

第 12 章

Nginx 的核心模块

在第 4 部分，我们主要介绍 Nginx 的各类模块。Nginx 的模块分为核心模块、标准 HTTP 模块、可选 HTTP 模块、邮件模块、第三方模块和补丁。

其中，Nginx 的核心模块包括主模块和事件模块两部分。

12.1 主模块指令

Nginx 的主模块是实现 Nginx 的基本功能的指令集，它们一般写在 Nginx 的配置文件的最上方。下面，我们来介绍 Nginx 主模块中包含的指令。

12.1.1 daemon 指令

语法：daemon on | off

默认值：on

示例：daemon off;

在生产环境中，请不要使用 daemon 和 master_process 指令。这些选项仅用于开发调试。当然，你也可以在生产环境中设置 daemon off，然后使用进程管理工具启动 Nginx，但是，平滑重启、升级等功能将无法使用。master_process off 绝对不应该用于生产环境。

12.1.2 env 指令

语法: `env VARIVAR=VALUE`

默认值: TZ

使用环境: main

此项指令用来定义变量集合，以下场合须更改环境变量，或者添加新的环境变量：

- 在零停机情况下平滑升级 Nginx 时；
- 启用 Nginx 内置 Perl 模块时使用；
- 被 Nginx 进程所使用。

如果没有明确定义 TZ 的值，默认情况下它集成老版本的值，且默认情况下内置的 Perl 模块总是可以使用 TZ 值的。

例如：

```
env MALLOC_OPTIONS;  
env PERL5LIB=/data/site/modules;  
env OPENSSL_ALLOW_PROXY_CERTS=1;
```

12.1.3 debug_points 指令

语法: `debug_points [stop | abort]`

默认值: none

示例: `debug_points stop;`

用于调试，在调试器内设置断点。

12.1.4 error_log 指令

语法: `error_log file [debug | info | notice | warn | error | crit]`

默认值: `${prefix}/logs/error.log`

示例: `debug_points stop;`

file 参数用来指定记录 Nginx 及 FastCGI 错误日志的文件路径。错误日志记录了服务器运行期间遇到的各种错误，以及一些普通的诊断信息，我们可以设置日志文件记录信息级别的高低，

控制日志文件记录信息的数量和类型。通常情况下, Nginx 分为 6 个错误级别, 其中 debug 级别最低, 记录的错误日志数量最多, 范围最广; 而 crit 级别最高, 只记录非常严重的错误, 一般在生产环境中使用。

日志中默认的错误级别:

- main 部分: error
- HTTP 部分: crit
- server 部分: crit

Nginx 支持将不同虚拟主机的日志存储在不同的位置, 这是个很有特色的功能。在 lighttpd 中, 它们一直拒绝提供类似的功能。代码 12-1 为针对不同虚拟主机提供不同日志的示例:

代码 12-1

```
error_log logs/main_error.log;

events {
    worker_connections 51200;
}

http {
    error_log logs/http_error.log error;
    server {
        server_name one.org;
        access_log logs/one.access;
        error_log logs/one.error error;
    }

    server {
        server_name two.org;
        access_log logs/two.access;
        error_log logs/two.error error;
    }
}
```

如果你在编译 Nginx 的时候, 使用了 --with-debug 指令, 则还可以使用:

```
error_log LOGFILE [debug_core | debug_alloc | debug_mutex | debug_event | debug_http
| debug_imap];
```

注意: error_log off 无法禁用日志, 这种写法将会创建一个名为 off 的日志文件。如果要禁用日志, 请用下面的写法:

```
error_log /dev/null crit;
```

12.1.5 log_not_found 指令

语法: `log_not_found on | off`

默认值: on

使用环境: `debug_points stop`;

启用或禁用 404 错误日志, 这个指令可以用来禁止 Nginx 记录找不到 `robots.txt` 和 `favicon.ico` 这类文件的错误信息。

示例:

```
location = /robots.txt {  
    log_not_found off;  
}
```

12.1.6 include 指令

语法: `include file | *`

默认值: 无

使用此指令, 可以包含任何你想要包含的配置文件。从 0.4.4 开始, `include` 指令开始支持文件名匹配, 例如:

```
include vhosts/*.conf;
```

注意: 直到 0.6.7 版本为止, `include` 文件的路径是相对于 `configure` 时由 `--prefix=<PATH>` 指令指定的路径而言的, 默认情况下是 `/usr/local/nginx`。如果在编译 `compiledNginx` 时你没有指定这个值, 请使用绝对路径。

从 0.6.7 开始, `include` 文件的路径实现归于 Nginx 配置文件 `nginx.conf` 的所在目录, 不再是 Nginx 编译时指定的路径。这个改进大大增加了 `include` 的灵活性。

12.1.7 lock_file 指令

语法: `lock_file file`

默认值: `compile-time option`

示例: `lock_file /var/log/lock_file`;

如果 Nginx 是由 `gcc`、`Intel C++` 或 `SunPro C++` 在 `i386`、`amd64` 平台上编译的, Nginx 可采用异步互斥进行访问控制。

12.1.8 master_process 指令

语法: master_process on | off

默认值: on

示例: master_process off;

生产环境中, 请不要使用 daemon 和 master_process 指令, 这些选项主要用于开发调试。

12.1.9 pid 指令

语法: pid file

默认值: compile-time option

示例: pid /var/log/nginx.pid;

pid 文件内记录着当前 Nginx 主进程的 ID 号, 可以通过 kill 命令发送信号给该 ID 号, 例如重新加载 Nginx 配置文件完成平滑重启: kill -HUP `cat /var/log/nginx.pid`

12.1.10 ssl_engine 指令

语法: ssl_engine engine

默认值: 系统默认依赖的引擎

此指令可以设置首选的 SSL 引擎。你可以通过命令行工具 openssl engine -t 找出系统目前支持的 SSL 引擎, 例如:

```
$ openssl engine -t
(cryptodev) BSD cryptodev engine
[ available ]
(dynamic) Dynamic engine loading support
```

12.1.11 timer_resolution 指令

语法: timer_resolution t

默认值: none

示例: timer_resolution 100ms;

该指令可以减少 gettimeofday() 函数获取当前时间的系统调用次数。在默认情况下,

gettimeofday()函数会在每一次 kevent()、epoll、/dev/poll、select()、poll()返回时被调用。如果你需要在日志中记录毫秒级的准确时间，或者毫秒级的准确反向代理响应时间，你要使用此指令。

12.1.12 try_files 指令

语法: try_files path1 [path2] uri

默认值: none

备注: 从 Nginx 0.7.27 版本开始提供此指令。

该指令可以按照参数顺序检查文件是否存在，以及返回第一个被找到的文件名。以 “/” 结尾的表示一个目录。如果文件没有找到，将被内部重定向到最后一个参数上。最后的参数是一个后备的 URI（统一资源标识符），它必须存在，否则会报内部错误。

代码 12-2 为在反向代理中使用 try_files 指令：

代码 12-2

```
location / {
    try_files /system/maintenance.html
    $uri $uri/index.html $uri.html @mongrel;
}

location @mongrel {
    proxy_pass http://mongrel;
}
```

代码 12-3 为在 FastCGI 中使用 try_files 指令：

代码 12-3

```
location / {
    try_files $uri $uri/ @drupal;
}

location ~ /\.php$ {
    try_files $uri @drupal;
    fastcgi_pass 127.0.0.1:8888;
    fastcgi_param SCRIPT_FILENAME /path/to$fastcgi_script_name;
    # other fastcgi_param
}

location @drupal {
    fastcgi_pass 127.0.0.1:8888;
    fastcgi_param SCRIPT_FILENAME /path/to/index.php;
    fastcgi_param QUERY_STRING q=$request_uri;
    # other fastcgi_param
}
```

在以上两个例子中，指令 `try_files`：

```
location / {  
    try_files $uri $uri/ @drupal;  
}
```

类似于以下的指令：

```
location / {  
    error_page 404 = @drupal;  
    log_not_found off;  
}
```

和

```
location ~ \.php$ {  
    try_files $uri @drupal;  
  
    fastcgi_pass 127.0.0.1:8888;  
    fastcgi_param SCRIPT_FILENAME /path/to$fastcgi_script_name;  
    # other fastcgi_param  
}
```

`try_files` 指令还可以在请求提交给 PHP 等 FastCGI 服务器之前，检查 PHP 文件是否存在。

在 Wordpress 博客系统和 Joomla! 内容发布系统中使用的示例如代码 12-4 所示：

代码 12-4

```
location / {  
    try_files $uri $uri/ @wordpress;  
}  
  
location ~ \.php$ {  
    try_files $uri @wordpress;  
  
    fastcgi_pass 127.0.0.1:8888;  
    fastcgi_param SCRIPT_FILENAME /path/to$fastcgi_script_name;  
    # other fastcgi_param  
}  
  
location @wordpress {  
    fastcgi_pass 127.0.0.1:8888;  
    fastcgi_param SCRIPT_FILENAME /path/to/index.php;  
    # other fastcgi_param  
}
```

12.1.13 user 指令

语法：user user [group]

默认值: nobody nobody

示例: user www users;

该指令用于指定运行 Nginx Worker 进程的用户和组, 默认的用户名和组名都是 nobody。如果组名没有指定, Nginx 则默认组名与用户名相同。

12.1.14 worker_cpu_affinity 指令

语法: user user [group]

默认值: nobody nobody

备注: 只能在 Linux 环境中使用。

使用此指令, 你可以为每个 Nginx Worker 进程绑定指定的一颗 CPU (双核 CPU 算做两颗), Nginx 采用 sched_setaffinity() 函数绑定 CPU。

示例如下:

```
worker_processes 4;  
worker_cpu_affinity 0001 0010 0100 1000;
```

分别给每个 worker 进程绑定一个 CPU:

```
worker_processes 2;  
worker_cpu_affinity 0101 1010;
```

在上例中, 第一个 Nginx Worker 进程绑定了 CPU0/CPU2, 第二个 Nginx Worker 进程绑定了 CPU1/CPU3。这种配置适用于超线程技术。

12.1.15 worker_priority 指令

语法: worker_priority [-] number

默认值: on

使用该选项可以给所有的 worker 进程分配优先值。

12.1.16 worker_processes 指令

语法: worker_processes number

默认值: 1

示例: `worker_processes 5;`

Nginx 可以使用多个 worker 进程, 原因如下:

(1) 使用 SMP 对称多处理方式, 充分发挥多核 CPU 的优势。SMP 的全称是“对称多处理”(Symmetrical Multi-Processing)技术, 是指在一个计算机上汇集了一组处理器(多 CPU), 各 CPU 之间共享内存子系统及总线结构。它是相对非对称多处理技术而言的、应用十分广泛的并行技术。在这种架构中, 一台电脑不再由单个 CPU 组成, 而同时由多个处理器运行操作系统的单一复本, 并共享内存和一台计算机的其他资源。虽然同时使用多个 CPU, 但是从管理的角度来看, 它们的表现就像一台单机一样。系统将任务队列对称地分布于多个 CPU 之上, 从而极大地提高了整个系统的数据处理能力。所有的处理器都可以平等地访问内存、I/O 和外部中断。在对称多处理系统中, 系统资源被系统中所有 CPU 共享, 工作负载能够均匀地分配到所有可用处理器之上。

(2) 能够减少进程在磁盘 I/O 阻塞上的延迟时间。

(3) 当使用 `select()/poll()` 模型时, 能够限制每个进程的连接数。

通过 `worker_processes` 和 `worker_connections` 两个指令可以计算出最大客户端连接数, 计算公式如下:

```
max_clients = worker_processes * worker_connections
```

12.1.17 worker_rlimit_core 指令

语法: `worker_processes number`

该指令用于指定每个 Nginx 进程的最大 core 文件大小。

12.1.18 worker_rlimit_nofile 指令

语法: `worker_rlimit_nofile limit`

示例: `worker_rlimit_nofile 65535;`

该指令用于指定 Nginx 进程可以打开的最大文件描述符数量。

12.1.19 worker_rlimit_sigpending 指令

语法: `worker_rlimit_sigpending limit`

示例: `worker_rlimit_sigpending 32768;`

(Linux 2.6.8 以上内核支持) 该指令指定调用进程的真正用户 ID 的排队数量。

12.1.20 working_directory 指令

语法: working_directory path

默认值: --prefix

该指令用于指定 Nginx 的工作目录, path 参数只能使用绝对路径。该指令的默认值为 Nginx 使用 configure 文件编译安装时, 参数--prefix==PATH 指定的路径。

12.2 主模块变量

在主模块指令中, 可以从以下变量获取相关信息:

\$nginx_version

当前运行的 Nginx 版本号。

\$pid

进程 ID 号。

\$realpath_root

Root 目录绝对路径。

12.3 事件模块指令

Nginx 的事件模块是控制 Nginx 处理访问连接的指令集, 跟主模块指令一样, 事件模块的指令也写在 Nginx 的配置文件的最上方区域。下面, 我们来介绍 Nginx 事件模块中包含的指令。

12.3.1 accept_mutex 指令

语法: accept_mutex [on | off]

默认值: on

Nginx 使用连接互斥锁进行顺序的 accept()系统调用。

12.3.2 accept_mutex_delay 指令

语法: `accept_mutex_delay Nms;`

默认值: 500ms

如果一个工作进程没有互斥锁,它将在最少 N 毫秒延迟之后再次尝试获取互斥锁。默认的延迟时间为 500 毫秒。

12.3.3 debug_connection 指令

语法: `debug_connection [ip|CIDR]`

默认值: none

从 Nginx 0.3.54 版本开始支持 CIDR 地址格式。这个选项用于记录 IP/网络的用户端侦错日志。

示例如下:

```
error_log /var/log/nginx/errors;
events {
    debug_connection 192.168.1.1;
}
```

12.3.4 use 指令

语法: `use [kqueue | rtsig | epoll | /dev/poll | select | poll | eventport]`

如果你在使用 `./configure` 脚步编译安装 Nginx 时,指定了一个以上的事件模型,则要告诉 Nginx 使用哪种事件模型。如果您的操作系统是 FreeBSD 4.1 以上版本,推荐使用 Kqueue 模型,如果您的操作系统是 Linux 2.6 以上内核版本,推荐使用 epoll 模型,如果您的操作系统是 Solaris 10,推荐使用 eventport 模型。Kqueue、epoll、eventport 分别属于 FreeBSD、Linux、Solaris 操作系统中高效的时间模型。

12.3.5 worker_connections 指令

语法: `worker_connections number`

该指令用于设置每个工作进程能够处理的连接数。通过 `worker_connections` 和 `worker_processes` (Nginx 工作进程数) 可以计算出 Nginx 服务器能够处理的最大连接数 `max_clients`, 计算公式如下:

$$\text{max_clients} = \text{worker_processes} * \text{worker_connections}$$

假设你开启了 8 个 Nginx 进程(worker_processes), 设置的 worker_connections 连接数为 32 768, 那么最大连接数 max_clients 为 $8 * 32768 = 262\ 144$ 。这只是理论值, 实际情况下, 受操作系统、文件描述符与端口数的限制, 最大连接数是达不到 262 144 的。

在反向代理情况下, 最大连接数变成了:

```
max_clients = worker_processes * worker_connections/4
```

因为, Internet Explorer 浏览器通常默认打开两个与服务器的连接, 而 Nginx 使用同一个文件描述符池中的描述符来连接后端。