IBM

developerWorks 中国 技术主题 Linux 文档库

Linux 技巧：让进程在后台可靠运行的几种方法

想让进程在断开连接后依然保持运行？如果该进程已经开始运行了该如何补救？ 如果有大量这类需求如何简化操作？

申毅，IBM 中国软件开发中心 WebSphere Portal 部门软件工程师。

2008 年 5 月 29 日

我们经常会碰到这样的问题，用 telnet/ssh 登录了远程的 Linux 服务器，运行了一些耗时较长的任务， 结果却由于网络的不稳定导致任务中途失败。如何让命令提交后不受本地关闭终端窗口/网络断开连接的干扰呢？下面举了一些例子， 您可以针对不同的场景选择不同的方式来处理这个问题。

nohup/setsid/&

场景：

如果只是临时有一个命令需要长时间运行，什么方法能最简便的保证它在后台稳定运行呢？

解决方法：

我们知道，当用户注销（logout）或者网络断开时，终端会收到 HUP（hangup）信号从而关闭其所有子进程。因此，我们的解决办法

hangup 名称的来由

在 Unix 的早期版本中，每个终端都会通过 modem 和系统通讯。当用户 logout 时，modem 就会挂断（hang up）电话。 同

就有两种途径：要么让进程忽略 HUP 信号，要么让进程运行在新的会话里从而成为不属于此终端的子进程。

理，当 modem 断开连接时，就会给终端发送 hangup 信号来通知其关闭所有子进程。

1. nohup

nohup 无疑是我们首先想到的办法。顾名思义，nohup 的用途就是让提交的命令忽略 hangup 信号。让我们先来看一下 nohup 的帮助信息：

```
NOHUP(1)                        User Commands                        NOHUP(1)

NAME
       nohup - run a command immune to hangups, with output to a non-tty

SYNOPSIS
       nohup COMMAND [ARG]...
       nohup OPTION

DESCRIPTION
       Run COMMAND, ignoring hangup signals.

       --help display this help and exit

       --version
              output version information and exit
```

可见，nohup 的使用是十分方便的，只需在要处理的命令前加上 nohup 即可，标准输出和标准错误缺省会被重定向到 nohup.out 文件中。一般我们可在结尾加上**"&"**来将命令同时放入后台运行，也可用"*>filename* 2>&1"来更改缺省的重定向文件名。

nohup 示例

```
[root@pvcent107 ~]# nohup ping www.ibm.com &
[1] 3059
nohup: appending output to `nohup.out'
[root@pvcent107 ~]# ps -ef |grep 3059
root      3059   984  0 21:06 pts/3    00:00:00 ping www.ibm.com
root      3067   984  0 21:06 pts/3    00:00:00 grep 3059
[root@pvcent107 ~]#
```

2。setsid

nohup 无疑能通过忽略 HUP 信号来使我们的进程避免中途被中断，但如果我们换个角度思考，如果我们的进程不属于接受 HUP 信号的终端的子进程，那么自然也就不会受到 HUP 信号的影响了。setsid 就能帮助我们做到这一点。让我们先来看一下 setsid 的帮助信息：

```
SETSID(8)                Linux Programmer's Manual                SETSID(8)

NAME
       setsid - run a program in a new session

SYNOPSIS
       setsid program [ arg ... ]

DESCRIPTION
       setsid runs a program in a new session.
```

可见 setsid 的使用也是非常方便的，也只需在要处理的命令前加上 setsid 即可。

setsid 示例

```
[root@pvcent107 ~]# setsid ping www.ibm.com
[root@pvcent107 ~]# ps -ef |grep www.ibm.com
root     31094     1  0 07:28 ?        00:00:00 ping www.ibm.com
root     31102 29217  0 07:29 pts/4    00:00:00 grep www.ibm.com
[root@pvcent107 ~]#
```

值得注意的是，上例中我们的进程 ID(PID)为31094，而它的父 ID（PPID）为1（即为 init 进程 ID），并不是当前终端的进程 ID。请将此例与nohup 例中的父 ID 做比较。

3。&

这里还有一个关于 subshell 的小技巧。我们知道，将一个或多个命名包含在“()”中就能让这些命令在子 shell 中运行中，从而扩展出很多有趣的功能，我们现在要讨论的就是其中之一。

当我们将"&"也放入“()”内之后，我们就会发现所提交的作业并不在作业列表中，也就是说，是无法通过jobs来查看的。让我们来看看为什么这样就能躲过 HUP 信号的影响吧。

subshell 示例

```
[root@pvcent107 ~]# (ping www.ibm.com &)
[root@pvcent107 ~]# ps -ef |grep www.ibm.com
root     16270     1  0 14:13 pts/4    00:00:00 ping www.ibm.com
root     16278 15362  0 14:13 pts/4    00:00:00 grep www.ibm.com
[root@pvcent107 ~]#
```

从上例中可以看出，新提交的进程的父 ID（PPID）为1（init 进程的 PID），并不是当前终端的进程 ID。因此并不属于当前终端的子进程，从而也就不会受到当前终端的 HUP 信号的影响了。

disown

场景：

我们已经知道，如果事先在命令前加上 nohup 或者 setsid 就可以避免 HUP 信号的影响。但是如果我们未加任何处理就已经提交了命令，该如何补救才能让它避免 HUP 信号的影响呢？

解决方法：

这时想加 nohup 或者 setsid 已经为时已晚，只能通过作业调度和 disown 来解决这个问题了。让我们来看一下 disown 的帮助信息：

```
disown [-ar] [-h] [jobspec ...]
        Without options, each jobspec is  removed  from  the  table  of
        active  jobs.   If  the -h option is given, each jobspec is not
        removed from the table, but is marked so  that  SIGHUP  is  not
        sent  to the job if the shell receives a SIGHUP.  If no jobspec
        is present, and neither the -a nor the -r option  is  supplied,
        the  current  job  is  used.  If no jobspec is supplied, the -a
        option means to remove or mark all jobs; the -r option  without
        a  jobspec  argument  restricts operation to running jobs.  The
        return value is 0 unless a jobspec does  not  specify  a  valid
```

```
       job.
```

可以看出，我们可以用如下方式来达成我们的目的。

用disown -h *jobspec*来使**某个作业**忽略HUP信号。

用disown -ah 来使**所有的作业**都忽略HUP信号。

用disown -rh 来使**正在运行的作业**忽略HUP信号。

需要注意的是，当使用过 disown 之后，会将把目标作业从作业列表中移除，我们将不能再使用jobs来查看它，但是依然能够用ps -ef查找到它。

灵活运用 CTRL-z

在我们的日常工作中，我们可以用 CTRL-z 来将当前进程挂起到后台暂停运行，执行一些别的操作，然后再用 fg 来将挂起的进程重新放回前台（也可用 bg 来将挂起的进程放在后台）继续运行。这样我们就可以在一个终端内灵活切换运行多个任务，这一点在调试代码时尤为有用。因为将代码编辑器挂起到后台再重新放回时，光标定位仍然停留在上次挂起时的位置，避免了重新定位的麻烦。

但是还有一个问题，这种方法的操作对象是作业，如果我们在运行命令时在结尾加了**"&"**来使它成为一个作业并在后台运行，那么就万事大吉了，我们可以通过jobs命令来得到所有作业的列表。但是如果并没有把当前命令作为作业来运行，如何才能得到它的作业号呢？答案就是用 CTRL-z（按住Ctrl键的同时按住z键）了！

CTRL-z 的用途就是将当前进程挂起（Suspend），然后我们就可以用jobs命令来查询它的作业号，再用bg *jobspec*来将它放入后台并继续运行。需要注意的是，如果挂起会影响当前进程的运行结果，请慎用此方法。

disown 示例1（如果提交命令时已经用“&”将命令放入后台运行，则可以直接使用“disown”）

```
[root@pvcent107 build]# cp -r testLargeFile largeFile &
[1] 4825
[root@pvcent107 build]# jobs
[1]+  Running                 cp -i -r testLargeFile largeFile &
[root@pvcent107 build]# disown -h %1
[root@pvcent107 build]# ps -ef |grep largeFile
```

```
root      4825   968  1 09:46 pts/4    00:00:00 cp -i -r testLargeFile largeFile
root      4853   968  0 09:46 pts/4    00:00:00 grep largeFile
[root@pvcent107 build]# logout
```

disown 示例2（如果提交命令时未使用“&”将命令放入后台运行，可使用 CTRL-z 和“bg”将其放入后台，再使用“disown”）

```
[root@pvcent107 build]# cp -r testLargeFile largeFile2

[1]+  Stopped                 cp -i -r testLargeFile largeFile2
[root@pvcent107 build]# bg %1
[1]+ cp -i -r testLargeFile largeFile2 &
[root@pvcent107 build]# jobs
[1]+  Running                 cp -i -r testLargeFile largeFile2 &
[root@pvcent107 build]# disown -h %1
[root@pvcent107 build]# ps -ef |grep largeFile2
root      5790  5577  1 10:04 pts/3    00:00:00 cp -i -r testLargeFile largeFile2
root      5824  5577  0 10:05 pts/3    00:00:00 grep largeFile2
[root@pvcent107 build]#
```

screen

场景：

我们已经知道了如何让进程免受 HUP 信号的影响，但是如果有大量这种命令需要在稳定的后台里运行，如何避免对每条命令都做这样的操作呢？

解决方法：

此时最方便的方法就是 screen 了。简单的说，screen 提供了 ANSI/VT100 的终端模拟器，使它能够在一个真实终端下运行多个全屏的伪终端。screen 的参数很多，具有很强大的功能，我们在此仅介绍其常用功能以及简要分析一下为什么使用 screen 能够避免 HUP 信号的影响。我们先看一下 screen 的帮助信息：

```
SCREEN(1)                                                       SCREEN(1)

NAME
       screen - screen manager with VT100/ANSI terminal emulation

SYNOPSIS
```

```
       screen [ -options ] [ cmd [ args ] ]
       screen -r [[pid.]tty[.host]]
       screen -r sessionowner/[[pid.]tty[.host]]

DESCRIPTION
       Screen  is  a  full-screen  window manager that multiplexes a physical
       terminal between several  processes  (typically  interactive  shells).
       Each  virtual  terminal provides the functions of a DEC VT100 terminal
       and, in addition, several control functions from the  ISO  6429  (ECMA
       48,  ANSI  X3.64)  and ISO 2022 standards (e.g. insert/delete line and
       support for multiple character sets).  There is a  scrollback  history
       buffer  for  each virtual terminal and a copy-and-paste mechanism that
       allows moving text regions between windows.
```

使用 screen 很方便，有以下几个常用选项：

用`screen -dmS` *`session name`*来建立一个处于断开模式下的会话（并指定其会话名）。

用`screen -list` 来列出所有会话。

用`screen -r` *`session name`*来重新连接指定会话。

用快捷键`CTRL-a d` 来暂时断开当前会话。

screen 示例

```
[root@pvcent107 ~]# screen -dmS Urumchi
[root@pvcent107 ~]# screen -list
There is a screen on:
        12842.Urumchi   (Detached)
1 Socket in /tmp/screens/S-root.

[root@pvcent107 ~]# screen -r Urumchi
```

当我们用“-r”连接到 screen 会话后，我们就可以在这个伪终端里面为所欲为，再也不用担心 HUP 信号会对我们的进程造成影响，也不用给每个命令前都加上“nohup”或者“setsid”了。这是为什么呢？让我来看一下下面两个例子吧。

1. 未使用 screen 时新进程的进程树

```
[root@pvcent107 ~]# ping www.google.com &
[1] 9499
[root@pvcent107 ~]# pstree -H 9499
init─┬─Xvnc
     ├─acpid
     ├─atd
     ├─2*[sendmail]
     ├─sshd─┬─sshd───bash───pstree
     │      └─sshd───bash───ping
```

我们可以看出，未使用 screen 时我们所处的 bash 是 sshd 的子进程，当 ssh 断开连接时，HUP 信号自然会影响到它下面的所有子进程（包括我们新建立的 ping 进程）。

2. 使用了 screen 后新进程的进程树

```
[root@pvcent107 ~]# screen -r Urumchi
[root@pvcent107 ~]# ping www.ibm.com &
[1] 9488
[root@pvcent107 ~]# pstree -H 9488
init─┬─Xvnc
     ├─acpid
     ├─atd
     ├─screen───bash───ping
     ├─2*[sendmail]
```

而使用了 screen 后就不同了，此时 bash 是 screen 的子进程，而 screen 是 init（PID为1）的子进程。那么当 ssh 断开连接时，HUP 信号自然不会影响到 screen 下面的子进程了。

总结

现在几种方法已经介绍完毕，我们可以根据不同的场景来选择不同的方案。nohup/setsid 无疑是临时需要时最方便的方法，disown 能帮助我们来事后补救当前已经在运行了的作业，而 screen 则是在大批量操作时不二的选择了。

参考资料

“系统管理员工具包：进程管理技巧”（developerWorks 中国，2006 年 5 月）介绍了 Linux 进程管理的更多技巧。

“Linux 技巧：使用 screen 管理你的远程会话”（developerWorks 中国，2007 年 7 月）介绍了 screen 的更多技巧。

在 developerWorks 中国网站 Linux 专区中学习更多 Linux 方面的知识。