**NGINX**

# 一 nginx 简介

## 1.1 Nginx是什么

**Nginx** ("engine x") 是一个高性能的HTTP和 反向代理 服务器，也是一个 IMAP/POP3/SMTP 服务器。 Nginx 是由 Igor Sysoev 为俄罗斯访问量第二的 Rambler.ru 站点开发的，第一个公开版本0.1.0发布于2004年10月4日。其将源代码以类BSD许可证的形式发布，因它的稳定性、丰富的功能集、示例配置文件和低系统资源的消耗而闻名。2011年6月1日，nginx 1.0.4发布。

**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，并在一个BSD-like 协议下发行。由俄罗斯的程序设计师Igor Sysoev所开发，供俄国大型的入口网站及搜索引擎Rambler(俄文:Рамблер)使用。其特点是占有内存少，并发能力强，事实上nginx的并发能力确实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、新浪、网易、腾讯、 淘宝等。

## 1.2 Nginx能干什么

1：直接支持Rails和PHP的程序

2：作为HTTP反向代理服务器

3：作为负载均衡服务器

4：作为邮件代理服务器

5：帮助实现前端动静分

互联网飞速发展的今天,大用户量高并发已经成为互联网的主体.怎样能让一个网站能够承载几万个或几十万个用户的持续访问呢？这是一些中小网站急需解决的问题。用单机tomcat搭建的网站，在比较理想的状态下能够承受的并发访问量在150到200左右。按照并发访问量占总用户数量的5%到10%这样计算，单点tomcat网站的用户人数在1500到4000左右。对于一个为全国范围提供服务的网站显然是不够用的，为了解决这个问题引入了负载均衡方法。负载均衡就是一个web服务器解决不了的问题可以通过多个web服务器来平均分担压力来解决，并发过来的请求被平均分配到多个后台web服务器来处理，这样压力就被分解开来。

负载均衡服务器分为两种一种是通过硬件实现的负载均衡服务器，简称硬负载 例如：f5。另一种是通过软件来实现的负载均衡，简称软负载: 例如apache和nginx。硬负载和软负载相比前者作用的网络层次比较多可以作用到socket接口的数据链路层对发出的请求进行分组转发但是价格成本比较贵，而软负载作用的层次在http协议层之上可以对http请求进行分组转发并且因为是开源的所以几乎是0成本，并且阿里巴巴，京东等电商网站使用的都是Nginx服务器。

## 1.3 Nginx特点

高稳定、高性能、资源占用少、功能丰富、模块化结构、支持热部署,单台nginx服务器可以轻松支持10万个并发。

# 二 nginx 安装及基本的使用

## 2.1安装步骤

1：需要gcc，系统自带了，没有的话，需要先安装

2：需要pcre，安装的命令示例如下： yum install pcre\*

3：需要zlib，安装的命令示例如下：yum install zlibzlib-devel

4：如果需要支持ssl的话，安装OpenSSL，安装的命令示例如下： yum install opensslopenssl-devel

5：上<http://nginx.org/>去下载源码包，然后进行解压安装，示例如下：

1. 先解压源码包，然后进入到这个包里面
2. 安装命令示例如下：
4. 第一步：./configure--prefix=/usr/local/nginx

--withhttp\_stub\_status\_module

--with-http\_ssl\_module 如果提示确少啥，就加上相应的选项，比如缺少pcre的话，就加上 --with-pcre=/usr/common/temp/pcre-8.34 （当然我们这里是不缺东西的）

（4）第二步：构建完后生成了makefile文件,依次 make&&make install

## 2.2安装配置项

编译参数可能会根据版本的不同进行变化，./configure --help查看编译参数列表，常见的选项如下：

--prefix=<path> - 安装路径，如果没有指定，默认为/usr/local/nginx。

--sbin-path=<path> -nginx可执行命令的文件，如果没有指定，默认为<prefix>/sbin/nginx。 --conf-path=<path> - 在没有使用-c参数指定的情况下nginx.conf的默认位置，如果没有指定，默认 为<prefix>/conf/nginx.conf。

--pid-path=<path> -nginx.pid的路径，如果没有在nginx.conf中通过“pid”指令指定，默认为 <prefix>/logs/nginx.pid。

--lock-path=<path> -nginx.lock文件路径，如果没有指定，默认为<prefix>/logs/nginx.lock。 --error-log-path=<path> - 当没有在nginx.conf中使用“error\_log”指令指定时的错误日志位置， 如果没有指定，默认为<prefix>/logs/error.log。

--http-log-path=<path> - 当没有在nginx.conf中使用“access\_log”指令指定时的访问日志位置， 如果没有指定，默认为<prefix>/logs/access.log。

--user=<user> - 当没有在nginx.conf中使用“user”指令指定时nginx运行的用户，如果没有指定， 默认为“nobody”。

--group=<group> - 当没有在nginx.conf中使用“user”指令指定时nginx运行的组，如果没有指定， 默认为“nobody”。

--builddir=DIR - 设置构建目录。

--with-rtsig\_module-启用rtsig模块。

--with-select\_module –without-select\_module-如果在configure的时候没有发现kqueue, epoll, rtsig或/dev/poll其中之一，select模块始终为启用状态。

--with-poll\_module –without-poll\_module-如果在configure的时候没有发现kqueue, epoll, rtsig或/dev/poll其中之一，poll模块始终为启用状态。

--with-http\_ssl\_module-启用ngx\_http\_ssl\_module，启用SSL支持并且能够处理HTTPS请求。需要 OpenSSL，在Debian系统中，对应的包为libssl-dev。

--with-http\_realip\_module-启用ngx\_http\_realip\_module

--with-http\_addition\_module-启用 ngx\_http\_addition\_module

--with-http\_sub\_module-启用ngx\_http\_sub\_module

--with-http\_dav\_module-启用ngx\_http\_dav\_module

--with-http\_flv\_module-启用ngx\_http\_flv\_module

--with-http\_stub\_status\_module-启用”serverstatus”（服务状态）页 --without-http\_charset\_module-禁用ngx\_http\_charset\_module

--without-http\_gzip\_module-禁用 ngx\_http\_gzip\_module，如果启用，需要zlib包。 --without-http\_ssi\_module-禁用ngx\_http\_ssi\_module

--without-http\_userid\_module-禁用ngx\_http\_userid\_module

--without-http\_access\_module-禁用ngx\_http\_access\_module --without-http\_auth\_basic\_module-禁用ngx\_http\_auth\_basic\_module

--without-http\_autoindex\_module-禁用ngx\_http\_autoindex\_module --without-http\_geo\_module-禁用ngx\_http\_geo\_module

--without-http\_map\_module-禁用ngx\_http\_map\_module

--without-http\_referer\_module-禁用ngx\_http\_referer\_module

--without-http\_rewrite\_module-禁用ngx\_http\_rewrite\_module。如果启用，需要PCRE包。

--without-http\_proxy\_module-禁用 ngx\_http\_proxy\_module

--without-http\_fastcgi\_module-禁用ngx\_http\_fastcgi\_module --without-http\_memcached\_module-禁用ngx\_http\_memcached\_module --without-http\_limit\_zone\_module-禁用ngx\_http\_limit\_zone\_module --without-http\_empty\_gif\_module-禁用ngx\_http\_empty\_gif\_module --without-http\_browser\_module-禁用ngx\_http\_browser\_module --without-http\_upstream\_ip\_hash\_module-禁用ngx\_http\_upstream\_ip\_hash\_module --with-http\_perl\_module-启用 ngx\_http\_perl\_module

--with-perl\_modules\_path=PATH - 为perl模块设置路径

--with-perl=PATH - 为perl库设置路径

--http-client-body-temp-path=PATH - 为http连接的请求实体临时文件设置路径，如果没有指定，默 认为<prefix>/client\_body\_temp

--http-proxy-temp-path=PATH - 为http代理临时文件设置路径，如果没有指定，默认为 <prefix>/proxy\_temp

--http-fastcgi-temp-path=PATH - 为httpfastcgi临时文件设置路径，如果没有指定，默认为 <prefix>/fastcgi\_temp

--without-http - 禁用HTTP服务

--with-mail - 启用IMAP4/POP3/SMTP代理模块

--with-mail\_ssl\_module-启用ngx\_mail\_ssl\_module

--with-cc=PATH - 设置C编译器路径

--with-cpp=PATH - 设置C预处理器路径

--with-cc-opt=OPTIONS - 变量CFLAGS中附加的参数，用于FreeBSD中的PCRE库，同样需要指定–withcc-opt=”-I /usr/local/include”，如果我们使用select()函数则需要同时增加文件描述符数量，可 以通过–with-cc-opt=”-D FD\_SETSIZE=2048”指定。

--with-ld-opt=OPTIONS - 通过连接器的附加参数，用于FreeBSD中的PCRE库，同样需要指定–withld-opt=”-L /usr/local/lib”。

--with-cpu-opt=CPU - 指定编译的CPU，可用的值为: pentium, pentiumpro, pentium3, pentium4, athlon, opteron, amd64, sparc32, sparc64, ppc64

--without-pcre-禁用PCRE库文件，同时将禁用HTTP rewrite 模块，如果要在”location”指令中使 用正则表达式，同样需要PCRE库。

--with-pcre=DIR - 设置PCRE库源文件路径。

--with-pcre-opt=OPTIONS - 在编译时为PCRE设置附加参数。

--with-md5=DIR - 设置md5库源文件路径。

--with-md5-opt=OPTIONS - 在编译时为md5设置附加参数。

--with-md5-asm - 使用md5汇编源。

--with-sha1=DIR - 设置sha1库源文件路径。

--with-sha1-opt=OPTIONS - 在编译时为sha1设置附加参数。

--with-sha1-asm - 使用sha1汇编源。

--with-zlib=DIR - 设置zlib库源文件路径。

--with-zlib-opt=OPTIONS - 在编译时为zlib设置附加参数。

--with-zlib-asm=CPU - 为指定的CPU使用zlib汇编源进行优化，可用值为: pentium, pentiumpro。

--with-openssl=DIR - 设置openssl库源文件路径。

--with-openssl-opt=OPTIONS - 在编译时为openssl设置附加参数。

--with-debug - 启用debug记录。

--add-module=PATH - 增加一个在PATH中的第三方模块

## 2.3测试运行

测试配置文件： 安装路径下的/nginx/sbin/nginx-t

启动： 安装路径下的/nginx/sbin/nginx

停止 安装路径下的/nginx/sbin/nginx-s stop 或者是： nginx-s quit

重启 安装路径下的/nginx/sbin/nginx-s reload

查看进程 ps-ef|grepnginx

安装过后，如果从外面访问不了，多半是被防火墙挡住了，可以关闭掉防火墙： /sbin/service iptables stop

# 三 nginx 基本配置

## 3.1基本配置文件

默认启动Nginx时，使用的配置文件是：安装路径/conf/nginx.conf 文件 可以在启动nginx的时候，通过-c来指定要读取的配置文件

常见的配置文件有如下几个：

nginx.conf：应用程序的基本配置文件

mime.types：MIME类型关联的扩展文件

fastcgi.conf：与fastcgi相关的配置

proxy.conf：与proxy相关的配置

sites.conf：配置Nginx提供的网站，包括虚拟主机

## 3.2 Nginx的进程结构

启动Nginx的时候，会启动一个Master进程，这个进程不处理任何客户端的 请求，主要用来产生worker进程，一个worker进程用来处理一个request。 n Nginx模块分为：核心模块、事件模块、标准Http模块、可选Http模块、邮件模 块、第三方模块和补丁等

Nginx基本模块：所谓基本模块，指的是Nginx默认的功能模块，它们提供的指 令，允许你使用定义Nginx基本功能的变量，在编译的时候不能被禁用，包括： 核心模块：基本功能和指令，如进程管理和安全 事件模块：在Nginx内配置网络使用的能力 配置模块：提供包含机制

常见的核心模块指令，大部分都是放置在配置文件的顶部 具体的指令，请参看nginx的官方文档，

<http://www.howtocn.org/nginx:directiveindex>

常见的events模块指令，大部分都是放置在配置文件的顶部 具体的指令，在上面那个文档里面，命令的context为events的就是events 模块指令，只能在events里面使用

# 四 核心，日志，事件常用指令

## 4.1核心模块的指令

error\_log 错误日志

Include 包含进来的文件

Pid 记录进程

User 指定用户，包括日志访问路径的权限等.指定谁来运行nginx

worker\_cpu\_affinity

Work\_connection 每一个进程可以连接的数目。

worker\_processes 最大工作进程数

error\_log 日志有6个级别：debug|info|notice|warn|error|crit

Nginx支持将不同的虚拟主机的日志记录在不同的地方，如下示例：

http{

error\_log logs/http\_error.log error;

server{

server\_name one;

access\_log logs/one\_access.log;

error\_log logs/one\_error.log error;

}

server{

server\_name two;

access\_log logs/two\_access.log;

error\_log logs/two\_error.log error;

}

}

注意：error\_log off不是禁用日志，而是创建一个名为off的日志，要禁用日志，可以这么写：error\_log /dev/null crit;

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"';



如上图所示

$remote\_addr 访问的ip

$remote\_user 请求的用户

$time\_local 访问的时间

$request 请求方式

$status 响应码

$body\_bytes\_sent"$http\_referer" 请求源

http\_x\_forwarded\_for:在经过代理时,代理把你的本来IP加在此头信息中,传输你的原始IP

等等，如上是nginx的日志，默认是main格式的日志，放在相对路劲logs文件夹下。

# 五 http模块配置与常用指令

## 5.1 核心配置区域

// 全局区

worker\_processes 1; // 有1个工作的子进程,可以自行修改,但太大无益,因为要争夺CPU,一般设置为 CPU数\*核数

Event {

// 一般是配置nginx连接的特性

// 如1个worker能同时允许多少连接

worker\_connections 1024; // 这是指 一个子进程最大允许连1024个连接

}

http {

//这个是协议级别

include mime.types;

default\_type application/octet-stream;

keepalive\_timeout 65;

gzip on;

server {

//这个是服务器级别

listen 80;

server\_name localhost;

location / {

//这个是请求级别

root html;

index index.html index.htm;

}

}

}

## 5.2 区分虚拟主机

### 5.2.1 域名的方式

server {

listen 80; #监听端口

server\_name a.com; #监听域名

location / {

root/var/www/a.com; #根目录定位

indexindex.html;

}

}

server {

listen 80; #监听端口

server\_name a.b.com; #监听域名

location / {

root/var/www/a.com; #根目录定位

indexindex.html;

}

}

### 5.2.2 端口号的方式

server {

listen 8080;

server\_name 192.168.1.204;

location / {

root /var/www/html8080;

index index.html;

}

}

server {

listen 8081;

server\_name 192.168.1.204;

location / {

root /var/www/html8080;

index index.html;

}

}

## 5.3 location区域

Location区段，通过指定模式来与客户端请求的URI相匹配，基本语法如下： location [=|~|~\*|^~|@] pattern{……}

### 5.2.1 / : 模糊匹配

server {

server\_name localhost;

location /abc{

……

}

}

那么，如下是对的：

<http://xxxxxx.com/abc>

<http://xxxxxx.com/abc?p1=11&p2=22>

<http://xxxxxx.com/abc>

<http://xxxxxx.com/abcde>

### 5.2.2 = : 精确匹配

server {

server\_name localhost;

location = /abc {

……

}

}

那么，如下是对的：

<http://xxxxxxx/abc>

<http://xxxxxxx/abc/?p1=11&p2=22>

如下是错的：

<http://xxxxxxx/abc>

<http://xxxxxxx/abcde>

### 5.2.3 ~ : 正则表达式区分大小写

server {

server\_name localhost;

location ~ ^/abc$ {

……

}

}

那么，如下是对的：

<http://xxxxxxx/abc>

<http://xxxxxxx/abc?p1=11&p2=22>

如下是错的：

<http://xxxxxxx/ABC>

<http://xxxxxxx/abc/>

<http://xxxxxxx/abcde>

### 5.2.4 ~\*:正则表达式不区分大小写

server {

server\_name sishuok.com;

location ~\* ^/abc$ {

……

}

}

那么，如下是对的：

http://xxxxxxxxx/abc

http://xxxxxxxxx/ABC

http://xxxxxxxxx/abc?p1=11&p2=22

如下是错的：

http://xxxxxxxxx/abc/

http://xxxxxxxxx/abcde

### 5.2.5:^~ 模糊匹配

不同的是，如果模式匹配， 那么就停止搜索其他模式了。

### 5.2.6 @ ：内部location区段

这些区段客户端不能访问，只可以由内部产生的请 求来访问，如try\_files或error\_page等

### 5.2.7 查找顺序

1：带有“=“的精确匹配优先

2：没有修饰符的精确匹配

3：正则表达式按照他们在配置文件中定义的顺序

4：带有“^~”修饰符的，开头匹配

5：带有“~”或“~\*” 修饰符的，如果正则表达式与URI匹配

6：没有修饰符的，如果指定字符串与URI开头匹配

## 5.4 location 案例

### 5.4.1 案例一

location = / { # 只匹配 / 的查询. [ configuration A ] }

location / { # 匹配任何以 / 开始的查询，但是正则表达式与一些较长的字符串将被首先匹配。 [ configuration B ] }

location ^~ /images/ { # 匹配任何以 /images/ 开始的查询并且停止搜索，不检查正则表达式。 [ configuration C ] }

location ~\* \.(gif|jpg|jpeg)$ { # 匹配任何以gif, jpg, or jpeg结尾的文件，但是所有 /images/ 目录的请求将在Configuration C中处 理。 [ configuration D ] }

各请求的处理如下例：

■/ → configuration A

■/documents/document.html → configuration B

■/images/1.gif → configuration C

■/documents/1.jpg → configuration D

### 5.4.2 案例二

location = / {

root /var/www/html/;

index index.htm index.html;

}

location / {

root /usr/local/nginx/html;

index index.html index.htm;

}

访问　　<http://xxx.com>/

定位流程是

1: 精准匹配中　”/”   ,得到index页为index.htm,是location的

2: 再次访问 /index.htm , 内部转跳uri已经是”/index.htm” ,一般匹配 “/”

根目录为/usr/local/nginx/html

3: 最终结果,访问了 /usr/local/nginx/html/index.htm

4: 返回结果如果linux下/usr/local/nginx/html目录下有index.htm就返回没有报404

### 5.4.3 案例三

location / {//模糊

root /usr/local/nginx/html;

index index.html index.htm;

}

location ~ image {//相对准确

root /var/www/image;

index index.html;

}

访问 http://xx.com/image/logo.png

1此时, “/” 与”/image/logo.png” 都能匹配

2图片真正会访问 /var/www/image/logo.png

3默认正则的优先级高于一般匹配

### 5.4.4 案例四

location / {//模糊

root /usr/local/nginx/html;

index index.html index.htm;

}

location /foo {//相对准确

root /var/www/html;

index index.html;

}

访问 http://xxx.com/foo

1对于uri “/foo”,   两个location的patt,都能匹配他们

2即‘/’能从左前缀匹配‘/foo’, ‘/foo’也能左前缀匹配’/foo’,

3此时, 真正访问 /var/www/html/index.html

4匹配的最高原则是：按目录最精细，准确的来定位，按访问的域来匹配

5取域的最小交集作为访问路径（存在特殊）。

# 六 反向代理，负载均衡，动静分离

## 6.1 正向代理

没有代理的时候，客户端访问服务器是直接访问的，为了找到专门过滤请求，或让请求排队的中间人我们找来了代理。比如我们的翻墙的软件，抢票的软件，它都是为客户端服务的。这个类型的代理我们就称之为正向代理。

真正的服务器

代理服务器

如上图所示，三个小人不直接请求原来的服务器了，而是通过代理服务器请求服务器，这就是正向代理，然而现在的问题是所有的请求都压在了一个服务器上了，那么怎么办呢。此时我们就要增加真实的服务器。

## 6.2 反向代理

真正的服务器

代理服务器

真正的服务器

真正的服务器

随着用户量的增加，我们需要在后端增加服务器，这时候客户端不知道怎么访问到每台响应的服务器，就需要把服务器的信息告诉代理，让代理服务器代为转发，此时代理服务器主要是为服务端转发服务的就是我们所说的反向代理了。

6.3 反向代理配置

upstream apachephp  {

    server 192.168.232.131:8080;

}

server {

    listen 80;

    server\_name  www.quancha.cn;

    access\_log  logs/quancha.access.log  main;

    error\_log   logs/quancha.error.log;

    root   html;

    index  index.html index.htm index.php;

    ## send request back to apache ##

    location / {

        proxy\_pass  http://apachephp;

   }

}

如上代理配置nginx监听的[www.quancha.cn:8080转发到了192.168.232.131:8080](http://www.quancha.cn:8080转发到了192.168.232.131:8080)端口<http://apachephp>作为代理的媒介用来做转发的依据这就是反向代理的配置。

## 6.3 动静分离

Nginx实现动静分离，其实就是在反向代理的时候，如果是静态资源，那么就直 接从Nginx发布的路径去读取，或者从静态资源服务器中读取，而不需要从动态资源服务器获取了，或者甚至不需要访问后台服务器。但是要注意：这种情况下需要保证后端跟前端的程序保持一致，可以使用Rsync 做服务端自动同步或者使用NFS、MFS分布式共享存储。

Nginx服务器

动态资源服务器

静态资源服务器

如上图所示，我们把项目的静态资源放在nginx服务器上直接返回给客户端，而动态解析的jsp则放在了后台服务器上。

upstream statics{

server s101:80 weight=1; #访问配置了静态页面的nginx

server s102:80 weight=1; #访问配置了静态页面的nginx

server s103:80 weight=1; #访问配置了静态页面的nginx

}

upstream tomcats{

server s101:8080 weight=1; #访问tomcat

server s102:8080 weight=1; #访问tomcat

server s103:8080 weight=1; #访问tomcat

}

server{

listen 80;

server\_name s100;

access\_log off;

location ~\* \.(png|html|js|css)$ {

proxy\_pass http://statics;

#所有以.png .html .js .css结尾的url进入此路径

}

location / {

proxy\_pass http://tomcats;

#其它url进入此路径

}

}

主机s100，s101, s102上分别配置nginx 和 tomcat(略)

server{

listen 80;

server\_name s101;

location / {

root html; #存放了静态页面的根目录

index index.html index.htm; #主页

}

}

## 6.4 负载均衡

Nginx通过upstream模块来实现简单的负载均衡

在upstream块内，定义一个服务器列表，默认的方式是轮询，如果要确定同一个 访问者发出的请求总是由同一个后端服务器来处理，可以设置ip\_hash，如： upstream myapp1 {

ip\_hash;

server localhost:18080;

server localhost:18081;

server localhost:18082;

}

server {

listen 8000;

location / {

proxy\_pass http://myapp1;

}

}

如上所示myapp1映射到upstream以后采用轮询的方式访问另外的三个服务器。

请注意：这个方法本质还是轮询，而且由于客户端的ip可能是不断变化的，比如动 态ip，代理，翻墙等等，因此ip\_hash并不能完全保证同一个客户端总是由同一 个服务器来处理。

## 6.5 GEO&&GEOIP

这两个模块主要用于做全局的负载均衡，可以根据不同的客户端ip来访问不同的 服务器，示例如下：

http{

geo $geo{

default default;

202.103.10.1/24 A;

179.9.0.3/24 B;

}

upstream default.server{

server 192.168.0.100;

}

upstream A.server{

server 192.168.0.101;

}

upstream B.server{

server 192.168.0.102;

}

server{

listen 80;

location / {

proxy\_passhttp://$geo.server$request\_uri;

}

}

}

# 七 常见功能配置总结

## 7.1 server配置为监听ip和端口

server{

listen 127.0.0.1:9080;

server\_name127.0.0.1;

}

## 7.2 server配置为监听域名和端口

server{

listen 80;

server\_name www.sishuok.com sishuok.com\*.sishuok.com;

}

## 7.3 向后台服务器传递客户端的真实ip

location ~ \.(jsp|action|mvc)$ {

proxy\_set\_headerHost $host;

proxy\_set\_headerX-Forwarded-For $remote\_addr;

proxy\_pass http://sishuok.com;

}

## 7.4 在负载均衡里面，实现后端服务器故障转移的配置

location ~ \.(jsp|action|mvc)$ {

proxy\_next\_upstream http\_502 http\_504 timeout;

proxy\_set\_headerHost $host;

proxy\_set\_headerX-Forwarded-For $remote\_addr;

proxy\_passhttp://server\_pool;

}

## 7.5 简单的防盗链

location / {

……

valid\_referersblocked sishuok.com\*.sishuok.com;

if($invalid\_referer){

rewrite ^/ http://sishuok.com;

}

}

## 7.6 简单的限制下载速度

location / {

limit\_rate 256K;

}

# 八 优化配置

## 8.1 参数配置

### 8.1.1 用户和组

生产环境下，最好是专为Nginx创建用户和组，并单独设置权限，这 样会更安全。例如： user nginxnginx

### 8.1.2 worker\_processes

通常配置成cpu的总核数，或者其2倍，性能会更好。这可以 减少进程间切换带的消耗。还可以同时使用worker\_cpu\_affinity来绑定cpu，使得每个worker进程独享一个 cpu，实现完全的并发，性能更好，不过这个只对linux系统有效

### 8.1.3 worker\_rlimit\_nofile

描述一个nginx进程打开的最多的文件数目。配置成跟 linux内核下文件打开数一致就可以了。可以通过ulimit-n 来查看，新装的系 统默认是1024，CentOS中可以如下方式进行修改：

在/etc/security/limits.conf最后增加：

\* soft nofile 65535

\* hard nofile 65535

\* soft nproc 65535

\* hard nproc 65535

### 8.1.4 worker\_connections

每个进程允许的最多连接数，默认是1024，可以设置大一些。 理论上并发总数是worker\_processes和worker\_connections的乘积， worker\_connections值的设置跟物理内存大小有关，因为系统可以打开的最大文件数和内 存大小成正比，一般1GB内存的机器上可以打开的文件数大约是10万左右，所以， 8.1.5

### 8.1.5 worker\_connections

的值需根据 worker\_processes 进程数目和系统可以打开的最大文件 总数进行适当地进行设置。

### 8.1.6 keepalive\_timeout

设置到65左右就可以

### 8.1.7 client\_header\_buffer\_size

设置请求的缓存，设置为4k，通常为系统分页大小的整数 倍，可以通过getconfPAGESIZE 来查看系统分页大小。

对打开文件设置缓存 open\_file\_cachemax=建议设置成和每个进程打开的最大文件数一致 inactive=60s; open\_file\_cache\_valid 90s; open\_file\_cache\_min\_uses2;

### 8.1.8 open\_file\_cache\_errorson

尽量开启Gzip压缩，gzip\_comp\_level通常设置成3-5，高了浪费CPU n Error日志优化：运行期间设置为crit，可以减少I/O

### 8.1.9 access日志优化

如果使用了其他统计软件，可以关闭日志，来减少磁盘写，或者写入内 存文件，提高I/O效率。

### 8.1.10 sendfile

指令指定 nginx 是否调用 sendfile 函数（zero copy 方式）来输出文件，通常 应设置成on，如果是下载等应用磁盘IO重负载应用，可设置为 off

Buffers size优化：如果buffer size太小就会到导致nginx使用临时文件存储response， 这会引起磁盘读写IO，流量越大问题越明显。

### 8.1.11 client\_body\_buffer\_size

处理客户端请求体buffer大小。用来处理POST提交数 据，上传文件等。

### 8.1.12 client\_body\_buffer\_size

需要足够大以容纳需要上传的POST数据。同 理还有后端的buffer数据。

### 8.1.13 worker\_priority

进程优先级设置：Linux系统中，优先级高的进程会占用更多的系统资 源，这里配置的是进程的静态优先级，取值范围-20到+19，-20级别最高。因此可以把这个 值设置小一点，但不建议比内核进程的值低（通常为-5） 合理设置静态资源的浏览器缓存时间，尽量用浏览器缓存

负载均衡锁accept\_mutex，建议开启，默认就是开启的

如果使用SSL的话，而且服务器上有SSL硬件加速设备的话，请开启硬件加速。