Nginx 信号控制

在生产环境中，Nginx 一般采用的是一个 master 进程，多个 worker 进程的模型，其中 master 进程不需要处理网络事件，它不负责业务的执行，只是通过管理 worker 进程来实现重启服务、平滑升级、更换日志文件、配置文件实时生效等功能，而 worker 进程则是用来提供服务，如静态文件服务、反向代理等功能。那么 Nginx 中是如何实现 master 的重启服务、平滑升级、更换日志文件以及配置文件实时生效的呢？另外，master 进程是如何将重启服务、平滑升级、更换日志文件以及配置文件实时生效等告知 worker 进程的呢？答案是信号。下面将结合代码来看看 Nginx 是如何实现上述功能的。

因为 Nginx 是利用信号来实现平滑升级、更换日志文件、配置文件实时生效、重启服务等功能的，所以在 Nginx 的启动过程中会向操作系统内核注册所使用到的信号，其代码实现如下：

```
int ngx_cdecl
main(int argc, char *const *argv)
{
    ……
    /*初始化信号*/
     if (ngx_init_signals(cycle->log) != NGX_OK) {
          return 1;
     }
    ……
}
ngx_init_signals()实现如下：
ngx_int_t
ngx_init_signals(ngx_log_t *log)
{
    ngx_signal_t          *sig;
    struct sigaction      sa;    //Linux 内核使用的信号

    /*遍历 signals 数组，向内核注册所有 nginx 支持的信号*/
    for (sig = signals; sig->signo != 0; sig++) {
        ngx_memzero(&sa, sizeof(struct sigaction));
        sa.sa_handler = sig->handler;    //设置信号发生时的处理函数
        sigemptyset(&sa.sa_mask);
        //向内核注册信号的回调方法
        if (sigaction(sig->signo, &sa, NULL) == -1) {
            ……
        }
    }
    return NGX_OK;
}
```

从 ngx_init_signals()函数的实现中我们可以看到，Nginx 通过系统调用 sigaction()将所支持的信号注册到操作系统内核，当内核捕捉到对应的信号时，就会调用回调函数进行信号处理。从代码中可以看到，其实 Nginx 支持的所有信号对应的处理函数都是同一个，即 ngx_signal_hanlder()。在这个函数中，会根据锁发生的信号，将 Nginx 中对应于该信号的一个全局变量置位，然后在 master 进程的处理循环中将会依据这个全局变量进行相应的动作，这部分的具体实现稍后介绍。

一般来说当环境中已经有 Nginx 进程（包括 master 进程、worker 进程）在运行了，才有所谓的平滑升级、更换日志文件或者配置文件实时生效等管理功能。那么 Nginx 是如何通过命令行控制这些功能的呢？Nginx 的做法是启动一个新的 Nginx 进程，来向环境中已经存在的 master 进程发送相应的控制信号，从而实现让已有服务执行相应的动作。那么新的 Nginx 进程是如何向环境中已有的 master 进程发送信号的呢？其代码实现如下：

```
int ngx_cdecl
main(int argc, char *const *argv)
{
    ……
    /*"nginx -s xxx"*/
     if (ngx_signal) {
          return ngx_signal_process(cycle, ngx_signal);
     }
    ……
}
```

从代码实现上可以看到新的 Nginx 进程执行完这个之后就返回退出了，因为这个新启动的进程就是用来发送信号的。那新的进程是如何向已有 master 进程发送信号的呢？答案是通过 kill 系统调用。总的来说也是分两步：首先是获取已运行 master 进程的存放在 nginx.pid 文件中的 pid，也就是 ngx_signal_process()函数的功能，其实现如下：

```
ngx_int_t
ngx_signal_process(ngx_cycle_t *cycle, char *sig)
{
    ssize_t                n;
    ngx_pid_t              pid;
    ngx_file_t             file;
    ngx_core_conf_t    *ccf;
    u_char                 buf[NGX_INT64_LEN + 2];

    /*获取核心模块存储配置项结构体指针*/
    ccf = (ngx_core_conf_t *) ngx_get_conf(cycle->conf_ctx, ngx_core_module);
    ngx_memzero(&file, sizeof(ngx_file_t));
```

```
        file.name = ccf->pid;    //ccf->pid 为 nginx.pid 文件
        file.log = cycle->log;
        /*以可读方式打开 nginx.pid 文件*/
        file.fd = ngx_open_file(file.name.data, NGX_FILE_RDONLY,
                                                NGX_FILE_OPEN,
       NGX_FILE_DEFAULT_ACCESS);
       ……
        /*读文件*/
        n = ngx_read_file(&file, buf, NGX_INT64_LEN + 2, 0);

        if (ngx_close_file(file.fd) == NGX_FILE_ERROR) {
              ……
        }
        if (n == NGX_ERROR) {
              return 1;
        }
        /*去掉结尾的控制字符*/
        while (n-- && (buf[n] == CR || buf[n] == LF)) { /* void */ }

        /*将字符串转换为数字，获取 master 进程 pid*/
        pid = ngx_atoi(buf, ++n);
       ……
        /*封装 kill 系统调用的信号发送函数*/
        return ngx_os_signal_process(cycle, sig, pid);
    }
```

其次，在获取到已运行 master 进程的 pid 后，调用 kill 命令将新的 Nginx 进程携带的信号发送给已运行 master 进程，这也正是 ngx_os_signal_process()函数的功能，其实现如下：

```
    ngx_int_t
    ngx_os_signal_process(ngx_cycle_t *cycle, char *name, ngx_pid_t pid)
    {
        ngx_signal_t    *sig;

        /*
         *遍历 nginx 内核支持的信号，找到与 name 相同的信号，通过 kill 系统
         *调用向 master 进程发送信号
         */
        for (sig = signals; sig->signo != 0; sig++) {
            if (ngx_strcmp(name, sig->name) == 0) {
                if (kill(pid, sig->signo) != -1) {
                    return 0;
                }
                ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                              "kill(%P, %d) failed", pid, sig->signo);
```

```
            }
        }
        return 1;
    }
```

这个时候新的 Nginx 进程的任务就完成了，然后就返回退出了，那么接下来已运行的 master 进程是如何处理的呢？这个就涉及到 master 进程的工作循环了。我们知道 master 进程是不对外提供服务的，而是专门用来管理 worker 进程的，那 Nginx 中是如何实现的呢？前面提到了，Nginx 中是通过新启动一个进程来给 master 进程发送信号的，所以 master 进程的工作循环中就是在等待信号的到来。信号到来之后，会触发信号处理函数，然后将信号对应的标志位置位，然后在 master 进程中就检测相应的标志位来进行相应的处理。在介绍 master 对于具体信号是如何处理之前看下 master 进程对哪些信号感兴趣，列举如下：

信号	信号对应的全局标志位变量	意义
QUIT	ngx_quit	优雅地关闭整个服务
TERM（INT）	ngx_terminate	强制关闭整个服务
USR1	ngx_reopen	重新打开文件
WINCH	ngx_noaccept	所有子进程不再接受处理新的连接
USR2	ngx_change_binary	平滑升级
HUP	ngx_reconfigure	配置文件实时生效（热启动）
CHLD	ngx_reap	有子进程结束，监控所有子进程

上面表格中的 CHLD 信号并不是有新的 Nginx 进程发送的，而是操作系统内核在检测到有子进程退出时会向父进程，即 master 发送一个 CHLD 信号，然后 master 进程在对此做进一步分析，亦即 ngx_reap_children()的功能。在 ngx_master_process_cycle()中我们可以看到，master 首先将其感兴趣的信号加入到了阻塞自己的信号集合中（通过 SIG_BLOCK 调用 sigprocmask()），在完成这些动作之后会调用 sigsuspend()将自己挂起，等待信号集合中的信号发生，将自己唤醒，然后依据信号发生后置位的全局变量（见上表）做相应的处理，其处理流程图如下：

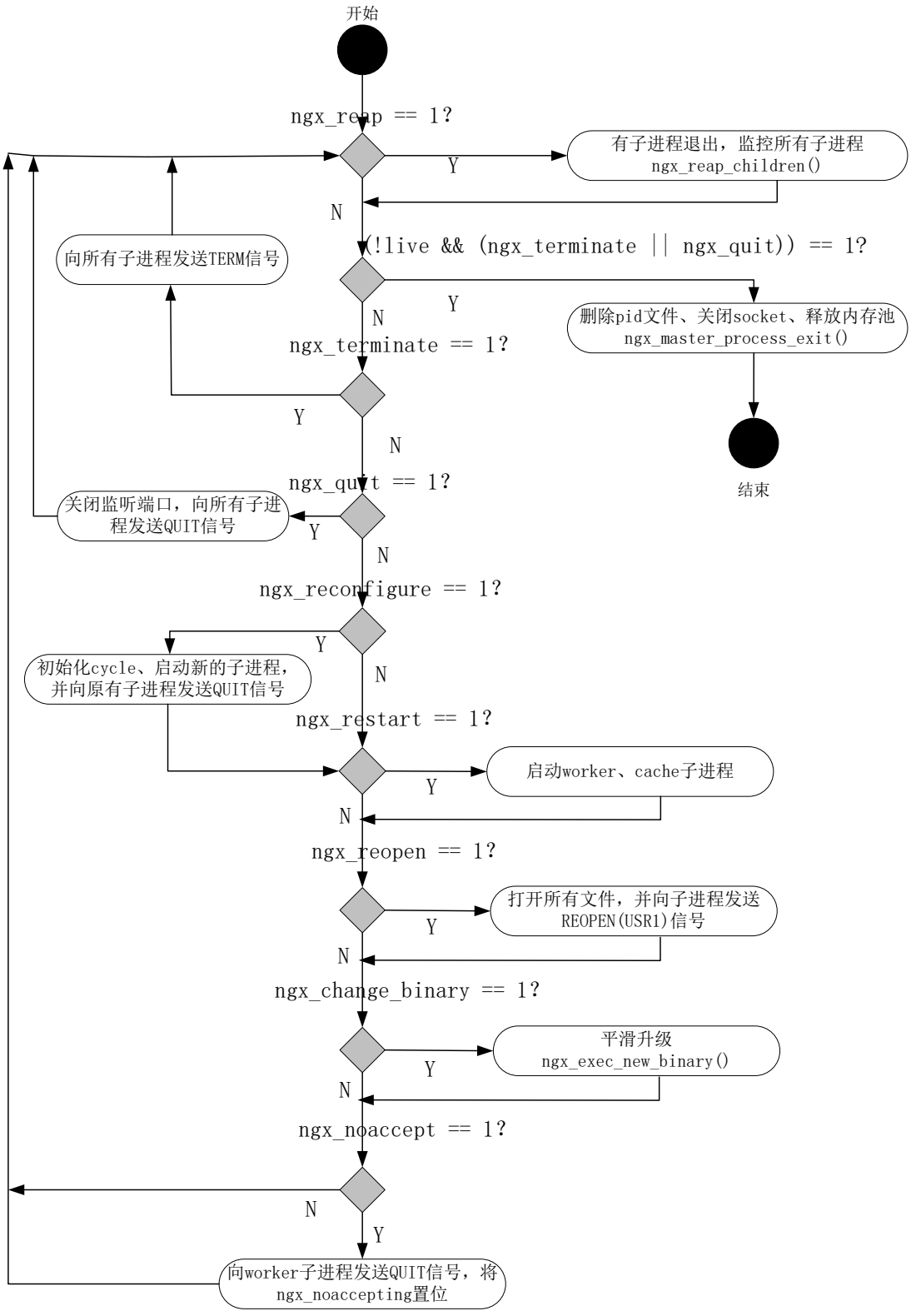

图 1 master 进程工作循环

从 master 进程的工作循环中我们可以看到，当 master 收到相应的信号，并做完自己处理流程后，会通过 ngx_signal_worker_processes()向 worker 进程发送相应的信号。举个例子，比如在收到了 QUIT 信号之后，master 会向所有的 worker

子进程也发送 QUIT 信号，以通知 worker 要优雅地退出（所谓的优雅，其实就是处理完现有连接，不再接受新的连接），并关闭监听的 socket 句柄。那么 worker 进程会对那些信号感兴趣，以及在收到 master 发送的相应信号后是如何处理的呢？这个就是 worker 工作循环的内容了。在 worker 进程中，收到了 master 发送的信号后，也会将在介绍 worker 工作循环之前，先来看下 worker 对那些信号感兴趣以及会将那些对应的全局变量置位，列举如下：

信号	信号对应的全局标志位变量	意义
QUIT	ngx_quit	优雅地关闭整个服务
TERM（INT）	ngx_terminate	强制关闭整个服务
USR1	ngx_reopen	重新打开文件

除了上面介绍的三个信号，在 worker 进程的工作循环中还可以看到另一个全局变量 ngx_exciting。这个标志位只有一个地方会设置它，也就是在收到 QUIT 信号后。ngx_quit 只有在首次设置为 1 时，才会将 ngx_exciting 置位。为什么呢？因为当 worker 进程收到 QUIT 信号后，它会知道自己需要优雅地关闭进程，即处理完现有连接并且不再接受新的连接，所以 ngx_exciting 表示的是一种正在退出的状态，即还有连接没有处理完。下面就是 worker 进程的工作流程：

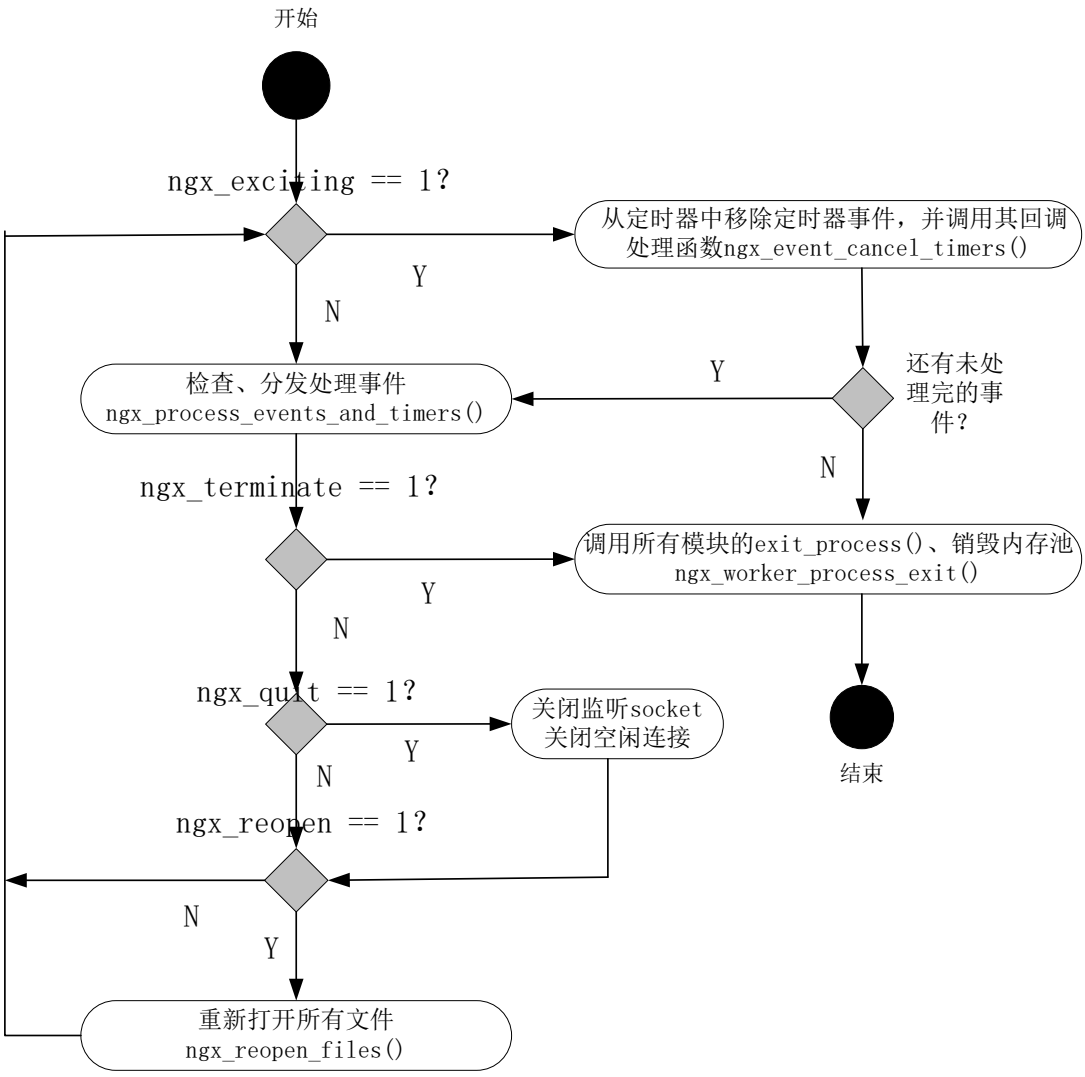

图 2 worker 进程工作循环

这里只是简要的说明了 master 进程和 worker 进程的信号处理流程，对于详细的处理流程，比如平滑升级、配置文件实时生效等，还是要通过阅读代码才能更好的理清细节。