Simpleperf介绍

hanpfei (/u/1109fa43aaf6) + 关注
2017.02.19 21:20* 字数 4747 阅读 2310 评论 2 喜欢 5
(/u/1109fa43aaf6)

什么是simpleperf

Simpleperf是Android平台的一个本地层性能分析工具。它的命令行界面支持与linux-tools perf大致相同的选项，但是它还支持许多Android特有的改进。

Simpleperf是Android开源项目（AOSP）的一部分。其源代码 位于 (https://link.jianshu.com?t=https%3A%2F%2Fandroid.googlesource.com%2Fplatform%2Fsystem%2Fextras%2F%2B%2Fmaster%2Fsimpleperf%2F)。其最新的文档 位于 (https://link.jianshu.com?t=https%3A%2F%2Fandroid.googlesource.com%2Fplatform%2Fsystem%2Fextras%2F%2Bshow%2Fmaster%2Fsimpleperf%2FREADME.md)。Bugs 和 功能需求可以提交到 githb上 (https://link.jianshu.com?t=http%3A%2F%2Fgithub.com%2Fandroid-ndk%2Fndk%2Fissues)。

Simpleperf是如何工作的

现代的CPU具有一个硬件组件，称为性能监控单元(PMU)。PMU具有一些硬件计数器，计数一些诸如 经历了多少次CPU周期，执行了多少条指令，或发生了多少次缓存未命中等的事件。

Linux内核将这些硬件计数器包装到硬件perf事件 (hardware perf events)中。此外，Linux内核还提供了独立于硬件的软件事件和跟踪点事件。Linux内核通过perf_event_open 系统调用将这些都暴露给了用户空间，这正是simpleperf所使用的机制。

Simpleperf具有三个主要的功能：stat，record 和 report。

Stat命令给出了在一个时间段内被分析的进程中发生了多少事件的摘要。以下是它的工作原理：

1. 给定用户选项，simpleperf通过对linux内核进行系统调用来启用分析。
2. Linux 内核在调度到被分析进程时启用计数器。
3. 分析之后，simpleperf从内核读取计数器，并报告计数器摘要。

Record 命令在一段时间内记录剖析进程的样本。它的工作原理如下：

1. 给定用户选项，simpleperf通过对linux内核进行系统调用来启用分析。
2. Simpleperf 在simpleperf 和 linux 内核之间创建映射缓冲区。
3. Linux 内核在调度到被分析进程时启用计数器。
4. 每次给定数量的事件发生时，linux 内核将样本转储到映射缓冲区。
5. Simpleperf 从映射缓冲区读取样本并生成 perf.data。

Report 命令读取 “perf.data” 文件及所有被剖析进程用到的共享库，并输出一份报告，展示时间消耗在了哪里。

主 simpleperf 命令

Simpleperf 支持一些子命令，包括 list，stat，record，report。每个子命令支持不同的选项。这一节只描述最重要的子命令和选项。要了解所有的子命令和选项，请使用 --help。

```
# List all subcommands.
$simpleperf --help

# Print help message for record subcommand.
$simpleperf record --help
```

simpleperf list

simpleperf list 被用于列出设备上所有可用的事件。由于应该和内核的差异，不同的设备可以支持不同的事件。

```
$simpleperf list
List of hw-cache events:
  branch-loads
  ...
List of hardware events:
  cpu-cycles
  instructions
  ...
List of software events:
  cpu-clock
  task-clock
  . . .
```

simpleperf stat

simpleperf stat 被用于获取被剖析程序或系统范围内的原始事件计数器信息。通过传入选项，我们可以选择使用哪些事件，监视哪个进程/线程，监视多长时间，以及打印的间隔。下面是一个例子。

```
# Stat using default events (cpu-cycles,instructions,...), and monitor
# process 7394 for 10 seconds.
$simpleperf stat -p 7394 --duration 10
Performance counter statistics:

 1,320,496,145  cpu-cycles         # 0.131736 GHz                     (100%)
   510,426,028  instructions       # 2.587047 cycles per instruction  (100%)
     4,692,338  branch-misses      # 468.118 K/sec                    (100%)
886.008130(ms)  task-clock         # 0.088390 cpus used               (100%)
           753  context-switches   # 75.121 /sec                      (100%)
           870  page-faults        # 86.793 /sec                      (100%)

Total test time: 10.023829 seconds.
```

选择事件

我们可以通过 -e 选项选择使用哪个事件。下面是例子：

```
# Stat event cpu-cycles.
$simpleperf stat -e cpu-cycles -p 11904 --duration 10

# Stat event cache-references and cache-misses.
$simpleperf stat -e cache-references,cache-misses -p 11904 --duration 10
```

当运行 stat 命令时，如果硬件事件的数量大于 PMU中可用的硬件计数器的数量，则内核在事件间共享硬件计数器，因此每个事件只在总时间中的一部分内被监视。在下面的例子中，每一行的最后都有一个百分比，展示了每个事件实际被监视的时间占总时间的百分比。

```
# Stat using event cache-references, cache-references:u,....
$simpleperf stat -p 7394 -e     cache-references,cache-references:u,cache-references
Performance counter statistics:

4,331,018  cache-references     # 4.861 M/sec    (87%)
3,064,089  cache-references:u   # 3.439 M/sec    (87%)
1,364,959  cache-references:k   # 1.532 M/sec    (87%)
   91,721  cache-misses         # 102.918 K/sec  (87%)
   45,735  cache-misses:u       # 51.327 K/sec   (87%)
   38,447  cache-misses:k       # 43.131 K/sec   (87%)
9,688,515  instructions         # 10.561 M/sec   (89%)

Total test time: 1.026802 seconds.
```

在上面的例子中，每个事件被监视的时间大概占总时间的 87%。但是，不保证任何一对事件总是在相同的时间被监视。如果我们想要让一些事件在同一时间被监视，我们可以使用 --group 选项。下面是一个例子。

```
# Stat using event cache-references, cache-references:u,....
$simpleperf stat -p 7394 --group cache-references,cache-misses --group cache-referen
Performance counter statistics:

3,638,900  cache-references     # 4.786 M/sec          (74%)
   65,171  cache-misses         # 1.790953% miss rate  (74%)
2,390,433  cache-references:u   # 3.153 M/sec          (74%)
   32,280  cache-misses:u       # 1.350383% miss rate  (74%)
  879,035  cache-references:k   # 1.251 M/sec          (68%)
   30,303  cache-misses:k       # 3.447303% miss rate  (68%)
8,921,161  instructions         # 10.070 M/sec         (86%)

Total test time: 1.029843 seconds.
```

选择监视目标

我们可以通过 -p 选项或 -t 选项选择监视哪个进程或线程。监视一个进程如同监视进程中的所有线程。Simpleperf 也可以 fork 一个子进程来运行新命令，然后监视子进程。下面是例子。

```
# Stat process 11904 and 11905.
$simpleperf stat -p 11904,11905 --duration 10

# Stat thread 11904 and 11905.
$simpleperf stat -t 11904,11905 --duration 10

# Start a child process running `ls`, and stat it.
$simpleperf stat ls
```

决定监视多长时间

当监视已有线程时，我们可以使用 --duration 选项决定监视多长时间。当监视执行一个新命令的子进程时，simpleperf 将一直监视子进程直至其结束。在这种情况下，我们可以使用 Ctrl-C 在任何时间停止监视。例子如下。

```
# Stat process 11904 for 10 seconds.
$simpleperf stat -p 11904 --duration 10

# Stat until the child process running `ls` finishes.
$simpleperf stat ls

# Stop monitoring using Ctrl-C.
$simpleperf stat -p 11904 --duration 10
^C
```

决定打印的间隔

当监视 perf 计数器时，我们还可以使用 --interval 选项决定打印的间隔。例子如下。

```
# Print stat for process 11904 every 300ms.
$simpleperf stat -p 11904 --duration 10 --interval 300

# Print system wide stat at interval of 300ms for 10 seconds (rooted device only).
# system wide profiling needs root privilege
$su 0 simpleperf stat -a --duration 10 --interval 300
```

在 systrace 中显示计数器

simpleperf 还可以与systrace一起工作来将计数器转储进收集的trace中。下面是一个执行系统范围的 stat 的例子。

```
# capture instructions (kernel only) and cache misses with interval of 300 milliseco
$su 0 simpleperf stat -e instructions:k,cache-misses -a --interval 300 --duration 15
# on host launch systrace to collect trace for 10 seconds
(HOST)$external/chromium-trace/systrace.py --time=10 -o new.html sched gfx view
# open the collected new.html in browser and perf counters will be shown up
```

simpleperf record

simpleperf record用于转储被剖析程序的记录。通过传入选项，我们可以选择使用哪个事件，监视哪个进程/线程，以什么频率转储记录，监视多长时间，以及将记录存储到哪里。

```
# Record on process 7394 for 10 seconds, using default event (cpu-cycles),
# using default sample frequency (4000 samples per second), writing records
# to perf.data.
$simpleperf record -p 7394 --duration 10
simpleperf I 07-11 21:44:11 17522 17522 cmd_record.cpp:316] Samples recorded: 21430.
```

选择事件

在大多数情况下，cpu-cycles 事件被用于评估消耗的CPU时间。作为一个硬件事件，它精确而高效。我们还可以通过 -e 选项使用其它事件。下面是一个例子。

```
# Record using event instructions.
$simpleperf record -e instructions -p 11904 --duration 10
```

选择监视目标

record命令中选择目标的方式与 stat 命令中的类似。例子如下。

```
# Record process 11904 and 11905.
$simpleperf record -p 11904,11905 --duration 10

# Record thread 11904 and 11905.
$simpleperf record -t 11904,11905 --duration 10

# Record a child process running `ls`.
$simpleperf record ls
```

设置记录的频率

我们可以通过 -f 或 -c 选项设置转储记录的频率。比如，-f 4000 意味着当监视的线程运行时每秒转储接近 4000 个记录。如果监视的线程一秒钟运行了 0.2 s（其它时间它可能被抢占或阻塞），simpleperf 每秒转储大约 4000 * 0.2 / 1.0 = 800 个记录。另一种方式是使用 -c 选项。比如，-c 10000 意味着每发生 10000 次事件转储一个记录。例子如下。

```
# Record with sample frequency 1000: sample 1000 times every second running.
$simpleperf record -f 1000 -p 11904,11905 --duration 10

# Record with sample period 100000: sample 1 time every 100000 events.
$simpleperf record -c 100000 -t 11904,11905 --duration 10
```

决定监视多长时间

record 命令中决定监视多长时间的方式与 stat 命令中的类似。例子如下。

```
# Record process 11904 for 10 seconds.
$simpleperf record -p 11904 --duration 10

# Record until the child process running `ls` finishes.
$simpleperf record ls

# Stop monitoring using Ctrl-C.
$simpleperf record -p 11904 --duration 10
^C
```

设置存储记录的路径

默认情况下，simpleperf 将记录保存至当前文件夹下的 perf.data 文件中。我们可以使用 -o 选项设置存储记录的路径。下面是一个例子。

```
# Write records to data/perf2.data.
$simpleperf record -p 11904 -o data/perf2.data --duration 10
```

simpleperf report

simpleperf report 被用来基于 simpleperf record 命令生成的 perf.data 产生报告。Report 命令将记录分组为不同的样本项，基于每个样本项包含的事件的多少对样本项排序，并打印每个样本项。通过传入选项，我们可以选择到哪里寻找被监视的程序使用的 perf.data 和 可执行二进制文件，过滤不感兴趣的记录，并决定如何分组记录。

下面是一个例子。记录被分为 4 个样本项，每项一行。有一些列，每列展示了属于一个样本项的信息片段。第一列是 Overhead，它展示了当前样本项中的事件占总事件的百分比。由于 perf 事件是 cpu-cycles，overhead 可被视为是每个函数占用的 cpu 的百分比。

```
# Reports perf.data, using only records sampled in libsudo-game-jni.so,
# grouping records using thread name(comm), process id(pid), thread id(tid),
# function name(symbol), and showing sample count for each row.
$simpleperf report --dsos /data/app/com.example.sudogame-2/lib/arm64/libsudo-game-jn
Cmdline: /data/data/com.example.sudogame/simpleperf record -p 7394 --duration 10
Arch: arm64
Event: cpu-cycles (type 0, config 0)
Samples: 28235
Event count: 546356211

Overhead  Sample  Command    Pid   Tid   Symbol
59.25%    16680   sudogame  7394  7394  checkValid(Board const&, int, int)
20.42%    5620    sudogame  7394  7394  canFindSolution_r(Board&, int, int)
13.82%    4088    sudogame  7394  7394  randomBlock_r(Board&, int, int, int, int, in
6.24%     1756    sudogame  7394  7394  @plt
```

设置存储记录的路径

默认情况下，simpleperf 读取当前目录下的 perf.data。我们可以使用 -i 选项选择从另一个文件读取记录。

```
$simpleperf report -i data/perf2.data
```

设置查找可执行二进制文件的路径

如果要生成函数符号报告，simpleperf 需要读取被监视的进程使用的可执行二进制文件来获取符号表和调试信息。默认情况下，路径是记录时被监视的进程使用的可执行二进制文件，然而，在生成报告时这些二进制文件可能不存在，或不包含符号表和调试信息。因此我们可以使用 --symfs 来重定向路径。下面是一个例子。

```
$simpleperf report
# In this case, when simpleperf wants to read executable binary /A/b,
# it reads file in /A/b.

$simpleperf report --symfs /debug_dir
# In this case, when simpleperf wants to read executable binary /A/b,
# it prefers file in /debug_dir/A/b to file in /A/b.
```

过滤记录

当生成报告时，可能不是对所有记录都感兴趣。Simpleperf 支持五钟过滤器来选择感兴趣的记录。下面是例子。

```
# Report records in threads having name sudogame.
$simpleperf report --comms sudogame

# Report records in process 7394 or 7395
$simpleperf report --pids 7394,7395

# Report records in thread 7394 or 7395.
$simpleperf report --tids 7394,7395

# Report records in libsudo-game-jni.so.
$simpleperf report --dsos /data/app/com.example.sudogame-2/lib/arm64/libsudo-game-jn

# Report records in function checkValid or canFindSolution_r.
$simpleperf report --symbols "checkValid(Board const&, int, int);canFindSolution_r(B
```

决定如何将记录分组为样本项

Simpleperf 使用 --sort 选项决定如何分组样本项。下面是例子。

```
# Group records based on their process id: records having the same process
# id are in the same sample entry.
$simpleperf report --sort pid

# Group records based on their thread id and thread comm: records having
# the same thread id and thread name are in the same sample entry.
$simpleperf report --sort tid,comm

# Group records based on their binary and function: records in the same
# binary and function are in the same sample entry.
$simpleperf report --sort dso,symbol

# Default option: --sort comm,pid,tid,dso,symbol. Group records in the same
# thread, and belong to the same function in the same binary.
$simpleperf report
```

simpleperf 的特性

Simpleperf 的工作方式与 linux-tools-perf 类似，但它有如下的提升：

1. Aware of Android environment. Simpleperf 在剖析时处理一些Android特有的情形。例如，它可以剖析 apk 中嵌入的共享库，从 .gnu_debugdata 段读取符号表和调试信息。如果可能，当出现错误时它会给出一些提示，如如何禁用perf_harden启用分析。
2. 记录时支持展开。如果我们想使用 -g 选项来记录并报告一个程序的调用图，我们需要转储每个记录中的用户栈和寄存器集合，然后展开栈来查找调用链。Simpleperf 支持在记录时展开，因此它无需在 perf.data 中存储用户栈。因而我们可以在设备上有限的空间内剖析更长的时间。
3. 支持脚本使Android上的剖析更方便。
4. 编译进静态的二进制文件。Simpleperf 是一个静态库，因此它无需支持运行共享库。这意味着对于 simpleperf 可以运行的的 Android 版本没有限制，尽管有些设备不支持剖析。

ndk中的Simpleperf工具

ndk 中的 simpleperf 工具包含三个部分：在 Android 设备上运行的 simpleperf 可执行文件，在主机上运行的 simpleperf 可执行文件，和 python 脚本。

设备上的 simpleperf

在设备上运行的 simpleperf 位于 bin/android 目录下。它包含不同体系架构的 Android 上运行的静态二进制文件。它们可被用于剖析运行在设备上的进程，并生成 perf.data。

主机上的 simpleperf

运行与主机上的 Simpleperfs 位于bin/darwin，bin/linux 和 bin/windows。它们可被用于在主机上解析 perf.data。

脚本

脚本被用于使得剖析和解析剖析结果更方便。app_profiler.py 被用于剖析一个 android 应用程序。它准备剖析环境，下载 simpleperf 到设备上，在主机上生成并拉出 perf.data。它由 app_profiler.config 配置。binary_cache_builder.py 被用于从设备拉出本地层二进制文件到主机上。它由 app_profiler.py 使用。annotate.py 被用于使用 perf.data 注解源文件。它由 annotate.config 配置。report.py 在一个 GUI 窗口中报告 perf.data。simpleperf_report_lib.py 被用于枚举 perf.data 中的样本。在内部它使用 libsimpleperf_report.so 来解析 perf.data。它可被用于将 perf.data 中的样本翻译为其它形式。使用 simpleperf_report_lib.py 的一个例子是 report_sample.py。

使用 simpleperf 工具的例子

这个部分展示了如何使用 simpleperf 工具来剖析一个 Android 应用。

准备一个可调试(debuggable)的应用

应用程序的包名是 com.example.sudogame。它包含 java 代码和 c++ 代码。我们需要运行一份在其 AndroidManifest.xml 元素中 android:debuggable=”true” 的app，因为我们不能为 non-debuggable apps 使用 run-as。应用应该已经安装在设备上了，而且我们可以通过 adb 连接设备。

使用命令行剖析

为了记录剖析数据，我们需要下载 simpleperf 和包含调试信息的本地库到设备上，运行 simpleperf 产生剖析数据：perf.data，并运行 simpleperf 为 perf.data 生成报告。步骤如下。

1. Enable profiling

```
$adb shell setprop security.perf_harden 0
```

2. 寻找运行 app 的进程

在 app 的上下文运行 `ps`。在 >=O 的设备上，用 `ps -e` 来替代。

```
$adb shell
angler:/ $ run-as com.example.sudogame
angler:/data/data/com.example.sudogame $ ps
u0_a93    10324 570   1030480 58104 SyS_epoll_ 00f41b7528 S com.example.sudogame
u0_a93    10447 10441 7716   1588  sigsuspend 753c515d34 S sh
u0_a93    10453 10447 9112   1644           0 7ba07ff664 R ps
```

因此是进程 10324 运行app。

3. 将 simpleperf 下载到 app 的 data 目录

首先我们需要找出 app 使用的是哪个体系架构。有许多中方式，这里我们只检查进程的映射。

```
angler:/data/data/com.example.sudogame $cat /proc/10324/maps | grep boot.art
70f34000-7144e000 r--p 00000000 fd:00 1082  /system/framework/arm/boot.oat
```

路径显示是它是 arm。因此我们将 simpleperf 下载到设备上的 arm 目录下。

```
$adb push bin/android/arm/simpleperf /data/local/tmp
$adb shell
angler:/ $ run-as com.example.sudogame
angler:/data/data/com.example.sudogame
$ cp /data/local/tmp/simpleperf .
```

4. 记录 perf.data

```
angler:/data/data/com.example.sudogame $./simpleperf record -p 10324 --duration 30
simpleperf I 01-01 09:26:39 10598 10598 cmd_record.cpp:341] Samples recorded: 49471.
angler:/data/data/com.example.sudogame $ls -lh perf.data
-rw-rw-rw- 1 u0_a93 u0_a93 2.6M 2017-01-01 09:26 perf.data
```

在记录时不要忘记运行 app。否则，我们可能无法获得样本，由于进程仍在休眠。

5. 为 perf.data 生成报告

有不同的方式来为 perf.data 生成报告。下面展示了一些例子。

报告不同线程中的样本。

```
angler:/data/data/com.example.sudogame $./simpleperf report --sort pid,tid,comm
Cmdline: /data/data/com.example.sudogame/simpleperf record -p 10324 --duration 30
Arch: arm64
Event: cpu-cycles (type 0, config 0)
Samples: 49471
Event count: 16700769019

Overhead  Pid    Tid    Command
66.31%    10324  10324  xample.sudogame
30.97%    10324  10340  RenderThread
. . .
```

报告主线程中不同二进制文件中的样本。

```
angler:/data/data/com.example.sudogame $./simpleperf report --tids 10324 --sort dso
...
Overhead  Sample  Shared Object
37.71%    9970    /system/lib/libc.so
35.45%    9786    [kernel.kallsyms]
8.71%     3305    /system/lib/libart.so
6.44%     2405    /system/framework/arm/boot-framework.oat
5.64%     1480    /system/lib/libcutils.so
1.55%     426     /data/app/com.example.sudogame-1/lib/arm/libsudo-game-jni.so
...
```

报告主线程中 libsudo-game-jni.so 中的不同函数的采样。

```
angler:/data/data/com.example.sudogame $./simpleperf report --tids 10324 --dsos  /da
...
Overhead  Sample  Symbol
8.94%     35      libsudo-game-jni.so[+1d54]
5.71%     25      libsudo-game-jni.so[+1dae]
5.70%     23      @plt
5.09%     22      libsudo-game-jni.so[+1d88]
4.54%     19      libsudo-game-jni.so[+1d82]
3.61%     14      libsudo-game-jni.so[+1f3c]
...
```

在上面的结果中，大多数符号是 二进制文件名[+virual_addr] 的形式呈现。那是由于设备上使用的 libsudo-game-jni.so 已经抛离了 .symbol 段。我们可以将带有调试信息的 libsudo-game-jni.so 下载到设备上。在 android studio工程中，它位于 app/build/intermediates/binaries/debug/arm/obj/armeabi-v7a/libsudo-game-jni.so。我们不得不下载 libsudo-game-jni.so 到与 perf.data 中记录的相同的相对路径（否则，simpleperf无法找到它）。在这个例子中，是/data/app/com.example.sudogame-1/lib/arm/libsudo-game-jni.so。

为包含了调试信息的库使用的符号生成报告。

```
$adb push app/build/intermediates/binaries/debug/arm/obj/armeabi-v7a/libsudo-game-jn
$adb shell
angler:/ $ run-as com.example.sudogame
angler:/data/data/com.example.sudogame $ mkdir -p data/app/com.example.sudogame-1/li
angler:/data/data/com.example.sudogame $cp /data/local/tmp/libsudo-game-jni.so data/
angler:/data/data/com.example.sudogame $./simpleperf report --tids 10324 --dsos  /da
. . .
Overhead  Sample  Symbol
75.18%    317     checkValid(Board const&, int, int)
14.43%    60      canFindSolution_r(Board&, int, int)
5.70%     23      @plt
3.51%     20      randomBlock_r(Board&, int, int, int, int, int)
. . .
```

报告一个函数中的采样。

```
angler:/data/data/com.example.sudogame $./simpleperf report --tids 10324 --dsos  /da
. . .
Overhead  Sample  VaddrInFile
11.89%    35      0x1d54
7.59%     25      0x1dae
6.77%     22      0x1d88
6.03%     19      0x1d82
. . .
```

6. 记录并报告调用图

调用图是显示了函数调用关系的树。下面是一个例子。

```
main() {
    FunctionOne();
    FunctionTwo();
}
FunctionOne() {
    FunctionTwo();
    FunctionThree();
}
callgraph:
    main-> FunctionOne
       |    |
       |    |-> FunctionTwo
       |    |-> FunctionThree
       |
       |-> FunctionTwo
```

基于调用图记录 dwarf

为了生成调用图，simpleperf 需要为每个记录生成调用链。Simpleperf 需要内核为每个记录转储用户栈和用户寄存器集，然后它追踪用户栈来查找函数调用链。为了解析调用链，它需要 dwarf 调用帧信息的支持，这些通常位于二进制文件的 .eh_frame 或 .debug_frame段。因此我们需要使用 --symfs 指出带有调试信息的 libsudo-game-jni.so 位于哪里。

```
data/data/com.example.sudogame
eperf record -p 10324 -g --symfs . --duration 30
f I 01-01 09:59:42 11021 11021 cmd_record.cpp:341] Samples recorded: 60700. Samples lo
```

注意内核无法转储 >= 64K 的用户栈，因此基于调用图的 dwarf 不要包含消耗了 >= 64K 栈的调用链。此外，由于我们需要转储每个记录的栈，则可能丢失一些记录。通常，失去一些记录没关系。

基于调用图记录栈帧

另外一种生成调用图的方式依赖内核为每个记录解析调用链。为了使它成为可能，内核需要能够识别每个函数调用的栈帧。这不总是可能的，因为编译器可能优化掉栈帧，或内核无法识别使用的栈帧风格。因此它如何工作视情况而定（它在 arm64 上工作的很好，但在 arm 上不行）。

```
angler:/data/data/com.example.sudogame
$./simpleperf record -p 10324 --call-graph fp --symfs . --duration 30
simpleperf I 01-01 10:03:58 11267 11267 cmd_record.cpp:341] Samples recorded: 56736.
```

报告调用图

报告累积的周期。在下面的表中，第一列是 “Children”，它是函数及那个函数调用的函数所占的cpu 周期百分比。第二列是 “Self”，它只是一个函数的cpu 周期百分比。比如，checkValid() 自身消耗 1.28% 的 cpus，但通过运行它自身及调用其它函数，它消耗了 29.43%。

```
angler:/data/data/com.example.sudogame $./simpleperf report --children --symfs .
. . .
Children  Self   Command          Pid    Tid    Shared Object
31.94%    0.00%  xample.sudogame  10324  10324  [kernel.kallsyms]
31.10%    0.92%  xample.sudogame  10324  10324  /system/lib/libc.so
29.43%    1.28%  xample.sudogame  10324  10324  /data/app/com.example.sudogame-1/lib
28.43%    0.34%  xample.sudogame  10324  10324  /system/lib/liblog.so
28.24%    0.00%  xample.sudogame  10324  10324  /system/lib/libcutils.so
28.10%    0.27%  xample.sudogame  10324  10324  /data/app/com.example.sudogame-1/lib
. . .
```

报告调用图。

```
angler:/data/data/com.example.sudogame $./simpleperf report -g --symfs . >report
angler:/data/data/com.example.sudogame $exit
angler:/ $cp /data/data/com.example.sudogame/report /data/local/tmp
angler:/ $exit
$adb pull /data/local/tmp/report .
$cat report
. . .
29.43%    1.28%  xample.sudogame  10324  10324  /data/app/com.example.sudogame-1/lib
       |
       -- checkValid(Board const&, int, int)
          |
          |--95.50%-- __android_log_print
          |    |--0.68%-- [hit in function]
          |    |
          |    |--51.84%-- __android_log_buf_write
          |    |    |--2.07%-- [hit in function]
          |    |    |
          |    |    |--30.74%-- libcutils.so[+c69d]
. . .
```

以 callee 模式报告调用图。我们还可以展示一个函数是如何被其它函数调用的。

```
angler:/data/data/com.example.sudogame $./simpleperf report -g callee --symfs . >rep
$adb shell run-as com.example.sudogame cat report >report
$cat report
. . .
28.43%    0.34%  xample.sudogame  10324  10324  /system/lib/liblog.so
       |
       -- __android_log_print
          |
          |--97.82%-- checkValid(Board const&, int, int)
          |    |--0.13%-- [hit in function]
          |    |
          |    |--94.89%-- canFindSolution_r(Board&, int, int)
          |    |    |--0.01%-- [hit in function]
          |    |    |
. . .
```

剖析 Java 代码

Simpleperf 只支持剖析 ELF 格式二进制文件中的本地指令。如果 java 代码由解释器执行，或使用 jit 缓存，则它不能由 simpleperf 剖析。由于 Android 支持提前编译，它可以将 java 字节码编译为包含调试信息的本地层指令。在 Android 版本 <= M 的设备上，我们需要 root 权限来编译包含调试信息的 java 字节码。然而，在 Android 版本 >= N 的设备上，我们无需 root 权限就可以做这些。

在Android N上

1. 将 java 代码完整编译为本地层指令。

```
$adb shell setprop debug.generate-debug-info true
$adb shell cmd package compile -f -m speed com.example.sudogame
// restart the app to take effect
```

2. 记录 perf.data

```
angler:/data/data/com.example.sudogame
$./simpleperf record -p 11826 -g --symfs . --duration 30
simpleperf I 01-01 10:31:40 11859 11859 cmd_record.cpp:341] Samples recorded: 50576.
```

3. 为 perf.data 生成报告

```
angler:/data/data/com.example.sudogame $./simpleperf report -g --symfs . >report
angler:/data/data/com.example.sudogame $exit
angler:/ $cp /data/data/com.example.sudogame/report /data/local/tmp
angler:/ $exit
$adb pull /data/local/tmp/report .
$cat report
. . .
21.14%    0.00%  xample.sudogame  11826  11826  /data/app/com.example.sudogame-1/oat
       |
       -- boolean com.example.sudogame.MainActivity.onOptionsItemSelected(android.vi
          |
           --99.99%-- void com.example.sudogame.GameView.startNewGame()
               |--0.01%-- [hit in function]
               |
               |--99.87%-- void com.example.sudogame.GameModel.reInit()
               |    |--0.01%-- [hit in function]
               |    |
               |    |--89.65%-- boolean com.example.sudogame.GameModel.canFindSoluti
               |    |    |
               |    |    |--99.95%-- Java_com_example_sudogame_GameModel_canFindSolu
               |    |    |    |
               |    |    |    |--99.49%-- canFindSolution(Board&)
               |    |    |    |    |--0.01%-- [hit in function]
               |    |    |    |    |
               |    |    |    |    |--99.97%-- canFindSolution_r(Board&, int, int)
               |    |    |    |    |           canFindSolution_r(Board&, int, int)
. . .
```

在 Android M上

在 M 设备上，我们需要 root 权限来强制 Android 将 java 代码完全编译为带调试信息的 ELF 二进制文件中的本地层指令。我们还需要 root 权限来读取编译后的本地层二进制文件（由于 installd 将它们写到了一个 uid/gid 是 system:install的目录下）。因而剖析 java 代码只能在 root 了的设备上完成。

```
$adb root
$adb shell setprop dalvik.vm.dex2oat-flags -g

# Reinstall the app.
$adb install -r app-debug.apk

# Change to the app’s data directory.
$ adb root && adb shell
device# cd `run-as com.example.sudogame pwd`

# Record as root as simpleperf needs to read the generated native binary.
device#./simpleperf record -p 25636 -g --symfs . -f 1000 --duration 30
simpleperf I 01-02 07:18:20 27182 27182 cmd_record.cpp:323] Samples recorded: 23552.
```

在 Android L上

在 L 设备上，我们也需要 root 权限来编译带调试信息的 app 并访问本地层二进制文件。

```
$adb root
$adb shell setprop dalvik.vm.dex2oat-flags --include-debug-symbols

# Reinstall the app.
$adb install -r app-debug.apk
```

使用脚本剖析

尽管使用命令行很灵活，但它可能太复杂了。因而我们提供了 pthon 脚本来帮助运行命令。

使用 app_profiler.py 记录

app_profiler.py 用于剖析 Android 应用程序。它设置剖析环境，下载 simpleperf 和带有调试信息的本地层库，运行 simpleperf 产生 perf.data，并从设备上将 perf.data 和二进制文件拉到主机上。它由 app_profiler.config 配置。下面是一个例子。

app_profiler.config:

```
app_package_name = “com.example.sudogame”
android_studio_project_dir = “/AndroidStudioProjects/SudoGame”  # absolute path of t
. . .
record_options = "-e cpu-cycles:u -f 4000 -g --dump-symbols --duration 30"
. . .
```

运行 app_profiler.py:

```
$python app_profiler.py
...
INFO:root:profiling is finished.
```

它将生成的 perf.data 拉到主机上，并从设备的 binary_cache 中收集二进制文件。

使用 report.py 生成报告

```
$python report.py -g
```

它生成一个GUI接口来报告数据。

使用 simpleperf_report_lib.py 处理样本

simpleperf_report_lib.py 提供了一个接口从 perf.data 读取样本。一个例子是 report_sample.py。

展示流程图

```
$python report_sample.py >out.perf
$stackcollapse-perf.pl out.perf >out.folded
$./flamegraph.pl out.folded >a.svg
```

注解源代码

annotate.py 读取 perf.data 和 binary_cache 下的二进制文件。然后它知道每个样本命中哪个 源文件:line。因此它可以注解源代码。annotate.py 由 annotate.config 配置。下面是一个例子。

annotate.config:

```
. . .
source_dirs = [“/AndroidStudio/SudoGame”]  # It is a directory containing source cod
. . .
```

运行 annotate.py:

```
$python annotate.py
```

它生成 annotated_files 目录。 annotated_files/summary 文件包含每个源文件的概要信息。例子如下。

```
/AndroidStudioProjects/SudoGame/app/src/main/jni/sudo-game-jni.cpp: accumulated_peri
  function (checkValid(Board const&, int, int)): line 99, accumulated_period: 23.564
  function (canFindSolution_r(Board&, int, int)): line 135, accumulated_period: 22.2
  function (canFindSolution(Board&)): line 166, accumulated_period: 22.233101%, peri
  function (Java_com_example_sudogame_GameModel_canFindSolution): line 470, accumula
  function (Java_com_example_sudogame_GameModel_initRandomBoard): line 430, accumula

  line 27: accumulated_period: 0.011729%, period: 0.000000%
  line 32: accumulated_period: 0.004362%, period: 0.000000%
  line 33: accumulated_period: 0.004427%, period: 0.000000%
  line 36: accumulated_period: 0.003303%, period: 0.000000%
  line 39: accumulated_period: 0.010367%, period: 0.004123%
  line 41: accumulated_period: 0.162219%, period: 0.000000%
```

annotated_files/ 还包含由 annotate.py 找到的经过注解的源文件。比如， libsudo-game-jni.cpp 中的 checkValid() 函数的一部分注解后如下。

```
/* [func] acc_p: 23.564356%, p: 0.908457% */static bool checkValid(const Board& boar
/* acc_p: 0.037933%, p: 0.037933%         */    int digit = board.digits[curR][curC]
/* acc_p: 0.162355%, p: 0.162355%         */    for (int r = 0; r < BOARD_ROWS; ++r)
/* acc_p: 0.020880%, p: 0.020880%         */        if (r == curR) {
/* acc_p: 0.034691%, p: 0.034691%         */            continue;
                                                   }
/* acc_p: 0.176490%, p: 0.176490%         */        if (board.digits[r][curC] == dig
/* acc_p: 14.957673%, p: 0.059022%        */            LOGI("conflict (%d, %d) (%d,
/* acc_p: 0.016296%, p: 0.016296%         */            return false;
                                                   }
                                               }
```

原文地址 (https://link.jianshu.com?t=https%3A%2F%2Fandroid.googlesource.com%2Fplatform%2Fprebuilts%2Fsimpleperf%2F%2B%2Fndk-r13%2FREADME.md)

参考文档：

- Simpleperf (https://link.jianshu.com?t=https%3A%2F%2Fandroid.googlesource.com%2Fplatform%2Fsystem%2Fextras%2F%2B%2Fmaster%2Fsimpleperf%2Fdoc%2FREADME.md)

小礼物走一走，来简书关注我

赞赏支持

Android 开发 (/nb/16446924)

hanpfei (/u/1109fa43aaf6)

写了 303738 字，被 242 人关注，获得了 382 个喜欢

(/u/1109fa43aaf6)

+关注

https://www.wolfcstech.com/

喜欢 | 5

更多分享

(http://cwb.assets.jianshu.io/notes/images/9263720/weibo/image_92a81a7e1

下载简书 App ▸

随时随地发现和创作内容

(/apps/download?utm_source=nbc)

被以下专题收入，发现更多相似内容

Android知识 (/c/3fde3b545a35?utm_source=desktop&utm_medium=notes-included-collection)

Android开发 (/c/d1591c322c89?utm_source=desktop&utm_medium=notes-included-collection)

Android... (/c/58b4c20abf2f?utm_source=desktop&utm_medium=notes-included-collection)

技术干货 (/c/38d96caffb2f?utm_source=desktop&utm_medium=notes-included-collection)

推荐阅读 更多精彩内容 ›(/)

网络优化实践探索文章 (/p/34242545c48c?utm_campaign=maleskine&ut…

携程App的网络性能优化实践 2016年携程App网络服务通道治理和性能优化实践 蘑菇街App Chromium网络栈实践 手机淘宝性能优化 无线性能优化：域名收敛 Facebook App对TLS的魔改造：实现0-RTT 腾讯HTT…

hanpfei (/u/1109fa43aaf6?utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

EGLSurfaces 和 OpenGL ES (/p/5bb1da34af25?utm_campaign=malesk…

OpenGL ES 定义了一个渲染图形的 API。它没有定义窗口系统。为了使 GLES 可以工作于各种平台之上，它被设计为与知道如何通过操作系统创建和访问窗口的库相结合。Android 使用的库称为 EGL。如果你想…

hanpfei (/u/1109fa43aaf6?utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

20岁的我，一点都不想虚度时光 (/p/ee4f3aba24ca?ut…

(/p/ee4f3aba24ca?utm_campaign=maleskine&utm_content=note&utm_medium=pc_all_hots&ut

文/梅梅 01 前几天和久别重逢的高中室友去看电影，《无问西东》。 看到清华一代代青春洋溢的年轻人，心里却总想着我那个平平常常的母亲。 她20岁的…

叫我梅梅呀 (/u/9205dd0ccddf?utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

《前任3》：没看到爱情，只看到了少女的油腻 (/p/b1c…

(/p/b1c162d31f71?utm_campaign=maleskine&utm_content=note&utm_medium=pc_all_hots&ut

趁着元旦放假，跑到电影院遛了一圈，怀着想看喜剧的心情，选了《前任3》，结果在笑声不断的影院，我被剧情震惊到，陷入深思。 故事很简单，总结起…

Miss余点 (/u/f351af6d6c85?utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

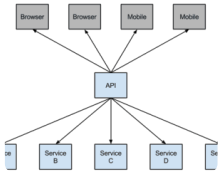
这些不为人知却超级好用的神器，推荐给你！ (/p/0586…

(/p/058671d37f4d?utm_campaign=maleskine&utm_content=note&utm_medium=pc_all_hots&ut

✍ 大家好，我是雅客。 由于平时要给大家写文章，录课程，所以平时会搜集一些非常好用的软件或者工具，来提高工作的效率和质量。 今天，我就来跟大…

雅客先生 (/u/e21f544cd102?utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

(/p/46fd0faecac1?



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

Spring Cloud (/p/46fd0faecac1?utm_campaign=maleskine&utm_conte…

Spring Cloud为开发人员提供了快速构建分布式系统中一些常见模式的工具（例如配置管理，服务发现，断路器，智能路由，微代理，控制总线）。分布式系统的协调导致了样板模式, 使用Spring Cloud开发人员可…

卡卡罗2017 (/u/d90908cb0d85?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

1、红帽性能调优——perf使用 (/p/675a850365eb?utm_campaign=males…

性能调优工具如 perf，Oprofile 等的基本原理都是对被监测对象进行采样，最简单的情形是根据 tick 中断进行采样，即在 tick 中断内触发采样点，在采样点里判断程序当时的上下文。 事件分为以下三种： 1）…

闪亮的蛤蟆 (/u/abeb6ca9bb86?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

百战程序员V1.2——尚学堂旗下高端培训_ Java1573题 (/p/49ad52bd5405…

百战程序员_ Java1573题 QQ群：561832648489034603 掌握80%年薪20万掌握50%年薪10万 全程项目穿插, 从易到难，含17个项目视频和资料持续更新，请关注www.itbaizhan.com 国内最牛七星级团队马士兵、…

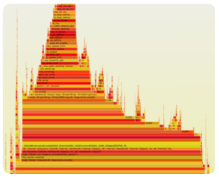
Albert陈凯 (/u/185a3c553fc6?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

面向对象的用电信息数据交换协议 (/p/94caedb70f65?utm_campaign=mal…

国家电网公司企业标准（Q/GDW）- 面向对象的用电信息数据交换协议 - 报批稿：20170802 前言： 排版 by Dr_Ting公众号：庭说移步 tingtalk.me 获得更友好的阅读体验 Q/GDW XXXX-201X《面向对象的用电信息…

庭说 (/u/a0d04c114c89?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/c62dac29c8d4?



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

了解你自己的程序 (/p/c62dac29c8d4?utm_campaign=maleskine&utm_c…

前言 因为在做nodejs程序的性能分析的时候，了解到了Perf和FlameGraph这两个神奇的工具，接着就知道了Brendan D. Gregg这个大神，跪着拜读了他的博客和他写的System Performance。从前写程序和调优只知…

泡沫与周期_白羊Jerry (/u/f5cf613b44a4?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

无题 (/p/03021d8e401c?utm_campaign=maleskine&utm_content=note…

无意中看到这个软件，心里说不出什么感觉，不觉中竟下载了。我好久没有写东西了，满案皆笔纸却荒芜在心里。我不知道是否还能写出东西。好多个问号在血液里跳动。些许跃跃欲试被担心按在了心里。


诉至荒凉 (/u/100a7953c3f9?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

随想 (/p/e58b34b0246f?utm_campaign=maleskine&utm_content=note…


荷花_a57a (/u/8f95046b1f0f?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/475acf2d7fb5?

utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

双溪之行（二）——画画前的激烈内心戏 (/p/475acf2d7fb5?utm_campaig…

林老师说，那画吧。 然后，我去登记领材料。 在此之前，从未接触过油画，面对颜料桌上的颜料，我不知道该如何选择颜色。我甚至不知道原来油画还有专用的稀释剂而不是用水（这是现场的黄老师后来告诉我的…


田螺姑娘FZ (/u/2312e441a4d3?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/ec4a0826bc36?

utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

今年上岗证的改革，低学历者不能再从事会计工作了吗? (/p/ec4a0826bc36…

会计证暂停之后，很多人只能眼巴巴的守着一个不能考的证，盼着它能重新恢复考试。这么空等绝对不是好办法，应该做点什么才行。 今天根据我看到的不同身份的人群，高顿小编给出点建议。 1、在校大学生 这…


高顿财经 (/u/03776fba7930?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

离开 (/p/c56e6423bd85?utm_campaign=maleskine&utm_content=note…

我想离开这里 虽然没有那么糟糕 但是 好想离开 离开，逃开 好沉重，好寂寞 在这片蓝天下 默默的寻找着 那可以放松的地方 那可以逃避到地方 不再


lavende陌酱 (/u/63435d9e9c65?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)