# TODO

<http://gityuan.com/2018/01/27/android-log/>

<https://blog.csdn.net/c_z_w/article/details/82187616>

ALOGV 在release版本不打印？？

# wtf

可怕的失败：报告一个永远不可能发生的情况。

大概是用来打印正常情况下永远不会发生bug？

的意思是what a terrible failure

抛异常设计技巧

**try** {  
 mPackageManagerService.systemReady();  
} **catch** (Throwable e) {  
 reportWtf(**"making Package Manager Service ready"**, e);  
}

# 日志缓存区大小

= **"persist.logd.size"**;

buffer大小设置的优先级顺序为:

1. persist.logd.size.xxx; 比如persist.logd.size.system;
2. persist.logd.size;
3. ro.logd.size;
4. LOG\_BUFFER\_SIZE, 即256k;
5. LOG\_BUFFER\_MIN\_SIZE, 即64k。

# Logcat

在Android中不同的log写到不同的设备中，共有/dev/log/system, /dev/log/main, /dev/log/radion, /dev/log/events四中类型。其中默认Log.v等写入/dev/log/main中。Slog写入/dev/log/system中。

<https://blog.csdn.net/hudashi/article/details/7073155>

## EventLog概述

在调试分析Android的过程中，经常会查看EventLog，它非常简洁明了地展现当前Activity各种状态，当然不至于此，比如还有window、surfaceFlinger、battery等其他的信息。

  如果在终端中输入：logcat -b events，就能够输出大量类似下面的日志信息

**[javascript]** [view plain](https://blog.csdn.net/qq1028850792/article/details/80287021) [copy](https://blog.csdn.net/qq1028850792/article/details/80287021)

1. 07-02 01:38:27.718  3623  3636 I am\_home\_stack\_moved: [0,0,1,1,startingNewTask]
2. 07-02 01:38:27.719  3623  3636 I wm\_task\_moved: [6,1,0]
3. 07-02 01:38:27.722  3623  3636 I am\_create\_task: [0,7]
4. 07-02 01:38:27.722  3623  3636 I am\_create\_activity: [0,224793551,7,com.android.settings/.Settings,android.intent.action
5. .MAIN,NULL,NULL,807403520]
6. 07-02 01:38:27.723  3623  3636 I wm\_task\_created: [7,1]
7. 07-02 01:38:27.723  3623  3636 I wm\_task\_moved: [7,1,1]

通过字面意思，就能得到不少信息量，比如am\_create\_activity，创建activity，但是后面括号中内容的具体含义，其实有很高的价值。既然Event日志辣么辣么重要

在Android源码中，是通过如下语句打印出event日志的：（这里我以ActivityStackSupervisor.java文件说明，其它文件方法一样）

**[java]** [view plain](https://blog.csdn.net/qq1028850792/article/details/80287021) [copy](https://blog.csdn.net/qq1028850792/article/details/80287021)

1. **import** android.util.EventLog;
2. **public** **final** **class** ActivityStackSupervisor **implements** DisplayListener {
3. **void** moveHomeStack(**boolean** toFront, String reason, ActivityStack lastFocusedStack) {
4. EventLog.writeEvent(EventLogTags.AM\_HOME\_STACK\_MOVED,
5. mCurrentUser, toFront ? 1 : 0, stacks.get(topNdx).getStackId(),
6. mFocusedStack == **null** ? -1 : mFocusedStack.getStackId(), reason);
7. }
8. }

EventLog.writeEvent

android\_util\_EventLog\_writeEvent\_Array //android\_util\_EventLog.cpp

android\_bWriteLog // system/core/include/log/Log.h

## 源码分析

### Event TAG

<https://blog.csdn.net/yaowei514473839/article/details/53513435>

日志

EventLog.*writeEvent*(EventLogTags.***LOCK\_SCREEN\_TYPE***, unlockMethod);

# system

Slog.*d*

adb shell logcat -b main -b system -b radio -b events -v time > logcat0630.log

adb bugreport > bugreport0622.txt

adb pull /data/anr

# EventLog

Sd

## 概述

在调试分析Android的过程中，比较常用的地查看EventLog，非常简洁明了地展现当前Activity各种状态，当然不至于此，比如还有window的信息。那么本文就列举以下am相关的tags含义。

本文涉及的源码类有EventLog.java, EventLogTags.java，另外tags格式的定义位于文件/system/etc/event-log-tags。

如果在终端输入：

logcat -b events

那么会输出大量类似这样的信息：

06-01 13:44:55.518 7361 8289 I am\_create\_service: [0,111484394,.StatService,10094,7769]

06-01 13:44:55.540 7361 8343 I am\_proc\_bound: [0,3976,com.android.providers.calendar]

06-01 13:44:55.599 7361 8033 I am\_create\_service: [0,61349752,.UpdateService,10034,1351]

06-01 13:44:55.625 7361 7774 I am\_destroy\_service: [0,61349752,1351]

...

通过字面意思，就能得到不少信息量，比如am\_create\_service，创建service，但是后面括号中内容的具体含义，其实有很高的价值。 接下来通过一张表格来展示含义。

## 二. EventLog

### 2.1 ActivityManager

| **Num** | **TagName** | **格式** | **功能** |
| --- | --- | --- | --- |
| 30001 | am\_finish\_activity | User,Token,TaskID,ComponentName,Reason |  |
| 30002 | am\_task\_to\_front | User,Task |  |
| 30003 | am\_new\_intent | User,Token,TaskID,ComponentName,Action,MIMEType,URI,Flags |  |
| 30004 | am\_create\_task | User ,Task ID |  |
| 30005 | am\_create\_activity | User ,Token ,TaskID ,ComponentName,Action,MIMEType,URI,Flags |  |
| 30006 | am\_restart\_activity | User ,Token ,TaskID,ComponentName |  |
| 30007 | am\_resume\_activity | User ,Token ,TaskID,ComponentName |  |
| 30008 | am\_anr | User ,pid ,Package Name,Flags ,reason | ANR |
| 30009 | am\_activity\_launch\_time | User ,Token ,ComponentName,time |  |
| 30010 | am\_proc\_bound | User ,PID ,ProcessName |  |
| 30011 | am\_proc\_died | User ,PID ,ProcessName |  |
| 30012 | am\_failed\_to\_pause | User ,Token ,Wanting to pause,Currently pausing |  |
| 30013 | am\_pause\_activity | User ,Token ,ComponentName |  |
| 30014 | am\_proc\_start | User ,PID ,UID ,ProcessName,Type,Component |  |
| 30015 | am\_proc\_bad | User ,UID ,ProcessName |  |
| 30016 | am\_proc\_good | User ,UID ,ProcessName |  |
| 30017 | am\_low\_memory | NumProcesses | Lru |
| 30018 | am\_destroy\_activity | User ,Token ,TaskID,ComponentName,Reason |  |
| 30019 | am\_relaunch\_resume\_activity | User ,Token ,TaskID,ComponentName |  |
| 30020 | am\_relaunch\_activity | User ,Token ,TaskID,ComponentName |  |
| 30021 | am\_on\_paused\_called | User ,ComponentName |  |
| 30022 | am\_on\_resume\_called | User ,ComponentName |  |
| 30023 | am\_kill | User ,PID ,ProcessName,OomAdj ,Reason | 杀进程 |
| 30024 | am\_broadcast\_discard\_filter | User ,Broadcast ,Action,ReceiverNumber,BroadcastFilter |  |
| 30025 | am\_broadcast\_discard\_app | User ,Broadcast ,Action,ReceiverNumber,App |  |
| 30030 | am\_create\_service | User ,ServiceRecord ,Name,UID ,PID |  |
| 30031 | am\_destroy\_service | User ,ServiceRecord ,PID |  |
| 30032 | am\_process\_crashed\_too\_much | User ,Name,PID |  |
| 30033 | am\_drop\_process | PID |  |
| 30034 | am\_service\_crashed\_too\_much | User ,Crash Count,ComponentName,PID |  |
| 30035 | am\_schedule\_service\_restart | User ,ComponentName,Time |  |
| 30036 | am\_provider\_lost\_process | User ,Package Name,UID ,Name |  |
| 30037 | am\_process\_start\_timeout | User ,PID ,UID ,ProcessName | timeout |
| 30039 | am\_crash | User ,PID ,ProcessName,Flags ,Exception,Message,File,Line | Crash |
| 30040 | am\_wtf | User ,PID ,ProcessName,Flags ,Tag,Message | Wtf |
| 30041 | am\_switch\_user | id |  |
| 30042 | am\_activity\_fully\_drawn\_time | User ,Token ,ComponentName,time |  |
| 30043 | am\_focused\_activity | User ,ComponentName |  |
| 30044 | am\_home\_stack\_moved | User ,To Front ,Top Stack Id ,Focused Stack Id ,Reason |  |
| 30045 | am\_pre\_boot | User ,Package |  |
| 30046 | am\_meminfo | Cached,Free,Zram,Kernel,Native | 内存 |
| 30047 | am\_pss | Pid, UID, ProcessName, Pss, Uss | 进程 |

下面列举**tag可能使用的部分场景**：

* am\_low\_memory：位于AMS.killAllBackgroundProcesses或者AMS.appDiedLocked，记录当前Lru进程队列长度。
* am\_pss：位于AMS.recordPssSampleLocked(
* am\_meminfo：位于AMS.dumpApplicationMemoryUsage
* am\_proc\_start:位于AMS.startProcessLocked，启动进程
* am\_proc\_bound:位于AMS.attachApplicationLocked
* am\_kill: 位于ProcessRecord.kill，杀掉进程
* am\_anr: 位于AMS.appNotResponding
* am\_crash:位于AMS.handleApplicationCrashInner
* am\_wtf:位于AMS.handleApplicationWtf
* am\_activity\_launch\_time：位于ActivityRecord.reportLaunchTimeLocked()，后面两个参数分别是thisTime和 totalTime.
* am\_activity\_fully\_drawn\_time:位于ActivityRecord.reportFullyDrawnLocked, 后面两个参数分别是thisTime和 totalTime
* am\_broadcast\_discard\_filter:位于BroadcastQueue.logBroadcastReceiverDiscardLocked
* am\_broadcast\_discard\_app:位于BroadcastQueue.logBroadcastReceiverDiscardLocked

Activity生命周期相关的方法:

* am\_on\_resume\_called: 位于AT.performResumeActivity
* am\_on\_paused\_called: 位于AT.performPauseActivity, performDestroyActivity
* am\_resume\_activity: 位于AS.resumeTopActivityInnerLocked
* am\_pause\_activity: 位于AS.startPausingLocked
* am\_finish\_activity: 位于AS.finishActivityLocked, removeHistoryRecordsForAppLocked
* am\_destroy\_activity: 位于AS.destroyActivityLocked
* am\_focused\_activity: 位于AMS.setFocusedActivityLocked, clearFocusedActivity
* am\_restart\_activity: 位于ASS.realStartActivityLocked
* am\_create\_activity: 位于ASS.startActivityUncheckedLocked
* am\_new\_intent: 位于ASS.startActivityUncheckedLocked
* am\_task\_to\_front: 位于AS.moveTaskToFrontLocked

Window相关

* wm\_task\_moved: 位于TaskStack.positionTask()
  + TaskId, toTop ? 1 : 0, position；
  + 206，1，3，是指吧TaskId=206的移动到栈顶(即该栈的长度为4)
* am\_home\_stack\_moved: 位于ASS.moveHomeStack
  + CurrentUser, toFront ? 1:0 , homStackId, FocusedStackId
  + 0,1,0,0, 是指userId=0, home栈顶的StackId=0, 当前focusedStackId=0,

### 2.2 Power

| **Num** | **TagName** | **格式** | **功能** |
| --- | --- | --- | --- |
| 2722 | battery\_level | level, voltage, temperature |  |
| 2723 | battery\_status | status,health,present,plugged,technology |  |
| 2730 | battery\_discharge | duration, minLevel,maxLevel |  |
| 2724 | power\_sleep\_requested | wakeLocksCleared | 唤醒锁数量 |
| 2725 | power\_screen\_broadcast\_send | wakelockCount |  |
| 2726 | power\_screen\_broadcast\_done | on, broadcastDuration, wakelockCount |  |
| 2727 | power\_screen\_broadcast\_stop | which,wakelockCount | 系统还没进入ready状态 |
| 2728 | power\_screen\_state | offOrOn, becauseOfUser, totalTouchDownTime, touchCycles |  |
| 2729 | power\_partial\_wake\_state | releasedorAcquired, tag |  |

部分含义：

* battery\_level: [19,3660,352] //剩余电量19%, 电池电压3.66v, 电池温度35.2℃
* power\_screen\_state: [0,3,0,0] // 灭屏状态(0), 屏幕超时(3). 当然还有其他设备管理策略(1),其他理由都为用户行为(2)
* power\_screen\_state: [1,0,0,0] // 亮屏状态(1)

下面列举**tag可能使用的部分场景**：

* power\_sleep\_requested: 位于PMS.goToSleepNoUpdateLocked
* power\_screen\_state:位于Notifer.handleEarlyInteractiveChange, handleLateInteractiveChange

## 三. EventLog完整语义分析

在源码EventLogTags.java中,有大量类似的定义,那么括号中数字是什么含义呢? (以进程启动为例)

30014 am\_proc\_start (User|1|5),(PID|1|5),(UID|1|5),(Process Name|3),(Type|3),(Component|3)

am\_proc\_start之后紧跟着的几个括号，其中括号里的内容**格式**如下：

(<name>|data type[|data unit])

(<名字>|数据类型[|数据单位])

那么(User|1|5) ==> 名字为User, 数据类型为1，数据单位为5，下面再来看看数据类型和数据单位：

### 3.1 数据类型

* 1: int
* 2: long
* 3: string
* 4: list

数据类型中int和string用得最多.

### 3.2 数据单位

* 1: Number of objects(对象个数)
* 2: Number of bytes(字节数)
* 3: Number of milliseconds(毫秒)
* 4: Number of allocations(分配个数)
* 5: Id
* 6: Percent(百分比)

### 3.3 实例解析

有了前面的准备知识，再来完整的看看如下语句：

am\_proc\_start (User|1|5),(PID|1|5),(UID|1|5),(Process Name|3),(Type|3),(Component|3) am\_proc\_start: [0,9227,10002,com.android.browser,content provider,com.android.browser/.provider.BrowserProvider2]

含义如下：

进程启动: UserId=0, pid=9227, uid=10002, ProcessName=com.android.browser, 数据类型=ContentProvider, 组件=com.android.browser/.provider.BrowserProvider2

## 餐卡破

http://gityuan.com/2016/05/15/event-log/