性能分析工具 systrace

Last edited by **caoquanli** 1 month ago

性能分析工具-Systrace

概述

Systrace是Google在android4.1版本之后推出的，对系统性能分析的工具，在调查UI性能方面更出色。在谷歌的官方认为保证系统能够连续不间断地提供每秒60帧的运行状态，系统就很流畅，。当出现掉帧时（也可称为Jank）也就是说系统卡顿的时候，需要知道当前整个系统所处的状态，systrace便是最佳的选择，它能检测android系统各个组件随着时间的运行状态，并能提示该如何有效地修复问题。简单点说就是，执行一条命令，接着执行你卡顿的步骤，生成一个HTML报告，根据这个报告，会告诉你哪里有问题，并告诉你修复它们的建议。接下来说说systrace如何使用以及如何解读。

Systrace运行

要运行systrace，请完成以下步骤：
1.下载并安装最新的Android SDK工具。
2.安装Python 。
3.连接使用USB调试将运行Android 4.3（API级别18）或更高版本的设备连接到开发系统 。
该systrace工具在Android SDK工具包中提供，位于android-sdk/platform-tools/systrace/。
需要进入到这个目录里面，去执行Systrace命令。

Systrace使用

命令行

```
python systrace.py [options] [category1] [category2] ... [categoryN]
```

[options]是一些命令参数 [category]是你觉有问题的系统模块

[options] 命令参数讲解:

options	解释
-o **< FILE >**	输出的目标文件，如果没有指定，就保存在当前的目录下面，名字为trace.html
-t N, –time=N	执行时间，默认5s
-b N, –buf-size=N	buffer大小（单位kB),用于限制trace总大小，默认无上限
-k **< KFUNCS >**，–ktrace = **< KFUNCS >**	追踪kernel函数，用逗号分隔
-a **< APP_NAME >**,–app = **< APP_NAME >**	追踪应用包名，用逗号分隔
–from-file=**< FROM_FILE >**	从文件中创建互动的systrace
-e **< DEVICE_SERIAL >**,–serial=**< DEVICE_SERIAL > **	指定设备
-l, –list-categories	列举可用的category

[category] 系统模块:

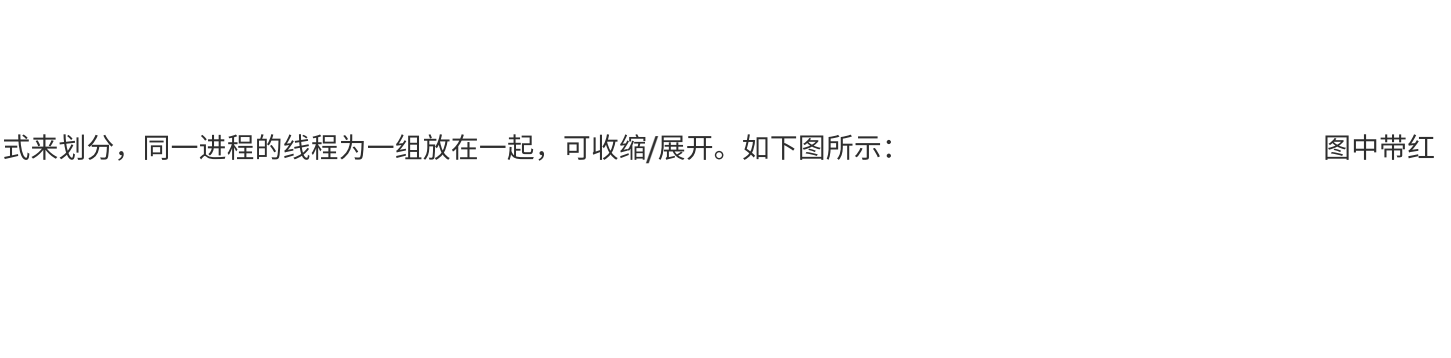
```
      gfx - Graphics
    input - Input
     view - View System
  webview - WebView
       wm - Window Manager
       am - Activity Manager
       sm - Sync Manager
    audio - Audio
    video - Video
```

```
       camera - Camera
         hal - Hardware Modules
         app - Application
         res - Resource Loading
       dalvik - Dalvik VM
           rs - RenderScript
       bionic - Bionic C Library
        power - Power Management
           pm - Package Manager
           ss - System Server
     database - Database
      network - Network
          adb - ADB
          pdx - PDX services
        sched - CPU Scheduling
          irq - IRQ Events
          i2c - I2C Events
         freq - CPU Frequency
         idle - CPU Idle
         disk - Disk I/O
          mmc - eMMC commands
         load - CPU Load
        workq - Kernel Workqueues
 memreclaim - Kernel Memory Reclaim
 regulators - Voltage and Current Regulators
   binder_driver - Binder Kernel driver
   binder_lock - Binder global lock trace
   pagecache - Page cache
```

这里看下几个比较常用的模块：
sched：CPU调度的信息，非常重要；你能看到CPU在每个时间段在运行什么线程；线程调度情况，比如锁信息。
gfx：Graphic系统的相关信息，包括SurfaceFlinger，VSYNC消息，Texture，RenderThread等；分析卡顿非常依赖这个。
view：View绘制系统的相关信息，比如onMeasure，onLayout等；滑动，对分析卡顿比较有帮助。
am：ActivityManager调用的相关信息，用来分析Activity的启动过程比较有效。
dalvik： 虚拟机相关信息，比如GC停顿等。
binder_driver：Binder驱动的相关信息，如果你怀疑是Binder IPC的问题，不妨打开这个。
core_services：SystemServer中系统核心Service的相关信息，分析特定问题用。
input：输入事件，列表滑动，桌面滑动等流畅性问题。

Systrace解读

在执行以上命令后，我们会生成一个trace.xml文件，我们需要使用Google Chrome浏览器打开(只能使用Google Chrome浏览器)，才能分析。打开后我们会发现一个图形化界面，横坐标是以时间为单位，纵坐标是以进程-线程的方式来划分，同一进程的线程为一组放在一起，可收缩/展开。如下图所示：



图中带红色指向箭头的几个都是可操作的
鼠标操作模式，就是从上至下以此为选择、移动、缩放和测量(时间间隔)的工具，需要先选中，后使用。
搜索栏可以搜索自己自定义的Systrace等。
最右边的那个框：可以查看这段时间alert的出现的次数。

1.Frames解读

我们可以看到每个app进程，都有一个Frames行，并且这一行都有个带有圆圈的F，正常情况带有绿色圆圈的F都是正常的，带有黄色圆圈或者红色圆圈的F都是有问题的帧，代表着这一帧超过16.6ms，这就是出现丢帧情况(Jank)。这时需要通过放大那一帧进一步分析问题。对于Android 5.0(API level 21)或者更高的设备，主要聚焦在UI Thread和Render Thread（渲染线程）这两个线程当中分析。对于更早的版本，则所有工作在UI Thread。

2.Alert解读

当我们发现这一帧有问题的时候，并且去点击带红色的圆圈F时，会在此页面的下面一栏出现一个提示信息如下图所

以：　　从图片中我们可以看到Description，告诉你了一些修复问题的建议。

3.Trace.html的一些导航操作

操作	作用
w	放大，[+shift]速度更快
s	缩小，[+shift]速度更快
a	左移，[+shift]速度更快
d	右移，[+shift]速度更快
f	放大当前选定区域
m	标记当前选定区域
v	高亮VSync
g	切换是否显示60hz的网格线
0	恢复trace到初始态，这里是数字0而非字母o
h	切换是否显示详情
/	搜索关键字
enter	显示搜索结果，可通过← →定位搜索结果
?	显示帮助功能

Systrace自定义

systrace没有办法在代码中控制跟踪运行的开始和结束；那么，如果我们要分析App的启动性能，我点了桌面图标，把Trace时间设置为10s，我怎么知道这10s中，哪段时间是我App的启动过程？在分析framework的代码的时候，我们经常会看到Trace.traceBegin，Trace.traceEnd这样的成对出现的代码，这就是自定义Systrace的代码。

java framework层的自定义：

```
import android.os.Trace;
Trace.traceBegin(long traceTag, String methodName)
//要分析的代码
//....
Trace.traceEnd(long traceTag)
```

appk层的自定义格式：

```
    import android.os.Trace;
    Trace.beginSection(String sectionName);
    // .. 其他代码
    // ...
    // .. 结束处
    Trace.endSection();
```

native framework层

```
#include<utils/Trace.h>
ATRACE_CALL();
```

例如

我觉得Activty的Oncreate在创建的过程中是有问题，因此就在onCreate中加如下代码：

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Trace.beginSection("sunchao_onCreate");
        //有问题的代码
        Trace.endSection();
```

因此我们得到trace就会带上"sunchao_onCreate"这个过程的运行时间段信息。如下：

我们可以在任意自己感兴趣的地方添加自定义的trace；一般来说，分析过程就是，你怀疑哪里有问题，就在那那个函数加上Label，运行一遍抓一个Trace，看看自己的猜测对不对；如果猜测正确，进一步加Label缩小范围，定位到具体的自定义函数，函数最终调用到系统内部，那就开启系统相关模块的Trace，继续定位；如果猜测错误，那就转移目标，一步步缩小范围，直至问题收敛。

Systrace例子

写个包为com.demo.sunchao.demo3的APK 代码如下：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {
    //下载按钮
    private Button download;
    //返回按钮
    private Button back;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        download = (Button) findViewById(R.id.bt);
        back = (Button) findViewById(R.id.back);
        back.setOnClickListener(this);
        download.setOnClickListener(this);
    }

    private void download() {
        try{
            Thread.sleep(4000);//休眠4秒
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onClick(View view) {
        if (view == bt) {
            download();//调用UI主线程下载函数
            Toast.makeText(MainActivity.this, "complete download", Toast.LENGTH_SHORT).show
        }
        if (view  == back) {
            finish();
        }
    }
}
```

安装这个APK，运行。我们发现当我们点击这个download按钮的时候，再去点击back键，发现出现了卡顿现象，过了几秒页面才关闭。 此时我们猜测是由于UI线程执行了耗时的操作导致的，因此为了验证我们的猜测，我们需要Systrace去分析。 打开APP页面，连接手机到电脑，在电脑终端上面执行python systrace.py -t 10 -a com.demo.sunchao.demo3，执行你卡顿的步骤将会生成一个trace.xml文件，打开文件如下：

我们可以看有红色的圆圈F，w键放大点击点击查看红色的圆圈F，会在页面的下方显

示如下：

从下面的描述，大致的意思是：生成这个框架的工作被分解了几毫秒，导

致了jank的出现。确保UI线程上的代码不会阻塞，因此我们可以发现，根据这个提示我们可以发现UI线程执行了大量耗时的操作我们可以根据这个提示，加上我们对代码的分析，然后在我们觉得有问题的地方加入Trace.beginSection，Trace.endSection来进行验证。

总结：

可能这个例子有点简单，实际分析的问题要比这个难多了，systrace命令的使用说到这里也没什么要注意的了；其实命令的使用不难，分析思路也很简单：合理假设-> 加自定义Label验证 -> (结论成立-> 缩小范围继续)／(结论推翻-> 重新假设) 这样一步一步直至问题收敛；如果你分析了一段时间，发现问题是发散的，一会儿觉得是这里有问题，一会儿又觉得是那里有问题，那么或许你的假设根本就是错误的；需要重新调整方向分析。

参考连接：https://developer.android.com/studio/command-line/systrace