

# 不错的 Nginx 配置介绍及性能调优

Nginx 是一款轻量级的 Web 服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器，在 BSD-like 协议下发行。其特点是占有内存少，并发能力强，事实上 nginx 的并发能力在同类型的网页服务器中表现较好，中国大陆使用 nginx 网站用户有：百度、京东、新浪、网易、腾讯、淘宝等。

可以在大多数 UnixLinux OS 上编译运行，并有 Windows 移植版。是一个很强大的高性能 Web 和反向代理服务，它具有很多非常优越的特性，在连接高并发的情况下，Nginx 是 Apache 服务不错的替代品：Nginx 在美国是做虚拟主机生意的老板们经常选择的软件平台之一，能够支持高达 50,000 个并发连接数的响应。

Nginx 作为负载均衡服务：Nginx 既可以在内部直接支持 Rails 和 PHP 程序对外进行服务，也可以支持作为 HTTP 代理服务 对外进行服务。Nginx 采用 C 进行编写，不论是系统资源开销还是 CPU 使用效率都比 Perlbal 要好很多。

## 一、Nginx 配置说明

### 1、server 代码块

server 代码块位于 http 代码块内部，每一个 server 都可以用来配置一个虚拟主机。也就是说，每一个 server 代表了一个虚拟服务器的配置信息。

可以添加多个 server 来配置多个虚拟主机。

server 中的主要配置有：

- listen 虚拟主机监听的端口
- server\_name 虚拟主机的域名或 IP 地址，可以配置多个（用空格隔开）
- root 虚拟主机的根目录
- index 虚拟主机的首页，也可以用 location 代码块来配置
- access\_log 虚拟主机的访问日志
- error\_log 虚拟主机的错误日志
- error\_page 错误页面

```
server {
    listen 80;
    server_name localhost;

    #access_log logs/host.access.log main;
    root "D:/phpStudy/WWW";
    location / {
        index index.html index.htm index.php 1.php;
        autoindex off;
    }
}
```

```
}

#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;
}

# proxy the PHP scripts to Apache listening on 127.0.0.1:80
#
#location ~ \.php$ {
#    proxy_pass http://127.0.0.1;
#}

# pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
#
location ~ \.php(.*)$ {
    fastcgi_pass 127.0.0.1:9000;
    fastcgi_index index.php;
    fastcgi_split_path_info ^((?U).+\.php)(/?.+)$;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_param PATH_INFO $fastcgi_path_info;
    fastcgi_param PATH_TRANSLATED $document_root$fastcgi_path_info;
    include fastcgi_params;
}

# deny access to .htaccess files, if Apache's document root
# concurs with nginx's one
#
#location ~ /\.ht {
#    deny all;
}
```

```
#}  
}
```

## 2、location 代码块

location 代码块位于 server 代码块内部。

location 用于配置虚拟主机的 URI，它是一个非常重要的配置。

可以给每一个 server（虚拟主机）配置多个 location。

可以根据不同的 URI 配置不同的 location，来处理不同的请求。

### 2.1、location 的语法格式

其中， = | ~ | ~\* | ^~ | @ 表示前缀，也叫修饰符，是可选的；uri 表示普通字符串或正则表达式，是必须的。

@ 这个修饰符非常特殊，后面跟一个普通字符串，用于定义特殊的类型，被定义的类型只能被 nginx 内部调用，用于内部的重定向。这个重定向纯粹是 nginx 内部的一个转发行为。

= 字符串完整匹配。

~ 区分大小写的正则匹配。

~\* 不区分大小写的正则匹配。

^~ 字符串前缀匹配，只要匹配到了，就不会再匹配其他的正则 location。

如果没有任何修饰符，也表示字符串前缀匹配，即字符串 location。

如果 location 中使用了修饰符 ~ 或者 ~\*，那么，这个 location 就是正则 location；否则，就是字符串 location。

```
location [ = | ~ | ~* | ^~ | @ ] uri {...}
```

### 2.2、多个 location 的匹配顺序

多个 location 的匹配顺序与 location 的位置顺序没有直接关系，匹配顺序为：

1、= 修饰符的优先级最高，表示完整匹配。如果匹配成功，则停止匹配其他 location。

2、字符串 location 的优先级第二；多个字符串 location 的匹配顺序为从长到短，也就是说优先选择长度最长的字符串匹配；匹配成功的字符串 location 如果使用了修饰符 ^~ 或者正好是精准匹配，则不会再去检验正则 location。

3、正则 location 的优先级低于字符串 location；多个 正则 location 会按照配置文件里的位置顺序进行匹配，如果匹配成功，就停止匹配。

注意：虽然字符串 location 的优先级高于正则 location。但是，如果匹配成功的字符串 location 中没有使用修饰符 ^~，也不是精准匹配，那么还会继续检测是否有匹配的正则 location。如果匹配到了正则 location，就立即使用该正则 location 并停止匹配；否则，才会使用字符串 location。

也就是说，匹配到的字符串 location 可能会被正则 location 所覆盖。

匹配成功的字符串 location，如果不想再继续检测匹配正则 location，有三种实现方式：

- 使用 = 修饰符，来进行完整匹配。
- 使用 ^~ 修饰符，仍然还是前缀匹配。
- 如果字符串匹配正好是精准的前缀匹配，也不会再去检测正则 location。这是一种隐式的实现方式。

**2.3、匹配模式及其顺序**

- 1、location = /string 字符串完整匹配，优先级最高。
  - 2、location ^~ /string 字符串前缀匹配（不检测正则 location）。
  - 3、location ~ pattern 正则匹配（区分大小写）。
  - 4、location ~\* pattern 正则匹配（不区分大小写）。
  - 5、location /string 不带修饰符的字符串前缀匹配。
  - 6、location / 默认匹配，如果一个请求没有匹配到其他的 location，就会匹配默认匹配。它相当于 switch 中的 default 。
- 说明：对于字符串 location，如果没有 = 修饰符，就都是前缀匹配；而正则 location，可能是前缀匹配、后缀匹配、中间匹配和完整匹配中的任意一种，这取决于正则表达式本身。

**2.4、配置默认主页**

```
location / {
    index index.html index.htm index.php l.php;
    autoindex off;
}
```

**2.5、配置反向代理**

```
location / {
    proxy_pass http://localhost:8888;

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

**2.6、URL 美化 (省略 index.php 入口文件)**

```
location / {  
    try_files $uri $uri/ /index.php?$query_string;  
}
```

## 2.7、upstream 代码块

upstream 代码块位于 http 代码块内部。

upstream 用于对服务器集群进行负载均衡的配置。

```
upstream name {  
    ip_hash;  
    server 192.168.1.100:8000;  
    server 192.168.1.100:8001 down;  
    server 192.168.1.100:8002 max_fails=3;  
    server 192.168.1.100:8003 fail_timeout=20s;  
    server 192.168.1.100:8004 max_fails=3 fail_timeout=20s;  
}
```

- ip\_hash: 手动指定调度算法。
- down: 表示该主机暂停服务。
- max\_fails: 表示失败最大次数, 超过失败最大次数就会暂停服务。
- fail\_timeout: 表示如果请求受理失败, 暂停指定的时间之后重新发起请求。

## 2.8、配置文件中的全局变量

\$args #这个变量等于请求行中的参数。

\$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\_body\_file #客户端请求主体信息的临时文件名。

\$request\_method #客户端请求的动作, 通常为 GET 或 POST。

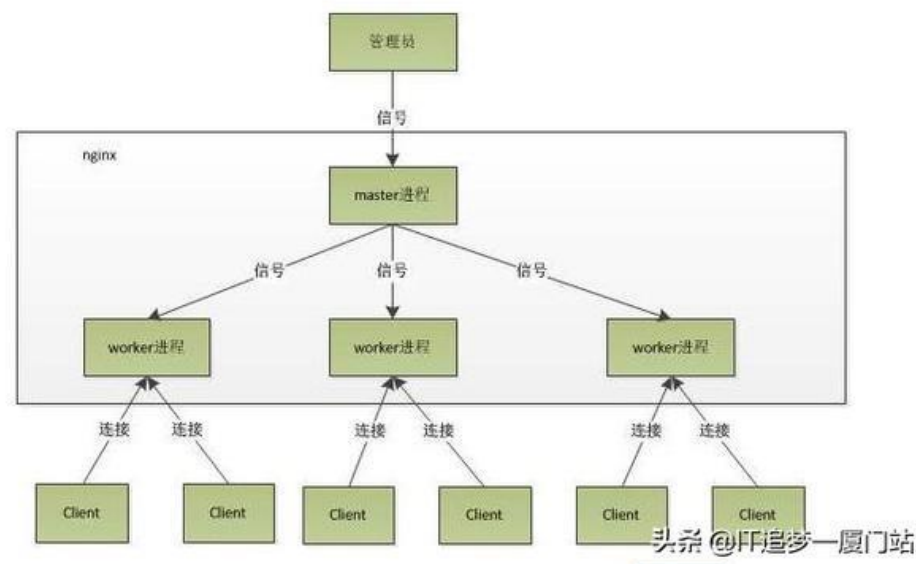
\$remote\_addr #客户端的 IP 地址。

\$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。  
\$query\_string #与\$args 相同。  
\$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。

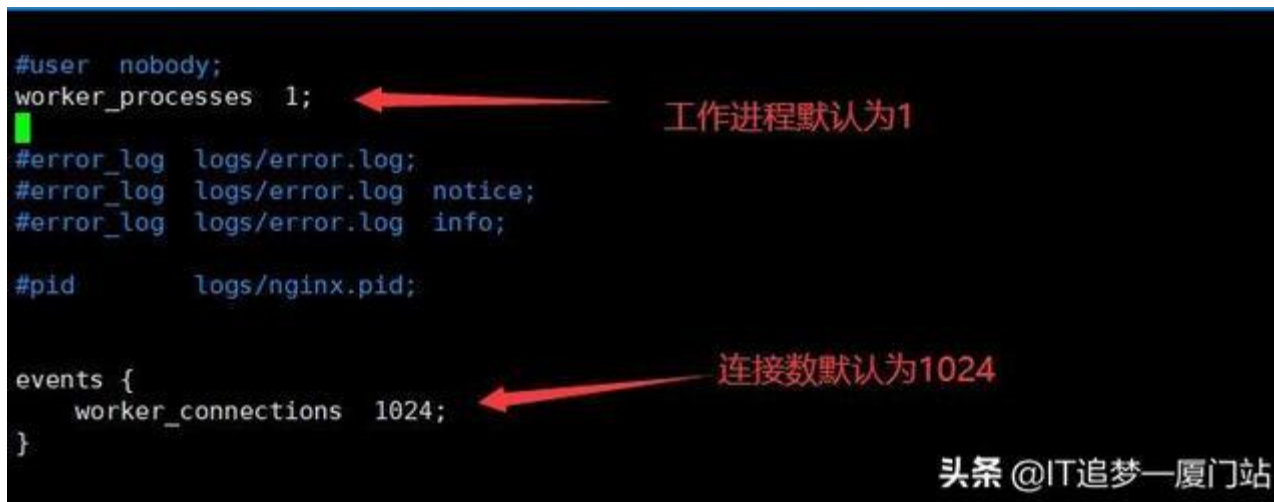
## 二、Nginx 性能调优

### 1、优化 Nginx 工作进程数及连接数

Nginx 有 master 和 worker 两种进程，master 进程用于管理 worker 进程，worker 进程用于 Nginx 服务。



而 worker 进程数默认为 1。单进程最大连接数为 1024。如下图（打开 Nginx 目录下的/conf/nginx.conf 文档），现在我们来对这两个数值进行调优



### 1.1、worker 进程设置

worker 进程数应该设置为服务器 CPU 的核数。所以我们得先查看一下本机的 CPU 核数，得到结果后，再设置上图中的 `worker_processes` 值

```
# 查看 CPU 核数
grep -c processor /proc/cpuinfo
```

### 1.2、调整最大连接数

控制 Nginx 单个进程允许的最大连接数的参数为 `worker_connections`，这个参数要根据服务器性能和内存使用量来调整。进程的最大连接数受 Linux 系统进程打开的最大文件数的限制，只有执行了 "`ulimit -HSn 65535`" 之后，`worker_connections` 才能生效。连接数包括代理服务器的连接、客户端的连接等，Nginx 总并发连接数 = `worker_processes` \* `worker_connections`。总数保持在 3w 左右即可。

```
worker_processes 2;
worker_cpu_affinity 01 10;
user nginx nginx;
events {
    use epoll;
    worker_connections 15000;
}
```

## 2、绑定 Nginx 进程到不同的 CPU 上

默认情况下，Nginx 的多个进程有可能跑在某一个 CPU 或 CPU 的某一核上，导致 Nginx 进程使用硬件的资源不均，因此绑定 Nginx 进程到不同的 CPU 上是为了充分利用硬件的多 CPU 多核资源。

```
[root@localhost ~]# grep -c processor /proc/cpuinfo # 查看 CPU 核数
2
worker_processes 2; # 2 核 CPU 的配置
worker_cpu_affinity 01 10;

worker_processes 4; # 4 核 CPU 的配置
worker_cpu_affinity 0001 0010 0100 1000;

worker_processes 8; # 8 核 CPU 的配置
worker_cpu_affinity 00000001 00000010 00000100 00001000 00010000 00100000 01000000 10000000;

[root@localhost ~]# /usr/local/nginx/sbin/nginx -t
[root@localhost ~]# /usr/local/nginx/sbin/nginx -s reload
```

## 3、优化 Nginx worker 进程打开的最大文件数

```
# worker 进程打开的最大文件数，可设置为优化后的 ulimit -HSn 的结果
worker_rlimit_nofile 65535;
```

## 4、开启高效文件传输模式

sendfile：参数用于开启文件的高效传输模式，该参数实际上是激活了 sendfile() 功能。

sendfile()：是作用于两个文件描述符之间的数据拷贝函数，这个拷贝操作是在内核之中的，被称为“零拷贝”。sendfile 比 read 和 write 函数要高效得多，因为 read 和 write 函数要把数据拷贝到应用层再进行操作。

tcp\_nopush：参数用于激活 Linux 上的 TCP\_CORK socket 选项，此选项仅仅当开启 sendfile 时才生效，tcp\_nopush 参数可以把 http response header 和文件的开始部分放在一个文件里发布，以减少网络报文段的数量。

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



```
sendfile on; # 开启文件的高效传输模式，减少文件在应用和内核之间的拷贝
tcp_nopush on; # 激活 TCP_CORK socket 选择，当数据包达到一定大小再发送
tcp_nodelay on; # 数据在传输的过程中不进缓存，有数据随时发送（只用在应答需要非常快速的情况下）
```

```
keepalive_timeout 65;
include vhosts/*.conf;
}
```

## 5、优化 Nginx 连接的超时时间

### 5.1：连接超时的作用

- 将无用的连接设置为尽快超时，可以保护服务器的系统资源（CPU、内存、磁盘）
- 当连接很多时，及时断掉那些建立好的但又长时间不做事的连接，以减少其占用的服务器资源
- 如果黑客攻击，会不断地和服务器建立连接，因此设置连接超时以防止大量消耗服务器的资源
- 如果用户请求了动态服务，则 Nginx 就会建立连接，请求 FastCGI 服务以及后端 MySQL 服务，设置连接超时，使得在用户容忍的时间内返回数据

### 5.2：连接超时存在的问题

- 服务器建立新连接是要消耗资源的，因此，连接超时时间不宜设置得太短，否则会造成并发很大，导致服务器瞬间无法响应用户的请求。
- 有些 PHP 站点会希望设置成短连接，因为 PHP 程序建立连接消耗的资源和时间相对要少些。
- 有些 Java 站点会希望设置成长连接，因为 Java 程序建立连接消耗的资源和时间要多一些，这是由语言的运行机制决定的。

### 5.3：设置超时时间

- keepalive\_timeout：用于设置客户端连接保持会话的超时时间，超过这个时间服务器会关闭该连接。
- client\_header\_timeout：用于设置读取客户端请求头数据的超时时间，如果超时客户端还没有发送完整的 header 数据，服务器将返回 "Request time out (408)" 错误。
- client\_body\_timeout：用于设置读取客户端请求主体数据的超时时间，如果超时客户端还没有发送完整的主体数据，服务器将返回 "Request time out (408)" 错误。
- send\_timeout：用于指定响应客户端的超时时间，如果超过这个时间，客户端没有任何活动，Nginx 将会关闭连接。
- tcp\_nodelay：默认情况下当数据发送时，内核并不会马上发送，可能会等待更多的字节组成一个数据包，这样可以提高 I/O 性能，但是，在每次只发送很少字节的业务场景中，使用 tcp\_nodelay 功能，等待时间会比较长。

```
http {
    include mime.types;
    server_names_hash_bucket_size 512;
```

```

default_type application/octet-stream;
sendfile on;
tcp_nodelay on;

keepalive_timeout 65;
client_header_timeout 15;
client_body_timeout 15;
send_timeout 25;

include vhosts/*.conf;
}

```

## 6、限制上传文件的大小

`client_max_body_size` 用于设置最大的允许客户端请求主体的大小。

在请求头中有 "Content-Length"，如果超过了此配置项，客户端会收到 413 错误，即请求的条目过大。

```

http {
    client_max_body_size 8m; # 设置客户端最大的请求主体大小为 8 M
}

```

## 7、FastCGI 相关参数调优

当 LNMP 组合工作时，用户通过浏览器输入域名请求 Nginx Web 服务：

- 如果请求的是静态资源，则由 Nginx 解析后直接返回给用户；
  - 如果是动态请求（如 PHP），那么 Nginx 就会把它通过 FastCGI 接口发送给 PHP 引擎服务（即 php-fpm）进行解析，如果这个动态请求要读取数据库数据，那么 PHP 就会继续请求 MySQL 数据库，以读取需要的数据，并最终通过 Nginx 服务把获取的数据返回给用户。
- 这就是 LNMP 环境的基本请求流程。

在 Linux 中，FastCGI 接口即为 socket，这个 socket 可以是文件 socket，也可以是 IP socket。

```

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

```

```
keepalive_timeout 65;
fastcgi_connect_timeout 240; # Nginx 服务器和后端 FastCGI 服务器连接的超时时间
fastcgi_send_timeout 240; # Nginx 允许 FastCGI 服务器返回数据的超时时间，即在在规定时间内后端服务器必须传完所有的数据，否则 Nginx 将断开这个连接
fastcgi_read_timeout 240; # Nginx 从 FastCGI 服务器读取响应信息的超时时间，表示连接建立成功后，Nginx 等待后端服务器的响应时间
fastcgi_buffer_size 64k; # Nginx FastCGI 的缓冲区大小，用来读取从 FastCGI 服务器端收到的第一部分响应信息的缓冲区大小
fastcgi_buffers 4 64k; # 设定用来读取从 FastCGI 服务器端收到的响应信息的缓冲区大小和缓冲区数量
fastcgi_busy_buffers_size 128k; # 用于设置系统很忙时可以使用的 proxy_buffers 大小
fastcgi_temp_file_write_size 128k; # FastCGI 临时文件的大小
# fastcgi_temp_path /data/nginx_fcgi_tmp; # FastCGI 临时文件的存放路径
fastcgi_cache_path /data/nginx_fcgi_cache levels=2:2 keys_zone=ngx_fcgi_cache:512m inactive=1d max_size=40g; # 缓存目录

server {
listen 80;
server_name www.abc.com;
location / {
root html/www;
index index.html index.htm;
}
location ~ .*\.php$ {
root html/www;
fastcgi_pass 127.0.0.1:9000;
fastcgi_index index.php;
include fastcgi.conf;
fastcgi_cache ngx_fcgi_cache; # 缓存 FastCGI 生成的内容，比如 PHP 生成的动态内容
fastcgi_cache_valid 200 302 1h; # 指定 http 状态码的缓存时间，这里表示将 200 和 302 缓存 1 小时
fastcgi_cache_valid 301 1d; # 指定 http 状态码的缓存时间，这里表示将 301 缓存 1 天
fastcgi_cache_valid any 1m; # 指定 http 状态码的缓存时间，这里表示将其他状态码缓存 1 分钟
fastcgi_cache_min_uses 1; # 设置请求几次之后响应被缓存，1 表示一次即被缓存
fastcgi_cache_use_stale error timeout invalid_header http_500; # 定义在哪些情况下使用过期缓存
fastcgi_cache_key http://$host$request_uri; # 定义 fastcgi_cache 的 key
}
}
```

```
}
```

## 8、gzip 压缩（在之前的讲解 vue 首页加载慢的一文中也有介绍 Nginx 压缩）

Nginx gzip 压缩模块提供了压缩文件内容的功能，用户请求的内容在发送到客户端之前，Nginx 服务器会根据一些具体的策略实施压缩，以节约网站出口带宽，同时加快数据传输效率，来提升用户访问体验。

需要压缩的对象有 html 、 js 、 css 、 xml 、 shtml ，图片和视频尽量不要压缩，因为这些文件大多都是已经压缩过的，如果再压缩可能反而变大。

另外，压缩的对象必须大于 1KB，由于压缩算法的特殊原因，极小的文件压缩后可能反而变大。

```
http {
    gzip on; # 开启压缩功能
    gzip_min_length 1k; # 允许压缩的对象的最小字节
    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的缓存
    gzip_http_version 1.1; # 压缩版本，用于设置识别 HTTP 协议版本
    gzip_comp_level 9; # 压缩级别，1 级压缩比最小但处理速度最快，9 级压缩比最高但处理速度最慢
    gzip_types text/plain application/x-javascript text/css application/xml; # 允许压缩的媒体类型
    gzip_vary on; # 该选项可以让前端的缓存服务器缓存经过 gzip 压缩的页面，例如用代理服务器缓存经过 Nginx 压缩的数据
}
```

## 9、配置 expires 缓存期限

Nginx expires 的功能就是给用户访问的静态内容设定一个过期时间。

当用户第一次访问这些内容时，会把这些内容存储在用户浏览器本地，这样用户第二次及以后继续访问该网站时，浏览器会检查加载已经缓存在用户浏览器本地的内容，就不会去服务器下载了，直到缓存的内容过期或被清除。

不希望被缓存的内容：广告图片、网站流量统计工具、更新很频繁的文件。

缓存期限参考：新浪缓存 15 天，京东缓存 25 年，淘宝缓存 10 年。

```
server {
    listen 80;
    server_name www.abc.com abc.com;
    root html/www;
    location ~ .*\. (gif|jpg|jpeg|png|bmp|swf|js|css)$ # 缓存的对象
    {
```

```
expires 3650d; # 缓存期限为 10 年
}
}
```

## 10、配置防盗链

防盗链：简单地说，就是其它网站未经许可，通过在其自身网站程序里非法调用其他网站的资源，然后在自己的网站上显示这些调用的资源，使得被盗链的那一端消耗带宽资源。

通过 HTTP referer 实现防盗链。

#第一种,匹配后缀

```
location ~ .*\. (gif|jpg|jpeg|png|bm|swf|flv|rar|zip|gz|bz2)$ { # 指定需要使用防盗链的媒体资源
    access_log off; # 不记录日志
    expires 15d; # 设置缓存时间
    valid_referers none blocked *.test.com *.abc.com; # 表示仅允许这些域名访问上面的媒体资源
    if ($invalid_referer) { # 如果域名不是上面指定的地址就返回 403
        return 403;
    }
}
```

#第二种,绑定目录

```
location /images {
    root /web/www/img;
    valid_referers none blocked *.spdir.com *.spdir.top;
    if ($invalid_referer) {
        return 403;
    }
}
```

## 11、操作系统优化

1、配置文件/etc/sysctl.conf，如下：

```
sysctl -w net.ipv4.tcp_syncookies=1 #防止一个套接字在有过多试图连接到达时引起过载
sysctl -w net.core.somaxconn=1024 #默认 128，连接队列
sysctl -w net.ipv4.tcp_fin_timeout=10 #timewait 的超时时间
sysctl -w net.ipv4.tcp_tw_reuse=1 #os 直接使用 timevait 的连接
sysctl -w net.ipv4.tcp_tw_recycle=0 #回收禁用
```

## 2、配置文件/etc/security/limits.conf，如下：

```
hard nofile 204800
soft nofile 204800
soft core unlimited
soft stack 204800
```

## 003、Nginx 的负载均衡方式

我的知乎：DarrenChan 陈驰

如果 Nginx 没有仅仅只能代理一台服务器的话，那它也不可能像今天这么火，Nginx 可以配置代理多台服务器，当一台服务器宕机之后，仍能保持系统可用。具体配置过程如下：

1. 在 http 节点下，添加 upstream 节点。

```
upstream linuxidc {
    server 10.0.6.108:7080;
    server 10.0.0.85:8980;
}
```

2. 将 server 节点下的 location 节点中的 proxy\_pass 配置为：http:// + upstream 名称，即“http://linuxidc”。

```
location / {
    root html;
    index index.html index.htm;
    proxy_pass http://linuxidc;
}
```

3. 现在负载均衡初步完成了。upstream 按照轮询（默认）方式进行负载，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。虽然这种方式简便、成本低廉。但缺点是：可靠性低和负载分配不均衡。适用于图片服务器集群和纯静态页面服务器集群。

除此之外，upstream 还有其它的分配策略，分别如下：

**weight**（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。如下所示，**10.0.0.88** 的访问比率要比 **10.0.0.77** 的访问比率高一倍。

```
upstream linuxidc{
    server 10.0.0.77 weight=5;
    server 10.0.0.88 weight=10;
}
```

**ip\_hash**（访问 ip）

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 favresin{
    ip_hash;
    server 10.0.0.10:8080;
    server 10.0.0.11:8080;
}
```

**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。与 **weight** 分配策略类似。

```
upstream favresin{
    server 10.0.0.10:8080;
    server 10.0.0.11:8080;
    fair;
}
```

**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。

注意：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 resinserver{
    server 10.0.0.10:7777;
    server 10.0.0.11:8888;
    hash $request_uri;
    hash_method crc32;
}
```

**upstream** 还可以为每个设备设置状态值，这些状态值的含义分别如下：

**down** 表示单前的 **server** 暂时不参与负载。

**weight** 默认为 1.**weight** 越大，负载的权重就越大。

**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。

**fail\_timeout**：**max\_fails** 次失败后，暂停的时间。

**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。

```
upstream bakend{ #定义负载均衡设备的 Ip 及设备状态
    ip_hash;
    server 10.0.0.11:9090 down;
    server 10.0.0.11:8080 weight=2;
    server 10.0.0.11:6060;
    server 10.0.0.11:7070 backup;
}
```

## 003、nginx 负载均衡的五种算法

### 目录

- 1.round robin（默认）
- 2.weight
- 3. IP\_hash
- 4.url\_hash（第三方）
- 5. fair（第三方）

### 正文

#### 1.round robin（默认）#

轮询方式，依次将请求分配到各个后台服务器中，默认的负载均衡方式。  
适用于后台机器性能一致的情况。  
挂掉的机器可以自动从服务列表中剔除。

#### 2.weight#

根据权重来分发请求到不同的机器中，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;
}
```

### 3. IP\_hash#

根据请求者 ip 的 hash 值将请求发送到后台服务器中，可以保证来自同一 ip 的请求被打到固定的机器上，可以解决 session 问题。

例如：

```
upstream bakend {
ip_hash;
server 192.168.0.14:88;
server 192.168.0.15:80;
}
```

### 4. url\_hash（第三方）#

根据请求的 url 的 hash 值将请求分到不同的机器中，当后台服务器为缓存的时候效率高。

例如：

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 默认为 1.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

## 5. fair（第三方）#

根据后台响应时间来分发请求，响应时间短的分发的请求多。

例如：

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

参考：<https://blog.csdn.net/chenyulancn/article/details/70800991>