紫光展锐技术月刊

应用APP流畅性分析方法

(2018年12月）

Contents

绘制原理

Android系统要求每一帧都要在 16ms 内绘制完成，平滑的完成一帧意味着任何特殊的帧需要执行所有的渲染代码（包括 framework 发送给 GPU 和 CPU 绘制到缓冲区的命令）都要在 16ms 内完成，保持流畅的体验。这个速度允许系统在动画和输入事件的过程中以约 60 帧每秒（ 1秒 / 0.016帧每秒 = 62.5帧/秒 ）的平滑帧率来渲染。

掉帧

掉帧是用户体验中一个非常核心的问题。丢弃了当前帧，并且之后不能够延续之前的帧率，这种不连续的间隔会容易会引起用户的注意，也就是我们常说的卡顿、不流畅。

引起掉帧的原因非常多，比如：

- 花了非常多时间重新绘制界面中的大部分东西，这样非常浪费CPU周期；
- 过度绘制严重，在绘制用户看不到的对象上花费了太多的时间
- 有一大堆动画重复了一遍又一遍，消耗 CPU 、 GPU 资源；
- 频繁的触发垃圾回收。

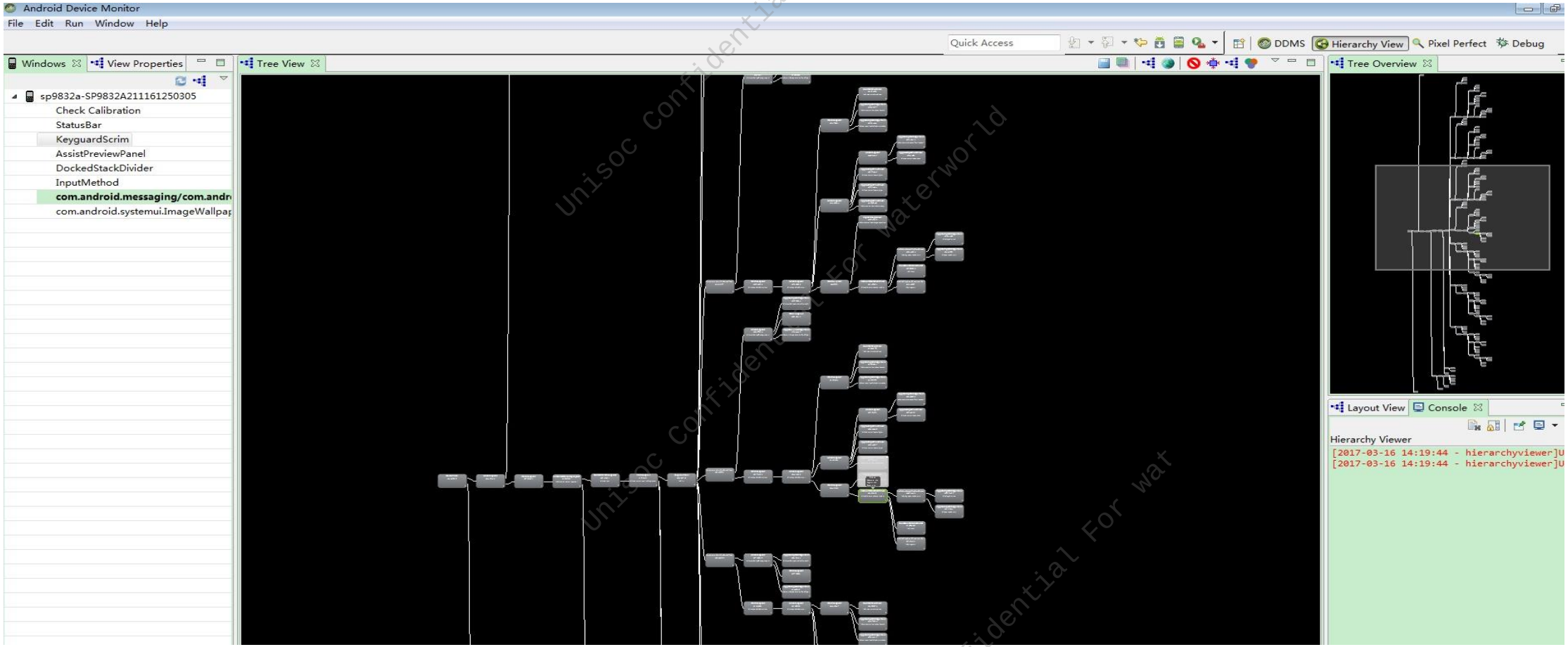
检测维度

根据业务的不同与所需要的测试粒度的不同，就会有不同的检测维度。

- 界面过度绘制；（检测过度绘制）、
- 渲染性能；（检测严格模式下的UI渲染性能呈现）
- 布局边界合理性；（检测元素显示的合理性）

Hierarchy View

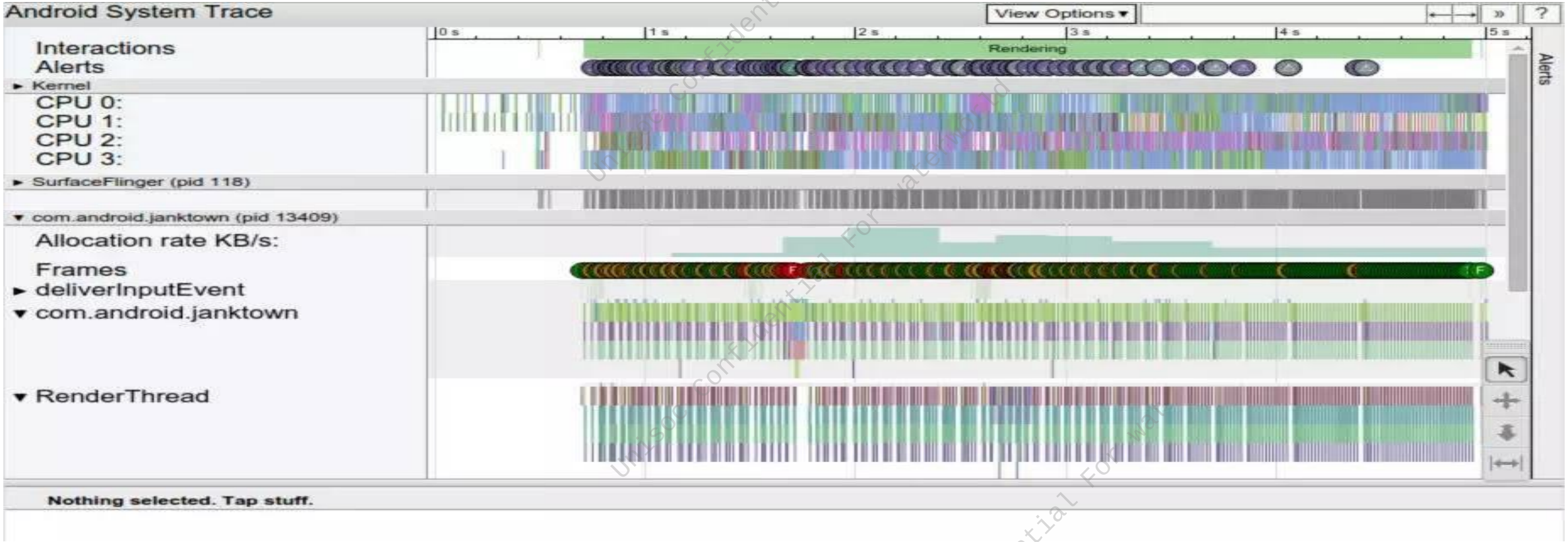
常用来查看界面的视图结构是否过于复杂，用于了解哪些视图过度绘制，又该如何进行改进



常用工具介绍

Systrace

用来跟踪 graphics 、view 和 window 的信息，发现一些深层次的问题。

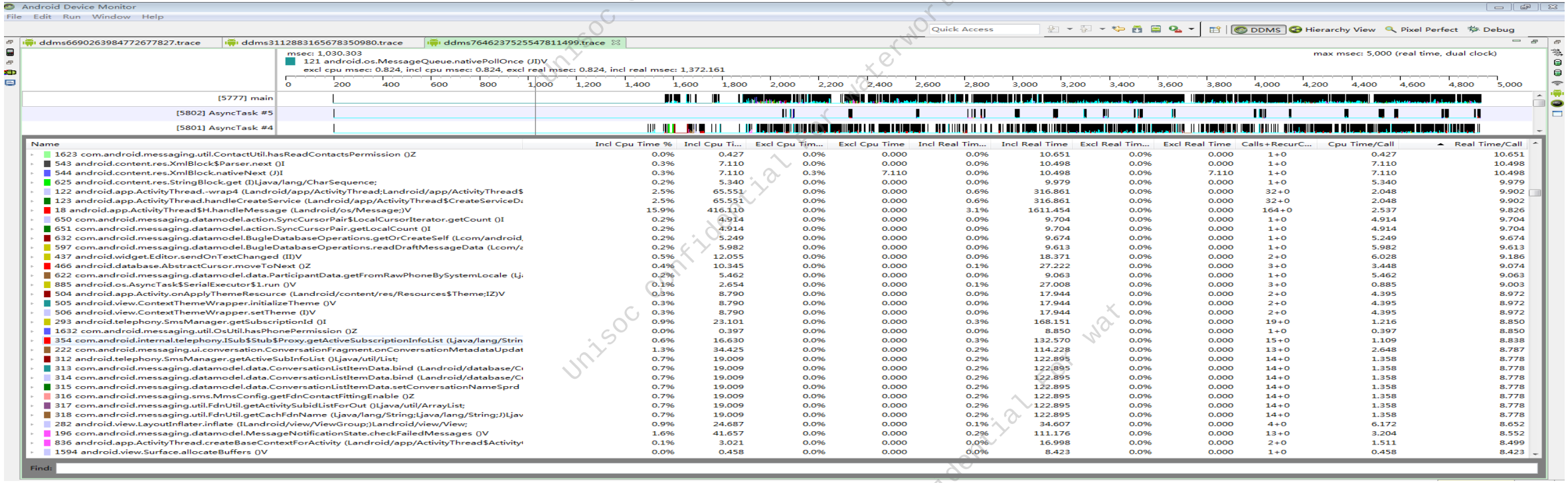


TraceView

有两方面用途：

1 查看跟踪代码的执行时间，分析哪些是耗时操作

2 可以用于跟踪方法的调用，尤其是Android Framework层的方法调用关系



常用工具介绍

名	描述
Name	该线程运行过程中所调用的函数名
Incl Cpu Time	某函数占用的CPU时间，包含内部调用其它函数的CPU时间
Excl Cpu Time	某函数占用的CPU时间，但不含内部调用其它函数所占用的CPU时间
Incl Real Time	某函数运行的真实时间（以毫秒为单位），内含调用其它函数所占用的真实时间
Excl Real Time	某函数运行的真实时间（以毫秒为单位），不含调用其它函数所占用的真实时间
Call+Recur Calls/Total	某函数被调用次数以及递归调用占总调用次数的百分比
Cpu Time/Call	某函数调用CPU时间与调用次数的比。相当于该函数平均执行时间
Real Time/Call	同CPU Time/Call类似，只不过统计单位换成了真实时间

OverDraw

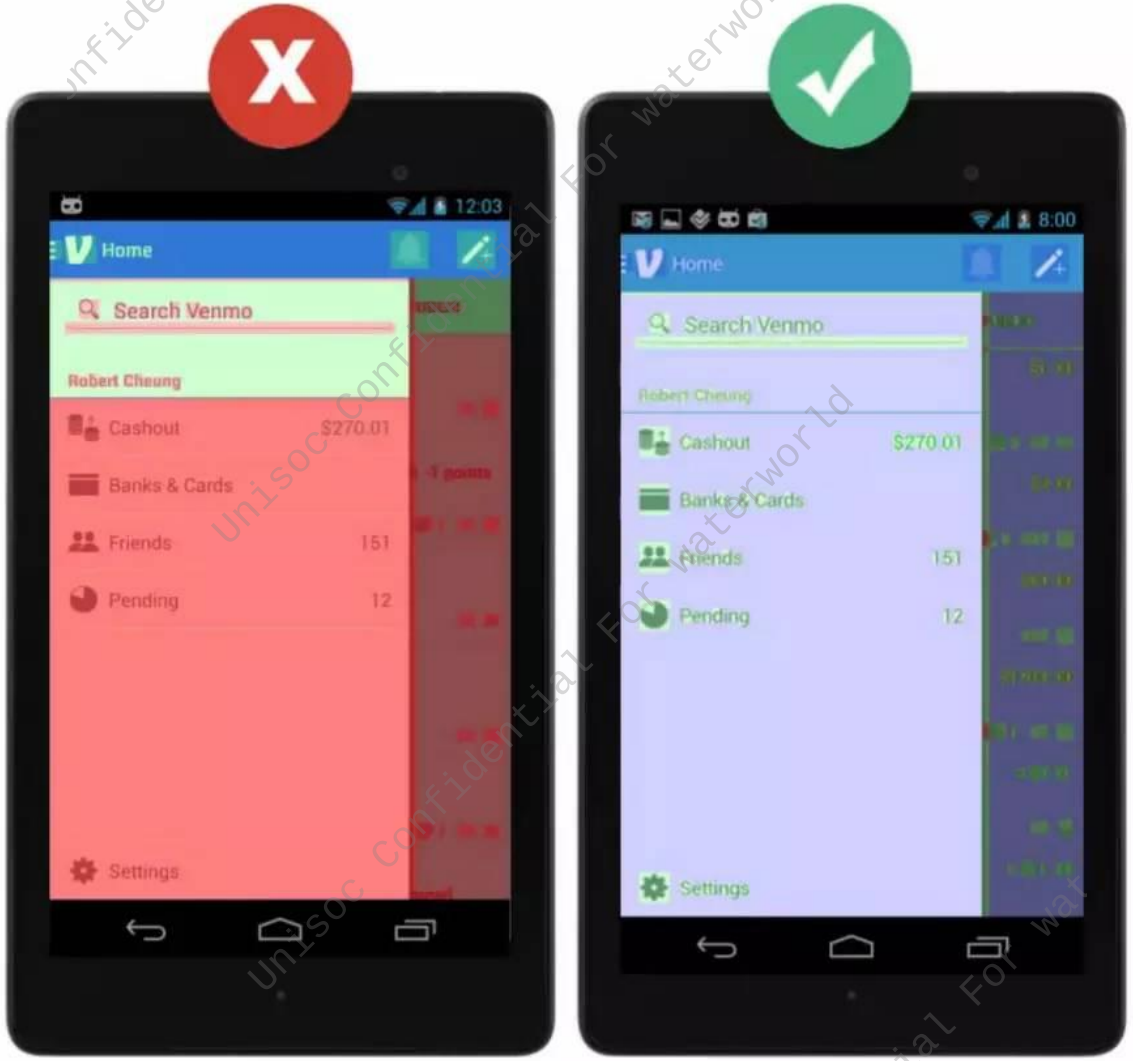
- 过度绘制是一个术语，表示某些组件在屏幕上的一个像素点的绘制次数超过 1 次

- 通过在 Android 设备的设置 APP 的开发者选项里打开 “ 调试 GPU 过度绘制 ” ，来查看应用所有界面及分支界面下的过度绘制情况，方便进行优化。

Android 会在屏幕上显示不同深浅的颜色来表示过度绘制：

- 没颜色：没有过度绘制，即一个像素点绘制了 1 次，显示应用本来的颜色；
- 蓝色：1倍过度绘制，即一个像素点绘制了 2 次；
- 绿色：2倍过度绘制，即一个像素点绘制了 3 次；
- 浅红色：3倍过度绘制，即一个像素点绘制了 4 次；
- 深红色：4倍过度绘制及以上，即一个像素点绘制了 5 次及以上；

常用工具介绍



GPU 呈现模式分析

- 通过在 Android 设备的设置 APP 的开发者选项里打开 “ GPU 呈现模式分析” 选项，选择 “ 在屏幕上显示为条形图;
- 这个工具会在Android 设备的屏幕上实时显示当前界面的最近 128 帧 的 GPU 绘制图形数据，包括 StatusBar 、NavBar 、 当前界面的 GPU 绘制图形柱状图数据。我们一般只需关心当前界面的 GPU 绘制图形数据即可

常用工具介绍

- 每一个柱状图都是由三种颜色构成：蓝、红、黄。

- 蓝色代表的是这一帧绘制 Display List 的时间。通俗来说，就是记录了需要花费多长时间在屏幕上更新视图。用代码语言来说，就是执行视图的 onDraw 方法，创建或更新每一个视图的 Display List 的时间。

- 红色代表的是这一帧 OpenGL 渲染 Display List 所需要的时间。通俗来说，就是记录了执行视图绘制的耗时。用代码语言来说，就是 Android 用 OpenGL ES 的 API 接口进行 2D 渲染 Display List 的时间。

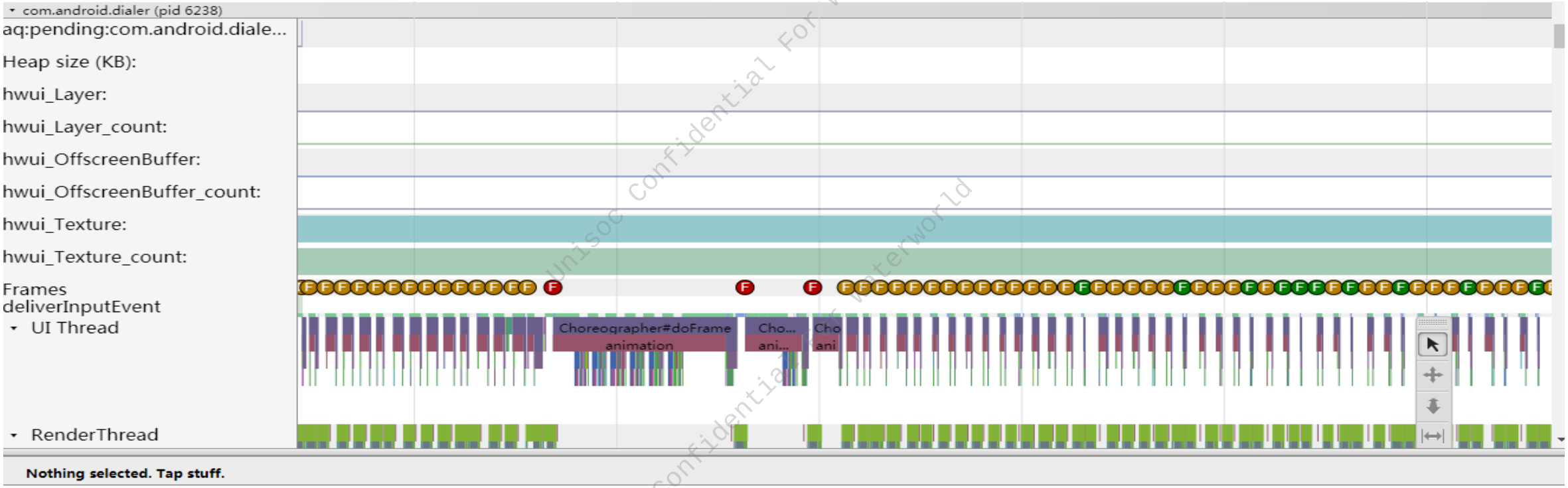
- 黄色代表的是这一帧 CPU 等待 GPU 处理的时间。通俗来说，就是 CPU 等待 GPU 发出接到命令的回复的等待时间。用代码语言来说，就是这是一个阻塞调用。

原则

- 对于性能问题，分析和改善有必要遵循以下原则：
- 一切看数据说话，不能跟着感觉走，感觉哪有问题就去改，很有可能会适得其反；
- 性能优化是一个持续的过程，需要不断地改善，不要想着一气呵成；
- 对于性能问题，不一定必须要改善，受限于架构或者其它原因某些问题可能会很难改善，必须要先保证能用，再才考虑好用。
- 改善后一定要验证，任何一个地方的改动都需要验证，避免因为改善性能问题导致其它的问题。

丢帧问题分析

Systrace中的帧



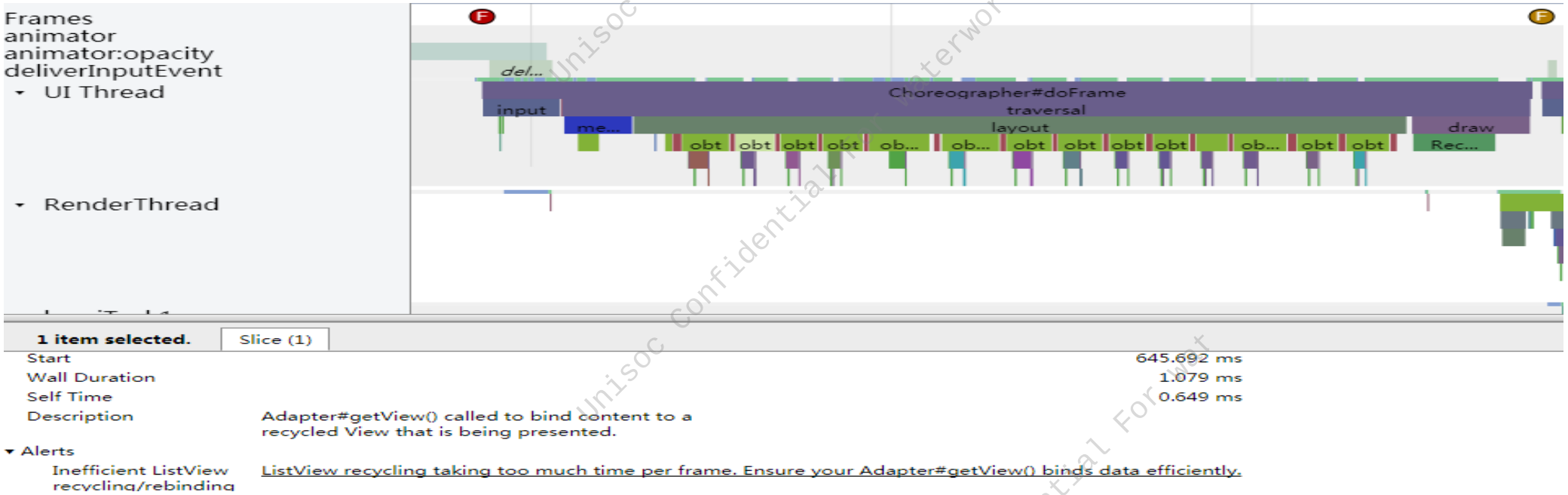
丢帧问题分析

导致掉帧的一些原因：

- 进入界面的时候layout过于复杂会导致进入时丢帧

体现在systrace中是每帧的doFrame过长。例如：

layout耗时过长做了十几次obtainview，从描述信息可以看到是Adapter#getView()进行的调用。

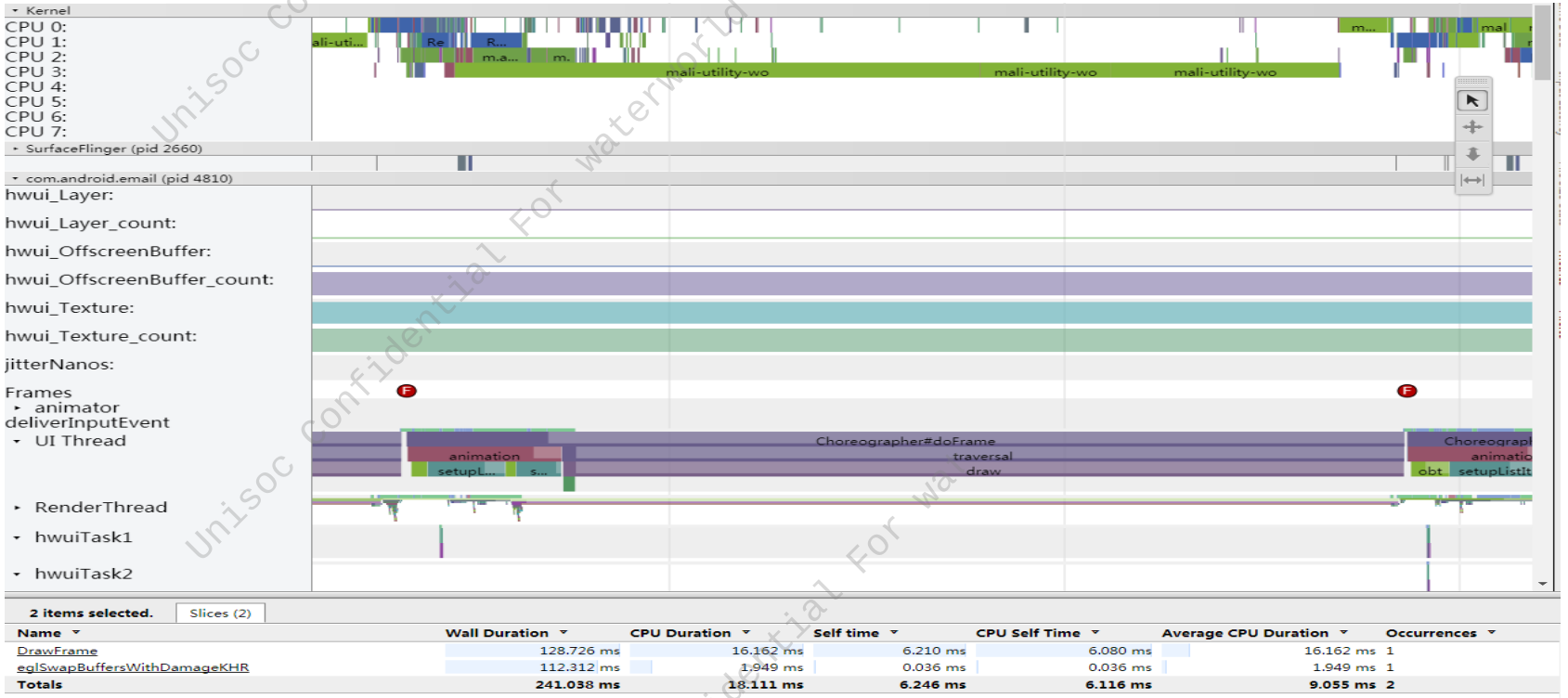


丢帧问题分析

导致掉帧的一些原因：

- GPU接口问题

体现在systrace中是每帧的DrawFrame过长。例如：eglSwapBuffersWithDamageKHR耗时长，一直在Sleeping状态,此时CPU一直在做mali-utility-wo，通过添加tag和查看代码可以定位到cobjp函数接口问题.



丢帧问题分析

导致掉帧的一些原因：

- 过多的动画效果，或者过度绘制严重

  如下图中动画效果惊人，由于画一帧的时间基本上是固定的16ms，所以过多的动画效果会拖延性能。

声明

本文件所含数据和信息都属于紫光展锐所有的机密信息，紫光展锐保留所有相关权利。本文件仅为信息参考之目的提供，不包含任何明示或默示的知识产权许可，也不表示有任何明示或默示的保证，包括但不限于满足任何特殊目的、不侵权或性能保证。本文件中的性能指标、测试结果和参数等，均为在紫光展锐内部研发和测试系统中获得的，仅供内部参考，若需要商用或量产，需要结合自身的软硬件测试环境进行全面的测试和调试。 当您接受这份文件时，即表示您同意本文件中内容和信息属于紫光展锐机密信息，且同意在未获得紫光展锐书面同意前，不使用或复制本文件的整体或部分，也不向任何其他方披露本文件内容。紫光展锐有权在未经事先通知的情况下，在任何时候对本文件做任何修改。除合同另有书面约定外，紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负责任何与本文件相关的直接或间接的、任何伤害或损失。

THANK YOU!