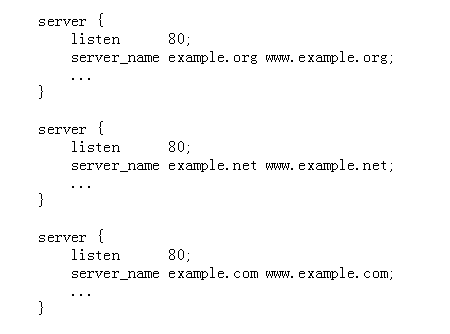
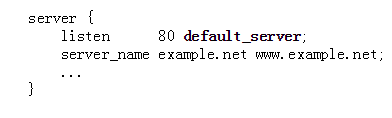
# nginx如何处理一个请求

nginx首先选定由那一个虚拟主机来处理请求

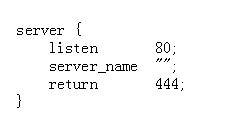
在这个配置中，nginx仅仅检查请求的host头以决定该请求由那个虚拟主机来处理，如果host头没有匹配任意一个虚拟主机，或者请求中根本没有包含host头，那nginx会将请求分发到定义在此端口上的默认虚拟主机，第一个被列出的虚拟主机就是nginx的默认虚拟主机。也可以显式的某个虚拟主机为默认虚拟主机(default\_server从0.8.21开始使用，在以前版本使用default代替)

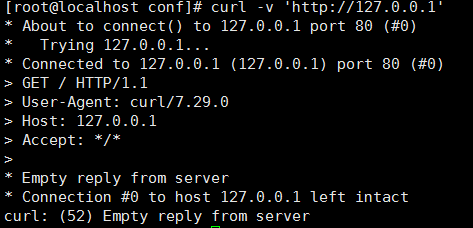
如果没有配置/etc/hosts，curl ‘http://127.0.0.1’ –H ‘host: abc’，是会被sever\_name abc接受的



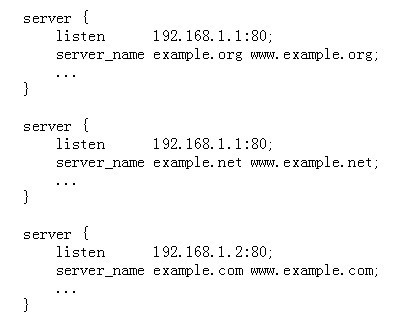
default\_server是监听端口的属性，而不是主机名的属性

如果不允许请求中缺少Host头，可以定义如下主机



设置主机名为空字符串以匹配未定义Host头的请求，而且返回了一个nginx特有的，非http标准码444，可以用来关闭连接

从0.8.48版本开始，server\_name “” 以成为主机名的默认设置，所以可以省略server\_name “”。而之前版本使用机器的hostname作为主机的默认设置



上面的配置中，nginx首先测试请求的IP地址和端口是否匹配某个server块中的listen指令配置。接着nginx继续测试请求host头是否匹配这个server块中的某个server\_name值，如果没有匹配则将这个请求交给默认主机。

默认服务器是监听端口的属性，所以不同的监听端口可以设置不同的默认服务器

# building nginx from sources

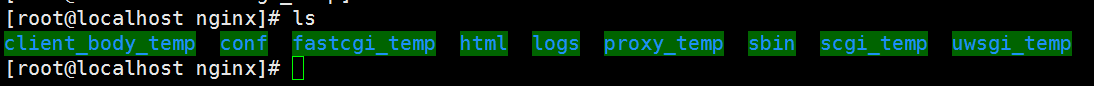
configure –help

打印帮助命令

--prefix=path

默认是/usr/local/nginx

定义一个目录，用来存放服务器文件，这个目录同样也是configure的相对路径以及nginx.conf文件定义的相对路径



--sbin-path=path

设置nginx的执行文件路径，这个路径仅仅在安装中有用，默认是prefix/sbin/nginx

--modules-path=path

定义一个目录，nginx的动态modules会被安装。默认是prefix/modules

--conf-path=path

默认是prefix/conf/nginx.conf

定义一个nginx.conf，如果有需要的话，每次启动可以指定不同的命令 –c filename

--error-log-path=path

设置错误日志路径，安装后，可以更改这个路径在nignx.conf里，使用error\_log指令

默认是prefix/logs/error.log

--pid-path=path

设置nginx.pid文件，也可以配置nginx.conf的pid指令。默认是prefix/logs/nginx.pid

--lock-path=path

设置lock文件，安装后也可以配置lock\_file指令。默认是prefix/logs/nginx.lock

--user=name

设置工作进程(worker process)以某非特权用户来运行（sets the name of an unprivileged user whose credentials will be used by worker processes）。安装后可以指定user指令来更改，默认为nobody

--group=name

sets the name of a group whose credentials will be used by worker processes. After installation, the name can always be changed in the nginx.conf configuration file using the [user](http://nginx.org/en/docs/ngx_core_module.html#user) directive. By default, a group name is set to the name of an unprivileged user

--build=name

设置nginx build name

--builddir=path

sets a build directory

--with-select\_module

--without-select\_module

开启或者禁止服务器使用select()方法，如果服务器不支持kqueue、epoll、/dev/poll该模块会被自动编译

--with-poll\_module

--without-poll\_module

开启或者禁止服务器使用poll()方法，如果服务器不支持kqueue、epoll、/dev/poll该模块会被自动编译

--with-threads

使用thread pools指令

--with-file-aio

使用asynchronous file I/O(AIO)在FreeBSD或者Linux

--with-http\_ssl\_module

这个模块不是默认编译，允许使用HTTPS\_protocol\_support模块，OpenSSL library需要编译

--with-http\_v2\_module

这个模块不是默认编译，允许使用HTTP/2模块

--with-http\_realip\_module

这个模块不是默认编译，允许使用ngx\_http\_realip\_module

--with-http\_addition\_module

这个模块不是默认编译，允许使用ngx\_http\_addition\_module

--with-http\_xslt\_module

--with-http\_xslt\_module=dynamic

这个模块不是默认编译，允许使用ngx\_http\_xslt\_module模块，转换XML响应使用一种或者多种XSLT风格

--with-http\_images\_filter\_module

--with-http\_images\_filter\_module=dynamic

这个模块不是默认编译，允许使用ngx\_http\_image\_filter\_module

--with-http\_geoip\_module

--with-http\_geoip\_module=dynamic

This module is not bulid by default，允许使用ngx\_http\_geoip\_module

--with-http\_sub\_module

This module is not bulid by default，允许使用ngx\_http\_sub\_module

--with-http\_dav\_module

This module is not bulid by default，允许使用ngx\_http\_dav\_module

--with-http\_flv\_module

This module is not bulid by default，允许使用ngx\_http\_flv\_module

--with-http\_mp4\_module

This module is not bulid by default，ngx\_http\_mp4\_module

--with-http\_gunzip\_module

This module is not bulid by default，ngx\_http\_gunzip\_module

--with-http\_gzip\_static\_module

This module is not bulid by default，ngx\_http\_gzip\_static\_module

--with-http\_auth\_request\_module

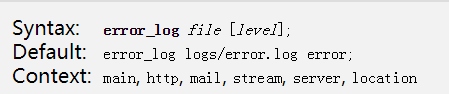
This module is not bulid by default，ngx\_http\_auth\_request\_module

--with-http\_random\_index\_module

This module is not bulid by default，ngx\_http\_random\_index\_module

# core functionality

## error\_log



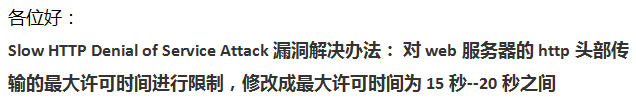
日志级别严重性从轻到重顺序:debug、info、notice、warn、error、crit、alert、emerg

如果file设置为stderr，nginx将会将日志输出到标准错误输出

设置某个日志级别将会使指定级别和更高级别的日志都被记录下来，如指定error会使nginx记录所有的error、crit、alert、emerg级别消息

debug日志需要—with-debug

# ngx\_http\_core\_module



## absolute\_redirect

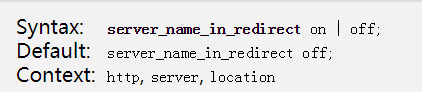
作用于http、server、location

默认absolute\_redirect on;

版本:>1.11.8

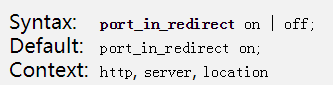
如果off，由nginx进行的重定向将是相对定位(redirects issued by nginx will be relative)

## server\_name\_in\_redirect



开启或者关闭nginx将server\_name指定的虚拟主机名用于发起重定向的功能。关闭此功能时，nginx将使用“HOST”头中的名字，如果没有HOST头，nginx将使用虚拟主机所在的IP地址

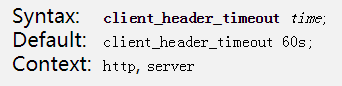
## port\_in\_redirect



开启或者关闭使用指定端口在绝对重定向中

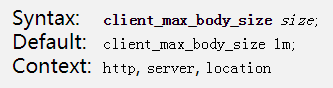
重定向中主机名的使用由server\_name\_in\_redirect指令控制

## client\_header\_timeout



定义读取客户端请求头部的超时。如果客户端在这段时间内没有传送完整的头部到nginx， nginx将返回错误408 (Request Time-out)到客户端

## client\_max\_body\_size



设置允许客户端请求正文的最大长度，请求长度由Content-Length请求头指定。如果超过设定值，将返回413(Request Entity Too Large)。浏览器不能正确显示这个错误。size为0可以使nginx不检查客户端请求正文长度

curl –H ‘Content-Length `expr 1024 \\* 1024 + 1`’ ‘ip’

当实际传的正文大于nginx限制，但是重写Content-Length，不会报错

curl的get访问最大是8173

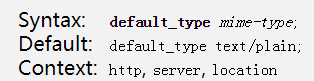
## connection\_pool\_size



允许微调为每个连接分配的内存，这个指令对nginx的性能影响非常小，一般不应该使用

默认值256在32位系统，512在64位系统

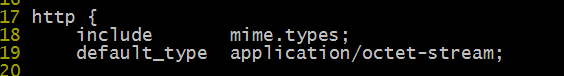
## default\_type



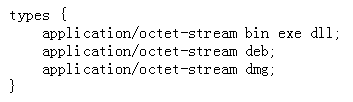
定义默认响应类型，设置文件扩展名和响应的MIME类型的映射表应该使用types指令

## types



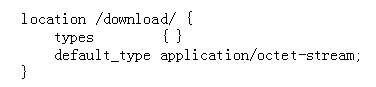


设置文件扩展名和响应的MIME类型的映射表，可以将多个扩展名映射到同一种类型

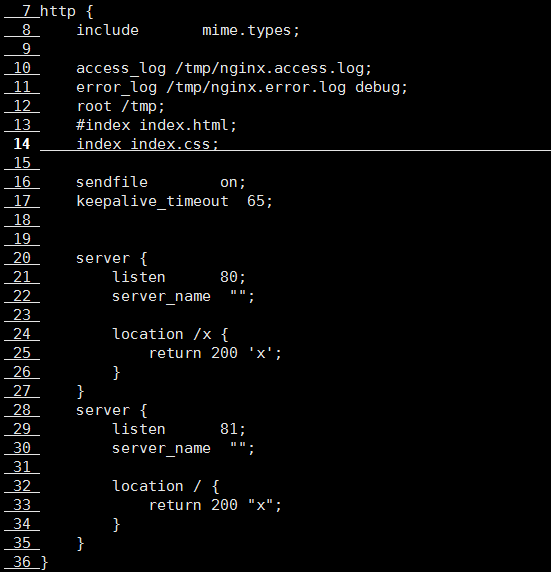


conf/mime.type包含了足够多的映射表

为了某个路径的所有请求生成MIME类型，可以使用下面配置

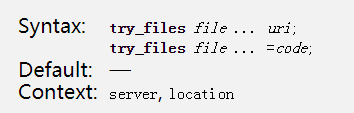


总结



请求uri为:80/，则content-type为text/css，该映射在conf/mime.types中，请求uri为/x，则响应text/plain，走默认default\_type

## try\_files



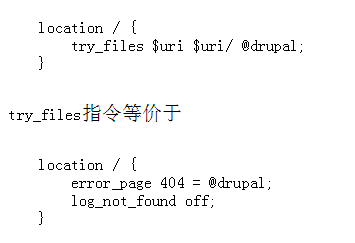
按照指定顺序检查文件是否存在，并且使用第一个找到的文件来处理请求。文件路径是根据root和alias指令，将file参数拼接而成

可以在名字尾部添加斜线以检查目录是否存在，比如”$uri/”

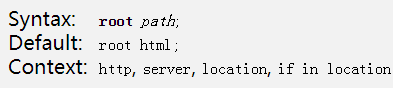
如果找不到，则按最后一个参数指定的uri进行内部跳转

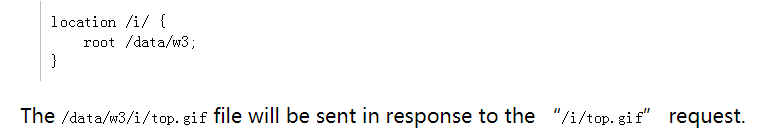


uri为/b，则try\_files /tmp/ccc找不到则尝试找/tmp/cc，找不到则发起内部重定向uri为ccc，最后返回的http\_code为500



## root

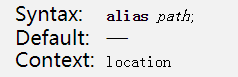




path值可以是变量，但是不能是$document\_root和$realpath\_root

仅仅是将uri拼到root值的后面

## alias



指定一个指定路径的替换路径

**alias和rewrite不能同时出现**

location /i/ {

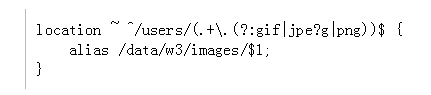
alias /data/w3/images/;

}

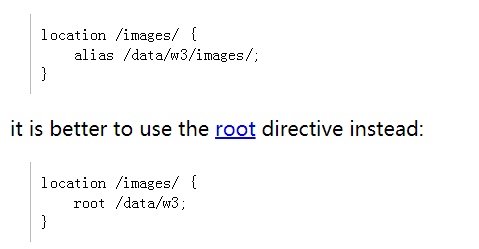
请求uri为/i/top.gif，则相应/data/w3/images/top.gif

path可以是变量，但是不能是$document\_root和$realpath\_root

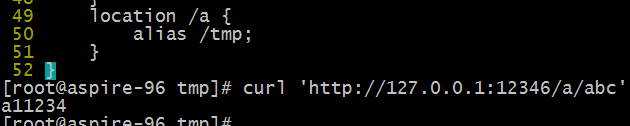
如果在定义了正则表达式的路径中使用了alias，那么正则表达式中应该含有匹配组，并且alias该引用这些匹配组来组成一个完整的文件匹配



如果路径对应指令path值的最后一部分，那么最好应该用root指令



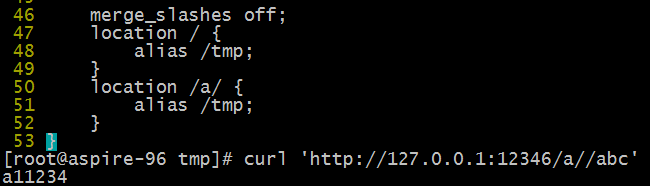
总结：



请求资源/tmp/abc



请求资源/tmpabc，404；因为两个”/”被转成了一个”/”



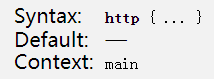
请求资源/tmp/abc，因为关闭了merge\_slashes，两个”/”不会转成一个”/”





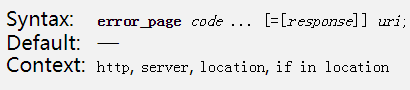
请求资源/tmp/abc

## http



为服务器提供配置文件上下文

## error\_page



为错误定义显示的URI



而且可以使用=response语法来改变响应码，404 =200（404后面要有一个空格）

C:\Users\ZHUXIA~1\AppData\Local\Temp\WeChat Files\71180b15be6e9f691771f2967e2a357.png

会再找location /empty.gif，或者root+/empty.gif

也可以进行重定向

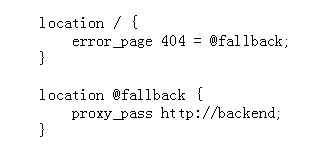
C:\Users\ZHUXIA~1\AppData\Local\Temp\1526867935(1).png

如果URI将被发送到一个被代理的服务器处理，或者发送到一个FastCGI服务器处理，这些后端服务器又返回了不同的响应码，那么这些响应码可以由本指令处理

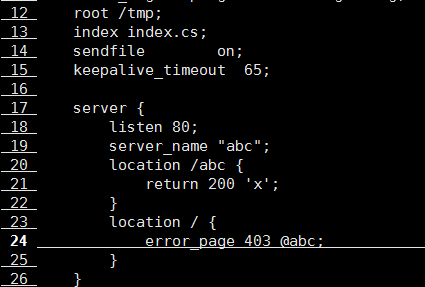
=左右都有一个空格

C:\Users\ZHUXIA~1\AppData\Local\Temp\WeChat Files\672ca9176ec427585eaecf47fde5f14.png

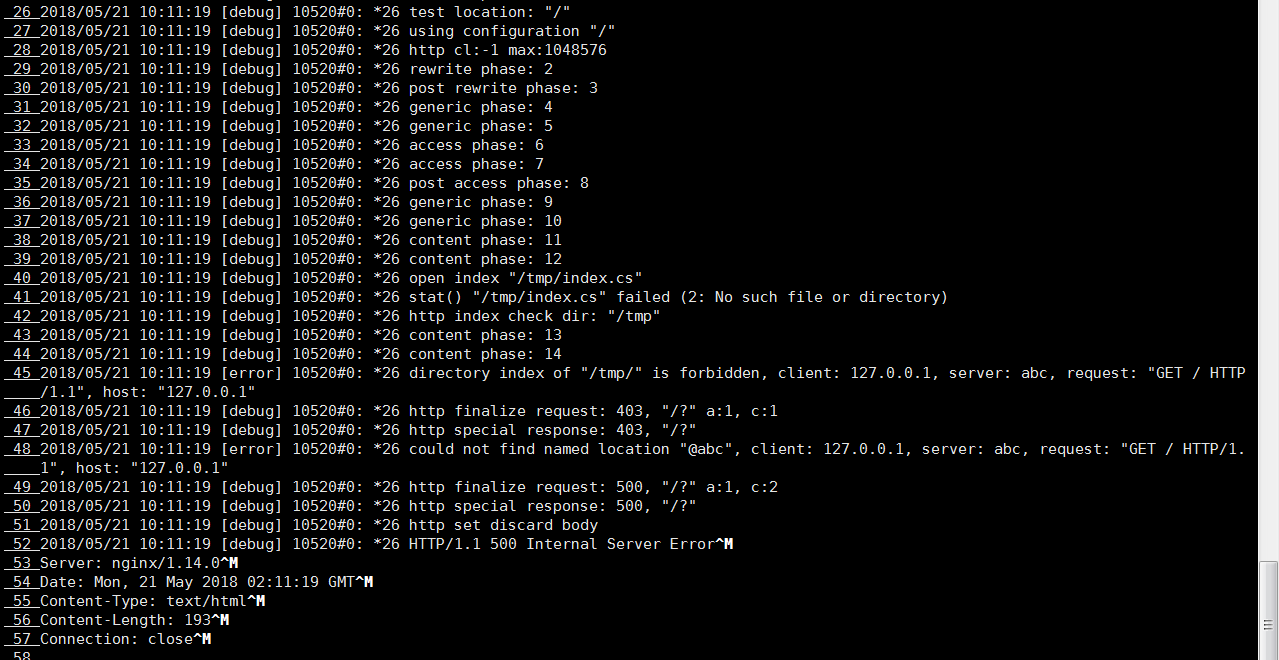
如果内部跳转无需改变uri，可以将错误处理转到一个命名路径



总结



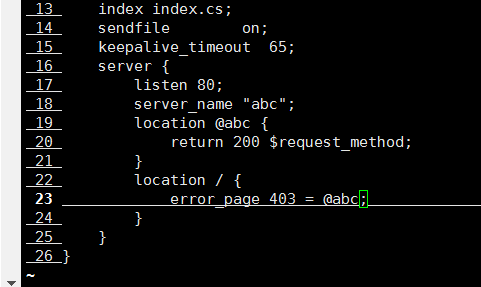
因为uri为/，因为没有location @abc，所以返回http\_code 500



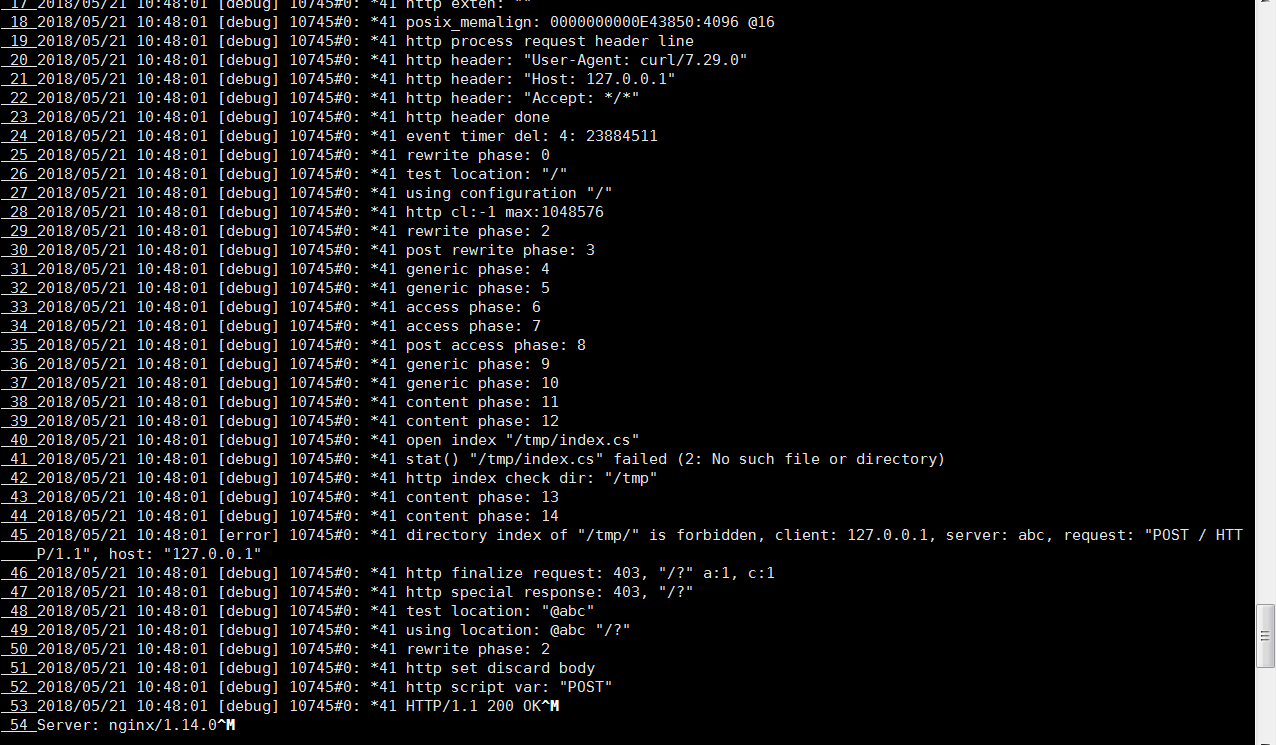
erro\_page 403 = @abc; location @abc; //=左右都有一个空格，http响应码使用@abc的

error\_page 403 =200 @abc; location @abc; //=左面有一个空格，http响应码使用200

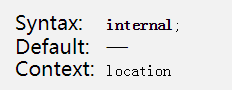
这个指令会导致内部重定向，并且客户端的请求方式为最初的请求方式



使用curl -X POST ‘http://127.0.0.1’ 返回POST



## interal

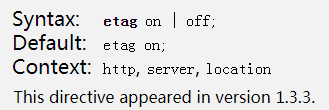


指定一个路径是否只能用于内部访问，如果是外部访问，客户端将收到404，内部访问是：

1. error\_page、index、random\_index、和try\_files指令引起的重定向
2. 由后端服务器返回的X-Accel-Redirect响应头引起的重定向
3. 由ngx\_http\_ssi\_module和ngx\_http\_addition\_module模块的include virtual指令产生的子请求
4. 用rewrite指令对请求进行修改

nginx限制每个请求只能最多进行10次内部重定向，以防配置错误引起请求处理出现问题，如果已经达到10次，nginx将返回500，同时日志中有rewrite or internal redirection cycle

## etag



开启或者关闭静态文件自动计算etag响应头

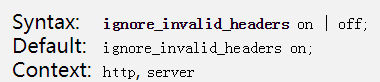
**只是在静态文件中使用**

## underscores\_in\_headers



允许或者禁止在客户端请求头中使用下划线。如果禁止，含有下划线的请求头将被标志为非法请求头并接受igonre\_invalid\_header指令处理

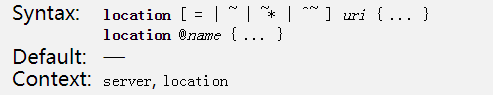
## ignore\_invalid\_headers



控制是否忽略非法的请求头字段名，合法的名字是**英文字母、数字、和连字符(-)**组成，当然也可以包含下划线，不过判断下划线是否合法需要根据underscores\_in\_headers指令来判断。

非法的请求头PHP的$\_SERVER是接受不到的

## location



根据请求的URI来设置配置

路径匹配在URI规范化以后进行，所谓规范化，就是先将URI中形如”%XX”的编码字符进行编码，再解析URI中的相对路径”.”和”..”部分，另外还可能会压缩相邻的两个或多个斜线成为一个斜线(需要merge\_slashes on;)

可以使用前缀字符串或者正则表达式定义路径。使用正则表达式需要在路径开始添加“~\*”前缀 (不区分大小写)，或者“~”前缀(区分大小写)。为了根据请求URI查找路径，nginx先检查前缀字符串定义的路径 (前缀路径)，在这些路径中找到能最精确匹配请求URI的路径。然后nginx按在配置文件中的出现顺序检查正则表达式路径， 匹配上某个路径后即停止匹配并使用该路径的配置，否则使用最大前缀匹配的路径的配置。

**路径可以嵌套**

****

如果最大前缀匹配的路径以“^~”开始，那么nginx不再检查正则表达式

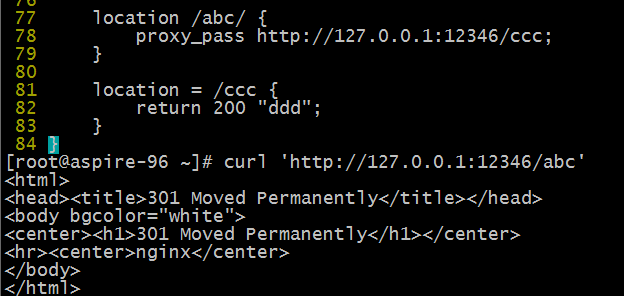
正则表达式中可以包含匹配组(0.7.40)，结果可以被后面的其他指令使用

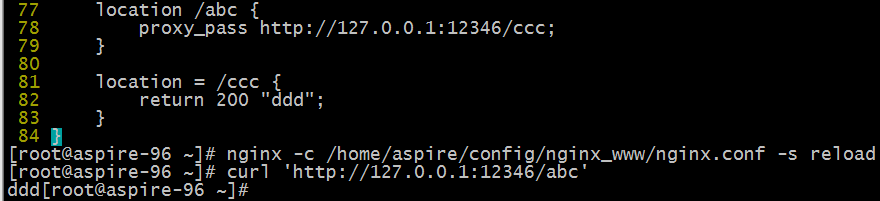
使用“=”前缀可以定义URI和路径的精确匹配。如果发现匹配，则终止路径查找。 **比如，如果请求“/”出现频繁，定义“location = /”可以提高这些请求的处理速度， 因为查找过程在第一次比较以后即结束。这样的路径明显不可能包含嵌套路径**

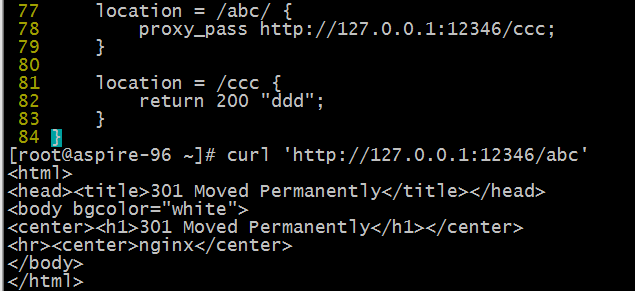
前缀“@”定义了命名路径。这种路径不在一般的请求处理中使用， 而是用在请求重定向中。这些路径不能嵌套，也不能包含嵌套路径

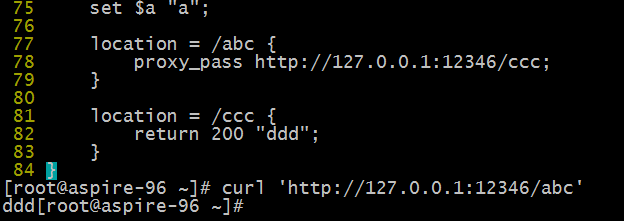
在精准匹配或者常规字符串匹配中，如果请求经过proxy\_pass、fastcgi\_pass、uwsgi\_pass、scgi\_pass、memcached\_pass、grpc\_pass，下面的例子会发生重定向

如果location /abc/ {return 200 “acv”;} uri为/abc是会open root+/abc的









总结：

~ 执行一个正则匹配，区分大小写，匹配不继续匹配

~\* 执行一个正则匹配，不区分大小写，匹配不继续匹配

^~ 前缀匹配，匹配成功则停止

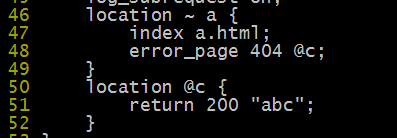
= 精准匹配，匹配成功停止匹配

/abc 常规字符串匹配

@ 定义一个命名的location，使用在内部定向时，如error\_page

所有请求都会被 location / {} 匹配

location @



=类型优先级最高，一旦匹配成功则不再查找其他

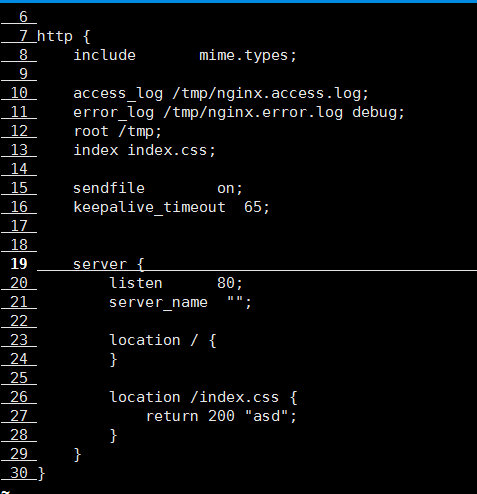
^~，优先级次之，一旦匹配则不再查找

正则匹配优先级次^~

常规字符串匹配，优先级最低，如果有多个能匹配的话，使用匹配最长的那个

Location =与~^不支持正则匹配，location = /abc$不是正则的结束符

**Location ^~ /a与location /a不能同时写**

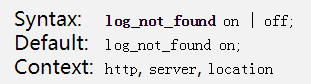


uri为/，然后走location /，root+uri+index有这个文件，然后发生内部重定向，此时uri变成/index.css，走location /index.css

### location、index失败情况

1. 如果uri /a，location /a {}，则会寻找root+/a，找不到则404
2. 如果uri /a，location /a {}，则会寻找root+/a，如果root+/a是目录，则uri变为/a/，发生内部重定向，找不到则403
3. 如果uri为/，没有匹配location，则会寻找root+index，找不到则403

## log\_not\_found



log\_not\_fount on/off;

在server、http、location**(location配置关闭模拟不出来还是会记录错误日志)**

开启或者关闭在error\_log中记录文件不存在的错误

默认开启





/tmp/index.css没有该文件，返回403；如果uri为/a，/tmp/a没有该文件，返回404

## log\_subrequest

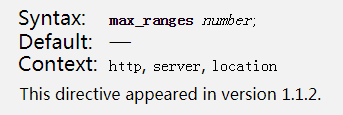
**没有模拟出来(proxy\_pass、rewrite、location @不是子请求)**

开启或者关闭在access\_log记录子请求的日志

配置与http、server、location

默认关闭

## max\_ranges



在http、server、location

version -> 1.1.2

如果请求中含有字节范围的请求头，max\_ranges指令可以限制此范围允许的最大值。如果请求头的值超过此限制，将按请求未携带此请求头的情况处理。默认不限制，设置为0将使nginx完全不支持HTTP字节范围特性

总结

return 200 “aaaa”，fastcgi、proxy不支持max\_ranges

静态文件支持

## merge\_slashes

在http、server中

默认merge\_slashes on

开启或者关闭将请求中URI相邻的两个或者更多斜线合并成一个的功能

压缩URI对于前缀匹配和正则匹配的正确性是很重要的，没有开启这个功能时，请求//script/one.php不能被location /scripts/匹配 **如果URI中包含base64编码内容，必须将斜线压缩调整成off**，因为base64编码本身会使用“/”字符，然而出于安全方面的考虑，最好还是不要关闭压缩

## msie\_padding

上下文http、server、location

默认msie\_padding on;

curl请求测试无效

在响应状态码大于或等于400的时候，在响应正文中添加一段注释，使响应正文达到512字节。**可以为MSIE客户端开启**

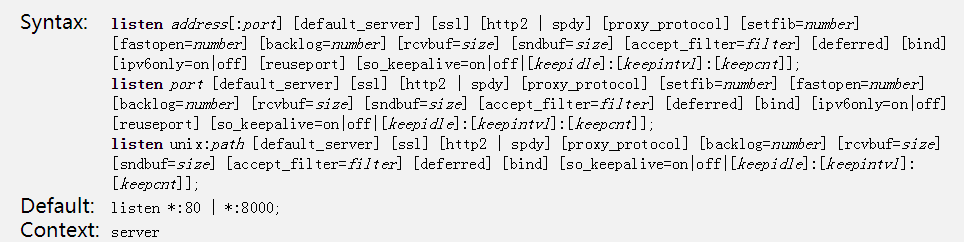
## msie\_refresh

在http、server、location

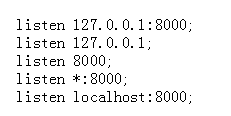
默认msie\_refresh off

**为MSIE客户端开启或者关闭用页面刷新取代页面重定向的功能**

## listen



设置nginx监听地址，nginx从这里接受请求。对于IP协议，这个地址就是address和port，对于UNIX域套接协议，这个地址就是path。address可以是hostname。一个listen只能指定一个address或者port



IPV6地址用方括号来表示

C:\Users\ZHUXIA~1\AppData\Local\Temp\1526277521(1).png

UNIX域套接字层使用unix：前缀

C:\Users\ZHUXIA~1\AppData\Local\Temp\1526277550(1).png

如果只定义了address，nginx将使用80端口

**在没有定义listen指令的情况下，如果以超级用户权限运行nginx，将监听80，否则监听\*:8000**

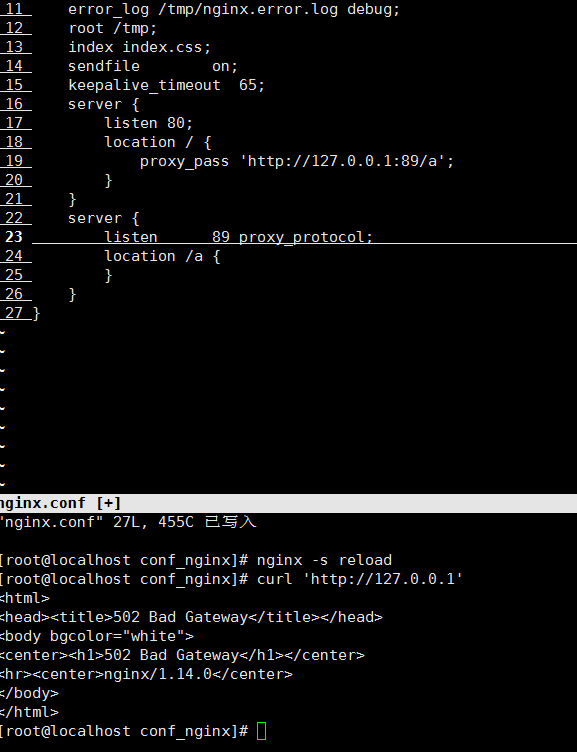
**如果listen 80；使用普通用户启动将失败，因为普通用户只能使用大于1024以上的端口**

**普通用户启动nginx，需要/usr/local/nginx目录的权限**

如果listen携带default\_server参数，当前虚拟主机将成为指定address:port的默认虚拟主机，如果任何listen都没有携带default\_server，那么第一个监听address:port的虚拟主机将被作为这个地址的默认虚拟主机（0.8.21以前这个参数是default）

**proxy\_protocol（1.5.12）**

允许指定在该端口上接受的请求都应该使用代理模式



**http2**

HTTP2参数（1.9 5）配置端口接受HTTP / 2连接。通常，为了工作，SSL参数也应该被指定，但是NGNIX也可以被配置为接受没有SSL的HTTP / 2连接

**setfib=number**

在0.8.44版本为监听套接字设置关联路由表FIB，仅仅工作在FreeBSD上

**backlog=number**

限制未接受accept连接的队列最大长度。FreeBSD和Mac OS X下，默认为-1，其他为511

**rcvbuf=size**

为监听套接字设置接受缓冲区大小(SO\_REVBUF)

**sndbuf=size**

为监听套接字设置发送缓冲区大小(SO\_SNDBUF)

**accept\_filter=filter**

为监听套接字设置接受过滤器的名称（SO\_ACCEPTFILTER）。每个到来的连接，接受过滤器先进行过滤，然后才将他们呈现给accept，可以接受的值是dataready和httpready。仅在FreeBSD和NetBSD5.0+系统下

**deferred**

在linux系统使用延迟的accept()(TCP\_DEFER\_ACCEPT)

**bind**

设置address:port单独调用一次bind（），这是因为当有多条listen指令监听不同地址下的相同端口，而其中一条listen指令监听了这个端口的所有地址\*:port时，nginx只会为\*:port调用一次bind。需要留意的是，这种情况下，nginx会调用getsockname()系统调用来确定接受请求的套接字地址。 如果为某个address:port定义了参数backlog、rcvbuf、 sndbuf、accept\_filter、deferred或者so\_keepalive， nginx总会为这个地址单独调用一次bind()绑定套接字

**ipv6only=on|off**

决定监听在通赔地址[::]上的IPV6套接字是只支持IPV6连接还是同时支持IPV4。默认打开，而且只能在nginx启动时候设置

**ssl（需要ngx\_http\_ssl\_module）**

与套接字层相关的系统调用无关，但是可以指定这个端口接受的连接应该以SSL模式工作

**so\_keepalive=on|off[keepidle]:[keepintv]:[keepcnt]**

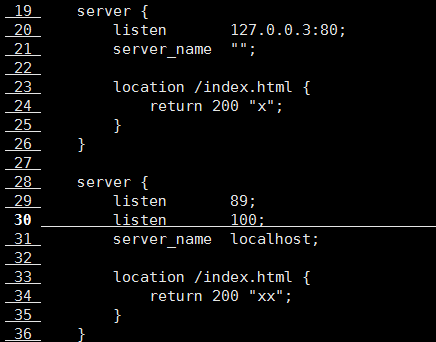
这个参数(1.1.11)为监听套接字配置“TCP keepalive”行为。 如果省略此参数，操作系统默认的设置将对此端口生效。 如果参数值设置为“on”，监听套接字的SO\_KEEPALIVE属性将被开启。 如果参数值设置为“off”，监听套接字的SO\_KEEPALIVE属性将被关闭。 有些操作系统支持为每个连接调整TCP长连接的参数。调整参数可以使用套接字选项TCP\_KEEPIDLE，TCP\_KEEPINTVL和TCP\_KEEPCNT。 在这些操作系统上(当前就是Linux 2.4+，嬀NetBSD 5+FreeBSD 9.0-STABLE)，可以使用keepidle，keepintvl和keepcnt参数来配置。 省略一到两个参数的话，对应套接字属性的系统默认设置将生效

so\_keepalive=30m::10

将 设置空闲超时(TCP\_KEEPIDLE)为30分钟， 设置探测次数(TCP\_KEEPCNT)为10次， 保留探测时间间隔(TCP\_KEEPINTVL)为系统默认值

总结：

一个server可以配置两个listen



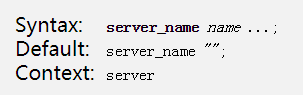
listen 80;//意思就是可以让任何ip来访问80端口

listen 127.0.0.1.; //只默认80端口或者8000

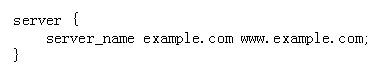
### netstat

netstat –tlnp

## server\_name

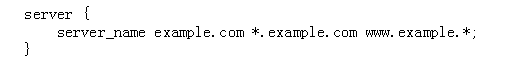


设置虚拟主机名字



第一个名字成为虚拟主机的主要名字

主机名中可以含有\*，用来替代名字开始部分和结束部分，这被称为通配符主机

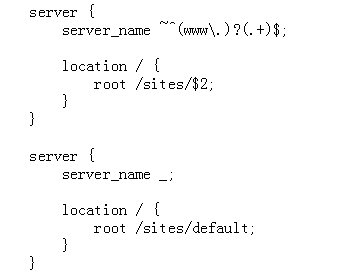


主机example.com和\*.example.com可以组合成一个.example.com

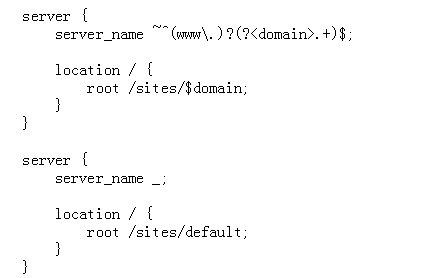
正则表达式主机，就是在名字前面补一个~^

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

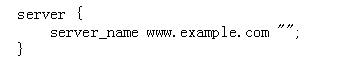
正则可以包含匹配组



可以创建变量名



可以定义空主机名



空主机名可以让虚拟主机处理没有“Host”请求头的请求，而不是让指定“地址：端口”的默认虚拟主机来处理

如果一个名字可以匹配多个指定的配置，比如同时匹配上通配符和正则表达式，则按优先级来处理

1. 确切的名字
2. 最长的以星号起始的通配符名字
3. 最长的以星号结束的通配符名字
4. 第一个匹配的正则表达式名字，按在配置文件中出现的顺序

总结

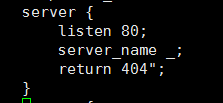
nginx的server\_name为~^actapi.\w+.dev.shop$，请求<http://actapi.zhuxiaoyu.dev.shop>，收到的$\_SERVER[“SERVER\_NAME”]为~^actapi.\w+.dev.shop$，$\_SERVER[“HTTP\_HOST”]为actapi.zhuxiaoyu.dev.shop



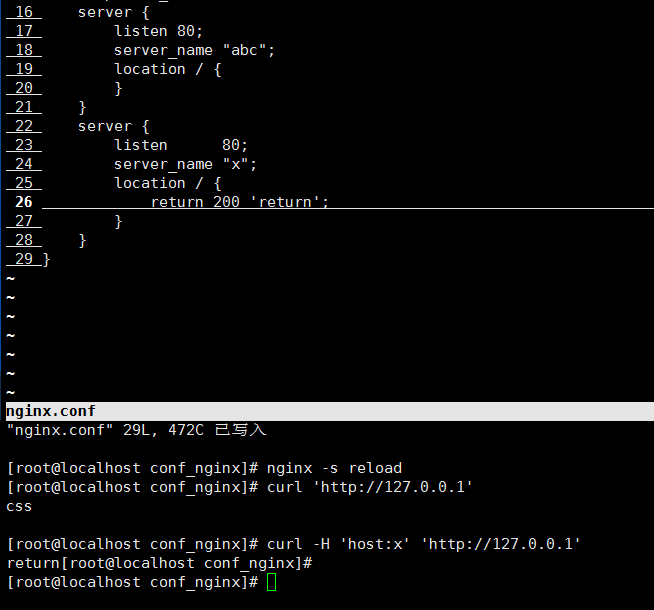
配置hosts 127.0.0.1 [www.baidu.com](http://www.baidu.com);请求[www.baidu.com就返回x](http://www.baidu.com就返回x)

如果server\_name每个server\_name都没匹配，就用第一个

如果要加一个默认的阻挡，匹配了listen而不匹配server\_name，就在**最前面**加一个server\_name \_;







### linux hosts

每台主机都有一个IP地址，用来区分当前是哪一个机器。因为IP不方便记忆，所以有了域名，每个域名对应一个IP地址，但是一个IP地址可以对应多个域名

每台机器都有一个主机名，用于主机于主机之间的区别。  
 vim /etc/hosts (主机名查询静态表，负责IP地址与主机名之间的映射，还包括主机名的别名)

主机名与域名：

主机名：

在一个局域网中，每台机器都有一个主机名，用于主机于主机之间的区别，可以为每台服务器设置主机名，以便于互相访问；就是机器本身的名字

域名：

域名是用来解析到IP的；在局域网中，主机名也可以解析到IP上

优先级：DNS缓存>hosts>DNS服务

192.168.1.100 a b

在网页中输入 http://a 和<http://b> 都打开192.168.1.100

127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4

hostname

显示主机名

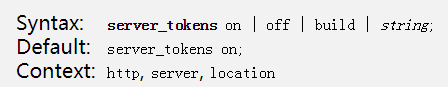
hostname abc

临时设置主机名为abc，关机失效

vim /proc/sys/kernel/hostname

为永久修改

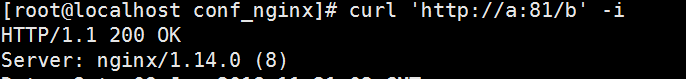
## server\_tokens



开启或者关闭在响应头中输出nginx版本号

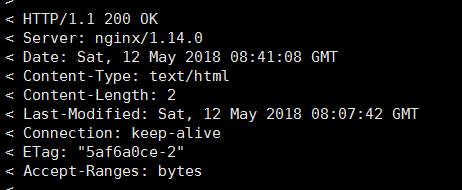
参数build可以开启显示./configure –build=name 在错误信息中

--build=8

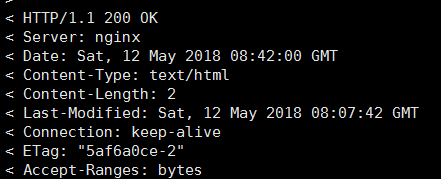


**使用string，一直报错**

**Additionally, as part of our**[**commercial subscription**](http://nginx.com/products/)**, starting from version 1.9.13 the signature on error pages and the “Server” response header field value can be set explicitly using the string with variables. An empty string disables the emission of the “Server” field.**

****

server\_tokens off;

****

## 长连接相关

### 长连接与短连接

HTTP的长连接和短连接本质上是TCP的长连接和短连接，HTTP属于应用层协议，在传输层使用TCP协议，在网络层使用IP协议。 IP协议主要解决网络路由和寻址问题，TCP协议主要解决如何在IP层之上可靠地传递数据包，使得网络上接收端收到发送端所发出的所有包，并且顺序与发送顺序一致

HTTP/1.0默认使用短连接，客户端与服务器每次进行HTTP操作都建立一次连接，连接后就中断连接。客户端浏览器访问的某个HTML或其他类型的Web页中包含有其他的Web资源（如JavaScript文件、图像文件、CSS文件等），每遇到这样一个Web资源，浏览器就会重新建立一个HTTP会话。

HTTP/1.1默认使用长连接，会在响应头中返回

Connection:keep-alive

在使用长连接的情况下，当一个网页打开完成后，客户端和服务器之间用于传输HTTP数据的TCP连接不会关闭，客户端再次访问这个服务器时，会继续使用这一条已经建立的连接。Keep-Alive不会永久保持连接，它有一个保持时间，可以在不同的服务器软件（如Apache）中设定这个时间。实现长连接需要客户端和服务端都支持长连接

如果HTTP/1.1版本请求报文不希望使用长连接，则要在HTTP请求报文首部加上Connection:close

有部分古老的HTTP1.0 代理不理解Keep-alive，而导致长连接失效：客户端–>代理–>服务端，客户端带有Keep-alive，而代理不认识，于是将报文原封不动转给了服务端，服务端响应了Keep-alive，也被代理转发给了客户端，于是保持了“客户端–>代理”连接和“代理–>服务端”连接不关闭，但是，当客户端第发送第二次请求时，代理会认为当前连接不会有请求了，于是忽略了它，长连接失效

常见的是客户端跟Nginx代理服务器使用HTTP1.1协议&长连接，而Nginx代理服务器跟后端服务器使用HTTP1.0协议&短连接

HTTP头部有了Keep-Alive这个值并不代表一定会使用长连接，客户端和服务器端都可以无视这个值，也就是不按标准来，譬如我自己写的HTTP客户端多线程去下载文件，就可以不遵循这个标准，并发的或者连续的多次GET请求，都分开在多个TCP通道中，每一条TCP通道，只有一次GET，GET完之后，立即有TCP关闭的四次握手，这样写代码更简单，这时候虽然HTTP头有Connection: Keep-alive，但不能说是长连接。正常情况下客户端浏览器、web服务端都有实现这个标准，因为它们的文件又小又多，保持长连接减少重新开TCP连接的开销很有价值

使用长连接之后，客户端、服务端怎么知道本次传输结束呢？两部分：1是判断传输数据是否达到了Content-Length指示的大小；2动态生成的文件没有Content-Length，它是分块传输（chunked），这时候就要根据chunked编码来判断，chunked编码的数据在最后有一个空chunked块，表明本次传输数据结束

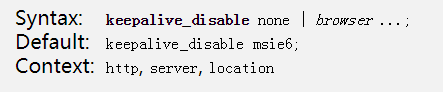
TCP的keep alive是检查当前TCP连接是否活着；HTTP的Keep-alive是要让一个TCP连接活久点。它们是不同层次的概念

TCP keep alive的表现：

当一个连接“一段时间”没有数据通讯时，一方会发出一个心跳包（Keep Alive包），如果对方有回包则表明当前连接有效，继续监控

http://blog.jobbole.com/104108/

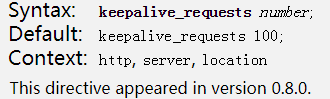
### keepalive\_disable



针对行为异常的浏览器关闭长连接功能。brower参数指定那些浏览器会受到影响，值为msie6表示在遇到POST请求时，关闭与老版本MSIE浏览器建立长连接。值为safari表示在遇到Mac OS X和类Mac OS X操作系统下的safari浏览器和类safari浏览器时，不与浏览器建立长连接。值为none表示为所有浏览器开启长连接功能

在nginx 1.1.18版本以前，safari将匹配所有操作系统上的safari和类safari浏览器，并默认不与这些浏览器建立长连接

### keepalive\_requests



设置通过一个长连接可以处理的最大请求数，请求超过此数值，长连接将关闭

### keepalive\_timeout

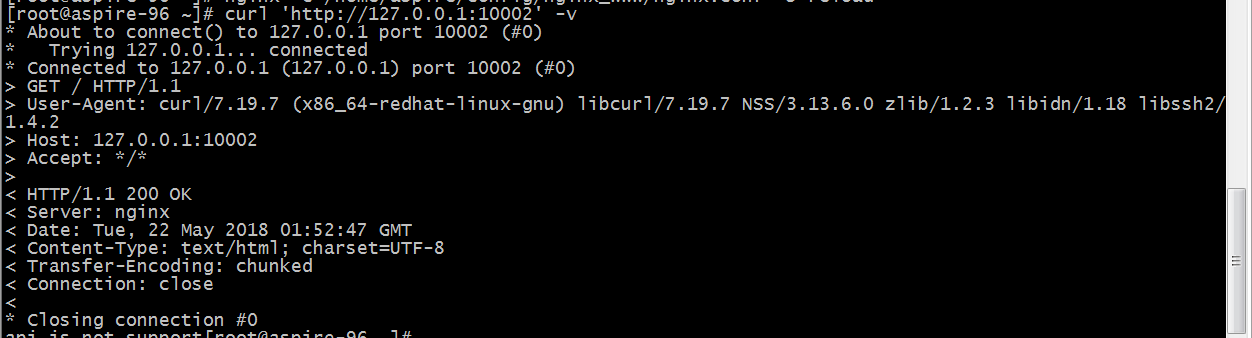


第一个参数设置客户端的长连接在服务器端保持的最长时间（在此时间客户端未发起新请求，则长连接关闭）。 第二个参数为可选项，设置“Keep-Alive: timeout=time”响应头的值。 可以为这两个参数设置不同的值

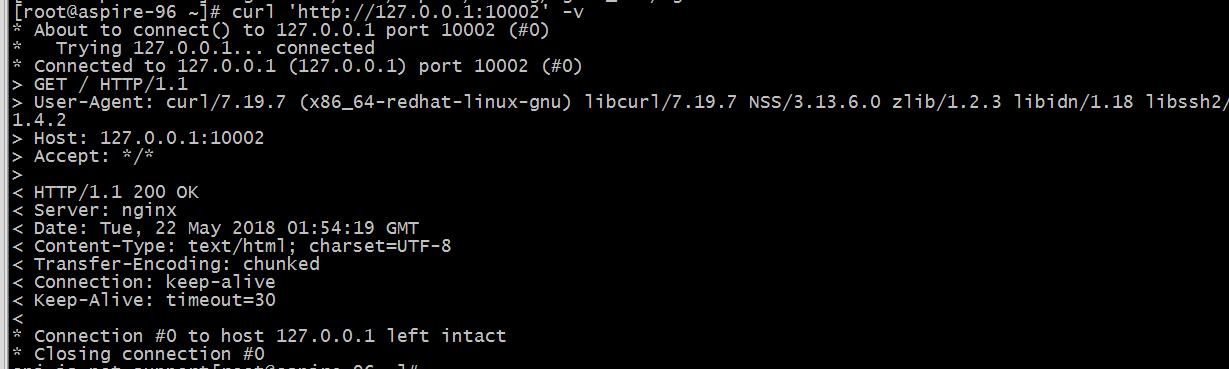
值0将禁止客户端长连接

“Keep-Alive: timeout=time”响应头可以被Mozilla和Konqueror浏览器识别和处理。 MSIE浏览器在大约60秒后会关闭长连接

keepalive\_timeout 0 20s;



keepalive\_timeout 20s 30s;



# ngx\_http\_addition\_module

ngx\_http\_addition\_module是一个过滤模块，用来添加内容在响应前和后

这个模块不默认编译，需要—with-http\_addition\_module

after和before相当于一个子请求

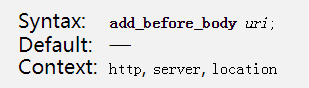
after和before相当于找root+定义路径的静态文件，如果找不到会报404

## add\_after\_body



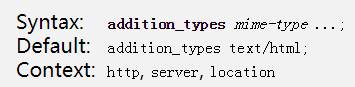
Adds the text returned as a result of processing a given subrequest after the response body. An empty string ("") as a parameter cancels addition inherited from the previous configuration level

## add\_before\_body



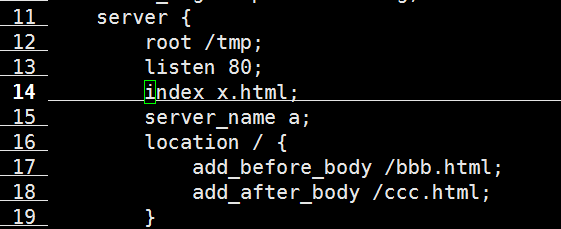
Adds the text returned as a result of processing a given subrequest before the response body. An empty string ("") as a parameter cancels addition inherited from the previous configuration level.

## addition\_types



允许使用添加响应消息的MIME类型

值为”\*”的话，就是所有的都可以使用添加响应



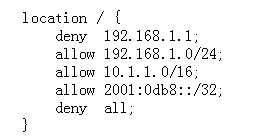
因为index.html，尾缀是html，所以可以使用add。如果index为index，则不可以使用add

# ngx\_http\_access\_module

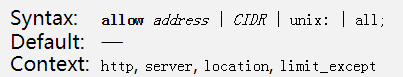
该模块允许限制某些IP地址的客户端访问

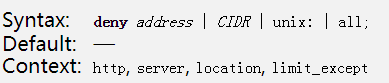
这个模块内置在nginx中。除非安装中使用了—with-http\_access\_module

也可以通过密码来限制，使用satisyf指令就可以同时通过IP地址和密码来限制



**按照顺序依次检查，直到匹配到第一条规则**。IPV4网络中只有10.1.1.0/16和192.168.1.0/24允许访问，但是192.168.1.1除外。很多情况下，ngx\_http\_geo\_module更适合





10.0.0.1-10.0.0.255为10.0.0.0/24

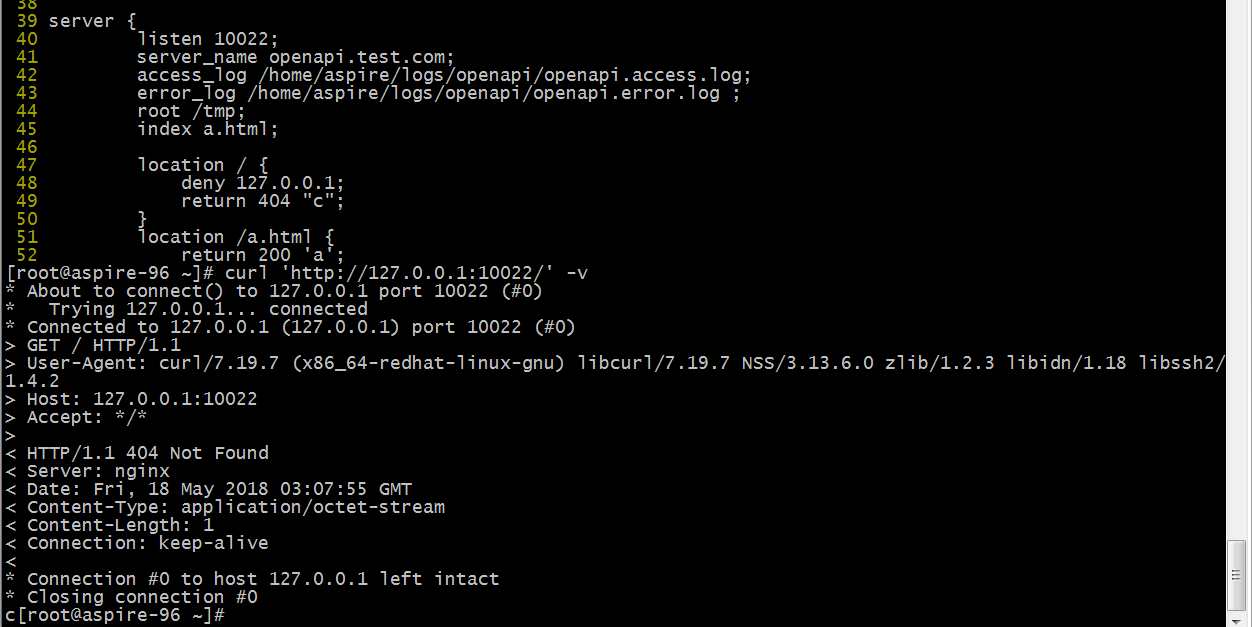
10.0.0.1-10.0.255.255为10.0.0.0/16

10.0.0.1-10.255.255.255为10.0.0.0/8

被deny调的ip，返回403

总结：

deny是可以禁调自己的本机的



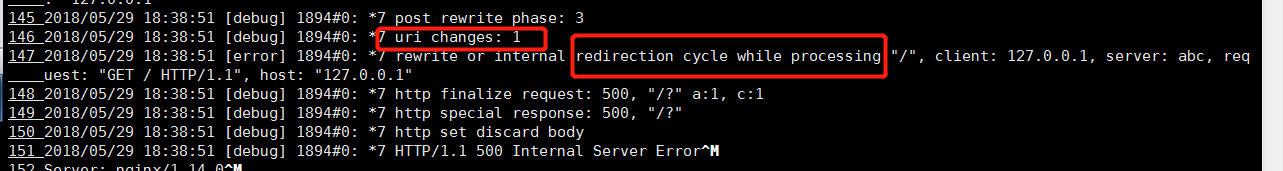
请求本来应该返回403，但是return 404，把http\_code给覆盖了

# ngx\_http\_rewrite\_module

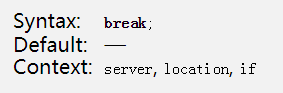
ngx\_http\_rewrite\_module模块允许正则替换uri，返回页面重定向

处理顺序：

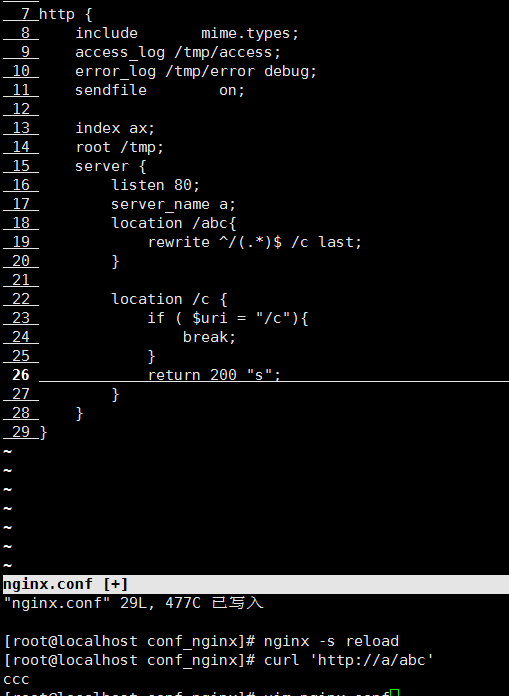
1. 处理在server级别中定义的模块指令
2. 为请求查找location
3. 处理在选中的location中定义模块指令，如果指令改变了URI，按新URI查找location，这个循环最多重复10次，之后nginx返回500



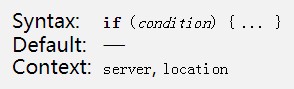
## break



停止处理当前这一轮的ngx\_http\_rewrite\_module指令集，并且使用root+改变后的uri的静态文件(在location里面，在location外面无效)



## if

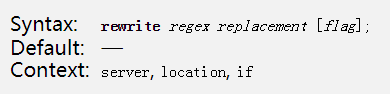


if和(之间有一个空格，运算符左右也是有空格的

condition可以是：

1. 变量名；如果变量值为空或0，则条件为假；（在1.0.1之前，变量以0开头会判断为false）
2. 使用“=”和“!=”运算符比较变量和字符串；
3. 使用“~”（大小写敏感）和“~\*”（大小写不敏感）运算符匹配变量和正则表达式。正则表达式可以包含匹配组，匹配结果后续可以使用变量$1..$9引用。如果正则表达式中包含字符“}”或者“;”，整个表达式应该被包含在单引号或双引号的引用中。
4. 使用“-f”和“!-f”运算符检查文件是否存在；
5. 使用“-d”和“!-d”运算符检查目录是否存在；
6. 使用“-e”和“!-e”运算符检查文件、目录或符号链接是否存在；
7. 使用“-x”和“!-x”运算符检查可执行文件；

## rewrite



如果指定的正则表达式能匹配URI，此URI将被replacement参数定义的字符串改写。rewrite指定按照配置的顺序执行。flag可以终止后续指令的执行，如果replacement以http、https开头，nginx将结束执行过程，并返回给客户端一个重定向

在location外面，last和break没区别

flag可以是其中的参数

1. last

停止执行当前这一轮的ngx\_http\_rewrite\_module，然后查找匹配后的uri的新location

1. break

停止执行当前这一轮的ngx\_http\_rewrite\_module

1. redirect

使用302状态码返回重定向

1. permanent

返回状态码为301的重定向

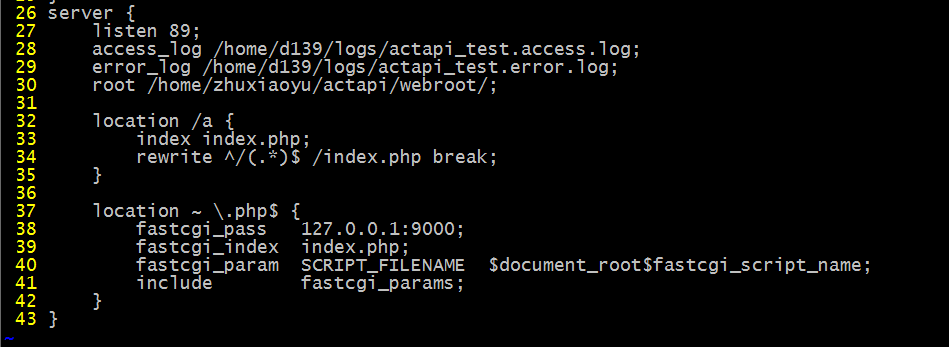
完整的重定向url将按照$scheme、server\_name\_in\_redirect和port\_int\_redirect指令的配置进行补全

如果replacement字符串包括新的请求参数，以往的请求参数会添加到新参数后面。如果不希望这样，在replacement字符串末尾加一个？

C:\Users\ZHUXIA~1\AppData\Local\Temp\1527770004(1).png

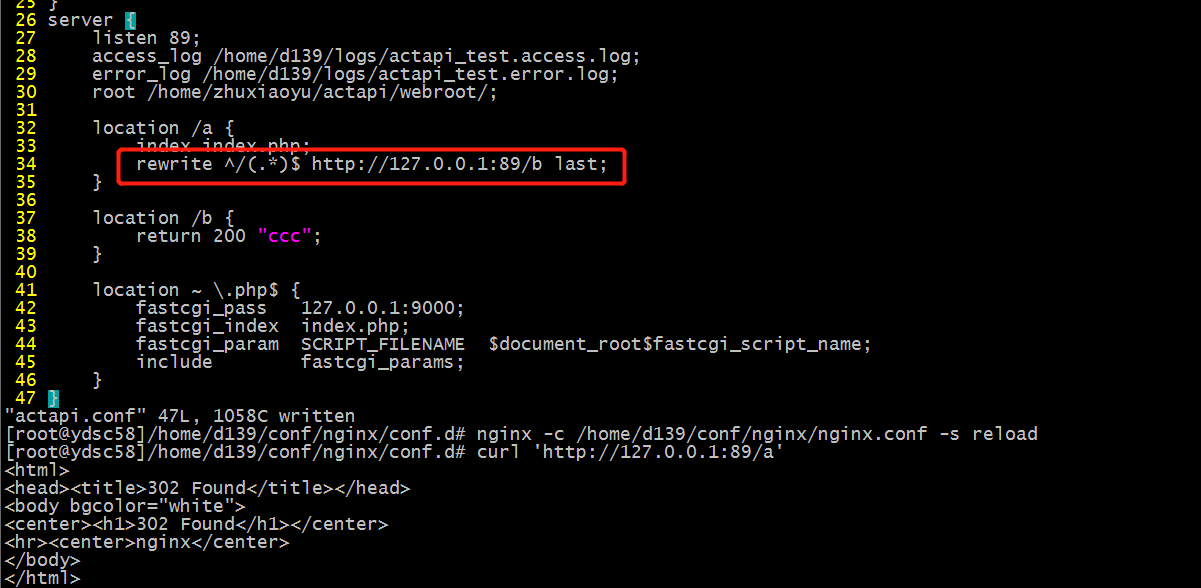
如果正则表达式中包括字符}或者;，整个表达式应该被包含在单引号或双引号中使用

总结

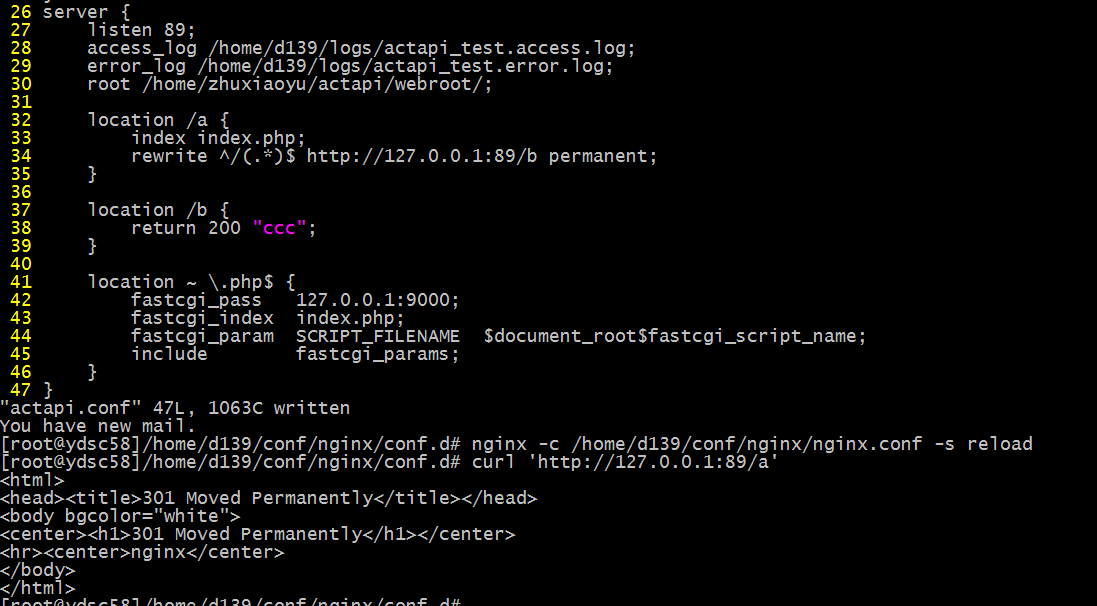


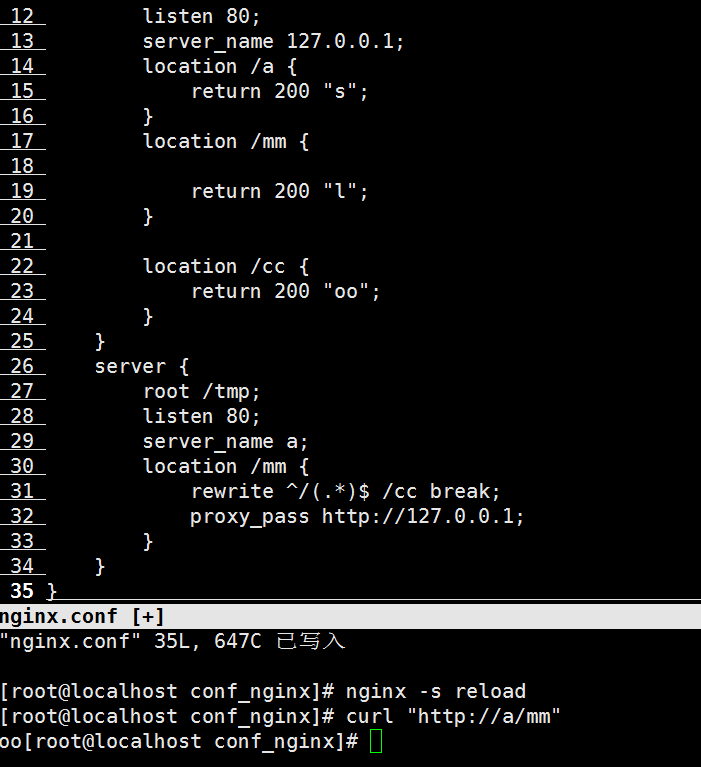
uri为/a，则rewrite到/index.php，由于是break，停止查找location，直接使用root+/index.php的静态文件

如果将break改为last，则会匹配到~ \.php$

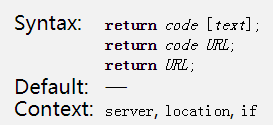


这样也是会发生302的





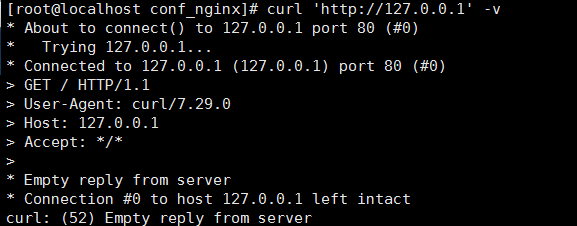
## return



停止处理并返回指定code给客户端，返回非标准状态码444可以直接关闭连接而不返回响应头

return 444 “a”;还是有响应头

return 444;没有响应头



可以在指令中指定重定向的URL（0.8.42版本之后），状态码为301、302、303和307，或者指定响应体文本

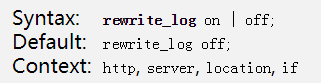
return <http://uri> 响应码为302

0.7.51以前只能返回204、400、402-406、408、410、411、413、416和500-504

直到1.1.16和1.0.13版本，307才被认为是一种重定向

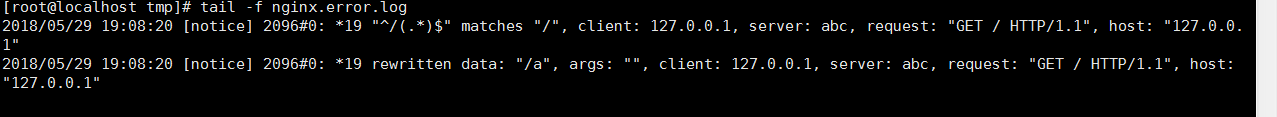
1.13.0可以返回308

## rewrite\_log

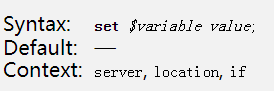


开启或者关闭将ngx\_http\_rewrite\_module模块指令的处理日志**以notice级别**记录到日志中

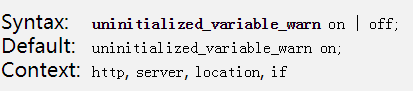
需要error\_log支持notice级别日志记录



## set

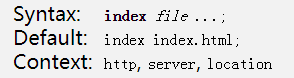


## uninitialized\_variable\_warn



是否记录未初始化的变量错误，warn

# ngx\_http\_index\_module



ngx\_http\_index\_module模块处理以斜杠字符为结尾的请求，这样的请求同样也能被ngx\_http\_autoindex\_module和ngx\_http\_random\_index\_module模块处理

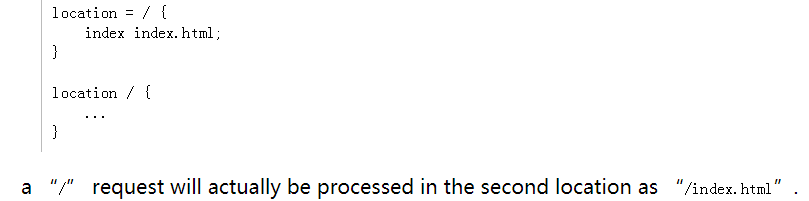
**只能处理以斜杠结尾的请求**

**如果root /tmp，uri为/t，不能被location匹配，则open index /tmp/t；如果uri为/t/，则open index /tmp/t/index.html**

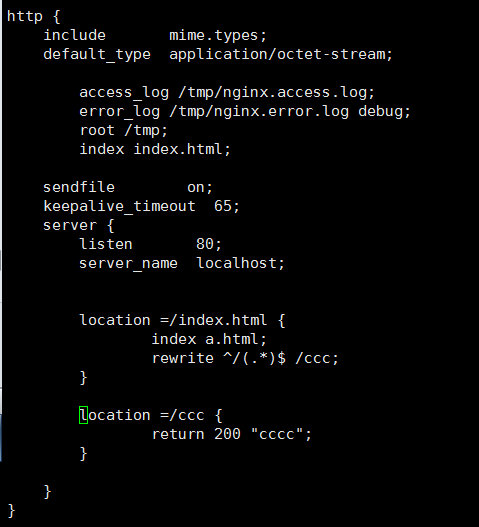
定义将要被作为默认页的文件，文件file名字可以包含变量。文件以配置中指定的顺序被nginx检查，index列表的最后一个元素可以是一个带有绝对路径的文件

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

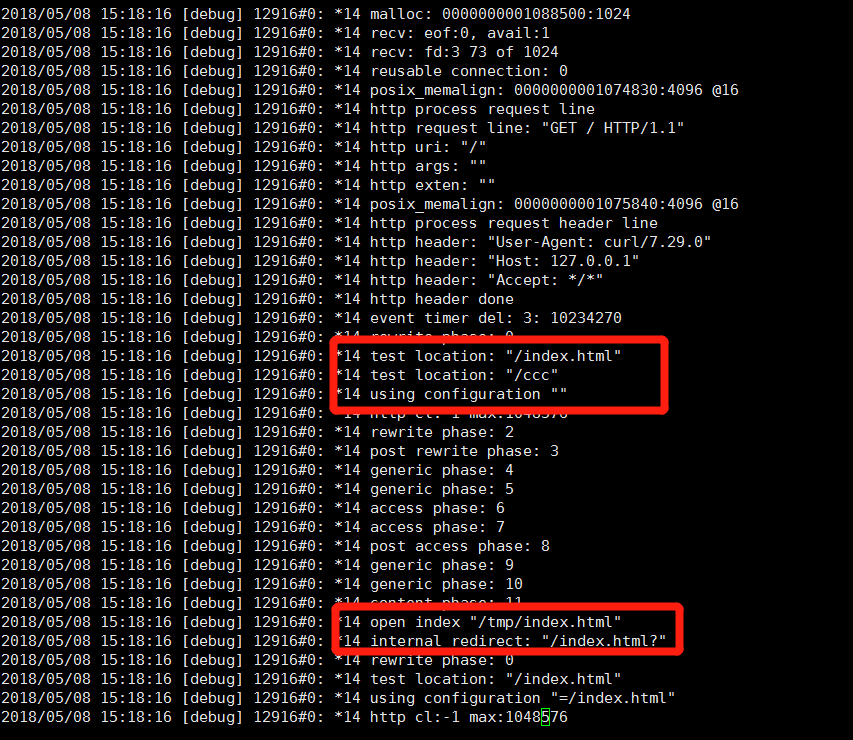
需要注意的是，index文件会引发内部重定向，请求可能会被其他location处理



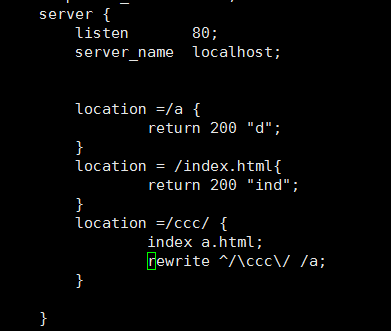
总结



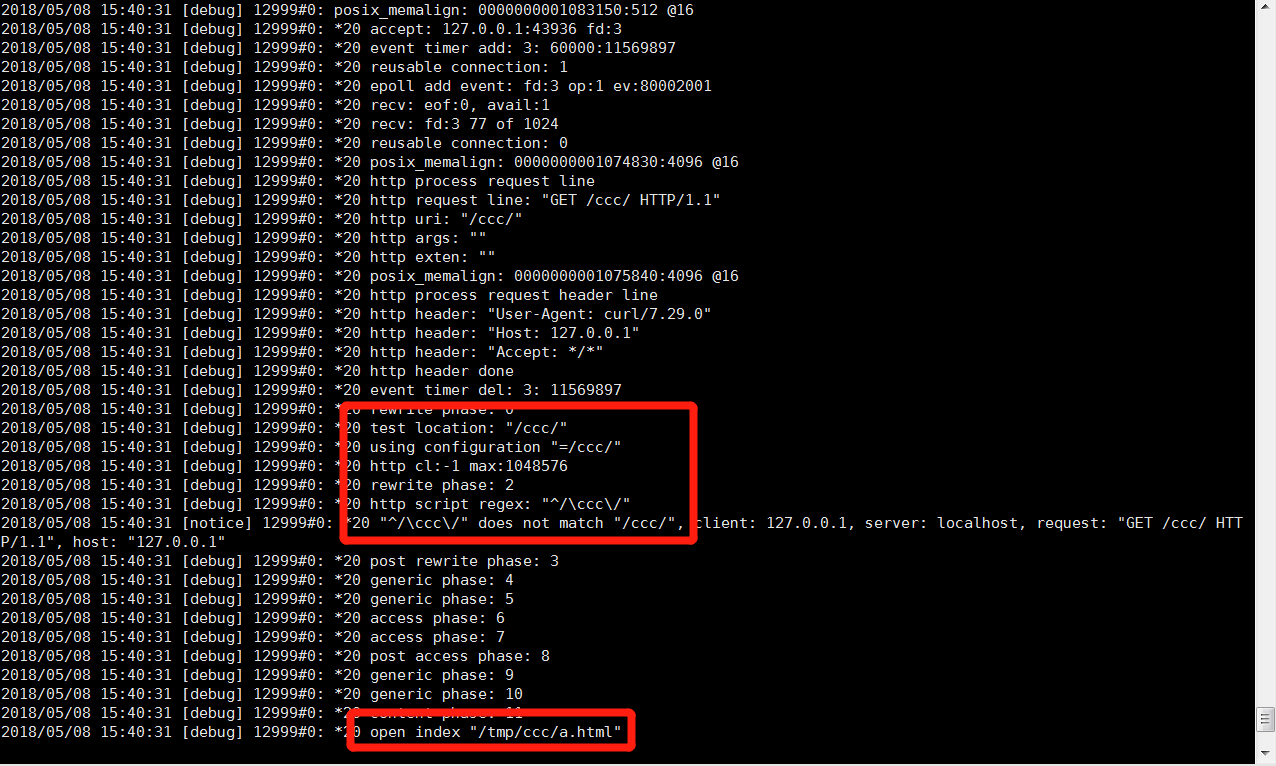
请求curl ‘http://127.0.0.1’



请求uri不能被所有location匹配，所以将uri后面拼上index的值，走默认处理，发生内部重定向，然后被location =/index.html匹配



curl ‘http://127.0.0.1/ccc/’，

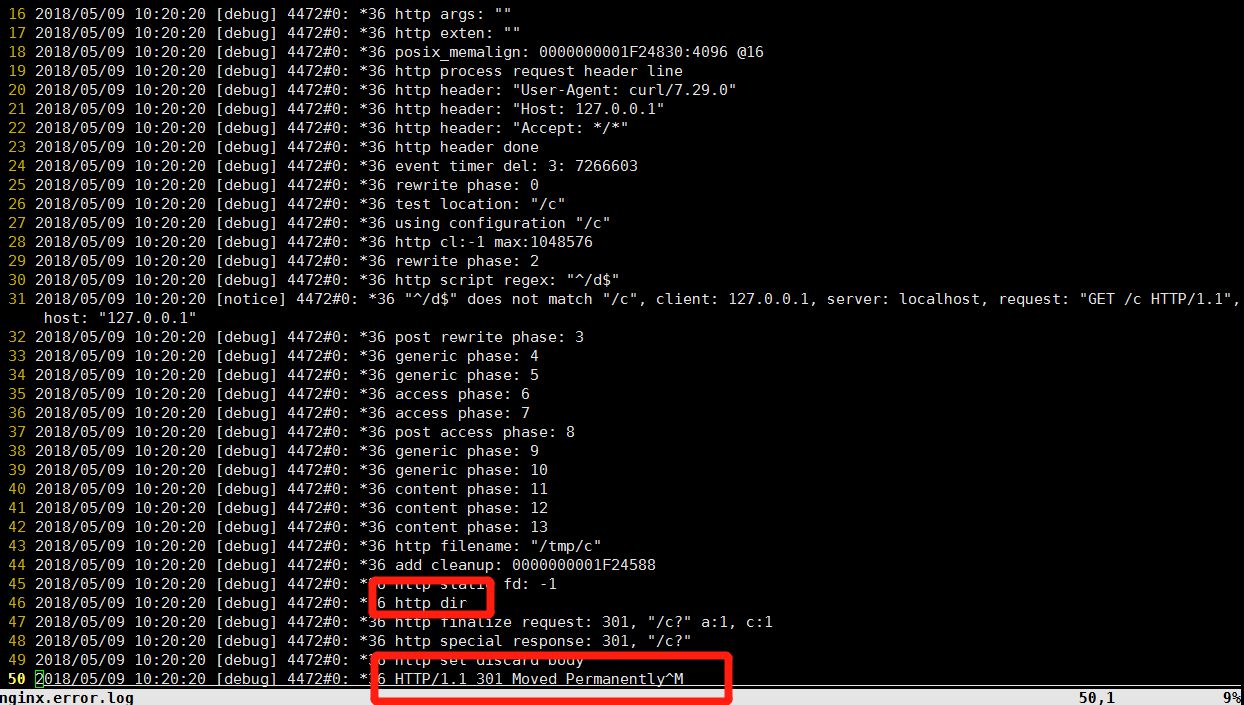


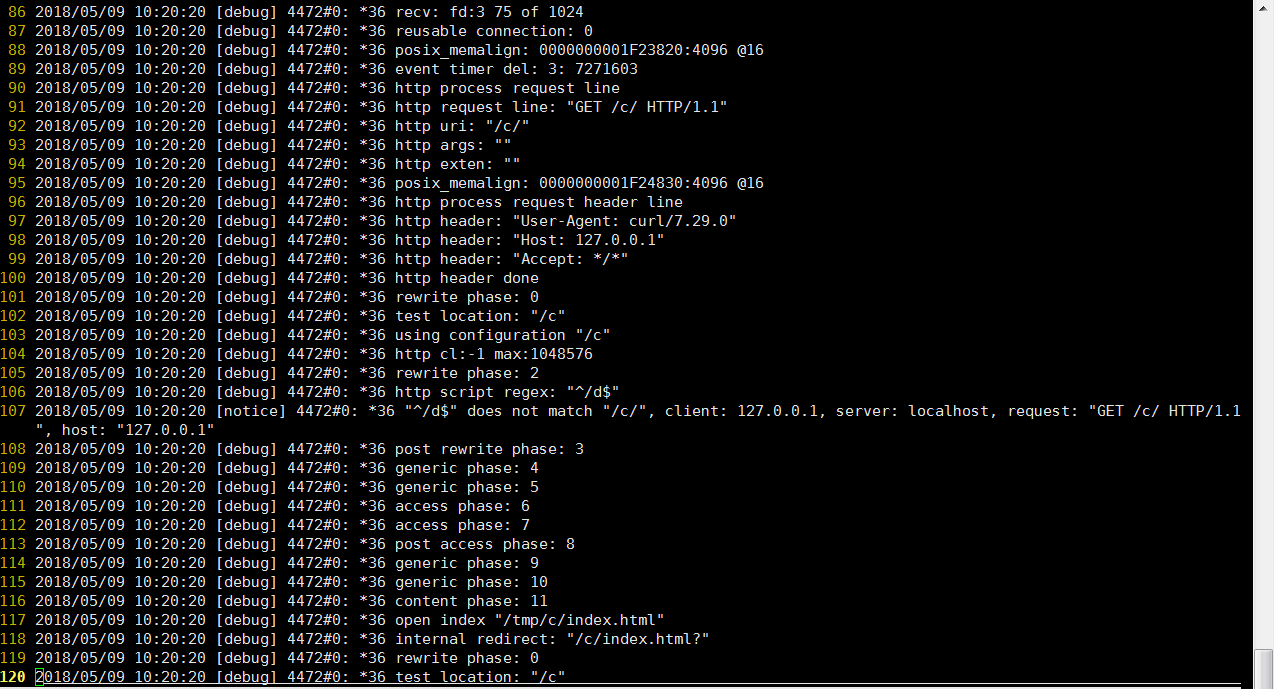
请求进location = /ccc/模块，然后走rewrite，正则处理的还是/ccc/，此时正则无法匹配，走index a.html，内部重定向uri/ccc/a.html，无法被匹配就打开/tmp/ccc/a.html；如果正则能被匹配，则index默认失效

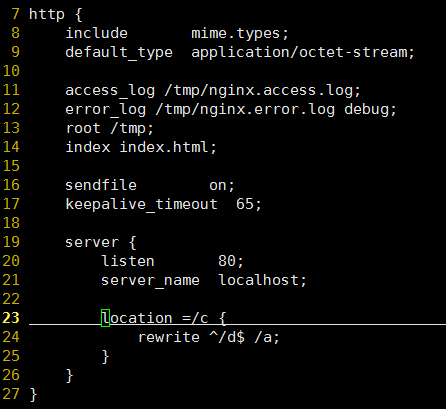


curl ‘http://127.0.0.1/c’

会被location /c匹配，然后rewrite失效，因为uri不以/结尾，所以index没用，此时/tmp/c如果是文件，则输出文件内容；如果c是目录，需要curl –L重定向，则会发生重定向，uri为/c/，又被location /c匹配，rewrite再次失效，走index，open /tmp/c/index



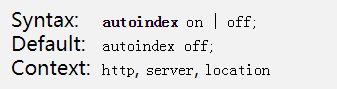




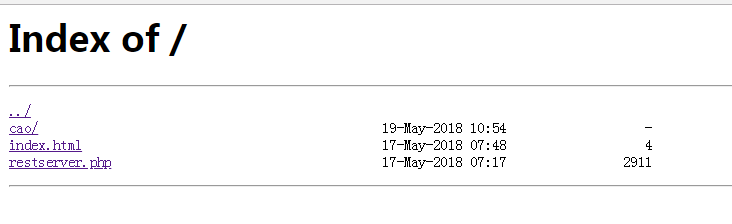
uri为/c，curl –L，rewrite不能匹配，open /tmp/c，发现是目录，返回重定向。自动重定向后uri为/c/，open /tmp/c/index.html

# ngx\_http\_autoindex\_module

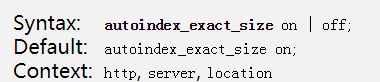
可以列出目录中的文件，**当ngx\_http\_index\_module模块找不到默认主页的时候**，会把请求转给ngx\_http\_autoindex\_module



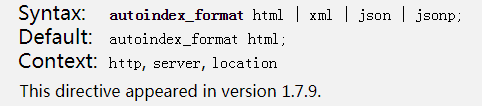
开启或者关闭列出目录中文件的功能



可以点，点击index.html就返回index.html内容；点击restserver.php，uri会变成/restserver.php，执行一次请求，如果没有php-fpm加载php，则直接回下载这个文件(能不能下载和这个文件的权限有关，如果不能下载则返回403)

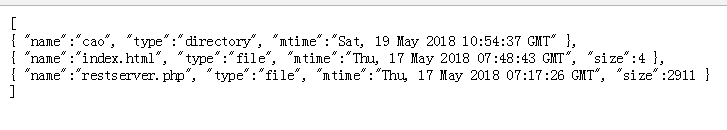


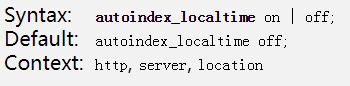
设置目录中列出的文件是显示精确大小，还是对KB、MB、GB进行四舍五入



设置目录列表的显示格式

xml输出将被ngx\_http\_xslt\_module转化



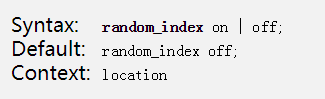


设置目录列出文件的时间是本地时间还是UTC时间

# ngx\_http\_random\_index\_module

在文件夹中随机选择一个文件作为默认页(uri为/结尾)，使用此模块需要先配置ngx\_http\_index\_modure

需要开启—with-http\_random\_index\_module



# ngx\_http\_log\_module

该模块按指定的格式写访问日志

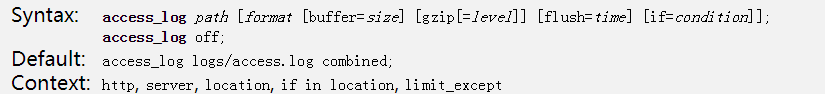
请求在处理结束时，会按请求路径的配置记录访问日志，如果请求处理期间产生了内部跳转，请求结束时的路径可能不同于原始的请求路径

如果请求location a；在location a里面发生内部跳转到location b;在location

b里面重新定义access\_log或者error\_log等，则location a里面的不会生效。

如果在location a里面proxy pass，则a和b的都会记录，并且a的http\_code是b给返回的

## access\_log



为访问日志设置路径、格式和缓冲区大小（nginx访问日志支持缓存）。在同一个配置层级里可以指定多个日志，特定值off会取消当前配置层级里面的所有access\_log。如果没有指定日志格式则会使用预定义的combined

如果buffer或者gzip（1.3.10、1.2.7）被使用，写操作会被缓存

buffer的大小不能超过磁盘原子性写入大小。FreeBSD系统无限制

如果缓存被使用，以下会发生实际写入文件的操作：

1. 下一个日志没有被缓存
2. if the buffered data is older than specified by the flush parameter (1.3.10, 1.2.7)
3. worker process正在打开日志文件或者正在关闭

如果gzip参数被使用，则buffer数据会先被压缩，然后再写入文件。参数可以指定为1-9,1为最快，9最慢但是压缩比最大。默认情况下，buffer为64K，gizp为1。由于日志数据被压缩，所以需要被解压或者使用zcat

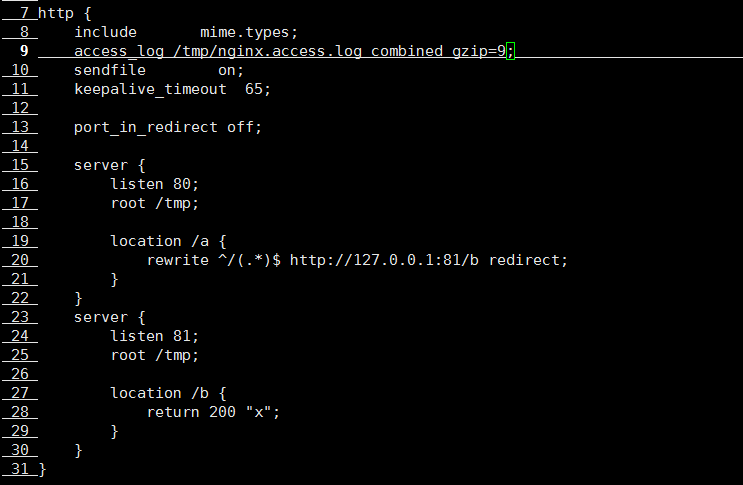
为了支持gzip功能，系统必须有zlib library模块

如果access\_log的路径值使用变量，则不能明着写buffer gzip flush，否则会报错

日志文件的路径可以包含变量(0.7.6+)，这样有一些限制

1. 工作进程使用的user应具有在目录里创建文件的权限
2. 写缓冲无效，不能指定gzip、flush、buffer
3. 每条日志写入都会打开和关闭文件，然而，频繁使用的文件描述符可以存储在[缓存](http://tengine.taobao.org/nginx_docs/cn/docs/http/ngx_http_log_module.html#open_log_file_cache)中， 在[open\_log\_file\_cache](http://tengine.taobao.org/nginx_docs/cn/docs/http/ngx_http_log_module.html#open_log_file_cache)指令的valid参数指定的时间里， 写操作能持续写到旧文件
4. 每次日志写入的操作都会检查请求的 [根目录](http://tengine.taobao.org/nginx_docs/cn/docs/http/ngx_http_core_module.html#root)是否存在， 如果不存在则日志不会被创建。 因此在一个层级里同时指定[root](http://tengine.taobao.org/nginx_docs/cn/docs/http/ngx_http_core_module.html#root) 和access\_log是一个不错的想法

总结

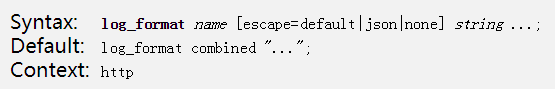


访问uri为b，不会马上记录日志，因为gzip开启，会有缓存;如果在缓存期间kill掉nginx，则会有日志

如果将nginx日志删掉，启动nginx和reload会新建这个日志文件；如果在运行期间删除nginx文件，则新请求不会创建nginx日志文件;日志路径包含变量的话，会在运行期间创建文件

如果指定了缓存或者gzip，还指定了flush=1m，则在缓存不超的情况下，会在1分钟后写入日志

## log\_format



定义日志的格式

$bytes\_sent

发送给客户端的字节数

$connection

连接序号

$connection\_requests

当前通过一个连接获得的请求数量

$msec

日志写入时间，毫秒

$pipe

如果请求是pipelined则为p，否则为.

$request\_length

请求的长度，包括请求行、头和请求正文

$request\_time

请求处理时间，为秒，精度为毫秒；从读入客户端第一个字节开始，直到把最后一个字符发送给客户端后写入日志位置

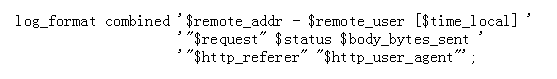
$status

状态码

$time\_local

本地时间，通用日志格式

发送给客户端的响应头有sent\_http\_前缀



要先定义log\_format，再定义access\_log

## 日志切割

1.mv access.log access.log.date

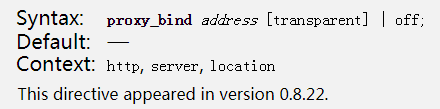
2.kill –USR1 ${cat /usr/local/logs/nginx.pid}

在有buffer情况下，如果还没有buffer出数据就mv了，则-USR1会马上刷新buffer的数据进access.log.date

# ngx\_http\_proxy\_module

该模块允许将请求转发给其他服务器

## proxy\_bind



使代理服务器的连接来自指定的本地IP和指定的端口(1.11.2)，参数可以包含变量(1.3.12)

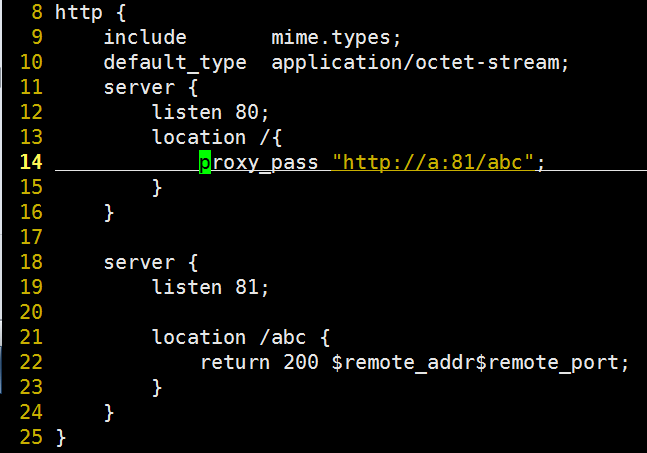
特殊值off(1.3.12)将取消proxy\_bind指定前一个级别继承的影响，这将允许系统自动分配本地IP和端口

值transparent(1.11.0)参数可以允许一个非本地IP连接代理服务器（需要配置路由表），比如

proxy\_bind $remote\_addr transparent;

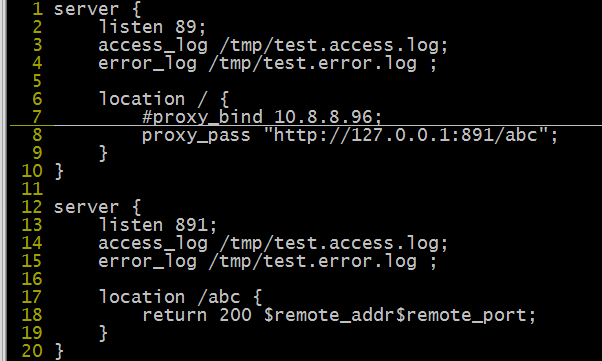
In order for this parameter to work, it is usually necessary to run nginx worker processes with the superuser privileges. On Linux it is not required (1.13.8) as if the transparent parameter is specified, worker processes inherit the CAP\_NET\_RAW capability from the master process. It is also necessary to configure kernel routing table to intercept network traffic from the proxied server.

总结：



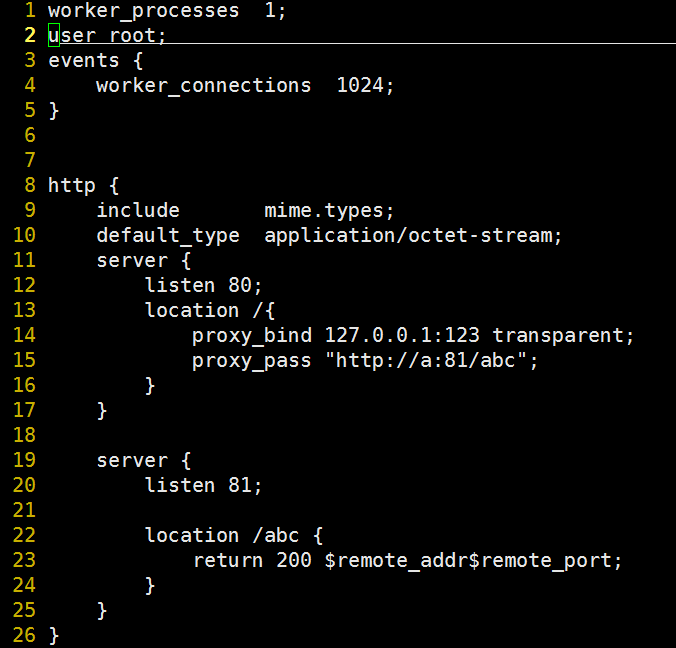
请求 curl ‘http://127.0.0.1’，输出127.0.0.1端口随机

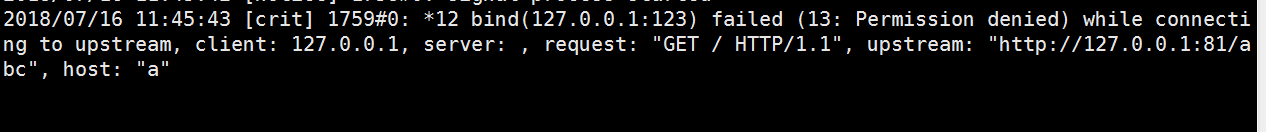
请求 curl ‘http://127.0.0.2’，输出127.0.0.1端口随机



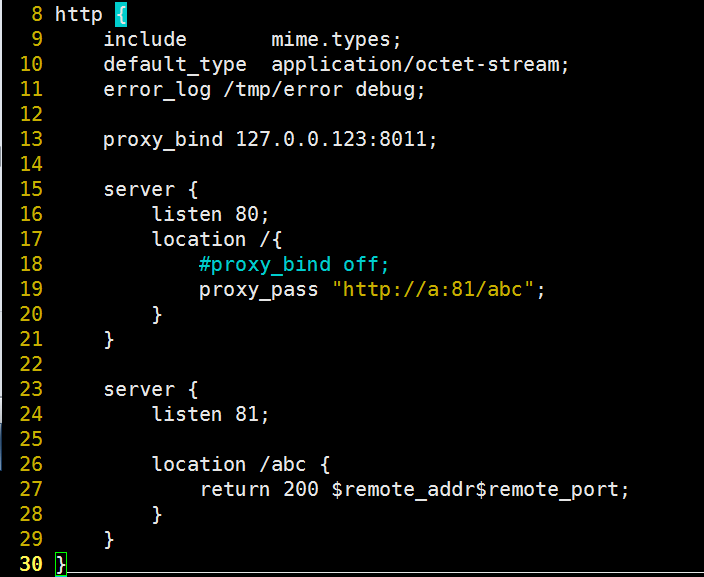
以上配置为10.8.8.57，如果用10.8.8.96请求89端口，则输出127.0.0.1端口随机

nginx:worker process需要超级管理员权限启动，如果bind的端口小于8000的话



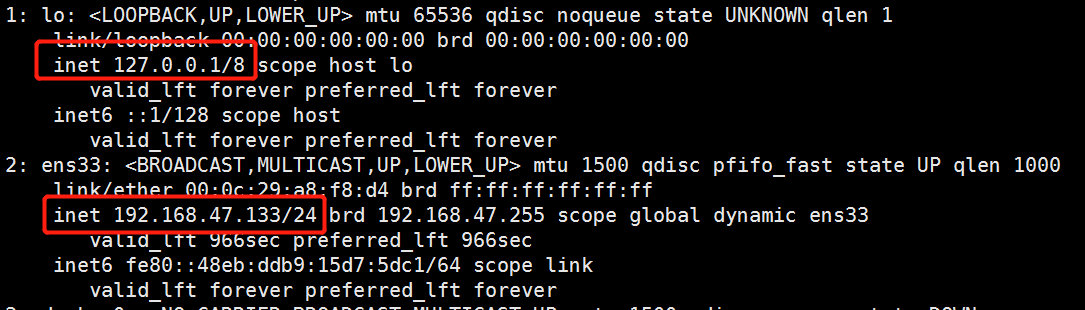


off参数

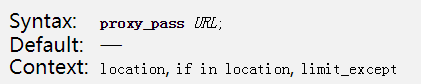


访问的话输出127.0.0.123:8011，如果将proxy\_bind off注释打开，则输出127.0.0.1随机端口

bind可以绑定ip addr中的地址



## proxy\_pass

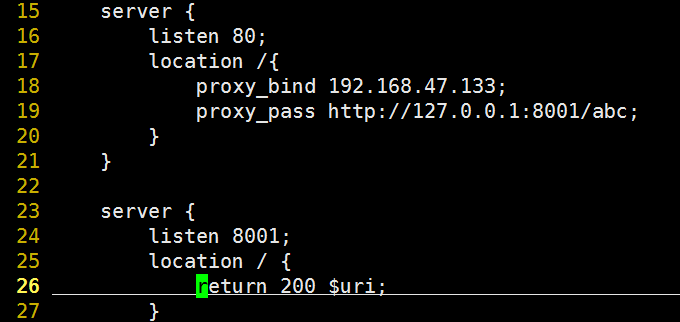


设置一个代理协议和地址

协议可以是http或者https，地址可以带端口和uri

注意：

pass的uri：



请求80，uri为/，则输出abc

uri为/aaa，输出abcaaa

get请求，输出get；post请求输出post

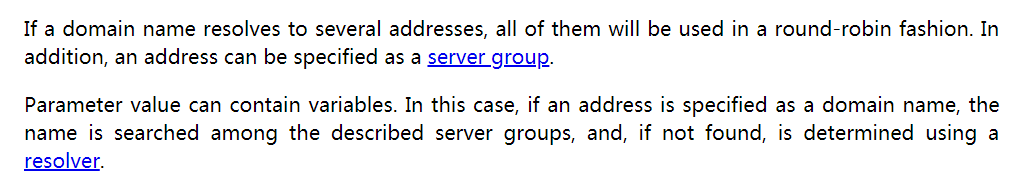
C:\Users\ZHUXIA~1\AppData\Local\Temp\1528771401(1).png

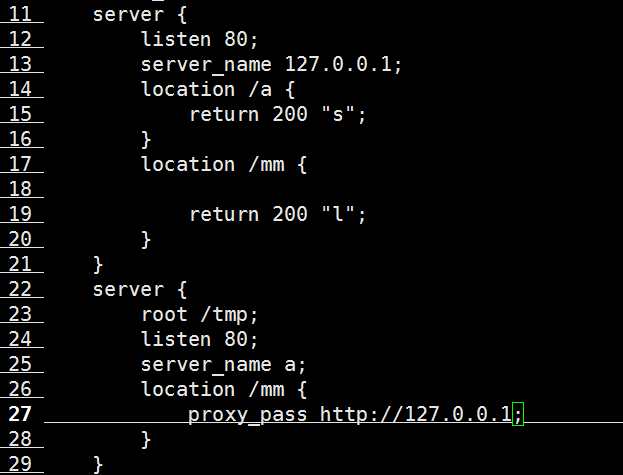
或者是

C:\Users\ZHUXIA~1\AppData\Local\Temp\1528771438(1).png

如果proxy\_pass使用了URI，当传送请求到后端服务器时，URI将被规范化

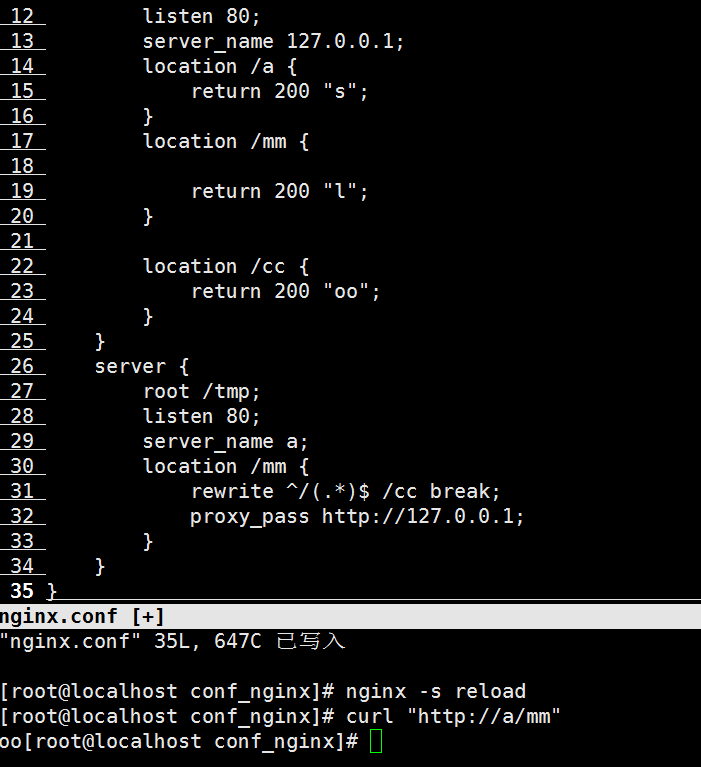
如果proxy\_pass没有使用URI，传送到后端服务器的URI是原始请求的URI



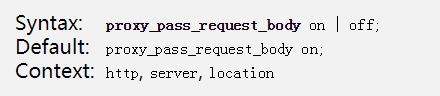


请求<http://a/mm> ，会被代理到<http://127.0.0.1/mm>，如果proxy\_pass <http://127.0.0.1/，则代理的uri为/>

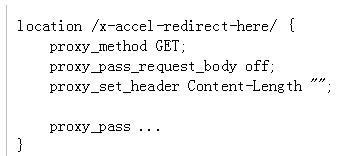
使用正则表达式定义路径，无法确定请求URI中应该被替代的部分



## proxy\_pass\_request\_body

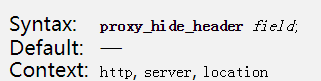


决定是否传送原始body给代理服务器



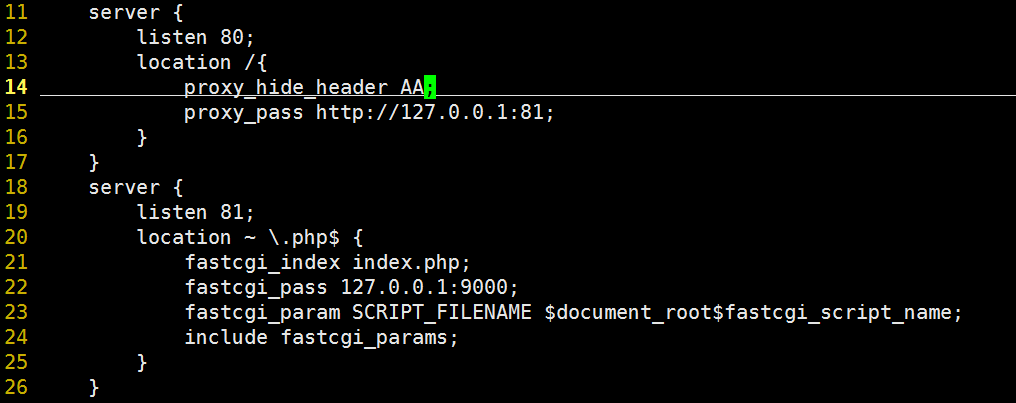
单独proxy\_pass\_request\_body off；php还是有body体，需要proxy\_set\_header Content-Length “”，proxy\_method GET

## proxy\_hide\_header

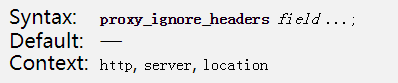


默认情况下，nginx不会传递Date，Server，X-Pad，X-Accel- 从代理服务器响应给客户端。proxy\_hide\_header指令设置额外的不传递字段

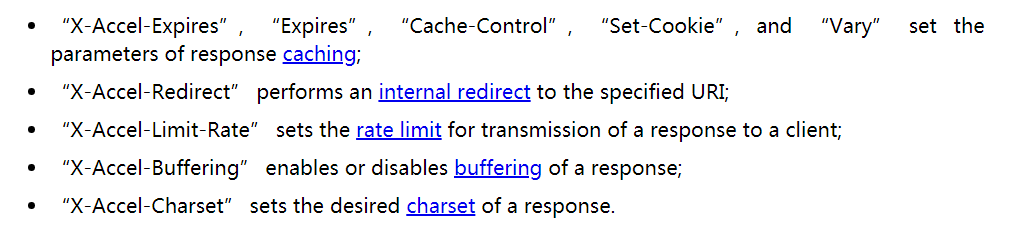
php指定header（”AA: 123”），以下配置不会响应给客户端AA头



## proxy\_ignore\_headers

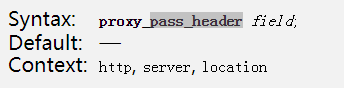


不处理代理服务器传过来的指定响应头，可以被设置为X-Accel-Redirect”, “X-Accel-Expires”, “X-Accel-Limit-Rate” (1.1.6), “X-Accel-Buffering” (1.1.6), “X-Accel-Charset” (1.1.6), “Expires”, “Cache-Control”, “Set-Cookie” (0.8.44), and “Vary” (1.7.7)



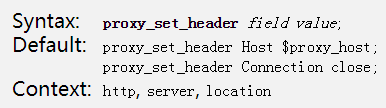
不清楚啥意思

## proxy\_pass\_header



允许传递被禁用的响应头给客户端

## proxy\_set\_header



允许重新定义或者追加header头到proxy服务器，可以包含文本、变量和他们的组合

指令会继承于上一个级别，如果当前级别没有proxy\_set\_header被定义

如果location A proxy给location B，父级有proxy\_set\_header，如果想不使用父级的，则要在AB分别都写proxy\_set\_header

### 关于获取代理服务器的真实IP

如果client C-> proxy A -> proxy B->server S，则获取的HTTP\_X\_FORWARDED\_FOR为C，A；S的remote\_addr为B

如果client C-> proxy A->server S，则获取的HTTP\_X\_FORWARDED\_FOR为C；S的remote\_addr为A

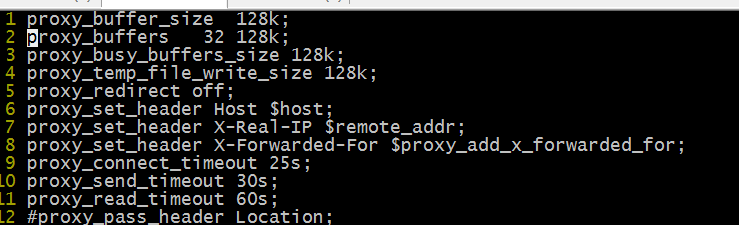
但是获取的HTTP\_X\_FORWARDED\_FOR需要指定

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

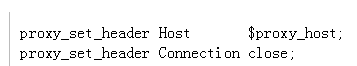
如果有HTTP\_X\_REAL\_IP，无论经过了多少层proxy，都是client的

获取HTTP\_X\_REAL\_IP需要指定

proxy\_set\_header X-REAL-IP $remote\_addr;



默认情况下会有两个默认的



**如果caching被使用，从原始服务器If-Modified-Since，If-Unmodified-Since，If-None-Match，If-Match，Range，If-Range的header头不会被传递给proxy服务器**

如果不想改变host，可以

proxy\_set\_header Host $http\_host;

然而，如果原始请求没有host，那么host不会给代理服务器发，最好使用$host，如果有请求Host字段，那么就使用请求的Host，如果没有则使用server\_name

proxy\_set\_header Host $host;

服务器名可以和后端服务器的端口一起传送

proxy\_set\_header Host $host:$proxy\_port;

如果某个请求头尾空，那么这个请求头不会传给代理服务器



curl ‘http://a/mm’，因为改变了代理host，所以代理后找server\_name a的location cc，找不到则找静态文件，找不到报404

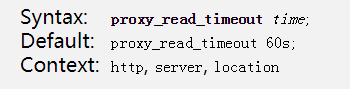
## proxy\_connect\_timeout



设置于代理服务器的连接超时时间

这个时间不能超过75秒

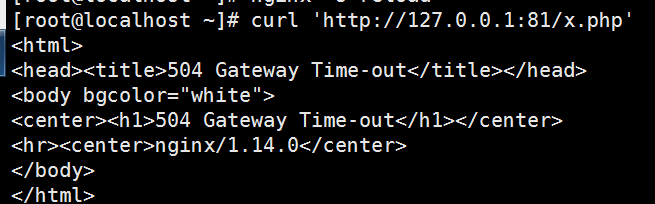
## proxy\_read\_timeout



Defines a timeout for reading a response from the proxied server. The timeout is set only between two successive read operations, not for the transmission of the whole response. If the proxied server does not transmit anything within this time, the connection is closed

定义一个从代理服务器的超时时间

就是两次读操作的成功间隔时间，如果php中第一次echo 1，第二次sleep 20，第三次echo 2；然后proxy\_read\_timeout 10s



## proxy\_send\_timeout



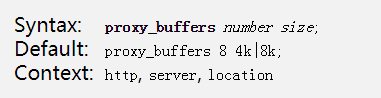
设置传输请求给代理服务器的超时时间，这个超时时间是相邻两次成功写操作的间隔时间，不是整个传输的时间。如果代理服务器没有接受任何数据，则连接关闭

## proxy\_buffer\_size



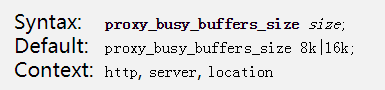
Sets the size of the buffer used for reading the first part of the response received from the proxied server. This part usually contains a small response header. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform. It can be made smaller, however

## proxy\_buffers



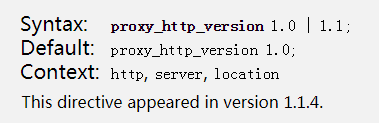
Sets the number and size of the buffers used for reading a response from the proxied server, for a single connection. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform

## proxy\_busy\_buffers\_size



当开启缓冲响应功能以后，在没有读取到全部响应情况下，写缓冲达到一定的大小时候，nginx会向客户端发一次响应，直到缓冲小于此值。剩余的缓冲区可以用于接受响应，如果需要，一部分内容将缓冲到临时文件。该大小默认是proxy\_buffer\_size和proxy\_buffers指令设置单块缓冲大小的两倍

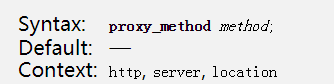
## proxy\_http\_version



设置代理HTTP协议版本，默认是1.0

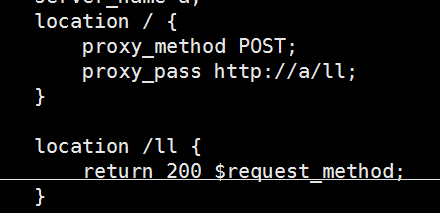
Version 1.1 is recommended(推荐) for use with [keepalive](http://nginx.org/en/docs/http/ngx_http_upstream_module.html#keepalive) connections and NTLM anthentication

## proxy\_method

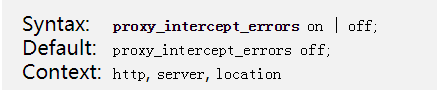


Specifies(指定，提出…的条件，使具有特性) the HTTP method to use in requests forwarded to(转发到) the proxied server instead of the method from the client request. Parameter value can contain variables (1.11.6)

指定代理方法而不是使用客户端请求的方法

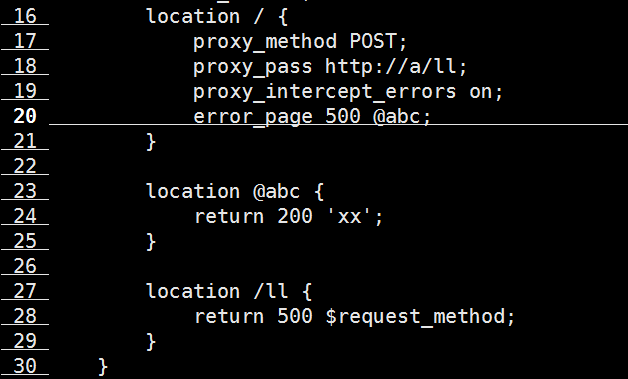


## proxy\_intercept\_errors



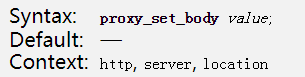
Determines whether proxied responses with codes greater than or equal to 300 should be passed to a client or be intercepted(拦截) and redirected to nginx for processing with the [error\_page](http://nginx.org/en/docs/http/ngx_http_core_module.html#error_page) directive

决定是否使用error\_page指令拦截或直接将响应码大于或等于300的响应直接传给客户端

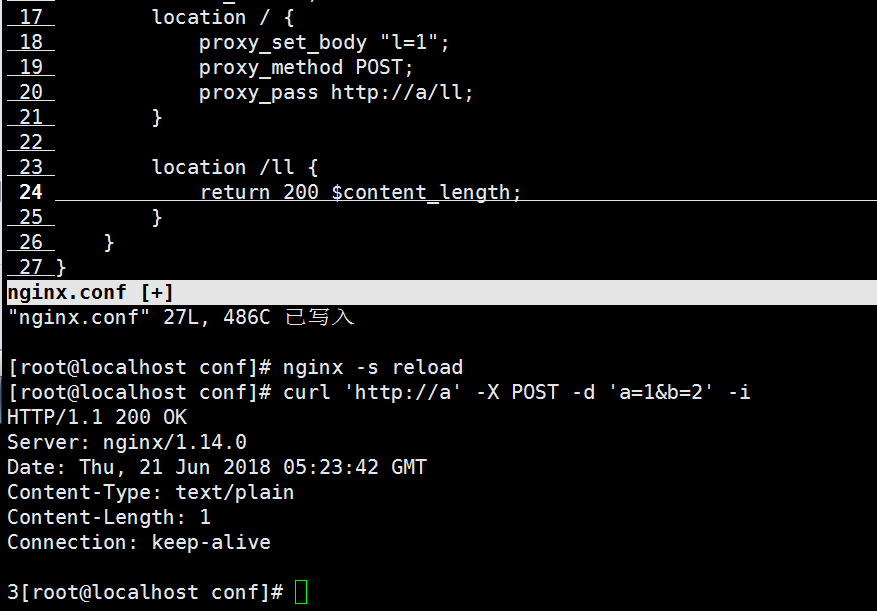


curl ‘http://a’ 返回xx，如果proxy\_intercept\_errors off，则返回POST

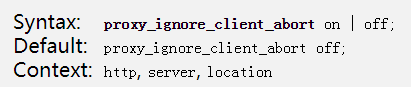
## proxy\_set\_body



Allows redefining(重新定义) the request body passed to the proxied server. The value can contain text, variables, and their combination



## proxy\_igonre\_client\_abort



决定nginx是否在客户端关闭连接的时候关闭与代理服务器的连接，without waiting for a response

Determines whether the connection with a proxied server should be closed when a client closes the connection without waiting for a response

## proxy\_redirect

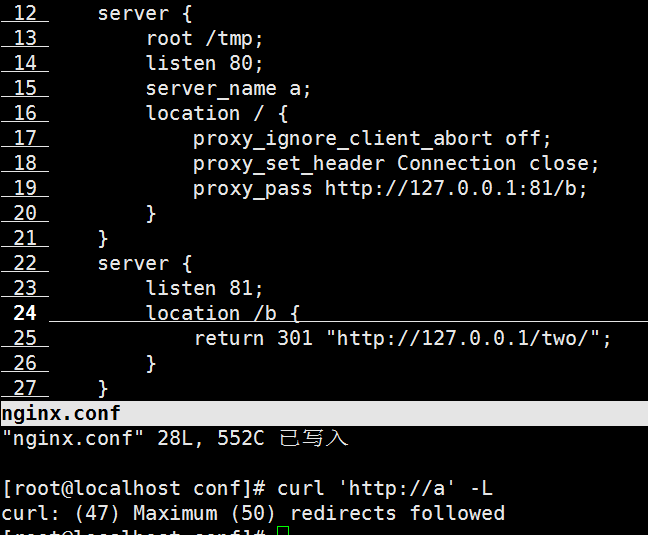


设置一个文本用来改变从后端服务器发过来的响应头Location和Refresh

假设代理服务器返回一个Location头，Location:http://localhost:8000/two/some/uri

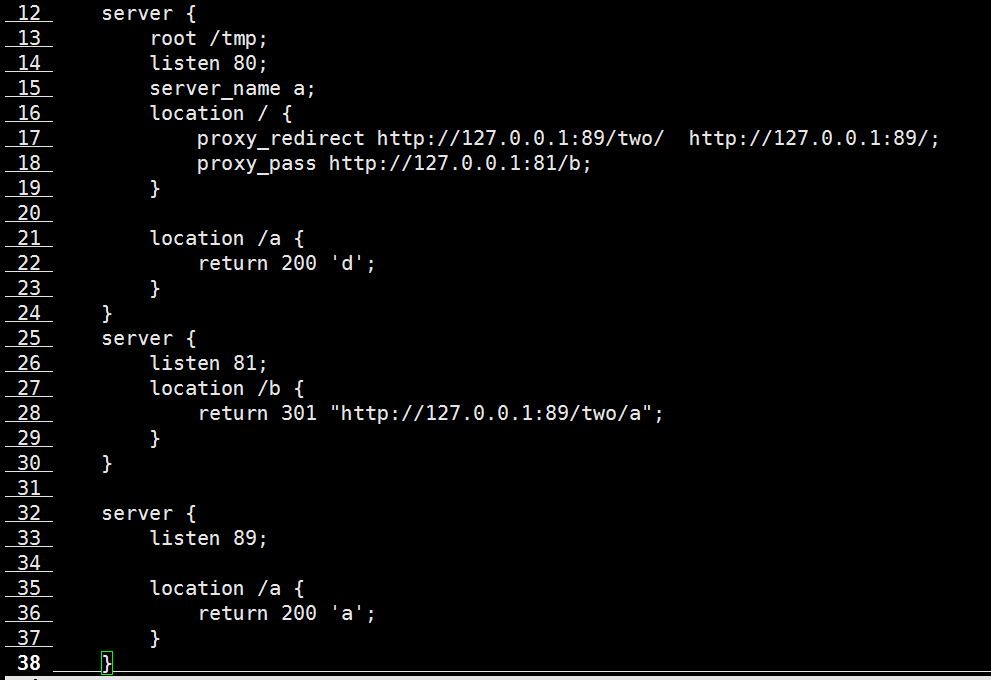
Suppose(假设) a proxied server returned the header field Location….

proxy\_redirect <http://location:8000/two/> <http://frontend/one/>;将会将Location重写为Location: <http://frontend/one/some/uri/，replacement>可以省略服务器名，此时将使用代理服务器的主域名和端口号





请求<http://a>，发生proxy\_pass，代理到listen 81;location /b，由于proxy\_redirect <http://127.0.0.1:89/two/将301>地址替换为/a，使用listen 80的端口和server\_name，匹配到location /a，返回d

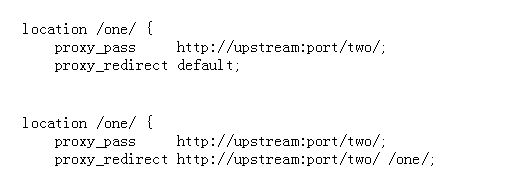


请求<http://a，proxy_redirect>将301rewrite为<http://127.0.0.1:89/a>

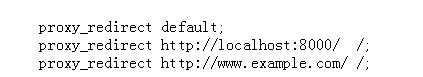


请求<http://a/x，发生301>到<http://127.0.0.1:81/b/a>，因为proxy\_redirect default，default就相当于proxy\_redirext proxy\_pass的值 location的值，这个配置中就相当于proxy\_redirect <http://127.0.0.1:81/b> /x;就相当于重定向到/x/a，返回d

以下配置等效



可以同时定义多个proxy\_redirect指令



off参数是所有同级别的配置无效



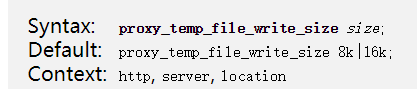


请求<http://a>，代理到端口81，发生301，因为proxy\_redirect / / ，相当于把<http://127.0.0.1:81/b/a>，替换为<http://127.0.0.1:81，发生50>次重定向报错

## proxy\_send\_lowat

指令在linux中无效

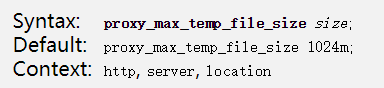
## Proxy\_temp\_file\_write\_size



Limits the size of data written to a temporary file at a time, when buffering of responses from the proxied server to temporary files is enabled. By default, size is limited by two buffers set by the[proxy\_buffer\_size](http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_buffer_size) and [proxy\_buffers](http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_buffers) directives. The maximum size of a temporary file is set by the [proxy\_max\_temp\_file\_size](http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_max_temp_file_size) directive

在开启缓冲后端服务器响应到临时文件的功能后，设置nginx每次写数据到临时文件的size(大小)限制。 size的默认值是[proxy\_buffer\_size](http://tengine.taobao.org/nginx_docs/cn/docs/http/ngx_http_proxy_module.html#proxy_buffer_size)指令和[proxy\_buffers](http://tengine.taobao.org/nginx_docs/cn/docs/http/ngx_http_proxy_module.html#proxy_buffers)指令定义的每块缓冲区大小的两倍， 而临时文件最大容量由[proxy\_max\_temp\_file\_size](http://tengine.taobao.org/nginx_docs/cn/docs/http/ngx_http_proxy_module.html#proxy_max_temp_file_size)指令设置

## Proxy\_max\_temp\_file\_size



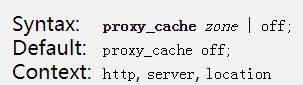
When [buffering](http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_buffering) of responses from the proxied server is enabled, and the whole response does not fit into the buffers set by the [proxy\_buffer\_size](http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_buffer_size) and [proxy\_buffers](http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_buffers) directives, a part of the response can be saved to a temporary file. This directive sets the maximum size of the temporary file. The size of data written to the temporary file at a time is set by the [proxy\_temp\_file\_write\_size](http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_temp_file_write_size) directive.

The zero value disables buffering of responses to temporary files

打开响应[缓冲](http://tengine.taobao.org/nginx_docs/cn/docs/http/ngx_http_proxy_module.html#proxy_buffering)以后，如果整个响应不能存放在[proxy\_buffer\_size](http://tengine.taobao.org/nginx_docs/cn/docs/http/ngx_http_proxy_module.html#proxy_buffer_size)和[proxy\_buffers](http://tengine.taobao.org/nginx_docs/cn/docs/http/ngx_http_proxy_module.html#proxy_buffers)指令设置的缓冲区内，部分响应可以存放在临时文件中。 这条指令可以设置临时文件的最大容量。而每次写入临时文件的数据量则由[proxy\_temp\_file\_write\_size](http://tengine.taobao.org/nginx_docs/cn/docs/http/ngx_http_proxy_module.html#proxy_temp_file_write_size)指令定义。

将此值设置为0将禁止响应写入临时文件

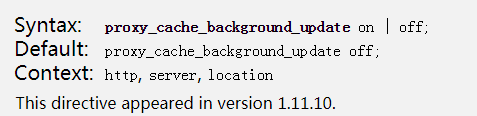
## proxy\_cache



定义一个页面缓存的共享区域，这个区域可以用于很多地方，off参数可以不从上层配置继承该指令

proxy\_cache cache\_www

## proxy\_cache\_background\_update



允许启动后台子请求来更新过期缓存，当过期的缓存返回给了客户端。应该注意的是，更新时，必须允许使用过期的缓存响应

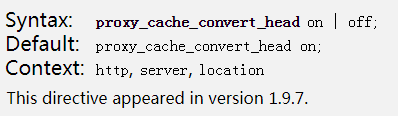
Allows starting a background subrequest(后台子请求) to update an expired(过期) cache item(项), while a stale(不新鲜的、陈旧的) cached response is returned to the client. Note that it is necessary to allow the usage of a stale cached response when it is being updated

## proxy\_cache\_bypass

定义不从缓冲中取出响应的条件，如果至少一个字符串条件非空而且非0，nginx就不会从缓冲中去取响应

C:\Users\ZHUXIA~1\AppData\Local\Temp\1529997064(1).png

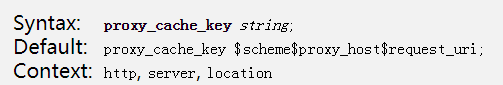
## proxy\_cache\_convert\_head



开启或者禁用将HEAD方法转变为GET用于caching，当off时候，cache\_key应该包含$request\_method

Enables or disables the conversion(转换) of the “HEAD” method to “GET” for caching. When the conversion is disabled, the [cache key](http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_cache_key) should be configured to include the $request\_method

## proxy\_cache\_key

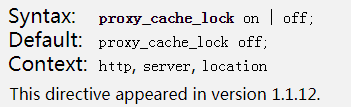


定义一个cache键

键名是proxy\_cache\_path，keys\_zone的值

proxy\_cache\_path /tmp/cache/test1 levels=1:2 keys\_zone=test1:1m;，可以proxy\_cache\_key test1;

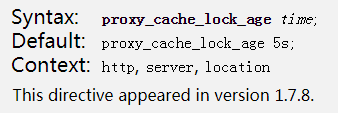
## proxy\_cache\_lock



开启此功能时，对于相同的请求，同时只允许一个请求发往后端，并根据[proxy\_cache\_key](http://tengine.taobao.org/nginx_docs/cn/docs/http/ngx_http_proxy_module.html#proxy_cache_key)指令的设置在缓存中植入一个新条目。 其他请求相同条目的请求将一直等待，直到缓存中出现相应的内容，或者锁在[proxy\_cache\_lock\_timeout](http://tengine.taobao.org/nginx_docs/cn/docs/http/ngx_http_proxy_module.html#proxy_cache_lock_timeout)指令设置的超时后被释放

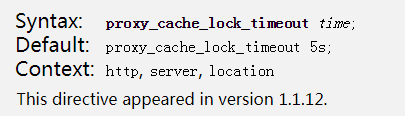
When enabled, only one request at a time will be allowed to populate(居住于、生活于) a new cache element identified(认出、标识) according(依照) to the [proxy\_cache\_key](http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_cache_key) directive by passing a request to a proxied server. Other requests of the same cache element will either wait for a response to appear in the cache or the cache lock for this element to be released(释放), up to the time set by the [proxy\_cache\_lock\_timeout](http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_cache_lock_timeout)directive.

## proxy\_cache\_lock\_age



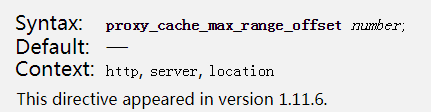
If the last request passed to the proxied server for populating a new cache element has not completed(完整的) for the specified time, one more request may be passed to the proxied server

## proxy\_cache\_lock\_timeout



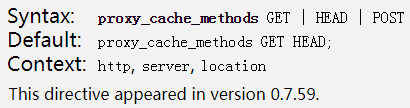
设置proxy\_cache\_key的超时时间，当超时，请求会发给proxy服务器，然而响应不会cached

## proxy\_cache\_max\_range\_offset



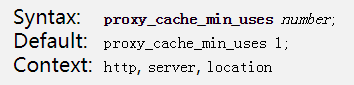
Sets an offset(抵消、偏移) in bytes for byte-range requests. If the range is beyond(超出) the offset, the range request will be passed to the proxied server and the response will not be cached

## proxy\_cache\_methods



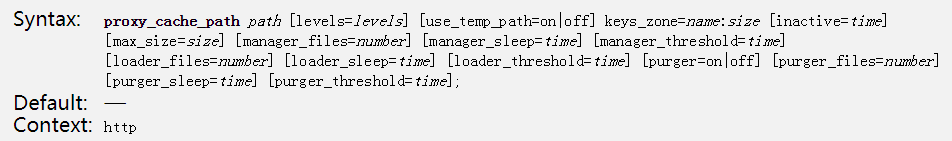
If the client request method is listed in(在列表中) this directive then the response will be cached. “GET” and “HEAD” methods are always added to the list, though it is recommended to specify them explicitly(虽然建议明确的指出他们). See also the [proxy\_no\_cache](http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_no_cache) directive.

## proxy\_cache\_min\_uses



设置在多少次请求以后才被缓存的最小次数

## proxy\_cache\_path



设置缓存的路径和其他参数，缓存数据是保存在文件中的，缓存的键和文件名都是在代理URL上执行MD5的结果

被缓存的响应首先先写入一个临时文件，然后进行重命名。从0.8.9版本开始，临时文件和缓存可以放在不同的文件系统。但请注意，这将导致文件在这两个文件系统中进行拷贝，而不是高效率的重命名操作。因此，针对任何路径，都建议将缓存和[proxy\_temp\_path](http://tengine.taobao.org/nginx_docs/cn/docs/http/ngx_http_proxy_module.html#proxy_temp_path)指令设置的临时文件目录放在同一文件系统

临时文件的目录是基于use\_temp\_path参数来设置的，如果设置为on，则使用给定proxy\_temp\_path命令设置的临时文件，如果是off，则临时文件将直接放入缓存目录中；建议off，避免文件在不同的文件系统中不必要的拷贝

此外，所有的键和缓存数据相关的(使用次数)信息都被存放在共享内存中。共享内存通过keys\_zone参数的name和size来定义。1m区域可以存储大约8000个键

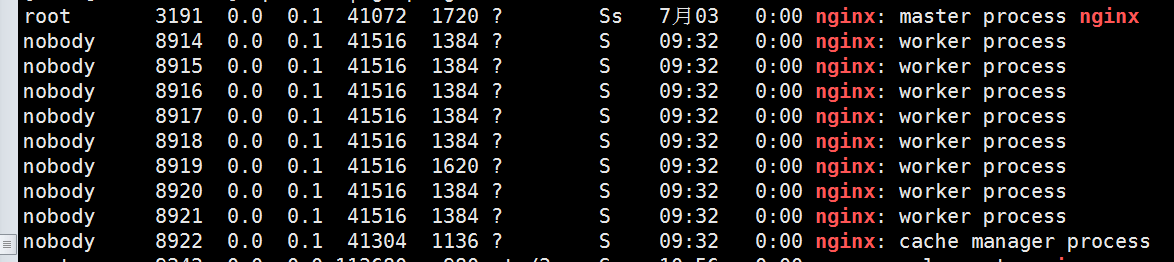
被缓存的数据如果在inactive参数指定的时间内未被访问，就会被从缓存中移除，不论它是否是刚产生的。inactive的默认值是10分钟

如果定义proxy\_cache\_valid any 10m，则以inactive为准，如inactive为10s，则10秒后删除

inactive 10d

inactive 10m

inactive 20s



特殊进程“cache manager”监控缓存的条目数量，如果超过max\_size参数设置的最大值，使用LRU算法移除缓存数据(如果缓存满，则删除掉最近最小使用的数据)

nginx新启动后不久，特殊进程“cache loader”就被启动。该进程将文件系统上保存的过去缓存的数据的相关信息重新加载到共享内存。加载过程分多次迭代完成，每次迭代，进程只加载不多于loader\_files参数指定的文件数量（默认值为100）。此外，每次迭代过程的持续时间不能超过loader\_threshold参数的值（默认200毫秒）。每次迭代之间，nginx的暂停时间由loader\_sleep参数指定（默认50毫秒）

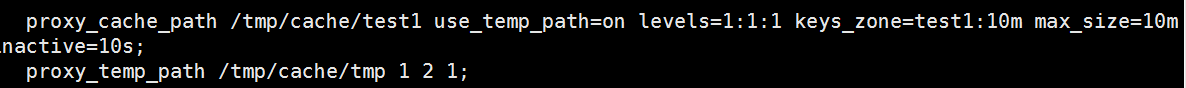
max\_size

最大cache空间，如果不指定，会使用掉所有的磁盘空间，当满后，删除最少使用的cache文件

path

定义缓存的目录

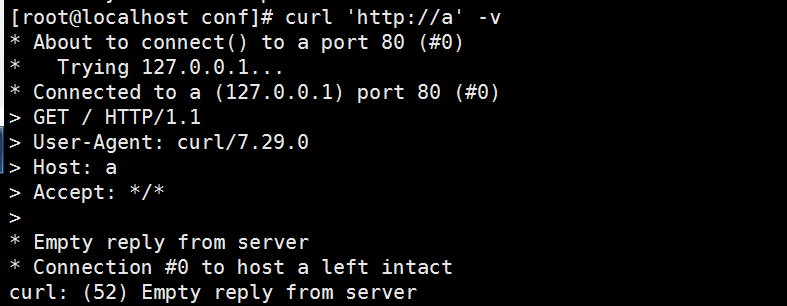
如果是以下配置



需要创建cache目录，然后test1和tmp会自动创建

如果运行期间把/tmp/cache删除，则访问会返回，nginx –s reload 不会自动创建/tmp/cache目录，需要手动创建。nginx –s reload 会自动创建/tmp/cache/test1 /tmp/cache/tmp

user\_temp\_path off的话，也是会创建/tmp/cache/tmp



levels

proxy\_cache\_path /data/nginx/cache levels=1:2

则缓存的文件

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

levels可接受3个级别，每个级别的值是1或者2

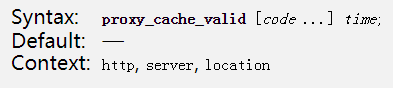
levels=1:2:2

levels=1:2:2代表/a/xx/dd这种 1:1:1代表/a/b/c这种

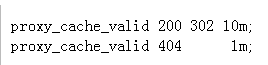
<http://phl.iteye.com/blog/2256857>

https://blog.csdn.net/dengjiexian123/article/details/53386586

## proxy\_cache\_valid



为不同的响应码设置响应时间



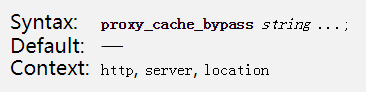
如果仅仅proxy\_cache\_valid 1m;这样只有200，301，302会被缓存

如果proxy\_cache\_valid any 1m; 可以缓存各种响应

缓存参数也可以直接在响应头中设定，这种方式的优先级高于使用proxy\_cache\_valid指令

“X-Accel-Expires”响应头可以以秒为单位设置响应的缓存时间，如果值为0，表示禁止缓存响应，如果值以@开始，表示自1970年1月1日以来的秒数，响应一直会被缓存到这个绝对时间点。 如果不含“X-Accel-Expires”响应头，缓存参数仍可能被“Expires”或者“Cache-Control”响应头设置。 如果响应头含有“Set-Cookie”，响应将不能被缓存。 这些头的处理过程可以使用指令proxy\_igonre\_headers忽略

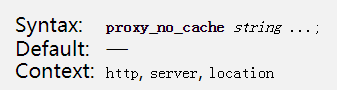
## proxy\_cache\_bypass



定义nginx不从缓存取响应的条件。如果至少一个字符串条件非空而且非“0”，nginx就不会从缓存中去取响应

C:\Users\ZHUXIA~1\AppData\Local\Temp\1530691405(1).png

## proxy\_no\_cache

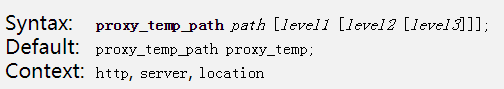


定义nginx不将响应写入缓存的条件。如果至少一个字符串条件非空而且非“0”，nginx就不将响应存入缓存

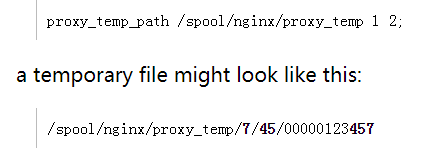
proxy\_no\_cache $cookie\_nocache(0) $arg\_nocache(1) $arg\_comment(0)，不会被cache

C:\Users\ZHUXIA~1\AppData\Local\Temp\1530675908(1).png

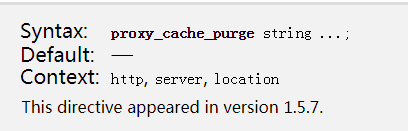
## proxy\_temp\_path



定义从后端服务器接受的临时文件的存放路径，可以为临时文件路径定义最大3层目录树

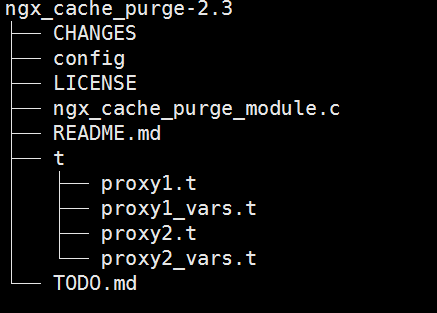


## proxy\_cache\_purage



需要额外安装模块ngx\_cache\_purge-2.3

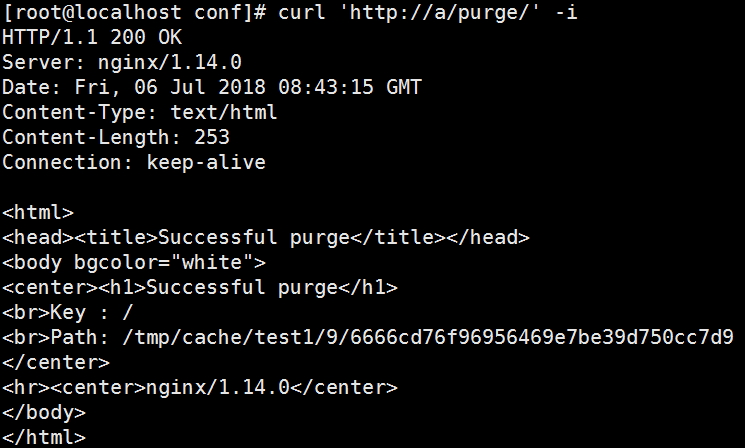
configure arguments: --add-module=../ngx\_cache\_purge-2.3



定义在什么情况下清除缓存

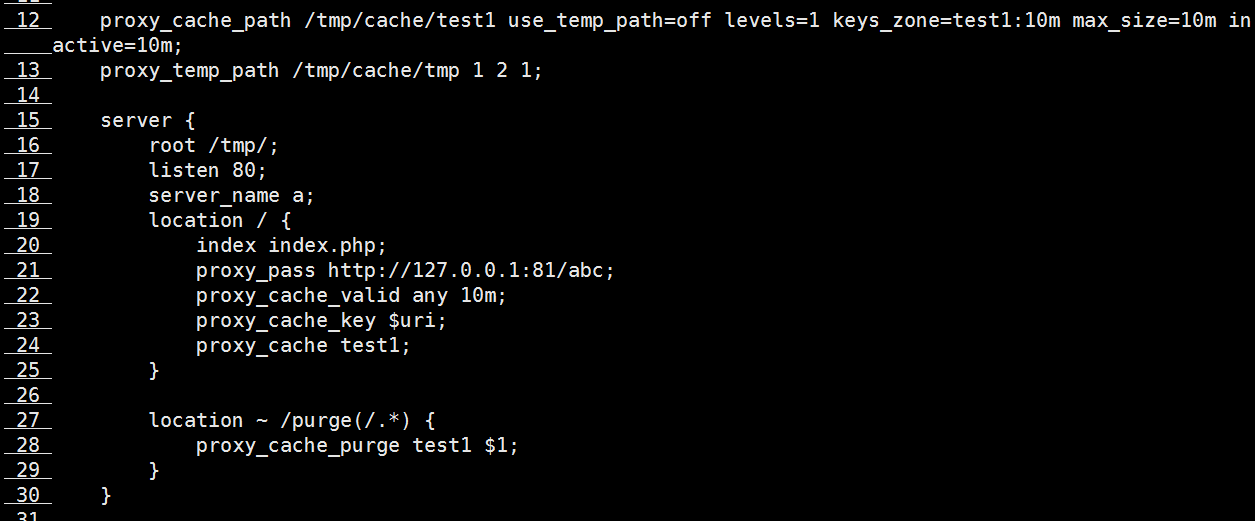
如果至少有一个参数，不为空或者不为0，然后清楚缓存的cache key

如果清除缓存的请求以\*结尾，则所有与通配符匹配的缓存都将被清除

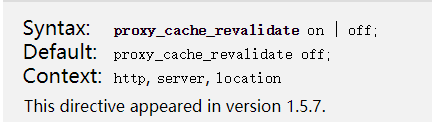


删除不成功返回404

需要请求curl ‘http://a/purge/’



## Proxy\_cache\_revalidate



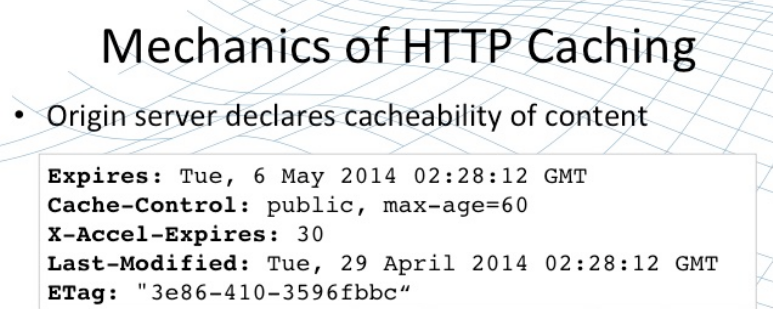
Enables revalidation(重新生效) of expired(过期) cache items using conditional requests with the “If-Modified-Since” and “If-None-Match” header fields

当请求头有If-Modified-Since和If-None-Match时候，允许过期缓存重新生效

## proxy cache总结



缓存服务器可以通过设置一些参数来忽略或者重写后端服务器的的缓存特性，但是后端服务器的缓存特性也是极其重要的



expires

最原始的配置策略，即设置过期时间，但是使用效率低下，目前绝大部分已经被cache-control取代

cache-control

定义缓存资源属性是public或者是private，并且设置缓存多久后过期

X-Accel-Expires

只有nginx才能识别的缓存特性header，优先级大于上面两个header，可以设置这个头来重新定义缓存特性

etag和last-modified

为捆绑出现，last-modified表示最后修改时间，并声明一个etag哈希值，作为缓存内容的标签，具有唯一性。客户端访问请求带有If‑Modified‑Since或者If‑None‑Match header，并申明自己的客户端带有静态缓存文件,以及文件修改日期和ETag值，如果服务器端的版本和Etag值与客户端一致，则服务端会直接返回304 not modified，这个验证流程是非常快的，并且节省网络带宽

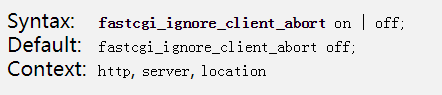
proxy启动cache，需要proxy\_cache，指定key\_zone

在nginx里面return 200 xxx，虽然能cache住，但是再次在缓存期内改变xxx，cache失效

# ngx\_http\_fastcgi\_module

该模块允许向FastCGI服务器传递请求

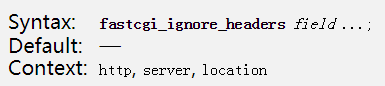
## fastcgi\_igonre\_client\_abort



Determines(决定、确定) whether(确定是否) the connection with a FastCGI server should be closed when a client closes the connection without waiting for a response

确定是否应该将与FastCGI的连接关闭，当客户端关闭连接而不等待响应的时候

## fastcgi\_ignore\_headers



禁止处理FastCGI服务器的响应头字段

可以禁止：

1.X-Accel-Redirect

2.X-Accel-Expires

3.X-Accel-Limit-Rate

4.X-Accel-Buffering

5. X-Accel-Charset

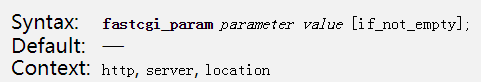
6.Expires

7.Cache-Control

8.Set-Cookie

9.Vary

## fastcgi\_param



设置一个parameter，该parameter向FastCGI服务器传递。value可以是变量和文本变量的结合

如果当前级别没有fastcgi\_param，则该指令从前一个级别继承

PHP所需要的最低配置，PHP的$\_SERVER是有$\_SERVER[“SCRIPT\_NAME”]和$\_SERVER[“SCRIPT\_FILENAME”]的

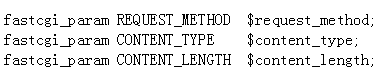
C:\Users\ZHUXIA~1\AppData\Local\Temp\1528080421(1).png

SCRIPT\_FILENAME在PHP中确定script

如果location /b{}，uri为/b/c/f/g，则$fastcgi\_script\_name为/b/c/f/g

QUERY\_STRING传递请求参数，GET的

处理POST请求，需要以下参数



$content\_length为body体长度，如果单纯get请求，$content\_length为0。如果POST请求，curl –d ”a=1”，该值为3

如果PHP编译时指定—enable-force-cgi-redirect（php –i | grep configure），则需要



如果提供if\_not\_empty(1.1.11)，则参数在不为空的情况下传递给fastCGI服务器

C:\Users\ZHUXIA~1\AppData\Local\Temp\1528081183(1).png

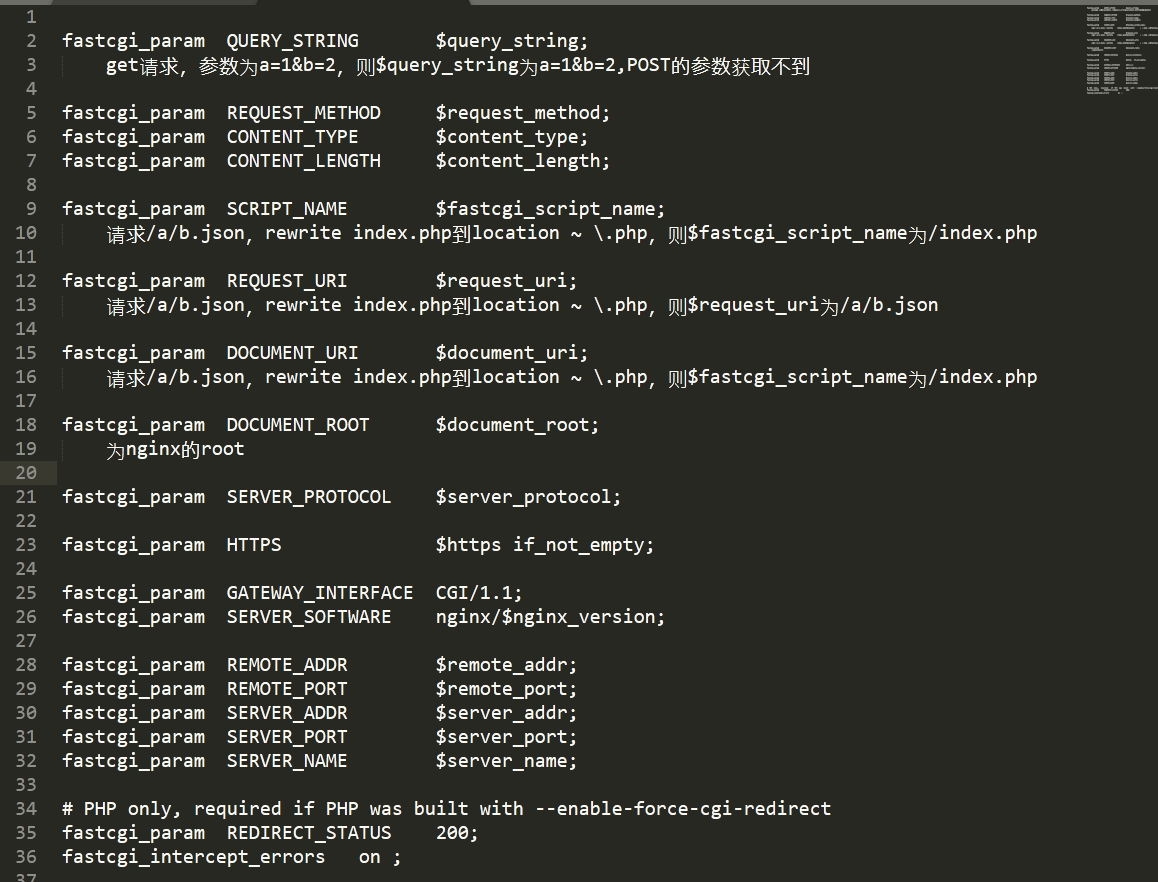
location /a {

alias /tmp/a/a;

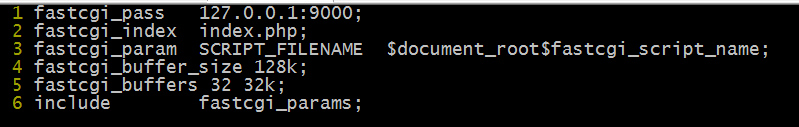
}

获取到的$document\_root为/tmp/a/a

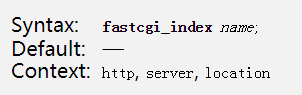
$document\_root参数，fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;



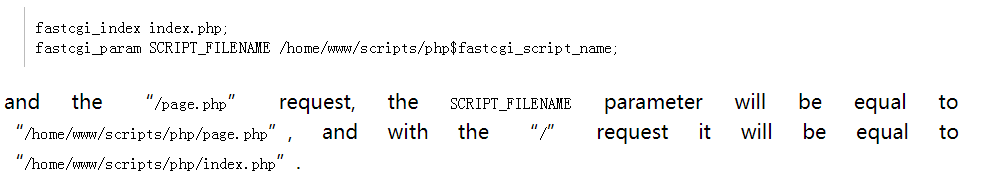
php\_fastcgi



## fastcgi\_index



如果请求以“/”结尾，则$fastcgi\_script\_name变量会在最后添加fastcig\_index的值



## fastcgi\_intercept\_errors



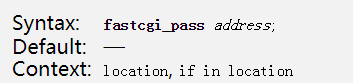
当FastCGI服务器响应码大于或者等于300时，决定是否直接将响应发给客户端，或者将响应转发给nginx由error\_page处理

Determines whether FastCGI server responses with codes greater than or equal to 300 should be passed to a client or be intercepted and redirected to nginx for processing with the error\_page directive.

如果是off，当fastcgi返回500，使用error\_page 500 @abc;，这样error\_page不会生效

on的话，error\_page会生效

## fastcgi\_pass



设置FastCGI服务器地址，地址可以被设置为domain或者是IP

fastcgi\_pass localhost:9000;

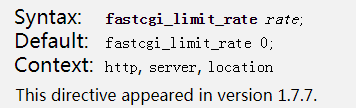
fastcgi\_pass unix: /tmp/fastcgi.socket;

如果域名解析为多个地址，他们将以循环方式使用，此外，可以指定一个地址作为服务器组（ngx\_http\_upstream\_module）

参数可以包含变量，在这种情况下，如果地址被指定为域名，则在所定义的服务器组中搜索名称，如果未找到，则使用resolver



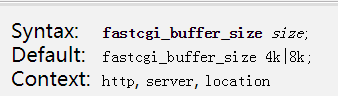
## fastcgi\_limit\_rate



限制从FastCGI服务器读取响应的速度，rate参数是每秒字节数，0的话就是不限制

每个请求都限制，如果nginx同时打开了两个连接到FastCGI，则限速将是值的两倍

## fastcgi\_buffer\_size

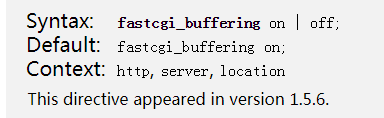


设置buffer大小用来读取FastCGI服务器的响应。这部分通常包含一个小的响应头。默认buffer大小等于一个内存页，4K或者8K，取决于系统

fastcgi\_buffer\_size 64K表示使用一个64KB的缓冲区读取应答的第一部分（应答头）

Sets the size of the buffer used for reading the first part of the response received from the FastCGI server. This part usually contains a small response header. By default, the buffer size is equal to one memory page. This is either 4K or 8K, depending on a platform. It can be made smaller, however

## fastcgi\_buffering



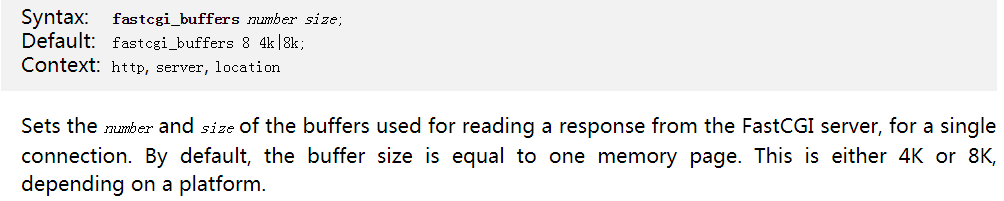
使用或者禁止缓冲(buffering)FastCGI的响应，当启用buffering时候，nginx将尽快从FastCGI服务器接受响应。保存在被fastcgi\_buffer\_size和fastcgi\_buffers指令设置的buffer中

如果整个响应没有被存到内存中，一部分会被写进临时文件在硬盘中，会被fastcgi\_max\_temp\_file\_size和fastcgi\_temp\_file\_write\_size和fastcgi\_temp\_path指令控制

When buffering is disabled, the response is passed to a client synchronously（同时、同步）, immediately（立即） as it is received. nginx will not try to read the whole response from the FastCGI server.The maximum size of the data that nginx can receive from the server at a time is set by the [fastcgi\_buffer\_size](http://nginx.org/en/docs/http/ngx_http_fastcgi_module.html#fastcgi_buffer_size) directive

buffering也可以用响应头X-Accel-Buffering的yes或者on来控制，这个可以被fastcgi\_igonre\_headers来控制

## fastcgi\_buffers



对于单个连接

指定需要多少和多大的缓冲区来缓冲FastCGI的应答，如果一个PHP脚本所产生的页面大小为256KB，那么会为其分配4个64KB的缓冲区来缓存；如果页面大小大于256KB，那么大于256KB的部分会缓存到fastcgi\_temp指定的路径中，但是这并不是好方法，因为内存中的数据处理速度要快于硬盘。一般这个值应该为站点中PHP脚本所产生的页面大小的中间值，如果站点大部分脚本所产生的页面大小为256KB，那么可以把这个值设置为“16 16k”、“4 64k”等

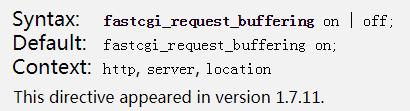
https://www.cnblogs.com/yezhaohui/p/4377662.html

## fastcgi\_busy\_buffers\_size

当buffering被使用，limits the total size of buffers that can be busy sending a response to the client while the response is not yet fully read(限制buffers的总大小，用来忙着响应给客户端，当响应尚未读完)。同时，其余buffers可以用来读取响应，如果需要，也可以写进temp

By default, size is limited by the size of two buffers set by the fastcgi\_buffer\_size and fastcgi\_buffers directives

## fastcgi\_request\_buffering

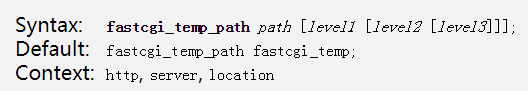


开启或者禁用buffer客户端请求body体

开启时，在向fastcgi服务器发送请求时，会从客户端读取整个body体

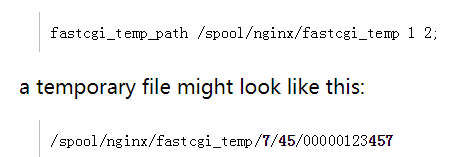
关闭时，客户端body体会立即传给FastCGI服务器。In this case, the request cannot be passed to the next server if nginx already started sending the request body

## fastcgi\_temp\_path

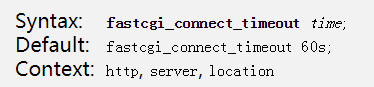


定义一个目录，用来存储从FastCGI服务器收到的数据。在指令目录下可以使用最多3级子目录层次结构

fastcgi\_temp 默认在/usr/local/nginx/fastcgi\_temp

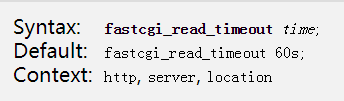


## fastcgi\_connect\_timeout



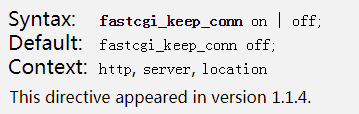
Defines a timeout for establishing(建立) a connection with a FastCGI server. It should be noted（应该注意的是） that this timeout cannot usually exceed 75 seconds（这个时间不能超过75s）

## fastcgi\_read\_timeout



Defines a timeout for reading a response from the FastCGI server. The timeout is set only between two successive read operations(超时时间仅设置于两次成功的读取操作), not for the transmission of the whole response（不是整个传输响应）. If the FastCGI server does not transmit anything within this time, the connection is closed

## fastcgi\_keep\_conn



默认情况下，a FastCGI server will close a connection right after（刚好在…之后） sending the response. However, when this directive is set to the value on, nginx will instruct(通知、命令) a FastCGI server to keep connections open. This is necessary(这是重要的), in particular,(特别是) for [keepalive](http://nginx.org/en/docs/http/ngx_http_upstream_module.html#keepalive) connections to FastCGI servers to function

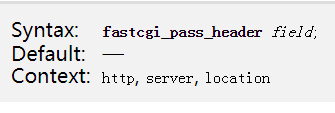
## fastcgi\_hide\_header



By default, nginx does not pass the header fields “Status” and “X-Accel-...” from the response of a FastCGI server to a client（默认情况下，nginx不传递FastCGI响应header字段Status和X-Accel-…给客户端）. The fastcgi\_hide\_header directive sets additional(额外的) fields that will not be passed. If, on the contrary(相反), the passing of fields needs to be permitted（允许）, the [fastcgi\_pass\_header](http://nginx.org/en/docs/http/ngx_http_fastcgi_module.html#fastcgi_pass_header)directive can be used

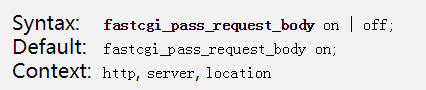
fastcgi\_hide\_header指令设置额外的不传递头字段。相反，如果允许头字段传递，需要使用fastcgi\_pass\_headerdirective

## fastcgi\_pass\_header



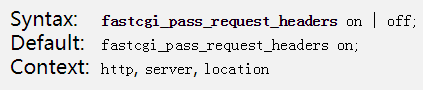
允许传递被不允许传递的头信息从FastCGI给客户端

## fastcgi\_pass\_request\_body



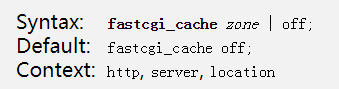
Indicates(表明、声明) whether the original request body is passed to the FastCGI server（是否将原始请求的body体传给FastCGI）

## fastcgi\_pass\_request\_headers



Indicates whether the header fields of the original request are passed to the FastCGI server

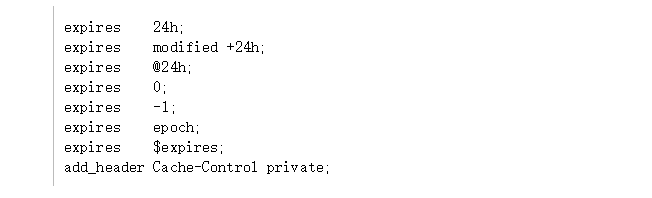
## fastcgi\_cache



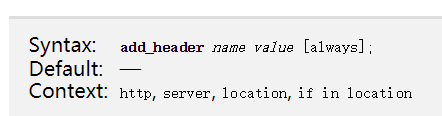
定义一个共享内存区用来使用caching，这个共享内存区可以用在很多地方。当off时候，当前级别可以从以前级别来继承配置

# ngx\_http\_headers\_module

该模块可以添加Expires、Cache-Control和任意的响应头



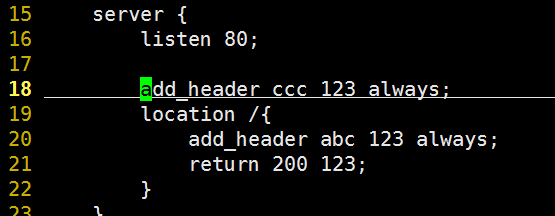
## add\_header



如果响应码为200、201、204、206、301、302、303、304、307、308则添加指定的响应头

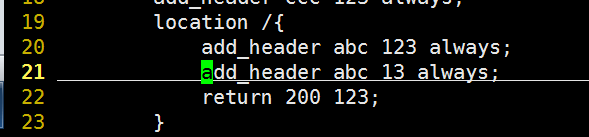
如果当前级别没有add\_header，则从上一个级别继承，仅仅当前级别没有的话

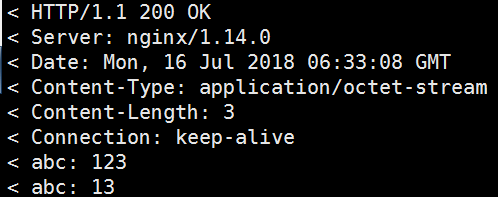
如果定义了always，则不管响应码为多少都添加header



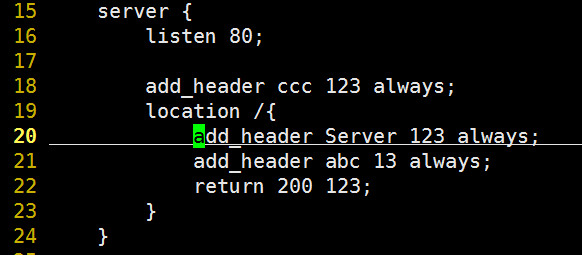
仅仅有abc 123生效

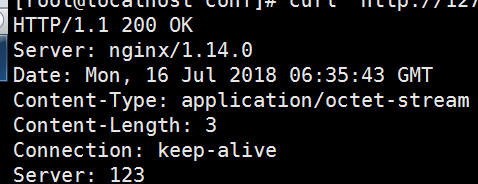
如果定义了两个abc 123，则输出的响应会有两个abc



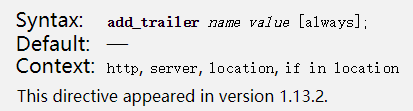


如果键为Server，则不会覆盖基本的Server输出





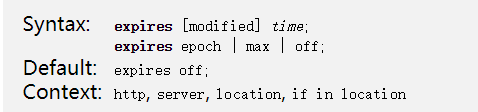
## add\_trailer(不知道怎么使用)



添加响应到响应头的末尾，如果响应码为200, 201, 206, 301, 302, 303, 307, 308

如果当前级别没有add\_header，则从上一个级别继承，仅仅当前级别没有的话

## expires



使用或者禁止添加或者改变expires和Cache-Control响应头在响应码为200, 201, 206, 301, 302, 303, 307, 308情况下

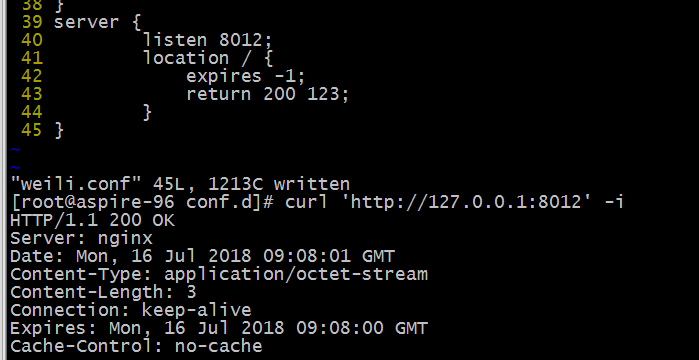
参数可以是正数或者负数

The time in the “Expires” field is computed as a sum of the current time and time specified in the directive. If the modified parameter is used (0.7.0, 0.6.32) then the time is computed as a sum of the file’s modification time and the time specified in the directive

此外，可以使用@，指定一天的时间

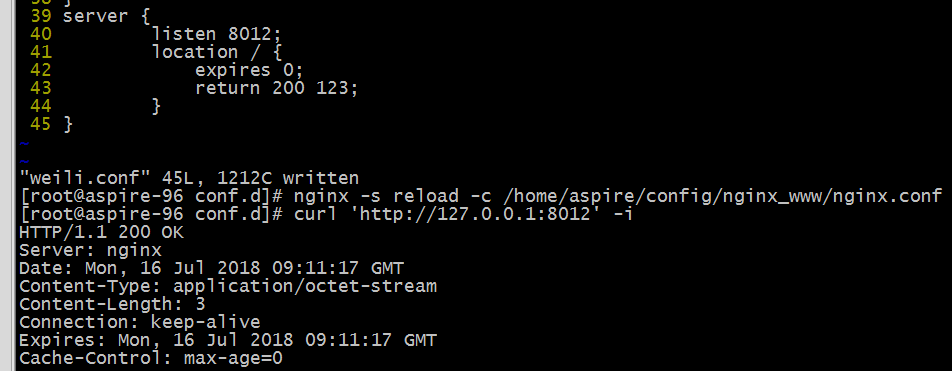
expire @15h30m;

如果time是负数，则Cache-Control:no-cache

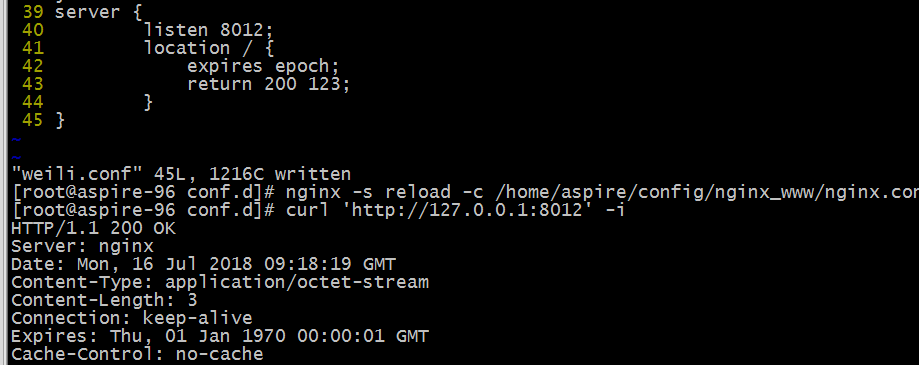


如果是正数或者0，则Cache-Control:max-age:t，t的值取决于time参数

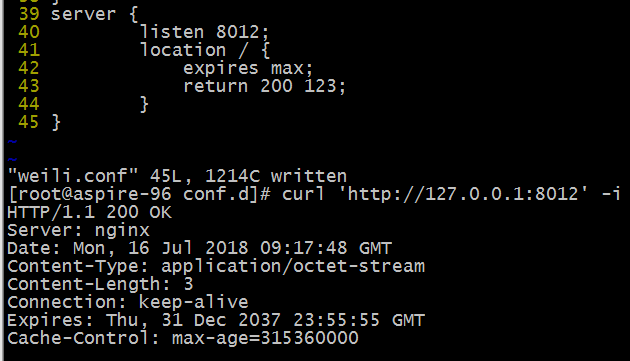




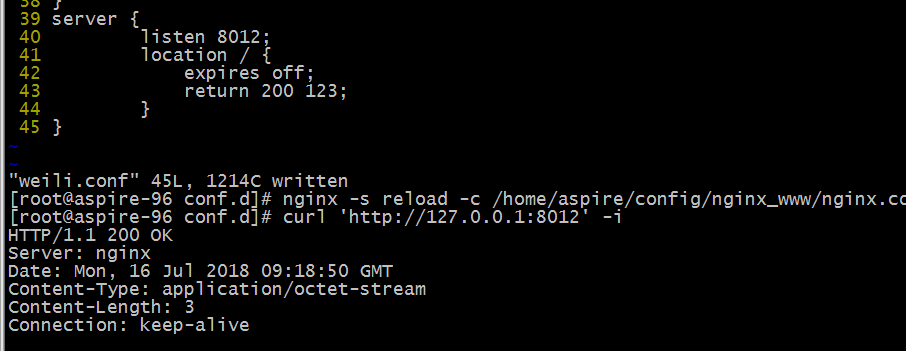
epoch参数指定Expires为Thu, 01 Jan 1970 00:00:01 GMT

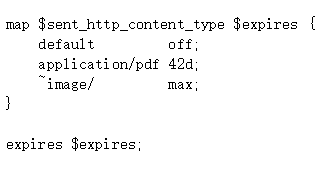


max参数指定

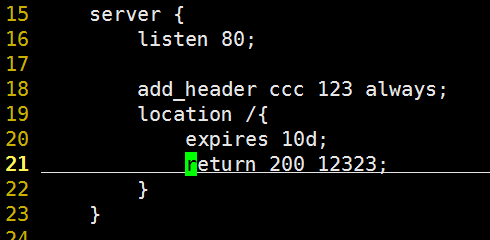


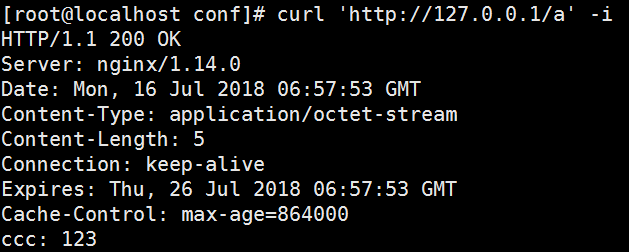
off参数可以禁用添加或者修改expire和Cache-Control响应



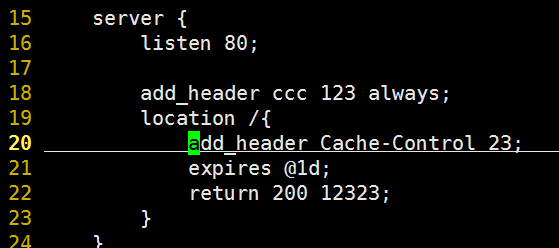


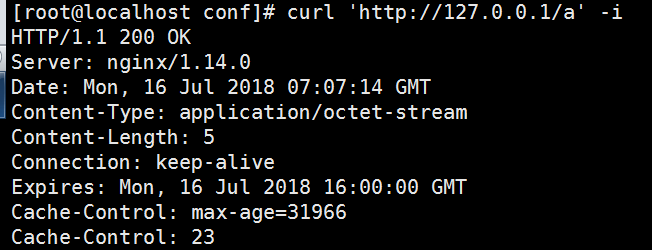
总结





如果定义了expire又指定了add\_header Cache-Control则





## 缓存的使用

### http\_code 304

304状态码不应该认为是一种错误，而是对客户端有缓存情况下服务器的一种响应

客户端在请求一个文件的时候，发现自己缓存的文件有Last Modified，那么请求中会包含If Modified Since，这个时间就是缓存文件的Last Modified，如果请求中有If Modified Since，就说明已经有缓存在客户端，服务端只要判断这个时间和当前请求的文件的修改时间就可以确定是返回200还是304

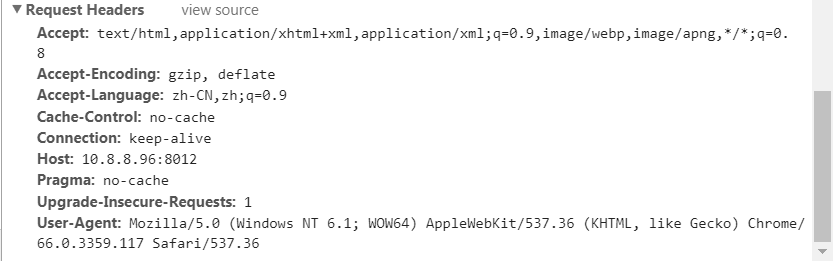
对于静态文件，如CSS、图片、js文件，服务器会自动完成Last Modified和If Modified Since的比较，完成缓存或者更新。对于动态页面，就是动态产生的页面，往往没有包含Last Modified信息，这样浏览器、网关都不会缓存，也就是在每次请求的时候都会完成一个200的请求

对于动态页面缓存加速，首先要在响应的header中增加一个Last Modified，其次根据请求中的If Modified Since和被请求的内容更新时间来返回200或者304

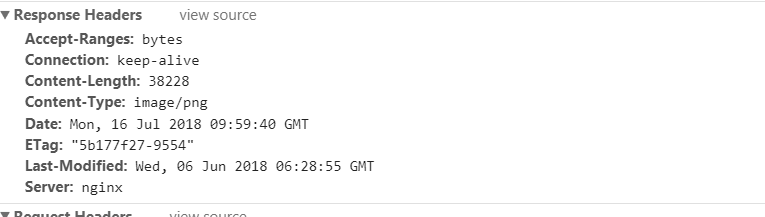
304只返回了一个HTTP header，从而大大降低了带宽的消耗

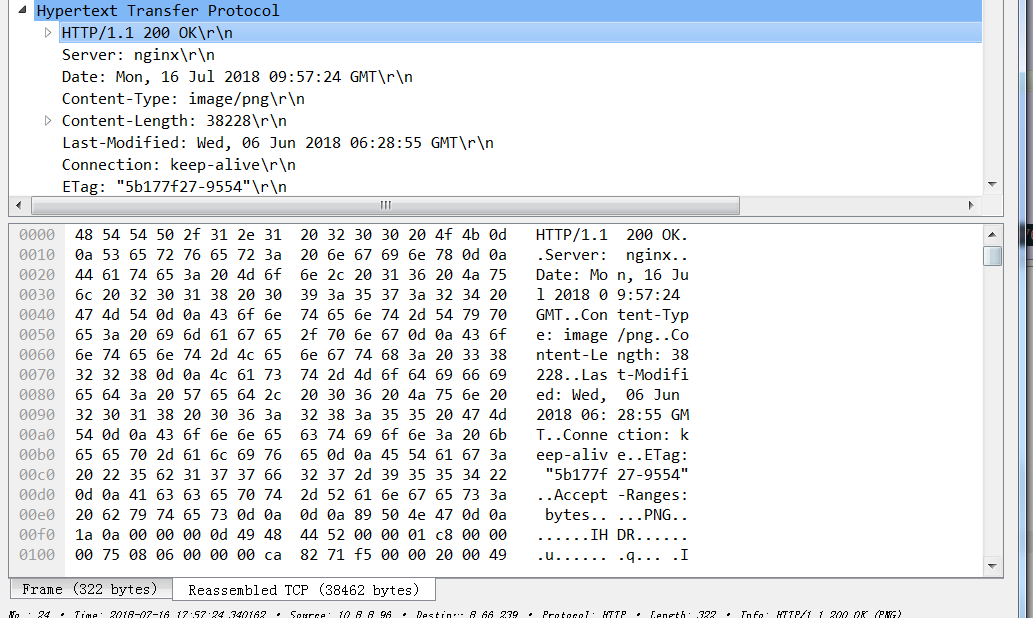
可以使用ctrl+F5强制刷新来返回200

第一次请求

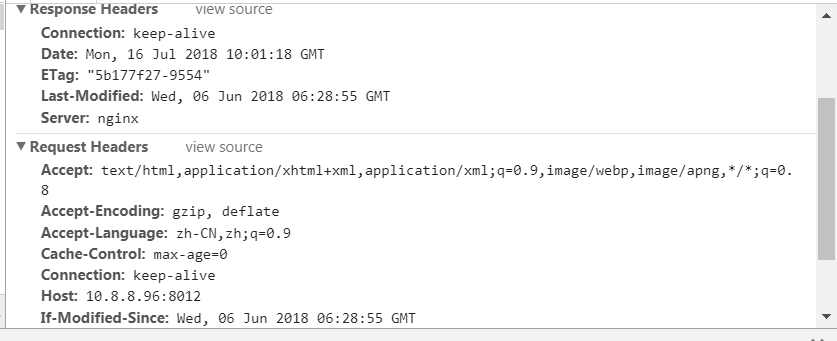


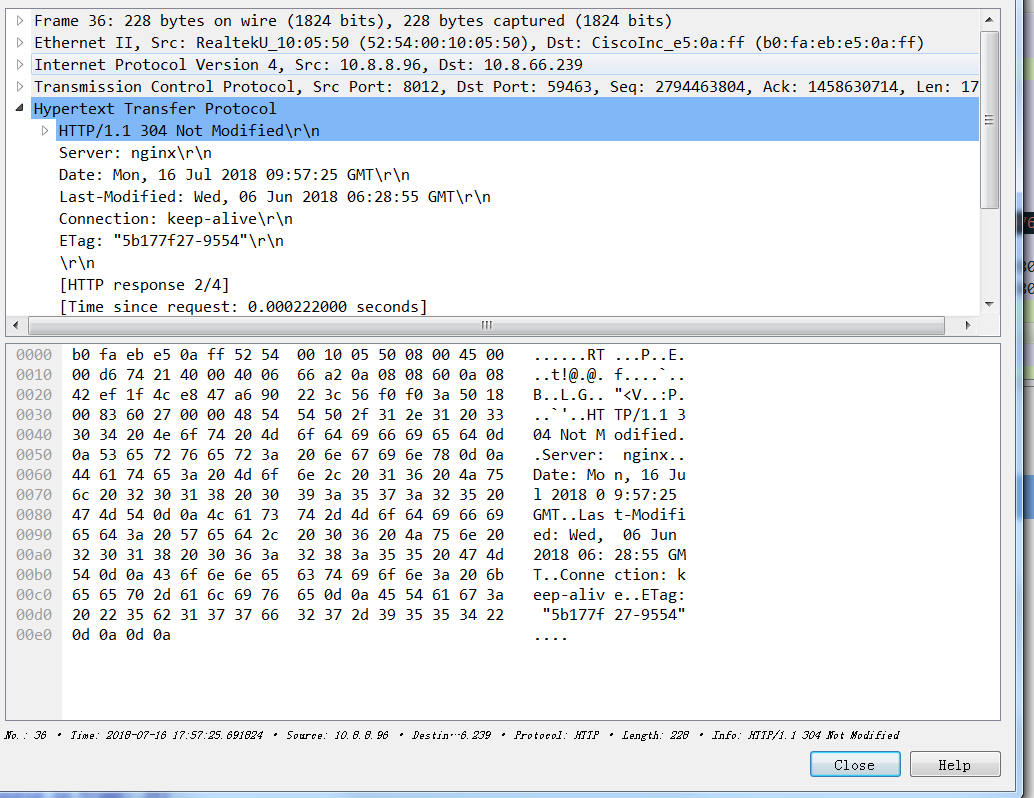
响应的Last-Modifiey是图片的修改时间





第二次请求304





### 协商缓存

If-Modified-Since/Last-Modified，服务器检查请求头If-modified-Since，如果请求文件最后修改时间相同则返回304

If-None-Match/Etag，服务器检查请求头If-None-Match的值与请求文件的etag，相同则返回304

### 强缓存

设置了expires或者cache-control(http 1.0不支持)，则请求不会传递给服务器

但是一般浏览器强缓存都不会生效，因为自带请求头Cache-Control:max-age=0

一般的浏览器会忽略Cache-Control和expires头，如果在同一个选项卡上很快的请求同一个url（刷新，F5或者command+R）

### ETag

用来标识一个资源，ETag可以是资源的hash值，也可以是一个内部维护的版本号，需要反映出资源的内部变化

### Cache-Control

max-age

从当前请求开始，允许获取的响应被重用的最长时间，单位是秒，如Cache-Control:max-age=60，标识响应可以再缓存和重用60秒。在max-age指定的时间内，浏览器不会向服务器发送任何请求，包括验证缓存是否有效的请求，服务器在这段时间内资源发生变化，那么浏览器将不能得到通知

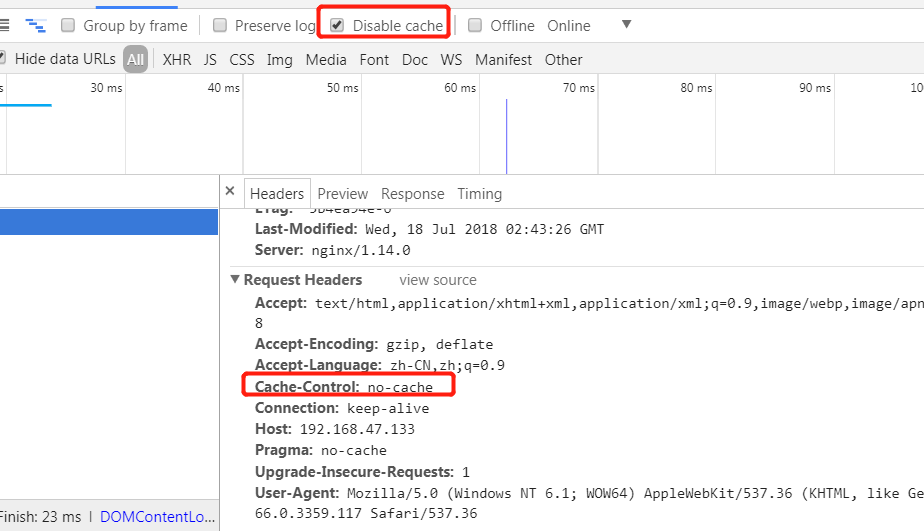
当请求max-age>0时，直接从浏览器的缓存中取得资源

当响应max-age>0时，代表客户端应该缓存多少秒

请求max-age<=0时，向服务器发送确认请求，是否有修改，有的话就200，没有就304

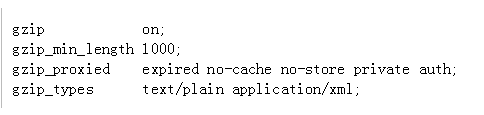
no-cache

强制每次请求直接发给服务器，不经过本地缓存校验

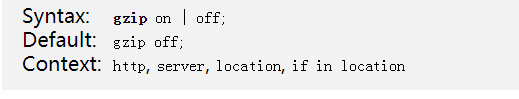


# ngx\_http\_gzip\_module

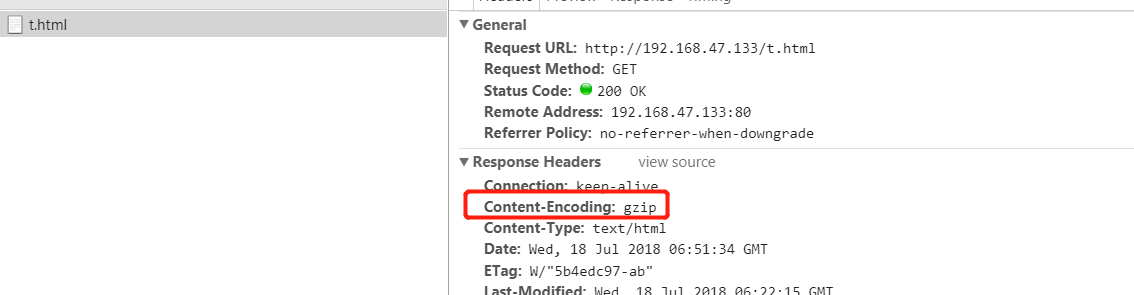
该模块使用gzip方法压缩响应，通常将是响应传输大小减半或者更小

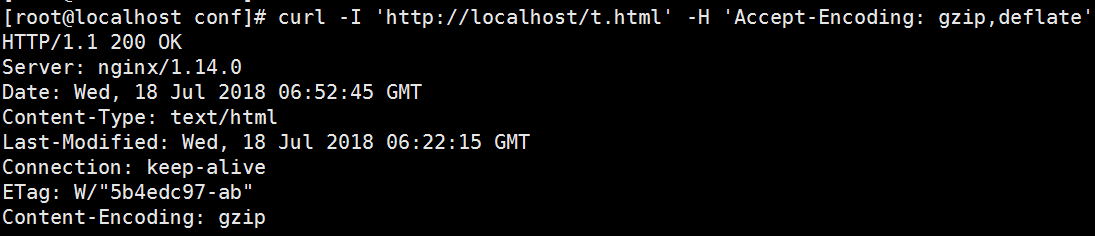
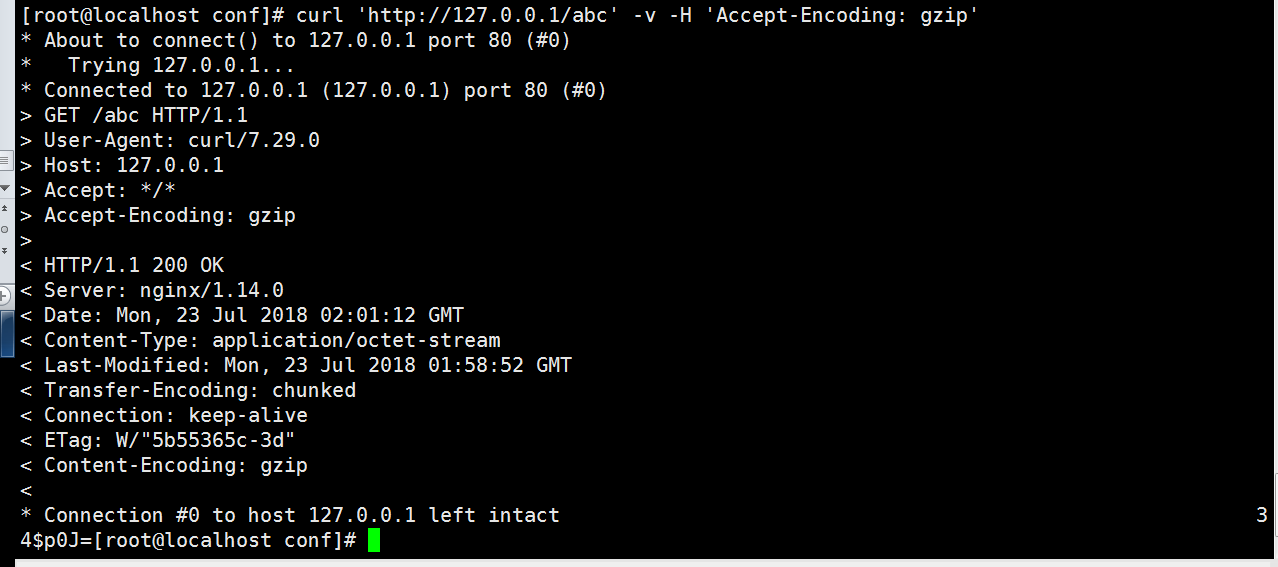


## gzip

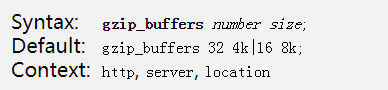


开启或者使用gzip



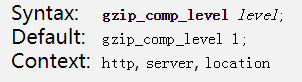


## gzip\_buffers



设置buffers的name和size用于压缩响应，默认情况下，size是一个内存页的大小，等于4K或者8K，取决于平台

## gzip\_comp\_level



设置压缩等级，可用的值为1-9

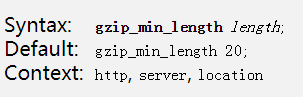
9最慢但是压缩比最高

## gzip\_http\_version



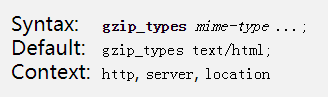
设置使用HTTP version请求开启压缩的最小限制

## gzip\_min\_length



设置被压缩的最小响应长度，这个长度仅仅取决于content-length

## gzip\_types

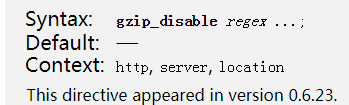


使除了text/html的响应都能压缩

特殊值\*将使所有的都得到压缩

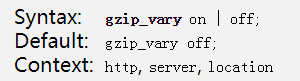
响应text/html都能被压缩

## gzip\_disable



禁用匹配上的User-Agent请求字段使用压缩

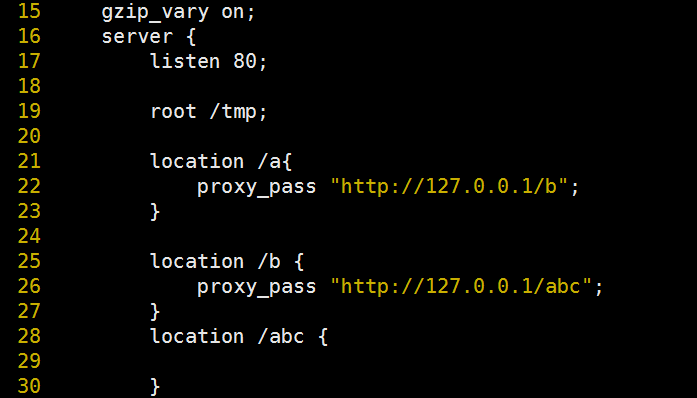
## gzip\_vary

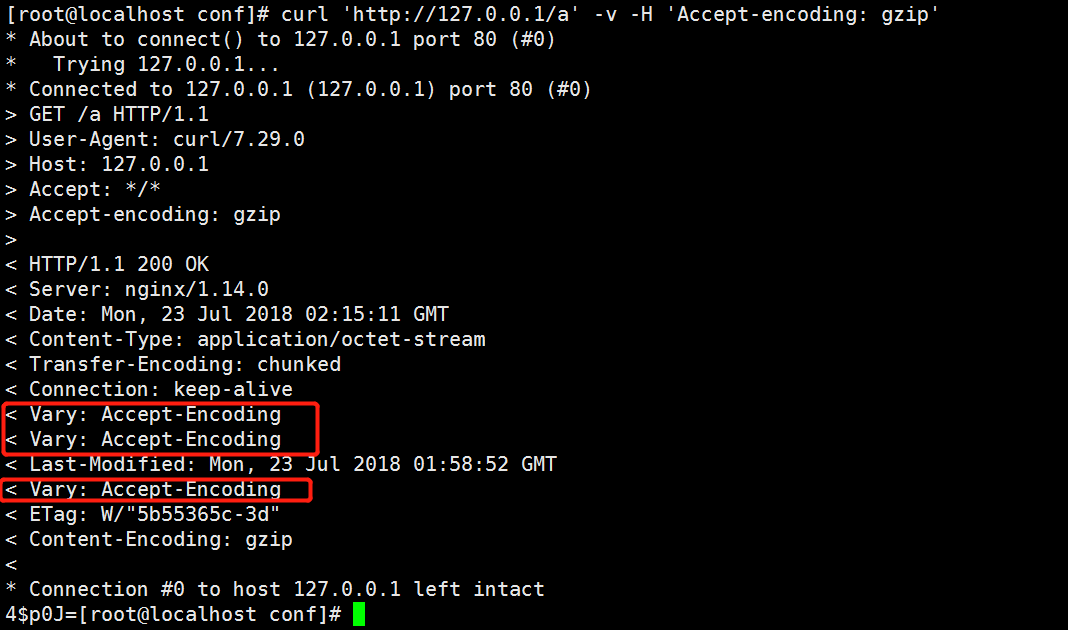


如果gzip，gzip\_static或者gunzip被开启，则在响应中体检Vary: Accept-Encoding字段



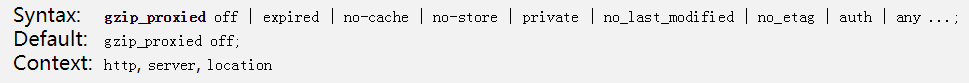
在代理情况下，gzip\_vary配置不能写在server外面，应该写在location /a里面





## gzip\_proxied

在反向代理情况下使用



Enables or disables gzipping of responses for proxied requests depending on the request and response. The fact that the request is proxied is determined by the presence of the “Via” request header field. The directive accepts multiple parameters:

off

disables compression for all proxied requests, ignoring other parameters;

expired

enables compression if a response header includes the “Expires” field with a value that disables caching;

no-cache

enables compression if a response header includes the “Cache-Control” field with the “no-cache” parameter;

no-store

enables compression if a response header includes the “Cache-Control” field with the “no-store” parameter;

private

enables compression if a response header includes the “Cache-Control” field with the “private” parameter;

no\_last\_modified

enables compression if a response header does not include the “Last-Modified” field;

no\_etag

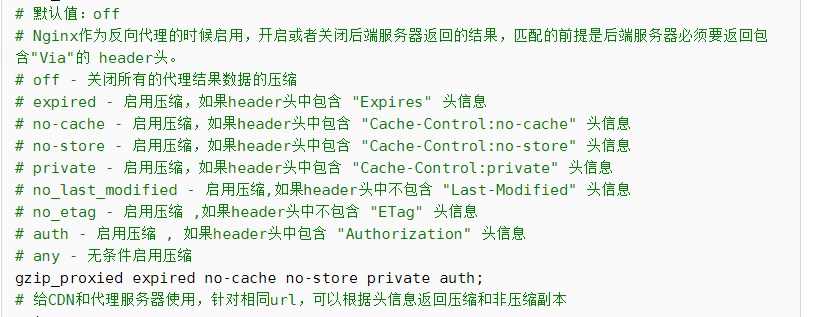
enables compression if a response header does not include the “ETag” field;

auth

enables compression if a request header includes the “Authorization” field;

any

enables compression for all proxied requests.



# ngx\_http\_gzip\_static\_module

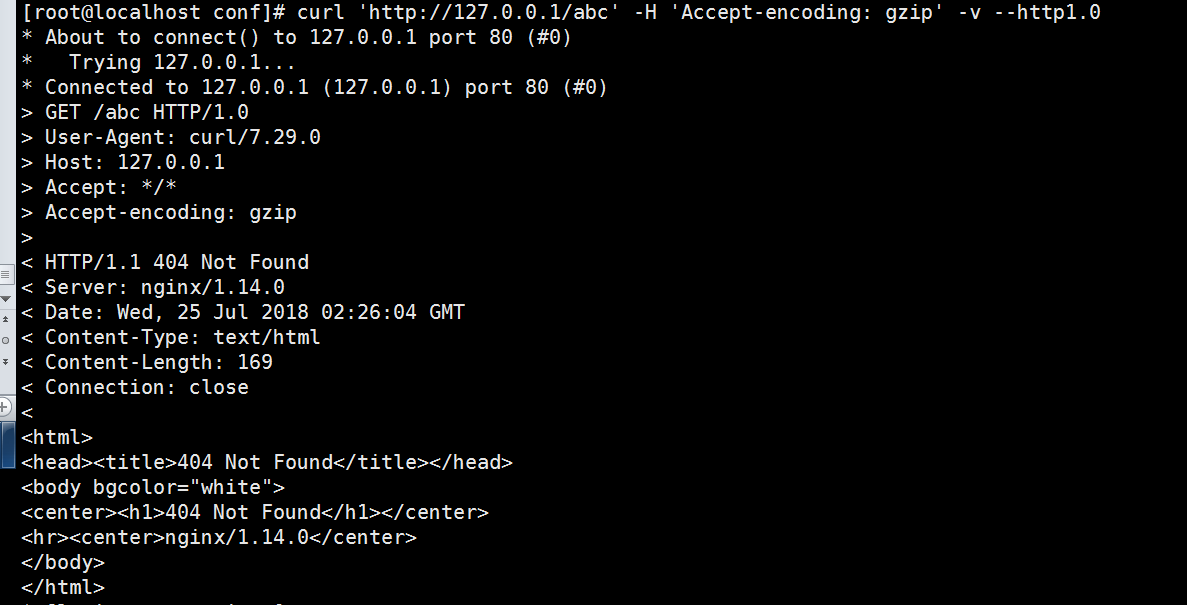
该模块允许发送以.gz为结尾的压缩文件替代常规文件

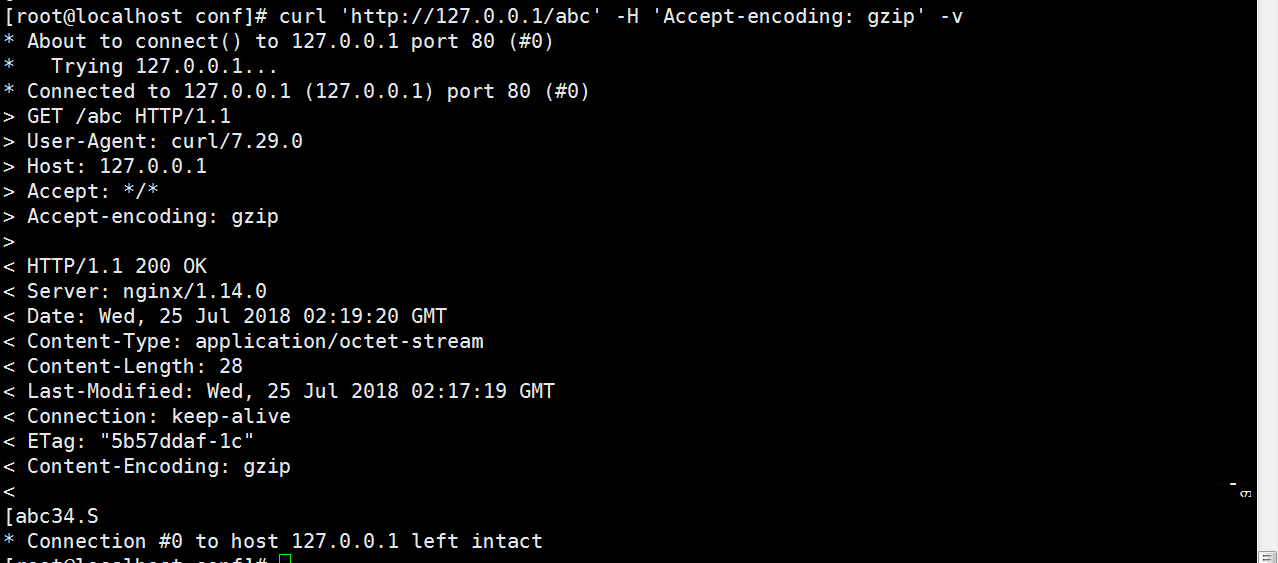
The ngx\_http\_gzip\_static\_module module allows sending precompressed（预压缩） files with the “.gz” filename extension（文件结尾） instead of regular（常规） files

这个模块不会默认编译，需要—with-http\_gzip\_static\_module

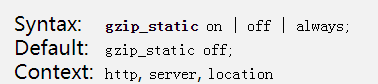
nginx的动态压缩是对每个请求先压缩再输出，会浪费很多cpu，可以直接读取已经压缩好的文件，文件名加.gz，而不是动态压缩，对于不支持gzip的请求则读取文件

优先级比gzip高，对于任何文件都会先查找是否有对应的gz文件，默认适用于HTTP 1.1





## gzip\_static



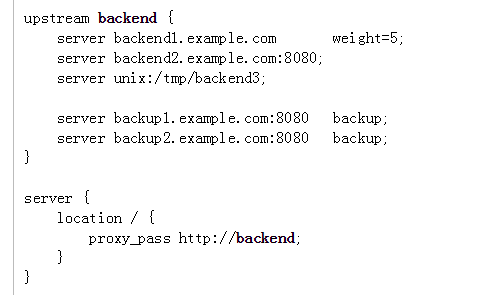
使用或者禁用检查压缩文件是否存在

always参数，gzipped文件将会被任何情况下使用，不检查客户端是否支持，在没有压缩文件在磁盘上或者ngx\_http\_gunzip\_module模块被使用的情况下非常有用

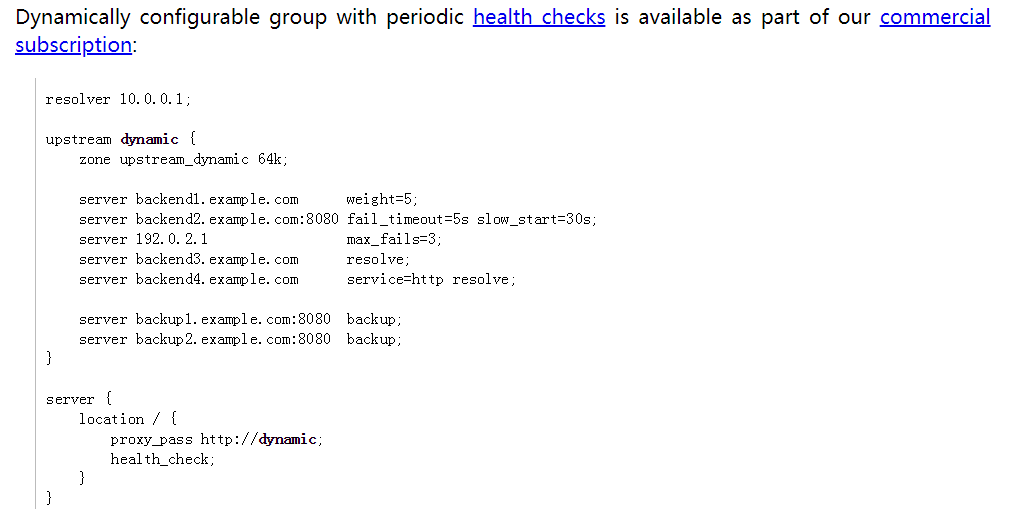
文件可以被命令行gzip压缩，或者其他兼容的压缩方式。建议原始文件和压缩文件的修改日志是相同的

# ngx\_http\_upstream\_module

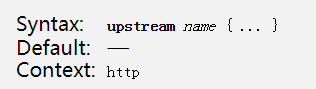
该模块可以定义一组服务器组，用于proxy\_pass，fastcgi\_pass，uwsgi\_pass，scgi\_pass，memcached\_pass，grpc\_pass



health\_checks是ngx\_http\_upstream\_hc\_module的指令，是商业版本



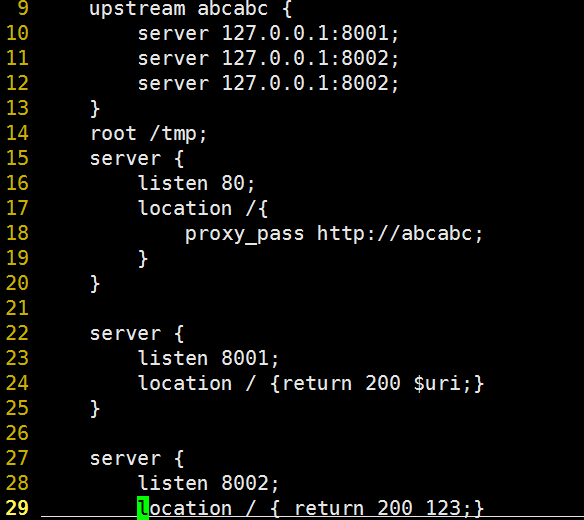
## upstream



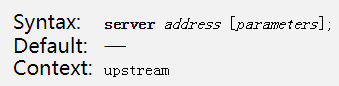
定义一个服务器组，服务可以监听不懂的端口，此外也可以混合监听TCP和UNIX-domain sockets



默认情况下，服务器将使用加权轮转方式请求，每7个请求会通过以下方式分发： 5个请求分到backend1.example.com， 一个请求分到第二个服务器，一个请求分到第三个服务器。 与服务器通信的时候，如果出现错误，请求会被传给下一个服务器，直到所有可用的服务器都被尝试过。 如果所有服务器都返回失败，客户端将会得到最后通信的那个服务器的（失败）响应结果



## server



server后面不能有uri，前面不能有协议如http://

定义服务器地址address和其他参数，地址可以是域名或者IP，端口可选，或者是指定unix前缀的unix套接字层，如果没有指定端口，就是用80，如果一个域名解析到多个IP，本质上是定义了多个server

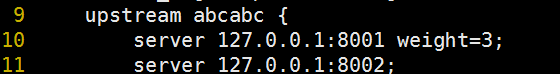
如果请求的服务器失败，则会直接请求下一个服务器

如果轮询81 82 83端口，proxy\_next\_upstrem http\_404，81 82 83都返回404，则输出为81/83

如果定义的upstream是abc，/etc/hosts也是abc，则使用upstream的

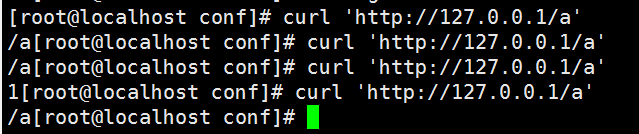
weight=number

服务器权重，默认是1



就是每4个请求，会分配给8001端口3次，8002端口1次；不是前3次分配8001；

如果没有8001端口，会直接不请求，直接请求8002



max\_fails=number

nginx与服务器通信尝试失败的次数，在fail\_timeout参数定义的时间内，如果失败的次数达到此值，nginx就认为服务器不可用，在下一个fail\_timeout时间段，服务器不会再被尝试。失败的尝试次数默认是1，设置为0就会停止统计尝试次数，认为服务器一直可用。可以通过proxy\_next\_upstream、fastcgi\_next\_upstream、memcached\_next\_upstream来配置什么是失败的尝试，默认配置时，404状态不认为是失败的尝试

fail\_timeout=time

统计失败尝试次数的时间段，服务器失败次数达到指定的尝试次数，服务器就被认为不可用

默认为10

backup

标记一个服务器为备用服务器

当主服务器不可用的时候，使用备用服务器

如果定义了多个backup，则在backup里面轮询

如果轮询81 82 83端口，proxy\_next\_upstrem http\_404，81 82 83都返回404，82端口为backup，则输出为82

down

标记一个服务器不可用

## zone



定义一个name和size用于共享内存，用于在每个工作进程中保存服务器组配置和运行时状态。相同的服务器有相同的配置，所以定义一个zone就可以了

