Android adb bugreport工具分析和使用

访问： 151492次
积分： 2400
等级： BLOG 5
排名： 第16080名

原创： 82篇 转载： 15篇
译文： 2篇 评论： 56条

标签： android bugreport

2016-07-19 11:53 5726人阅读 评论(0) 收藏

分类：

Android升华之路之系统篇（6）

目录(?) [+]

bugreport是什么，怎么用

Android系统想要成为一个功能完备，生态繁荣的操作系统，那就

用开发中，app程序的调试分析是日常生产中进程会进行的工作。

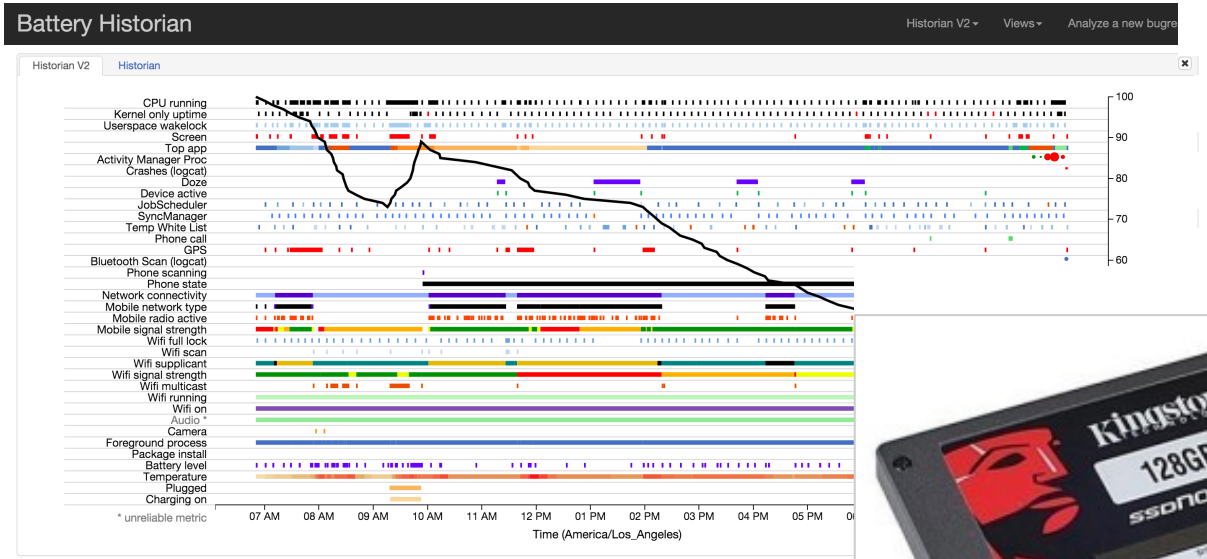
文章存档

2017年09月 (1)
2017年03月 (1)

统平台和某个app在运行一段时间之内的所有信息，专门开发了bugreport工具。这个工具使用起来十分简单，只要在终端执行（linux或者win）：

```
1 adb bugreport > bugreport.txt
```

即可生成bugreport文件。但是有一个问题是，这个生成的文件有的时候异常庞大，能够达到15M+,想一想对于一个txt文本格式的文件内容长度达到了15M+是一个什么概念，如果使用文本工具打开查看将是一个噩梦。因此google针对android 5.0（api 21）以上的系统开发了一个叫做battery historian这个工具就是用来解析这个txt文本文件，然后使用web图形的形式展现出来，这样出来的效化，更加可读。它的基本界面像下面这个样子：



目前google已经将bettery historian开源了，开源项目的地址：

https://github.com/google/battery-historian

google写了一个比较详细的说明文档，大家可以自行查阅一下。这

1. Brightness
2. CPU running
3. Charging on
4. Charging status
5. Health
6. JobScheduler
7. Kernel only uptime
8. Level
9. Package active
10. Partial wakelock
11. Phone scanning
12. Phone state
13. Plug
14. Plugged
15. Screen
16. Temperature
17. Top app
18. Voltage
19. Wifi on
20. Wifi running
21. Wifi supplicant

数据还是比较详细的。

当然，同样的bugreport数据也可以有不同的解析和阅读方式，你

话，你还有别的选择，那就是选择Sony开源的ChkBugReport，这

视角去解读bugreport文件，界面简单明了：

这个项目的文档：

http://developer.sonymobile.com/2012/01/25/new-bugreport-analysis-t

开源地址首页：

https://github.com/sonyxperiadev/ChkBugReport

这里说明一下，笔者使用过ChkBugReport这个工具，感觉很不错

外ChkBugReport这个工具还有一点bug，不过不影响使用。

关闭

bugreport的原理是什么？

下面我们简要分析一下adb bugreport运行的原理。我们知道，使用bugreport只要执行adb bugreport命令就可以了，因此我们的分析肯定是从adbd这个daemon进程开始，我们查看这个进程的代码的时候发现这里处理了bugreport选项：

adb_commandline@system/core/adb/commandline.cpp

```
else if (!strcmp(argv[0], "bugreport")) {
    if (argc != 1) return usage();
    return send_shell_command(ttype, serial, "shell:bugreport");
}
```

我们可以清楚地看到，这里判断如果附带的参数是bugreport的话，那就直接调用send_shell数处理，这个函数的代码比较简单，我们就不分析了，这个函数的功能就是使用shell执行参令，因此我们这里相当于执行了bugreport命令。

在android设备中，bugreport命令存在于system/bin/目录下，这是一个可执行文件，所以我们可执行文件实现的地方，它的实现代码在/frameworks/native/cmds/

Name		Date	Size
..		13-Oct-2015	4 KiB
Android.mk	H A D	13-Oct-2015	198
bugreport.cpp	H A D	13-Oct-2015	2.9 KiB

我们看到，bugreport的实现是比较简单的，只有一个Android.mk和Android.mk文件：

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)

LOCAL_SRC_FILES:= bugreport.cpp

LOCAL_MODULE:= bugreport

LOCAL_CFLAGS := -Wall

LOCAL_SHARED_LIBRARIES := libcutils

include $(BUILD_EXECUTABLE)

```

这里我们看到该目录下的代码会被编译成一个名字叫做bugreport的可执行文件，这就是我

现在我们看一下bugreport.cpp文件的实现，这个文件中代码比较简单，只有一个main函数：

```
// This program will trigger the dumpstate service to start a call to
// dumpstate, then connect to the dumpstate local client to read the
// output. All of the dumpstate output is written to stdout, including
// any errors encountered while reading/writing the output.
int main() {
  // Start the dumpstate service.
  property_set("ctl.start", "dumpstate");

  // Socket will not be available until service starts.
  int s;
  for (int i = 0; i < 20; i++) {
    s = socket_local_client("dumpstate", ANDROID_SOCKET_
                            SOCK_STREAM);
    if (s >= 0)
      break;
    // Try again in 1 second.
    sleep(1);
  }

  if (s == -1) {
```

```
  printf("Failed to connect to dumpstate service: %s\n", strerror(errno));
  return 1;
}

// Set a timeout so that if nothing is read in 3 minutes, we'll stop
// reading and quit. No timeout in dumpstate is longer than 60 seconds,
// so this gives lots of leeway in case of unforeseen time outs.
struct timeval tv;
tv.tv_sec = 3 * 60;
tv.tv_usec = 0;
if (setsockopt(s, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv)) == -1) {
  printf("WARNING: Cannot set socket timeout: %s\n", strerror(errno));
}

while (1) {
  char buffer[65536];
  ssize_t bytes_read = TEMP_FAILURE_RETRY(read(s, buffer, sizeof(buffer)));
  if (bytes_read == 0) {
    break;
  } else if (bytes_read == -1) {
    // EAGAIN really means time out, so change the errno.
    if (errno == EAGAIN) {
      errno = ETIMEDOUT;
    }
    printf("\nBugreport read terminated abnormally (%s).\
    break;
  }

  ssize_t bytes_to_send = bytes_read;
  ssize_t bytes_written;
  do {
    bytes_written = TEMP_FAILURE_RETRY(write(STDOUT_F
                                       buffer + bytes_read - bytes_to_
                                       bytes_to_send));
    if (bytes_written == -1) {
```

```
56          printf("Failed to write data to stdout: read %zd, trying to send %zd (%s)\n",
57                bytes_read, bytes_to_send, strerror(errno));
58          return 1;
59        }
60        bytes_to_send -= bytes_written;
61      } while (bytes_written != 0 && bytes_to_send > 0);
62    }
63
64    close(s);
65    return 0;
66  }
```

这里的代码非常简单，主要的逻辑就是：

1.启动dumpstate service

2. 和dumpstate service建立socket链接

3. 从socket中读取数据，并且答应到stdout中

4. 读取完成之后关闭socket，然后退出

因此，我们分析的重点需要转移到dumpstate中了。这里说明一下，前面启动dumpstate servic

用系统属性来实现，这个属性的改变消息会被init进程收到，然后

dumpstate其实也是一个可执行文件，也存在于system/bin目录下。

dumpstate，只是bugreport将dumpstate包装了一下而已。

现在我们需要分析一下dumpstate的实现，它的实现代码在：frame

们看下这个目录下的代码结构：

Name		Date	Size
..		13-Oct-2015	4 KiB
Android.mk	H A D	13-Oct-2015	510
dumpstate.c	H A D	13-Oct-2015	26.2 KiB
dumpstate.h	H A D	13-Oct-2015	2.8 KiB
libdumpstate_default.c	H A D	13-Oct-2015	677
utils.c	H A D	13-Oct-2015	24.1 KiB

这里的代码也是十分简单，只要少数的几个实现文件，其中main函数在dumpstate.c文件中，这个main函数我们这里不详细分析了，总结下它的主要工作：

1. 根据启动参数，初始化相关资源

2. 如果启动参数中带有-s的话（init启动会加上这个参数），就表示使用socket，那么就启动在这个socket中等待链接。

3. 如果client端（也就是bugreport进程）链接成功，那就初始化所要用到的内存，并且设置优先级，防止被OOM干掉。

4. 然后使用vibrator震动一下（如果设备有这个硬件的话），提示用户开始截取log了

5. 调用dumpstate函数开始真正的dump工作

6. dump完成之后再次调用vibrator震动3次，提示用户dump完成。

现在我们看下dumpstate函数的实现：

```
1  /* dumps the current system state to stdout */
2  static void dumpstate() {
3      unsigned long timeout;
4      time_t now = time(NULL);
5      char build[PROPERTY_VALUE_MAX], fingerprint[PROPERT
6      char radio[PROPERTY_VALUE_MAX], bootloader[PROPER
7      char network[PROPERTY_VALUE_MAX], date[80];
8      char build_type[PROPERTY_VALUE_MAX];
```

```
9
10  property_get("ro.build.display.id", build, "(unknown)");
11  property_get("ro.build.fingerprint", fingerprint, "(unknown)");
12  property_get("ro.build.type", build_type, "(unknown)");
13  property_get("ro.baseband", radio, "(unknown)");
14  property_get("ro.bootloader", bootloader, "(unknown)");
15  property_get("gsm.operator.alpha", network, "(unknown)");
16  strftime(date, sizeof(date), "%Y-%m-%d %H:%M:%S", localtime(&now));
17
18  printf("============================================================\n");
19  printf("== dumpstate: %s\n", date);
20  printf("============================================================\n");
21
22  printf("\n");
23  printf("Build: %s\n", build);
24  printf("Build fingerprint: '%s'\n", fingerprint); /* format is important for other tool
25  printf("Bootloader: %s\n", bootloader);
26  printf("Radio: %s\n", radio);
27  printf("Network: %s\n", network);
28
29  printf("Kernel: ");
30  dump_file(NULL, "/proc/version");
31  printf("Command line: %s\n", strtok(cmdline_buf, "\n"));
32  printf("\n");
33
34  dump_dev_files("TRUSTY VERSION", "/sys/bus/platform/
35  run_command("UPTIME", 10, "uptime", NULL);
36  dump_files("UPTIME MMC PERF", mmcblk0, skip_not_sta
37  dump_file("MEMORY INFO", "/proc/meminfo");
38  run_command("CPU INFO", 10, "top", "-n", "1", "-d", "1",
39  run_command("PROCRANK", 20, "procrank", NULL);
40  dump_file("VIRTUAL MEMORY STATS", "/proc/vmstat");
41  dump_file("VMALLOC INFO", "/proc/vmallocinfo");
42  dump_file("SLAB INFO", "/proc/slabinfo");
43  dump_file("ZONEINFO", "/proc/zoneinfo");
```

```
44  dump_file("PAGETYPEINFO", "/proc/pagetypeinfo");
45  dump_file("BUDDYINFO", "/proc/buddyinfo");
46  dump_file("FRAGMENTATION INFO", "/d/extfrag/unusable_index");
47
48  dump_file("KERNEL WAKELOCKS", "/proc/wakelocks");
49  dump_file("KERNEL WAKE SOURCES", "/d/wakeup_sources");
50  dump_file("KERNEL CPUFREQ", "/sys/devices/system/cpu/cpu0/cpufreq/stats/time_in_state");
51  dump_file("KERNEL SYNC", "/d/sync");
52
53  run_command("PROCESSES", 10, "ps", "-P", NULL);
54  run_command("PROCESSES AND THREADS", 10, "ps", "-t", "-p", "-P", NULL);
55  run_command("PROCESSES (SELINUX LABELS)", 10, "ps", "-Z", NULL);
56  run_command("LIBRANK", 10, "librank", NULL);
57
58  do_dmesg();
59
60  run_command("LIST OF OPEN FILES", 10, SU_PATH, "root", "lsof", NULL);
61  for_each_pid(do_showmap, "SMAPS OF ALL PROCESSES");
62  for_each_tid(show_wchan, "BLOCKED PROCESS WAIT-CHANNELS");
63
64  if (screenshot_path[0]) {
65      ALOGI("taking screenshot\n");
66      run_command(NULL, 10, "/system/bin/screencap", "-
67      ALOGI("wrote screenshot: %s\n", screenshot_path);
68  }
69
70  // dump_file("EVENT LOG TAGS", "/etc/event-log-tags");
71  // calculate timeout
72  timeout = logcat_timeout("main") + logcat_timeout("syst
73  if (timeout < 20000) {
74      timeout = 20000;
75  }
76  run_command("SYSTEM LOG", timeout / 1000, "logcat",
77  timeout = logcat_timeout("events");
78  if (timeout < 20000) {
```

```
79      timeout = 20000;
80  }
81  run_command("EVENT LOG", timeout / 1000, "logcat", "-b", "events", "-v", "threadtime", "-d", "*:v'
82  timeout = logcat_timeout("radio");
83  if (timeout < 20000) {
84      timeout = 20000;
85  }
86  run_command("RADIO LOG", timeout / 1000, "logcat", "-b", "radio", "-v", "threadtime", "-d", "*:v",
87
88  run_command("LOG STATISTICS", 10, "logcat", "-b", "all", "-S", NULL);
89
90  /* show the traces we collected in main(), if that was done */
91  if (dump_traces_path != NULL) {
92      dump_file("VM TRACES JUST NOW", dump_traces_path);
93  }
94
95  /* only show ANR traces if they're less than 15 minutes old */
96  struct stat st;
97  char anr_traces_path[PATH_MAX];
98  property_get("dalvik.vm.stack-trace-file", anr_traces_path, "");
99  if (!anr_traces_path[0]) {
100     printf("*** NO VM TRACES FILE DEFINED (dalvik.vm.stack-trace-file)\n\n");
101 } else {
102   int fd = TEMP_FAILURE_RETRY(open(anr_traces_path,
103                               O_RDONLY | O_CLOEXEC | O_NOFC
104   if (fd < 0) {
105       printf("*** NO ANR VM TRACES FILE (%s): %s\n\n", a
106   } else {
107       dump_file_from_fd("VM TRACES AT LAST ANR", anr_t
108   }
109 }
110
111 /* slow traces for slow operations */
112 if (anr_traces_path[0] != 0) {
113     int tail = strlen(anr_traces_path)-1;
```

```
114     while (tail > 0 && anr_traces_path[tail] != '/') {
115         tail--;
116     }
117     int i = 0;
118     while (1) {
119         sprintf(anr_traces_path+tail+1, "slow%02d.txt", i);
120         if (stat(anr_traces_path, &st)) {
121             // No traces file at this index, done with the files.
122             break;
123         }
124         dump_file("VM TRACES WHEN SLOW", anr_traces_path);
125         i++;
126     }
127 }
128
129 int dumped = 0;
130 for (size_t i = 0; i < NUM_TOMBSTONES; i++) {
131     if (tombstone_data[i].fd != -1) {
132         dumped = 1;
133         dump_file_from_fd("TOMBSTONE", tombstone_data[i].name, tombstone_data[i
134         tombstone_data[i].fd = -1;
135     }
136 }
137 if (!dumped) {
138     printf("*** NO TOMBSTONES to dump in %s\n\n", TO
139 }
140
141 dump_file("NETWORK DEV INFO", "/proc/net/dev");
142 dump_file("QTAGUID NETWORK INTERFACES INFO", "/pr
143 dump_file("QTAGUID NETWORK INTERFACES INFO (xt)",
144 dump_file("QTAGUID CTRL INFO", "/proc/net/xt_qtaguid
145 dump_file("QTAGUID STATS INFO", "/proc/net/xt_qtaguic
146
147 if (!stat(PSTORE_LAST_KMSG, &st)) {
148     /* Also TODO: Make console-ramoops CAP_SYSLOG pr
```

```
149     dump_file("LAST KMSG", PSTORE_LAST_KMSG);
150 } else {
151     /* TODO: Make last_kmsg CAP_SYSLOG protected. b/5555691 */
152     dump_file("LAST KMSG", "/proc/last_kmsg");
153 }
154
155 /* kernels must set CONFIG_PSTORE_PMSG, slice up pstore with device tree */
156 run_command("LAST LOGCAT", 10, "logcat", "-L", "-v", "threadtime",
157                                 "-b", "all", "-d", "*:v", NULL);
158
159 /* The following have a tendency to get wedged when wifi drivers/fw goes belly-u
160
161 run_command("NETWORK INTERFACES", 10, "ip", "link", NULL);
162
163 run_command("IPv4 ADDRESSES", 10, "ip", "-4", "addr", "show", NULL);
164 run_command("IPv6 ADDRESSES", 10, "ip", "-6", "addr", "show", NULL);
165
166 run_command("IP RULES", 10, "ip", "rule", "show", NULL);
167 run_command("IP RULES v6", 10, "ip", "-6", "rule", "show", NULL);
168
169 dump_route_tables();
170
171 run_command("ARP CACHE", 10, "ip", "-4", "neigh", "sho
172 run_command("IPv6 ND CACHE", 10, "ip", "-6", "neigh",
173
174 run_command("IPTABLES", 10, SU_PATH, "root", "iptable
175 run_command("IP6TABLES", 10, SU_PATH, "root", "ip6ta
176 run_command("IPTABLE NAT", 10, SU_PATH, "root", "ipt
177 /* no ip6 nat */
178 run_command("IPTABLE RAW", 10, SU_PATH, "root", "ipt
179 run_command("IP6TABLE RAW", 10, SU_PATH, "root", "i
180
181 run_command("WIFI NETWORKS", 20,
182         SU_PATH, "root", "wpa_cli", "IFNAME=wlan0", "list_
183
```

```
184 #ifdef FWDUMP_bcmdhd
185     run_command("ND OFFLOAD TABLE", 5,
186             SU_PATH, "root", "wlutil", "nd_hostip", NULL);
187
188     run_command("DUMP WIFI INTERNAL COUNTERS (1)", 20,
189             SU_PATH, "root", "wlutil", "counters", NULL);
190
191     run_command("ND OFFLOAD STATUS (1)", 5,
192             SU_PATH, "root", "wlutil", "nd_status", NULL);
193
194 #endif
195     dump_file("INTERRUPTS (1)", "/proc/interrupts");
196
197     run_command("NETWORK DIAGNOSTICS", 10, "dumpsys", "connectivity", "--diag",
198
199 #ifdef FWDUMP_bcmdhd
200     run_command("DUMP WIFI STATUS", 20,
201             SU_PATH, "root", "dhdutil", "-i", "wlan0", "dump", NULL);
202
203     run_command("DUMP WIFI INTERNAL COUNTERS (2)", 20,
204             SU_PATH, "root", "wlutil", "counters", NULL);
205
206     run_command("ND OFFLOAD STATUS (2)", 5,
207             SU_PATH, "root", "wlutil", "nd_status", NULL);
208 #endif
209     dump_file("INTERRUPTS (2)", "/proc/interrupts");
210
211     print_properties();
212
213     run_command("VOLD DUMP", 10, "vdc", "dump", NULL);
214     run_command("SECURE CONTAINERS", 10, "vdc", "asec"
215
216     run_command("FILESYSTEMS & FREE SPACE", 10, "df", N
217
218     run_command("LAST RADIO LOG", 10, "parse_radio_log'
```

```
219
220     printf("------ BACKLIGHTS ------\n");
221     printf("LCD brightness=");
222     dump_file(NULL, "/sys/class/leds/lcd-backlight/brightness");
223     printf("Button brightness=");
224     dump_file(NULL, "/sys/class/leds/button-backlight/brightness");
225     printf("Keyboard brightness=");
226     dump_file(NULL, "/sys/class/leds/keyboard-backlight/brightness");
227     printf("ALS mode=");
228     dump_file(NULL, "/sys/class/leds/lcd-backlight/als");
229     printf("LCD driver registers:\n");
230     dump_file(NULL, "/sys/class/leds/lcd-backlight/registers");
231     printf("\n");
232
233     /* Binder state is expensive to look at as it uses a lot of memory. */
234     dump_file("BINDER FAILED TRANSACTION LOG", "/sys/kernel/debug/binder/failed
235     dump_file("BINDER TRANSACTION LOG", "/sys/kernel/debug/binder/transaction_lo ",
236     dump_file("BINDER TRANSACTIONS", "/sys/kernel/debug/binder/transactions");
237     dump_file("BINDER STATS", "/sys/kernel/debug/binder/stats");
238     dump_file("BINDER STATE", "/sys/kernel/debug/binder/state");
239
240     printf("=========================================================\n");
241     printf("== Board\n");
242     printf("=========================================
243
244     dumpstate_board();
245     printf("\n");
246
247     /* Migrate the ril_dumpstate to a dumpstate_board()? */
248     char ril_dumpstate_timeout[PROPERTY_VALUE_MAX] = {
249     property_get("ril.dumpstate.timeout", ril_dumpstate_tim
250     if (strnlen(ril_dumpstate_timeout, PROPERTY_VALUE_MA
251         if (0 == strncmp(build_type, "user", PROPERTY_VALUE_
252             // su does not exist on user builds, so try running w
253             // This way any implementations of vril-dump that
```

```
254         // root can run on user builds.
255         run_command("DUMP VENDOR RIL LOGS", atoi(ril_dumpstate_timeout),
256                 "vril-dump", NULL);
257     } else {
258         run_command("DUMP VENDOR RIL LOGS", atoi(ril_dumpstate_timeout),
259                 SU_PATH, "root", "vril-dump", NULL);
260     }
261 }
262
263 printf("========================================================\n");
264 printf("== Android Framework Services\n");
265 printf("========================================================\n");
266
267 /* the full dumpsys is starting to take a long time, so we need
268    to increase its timeout.  we really need to do the timeouts in
269    dumpsys itself... */
270 run_command("DUMPSYS", 60, "dumpsys", NULL);
271
272 printf("========================================================\n");
273 printf("== Checkins\n");
274 printf("========================================================\n");
275
276 run_command("CHECKIN BATTERYSTATS", 30, "dumpsys
277 run_command("CHECKIN MEMINFO", 30, "dumpsys", "m
278 run_command("CHECKIN NETSTATS", 30, "dumpsys", "ne
279 run_command("CHECKIN PROCSTATS", 30, "dumpsys", "
280 run_command("CHECKIN USAGESTATS", 30, "dumpsys",
281 run_command("CHECKIN PACKAGE", 30, "dumpsys", "pa
282
283 printf("========================================
284 printf("== Running Application Activities\n");
285 printf("========================================
286
287 run_command("APP ACTIVITIES", 30, "dumpsys", "activit
288
```

```
289     printf("========================================================\n");
290     printf("== Running Application Services\n");
291     printf("========================================================\n");
292
293     run_command("APP SERVICES", 30, "dumpsys", "activity", "service", "all", NULL);
294
295     printf("========================================================\n");
296     printf("== Running Application Providers\n");
297     printf("========================================================\n");
298
299     run_command("APP SERVICES", 30, "dumpsys", "activity", "provider", "all", NULL);
300
301
302     printf("========================================================\n");
303     printf("== dumpstate: done\n");
304     printf("========================================================\n");
305 }
```

上面的代码比较长，是因为所要dump的模块太多，但是基本逻辑还是比较清楚的，我们看到

来源就是：

1.系统属性

2./proc和/sys节点文件

3.执行shell命令获得相关输出

4.logcat输出

5.Android Framework Services信息基本使用dumpsys命令通过binde

这里我们需要看一下dumpsys命令的实现，这个命令也是比较简单

```
1  int main(int argc, char* const argv[])
2  {
3      signal(SIGPIPE, SIG_IGN);
4      sp<IServiceManager> sm = defaultServiceManager();
5      fflush(stdout);
6      if (sm == NULL) {
7          ALOGE("Unable to get default service manager!");
8          aerr << "dumpsys: Unable to get default service manager!" << endl;
9          return 20;
10     }
11
12     Vector<String16> services;
13     Vector<String16> args;
14     bool showListOnly = false;
15     if ((argc == 2) && (strcmp(argv[1], "-l") == 0)) {
16         showListOnly = true;
17     }
18     if ((argc == 1) || showListOnly) {
19         services = sm->listServices();
20         services.sort(sort_func);
21         args.add(String16("-a"));
22     } else {
23         services.add(String16(argv[1]));
24         for (int i=2; i<argc; i++) {
25             args.add(String16(argv[i]));
26         }
27     }
28
29     const size_t N = services.size();
30
31     if (N > 1) {
32         // first print a list of the current services
33         aout << "Currently running services:" << endl;
34
35         for (size_t i=0; i<N; i++) {
```

```
36          sp<IBinder> service = sm->checkService(services[i]);
37          if (service != NULL) {
38              aout << "  " << services[i] << endl;
39          }
40      }
41  }
42
43  if (showListOnly) {
44      return 0;
45  }
46
47  for (size_t i=0; i<N; i++) {
48      sp<IBinder> service = sm->checkService(services[i]);
49      if (service != NULL) {
50          if (N > 1) {
51              aout << "------------------------------------------------------------"
52                   "------------------" << endl;
53              aout << "DUMP OF SERVICE " << services[i] << ":" << endl;
54          }
55          int err = service->dump(STDOUT_FILENO, args);
56          if (err != 0) {
57              aerr << "Error dumping service info: (" << strerror(err)
58                   << ") " << services[i] << endl;
59          }
60      } else {
61          aerr << "Can't find service: " << services[i] << endl;
62      }
63  }
64
65  return 0;
66  }
```

我们看到它的代码逻辑就是，通过Binder的SM查找参数中的servic

```
1 int err = service->dump(STDOUT_FILENO, args);
```

这句来调用service的dump函数。

dumpstate会调用到所有binder中的service的dump函数，因为dumpstate函数执行了这一句：

```
1 /* the full dumpsys is starting to take a long time, so we need
2   to increase its timeout.  we really need to do the timeouts in
3   dumpsys itself... */
4 run_command("DUMPSYS", 60, "dumpsys", NULL);
```

直接执行dumpsys，没有参数，并且注释中也说的很清楚，就是采集所有的信息。这会执行

dump函数（执行dumpsys | grep “DUMP OF SERVICE”可以看到）：

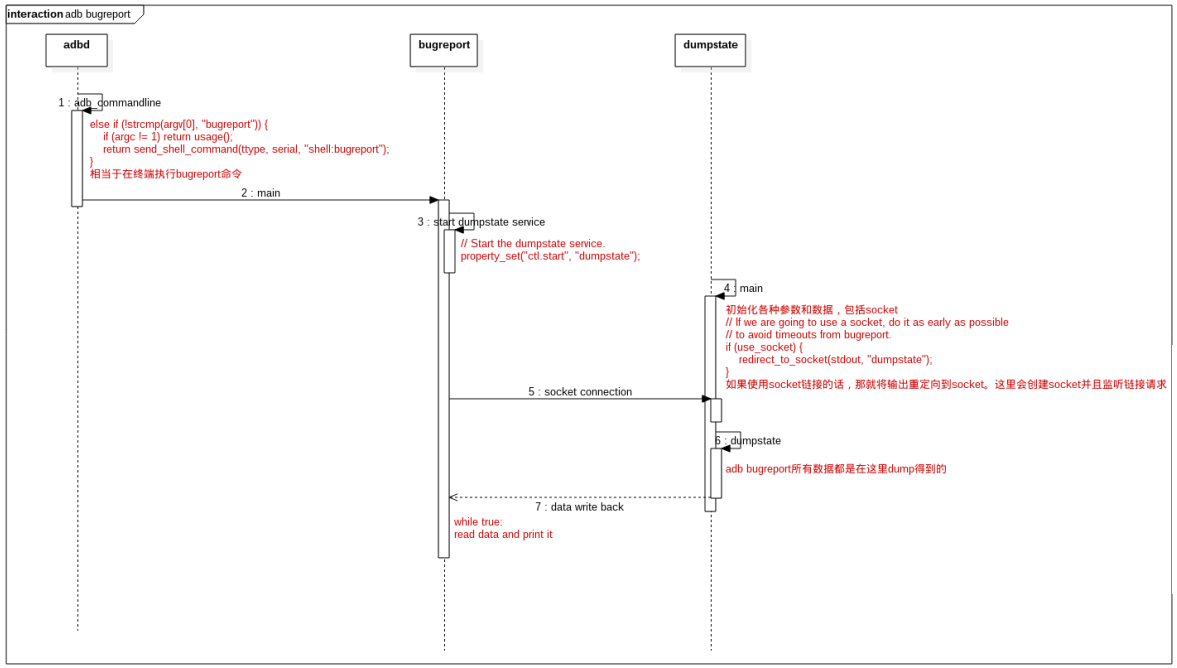
```
1  DUMP OF SERVICE DockObserver:
2  DUMP OF SERVICE SurfaceFlinger:
3  DUMP OF SERVICE accessibility:
4  DUMP OF SERVICE account:
5  DUMP OF SERVICE activity:
6  DUMP OF SERVICE alarm:
7  DUMP OF SERVICE android.security.keystore:
8  DUMP OF SERVICE android.service.gatekeeper.IGateKeepe
9  DUMP OF SERVICE appops:
10 DUMP OF SERVICE appwidget:
11 DUMP OF SERVICE assetatlas:
12 DUMP OF SERVICE audio:
13 DUMP OF SERVICE backup:
14 DUMP OF SERVICE battery:
15 DUMP OF SERVICE batteryproperties:
16 DUMP OF SERVICE batterystats:
17 DUMP OF SERVICE bluetooth_manager:
18 DUMP OF SERVICE carrier_config:
19 DUMP OF SERVICE clipboard:
```

```
20 DUMP OF SERVICE commontime_management:
21 DUMP OF SERVICE connectivity:
22 DUMP OF SERVICE consumer_ir:
23 DUMP OF SERVICE content:
24 DUMP OF SERVICE country_detector:
25 DUMP OF SERVICE cpuinfo:
26 DUMP OF SERVICE dbinfo:
27 DUMP OF SERVICE device_policy:
28 DUMP OF SERVICE deviceidle:
29 DUMP OF SERVICE devicestoragemonitor:
30 DUMP OF SERVICE diskstats:
31 DUMP OF SERVICE display:
32 DUMP OF SERVICE display.qservice:
33 DUMP OF SERVICE dreams:
34 DUMP OF SERVICE drm.drmManager:
35 DUMP OF SERVICE dropbox:
36 DUMP OF SERVICE ethernet:
37 DUMP OF SERVICE fingerprint:
38 DUMP OF SERVICE gfxinfo:
39 DUMP OF SERVICE graphicsstats:
40 DUMP OF SERVICE imms:
41 DUMP OF SERVICE input:
42 DUMP OF SERVICE input_method:
43 DUMP OF SERVICE iphonesubinfo:
44 DUMP OF SERVICE isms:
45 DUMP OF SERVICE isub:
46 DUMP OF SERVICE jobscheduler:
47 DUMP OF SERVICE launcherapps:
48 DUMP OF SERVICE location:
49 DUMP OF SERVICE lock_settings:
50 DUMP OF SERVICE media.audio_flinger:
51 DUMP OF SERVICE media.audio_policy:
52 DUMP OF SERVICE media.camera:
53 DUMP OF SERVICE media.camera.proxy:
54 DUMP OF SERVICE media.player:
```

```
55 DUMP OF SERVICE media.radio:
56 DUMP OF SERVICE media.resource_manager:
57 DUMP OF SERVICE media.sound_trigger_hw:
58 DUMP OF SERVICE media_projection:
59 DUMP OF SERVICE media_router:
60 DUMP OF SERVICE media_session:
61 DUMP OF SERVICE meminfo:
62 DUMP OF SERVICE midi:
63 DUMP OF SERVICE mount:
64 DUMP OF SERVICE netpolicy:
65 DUMP OF SERVICE netstats:
66 DUMP OF SERVICE network_management:
67 DUMP OF SERVICE network_score:
68 DUMP OF SERVICE nfc:
69 DUMP OF SERVICE notification:
70 DUMP OF SERVICE package:
71 DUMP OF SERVICE permission:
72 DUMP OF SERVICE persistent_data_block:
73 DUMP OF SERVICE phone:
74 DUMP OF SERVICE power:
75 DUMP OF SERVICE print:
76 DUMP OF SERVICE processinfo:
77 DUMP OF SERVICE procstats:
78 DUMP OF SERVICE restrictions:
79 DUMP OF SERVICE rttmanager:
80 DUMP OF SERVICE samplingprofiler:
81 DUMP OF SERVICE scheduling_policy:
82 DUMP OF SERVICE search:
83 DUMP OF SERVICE sensorservice:
84 DUMP OF SERVICE serial:
85 DUMP OF SERVICE servicediscovery:
86 DUMP OF SERVICE simphonebook:
87 DUMP OF SERVICE sip:
88 DUMP OF SERVICE statusbar:
89 DUMP OF SERVICE telecom:
```

```
90  DUMP OF SERVICE telephony.registry:
91  DUMP OF SERVICE textservices:
92  DUMP OF SERVICE trust:
93  DUMP OF SERVICE uimode:
94  DUMP OF SERVICE updatelock:
95  DUMP OF SERVICE usagestats:
96  DUMP OF SERVICE usb:
97  DUMP OF SERVICE user:
98  DUMP OF SERVICE vibrator:
99  DUMP OF SERVICE voiceinteraction:
100 DUMP OF SERVICE wallpaper:
101 DUMP OF SERVICE webviewupdate:
102 DUMP OF SERVICE wifi:
103 DUMP OF SERVICE wifip2p:
104 DUMP OF SERVICE wifiscanner:
105 DUMP OF SERVICE window:
```

这里总结以下，上面的bugreport整体逻辑如下图描述（如果图片太小看不清，请下载图片并查看）：



adb bugreport的其他选项

bugreport本身并没有什么选项，主要是通过dumpsys等命令配合完

页：https://github.com/google/battery-historian，以下是个总结：

1）. 重置电池统计信息：

```
adb shell dumpsys batterystats --reset
```

2）. Wakelock analysis全部wakelock信息：

```
adb shell dumpsys batterystats --enable full-wake-history
```

3）. Kernel trace analysis分析内核，主要分析wakeup source和wakelock activities，首先使能kernel分析：

```
$ adb root
$ adb shell

# Set the events to trace.
$ echo "power:wakeup_source_activate" >> /d/tracing/set_event
$ echo "power:wakeup_source_deactivate" >> /d/tracing/set_event

# The default trace size for most devices is 1MB, which is relatively low and might ca
# 8MB to 10MB should be a decent size for 5-6 hours of logging.

$ echo 8192 > /d/tracing/buffer_size_kb

$ echo 1 > /d/tracing/tracing_on
```

然后获得log：

```
$ echo 0 > /d/tracing/tracing_on
$ adb pull /d/tracing/trace <some path>

# Take a bug report at this time.
$ adb bugreport > bugreport.txt
```


顶 1　　踩 0

- 上一篇 Android 6.0 Overview Screen实现原理
- 下一篇 Android 6.0指纹识别App开发demo

相关文章推荐

- adb shell bugreport分析
- Python即将成为第一语言
- android bugreport. .
- 构建企业级高性能OLAP引擎--董西成
- Android bugreport工具分析和使用
- JDK9新特性解读
- Android 建立手机与手表数据同步机制总结
- 华为工程师，带你实战C++
- 聊一聊binder driver导致的系统重启问题
- Android自定义控件全知道
- adb启动程序命令：adb shell am start
- TensorFlow入门基础知识详解
- android跨进程通信（IPC）：使用AIDL
- Android 通过adb shell am broadcast发送广
- idapython import site failed
- 一键分析Android的BugReport

查看评论

暂无评论