Nginx Process Model

# 概述

Nginx采用Master/Worker进程模型。Master管理Worker(s)，推荐采用一个CPU核对应一个Worker，以减少进程或线程切换开销，对于读磁盘文件多的应用，则推荐一个CPU核对应2个Worker，以便利用IO等待的时间。相对应于Apache的 “[hybrid multi-threaded multi-process](http://httpd.apache.org/docs/2.2/mod/worker.html)”，Nginx的进程模型是“single threaded multi-process”。进程模型的差异是Nginx比Apache快的一个原因。得到高性能的同时，牺牲了开发的简单性，增加了业务状态机的复杂性，因为单线程的模型中不能有堵塞的操作(读磁盘文件操作目前还是堵塞的，以后可能会改为非堵塞的)。

# 框架

# architecture.png

# 数据结构

# data_structure.png

## cycle

全局变量ngx\_cycle主要是保存了配置数据、内存管理池、监听Sockets列表等数据。

## process

全局变量ngx\_processes是一个ngx\_process\_t的数组（1024个），用以管理wokers。

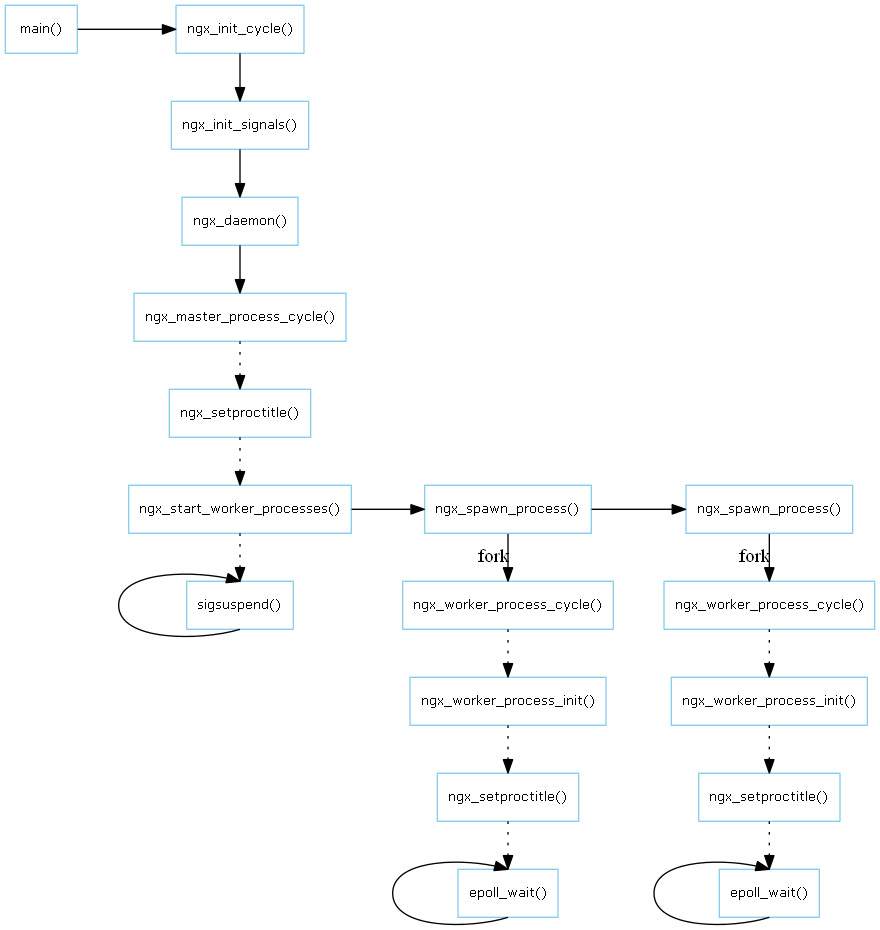
全局变量ngx\_last\_process记录ngx\_processes数组已用的位置。

## ngx\_process\_t

ngx\_process\_t 是一个关键的结构体，它抽象了一个Worker进程:

* pid : worker的进程号
* proc/data : worker 运行的函数和参数，它通常是ngx\_worker\_process\_cycle(ngx\_cycle)
* RESPAWN : 表示如果worker 退出，需要重新创建它
* JUST\_SPAWN : 表示 worker 刚被创建，不接收Master发的信号
* EXITED : 表示 worker已经退出
* EXITING : 表示 worker正在退出
* DETACHED : 用于upgrade

# 启动过程



## ngx\_init\_cycle()

读取配置数据到ngx\_cycle中，并打开Listen Sockets

## ngx\_init\_signals()

安装信号处理函数

## ngx\_daemon()

fork后台进程，标准输入输出定向到/dev/null

## ngx\_master\_process\_cycle()

Master进程的主循环，先创建Wokers，再进入等待信号/处理信号的循环中

## ngx\_setproctitle()

设置进程的title，ps 命令显示

## ngx\_start\_worker\_processes()

创建若干个Worker，Worker个数可通过worker\_processes项配置

## ngx\_spawn\_process()

从全局变量数组ngx\_processes中找一个空位，通过系统调用fork出一个Worker

## ngx\_worker\_process\_cycle()

Worker进程的主循环（等待事件/处理事件）

## ngx\_worker\_process\_init()

初始化Worker进程，设置资源限制、用户和组，绑定CPU

# 情境分析

## Startup

[lingjf@dell:/usr/local/nginx$sudo ./nginx](mailto:lingjf@dell:/usr/local/nginx$./nginx)

PID PPID USER COMMAND

33126 1 root nginx: master process /usr/local/nginx

33127 33126 nobody nginx: worker process

33128 33126 nobody nginx: worker process

## Graceful Shutdown

[lingjf@dell:/usr/local/nginx$sudo ./nginx](mailto:lingjf@dell:/usr/local/nginx$./nginx) -s quit

1. 新创建的nginx进程（Signaller）从pid文件(nginx.pid)中获得Master进程号。
2. Signaller通过系统调用kill向Master发送SIGQUIT（NGX\_SHUTDOWN\_SIGNAL）信号，并退出。
3. Master收到SIGQUIT后向所有的Worker通过kill发SIGQUIT。
4. Worker收到信号后关闭资源（Sockets、Pool等）退出(exit)。
5. OS向Master发送SIGCHLD。
6. Master收到SIGCHLD后调用系统调用waitpid()获得Worker的pid和状态并更新ngx\_processes数组(信号处理函数中)。
7. Master检查是否所有的Worker都退出，如果还有没退出，则在sigsuspend()继续等待。
8. Master删除pid文件，清理资源并退出。

main()🡪ngx\_signal\_process()🡪ngx\_os\_signal\_process()🡪kill()🡪ngx\_signal\_handler()🡪ngx\_master\_process\_cycle()

🡪ngx\_signal\_worker\_processes()🡪kill()🡪ngx\_signal\_handler()🡪ngx\_worker\_process\_cycle()🡪ngx\_worker\_process\_exit()

🡪ngx\_signal\_handler()🡪ngx\_process\_get\_status()🡪waitpid()🡪ngx\_master\_process\_cycle()🡪ngx\_master\_process\_exit()

## Fast Shutdown

[lingjf@dell:/usr/local/nginx$sudo ./nginx](mailto:lingjf@dell:/usr/local/nginx$./nginx) -s stop

过程基本可Graceful Shutdown相同，不同点有：

* 发送的信号是SIGTERMs (NGX\_TERMINATE\_SIGNAL)。
* Master等待超时后向Worker发SIGKILL，强制Worker退出。

## Worker Exception

Worker异常退出后，OS会向Master发送SIGCHLD信号。Master收到信号后会调用waitpid()获得Worker的pid和状态并更新ngx\_processes数组。再根据ngx\_processes重新创建Worker。

ngx\_signal\_handler()🡪ngx\_worker\_process\_cycle()🡪ngx\_reap\_children()🡪ngx\_spawn\_process()

## Change configuration

[lingjf@dell:/usr/local/nginx$sudo ./nginx](mailto:lingjf@dell:/usr/local/nginx$./nginx) -s reload

1. 新创建的nginx进程（Signaller）从pid文件(nginx.pid)中获得Master进程号。
2. Signaller通过系统调用kill向Master发送SIGHUP（NGX\_RECONFIGURE\_SIGNAL）信号，并退出。
3. Master收到信号后根据新的配置生成新的Cycle。
4. 创建若干个新的Worker，此时新老Worker并存。
5. Master向老Workers发送SIGQUIT(NGX\_SHUTDOWN\_SIGNAL)。
6. 老Workers退出。

## Rotate log files

[lingjf@dell:/usr/local/nginx$sudo ./nginx](mailto:lingjf@dell:/usr/local/nginx$./nginx) -s reopen

略

## Upgrade binary on fly

lingjf@dell:/usr/local/nginx$mv nginx nginx\_old

lingjf@dell:/usr/local/nginx$cp nginx\_new nginx

lingjf@dell:/usr/local/nginx$sudo kill -USR2 `cat logs/nginx.pid`

lingjf@dell:/usr/local/nginx$sudo kill -QUIT `cat logs/nginx.pid.oldbin`

1. 使用新的二进制文件替换老的文件。Windows不可以替换正在运行的文件，但Linux是可以的。http://stackoverflow.com/questions/1712033/replacing-a-running-executable-in-linux
2. 向老Master发送SIGUSR2(NGX\_CHANGEBIN\_SIGNAL)信号。
3. 老Master收到信号后，把Listening Sockets fd 存入环境变量NGX\_VAR。
4. 老Master把pid文件更名为nginx.pid.oldbin。
5. 老Master通过系统调用execve() 运行新的二进制文件。
6. 新Master解析环境变量NGX\_VAR，并继承Listening Sockets。
7. 新二进制文件启动完成。此时有新老Master和新老Workers以及新老pid文件。
8. 向老Master发送SIGQUIT信号。
9. 老Master/Workers退出。

# Prototype

<https://github.com/lingjf/nginx_process>