## Bitmap OOM

见图片OOM文档

## ListView优化

## UI Review（视图检查）

### 减少视图层级

减少视图层级可以有效的减少内存消耗，因为视图是一个树形结构，每次刷新和渲染都会遍历一次。

### ViewStub标签

此标签可以使UI在特殊情况下，直观效果类似于设置View的不可见性，但是其更大的意义在于被这个标签所包裹的Views在默认状态下不会占用任何内存空间。

### include标签

可以通过这个标签直接加载外部的xml到当前结构中，是复用UI资源的常用标签。

### merge标签

它在优化UI结构时起到很重要的作用。目的是通过删减多余或者额外的层级，从而优化整个Android Layout的结构。

## 使用ThreadPool而不是每次new Thread

## 减少不必要的全局变量

尽量避免static成员变量引用资源耗费过多的实例,比如Context。

因为Context的引用超过它本身的生命周期，会导致Context泄漏。所以尽量使用Application这种Context类型。

你可以通过调用Context.getApplicationContext()或 Activity.getApplication()轻松得到Application对象。

## Cursor（游标）回收

Cursor是Android查询数据后得到的一个管理数据集合的类，在使用结束以后。应该保证Cursor占用的内存被及时的释放掉，而不是等待GC来处理。并且Android明显是倾向于编程者手动的将Cursor close掉，因为在源代码中我们发现，如果等到垃圾回收器来回收时，会给用户以错误提示。

## Receiver（接收器）回收

调用registerReceiver()后未调用unregisterReceiver().

当我们Activity中使用了registerReceiver()方法注册了BroadcastReceiver，一定要在Activity的生命周期内调用unregisterReceiver()方法取消注册   
也就是说registerReceiver()和unregisterReceiver()方法一定要成对出现，通常我们可以重写Activity的onDestory()方法，在onDestory里进行unregisterReceiver操作

## Stream/File（流/文件）回收

主要针对各种流，文件资源等等如：

InputStream/OutputStream，SQLiteOpenHelper，SQLiteDatabase，Cursor，文件，I/O，Bitmap图片等操作等都应该记得显示关闭。

## 避免创建不必要的对象

最常见的例子就是当你要频繁操作一个字符串时，使用StringBuffer代替String。

还比如：使用int数组而不是Integer数组。

避免创建短命的临时对象，减少对象的创建就能减少垃圾收集，进而减少对用户体验的影响。

## 避免内部Getters/Setters

在Android中，虚方法调用的代价比直接字段访问高昂许多。通常根据面向对象语言的实践，在公共接口中使用Getters和Setters是有道理的，但在一个字段经常被访问的类中宜采用直接访问。

## for循环

访问成员变量比访问本地变量慢得多，如下面一段代码：

**for**(**int** i =0; i < **this**.mCount; i++)  {}

永远不要在for的第二个条件中调用任何方法，如下面一段代码：

**for**(**int** i =0; i < **this**.getCount(); i++) {}

对上面两个例子最好改为：

**int** count = **this**.mCount; / **int** count = **this**.getCount();

**for**(**int** i =0; i < count; i++)  {}