性能优化：

参考http://blog.csdn.net/gs12software/article/details/51234454

ListView的复用

合理使用缓存

界面UI的层次

使用性能分析工具

数据结构、算法

减少new 的对象

常量字段要声明为“static final”，因为这样常量会被存放在dex文件的静态字段初始化器中被直接访问，否则在运行时需要通过编译时自动生成的一些函数来初始化。此规则只对基本类型和String类型有效。

**避免内部的Getters/Setters**   
面向对象设计中，字段访问使用Getters/Setters通常是一个好的原则，但是在Android开发中限于硬件条件，除非字段需要被公开访问，否则如果只是有限范围内的内部访问（例如包内访问）则不建议使用Getters/Setters。在开启JIT时，直接访问的速度比间接访问要快7倍。

**采用**<merge>**优化布局层数。 采用**<include＞**来共享布局**

广播BroadCast动态注册时，记得要在调用者生命周期结束时unregisterReceiver,防止内存泄漏。即及时释放一些对象，避免占用内存。

**Bitmap的使用**

**尽量不要使用整张的大图作为资源文件，尽量使用9path图片**

**1.UI卡顿常见原因：**

在UI线程中做了耗时操作，导致UI线程卡顿；

布局Layout过于复杂，无法在16ms内完成渲染；

同一时间动画执行的次数过多，导致CPU或GPU负载过重；

View过度绘制，导致某些像素在同一帧时间内被绘制多次，从而使CPU或GPU负载过重；

View频繁的触发measure、layout，导致measure、layout累计耗时过多及整个View频繁的重新渲染；

内存频繁触发GC过多（同一帧中频繁创建内存），导致暂时阻塞渲染操作；

冗余资源及逻辑等导致加载和执行缓慢；

使用Traceview进行分析优化

**使用Systrace进行分析优化**

Android内存泄露检测

众所周知，在Java中有些对象的生命周期是有限的，当它们完成了特定的逻辑后将会被垃圾回收；但是，如果在对象的生命周期本来该被垃圾回收时这个对象还被别的对象所持有引用，那就会导致内存泄漏；这样的后果就是随着我们的应用被长时间使用，他所占用的内存越来越大。

使用LeakCanary内存泄漏检测工具