[**图片缓存之内存缓存技术LruCache,软引用**](http://blog.chinaunix.net/uid-26930580-id-4138306.html)

每当碰到一些大图片的时候，我们如果不对图片进行处理就会报OOM异常，  
这个问题曾经让我觉得很烦恼，后来终于得到了解决，  
那么现在就让我和大家一起分享一下吧。  
这篇博文要讲的图片缓存机制，我接触到的有两钟，一种是软引用，另一种是内存缓存技术。  
先来看下两者的使用方式，再来作比较。  
除了加载图片时要用到缓存处理，还有一个比较重要的步骤要做，就是要先压缩图片。  
  
**1、压缩图片**  
至于要压缩到什么状态就要看自己当时的处境了，压缩图片的时候既要达到一个小的值，又不能让其模糊  
，更不能拉伸图片。

1. /\*\*
2. \* 加载内存卡图片
3. \*/
4. BitmapFactory.Options options = new BitmapFactory.Options();
5. options.inJustDecodeBounds = true; // 设置了此属性一定要记得将值设置为false
6. Bitmap bitmap = null;
7. bitmap = BitmapFactory.decodeFile(url, options);
8. int be = (int) ((options.outHeight > options.outWidth ? options.outHeight / 150
9. : options.outWidth / 200));
10. if (be <= 0) // 判断200是否超过原始图片高度
11. be = 1; // 如果超过，则不进行缩放
12. options.inSampleSize = be;
13. options.inPreferredConfig = Bitmap.Config.ARGB\_4444;
14. options.inPurgeable = true;
15. options.inInputShareable = true;
16. options.inJustDecodeBounds = false;
17. try {
18. bitmap = BitmapFactory.decodeFile(url, options);
19. } catch (OutOfMemoryError e) {
20. System.gc();
21. Log.e(TAG, "OutOfMemoryError");
22. }

**2、软引用：**  
只要有足够的内存，就一直保持对象，直到发现内存吃紧且没有Strong Ref时才回收对象。  
我们可以这样定义：map里面的键是用来放图片地址的，既可以是网络上的图片地址，也可以SDcard上的图片地址，  
map里面的值里面放的是持有软引用的Bitmap，当然如果你要放Drawable，那也是可以的。

1. private Map<String, SoftReference<Bitmap>> imageMap =

 new HashMap<String, SoftReference<Bitmap>>();

接下来就让我再介绍一下如何具体加载图片：  
步骤：（1）先通过URL查看缓存中是否有图片，如果有，则直接去缓存中取得。  
           如果没有，就开线程重新去网上下载。  
      （2）下载完了之后，就把图片放在缓存里面，方便下次可以直接从缓存中取得。

1. public Bitmap loadBitmap(final String imageUrl,final ImageCallBack imageCallBack) {
2. SoftReference<Bitmap> reference = imageMap.get(imageUrl);
3. if(reference != null) {
4. if(reference.get() != null) {
5. return reference.get();
6. }
7. }
8. final Handler handler = new Handler() {
9. public void handleMessage(final android.os.Message msg) {
10. //加入到缓存中
11. Bitmap bitmap = (Bitmap)msg.obj;
12. imageMap.put(imageUrl, new SoftReference<Bitmap>(bitmap));
13. if(imageCallBack != null) {
14. imageCallBack.getBitmap(bitmap);
15. }
16. }
17. };
18. new Thread(){
19. public void run() {
20. Message message = handler.obtainMessage();
21. message.obj = downloadBitmap(imageUrl);
22. handler.sendMessage(message);
23. }
24. }.start();
25. return null ;
26. }
27. // 从网上下载图片
28. private Bitmap downloadBitmap (String imageUrl) {
29. Bitmap bitmap = null;
30. try {
31. bitmap = BitmapFactory.decodeStream(new URL(imageUrl).openStream());
32. return bitmap ;
33. } catch (Exception e) {
34. e.printStackTrace();
35. return null;
36. }
37. }
38. public interface ImageCallBack{
39. void getBitmap(Bitmap bitmap);
40. }

**2、内存缓存技术**  
另外一种图片缓存的方式就是内存缓存技术。在Android中，有一个叫做LruCache类专门用来做图片缓存处理的。  
它有一个特点，当缓存的图片达到了预先设定的值的时候，那么近期使用次数最少的图片就会被回收掉。  
步骤：（1）要先设置缓存图片的内存大小，我这里设置为手机内存的1/8,手机内存的获取方式：int MAXMEMONRY = (int) (Runtime.getRuntime() .maxMemory() / 1024);  
   （2）LruCache里面的键值对分别是URL和对应的图片  
   （3）重写了一个叫做sizeOf的方法，返回的是图片数量。

1. private LruCache<String, Bitmap> mMemoryCache;
2. private LruCacheUtils() {
3. if (mMemoryCache == null)
4. mMemoryCache = new LruCache<String, Bitmap>(
5. MAXMEMONRY / 8) {
6. @Override
7. protected int sizeOf(String key, Bitmap bitmap) {
8. // 重写此方法来衡量每张图片的大小，默认返回图片数量。
9. return bitmap.getRowBytes() \* bitmap.getHeight() / 1024;
10. }
11. //移除图片
12. @Override
13. protected void entryRemoved(boolean evicted, String key,
14. Bitmap oldValue, Bitmap newValue) {
15. Log.v("tag", "hard cache is full , push to soft cache");
17. }
18. };
19. }

     （4）下面的方法分别是清空缓存、添加图片到缓存、从缓存中取得图片、从缓存中移除。移除和清除缓存是必须要做的事，因为图片缓存处理不当就会报内存溢出，所以一定要引起注意。

1. public void clearCache() {
2. if (mMemoryCache != null) {
3. if (mMemoryCache.size() > 0) {
4. Log.d("CacheUtils",
5. "mMemoryCache.size() " + mMemoryCache.size());
6. mMemoryCache.evictAll();
7. Log.d("CacheUtils", "mMemoryCache.size()" + mMemoryCache.size());
8. }
9. mMemoryCache = null;
10. }
11. }
12. public synchronized void addBitmapToMemoryCache(String key, Bitmap bitmap) {
13. if (mMemoryCache.get(key) == null) {
14. if (key != null && bitmap != null)
15. mMemoryCache.put(key, bitmap);
16. } else
17. Log.w(TAG, "the res is aready exits");
18. }
19. public synchronized Bitmap getBitmapFromMemCache(String key) {
20. Bitmap bm = mMemoryCache.get(key);
21. if (key != null) {
22. return bm;
23. }
24. return null;
25. }
26. /\*\*
27. \* 移除缓存
28. \*
29. \* @param key
30. \*/
31. public synchronized void removeImageCache(String key) {
32. if (key != null) {
33. if (mMemoryCache != null) {
34. Bitmap bm = mMemoryCache.remove(key);
35. if (bm != null)
36. bm.recycle();
37. }
38. }
39. }

注：Android用LruCache来取代原来强引用和软引用实现内存缓存，因为据说自2.3以后Android将更频繁的调用GC，导致软引用缓存的数据极易被释放。LruCache使用一个LinkedHashMap简单的实现内存的缓存，没有软引用，都是强引用。如果添加的数据大于设置的最大值，就删除最先缓存的数据来调整内存。他的主要原理在trimToSize方法中。需要了解两个主要的变量size和maxSizemaxSize是通过构造方法初始化的值，他表示这个缓存能缓存的最大值是多少。size在添加和移除缓存都被更新值，他通过safeSizeOf这个方法更新值。safeSizeOf默认返回1，但一般我们会根据maxSize重写这个方法，比如认为maxSize代表是KB的话，那么就以KB为单位返回该项所占的内存大小。

public void trimToSize(int maxSize)

{

while (true)

{

K key;V value;synchronized (this)

{

if (size < 0 || (map.isEmpty() && size != 0))

{

throw new IllegalStateException(getClass().getName() + ".sizeOf() is reporting inconsistent results!");

}

if (size <= maxSize)

{ break;

}

Map.Entry<K, V> toEvict = map.eldest();

if (toEvict == null)

{ break; }

key = toEvict.getKey();

value = toEvict.getValue();

map.remove(key);

size -= safeSizeOf(key, value); evictionCount++;}

entryRemoved(true, key, value, null);}}

除异常外首先会判断size是否超过maxSize,，如果超过了就取出最先插入的缓存，如果不为空就删掉（一般来说只要map不为空都不会返回null，因为他是个双休链表），并把size减去该项所占的大小。这个操作将一直循环下去，直到size比maxSize小或者缓存为空。

**4、两者的比较**  
说到这里，我觉得有必要来进行一下比较了。  
网上有很多人使用软引用加载图片的多 ，但是现在已经不再推荐使用这种方式了，  
（1）因为从 Android 2.3 (API Level 9)开始，垃圾回收器会更倾向于回收持有软引用或弱引用的对象，这让软引用和弱引用变得不再可靠。  
（2）另外，Android 3.0 (API Level 11)中，图片的数据会存储在本地的内存当中，因而无法用一种可预见的方式将其释放，这就有潜在的风险造成应用程序的内存溢出并崩溃，  
所以我这里用得是LruCache来缓存图片，当存储Image的大小大于LruCache设定的值，系统自动释放内存，  
这个类是3.1版本中提供的，如果你是在更早的Android版本中开发，则需要导入android-support-v4的jar包。