

Traceview 사용법

Trace File은 traceview와 dmtracedump 2가지를 이용하여 분석을 할 수 있습니다. traceview는 timeline과 profile을 이용하여 성능체크 위주로 사용할 수 있고, dmtracedump는 call-stack diagrams으로 보여주지만 Graphviz 라는 유틸을 설치하여야 합니다.

우선 Trace File을 만들기 위해 해당 어플리케이션에서 해야 할 일을 설명하고 traceview의 구성과 file format에 대해 알아본 뒤에 dmtracedump의 사용법을 알아 보겠습니다.

이 문서는 <http://developer.android.com/guide/developing/tools/traceview.html> 의 내용을 바탕으로 만들어 졌습니다.

Creating Trace Files

To create the trace files, include the Debug class.

애플리케이션에 디버그 패키지를 import한다.

```
import android.os.Debug;
```

start tracing point

트레이스 정보 수집을 시작하려는 지점에서 Debug.startMethodTracing()을 호출한다.

```
// 이 메소드는 항상 트레이스 정보를 타겟 시스템의 sd카드에 저장한다.
```

```
Debug.startMethodTracing("test"); // "sdcard/test.trace" file
```

stop tracing point

수집을 멈추려는 지점에서 Debug.stopMethodTracing()을 호출한다.

```
Debug.stopMethodTracing();
```

For example,

```
....
import android.os.Debug;
....

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ....
    Debug.startMethodTracing("test");
    ....
}

....
@Override
protected void onDestroy() {
    super.onDestroy();
    Debug.stopMethodTracing();
    ....
}

....
```

Copying Trace Files to a Host Machine

```
adb pull /sdcard/test.trace /tmp
```

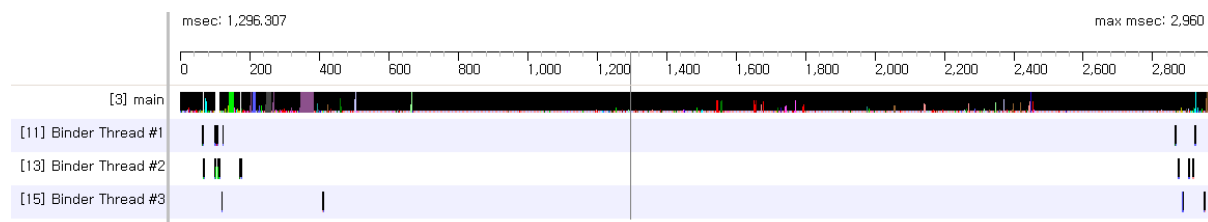
Viewing Trace Files in Traceview

```
traceview /tmp/test
```

Traceview loads the log files and displays their data in a window that has two panels:

- A timeline panel -- describes when each thread and method started and stopped.
- A profile panel -- provides a summary of what happened inside a method

Timeline panel



Profile Panel

The profile panel shows a summary of all the time spent in a method.

Name	Incl %	Inclusive	Excl %	Exclusive	Calls+Recur Calls/Total	Time/Call
0 (toplevel)	100.0%	2960.072	2.1%	61.510	4+0	740.018
1 android.os.Handler.dispatchMessage (Landroid/os/Handler;Ljava/lang/Object;)V	77.7%	2301.021	1.1%	31.186	569+0	4.044
2 com.digitalaria.fxui.FxuiActivity\$FxuiView\$CoreProc.run ()V	63.4%	1877.441	5.1%	151.671	550+0	3.414
3 android.os.Handler.handleCallback (Landroid/os/Handler;Ljava/lang/Runnable;)V	63.4%	1876.156	1.1%	31.346	550+0	3.411
4 com.digitalaria.fxui.FxuiActivity\$FxuiView\$1.run ()V	62.3%	1844.810	1.1%	32.532	550+0	3.354
5 com.digitalaria.fxui.FxuiActivity\$FxuiView\$CoreDrew.run ()V	61.2%	1812.278	1.5%	45.242	550+0	3.295
6 com.digitalaria.fxui.FxPlayer.process ()Z	27.1%	802.878	2.7%	79.489	550+0	1.460
7 com.digitalaria.fxui.FxPlayer.getNextProcessInterval ()I	18.9%	558.864	2.4%	71.520	550+0	1.016
8 java.util.concurrent.locks.ReentrantLock.unlock ()V	17.9%	528.720	2.2%	64.341	1106+0	0.478
9 java.util.concurrent.locks.AbstractQueuedSynchronizer.doRelease ()V	15.7%	464.379	2.1%	62.712	1106+0	0.420
10 android.os.MessageQueue.next ()Landroid/os/Message;	15.5%	459.166	6.0%	178.146	569+0	0.807
11 java.util.concurrent.locks.ReentrantLock.lock ()V	14.3%	421.978	2.0%	60.277	1106+0	0.382
12 java.util.concurrent.locks.ReentrantLock\$Sync.tryLock ()Z	13.6%	401.667	6.5%	193.050	1106+0	0.363
13 android.view.ViewRoot.handleMessage (Landroid/os/Message;)V	12.4%	367.221	0.0%	1.344	10+0	36.722
14 java.util.concurrent.locks.ReentrantLock\$NonfairSync.lock ()V	12.2%	361.701	4.5%	133.751	1106+0	0.327
15 android.view.ViewRoot.performTraversals ()V	11.3%	334.140	0.1%	3.323	3+0	111.380
16 android.view.View.postDelayed (Ljava/lang/Runnable;J)V	10.5%	309.855	1.1%	32.399	550+0	0.563
17 java.util.concurrent.locks.ReentrantLock\$Sync.doLock ()V	9.8%	293.456	1.5%	45.147	550+0	0.534

Traceview File Format

Data File Format

The data file is binary, structured as follows (all values are stored in little-endian order):

- * File format:
- * header
- * record 0
- * record 1
- * ...

- *
 - * Header format:
 - * u4 magic 0x574f4c53 ('SLOW')
 - * u2 version
 - * u2 offset to data
 - * u8 start data/time in usec
 - *
 - * Record format:
 - * u1 thread ID
 - * u4 method ID | method action
 - * u4 time delta since start, in usec

The application is expected to parse all of the header fields, then seek to "offset to data" from the start of the file. From there it just reads 9-byte records until EOF is reached. u8 start data/time in usec is the output from `gettimeofday()`. It's mainly there so that you can tell if the output was generated yesterday or three months ago. method action sits in the two least-significant bits of the method word. The currently defined meanings are:

- 0 - method entry
- 1 - method exit
- 2 - method "exited" when unrolled by exception handling
- 3 - (reserved)

An unsigned 32-bit integer can hold about 70minutes of time in microseconds.

Key File Format

The key file is a plain text file divided into three sections. Each section starts with a keyword that begins with '*'. If you see a '*' at the start of a line, you have found the start of a new section.

An example file might look like this:

```
*version
1
data-file-overflow=false
clock=thread-cpu
elapsed-time-usec=10235287
num-method-calls=84009
clock-call-overhead-nsec=21886
vm=dalvik
*threads
3  main
15 Binder Thread #3
13 Binder Thread #2
11 Binder Thread #1
9  JDWP
7  Signal Catcher
5  HeapWorker
*methods
0x412c0850  android/view/ViewTreeObserver$InternalInsetsInfo  <init>  ()V
0x4108bf90  android/graphics/Bitmap ni  ()I Bitmap.java 1047
```

[...]
*end

version section

The first line is the file version number, currently 1.

The second line indicates that we use a common clock across all threads.

A future version may use pre-thread CPU time counters that are independent for every thread.

threads section

One line per thread. Each line consists of two parts: the thread ID, followed by a tab, followed by the thread name. There are few restrictions on what a valid thread name is, so include everything to the end of the line.

methods section

One line per method entry or exit. A line consists of four pieces, separated by tab marks: method-ID [TAB] class-name [TAB] method-name [TAB] signature. Only the methods that were actually entered or

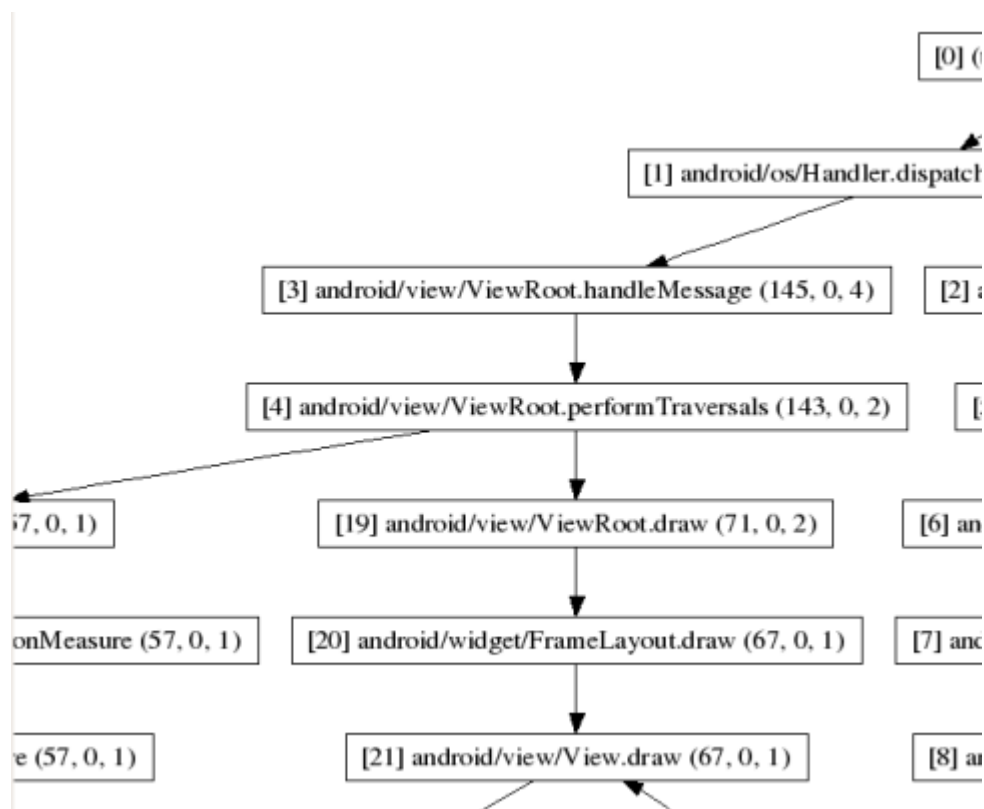
exited are included in the list. Note that all three identifiers are required to uniquely identify a method.

Neither the threads nor methods sections are sorted.

Using dmtracedump

Dmtracedump, a tool that gives you an alternate way of generating graphical call-stack diagrams from trace log files.

The tool uses the Graphviz Dot utility to create the graphical output, so you need to install Graphviz before running dmtracedump.



For each node, dmtracedump shows <ref> callname (<inc-ms>, <exc-ms>, <numcalls>), where

- <ref> -- Call reference number, as used in trace logs
- <inc-ms> -- Inclusive elapsed time (milliseconds spent in method, including all child methods)
- <exc-ms> -- Exclusive elapsed time (milliseconds spent in method, not including any child methods)
- <numcalls> -- Number of calls

The usage for dmtracedump is:

dmtracedump [-ho] [-s sortable] [-d trace-base-name] [-g outfile] <trace-base-name>

Option	Description
-d <trace-base-name>	Diff with this trace name
-g <outfile>	Generate output to <outfile>
-h	Turn on HTML output
-o	Dump the trace file instead of profiling
-s <sortable>	URL base to the location of the sortable javascript file
-t <percent>	Minimum threshold for including child nodes in the graph (child's inclusive time as a percentage of parent inclusive time). If this option is not used, the default threshold is 20%.