深入 Android 大图片处理机制

## 图片是1M，真的在内存中差不多也会占用大概1M吧？

一张图片到底占用多大内存？肯定需要根据图片的像素尺寸进行计算，假设一个像素占4个字节（默认是：ARGB\_8888）。

**占用内存 = 横向像素数 x 纵向像素数 x 每个像素占用内存（默认是4）**

例如：现在有个1M的图片尺寸是 3500 x 2500，计算 3500 x 2500 x 4 / 1024 / 1024 = 31M，真是没想到啊。。。1张图片足以上你的app抛出一个OOM。

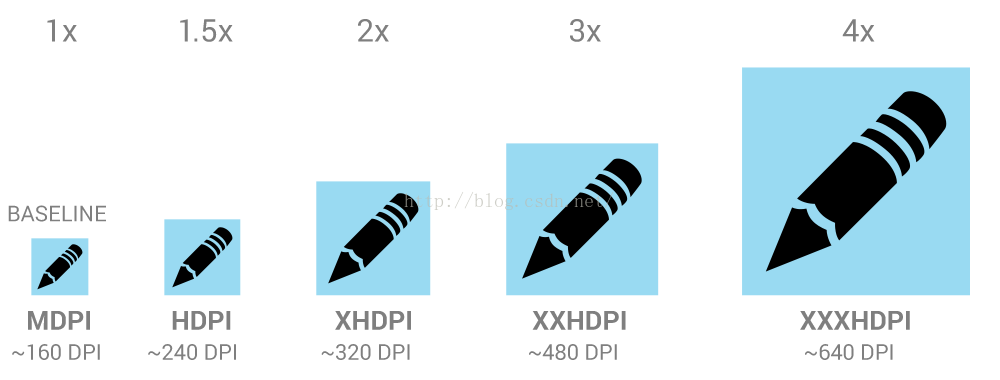
原因猜测是该图片是jpg格式，使用的相应的压缩算法，所以是图片很小，而在Android使用Bitmap时需要原始的数据，所以需要按照上面的公式进行计算。

# 放错对应的分辨率文件夹有那么严重吗？

如果你把一张图片放在了mdpi，而用户的手机是xhdpi的话，你会发现你的图片实际所占用的内存会比计算值（按照上面的方法做计算）多4倍。所以如果你把上面那张图片放在了mdpi目录下面，这简直就是自残啊：占用内存120M。

这是因为安卓系统会把mdpi的图片尺寸进行扩大以保证显示的尺寸相同。而相反的如果你把图片放到了xhdpi，而你的手机是mdpi的话，你的图片会被压缩成原来的1/4。

具体的比例关系：（NX 表示表示长和宽是mdpi的N倍，总内存就是N\*N倍）



所以图片可不能乱放。小小的操作会吃掉成倍的内存。因此有如下解决方法：

1. 针对不同分辨率切图。
2. 现的手机基本是xhdpi及以上，这也是为什么很多商业项目在切图的时候只会对应xhdpi的手机。
3. 大图片压缩的时候一定要全部加载到内存再进行压缩吗？

当然不是，可以使用BitmapFactory.decode()，传入一个Options对象，进行某些设置即可读取到该图片的宽高、类型，而不需要读取整个图片到内存。读取了图片的宽和高之后，可以按照想要的比例进行压缩读取。我的理解是如果你设置的压缩比是2x2（横向纵向像素各压缩2倍，总压缩比4倍），在BitmapFactory进行decode的时候知道了你的压缩比，他会选择性的读取某些像素点（具体算法未知，可以简单的想成比如原图是100x100 px，现在我们获得到了他的输入流，只是平均的从输入流读取50x50 px像素），这样就实现了大图片不加载到内存先压缩。

这种做法可以用在本地图片的显示处理，同时也可以用在加载网络图片的时候。都是为了避免OOM，同时实现对大图片进行压缩。

具体demo如下：

**public class MainActivity extends AppCompatActivity {**

**ImageView imageView;**

**Handler handler=new Handler(){**

**@Override**

**public void handleMessage(Message msg) {**

**super.handleMessage(msg);**

**imageView.setImageBitmap((Bitmap) msg.obj);**

**}**

**};**

**@Override**

**protected void onCreate(Bundle savedInstanceState) {**

**super.onCreate(savedInstanceState);**

**setContentView(R.layout.\*activity\_main\*);**

**imageView= (ImageView) findViewById(R.id.\*imageview\_main\_activity\_image\*);**

**setBitmapFromNet();**

**}**

**public Bitmap setBitmapFromDisk(){**

**BitmapFactory.Options options=new BitmapFactory.Options();**

**//设置只加载图片的格式尺寸信息到内存，不加载具体的图片字节。**

**options.inJustDecodeBounds=true;**

**Bitmap bitmap=BitmapFactory.\*decodeResource\*(getResources(),R.drawable.\*big\*,options);**

**//获取图片的高度和宽度**

**int height=options.outHeight;**

**int width=options.outWidth;**

**//获取图片的类型**

**String imageType = options.outMimeType;**

**Log.\*d\*("imageTest",height+" "+width+" "+imageType);**

**System.\*out\*.println(height+" "+width+" "+imageType);**

**//长&&宽压缩的比例,内存占用的比例关系是平方倍**

**options.inSampleSize=2;**

**options.inJustDecodeBounds=false;**

**bitmap=BitmapFactory.\*decodeResource\*(getResources(),R.drawable.\*big\*,options);**

**return bitmap;**

**}**

**public void setBitmapFromNet(){**

**Thread t =new Thread(){**

**@Override**

**public void run() {**

**super.run();**

**try {**

**URL url=new URL("[http://img.taopic.com/uploads/allimg/130501/240451-13050106450911.jpg](http://img.taopic.com/uploads/allimg/130501/240451-13050106450911.jpg)");**

**HttpURLConnection connection= (HttpURLConnection) url.openConnection();**

**System.\*out\*.println(" code :"+connection.getResponseCode());**

**BitmapFactory.Options options=new BitmapFactory.Options();**

**//设置只加载图片的格式尺寸信息到内存，不加载具体的图片字节。**

**options.inJustDecodeBounds=true;**

**//获取图片的高度和宽度**

**InputStream is=connection.getInputStream();**

**Bitmap bitmap=BitmapFactory.\*decodeStream\*(is,null,options);**

**int height=options.outHeight;**

**int width=options.outWidth;**

**//获取图片的类型**

**String imageType = options.outMimeType;**

**Log.\*d\*("imageTest",height+" "+width+" "+imageType);**

**System.\*out\*.println(height+" "+width+" "+imageType);**

**connection= (HttpURLConnection) url.openConnection();**

**options.inJustDecodeBounds=false;**

**options.inSampleSize=2;**

**//获取图片的高度和宽度**

**bitmap=BitmapFactory.\*decodeStream\*(connection.getInputStream(),null,options);**

**Message m=new Message();**

**m.obj=bitmap;**

**handler.sendMessage(m);**

**} catch (MalformedURLException e) {**

**e.printStackTrace();**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**};**

**t.start();**

**}**

**}**

更详细的压缩步骤如下。

首先请看一行代码：  
mImageView.setImageResource(R.drawable.my\_image)

这是一行从资源文件中加载图片到ImageView的代码。通常这段代码没什么问题，但有些情况下，你需要对这段代码进行优化。例如当图片的尺寸远远大于ImageView的尺寸时，或者当你要在一个ListView或GridView中批量加载一些大小未知的图片时。实际上，以上这行代码会在运行时使用BitmapFactory.decodeStream()方法将资源图片生成一个Bitmap，然后由这个Bitmap生成一个Drawable，最后再将这个Drawable设置到ImageView。由于在过程中生成了Bitmap，因此如果你使用的图片过大，就会导致性能和内存占用的问题。另外，需要优化的情形不止这一种，这里就不再列举。

下面分步说明使用代码来减小Bitmap的尺寸从而达到减小内存占用的方法。

**1. 获取原图片尺寸**

通常我们使用BitmapFactory.decodeResource()方法来从资源文件中读取一张图片并生成一个Bitmap。但如果使用一个BitmapFactory.Options对象，并把该对象的inJustDecodeBounds属性设置为true，decodeResource()方法就不会生成Bitmap对象，而仅仅是读取该图片的尺寸和类型信息：

**BitmapFactory.Options options = new BitmapFactory.Options();**

**options.inJustDecodeBounds = true;**

**BitmapFactory.decodeResource(getResources(), R.id.myimage, options);**

**intimageHeight = options.outHeight;**

**intimageWidth = options.outWidth;**

**String imageType = options.outMimeType;**

**2. 根据原图尺寸和目标区域的尺寸计算出合适的Bitmap尺寸**

BitmapFactory.Options类有一个参数inSampleSize，该参数为int型，他的值指示了在解析图片为Bitmap时在长宽两个方向上像素缩小的倍数。inSampleSize的默认值和最小值为1（当小于1时，解码器将该值当做1来处理），且在大于1时，该值只能为2的幂（当不为2的幂时，解码器会取与该值最接近的2的幂）。

例如，当inSampleSize为2时，一个2000 x 1000的图片，将被缩小为1000 x 500，相应地，它的像素数和内存占用都被缩小为了原来的1/4：

**publicstatic int calculateInSampleSize(**

**BitmapFactory.Options options, int reqWidth, int reqHeight) {**

**// 原始图片的宽高**

**finalint height = options.outHeight;**

**finalint width = options.outWidth;**

**intinSampleSize = 1;**

**if(height > reqHeight || width > reqWidth) {**

**finalint halfHeight = height / 2;**

**finalint halfWidth = width / 2;**

**// 在保证解析出的bitmap宽高分别大于目标尺寸宽高的前提下，取可能的inSampleSize的最大值**

**while((halfHeight / inSampleSize) > reqHeight**

**&& (halfWidth / inSampleSize) > reqWidth) {**

**inSampleSize \*= 2;**

**}**

**}**

**returninSampleSize;**

**}**

**3. 根据计算出的inSampleSize生成Bitmap**

**publicstatic Bitmap decodeSampledBitmapFromResource(Resources res, int resId,**

**intreqWidth, int reqHeight) {**

**// 首先设置 inJustDecodeBounds=true 来获取图片尺寸**

**finalBitmapFactory.Options options = new BitmapFactory.Options();**

**options.inJustDecodeBounds = true;**

**BitmapFactory.decodeResource(res, resId, options);**

**// 计算 inSampleSize 的值**

**options.inSampleSize = calculateInSampleSize(options, reqWidth, reqHeight);**

**// 根据计算出的 inSampleSize 来解码图片生成Bitmap**

**options.inJustDecodeBounds = false;**

**returnBitmapFactory.decodeResource(res, resId, options);**

**}**

这里有一点要注意，就是要在第二遍decode之前把inJustDecodeBounds设置回false。

**4. 调用以上的decodeSampledBitmapFromResource方法，使用自定尺寸的Bitmap**

如果你要将一张大图设置为一个100\*100的缩略图，执行以下代码：

**mImageView.setImageBitmap(decodeSampledBitmapFromResource(**

**getResources(), R.id.myimage, 100,100));**

到此，使用decodeResource()方法将一个大图解析为小尺寸bitmap的应用就完成了。同理还可以使用decodeStream()、decodeFile()等方法做相同的事，原理是一样的。