Nginx配置详解

Nginx常见的配置文件有以下几个

nginx.conf：应用程序的基本配置文件

mime.types：MIME类型关联的扩展文件

fastcgi.conf：与fastcgi相关的配置

proxy.conf：与proxy相关的配置

sites.conf：配置Nginx提供的网站，包括虚拟主机

nginx.conf配置

Nginx的配置是在nginx.conf中进行配置的

主要由三部分组成

全局块

events块

http块

全局块

从配置文件开始到events块之间的内容是全局块，主要用来配置Nginx服务器整体运行的配置指令，主要包括配置运行Nginx的用户组、允许生成的worker process数，进行PID存放路径、日志存放路径和类型以及配置文件的引入

#nginx中worker进程的用户和用户组

#user nobody;

# worker进程启动的数量，值越大，处理并发数量越多

# worker进程用于处理客户的连接

# 通常为cpu数量

worker\_processes 1;

#全局错误日志定义 位置 级别([ debug | info | notice | warn | error | crit ])

#error\_log logs/error.log;

#error\_log logs/error.log notice;

#error\_log logs/error.log info;

#进程文件存放位置

#pid logs/nginx.pid;

#一个nginx进程打开的最多文件描述符数目,理论值应该是最多打开文件数（系统的值ulimit -n）与nginx进程数相除,但是nginx分配请求并不均匀,所以建议与ulimit -n的值保持一致.

worker\_rlimit\_nofile 65535;

error\_log off不是禁用日志，而是创建一个名为off的日志，要禁用日志，可以这么写：error\_log /dev/null crit;

##### events块

主要是影响Nginx服务器与用户网络的连接，常用的配置包括是否开启对多worker process下的网络连接进行序列化，是否允许同时接受多个网络连接，选取那种事件驱动模型来处理连接请求，每个worker process可以同时支持的并发

events {

# 使用连接方式

#use [ kqueue | rtsig | epoll | /dev/poll | select | poll ];

#epoll模型是Linux内核中的高性能网络I/O模型，如果在mac上面，就用kqueue模型。

use kqueue;

# 每个worker进程可以同时支持的连接数为1024

# nginx支持的最大并发数是worker\_processes\*worker\_connections

#如果是作为反向代理服务器的话，最大并发数是worker\_processes\*worker\_connections/2,因为反向代理服务器，每个并发会建立与客户端的连接和与后端的连接，会占用两个连接

worker\_connections 1024;

# on|off,默认为on,将会对多个nginx进程接收连接进行序列化，防止多个进程对连接的争抢

accept\_mutex on;

# on|off,默认为off, 若为on，则每个nginx服务器的进程都可以接收多个新到达的网络连接

multi\_accept on;

}

##### http块

Nginx配置中修改最频繁的配置就是http块的配置，代理、缓存和日志定义等绝大多数功能和第三方模块的配置都在这个模块

http块又分为了http全局块和server块

###### http全局块

主要配置的是文件的引入、MIME-TYPE定义、日志自定义、是否使用sendfile传输文件、连接超时时间、单链接请求上限等

http { #协议级别

# 文件扩展名与文件类型映射表

include mime.types;

# 默认文件类型

default\_type application/octet-stream;

# 在发送给客户端响应中允许禁用Http1.1标准的块传输编码

chunked\_transfer\_encoding

#默认编码

#charset utf-8;

#服务器名字的hash表大小

server\_names\_hash\_bucket\_size 128;

#请求头大小限制

client\_header\_buffer\_size 32k;

#设定请求缓冲，请求的头信息不能大于该值，否则会报Request URI too Large 414

large\_client\_header\_buffers 4 64k;

#上传文件大小限制

client\_max\_body\_size 8m;

# 开启目录列表访问,合适下载服务器,默认关闭.

# 显示目录

autoindex on;

# 显示文件大小 默认为on,显示出文件的确切大小,单位是bytes 改为off后,显示出文件的大概大小,单位是kB或者MB或者GB

autoindex\_exact\_size on;

# 显示文件时间 默认为off,显示的文件时间为GMT时间 改为on后,显示的文件时间为文件的服务器时间

autoindex\_localtime on;

#日志格式化 main是日志格式的名称

#log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '

# '$status $body\_bytes\_sent "$http\_referer" '

# '"$http\_user\_agent" "$http\_x\_forwarded\_for"';

# 日志位置 日志的格式使用main

# access\_log path [format]

# path 是文件存放的位置 format为可选，指定日志的格式，使用log\_format来进行格式的配置

#access\_log logs/access.log main;

# 配置日志是否开启 on|off

access\_log off;

# sendfile 高效传输文件的模式

sendfile on;

# 默认为0，每次调用sendfile()传输的数据量最大值，如果设置为0，则没有限制

sendfile\_max\_chunk 128k;

# 将文件一次性一起传输给客户端，提高网络包的传输效率,建议开启

#tcp\_nopush on;

# 开启实时传输，与tcp\_nopush相反，只在长连接下有效，此时nginx不缓存数据，一段一段的发送

#tcp\_nodelay on;

#连接超时时间

keepalive\_timeout 65;

# 用于限制用户通过某一连接向nginx服务器发送请求的次数

keepalive\_requests 100;

# FastCGI相关参数是为了改善网站的性能：减少资源占用,提高访问速度.下面参数看字面意思都能理解.

fastcgi\_connect\_timeout 300;

fastcgi\_send\_timeout 300;

fastcgi\_read\_timeout 300;

fastcgi\_buffer\_size 64k;

fastcgi\_buffers 4 64k;

fastcgi\_busy\_buffers\_size 128k;

fastcgi\_temp\_file\_write\_size 128k;

#gzip压缩

#gzip on;

#允许压缩的页面的最小字节数,页面字节数从header偷得content-length中获取.默认是0,不管页面多大都进行压缩.建议设置成大于1k的字节数,小于1k可能会越压越大

gzip\_min\_length 1k;

#表示申请4个单位为16k的内存作为压缩结果流缓存,默认值是申请与原始数据大小相同的内存空间来存储gzip压缩结果

gzip\_buffers 4 16k;

#压缩版本（默认1.1,目前大部分浏览器已经支持gzip解压.前端如果是squid2.5请使用1.0）

gzip\_http\_version 1.1;

#压缩等级.1压缩比最小,处理速度快.9压缩比最大,比较消耗cpu资源,处理速度最慢,但是因为压缩比最大,所以包最小,传输速度快

gzip\_comp\_level 2;

#压缩类型,默认就已经包含text/html,所以下面就不用再写了,写上去也不会有问题,但是会有一个warn.

gzip\_types text/plain application/x-javascript text/css application/xml;

#选项可以让前端的缓存服务器缓存经过gzip压缩的页面.例如:用squid缓存经过nginx压缩的数据

gzip\_vary on;

#实际的服务器列表

#upstream myserver {

# server 127.0.0.1:8080;

# server 127.0.0.1:8082;

#}

# 虚拟主机设置

server{

}

include servers/\*;

}

server块配置虚拟主机

一个http块可以包含多个server块，每个server块就相当于一台虚拟主机

server块主要是虚拟主机的配置，为了充分利用服务器资源，一台nginx服务器一般都同时配置N个虚拟域名主机，即多个域名对应同一个80端口，每台虚拟主机都可以是一个独立的网站

Nginx可以配置多种类型的虚拟主机

基于IP

基于域名

基于端口

又分为了server全局块和location块，在server全局块中常见的配置就是虚拟主机监听配置以及虚拟主机的名称

server { # 服务器级别

#监听的端口号 (可以配置监听 ip、端口)

listen 8080;

#虚拟主机的名称，访问域名

server\_name localhost;

#编码格式

#charset koi8-r;

#虚拟主机访问日志定义

#access\_log logs/host.access.log main;

# url匹配

location / { #请求级别

root html;

index index.html index.htm;

}

#error\_page 404 /404.html;

# redirect server error pages to the static page /50x.html

#

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root html;

}

# deny access to .htaccess files, if Apache's document root

# concurs with nginx's one

# 禁止访问.ht文件

#location ~ /\.ht {

# 禁止下载

# deny all;

#}

###### }

###### location块

每个server块可以包含多个location块，允许对不同的uri进行不同的配置，对特定的请求进行处理，如进行地址定向、数据缓存等功能

**匹配规则**

ocation有多种匹配规则[=|~|~\*|^~|@] /url/

没有修饰符表示必须以指定模式开头

# 匹配以/abc开头的

location /abc {

}

# 以/abc开头的

http://zhhll.icu/abc

http://zhhll.icu/abc?id=11

http://zhhll.icu/abc/

<http://zhhll.icu/abcd>

=表示精确匹配，优先级最高

# 仅匹配/abc，可以有参数

location = /abc {

}

# 可以匹配

http://zhhll.icu/abc

http://zhhll.icu/abc?id=11

# 不能匹配

http://zhhll.icu/abc/

<http://zhhll.icu/abcd>

* ^~ 类似于无修饰符，表示uri以某个字符串开头，即^~/spring/ 可以被所有以/spring/开头的匹配到
* ~为区分大小写的正则匹配，如果使用!~则表示取反，不匹配的正则

location ~ /abc {

}

# 可以匹配

http://zhhll.icu/abc

http://zhhll.icu/abc?id=11

# 不能匹配

http://zhhll.icu/Abc

http://zhhll.icu/abc/

http://zhhll.icu/abcd

~\*为不区分大小写的正则匹配，如果使用!~\*则表示取反，不匹配的正则   
location ~\* /abc {

}

# 可以匹配

http://zhhll.icu/abc

http://zhhll.icu/abc?id=11

http://zhhll.icu/Abc

# 不能匹配

http://zhhll.icu/abc/

http://zhhll.icu/abcd

* / 通用匹配，任何请求都会匹配到
* @ 客户端不能访问，只能由内部产生的请求来访问

静态文件路径

可以使用root或alias来指定静态资源文件路径，root可以配置在http块、server块、location块以及if块中，alias只能配置在location块中

root

使用root来表示请求的url时会进行完整的拼接，即如果请求使用的uri是/pic/index.html的话，使用root实际获取的是/data/www/web/pic/index.htm

location /pic/ {

root /data/www/web;

}  
**alias**alias会将location中配置的路径丢弃掉，即如果请求使用的uri是/pic/index.html的话，使用alias实际获取的是/data/www/web/index.html

location /pic/ {

alias /data/www/web;

}

Nginx常见配置参数

server\_names\_hash\_max\_size 1024;

server\_names\_hash\_bucket\_size 512;

client\_header\_buffer\_size 32k; # 客户端请求头部的缓冲区大小，有时候请求头过大，可以适时增大该配置

large\_client\_header\_buffers 4 32k;

client\_max\_body\_size 10m;#设置最大的允许客户端请求主体的大小(上传文件大小限制), 默认为1m，如果大于该设置，会返回413

client\_body\_buffer\_size 128k; 为了阻止临时文件写到磁盘，可以通过该指令为客户端请求体设置缓存大小

client\_body\_in\_single\_buffer 为了减少拷贝操作，该指令强制nginx将整个客户端请求体保存在单个缓存中

client\_body\_temp\_path 定义一个路径用于保存客户端请求体

keepalive\_timeout 60; #客户端连接超时时间，单位是秒, 默认是75秒

client\_header\_timeout #设置请求头的超时时间，可以设置低些，如10秒

client\_body\_timeout #设置请求体的超时时间，可以设置低些，如10秒

reset\_timeout\_connection #关闭不响应的客户端连接，这将会释放那个客户端所占有的内存空间，建议开启

send\_timeout #指定客户端的响应超时时间，如果在两次客户端读取操作之间，客户端没有读取任何数据，Nginx就会关闭连接，默认是60秒，可以设置小一些，如10秒

sendfile on; # 开启高效传输模式(开启或禁用sendfile函数),默认为off

tcp\_nopush on; #将文件一次性一起传输给客户端，提高网络包的传输效率,建议开启

tcp\_nodelay on; #此时nginx不缓存数据，一段一段的发送(允许或禁止TCP\_NODELAY)，仅适用于keep-alive连接

ssi on; # 开启ssi支持，默认为false

ssi\_silent\_errors on; # 设置为on表示在处理ssi文件时不输出错误信息，默认为false

ssi\_types text/html; # 默认支持html ,如果需要支持shtml(服务器执行脚本)，需要设置为ssi\_types text/shtml

server\_tokens off; # 关闭nginx版本号的显示，默认为on

Nginx优化相关参数

worker\_processes 2; # 配置生成的worker process数量，进程数，一般为cpu核数

worker\_rlimit\_nofile 65536; # 一个nginx进程打开的最多文件描述符数目，一般设置为与系统设定的值相同(ulimit -n)

worker\_cpu\_affinity 01 10;# 为每个进程分配CPU的工作内核

use epoll; # 使用epoll的IO模型，epoll是linux内核为处理大量文件描述符而做了改进的poll，它能显著提高程序在大量连接中只有少数活跃的情况下的CPU利用率

worker\_connections 1024 # 每个进程允许的最多连接数，理论上每台nginx服务器的最大连接数为worker\_processes\*worker\_connections

keepalive\_timeout 60 #keepalive超时时间，客户端到服务端的连接持续有效时间，当出现对服务器的后继请求时，可以避免重新建立连接

client\_header\_buffer\_size 4k 客户端请求头部的缓冲区大小，可以根据系统分页大小来设置，一般一个请求的头部大小不会超过1k

open\_file\_cache max=102400 inactive=20s 为打开文件指定缓存，默认没有启用，max指定缓存数量，建议和打开文件数一致，inactive是指经过多长时间文件没有请求后删除缓存

open\_file\_cache\_vaild 30s 这个是指多长时间检查一次缓存的有效信息

open\_file\_cache\_min\_uses 1 open\_file\_cache指令中的inactive参数时间内文件的最少使用次数，如果超过这个数字，文件描述符一直是在缓存中打开的，如上例，如果有一个文件在inactive时间内一次没被使用，将被移除

events {

use epoll; # IO模型

worker\_connections 20000; # 一个nginx进程的连接数，nginx服务器允许的同事连接的客户端最大数量Client = worker\_processes \* worker\_connections/2;

}

#### Nginx常见用法

##### 依据UA屏蔽爬虫

if ($http\_user\_agent ~\* "qihoobot|Baiduspider|Googlebot|Googlebot-Mobile|Googlebot-Image|Mediapartners-Google|Adsbot-Google|Feedfetcher-Google|Yahoo! Slurp|Yahoo! Slurp China|YoudaoBot|Sosospider|Sogou

spider|Sogou web spider|MSNBot|ia\_archiver|Tomato Bot") {

return 403;

}

##### 屏蔽IP访问

allow 133.27.182.82;

allow 113.106.18.0/24;

allow 121.201.104.0/24;

deny all;

##### 使用Auth权限访问

auth\_basic "bbs-auth";

auth\_basic\_user\_file /usr/local/nginx/conf/bbsauthpwd;

##### 限制带宽

##### auth\_basic "bbs-auth";

##### auth\_basic\_user\_file /usr/local/nginx/conf/bbsauthpwd;

##### 限制连接

# 定义了一个名为“down”，10M大小，以连接IP为key的连接数据存储空间

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

# 读取名为`down`连接数据存储空间的数据，限制每个key(上面是以ip作为IP) 最大同时连接数为4

location ~ .\*\.(rar|zip|apk)?$ {

limit\_conn down 4;

limit\_rate 150k;

}

limit\_conn\_log\_level notice: 指定当触发limit的时候日志打印级别

##### 限制请求

# 定义一个名为”one”, 10M大小，每秒1个请求的请求数据存储空间

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

# 引用名为“one”的存储空间，burst为等待请求数量数，当等待请求数量超过50个时，则抛出503错误,nodelay 针对的是 burst 参数，burst=50 nodelay 表示这50个请求立马处理，不能延迟，相当于特事特办。不过，即使这20个突发请求立马处理结束，后续来了请求也不会立马处理。burst=50 相当于缓存队列中占了50个坑，即使请求被处理了，这20个位置这只能按 100ms一个来释放

limit\_req zone=one burst=50 nodelay;

limit\_req\_log\_level notice: 指定当触发limit的时候日志打印级别

##### 实时显示Nginx运行状况

在安装nginx是编译http\_stub\_status\_module即可，使用参数为–with-http\_stub\_status\_module

##### 设置错误页面

error\_page 404 /404.html

##### 在负载均衡里面，实现后端服务器故障转移的配置

location ~ \.(jsp)$ {

proxy\_next\_upstream http\_502 http\_504 timeout;

proxy\_set\_header Host $host;

proxy\_set\_header X-Forwarded-For $remote\_addr;

proxy\_pass http://server\_pool;

}

Nginx 常见错误码

301 永久重定向

302 临时重定向

403 禁止访问

404 文件不存在

413 文件上传超过限制

500 服务器错误

502 后台服务器无响应

504 Nginx超时，请求过多，工作进程不足