**Android 性能优化方案**

## 一、前言

性能优化是每个android 开发者必须关注的问题。做好性能优化有助开于发者创建更快更优秀的Android App。以下主要从界面优化、内存优化，电量优化，代码优化等四个方面介绍了如何找出性能问题以及提升性能的方法和建议。

## 二、界面优化

大多数用户感知到的卡顿等性能问题的最主要根源都是因为渲染性能。从设计师的角度，他们希望App能够有更多的动画，图片等时尚元素来实现流畅的用 户体验。但是Android系统很有可能无法及时完成那些复杂的界面渲染操作。Android系统每隔16ms发出VSYNC信号，触发对UI进行渲染， 如果每次渲染都成功，这样就能够达到流畅的画面所需要的60fps，为了能够实现60fps，这意味着程序的大多数操作都必须在16ms内完成。

如果你的某个操作花费时间是24ms，系统在得到VSYNC信号的时候就无法进行正常渲染，这样就发生了丢帧现象。那么用户在32ms内看到的会是同一帧画面。

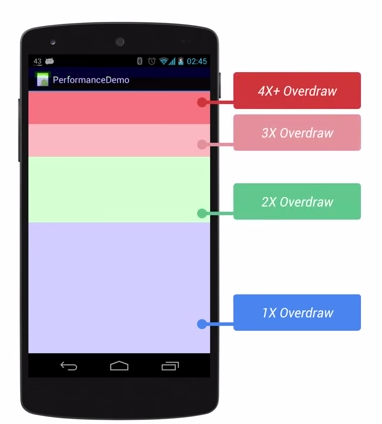
用户容易在UI执行动画或者滑动ListView的时候感知到卡顿不流畅，是因为这里的操作相对复杂，容易发生丢帧的现象，从而感觉卡顿。有很多原 因可以导致丢帧，也许是因为你的layout太过复杂，无法在16ms内完成渲染，有可能是因为你的UI上有层叠太多的绘制单元，还有可能是因为动画执行 的次数过多。这些都会导致CPU或者GPU负载过重。我们可以通过一些工具来定位问题。

#### 1、Overdraw

Overdraw(过度绘制)描述的是屏幕上的某个像素在同一帧的时间内被绘制了多次。在多层次的UI结构里面，如果不可见的UI也在做绘制的操作，这就会导致某些像素区域被绘制了多次。这就浪费大量的CPU以及GPU资源。

当设计上追求更华丽的视觉效果的时候，我们就容易陷入采用越来越多的层叠组件来实现这种视觉效果的怪圈。这很容易导致大量的性能问题，为了获得最佳的性能，我们必须尽量减少Overdraw的情况发生。

我们可以通过手机设置里面的开发者选项，打开Show GPU Overdraw的选项，可以观察UI上的Overdraw情况。





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。

Overdraw有时候是因为你的UI布局存在大量重叠的部分，还有的时候是因为非必须的重叠背景。例如某个Activity有一个背景，然后里面 的Layout又有自己的背景，同时子View又分别有自己的背景。仅仅是通过移除非必须的背景图片，这就能够减少大量的红色Overdraw区域，增加 蓝色区域的占比。这一措施能够显著提升程序性能。

#### 2、Profile GPU Rendering

打开手机里面的开发者选项，选择Profile GPU Rendering，选中On screen as bars的选项。选择了这样以后，我们可以在手机画面上看到丰富的GPU绘制图形信息。



每一条柱状线都包含三部分，蓝色代表测量绘制Display List的时间，红色代表OpenGL渲染Display List所需要的时间，黄色代表CPU等待GPU处理的时间。

随着界面的刷新，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，柱状图越高表示花费的渲染时间越长。

中间有一根绿色的横线，代表16ms，我们需要确保每一帧花费的总时间都低于这条横线，这样才能够避免出现卡顿的问题。

如果发现绘制时间过长的，我们可以通过HierarchyViewer这个工具来查看布局，使得布局尽量扁平化，移除非必需的UI组 件，这些操作能够减少Measure，Layout的计算时间。

## 三、内存优化

#### 1、内存抖动

虽然Android有自动管理内存的机制，但是对内存的不恰当使用仍然容易引起严重的性能问题。Android里面是一个三级Generation的内存模型，最近分配的对象会存放在Young Generation区域，当这个对象在这个区域停留的时间达到一定程度，它会被移动到Old Generation，最后到Permanent Generation区域。系统会根据内存中不同的内存数据类型分别执行不同的GC操作。例如，最近刚分配的对象会放在Young Generation区域，这个区域的对象通常都是会快速被创建并且很快被销毁回收的，同时这个区域的GC操作速度也是比Old Generation区域的GC操作速度更快的。除了速度差异之外，执行GC操作的时候，任何线程的任何操作都会需要暂停，等待GC操作完成之后，其他操作才能够继续运行。

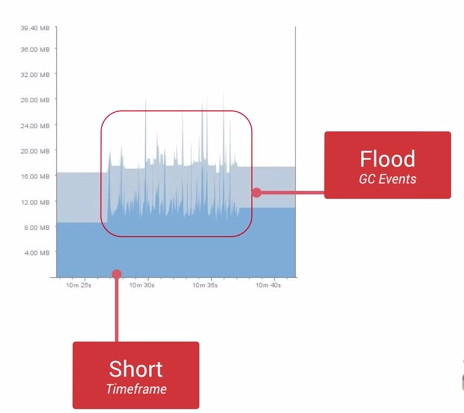
通常来说，单个的GC并不会占用太多时间，但是大量不停的GC操作则会显著占用帧间隔时间(16ms)。如果在帧间隔时间里面做了过多的GC操作，那么自然其他类似计算，渲染等操作的可用时间就变得少了。

导致GC频繁执行有两个原因：

•Memory Churn内存抖动，内存抖动是因为大量的对象被创建又在短时间内马上被释放。

•瞬间产生大量的对象会严重占用Young Generation的内存区域，当达到阀值，剩余空间不够的时候，也会触发GC。即使每次分配的对象占用了很少的内存，但是他们叠加在一起会增加 Heap的压力，从而触发更多其他类型的GC。这个操作有可能会影响到帧率，并使得用户感知到性能问题。

解决上面的问题有简洁直观方法，如果你在Memory Monitor里面查看到短时间发生了多次内存的涨跌，这意味着很有可能发生了内存抖动。



#### 2、内存泄漏

虽然Java有自动回收的机制，可是这不意味着Java中不存在内存泄漏的问题，而内存泄漏会很容易导致严重的性能问题。

内存泄漏指的是那些程序不再使用的对象无法被GC识别，这样就导致这个对象一直留在内存当中，占用了宝贵的内存空间。显然，这还使得每级Generation的内存区域可用空间变小，GC就会更容易被触发，从而引起性能问题。

寻找内存泄漏并修复这个漏洞是件很棘手的事情，你需要对执行的代码很熟悉，清楚的知道在特定环境下是如何运行的，然后仔细排查。

为了寻找内存的性能问题，我们可以使用一些工具。

•Memory Monitor：查看整个app所占用的内存，以及发生GC的时刻，短时间内发生大量的GC操作是一个危险的信号。

•Allocation Tracker：使用此工具来追踪内存的分配，前面有提到过。

•Heap Tool：查看当前内存快照，便于对比分析哪些对象有可能是泄漏。

## 四、电量优化

## 五、网络优化

## 六、代码优化

#### 1、Analyze代码分析

Analyze代码分析⼯工具，它能够分析出代码中的很多问题，⽐比如⽆无效变量量、类型检测、内存相关问题等等。对于代码中的⼩小问题，或者开发者不不⼩小⼼心犯下的错误，Analyze都能分析出来，并给出⼀一些解决问题的建议。

Analyze能够检测出⼤大部分的代码问题，并且提出解决问题的⽅方案，使修改这些问题变更加简单。

#### 2、腾讯漏洞扫描

使用腾讯漏洞扫描工具，对app 进行组件公开安全检测、AndroidManifest.xml配置安全检测、Intent劫持风险安全检测、数据存储安全检测、WebView高危接口安全检测、数据传输安全检测、拒绝服务攻击安全检测等其他安全风险安全检测，及时发现安全漏洞并修复。



## 七、实施计划

1、4.2版本理财师功能开发完成后，可以对发现页及内部所有页面进行性能优化。

2、以后每个迭代根据迭代功能针对相应的功能模块进行性能和代码优化。

3、计划第三季度，能够完成所有功能都优化一遍。

4、以后每个季度，要对app进⾏行一次全功能的性能和代码优化