在Linux下，ps是非常普通的命令:

ps -aux 查看所有正在运行的进程

ps | grep xxx 显示过滤指定名称的进程

可是以上的命令在Android上，都不起作用！ What？！！！Android不是基于Linux系统的吗？对，准确来说，Android是Google基于Linux的内核改造出来的。那也意味着，Android是一个Linux的修改版，这样子也解析了同样的命令，不能执行的问题了。

对于前面提到的命令，我们可以这样子的命令来替代:

ps 可以查看所有正在运行的进程

ps xxx 显示过滤指定名称的进程

和原来的Linux相比，貌似更人性化了，或者说，功能简洁了许多，毕竟Android的toolbox是为适应大多数移动设备而设计的。

接着我们一起学习一下Android的ps命令参数: -t -x -P -p -c [pid|name]

* -t 显示进程下的线程列表
* -x 显示进程耗费的用户时间和系统时间，格式:(u:0, s:0)，单位:秒(s)
* -P 显示调度策略，通常是bg或fg，当获取失败将会是un和er
* -p 显示进程的优先级和nice等级
* -c 显示进程耗费的CPU时间 (可能不兼容Android 4.0以前的老版本系统)
* [pid] 过滤指定的进程PID
* [name] 过滤指定的进程NAME

和Linux的ps不同，Linux的ps命令可以这样子：ps -aux，Android上ps的参数并不能一起使用，如ps -txPc.

小提示: 内核根据进程的nice值决定进程需要多少处理器时间. nice值的取值范围是是: -20到20. 一个具有-20 的 nice 值的进程有很高的优先级. 一个 nice 值为 20 的进程的优先级则很低.

如何查看ANDROID进程信息呢?

可以打开Adb shell，然后我们就有2种方法：

**方法1：直接输入ps命令**

输入之后，我们就可以看到如下的信息：

**[plain]** [view plaincopy](http://blog.csdn.net/tcpipstack/article/details/8541980)

1. # ps
2. ps
3. USER     PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
4. root      1     0     276    188   c0099f1c 000086e8 S /init
5. root      2     0     0      0     c004df64 00000000 S kthreadd
6. root      3     2     0      0     c003fa28 00000000 S ksoftirqd/0
7. root      4     2     0      0     c004abc0 00000000 S events/0
8. root      5     2     0      0     c004abc0 00000000 S khelper
9. root      6     2     0      0     c004abc0 00000000 S suspend
10. root      7     2     0      0     c004abc0 00000000 S kblockd/0
11. root      8     2     0      0     c004abc0 00000000 S cqueue
12. root      9     2     0      0     c01780d0 00000000 S kseriod
13. root      10    2     0      0     c004abc0 00000000 S kmmcd
14. root      11    2     0      0     c006efa8 00000000 S pdflush
15. root      12    2     0      0     c006efa8 00000000 S pdflush
16. root      13    2     0      0     c0073480 00000000 S kswapd0
17. root      14    2     0      0     c004abc0 00000000 S aio/0
18. root      22    2     0      0     c0175900 00000000 S mtdblockdc

那么我们禁不住要问:

**USER     PID   PPID  VSIZE  RSS     WCHAN    PC         NAME**

那么这些项各代表着什么意思呢？

1. **USER：  进程的当前用户；**
2. **PID   ： 毫无疑问, process ID的缩写，也就进程号；**
3. **PPID  ：process parent ID，父进程ID**
4. **VSIZE  ： virtual size，进程虚拟地址空间大小；**
5. **RSS    ： 进程正在使用的物理内存的大小；**
6. **WCHAN  ：进程如果处于休眠状态的话，在内核中的地址；**
7. **PC  ： program counter，**
8. **NAME: process name，进程的名称**

**和Linux ps命令对比**

Android的内核还是Linux的，我们可以和Linux下ps aux命令的各项参数进行对比，可见：

**[plain]** [view plaincopy](http://blog.csdn.net/tcpipstack/article/details/8541980)

1. root@long-desktop:~# ps aux
2. USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
3. root         1  0.0  0.2   2804  1684 ?        Ss   21:11   0:01 /sbin/init
4. root         2  0.0  0.0      0     0 ?        S    21:11   0:00 [kthreadd]
5. root      2286  0.0  0.4   8888  3172 ?        Ss   21:52   0:00 sshd: root@pts/0
6. root      2352  0.0  0.2   4684  2048 pts/0    Ss+  21:52   0:00 -bash
7. root      2454  0.4  0.4   8888  3164 ?        Rs   22:06   0:00 sshd: root@pts/1
8. root      2518  0.1  0.2   4684  2004 pts/1    Ss   22:06   0:00 -bash
9. root      2551  0.0  0.1   2736  1088 pts/1    R+   22:06   0:00 ps aux

在Linux下ps命令各项含义如下所示：

* %CPU 进程的cpu占用率
* %MEM 进程的内存占用率
* VSZ 进程所使用的虚存的大小
* RSS 进程使用的驻留集大小或者是实际内存的大小
* TTY 与进程关联的终端（tty）
* STAT 检查的状态：进程状态使用字符表示的，如R（running正在运行或准备运行）、S（sleeping睡眠）、I（idle空闲）、Z (僵死)、D（不可中断的睡眠，通常是I/O）、P（等待交换页）、W（换出,表示当前页面不在内存）、N（低优先级任务）T(terminate终 止)、W has no resident pages
* START （进程启动时间和日期）
* TIME ;（进程使用的总cpu时间）
* COMMAND （正在执行的命令行命令）
* NI (nice)优先级
* PRI 进程优先级编号
* PPID 父进程的进程ID（parent process id）
* SID 会话ID（session id）
* WCHAN 进程正在睡眠的内核函数名称；该函数的名称是从/root/system.map文件中获得的。
* FLAGS 与进程相关的数字标识

**android ps命令实现的源码**

android下ps命令的源码的位置：android/system/core/toolbox/ps.c，其实现如下：

**[cpp]** [view plaincopy](http://blog.csdn.net/tcpipstack/article/details/8541980)

1. **int** ps\_main(**int** argc, **char** \*\*argv)
2. {
3. DIR \*d;
4. **struct** dirent \*de;
5. **char** \*namefilter = 0;
6. **int** pidfilter = 0;
7. **int** threads = 0;
9. d = opendir("/proc");
10. **if**(d == 0) **return** -1;
12. **while**(argc > 1){
13. **if**(!strcmp(argv[1],"-t")) {
14. threads = 1;
15. } **else** **if**(!strcmp(argv[1],"-x")) {
16. display\_flags |= SHOW\_TIME;
17. } **else** **if**(!strcmp(argv[1],"-p")) {
18. display\_flags |= SHOW\_PRIO;
19. }  **else** **if**(isdigit(argv[1][0])){
20. pidfilter = atoi(argv[1]);
21. } **else** {
22. namefilter = argv[1];
23. }
24. argc--;
25. argv++;
26. }
28. printf("USER     PID   PPID  VSIZE RSS   %sWCHAN    PC         NAME\n",
29. (display\_flags&SHOW\_PRIO)?"PRIO  NICE  RTPRI SCHED ":"");
30. **while**((de = readdir(d)) != 0){
31. **if**(isdigit(de->d\_name[0])){
32. **int** pid = atoi(de->d\_name);
33. **if**(!pidfilter || (pidfilter == pid)) {
34. ps\_line(pid, 0, namefilter);
35. **if**(threads) ps\_threads(pid, namefilter);
36. }
37. }
38. }
39. closedir(d);
40. **return** 0;
41. }

我们可以得到每一行数据是如何获得的：

**[cpp]** [view plaincopy](http://blog.csdn.net/tcpipstack/article/details/8541980)

1. **static** **int** ps\_line(**int** pid, **int** tid, **char** \*namefilter)
2. {
3. **char** statline[1024];
4. **char** cmdline[1024];
5. **char** user[32];
6. **struct** stat stats;
7. **int** fd, r;
8. **char** \*ptr, \*name, \*state;
9. **int** ppid, tty;
10. unsigned wchan, rss, vss, eip;
11. unsigned utime, stime;
12. **int** prio, nice, rtprio, sched;
13. **struct** passwd \*pw;
15. sprintf(statline, "/proc/%d", pid);
16. stat(statline, &stats);
18. **if**(tid) {
19. sprintf(statline, "/proc/%d/task/%d/stat", pid, tid);
20. cmdline[0] = 0;
21. } **else** {
22. sprintf(statline, "/proc/%d/stat", pid);
23. sprintf(cmdline, "/proc/%d/cmdline", pid);
24. fd = open(cmdline, O\_RDONLY);
25. **if**(fd == 0) {
26. r = 0;
27. } **else** {
28. r = read(fd, cmdline, 1023);
29. close(fd);
30. **if**(r < 0) r = 0;
31. }
32. cmdline[r] = 0;
33. }
35. fd = open(statline, O\_RDONLY);
36. **if**(fd == 0) **return** -1;
37. r = read(fd, statline, 1023);
38. close(fd);
39. **if**(r < 0) **return** -1;
40. statline[r] = 0;
42. ptr = statline;
43. nexttok(&ptr); // skip pid
44. ptr++;          // skip "("
46. name = ptr;
47. ptr = strrchr(ptr, ')'); // Skip to \*last\* occurence of ')',
48. \*ptr++ = '\0';           // and null-terminate name.
50. ptr++;          // skip " "
51. state = nexttok(&ptr);
52. ppid = atoi(nexttok(&ptr));
53. nexttok(&ptr); // pgrp
54. nexttok(&ptr); // sid
55. tty = atoi(nexttok(&ptr));
57. nexttok(&ptr); // tpgid
58. nexttok(&ptr); // flags
59. nexttok(&ptr); // minflt
60. nexttok(&ptr); // cminflt
61. nexttok(&ptr); // majflt
62. nexttok(&ptr); // cmajflt
63. #if 1
64. utime = atoi(nexttok(&ptr));
65. stime = atoi(nexttok(&ptr));
66. #else
67. nexttok(&ptr); // utime
68. nexttok(&ptr); // stime
69. #endif
70. nexttok(&ptr); // cutime
71. nexttok(&ptr); // cstime
72. prio = atoi(nexttok(&ptr));
73. nice = atoi(nexttok(&ptr));
74. nexttok(&ptr); // threads
75. nexttok(&ptr); // itrealvalue
76. nexttok(&ptr); // starttime
77. vss = strtoul(nexttok(&ptr), 0, 10); // vsize
78. rss = strtoul(nexttok(&ptr), 0, 10); // rss
79. nexttok(&ptr); // rlim
80. nexttok(&ptr); // startcode
81. nexttok(&ptr); // endcode
82. nexttok(&ptr); // startstack
83. nexttok(&ptr); // kstkesp
84. eip = strtoul(nexttok(&ptr), 0, 10); // kstkeip
85. nexttok(&ptr); // signal
86. nexttok(&ptr); // blocked
87. nexttok(&ptr); // sigignore
88. nexttok(&ptr); // sigcatch
89. wchan = strtoul(nexttok(&ptr), 0, 10); // wchan
90. nexttok(&ptr); // nswap
91. nexttok(&ptr); // cnswap
92. nexttok(&ptr); // exit signal
93. nexttok(&ptr); // processor
94. rtprio = atoi(nexttok(&ptr)); // rt\_priority
95. sched = atoi(nexttok(&ptr)); // scheduling policy
97. tty = atoi(nexttok(&ptr));
99. **if**(tid != 0) {
100. ppid = pid;
101. pid = tid;
102. }
104. pw = getpwuid(stats.st\_uid);
105. **if**(pw == 0) {
106. sprintf(user,"%d",(**int**)stats.st\_uid);
107. } **else** {
108. strcpy(user,pw->pw\_name);
109. }
111. **if**(!namefilter || !strncmp(name, namefilter, strlen(namefilter))) {
112. printf("%-8s %-5d %-5d %-5d %-5d", user, pid, ppid, vss / 1024, rss \* 4);
113. **if**(display\_flags&SHOW\_PRIO)
114. printf(" %-5d %-5d %-5d %-5d", prio, nice, rtprio, sched);
115. printf(" %08x %08x %s %s", wchan, eip, state, cmdline[0] ? cmdline : name);
116. **if**(display\_flags&SHOW\_TIME)
117. printf(" (u:%d, s:%d)", utime, stime);
118. printf("\n");
119. }
120. **return** 0;
121. }

**方法2： 进入/proc文件夹**

在/proc文件夹下有很多对应进程ID号的子文件夹：

**[plain]** [view plaincopy](http://blog.csdn.net/tcpipstack/article/details/8541980)

1. drwxr-xr-x  23 root       root             4096 2012-10-01 02:09 ../
2. dr-xr-xr-x   7 root       root                0 2013-01-25 21:10 1/
3. dr-xr-xr-x   7 root       root                0 2013-01-25 21:10 10/
4. dr-xr-xr-x   7 root       root                0 2013-01-25 21:11 1001/
5. dr-xr-xr-x   7 root       root                0 2013-01-25 21:11 1076/
6. dr-xr-xr-x   7 root       root                0 2013-01-25 21:10 11/
7. dr-xr-xr-x   7 root       root                0 2013-01-25 21:10 12/
8. dr-xr-xr-x   7 root       root                0 2013-01-25 21:10 13/
9. dr-xr-xr-x   7 gdm        gdm                 0 2013-01-25 21:11 1345/
10. dr-xr-xr-x   7 root       root                0 2013-01-25 21:11 1353/
11. dr-xr-xr-x   7 root       root                0 2013-01-25 21:11 1375/
12. dr-xr-xr-x   7 root       root                0 2013-01-25 21:10 14/

我们可以进入对应的文件夹内，可以看到有以下信息，就可以查询到你的进程信息了。