机器的各种分辨率来自动适应

### 别让你的程序卡掉，或者，抛出异常

作为一个单独的小结，我理解，上述问题是多么的重要，以及，令人抓狂

你可以试试每隔60ms来播放1万张40K+图片，并且按照上述方法来进行展示

但是，依然，程序能挂掉，或者，屏幕已经全黑了（你会很确定，你的程序没有死掉，但是，你看到的一切全黑了）

你查看logcat，你会发现：imageref\_ashmem create failed <(null)>

可以google，或者stackoverflow，它会告诉你，你需要释放bitmap

你可能会反问，都不会再OOM了，为什么还要释放bitmap，答案，很简单，因为黑屏，因为异常（开玩笑的，如果你知道更确认的原因，请告诉我，请尽量使用通俗易懂的语言，3x）

如何释放一个bitmap：bitmap.recycle()

bitmap会立刻释放吗：不会，但是会在某个时刻被释放

它会立刻引发GC吗：不会（说到这里，如果在你的程序里，频繁的调用gc，或 者被动的被调用gc（如：频繁的调用setbackgroundresource，且每次都是个大图片），那么你将有幸看到你的程序离迟钝只有一步之遥 了，所以，请不要这样做，因为，它只会让性能更差，你可以实验一下）

它会引发什么样的问题：你可能有机会看到这样一个异常：{你引用了一个已经被recycled的bitmap，然后程序就这样挂了}，所以，请你考虑应该进行释放操作的时机，给一个参考：你在显示当前图片的时候，也许，你愿意将之前的图片给释放掉

## 我们该鄙视诸如setbackgroundresource此类吗

的确，setbackgroundresource让人抓狂：它缓存了太多东西，导致OOM

但是，它的存在是为了它的另外一面：当下次访问该图片的时候，能快速响应

接下来的一个问题，下次访问是在什么时候，停100ms再访问这个图片算很长吗？

不幸的是，是的，100ms再访问，会显的很长

当你在编写一个撞球的游戏，每100ms刷新一次屏幕，是那么的漫长，这时， 你巴不得刚设置完view的img后，就立刻开始下一次view的img设置（因为android的刷新是有时间的，所以，这样做，是有意义的）（啊，这 个球很快，这一秒还在这，下一秒已经在那了，太快了，瞬间转移，也许，你也认为100ms一帧是漫长的）

从bitmapdrawable中实验来看，图片加载的时间为20ms，那么，这个时间很快吗？有直接从缓存里面取来的快吗？

所以，我们不应该鄙视

## 一些参考资料

google上有很多的处理OOM的资料，在此，我贴出一些：

http://blog.sina.com.cn/s/blog\_5de73d0b010117ix.html

http://www.cnblogs.com/siyiganshou/

http://blog.csdn.net/imyfriend/article/details/8039767