**一、Android性能优化**  
1.http用gzip压缩，设置连接超时时间和响应超时时间  
http请求按照业务需求，分为是否可以缓存和不可缓存，那么在无网络的环境中，仍然通过缓存的httpresponse浏览部分数据，实现离线阅读。  
2.listview 性能优化  
1).复用convertView  
在getItemView中，判断convertView是否为空，如果不为空，可复用。如果couvertview中的view需要添加listerner，代码一定要在if(convertView==null){}之外。  
2).异步加载图片  
item中如果包含有webimage，那么最好异步加载  
3).快速滑动时不显示图片  
当快速滑动列表时（SCROLL\_STATE\_FLING），item中的图片或获取需要消耗资源的view，可以不显示出来；而处于其他两种状态（SCROLL\_STATE\_IDLE 和SCROLL\_STATE\_TOUCH\_SCROLL），则将那些view显示出来  
3.使用线程池，分为核心线程池和普通线程池，下载图片等耗时任务放置在普通线程池，避免耗时任务阻塞线程池后，导致所有异步任务都必须等待  
4.异步任务，分为核心任务和普通任务，只有核心任务中出现的系统级错误才会报错，异步任务的ui操作需要判断原activity是否处于激活状态  
5.尽量避免static成员变量引用资源耗费过多的实例,比如Context  
6.使用WeakReference代替强引用，弱引用可以让您保持对对象的引用，同时允许GC在必要时释放对象，回收内存。对于那些创建便宜但耗费大量内存的对象，即希望保持该对象，又要在应用程序需要时使用，同时希望GC必要时回收时，可以考虑使用弱引用。  
7.超级大胖子Bitmap  
及时的销毁(Activity的onDestroy时，将bitmap回收)  
设置一定的采样率  
巧妙的运用软引用  
drawable对应resid的资源，bitmap对应其他资源

8.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  
9.线程也是造成内存泄露的一个重要的源头。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  
10.如果ImageView的图片是来自网络，进行异步加载  
11.应用开发中自定义View的时候，交互部分，千万不要写成线程不断刷新界面显示，而是根据TouchListener事件主动触发界面的更新

**二、Android UI优化**  
1.layout组件化，尽量使用merge及include复用  
2.使用styles，复用样式定义  
3.软键盘的弹出控制，不要让其覆盖输入框  
4.数字、字母和汉字混排占位问题：将数字和字母全角化。由于现在大多数情况下我们的输入都是半角，所以 字母和数字的占位无法确定，但是一旦全角化之后，数字、字母的占位就和一个汉字的占位相同了，这样就可以避免由于占位导致的排版问题。  
5.英文文档排版：textview自动换行时要保持单词的完整性，解决方案是计算字符串长度，然后手动设定每一行显示多少个字母并加上‘n‘  
6.复杂布局使用RelativeLayout  
7.自适应屏幕，使用dp替代pix  
8.使用android:layout\_weight或者TableLayout制作等分布局  
9.使用animation-list制作动画效果

**三：Android注释**

严格来说这个应该属于命名规范的范畴。注释一方面是帮助自己记忆 ，另一方面是团队协作中的一个规范，好的注释配合好的命名规范，可以省去很多沟通上的成本。 注释至少要有如下几方面的内容：

1. 该接口（或类）的作用。注意写的应该是作用，而不是你做了什么；
2. 参数列表的各个参数说明；
3. 返回值的说明；
4. 如果有异常抛出，对抛出异常的说明；
5. 如果注释是在类上的，总得留个作者吧，免得以后出了问题都找不到原作者。
6. 其他你认为有必要解释。

**四：内存管理**

Android 虽然延续了 Java 的垃圾回收机制，但是并不意味 Android 应用程序就不会出现内存问题。在 Android 中引起内存开销过大的往往是 BitMap 对象。BitMap.java实际上是 Skia 引擎中对图片处理部分的 Java 层代码而已（真正工作的是 C++层代码，通过 JNI 封装，最后提供 Java 层的接口），那么你创建 BitMap 对象实际上是创建了两部分内存，一部分是 Java 层的，就是 BitMap对象，Java 的垃圾回收会在合适的时机回收这一部分内存。另一部分是 C++层面的，也就是通过 JNI 调用 C++层的代码分配的那一部分内存。Java 的垃圾回收是不会回收这一部分内存的，所以如果不手动释放的话就容易引起内存问题。

**五：千万不要阻塞用户主线程**

用户主线程就是 UI 线程，主要负责 UI 的绘制（除 SurfaceView 外，其他 View 对象都是需要在 UI 线程中进程操作的）。为了保证 App 的交互尽可能的流程，请不要在 UI 线程中进行耗时操作（文件读写、Http 请求（4.0之前可以在主线程中发起）等）。否则会引起两种可能的问题：第一是造成用户交互极度不流畅，第二容易触发 ANR 的超时机制（UI 线程5秒，广播10秒）。

**六：严格把控生命周期（Activity、Service、ContentProvider 等）**

在每一个生命周期事件中，明确应该做什么不应该做什么是很有必要的，不然也会容易造成各种莫名其妙的问题（比如 onCreate 中使用了 onResume 中才初始化的对象）。

**七:使用 XML 文件进行 UI 布局时，应该尽量减少 Layout 的嵌套层级。Layout 的过度嵌套会造成渲染时资源开销过大的问题。**

**八：资源文件的使用**

资源文件包括图片、字符串、尺寸、颜色等等。在使用尺寸资源的时候应该尽量使用像素无关的单位，比如 dp 和 sp。而字符串资源（比如 Button 上显示的名称）也应该尽可能的抽离出来，使用 res/value 下的xml 文件进行维护。一方面方便日后管理，另一方面方便国际化。

**九：多线程以及线程池的使用**

前面说过应该尽量避免在主线程中执行耗时操作，那么多线程就变得很有必要。对于 Java 来说，线程的创建与销毁是非常占用资源的，线程的滥用（随手 new Thread 等）会造成 App 整体性能的下降。Java 提供了Executors的线程池方案，而 Android 自身也提供了AsyncTask 这样的异步任务方案（实际上也是线程池）。

**十：final 和 static 关键字的合理使用**

很多人觉得这是很基础的东西，但是 final 和 static 关键字的合理使用能够有效提升代码的执行效率，而不合理使用则后患无穷。

**十一： Android 设备的内存资源是极度珍贵的**

合理的使用、回收内存也是一种好的编程习惯。Java 对象的引用类型会影响到垃圾回收对象的时机。Java 有强引用、 软引用、 弱引用、虚引用，以及 Android 增加的 Lru 内存管理。建议了解一下这四种引用类型的特点以及 Lru 内存管理的具体实现。

**十二：接口和抽象类**

这是老生常谈的话题了，但却是永恒的话题。接口和抽象类的合理使用，可以增加代码的可维护性和扩展性。接口和抽象类也是各种设计模式的基石。

**十三：软件设计的六大设计原则**

1. 针对接口编程，不针对实现编程
2. 单一职责原则
3. 开放封闭原则
4. 里氏代换原则
5. 迪米特法则
6. 合成聚合复用原则

**十四：统一项目的编码格式（推荐使用 UTF-8）**

如果多人协作，这种举措显得尤为重要。由此引申出来的另外一个规范就是，规范统一命名风格，即团队中使用相同的命名风格

**十五：使用友盟分享 SDK**

需要执行分享的 Activity 请不要为该 Activity 设置android:process属性。比如你的 App 运行在 com.fuzhong.test 进程，需要产生分享动作的Activity 设置 android:proces=":com.fuzhong.hello" ，那么新浪微博就会出现你设置的分享内容没有显示的问题。该 Bug 已经提交给友盟的技术人员，但是 N 久没有得到修复。  
 **十六：上线之前一定要使用正式签名打包**

我废话不多说，发布生产环境之前必须打正式签名包

**十七：Activity 中尽可能少的创建 Handler 对象，创建一个主线程 Handler，一个后台 HandlerThread 就可以了。  
  
十八：onCreate(Bundle savedInstanceState) 切记将super.onCreate(savedInstanceState);放在一切业务的前面。**  
  
  
**十九：要考虑到Activity和进程被杀掉的情况**

通常情况退出Activity外，还有Activity因其他原因被杀的情况，比如系统内存过低，系统配置变更，有异常等等，要考虑和测试这种情况，特别是Activity处理重要的数据时，做好的数据的保存。  
  
  
**二十：不要用四大组件去实现接口**

一是组件的对象都比较大，实现接口比较浪费，而且让代码更不易读和理解； 另外更重要的是导致多方引用，可能会引发内存泄露。  
  
  
**二十一：用getApplication()来取Context当参数**

对于需要使用Context对象作为参数的函数，要使用getApplication()获取Context对象当参数，而不要使用this，除非你需要特定的组件实例！getApplication()返回的Context是属于Application的，它会在整个应用的生命周期内存在，远大于某个组件的生命周期，所以即使某个引用长期持有Context对象也不会引发内存泄露。  
  
  
**二十二：要考虑多分辨率**

至少为hdpi, mdpi, ldpi准备图片和布局。元素的单位也尽可能的使用dip而不要用px。如果要求对包大小有限制，那么我们呢只需要做一个大分辨率的图片，比如xhdpi：至少960\*720