**布局方面**

布局层级不要太复杂,尽量减少无用布局和布局嵌套,可以使用布局工具Hierarchy Viewer来检查和优化布局;

将重复的复杂布局或者view层级较多的布局抽取出来,使用include嵌套,是代码结构清晰

使用viewstub 动态加载不常用的布局,加快布局渲染呈现

根布局尽量少使用RelactivityLayout,因为RelactivityLayout每次需要测量2次

(（1）RelativeLayout慢于LinearLayout是因为它会让子View调用2次measure过程，而后者只需一次，但是有weight属性存在时，后者同样会进行两次measure。

（2）RelativeLayout的子View如果高度和RelativeLayout不同，会引发效率问题，可以使用padding代替margin以优化此问题。

（3）在不响应层级深度的情况下，使用Linearlayout而不是RelativeLayout。)

**代码逻辑方面**

，什么是内存泄漏，android在什么情况下容易产生内存泄漏

说到内存泄漏就不得不提内存溢出。

内存溢出 out of memory，是指程序在申请内存时，没有足够的内存空间供其使用，出现out of memory；比如申请了一个integer,但给它存了long才能存下的数，那就是内存溢出。

内存泄露 memory leak，是指程序在申请内存后，无法释放已申请的内存空间，一次内存泄露危害可以忽略，但内存泄露堆积后果很严重。内存溢出导致了内存泄漏。

**预防内存溢出和内存泄露**

1. 资源释放问题  
程序代码的问题，长期保持某些资源，如Context、Cursor、IO流的引用，资源得不到释放  
造成内存泄露。  
2. 对象内存过大问题  
保存了多个耗用内存过大的对象（如Bitmap、XML文件），造成内存超出限制, Bitmap对象不再使用时调用recycle()释放内存。  
3. static关键字的使用问题  
static是[**Java**](http://lib.csdn.net/base/javaee)中的一个关键字，当用它来修饰成员变量时，那么该变量就属于该类，而不是  
该类的实例。所以用static修饰的变量，它的生命周期是很长的，如果用它来引用一些资源耗费  
过多的实例（Context的情况最多），这时就要谨慎对待了。  
public class ClassName {  
private static Context mContext;  
//省略  
}  
以上的代码是很危险的，如果将 Activity 赋值到 mContext 的话。那么即使该 Activity 已经  
onDestroy，但是由于仍有对象保存它的引用，因此该Activity依然不会被释放。

4. 线程导致内存溢出  
线程产生内存泄露的主要原因在于线程生命周期的不可控。

5，[**数据库**](http://lib.csdn.net/base/mysql)游标忘记回收等

那么针对上面的问题我们怎么避免呢

6 图片过大导致OOM 使用图片缓存机制,介绍一下图片缓存……

3、构造Adapter时，没有使用缓存的 convertView  
在使用ListView的时候通常会使用Adapter，那么我们应该尽可能的使用ConvertView。  
为什么要使用convertView?  
当convertView为空时，用setTag()方法为每个View绑定一个存放控件的ViewHolder对象。当 convertView 不为空，重复利用已经创建的 view 的时候，使用 getTag()方法获取绑定的ViewHolder对象，这样就避免了findViewById对控件的层层查询，而是快速定位到控件。

**适配方面**