Nginx模块参考手册中文版

原英文文档来源于Nginx维基，网页版会和维基同步更新，由于是第一次翻译英文文档，错误在所难免，如果在阅读文档过程中发现翻译不当的地方，请点击这里与作者联系，谢谢！

目录

E-mail: qzao16@21cn.com

MSN: qzao16@21cn.com

QQ: 179401

编译Nginx (Compiling Nginx)

回目录

Nginx模块必须在编译的时候指定，完整的编译选项，可用的模块可以参考安装指导

下面是一个例子:

```
./configure \
  --prefix=/usr \
  --sbin-path=/usr/sbin/nginx \
  --conf-path=/etc/nginx/nginx.conf \
  --error-log-path=/var/log/nginx/error.log \
  --pid-path=/var/run/nginx/nginx.pid  \
  --lock-path=/var/lock/nginx.lock \
  --user=nginx \
  --group=nginx \
  --with-http_ssl_module \
  --with-http_flv_module \
  --with-http_gzip_static_module \
  --http-log-path=/var/log/nginx/access.log \
  --http-client-body-temp-path=/var/tmp/nginx/client/ \
  --http-proxy-temp-path=/var/tmp/nginx/proxy/ \
  --http-fastcgi-temp-path=/var/tmp/nginx/fcgi/
```

更多编译参数与可用的模块信息请运行./configure --help

前进

Nginx核心模块（Nginx Core Modules）

回目录

这些模块对于Nginx是必须的

主模块（Main Module）

回目录

· 摘要

包含一些Nginx的基本控制功能

· 指令

daemon

语法：daemon on | off

默认值：on

```
daemon off;
```

生产环境中不要使用"daemon"和"master_process"指令，这些指令仅用于开发调试。虽然可以使用daemon off在生产环境中，但对性能提升没有任何帮助，但是在生产环境中永远不要使用master_process off。

env

语法：env VAR|VAR=VALUE

默认值：TZ

使用字段：main

这个命令允许其限定一些环境变量的值，在以下的情况下会创建或修改变量的值：

· 在不停机情况下升级、增加或删除一些模块时继承的变量

· 使用嵌入式perl模块

· 使用工作中的进程，必须记住，某些类似系统库的行为管理仅在变量初始化时频繁地使用库文件，即仍然可以用之前给定的命令设置，上面提到的零停机更新文件是一个例外（此句不知怎么翻，原文：for use by working processes. However it is necessary to keep in mind, that management of behaviour of system libraries in a similar way probably not always as frequently

libraries use variables only during initialization, that is still before they can be set by means of the given instruction. Exception to it is the above described updating an executed file with zero downtime.)

如果没有明确的定义TZ的值,默认情况下它总是继承的，并且内置的Perl模块总是可以使用TZ的值。

例:

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

debug_points

语法: debug_points [stop | abort]

默认值: none (无)

```
debug_points stop;
```

在Nginx内部有很多断言，如果debug_points的值设为stop时，那么触发断言时将停止Nginx并附加调试器。如果debug_point的值设为abort,那么触发断言时将创建内核文件。

error_log

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

默认值: ¥{prefix}/logs/error.log

指定Nginx服务 (与FastCGI) 错误日志文件位置。

每个字段的错误日志等级默认值:

1、main字段 - error

2、HTTP字段 - crit

3、server字段 - crit

Nginx支持为每个虚拟主机设置不同的错误日志文件，这一点要好于lighttpd，详细为每个虚拟主机配置不同错误日志的例子请参考: SeparateErrorLoggingPerVirtualHost和mailing list thread on separating error logging per virtual host

如果你在编译安装Nginx时加入了--with-debug参数，你可以使用以下配置:

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

注意error_log off并不能关闭日志记录功能，而会将日志文件写入一个文件名为off的文件中，如果你想关闭错误日志记录功能，应使用以下配置：

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

同样注意0.7.53版本，nginx在使用配置文件指定的错误日志路径前将使用编译时指定的默认日志位置，如果运行nginx的用户对该位置没有写入权限，nginx将输出如下错误：

```
[alert]: could not open error log file: open() "/var/log/nginx/error.log" failed (13: Permission denied)
```

log_not_found

语法：log_not_found on | off

默认值：on

使用字段：location

这个参数指定了是否记录客户端的请求出现404错误的日志，通常用于不存在的robots.txt和favicon.ico文件，例如：

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

include

语法：include file | *

默认值：none

你可以包含一些其他的配置文件来完成你想要的功能。

0.4.4版本以后，include指令已经能够支持文件通配符：

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

注意：直到0.6.7版本，这个参数包含的文件相对路径随你在编译时指定的--prefix=PATH目录而决定，默认是/usr/local/nginx，如果你不想指定这个目录下的文件，请写绝对路径。

0.6.7版本以后指定的文件相对路径根据nginx.conf所在的目录而决定，而不是prefix目录的路径。

lock_file

语法：lock_file file

默认值：编译时指定

```
lock_file  /var/log/lock_file;
```

Nginx使用连接互斥锁进行顺序的accept()系统调用，如果Nginx在i386,amd64,sparc64,与ppc64环境下使用gcc,Intel C++,或SunPro C++进行编译，Nginx将采用异步互斥进行访问控制，在其他情况下锁文件会被使用。

master_process

语法：master_process on | off

默认值：on

```
master_process  off;
```

生产环境中不要使用"daemon"和"master_process"指令，这些选项仅用于开发调试。

pid

语法：pid file

默认值：编译时指定

```
pid /var/log/nginx.pid;
```

指定pid文件，可以使用kill命令来发送相关信号，例如你如果想重新读取配置文件，则可以使用：kill -HUP `cat /var/log/nginx.pid`

ssl_engine

语法：ssl_engine engine

默认值：依赖于系统环境

这里可以指定你想使用的OpenSSL引擎，你可以使用这个命令找出哪个是可用的：openssl engine -t

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

timer_resolution

语法: timer_resolution t

默认值: none

```
timer_resolution  100ms;
```

这个参数允许缩短gettimeofday()系统调用的时间，默认情况下gettimeofday()在下列都调用完成后才会被调用: kevent(), epoll, /dev/poll, select(), poll()。

如果你需要一个比较准确的时间来记录¥upstream_response_time或者¥msec变量，你可能会用到timer_resolution

try_files

语法: try_files path1 [path2] uri

默认值: none

可用版本: 0.7.27

按照顺序检查存在的文件，并且返回找到的第一个文件，斜线指目录: ¥uri / 。如果在没有找到文件的情况下，会启用一个参数为last的内部重定向到，这个last参数“必须”被设置用来返回URL，否则会产生一个内部错误。

在代理Mongrel中使用:

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

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

在Drupal / FastCGI中:

```
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
}
```

在这个例子中，这个try_files指令：

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

等同于下列配置：

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

这段：

```
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
}
```

指try_files在将请求提交到FastCGI服务之前检查存在的php文件。

一个在Wordpress和Joomla中的例子：

```
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
}
```

user

语法：user user [group]

默认值：nobody nobody

如果主进程以root运行，Nginx将会调用setuid()/setgid()来设置用户/组，如果没有指定组，那么将使用与用户名相同的组，默认情况下会使用nobody用户与nobody组（或者nogroup），或者在编译时指定的--user=USER和--group=GROUP的值。

```
user www users;
```

worker_cpu_affinity

语法: worker_cpu_affinity cpumask [cpumask...]

默认值: none

仅支持linux系统。

这个参数允许将工作进程指定到cpu，它调用sched_setaffinity()函数

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

指定每个进程到一个CPU:

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

指定第一个进程到CPU0/CPU2，指定第二个进程到CPU1/CPU3，对于HTT处理器来说是一个不错的选择。

worker_priority

语法: worker_priority [-] number

默认值: on

这个选项可以用来设置所有工作进程的优先级（即linux系统中的nice），它调用setpriority()。

worker_processes

语法: worker_processes number

默认值: 1

```
worker_processes 5;
```

由于以下几点原因，Nginx可能需要运行不止一个进程

- 使用了SMP（对称多处理技术）。
- 当服务器在磁盘I/O出现瓶颈时为了减少响应时间。
- 当使用select()/poll()限制了每个进程的最大连接数时。

在事件模块这一章中我们将使用worker_processes和worker_connections来计算理论最大连接数

(max_clients)：

max_clients = worker_processes * worker_connections

worker_rlimit_core

语法: worker_rlimit_core size

默认值:

允许的每个进程核心文件最大值。

worker_rlimit_nofile

语法: worker_rlimit_nofile limit

默认值:

进程能够打开的最多文件描述符数

worker_rlimit_sigpending

语法: worker_rlimit_sigpending limit

默认值:

linux内核2.6.8以后，限制调用的进程中真实用户队列可能使用的信号数量。

working_directory

语法: working_directory path

默认值: --prefix

程序的工作目录，一般只用来指定核心文件位置，Nginx仅使用绝对路径，所有在配置文件中的相对路径会转移到--prefix==PATH

·变量

¥nginx_version

目前运行中的Nginx版本

¥pid

进程ID号

¥realpath_root

未标记

· 参考文档

前进->事件模块 (Events Module)

事件模块（Events Module）

回目录

· 摘要

控制Nginx处理连接的方式

· 指令

accept_mutex

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

accept_mutex_delay

语法: accept_mutex_delay Nms;

默认值: 500ms

如果一个进程没有互斥锁，它将至少在这个值的时间后被回收，默认是500ms

debug_connection

语法: debug_connection [ip | CIDR]

默认值: none

0.3.54版本后，这个参数支持CIDR地址池格式。

这个参数可以指定只记录由某个客户端IP产生的debug信息。

当然你也可以指定多个参数。

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

devpoll_changes

devpoll_events

kqueue_changes

kqueue_events

epoll_events

语法: devpoll_changes

默认值:

这些参数指定了按照规定方式传递到或者来自内核的事件数，默认devpoll的值为32，其余为512。

multi_accept

语法: multi_accept [on | off]

默认值: off

multi_accept在Nginx接到一个新连接通知后调用accept()来接受尽量多的连接

rtsig_signo

语法: rtsig_signo

默认值:

Nginx在rtsig模式启用后使用两个信号，该指令指定第一个信号编号，第二个信号编号为第一个加1

默认rtsig_signo的值为SIGRTMIN+10 (40)。

rtsig_overflow_events

rtsig_overflow_test

rtsig_overflow_threshold

语法: rtsig_overflow_*

默认值:

这些参数指定如何处理rtsig队列溢出。当溢出发生在nginx清空rtsig队列时，它们将连续调用poll()和rtsig.poll()来处理未完成的事件，直到rtsig被排空以防止新的溢出，当溢出处理完毕，nginx再次启用rtsig模式。

rtsig_overflow_events specifies指定经过poll()的事件数，默认为16

rtsig_overflow_test指定poll()处理多少事件后nginx将排空rtsig队列，默认值为32

rtsig_overflow_threshold只能运行在Linux 2.4.x内核下，在排空rtsig队列前nginx检查内核以确定队列是怎样被填满的

默认值为1/10，“rtsig_overflow_threshold 3”意为1/3。

use

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

默认值:

如果你在./configure的时候指定了不止一个事件模型，你可以通过这个参数告诉nginx你想使用哪一个事件模型，默认情况下nginx在编译时会检查最适合你系统的事件模型。

你可以在这里看到所有可用的事件模型并且如果在./configure时激活它们。

worker_connections

语法: worker_connections

默认值:

worker_connections和worker_proceses（见主模块）允许你计算理论最大连接数:

最大连接数 = worker_processes * worker_connections

在反向代理环境下:

最大连接数 = worker_processes * worker_connections/4

由于浏览器默认打开2个连接到服务器，nginx使用来自相同地址池的fds（文件描述符）与前后端相连接

· 参考文档

前进->Nginx标准HTTP模块

Nginx标准HTTP模块（Standard HTTP Modules）

回目录

这些模块默认会全部编译进Nginx除非手工指定某个模块在configure时排除。

HTTP核心模块（HTTP Core）

回目录

· 摘要

Nginx处理HTTP的核心功能模块

· 指令

alias

语法：alias file-path|directory-path;

默认值：no

使用字段：location

这个指令指定一个路径使用某个某个，注意它可能类似于root，但是document root没有改变，请求只是使用了别名目录的文件。

```
location  /i/ {
  alias  /spool/w3/images/;
}
```

上个例子总，请求"/i/top.gif"将返回这个文件 "/spool/w3/images/top.gif"。

Alias同样可以用于正则指定的location，如：

```
location ~ ^/download/(.*)$ {
  alias /home/website/files/$1;
}
```

请求"/download/book.pdf"将返回"/home/website/files/book.pdf"文件。

当然也可以在别名目录字段中使用变量。

client_body_in_file_only

语法：client_body_in_file_only on|off

默认值：off

使用字段：http，server，location

这个指令始终存储一个连接请求实体到一个文件即使它只有0字节。

注意：如果这个指令打开，那么一个连接请求完成后，所存储的文件并不会删除。

这个指令可以用于debug调试和嵌入式Perl模块中的¥r->request_body_file。

client_body_in_single_buffer

语法：client_body_in_single_buffer

默认值：off

使用字段：http，server，location

这个指令(0.7.58版本)指定将一个完整的连接请求放入缓冲区，当使用¥request_body时推荐使用这个指令以减少复制操作。

如果无法将一个请求放入单个缓冲区，将会被放入磁盘。

client_body_buffer_size

语法：client_body_buffer_size the_size

默认值：8k/16k

使用字段：http，server，location

这个指令可以指定连接请求使用的缓冲区大小。

如果连接请求超过缓存区指定的值，那么这些请求或部分请求将尝试写入一个临时文件。

默认值为两个内存分页大小值，根据平台的不同，它可能是8k或16k。

client_body_temp_path

语法：client_body_temp_path dir-path [level1 [level2 [level3]

默认值：client_body_temp

使用字段：http，server，location

这个指令指定连接请求试图写入文件的目录路径。

可以指定三级目录结构，如：

```
client_body_temp_path  /spool/nginx/client_temp 1 2;
```

那么它的目录结构可能是这样:

```
/spool/nginx/client_temp/7/45/00000123457
```

client_body_timeout

语法: client_body_timeout time

默认值: 60

使用字段: http, server, location

这个指令指定一个请求读取超时时间。

这个选项仅设置与客户端与服务器建立连接过程中，如果连接超过这个时间而客户端没有任何响应，Nginx将返回一个"Request time out" (408)错误

client_header_buffer_size

语法: client_header_buffer_size size

默认值: 1k

使用字段: http, server

这个指令指定客户端请求的http头部缓冲区大小

绝大多数情况下一个头部请求的大小不会大于1k

不过如果有来自于wap客户端的较大的cookie它可能会大于1k，Nginx将分配给它一个更大的缓冲区，这个值可以在large_client_header_buffers里面设置。

client_header_timeout

语法: client_header_timeout time

默认值: 60

使用字段: http, server

这个指令指定客户端请求的http头部超时时间。

这个选项仅设置与客户端与服务器建立连接并且读取http头的过程中，如果连接超过这个时间而客户端没有任何响应，Nginx将返回一个"Request time out" (408)错误。

client_max_body_size

语法: client_max_body_size size

默认值: client_max_body_size 1m

使用字段: http, server, location

这个指令指定允许客户端请求的最大的单个文件字节数，它出现在请求头部的Content-Length字段。

如果请求大于指定的值，客户端将收到一个"Request Entity Too Large" (413)错误。

需要记住，浏览器并不知道怎样显示这个错误。

default_type

语法: default_type MIME-type

默认值: default_type text/plain

使用字段: http, server, location

在基础MIME视图没有指定任何信息的情况下可以设置默认MIME类型。

参考types。

```
location = /proxy.pac {
  default_type application/x-ns-proxy-autoconfig;
}
location = /wpad.dat {
  rewrite . /proxy.pac;
  default_type application/x-ns-proxy-autoconfig;
}
```

directio

语法: directio [size|off]

默认值: directio off

使用字段: http, server, location

这个参数指定在读取文件大小大于指定值的文件时使用O_DIRECT (FreeBSD, Linux) , F_NOCACHE (Mac OS X) 或者调用directio()函数 (Solaris) , 当一个请求用到这个参数时会关掉sendfile，通常这个参数用于大文件。

```
directio  4m;
```

error_page

语法: error_page code [code...] [= | =answer-code] uri | @named_location

默认值: no

使用字段: http, server, location, location 中的if字段

这个参数可以为错误代码指定相应的错误页面

```
error_page   404          /404.html;
error_page   502 503 504  /50x.html;
error_page   403          http://example.com/forbidden.html;
error_page   404          = @fetch;
```

同样，你也可以修改返回的错误代码:

```
error_page 404 =200 /.empty.gif;
```

如果一个错误的响应经过代理或者FastCGI服务器，这个服务器将返回不同的响应码，如200, 302, 401或404，那么可以指定响应码重定向:

```
error_page   404  =  /404.php;
```

如果在重定向时不希望改变URI，可以将错误页面重定向到一个命名的location字段中:

```
location / (
    error_page 404 = @fallback;
)

location @fallback (
    proxy_pass http://backend;
)
```

if_modified_since

语法: if_modified_since [off|exact|before]

默认值: if_modified_since exact

使用字段: http, server, location

这个参数 (0.7.24) 指定了如何将文件修改的时间与请求头中的"If-Modified-Since"时间相比较。

- off : 不检查请求头中的"If-Modified-Since"
- exact: 精确匹配
- before: 文件修改时间应小于请求头中的"If-Modified-Since"时间

index

语法: index file [file...]

默认值: index index.html

使用字段: http, server, location

这个参数哪个（些）文件将被用于主页，可以在文件名中使用变量，一个包含绝对路径的文件可以把它放到末尾，例:

```
index  index.$geo.html  index.0.html  /index.html;
```

如果你想自动生成一个目录下的文件列表的主页，可以用autoindex on指令。

internal

语法: internal

默认值: no

使用字段: location

internal指令指定某个location只能被“内部的”请求调用，外部的调用请求会返回"Not found" (404)

“内部的”是指下列类型:

- 指令error_page重定向的请求。
- ngx_http_ssi_module模块中使用include virtual指令创建的某些子请求。
- ngx_http_rewrite_module模块中使用rewrite指令修改的请求。

一个防止错误页面被用户直接访问的例子:

```
error_page 404 /404.html;
location  /404.html {
  internal;
}
```

keepalive_timeout

语法: keepalive_timeout [time] [time]

默认值: keepalive_timeout 75

使用字段: http, server, location

参数的第一个值指定了客户端与服务器长连接的超时时间，超过这个时间，服务器将关闭连接。
参数的第二个值（可选）指定了HTTP头中Keep-Alive: timeout=time的time值，这个值可以帮助一些浏览器关闭连接，以便服务器不用重复关闭，如果不指定这个参数，nginx不会在http头中发送Keep-Alive信息。（但这并不是指怎样将一个连接“Keep-Alive”）
参数的这两个值可以不相同
下面列出了一些服务器如何处理包含Keep-Alive的头:

- MSIE和Opera将Keep-Alive: timeout=N头忽略。
- MSIE保持一个活动的连接60-65秒，然后发送一个TCP RST。
- Opera将一直保持一个连接处于活动状态。
- Mozilla将一个连接在timeout=N的基础上增加大约1-10秒。
- Konqueror保持一个连接大约N秒。

keepalive_requests

语法: keepalive_requests n
默认值: keepalive_requests 100
使用字段: http, server, location
服务器保持长连接的请求数。

large_client_header_buffers

语法: large_client_header_buffers number size
默认值: large_client_header_buffers 4 4k/8k
使用字段: http, server
指令指定客户端请求的一些比较大的头文件到缓冲区的最大值，如果一个请求的URI大小超过这个值，服务器将返回一个"Request URI too large" (414)，同样，如果一个请求的头部字段大于这个值，服务器将返回"Bad request" (400)。
缓冲区根据需求的不同是分开的。
默认一个缓冲区大小为操作系统中分页文件大小，通常是4k或8k，如果一个连接请求将状态转换为keep-alive，这个缓冲区将被释放。

limit_except

语法：limit_except methods {...}

默认值：no

使用字段：location

可以比较容易得在location字段中做一些http动作的限制。

ngx_http_access_module和ngx_http_auth_basic_module模块有很强的访问控制功能。

```
limit_except  GET {
  allow  192.168.1.0/32;
  deny   all;
}
```

limit_rate

语法：limit_rate speed

默认值：no

使用字段：http，server，location，location中的if字段

限制将应答包传送到客户端的速度，单位为字节/秒，限制仅对一个连接有效，即如果一个客户端打开2个连接，则它的速度是这个值乘以二。

基于一些不同的状况，如果要在server字段来限制一些连接的速度，那么这个参数并不适用，不过你可以选择设置¥limit_rate变量的值来达到目的：

```
server {
  if ($slow) {
    set $limit_rate  4k;
  }
}
```

同样可以通过设置X-Accel-Limit-Rate头 (NginxXSendfile) 来控制proxy_pass返回的应答。不依赖于X-Accel-Redirect头。

limit_rate_after

语法：limit_rate_after time

默认值：limit_rate_after 1m

使用字段：http，server，location，location中的if字段

这个指令和上面的一起使用：

```
limit_rate_after 1m;
limit_rate 100k;
```

listen

语法: listen address:port [default [backlog=num | rcvbuf=size | sndbuf=size | accept_filter=filter | deferred | bind | ssl]]

默认值: listen 80

使用字段: server

listen指令指定了server{...}字段中可以被访问到的ip地址及端口号，可以只指定一个ip，一个端口，或者一个可解析的服务器名。

```
listen 127.0.0.1:8000;
listen 127.0.0.1;
listen 8000;
listen *:8000;
listen localhost:8000;
```

ipv6地址格式（0.7.36）在一个方括号中指定:

```
listen [::]:8000;
listen [fe80::1];
```

当linux（相对于FreeBSD）绑定IPv6[::]，那么它同样将绑定相应的IPv4地址，如果在一些非ipv6服务器上仍然这样设置，可能会绑定失败，当然你可以使用完整的地址来代替[::]，也可以使用"default ipv6only=on"选项来制定这个listen字段仅绑定ipv6地址，注意这个选项仅对这行listen生效，而不影响server块中指定的其他listen字段。

```
listen [2a02:750:5::123]:80;
listen [::]:80 default ipv6only=on;
```

只有ip地址需要指定，端口默认为80。

如果指令有default参数，那么这个server块将是一个“地址:端口”对的默认服务器，这对于你想为那些不匹配server_name指令中的主机名指定默认server块的基于域名的虚拟主机非常有用，如果没有指令带有default参数，那么默认服务器将使用第一个出现“IP:端口”对的server块。

listen允许为系统调用的listen(2)和bind(2)传递一些其他的参数，这些参数必须用在default参数之后:

backlog=num -- 指定调用listen(2)时backlog的值，默认会被-1。

rcvbuf=size -- 为正在监听的端口指定SO_RCVBUF的值。

sndbuf=size -- 为正在监听的端口指定SO_SNDBUF的值。

accept_filter=filter -- 为accept-filter指定一个名称。

· 仅用于FreeBSD，它可能有两个过滤器，dataready与httpready，仅在最终版本的FreeBSD (FreeBSD: 6.0, 5.4-STABLE与4.11-STABLE) 上，为他们发送-HUP信号可能会改变accept-filter。

deferred -- 在linux系统上延迟accept(2)并使用一个辅助的参数: TCP_DEFER_ACCEPT。

bind -- 指出必须使bind(2)分离构建。

· 主要指这里的“地址:端口”对，实际上如果这些指令描述为监听同一个端口，但是每个不同的地址和每条不同的指令均监听的是这个端口的所有地址，那么nginx只将bind(2)构建于*:port。这种情况下必须考虑并且确定哪个地址上有连接到达，完成系统调用getsockname()。但是如果使用了parameters backlog, rcvbuf, sndbuf, accept_filter或deferred这些参数，那么将总是将这个“地址:端口”对分离。

ssl -- 这个参数 (0.7.14) 并不关联listen(2)和bind(2)系统调用。

· 被指定这个参数的listen将被允许工作在SSL模式，这将允许服务器同时工作在HTTP和HTTPS两种协议下，例如:

```
listen  80;
listen  443 default ssl;
```

一个使用这些参数的完整例子:

```
listen  127.0.0.1 default accept_filter=dataready backlog=1024;
```

0.8.21版本以后nginx可以监听unix套接口:

```
listen unix:/tmp/nginx1.sock;
```

location

语法: location [=|~|~*|^~|@] /uri/ { ... }

默认值: no

使用字段: server

这个参数根据URI的不同需求来进行配置，可以使用字符串与正则表达式匹配，如果要使用正则表达式，你必须指定下列前缀:

1、~* 不匹配大小写。

2、~ 匹配大小写。

要确定该指令匹配特定的查询，程序将首先对字符串进行匹配，字符串匹配将作为查询的开始，最确切的匹配

将被使用。然后，正则表达式的匹配查询开始，匹配查询的第一个正则表达式找到后会停止搜索，如果没有找到正则表达式，将使用字符串的搜索结果。

在一些操作系统，如Mac OS X和Cygwin，字符串将通过不区分大小写的方式完成匹配（0.7.7），但是，比较仅限于单字节的语言环境。

正则表达式可以包含捕获（0.7.40），并用于其它指令中。

可以使用“^~”标记禁止在字符串匹配后检查正则表达式，如果最确切的匹配location有这个标记，那么正则表达式不会被检查。

使用“=”标记可以在URI和location之间定义精确的匹配，在精确匹配完成后并不进行额外的搜索，例如有请求“/”发生，则可以使用“location = /”来加速这个处理。

即使没有“=”和“^~”标记，精确的匹配location在找到后同样会停止查询。

下面是各种查询方式的总结：

1、前缀“=”表示精确匹配查询，如果找到，立即停止查询。

2、指令仍然使用标准字符串，如果匹配使用“^~”前缀，停止查询。

3、正则表达式按照他们在配置文件中定义的顺序。

4、如果第三条产生一个匹配，这个匹配将被使用，否则将使用第二条的匹配。

例：

```
location  = / {
  # 只匹配 / 的查询.
  [ configuration A ]
}
location  / {
  # 匹配任何以 / 开始的查询，但是正则表达式与一些较长的字符串将被首先匹配。
  [ configuration B ]
}
location ^~ /images/ {
  # 匹配任何以 /images/ 开始的查询并且停止搜索，不检查正则表达式。
  [ configuration C ]
}
location ~* \.(gif|jpg|jpeg)$ {
  # 匹配任何以gif, jpg, or jpeg结尾的文件，但是所有 /images/ 目录的请求将在Configuration C中处理。
  [ configuration D ]
}
```

各请求的处理如下例：

- / -> configuration A
- /documents/document.html -> configuration B
- /images/1.gif -> configuration C
- /documents/1.jpg -> configuration D

注意你可以以任何顺序定义这4个配置并且匹配结果是相同的，但当使用嵌套的location结构时可能会将配置

文件变的复杂并且产生一些比较意外的结果。

标记“@”指定一个命名的location，这种location并不会在正常请求中执行，它们仅使用在内部重定向请求中。（查看error_page和try_files）

log_not_found

语法：log_not_found [on|off]

默认值：log_not_found on

使用字段：http, server, location

这个指令指定是否将一些文件没有找到的错误信息写入error_log指定的文件中。

log_subrequest

语法：log_subrequest [on|off]

默认值：log_subrequest off

使用字段：http, server, location

这个指令指定是否将一些经过rewrite rules和/或SSI requests的子请求请求写入access_log指定的文件中。

msie_padding

语法：msie_padding [on|off]

默认值：msie_padding on

使用字段：http, server, location

这个指令指定开启或关闭MSIE浏览器和chrome浏览器（0.8.25+）的msie_padding特征，当这个功能开启，nginx将为响应实体分配最小512字节，以便响应大于或等于400的状态代码。

这个指令预防在MSIE和chrome浏览器中激活“友好的”HTTP错误页面，以便不在服务器端隐藏更多的错误信息。

msie_refresh

语法： msie_refresh [on|off]

默认值: msie_refresh off

使用字段: http, server, location

这个指令允许或拒绝为MSIE发布一个refresh而不是做一次redirect

open_file_cache

语法: open_file_cache max = N [inactive = time] | off

默认值: open_file_cache off

使用字段: http, server, location

这个指令指定缓存是否启用, 如果启用, 将记录文件以下信息:

· 打开的文件描述符, 大小信息和修改时间。

· 存在的目录信息。

· 在搜索文件过程中的错误信息 -- 没有这个文件、无法正确读取, 参考open_file_cache_errors

指令参数:

· max - 指定缓存的最大数目, 如果缓存溢出, 最长使用过的文件 (LRU) 将被移除。

· inactive - 指定缓存文件被移除的时间, 如果在这段时间内文件没被下载, 默认为60秒。

· off - 禁止缓存。

例:

```
open_file_cache max=1000 inactive=20s;
open_file_cache_valid    30s;
open_file_cache_min_uses 2;
open_file_cache_errors   on;
```

open_file_cache_errors

语法: open_file_cache_errors on | off

默认值: open_file_cache_errors off

使用字段: http, server, location

这个指令指定是否在搜索一个文件是记录cache错误。

open_file_cache_min_uses

语法: open_file_cache_min_uses number

默认值: open_file_cache_min_uses 1

使用字段: http, server, location

这个指令指定了在open_file_cache指令无效的参数中一定的时间范围内可以使用的最小文件数，如果使用更大的值，文件描述符在cache中总是打开状态。

open_file_cache_valid

语法: open_file_cache_valid time

默认值: open_file_cache_valid 60

使用字段: http, server, location

这个指令指定了何时需要检查open_file_cache中缓存项目的有效信息。

optimize_server_names

语法: optimize_server_names [on|off]

默认值: optimize_server_names on

使用字段: http, server

这个指令指定是否在基于域名的虚拟主机中开启最优化的主机名检查。

尤其是检查影响到使用主机名的重定向，如果开启最优化，那么所有基于域名的虚拟主机监听的一个“地址:端口对”具有相同的配置，这样在请求执行的时候并不进行再次检查，重定向会使用第一个server name。

如果重定向必须使用主机名并且在客户端检查通过，那么这个参数必须设置为off。

注意: 这个参数不建议在nginx 0.7.x版本中使用，请使用server_name_in_redirect。

port_in_redirect

语法: port_in_redirect [on|off]

默认值: port_in_redirect on

使用字段: http, server, location

这个指令指定是否在让nginx在重定向操作中对端口进行操作。

如果这个选项打开，在重定向请求中nginx不会在url中添加端口。

recursive_error_pages

语法: recursive_error_pages [on|off]

默认值: recursive_error_pages off

使用字段: http, server, location

recursive_error_pages指定启用除第一条error_page指令以外其他的error_page。

resolver

语法: resolver address

默认值: no

使用字段: http, server, location

指定DNS服务器地址, 如:

```
resolver 127.0.0.1;
```

resolver_timeout

语法: resolver_timeout time

默认值: 30s

使用字段: http, server, location

解析超时时间。如:

```
resolver_timeout 5s;
```

root

语法: root path

默认值: root html

使用字段: http, server, location ,location中的if字段

请求到达后的文件根目录。

下例中:

```
location  /i/ {
```

```
  root   /spool/w3;
}
```

如果请求"/i/top.gif"文件，nginx将转到"/spool/w3/i/top.gif"文件。你可以在参数中使用变量。

注意：在请求中root会添加这个location到它的值后面，即"/i/top.gif"并不会请求"/spool/w3/top.gif"文件，如果要实现上述类似于apache alias的功能，可以使用alias指令。

satisfy_any

语法：satisfy_any [on|off]

默认值：satisfy_any off

使用字段：location

指令可以检查至少一个成功的密码效验，它在NginxHttpAccessModule或NginxHttpAuthBasicModule这两个模块中执行。

```
location / {
  satisfy_any  on;
  allow  192.168.1.0/32;
  deny   all;
  auth_basic            "closed site";
  auth_basic_user_file  conf/htpasswd;
}
```

send_timeout

语法：send_timeout the time

默认值：send_timeout 60

使用字段：http, server, location

指令指定了发送给客户端应答后的超时时间，Timeout是指没有进入完整established状态，只完成了两次握手，如果超过这个时间客户端没有任何响应，nginx将关闭连接。

sendfile

语法：sendfile [on|off]

默认值：sendfile off

使用字段：http, server, location

是否启用sendfile()函数。

server

语法：server {...}

默认值：no

使用字段：http

server字段包含虚拟主机的配置。

没有明确的机制来分开基于域名（请求中的主机头）和基于IP的虚拟主机。

可以通过listen指令来指定必须连接到这个server块的所有地址和端口，并且在server_name指令中可以指定所有的域名。

server_name

语法：server_name name [...]

默认值：server_name hostname

使用字段：server

这个指令有两个作用：

·将HTTP请求的主机头与在nginx配置文件中的server{...}字段中指定的参数进行匹配，并且找出第一个匹配结果。这就是如何定义虚拟主机的方法，域名遵循下述优先级规则：

1、完整匹配的名称。

2、名称开始于一个文件通配符：*.example.com。

3、名称结束于一个文件通配符：www.example.*。

4、使用正则表达式的名称。

如果没有匹配的结果，nginx配置文件将安装以下优先级使用[#server server { ... }]字段：

1、listen指令被标记为default的server字段。

2、第一个出现listen（或者默认的listen 80）的server字段。

·如果server_name_in_redirect被设置，这个指令将用于设置HTTP重定向的服务器名。

例：

```
server {
  server_name   example.com  www.example.com;
}
```

第一个名称为服务器的基本名称，默认名称为机器的hostname。

当然，可以使用文件通配符：

```
server {
  server_name   example.com  *.example.com  www.example.*;
}
```

上述例子中的前两个名称可以合并为一个：

```
server {
  server_name  .example.com;
}
```

同样可以使用正则表达式。名称前面加“~”：

```
server {
  server_name   www.example.com   ~^www\d+\.example\.com$;
}
```

如果客户端请求中没有主机头或者没有匹配server_name的主机头，服务器基本名称将被用于一个HTTP重定向，你可以只使用“*”来强制nginx在HTTP重定向中使用Host头（注意*不能用于第一个名称，不过你可以用一个很傻逼的名称代替，如“_”）

```
server {
  server_name example.com *;
}
server {
  server_name _ *;
}
```

在nginx0.6.x中有稍许改变：

```
server {
  server_name _;
}
```

0.7.12版本以后已经可以支持空服务器名，以处理那些没有主机头的请求：

```
server {
  server_name "";
}
```

server_name_in_redirect

语法：server_name_in_redirect on|off

默认值：server_name_in_redirect on

使用字段：http, server, location

如果这个指令打开，nginx将使用server_name指定的基本服务器名作为重定向地址，如果关闭，nginx将使用请求中的主机头。

server_names_hash_max_size

语法: server_names_hash_max_size number

默认值: server_names_hash_max_size 512

使用字段: http

服务器名称哈希表的最大值，更多信息请参考nginx Optimizations。

server_names_hash_bucket_size

语法: server_names_hash_bucket_size number

默认值: server_names_hash_bucket_size 32/64/128

使用字段: http

服务器名称哈希表每个页框的大小，这个指令的默认值依赖于cpu缓存。更多信息请参考nginx Optimizations。

server_tokens

语法: server_tokens on|off

默认值: server_tokens on

使用字段: http, server, location

是否在错误页面和服务器头中输出nginx版本信息。

tcp_nodelay

语法: tcp_nodelay [on|off]

默认值: tcp_nodelay on

使用字段: http, server, location

这个指令指定是否使用socket的TCP_NODELAY选项，这个选项只对keep-alive连接有效。

点击这里了解更多关于TCP_NODELAY选项的信息。

tcp_nopush

语法: tcp_nopush [on|off]

默认值: tcp_nopush off

使用字段: http, server, location

这个指令指定是否使用socket的TCP_NOPUSH (FreeBSD) 或TCP_CORK (linux) 选项, 这个选项只在使用sendfile时有效。

设置这个选项的将导致nginx试图将它的HTTP应答头封装到一个包中。

点击这里查看关于TCP_NOPUSH和TCP_CORK选项的更多信息。

try_files

语法: try_files file1 [file2 ... filen] fallback

默认值: none

使用字段: location

这个指令告诉nginx将测试参数中每个文件的存在, 并且URI将使用第一个找到的文件, 如果没有找到文件, 将请求名为fallback (可以使任何名称) 的location字段, fallback是一个必须的参数, 它可以是一个命名的location或者可靠的URI。

例:

```
location / {
  try_files index.html index.htm @fallback;
}

location @fallback {
  root /var/www/error;
  index index.html;
}
```

types

语法: types {...}

使用字段: http, server, location

这个字段指定一些扩展的文件对应方式与应答的MIME类型, 一个MIME类型可以有一些与其类似的扩展。默认使用以下文件对应方式:

```
types {
  text/html    html;
  image/gif    gif;
  image/jpeg   jpg;
}
```

完整的对应视图文件为conf/mime.types，并且将被包含。

如果你想让某些特定的location的处理方式使用MIME类型：application/octet-stream，可以使用以下配置：

```
location /download/ {
  types         { }
  default_type  application/octet-stream;
}
```

·变量

core module 支持一些内置的变量，与apache使用的变量相一致。

首先，一些变量是代表了客户端请求标题的字段，如：¥http_user_agent，¥http_cookie等等。

除此之外，下列是一些其他变量：

¥arg_PARAMETER

这个变量包含在查询字符串时GET请求PARAMETER的值。

¥args

这个变量等于请求行中的参数。

¥binary_remote_addr

二进制码形式的客户端地址。

¥body_bytes_sent

未知。

¥content_length

请求头中的Content-length字段。

¥content_type

请求头中的Content-Type字段。

¥cookie_COOKIE

cookie COOKIE的值。

¥document_root

当前请求在root指令中指定的值。

¥document_uri

与¥uri相同。

¥host

请求中的主机头字段，如果请求中的主机头不可用，则为服务器处理请求的服务器名称。

¥is_args

如果¥args设置，值为"?"，否则为""。

¥limit_rate

这个变量可以限制连接速率。

¥query_string

与¥args相同。

¥remote_addr

客户端的IP地址。

¥remote_port

客户端的端口。

¥remote_user

已经经过Auth Basic Module验证的用户名。

¥request_filename

当前连接请求的文件路径，由root或alias指令与URI请求生成。

¥request_body

这个变量（0.7.58+）包含请求的主要信息。在使用proxy_pass或fastcgi_pass指令的location中比较有意义。

¥request_body_file

客户端请求主体信息的临时文件名。

¥request_completion

未知。

¥request_method

这个变量是客户端请求的动作，通常为GET或POST。
包括0.8.20及之前的版本中，这个变量总为main request中的动作，如果当前请求是一个子请求，并不使用这个当前请求的动作。

¥request_uri

这个变量等于包含一些客户端请求参数的原始URI，它无法修改，请查看¥uri更改或重写URI。

¥scheme

HTTP方法（如http，https）。按需使用，例：

```
rewrite  ^(.+)$  $scheme://example.com$1  redirect;
```

¥server_addr

服务器地址，在完成一次系统调用后可以确定这个值，如果要绕开系统调用，则必须在listen中指定地址并且使用bind参数。

¥server_name

服务器名称。

¥server_port

请求到达服务器的端口号。

¥server_protocol

请求使用的协议，通常是HTTP/1.0或HTTP/1.1。

¥uri

请求中的当前URI(不带请求参数，参数位于¥args)，可以不同于浏览器传递的¥request_uri的值，它可以通过内部重定向，或者使用index指令进行修改。

·参考文档

Original Documentation

Nginx Http Core Module

前进->HTTP负载均衡模块 (HTTP Upstream)

HTTP负载均衡模块（HTTP Upstream）

回目录

·摘要

这个模块为后端的服务器提供简单的负载均衡（轮询（round-robin）和连接IP（client IP））

如下例：

```
upstream backend  {
  server backend1.example.com weight=5;
  server backend2.example.com:8080;
  server unix:/tmp/backend3;
}

server {
  location / {
    proxy_pass  http://backend;
  }
}
```

·指令

ip_hash

语法：ip_hash

默认值：none

使用字段：upstream

这个指令将基于客户端连接的IP地址来分发请求。

哈希的关键字是客户端的C类网络地址，这个功能将保证这个客户端请求总是被转发到一台服务器上，但是如果这台服务器不可用，那么请求将转发到另外的服务器上，这将保证某个客户端有很大概率总是连接到一台服务器。

无法将权重（weight）与ip_hash联合使用来分发连接。如果有某台服务器不可用，你必须标记其为“down”，如下例：

```
upstream backend {
  ip_hash;
  server   backend1.example.com;
```

```
  server   backend2.example.com;
  server   backend3.example.com  down;
  server   backend4.example.com;
}
```

server

语法: server name [parameters]

默认值: none

使用字段: upstream

指定后端服务器的名称和一些参数，可以使用域名，IP，端口，或者unix socket。如果指定为域名，则首先将其解析为IP。

· weight = NUMBER – 设置服务器权重，默认为1。

· max_fails = NUMBER – 在一定时间内（这个时间在fail_timeout参数中设置）检查这个服务器是否可用时产生的最多失败请求数，默认为1，将其设置为0可以关闭检查，这些错误在proxy_next_upstream或fastcgi_next_upstream（404错误不会使max_fails增加）中定义。

· fail_timeout = TIME – 在这个时间内产生了max_fails所设置大小的失败尝试连接请求后这个服务器可能不可用，同样它指定了服务器不可用的时间（在下一次尝试连接请求发起之前），默认为10秒，fail_timeout与前端响应时间没有直接关系，不过可以使用proxy_connect_timeout和proxy_read_timeout来控制。

· down – 标记服务器处于离线状态，通常和ip_hash一起使用。

· backup – (0.6.7或更高)只用于本服务器，如果所有的非备份服务器都宕机或繁忙。

示例配置

```
upstream  backend  {
  server   backend1.example.com    weight=5;
  server   127.0.0.1:8080          max_fails=3  fail_timeout=30s;
  server   unix:/tmp/backend3;
}
```

upstream

语法: upstream name { ... }

默认值: none

使用字段: http

这个字段设置一群服务器，可以将这个字段放在proxy_pass和fastcgi_pass指令中作为一个单独的实体，它们可以可以是监听不同端口的服务器，并且也可以是同时监听TCP和Unix socket的服务器。

服务器可以指定不同的权重，默认为1。

示例配置

```
upstream backend {
  server backend1.example.com weight=5;
  server 127.0.0.1:8080       max_fails=3  fail_timeout=30s;
  server unix:/tmp/backend3;
}
```

请求将按照轮询的方式分发到后端服务器，但同时也会考虑权重。

在上面的例子中如果每次发生7个请求，5个请求将被发送到backend1.example.com，其他两台将分别得到一个请求，如果有一台服务器不可用，那么请求将被转发到下一台服务器，直到所有的服务器检查都通过。如果所有的服务器都无法通过检查，那么将返回给客户端最后一台工作的服务器产生的结果。

· 变量

版本0.5.18以后，可以通过log_module中的变量来记录日志:

```
log_format timing '$remote_addr - $remote_user [$time_local]  $request '
  'upstream_response_time $upstream_response_time '
  'msec $msec request_time $request_time';

log_format up_head '$remote_addr - $remote_user [$time_local]  $request '
  'upstream_http_content_type $upstream_http_content_type';
```

¥upstream_addr

前端服务器处理请求的服务器地址

¥upstream_cache_status

0.8.3版本中其值可能为:

· MISS

· EXPIRED - expired。请求被传送到后端。

· UPDATING - expired。由于proxy/fastcgi_cache_use_stale正在更新，将使用旧的应答。

· STALE - expired。由于proxy/fastcgi_cache_use_stale，后端将得到过期的应答。

· HIT

¥upstream_status

前端服务器的响应状态。

¥upstream_response_time

单位为毫秒，不同的值以逗号或冒号分开。

¥upstream_http_¥HEADER

随意的HTTP协议头，如：

```
$upstream_http_host
```

· 参考文档

Original Documentation

Nginx Http Upstream Module

前进->HTTP访问控制模块（HTTP Access）

HTTP访问控制模块（HTTP Access）

回目录

·摘要

这个模块提供简单的基于主机的访问控制。

ngx_http_access_module这个模块可以详细的检查客户端IP，并且按顺序执行第一条匹配的规则。

如下例：

```
location / {
  deny    192.168.1.1;
  allow   192.168.1.0/24;
  allow   10.1.1.0/16;
  deny    all;
}
```

上面的例子中仅允许192.168.1.0/24和10.1.1.0/16网络段访问，但192.168.1.1是个例外。

如果要实施很多复杂的规则，那么最好使用GeoIP module模块。

·指令

allow

语法：allow [address | CIDR | all]

默认值：no

使用字段：http, server, location, limit_except

指令指定了允许访问的IP或网络段。

deny

语法：deny [address | CIDR | all]

默认值：no

使用字段：http, server, location, limit_except

指令指定了拒绝访问的IP或网络段。

· 提示和技巧

HttpAccess模块可以和error_page指令搭配使用来重定向一个未经验证的访问请求。

```
error_page  403  http://example.com/forbidden.html;
location / {
  deny    192.168.1.1;
  allow   192.168.1.0/24;
  allow   10.1.1.0/16;
  deny    all;
}
```

· 参考文档

Original Documentation

Nginx Http Access Module

前进->HTTP基本认证模块 (HTTP Auth Basic)

HTTP基本认证模块 (HTTP Auth Basic)

回目录

· 摘要

这个模块提供基于用户名与密码的验证来保护你的站点或站点的一部分。

如下例:

```
location  /  {
  auth_basic            "Restricted";
  auth_basic_user_file  conf/htpasswd;
}
```

· 指令

auth_basic

语法: auth_basic [text|off]

默认值: auth_basic off

使用字段: http, server, location, limit_except

指令包含一个具有测试用户名和密码的HTTP基本认证，指定的参数将用于认证域。如果将值设置为“off”则忽略下级指令继承的动作。

auth_basic_user_file

语法: auth_basic_user_file the_file

默认值: no

使用字段: http, server, location, limit_except

指令为验证域指定了密码文件，0.6.7版本以后这里指定的文件是nginx.conf所在目录的相对路径，而不是--prefix指定的路径。

这个文件格式如下:

```
user:pass
user2:pass2:comment
user3:pass3
```

密码字段必须经过crypt(3)函数加密，你可以使用Apache的htpasswd程序生成密码。

· 参考文档

Original Documentation

Nginx Http Auth Basic Module

前进->HTTP目录清单生成模块 (HTTP Auto Index)

HTTP目录清单生成模块（HTTP Auto Index）

回目录

· 摘要

这个模块提供自动目录列表。

连接请求仅在ngx_http_index_module中没有找到主页文件时才会请求这个模块。

如下例:

```
location  /  {
  autoindex  on;
}
```

· 指令

autoindex

语法: autoindex [on|off]

默认值: autoindex off

使用字段: http, server, location

是否使用自动目录列表。

autoindex_exact_size

语法: autoindex_exact_size [on|off]

默认值: autoindex_exact_size on

使用字段: http, server, location

指定生成的自动目录文件大小，可以是精确到bytes或者使用KB, MB或GB。

autoindex_localtime

语法: autoindex_localtime [on|off]

默认值: autoindex_localtime off

使用字段: http, server, location

是否在目录列表文件中显示本地时间（GMT时间），默认为关。

· 参考文档

Original Documentation

Nginx Auto Index Module

前进->浏览器相关模块（Browser）

浏览器相关模块（Browser）

回目录

· 摘要

这个模块按照请求头中的“User-agent”来创建一些变量：

· $modern_browser – 如果浏览器被识别为一个当前流行的浏览器，这个值等于指令modern_browser_value指定的值。

· $ancient_browser – 如果浏览器被识别为一个比较旧的浏览器，这个值等于指令ancient_browser_value指定的值。

· $msie – 如果浏览器被识别为MSIE，这个值为1。

如果不需要这个模块，可以在编译nginx时增加--without-http_browser_module参数。

配置实例：

为指定的浏览器指定主页文件：

```
modern_browser_value "modern.";
modern_browser msie 5.5;
modern_browser gecko 1.0.0;
modern_browser opera 9.0;
modern_browser safari 413;
modern_browser konqueror 3.0;
index index.${modern_browser}html index.html;
```

将一些来自比较旧的浏览器重定向：

```
modern_browser msie 5.0;
modern_browser gecko 0.9.1;
modern_browser opera 8.0;
modern_browser safari 413;
modern_browser konqueror 3.0;
modern_browser unlisted;
ancient_browser Links Lynx Netscape4;

if ($ancient_browser){
  rewrite  ^  /ancient.html;
}
```

· 指令

ancient_browser

语法: ancient_browser line [line...]

默认值: no

使用字段: http, server, location

在"User-agent"字段中的浏览器被识别为旧浏览器时，这个指令指定一些子链。

一个比较特殊的字段是"netscape4"，它对应正则表达式"^Mozilla/[1-4] "。

ancient_browser_value

语法: ancient_browser_value line

默认值: ancient_browser_value 1

使用字段: http, server, location

指定变量¥ancient_browser的值。

modern_browser

语法: modern_browser browser version|unlisted

默认值: no

使用字段: http, server, location

指令将指定哪个版本的浏览器将被认为是目前流行的。

可以指定的浏览器为: msie, gecko (基于Mozilla的浏览器) opera, safari, konqueror。

可以使用的版本格式为X, X.X, X.X.X, 或X.X.X.X，每个的最大值为4000, 4000.99, 4000.99.99,和4000.99.99.99。

一个特殊的值"unlisted"在被认为是流行的浏览器中指定，而不是通过modern_browser或ancient_browser指令。

如果请求头中没有"User-agent"字段，那么这个浏览器被认为是古老的 (除非指定“modern_browser unlisted”) 。

modern_browser_value

语法: modern_browser_value line

默认值: modern_browser_value 1

使用字段: http, server, location

指定¥modern_browser变量的值。

· 示例配置

示例配置: 仅支持最近版本的Chrome, Firefox, Internet Explorer, Safari, Mobile Safari和Palm Pre。

· 参考文档

Original Documentation

Nginx Http Browser Module

前进->字符集设置模块 (Charset)

字符集设置模块（Charset）

回目录

· 摘要

这个模块将在应答头中为"Content-Type"字段添加字符编码。

此外，这个模块可以将数据重新编码，只能在单向对其进行重新编码，即来自服务器到达客户端。

配置实例：

```
charset         windows-1251;
source_charset  koi8-r;
```

· 指令

charset

语法：charset encoding|off

默认值：charset off

使用字段：http，server，location，location中的if字段

这个指令使应答头中的"Content-Type"字段使用指定的编码集，如果这个字符集与source_charset指令设置的字符集不相同，将重新编码字符集，参数off表明不在应答头中添加"Content-Type"信息。

charset_map

语法：charset_map encoding1 encoding2 {...}

默认值：no

使用字段：http，server，location

charset_map指定了一个编码转换表，同时会创建一个反向转换表，代码均使用十六进制，如果在80-FF范围内没有被记录的代码，它们将被标记为"？"。

如下例：

```
charset_map  koi8-r  windows-1251 {
  C0  FE ; # small yu
  C1  E0 ; # small a
  C2  E1 ; # small b
  C3  F6 ; # small ts
  # ...
}
```

将koi8-r转换为Windows-1251的完整转换表为conf/koi-win。

override_charset

语法: override_charset on|off

默认值: override_charset off

使用字段: http, server, location, if中的location字段

参数指定在代理服务器或者FastCGI服务器上取得的应答头中存在"Content-Type"字段，将为应答启用编码转换，如果允许编码转换，将使用应答头中指定的编码对其初始化。

注意如果是在一个子查询中取得的应答，会始终将应答中的编码转换为基础编码，并不依赖于override_charset指令。

source_charset

语法: source_charset encoding

默认值: no

使用字段: http, server, location, if中的location字段

参数指定了应答中的初始代码，如果这个参数与charset指令中的不同，将启用编码转换。

· 参考文档

Original Documentation

Nginx Http Charset Module

前进->Empty GIF模块 (Empty GIF)

Empty GIF模块（Empty GIF）

回目录

·摘要

这个模块在内存中保存一个能够很快传递的1×1透明GIF。

简单用法：

```
location = /_.gif {
  empty_gif;
}
```

·指令

empty_gif

语法：empty_gif

默认值：n/a

使用字段：location

·参考文档

Original Documentation

Nginx Http Empty GIF Module

前进->FastCGI模块（FastCGI）

FastCGI模块 (FastCGI)

回目录

· 摘要

这个模块允许nginx同FastCGI协同工作，并且控制哪些参数将被安全传递。

例:

```
location / {
  fastcgi_pass   localhost:9000;
  fastcgi_index  index.php;

  fastcgi_param  SCRIPT_FILENAME  /home/www/scripts/php$fastcgi_script_name;
  fastcgi_param  QUERY_STRING     $query_string;
  fastcgi_param  REQUEST_METHOD   $request_method;
  fastcgi_param  CONTENT_TYPE     $content_type;
  fastcgi_param  CONTENT_LENGTH   $content_length;
}
```

一个在缓存中的实例:

```
http {
  fastcgi_cache_path   /path/to/cache  levels=1:2
                       keys_zone=NAME:10m
                       inactive=5m     clean_time=2h00m;

  server {
    location / {
      fastcgi_pass    http://127.0.0.1;
      fastcgi_cache   NAME;
      fastcgi_cache_valid   200 302  1h;
      fastcgi_cache_valid   301      1d;
      fastcgi_cache_valid   any      1m;
      fastcgi_cache_min_uses  1;
      fastcgi_cache_use_stale error  timeout invalid_header http_500;
    }
  }
}
```

0.7.48以后，缓存遵循后端服务器的Cache-Control, Expires等。

· 指令

fastcgi_buffers

语法: fastcgi_buffers the_number is_size;

默认值: fastcgi_buffers 8 4k/8k;

使用字段: http, server, location

这个参数指定了从FastCGI服务器到来的应答，本地将用多少和多大的缓冲区读取，默认这个参数等于分页大小，根据环境的不同可能是4K, 8K或16K。

fastcgi_buffer_size

语法: fastcgi_buffer_size the_size ;

默认值: fastcgi_buffer_size 4k/8k ;

使用字段: http, server, location

这个参数指定将用多大的缓冲区来读取从FastCGI服务器到来应答的第一部分。

通常来说在这个部分中包含一个小的应答头。

默认这个值为fastcgi_buffers指令中的每块大小，可以将这个值设置更小。

fastcgi_cache

语法: fastcgi_cache zone;

默认值: off

使用字段: http, server, location

为缓存实际使用的共享内存指定一个区域，相同的区域可以用在不同的地方。

fastcgi_cache_key

语法: fastcgi_cache_key line ;

默认值: none

使用字段: http, server, location

设置缓存的关键字，如:

```
fastcgi_cache_key localhost: 9000 $ request_uri;
```

fastcgi_cache_methods

语法: fastcgi_cache_methods [GET HEAD POST];

默认值: fastcgi_cache_methods GET HEAD;

使用字段: main,http,location

无法禁用GET/HEAD ，即使你只是这样设置:

```
fastcgi_cache_methods  POST;
```

fastcgi_cache_min_uses

语法: fastcgi_cache_min_uses n

默认值: fastcgi_cache_min_uses 1

使用字段: http, server, location

未知。

fastcgi_cache_path

语法: fastcgi_cache_path /path/to/cache [levels=m:n keys_zone=name:time inactive=time clean_time=time]

默认值: none

使用字段: http, server, location

未知。

fastcgi_cache_use_stale

语法: fastcgi_cache_use_stale [updating|error|timeout|invalid_header|http_500]

默认值: fastcgi_cache_use_stale off;

使用字段: http, server, location

未知。

fastcgi_cache_valid

语法: fastcgi_cache_valid [http_error_code|time]

默认值: none

使用字段: http, server, location

未知。

fastcgi_index

语法: fastcgi_index file

默认值: none

使用字段: http, server, location

如果URI以斜线结尾，参数将加到URI后面，这个值将存储在变量¥fastcgi_script_name中。

fastcgi_hide_header

语法: fastcgi_hide_header name

使用字段: http, server, location

默认情况下nginx不会将来自FastCGI服务器的"Status"和"X-Accel-..."头传送到客户端，这个参数也可以隐藏某些其它的头。

如果必须传递"Status"和"X-Accel-..."头，则必须使用fastcgi_pass_header强制其传送到客户端。

fastcgi_ignore_client_abort

语法: fastcgi_ignore_client_abort on|off

默认值: fastcgi_ignore_client_abort off

使用字段: http, server, location

如果当前连接请求FastCGI服务器失败，为防止其与nginx服务器断开连接，可以用这个指令。

fastcgi_intercept_errors

语法: fastcgi_intercept_errors on|off

默认值: fastcgi_intercept_errors off

使用字段: http, server, location

这个指令指定是否传递4xx和5xx错误信息到客户端，或者允许nginx使用error_page处理错误信息。

你必须明确的在error_page中指定处理方法使这个参数有效，正如Igor所说“如果没有适当的处理方法，nginx不会拦截一个错误，这个错误不会显示自己的默认页面，这里允许通过某些方法拦截错误。

fastcgi_max_temp_file_size

语法：fastcgi_max_temp_file_size 0

默认值：？

使用字段：？

根据源代码关闭FastCGI缓冲。

fastcgi_param

语法：fastcgi_param parameter value

默认值：none

使用字段：http, server, location

指定一些传递到FastCGI服务器的参数。

可以使用字符串，变量，或者其组合，这里的设置不会继承到其他的字段，设置在当前字段会清除掉任何之前的定义。

下面是一个PHP需要使用的最少参数：

```
  fastcgi_param  SCRIPT_FILENAME  /home/www/scripts/php$fastcgi_script_name;
  fastcgi_param  QUERY_STRING     $query_string;
```

PHP使用SCRIPT_FILENAME参数决定需要执行哪个脚本，QUERY_STRING包含请求中的某些参数。

如果要处理POST请求，则需要另外增加三个参数：

```
  fastcgi_param  REQUEST_METHOD   $request_method;
  fastcgi_param  CONTENT_TYPE     $content_type;
  fastcgi_param  CONTENT_LENGTH   $content_length;
```

如果PHP在编译时带有--enable-force-cgi-redirect，则必须传递值为200的REDIRECT_STATUS参数：

```
fastcgi_param  REDIRECT_STATUS  200;
```

fastcgi_pass

语法：fastcgi_pass fastcgi-server

默认值：none

使用字段: http, server, location

指定FastCGI服务器监听端口与地址，可以是本机或者其它:

```
fastcgi_pass   localhost:9000;
```

使用Unix socket:

```
fastcgi_pass   unix:/tmp/fastcgi.socket;
```

同样可以使用一个upstream字段名称:

```
upstream backend  {
  server   localhost:1234;
}

fastcgi_pass   backend;
```

fastcgi_pass_header

语法: fastcgi_pass_header name

默认值: none

使用字段: http, server, location

fastcgi_read_timeout

语法: fastcgi_read_timeout time

默认值: 60

使用字段: http, server, location

前端FastCGI服务器的响应超时时间，如果有一些直到它们运行完才有输出的长时间运行的FastCGI进程，或者在错误日志中出现前端服务器响应超时错误，可能需要调整这个值。

fastcgi_redirect_errors

语法: fastcgi_redirect_errors on|off

将来自fastcgi_intercept_errors的错误重命名。参数传递到FastCGI服务器。

请求头以参数的形式传递到FastCGI服务器，在FastCGI的应用程序或者脚本中运行，这些参数通常是环境变量的形式，例如: User-agent头是以HTTP_USER_AGENT形式转发，另外可以借助fastcgi_param指令可以传递任意参数的HTTP请求头。

fastcgi_split_path_info

语法: fastcgi_split_path_info regex

使用字段: location

可用版本: 0.7.31以上

```
location ~ ^(.+\.php)(.*)$ {
...
fastcgi_split_path_info ^(.+\.php)(.*)$;
fastcgi_param SCRIPT_FILENAME /path/to/php$fastcgi_script_name;
fastcgi_param PATH_INFO $fastcgi_path_info;
fastcgi_param PATH_TRANSLATED $document_root$fastcgi_path_info;
...
}
```

fastcgi_store

语法: fastcgi_store [on | off | path]

默认值: fastcgi_store off

使用字段: http, server, location

制定了存储前端文件的路径，参数on指定了将使用root和alias指令相同的路径，off禁止存储，此外，参数中可以使用变量使路径名更明确:

```
fastcgi_store   /data/www$original_uri;
```

应答中的"Last-Modified"头将设置文件的最后修改时间，为了使这些文件更加安全，可以将其在一个目录中存为临时文件，使用fastcgi_temp_path指令。

这个指令可以用在为那些不是经常改变的后端动态输出创建本地拷贝的过程中。如:

```
location /images/ {
  root                /data/www;
  error_page          404 = /fetch$uri;
}

location /fetch {
  internal;

  fastcgi_pass           fastcgi://backend;
  fastcgi_store          on;
  fastcgi_store_access   user:rw  group:rw  all:r;
  fastcgi_temp_path      /data/temp;

  alias                  /data/www;
}
```

fastcgi_store并不是缓存，某些需求下它更像是一个镜像。

fastcgi_store_access

语法: fastcgi_store_access users:permissions [users:permission ...]

默认值: fastcgi_store_access user:rw

使用字段: http, server, location

这个参数指定创建文件或目录的权限，例如:

```
fastcgi_store_access  user:rw  group:rw  all:r;
```

如果要指定一个组的人的相关权限，可以不写用户，如:

```
fastcgi_store_access  group:rw  all:r;
```

· 变量

¥fastcgi_script_name

这个变量等于一个以斜线结尾的请求URI加上fastcgi_index给定的参数。可以用这个变量代替SCRIPT_FILENAME 和PATH_TRANSLATED，以确定php脚本的名称。

如下例，请求"/info/":

```
fastcgi_index  index.php;
 fastcgi_param  SCRIPT_FILENAME  /home/www/scripts/php$fastcgi_script_name;
```

SCRIPT_FILENAME等于"/home/www/scripts/php/info/index.php"。

· 参考文档

Original Documentation

Nginx Http FastCGI Module

前进->Geo模块 (Geo)

Geo模块（Geo）

回目录

· 摘要

这个模块创建一些变量，其值依赖于客户端的IP地址：

如下例：

```
geo  $geo  {
  default          0;
  127.0.0.1/32     2;
  192.168.1.0/24   1;
  10.1.0.0/16      1;
}
```

· 指令

geo

语法：geo [¥ip_variable] ¥variable { ... }

默认值：none

使用字段：http

这个指令指定了一个客户端IP的所属国家，默认情况下它会查找¥remote_addr变量，但在0.7.27版本以后可以手工指定。

```
 geo  $arg_remote_addr $geo {
   ...;
 }
```

使用CIDR地址格式，另外，有4个特殊的参数：

· delete – 删除指定的网络（0.7.23）

· default – 将一些没有定义的地址替换为0.0.0.0/0。

· include – 具有地址信息的文本文件，可以包含多个。

· proxy – 指定代理服务器地址（0.8.7）。

· ranges – 指定使用以地址池的形式定义地址(0.7.23)，这个参数必须放在首位。

```
geo  $country  {
  default          no;
  include          conf/geo.conf;
  127.0.0.0/24     us;
  127.0.0.1/32     ru;
  10.1.0.0/16      ru;
  192.168.1.0/24   uk;
}
```

conf/geo.conf文件内容:

```
10.2.0.0/16      ru;
192.168.2.0/24   ru;
```

该值将使用最大的参数, 例如127.0.0.1将为“ru”,而不是“us”。

一个使用ranges的例子:

```
geo  $country  {
  ranges;
  default                     no;
  127.0.0.0-127.0.0.0         us;
  127.0.0.1-127.0.0.1         ru;
  127.0.0.1-127.0.0.255       us;
  10.1.0.0-10.1.255.255       ru;
  192.168.1.0-192.168.1.255   uk;
}
```

· 参考文档

HWLoadbalancerCheckErrors

Creating geo.conf From MaxMind GeoIP Country Database

Original Documentation

Nginx Http Geo Module

前进->Gzip压缩模块 (Gzip)

Gzip模块（Gzip）

回目录

·摘要

这个模块允许在文件传输过程中使用gzip压缩。

如下例:

```
gzip              on;
gzip_min_length   1000;
gzip_proxied      expired no-cache no-store private auth;
gzip_types        text/plain application/xml;
gzip_disable      "MSIE [1-6]\.";
```

可以使用¥gzip_ratio变量指定压缩比率。

·指令

gzip

语法: gzip on|off

默认值: gzip off

使用字段: http, server, location, location中的if字段

指定是否启用gzip压缩。

gzip_buffers

语法: gzip_buffers number size

默认值: gzip_buffers 4 4k/8k

使用字段: http, server, location

指定缓存压缩应答的缓冲区数量和大小，如果不设置，一个缓存区的大小为分页大小，根据环境的不同可能是4k或8k。

gzip_comp_level

语法: gzip_comp_level 1..9

默认值: gzip_comp_level 1

使用字段: http, server, location

指定压缩等级，其值从1到9，1为最小化压缩（处理速度快），9为最大化压缩（处理速度慢）。

gzip_disable

语法: gzip_disable regex

使用字段: http, server, location

使用正则表达式来指定某些不需要gzip压缩的浏览器（将和User-Agents进行匹配）。依赖于PCRE库。在0.6.23版本中首次使用。

0.7.63版本以后，你可以为IE5.5和IE6 SP1使用msie6参数来禁止gzip压缩。

```
gzip_disable     "msie6";
```

gzip_http_version

语法: gzip_http_version 1.0|1.1

默认值: gzip_http_version 1.1

使用字段: http, server, location

是否根据HTTP请求版本来启用gzip压缩。

当HTTP版本为1.0时，Vary: Accept-Encoding没有被设置，这将引起某些代理缓存失效，可以使用add_header，同样，在使用这个版本时Content-Length也没有设置，因此Keepalive不能在这个版本使用。

gzip_min_length

语法: gzip_min_length length

默认值: gzip_min_length 0

使用字段: http, server, location

设置被压缩的最小请求，少于这个值的请求将不会被压缩，这个值由请求头中的Content-Length字段决定。

gzip_proxied

语法: gzip_proxied [off|expired|no-cache|no-store|private|no_last_modified|no_etag|auth|any] ...

默认值: gzip_proxied off

使用字段: http, server, location

根据某些请求和应答来决定是否在对代理请求的应答启用压缩，事实上，代理请求取决于请求头中的“Via”字段，指令中可以同时指定多个不同的参数:

- off – 为所有代理请求禁用压缩。
- expired – 当“Expires”头禁用缓存时启用压缩。
- no-cache – 当“Cache-Control”头设置为no-cache时启用压缩。
- no-store – 当“Cache-Control”头设置为no-store时启用压缩。
- private – 当“Cache-Control”头设置为private时启用压缩。
- no_last_modified – 当“Last-Modified”没有定义时启用压缩。
- no_etag – 没有“ETag”头时启用压缩。
- auth – 当有一个“Authorization”头时启用压缩。
- any – 为所有请求启用压缩。

gzip_types

gzip_types mime-type [mime-type ...]

默认值: gzip_types text/html

使用字段: http, server, location

为除“text/html”之外的MIME类型启用压缩，“text/html”总是会被压缩。

gzip_vary

ggzip_vary on|off

默认值: gzip_vary off

使用字段: http, server, location

启用应答头“Vary: Accept-Encoding”，注意，由于一个bug将导致IE 4-6无法缓存内容。

· 参考文档

Original Documentation

Nginx Http Gzip Module

前进->HTTP头处理模块 (HTTP Headers)

HTTP头处理模块（HTTP Headers）

回目录

· 摘要

这个模块允许设置任意的HTTP头。

如下例:

```
  expires       24h;
  expires       0;
  expires       -1;
  expires       epoch;
  add_header    Cache-Control  private;
```

· 指令

add_header

语法: add_header name value

默认值: none

使用字段: http, server, location

当服务器应答代码为200, 204, 301, 302或304时为HTTP应答添加头。

这个值可以使用变量

注意这个指令只会在输出的头部中增加某个新字段，而并不能对某些已经定义的头（如server）进行重写，如果要实现这个操作可以使用第三方模块headers_more。

expires

语法: expires [time|epoch|max|off]

默认值: expires off

使用字段: http, server, location

在应答头中是否开启对“Expires”和“Cache-Control”的增加和修改操作。

可以指定一个正或负的时间值，Expires头中的时间根据目前时间和指令中指定的时间的和来获得。

epoch表示自1970年一月一日00:00:01 GMT的绝对时间，max指定Expires的值为2037年12月31日23:59:59，Cache-Control的值为10 years。

Cache-Control头的内容随预设的时间标识指定:

· 设置为负数的时间值:Cache-Control: no-cache。

· 设置为正数或0的时间值：Cache-Control: max-age = #，这里#的单位为秒，在指令中指定。

参数off禁止修改应答头中的"Expires"和"Cache-Control"。

注意：expires仅仅适用于200，204，301，302,和304应答

· 参考文档

Original Documentation

Nginx Http Headers Module

使用第三方模块来增加，修改或清除输入和输出的头部

前进->默认主页设置模块（Index）

默认主页设置模块（Index）

回目录

· 摘要

如果URL中没有指定文件，则设置一个默认主页。

如下例：

```
  index index.html;
```

可以指定多个文件，如果第一个文件没有找到，将会查找后面指定的文件：

```
  index index.html index.htm;
```

· 指令

index

语法：index file-path [file-path [...]];

默认值：no

使用字段：server, location

· 参考文档

Nginx Http Index Module

前进->HTTP Referer模块（HTTP Referer）

HTTP Referer模块 (HTTP Referer)

回目录

· 摘要

当一个请求头的Referer字段中包含一些非正确的字段，这个模块可以禁止这个请求访问站点。

这个头可以随意的伪造，因此，使用这个模块并不能100%的阻止这些请求，绝大多数拒绝的请求来自一些典型的浏览器，可以认为这些典型的浏览器并不能提供一个"Referer"头，甚至是那些正确的请求。

如下例:

```
 location /photos/ {
  valid_referers none blocked www.mydomain.com mydomain.com;

  if ($invalid_referer) {
    return   403;
  }
}
```

· 指令

valid_referers

语法: valid_referers [none|blocked|server_names] ...

默认值: none

使用字段: server, location

这个指令在referer头的基础上为 ¥invalid_referer 变量赋值，其值为0或1。

可以使用这个指令来实现防盗链功能，如果valid_referers列表中没有Referer头的值， ¥invalid_referer将被设置为1（参照前例）。

参数可以使如下形式:

· none意为不存在的Referer头

· blocked意为根据防火墙伪装Referer头，如: “Referer: XXXXXXX”。

· server_names为一个或多个服务器的列表，0.5.33版本以后可以在名称中使用“*”通配符。

· 参考文档

Original Documentation

Nginx Http Referer Module

前进->HTTP Limit Zone模块 (HTTP Limit Zone)

HTTP Limit Zone模块 (HTTP Limit Zone)

回目录

· 摘要

这个模块可以为一个地址指定的会话或者某些特殊情况限制同时连接数,

如下例:

```
http {
  limit_zone   one  $binary_remote_addr  10m;

  server {
    location /download/ {
      limit_conn   one  1;
    }
  }
}
```

· 指令

limit_zone

语法: limit_zone zone_name ¥variable memory_max_size

默认值: no

使用字段: http

指令描述会话状态存储区域。

会话的数目按照指定的变量来决定, 它依赖于使用的变量大小和memory_max_size的值。

如下例:

```
limit_zone one $binary_remote_addr 10m;
```

客户端的地址将用于会话, 注意¥binary_remote_addr变量将替换¥remote_addr而被使用。

¥remote_addr 变量的值的长度可以是7到15字节, 因此大小指定为32或64字节。

¥binary_remote_addr 变量的值的长度总是4字节, 大小总是32字节。

当会话状态储存区域为1M时理论上可以处理32000个会话, 每个会话大小为32字节。

limit_conn

语法: limit_conn zone_name max_clients_per_ip

默认值: no

使用字段: http, server, location

指令指定一个会话的最大同时连接数，超过这个数字的请求将被返回"Service unavailable" (503)代码。

如下例:

```
limit_zone   one  $binary_remote_addr  10m;

server {
  location /download/ {
  limit_conn   one  1;
}
```

这将指定一个地址只能同时存在一个连接。

· 参考文档

Original Documentation

Nginx Http Limit Zone Module

前进->HTTP Limit Requests模块 (HTTP Limit Requests)

HTTP Limit Requests模块 (HTTP Limit Requests)

回目录

· 摘要

这个模块允许为一个指定的会话或者某个特殊情况限制请求数目。

· 示例配置

```
http {
   limit_req_zone  $binary_remote_addr  zone=one:10m   rate=1r/s;

   ...

   server {

       ...

       location /search/ {
           limit_req   zone=one  burst=5;
       }
```

· 指令

语法: limit_req_zone ¥session_variable zone=name_of_zone:size rate=rate

默认值: none

使用字段: http

指令描述会话状态存储区域。

指令描述会话状态存储的某个区域，会话的值根据给定的变量，如下例:

```
limit_req_zone  $binary_remote_addr  zone=one:10m   rate=1r/s;
```

在这种情况下，将为一个名为“one”的区域分配10MB，这个区域的平均查询速度为每秒最多1个请求。

会话将追踪每个用户，但是注意它替换了变量¥remote_addr，我们使用的是¥binary_remote_addr，减少会话的大小为64字节，一个1MB的区域可以包含大约16000个会话状态。

速度可以设置为每秒处理请求数和每分钟处理请求数，其值必须是整数，所以如果你需要指定每秒处理少于1个的请求，2秒处理一个请求，可以使用 "30r/m"。

当会话状态储存区域为1M时理论上可以处理32000个会话，每个会话大小为32字节。

语法： limit_req zone=zone burst=burst [nodelay]

默认值：none

使用字段：http, server, location

这个指令指定区域（zone）可能的最大请求爆发值(burst),如果其值超过这个数，请求被延时，以便查询按照给定的速度处理。多余的请求将被延迟直到他们的数目小于burst值，在这种情况下，请求将得到"Service unavailable" (503)代码，默认burst的值为0。

如下例：

```
limit_req_zone  $binary_remote_addr  zone=one:10m   rate=1r/s;

    server {
        location /search/ {
            limit_req   zone=one  burst=5;
        }
```

允许一个用户平均每秒处理不超过1个请求，这个区域最多同时处理不超过5个查询，如果在burst值之外的额外请求不可用，可以使用nodelay参数：

```
limit_req   zone=one  burst=5  nodelay;
```

· 参考文档

Nginx Http Limit Requests Module

前进->日志模块（Log）

日志模块（Log）

回目录

· 摘要

控制nginx如何记录请求日志。

例:

```
log_format gzip '$remote_addr - $remote_user [$time_local]  '
                '"$request" $status $bytes_sent '
                '"$http_referer" "$http_user_agent" "$gzip_ratio"';

access_log  /spool/logs/nginx-access.log  gzip  buffer=32k;
```

关于记录nginx错误日志请参考HTTP核心模块。

· 指令

access_log

语法: access_log path [format [buffer=size | off]]

默认值: access_log log/access.log combined

使用字段: http, server, location

参数为连接日志指定了路径，格式和缓冲区大小。使用“off”将在当前的字段中清除access_log的所有参数，如果没有指定日志格式，默认为“combined”。缓冲区大小不能超过写入磁盘文件的最小大小。

日志文件路径可以包含变量（0.7.4以上版本），但是有一些限制:

· nginx指定的用户必须有创建日志文件的权限。

· 缓冲区不会工作

· 每个到来的连接，日志文件将被打开并且在记录日志后迅速关闭，然而，频繁使用的文件描述符将被保存到open_log_file_cache中，关于日志的轮询记录，必须记住随着时间的过去（使用open_log_file_cache的valid参数设置），日志仍然在旧的文件中记录。

nginx支持为每个location指定强大的日志记录。同样的连接可以在同一时间输出到不止一个的日志中，更多

信息请查看Multiple access_log directives in different contexts

log_format

语法: log_format name format [format ...]

默认值: log_format combined "..."

使用字段: http server

描述记录日志的格式，格式中可以使用大多数变量，也包括一些在写入日志文件过程中定义的变量:

·¥body_bytes_sent，减去应答头后传送给客户端的字节数，这个变量兼容apache模块mod_log_config的%B参数（在0.3.10前这个变量为¥apache_bytes_sent）。

·¥bytes_sent，传送给客户端的字节数。

·¥connection，连接数。

·¥msec，正在写入日志条目的当前时间（精确到百万分之一秒）

·¥pipe，如果请求为管道的。

·¥request_length，请求主体的长度。

·¥request_time，从一个请求发出到而使nginx工作的时间，单位为毫秒（0.5.19版本后可以使用秒为单位）。

·¥status，应答的状态（代码）。

·¥time_local，写入普通日志格式的当地时间（服务器时间）。

传送到客户端的头中的变量以"sent_http_"标记开头，如: ¥sent_http_content_range。

注意其他模块产生的变量同样可以写入日志，例如你可以记录前端负载均衡应答头使用“upstream_http_”开头的变量，具体请查看负载均衡模块。

nginx有一个预定的日志格式称为combined:

```
log_format combined '$remote_addr - $remote_user [$time_local]  '
                    '"$request" $status $body_bytes_sent '
                    '"$http_referer" "$http_user_agent"';
```

open_log_file_cache

语法: open_log_file_cache max=N [inactive=time] [min_uses=N] [valid=time] | off

默认值: open_log_file_cache off

使用字段: http server location

这个指令为频繁使用的日志文件描述符所在的路径变量设置缓存。

指令选项:

· max – 缓存中存储的最大文件描述符数。

· inactive – 设置缓存中在某个时间段内没有使用的文件描述符将被移除，默认为10秒。

· min_uses – 在一定时间内（inactive指定），一个文件描述符最少使用多少次后被放入缓存，默认为1。

· valid – 设置检查同名文件存在的时间，默认是60秒。

· off – 关闭缓存。

· 参考文档

Original Documentation

Nginx Http Log Module

前进->Map模块（Map）

Map模块（Map）

回目录

·摘要

这个模块允许你分类或者同时映射多个值到多个不同值并储存到一个变量中。

如下例:

```
map  $http_host  $name  {
  hostnames;

  default          0;

  example.com      1;
  *.example.com    1;
  test.com         2;
  *.test.com       2;
  .site.com        3;
}
```

一个典型的使用映射的例子是代替一个含有很多服务器的/location或者重定向:

```
map $uri $new {
  default        http://www.domain.com/home/;

  /aa            http://aa.domain.com/;
  /bb            http://bb.domain.com/;
  /john          http://my.domain.com/users/john/;
}

server {
  server_name   www.domain.com;
  rewrite  ^    $new   redirect;
}
```

·指令

map

语法: map ¥var1 ¥var2 { ... }

默认值: none

使用字段: http

指定一个变量使用的映射表，有三个指定的参数:

· default – 指定如果没有匹配结果将使用的默认值。

· hostnames – 允许对类似主机名的值进行简单的查询匹配，第一个点后面的部分将用作精确的主机名。例如：

```
*.example.com  1;
```

而不是写成两个：

```
example.com    1;
*.example.com  1;
```

可以只写成一条：

```
.example.com   1;
```

· include – 包含一个含有映射值的文件，可以包含多个。

map_hash_max_size

语法：map_hash_max_size number

默认值：map_hash_max_size 2048

使用字段：http

这个指令设置映射表对应的哈希表的最大值，更多信息可以参考优化章节。

map_hash_bucket_size

语法：map_hash_bucket_size n

默认值：map_hash_bucket_size 32/64/128

使用字段：http

这个指令指定一个映射表中的变量在哈希表中的最大值，这个值取决于处理器的缓存，更多信息可以参考优化章节。

· 参考文档

Original Documentation

Nginx Http Map Module

前进->Memcached模块（Memcached）

Memcached模块（Memcached）

回目录

·摘要

使用这个模块简单的处理缓存，这个模块将不断的进行完善。

示例配置:

```
server {
  location / {
    set $memcached_key $uri;
    memcached_pass     name:11211;
    default_type       text/html;
    error_page         404 = /fallback;
  }

  location = /fallback {
    proxy_pass backend;
  }
}
```

在nginx0.7.x中:

```
server {
  location / {
    set $memcached_key $uri;
    memcached_pass     name:11211;
    default_type       text/html;
    error_page         404 @fallback;
  }

  location @fallback {
    proxy_pass backend;
  }
}
```

·指令

memcached_pass

语法: memcached_pass [name:port]

默认值: none

使用字段: http, server, location

后端需要在memcached中设置一些数据，memcached key为“"/uri?args”。
在0.5.9版本之后memcached key存储在变量¥memcached_key中。

memcached_connect_timeout

语法: memcached_connect_timeout [time]
默认值: 60000
使用字段: http, server, location
连接memcached的超时时间，单位为毫秒。

memcached_read_timeout

语法: memcached_read_timeout [time]
默认值: 60000
使用字段: http, server, location
读取memcached数据的超时时间，单位为毫秒。

memcached_send_timeout

语法: memcached_send_timeout [time]
默认值: 60000
使用字段: http, server, location
发送memcached数据的超时时间，单位为毫秒。

memcached_buffer_size

语法: memcached_buffer_size [size]
默认值: see getpagesize(2)
使用字段: http, server, location
发送/收到的缓冲区大小，单位是字节。

memcached_next_upstream

语法: memcached_next_upstream [error | timeout | invalid_response | not_found | off]

默认值: error timeout

使用字段: http, server, location

指定在哪种错误状态下请求将转发到另外的负载均衡服务器，仅当memcached_pass有两个或两个以上值的时候使用。

· 变量

¥memcached_key

memcached key的值。

· 参考文档

使用memcache加速nginx

Nginx Http Memcached Module 第三方模块memc

前进->HTTP代理模块 (HTTP Proxy)

HTTP代理模块（HTTP Proxy）

回目录

·摘要

这个模块可以转发请求到其他的服务器。

HTTP/1.0无法使用keepalive（后端服务器将为每个请求创建并且删除连接）。nginx为浏览器发送HTTP/1.1并为后端服务器发送HTTP/1.0，这样浏览器就可以为浏览器处理keepalive。

如下例：

```
location / {
  proxy_pass        http://localhost:8000;
  proxy_set_header  X-Real-IP  $remote_addr;
}
```

注意当使用http proxy模块（甚至FastCGI），所有的连接请求在发送到后端服务器之前nginx将缓存它们，因此，在测量从后端传送的数据时，它的进度显示可能不正确。

·指令

proxy_buffer_size

语法：proxy_buffer_size the_size

默认值：proxy_buffer_size 4k/8k

使用字段：http, server, location

设置从被代理服务器读取的第一部分应答的缓冲区大小。

通常情况下这部分应答中包含一个小的应答头。

默认情况下这个值的大小为指令proxy_buffers中指定的一个缓冲区的大小，不过可以将其设置为更小。

proxy_buffering

语法：proxy_buffering on|off

默认值: proxy_buffering on

使用字段: http, server, location

为后端的服务器启用应答缓冲。

如果启用缓冲，nginx假设被代理服务器能够非常快的传递应答，并将其放入缓冲区，可以使用proxy_buffer_size和proxy_buffers设置相关参数。

如果响应无法全部放入内存，则将其写入硬盘。

如果禁用缓冲，从后端传来的应答将立即被传送到客户端。

nginx忽略被代理服务器的应答数目和所有应答的大小，接受proxy_buffer_size所指定的值。

对于基于长轮询的Comet应用需要关闭这个指令，否则异步的应答将被缓冲并且Comet无法正常工作。

proxy_buffers

语法: proxy_buffers the_number is_size;

默认值: proxy_buffers 8 4k/8k;

使用字段: http, server, location

设置用于读取应答（来自被代理服务器）的缓冲区数目和大小，默认情况也为分页大小，根据操作系统的不同可能是4k或者8k。

proxy_busy_buffers_size

语法: proxy_busy_buffers_size size;

默认值: proxy_busy_buffers_size ["#proxy buffer size"] * 2;

使用字段: http, server, location, if

未知。

proxy_cache

语法: proxy_cache zone_name;

默认值: None

使用字段: http, server, location

设置一个缓存区域的名称，一个相同的区域可以在不同的地方使用。在0.7.48后，缓存遵循后端的Cache-

Control, Expires以及其他等。缓存依赖代理的缓冲区，如果proxy_buffers设置为off，将不会生效。

proxy_cache_key

语法: proxy_cache_key line;

默认值: ¥scheme¥proxy_host¥request_uri;

使用字段: http, server, location

指令指定了包含在缓存中的缓存关键字。

```
proxy_cache_key "$host$request_uri$cookie_user";
```

注意默认情况下服务器的主机名并没有包含到缓存关键字中，如果你为你的站点在不同的location中使用二级域，你可能需要在缓存关键字中包换主机名:

```
proxy_cache_key "$scheme$host$request_uri";
```

proxy_cache_path

语法: proxy_cache_path path [levels=number] keys_zone=zone_name:zone_size [inactive=time] [max_size=size];

默认值: None

使用字段: http

指令指定缓存的路径和一些其他参数，缓存的数据存储在文件中。缓存的文件名和key为代理URL的MD5码。levels参数指定缓存的子目录数，例如:

```
proxy_cache_path  /data/nginx/cache  levels=1:2   keys_zone=one:10m;
```

文件名类似于:

/data/nginx/cache/c/29/b7f54b2df7773722d382f4809d65029c

所有活动的key和元数据存储在共享的内存区域中，这个区域用keys_zone参数指定，如果在inactive参数指定的时间内缓存的数据没有被请求则被删除，默认inactive为10分钟。

cache manager进程控制磁盘的缓存大小，在max_size参数中定义，超过其大小后最少使用数据将被删除。

区域的大小按照缓存页面数的比例进行设置，一个页面（文件）的元数据大小按照操作系统来定，FreeBSD/i386下为64字节，FreeBSD/amd64下为128字节，当区域满了以后key将按照LRU（最近最少使用算法）进行处理。

proxy_cache_path和proxy_temp_path应该使用在相同的文件系统上。

proxy_cache_methods

语法: proxy_cache_methods [GET HEAD POST];

默认值: proxy_cache_methods GET HEAD;

使用字段: http, server, location

GET/HEAD用来装饰语句，即你无法禁用GET/HEAD即使你只使用下列语句设置:

```
proxy_cache_methods POST;
```

proxy_cache_min_uses

语法: proxy_cache_min_uses the_number;

默认值: proxy_cache_min_uses 1;

使用字段: http, server, location

多少次的查询后应答将被缓存，默认1。

proxy_cache_valid

语法: proxy_cache_valid reply_code [reply_code ...] time;

默认值: None

使用字段: http, server, location

为不同的应答设置不同的缓存时间，例如:

```
  proxy_cache_valid  200 302  10m;
  proxy_cache_valid  404      1m;
```

为应答代码为200和302的设置缓存时间为10分钟，404代码缓存1分钟。

如果只定义时间:

```
 proxy_cache_valid 5m;
```

那么只对代码为200，301和302的应答进行缓存。

同样可以使用any参数任何应答。

```
  proxy_cache_valid  200 302 10m;
  proxy_cache_valid  301 1h;
  proxy_cache_valid  any 1m;
```

proxy_cache_use_stale

语法: proxy_cache_use_stale
[error|timeout|updating|invalid_header|http_500|http_502|http_503|http_504|http_404|off] [...];
默认值: proxy_cache_use_stale off;
使用字段: http, server, location
这个指令告诉nginx何时从代理缓存中提供一个过期的响应，参数类似于proxy_next_upstream指令。
为了防止缓存失效（在多个线程同时更新本地缓存时），你可以指定'updating'参数，它将保证只有一个线程去更新缓存，并且在这个线程更新缓存的过程中其他的线程只会响应当前缓存中的过期版本。

proxy_connect_timeout

语法: proxy_connect_timeout timeout_in_seconds
使用字段: http, server, location
指定一个连接到代理服务器的超时时间，这个时间并不是指服务器传回页面的时间，而是proxy_read_timeout的声明。无论何时你的代理服务器都是正常运行的，但是如果服务器遇到一些状况（例如没有足够的线程去处理请求，请求将被放在一个连接池中延迟处理），那么这个声明无助于服务器去建立连接。

proxy_headers_hash_bucket_size

语法: proxy_headers_hash_bucket_size size;
默认值: proxy_headers_hash_bucket_size 64;
使用字段: http, server, location, if
设置哈希表中存储的每个数据大小（参考解释）。

proxy_headers_hash_max_size

语法: proxy_headers_hash_max_size size;
默认值: proxy_headers_hash_max_size 512;
使用字段: http, server, location, if

设置哈希表的最大值（参考解释）。

proxy_hide_header

语法: proxy_hide_header the_header

使用字段: http, server, location

nginx不对从被代理服务器传来的"Date", "Server", "X-Pad"和"X-Accel-..."应答进行转发，这个参数允许隐藏一些其他的头部字段，但是如果上述提到的头部字段必须被转发，可以使用proxy_pass_header指令，例如：需要隐藏MS-OfficeWebserver和AspNet-Version可以使用如下配置:

```
location / {
  proxy_hide_header X-AspNet-Version;
  proxy_hide_header MicrosoftOfficeWebServer;
}
```

当使用X-Accel-Redirect时这个指令非常有用。例如，你可能要在后端应用服务器对一个需要下载的文件设置一个返回头，其中X-Accel-Redirect字段即为这个文件，同时要有恰当的Content-Type，但是，重定向的URL将指向包含这个文件的文件服务器，而这个服务器传递了它自己的Content-Type，可能这并不是正确的，这样就忽略了后端应用服务器传递的Content-Type。为了避免这种情况你可以使用这个指令:

```
location / {
  proxy_pass http://backend_servers;
}

location /files/ {
  proxy_pass http://fileserver;
  proxy_hide_header Content-Type;
```

proxy_ignore_client_abort

语法: proxy_ignore_client_abort [on|off]

默认值: proxy_ignore_client_abort off

使用字段: http, server, location

防止在客户端自己终端请求的情况下中断代理请求。

proxy_ignore_headers

语法: proxy_ignore_headers name [name ...]

默认值: none

使用字段: http, server, location

这个指令(0.7.54+) 禁止处理来自代理服务器的应答。

可以指定的字段为"X-Accel-Redirect", "X-Accel-Expires", "Expires"或"Cache-Control"。

proxy_intercept_errors

语法: proxy_intercept_errors [on|off]

默认值: proxy_intercept_errors off

使用字段: http, server, location

使nginx阻止HTTP应答代码为400或者更高的应答。

默认情况下被代理服务器的所有应答都将被传递。

如果将其设置为on则nginx会将阻止的这部分代码在一个error_page指令处理，如果在这个error_page中没有匹配的处理方法，则被代理服务器传递的错误应答会按原样传递。

proxy_max_temp_file_size

语法: proxy_max_temp_file_size size;

默认值: proxy_max_temp_file_size 1G;

使用字段: http, server, location, if

当代理缓冲区过大时使用一个临时文件的最大值，如果文件大于这个值，将同步传递请求而不写入磁盘进行缓存。

如果这个值设置为零，则禁止使用临时文件。

proxy_method

语法: proxy_method [method]

默认值: None

使用字段: http, server, location

允许代理一些其他的HTTP访问方法。

如果指定这个指令，nginx只允许一个请求中含有单个HTTP访问方法的请求（在该指令中指定），所以对于代理Subversion这样的请求并不是很有用。

示例配置:

```
  proxy_method PROPFIND;
```

proxy_next_upstream

语法: proxy_next_upstream [error|timeout|invalid_header|http_500|http_502|http_503|http_504|http_404|off]

默认值: proxy_next_upstream error timeout

使用字段: http, server, location

确定在何种情况下请求将转发到下一个服务器:

- error – 在连接到一个服务器, 发送一个请求, 或者读取应答时发生错误。
- timeout – 在连接到服务器, 转发请求或者读取应答时发生超时。
- invalid_header – 服务器返回空的或者错误的应答。
- http_500 – 服务器返回500代码。
- http_502 – 服务器返回502代码。
- http_503 – 服务器返回503代码。
- http_504 – 服务器返回504代码。
- http_404 – 服务器返回404代码。
- off – 禁止转发请求到下一台服务器。

转发请求只发生在没有数据传递到客户端的过程中。

proxy_pass

语法: proxy_pass URL

默认值: no

使用字段: location, location中的if字段

确定需要代理的URL, 端口和socket。

可以使用主机名和端口, 例如:

```
proxy_pass http://localhost:8000/uri/;
```

也可以使用unix socket:

```
proxy_pass unix:/path/to/backend.socket:/uri/;
```

请求转发到服务器的URI部分（location字段的后面部分），将为proxy_pass指定的值的URI。例如你的location字段为/name/，proxy_pass为http://127.0.0.1，则请求/name/test将被转发至http://127.0.0.1/test。

但是对于上述规则有两个例外，那时将无法确定被替换的location：

location使用正则表达式。

在使用代理的location中利用rewrite指令改变URI，使用这个配置可以更加精确的处理请求（break）：

```
location  /name/ {
  rewrite      /name/([^/] +)  /users?name=$1  break;
  proxy_pass   http://127.0.0.1;
}
```

这些情况下URI并没有被映射传递。

此外，可能需要指明URI将使用同样的方式转发，因为它是来自客户端，而不是以处理过的形式发送。

在其工作过程中：

· 两个以上的斜杠将被替换为一个： "//" -- "/"；

· 删除引用的当前目录："/./" -- "/"；

· 删除引用的先前目录："/dir /../" -- "/"。

如果在服务器上必须以未经过处理的形式发送URI，那么在这个指令中必须使用未指定URI的URL：

```
location  /some/path/ {
  proxy_pass   http://127.0.0.1;
}
```

在指令中使用变量是一种比较特殊的情况：被请求的URL不会使用并且你必须完全手工标记URL。

这意味着下列的配置并不能让你方便的进入某个你想要的虚拟主机目录，代理总是将它转发到相同的URL（在一个server字段的配置）：

```
location / {
  proxy_pass
http://127.0.0.1:8080/VirtualHostBase/https/$server_name:443/some/path/VirtualHostRoot;
}
```

解决方法是使用rewrite和proxy_pass的组合：

```
location / {
  rewrite ^(.*)$ /VirtualHostBase/https/$server_name:443/some/path/VirtualHostRoot/$1 break;
  proxy_pass   http://127.0.0.1:8080;
}
```

这种情况下请求的URL将被重写， proxy_pass中的拖尾斜杠并没有实际意义。

proxy_pass_header

语法: proxy_pass_header the_name

使用字段: http, server, location

这个指令允许为应答转发一些隐藏的头部字段。

如:

```
location / {
  proxy_pass_header X-Accel-Redirect;
}
```

proxy_pass_request_body

语法: proxy_pass_request_body [on | off] ;

默认值: proxy_pass_request_body on;

使用字段: http, server, location

可用版本: 0.1.29

proxy_pass_request_headers

语法: proxy_pass_request_headers [on | off] ;

默认值: proxy_pass_request_headers on;

使用字段: http, server, location

可用版本: 0.1.29

proxy_redirect

语法: proxy_redirect [default|off|redirect replacement]

默认值: proxy_redirect default

使用字段: http, server, location

如果需要修改从被代理服务器传来的应答头中的"Location"和"Refresh"字段，可以用这个指令设置。

假设被代理服务器返回Location字段为: http://localhost:8000/two/some/uri/

这个指令:

```
proxy_redirect http://localhost:8000/two/ http://frontend/one/;
```

将Location字段重写为http://frontend/one/some/uri/。

在代替的字段中可以不写服务器名:

```
proxy_redirect http://localhost:8000/two/ /;
```

这样就使用服务器的基本名称和端口，即使它来自非80端口。

如果使用“default”参数，将根据location和proxy_pass参数的设置来决定。

例如下列两个配置等效:

```
location /one/ {
  proxy_pass       http://upstream:port/two/;
  proxy_redirect   default;
}

location /one/ {
  proxy_pass       http://upstream:port/two/;
  proxy_redirect   http://upstream:port/two/   /one/;
}
```

在指令中可以使用一些变量:

```
proxy_redirect   http://localhost:8000/    http://$host:$server_port/;
```

这个指令有时可以重复:

```
  proxy_redirect   default;
  proxy_redirect   http://localhost:8000/    /;
  proxy_redirect   http://www.example.com/   /;
```

参数off将在这个字段中禁止所有的proxy_redirect指令:

```
  proxy_redirect   off;
  proxy_redirect   default;
  proxy_redirect   http://localhost:8000/    /;
  proxy_redirect   http://www.example.com/   /;
```

利用这个指令可以为被代理服务器发出的相对重定向增加主机名:

```
proxy_redirect   /   /;
```

proxy_read_timeout

语法: proxy_read_timeout the_time

默认值: proxy_read_timeout 60

使用字段: http, server, location

决定读取后端服务器应答的超市时间，它决定nginx将等待多久时间来取得一个请求的应答。超时时间是指完成了两次握手后并且状态为established的超时时间，而不是所有的应答时间。

相对于proxy_connect_timeout，这个时间可以扑捉到一台将你的连接放入连接池延迟处理并且没有数据传送

的服务器，注意不要将此值设置太低，某些情况下代理服务器将花很长的时间来获得页面应答（如当接收一个需要很多计算的报表时），当然你可以设置多个不同的location。
如果被代理服务器在设置的时间内没有传递数据，nginx将关闭连接。

proxy_redirect_errors

不推荐使用，请使用 proxy_intercept_errors。

proxy_send_lowat

语法: proxy_send_lowat [on | off]
默认值: proxy_send_lowat off;
使用字段: http, server, location, if
设置SO_SNDLOWAT，这个指令仅用于FreeBSD。

proxy_send_timeout

语法: proxy_send_timeout time_in_seconds
默认值: proxy_send_timeout 60
使用字段: http, server, location
设置代理服务器转发请求的超时时间，同样指完成两次握手后的时间，如果超过这个时间代理服务器没有数据转发到被代理服务器，nginx将关闭连接。

proxy_set_body

语法: proxy_set_body [on | off]
默认值: proxy_set_body off;
使用字段: http, server, location, if
可用版本: 0.3.10。

proxy_set_header

语法: proxy_set_header header value

默认值: Host and Connection

使用字段: http, server, location

这个指令允许将发送到被代理服务器的请求头重新定义或者增加一些字段。

这个值可以是一个文本，变量或者它们的组合。

proxy_set_header在指定的字段中没有定义时会从它的上级字段继承。

默认只有两个字段可以重新定义:

```
proxy_set_header Host $proxy_host;
proxy_set_header Connection Close;
```

未修改的请求头“Host”可以用如下方式传送:

```
proxy_set_header Host $http_host;
```

但是如果这个字段在客户端的请求头中不存在，那么将没有数据转发被代理服务器。

这种情况下最好使用¥Host变量，它的值等于请求头中的"Host"字段或服务器名:

```
proxy_set_header Host $host;
```

此外，可以将被代理的端口与服务器名称一起传递:

```
proxy_set_header Host $host:$proxy_port;
```

如果设置为空字符串，则不会传递头部到后端，例如下列设置将禁止后端使用gzip压缩:

```
proxy_set_header  Accept-Encoding  "";
```

proxy_store

语法: proxy_store [on | off | path]

默认值: proxy_store off

使用字段: http, server, location

这个指令设置哪些传来的文件将被存储，参数"on"保持文件与alias或root指令指定的目录一致，参数"off"将关闭存储，路径名中可以使用变量:

```
proxy_store   /data/www$original_uri;
```

应答头中的"Last-Modified"字段设置了文件最后修改时间，为了文件的安全，可以使用proxy_temp_path指定一个临时文件目录。

这个指令为那些不是经常使用的文件做一份本地拷贝。从而减少被代理服务器负载。

```
location /images/ {
  root                /data/www;
  error_page          404 = /fetch$uri;
}

location /fetch {
  internal;
  proxy_pass          http://backend;
  proxy_store         on;
  proxy_store_access  user:rw  group:rw  all:r;
  proxy_temp_path     /data/temp;
  alias               /data/www;
}
```

或者通过这种方式:

```
location /images/ {
  root                /data/www;
  error_page          404 = @fetch;
}

location @fetch {
  internal;

  proxy_pass          http://backend;
  proxy_store         on;
  proxy_store_access  user:rw  group:rw  all:r;
  proxy_temp_path     /data/temp;

  root                /data/www;
}
```

注意proxy_store不是一个缓存，它更像是一个镜像。

proxy_store_access

语法: proxy_store_access users:permissions [users:permission ...]

默认值: proxy_store_access user:rw

使用字段: http, server, location

指定创建文件和目录的相关权限，如:

```
proxy_store_access  user:rw  group:rw  all:r;
```

如果正确指定了组和所有的权限，则没有必要去指定用户的权限:

```
proxy_store_access  group:rw  all:r;
```

proxy_temp_file_write_size

语法: proxy_temp_file_write_size size;

默认值: proxy_temp_file_write_size ["#proxy buffer size"] * 2;

使用字段: http, server, location, if

设置在写入proxy_temp_path时数据的大小，在预防一个工作进程在传递文件时阻塞太长。

proxy_temp_path

语法: proxy_temp_path dir-path [level1 [level2 [level3] ;

默认值: 在configure时由--http-proxy-temp-path指定

使用字段: http, server, location

类似于http核心模块中的client_body_temp_path指令，指定一个地址来缓冲比较大的被代理请求。

proxy_upstream_fail_timeout

0.5.0版本后不推荐使用，请使用http负载均衡模块中server指令的fail_timeout参数。

proxy_upstream_fail_timeout

0.5.0版本后不推荐使用，请使用http负载均衡模块中server指令的max_fails参数。

· 变量

该模块中包含一些内置变量，可以用于proxy_set_header指令中以创建头部。

¥proxy_host

被代理服务器的主机名与端口号。

¥proxy_host

被代理服务器的端口号。

¥proxy_add_x_forwarded_for

包含客户端请求头中的"X-Forwarded-For"，与¥remote_addr用逗号分开，如果没有"X-Forwarded-For"请求

头，则¥proxy_add_x_forwarded_for等于¥remote_addr。

· 参考文档

Original Documentation

Nginx Http Proxy Module

前进->URL重写模块（Rewrite）

URL重写模块（Rewrite）

回目录

·摘要

这个模块允许使用正则表达式重写URI（需PCRE库），并且可以根据相关变量重定向和选择不同的配置。
如果这个指令在server字段中指定，那么将在被请求的location确定之前执行，如果在指令执行后所选择的location中有其他的重写规则，那么它们也被执行。如果在location中执行这个指令产生了新的URI，那么location又一次确定了新的URI。
这样的循环可以最多执行10次，超过以后nginx将返回500错误。

·指令

break

语法：break
默认值：none
使用字段：server，location，if
完成当前设置的规则，停止执行其他的重写指令。
示例：

```
if ($slow) {
  limit_rate  10k;
  break;
}
```

if

语法：if（condition）{ ... }
默认值：none
使用字段：server，location

判断一个条件，如果条件成立，则后面的大括号内的语句将执行，相关配置从上级继承。

可以在判断语句中指定下列值：

· 一个变量的名称；不成立的值为：空字符传""或者一些用“0”开始的字符串。

· 一个使用=或者!=运算符的比较语句。

· 使用符号~*和~模式匹配的正则表达式：

· ~为区分大小写的匹配。

· ~*不区分大小写的匹配（firefox匹配FireFox）。

· !~和!~*意为“不匹配的”。

· 使用-f和!-f检查一个文件是否存在。

· 使用-d和!-d检查一个目录是否存在。

· 使用-e和!-e检查一个文件，目录或者软链接是否存在。

· 使用-x和!-x检查一个文件是否为可执行文件。

正则表达式的一部分可以用圆括号，方便之后按照顺序用¥1-¥9来引用。

示例配置：

```
if ($http_user_agent ~ MSIE) {
  rewrite  ^(.*)$  /msie/$1  break;
}

if ($http_cookie ~* "id=([^;] +)(?:;|$)" ) {
  set  $id  $1;
}

if ($request_method = POST ) {
  return 405;
}

if (!-f $request_filename) {
  break;
  proxy_pass  http://127.0.0.1;
}

if ($slow) {
  limit_rate  10k;
}

if ($invalid_referer) {
  return   403;
}

if ($args ~ post=140){
  rewrite ^ http://example.com/ permanent;
}
```

内置变量¥invalid_referer用指令valid_referers指定。

return

语法：return code

默认值：none

使用字段：server，location，if

这个指令结束执行配置语句并为客户端返回状态代码，可以使用下列的值：204，400，402-406，408，410，411，413，416与500-504。此外，非标准代码444将关闭连接并且不发送任何的头部。

rewrite

语法：rewrite regex replacement flag

默认值：none

使用字段：server，location，if

按照相关的正则表达式与字符串修改URI，指令按照在配置文件中出现的顺序执行。

注意重写规则只匹配相对路径而不是绝对的URL，如果想匹配主机名，可以加一个if判断，如：

```
if ($host ~* www\.(.*)) {
  set $host_without_www $1;
  rewrite ^(.*)$ http://$host_without_www$1 permanent; # $1为'/foo'，而不是'www.mydomain.com/foo'
}
```

可以在重写指令后面添加标记。

如果替换的字符串以http://开头，请求将被重定向，并且不再执行多余的rewrite指令。

标记可以是以下的值：

- last – 完成重写指令，之后搜索相应的URI或location。
- break – 完成重写指令。
- redirect – 返回302临时重定向，如果替换字段用http://开头则被使用。
- permanent – 返回301永久重定向。

注意如果一个重定向是相对的（没有主机名部分），nginx将在重定向的过程中使用匹配server_name指令的“Host”头或者server_name指令指定的第一个名称，如果头不匹配或不存在，如果没有设置server_name，将使用本地主机名，如果你总是想让nginx使用“Host”头，可以在server_name使用“*”通配符（查看http核心模块中的server_name）。例如：

```
rewrite  ^(/download/.*)/media/(.*)\..*$  $1/mp3/$2.mp3  last;
rewrite  ^(/download/.*)/audio/(.*)\..*$  $1/mp3/$2.ra   last;
return   403;
```

但是如果我们将其放入一个名为/download/的location中，则需要将last标记改为break，否则nginx将执行10次循环并返回500错误。

```
location /download/ {
```

```
  rewrite  ^(/download/.*)/media/(.*)\..*$  $1/mp3/$2.mp3  break;
  rewrite  ^(/download/.*)/audio/(.*)\..*$  $1/mp3/$2.ra   break;
  return   403;
}
```

如果替换字段中包含参数，那么其余的请求参数将附加到后面，为了防止附加，可以在最后一个字符后面跟一个问号：

```
rewrite  ^/users/(.*)$  /show?user=$1?  last;
```

注意：大括号（{和}），可以同时用在正则表达式和配置块中，为了防止冲突，正则表达式使用大括号需要用双引号（或者单引号）。例如要重写以下的URL：

```
/photos/123456
```

为：

```
/path/to/photos/12/1234/123456.png
```

则使用以下正则表达式（注意引号）：

```
rewrite  "/photos/([0-9] {2})([0-9] {2})([0-9] {2})" /path/to/photos/$1/$1$2/$1$2$3.png;
```

同样，重写只对路径进行操作，而不是参数，如果要重写一个带参数的URL，可以使用以下代替：

```
if ($args ^~ post=100){
  rewrite ^ http://example.com/new-address.html? permanent;
}
```

注意¥args变量不会被编译，与location过程中的URI不同（参考http核心模块中的location）。

set

语法：set variable value

默认值：none

使用字段：server, location, if

指令设置一个变量并为其赋值，其值可以是文本，变量和它们的组合。

uninitialized_variable_warn

语法：uninitialized_variable_warn on|off

默认值：uninitialized_variable_warn on

使用字段：http, server, location, if

开启或关闭在未初始化变量中记录警告日志。

事实上，rewrite指令在配置文件加载时已经编译到内部代码中，在解释器产生请求时使用。

这个解释器是一个简单的堆栈虚拟机，如下列指令：

```
location /download/ {
  if ($forbidden) {
    return   403;
  }
  if ($slow) {
    limit_rate  10k;
  }
  rewrite  ^/(download/.*)/media/(.*)\..*$  /$1/mp3/$2.mp3  break;
```

将被编译成以下顺序：

```
  variable $forbidden
  checking to zero
  recovery 403
  completion of entire code
  variable $slow
  checking to zero
  checkings of regular expression
  copying "/"
  copying $1
  copying "/mp3/"
  copying $2
  copying "..mpe"
  completion of regular expression
  completion of entire sequence
```

注意并没有关于limit_rate的代码，因为它没有提及ngx_http_rewrite_module模块，“if”块可以类似"location"指令在配置文件的相同部分同时存在。

如果¥slow为真，对应的if块将生效，在这个配置中limit_rate的值为10k。

指令：

```
rewrite  ^/(download/.*)/media/(.*)\..*$  /$1/mp3/$2.mp3  break;
```

如果我们将第一个斜杠括入圆括号，则可以减少执行顺序：

```
rewrite  ^(/download/.*)/media/(.*)\..*$  $1/mp3/$2.mp3  break;
```

之后的顺序类似如下：

```
  checking regular expression
  copying $1
  copying "/mp3/"
  copying $2
  copying "..mpe"
  completion of regular expression
  completion of entire code
```

· 参考文档

Original Documentation

Nginx Http Rewrite Module

前进->SSI模块（SSI）

SSI模块（SSI）

回目录

· 摘要

这个模块为处理服务器端包含（SSI）的输入提供一个过滤器，目前所支持的SSI命令并不完善。

如下例：

```
location / {
  ssi  on;
}
```

· 指令

ssi

语法：ssi [on | off]

默认值：ssi off

使用字段：http, server, location, location中的if字段

启用SSI处理。

注意如果启用SSI，那么Last-Modified头和Content-Length头不会传递。

ssi_silent_errors

语法：ssi_silent_errors [on|off]

默认值：ssi_silent_errors off

使用字段：http, server, location

如果在处理SSI的过程中出现“[an error occurred while processing the directive]”错误，禁止将其输出。

ssi_types

语法: ssi_types mime-type [mime-type ...]

默认值: ssi_types text/html

使用字段: http, server, location

默认只解析text/html类型，这个参数可以指定其他的MIME类型。

ssi_value_length

语法: ssi_value_length length

默认值: ssi_value_length 256

使用字段: http, server, location

定义允许SSI使用的参数值的长度。

SSI 命令

命令格式如下:

```
<!--#  command parameter1=value parameter2=value... -->
```

支持的SSI命令如下:

block

命令描述一个可以在include命令中使用的块，在块中可以存在SSI命令。

· name – 块的名称，如:

```
<!--# block name="one" -->
the silencer
<!--# endblock ---->
```

config

为SSI指定一些配置参数。

· errmsg – 在SSI处理过程中得到的错误字段，默认字符串为: [an error occurred while processing the directive]。

· timefmt – 时间字符串的格式，strftime(3)函数使用，默认为:

```
"%A, %d-%b-%Y %H:%M:%S %Z"
```

在时间中包含秒可以使用“%s”

echo

打印一个变量。

· var – 变量的名称。

· default – 如果变量为空，则显示这个字符串，默认为“none”。如:

```
<!--# echo var="name" default="no" -->
```

与下列等效:

```
<!--# if expr="$name" --><!--# echo var="name" --><!--# else -->no<!--# endif -->
```

if/elif/else/endif

根据条件包含其他字段或者指令，用法:

```
<!--# if expr="..." -->
...
<!--# elif expr="..." -->
...
<!--# else -->
...
<!--# endif -->
```

上述例子中只有一个嵌套为真。

· expr – 判定一个表达式，可以是变量:

```
<!--# if expr="$name" -->
```

比较字符串:

```
<!--# if expr="$name = text" -->
<!--# if expr="$name != text" -->
```

或者匹配正则:

```
<!--# if expr="$name = /text/" -->
<!--# if expr="$name != /text/" -->
```

如果使用变量，则用他们的值代替。

include

包含一个其他来源的记录。

· file – 包含一个文件，如:

```
<!--# include file="footer.html" -->
```

· virtual – 包含一个请求，如:

```
<!--# include virtual="/remote/body.php?argument=value" -->
```

"file"和"virtual"主要是一些历史性的差别，"file"等同于暗示"wait"选项的"virtual"，在某点上反映与apache等值，但是现在他们基本都处理同样的操作，都可以处理一个URI或者一个静态文件。

多个请求将并行发出，如果需要按顺序发出，请使用“wait”选项

· stub – 如果请求为空或返回一个错误后使用的默认块。

```
<!--# block name="one" --> <!--# endblock -->
<!--# include virtual="/remote/body.php?argument=value" stub="one" -->
```

· wait – 当设置为yes时，在当前的请求未完成之前剩余的SSI不会进行判定，例如:

```
<!--# include virtual="/remote/body.php?argument=value" wait="yes" -->
```

set

设置一个变量。

· var – 变量。

· value – 包含变量名变量的值，它们将被判定。

· 变量

¥date_local

本地时区的当前时间，选项 "timefmt"可以指定格式。

¥date_gmt

当前的格林尼治时间，选项 "timefmt"可以指定格式。

· 参考文档

Original Documentation

Nginx Http SSI Module

前进->User ID模块 (User ID)

User ID模块 (User ID)

回目录

· 摘要

模块ngx_http_userid_module为连接发布cookie，主要使用¥uid_got和¥uid_set变量，注意：¥uid_got无法¥uid_set在SSI中取得，因为SSI过滤模块工作在userid模块过滤之前。

这个模块相当于Apache的mod_uid模块。

示例配置：

```
userid          on;
userid_name     uid;
userid_domain   example.com;
userid_path     /;
userid_expires  365d;
userid_p3p      'policyref="/w3c/p3p.xml", CP="CUR ADM OUR NOR STA NID"';
```

· 指令

userid

语法：userid [on|v1|log||off]

默认值：userid off

使用字段：http, server, location

是否启用发出cookie或者记录到被请求的cookie：

· on – 启用版本2的cookie并记录。

· v1 – 启用版本1的cookie并记录。

· log – 不传送cookie，但是写入日志。

· off – 禁用cookie。

userid_domain

语法: userid_domain [name | none]

默认值: userid_domain none

使用字段: http, server, location

指定cookie的域名，参数“none”不对任何域名发出cookie。

userid_expires

语法: userid_expires [time | max]

默认值: none

使用字段: http, server, location

设置cookie的过期时间。

参数设置并发出浏览器对于cookie的实效时间，值“max”指定过期时间为: 2037年12月31日23:55:55 GMT，这是某些旧浏览器所能识别的最大时间。

userid_name

语法: userid_name name

默认值: userid_name uid

使用字段: http, server, location

设置cookie的名称。

userid_p3p

语法: userid_p3p line

默认值: none

使用字段: http, server, location

为和cookie一起传递的P3P头指定一个值。

userid_path

语法: userid_path path

默认值: userid_path /

使用字段: http, server, location

设置cookie路径。

userid_service

语法: userid_service number

默认值: userid_service address

使用字段: http, server, location

设置cookie发布的服务器地址，如果不设置，版本一的cookie将其设置为0，版本二将其设置为服务器IP。

· 参考文档

Original Documentation

Nginx Http User ID Module

前进->可选HTTP模块

Nginx可选HTTP模块（Optional HTTP modules）

回目录

如果要使用这些模块，则必须在编译时指定相关的编译参数。

4.14 Secure Link模块 (Secure Link)

4.15 Image Filter模块 (Image Filter)

HTTP Addition模块 (HTTP Addition)

回目录

· 摘要

这个模块可以在当前的location之前或者之后增加别的location。

它作为一个输出过滤器执行，包含到其他location中的主请求和子请求不会被完全缓冲，并且仍然以流的形式传递到客户端，因为最终应答体的长度在传递HTTP头的时候是未知的，HTTP的chunked编码总是在这里使用。

安装

默认情况下这个模块是没有编译的，如果要使用则需要在编译时指定下列参数:

```
./configure --with-http_addition_module
```

示例配置:

```
location / {
  add_before_body   /before_action;
  add_after_body    /after_action;
```

一些限制

在0.8.17版本中如果当前的location请求一个请求自身的子请求，包含的location不会被添加。如以下配置:

```
location /foo {
  add_before_body /bar;
}

location /bar {
  add_before_body  /baz;
}
```

连接到/foo的请求并不会将/baz对应的location添加到字段中。

同样注意，在定义需要包含的location时只能使用字符串，而不能使用变量，所以以下配置:

```
location / {
  set $before_action /before_action;
  add_before_body $before_action;
}
```

并不能正常工作（虽然在测试配置文件正确性的时候可以通过）

· 指令

add_before_body

语法: add_before_body uri

默认值: no

使用字段: http, server, location

在应答体的前面增加URI，为一个处理结果发出子请求。

add_after_body

语法: add_after_body uri

默认值: no

使用字段: http, server, location

在应答体的后面增加URI，为一个处理结果发出子请求。

addition_types

语法: addition_types mime-type [mime-type ...]

默认值: text/html

使用字段: http, server, location

指令（0.7.9）允许增加让该location处理的其他MIME类型（默认为"text/html"）。（0.8.17之前这个指令在源代码中被拼写成"addtion_types"，不过在0.8.17版本后修复了。）

· 参考文档

Original Documentation

Nginx Http Addition Module

前进->嵌入式Perl模块（Embedded Perl）

嵌入式Perl模块（Embedded Perl）

回目录

·摘要

这个模块允许nginx使用SSI调用perl或直接执行perl

在编译时安装模块

默认这个模块为不可用，如果想使用这个模块，则必须在编译时指定--with-http_perl_module。

系统必须有Perl 5.6.1以上版本。

已知的问题

这个模块并不完善，因此可能会出现一些bug如：

1、如果perl脚本执行延时操作，（如dns解析，数据库查询等。）那么运行perl脚本的工作进程将一直处于完全占用状态，因此需要perl脚本尽量简短，并且很快执行。

2、nginx在重载配置文件时可能导致内存溢出（通过′kill -HUP <pid>>′）

实例：

```
http {
  perl_modules  perl/lib;
  perl_require  hello.pm;

  perl_set  $msie6  '
  sub {
    my $r = shift;
    my $ua = $r->header_in("User-Agent");
    return "" if $ua =~ /Opera/;
    return "1" if $ua =~ / MSIE [6-9] \.\d+/;
    return "";
  }
 ';

  server {
    location / {
      perl  hello::handler;
    }
  }
}
```

hello.pm的内容:

```
package hello;
use nginx;

sub handler {
  my $r = shift;
  $r->send_http_header("text/html");
  return OK if $r->header_only;

  $r->print("hello!\n<br/>");
  $r->rflush;

  if (-f $r->filename or -d _) {
    $r->print($r->uri, " exists!\n");
  }

  return OK;
}

1;
__END__
```

· 指令

perl

语法: perl module::function | 'sub {...}'

默认值: no

使用字段: location

指出一个location中所要使用的perl函数。

perl_modules

语法: perl_modules path

默认值: no

使用字段: http

为perl模块增加额外的路径, 0.6.7版本以后这个路径与nginx.conf所在目录有关, 而不是编译时指定的perfix目录。

perl_require

语法: perl_require module

默认值: no

使用字段: http

可以指定多个perl_require指令。

perl_set

语法: perl_set module::function | 'sub {...}'

默认值: no

使用字段: http

未知。

· SSI中调用perl。

指令格式如下:

```
<!- # perl sub="module::function" arg="parameter1" arg="parameter2"... >
```

对象¥r的请求方法:

· ¥r->args - 返回请求的参数。

· ¥r->discard_request_body - 告诉nginx忽略请求主体。

· ¥r->filename - 更具URI的请求返回文件名。

¥r->has_request_body(function) - 如果没有请求主体, 返回0, 但是如果请求主体存在, 那么建立传递的函数并返回1, 在程序的最后, nginx将调用指定的处理器。例如:

```
package hello;

use nginx;

sub handler {
  my $r = shift;

  if ($r->request_method ne "POST") {
    return DECLINED;
  }

  if ($r->has_request_body(\&post)) {
    return OK;
  }

  return 400;
}

sub post {
  my $r = shift;
  $r->send_http_header;
```

```
  $r->print("request_body: \"", $r->request_body, "\"<br/>");
  $r->print("request_body_file: \"", $r->request_body_file, "\"<br/>\n");
  return OK;
}

1;

__END__
```

·¥r->header_in(header) – 检索一个HTTP请求头。

·¥r->header_only – 在我们只需要返回一个应答头时为真。

·¥r->header_out(header, value) – 设置一个应答头。

·¥r->internal_redirect(uri) – 使内部重定向到指定的URI，重定向仅在完成perl脚本后发生。

·¥r->print(args, ...) – 为客户端传送数据。

·¥r->request_body – 在请求主体未记录到一个临时文件时为客户返回这个请求主体。为了使客户端的请求主体保证在内存里，可以使用client_max_body_size限制它的大小并且为其使用的缓冲区指定足够的空间。

·¥r->request_body_file – 返回存储客户端需求主体的文件名，这个文件必须在请求完成后被删除，以便请求主体始终能写入文件，需要指定client_body_in_file_only为on。

·¥r->request_method – 返回请求的HTTP动作。

·¥r->remote_addr – 返回客户端的IP地址。

·¥r->rflush – 立即传送数据到客户端。

·¥r->sendfile(file [, displacement [, length]) – 传送给客户端指定文件的内容，可选的参数表明只传送数据的偏移量与长度，精确的传递仅在perl脚本执行完毕后生效。

·¥r->send_http_header(type) – 为应答增加头部，可选参数“type”在应答标题中确定Content-Type的值。

·¥r->sleep(milliseconds, handler) – 设置为请求在指定的时间使用指定的处理方法和停止处理，在此期间nginx将继续处理其他的请求，超过指定的时间后，nginx将运行安装的处理方法，注意你需要为处理方法通过一个reference，在处理器间转发数据你可以使用¥r->variable()。如下例:

```
package hello;

use nginx;

sub handler {
   my $r = shift;

  $r->discard_request_body;
  $r->variable("var", "OK");
  $r->sleep(1000, \&next);

  return OK;
}
```

```
sub next {
  my $r = shift;

  $r->send_http_header;
  $r->print($r->variable("var"));

  return OK;
}

1;
__END__
```

· ¥r->status(code) - 设置HTTP应答代码。

· ¥r->unescape(text) - 以%XX的形式编码text。

· ¥r->uri - 返回请求的URI。

· ¥r->variable(name[, value]) - 返回一个指定变量的值，变量为每个查询的局部变量。

· 参考文档

Original Documentation

Nginx Embedded Perl Module

前进->FLV模块 (FLV)

FLV模块 (FLV)

回目录

· 摘要

这个模块支持对FLV (flash) 文件的拖动播放。

nginx处理文件的过程为:

· 为请求的文件增加FLV头。

· 根据请求参数中的start=XXX转发文件。

使用这个模块必须在编译时指定--with-http_flv_module参数。

示例配置:

```
location ~ \.flv$ {
  flv;
}
```

· 指令

flv

语法: flv

默认值: None

使用字段: location

是否对这个location中的文件开启FLV。

· 参考文档

Original Documentation

Nginx Flv Stream Module

前进->Gzip Precompression模块 (Gzip Precompression)

Gzip Precompression模块（Gzip Precompression）

回目录

·摘要

这个模块在一个预压缩文件传送到开启Gzip压缩的客户端之前检查是否已经存在以“.gz”结尾的压缩文件，这样可以防止文件被重复压缩。

这个模块在0.6.24后可用，如果要使用它则需要在编译时指定如下参数：

```
./configure --with-http_gzip_static_module
```

示例配置：

```
gzip_static on;

gzip_http_version   1.1;
gzip_proxied        expired no-cache no-store private auth;
gzip_disable        "MSIE [1-6]\.";
gzip_vary           on;
```

·指令

gzip_static

语法：gzip_static on|off

默认值：gzip_static off

使用字段：http, server, location

启用模块，你需要确保压缩文件与未压缩文件的时间戳一致。

gzip_http_version

参考Gzip 压缩模块。

gzip_proxied

参考Gzip 压缩模块。

gzip_disable

参考Gzip 压缩模块。

gzip_vary

参考Gzip 压缩模块。

· 参考文档

Gzip 压缩模块

前进->Random Index模块 (Random Index)

Random Index模块（Random Index）

回目录

·摘要

从目录中选择一个随机主页：

示例配置：

```
location  /  {
  random_index  on;
}
```

·指令

random_index

语法：random_index [on|off]

默认值：off

使用字段：location

如果在指定的location中指定，将为每一个请求扫描指定目录中的文件，并且随机选择一个代替index.html，但是不会选择以“.”开头的文件。

·参考文档

Original Documentation

Nginx Http Random Index Module

前进->GeoIP模块（GeoIP）

GeoIP模块（GeoIP）

回目录

· 摘要

这个模块基于客户端的IP地址创建一些ngx_http_geoip_module变量，并与MaxMindGeoIP文件进行匹配，该模块仅用于0.7.63和0.8.6版本之后。

使用前提

这个模块需要geo数据库和读取数据库的库文件。

```
#Get the free database of geo_city
wget http://geolite.maxmind.com/download/geoip/database/GeoLiteCity.dat.gz
#Get the free database of geo_coundty
wget http://geolite.maxmind.com/download/geoip/database/GeoLiteCountry/GeoIP.dat.gz
#Get the libgeoip. In debian you can do like this:
sudo apt-get install libgeoip-dev
#In other systems, you can download the source and compile it youself.
wget http://geolite.maxmind.com/download/geoip/api/c/GeoIP.tar.gz
```

CentOS中可以使用yum:

```
yum install geoip-devel
```

编译:

```
./configure --with-http_geoip_module
```

示例配置:

```
http {
    geoip_country  GeoIP.dat;
    geoip_city     GeoLiteCity.dat;
    ...
```

· 指令

geoip_country

语法：geoip_country path/to/db.dat;

默认值：none

使用字段：http

这个指令决定参观者IP所在国家需要使用的.dat文件，设置后该模块会创建以下变量：

·¥geoip_country_code; - 两个字母的国家代码，如："RU", "US"。

·¥geoip_country_code3; - 三个字母的国家代码，如："RUS", "USA"。

·¥geoip_country_name; - 国家的完整名称，如："Russian Federation", "United States"。

如果只需要国家名，则可以使用geoip_country数据库(1.1M)，因为geoip_city数据库大小为43M，并且它们在进行查找时是完全缓存到内存中的。

geoip_city

语法：geoip_city path/to/db.dat;

默认值：none

使用字段：http

这个指令决定参观者IP所在国家、地区和城市需要使用的.dat文件，设置后该模块会创建以下变量：

·¥geoip_country_code - 两个字母的国家代码，如："RU", "US"。

·¥geoip_country_code3 - 三个字母的国家代码，如："RUS", "USA"。

·¥geoip_country_name - 国家的完整名称，如："Russian Federation", "United States"（如果可用）。

·¥geoip_region - 地区的名称（类似于省，地区，州，行政区，联邦土地等），如："Moscow City", "DC"（如果可用）。

·¥geoip_city - 城市名称，如"Moscow", "Washington"（如果可用）。

·¥geoip_postal_code - 邮政编码（如果可用）。

·¥geoip_city_continent_code（如果可用）。

·¥geoip_latitude - 所在维度（如果可用）。

·¥geoip_longitude - 所在经度（如果可用）。

·参考文档

Original Documentation

Nginx Http GeoIP Module

前进->Real IP模块 (Real IP)

Real IP模块 (Real IP)

回目录

· 摘要

这个模块允许修改请求头中客户端的ip地址（如X-Real-IP和X-Forwarded-For）。

这对于一些工作在七层负载均衡器后面的nginx非常有用，请求来自本地IP，但是代理服务器在请求头中增加了客户端的真实IP。

要使用这个模块必须在编译时指定下列编译参数:

```
--with-http_realip_module
```

使用须知：你将创建一个信任代理的列表（见下文）并且头部中第一个不被信任的IP将作为客户端IP（参考Apache的mod_extract）。

示例配置:

```
set_real_ip_from   192.168.1.0/24;
set_real_ip_from   192.168.2.1;
real_ip_header     X-Real-IP;
```

· 指令

set_real_ip_from

语法: set_real_ip_from [the address|CIDR]

默认值: none

使用字段: http, server, location

这个指令指定信任的代理IP，它们将会以精确的替换IP转发。

real_ip_header

语法: real_ip_header [X-Real-IP|X-Forwarded-For]

默认值：real_ip_header X-Real-IP

使用字段：http, server, location

设置需要使用哪个头来确定替换的IP地址。

· 参考文档

Original Documentation

Nginx Http Real IP Module

前进->SSL模块 (SSL)

SSL模块（SSL）

回目录

· 摘要

这个模块提供HTTPS支持

支持通过以下两个限制检察客户端证书:

· 0.8.7之前的版本无法为过期证书指定列表。

· 如果你有一个证书链文件（有时称为一个中级证书），你并不需要像apache那样对每个都进行指定，你只需要将证书链中的信息追加到主证书文件中（通过cat chain.crt >> mysite.com.crt命令），之后可以不适用证书链文件，只需要让nginx指向主证书文件。

默认情况下模块并未被安装，如果要使用该模块则需要在编译时指定--with-http_ssl_module参数，安装模块依赖于OpenSSL库和一些引用文件，通常这些文件并不在同一个软件包中。

下面的例子是一个使用SSL的示例配置，为了减少CPU负载，可以将其指到一个工作进程，并且启用keepalive。

```
worker_processes 1;
http {
  server {
    listen               443;
    ssl                  on;
    ssl_certificate      /usr/local/nginx/conf/cert.pem;
    ssl_certificate_key  /usr/local/nginx/conf/cert.key;
    keepalive_timeout    70;
  }
}
```

如果使用证书链文件，只需要将其追加到你的.crt文件（上例中为cret.pem），你自己的证书需要位于文件的顶部，否则密钥无法对其进行匹配。

0.7.14版本以后通常将ssl参数使用到listen指令中:

```
server {
  listen 443 ssl;
  ssl_certificate      /usr/local/nginx/conf/cert.pem;
  ssl_certificate_key  /usr/local/nginx/conf/cert.key;
  ...
}
```

生成证书

可以通过以下步骤生成一个简单的证书:

```
$ cd /usr/local/nginx/conf
$ openssl genrsa -des3 -out server.key 1024
$ openssl req -new -key server.key -out server.csr
$ cp server.key server.key.org
$ openssl rsa -in server.key.org -out server.key
$ openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

在nginx.conf中配置这个证书:

```
server {
    server_name YOUR_DOMAINNAME_HERE;
    listen 443;
    ssl on;
    ssl_certificate /usr/local/nginx/conf/server.crt;
    ssl_certificate_key /usr/local/nginx/conf/server.key;
}
```

重启nginx。

这样就可以通过以下方式访问:

```
https://YOUR_DOMAINNAME_HERE
```

在多个server块中使用通配符证书。

在某些情况下你可能需要在一个非安全验证的域中指定一个或多个安全验证的子域，并且能够在HTTP和HTTPS子域中共享资源，要实现这种功能则需要带通配符的子域，如*.nginx.org，下面的例子中表示如何配置一个标准的www子域，一个安全验证子域，并且两个子域均能访问的共享图片域。

使用这种配置最好http块中指定一个证书文件和密钥用于多个server，这样在每个server中都会继承它的一个工作拷贝。

```
ssl_certificate      common.crt;
ssl_certificate_key  common.key;

server {
  listen           80;
  server_name      www.nginx.org;
  ...
}

server {
  listen           443 ssl;
  server_name      secure.nginx.org;
  ...
}

server {
  listen           80;
  listen           443;
  server_name      images.nginx.org;
```

```
  ...
}
```

· 指令

ssl

语法: ssl [on|off]

默认值: ssl off

使用字段: main, server

开启HTTPS。

ssl_certificate

语法: ssl_certificate file

默认值: ssl_certificate cert.pem

使用字段: main, server

为这个虚拟主机指定PEM格式的证书文件，一个文件可以包含其他的证书，同样，密钥也必须是PEM格式，0.6.7版本以后，这里的路径为相对于nginx.conf的路径，而不是编译时的prefix路径。

ssl_certificate_key

语法: ssl_certificate_key file

默认值: ssl_certificate_key cert.pem

使用字段: main, server

为这个虚拟主机指定PEM格式的密钥，0.6.7版本以后，这里的路径为相对于nginx.conf的路径，而不是编译时的prefix路径。

ssl_client_certificate

语法: ssl_client_certificate file

默认值: none

使用字段: main, server

指出PEM格式的证书认证文件，在检查客户端证书时使用。

ssl_dhparam

语法: ssl_dhparam file

默认值: none

使用字段: main, server

指出PEM格式并带有Diffie-Hellman参数的文件，用于TLS会话。

ssl_ciphers

语法: ssl_ciphers file

默认值: ssl_ciphers ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP

使用字段: main, server

指出允许的密码，密码指定为OpenSSL支持的格式，如:

```
ssl_ciphers  ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
```

使用下列命令查看完整格式列表:

```
openssl ciphers
```

ssl_crl

语法: ssl_crl file

默认值: none

使用字段: http, server

指定一个PEM格式的证书吊销列表文件，用于检查客户端证书 (0.8.7以后版本)

ssl_prefer_server_ciphers

语法: ssl_prefer_server_ciphers [on|off]

默认值: ssl_prefer_server_ciphers off

使用字段: main, server

依赖SSLv3和TLSv1协议的服务器密码将优先于客户端密码。

ssl_protocols

语法: ssl_protocols [SSLv2] [SSLv3] [TLSv1]

默认值: ssl_protocols SSLv2 SSLv3 TLSv1

使用字段: main, server

指定要使用的SSL协议。

ssl_verify_client

语法: ssl_verify_client on|off|ask

默认值: ssl_verify_client off

使用字段: main, server

启用客户端证书审核，参数“ask”在客户端主动提出时检查证书。

ssl_verify_depth

语法: ssl_verify_depth number

默认值: ssl_verify_depth 1

使用字段: main, server

设置客户证书认证链的长度。

ssl_session_cache

语法: ssl_session_cache off|none|builtin:size and/or shared:name:size

默认值: ssl_session_cache off

使用字段: main, server

设置储存SSL会话的缓存类型和大小。

缓存类型为:

· off – 强制关闭: nginx告诉客户端这个会话已经不能被再次使用。

· none – 非强制关闭：nginx告诉客户端这个会话可以被再次使用，但是nginx实际上并不使用它们，这是为某些使用ssl_session_cache的邮件客户端提供的一种变通方案，可以使用在邮件代理和HTTP服务器中。

· builtin – 内建OpenSSL缓存，仅能用在一个工作进程中，缓存大小在会话总数中指定，注意：如果要使用这个类型可能会引起内存碎片问题，具体请查看下文中参考文档。

· shared – 缓存在所有的工作进程中共享，缓存大小指定单位为字节，1MB缓存大概保存4000个会话，每个共享的缓存必须有自己的名称，相同名称的缓存可以使用在不同的虚拟主机中。

可以同时使用两个缓存类型，如：

```
ssl_session_cache  builtin:1000  shared:SSL:10m;
```

然而仅当builtin没有影响共享缓存时会使用。

ssl_session_timeout

语法：ssl_session_timeout time

默认值：ssl_session_timeout 5m

使用字段： main, server

设置客户端能够反复使用储存在缓存中的会话参数时间。

ssl_engine

语法：ssl_engine

指定使用的OpenSSL引擎，如Padlock，需要比较新版本的OpenSSL。

· 内置变量

ngx_http_ssl_module模块支持下列内建变量：

· ¥ssl_cipher – 返回建立的SSL连接中那些使用的密码字段。

· ¥ssl_client_serial – 返回建立的SSL连接中客户端证书的序列号。

· ¥ssl_client_s_dn – 返回建立的SSL连接中客户端证书的DN主题字段。

· ¥ssl_client_i_dn – 返回建立的SSL连接中客户端证书的DN发行者字段。

· ¥ssl_protocol – 返回建立的SSL连接中使用的协议。

· ¥ssl_session_id – 需要0.8.20以上版本。

· ¥ssl_client_cert

· ¥ssl_client_raw_cert

· ¥ssl_verify – 如果客户端证书审核通过，这个变量为“SUCCESS”。

· 非标准错误代码

这个模块支持一些非标准的HTTP错误代码，可以借助error_page指令用于调试：

· 495 – 检查客户端证书时发生错误。

· 496 – 客户端无法提供必须的证书。

· 497 – 传送到HTTPS的普通请求。

在请求完整的废除后调试完成，并取得一些内置变量，如：¥request_uri，¥uri，¥arg等。

· 参考文档

Original Documentation

注册一个SSL证书。

ssl_session_cache中的内存碎片与初始状态。

Nginx Http SSL Module

Stub Status模块（Stub Status）

Stub Status模块（Stub Status）

回目录

· 摘要

这个模块可以取得一些nginx的运行状态。

要使用这个模块必须在编译时指定下列编译参数：

```
--with-http_stub_status_module
```

示例配置：

```
location /nginx_status {
  # copied from http://blog.kovyrin.net/2006/04/29/monitoring-nginx-with-rrdtool/
  stub_status on;
  access_log   off;
  allow SOME.IP.ADD.RESS;
  deny all;
}
```

· 指令

stub_status

语法：stub_status on

默认值：none

使用字段：location

在这个location中启用状态监测。

这个模块可以生成一个类似mathopd状态页面的纯文本信息：

```
Active connections: 291
server accepts handled requests
  16630948 16630948 31070465
Reading: 6 Writing: 179 Waiting: 106
```

active connections -- 所有打开的连接，包括连接到后端服务器的。

server accepts handled requests -- nginx已经接受并处理16630948个连接，31070465个请求（每个连接

1.8个请求）。

reading -- 读取的请求头。

writing -- 读取的请求主体，处理的请求或者写入的应答。

waiting -- keepalive连接数，等于active - (reading + writing)。

· 参考文档

使用rrd监控nginx

rrd readme

Nginx Http Stub Status Module

前进->Substitution模块 (Substitution)

Substitution模块（Substitution）

回目录

· 摘要

这个模块可以能够在nginx的应答中搜索并替换文本。

要使用这个模块必须在编译时指定下列编译参数：

```
--with-http_sub_module option
```

示例配置：

```
location / {
  sub_filter      </head>
  '</head><script language="javascript" src="$script"></script>';
  sub_filter_once on;
}
```

· 指令

sub_filter

语法：sub_filter text substitution

默认值：none

使用字段：http, server, location

指令允许替换nginx应答体中的一些文本到另外的值，匹配不区分大小写，替换文本可以包含变量，每个location中只能指定一个替换规则。

sub_filter_once

语法：sub_filter_once on|off

默认值：sub_filter_once on

使用字段：http, server, location

设置为off将替换应答中出现的所有匹配字段，默认只会替换第一个出现的匹配值。

sub_filter_types

语法：sub_filter_types mime-type [mime-type ...]

默认值：sub_filter_types text/html

使用字段：http, server, location

设置sub_filter检查包含的MIME类型，默认只有text/html。

·参考文档

Original Documentation

Nginx Http Substitution Module

前进->WebDAV模块 (WebDAV)

WebDAV模块（WebDAV）

回目录

·摘要

这个模块增加一些HTTP和webdav扩展动作（PUT，DELETE，MKCOL，COPY和MOVE）。

要使用这个模块必须在编译时指定下列编译参数：

```
./configure --with-http_dav_module
```

示例配置：

```
location / {
  root     /data/www;
  client_body_temp_path  /data/client_temp;

  dav_methods  PUT DELETE MKCOL COPY MOVE;

  create_full_put_path   on;
  dav_access             group:rw  all:r;

  limit_except  GET {
    allow  192.168.1.0/32;
    deny   all;
  }
}
```

·指令

dav_access

语法：dav_access user:permissions [users:permissions] ...

默认值：dav_access user:rw

使用字段：http，server，location

为文件和目录指定权限，例如：

```
dav_access user:rw group:rw all:r;
```

在指定了正确的group和all后，可以不指定user：

```
dav_access group:rw all:r;
```

dav_methods

语法: dav_methods [off|put|delete|mkcol|copy|move] ...

默认值: dav_methods off

使用字段: http, server, location

启用的扩展动作，参数“off”将禁止这些扩展动作。

PUT动作的目标文件必须在存储临时文件的目录中存在（location字段的client_body_temp_path指令指定）。

当PUT创建一个文件后，将用Date头为其指定修改时间。

create_full_put_path

语法: create_full_put_path on|off

默认值: create_full_put_path off

使用字段: http, server, location

默认情况下，PUT动作只能在存在的目录中创建文件，这个指令可以允许其创建必须的目录。

· 参考文档

Original Documentation

Nginx Http Dav Module

前进->Google Perftools模块 (Google Perftools)

Google Perftools模块 (Google Perftools)

回目录

· 摘要

这个模块可以启用google性能分析工具套件，模块在版本0.6.29增加。

要使用这个模块必须在编译时指定下列编译参数:

```
./configure --with-google_perftools_module
```

示例配置:

```
google_perftools_profiles /path/to/profile;
```

Profile以/path/to/profile.<worker_pid>存储。

· 指令

google_perftools_profiles

语法: google_perftools_profiles path

默认值: none

使用字段: http, server, location

指令为从google性能分析套件中收集的文件指定一个基本文件名，程序的PID将追加到这个基本文件名后面。

· 参考文档

Nginx Google Perftools Module

前进->XSLT模块 (XSLT)

XSLT模块（XSLT）

回目录

· 摘要

这个模块是一个过滤器，它可以通过XSLT模板转换XML应答。

要使用这个模块必须在编译时指定下列编译参数（0.7.8后版本可用）：

```
./configure --with-http_xslt_module
```

示例配置：

```
location / {
  xml_entities       /site/dtd/entities.dtd;
  xslt_stylesheet    /site/xslt/one.xslt   param=value;
  xslt_stylesheet    /site/xslt/two.xslt;
}
```

· 指令

xml_entities

语法：xml_entities <path>

默认值：no

使用字段：http, server, location

指定一个描述基本XML文档标记的DTD文件（XML实体），由于技术原因无法对正在处理的XML指定实体，但是会使用这个DTD文件，在这个文件中没有必要为处理的XML指定结构，只需要声明基本的XML文档标记。如：

```
<! ENTITY of nbsp “  “>
```

xslt_stylesheet

语法：xslt_stylesheet template [parameter[[parameter...]]

默认值：no

使用字段：http, server, location

指定一个使用它自己参数的XSLT模板，参数指定为如下：

```
param=value
```

你可以为每行指定任何参数，使用“：”将每个参数分开，如果参数本身包含“：”字符请用“%3A”代替，另外，如果应用包含非数字与字符的字符串参数时需要使用单引号或双引号，并且引用它们依赖于libxslt。如：

```
param1='http%3A//www.example.com': param2=value2
```

可以在参数中使用变量，参数的实体字段可以用一个变量代替：

```
  location / {
    xslt_stylesheet /site/xslt/one.xslt
    $arg_xslt_params
    param1='$value1': param2=value2
    param3=value3;
  }
```

可以指定多个模板。

xslt_types

语法：xslt_types mime-type [mime-type...]

默认值：xslt_types text/xml

使用字段：http, server, location

可以让其处理除“text/xml”之外的mime类型，如果XSLT的输出模式为HTML，那么应答的MIME类型将修改为“text/HTML”。

·参考文档

Nginx Http Xslt Module

前进->Secure Link模块 (Secure Link)

Secure Link模块 (Secure Link)

回目录

· 摘要

这个模块为一个必需的安全性令牌检查请求网址。

要使用这个模块必须在编译时指定下列编译参数（0.7.18后版本可用）：

```
--with-http_secure_link_module
```

示例配置:

```
location /prefix/ {
    secure_link_secret   secret_word;

    if ($secure_link = "") {
        return 403;
    }
}
```

· 指令

secure_link_secret

语法: secure_link_secret secret_word

默认值: none

使用字段: location

指令为审核请求指定一个秘密字段，一个被保护连接的完整网址如下:

/prefix/hash/reference

hash通过以下函数计算:

```
md5 (reference, secret_word);
```

prefix为location块的范围，但是不能为“/”，secure_link只能用在非根路径中。

· 变量

¥secure_link

自动设置到网址的reference部分，并与prefix和hash分开，如果hash不正确，将返回一个空字符串。

· 参考文档

Nginx Http Secure Link Module

前进->Image Filter模块 (Image Filter)

Image Filter模块 (Image Filter)

回目录

· 摘要

这个模块可以在传递图片文件 (JPEG, GIF和PNG) 时对其做一些改变 。

要使用这个模块必须在编译时指定下列编译参数 (0.7.54后版本可用) :

```
--with-http_image_filter_module
```

这个模块依赖于libgd, 推荐使用近期的版本。

示例配置:

```
location /img/ {
    proxy_pass      http://backend;
    image_filter    resize  150 100;
    error_page      415   = /empty;
}

location = /empty {
    empty_gif;
}
```

· 指令

image_filter

语法: image_filter (test|size|resize width height|crop width height)

默认值: none

使用字段: location

指定需要变化的类型。如下:

· test: 核实应答确实为一个图片格式, 否则返回415错误。

· size: JSON格式的图片信息, 如:

```
{ "img" : { "width": 100, "height": 100, "type": "gif" } }
```

或许是一个错误:

{}

· resize:减少图片到指定大小。

· crop:减少图片到指定大小并对超过大小的部分进行裁剪。

image_filter_buffer

语法: image_filter_buffer size

默认值: 1M

使用字段: http, server, location

读取图片的最大值。

image_filter_jpeg_quality

语法: image_filter_jpeg_quality [0...100]

默认值: 75

使用字段: http, server, location

设置处理JPEG图片时损失的质量比率，最大推荐值为95。

image_filter_transparency

语法: image_filter_transparency on|off

默认值: on

使用字段: http, server, location

这个指令允许你在GIF和基于调色板的PNG中禁用图像透明，以提高图像质量。

无论是否设置这个参数，真色彩PNG的alpha通道总是存在。

注意: 灰度PNG并未进过测试，不过将作为真色彩进行处理。

· 参考文档

Original Documentation

Nginx Http Image Filter Module

前进->邮件模块 (Mail modules)

Nginx邮件模块（Mail modules）

回目录

Nginx邮件模块

邮件核心模块 (Mail Core)

回目录

· 邮件代理配置

nginx可以处理和代理以下的邮件协议:

· IMAP

· POP3

· SMTP

· 认证

nginx使用外部的HTTP类服务器来了解它将连接到哪个后端的IMAP/POP。

nginx在HTTP头中通过认证信息:

```
GET /auth HTTP/1.0
Host: auth.server.hostname
Auth-Method: plain
Auth-User: user
Auth-Pass: password
Auth-Protocol: imap
Auth-Login-Attempt: 1
Client-IP: 192.168.1.1
```

合适的应答为:

```
HTTP/1.0 200 OK       # 这个字段实际上是被忽略或者可能不存在。
Auth-Status: OK
Auth-Server: 192.168.1.10
Auth-Port: 110
Auth-User: newname    # 如果你连接到一个后端可以不理会这个用户名。
```

当为POP3认证使用APOP时, 你必须返回Auth-Pass:

```
HTTP/1.0 200 OK       # 这个字段实际上是被忽略或者可能不存在。
Auth-Status: OK
Auth-Server: 192.168.1.10
Auth-Port: 110
Auth-User: newname    # 如果你连接到一个后端可以不理会这个用户名。
Auth-Pass: password   # 这里必须为明文的用户名密码。
```

失败的应答为:

```
HTTP/1.0 200 OK       # 这个字段实际上是被忽略或者可能不存在。
Auth-Status: Invalid login or password
Auth-Wait: 3          # nginx将在3秒后重新读取客户端的用户名与密码。
```

· 指令

auth

0.5.15版本后重命名为pop3_auth，见下文。

imap_capabilities

语法: imap_capabilities "capability1" ["capability2" .. "capabilityN"]

默认值: "IMAP4" "IMAP4rev1" "UIDPLUS"

使用字段: main, server

在客户端发布IMAP命令CAPABILITY时，设置IMAP协议扩展列表，STARTTLS在你使用starttls指令时会自动添加。

目前标准的IMAP扩展在www.iana.org发布。

```
mail {
  imap_capabilities NAMESPACE SORT QUOTA;
}
```

默认值仍然会被设置！？

imap_client_buffer

语法: imap_client_buffer size

默认值: 4K/8K

使用字段: main, server

为IMAP命令设置读取缓冲区大小，默认值为分页大小（根据系统不同为4k或8k）。

listen

语法: listen address:port [bind]

默认值: no

使用字段: server

listen指令指定了server{...}字段中可以被访问到的ip地址及端口号，可以只指定一个ip，一个端口，或者一个可解析的服务器名。

```
listen 127.0.0.1:8000;
listen 127.0.0.1;
listen 8000;
listen *:8000;
listen localhost:8000;
```

ipv6地址格式（0.7.58）在一个方括号中指定:

```
listen [::]:8000;
listen [fe80::1];
```

指令中可以指出系统调用bind(2)。

bind -- 指出必须为这个“地址: 端口”对独立构建bind(2)，如果多个指令监听同一端口但是不同的地址和某个listen指令为这个端口（*:port）监听所有地址，那么nginx仅构建bind(2)到*:port，在这种情况下地址通过系统调用getsockname()取得。

pop3_auth

语法: pop3_auth [plain] [apop] [cram-md5]

默认值: plain

使用字段: main, server

为POP3客户端设置允许的认证动作:

- plain - USER/PASS,AUTH PLAIN,AUTH LOGIN
- apop - APOP
- cram-md5 - AUTH CRAM-MD5

pop3_capabilities

语法: pop3_capabilities "capability1" ["capability2" .. "capabilityN"]

默认值: "TOP" "USER" "UIDL"

使用字段: main, server

在客户端发布POP3命令CAPA时，设置POP3协议扩展列表，STLS在你使用starttls指令时会自动添加，SASL通过指令pop3_auth添加。

protocol

语法: protocol [pop3 | imap | smtp] ;

默认值: IMAP

使用字段: server

为这个server块设置邮件协议。

server

语法: server {...}

默认值: no

使用字段: mail

指定一个虚拟服务器配置。

没有明确的机制来分开基于域名（请求中的主机头）和基于IP的虚拟主机。

可以通过listen指令来指定必须连接到这个server块的所有地址和端口，并且在server_name指令中可以指定所有的域名。

server_name

语法: server_name name fqdn_server_host

默认值: 主机名，通过调用gethostname()取得。

使用字段: mail, server

指定虚拟主机名。如:

```
server {
  server_name   example.com  www.example.com;
}
```

第一个名称为服务器的基本名称，默认名称为机器的hostname。当然，可以使用文件通配符:

```
server {
  server_name   example.com  *.example.com;
}
```

上述例子中的两个名称可以合并为一个:

```
server {
```

```
  server_name  .example.com;
}
```

如果客户端请求中没有主机头或者没有匹配server_name的主机头，服务器基本名称将被用于一个HTTP重定向，你可以只使用“*”来强制nginx在HTTP重定向中使用Host头（注意*不能用于第一个名称，不过你可以用一个很傻逼的名称代替，如“_”）

```
server {
  server_name example.com *;
}
server {
  server_name _ *;
}
```

smtp_auth

语法: smtp_auth [login] [plain] [cram-md5] ;

默认值: login plain

使用字段: main, server

为SMTP客户端设置允许的认证动作:

· login – AUTH LOGIN

· plain – AUTH PLAIN

· cram-md5 – AUTH CRAM-MD5

smtp_capabilities

语法: smtp_capabilities “capability1” [“capability2” .. “capabilityN”]

默认值: no

使用字段: main, server

在客户端发布命令EHLO时，设置SMTP协议扩展列表，这个列表使用smtp_auth指令中启用的动作自动扩展。

so_keepalive

语法: so_keepalive on|off;

默认值: off

使用字段: main, server

为后端的IMAP/POP3设置socket SO_KEEPALIVE选项。FreeBSD中keepalive参数使用于所有的连接，并且可以通

过内核参数（sysctl net.inet.tcp.always_keepalive）关闭。

timeout

语法：timeout milliseconds;

默认值：60000

使用字段：main, server

设置到后端代理连接的超时时间。

·参考文档

Nginx Mail Core Module

前进->邮件认证模块（Mail Auth）

邮件认证模块 (Mail Auth)

回目录

· 摘要

示例配置:

```
  auth_http           localhost:9000/cgi-bin/nginxauth.cgi;
  auth_http_timeout   5;
}
```

· 指令

auth_http

语法: auth_http URL

默认值: no

使用字段: mail, server

为认证设置网址到扩展HTTP类服务器。到达的协议请查看邮件核心模块中的pop3_auth指令。

auth_http_header

语法: auth_http_header header value

默认值: no

使用字段: mail, server

在认证过程中增加一个HTTP头和它的值, 可以使用一个共享的秘密短语确保请求总是通过nginx应答。如:

```
auth_http_header X-NGX-Auth-Key "secret_string";
```

auth_http_timeout

语法: auth_http_timeout milliseconds;

默认值：60000

使用字段：mail, server

为认证进程设置超时时间。

· 参考文档

Nginx Mail Auth Module

前进->邮件代理模块 (Mail Proxy)

邮件代理模块 (Mail Proxy)

回目录

· 摘要

nginx可以代理IMAP, POP3,和SMTP协议。

· 指令

proxy

语法: proxy on | off
默认值: off
使用字段: mail, server
设置是否启用邮件代理。

proxy_buffer

语法: proxy_buffer size
默认值: 4K/8K
使用字段: mail, server
为代理连接设置缓冲区大小，默认为分页大小，根据不同的操作系统可能是4k或8k。

proxy_pass_error_message

语法: proxy_pass_error_message on | off
默认值: off
使用字段: mail, server
可以把从后端获取的错误认证信息传递到客户端，通常如果通过了nginx的认证，那么后端的错误信息无法传

递到客户端。

但是一些正确密码应答中的POP3错误，如CommuniGatePro通知用户一个关于邮箱超出容量限制（或者其它事件）将在认证中周期性的发出错误，在这种情况下有必要打开proxy_error_message。

proxy_timeout

语法: proxy_timeout time

默认值: 24h

使用字段: mail, server

为代理连接设置超时时间。

xclient

语法: xclient on | off

默认值: on

使用字段: mail, server

是否为SMTP后端连接启用XCLIENT命令，这将允许后端强制客户端连接建立在IP/HELO/LOGIN上。

如果xclient启用，那么nginx首先转发到后端:

```
EHLO server_name
```

然后:

```
XCLIENT PROTO=ESMTP HELO=client_helo ADDR=client_ip LOGIN=authentificated_user NAME=[UNAVAILABLE]
```

· 参考文档

Nginx Mail Proxy Module

前进->邮件SSL认证模块 (Mail SSL)

邮件SSL认证模块 (Mail SSL)

回目录

· 摘要

这个模块为POP3/IMAP/SMTP提供SSL/TLS支持。配置与HTTP SSL模块基本相同，但是不支持检察客户端证书。

· 指令

ssl

语法: ssl on | off
默认值: ssl off
使用字段: mail, server
为这个虚拟主机启用SSL。

ssl_certificate

语法: ssl_certificate file
默认值: cert.pem
使用字段: mail, server
为这个虚拟主机指定PEM格式的证书文件，相同的文件可以包含其它的证书，同样密钥为PEM格式。

ssl_certificate_key

语法: ssl_certificate_key file
默认值: cert.pem
使用字段: mail, server
为这个虚拟主机指定PEM格式的密钥。

ssl_ciphers

语法: ssl_ciphers file ciphers

默认值: ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP

使用字段: mail, server

指出允许的密码，密码指定为OpenSSL支持的格式。

ssl_prefer_server_ciphers

语法: ssl_prefer_server_ciphers on | off

默认值: off

使用字段: mail, server

依赖SSLv3和TLSv1协议的服务器密码将优先于客户端密码。

ssl_protocols

语法: ssl_protocols [SSLv2] [SSLv3] [TLSv1]

默认值: SSLv2 SSLv3 TLSv1

使用字段: mail, server

指定要使用的SSL协议。

ssl_session_cache

语法: ssl_session_cache [builtin[:size [shared:name:size]

默认值: builtin:20480

使用字段: mail, server

设置储存SSL会话的缓存类型和大小。

缓存类型为:

· builtin - 内建OpenSSL缓存，仅能用在一个工作进程中，缓存大小在会话总数中指定，注意: 如果要使用这个类型可能会引起内存碎片问题，具体请查看下文中参考文档。

· shared – 缓存在所有的工作进程中共享，缓存大小指定单位为字节，1MB缓存大概保存4000个会话，每个共享的缓存必须有自己的名称，相同名称的缓存可以使用在不同的虚拟主机中。

可以同时使用两个缓存类型，如：

```
ssl_session_cache  builtin:1000  shared:SSL:10m;
```

然而仅当builtin没有影响共享缓存时会使用。

ssl_session_timeout

语法：ssl_session_timeout time

默认值：5m

使用字段：mail, server

设置客户端能够反复使用储存在缓存中的会话参数时间。

starttls

语法：starttls on | off | only

默认值：off

使用字段：mail, server

· on – 允许为IMAP/SMTP使用STARTTLS和为POP3使用STLS。

· off – 禁止命令STLS和STARTTLS。

· only – 在客户端使用TLS启用STLS和STARTTLS。

· 参考文档

Nginx Mail SSL Module

前进–>第三方模块（3rd party modules）

第三方模块（3rd Party Modules）

回目录

这些模块虽然没有正式被官方支持，但是可以帮助用户完成不少的功能。使用过程中请自行承担遇到的问题。

在nginx源代码目录中使用下列命令添加第三方模块：

```
./configure --add-module=/path/to/module1/source \
            --add-module=/path/to/module2/source
```

可以根据需求使用多个--add-module。

可能根据不同的模块需要一些其他的库，请根据模块自行安装。

Evan Miller编写了nginx模块开发指南，但是某些部分有些陈旧（这个指南已经有中文版，译者姚伟斌，请点击这里下载）。

Nginx部分优化选项

回目录

哈希表（hash tables）

为了能使nginx更加快速的处理请求，nginx使用哈希表。

在nginx启动和加载配置期间，考虑到保存键（key）和值（包括失效的关键字）的哈希表存储单元不至于超出设定大小，nginx将为哈希表选择可能的最小值。

直到哈希表超过参数设置大小时才重新进行选择。大多数的哈希是指令，即可以修改它们的参数，例如，服务器名称的哈希通过指令server_names_hash_max_size和server_names_hash_bucket_size来约束，参数哈希框大小总是等于哈希表的大小，即处理器高速缓存区大小的倍数，这将加速处理器中key的搜索速度，减少内存的存取数，如果哈希框大小等于一个处理器的高速缓存区大小，那么在搜索key时内存的转换数在最坏的情况下将等于2，首先决定框的地址，其次在框中搜索key。因此，如果nginx发布需要增加哈希最大值或者哈希框大小，那么首先需要增加第一个参数的大小。

事件模型（Event Models）

nginx支持使用下列的方式处理连接，这些方式可以通过use指令指定。

· select – 标准方式，如果当前平台没有其他有效的方式，则会默认编译。你可以使用--with-select_module和--without-select_module编译参数来启用或禁止该模块。

· poll – 标准方式，如果当前平台没有其他有效的方式，则会默认编译。你可以使用--with-poll_module和--without-poll_module编译参数来启用或禁止该模块。

· kqueue – 高效方式，适用于FreeBSD 4.1+, OpenBSD 2.9+, NetBSD 2.0和MacOS X。运行于多处理器的MacOS X使用kqueue可能引起某些问题。

· epoll – 高效方式，适用于Linux 2.6+。在某些平台，例如SuSE 8.2，它们有一些关联包使2.4版本内核就能够支持epoll。

· rtsig – 可执行的实时信号，运行于Linux 2.2.19+。默认情况下系统整体无法有超过1024个POSIX实时（队

列的）信号，显然这对于高负载服务器是不够用的，因此可以通过内核参数/proc/sys/kernel/rtsig-max增加这个队列大小，然而，Linux 2.6.6-mm2以后，这个参数不再可用，并且每个处理器都是一个单独的信号队列，其大小通过RLIMIT_SIGPENDING指定，当队列溢出时，nginx将丢弃它们并且使用poll方式处理连接直到他们恢复正常。

· /dev/poll – 高效方式，适用于Solaris 7 11/99+, HP/UX 11.22+ (eventport), IRIX 6.5.15+和Tru64 UNIX 5.1A+。

· eventport – 高效方式，适用于Solaris 10,为了不引起内核错误，需要安装这个包

· 参考文档

NginxOptimizations

哈希表（俄文）

事件模型（俄文）

请访问headers_more 模块。

请访问memc 模块。

Accept Language模块（AcceptLanguageModule）

回目录

· 摘要

分析Accept-Language头并且从一个支持的语言环境列表中为其给出一个最合适的语言环境。

下载

· nginx_accept_language_module

这个模块并没有默认添加到源代码树中，具体查看下文。

可用的版本请点击这里。

示例配置

```
set_from_accept_language $lang en ja pl;
```

$lang为存储语言环境的变量，并且en ja pl为你的站点所支持的语言环境。

如果accept_language的值在你的站点上没有可用的语言环境，它将被设置为你的站点支持列表中第一个值（这个例子中是en）。

安装

点击这里下载模块。

解包并且在编译nginx时增加如下参数:

```
./configure --add-module=path/to/nginx_accept_language_module
```

为什么我要使用它

我在一个多语言站点上使用merb页面缓存，并且我需要一种从缓存中读取正确语言的方法，不久我将这个例子

放在http://gom-jabbar.org。

bugs

请发送邮件到dev@gom-jabbar.org。

· 参考文档

Nginx Accept Language Module

前进->Access Key Module模块 (Access Key Module)

HTTP Access Key模块 (AcceptLanguageModule)

回目录

· 摘要

这个模块并没有默认添加到源代码树中，具体查看下文。

这个模块拒绝一个连接除非请求URL包含一个access key，access key可以根据客户端IP地址或者其它变量创建，这样可以将动态下载限制于某些客户端。

示例配置:

```
location /download {
  accesskey             on;
  accesskey_hashmethod  md5;
  accesskey_arg         "key";
  accesskey_signature   "mypass$remote_addr";
}
```

一个客户端连接可以被定向到如: http://example.com/download/file.zip?key=09093abeac094

下载该模块

· 指令

accesskey

语法: accesskey [on|off]

默认值: accesskey off

使用字段: main, server, location

启用access-key限制。

accesskey_arg

语法: accesskey_arg "string"

默认值: accesskey "key"

使用字段: main, server, location

URL中包含的access key参数。

accesskey_hashmethod

语法: accesskey_hashmethod [md5|sha1]

默认值: accesskey_hashmethod md5

使用字段: main, server, location

是否在创建key的时候使用MD5或SHA-1。

accesskey_signature

语法: accesskey_signature "string"

默认值: accesskey_signature "¥remote_addr"

使用字段: main, server, location

创建access key的字符串, 包含独一无二的客户端地址的¥remote_addr变量创建的key。最好包含一个秘密短语确保key的完整性, 如 "myPassWord¥remote_addr"。

· 安装

下载: File:Nginx-accesskey-2.0.3.tar.gz

解包, 编辑config文件替换"¥HTTP_ACCESSKEY_MODULE"为"ngx_http_accesskey_module", 然后编译nginx:

```
./configure --add-module=path/to/nginx-accesskey
```

· 其它

Nginx Http Access Key Module

OWOX project

前进->Auth PAM 模块 (Auth PAM Module)

Circle GIF 模块 (Circle GIF Module)

回目录

· 摘要

这个模块可以根据URL指定的颜色与大小生成一个圆形图片，这个图片可以很快速的生成，速度要比读取磁盘快很多。这个模块可以帮助网站设计师修改“round corners”的颜色，并且不启用Photoshop。

示例配置:

```
location /circles {
    circle_gif;
}
```

下载该模块

· 指令

circle_gif

语法: circle_gif

默认值: n/a

使用字段: location

circle_gif_min_radius

语法: circle_min_radius

默认值: 10

使用字段: location

生成图片的最小半径，像素为单位。

circle_gif_max_radius

语法：circle_max_radius

默认值：20

使用字段：location

生成图片的最大半径，像素为单位。

circle_gif_step_radius

语法：circle_step_radius

默认值：2

使用字段：location

· 用法

在指定location的URL后面添加如下可以生成图片：

```
<background color>/<foreground color>/<radius>.gif
```

radius为图片半径（像素为单位），colors为24位十六进制颜色（如 "ffffff"为白，"000000"为黑），例如，以下示例将生成一个半径为20像素白底的黑色圆形：

```
/circles/ffffff/000000/20.gif
```

· 安装

下载：File:Nginx_circle_gif-0.1.3.tar.gz

解包，然后编译nginx：

```
./configure --add-module=path/to/circle_gif/directory
```

· Bugs

提交报告给Evan Miller

· 参考文档

Nginx Http Circle Gif Module

前进->Echo 模块（Echo Module）

Events模块（Events Module）

回目录

·摘要

配置start/stop事件

示例配置:

```
on_start /opt/f.sh;
on_stop kill `cat /tmp/f.pid`;
on_stop rm -f /tmp/f.sock;
```

作者: Anton Dutov

下载该模块

源代码

文档: 英文, 俄文

·指令

on_start

语法: on_start <system command>

默认值: none

使用字段: server

示例: on_start echo “started” > /tmp/test && date >> /tmp/test;

在服务启动时执行一些系统命令。

on_stop

语法: on_stop <system command>

默认值: none

使用字段: server

示例: on_stop echo “stopped” > /tmp/test && date >> /tmp/test;

在服务停止时执行一些系统命令。

· 示例配置

Mercurial (代理)

```
server {
    listen                  80;
    server_name             hg.dutov.org;
    server_name_in_redirect  off;
    access_log              /var/log/nginx/org.dutov.hg.access.log  main;

    on_start hg serve
        --style paper
        --webdir-conf /var/www/org.dutov.hg/org.dutov.hg.hgweb
        --pid-file    /var/run/org.dutov.hg.pid
        -A /dev/null
        -E /dev/null
        -d  -a 127.0.0.1 -p 8080;

    on_stop kill `cat -- /var/run/org.dutov.hg.pid`;
    on_stop rm -f -- /var/run/org.dutov.hg.pid;

    location / {
        if ($request_method = POST ) {
            return 405;
        }
        proxy_pass          http://127.0.0.1:8080/;
        proxy_redirect      off;
        proxy_set_header    Host             $host;
        proxy_set_header    X-Real-IP        $remote_addr;
        proxy_set_header    X-Forwarded-For  $remote_addr;

        client_max_body_size       10m;
        client_body_buffer_size    128k;
        proxy_connect_timeout      90;
        proxy_send_timeout         90;
        proxy_read_timeout         90;
        proxy_buffer_size          4k;
        proxy_buffers              4 32k;
        proxy_busy_buffers_size    64k;
        proxy_temp_file_write_size 10m;
    }
}
```

Redmine

```
server {
    listen                  80;
    server_name             dev.dutov.org;
    server_name_in_redirect  off;
    access_log              /var/log/nginx/org.dutov.dev.access.log  main;
    root                    /var/www/org.dutov.dev/public;

    on_start export RAILS_ENV="production" && /usr/bin/spawn-fcgi
        -u www-data
        -g www-data
        -P /var/run/org.dutov.dev.pid
```

```
        -s /var/run/org.dutov.dev.sock
        -f /var/www/org.dutov.dev/public/dispatch.fcgi;

    on_stop kill `cat -- /var/run/org.dutov.dev.pid`;
    on_stop rm -f -- /var/run/org.dutov.dev.pid;

    location ~ ^/(javascripts|stylesheets|themes|images|help)/ {
        expires    30d;
        access_log off;
    }

    location / {
        fastcgi_pass    unix:/var/run/org.dutov.dev.sock;
        include         /etc/nginx/fastcgi_params;
    }
}
```

· 参考文档

Nginx Events Module

前进->Expressz模块（Expressz Module）

Expressz模块（Expressz Module）

回目录

项目未完成。。。。

作者：Jason Giedymin at AcronymLabs

·参考文档

Nginx Expressz Module

前进->EY Balancer模块（EY Balancer Module）

EY Balancer模块 (EY Balancer Module)

回目录

· 摘要

为nginx增加一个请求队列以便通过上游服务器允许的并发请求。

作者: Ry Dahl

点击这里下载

在一个upstream指令中使用“max_connections N;”意为每个上游服务器每次给予N个请求，例如，如果你有两台上游服务器并且使用"max_connections 1;" 指令，那么将每次产生两个请求依次在每个上游服务器上。

一些分析报告: http://four.livejournal.com/955976.html

· 用法

```
upstream mongrels {
  server 127.0.0.1:8001;
  server 127.0.0.1:8002;
  max_connections 1;
}
```

· 安装

这个模块依赖nginx安装包，它同样包含一个Makefile以便能够更容易将其安装到nginx并且执行测试，如果使用Makefile则必须修改其第一行指向nginx源代码树，如下例:

```
tar -zxf nginx-0.6.35.tar.gz
tar -xzf ngx_max_connections-0.0.5.tar.gz
cd nginx-0.6.35
patch -p0 < ../ngx_max_connections-0.0.5/patches/nginx-0.6.35.patch
cd ../ngx_max_connections-0.0.5
vim Makefile #### 编辑文件第一行
make configure
make
make test    #### 需要ruby, rubygems, rack,和httperf
```

· 参考文档

Nginx Expressz Module

前进->Fancy Indexes模块 (Fancy Indexes Module)

Fancy Indexes模块 (Fancy Indexes Module)

回目录

· 摘要

类似于autoindex模块，但是可以提供更友好的目录清单主页。

作者: Adrian Perez de Castro

点击这里下载

对这个模块有兴趣的请参考维基或者README。

· 参考文档

Fancy Indexes Module

前进->GeoIP模块 (GeoIP Module)

GeoIP模块 (GeoIP Module)

回目录

· 摘要

根据MaxMindGeoIP接口查找国家代码。

作者: SPIL GAMES

点击这里下载

该模块只适用于nginx0.7以上版本。

改模块基于标准模块中的GEO模块，区别在于它并不约束于某一块配置，而使用MaxMind二进制API。它在二进制GeoLite Country和GeoIP Country数据库中提供更快速的查询。

使用二进制数据库和API的好处之一是能够通过geoipupdate功能获得自动化的更新，这正是SPIL GAMES编写该模块的原因之一。

示例配置:

```
geoip_country_file /absolute/path/to/GeoIP.dat;
location /geoip/ {
  rewrite .* /?country=$geoip_country_code;
}
```

指令

geoip_country_file

语法: geoip_country_file path

默认值: n/a

使用字段: http

启用解析下文中的¥geoip_country_* 变量。

· 变量

支持下列变量:

¥geoip_country_code, 从GeoIP country API解析的2位国家代码。

¥geoip_country_code3, 从GeoIP country API解析的3位国家代码。

¥geoip_country_name, 从GeoIP country API解析的完整国家名称。

· 安装

下载: File:Nginx-geoip-0.2.tar.gz

解包, 并且在编译nginx时加入下列参数:

```
. --add-module=/path/to/nginx-geoip-0.2
```

安装前请确保你的系统中已经安装了 GeoIP C API库。

· Bugs

请提交报告到matthijs@spilgames.com。

· 参考文档

GeoIP Module

前进->Headers More模块 (Headers More Module)