# 一、何为性能问题

## 1 响应时间

从用户操作开始到系统给用户以正确反馈的时间；ANR问题是此类性能问题的主要体现

## 2 FPS(Frame Per Second)

另一种是系统响应后画面更新的频率。60fps是人类感知视觉更新的极限，所以屏幕每帧的更新时间<16ms时，视觉没有停顿；4.0之后Android系统每隔16ms发出VSYNC信号，对CPU、GPU、显示屏同步触发对UI进行渲染。如果每次渲染时间都在16ms之内，则视觉流畅。

# 二、性能调优方式

从性能问题的分析上，可以得出性能调优过程包括缩短响应时间和提高FPS。优化的方法可以归纳为三种：

## (1) 降低执行时间 这部分包括：

a. GPU渲染优化

b. 布局优化

c. 内存优化（详见内存优化部分）

d. 缓存(包括对象缓存、IO缓存、网络缓存)

e. 数据存储类型优化

f. 算法优化

g. 逻辑优化

## (2) 同步改异步

利用多线程提高FPS

## (3) 提前或延迟操作

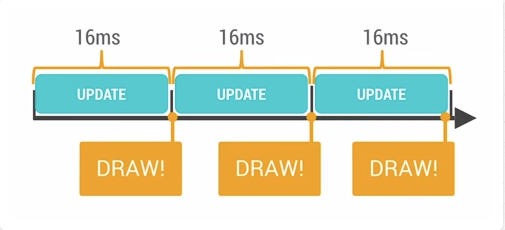
错开时间段提高FPS

以下详细说

## (1) 降低执行时间

### a. GPU渲染优化

Android系统每隔16ms发出VSYNC信号，无论是CPU、GPU、还是屏幕画面更新，更新在16ms之上就会出现丢帧的情况。如下图



先说GPU：

两个工具：

1）开发者选项 --》Profile GPU Rendering

我们可以在画面上看到GPU绘制图形信息，分别关于StatusBar，NavBar，激活的程序Activity区域的GPU Rending信息。界面上会滚动显示垂直的柱状图来表示每帧画面所需要渲染的时间，中间有条绿线，代表16ms阈值。每一条柱状线都包含三部分，蓝色：测量绘制Display List的时间，红色：OpenGL渲染Display List所需要的时间，黄色：CPU等待GPU处理的时间。**超过16ms阈值线，则说明存在卡顿，是判断性能的直观定性的标准。**

****

2）开发者选项--》Show GPU Overdraw

Overdraw(过度绘制)指屏幕上的某个像素在同一帧的时间内被绘制了多次。多个层级嵌套的页面容易出现。蓝色，淡绿，淡红，深红代表了4种不同程度的Overdraw情况，我们的目标就是尽量减少红色Overdraw，看到更多的蓝色区域，消灭深红的区域。



**实际优化经验：**

1 BaseActivity getWindow().getDecorView().setBackground(null); 每个Activity的mDecorView默认有背景图，而mDecorView一般被layout所覆盖，背景图无用，可以置为null；如果需要为Activity设置背景，可以考虑Theme的方式；transparent的背景色不会增加重绘；

2 View的setFadingEdgeEnable属性会将边缘透明度过度绘制，将重绘次数提高到深红色，性能敏感界面不要使用此属性；

3 layout层级优化，可以降低mDisplayList层级

4 View的DrawingCache（SOFTWARE\_CACHE）软缓存，GPU还将绘制缓存bitmap，增加绘制次数

### b. Layout布局优化

在屏幕更新的过程中，CPU处理的上层应用可以涉及到的，主要包含onDraw onMeasure onLayout。通常来说，Android需要把XML布局文件转换成GPU能够识别并绘制的对象。这个操作是在DisplayList的帮助下完成的。DisplayList通过每一View层的onMeasure onLayout，持有要交给GPU绘制到屏幕上的数据信息。布局应遵循宽而浅，避免窄而深的层级，在三个关键函数中避免任何耗时的操作。层级少的DisplayList也为GPU渲染节省了时间。

**尤其要注意避免不必要的RequestLayout,Invalidate**（注意哪些方法本身就含有invalidate方法，如notifyDataSetChanged）**也是较少layout时间的重要方法**

**具体方法及实际优化经验：**

1 layaout中默认不显示的部分，可以viewstub；例如加载过程，出错信息等。注意inflate后view变量的保存

2 include不多说

3 merge Activity的布局顶节点是FrameLayout；某布局作为子布局被其他布局include时，使用merge当作该布局的顶节点，两种情况可以节省一层layout

4 多个linearLayout可以RelativeLayout代替，复杂布局可以先考虑GridLayout，不要使用多级嵌套

5 一个ImageView和TextView组合的LinearLayout可以用compound drawable代替

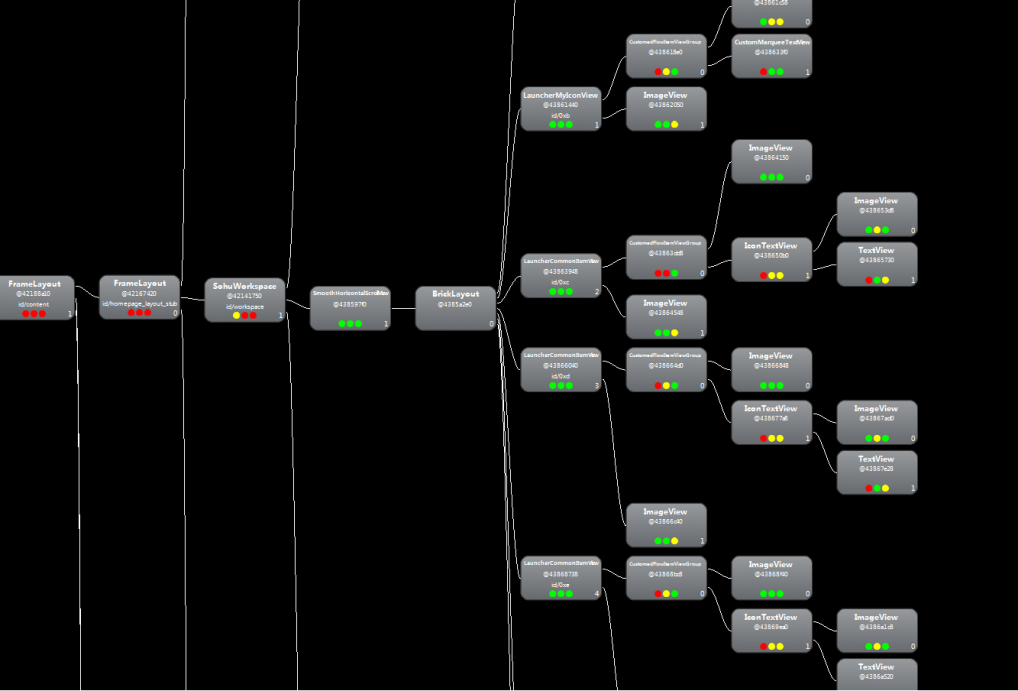
6 如果一个layout只有一个子控件，没有兄弟控件，并且不是一个ScrollView，没有背景，则可以移除之

7 listview之间间隙，GridView行、列间隙等，使用控件的本身属性 例如 android:dividerHeight = 18dp 而不要使用layout嵌套的方式实现

8 减少不必要的inflate，已经inflate的view要保存，比如AdapterView中的viewHodler

一个工具：

**HierarchyViewer，查看布局，点击ProfileNode后可以看每一层onDraw onMeasure onLayout所用时间；sdk的lint工具也可以检查layout布局**

****

### c. 内存优化（以下详细讲）

### d. 缓存方案

线程池、图片双（三）级缓存、消息缓存（handler.sendMessage(handler.obtainMessage(0, object));）、网络缓存、文件I\O缓存（BufferedInputStream、BufferReader替代InputStream等）

### e. 数据存储类型优化

**1. 数据类型选择**字符串拼接用StringBuilder代替String，在非并发情况下用StringBuilder代替StringBuffer。  
使用SoftReference、WeakReference相对正常的强应用来说更有利于系统垃圾回收  
final类型存储在常量区中读取效率更高

2.**数据结构选择**

Android也提供了一些性能更优的数据类型，如SparseArray、SparseBooleanArray、SparseIntArray、Pair。Sparse系列的数据结构是为key为int情况的特殊处理，采用二分查找及简单的数组存储，加上不需要泛型转换的开销，相对Map来说性能更优

### f.  算法优化

### g. 逻辑优化

## (2) 同步改异步

对于耗时操作往往需要放入新的线程中异步处理，（耗时操作包含数据库操作、文件操作、网络请求、复杂算法），这种情况要注意ANR（Event事件ANR和广播ANR）

如何避免ANR：以下来自developer.android.com

1 在Activity的关键生命周期函数onCreate onResume中，执行尽量少的事情，也可以提升页面启动速度

2 耗时操作可以在asyncTask中做，或者new HandlerThread 在新的Thread中做，注意设置Process.setThreadPriority()并传入THREAD\_PRIORITY\_BACKGROUND参数，否则新thread和UI线程等级同一样可能卡。

3 BroadcastReceiver只有10s的onReceive处理时间，可以启动一个IntentService来处理耗时操作

4 在加载过程中，增加loading提示

HandlerThread和AsyncTask的使用场景：

AsyncTask是android的轻量级异步类，简单易用，但在多个异步操作并和UI线程通信时很复杂，HandlerThread正好相反，多个后台任务简单清晰

## (3) 提前或延迟操作

### a.延时操作

1 在Activity Fragment Service BroadcastReceiver的Key生命周期中，耗时操作放入新线程外，动画等UI操作可以延时，handler.postDelayed，handler.postAtTime，handler.sendMessageDelayed，View.postDelayed；*在BaseFragment中将loadData操作放入onStart后进行*

2 页面的翻滚，大幅度动画场景下，将耗时UI操作，其他动画放入其后；特别是页面滑动中，要把海报填充、放大动画、甚至inflateview等操作滞后。

影视日历的滑动优化，以及max版首页优化，SmoothListView滑动优化都是监听滚动事件，在滚动之后onPageStateChange方法后实现的其他UI操作。

### b.提前操作

1、将一些操作放入loading页面或者欢迎页面做，并保存起来，来换取进入页面的后的性能。

2、将一些不经常变动的数据和信息保存起来，启动时先加载已有数据，启动之后访问网络更新数据也可以提高第一次加载的速度。

3、对于一些特定场景，比如应用刚启动，动画过程中等等，都可以利用TraceView来查看性能的瓶颈点，再根据瓶颈点来有针对性的解决。

参考文献：

Trinea博客性能篇：http://www.trinea.cn/android/performance/

TraceView的使用：<http://www.trinea.cn/android/android-performance-demo/>

《Android性能优化典范》中文翻译：<http://hukai.me/android-performance-patterns/>

官方开发文档：http://developer.android.com/

参考文档：

同级svn目录下：

《狐派launcher性能优化小结》

《狐派launcher性能分析方案（草案）》

# 一、内存泄露可以引发很多的问题：

1.程序卡顿，响应速度慢（内存占用高时JVM虚拟机会频繁触发GC）

2.莫名消失（当你的程序所占内存越大，它在后台的时候就越可能被干掉。反之内存占用越小，在后台存在的时间就越长）

3.直接崩溃（OutOfMemoryError）

# 二、5R：

通过如下的5R方法来对ANDROID内存进行优化：

**1.Reckon（计算）**

首先需要知道你的app所消耗内存的情况，知己知彼才能百战不殆

**2.Reduce（减少）**

消耗更少的资源

**3.Reuse（重用）**

当第一次使用完以后，尽量给其他的使用

**4.Recycle（回收）**

返回资源

**5.Review（检查）**

回顾检查你的程序，看看设计或代码有什么不合理的地方。

## Reckon(计算、监控内存)

从系统内存（system ram）和堆内存(heap)两个方面介绍一些查看和计算内存使用情况的方法：

ram和heap的区别和关系如下：包含关系

RAM（random access memory）随机存取存储器。说白了就是内存。

一般Java在内存分配时会涉及到以下区域：

**寄存器（Registers）：**速度最快的存储场所，因为寄存器位于处理器内部**，**我们在程序中无法控制

**栈（Stack）：**存放基本类型的数据和对象的引用，但对象本身不存放在栈中，而是存放在堆中

**堆（Heap）：**堆内存用来存放由new创建的对象和数组。在堆中分配的内存，由Java虚拟机的自动垃圾回收器（GC）来管理。（**堆内存）**

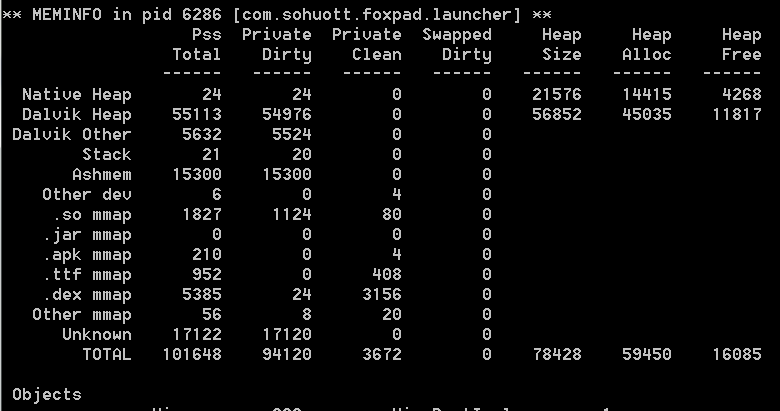
**静态域（static field）：**  静态存储区域就是指在固定的位置存放应用程序运行时一直存在的数据，Java在内存中专门划分了一个静态存储区域来管理一些特殊的数据变量如静态的数据变量

**常量池（constant pool）：**虚拟机必须为每个被装载的类型维护一个常量池。常量池就是该类型所用到常量的一个有序集和，包括直接常量（string,integer和floating point常量）和对其他类型，字段和方法的符号引用。

**非RAM存储：**硬盘等永久存储空间

**System Ram(系统内存)：**

观察和计算系统内存使用情况，可以使用Android提供给我们的两个工具**procstats**，**meminfo**。前者侧重于后台的内存使用，后者是运行时的内存使用。主要说meminfo，通过命令adb shell dumpsys meminfo



各个类型是什么意思，可以参考文章<http://blog.csdn.net/a396901990/article/details/37914465>

依据：PSS实际物理内存占用

存疑：meminfo 中dalvik和native的堆内存之和与heap是否有关系

**Heap（堆内存）：**

MAT工具导出的内存大小，以及DDMS中呈现的heap大小

我们在监控内存时，监控meminfo和heap内存都是可以的，二者基本保持线性关系

androidStudio对于内存有memoryMonitor的监控，可以更加连续和直观监控内存情况

谈MAT的用法 主要分为定向分析和初期判断

<http://www.ibm.com/developerworks/cn/opensource/os-cn-ecl-ma/index.html>

## Reduce(减少内存使用，减少内存泄漏几率) Reuse (重用机制)

### Bitmap：

**bitmap是内存大户，降低其内存占用是减少内存使用的主要方法**

1 bitmap替代品：color gradient 外发光 等android特效代替资源文件，几乎没有内存占用

2 根据容器的大小裁剪压缩bitmap大小；对分辨率不敏感可以设置BitmapFactory.Options；

也可以改变bitmap通道值ARGB\_8888

3 可以用9patch情况，不用整张图

4 使用第三方库加载bitmap，第三方库一般采用二级或三级缓存，可以对不再使用的bitmap及时释放，重用已有的内存；如果需要自己处理，不要view.setBackground(),占用很大

5 第三方库包含UniversalImageLoader，picasso square**，**Fresco facebook三级缓存等

6 在Adapter中使用convertView和viewHolder来实现

### 降低内存泄露最佳实践tips：

1. **Context对象泄漏**

1）、如果一个类持有Context对象的强引用，就需要检查其生存周期是否比Context对象更长。否则就可能发生Context泄漏。例如Activity实例注册系统listener，onDestroy要解注册；Activity实例传入Application，传入Service等（组件之间的关系要解耦）

2）、View持有其创建所在Context对象的引用，如果将View对象传递给其它生存周期比View所在Context更长的强引用，就可能会引起内存泄漏。

例如View#setTag(int, Object)的内存泄漏

3）、把Context对象赋给static变量。

1. **Handler对象泄漏**

发送到Handler的Message实际上是加入到了主线程的消息队列等待处理，每一个Message持有其目标Handler的强引用，Runnable也是如此。当使用内部类（包括匿名类）来创建Handler的时候，Handler对象会隐式地持有一个外部类对象（通常是一个Activity）的引用。而如果Message或Runnable在Activity关闭之后才发出，那么会有泄露。解决方法是static内部类，采用weakreference联系activity

1. **Drawable.Callback引起的内存泄漏**

Drawable对象持有Drawable.callback的引用。当把一个Drawable对象设置到一个View时，      Drawable对象会持有该View的引用作为Drawable.Callback。解决方法：尽量不要在static成员中保存Drawable对象，或者在需要时调用Drawable#setCallback(null)

1. **线程问题**

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。

public class MyActivity extends Activity {   
@Override   
public void onCreate(Bundle savedInstanceState) {   
    super.onCreate(savedInstanceState);   
    setContentView(R.layout.main);   
    new MyThread().start();   
}   
  
private class MyThread extends Thread{   
@Override   
    public void run() {   
    super.run();   
//do somthing   
}   
}   
}

由于MyThread是内部类，所以会保留一份Activity实例，当Thread任务没有执行完，activity被销毁或者横竖屏转换，则会有泄露。对于AsyncTask则情况更为严重，由于内部采用ThreadPoolExecutor外部无法控制Thread周期则问题更大。解决办法，static内部类以及软引用

1. **Cursor问题**

Cursor是android数据库查询结果管理数据集合的类，使用中如果没有释放，编译会提示，但注意CursorAdapter的情况，要手动在onDestroy中释放;InputStream/OutputStream，SQLiteOpenHelper，SQLiteDatabase，Cursor，文件，I/O，Bitmap图片等操作等都应该记得显式关闭。和之前介绍的Cursor道理类似。

### 小结：

1. 保存对象前要三思（static）

I. 对象本身有无隐含的引用

II. 保存后何时能够回收

2. 要了解常见的隐含引用

I. anonymous class outer class

II. View to context

3. 创建大对象时，要检查它的生命周期

## Recycle & Review

recycle的问题已经在之前说过，代码的行文中要注意资源的回收，详见防止泄露小tips

**Code Review（代码检查）：**以上所说的易泄漏点

**UI Review（视图检查）：**

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；sdk自带工具hierarchyviewer可以有效直观分析此类问题。此部分问题会在性能优化中展开。说几点tips

1 善用 include merge viewstub等方式布局 2 java代码布局比xml布局性能好，某些情况下可以考虑

3 利用系统定义的颜色定义 style drawable等

**Desgin Review（设计检查）：**

1 在视觉组件、常用控件的基类中避免重复的工作，子类不做基类中重复内容

2 对于较为复杂的处理，是否有成熟稳定的开源控件可以借鉴，可以在开源控件基础上定制。

参考文档：同级svn目录下

《Foxpad内存优化问题》

《Wed\_230\_400\_Putting Your App On A Memory Diet》