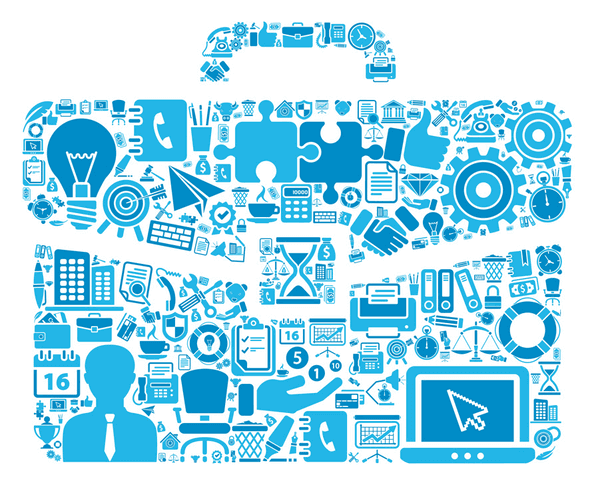
# Linux 守护进程的启动方法

["守护进程"](http://baike.baidu.com/view/53123.htm)（daemon）就是一直在后台运行的进程（daemon）。

本文介绍如何将一个 Web 应用，启动为守护进程。



一、问题的由来

Web应用写好后，下一件事就是启动，让它一直在后台运行。

这并不容易。举例来说，下面是一个最简单的Node应用server.js，只有6行。

var http = require('http');

http.createServer(function(req, res) {

res.writeHead(200, {'Content-Type': 'text/plain'});

res.end('Hello World');

}).listen(5000);

你在命令行下启动它。

$ node server.js

看上去一切正常，所有人都能快乐地访问 5000 端口了。但是，一旦你退出命令行窗口，这个应用就一起退出了，无法访问了。

怎么才能让它变成系统的守护进程（daemon），成为一种服务（service），一直在那里运行呢？

二、前台任务与后台任务

上面这样启动的脚本，称为"前台任务"（foreground job）。它会独占命令行窗口，只有运行完了或者手动中止，才能执行其他命令。

变成守护进程的第一步，就是把它改成"后台任务"（background job）。

$ node server.js &

只要在命令的尾部加上符号&，启动的进程就会成为"后台任务"。如果要让正在运行的"前台任务"变为"后台任务"，可以先按ctrl + z，然后执行bg命令（让最近一个暂停的"后台任务"继续执行）。

"后台任务"有两个特点。

1. 继承当前 session （对话）的标准输出（stdout）和标准错误（stderr）。因此，后台任务的所有输出依然会同步地在命令行下显示。
2. 不再继承当前 session 的标准输入（stdin）。你无法向这个任务输入指令了。如果它试图读取标准输入，就会暂停执行（halt）。

可以看到，"后台任务"与"前台任务"的本质区别只有一个：是否继承标准输入。所以，执行后台任务的同时，用户还可以输入其他命令。

三、SIGHUP信号

变为"后台任务"后，一个进程是否就成为了守护进程呢？或者说，用户退出 session 以后，"后台任务"是否还会继续执行？

Linux系统是这样设计的。

1. 用户准备退出 session
2. 系统向该 session 发出SIGHUP信号
3. session 将SIGHUP信号发给所有子进程
4. 子进程收到SIGHUP信号后，自动退出

上面的流程解释了，为什么"前台任务"会随着 session 的退出而退出：因为它收到了SIGHUP信号。

那么，"后台任务"是否也会收到SIGHUP信号？

这由 Shell 的huponexit参数决定的。

$ shopt | grep huponexit

执行上面的命令，就会看到huponexit参数的值。

大多数Linux系统，这个参数默认关闭（off）。因此，session 退出的时候，不会把SIGHUP信号发给"后台任务"。所以，一般来说，"后台任务"不会随着 session 一起退出。

四、disown 命令

通过"后台任务"启动"守护进程"并不保险，因为有的系统的huponexit参数可能是打开的（on）。

更保险的方法是使用disown命令。它可以将指定任务从"后台任务"列表（jobs命令的返回结果）之中移除。一个"后台任务"只要不在这个列表之中，session 就肯定不会向它发出SIGHUP信号。

$ node server.js &

$ disown

执行上面的命令以后，server.js进程就被移出了"后台任务"列表。你可以执行jobs命令验证，输出结果里面，不会有这个进程。

disown的用法如下。

# 移出最近一个正在执行的后台任务

$ disown

# 移出所有正在执行的后台任务

$ disown -r

# 移出所有后台任务

$ disown -a

# 不移出后台任务，但是让它们不会收到SIGHUP信号

$ disown -h

# 根据jobId，移出指定的后台任务

$ disown %2

$ disown -h %2

五、标准 I/O

使用disown命令之后，还有一个问题。那就是，退出 session 以后，如果后台进程与标准I/O有交互，它还是会挂掉。

还是以上面的脚本为例，现在加入一行。

var http = require('http');

http.createServer(function(req, res) {

console.log('server starts...'); // 加入此行

res.writeHead(200, {'Content-Type': 'text/plain'});

res.end('Hello World');

}).listen(5000);

启动上面的脚本，然后再执行disown命令。

$ node server.js &

$ disown

接着，你退出 session，访问5000端口，就会发现连不上。

这是因为"后台任务"的标准 I/O 继承自当前 session，disown命令并没有改变这一点。一旦"后台任务"读写标准 I/O，就会发现它已经不存在了，所以就报错终止执行。

为了解决这个问题，需要对"后台任务"的标准 I/O 进行重定向。

$ node server.js > stdout.txt 2> stderr.txt < /dev/null &

$ disown

上面这样执行，基本上就没有问题了。

六、nohup 命令

还有比disown更方便的命令，就是nohub。

$ nohup node server.js &

nohup命令对server.js进程做了三件事。

* 阻止SIGHUP信号发到这个进程。
* 关闭标准输入。该进程不再能够接收任何输入，即使运行在前台。
* 重定向标准输出和标准错误到文件nohup.out。

也就是说，nohup命令实际上将子进程与它所在的 session 分离了。

注意，nohup命令不会自动把进程变为"后台任务"，所以必须加上&符号。

七、Screen 命令与 Tmux 命令

另一种思路是使用 terminal multiplexer （终端复用器：在同一个终端里面，管理多个session），典型的就是[Screen](https://www.gnu.org/software/screen/" \t "_blank) 命令和 [Tmux](https://tmux.github.io/" \t "_blank) 命令。

它们可以在当前 session 里面，新建另一个 session。这样的话，当前 session 一旦结束，不影响其他 session。而且，以后重新登录，还可以再连上早先新建的 session。

Screen 的用法如下。

# 新建一个 session

$ screen

$ node server.js

然后，按下ctrl + A和ctrl + D，回到原来的 session，从那里退出登录。下次登录时，再切回去。

$ screen -r

如果新建多个后台 session，就需要为它们指定名字。

$ screen -S name

# 切回指定 session

$ screen -r name

$ screen -r pid\_number

# 列出所有 session

$ screen -ls

如果要停掉某个 session，可以先切回它，然后按下ctrl + c和ctrl + d。

Tmux 比 Screen 功能更多、更强大，它的基本用法如下。

$ tmux

$ node server.js

# 返回原来的session

$ tmux detach

除了tmux detach，另一种方法是按下Ctrl + B和d ，也可以回到原来的 session。

# 下次登录时，返回后台正在运行服务session

$ tmux attach

如果新建多个 session，就需要为每个 session 指定名字。

# 新建 session

$ tmux new -s session\_name

# 切换到指定 session

$ tmux attach -t session\_name

# 列出所有 session

$ tmux list-sessions

# 退出当前 session，返回前一个 session

$ tmux detach

# 杀死指定 session

$ tmux kill-session -t session-name

八、Node 工具

对于 Node 应用来说，可以不用上面的方法，有一些专门用来启动的工具：[forever](https://github.com/foreverjs/forever" \t "_blank)，[nodemon](http://nodemon.io/" \t "_blank) 和 [pm2](http://pm2.keymetrics.io/)。

forever 的功能很简单，就是保证进程退出时，应用会自动重启。

# 作为前台任务启动

$ forever server.js

# 作为服务进程启动

$ forever start app.js

# 停止服务进程

$ forever stop Id

# 重启服务进程

$ forever restart Id

# 监视当前目录的文件变动，一有变动就重启

$ forever -w server.js

# -m 参数指定最多重启次数

$ forever -m 5 server.js

# 列出所有进程

$ forever list

nodemon一般只在开发时使用，它最大的长处在于 watch 功能，一旦文件发生变化，就自动重启进程。

# 默认监视当前目录的文件变化

$ nodemon server.js

＃ 监视指定文件的变化

$ nodemon --watch app --watch libs server.js

pm2 的功能最强大，除了重启进程以外，还能实时收集日志和监控。

# 启动应用

$ pm2 start app.js

# 指定同时起多少个进程（由CPU核心数决定），组成一个集群

$ pm2 start app.js -i max

# 列出所有任务

$ pm2 list

# 停止指定任务

$ pm2 stop 0

＃ 重启指定任务

$ pm2 restart 0

# 删除指定任务

$ pm2 delete 0

# 保存当前的所有任务，以后可以恢复

$ pm2 save

# 列出每个进程的统计数据

$ pm2 monit

# 查看所有日志

$ pm2 logs

# 导出数据

$ pm2 dump

# 重启所有进程

$ pm2 kill

$ pm2 resurect

# 启动web界面 [http://localhost](http://localhost/):9615

$ pm2 web

十、Systemd

除了专用工具以外，Linux系统有自己的守护进程管理工具 Systemd 。它是操作系统的一部分，直接与内核交互，性能出色，功能极其强大。我们完全可以将程序交给 Systemd ，让系统统一管理，成为真正意义上的系统服务。

下一篇文章，我就来介绍 Systemd。