## 布局优化

重用布局文件，关键词include标签，merge标签。

ViewStub，在需要的时候才去加载布局。没有大小，没有绘制功能，也不参与布局，资源消耗很低，不影响性能。

## 内存优化

获取手机中分配给每个应用内存大小的方法是：

ActivityManager manager = (ActivityManager)getSystemService(Context.ACTIVITY\_SERVICE);

int heapSize = manager.getMemoryClass();

该方法返回的就是系统为每个应用分配的内存大小。

GC全称是：Garbage Collection。

节制的使用service，如果应用中需要Service来执行后台任务的话，一定注意只有放任务正在执行的时候才应该让service运行起来，当任务执行完毕后停止service的时候，要小心service停止失败导致的内存泄漏。当然也更加推荐使用IntentService，这种service最大的优点是后台任务执行结束自动停止。

当界面不可见的时候释放内存

避免在Bitmap上浪费内存

使用优化过的数据集合

尽量避免使用依赖注入

## 性能编码优化

**避免创建不必要的对象**：如果需要拼接一个长字符串，可以优先考虑使用StringBuffer或者StringBuilder来进行拼接，而不是用加号连接符，因为使用加号连接符会创建过多的对象，拼接的字符串越长，加号连接符的性能越低。

**静态优于抽象**：如果是调用一个对象中的某一个方法来完成一项通用的功能，可以将这个方法设置成静态方法。这个设置可以让程序的运行速度提高百分之15到百分之二十。

**对常量使用static final 修饰符**：

**使用增强型for循环语法**：

**多使用系统封装好的API:**系统封装好的api性能会更好，执行效率会更高，假如我们要实现一个数据拷贝的功能，使用循环的方式来对数组中的每一个元素进行遍历赋值可是可以的，但是我们也可以直接使用系统提供的System.arraycopy方法，这个方法能将执行效率提高好几倍。

**避免在内部调用getters和setters方法**：

## APP性能优化二

APP使用起来流畅，不卡顿

APP使用起来省电，还省流量

APP使用要稳定，不闪退等bug的出现

APP的安装包要尽量的变小

造成不流畅卡顿的原因有：UI的绘制层级深，页面复杂，刷新不合理；数据处理上，主要原因是数据量太大；UI的过度绘制，绘制的页面有好几层View，底层的View都隐藏了，这种情况就会造成过度绘制。

卡顿的优化方案有哪些：

不要在主线程中进行网络访问，大文件的io操作；

绘制UI时，尽量减少绘制UI的层次，减少不必要的View嵌套，可以使用一些工具来检测，比如：Hierarchy Viewer；

提高显示速度,使用ViewStub：当加载的时候才会占用。不加载的时候就是隐藏的，仅仅占用位置；

在view层级相同的情况下，尽量使用 LinerLayout而不是RelativeLayout；因为RelativeLayout在测量的时候会测量二次，而LinerLayout测量一次，可以看下它们的源码；

Listview的优化相关内容；

尽量避免过度绘制overDraw。

Android中电量分析工具：在android5.0以后，Google官方api提供了一个电量分析工具，叫做：Battery Historian。可以直观方便的看到电量的消耗情况。

一个对图片能进行无损压缩的插件：TinyPNG.。

APP安装包大小的优化：res资源优化、代码的优化、lib资源优化、代码混淆

只使用一套图片，使用高分辨率的图片；

移除无用的依赖库，libs等无用东西；

Lib优化：动态下载资源，一些模块化的插件动态加载

内存泄漏出现的场景：

 单例中引用的上下文Context，引用了Activity中的Context, 这样会造成内存泄漏，要引用Application中的Context;

 资源性对象未关闭。比如Cursor、File文件等，往往都用了一些缓冲，在不使用时，应该及时关闭它们。

 注册对象未注销。比如事件注册后未注销，会导致观察者列表中维持着对象的引用。

类的静态变量持有大数据对象。

 非静态内部类的静态实例。

 Handler临时性内存泄漏。如果Handler是非静态的，容易导致 Activity 或 Service 不会被回收。

 容器中的对象没清理造成的内存泄漏。

 WebView。WebView 存在着内存泄漏的问题，在应用中只要使用一次 WebView，内存就不会被释放掉。

内存的优化方案：

 对象引用。强引用、软引用、弱引用、虚引用四种引用类型，根据业务需求合理使用不同，选择不同的引用类型。

 减少不必要的内存开销。注意自动装箱，增加内存复用，比如有效利用系统自带的资源、视图复用、对象池、Bitmap对象的复用。

 使用最优的数据类型。比如针对数据类容器结构，可以使用ArrayMap数据结构，避免使用枚举类型，使用缓存Lrucache等等。

 图片内存优化。可以设置位图规格，根据采样因子做压缩，用一些图片缓存方式对图片进行管理等等。