Nginx配置与性能调优

主讲老师： fox老师

Nginx常规配置

location匹配

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

```
/ 基于uri目录匹配
=表示把URI作为字符串，以便与参数中的uri做完全匹配
～表示正则匹配URI时是字母大小写敏感的
～*表示正则匹配URI时忽略字母大小写问题
^～表示正则匹配URI时只需要其前半部分与uri参数匹配即可
```

```
location = /baidu.html {
    proxy_pass http://www.baidu.com;
}

location /static {
    #root的处理结果是：root路径＋location路径
    #root /usr/www/;
    #alias的处理结果是：使用alias路径替换location路径
    alias /usr/www/static/;
    index index.html;
}

#下载限速
location /download {
    alias /usr/www/download/;
    # 限速2m
    limit_rate 2m;
    # 30m之后限速
    limit_rate_after 30m;
}

location ~* \.(gif|png|jpg|css|js)$ {
   root /usr/www/static/;
}
```

日志配置

```
log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
access_log  logs/$host.access.log  main;
```

rewrite重定向

指令语法：rewrite regex replacement[flag];

应用位置：server、location、if

rewrite是实现URL重定向的重要指令，他根据regex(正则表达式)来匹配内容跳转到replacement，结尾是flag标记

```
 location / {
        rewrite ^/ http://www.baidu.com;
  }
```

```
location / {
            rewrite '^/images/(.*)\.(png|jpg)$' /fox?file=$1.$2;
            set $image_file $1;
            set $image_type $2;
        }

        location /fox {
           root html;
           # 按顺序检查文件是否存在，返回第一个找到的文件.
           #如果所有的文件都找不到，会进行一个内部重定向到最后一个参数
           try_files /$arg_file /image404.html;
        }

        location /image404.html {
            return 404 "image not found exception";
        }
```

gzip压缩策略

浏览器请求会告诉服务端当前浏览器支持的压缩类型，服务端会将数据根据浏览器支持的压缩类型进行压缩返回

对静态资源进行压缩

```
# 开启gzip
gzip on;
# 开始压缩的最小长度
gzip_min_length 1000;  #1k
# 压缩级别(级别越高,压的越小,越浪费CPU计算资源)
gzip_comp_level 5; # 1-9
# 对哪些类型的文件用压缩 如txt,xml,html ,css
# jpg,png本身就是压缩格式，不建议用gzip
gzip_types      text/plain application/json application/x-javascript
application/css application/xml application/xml+rss text/javascript;

location /static {
    alias /usr/local/apps/static/dlib-19.18/;
    # 显示目录
    autoindex on;
}
```

反向代理

正向代理是指客户端与目标服务器之间增加一个代理服务器，客户端直接访问代理服务器，在由代理服务器访问目标服务器并返回客户端并返回 。这个过程当中客户端需要知道代理服务器地址，并配置连接。

```
# 正向代理
location = /baidu.html {
    proxy_pass http://www.baidu.com;
}
```

反向代理是指客户端访问目标服务器，在目标服务内部有一个统一接入网关将请求转发至后端真正处理的服务器并返回结果。这个过程当中客户端不需要知道代理服务器地址，代理对客户端而言是透明的。

```
location = /fox {
    proxy_pass http://127.0.0.1:8080/;
}
```

代理相关参数

```
proxy_pass            # 代理服务
proxy_redirect off;   # 是否允许重定向
proxy_set_header Host $host; # 传 header 参数至后端服务
proxy_set_header X-Forwarded-For $remote_addr; # 设置request header 即客户端IP地址
proxy_connect_timeout 90; # 连接代理服务超时时间
proxy_send_timeout 90; # 请求发送最大时间
proxy_read_timeout 90;  # 读取最大时间
proxy_buffer_size 4k;
proxy_buffers 4 32k;
proxy_busy_buffers_size 64k;
proxy_temp_file_write_size 64k;
```

负载均衡

```
upstream backend {
    ip_hash;
    server 127.0.0.1:8088 weight=1;
    server 127.0.0.1:8080 weight=1;
    #server 127.0.0.1:9080 backup;

}

server{

    listen       80;
    server_name  localhost;

    location / {
        proxy_set_header Host   $host;
        proxy_set_header X-Real-IP  $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        # 反向代理
        proxy_pass http://backend/;
    }
```

```
}
```

upstream 相关参数:

```
service 反向服务地址 加端口
weight  权重
max_fails   失败多少次 认为主机已挂掉则，踢出
fail_timeout    踢出后重新探测时间
backup  备用服务
max_conns   允许最大连接数
slow_start  当节点恢复，不立即加入,而是等待 slow_start 后加入服务对列
```

负载均衡算法:

```
list 找一个
轮询 + 权重
随机  + 权重
ip_hash
url_hash
一致性hash
最小活跃数
最短响应时间
```

1、轮询（默认） 每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，能自动剔除。

```
upstream backserver {
 server 192.168.0.14;
 server 192.168.0.15;
}
```

2、指定权重 指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。

```
upstream backserver {
 server 192.168.0.14 weight=8;
 server 192.168.0.15 weight=10;
}
```

3、IP绑定 ip_hash 每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。

```
upstream backserver {
 ip_hash;
 server 192.168.0.14:8888;
 server 192.168.0.15:8080;
}
```

4、least_conn

把请求转发给连接数较少的后端服务器。轮询算法是把请求平均的转发给各个后端，使它们的负载大致相同；但是，有些请求占用的时间很长，会导致其所在的后端负载较高。这种情况下，least_conn这种方式就可以达到更好的负载均衡效果。

```
upstream backserver {
        least_conn;    #把请求转发给连接数较少的后端服务器
        server localhost:8080   weight=2;
        server localhost:8081;
        server localhost:8082 backup;
        server localhost:8083   max_fails=3 fail_timeout=20s;
    }
```

5、fair（第三方） 按后端服务器的响应时间来分配请求，响应时间短的优先分配。

```
upstream backserver {
 server server1;
 server server2;
 fair;
}
```

6、url_hash（第三方） 按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，后端服务器为缓存时比较有效。

```
upstream backserver {
 server squid1:3128;
 server squid2:3128;
 hash $request_uri;
 hash_method crc32;
}
```

代理缓存

代理缓存,获取服务器端内容进行缓存

http://nginx.org/en/docs/http/ngx_http_proxy_module.html

```
#proxy_cache_path 缓存路径
#levels 缓存层级及目录位数
#keys_zone 缓存区内存大小
#inactive 有效期
proxy_cache_path /tmp/nginx/cache levels=1:2 keys_zone=my_cache:10m max_size=10g
                        inactive=60m use_temp_path=off;

server {

        listen 80;
        server_name  localhost *.yuanma.com;

        location / {
          proxy_set_header Host   $host;
          proxy_set_header X-Real-IP  $remote_addr;
          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

          proxy_pass http://backend/;

          proxy_cache my_cache;
          #以全路径md5值做做为Key
          proxy_cache_key $host$uri$is_args$args;
```

```
            proxy_cache_valid 200 304 302 1d;
    }
    }
```

缓存参数详细说明：

父元素	名称	描述
http	proxy_cache_path	指定缓存区的根路径
	levels	缓存目录层级最高三层，每层1~2个字符表示。如1:1:2 表示三层。
	keys_zone	缓存块名称 及内存块大小。如 cache_item:500m 。表示声明一个名为cache_item 大小为500m。超出大小后最早的数据将会被清除。
	inactive	最长闲置时间 如:10d 如果一个数据被闲置10天将会被清除
	max_size	缓存区硬盘最大值。超出闲置数据将会被清除
location	proxy_cache	指定缓存区，对应keys_zone 中设置的值
	proxy_cache_key	通过参数拼装缓存key 如：$host$uriis_argsargs 则会以全路径md5值做做为Key
	proxy_cache_valid	为不同的状态码设置缓存有效期

缓存清除， 添加ngx_cache_purge模块

```
#下载ngx_cache_purge 模块包
wget http://labs.frickle.com/files/ngx_cache_purge-2.3.tar.gz
#查看已安装模块
./sbin/nginx -V
#进入nginx安装包目录 重新构建 --add-module为模块解压的全路径
./configure --prefix=/usr/local/nginx --with-http_stub_status_module --with-
http_ssl_module --with-debug --add-module=/root/ngx_cache_purge-2.3
#重新编译
make
#热升级
```

清除缓存

```
location ~ /clear(/.*) {
  #允许访问的IP
   allow           192.168.3.1;
   #禁止访问的IP
   deny            all;
   #配置清除指定缓存区和路径(与proxy_cache_key一致)
   proxy_cache_purge    my_cache $host$1$is_args$args;
}
```

HTTPS配置

生成自签名证书

https://www.cnblogs.com/hnxxcxg/p/7610582.html

```
# 使用openssl工具生成一个RSA私钥
openssl genrsa -des3 -out server.key 1024
# 生成CSR（证书签名请求）
 openssl req -new -key server.key -out server.csr
# 删除私钥中的密码
openssl rsa -in server.key -out server.key
# 生成自签名证书
openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

配置https服务： http://nginx.org/en/docs/http/configuring_https_servers.html

```
server {
    listen              443 ssl;
    server_name         fox.yuanma.com;
    ssl_certificate     server.crt;
    ssl_certificate_key server.key;

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

}
```

tomcat 配置，可以不用配置

```
<Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="443" proxyPort="443"  />
```

Nginx性能调优

1.nginx默认是没有开启利用多核cpu的配置的。需要通过增加worker_cpu_affinity配置参数来充分利用多核cpu。

通过绑定Nginx worker进程到指定的CPU内核，不会出现多个worker进程都在抢同一个CPU的情况。

```
#2核cpu，开启2个进程
worker_processes     2;
worker_cpu_affinity 01 10;

#4个cpu，开启4个进程
worker_processes     4;
worker_cpu_affinity 0001 0010 0100 1000;
```

2.配置worker进程最大打开文件数， 避免nginx出现“too many open files”错误

```
worker_rlimit_nofile 65535;
```

3.设置io模型， select,poll,epoll(linux),kqueue(unix) 。

设置并发连接数 worker_connections ， 默认1024

是否打开accept锁 语法：accept_mutex[on|off] 默认：accept_mutex on;

accept_mutex是Nginx的负载均衡锁，当某一个worker进程建立的连接数量达到worker_connections配置的最大连接数的7/8时，会大大地减小该worker进程试图建立新TCP连接的机会，accept锁默认是打开的，如果关闭它，那么建立TCP连接的耗时会更短，但worker进程之间的负载会非常不均衡，因此不建议关闭它。

使用accept锁后到真正建立连接之间的延迟时间 语法：accept_mutex_delay Nms; 默认：accept_mutex_delay 500ms;

```
events{
    use epoll;
    worker_connections  10240;
    accept_mutext on;
    accept_mutex_delay 500ms;
}
```