# Nginx学习笔记

参考：<https://www.nginx.com/resources/wiki/start/>

## 起步

### Nginx安装

#### Ubuntu版

（其它系统参见<http://nginx.org/en/docs/install.html>）

Stable版本

下载密钥来签名nginx包和存储库（http://nginx.org/en/linux\_packages.html）

执行指令：

sudo apt-key add nginx\_signing.key

把下行代码中的codename替换成ubuntu发行名称，并加到 /etc/apt/sources.list结尾。

deb http://nginx.org/packages/ubuntu/ *codename* nginx

deb-src http://nginx.org/packages/ubuntu/ *codename* nginx

apt-get update

apt-get install nginx

### 命令管理

https://www.nginx.com/resources/wiki/start/topics/tutorials/commandline/

#### 启动Nginx

/usr/sbin/nginx

#### 基本命令

形如 nginx –v

|  |  |
| --- | --- |
| -?, -h | Print help.打印帮助 |
| -v | Print version.打印版本 |
| -V | Print NGINX version, compiler version and configure parameters.打印nginx版本，编译版本，配置信息 |
| -t | Don’t run, just test the configuration file. NGINX checks configuration for correct syntax and then try to open files referred in configuration.测试nginx配置信息是否正确，试着打开与配置关联的文件。 |
| -q | Suppress non-error messages during configuration testing. |
| -s signal | Send signal to a master process: stop, quit, reopen, reload. (version >= 0.7.53)发送信号给主进程 |
| -p prefix | Set prefix path (default: /usr/local/nginx/). (version >= 0.7.53)设置前缀地址 |
| -c filename | Specify which configuration file NGINX should use instead of the default.指定配置信息 |
| -gdirectives | Set [global](http://nginx.org/en/docs/http/ngx_http_core_module.html) directives. (version >= 0.7.4)设置全局指令 |

注意：nginx仅有很少的命令参数，它的配置全部由配置信息配置。

Nginx有两种方式来控制正在运行的系统

第一种 通过nginx –s signal来控制

* stop — fast shutdown
* quit — graceful shutdown
* reload — reloading the configuration file
* reopen — reopening the log files

第二中 根据NGINX主进程id来发送信号进行控制

ps –ax | grep nginx来获取ngnix进程id或者在配置信息中首部有指定pid地址，可查看。

可使用形如以下指令进行操作

kill -QUIT **$(** cat /usr/local/nginx/logs/nginx.pid **)**

|  |  |
| --- | --- |
| TERM, INT | Quick shutdown 快速关闭 |
| QUIT | Graceful shutdown 安全优雅的关闭 |
| KILL | Halts a stubborn process 强行关掉某个进程 |
| HUP | Configuration reload 重新加载配置信息，启动新的进程，安全关闭旧进程  Start the new worker processes with a new configuration  Gracefully shutdown the old worker processes |
| USR1 | Reopen the log files |
| USR2 | Upgrade Executable on the fly平滑升级 |
| WINCH | Gracefully shutdown the worker processes安全关闭工作经常 |

#### 使用信号设置配置

/usr/sbin/nginx -c /etc/nginx/nginx.conf

当nginx收到一个HUP指令时候，它会试图解析配置信息（被指定的那个，如果没指定，则使用默认的）。如果成功加载，试图使用新的配置信息，即重新打开log文件和监听的socket。如果成功，nginx运行新的工作进程和信号来安全的关闭旧进程。注意：旧的工作进程关闭监听的socket但是会继续给正在服务的客户端服务。当服务完所有的客户端，才关闭。如果nginx不能成功的加载新的配置信息，它将继续执行旧的配置信息。

#### Nginx平滑升级（USR2指令）

即nginx升级或者添加或移除服务器模块。你可以在没有服务器中断或者请求遗失的情况下进行。

步骤如下：

1. 用新的版本更换旧的二进制文件
2. 给主进程发送USR2信号

它会更改它的.pid文件为.pid.oldbin，然后执行新的二进制文件从而启动新的主进程和新的工作进程。

在这段时间，有两个nginx实例在运行，一起处理新来的请求。为了把旧的工作进程给淘汰掉，你必须发送WINCH指令给旧的主进程来使它安全的关闭。

在这之后旧的工作进程会全部退出，新的工作进程将处理新来的请求。

在这个时候你依然能还原旧的服务器，因为它还没有关闭自己的socket。它会执行下面几步：

1. 发送HUP信号给旧的主进程，这会在没有重新加载配置文件的情况下启动工作进程。
2. 发送QUIT信号给新的主进程来安全的退出它的工作进程。
3. 发送TERM信号给新的住精彩来强制它退出。
4. 如果因为某些原因新的工作进程没有退出，那么发送KILL信号给它们。

当一个新的主进程退出之后，旧的主进程移除它的.pid文件中.oldbin后缀。所有事都如最初。

如果更新成功，你想继续保持新的服务，发送QUIT信号给旧的主进程来关闭它。

### 陷阱和常见错误

https://www.nginx.com/resources/wiki/start/topics/tutorials/config\_pitfalls/

#### 不要使用chmod 777。可使用namei –om /path/to/check来查询权限。

#### 在location 模块的root

坏配置

**server** {

**server\_name** www.example.com;

**location** / {

**root** /var/www/nginx-default/;

*# [...]*

}

**location** /foo {

**root** /var/www/nginx-default/;

*# [...]*

}

**location** /bar {

**root** /var/www/nginx-default/;

*# [...]*

}

}

把root放到location模块可以正常运行，但是当你添加root模块到所有location，则没有被匹配到的location模块将没有root。好的配置如下：

**server** {

**server\_name** www.example.com;

**root** /var/www/nginx-default/;

**location** / {

*# [...]*

}

**location** /foo {

*# [...]*

}

**location** /bar {

*# [...]*

}

}

#### 太多index指令

坏配置：

**http** {

**index** index.php index.htm index.html;

**server** {

**server\_name** www.example.com;

**location** / {

**index** index.php index.htm index.html;

*# [...]*

}

}

**server** {

**server\_name** example.com;

**location** / {

**index** index.php index.htm index.html;

*# [...]*

}

**location** /foo {

**index** index.php;

*# [...]*

}

}

}

不用重复那么多行index，简单使用它一次就够了。它只需要在你的http{}模块里定义，然后将会继承下去。

好的配置：

**http** {

**index** index.php index.htm index.html;

**server** {

**server\_name** www.example.com;

**location** / {

*# [...]*

}

}

**server** {

**server\_name** example.com;

**location** / {

*# [...]*

}

**location** /foo {

*# [...]*

}

}

}

#### if是邪恶的

if指令在上下文中使用时候有问题，甚至可能导致segfaults。尽量不要使用它。

* 只有在[return](http://nginx.org/en/docs/http/ngx_http_rewrite_module.html?&_ga=1.114985991.1803309096.1455860144#return) 和[rewrite](http://nginx.org/en/docs/http/ngx_http_rewrite_module.html" \l "rewrite) ... last上下文中if才能保证100%正确。

只有在合适的测试和了解if之后才使用它。

##### 怎么替换它

使用[try\_files](http://nginx.org/en/docs/http/ngx_http_core_module.html?&_ga=1.146826679.1803309096.1455860144" \l "try_files)。某些情况下使用[return](http://nginx.org/en/docs/http/ngx_http_rewrite_module.html?&_ga=1.114985991.1803309096.1455860144" \l "return) 和[rewrite](http://nginx.org/en/docs/http/ngx_http_rewrite_module.html" \l "rewrite) ... last。某些情况下把ifs移动到server等级。（这里它是安全的因为只有rewrite模块指令被允许在里面）。

例如：以下设置可以用来安全地更改用来处理请求的location。

**location** / {

**error\_page** 418 = @other;

**recursive\_error\_pages** on;

**if** ($something) {

**return** 418;

}

*# some configuration*

**...**

}

location @other {

*# some other configuration*

**...**

}

在某些时候好，使用嵌入的脚本模块（[embedded perl](http://nginx.org/en/docs/http/ngx_http_perl_module.html), or various [NGINX 3rd Party Modules](https://www.nginx.com/resources/wiki/modules/))来处理。

##### 例子

这些例子来跟你解释为什么if是邪恶的，最后不要在尝试它。

*# Here is collection of unexpectedly buggy configurations to show that*

*# if inside location is evil.*

*# only second header will be present in response*

*# not really bug, just how it works*

**location** /only-one-if {

**set** $true 1;

**if** ($true) {

**add\_header** X-First 1;

}

**if** ($true) {

**add\_header** X-Second 2;

}

**return** 204;

}

*# request will be sent to backend without uri changed*

*# to '/' due to if*

**location** /proxy-pass-uri {

**proxy\_pass** http://127.0.0.1:8080/;

**set** $true 1;

**if** ($true) {

*# nothing*

}

}

*# try\_files wont work due to if*

**location** /if-try-files {

**try\_files** /file @fallback;

**set** $true 1;

**if** ($true) {

*# nothing*

}

}

*# nginx will SIGSEGV*

**location** /crash {

**set** $true 1;

**if** ($true) {

*# fastcgi\_pass here*

**fastcgi\_pass** 127.0.0.1:9000;

}

**if** ($true) {

*# no handler here*

}

}

*# alias with captures isn't correcly inherited into implicit nested*

*# location created by if*

**location** ~\* ^/if-and-alias/(?<file>.\*) {

**alias** /tmp/$file;

**set** $true 1;

**if** ($true) {

*# nothing*

}

}

BAD:

**server** {

**server\_name** example.com \*.example.com;

**if** ($host ~\* ^www\.(.+)) {

**set** $raw\_domain $1;

**rewrite** ^/(.\*)$ $raw\_domain/$1 permanent;

}

*# [...]*

}

**}**

这个例子中无论请求的是子域还是普通的网址，这个指令总是要检查host的值，这非常低效。

GOOD:

**server** {

**server\_name** www.example.com;

**return** 301 $scheme://example.com$request\_uri;

}

**server** {

**server\_name** example.com;

*# [...]*

}

检查文件是否存在

BAD:

**server** {

**root** /var/www/example.com;

**location** / {

**if** (!-f $request\_filename) {

**break**;

}

}

}

GOOD:

**server** {

**root** /var/www/example.com;

**location** / {

**try\_files** $uri $uri/ /index.html;

}

}

##### 为什么这个会发生而且一直没被解决呢？

Directive “if” is part of rewrite module which evaluates instructions imperatively. On the other hand, NGINX configuration in general is declarative. At some point due to users demand an attempt was made to enable some non-rewrite directives inside “if”, and this lead to situation we have now. It mostly works, but... see above.在另一方面，NGINX配置文件一般是声明的，在一些地方由于用户需要尝试去是的某些if中的non-rewritezhi指令。

看起来唯一的解决办法是使得if中的non-rewrite指令完全失效。这会破坏很多里面配置。

##### 建议：

如果你想使用if。请确保你明白它怎么工作。一些基本的想法可能会发现如[这里](http://agentzh.blogspot.com/2011/03/how-nginx-location-if-works.html)。做正确的测试。

#### Front controller模式的web应用程序

1.不同应用程序使用的参数名不同

2.有些程序甚至不需要查询字段，它们能直接读取request\_uri。

#### 未完待续

### 使用nginx作为httpload balancer

#### 介绍

跨多个应用程序实例负载均衡是优化资源利用率​​，最大限度地提高吞吐量，降低延迟，并确保容错配置一个常用的技术。它可以使用的nginx作为一个非常有效的HTTP负载平衡器将流量分配到多个应用服务器并提高性能，可扩展性和Web应用程序的可靠性。

#### 负载均衡办法

1. 轮询（默认）：发向应用服务器的请求以轮询的形式去分发。
2. 最少连接：下一个请求被分发到有最少活动连接的服务器上。
3. ip哈希：一个哈希的办法用来决定哪个服务器应该被选择来处理下一个请求。基于客户的IP地址来决定。

#### 默认负载均衡配置

一个最简易的负载均衡配置如下：

http {

upstream myapp1 {

server srv1.example.com;

server srv2.example.com;

server srv3.example.com;

}

server {

listen 80;

location / {

proxy\_pass http://myapp1;

}

}

}

以上配置中没有指定负载均衡办法，则默认为轮询。所有的请求被myapp1代理，nginx使用http负载均衡来分发请求。

在nginx中的反向代理包括http，https，fastcgi，uwsgi，scgi和memcached。

## 模块结构及应用

### Nginx架构初探

nginx在启动后，会有一个master进程和多个worker进程。master进程主要用来管理worker进程，包含：接收来自外界的信号，向各worker进程发送信号，监控worker进程的运行状态，当worker进程退出后(异常情况下)，会自动重新启动新的worker进程。而基本的网络事件，则是放在worker进程中来处理了。多个worker进程之间是对等的，他们同等竞争来自客户端的请求，各进程互相之间是独立的。一个请求，只可能在一个worker进程中处理，一个worker进程，不可能处理其它进程的请求。worker进程的个数是可以设置的，一般我们会

设置与机器cpu核数一致，这里面的原因与nginx的进程模型以及事件处理模型是分不开的。



### Nginx怎么处理请求

#### 基于域名的虚拟服务器

server {

listen 80;

server\_name example.org www.example.org;

...

}

server {

listen 80;

server\_name example.net www.example.net;

...

}

server {

listen 80;

server\_name example.com www.example.com;

...

}

以上配置nginx会根据请求头中的“Host”来决定请求将转发到那个服务器。如果它的值没有匹配，或者不包含头字段，则nginx会将请求转发到这个端口上的默认服务器。Nginx标准情况下把第一个server当成默认服务器。可以使用下面配置进行明确指定。

server {

listen 80 **default\_server**;

server\_name example.net www.example.net;

...

}

注意：这个default\_server是监听端口的属性而不是一个服务器名称。

#### 怎么防止未定义服务器名的服务器处理请求

如果没有Host头字段的请求不被允许，服务器可以放弃这个请求，如下定义：

server {

listen 80;

server\_name "";

return 444;

}

Server\_name为空能匹配不存在”Host”头字段的请求。

#### 混合的基于域名和基于ip的虚拟服务器

server {

listen 192.168.1.1:80;

server\_name example.org www.example.org;

...

}

server {

listen 192.168.1.1:80;

server\_name example.net www.example.net;

...

}

server {

listen 192.168.1.2:80;

server\_name example.com www.example.com;

...

}

以上配置中，nginx首先测试请求中的IP地址和端口和server模块监听指令。然后它测试请求中的头字段和server中匹配ip地址和端口的模块的server\_name实体。如果server\_name没找到，requests会被默认服务器处理。

注意：由于default\_server是监听端口的属性，则不同端口能定义不同默认服务器。如下：

server {

listen 192.168.1.1:80;

server\_name example.org www.example.org;

...

}

server {

listen 192.168.1.1:80 **default\_server**;

server\_name example.net www.example.net;

...

}

server {

listen 192.168.1.2:80 **default\_server**;

server\_name example.com www.example.com;

...

}

#### 一个简单的PHP站点配置

server {

listen 80;

server\_name example.org www.example.org;

root /data/www;

location / {

index index.html index.php;

}

location ~\* \.(gif|jpg|png)$ {

expires 30d;

}

location ~ \.php$ {

fastcgi\_pass localhost:9000;

fastcgi\_param SCRIPT\_FILENAME

$document\_root$fastcgi\_script\_name;

include fastcgi\_params;

}

}

Nginx首先不管它列出的顺序，来通过文字字符串搜索最具体的前缀地址。以上配置中唯一的前缀是“/”，因此它匹配任何请求并作为最后的一种处理手段。然后nginx检查在配置文件中列出的正则表达式所给的地址，第一个匹配的会停止搜索，nginx将使用这个地址。如果没有正则表达式匹配一个请求，那么nginx会使用之前发现的最具体的前缀地址。

注意：所有类型的地址只测试不带参数的请求行中的URI部分。这是因为在查询字符中的参数会以好几种方式给出。例如：

/index.php?user=john&page=1

/index.php?page=1&user=john

此外，任何人能再查询字符中请求任何东西。

/index.php?page=1&something+else&user=john

先分析一下以上配置信息：

一个“/logo.gif”请求首先被前缀地址“/”匹配，然后被正则表达式“\.(gif|jpg|png)$”，因此它将被后面地址所处理。使用指令“root /data/www”这个请求被映射到文件/data/www/logo.gif，然后这文件会被发送到客户端。

一个”/index.php”请求也首先匹配前缀地址”/”然后被正则表达式“\.(php)$”。因此，它将被后面地址处理然后这个请求将被分发到在localhost:9000监听的FastCGIf服务器。fastcgi\_param指令将FastCGI的SCRIPT\_FILENAME参数设置为“/data/www/index.php”。$document\_root变量等于root指令，$fastcgi\_script\_name等于request的URI，这里为“index.php”。

一个“/about.html”请求仅仅被前缀地址“/”匹配。因此，它将被该地址处理。使用指令“root /data/www”，请求被映射到文件/data/www/about.html，然后这个文件被发送会客户端。

处理请求“/”会更复杂。它仅被前缀地址“/”匹配。因此它被该地址处理。然后index指令根据它的参数和“root /data/www”指令测试index文件是否存在。如果/data/www/index.html文件不存在，但是/data/www/index.php存在。那么指令会重定向到“/index.php”。然后nginx再一次查询地址，如果请求被客户端发送过来。由我们之前所见，重定向请求最终会被FastCGI服务器处理。

### 服务器名称

服务器名称由server\_name指令定义。它们可以用精确的名称，通配符或者正则表达式定义。如下可见：

server {

listen 80;

server\_name example.org www.example.org;

...

}

server {

listen 80;

server\_name \*.example.org;

...

}

server {

listen 80;

server\_name mail.\*;

...

}

server {

listen 80;

server\_name ~^(?<user>.+)\.example\.net$;

...

}

当使用名称进行虚拟服务器搜索，如果名称匹配不止一个指定名称。则第一个匹配的服务器将被选择。优先顺序如下：

1. 确切名称
2. 以\*开头的最长通配符名称
3. 以\*结尾的最长通配符名称
4. 第一个被匹配的正则表达式（在配置文件中出现的顺序）

#### 通配符名称

通配符名称可能包含\*在名字的开头或结尾，而且在一个.的边界。而正则表达式没有这种限制。”\*.example.org”不仅能匹配[www.example.org](http://www.example.org)还能匹配[www.sub.example.org](http://www.sub.example.org).

一个特殊的通配符”.example.org”，既能匹配确切地址”example.org”,还能匹配”\*.example.org”。

#### 正则表达式名称

Nginx使用的正则表达式和perl中的是兼容的。为了使用正则表达式，服务器名称要以波浪符号开头：

server\_name ~^www\d+\.example\.net$;

否则将被对待成确切地址或者通配符地址。不要忘了设置”^””$”锚，语法上不需要，到时逻辑上需要。注意域名的.需要被\转义。包含{}的字符应该被引用：

server\_name "~^(?<name>\w\d**{**1,3**}**+)\.example\.net$";

否则nginx会失败并展示错误信息：

directive "server\_name" is not terminated by ";" in ...

一个命名的正则表达式捕获后能被当成后面的变量：

server {

server\_name ~^(www\.)?(**?<domain>**.+)$;

location / {

root /sites/**$domain**;

}

}

PCRE包能命名捕获使用以下语法：

|  |  |
| --- | --- |
| ?<*name*> | Perl 5.10 compatible syntax, supported since PCRE-7.0 |
| ?'*name*' | Perl 5.10 compatible syntax, supported since PCRE-7.0 |
| ?P<*name*> | Python compatible syntax, supported since PCRE-4.0 |

如果nginx运行失败并展示如下：

pcre\_compile() failed: unrecognized character after (?< in ...

说明PCRE包过时了，”?P<name>”应该被取代。

命名捕获也可以以数字形式:

server {

server\_name ~^(www\.)?(.+)$;

location / {

root /sites/**$2**;

}

}

#### 其它名称（Miscellaneous names）

有很多可以被特殊处理的服务器名。

如果想在非默认的服务器模块上处理没有“host”头字段的请求，空名需要被指定：

server {

listen 80;

server\_name example.org www.example.org "";

...

}

如果在server模块中server\_name没有被指定，那么将使用空名作为server\_name.

注意：版本0.8.48以前使用hostname作为server\_name。

如果有人使用IP地址而不是域名发请求，则“host”请求头字段包含ip地址和请求，它将被使用ip地址作为server name的服务器处理：

server {

listen 80;

server\_name example.org

www.example.org

""

**192.168.1.1**

;

...

}

“\_”server\_name 如下：

server {

listen 80 default\_server;

server\_name \_;

return 444;

}

某个无效域名（不会与真正名称相交）

使用server\_name指令没有办法指定catch-all name或者default server。可以使用端口定义，如\*：80，\*：8080：

server {

listen 80;

listen 8080 default\_server;

server\_name example.net;

...

}

server {

listen 80 default\_server;

listen 8080;

server\_name example.org;

...

}