ngx\_http\_limit\_conn\_module模块 (限制用户并发连接数及请求数模块)

ngx\_http\_limet\_req\_module模块 （限制Nginx request processing rate 根据定义的key）

ngx\_http\_core\_module模块 （包含一些核心的http参数配置，对应Nginx的配置区块部分）

ngx\_http\_access\_module模块 （访问控制模块，用来控制网站用户对Nginx的访问）

ngx\_http\_fastcgi \_module模块 （fastcgi模块，和动态应用相关的模块，例如PHP）

ngx\_http\_proxy \_module模块 （proxy代理模块）

ngx\_http\_upstream \_module模块 （负载均衡模块，可以实现网站的负载均衡功能及节点的健康检查）

ngx\_http\_auth\_basic\_module模块 (Web认证模块，设置Web用户通过账号密码访问Nginx)

ngx\_http\_stub\_status\_module模块

ngx\_http\_log\_module模块 (访问日志模块，以指定的格式记录Nginx客户访问日志等信息)

ngx\_http\_gzip\_module模块

ngx\_http\_ssl\_module模块 (ssl模块，用于加密的http连接，如 https)

ngx\_http\_rewrite\_module模块 （URL地址重写模块）

ngx\_http\_referer\_module模块

ngx\_http\_headers\_module模块

**ngx\_http\_limet\_req\_module模块**

### ****一. 前言****

在《[nginx限制连接数ngx\_http\_limit\_conn\_module模块](http://www.ttlsa.com/html/3180.html)》我们说到了ngx\_http\_limit\_conn\_module 模块，来限制连接数。那么请求数的限制该怎么做呢？这就需要通过ngx\_http\_limit\_req\_module 模块来实现，该模块可以通过定义的 键值来限制请求处理的频率。特别的，可以限制来自单个IP地址的请求处理频率。 限制的方法如同漏斗，每秒固定处理请求数，推迟过多请求。

### ****二.**** ngx\_http\_limit\_req\_module模块指令

limit\_req\_zone  
语法: limit\_req\_zone $variable zone=name:size rate=rate;  
默认值: none  
配置段: http  
设置一块共享内存限制域用来保存键值的状态参数。 特别是保存了当前超出请求的数量。 键的值就是指定的变量（空值不会被计算）。如

|  |  |
| --- | --- |
| 1 | limit\_req\_zone $binary\_remote\_addr zone=one:10m rate=1r/s; |

说明：区域名称为one，大小为10m，平均处理的请求频率不能超过每秒一次。  
键值是客户端IP。  
使用$binary\_remote\_addr变量， 可以将每条状态记录的大小减少到64个字节，这样1M的内存可以保存大约1万6千个64字节的记录。  
如果限制域的存储空间耗尽了，对于后续所有请求，服务器都会返回 503 (Service Temporarily Unavailable)错误。  
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”。

limit\_req\_log\_level  
语法: limit\_req\_log\_level info | notice | warn | error;  
默认值: limit\_req\_log\_level error;  
配置段: http, server, location  
设置你所希望的日志级别，当服务器因为频率过高拒绝或者延迟处理请求时可以记下相应级别的日志。 延迟记录的日志级别比拒绝的低一个级别；比如， 如果设置“limit\_req\_log\_level notice”， 延迟的日志就是info级别。

limit\_req\_status  
语法: limit\_req\_status code;  
默认值: limit\_req\_status 503;  
配置段: http, server, location  
该指令在1.3.15版本引入。设置拒绝请求的响应状态码。

limit\_req  
语法: limit\_req zone=name [burst=number] [nodelay];  
默认值: —  
配置段: http, server, location  
设置对应的共享内存限制域和允许被处理的最大请求数阈值。 如果请求的频率超过了限制域配置的值，请求处理会被延迟，所以所有的请求都是以定义的频率被处理的。 超过频率限制的请求会被延迟，直到被延迟的请求数超过了定义的阈值，这时，这个请求会被终止，并返回503 (Service Temporarily Unavailable) 错误。这个阈值的默认值为0。如：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | limit\_req\_zone $binary\_remote\_addr zone=ttlsa\_com:10m rate=1r/s;  server {      location /www.ttlsa.com/ {          limit\_req zone=ttlsa\_com burst=5;      }  } |

限制平均每秒不超过一个请求，同时允许超过频率限制的请求数不多于5个。  
如果不希望超过的请求被延迟，可以用nodelay参数,如：

|  |  |
| --- | --- |
| 1 | limit\_req zone=ttlsa\_com burst=5 nodelay; |

### ****三. 完整实例配置****

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | http {  limit\_req\_zone $binary\_remote\_addr zone=ttlsa\_com:10m rate=1r/s;    server {  location  ^~ /download/ {  limit\_req zone=ttlsa\_com burst=5;  alias /data/www.ttlsa.com/download/;          }  }  } |

可能要对某些IP不做限制，需要使用到白名单。名单设置参见后续的文档，我会整理一份以供读者参考。请专注。

如需转载请注明出处：<http://www.ttlsa.com/html/3185.html>

<http://www.ttlsa.com/nginx/nginx-limiting-the-number-of-requests-ngx_http_limit_req_module-module/>

**ngx\_http\_limit\_conn\_module**

### ****一. 前言****

我们经常会遇到这种情况，服务器流量异常，负载过大等等。对于大流量恶意的攻击访问，会带来带宽的浪费，服务器压力，影响业务，往往考虑对同一个ip的连接数，并发数进行限制。下面说说ngx\_http\_limit\_conn\_module 模块来实现该需求。该模块可以根据定义的键来限制每个键值的连接数，如同一个IP来源的连接数。并不是所有的连接都会被该模块计数，只有那些正在被处理的请求（这些请求的头信息已被完全读入）所在的连接才会被计数。

### 二. ngx\_http\_limit\_conn\_module指令解释

limit\_conn\_zone  
语法: limit\_conn\_zone $variable zone=name:size;  
默认值: none  
配置段: http  
该指令描述会话状态存储区域。键的状态中保存了当前连接数，键的值可以是特定变量的任何非空值（空值将不会被考虑）。$variable定义键，zone=name定义区域名称，后面的limit\_conn指令会用到的。size定义各个键共享内存空间大小。如：

limit\_conn\_zone $binary\_remote\_addr zone=addr:10m;

注释：客户端的IP地址作为键。注意，这里使用的是$binary\_remote\_addr变量，而不是$remote\_addr变量。  
$remote\_addr变量的长度为7字节到15字节，而存储状态在32位平台中占用32字节或64字节，在64位平台中占用64字节。  
$binary\_remote\_addr变量的长度是固定的4字节，存储状态在32位平台中占用32字节或64字节，在64位平台中占用64字节。  
1M共享空间可以保存3.2万个32位的状态，1.6万个64位的状态。  
如果共享内存空间被耗尽，服务器将会对后续所有的请求返回 503 (Service Temporarily Unavailable) 错误。  
limit\_zone 指令和limit\_conn\_zone指令同等意思，已经被弃用，就不再做说明了。

limit\_conn\_log\_level  
语法：limit\_conn\_log\_level info | notice | warn | error  
默认值：error  
配置段：http, server, location  
当达到最大限制连接数后，记录日志的等级。

limit\_conn  
语法：limit\_conn zone\_name number  
默认值：none  
配置段：http, server, location  
指定每个给定键值的最大同时连接数，当超过这个数字时被返回503 (Service Temporarily Unavailable)错误。如：

imit\_conn\_zone $binary\_remote\_addr zone=addr:10m;

server {

location /www.ttlsa.com/ {

limit\_conn addr 1;

}

}

同一IP同一时间只允许有一个连接。  
当多个 limit\_conn 指令被配置时，所有的连接数限制都会生效。比如，下面配置不仅会限制单一IP来源的连接数，同时也会限制单一虚拟服务器的总连接数：

lmit\_conn\_zone $binary\_remote\_addr zone=perip:10m;

limit\_conn\_zone $server\_name zone=perserver:10m;

server {

limit\_conn perip 10;

limit\_conn perserver 100;

}

[warning]limit\_conn指令可以从上级继承下来。[/warning]

limit\_conn\_status  
语法: limit\_conn\_status code;  
默认值: limit\_conn\_status 503;  
配置段: http, server, location  
该指定在1.3.15版本引入的。指定当超过限制时，返回的状态码。默认是503。

limit\_rate  
语法：limit\_rate rate  
默认值：0  
配置段：http, server, location, if in location  
对每个连接的速率限制。参数rate的单位是字节/秒，设置为0将关闭限速。 按连接限速而不是按IP限制，因此如果某个客户端同时开启了两个连接，那么客户端的整体速率是这条指令设置值的2倍。

### ****三. 完整实例配置****

http {

limit\_conn\_zone $binary\_remote\_addr zone=limit:10m;

limit\_conn\_log\_level info;

server {

location ^~ /download/ {

limit\_conn limit 4;

limit\_rate 200k;

alias /data/www.ttlsa.com/download/;

}

}

}

### ****四. 使用注意事项****

事务都具有两面性的。ngx\_http\_limit\_conn\_module 模块虽说可以解决当前面临的并发问题，但是会引入另外一些问题的。如前端如果有做LVS或反代，而我们后端启用了该模块功能，那不是非常多503错误了？这样的话，可以在前端启用该模块，要么就是设置白名单，白名单设置参见后续的文档，我会整理一份以供读者参考。

可以与ngx\_http\_limit\_req\_module 模块结合起来使用，以达到最好效果。《[nginx限制请求数ngx\_http\_limit\_req\_module模块](http://www.ttlsa.com/html/3185.html)》。

如需转载请注明出处：http://www.ttlsa.com/html/3180.html

<http://www.ttlsa.com/nginx/nginx-limited-connection-number-ngx_http_limit_conn_module-module/>

### ngx\_http\_core\_module

**模块的功能**

模拟反代基于tcp或udp的服务连接，即工作于传输层的反代或调度器。

server { ... }

配置一个虚拟主机

server {

listen address[:PORT]|PORT;

server\_name SERVER\_NAME;

root /PATH/TO/DOCUMENT\_ROOT;

}

1、listen PORT|address[:port]|unix:/PATH/TO/SOCKET\_FILElisten address[:port] [default\_server] [ssl] [http2 | spdy] [backlog=number] [rcvbuf=size] [sndbuf=size]

default\_server #设定为默认虚拟主机

ssl #限制仅能够通过ssl连接提供服务

backlog=number #超过并发连接数后，新请求进入后援队列的长度

rcvbuf=size #接收缓冲区大小

sndbuf=size #发送缓冲区大小

注意：

(1) 基于port;

listen PORT; 指令监听在不同的端口

(2) 基于ip的虚拟主机

listen IP:PORT; IP 地址不同

(3) 基于host

nameserver\_name fqdn; 指令指向不同的主机名

(4) default\_server

不加这个的话默认页面是/etc/nginx/conf.d 文件下排序的第一个配置文件为主

2、server\_name name ...;

虚拟主机的主机名称后可跟多个由空白字符分隔的字符串

支持\*通配任意长度的任意字符

server\_name \*.a.com www.a.\*

支持~起始的字符做正则表达式模式匹配，性能原因慎用

server\_name ~^www\d+\.a\.com$

\d 表示[0-9]

匹配优先级机制从高到低：

(1) 首先是字符串精确匹配 如：www.a.com

(2) 左侧\*通配符 如：\*.a.com

(3) 右侧\*通配符 如：www.a.\*

(4) 正则表达式 如：~^.\*\.a\.com$

(5) default\_server

3、tcp\_nodelay on | off;

在keepalived模式下的连接是否启用TCP\_NODELAY选项

当为off时，延迟发送，合并多个请求后再发送

默认On时，不延迟发送

可用位置：http, server, location

4、sendfile on | off;

是否启用sendfile功能，在内核中封装报文直接发送默认Off

可用位置：http, server, location, if in location

5、server\_tokens on | off | build | string

是否在响应报文的Server首部显示nginx版本

可用位置：http, server, location

**示例1：虚拟主机**

server{

listen 80;

server\_name www.a.com;

root /data/web1/;

} #1、以server\_name 定义虚拟主机

server{

listen 192.168.109.143:98

root /data/web2/;

} #2、以ip加端口定义虚拟主机

**定义路径相关的配置**

6、root

设置web资源的路径映射；用于指明请求的URL所对应的文档的目录路径

可用位置：http, server, location, if in location

server {

...

root /data/www/vhost1;

}

示例：

http://www.a.com/images/logo.jpg

--> /data/www/vhosts/images/logo.jpg

7、location [ = | ~ | ~\* | ^~ ] uri{ ... }

location @name { ... }

在一个server中location配置段可存在多个，用于实现从uri到文件系统的路径映射；ngnix会根据用户请求的URI来检查定义的所有location，并找出一个最佳匹配，而后应用其配置

可用位置：server, location

示例：

server {...

server\_name www.a.com;

location /images/ {

root /data/imgs/;

}

}

http://www.a.com/images/logo.jpg

--> /data/imgs/images/logo.jpg #访问这个图片真实路径

=：对URI做精确匹配；

location = / {

...

}

http://www.a.com/ 匹配

http://www.a.com/index.html 不匹配

^~：对URI的最左边部分做匹配检查，不区分字符大小写

~：对URI做正则表达式模式匹配，区分字符大小写

~\*：对URI做正则表达式模式匹配，不区分字符大小写

不带符号：匹配起始于此uri的所有的uri

匹配优先级从高到低：=, ^~, ～/～\*, 不带符号

location = / {

[ configuration A ]

}

location / {

[ configuration B ]

}

location /documents/ {

[ configuration C ]

}

location ^~ /images/ {

[ configuration D ]

}

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

[ configuration E ]

}

示例：

root /vhosts/www/htdocs/

http://www.a.com/index.html

--> /vhosts/www/htdocs/index.html

server {

root /vhosts/www/htdocs/

location /admin/ {

root /webapps/app1/data/

}

}

http://www.a.com/admin/index.html

--> /webapps/app1/data/admin/index.html

8、alias path;

路径别名，文档映射的另一种机制

仅能用于location上下文

示例：

访问: http://www.a.com/bbs/index.php

location /bbs/ {

alias /web/forum/;

} --> 真实路径 /web/forum/index.html

location /bbs/ {

root /web/forum/;

} --> 真实路径 /web/forum/bbs/index.html

注意：location中使用root指令和alias指令的意义不同

(a) root， 给定的路径对应于location中的/uri/左侧的/

(b) alias，给定的路径对应于location中的/uri/右侧的/

9、error\_page code ... [=[response]] uri;

模块：ngx\_http\_core\_module

定义错误页，以指定的响应状态码进行响应

可用位置：http, server, location, if in location

error\_page 404 /404.html

error\_page 404 =200 /404.html

示例：

server{

listen 80;

server\_name www.b.com;

error\_page 404 /404.html; #页面是404错误换成默认页面

location /404.html {

root /data/error\_html/;#默认页面是/data/error\_html/404.html页面

}

root /data/web2;

}

server{

listen 80;

server\_name www.b.com;

error\_page 404 =200 /404.html; #=200意思是请求的页面状态码不会出现404 把404 变成200

location /404.html {

root /data/error\_html/;

}

root /data/web2;

}

10、try\_files file ... uri;

try\_files file ... =code;

可用位置：server, location

按顺序检查文件是否存在，返回第一个找到的文件或文件夹（结尾加斜线表示为文件夹），如果所有的文件或文件夹都找不到，会进行一个内部重定向到最后一个参数。只有最后一个参数可以引起一个内部重定向，之前的参数只设置内部URI的指向。最后一个参数是回退URI且必须存在，否则会出现内部500错误

location /images/ {

try\_files $uri /images/default.gif; #找不到图片 默认图片代替

}

location / {

try\_files $uri $uri/index.html $uri.html =404;

}

**定义客户端请求的相关配置**

11、keepalive\_timeout timeout [header\_timeout];

设定保持连接超时时长，0表示禁止长连接，默认为75s

可用位置：http, server, location

12、keepalive\_requests number;

在一次长连接上所允许请求的资源的最大数量 默认为100

可用位置：http, server, location

13、keepalive\_disable none | browser ...

对哪种浏览器禁用长连接

可用位置：http, server, location

14、send\_timeout time;

向客户端发送响应报文的超时时长，此处是指两次写操作之间的间隔时长，而非整个响应过程的传输时长.如果客户端在这段时间内没有收到任何东西，连接将被关闭

可用位置：http, server, location

15、client\_body\_buffer\_size size;

用于接收每个客户端请求报文的body部分的缓冲区大小；默认为16k；超出此大小时，其将被暂存到磁盘上的由client\_body\_temp\_path指令所定义的位置

可用位置：http, server, location

16、client\_body\_temp\_path path [level1 [level2 [level3]]];

设定用于存储客户端请求报文的body部分的临时存储路径及子目录结构和数量

可用位置：http, server, location

目录名为16进制的数字；

client\_body\_temp\_path /var/tmp/client\_body 1 2 2

1 1级目录占1位16进制，即2^4=16个目录0-f

2 2级目录占2位16进制，即2^8=256个目录00-ff

2 3级目录占2位16进制，即2^8=256个目录00-ff

**对客户端进行限制的相关配置**

17、limit\_rate rate;

限制响应给客户端的传输速率，单位是bytes/second 默认值0表示无限制

可用位置：http, server, location, if in location

18、limit\_except method ... { ... }，仅用于location

限制客户端使用除了指定的请求方法之外的其它方法

method:GET, HEAD, POST, PUT, DELETE MKCOL, COPY, MOVE, OPTIONS, PROPFIND,PROPPATCH, LOCK, UNLOCK, PATCH

limit\_except GET {

allow 192.168.1.0/24;

deny all;

} 除了GET和HEAD之外其它方法仅允许192.168.1.0/24网段主机使用

**文件操作优化的配置**

19、aio on | off | threads[=pool];

是否启用aio功能 该指令出现在0.8.11版本中。

可用位置：http, server, location

20、directio size | off;

是否同步（直接）写磁盘，而非写缓存，在Linux主机启用O\_DIRECT标记，则文件大于等于给定大小时使用，例如:directio 4m 大于4M直接写磁盘

可用位置：http, server, location

21、open\_file\_cache off;

open\_file\_cachemax=N [inactive=time];

可用位置：http, server, location

nginx可以缓存以下三种信息：

(1) 文件元数据：文件的描述符、文件大小和最近一次的修改时间

(2) 打开的目录结构

(3) 没有找到的或者没有权限访问的文件的相关信息

max=N：可缓存的缓存项上限；达到上限后会使用LRU算法实现管理

LRU算法参考：http://blog.csdn.net/xiao\_mrs\_li/article/details/69257694

inactive=time：缓存项的非活动时长，在此处指定的时长内未被命中的或命中的次数少于open\_file\_cache\_min\_uses指令所指定的次数的缓存项即为非活动项，将被删除

22、open\_file\_cache\_errors on | off;

是否缓存查找时发生错误的文件一类的信息默认值为off

可用位置：http, server, location

23、open\_file\_cache\_min\_uses number;

open\_file\_cache指令的inactive参数指定的时长内，至少被命中此处指定的次数方可被归类为活动项 默认值为1

可用位置：http, server, location

24、open\_file\_cache\_validtime;

缓存项有效性的检查频率默认值为60s

可用位置：http, server, location

原文链接：<https://blog.csdn.net/qq_37187976/article/details/79495843>

**ngx\_http\_access\_module模块**

这个模块只有2个指令

**格式：**

allow address | CIDR | unix: | all; #允许访问

deny address | CIDR | unix: | all; #拒绝访问

**示例**

location / {

root html;

index index.html index.htm;

allow 192.168.253.129; ##允许192.168.253.129访问

deny 192.168.253.139; ##拒绝192.168.253.139访问

}

## ngx\_http\_fastcgi\_module

## 前言

在Apache中php页面内容通常是通过fastcgi代理传递到php-fpm服务器上进行处理的。同理在nginx服务中也也支持类似的做法。nginx服务支持FastCGI模式，能够快速高效地处理php动态请求的内容。而nginx的FastCGI模式对应的模块即为ngx\_http\_fastcgi\_module。那么下面我们就来一起看下该模块的常用指令以及简单的示列。

**1、fastcgi\_pass address;**  
配置段： location, if in location  
此指令用于指定一个fastcgi服务器的地址，可以服务器的主机名或IP+端口的方式指定。  
示例：

location ~\* \.php$ {

....

fastcgi\_pass 10.10.10.13:9000;

...

}

**2、fastcgi\_index name;**  
配置段：http, server, location  
设置fastcgi的默认主页资源，如果URI以斜线结尾，文件名将追加到URI后面，这个值将存储在变量$fastcgi\_script\_name中。例如：

fastcgi\_index index.php;

fastcgi\_param SCRIPT\_FILENAME /home/www/scripts/php$fastcgi\_script\_name;

请求uri/page.php的参数SCRIPT\_FILENAME将被设置为/home/www/scripts/php/page.php，

但是"/"为"/home/www/scripts/php/index.php"。

**3、fastcgi\_param parameter value [if\_not\_empty];**  
配置段：http, server, location  
此指令用于指定要传递给FastCGI服务器的参数，参数可以为文本、变量和两者之间的组合。  
**示例1：**

#将匹配的php内容送到fastcgi服务器的/usr/share/nginx/html目录下进行处理，即指定fastcgi服务器上用于存放PHP内容的目录

location ~\* \.php$ {

root /usr/share/nginx/html;

fastcgi\_pass 127.0.0.1:9000;

fastcgi\_index index.php;

fastcgi\_param SCRIPT\_FILENAME /usr/share/nginx/html$fastcgi\_script\_name;

include fastcgi\_params; #调用nginx的变量定义；

}

**示例2：**

#将状态页面status和检测页面ping送到fastcgi进行处理

location ~\* ^/(status|ping)$ {

include fastcgi\_params;

fastcgi\_pass 127.0.0.1:9000;

fastcgi\_param SCRIPT\_FILENAME $fastcgi\_script\_name;

}

**4、fastcgi\_cache\_path path [levels=levels] [use\_temp\_path=on|off] keys\_zone=name:size [inactive=time] [max\_size=size] [manager\_files=number] [manager\_sleep=time] [manager\_threshold=time] [loader\_files=number] [loader\_sleep=time] [loader\_threshold=time] [purger=on|off]** [purger\_files=number] [purger\_sleep=time] [purger\_threshold=time];  
配置段：http  
此指令用于定义fastcgi代理的缓存路径，缓存位置为nginx服务器上的文件路径。  
levels=levels：缓存目录的层级数量，以及每一级的目录数量；格式为：levels=ONE:TWO:THREE，如：leves=1:2:2。  
keys\_zone=name:size：指定映射的内存空间的名称及大小  
inactive=time：指定缓存文件如果在指定的时间内都没有被访问，则文件会被cache manager进程删除掉。  
max\_size=size：指定此缓存空间的大小上限。  
**示例：**  
fastcgi\_cache\_path /data/nginx/cache levels=1:2 keys\_zone=one:10m;

**- 5、fastcgi\_cache zone | off;**  
配置段： http, server, location  
调用使用fastcgi\_cache\_path指定设置的缓存空间来缓存数据。  
**示例：**

fastcgi\_cache one;

**- 6、fastcgi\_cache\_key string;**  
配置段： http, server, location  
定义缓存的key字符串。

**- 7、fastcgi\_cache\_methods GET | HEAD | POST ...;**  
配置段：http, server, location  
设置哪些请求方法能够使用fastcgi缓存。

**- 8、fastcgi\_cache\_min\_uses number;**  
配置段：http, server, location  
缓存空间中的缓存项在inactive定义的非活动时间内至少要被访问到此处所指定的次数方可被认作活动项；

**- 9、fastcgi\_cache\_valid [code ...] time;**  
配置段： http, server, location  
根据不同的页面响应码设置不同的缓存时间。  
示例：

location ~\* \.php$ {

...

fastcgi\_cache fcgi;

fastcgi\_cache\_key $request\_uri;

fastcgi\_cache\_valid 200 302 10m; #状态码为200和302的页面内容缓存10分钟

fastcgi\_cache\_valid 301 1h;

fastcgi\_cache\_valid any 1m;

...

}

**- 10、fastcgi\_keep\_conn on | off;**  
配置段：http, server, location  
用于开启fastcgi长连接机制。

###### 综合示例

server {

listen 80;

server\_name www.ilinux.io;

index index.php index.html;

location / {

root /data/nginx/html;

proxy\_pass http://10.10.10.12:80;

}

location ~\* \.php$ {

fastcgi\_pass 10.10.10.11:9000; #指定fastcgi服务器

fastcgi\_index index.php; #指定默认的fastcgi主页

include fastcgi\_params; #调用nginx的变量定义

fastcgi\_param SCRIPT\_FILENAME /data/apps/$fastcgi\_script\_name; #指定fastcgi服务器上的php目录

fastcgi\_cache fcache; #调用fache缓存空间

fastcgi\_cache\_key $request\_uri; #设置缓存的key为$request\_uri

fastcgi\_cache\_valid 200 302 10m; #状态码为200和302的页面缓存10分钟

fastcgi\_cache\_valid 301 1h; #状态码为301的页面缓存1小时

fastcgi\_cache\_valid any 1m; #剩下的都缓存1分钟

fastcgi\_keep\_conn on; #开启长连接

}

location ~\* ^/(status|ping)$ {

fastcgi\_pass 10.10.10.11:9000;

include fastcgi\_params;

fastcgi\_param SCRIPT\_FILENAME /data/apps/$fastcgi\_script\_name;

}

}

<https://www.jianshu.com/p/097f82c9e4a4>

# [Nginx-http\_proxy\_module模块](https://www.cnblogs.com/yanjieli/p/10682685.html)

# Nginx 反向代理之 http\_proxy\_module 模块

**proxy\_pass指定属于 ngx\_http\_proxy\_module 模块，此模块可以将请求转发到另一台服务器，在实际的反向代理工作中，会通过 location 功能匹配指定的 URI，然后把接收到的符合匹配 URI 的请求通过 proxy\_pass 抛给定义好的 upstream 节点池。**

该指令官网地址见：<http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_pass>

**官网 proxy\_pass 使用 案例：**

1）将匹配 URI 为 name 的请求抛给 http://127.0.0.1/remote/

location /name/ {

proxy\_pass http://127.0.0.1/remote/;

}

2）将匹配 URI 为 some/path 的请求抛给 http://127.0.0.1

location /some/path/ {

proxy\_pass http://127.0.0.1;

}

3）将匹配 URI 为 name 的请求应用指定的 rewrite 规则，然后抛给 http://127.0.0.1

location /name/ {

rewrite /name/([^/]+) /users?name=$1 break;

proxy\_pass http://127.0.0.1;

}

## http\_proxy\_module 模块参数说明

Nginx 的代理功能是通过 http\_proxy 模块来实现的。默认在安装 Nginx 时已经安装了 http\_proxy 模块，因此可直接使用 http\_proxy 模块

[](https://img2018.cnblogs.com/blog/1210730/201904/1210730-20190410121714783-72008376.png)

<https://www.cnblogs.com/yanjieli/archive/2019/04/10/10682685.html#autoid-0-0-0>

## ngx\_http\_upstream\_module

**1.upstream**

**1.1.upstream语法**

Syntax: upstream name { ... }

Default: —

Context: http

**备注解析**

**1.upstream 配置http作用域下面**

**1.2.配置样例**

upstream backend {

server backend1.example.com weight=5;

server backend2.example.com;

server 127.0.0.1:8080 max\_fails=3 fail\_timeout=30s;

server backup1.example.com backup;

}

**备注-解析**

1.默认情况下，请求使用加权循环平衡方法在服务器之间分布。

2.每7个请求的分发如下:

2.1.五个请求进入backend1.example.com,

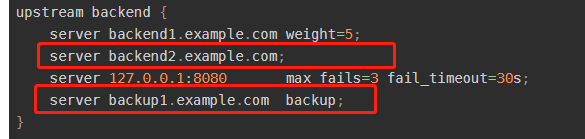
2.2.一个请求进入backend2.example.com

2.3.一个请求进入127.0.0.1:8080

2.4.如果在请求其中一个server时发生异常,会进入下一个server,直到所有的server都被重试

2.5.如果无法从任何服务器获得成功的响应，客户机将接收与最后一个服务器通信的结果。

**2.server**



**2.1.概念**

1.指定server的地址以及其他参数

2.地址可以指定为域名或者ip(或者带有端口,如果未指定端口,默认80)

3.其中server 指定的域名可以解析为多个IP地址的域名同时定义多个服务器。

**2.2.server 属性**

**2.2.1.weight**

1.设置server服务的权重,默认1

**案例:**

upstream backend {

server 127.0.0.1:8080 weight=2;

server 127.0.0.1:8081 weight=3;

server 127.0.0.1:8082 weight=5;

server 127.0.0.1:8083 ; # 默认相当于 weight=1;

}

**解析**

1.每进来(2+3+5+1)11个请求,会有

2个进入127.0.0.1:8080,

3个进入127.0.0.1:8081,

5个进入127.0.0.1:8082,

1个进入127.0.0.1:8083

**2.2.2.max\_conns**

**案例**

upstream backend {

server 127.0.0.1:8080 max\_conns=100;

server 127.0.0.1:8083 ; # 默认相当于 max\_conns=0;

}

**解析**

1.设置限制与代理服务器同时活动的最大连接数;

2.默认max\_conns的值为0,意味着没有限制;

**2.2.3.max\_fails**

**案例**

upstream backend {

server 127.0.0.1:8080 max\_fails=3 fail\_timeout=30s;

server 127.0.0.1:8081; #默认 相当于max\_fails=1

}

**解析**

1.设置请求server失败最大重试次数

2.与fail\_timeout 属性组合使用

3.默认失败重试最大次数 1

4. server 127.0.0.1:8080 max\_fails=3 fail\_timeout=30s;

max\_fails=3 fail\_timeout=30s代表在30秒内请求某一应用失败3次， 认为该应用宕机，

后等待30秒，这期间内不会再把新请求发送到宕机应用，而是直接发到正常的那一台，

等待的这30秒时间到后再有请求进来继续尝试连接宕机应用且仅尝试1次，

如果还是失败，则继续等待30秒...以此循环，直到恢复。

5.

**2.2.4.fail\_timeout**

**样例**

upstream backend {

server 127.0.0.1:8080 fail\_timeout=10s;

server 127.0.0.1:8081; #默认 相当于fail\_timeout=10s 10秒

}

**解析**

1.在指定的fail\_timeout时间范围内,超过max\_fails不成功的尝试次数,认为应用服务不可用;

2.fail\_timeout属性与max\_fails属性组合使用;

**2.2.5.backup**

upstream backend {

server 127.0.0.1:8080 fail\_timeout=10s;

server 127.0.0.1:8081 back;

}

1.标示 该server为备份的server 当主要的server不可用时,会请求到备份server中

2.如上127.0.0.1:8080 不可用时,会请求到127.0.0.1:8081

**2.2.6.down**

upstream backend {

server 127.0.0.1:8080;

server 127.0.0.1:8081 down;

}

**解析**

1.标示该server 已经永久不可用,宕机

**2.2.7.resolve**

http {

resolver 10.0.0.1;

upstream u {

server example.com resolve;

}

}

1.主要用于DNS解析,与http模块下的resolver 组合使用

**2.2.7.ip\_hash**

upstream backend {

ip\_hash;

server backend1.example.com;

server backend2.example.com;

server backend3.example.com down;

server backend4.example.com;

}

**解析**

1.指定组应使用负载平衡方法，其中基于客户端IP地址的请求在服务器之间分布。

2.客户端IPv4地址或整个IPv6地址的前三个八进制数用作哈希键。

3.该方法确保来自同一客户端的请求始终传递到同一服务器，除非该服务器不可用。

4.在后一种情况下，客户机请求将被传递到另一台服务器。

最有可能的是，它也将始终是相同的服务器

<https://blog.csdn.net/u014636209/article/details/87098898>

**ngx\_http\_auth\_basic\_module模块**

实现基于用户的访问控制，使用basic机制进行用户认证；

修改配置文件

location / {

root html;

index index.html index.htm;

auth\_basic "Admin"; ##认证对话框的提示字符串显示的内容

auth\_basic\_user\_file /usr/local/nginx/conf/htpasswd; ##存放认证用的用户名和文件，需要用htpasswd命令生成

}

**安装一下htpasswd命令**

yum install httpd-tools

**生成nginx用户和密码文件**

[root@localhost nginx]# htpasswd -c /usr/local/nginx/conf/htpasswd nginx

New password:

Re-type new password:

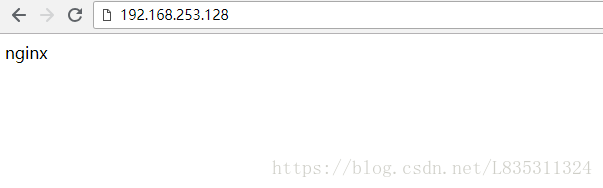
Adding password for user nginx

**重启一下nginx然后访问**

nginx -s reload



**然后用我们刚才设置的用户密码登陆就可以了**



**ngx\_http\_stub\_status\_module模块**

这个模块主要是用来查看http的状态信息的，使用方式直接在location里面加stub\_status;就可以了

示例

location /admin/ {

alias /var/www/html/;

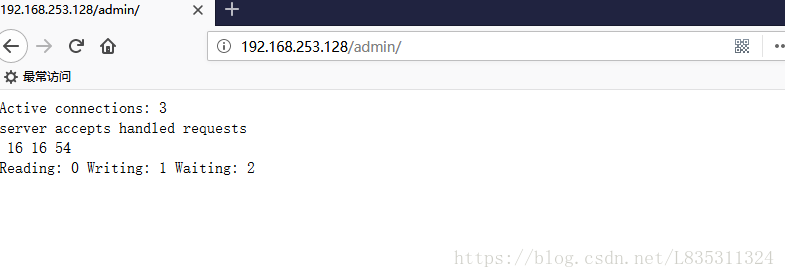
stub\_status;##重点是这一个指令

}

重新装载一下配置文件

nginx -s reload

接下来我们访问网站



Active connections: 活动状态的连接数；

accepts：已经接受的客户端请求的总数；

handled：已经处理完成的客户端请求的总数；

requests：客户端发来的总的请求数；

Reading：处于读取客户端请求报文首部的连接的连接数；

Writing：处于向客户端发送响应报文过程中的连接数；

Waiting：处于等待客户端发出请求的空闲连接数；

**ngx\_http\_log\_module模块**

nginx日志模块

几个重要的指令

log\_format name string …;

string可以使用nginx核心模块及其它模块内嵌的变量；

access\_log path [format [buffer=size] [gzip[=level]] [flush=time] [if=condition]];

access\_log off;

访问日志文件路径，格式及相关的缓冲的配置；

open\_log\_file\_cache max=N [inactive=time] [min\_uses=N] [valid=time];

open\_log\_file\_cache off;

缓存各日志文件相关的元数据信息；

max：缓存的最大文件描述符数量；

min\_uses：在inactive指定的时长内访问大于等于此值方可被当作活动项；

inactive：非活动时长；

valid：验正缓存中各缓存项是否为活动项的时间间隔；

**ngx\_http\_gzip\_module模块**

压缩模块，有利于传输数据的大小减少，但是cpu使用会变高。因为要对传输的数据进行压缩。

1. gzip on | off;

开启或关闭压缩功能

2.gzip\_comp\_level level;

设置压缩比，一般使用6

3. gzip\_disable regex …;

对请求报文的“User-Agent”匹配成功的请求，不进行压缩。

4. gzip\_min\_length length;

启用压缩功能的响应报文大小阈值；大于某个值才开启压缩功能

5. gzip\_buffers number size;

支持实现压缩功能时为其配置的缓冲区数量及每个缓存区的大小；

6. gzip\_proxied off | expired | no-cache | no-store | private | no\_last\_modified | no\_etag | auth | any …;

nginx作为代理服务器接收到从被代理服务器发送的响应报文后，在何种条件下启用压缩功能的；

off：对代理的请求不启用

no-cache, no-store，private：表示从被代理服务器收到的响应报文首部的Cache-Control的值为此三者中任何一个，则启用压缩功能；

gzip\_types mime-type …;

压缩过滤器，仅对此处设定的MIME类型的内容启用压缩功能；

**示例：**

gzip on;

gzip\_comp\_level 6;

gzip\_min\_length 64;

gzip\_proxied any;

gzip\_types text/xml text/css application/javascript;

ngx\_http\_ssl\_module模块

**设置https连接的模块**

1. ssl on | off;

是否开启ssl连接

2. ssl\_certificate file;

当前虚拟主机使用PEM格式的证书文件；

3. ssl\_certificate\_key file;

当前虚拟主机上与其证书匹配的私钥文件；

4. ssl\_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2];

支持ssl协议版本，默认为后三个；

5. ssl\_session\_cache off | none | [builtin[:size]] [shared:name:size];

builtin[:size]：使用OpenSSL内建的缓存，此缓存为每worker进程私有；

[shared:name:size]：在各worker之间使用一个共享的缓存；

6. ssl\_session\_timeout time;

客户端一侧的连接可以复用ssl session cache中缓存 的ssl参数的有效时长；

**先自建一个CA，创建证书和私钥文件**

(umask 077; openssl genrsa -out /etc/pki/CA/private/cakey.pem 4096)

openssl req -new -x509 -key /etc/pki/CA/private/cakey.pem -out /etc/pki/CA/cacert.pem -days 3655 #然后填写相关信息

mkdir -pv /etc/pki/CA/{certs,crl,newcerts}

touch /etc/pki/CA/{serial,index.txt}

echo 01 > /etc/pki/CA/serial

mkdir /usr/local/nginx/ssl

cd /usr/local/nginx/ssl

(umask 077; openssl genrsa -out /usr/local/nginx/ssl/ssl.key 2048)

openssl req -new -key /usr/local/nginx/ssl/ssl.key -out /usr/local/nginx/ssl/ssl.csr -days 365 ##填写相关信息。域名这里设置为www.ice.com.

openssl ca -in /usr/local/nginx/ssl/ssl.csr -out /usr/local/nginx/ssl/ssl.crt

**修改配置文件/usr/local/nginx/conf/nginx.conf**

**配置示例：**

server {

listen 443 ssl;

server\_name www.ice.com;

root /var/www/html;

ssl on;

ssl\_certificate /usr/local/nginx/ssl/ssl.crt;

ssl\_certificate\_key /usr/local/nginx/ssl/ssl.key;

ssl\_session\_cache shared:sslcache:20m;

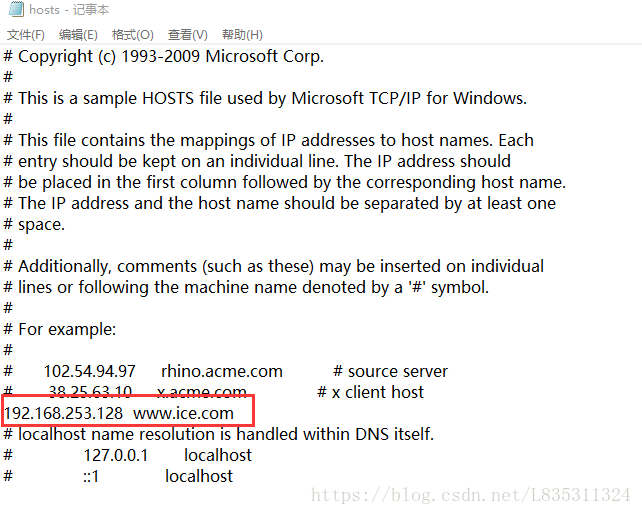
location / {

index index.html index.htm;

}

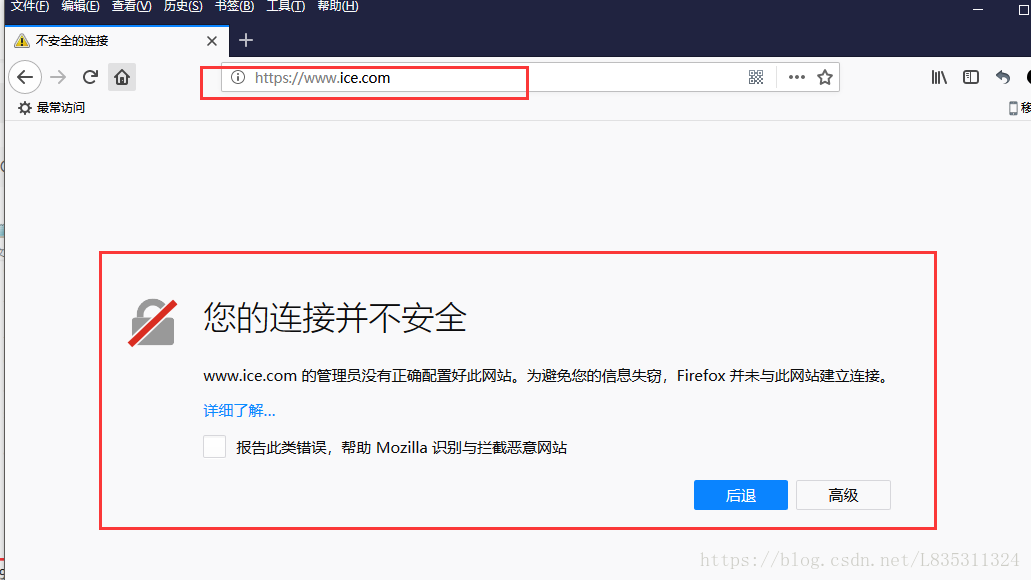
}

配置完成之后我们需要设置一下域名解析  
在这个目录下C:\Windows\System32\drivers\etc  
找到hosts文件并修改



保存退出

然后访问一下

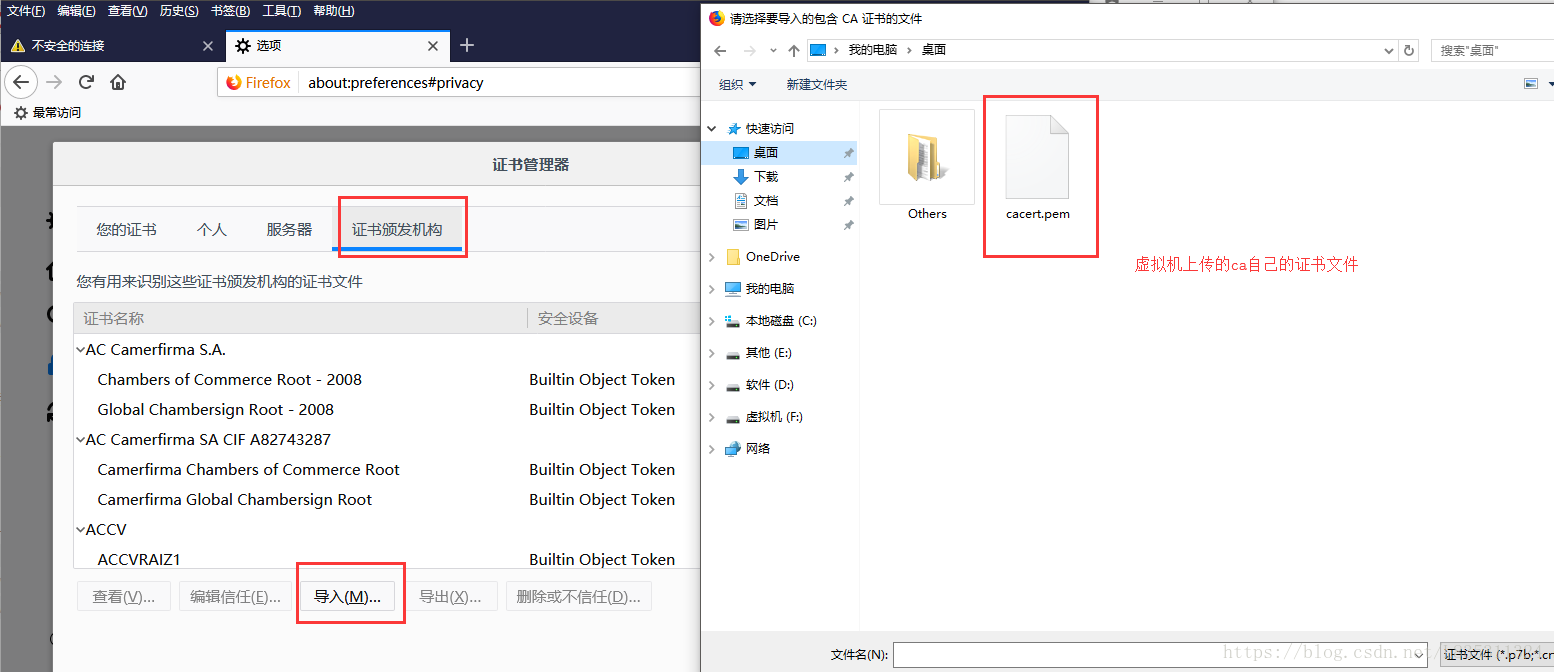


因为ca是自建的所以说不安全，我们在浏览器导入ca的证书，就可以了 。  
我们先吧虚拟机的ca文件复制到物理机

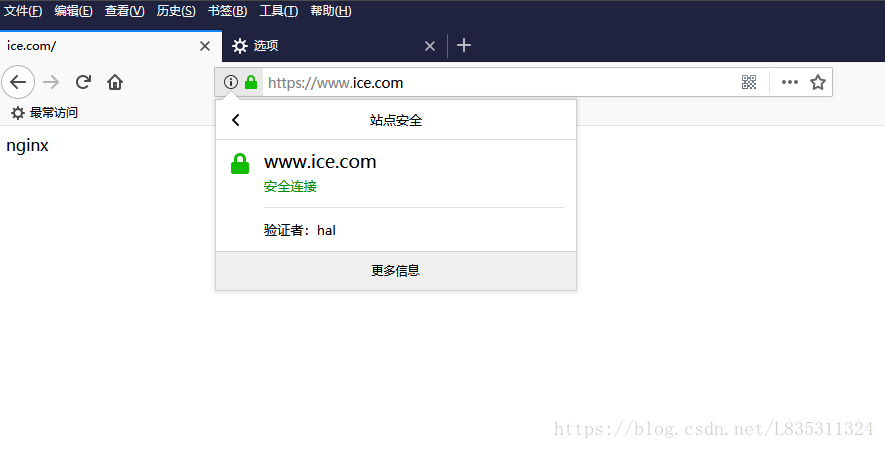
sz /etc/pki/CA/cacert.pem ##这个命令需要安装 lrzsz 安装方法 yum install lrzsz -y

导入证书





然后再次访问



这里显示安全链接了

**ngx\_http\_rewrite\_module模块**

常用的几个指令

1、rewrite regex replacement [flag]

将用户请求的URI基于regex所描述的模式进行检查，匹配到时将其替换为replacement指定的新的URI；

注意：如果在同一级配置块中存在多个rewrite规则，那么会自下而下逐个检查；被某条件规则替换完成后，会重新一轮的替换检查，因此，隐含有循环机制；[flag]所表示的标志位用于控制此循环机制；

[flag]：

last：如果规则有很多条。这里重写完成一次之后就会重新开始匹配规则，直至最后一条。也就是说。如果规则写的不好很容易造成死循环，不停的重写规则。

break：重写完成之后不再从头再次匹配规则。直接跳出循环

redirect：重写完成后以临时重定向方式直接返回重写后生成的新URI给客户端，由客户端重新发起请求；不能以http://或https://开头；

permanent:重写完成后以永久重定向方式直接返回重写后生成的新URI给客户端，由客户端重新发起请求；

示例比如我们要将http请求重定向到https请求。我们可以在http的server里面这样写里面这样写

rewrite (.\*)$ https://www.ice.com$1 break;

具体可以看官方文档  
<http://nginx.org/en/docs/http/ngx_http_rewrite_module.html#return>

**ngx\_http\_referer\_module模块**

可以基于这个模块做防盗链。Referer 是 HTTP 请求header 的一部分，当浏览器（或者模拟浏览器行为）向web 服务器发送请求的时候，头信息里有包含 Referer 。比如我在www.google.com 里有一个www.baidu.com 链接，那么点击这个www.baidu.com ，它的header 信息里就有：

Referer=http://www.google.com

valid\_referers none | blocked | server\_names | string …;

定义referer首部的合法可用值；

none：请求报文首部没有referer首部；

blocked：请求报文的referer首部没有值；

server\_names：参数，其可以有值作为主机名或主机名模式；

arbitrary\_string：直接字符串，但可使用\*作通配符；

regular expression：被指定的正则表达式模式匹配到的字符串；要使用~打头，例如 ~.\*.ice.com；

**配置示例：**

valid\_referers none block server\_names \*.ice.com ; #3设置有效的referers

if($invalid\_referer) {

return 403; ##如果referer是不再我们定义的valid\_referers列表中，则返回一个403拒绝访问

**ngx\_http\_headers\_module模块**

向由代理服务器响应给客户端的响应报文添加自定义首部，或修改指定首部的值；

**1、add\_header name value [always];**

添加自定义首部；

add\_header X-Via $server\_addr;

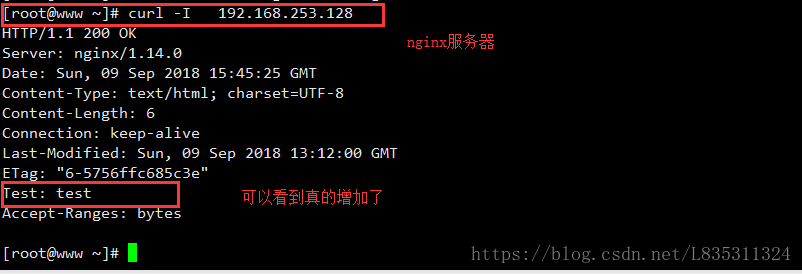
add\_header X-Accel $server\_name;

**2、expires [modified] time;**

expires epoch | max | off;

用于定义Expire或Cache-Control首部的值；

测试：在location中加上add\_header Test test; 在响应头部增加一个Test首部。内容是test。然后重启nginx，访问测试一下



<https://blog.csdn.net/L835311324/article/details/82560275>