配置文件说明

```
#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，
```

```
121
122 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
123
124 被移除。
125
126 #语法:open_file_cache_min_uses number 默认
    值:open_file_cache_min_uses 1
127
128 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
129
130 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.
131
132 open_file_cache_min_uses 1;
133 #语法:open_file_cache_errors on | off 默认
    值:open_file_cache_errors off
134
135 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.
136
137 open_file_cache_errors on;
138
139 }
140
141 #设定http服务器，利用它的反向代理功能提供负载均衡支持
142
143 http
144
145 {
146
147 #文件扩展名与文件类型映射表
148
149 include mime.types;
150
151 #默认文件类型
```

```
152
153 default_type application/octet-stream;
154
155 #默认编码
156
157 #charset utf-8;
158
159 #服务器名字的hash表大小
160
161 #保存服务器名字的hash表是由指令
    server_names_hash_max_size 和
162
163 server_names_hash_bucket_size所控制的。参数hash
    bucket size总是等于hash表的大
164
165 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
166
167 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
168
169 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
170
171 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
172
173 首要的是增大前一个参数的大小.
174
175 server_names_hash_bucket_size 128;
176
177 #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
178
179 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
180
```

```
令getconf PAGESIZE取得。

client_header_buffer_size 32k;

#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。

large_client_header_buffers 4 64k;

#设定通过nginx上传文件的大小

client_max_body_size 8m;

#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。

#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。

sendfile on;

#开启目录列表访问，合适下载服务器，默认关闭。

```

```
autoindex on;

#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

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;
```

```
241
242 #指定连接到后端FastCGI的超时时间
243
244 fastcgi_connect_timeout 300;
245
246 #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
247
248 时间
249
250 fastcgi_send_timeout 300;
251
252 #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
253
254 fastcgi_read_timeout 300;
255
256 #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
257
258 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
259
260 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
261
262 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
263
264 fastcgi_buffer_size 64k;
265
266 #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
267
268 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
269
```

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

```
297
298 fastcgi_cache TEST
299
300 #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，
301
302 其他为1分钟
303
304 fastcgi_cache_valid 200 302 1h;
305
306 fastcgi_cache_valid 301 1d;
307
308 fastcgi_cache_valid any 1m;
309
310 #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
311
312 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
313
314 fastcgi_cache_min_uses 1;
315
316 #gzip模块设置
317
318 #开启压缩
319
320 gzip on;
321
322 # 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。
323
324 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。
325 gzip_min_length 2k;
326
327 # 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
328
```

```
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]\.";
```

```
362
363 # 默认值：off
364
365 # 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
366
367 务器必须要返回包含"Via"的 header头。
368
369 # off - 关闭所有的代理结果数据的压缩
370
371 # expired - 启用压缩，如果header头中包含 "Expires" 头信息
372
373 # no-cache - 启用压缩，如果header头中包含 "Cache-Control:no-cache" 头信息
374
375 # no-store - 启用压缩，如果header头中包含 "Cache-Control:no-store" 头信息
376
377 # private - 启用压缩，如果header头中包含 "Cache-Control:private" 头信息
378
379 # no_last_modified - 启用压缩,如果header头中不包含 "Last-Modified" 头信息
380
381 # no_etag - 启用压缩 ,如果header头中不包含 "ETag" 头信息
382
383 # auth - 启用压缩 , 如果header头中包含 "Authorization" 头信息
384
385 # any - 无条件启用压缩
386
387 gzip_proxied expired no-cache no-store private auth;
388
```

```
# 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。
```

```
422
423 #例如
424
425 #upstream bakend {
426
427 # server 192.168.0.14 weight=10;
428
429 # server 192.168.0.15 weight=10;
430
431 #}
432
433 #2、ip_hash
434
435 #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
436
437 session的问题。
438
439 #例如：
440
441 #upstream bakend {
442
443 # ip_hash;
444
445 # server 192.168.0.14:88;
446
447 # server 192.168.0.15:80;
448
449 #}
450
451 #3、fair（第三方）
452
453 #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。
454
455 #upstream backend {
456
457 # server server1;
```

```
458
459 # server server2;
460
461 # fair;
462
463 #}
464
465 #4、url_hash（第三方）
466
467 #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
468
469 缓存时比较有效。
470
471 #例：在upstream中加入hash语句，server语句中不能写入weight等其他的参数，
472
473 hash_method是使用的hash算法
474
475 #upstream backend {
476
477 # server squid1:3128;
478
479 # server squid2:3128;
480
481 # hash $request_uri;
482
483 # hash_method crc32;
484
485 #}
486
487 #tips:
488
489 #upstream bakend{#定义负载均衡设备的Ip及设备状态}{
490
491 # ip_hash;
492
```

```
# 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

```

```
525 中用来做debug
526
527 #client_body_temp_path设置记录文件的目录 可以设置最多
    3层目录
528
529 #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
530
531 }
532 #虚拟主机的配置
533
534 server
535
536 {
537
538 #监听端口
539
540 listen 80;
541
542 #域名可以有多个，用空格隔开
543
544 server_name www.xx.com xx.com;
545
546 index index.html index.htm index.php;
547
548 root /data/www/xx;
549
550 #对******进行负载均衡
551
552 location ~ .*.(php|php5)?$
553
554 {
555
556 fastcgi_pass 127.0.0.1:9000;
557
558 fastcgi_index index.php;
559
```

```
include fastcgi.conf;

}

#图片缓存时间设置

location ~ .*.(gif|jpg|jpeg|png|bmp|swf)$

{

expires 10d;

}

#JS和CSS缓存时间设置

location ~ .*.(js|css)?$

{

expires 1h;

}

#日志格式设定

#$remote_addr与$http_x_forwarded_for用以记录客户端的ip地址；

#$remote_user：用来记录客户端用户名称；

#$time_local： 用来记录访问时间与时区；

#$request： 用来记录请求的url与http协议；

#$status： 用来记录请求状态；成功是200，

```

```
#$body_bytes_sent ：记录发送给客户端文件主体内容大小；

#$http_referer：用来记录从那个页面链接访问过来的；

#$http_user_agent：记录客户浏览器的相关信息；

#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。

#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。

log_format access '$remote_addr - $remote_user [$time_local]

"$request" '

'$status $body_bytes_sent "$http_referer" '

'"$http_user_agent" $http_x_forwarded_for';

#定义本虚拟主机的访问日志

access_log /usr/local/nginx/logs/host.access.log main;

access_log /usr/local/nginx/logs/host.access.404.log log404;

#对 "/" 启用反向代理

location / {
```

```
627
628 proxy_pass http://127.0.0.1:88;
629 proxy_redirect off;
630
631 proxy_set_header X-Real-IP $remote_addr;
632
633 #后端的Web服务器可以通过X-Forwarded-For获取用户真实IP
634
635 proxy_set_header X-Forwarded-For
    $proxy_add_x_forwarded_for;
636
637 #以下是一些反向代理的配置，可选。
638
639 proxy_set_header Host $host;
640
641 #允许客户端请求的最大单文件字节数
642
643 client_max_body_size 10m;
644
645 #缓冲区代理缓冲用户端请求的最大字节数，
646
647 #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
648
649 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
650
651 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。
652
653 #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
654
655 500 Internal Server Error错误
656
657 client_body_buffer_size 128k;
658
```

```
#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;
```

```
692
693 #proxy_buffers缓冲区，网页平均在32k以下的设置
694
695 #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
696
697 小，根据操作系统的不同可能是4k或者8k
698
699 proxy_buffers 4 32k;
700
701 #高负荷下缓冲大小（proxy_buffers*2）
702
703 proxy_busy_buffers_size 64k;
704
705 #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
706
707 太长
708
709 #设定缓存文件夹大小，大于这个值，将从upstream服务器传
710
711 proxy_temp_file_write_size 64k;
712
713 }
714
715 #设定查看Nginx状态的地址
716 location /NginxStatus {
717
718 stub_status on;
719
720 access_log on;
721
722 auth_basic "NginxStatus";
723
724 auth_basic_user_file confpasswd;
725
```

```
#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;

}

}

}

```