检测和解决Android应用的性能问题

devtf.cn

前言

无论你的应用多么有创新性、有用，如果它卡得要命，或者非常消耗内存，那么没人会愿意使用它。

因此，性能变得尤为重要。当你忙碌于构建精美的用户界面或者完成新的特性时，你可能容易忘却掉一些性能相关的事情。

这也是为什么有Google Play的应用审核机制的原因之一。

这篇文章中，你会看到每个Android工程师需要了解的一些性能问题。你将会学会使用Android SDK提供的、已安装在你的设备中的工具来测试这些问题是否发生在你自己的应用中。

如果在你的应用中发现了一个性能问题，你肯定会想修复它。我们还会看一看如何使用Android SDK 工具来获取更多关于那些没有覆盖到的性能问题的相关信息。一旦你有了这些信息，你将会对如何提升应用性能有一个更深刻的理解，并且能够构建出让用户喜爱的App。

过度绘制

步骤1：问题描述

你应用的用户界面是连接用户的纽带，但是创建漂亮的界面只是挑战的其中一面，你还需要确保用户界面流畅的运行。

一个常见的问题就是用户界面卡顿，出现这种情况的原因可能是**overdraw**。Overdraw是屏幕上的某个像素在同一帧的时间内被绘制了多次。

例如，想象一下一个有蓝色的背景文本，Android不仅会绘制对用户可见的蓝色区域，而是会绘制整个蓝色的背景以及上面的文本。这意味着一些像素会被两次绘制，这就是过度绘制。

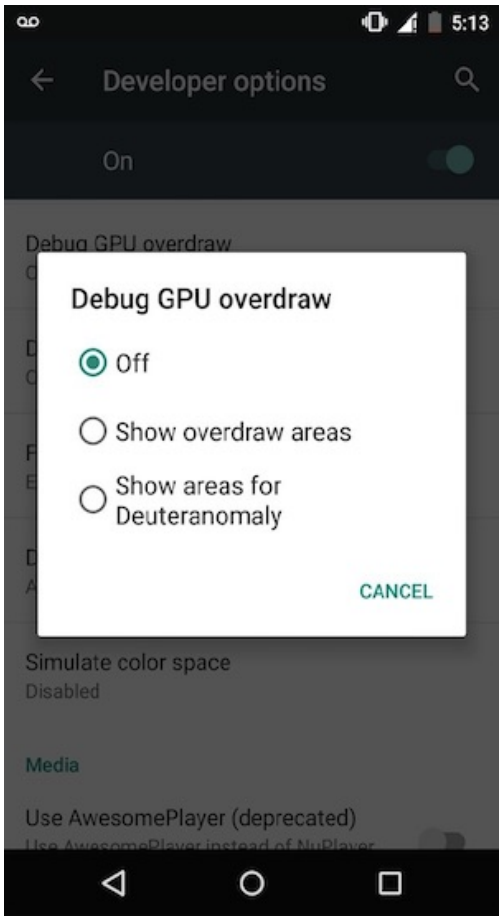
一些如上述例子所说的过度绘制示例是不可避免的。然而，过多的多度绘制会引发明显的性能问题，因此你必须将过度绘制的可能性降到最小。

检测应用中的过度绘制相对来说比较简单。大量的过度绘制会引出其他用户界面相关问题，例如视图层级过于复杂等。基于这些原因，当你测试你的App的性能问题时，从过度绘制开始是一个明智的选择。

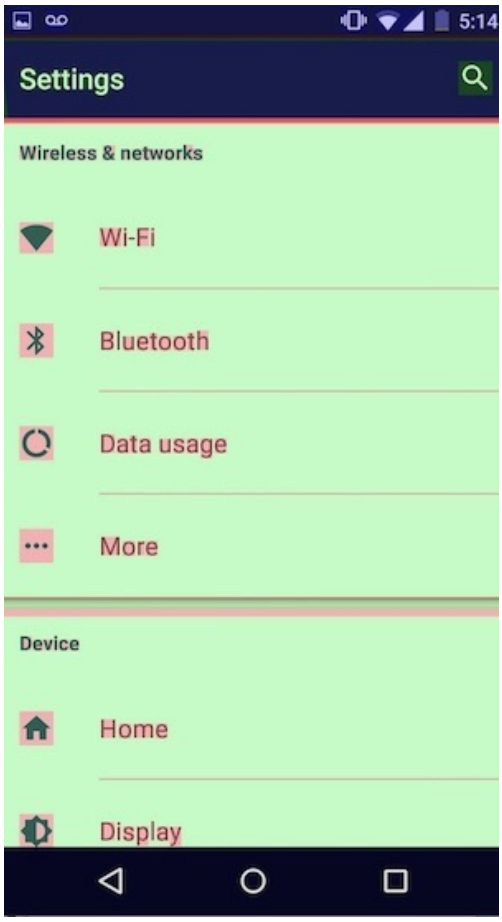
步骤2：检测过度绘制

好消息是你的Android设备上已经内置了检测过度绘制的工具。

因此你需要做的第一步就是安装你要测试的App到你的设备中。然后打开设置页面，选择开发选项（Developer Options）->调试GPU 过度绘制（Debug GPU Overdraw），然后选择“显示过度绘制区域（Show overdraw area）”。如下图所示。



这个工具使用色块来代表不同数量的过度绘制。剩下的事情就是启动你要测试的应用，然后观察它的过度绘制情况。



- **没颜色**：没有过度绘制，也就是说一个像素只被绘制了一次。
- **蓝色**：过度绘制了一次，也就是一个像素点被绘制了两次。
- **绿色**：过度绘制了2次. 也就是一个像素点被绘制了三次，通常，你需要集中精力优化过度绘制次数大于等于2次的情况。
- **浅红色**：过度绘制3次。这取决于你的应用，小范围的3次过度绘制可能是不可避免的，但是如果你看到了中等或者大量的红色区域那么你就需要查找出现这个问题的原因了。
- **深红色**：过度绘制4次，像素点被绘制了5次，甚至更多次。出现这种问题你绝逼要找到原因，并且解决它。

步骤3： 最小化过度绘制

一旦你发现了某个区域有严重的过度绘制，最简单的方法就是打开你应用的xml文件找到过度重叠的区域，特别是那些不可见的drawable对象和被绘制在其他控件上的背景，以此来降低这些地方的过度绘制。

你也应该查找那些背景属性设置为白色，并且它的父视图背景也设置为白色的区域。所有这些都会引起严重的过度绘制。

Android系统能自动的降低一些简单的过度绘制，但是这些对于复杂的自定义View并没有什么价值，因为Android不会知道你如何绘制你的内容。

如果你在App中使用了复杂的自定义View，你可以为使用clipRect函数为你的视图定义可绘制区域的边界。更新相关信息可以参考official Android documentation.

2. Android 图形渲染

步骤1：问题描述

另一个常见的性能问题就是应用的视图层级。为了渲染每个视图，Android都会经历这三个阶段：

花在这三个阶段的时间与View层级中的View的数量成正比，这就意味着降低App渲染时间的最简单的方法就是识别和移除那些并没有什么卵用的UI元素。

即使在你的视图层级上的所有View都是必须的，不同的布局方式也可能对测量过程产生重要的影响。通常来说，你的视图层级越深，花在测量视图的时间就越长。

在视图渲染期间，每个View都要向它的父View提供它自己的尺寸。如果父view发现了任意一个尺寸问题，那么它会强制要求所有的子视图重新测量。

即使没有错误发生，重新测量也可能出现。例如，为了正确的进行布局RelativeLayout通常会对它们的子视图进行两次测量。子视图使用了layout_weight属性的LinearLayout也会对它的子视图进行两次测量。

这些都取决于你的布局方式，测量和重新测量的代价非常昂贵，它会严重影响你的渲染速度。

确保你的用户界面渲染流畅的关键就是移除任何非必须的View以及减少你的View层级。

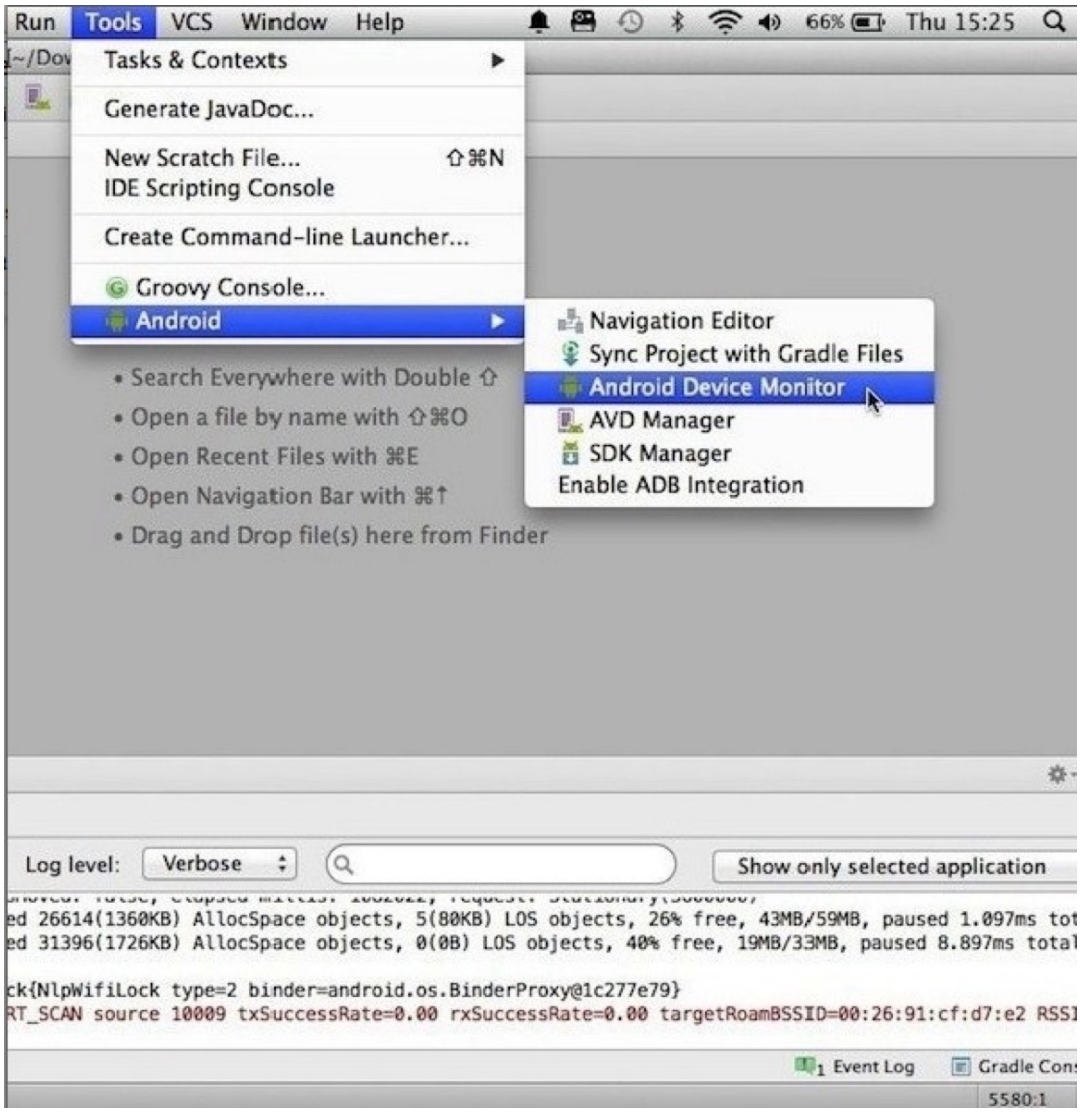
Hierarchy Viewer是一个能够将你完整的View层级可视化的工具，这个工具能够帮助你发现冗余的View以及嵌套的布局。

步骤2：使用 Hierarchy Viewer

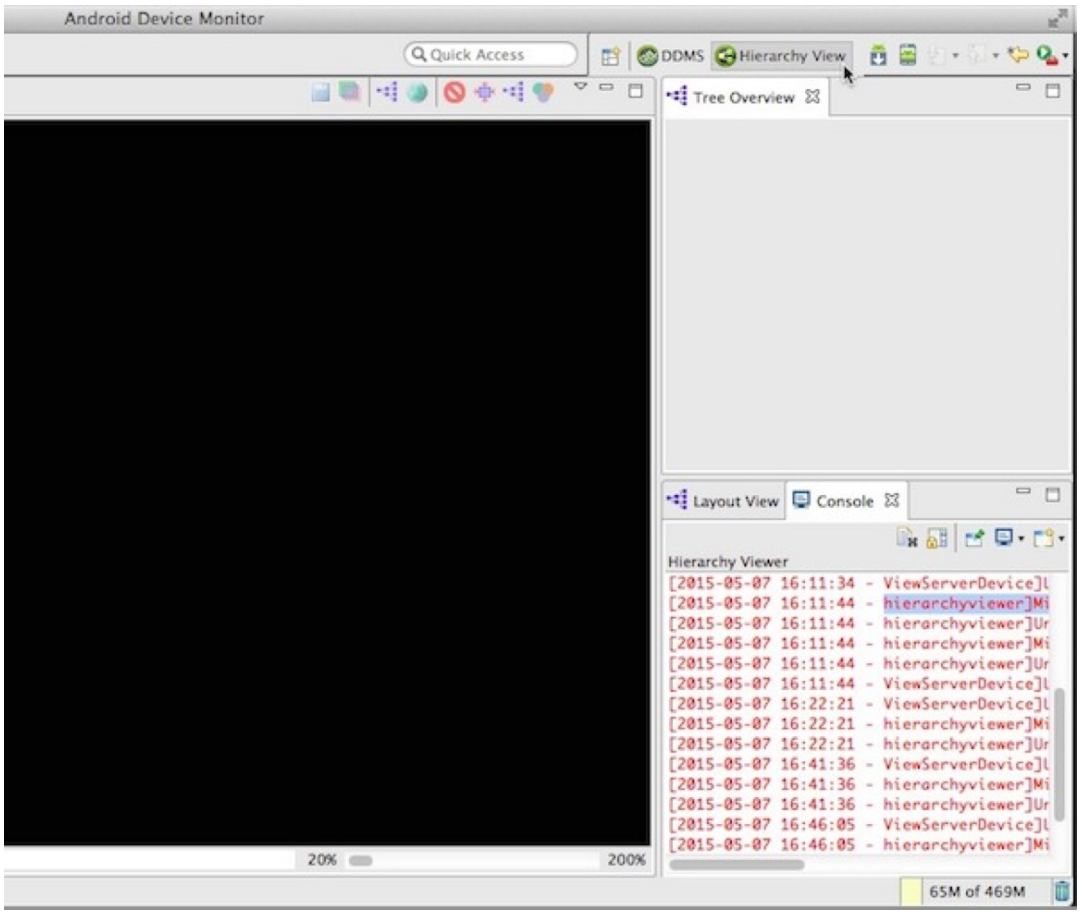
在我们进一步了解**Hierarchy Viewer**之前，你需要知道它的一些规则。首先**Hierarchy Viewer**只能与正在运行的App进行交互，而不是你的源代码。这就是说你需要将App安装到你的设备或者模拟器上。

还有一个最重要的问题，就是默认情况下**Hierarchy Viewer**只能与运行开发版Android系统的设备进行交互（译者注: 一般来说，使用模拟器即可）。如果你没有开发设备，那你需要添加ViewServer
class到你的应用中。

了解这些之后就让我们打开Android Studio，并且选择”tools” -> “Android” -> “Android Device Monitor”，如图所示。



然后点击Hierarchy View按钮，如下图所示。



屏幕左边的Windows标签下列出了所有Android设备和模拟器。选择你的设备后，你会看到你设备上运行的所有进程。选中你要检测的进程，然后你会看到三个自动更新的视图层级区域。

这三个窗口提供了视图层级的三个不同可视化展示。

- **Tree View:** **** 视图层级窗口，每个节点代表了一个View;
- **Tree Overview:** 整个视图层级的缩略布局;
- **Layout View:** 当前视图层级的轮廓.

Hierarchy View中有三个窗口。如果你在一个窗口中选择了一个View，那么它会在另外两个中高亮显示。你能同时使用这三个窗口查找View层级中的冗余视图。

如果你不确定一个View是否是UI界面中的必须元素，最简单的方法就是到Tree View窗口点击这个节点。你将会看到该View是如何显示在屏幕的预览，此时你就可以确切地知道该View是否是必须的。

但是即使一个View对最终的渲染效果有贡献也并不意味着它不会引起严重的性能问题。你已

经看到了如何通过Hierarchy Viewer来找到明显的嵌套布局，但是如果这引起的性能问题并不那么明显呢？或者还有其他的原因使得该视图渲染得非常慢？

好消息就是你还可以通过Hierarchy Viewer来剖析每个View在不同的渲染阶段的耗时。当性能问题的原因不那么明显时，这是你发现问题的另一途径。

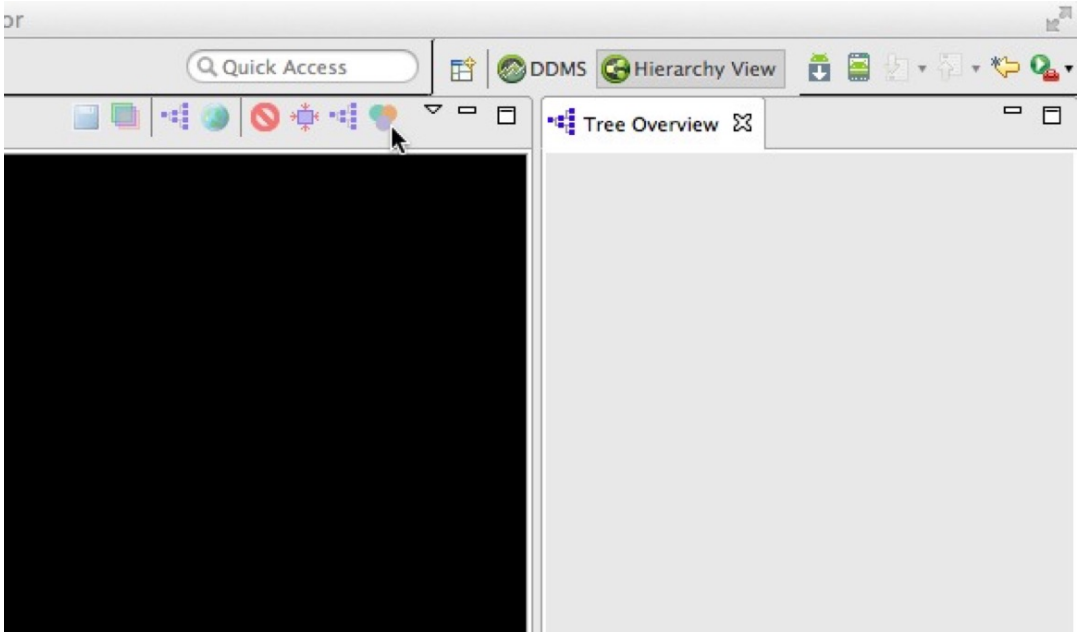
下个章节我将为你展示如何通过Hierarchy Viewer来剖析每个View的渲染时间来找到潜伏在问题表面的性能问题。

步骤3：节点的性能分析

定位你的用户界面瓶颈的最简单方法就是收集每个View分别完成测量、布局、绘制的时间。

你不仅可以通过Hierarchy Viewer收集这些信息，Hierarchy Viewer还可以通俗易懂地向你展示这些数据，因此你可以通过这种形式来找到性能问题。

Hierarchy Viewer默认并不会显示渲染时间。你需要到Tree View窗口添加这个信息，然后选择你想要测试的根节点。下一步，在Hierarchy Viewer上点击由绿、红、紫的三个圆形色块组成的按钮，如图所示。



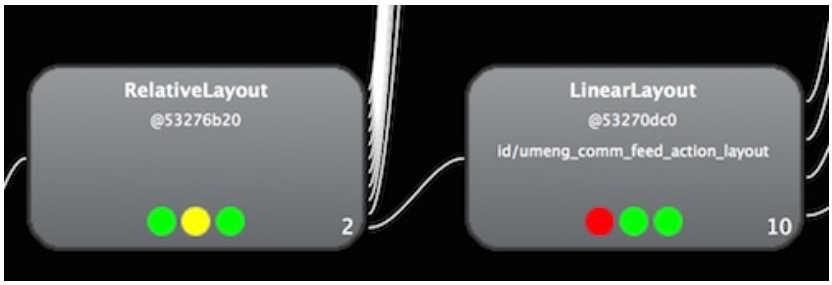
三个圆点色块就会显示在每个节点上，从左到右，这些圆点分别代表：

- 用于测量的时间
- 用于布局的时间

- 用于绘制的时间

每个圆点都有颜色：

- **绿色** 代表该View的渲染速度至少要快于一半以上的其他参与测试的节点。例如，一个在布局位置上的绿色的圆点代表它的布局速度要快于50%以上的其他节点；
- **黄色** 代表该View慢于50%以上的其他节点;
- **红色** 代表该View的渲染速度比其他所有参与测试的节点都慢。



当收集了这些数据之后，你不仅知道哪些View需要优化，你还会确切地知道是在渲染的哪个阶段导致的问题。

哪些黄色、红色的地方就是你需要开始优化的地方，这些性能指标与该视图层级下的其他剖析节点也有关系。换句话说，你肯定会有一些视图渲染得比其他的慢。

在开始改良你的View相关的代码之前，摸着你的良心问一句该View渲染得比其他视图慢是否有一个合理的原因，如果答案是否定的，那么就开始你的View优化之旅吧。

3. Memory Leaks 内存泄漏

步骤1：问题描述

Android是一个自动管理内存的开发环境，不要让这个句话蒙蔽了，因为内存泄漏依旧是可能发生的。这是因为垃圾回收器只会移除那些不可达的对象。如果它不是一个不可达的对象，那么该对象就不会被释放掉。

这些不可达的对象阴魂不散，聚集在你的堆内存中，占用App的内存控件。如果你继续泄漏对象，那么可用的内存空间将会越来越小，GC操作就会频繁触发。

有两个原因表明这是一个坏消息。首先，GC操作通常不会明显地影响你的App性能，但是当内存控件较小时大量的GC操作会使你的App变慢，此时UI就不会那么流畅了。第二问题是移

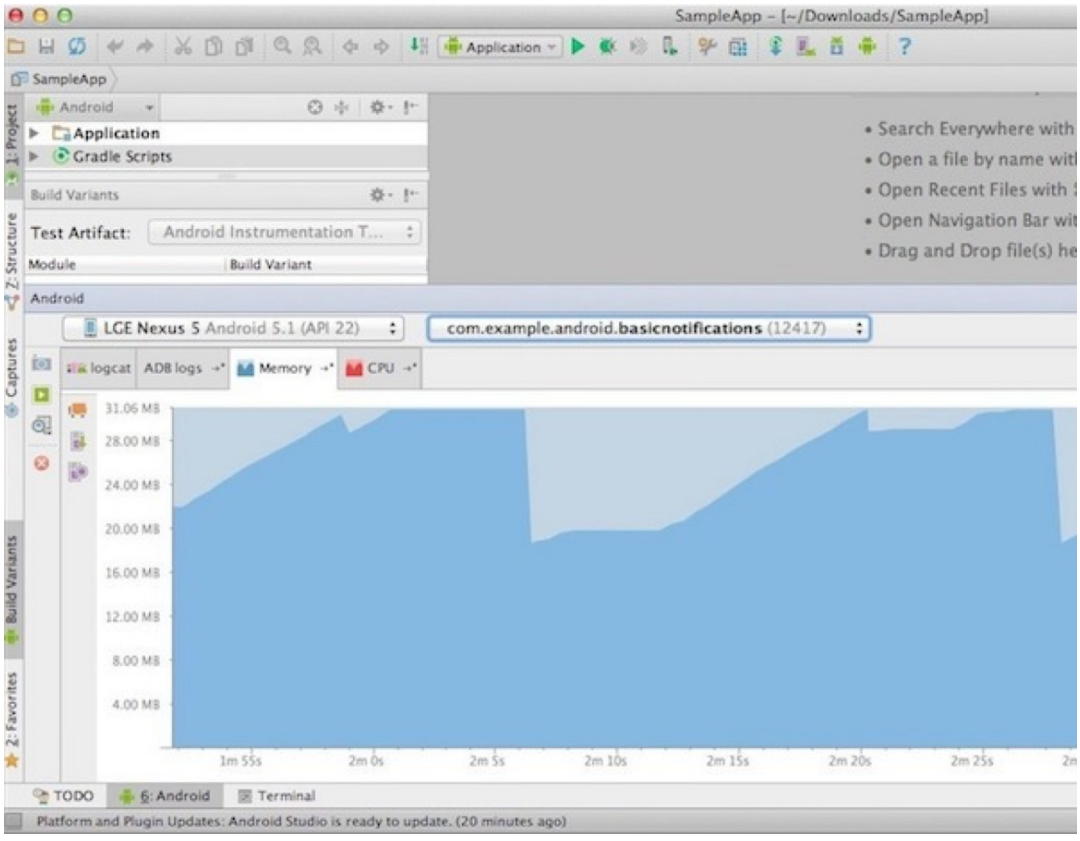
动设备的内存空间相对来说较小，因此内存泄漏会快速地升级为内存溢出，导致应用Crash。

内存泄漏难以被检测出。可能只有当用户开始抱怨你的应用时你才能发觉内存泄漏问题。幸运地是，Android SDK提供了一些有用的工具来让你找到这些问题。（译者注：Square的开源库LeakCanary是查找内存泄漏的优秀工具,强烈建议大家使用）。

步骤2：内存监视器 （Memory Monitor）

Memory Monitor是一个能够实时获取应用内存使用情况的工具。需要注意的是这个工具只能作用于正在运行的应用，因此确保你的要测试的应用已经安装到你的设备中，并且你的设备已经连接到你的电脑上。

Memory Monitor已经内置在Android Studio中，因此你可以点击Android Studio的底部的”Memory”这个tab来切换到内存监视页面。当你切换到该页面的时候，Memory Monitor就开始记录你的内存使用情况了。



如果Memory Monitor没有开始记录，那么确保你的设备是已经被选中的状态。

如果Memory Monitor提示**No debuggable applications**，那么你可以打开Android Studio的”Tools”菜单，选择”Android”，然后确保选中了**Enable adb integration**。这个功能还不是很

稳定，所以有时候你需要手动切换它的状态。你也可以断开设备与电脑的连接，然后再重连，这样可能就OK了。

一旦Memory Monitor检测到正在运行的应用，它就会显示这个应用的内存使用情况。已使用的内存会被表示为深蓝色，未分配的内存则会变为浅蓝色。

花一些时间与你的设备交互，并且关注你的内存使用情况。最终已分配的内存会增长，直到没有内存可用。此时，系统就会释放触发GC释放内存，当你看到已分配的内存明显的下降时就代表GC操作被触发了。

GC通常情况下会将无用的内存释放掉，但是当你看到App在短时间内快速增长或者GC变得非常频繁，此时你就需要倍加小心了，这就是发生内存泄漏的信号！

如果你通过Memory Monitor来追踪一个可疑的内存泄漏问题，你可能会看到Android系统会为你的App增大可用内存，TODO：。

最终，你可能会看到你的App消耗了非常多的内存以至于系统无法再给你的应用更多的可用内存。如果你看到这种场景，那么说明你在内存使用上犯了很严重的错误。

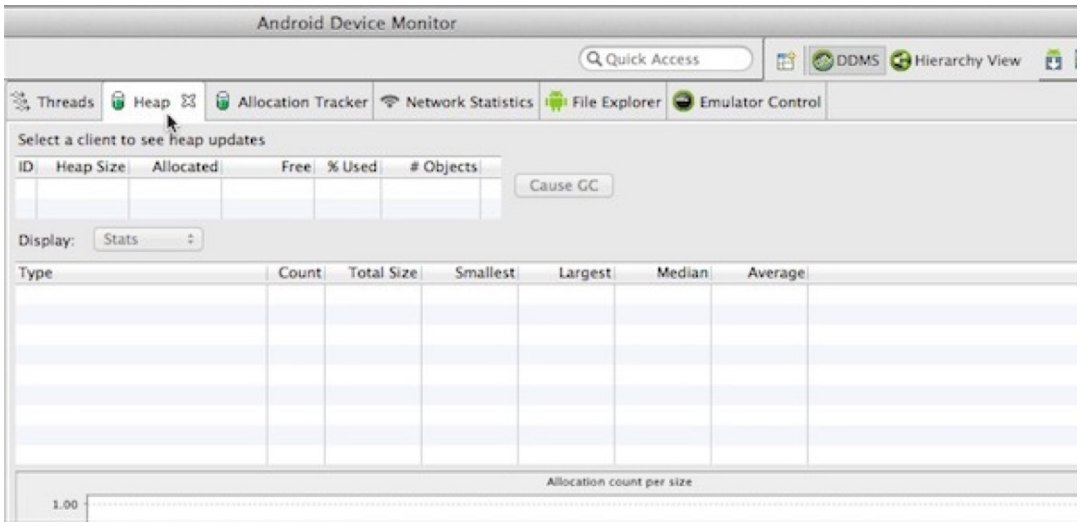
步骤3：Android Device Monitor

另一个能够帮助你收集更新关于内存泄漏信息和其他内存相关问题的工具是Android Device Monitor的DDMMS工具下的**Heap**。

Heap工具能够通过显示系统为你分配了多少内存来帮助你诊断内存泄漏问题。正如上面提到的，如果已分配的内存不断地增长，那么这是发生内存泄漏的明显信号。

但是这个工具还提供了许多关于你的应用堆内存使用情况的数据，包含你的App内分配的各种对象、分配的对象数量以及这些对象占用了多少空间。这些额外的信息对于你追踪内存泄漏极为有用。

你可以在Android Device Monitor工具中选择DDMS，在Devices中选择你要检测的App。然后选择Heap标签，如图所示。然后花一些时间与你的App进行交互以收集内存信息。



heap输出信息会在GC事件之后，因为你可以手动点击**Cause GC**来触发GC，使得Heap内存数据尽快地显示出来。

一旦GC事件被触发了，heap标签下就会更新App的堆内存使用信息，这些信息会在每次GC时更新。

总结

在这篇文章中，我们学习了一些开发中最常见的性能问题，过度绘制、内存泄漏、缓慢的UI渲染。

相信你已经掌握了如何使用工具来检查这些问题，以及如何获取更新的信息来判断你的应用中是否出现了这些性能问题。你有越多的信息，就越容易追踪到问题的原因并且修复它。

Android SDK有很多工具可以供你诊断和定位性能问题。如果你想学习更多这方面的知识，你可以访问u这两篇官方文档 Traceview and dmtracedump和 Allocation Tracker.