



APP 性能优化与分析CPU篇（一）-CPU Profiler

 林栩 关注

 0.287 2020.11.29 22:30:28 字数 3,028 阅读 1,220

APP 性能优化与分析CPU篇（一）-CPU Profiler

一、前言

在APP的性能分析中，CPU的占用分析是一个绕不过的话题。当一个APP占用的CPU过高时，会使手机的耗电增加、发烫，严重的还会造成响应速度变慢、ANR等严重的问题。所以当APP出现以上性能问题，我们首先应该做得就是分析APP对于CPU的性能影响。

在Android中CPU性能分析工具很多，恰好Android Studio更新了新的CPU分析工具，所以我们就首先来看一下Android Studio自带的CPU Profiler的使用。

本篇暂不涉及实战解决问题，等我们讲完了常用的CPU分析工具后，再来结合一个案例，巩固一下几种工具的使用和优缺点。

二、CPU Profiler 概览

- 打开CPU Profiler

1.在Android Studio（以下简称AS）中点击Profile按钮

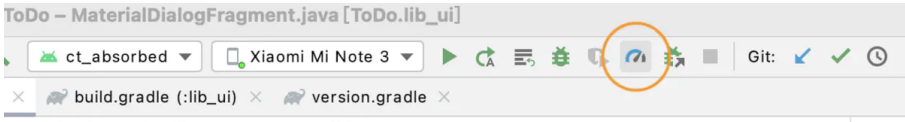


图1

2.AS中弹出如下界面，图片中标记出来的部分就是CPU Profiler

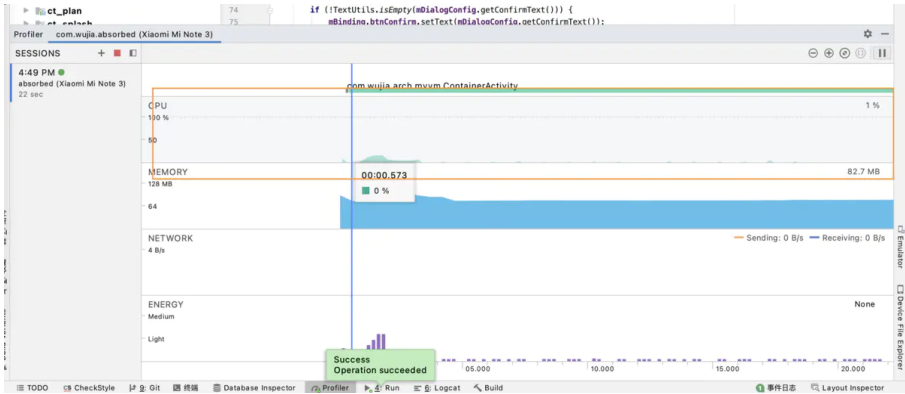


图2

3.点击图2标记的部分，查看CPU Profiler的详细信息

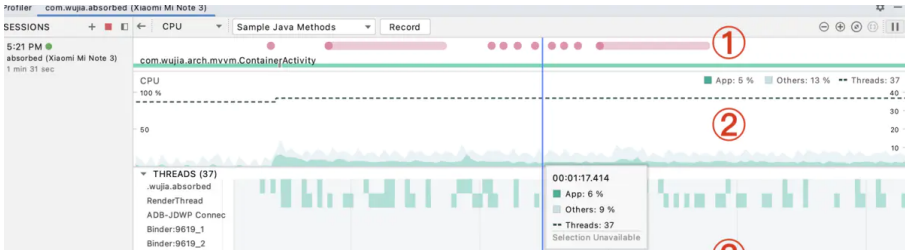


图3

如图3所示，CPUProfiler的默认视图包括以下两个重要的时间轴

- ①事件时间轴：显示应用中的 Activity 在其生命周期内不断转换而经历各种不同状态的过程，并指示用户与设备的交互，包括屏幕旋转事件、点击事件、长按事件等等。
 - ② CPU时间轴：显示应用的实时 CPU 使用率（以占总可用 CPU 时间的百分比表示）以及应用当前使用的线程总数。此时间轴还显示其他进程（如系统进程或其他应用）的 CPU 使用率，以便您可以将其与您应用的使用率进行对比。您可以通过沿时间轴的水平轴移动鼠标来检查历史 CPU 使用率数据
 - ③ 线程活动时间轴：列出属于应用进程的每个线程，并使用下面列出的颜色在时间轴上指示它们的活动。记录跟踪数据后，您可以从此时间轴上选择一个线程，以在跟踪数据窗格中检查其数据。
- 绿色：表示线程处于活动状态或准备使用 CPU。也就是说，它处于正在运行或可运行状态。
- 黄色：表示线程处于活动状态，但它正在等待一项 I/O 操作（如磁盘或网络 I/O），然后才能完成它的工作。
- 灰色：表示线程正在休眠且没有消耗任何 CPU 时间。当线程需要访问尚不可用的资源时，有时会发生这种情况。在这种情况下，要么线程自主进入休眠状态，要么内核将线程置于休眠状态，直到所需的资源可用。
- 白色：表示线程已经死亡。

CPU Profiler 还会报告 Android Studio 和 Android 平台添加到应用进程的线程的 CPU 使用率，这些线程包括 JDWP、Profile Saver、Studio:VMStats、Studio:Perfa 和 Studio:Heartbeat 等（然而，它们在线程活动时间轴上显示的确切名称可能有所不同）。也就是说CPU Profiler 会报告一些与并非应用中的线程状态，以便开发者确定线程活动和 CPU 使用率实际在何时是由您的应用的代码引发。

CPU时间轴的右上角我们还能看到APP：5%，Other：13%，Thread：37的图例，它分别表示当前APP的CPU占用百分比，非当前应用占据的CPU百分比，以及当前的线程总数。（Hint：我们应该把应用内由开发者创建的每一个线程赋予一个独特的名字，这样可以方便我们后续调试时，快速找到它。关于如何给线程命名，请参考这个类的写法：[TaskExecutors](#)）

三、记录、追踪CPU占用

多数情况下，我们仅仅只是观察CPU的占用率还是不够的，我们需要精确到某个占据多少CPU时间，这时候就需要进一步，追踪CPU的详细活动。

1.选择记录配置

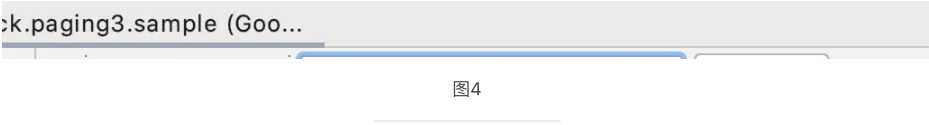


图4

图4中包含了4种不同的配置

- ①Sample Java Methods(采样Java方法)：在应用的 Java 代码执行期间，频繁捕获应用的调用堆栈。
分析器会比较捕获的数据集，以推导与应用的 Java 代码执行有关的时间和资源使用信息。
基于采样的跟踪存在一个固有的问题，那就是如果应用在捕获调用堆栈后进入一个方法且在下次捕获前退出该方法，则分析器不会记录该方法调用。如果想要跟踪生命周期如此短的方法，应使用检测跟踪（instrumented tracing）。
- ②Trace Java Methods(跟踪 Java 方法)：在运行时检测应用，以在每个方法调用开始和结束时记录一个时间戳。
系统会收集并比较这些时间戳，以生成方法跟踪数据，包括时间信息和 CPU 使用率。
请注意，与检测每个方法关联的开销会影响运行时性能，并且可能会影响分析数据；对于生命周期相对较短的方法，这一点更为明显。此外，如果应用在短时间内执行大量方法，则分析器可能很快就会超出其文件大小限制，因而不能再记录更多跟踪数据。
- ③Smample C/C++ Functions(对 C/C++ 函数采样)：捕获应用的原生线程的采样跟踪数据。
要使用此配置，必须将应用部署到搭载 Android 8.0（API 级别 26）或更高版本的设备上。
在内部，此配置使用 simpleperf 跟踪应用的原生代码。如果要为 simpleperf 指定其他选项，如对特定设备 CPU 采样指定高精度采样持续时间，您可以从命令行使用 simpleperf。
- ④Trace System Calls(跟踪系统调用)：捕获精细的详细信息，以便检查应用与系统资源的交互情况。
此跟踪配置在 systrace 的基础上构建而成。所以更建议使用Systrace，关于Systrace如何使用，后续再做介绍。

2.开始追踪

这里为了方便起见，我们选择Sample Java Methods，然后点击Record按钮，CPU Profiler会显示正在进行的记录的状态、持续时间和类型。我们实际的性能分析时，要同步的操作APP，这样才能得到最真实的数据。最后点击Stop按钮，得到该时间端内CPU的状态。

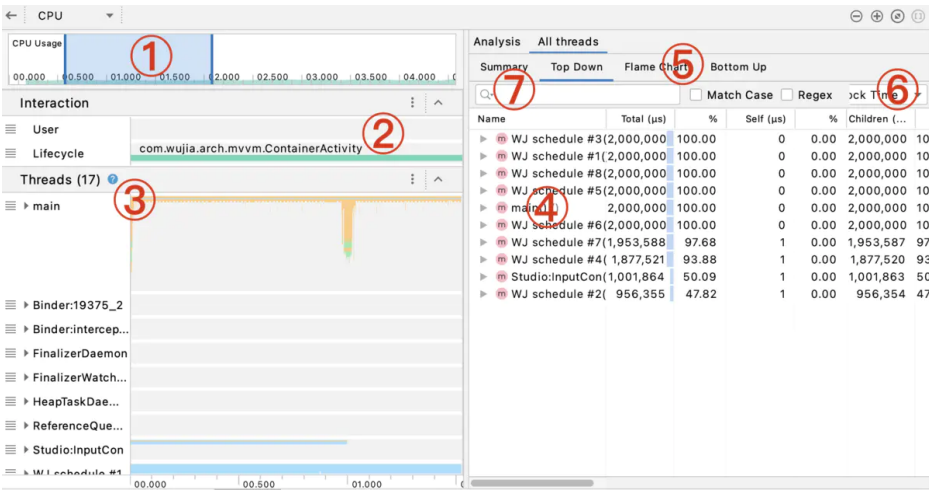


图5

- ①所选范围：确定在跟踪窗格中检查的记录时间部分。首次录制跟踪时，CPU 探查器会自动在 CPU 时间线中选择录制的整个长度。要仅检查跟踪数据记录的部分时间范围，请拖动高亮显示区域的边缘。
- ②交互时间轴：沿时间线显示用户交互和应用程序生命周期事件。(图5每一个粉红的圆点，表示一个事件)

⌚**线程时间轴**：显示沿时间线的每个线程的线程状态活动（如运行、睡眠等）和呼叫图（或在系统跟踪中跟踪事件图表）。

- 使用Ctrl+鼠标滚轮可放大/缩小所选时间范围
- 按住“空格键”左右移动鼠标，可向左/右移动时间轴
- 双击线程名称或按 Enter，而选择线程展开或折叠线程。
- 选择一个线程以查看🔍**分析窗格**中的其他信息。按住 Shift 可以选择多个线程。
- 选择方法调用以查看🔍**分析窗格**中的其他信息。

🔍**分析窗格**：显示所选时间范围和线程或方法调用的跟踪数据。在此窗格中，可以选择如何查看每个堆栈跟踪情况，以及如何测量执行时间。

🔍**分析窗格选项卡**：选择如何显示跟踪详细信息。有关每个选项的详细信息，后续再来详细介绍。

🕒**时间参考菜单**：选择以下选项之一以确定如何测量每个呼叫的计时信息（仅在示例/跟踪Java方法中支持）：

- **Wall clock time**：表示实际经过的时间。
- **Thread time**：表示实际经过的时间减去线程在该时间内没有消耗 CPU 资源的时间。它始终小于Wall clock time。Thread time可以更好地了解线程实际占用CPU的时间。

🔍**过滤器**：按函数、方法、类或软件包名称过滤跟踪数据。在 **Call chart** 和 **Flame chart** 标签中，会突出显示包含符合搜索查询条件的调用、软件包或类的调用堆栈。在 **Top down** 和 **Bottom up** 标签中，这些调用堆栈优先于其他跟踪结果。同时还可以通过勾选搜索字段旁边的相应框来启用以下选项：

Regex：若要在搜索中包含正则表达式，请使用此选项。

Match case：如果搜索对大小写敏感，请使用此选项

简要了解了CPU Profiler记录追踪数据之后窗口布局后，我们来详细了解一下🔍**分析窗格选项卡**里面包含了四个选项的含义

Summary(摘要)

我们在线程时间轴选择一个线程，点击Summary，看到如下界面

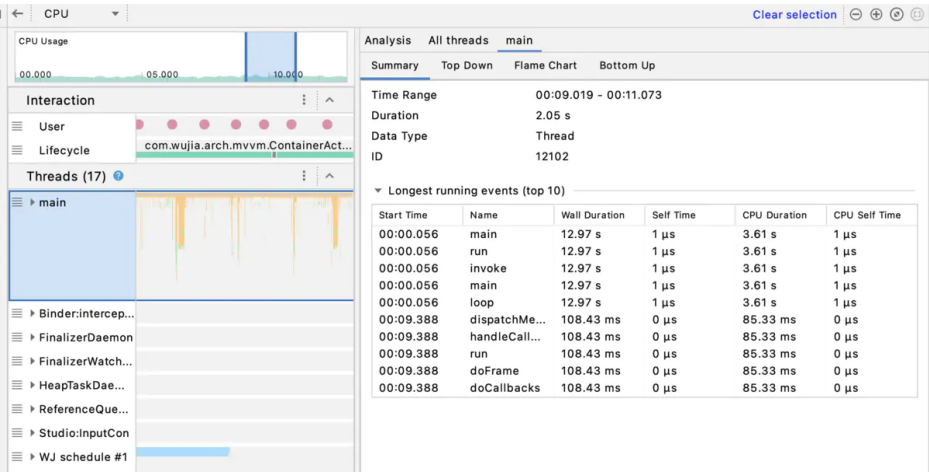


图6

Start Time：开启时间

Name：方法名

Wall Duration：代码执行时间

CPU Duration：代码消耗cpu的时间

Top Down

Top Down 标签显示一个调用列表，在该列表中展开方法或函数节点会显示它的被调用方

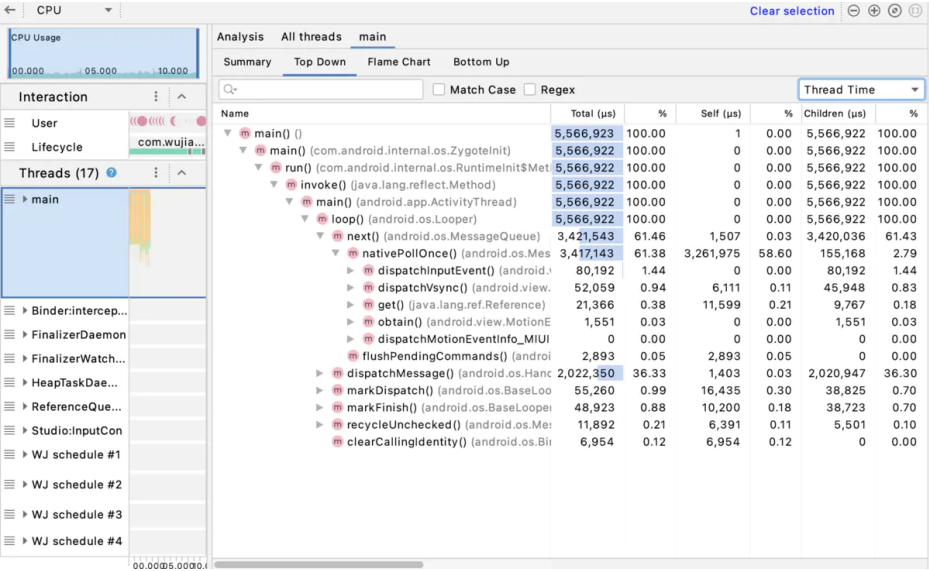


图7

Self：方法或函数调用在执行自己的代码（而非被调用方的代码）上所花的时间。
Children：方法或函数调用在执行它的被调用方（而非自己的代码）上所花的时间。
Total：方法的 Self 时间和 Children 时间的总和。这表示应用在执行调用上所花的总时间。
通过Top Down我们可以快速发现最耗时的方法，并逐步推导出它向下的调用链。

Bottom Up

Bottom Up标签显示一个调用列表，在该列表中展开函数或方法节点会显示它的调用方

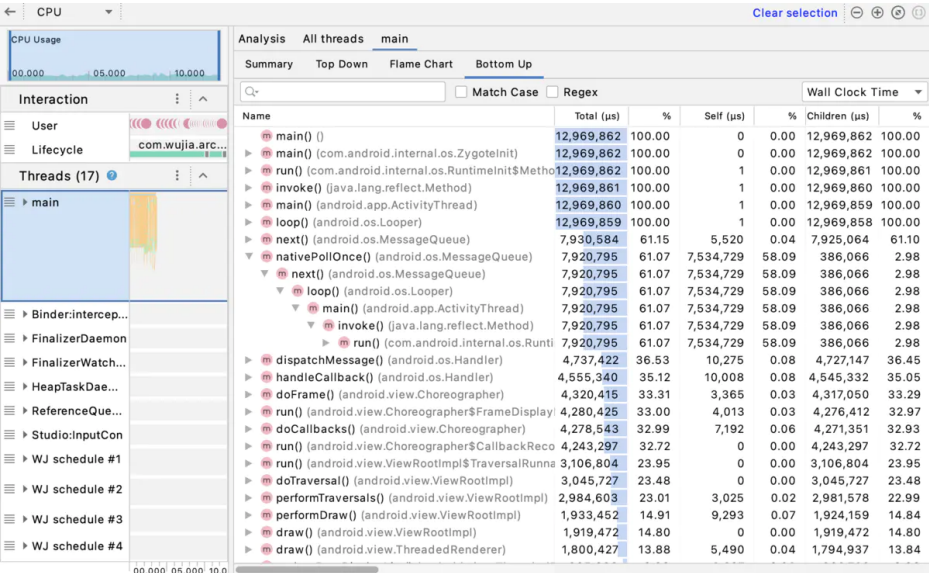


图8

例如上图中，展开nativePollOnce()方法，下方的next调用nativePollOnce，loop调用next，依次类推。
通过Bottom Up我们可以快速发现最耗时的方法，并逐步推导出它向上的调用链。

Flame chart

Flame Chart 标签提供一个倒置的调用图表，用来汇总完全相同的调用堆栈。

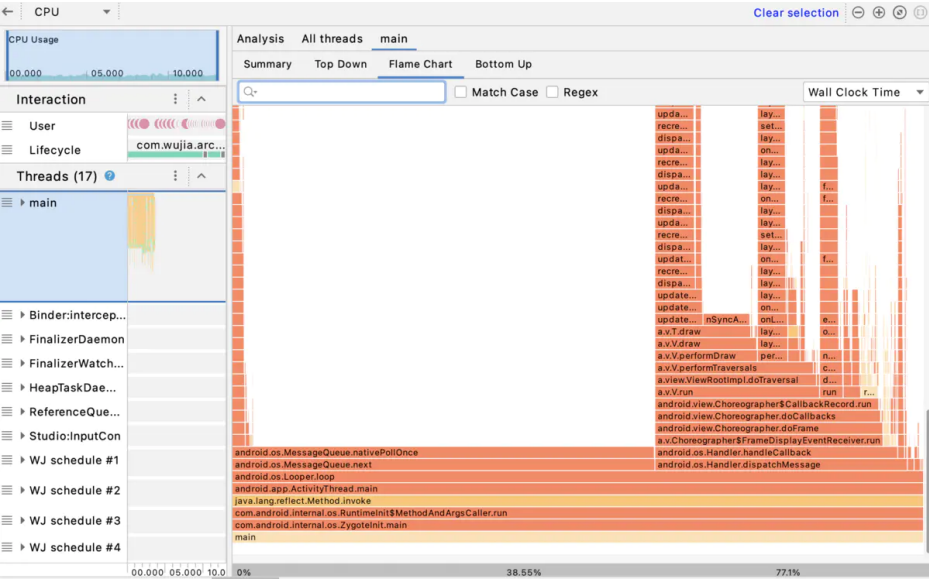


图9

在火焰图中从下往上表示方法的调用链，横条越长，表示该方法占用CPU的时间越长。通过火焰图的分叉可以更直观的看到一个方法同步调用了多少个子方法。

总结

本文讲解了Android Studio自带的CPU分析工具，CPU Profiler的基本使用方法。CPU Profiler是Android官方力荐使用的CPU性能分析工具，但是它缺点很明显，就是做性能分析的人必须有APP的源码，所以它只适合APP的开发人员使用。

下一篇我们来认识一个更为强大的性能分析工具Systrace，它既适合专门的测试、性能分析人员使用，也同样是开发人员的利器。

参考资料

Android官方资料[Identify CPU hot spots](#)

1人点赞 >

Android 性能优化