Nginx教程

> 作者：康哥

> 微信公众号：码上有猿

Windows：

```
1.conf目录：存放配置文件的目录，包含主配置文件nginx.conf，是我们经常修改的配置文件。
2.contrib目录：存放开源爱好者共享的代码。
3.docs目录：存放文档资料。
4.html目录：默认存放了Nginx的错误页面和欢迎页面。
5.logs目录：默认存放了访问日志、错误日志和Nginx主进程pid文件。
6.temp目录：临时目录，用于存放Nginx运行时产生的临时文件。
7.nginx.exe:可执行程序，常用于Nginx服务的启动、停止等管理工作。
```

linux：

```
1.*_temp目录：共有5个temp结尾的目录，用于存放Nginx运行时产生的临时文件。
2.conf目录：存放配置文件的目录，包含主配置文件nginx.conf，是我们经常修改的配置文件。
3.html目录：默认存放了Nginx的错误页面和欢迎页面等。
4.logs目录：默认存放了访问日志和错误日志文件。
5.sbin目录：默认存放了Nginx的二进制命令，常用于Nginx服务的启动、停止等管理工作。
```

配置文件详解

```
#定义Nginx运行的用户和用户组
# user nobady nobady;

#nginx进程数，建议设置为等于CPU总核心数,默认为1。
worker_processes 8;

#全局错误日志定义类型，[ debug | info | notice | warn | error | crit ]
error_log /usr/local/nginx/logs/error.log info;

#进程pid文件,指定nginx进程运行文件存放地址
pid /usr/local/nginx/logs/nginx.pid;

#指定进程可以打开的最大描述符：数目
#工作模式与连接数上限
#这个指令是指当一个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（ulimit -n）与nginx进程数相除，但是nginx分配请求并不是那么均匀，所以最好与ulimit -n的值保持一致。
#现在在linux 2.6内核下开启文件打开数为65535，worker_rlimit_nofile就相应应该填写65535。
#这是因为nginx调度时分配请求到进程并不是那么的均衡，所以假如填写10240，总并发量达到3-4万时就有进程可能超过10240了，这时会返回502错误。
```

```
worker_rlimit_nofile 65535;


events
{
    #参考事件模型，use [ kqueue | rtsig | epoll | /dev/poll | select | poll ]; epoll模型
    #是Linux 2.6以上版本内核中的高性能网络I/O模型，linux建议epoll，如果跑在FreeBSD上面，就用kqueue模型。
    #补充说明：
    #与apache相类，nginx针对不同的操作系统，有不同的事件模型
    #A）标准事件模型
    #Select、poll属于标准事件模型，如果当前系统不存在更有效的方法，nginx会选择select或poll
    #B）高效事件模型
    #Kqueue：使用于FreeBSD 4.1+, OpenBSD 2.9+, NetBSD 2.0 和 MacOS X.使用双处理器的MacOS X系统使用kqueue可能会造成内核崩溃。
    #Epoll：使用于Linux内核2.6版本及以后的系统。
    #/dev/poll：使用于Solaris 7 11/99+，HP/UX 11.22+ (eventport)，IRIX 6.5.15+ 和 Tru64 UNIX 5.1A+。
    #Eventport：使用于Solaris 10。 为了防止出现内核崩溃的问题， 有必要安装安全补丁。
    use epoll;

    #单个进程最大连接数（最大连接数=连接数*进程数）
    #根据硬件调整，和前面工作进程配合起来用，尽量大，但是别把cpu跑到100%就行。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_connections 65535;

    #keepalive超时时间，默认是60s，切记这个参数也不能设置过大！否则会导致许多无效的http连接占据着nginx的连接数，终nginx崩溃！
    keepalive_timeout 60;

    #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设置，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大小。
    #分页大小可以用命令getconf PAGESIZE 取得。
    #[root@web001 ~]# getconf PAGESIZE
    #4096
    #但也有client_header_buffer_size超过4k的情况，但是client_header_buffer_size该值必须设置为“系统分页大小”的整倍数。
    client_header_buffer_size 4k;

    #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 max=65535 inactive=60s;

    #这个是指多长时间检查一次缓存的有效信息。
    #语法:open_file_cache_valid time 默认值:open_file_cache_valid 60 使用字段:http, server, location 这个指令指定了何时需要检查open_file_cache中缓存项目的有效信息.
    open_file_cache_valid 60s;

    #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，它将被移除。
    #语法:open_file_cache_min_uses number 默认值:open_file_cache_min_uses 1 使用字段:http, server, location  这个指令指定了在open_file_cache指令无效的参数中一定的时间范围内可以使用的最小文件数,如果使用更大的值,文件描述符在cache中总是打开状态.
    open_file_cache_min_uses 1;
```

```
60
61      #语法:open_file_cache_errors on | off 默认值:open_file_cache_errors off 使用字段:http, server, location 这个指令指定是否在搜索一个文件是记录cache错误.
62      open_file_cache_errors on;
63  }
64
65
66
67  #设定http服务器，利用它的反向代理功能提供负载均衡支持
68  http
69  {
70      #文件扩展名与文件类型映射表
71      include mime.types;
72
73      #默认文件类型
74      default_type application/octet-stream;
75
76      #默认编码
77      #charset utf-8;
78
79      #服务器名字的hash表大小
80      #保存服务器名字的hash表是由指令server_names_hash_max_size 和 server_names_hash_bucket_size所控制的。参数hash bucket size总是等于hash表的大小，并且是一路处理器缓存大小的倍数。在减少了在内存中的存取次数后，使在处理器中加速查找hash表键值成为可能。如果hash bucket size等于一路处理器缓存的大小，那么在查找键的时候，最坏的情况下在内存中查找的次数为2。第一次是确定存储单元的地址，第二次是在存储单元中查找键 值。因此，如果Nginx给出需要增大hash max size 或 hash bucket size的提示，那么首要的是增大前一个参数的大小.
81      server_names_hash_bucket_size 128;
82
83      #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设置，一般一个请求的头部大小不会超过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大小。分页大小可以用命令getconf PAGESIZE取得。
84      client_header_buffer_size 32k;
85
86      #客户请求头缓冲大小。nginx默认会用client_header_buffer_size这个buffer来读取header值，如果header过大，它会使用large_client_header_buffers来读取。
87      large_client_header_buffers 4 64k;
88
89      #设定通过nginx上传文件的大小
90      client_max_body_size 8m;
91
92      #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，对于普通应用设为 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。注意：如果图片显示不正常把这个改成off。
93      #sendfile指令指定 nginx 是否调用sendfile 函数（zero copy 方式）来输出文件，对于普通应用，必须设为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IO处理速度，降低系统uptime。
94      sendfile on;
95
96      #开启目录列表访问，合适下载服务器，默认关闭。
97      autoindex on;
98
99      #此选项允许或禁止使用socke的TCP_CORK的选项，此选项仅在使用sendfile的时候使用,告诉nginx在一个数据包里发送所有头文件，而不一个接一个的发送。就是说数据包不会马上传送出去，等到数据包最大时，一次性的传输出去，这样有助于解决网络堵塞
100     tcp_nopush on;
```

```
    #告诉nginx不要缓存数据，而是一段一段的发送--当需要及时发送数据时，就应该给应用设置这个属性，这样发送一小块数据信息时就不能立即得到返回值
    tcp_nodelay on;

    #长连接超时时间，单位是秒
    keepalive_timeout 120;

    #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。
    #这个指令为FastCGI缓存指定一个路径，目录结构等级，关键字区域存储时间和非活动删除时间
    fastcgi_cache_path /usr/local/nginx/fastcgi_cache levels=1:2 keys_zone=TEST:10m inactive=5m;
    #指定连接到后端FastCGI的超时时间
    fastcgi_connect_timeout 300;
    #向FastCGI传送请求的超时时间，这个值是指已经完成两次握手后向FastCGI传送请求的超时时间
    fastcgi_send_timeout 300;
    #接收FastCGI应答的超时时间，这个值是指已经完成两次握手后接收FastCGI应答的超时时间
    fastcgi_read_timeout 300;
    #指定读取FastCGI应答第一部分 需要用多大的缓冲区,这里可以设置为fastcgi_buffers指令指定的缓冲区大小，上面的指令指定它将使用1个 16k的缓冲区去读取应答的第一部分，即应答头，其实这个应答头一般情况下都很小（不会超过1k），但是你如果在fastcgi_buffers指令中指定了缓冲区的大小，那么它也会分配一个fastcgi_buffers指定的缓冲区大小去缓存
    fastcgi_buffer_size 64k;
    #指定本地需要用多少和多大的缓冲区来 缓冲FastCGI的应答，如上所示，如果一个php脚本所产生的页面大小为256k，则会为其分配16个16k的缓冲区来缓存，如果大于256k，增大 于256k的部分会缓存到fastcgi_temp指定的路径中， 当然这对服务器负载来说是不明智的方案，因为内存中处理数据速度要快于硬盘，通常这个值 的设置应该选择一个你的站点中的php脚本所产生的页面大小的中间值，比如你的站点大部分脚本所产生的页面大小为 256k就可以把这个值设置为16 16k，或者4 64k 或者64 4k，但很显然，后两种并不是好的设置方法，因为如果产生的页面只有32k，如果用4 64k它会分配1个64k的缓冲区去缓存，而如果使用64 4k它会分配8个4k的缓冲区去缓存，而如果使用16 16k则它会分配2个16k去缓存页面，这样看起来似乎更加合理•
    fastcgi_buffers 4 64k;
    #这个指令我也不知道是做什么用，只知道默认值是fastcgi_buffers的两倍
    fastcgi_busy_buffers_size 128k;
    #在写入fastcgi_temp_path时将用多大的数据块，默认值是fastcgi_buffers的两倍
    fastcgi_temp_file_write_size 128k;
    #开启FastCGI缓存并且为其制定一个名称。个人感觉开启缓存非常有用，可以有效降低CPU负载，并且防止502错误。但是这个缓存会引起很多问题，因为它缓存的是动态页面。具体使用还需根据自己的需求
    fastcgi_cache TEST
    #为指定的应答代码指定缓存时间，如上例中将200，302应答缓存一小时，301应答缓存1天，其他为1分钟
    fastcgi_cache_valid 200 302 1h;
    fastcgi_cache_valid 301 1d;
    fastcgi_cache_valid any 1m;

    #缓存在fastcgi_cache_path指令inactive参数值时间内的最少使用次数，如上例，如果在5分钟内某文件1次也没有被使用，那么这个文件将被移除
    fastcgi_cache_min_uses 1;


    #gzip模块设置
    #开启压缩
    gzip  on;
    # 设置允许压缩的页面最小字节数，页面字节数从header头得content-length中进行获取。默认值是0，不管页面多大都压缩。建议设置成大于2k的字节数，小于2k可能会越压越大。
```

```
    gzip_min_length 2k;
    # 设置系统获取几个单位的缓存用于存储gzip的压缩结果数据流。 例如 4 4k 代表以4k为单位，按照原始数据大小以4k为单位的4倍申请内存。 4 8k 代表以8k为单位，按照原始数据大小以8k为单位的4倍申请内存。
    # 如果没有设置，默认值是申请跟原始数据相同大小的内存空间去存储gzip压缩结果。
    gzip_buffers 4 16k;
    #压缩级别，1-10，数字越大压缩的越好，也越占用CPU时间
    gzip_comp_level 5;
    # 默认值: gzip_types text/html (默认不对js/css文件进行压缩)
    # 压缩类型，匹配MIME类型进行压缩
    # 不能用通配符 text/*
    # (无论是否指定)text/html默认已经压缩
    # 设置哪压缩种文本文件可参考 conf/mime.types
    gzip_types text/plain application/x-javascript text/css application/xml;
    # 值为1.0和1.1 代表是否压缩http协议1.0，选择1.0则1.0和1.1都可以压缩
    gzip_http_version 1.0
    # IE6及以下禁止压缩
    gzip_disable "MSIE [1-6]\.";
    # 默认值：off
    # Nginx作为反向代理的时候启用，开启或者关闭后端服务器返回的结果，匹配的前提是后端服务器必须要返回包含"Via"的 header头。
    # off - 关闭所有的代理结果数据的压缩
    # expired - 启用压缩，如果header头中包含 "Expires" 头信息
    # no-cache - 启用压缩，如果header头中包含 "Cache-Control:no-cache" 头信息
    # no-store - 启用压缩，如果header头中包含 "Cache-Control:no-store" 头信息
    # private - 启用压缩，如果header头中包含 "Cache-Control:private" 头信息
    # no_last_modified - 启用压缩,如果header头中不包含 "Last-Modified" 头信息
    # no_etag - 启用压缩 ,如果header头中不包含 "ETag" 头信息
    # auth - 启用压缩 , 如果header头中包含 "Authorization" 头信息
    # any - 无条件启用压缩
    gzip_proxied expired no-cache no-store private auth;
    # 给CDN和代理服务器使用，针对相同url，可以根据头信息返回压缩和非压缩副本
    gzip_vary on;



    #开启限制IP连接数的时候需要使用
    #limit_zone crawler $binary_remote_addr 10m;



    #负载均衡配置
    upstream www.xx.com {

        #upstream的负载均衡，weight是权重，可以根据机器配置定义权重。weigth参数表示权值，权值越高被分配到的几率越大。
        server 192.168.80.121:80 weight=3;
        server 192.168.80.122:80 weight=2;
        server 192.168.80.123:80 weight=3;

        #nginx的upstream目前支持4种方式的分配
        #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。
        #2、weight
        #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。
        #例如：
```

```
        #upstream bakend {
        #    server 192.168.0.14 weight=10;
        #    server 192.168.0.15 weight=10;
        #}
        #2、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。
        #例如：
        #upstream bakend {
        #    ip_hash;
        #    server 192.168.0.14:88;
        #    server 192.168.0.15:80;
        #}
        #3、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 backend {
        #    server server1;
        #    server server2;
        #    fair;
        #}
        #4、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。
        #例：在upstream中加入hash语句，server语句中不能写入weight等其他的参数，hash_method是使用的hash算法
        #upstream backend {
        #    server squid1:3128;
        #    server squid2:3128;
        #    hash $request_uri;
        #    hash_method crc32;
        #}

        #tips:
        #upstream bakend{#定义负载均衡设备的Ip及设备状态}{
        #    ip_hash;
        #    server 127.0.0.1:9090 down;
        #    server 127.0.0.1:8080 weight=2;
        #    server 127.0.0.1:6060;
        #    server 127.0.0.1:7070 backup;
        #}
        #在需要使用负载均衡的server中增加 proxy_pass http://bakend/;

        #每个设备的状态设置为:
        #1.down表示单前的server暂时不参与负载
        #2.weight为weight越大，负载的权重就越大。
        #3.max_fails：允许请求失败的次数默认为1.当超过最大次数时，返回proxy_next_upstream模块定义的错误
        #4.fail_timeout:max_fails次失败后，暂停的时间。
        #5.backup：  其它所有的非backup机器down或者忙的时候，请求backup机器。所以这台机器压力会最轻。

        #nginx支持同时设置多组的负载均衡，用来给不用的server来使用。
        #client_body_in_file_only设置为On 可以讲client post过来的数据记录到文件中用来做debug
        #client_body_temp_path设置记录文件的目录 可以设置最多3层目录
        #location对URL进行匹配.可以进行重定向或者进行新的代理 负载均衡
    }

```

```
243
244
245     #虚拟主机的配置
246     server
247     {
248         #监听端口
249         listen 80;
250
251         #域名可以有多个，用空格隔开
252         server_name www.xx.com xx.com;
253         index index.html index.htm index.php;
254         root /data/www/xx;
255
256         #对******进行负载均衡
257         location ~ .*.(php|php5)?$
258         {
259             fastcgi_pass 127.0.0.1:9000;
260             fastcgi_index index.php;
261             include fastcgi.conf;
262         }
263
264         #图片缓存时间设置
265         location ~ .*.(gif|jpg|jpeg|png|bmp|swf)$
266         {
267             expires 10d;
268         }
269
270         #JS和CSS缓存时间设置
271         location ~ .*.(js|css)?$
272         {
273             expires 1h;
274         }
275
276         #日志格式设定
277         #$remote_addr与$http_x_forwarded_for用以记录客户端的ip地址；
278         #$remote_user：用来记录客户端用户名称；
279         #$time_local： 用来记录访问时间与时区；
280         #$request： 用来记录请求的url与http协议；
281         #$status： 用来记录请求状态；成功是200，
282         #$body_bytes_sent ：记录发送给客户端文件主体内容大小；
283         #$http_referer：用来记录从那个页面链接访问过来的；
284         #$http_user_agent：记录客户浏览器的相关信息；
285         #通常web服务器放在反向代理的后面，这样就不能获取到客户的IP地址了，通过
    $remote_add拿到的IP地址是反向代理服务器的iP地址。
286         #反向代理服务器在转发请求的http头信息中，可以增加x_forwarded_for信息，用以记
    录原有客户端的IP地址和原来客户端的请求的服务器地址。
287         log_format access '$remote_addr - $remote_user [$time_local]
    "$request" '
288         '$status $body_bytes_sent "$http_referer" '
289         '"$http_user_agent" $http_x_forwarded_for';
290
291         #定义本虚拟主机的访问日志
292         access_log  /usr/local/nginx/logs/host.access.log  main;
293         access_log  /usr/local/nginx/logs/host.access.404.log  log404;
294
295         #对 "/" 启用反向代理
296         location / {
297             proxy_pass http://127.0.0.1:88;
```

```
            proxy_redirect off;
            proxy_set_header X-Real-IP $remote_addr;

            #后端的Web服务器可以通过X-Forwarded-For获取用户真实IP
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

            #以下是一些反向代理的配置，可选。
            proxy_set_header Host $host;

            #允许客户端请求的最大单文件字节数
            client_max_body_size 10m;

            #缓冲区代理缓冲用户端请求的最大字节数，
            #如果把它设置为比较大的数值，例如256k，那么，无论使用firefox还是IE浏览器，来提交任意小于256k的图片，都很正常。如果注释该指令，使用默认的client_body_buffer_size设置，也就是操作系统页面大小的两倍，8k或者16k，问题就出现了。
            #无论使用firefox4.0还是IE8.0，提交一个比较大，200k左右的图片，都返回500 Internal Server Error错误
            client_body_buffer_size 128k;

            #表示使nginx阻止HTTP应答代码为400或者更高的应答。
            proxy_intercept_errors on;

            #后端服务器连接的超时时间_发起握手等候响应超时时间
            #nginx跟后端服务器连接超时时间(代理连接超时)
            proxy_connect_timeout 90;

            #后端服务器数据回传时间(代理发送超时)
            #后端服务器数据回传时间_就是在规定时间之内后端服务器必须传完所有的数据
            proxy_send_timeout 90;

            #连接成功后，后端服务器响应时间(代理接收超时)
            #连接成功后_等候后端服务器响应时间_其实已经进入后端的排队之中等候处理（也可以说是后端服务器处理请求的时间）
            proxy_read_timeout 90;

            #设置代理服务器（nginx）保存用户头信息的缓冲区大小
            #设置从被代理服务器读取的第一部分应答的缓冲区大小，通常情况下这部分应答中包含一个小的应答头，默认情况下这个值的大小为指令proxy_buffers中指定的一个缓冲区的大小，不过可以将其设置为更小
            proxy_buffer_size 4k;

            #proxy_buffers缓冲区，网页平均在32k以下的设置
            #设置用于读取应答（来自被代理服务器）的缓冲区数目和大小，默认情况也为分页大小，根据操作系统的不同可能是4k或者8k
            proxy_buffers 4 32k;

            #高负荷下缓冲大小（proxy_buffers*2）
            proxy_busy_buffers_size 64k;

            #设置在写入proxy_temp_path时数据的大小，预防一个工作进程在传递文件时阻塞太长
            #设定缓存文件夹大小，大于这个值，将从upstream服务器传
            proxy_temp_file_write_size 64k;
        }


        #设定查看Nginx状态的地址
```

```
        location /NginxStatus {
            stub_status on;
            access_log on;
            auth_basic "NginxStatus";
            auth_basic_user_file confpasswd;
            #htpasswd文件的内容可以用apache提供的htpasswd工具来产生。
        }

        #本地动静分离反向代理配置
        #所有jsp的页面均交由tomcat或resin处理
        location ~ .(jsp|jspx|do)?$ {
            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://127.0.0.1:8080;
        }

        #所有静态文件由nginx直接读取不经过tomcat或resin
        location ~ .*.
(htm|html|gif|jpg|jpeg|png|bmp|swf|ioc|rar|zip|txt|flv|mid|doc|ppt|
        pdf|xls|mp3|wma)$
        {
            expires 15d;
        }

        location ~ .*.(js|css)?$
        {
            expires 1h;
        }
    }
}
```

日志管理

- 通过访问日志，你可以得到用户地域来源、跳转来源、使用终端、某个URL访问量等相关信息
- 通过错误日志，你可以得到系统某个服务或server的性能瓶颈等

日志格式

- 日志生成的到Nginx根目录logs/access.log文件，默认使用“main”日志格式，也可以自定义格式
- 默认“main”日志格式

```
log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                  '$status $body_bytes_sent "$http_referer" '
                  '"$http_user_agent" "$http_x_forwarded_for"';

$remote_addr：客户端的ip地址(代理服务器，显示代理服务ip)
$remote_user：用于记录远程客户端的用户名称（一般为“-”）
$time_local：用于记录访问时间和时区
$request：用于记录请求的url以及请求方法
$status：响应状态码，例如：200成功、404页面找不到等。
$body_bytes_sent：给客户端发送的文件主体内容字节数
```

```
$http_user_agent：用户所使用的代理（一般为浏览器）
$http_x_forwarded_for：可以记录客户端IP，通过代理服务器来记录客户端的ip地址
$http_referer：可以记录用户是从哪个链接访问过来的

```

日志切割

- nginx的日志文件没有rotate功能
- 编写每天生成一个日志，我们可以写一个nginx日志切割脚本来自动切割日志文件
  - 第一步就是重命名日志文件 （不用担心重命名后nginx找不到日志文件而丢失日志。在你未重新打开原名字的日志文件前，nginx还是会向你重命名的文件写日志，Linux是靠文件描述符而不是文件名定位文件 ）
  - 第二步向nginx主进程发送USR1信号
    - nginx主进程接到信号后会从配置文件中读取日志文件名称
    - 重新打开日志文件 (以配置文件中的日志名称命名)，并以工作进程的用户作为日志文件的所有者
    - 重新打开日志文件后，nginx主进程会关闭重名的日志文件并通知工作进程使用新打开的日志文件
    - 工作进程立刻打开新的日志文件并关闭重名名的日志文件
    - 然后你就可以处理旧的日志文件了。[或者重启nginx服务]
- **nginx日志按每分钟自动切割脚本如下**：
  - 新建shell脚本：

```
vi   /opt/nginx/nginx_log.sh
```

```
#!/bin/bash
#设置日志文件存放目录
LOG_HOME="/opt/nginx/logs/"

#备分文件名称
LOG_PATH_BAK="$(date -d yesterday +%Y%m%d%H%M)".access.log

#重命名日志文件
mv ${LOG_HOME}/access.log ${LOG_HOME}/${LOG_PATH_BAK}.log

#向nginx主进程发信号重新打开日志
kill -USR1 `cat /opt/nginx/logs/nginx.pid`
```

  - 创建crontab设置作业
    - 设置日志文件存放目录crontab -e

```
*/1 * * * *  sh /opt/nginx/nginx_log.sh
```

语法说明

Location

语法规则：

```
location [=|~|~*|^~] /uri/ {… }
```

- 首先匹配 =
- 其次匹配^~
- 其次是按文件中顺序的正则匹配
- 最后是交给 /通用匹配
- 当有匹配成功时候，停止匹配，按当前匹配规则处理请求

符号含义：

```
#表示精确匹配
=
#表示uri以某个常规字符串开头，理解为匹配 url路径即可
#nginx不对url做编码，因此请求为/static/20%/aa，可以被规则^~ /static/ /aa匹配到（注意是空格）
^~
#表示区分大小写的正则匹配
~
#表示不区分大小写的正则匹配
~*
#!~和!~*分别为区分大小写不匹配及不区分大小写不匹配的正则
!~和!~*
#用户所使用的代理（一般为浏览器）
/
#可以记录客户端IP，通过代理服务器来记录客户端的ip地址
$http_x_forwarded_for
#可以记录用户是从哪个链接访问过来的
$http_referer
```

常用规则：

- 直接匹配网站根，通过域名访问网站首页比较频繁，使用这个会加速处理
  - 第一个必选规则

    ```
    location = / {
        proxy_passhttp://tomcat:8080/index
     }
    ```

  - 第二个必选规则是处理静态文件请求，这是nginx作为http服务器的强项
    - 有两种配置模式，目录匹配或后缀匹配,任选其一或搭配使用

```
    location ^~ /static/ {
       # 请求/static/a.txt 将被映射到实际目录文件:/webroot/res/static/a.txt
       root /webroot/res/;
    }

    location ~* \.(gif|jpg|jpeg|png|css|js|ico)${
       root /webroot/res/;
    }
```

- 第三个规则就是通用规则，用来转发动态请求到后端应用服务器

```
    location / {
       proxy_pass http://tomcat:8080/
    }
```

总结：

- 先判断精准命中，如果命中，立即返回结果并结束解析过程
- 判断普通命中，如果有多个命中，“记录”下来“最长”的命中结果（记录但不结束，最长的为准）
- 继续判断正则表达式的解析结果，按配置里的正则表达式顺序为准，由上至下开始匹配，一旦匹配成功1个，立即返回结果，并结束解析过程
- 普通命中顺序无所谓，是因为按命中的长短来确定。正则命中，顺序有所谓，因为是从前入往后命中的

ReWrite

- rewrite只能放在server{},location{},if{}中 ， 并且只能对域名后边的除去传递的参数外的字符串起作用
- Nginx提供的全局变量或自己设置的变量，结合正则表达式和标志位实现url重写以及重定向
- Rewrite主要的功能就是实现URL的重写
- Nginx的Rewrite规则采用Pcre，perl兼容正则表达式的语法规则匹配，如果需要Nginx的Rewrite功能，在编译Nginx之前，需要编译安装PCRE库
- 通过Rewrite规则，可以实现规范的URL、根据变量来做URL转向及选择配置

相关指令：

- break ：
  - 默认值： none
  - 使用范围： if，server，location
  - 作用： 完成当前的规则集，不再处理rewrite指令，需要和last加以区分
- if (condition) {....}
  - 默认值： none
  - 使用范围： server，location
  - 作用：用于检测一个条件是否符合，符合则执行大括号内的语句。不支持嵌套，不支持多个条件&&或||处理
- return
  - 默认值： none

    - 使用范围： server，if，location
    - 作用：用于结束规则的执行和返回状态码给客户端
- rewrite regex replacement flag
    - 使用范围： server，if，location
    - 作用：该指令根据表达式来重定向URI，或者修改字符串。指令根据配置文件中的顺序来执行。注意重写表达式只对相对路径有效
- uninitialized_variable_warn on|off
    - 默认值： on
    - 使用范围：http,server,location,if
    - 作用：该指令用于开启和关闭未初始化变量的警告信息，默认值为开启
- set variable value
    - 默认值： none
    - 作用：该指令用于定义一个变量，并且给变量进行赋值。变量的值可以是文本、一个变量或者变量和文本的联合，文本需要用引号引起来

rewrite全局变量表：

变量	含义
$args	这个变量等于请求行中的参数，同$query_string
$content length	请求头中的Content-length字段。
$content_type	请求头中的Content-Type字段。
$document_root	当前请求在root指令中指定的值。
$host	请求主机头字段，否则为服务器名称。
$http_user_agent	客户端agent信息
$http_cookie	客户端cookie信息
$limit_rate	这个变量可以限制连接速率。
$request_method	客户端请求的动作，通常为GET或POST。
$remote_addr	客户端的IP地址。
$remote_port	客户端的端口。
$remote_user	已经经过Auth Basic Module验证的用户名。
$request_filename	当前请求的文件路径，由root或alias指令与URI请求生成。
$scheme	HTTP方法（如http，https）。
$server_protocol	请求使用的协议，通常是HTTP/1.0或HTTP/1.1。
$server_addr	服务器地址，在完成一次系统调用后可以确定这个值。
$server_name	服务器名称。
$server_port	请求到达服务器的端口号。
$request_uri	包含请求参数的原始URI，不包含主机名，如”/foo/bar.php?arg=baz”。
$uri	不带请求参数的当前URI，$uri不包含主机名，如”/foo/bar.html”。
$document_uri	与$uri相同。

语法规则：

操作符	含义
= ,!=	比较的一个变量和字符串
~，~*	与正则表达式匹配的变量，如果这个正则表达式中包含}，;则整个表达式需要用"或'包围
-f，!-f	检查一个文件是否存在
-d, !-d	检查一个目录是否存在
-e，!-e	检查一个文件、目录、符号链接是否存在
-x，!-x	检查一个文件是否可执行

IF指令：

```
    if  语法格式
     if 空格 (条件) {
        重写模式
     }

        # 限制浏览器访问
    if ($http_user_agent ~ Firefox) {
      rewrite ^(.*)$ /firefox/$1 break;
    }

    if ($http_user_agent ~ MSIE) {
        rewrite ^(.*)$ /msie/$1 break;
    }

    if ($http_user_agent ~ Chrome) {
        rewrite ^(.*)$ /chrome/$1 break;
    }
```

return指令：

```
   # 限制IP访问
    if  ($remote_addr = 192.168.197.142) {
       return 403;
    }
```

- 首先从日志查出ip
- 修改conf配置文件
- 重启配置

rewrite指令：

- 判断目录是否存在
- 服务器内部的rewrite和302跳转不一样.跳转的话URL都变了,变成重新http请求index.html,而内部rewrite,上下文没变

```
  if (!-e $document_root$fastcgi_script_name) {
        rewrite ^.*$ /index.html break;
    }
```

set指令：

- set指令是设置变量用的,可以用来达到多条件判断时作标志用
- 判断IE并重写,且不用break;我们用set变量来达到目的

```
    if ($http_user_agent ~* msie) {
        set $isie 1;
     }

     if ($fastcgi_script_name = ie.html) {
        set $isie 0;
     }

     if ($isie 1) {
        rewrite ^.*$ ie.html;
     }
```

运行过程

默认情况下，运行中的Nginx会包含如下进程：

- 主进程master process（父进程）
  - 主进程充当监控进程
  - 负责整个进程组与管理用户的交互接口，同时对进程进行监护
  - 它不需要处理网络事件，不负责业务执行，只会通过管理worker进程来实现重启服务、关闭服务、配置文件生效等功能
- 子进程worker process（工作进程）
  - 子进程充当工作进程
  - 负责完成具体的任务
  - 完成用户请求接收与返回用户数据，以及与后端应用服务器的数据交互等工作

原理介绍：

- Nginx 的高并发得益于其采用了 epoll 模型，与传统的服务器程序架构不同，epoll 是linux 内核 2.6 以后才出现的
- Nginx 采用 epoll 模型，异步非阻塞，而 Apache 采用的是select 模型
  - Select 特点：select 选择句柄的时候，是遍历所有句柄，也就是说句柄有事件响应时，select 需要遍历所有句柄才能获取到哪些句柄有事件通知，因此效率是非常低
  - epoll 的特点：epoll 对于句柄事件的选择不是遍历的，是事件响应的，就是句柄上事件来就马上选择出来，不需要遍历整个句柄链表，因此效率非常高

负载均衡

https://mp.weixin.qq.com/s?__biz=MzIxMjg4NDU1NA==&mid=2247483785&idx=1&sn=66b01f7b73a5a3ea60133585a6a69d6d&chksm=97be0caca0c985ba9f080117c78874edc97f8d653c597be2e844c934d834b2f1bcff927b4ea7&scene=21#wechat_redirect

https://mp.weixin.qq.com/s?__biz=MzIxMjg4NDU1NA==&mid=2247483847&idx=1&sn=4d3c6d6c331342cc930cb4cfc0ca465c&chksm=97be0ce2a0c985f427f0462a9be9dd49b963a1e09ea86ef68f081afb8a21f96f247708fbc52d&scene=21#wechat_redirect

动静分离

- 实现：
  - 通过 location 指定不同的后缀名实现不同的请求转发
  - 通过 expires 参数设置，可以使浏览器缓存过期时间，减少与服务器之前的请求和流量
    - Expires 定义：是给一个资源设定一个过期时间，也就是说无需去服务端验证，直接通过浏览器自身确认是否过期即可，所以不会产生额外的流量
    - 此种方法非常适合不经常变动的资源。（如果经常更新的文件，不建议使用 Expires 来缓存）
    - 这里设置 3d，表示在这 3 天之内访问这个 URL，发送一个请求，比对服务器该文件最后更新时间没有变化，则不会从服务器抓取，返回状态码 304，如果有修改，则直接从服务器重新下载，返回状态码 200

示例：

- 新建一个web工程，并设置一张图片
- 修改nginx配置文件：

```
server {
        listen       80;
        server_name  a;


        location ~ .*\.(html|htm|gif|jpg|jpeg|bmp|png|ico|txt|js|css)$
        {
        root /opt/apache-tomcat-8.5.31/webapps/;
        expires 30d;
        }
    }
```

- 测试
- F12查看请求

- 再次刷新测试
- 查看控制器请求

高并发处理

https://mp.weixin.qq.com/s?__biz=MzIxMjg4NDU1NA==&mid=2247483785&idx=1&sn=66b01f7b73a5a3ea60133585a6a69d6d&chksm=97be0caca0c985ba9f080117c78874edc97f8d653c597be2e844c934d834b2f1bcff927b4ea7&scene=21#wechat_redirect