图片处理优化

一、大图片的加载显示(避免 OOM 问题)

1. 问题: 如果将大图片加载到内存中来显示, 可能会导致内存溢出(OOM)
2. 解决思路: 对图片进行压缩加载(本质上只是读取了图片文件的部分数据)
3. 具体办法:
   1) 得到图片的宽高的方式:

```
BitmapFactory.Options options = new BitmapFactory.Options();
options.inJustDecodeBounds = true;  //设置只读取图片文件的边框,这样就不会加载整个图片文件
BitmapFactory.decodeResource(getResources(), R.id.myimage, options);
int imageHeight = options.outHeight;  //图片的高
int imageWidth = options.outWidth;  //图片的宽
String imageType = options.outMimeType;
```

注意: 为了避免 OOM 异常，最好在解析每张图片的时候都先检查一下图片的大小，除非你非常信任图片的来源，保证这些图片都不会超出你程序的可用内存

   2) 计算取样比例的方式:

```
public static int calculateInSampleSize(BitmapFactory.Options options,
        int reqWidth, int reqHeight) {
    // 源图片的高度和宽度
    final int height = options.outHeight;
    final int width = options.outWidth;
    int inSampleSize = 1;
    if (height > reqHeight || width > reqWidth) {
        // 计算出实际宽高和目标宽高的比率
        final int heightRatio = Math.round((float) height / (float) reqHeight);
        final int widthRatio = Math.round((float) width / (float) reqWidth);
        // 选择宽和高中最小的比率作为inSampleSize的值，这样可以保证最终图片的宽和高
        // 一定都会大于等于目标的宽和高。
        inSampleSize = heightRatio < widthRatio ? heightRatio : widthRatio;
    }
    return inSampleSize;
```

注意: 如果 inSampleSize=3, 表示 宽和高上只读取原来数据的 1/3, 这样整体大小压缩为原来的 1/9

   3) 整体处理:

```
public static Bitmap decodeSampledBitmapFromResource(Resources res, int resId,
        int reqWidth, int reqHeight) {
    // 第一次解析将inJustDecodeBounds设置为true，只获取图片宽,高大小
    final BitmapFactory.Options options = new BitmapFactory.Options();
    options.inJustDecodeBounds = true; //不会加载整个图片文件
    BitmapFactory.decodeResource(res, resId, options);
    // 调用上面定义的方法计算inSampleSize值
    options.inSampleSize = calculateInSampleSize(options, reqWidth, reqHeight);
    // 使用获取到的inSampleSize值再次解析图片
    options.inJustDecodeBounds = false; //会根据inSampleSize加载图片的部分数据到内存
    return BitmapFactory.decodeResource(res, resId, options);
}
```

调用代码如下:

```
mImageView.setImageBitmap(decodeSampledBitmapFromResource(getResources(), R.id.myimage, 100, 100));
//将任意一张图片压缩成100*100的缩略图，并在ImageView上展示
```

二、缓存图片对象

1. 在内存中缓存图片对象(Bitmap)，不要直接用 Map<String, Bitmap>类型的容器，因为这样会导致 bitmap 对象太多太大时而没有去释放，最终导致 OOM
2. 在 Android2.3 之前，一般都用 Map<String, SoftReference<Bitmap>>结构容器来缓存 Bitmap 对象，这样在内存不太充足时，垃圾回收器会将软引用对象释放.但从 2.3 开始，垃圾回收器可能在正常情况下就回收软引用对象，这样会降低缓存的效果
3. Android 的 v4 兼容包中提供了 LruCache 来做缓存容器，它的基本原理为:把最近使用的对象用强引用存储在 LinkedHashMap 中，并且把最近最少使用的对象在缓存值达到预设定值之前从内存中移除

```
private LruCache<String, Bitmap> mMemoryCache; //缓存Bitmap的容器

@Override
protected void onCreate(Bundle savedInstanceState) {
    // 获取到可用内存的最大值，使用内存超出这个值会引起OutOfMemory异常。
    // LruCache通过构造函数传入缓存值，以KB为单位。
    int maxMemory = (int) (Runtime.getRuntime().maxMemory() / 1024);
    // 使用最大可用内存值的1/8作为缓存的大小。
    int cacheSize = maxMemory / 8;
    mMemoryCache = new LruCache<String, Bitmap>(cacheSize) {
        @Override
        protected int sizeOf(String key, Bitmap bitmap) {
            // 重写此方法来衡量每张图片的大小，默认返回图片数量。
            return bitmap.getByteCount() / 1024;
        }
    };
}

public void addBitmapToMemoryCache(String key, Bitmap bitmap) {
    if (getBitmapFromMemCache(key) == null) {
        mMemoryCache.put(key, bitmap);
    }
}

public Bitmap getBitmapFromMemCache(String key) {
    return mMemoryCache.get(key);
}
```

三、主动释放 bitmap 对象

```
if(!bmp.isRecycle() ){
   bmp.recycle()     // 回收图片所占的内存
   system.gc()       // 提醒系统及时回收
}
```

四、设置加载图片的色彩模式

1. Android 中有四种，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
2. 默认的模式显示的图片质量是最高的，但也是占用内存最大的，而如果使用 ARGB_4444 模式，占用的内存就会减少为 1/2，但显示效果差别不明显

```
BitmapFactory.Options options = new BitmapFactory.Options ( ) ;
options.inPreferredConfig = Bitmap.Config.ARGB_4444;
Bitmap img = BitmapFactory.decodeFile ("/sdcard/1.png", options) ;
```