链滴

nginx 配置文件详解以及调优

作者：iwang-peng
原文链接：https://ld246.com/article/1577439994880
来源网站：链滴

一、nginx的安装部署

(1)使用yum源直接安装

yum install nginx -y

(2)源码编译安装

nginx源码包下载官网地址：http://nginx.org/en/download.html

此处以新版本nginx-1.17.7.tar.gz为例，注意该版本不仅支持http七层代理，还支持tcp四层代理。

①下载源码包并解压包

wget http://nginx.org/download/nginx-1.17.7.tar.gz && tar zxvf nginx-1.17.7.tar.gz && cd nginx-1.17.7

②安装编译环境

yum -y install gcc pcre pcre-devel zlib zlib-devel openssl openssl-devel

③编译安装nginx

./configure --prefix=/usr/local/nginx-1.17 --with-http_ssl_module --with-http_stub_status_module --with-http_realip_module --with-threads

当然，如果需要其他模块，请自行添加其它模块参数,官网地址：http://nginx.org/en/docs/configur.html。

make && make install

二、配置文件优化

首先查看nginx配置文件的构成，如下：

```
...              #全局块

events {         #events块
   ...
}

http      #http块
{
    ...   #http全局块
    server        #server块
    {
        ...       #server全局块
        location [PATTERN]   #location块
        {
            ...
        }
        location [PATTERN]
        {
            ...
```

原文链接：nginx 配置文件详解以及调优

```
        }
    }
    server
    {
      ...
    }
    ...     #http全局块
}
```

(1)全局块配置调优

```
# nginx进程数，建议按照cpu数目来指定，一般跟cpu核数相同或为它的倍数。
worker_processes 8;

#为每个进程分配cpu，上例中将8个进程分配到8个cpu，当然可以写多个，或者将一个进程分配到多cpu。
worker_cpu_affinity 00000001 00000010 00000100 00001000 00010000 00100000 01000000 0000000;

#收到新连接通知后接受尽可能多的连接，作用于event
multi_accept on;

#Nginx最多可以打开文件数，与ulimit -n保持一致，如：worker_rlimit_nofile 65535;
worker_rlimit_nofile 65535;
```

(2)event段调优配置

```
events {

#作用于event的I/O多路复用模型
  use epoll;

#是单个worker进程允许客户端最大连接数，这个数值一般根据服务器性能和内存来制
  worker_connections 65535;

#收到新连接通知后接受尽可能多的连接，作用于event
  multi_accept on;
}
```

(3)http全局段调优

```
http {
include mime.types;
default_type application/octet-stream;
.....
sendfile on;
tcp_nopush on;
.....
```

sendfile ：开启高效文件传输模式，sendfile指令指定nginx是否调用sendfile函数来输出文件，于普通应用设为 on，如果用来进行下载等应用磁盘IO重负载应用，可设置为off，以平衡磁盘与网络I/O处理速度，降低系统的负载。
tcp_nopush：必须在sendfile开启模式才有效，防止网路阻塞，积极的减少网络报文段的数量
 tcp_nodelay:

- **连接超时时间**

主要目的是保护服务器资源，CPU，内存，控制连接数，因为建立连接也是需要消耗资源的，如：

```
keepalive_timeout 60;
keepalive_requests 10240;
tcp_nodelay on;
client_header_buffer_size 4k;
open_file_cache max=102400 inactive=20s;
open_file_cache_valid 30s;
open_file_cache_min_uses 1;
client_header_timeout 15;
client_body_timeout 15;
reset_timedout_connection on;
send_timeout 15;
server_tokens off;
client_max_body_size 500m;
proxy_connect_timeout 30s;
proxy_send_timeout 120s;
proxy_read_timeout 120s;
```

keepalived_timeout 60：客户端连接保持会话超时时间，超过这个时间，服务器断开这个链接

keepalive_requests 10240：参数限制了一个 HTTP 长连接最多可以处理完成的最大请求数, 默认是100。当连接处理完成的请求数达到最大请求数后，将关闭连接。

tcp_nodelay：也是防止网络阻塞，不过要包涵在keepalived参数才有效

client_header_buffer_size 4k：客户端请求头部的缓冲区大小，这个可以根据你的系统分页大小来置，一般一个请求头的大小不会超过 1k，不过由于一般系统分页都要大于1k，所以这里设置为分页小。分页大小可以用命令getconf PAGESIZE取得。

open_file_cache max=102400 inactive=20s：这个将为打开文件指定缓存，默认是没有启用的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被请求后删除缓存。

open_file_cache_valid 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 时内一次没被使用，它将被移除。

client_header_timeout：设置请求头的超时时间。我们也可以把这个设置低些，如果超过这个间没有发送任何数据，nginx将返回request time out的错误

client_body_timeout：设置请求体的超时时间。我们也可以把这个设置低些，超过这个时间没发送任何数据，和上面一样的错误提示

reset_timeout_connection：告诉nginx关闭不响应的客户端连接。这将会释放那个客户端所占的内存空间。

send_timeout：响应客户端超时时间，这个超时时间仅限于两个活动之间的时间，如果超过这个间，客户端没有任何活动，nginx关闭连接

server_tokens：并不会让nginx执行的速度更快，但它可以关闭在错误页面中的nginx版本数字这样对于安全性是有好处的。

client_max_body_size：上传文件大小限制

proxy_connect_timeout：与后端服务器建立连接的超时时间。注意这个一般不能大于75秒

proxy_read_timeout：从后端服务器读取响应的超时

- **gzip调优**

```
gzip on;
gzip_min_length 2k;
gzip_buffers    4 32k;
gzip_http_version 1.1;
gzip_comp_level 6;
gzip_typestext/plain text/css text/javascriptapplication/json application/javascript application
x-javascriptapplication/xml;
gzip_vary on;
gzip_proxied any;
gzip_disable "MSIE [1-6].";
```

gzip on：开启压缩功能

gzip_min_length 1k：设置允许压缩的页面最小字节数，页面字节数从header头的Content-Lengt中获取，默认值是0，不管页面多大都进行压缩，建议设置成大于1K，如果小与1K可能会越压越大。

gzip_buffers 4 32k：压缩缓冲区大小，表示申请4个单位为32K的内存作为压缩结果流缓存，默认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_http_version 1.1：压缩版本，用于设置识别HTTP协议版本，默认是1.1，目前大部分浏览器经支持GZIP解压，使用默认即可

gzip_comp_level 6：压缩比例，用来指定GZIP压缩比，1压缩比最小，处理速度最快，9压缩比最，传输速度快，但是处理慢，也比较消耗CPU资源。

gzip_types text/css text/xml application/javascript：用来指定压缩的类型，‘text/html’类总是会被压缩。默认值: gzip_types text/html (默认不对js/css文件进行压缩)

gzip_vary on：varyheader支持，改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Squid缓存经过nginx压缩的数据

gzip_disable：禁用IE6以下的gzip压缩，IE6的某些版本对gzip的压缩支持很不好

- **缓存设置**

```
#设置缓存空间，存储缓存文件
proxy_cache_path /data/nginx-cache levels=1:2 keys_zone=nginx-cache:20m max_size=50g i
active=168h;

#在location中使用缓存空间，pathname是项目的目录，请自定义
location /pathname {
    proxy_set_header X-Real-IP $remote_addr;
    proxy_cache nginx-cache;
    proxy_cache_valid 200 304 302 5d;
    proxy_cache_valid any 5d;
    proxy_cache_key '$host:$server_port$request_uri';
    add_header X-Cache '$upstream_cache_status from $host';
    proxy_set_header X-Real-IP $remote_addr;
    proxy_pass http://localhost/pathname;
}

#打开文件的缓存配置
#为打开文件指定缓存，默认是没有启用的，max 指定缓存数量，建议和打开文件数一致，inactive
指经过多长时间文件没被请求后删除缓存。
open_file_cache max=65535 inactive=60s;
```

```
#文件描述符一直是在缓存中打开的，如果有一个文件在inactive时间内一次没被使用，它将被移除。
open_file_cache_min_uses 1;

#指定多长时间检查一次缓存的有效信息。
open_file_cache_valid 80s;
```

- **访问限流调优**

```
#限制用户连接数来预防DOS攻击
limit_conn_zone $binary_remote_addr zone=perip:10m;
limit_conn_zone $server_name zone=perserver:10m;
#限制同一客户端ip最大并发连接数
limit_conn perip 2;
#限制同一server最大并发连接数
limit_conn perserver 20;
#限制下载速度，根据自身服务器带宽配置
limit_rate 300k;
```

- **正向代理**

```
server {
      listen 8080;
      resolver 8.8.8.8;
      resolver_timeout 5s;
      proxy_connect;
      proxy_connect_allow 443 563;
      proxy_connect_connect_timeout 10s;
      proxy_connect_read_timeout 10s;
      proxy_connect_send_timeout 10s;
      location / {
          proxy_pass $scheme://$host$request_uri;
          proxy_set_header Host $http_host;
          proxy_buffers 256 4k;
          proxy_max_temp_file_size 0;
          proxy_connect_timeout 30;
      }
      access_log /var/log/proxy/access.log main;
      error_log /var/log/proxy/error.log;
  }
```

配置终端代理

```
# 在 /etc/profile 文件中增加如下三项。
export proxy="http://{proxy_server_ip}:8080"
export http_proxy=$proxy
export https_proxy=$proxy

# 使配置生效
 source /etc/profile
```

- **反向代理**

```
location / {
        proxy_pass http://localhost:8088;
```

原文链接：nginx 配置文件详解以及调优

```
        proxy_redirect     off;
        proxy_set_header   Host            $host;
        proxy_set_header   X-Real-IP       $remote_addr;
        proxy_set_header   X-Forwarded-For  $proxy_add_x_forwarded_for;
    }
```

- **负载均衡**

```
upstream myproject{
  server 0.0.0.0:8082 fail_timeout=5 max_fails=3;
  server 0.0.0.0:8083 fail_timeout=5 max_fails=3;
  ip_hash;
}

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

- **https设置**

①下载证书

```
server {
    listen 80;
    server_name xxx.com;   #
    rewrite ^(.*)$ https://$host$1 permanent;   #把http的域名请求转成https


server {
    listen 443;
    server_name xxx.com; # 你的域名
    ssl on;
  #  root /var/www/bjubi.com; #前台文件存放文件夹，可改成别的
  #  index index.html index.htm;  #上面配置的文件夹里面的index.html
    ssl_certificate  cert/xxx.com.crt;   #改成你的证书的名字
    ssl_certificate_key cert/xxx.com.key;   #证书的名字
    ssl_session_timeout 5m;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE:ECDH:AES:HIGH:!NULL:!aNULL:!MD5:
ADH:!RC4;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers on;
    location / {
        proxy_pass http://127.0.0.1:8080;
                proxy_redirect          off;
                proxy_set_header        Host $host;
                proxy_set_header        X-Real-IP $remote_addr;
                proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

- **配置文件完整版，供参考**

```
worker_processes  8;
worker_cpu_affinity 00000001  00000010  00000100  00001000  00010000  00100000  010000
```

```
0  10000000;
 multi_accept on;
worker_rlimit_nofile 65535;

events {
    worker_connections  65535;
    multi_accept on;
    use epoll;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format main '{"@timestamp":"$time_iso8601",'
    '"host":"$server_addr",'
    '"clientip":"$remote_addr",'
    '"size":$body_bytes_sent,'
    '"responsetime":$request_time,'
    '"upstreamtime":"$upstream_response_time",'
    '"upstreamhost":"$upstream_addr",'
    '"http_host":"$host",'
    '"url":"$uri",'
    '"xff":"$http_x_forwarded_for",'
    '"referer":"$http_referer",'
    '"agent":"$http_user_agent",'
    '"status":"$status"}';

    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;

    keepalive_timeout 60;
    keepalive_requests 10240;
    tcp_nodelay on;
    client_header_buffer_size 4k;
   open_file_cache max=102400 inactive=20s;
   open_file_cache_valid 30s;
   open_file_cache_min_uses 1;
   client_header_timeout 15;
   client_body_timeout 15;
   reset_timedout_connection on;
   send_timeout 15;
   server_tokens off;
   client_max_body_size 500m;
   proxy_connect_timeout 30s;
   proxy_send_timeout 120s;
   proxy_read_timeout 120s;

    limit_conn_zone $binary_remote_addr zone=perip:10m;
    limit_conn_zone $server_name zone=perserver:10m;
    limit_conn perip 2;
    limit_conn perserver 20;
    limit_rate 300k;
```

```
    proxy_cache_path /data/nginx-cache levels=1:2 keys_zone=nginx-cache:20m max_size=50g
inactive=168h;

    gzip on;
    gzip_min_length 1k;
    gzip_buffers 4 16k;
    gzip_http_version 1.1;
    gzip_comp_level 4;
    gzip_types text/plain text/css application/json application/x-javascript text/xml application/
ml application/xml+rss text/javascript;
    gzip_vary on;
    gzip_disable "MSIE [1-6].";

    include /etc/nginx/conf.d/*.conf;
}


server {
    listen 443;
    server_name www.xxx.com;
    ssl on;
    ssl_certificate cert/证书名称.pem;
    ssl_certificate_key cert/证书名称.key;
    ssl_session_timeout 5m;
    # SSL协议配置
    ssl_protocols SSLv2 SSLv3 TLSv1.2;
    ssl_ciphers ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
    ssl_prefer_server_ciphers on;

    valid_referers none blocked server_names
            *.ygoclub.com;

    #日志配置
    access_log  /var/log/nginx/access.log  main gzip=4 flush=5m;
    error_log  /var/log/nginx/error.log  error;

    location ~ .*\.(eot|svg|ttf|woff|jpg|jpeg|gif|png|ico|cur|gz|svgz|mp4|ogg|ogv|webm) {
        proxy_cache nginx-cache;
        proxy_cache_valid 200 304 302 5d;
        proxy_cache_key '$host:$server_port$request_uri';
        add_header X-Cache '$upstream_cache_status from $host';
        #所有静态文件直接读取硬盘
        root /usr/share/nginx/html;
        expires 30d; #缓存30天
    }

    location ~ .*\.(js|css)?$
    {
        proxy_cache nginx-cache;
        proxy_cache_valid 200 304 302 5d;
        proxy_cache_key '$host:$server_port$request_uri';
        add_header X-Cache '$upstream_cache_status from $host';
        #所有静态文件直接读取硬盘
```

```
        root /usr/share/nginx/html;
        expires      12h;
    }

    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }

    location /druid {
        proxy_set_header X-Real-IP $remote_addr;
        proxy_cache nginx-cache;
        proxy_cache_valid 200 10m;
        proxy_pass http://xxxx/druid;
    }

    location /api {
       proxy_set_header X-Real-IP $remote_addr;
       proxy_pass http://xxxx/api;
    }

    # redirect server error pages to the static page /50x.html
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }
}
```

- **后续**

当然，nginx调优好了，如果系统不进行优化，照样也是无用，以下对linux系统进行tcp方面的优化。

```
cat /etc/sysctl.conf
net.ipv4.tcp_max_syn_backlog = 65536
net.core.netdev_max_backlog =  36768
net.core.somaxconn = 36768
net.ipv4.tcp_max_tw_buckets = 5000
net.ipv4.tcp_keepalive_time = 30

net.core.wmem_default = 8588608
net.core.rmem_default = 8588608
net.core.rmem_max = 16877216
net.core.wmem_max = 16877216


net.ipv4.tcp_mem = 786432 2097152 3145728
net.ipv4.tcp_rmem = 32768 436600 16777216
net.ipv4.tcp_wmem = 8192 436600 16777216
net.ipv4.tcp_max_orphans = 3376800
net.ipv4.ip_local_port_range = 1024  65535


 sysctl -p

 cat /etc/security/limit.conf
```

```
* hard nofile 65535
* soft nofile 65535
```

net.ipv4.tcp_max_syn_backlog：表示SYN队列的长度，默认为1024，加大队列长度为8192，可容纳更多等待连接的网络连接数。

net.core.netdev_max_backlog：表示当每个网络接口接收数据包的速率比内核处理这些包的速率时，允许发送到队列的数据包的最大数目，一般默认值128（可能不同的linux系统该数值也不同）。nginx服务器中定义的NGX_LISTEN_BACKLOG默认为511

net.core.somaxconn：表示socket监听（listen）的backlog上限

** net.ipv4.tcp_max_tw_buckets = 5000**：如果想把timewait降下了就要把tcp_max_tw_bucket值减小，默认是180000

net.ipv4.tcp_keepalive_time：当keepalive起用的时候，TCP发送keepalive消息的频度。缺省是2时。我们可以调短时间跨度

**net.core.wmem_default **：默认的TCP数据发送窗口大小（字节）。

net.core.rmem_default：默认的TCP数据接收窗口大小（字节）。

**net.core.rmem_max **：最大的TCP数据接收窗口（字节）。

net.core.wmem_max：最大的TCP数据发送窗口（字节）。

**net.ipv4.tcp_tw_reuse **：开启重用功能，允许将TIME-WAIT状态的sockets重新用于新的TCP连接

net.ipv4.tcp_mem：意思是:net.ipv4.tcp_mem[0]:低于此值,TCP没有内存压力.net.ipv4.tcp_mem[1:在此值下,进入内存压力阶段.net.ipv4.tcp_mem[2]:高于此值,TCP拒绝分配socket.上述内存单位是页,不是字节

net.ipv4.tcp_wmem：TCP写buffer

net.ipv4.tcp_rmem：TCP读buffer

net.ipv4.tcp_max_orphans：系统中最多有多少个TCP套接字不被关联到任何一个用户文件句柄上如果超过这个数字，孤儿连接将即刻被复位并打印出警告信息。这个限制仅仅是为了防止简单的DoS击