第二十八章：Nginx应用详解及配置

**一、Nginx简介；**

**二、Nginx实现原理；**

**三、Nginx支持高并发的原因；**

**四、案例：搭建Nginx网站服务；**

**一、Nginx简介；**

**概述：**Nginx是一款由俄罗斯开发的开源的高性能HTTP服务器和反向代理服务器，同时支持IMAP/POP3/SMTP代理服务，其性能优势着为显著，官网上称：单台nginx服务器可以处理50000并发；

**特点：**高性能、稳定、消耗硬件资源小、能够处理大并发，主要用于静态的解析，动静页面的分离；

**优势：**

1.作为Web服务器，nginx处理静态文件、索引文件以及自动索引效率非常高。

2.作为代理服务器，Nginx可以实现无缓存的反向代理加速，提高网站运行速度。

3.作为负载均衡服务器，Nginx既可以在内部直接支持Rails和PHP，也可以支持HTTP代理服务器，对外进行服务。同时支持简单的容错和利用算法进行负载均衡。

1.在性能方面，Nginx在实现上非常注重效率。它采用内核Poll模型，可以支持更多的并发连接，最大可以支持对50 000个并发连接数的响应，而且占用很低的内存资源。

2.在稳定性方面，Nginx采取了分阶段资源分配技术，使得对CPU与内存的占用率非常低。Nginx官方表示Nginx保持10 000个没有活动的连接，这些连接只占2.5M内存，因此，类似DOS这样的攻击对Nginx来说基本上是没有任何作用的。

3.在高可用性方面，Nginx支持热部署，启动速度特别迅速，因此可以在不间断服务的情况下，对软件版本或者配置进行升级，即使运行数月也无需重新启动，几乎可以做到7\*24小时的不间断运行。

**二、Nginx实现原理；**

**Nginx核心组件：**

  核心模块：HTTP模块、EVENT事件模块、MAIL模块。

基础模块：HTTP Access模块、HTTP FastCGI模块、HTTP Proxy模块、HTTP Rewrite模块

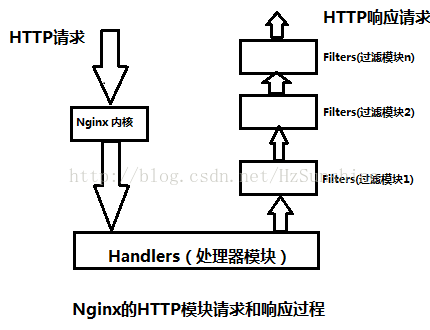
第三方模块：HTTP Upstream Request Hash模块、Notice模块、HTTP Access Key模块。

**Nginx模块分类（基于功能）：**

Handlers：处理器模块，此类模块直接处理请求，并进行输出内容和修改headers信息等操作。Handlers处理器模块一般只能有一个。

Filters：过滤器模块，此类模块主要对其他处理器模块输出的内容进行修改操作，最后由Nginx输出。

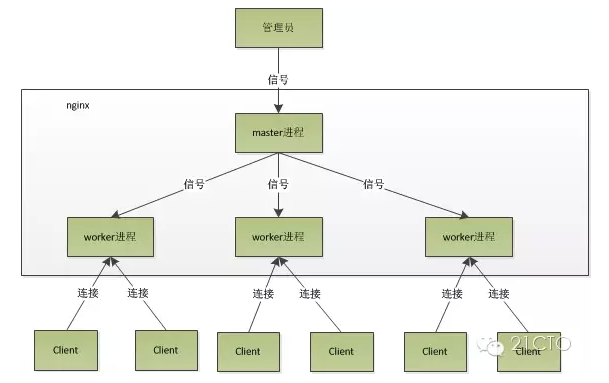
Proxies：代理类模块，此类模块是Nginx的HTTP Upstream之类的模块，这些模块主要与后端一些服务比如FastCGI等进行交互，实现服务代理和负载均衡等功能。



**Nginx的进程模型：**

单工作进程模式：除主进程外，还有一个工作进程，工作进程是单线程的，默认为此模式；

多工作进程模式：每个工作进程包含多个线程；



**master进程：**

1.接收外界传递给Nginx的信号，进而管理服务的状态等；

2.管理worker进程，向各worker进程发送信号，监控worker进程的运行状态，当worker进程异常情况下退出后，会自动重新启动新的worker进程；

3.master进程充当整个进程组与用户的交互接口，同时对进程进行监护。它不需要处理网络事件，不负责业务的执行，只会通过管理worker进程来实现重启服务、平滑升级、更换日志文件、配置文件实时生效等功能。

**worker进程：**

1.处理基本的网络事件，多个worker进程之间是对等的，他们同等竞争来自客户端的请求，各进程互相之间是独立的。一个请求，只可能在一个worker进程中处理，一个worker进程，不可能处理其它进程的请求。worker进程的个数是可以设置的，一般我们会设置与机器cpu核♥数一致；

**扩展：**

http://blog.csdn.net/hguisu/article/details/8930668 ## nginx实现原理

**三、Nginx支持高并发的原因；**

**I/O模型之select：**

1.每个连接对应一个描述。select模型受限于 FD\_SETSIZE（即进程最大打开的描述符数），linux2.6.35为1024,实际上linux每个进程所能打开描数字的个数仅受限于内存大小，然而在设计select的系统调用时，却是参考FD\_SETSIZE的值。可通过重新编译内核更改此值，但不能根治此问题，对于百万级的用户连接请求即便增加相应进程数，仍显得杯水车薪；  
2、select每次请求都会扫描一个文件描述符的集合，这个集合的大小是作为select第一个参数传入的值。但是每个进程所能打开文件描述符若是增加了，扫描的效率也将减小；  
3、内核到用户空间，采用内存复制方式传递信息，这样就增加了不必要的复制延迟；  
**I/O模型之epoll模型：**

1.请求无文件描述字大小限制，仅与内存大小相关；

2.epoll返回时已经明确的知道哪个socket fd发生了什么事件，不用像select那样再一个个比对；

3.内核到用户空间，采用共享内存方式传递消息，使用mmap加速内核与用户空间的消息传递；

**apache：**Apache 2.2.9之前只支持select模型，2.2.9之后支持epoll模型；

**Nginx：**支持epoll模型；

**四、案例：搭建Nginx网站服务；**

**案例环境：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统类型 | IP地址 | 主机名 | 所需软件 | 硬件 |
| Centos 6.5 64bit | 192.168.100.150 | www.linuxfan.cn | nginx-1.12.2.tar.gz | 内存：2G  CPU：8核 |

**案例步骤：**

* 安装nginx程序；
* 优化nginx服务并启动服务；
* 客户端访问测试；
* 开启nginx的状态监听模块；
* 客户端访问nginx的状态监听界面；
* 企业级优化Nginx服务；
* 访问测试优化后nginx服务；
* 安装webbench压力测试工具，进行测试nginx性能；
* 自主学习：Nginx服务器内核优化；
* **安装nginx程序；**

[root@www ~]# rpm -e httpd --nodeps

[root@www ~]# yum -y install pcre-devel zlib-devel

[root@www ~]# useradd -M -s /sbin/nologin nginx

[root@www ~]# tar zxvf nginx-1.12.2.tar.gz -C /usr/src/

[root@www ~]# cd /usr/src/nginx-1.12.2/

[root@www nginx-1.12.2]# ./configure --prefix=/usr/local/nginx --user=nginx --group=nginx --with-http\_stub\_status\_module

注解：

--prefix=/usr/local/nginx ##指定安装位置

--user=nginx --group=nginx ##指定运行服务的用户和组

--with-http\_stub\_status\_module ##开启状态监听模块

--conf-path= ##指向配置文件存放位置

--error-log-path= ##指向错误日志存放位置

--pid-path= ##指向pid文件存放位置

--with-rtsig\_module ##启用rtsig模块支持（实时信号）

--with-select\_module ##启用select模块支持（一种轮询模式,不推荐在高载环境下使用）禁用：--without-select\_module

--with-http\_ssl\_module ##启用ngx\_http\_ssl\_module支持（使支持https请求，需已安装openssl）

--with-http\_xslt\_module ##启用ngx\_http\_xslt\_module支持（过滤转换XML请求）

--with-http\_image\_filter\_module ##启用ngx\_http\_image\_filter\_module支持（传输JPEG/GIF/PNG 图片的一个过滤器）（默认为不启用，要用到gd库）

--with-http\_gzip\_static\_module ##启用ngx\_http\_gzip\_static\_module支持（在线实时压缩输出数据流）

--with-http\_degradation\_module   ##启用ngx\_http\_degradation\_module支持（允许在内存不足的情况下返回204或444码）

--without-http\_access\_module ##禁用ngx\_http\_access\_module支持（该模块提供了一个简单的基于主机的访问控制，允许或拒绝基于ip地址）

--without-http\_auth\_basic\_module ##禁用ngx\_http\_auth\_basic\_module（该模块是可以使用用户名和密码基于http基本认证方法，来保护你的站点或其部分内容）

---without-http\_rewrite\_module ##禁用ngx\_http\_rewrite\_module支持（该模块允许使用正则表达式改变URL）

--without-http\_fastcgi\_module ##禁用ngx\_http\_fastcgi\_module支持（该模块允许Nginx 与FastCGI 进程交互，并通过传递参数来控制FastCGI 进程工作。）

[root@www nginx-1.12.2]# make &&make install

[root@www nginx-1.12.2]# ls /usr/local/nginx/

client\_body\_temp conf fastcgi\_temp html logs proxy\_temp sbin scgi\_temp uwsgi\_temp

[root@www nginx-1.12.2]# cd

* **优化nginx服务并启动服务；**

[root@www ~]# ln -s /usr/local/nginx/sbin/nginx /usr/local/sbin/ ##优化命令执行路径

[root@www ~]# vi /etc/init.d/nginx

#!/bin/bash

# chkconfig: - 99 20

# description: Nginx Server Control Script

NP="/usr/local/nginx/sbin/nginx"

NPF="/usr/local/nginx/logs/nginx.pid"

case "$1" in

start)

$NP;

if [ $? -eq 0 ]

then

echo "nginx is starting!! "

fi

;;

stop)

kill -s QUIT $(cat $NPF)

if [ $? -eq 0 ]

then

echo "nginx is stopping!! "

fi

;;

restart)

$0 stop

$0 start

;;

reload)

kill -s HUP $(cat $NPF)

if [ $? -eq 0 ]

then

echo "nginx config file is reload! "

fi

;;

\*)

echo "Usage: $0 {start|stop|restart|reload}"

exit 1

esac

exit 0

[root@www ~]# chmod +x /etc/init.d/nginx

[root@www ~]# chkconfig --add nginx

[root@www ~]# chkconfig nginx on

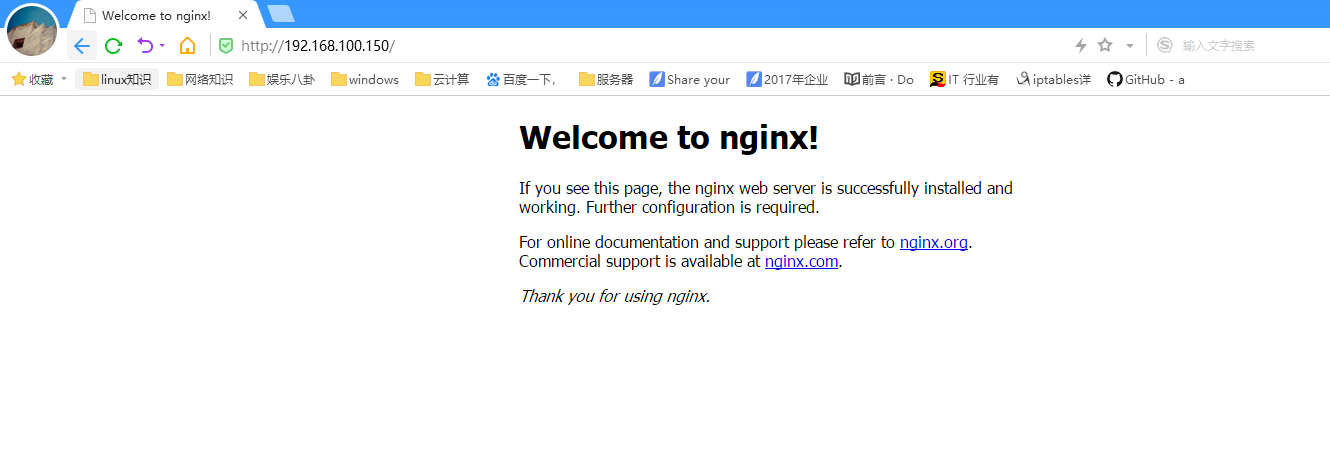
[root@www ~]# /etc/init.d/nginx start

nginx is starting!!

[root@www ~]# netstat -utpln |grep nginx

tcp 0 0 0.0.0.0:80 0.0.0.0:\* LISTEN 3713/nginx

* **客户端访问测试；**



* **开启nginx的状态监听模块；**

[root@www ~]# vi /usr/local/nginx/conf/nginx.conf ##编辑配置文件在server中添加如下行：

47 location /status {

48 stub\_status on;

49 access\_log off;

50 }

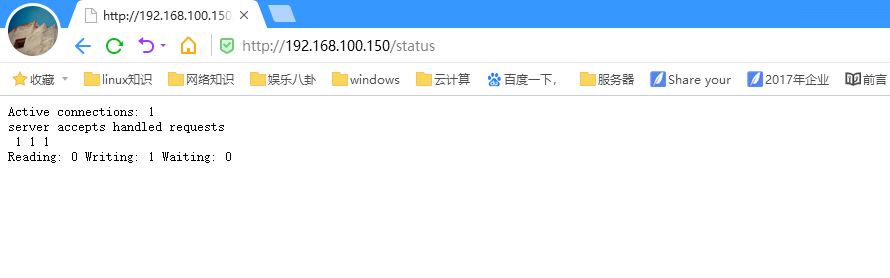
[root@www ~]# /etc/init.d/nginx restart

nginx is stopping!!

nginx is starting!!

* **客户端访问nginx的状态监听界面；**

http://192.168.100.150/status



活动的连接数

已处理的连接数 成功的tcp握手次数 已处理的请求数

* **企业级优化Nginx服务；**

[root@www ~]# vi /usr/local/nginx/conf/nginx.conf

worker\_processes 8;

worker\_cpu\_affinity 00000001 00000010 00000100 00001000 00010000 00100000 01000000 10000000;

error\_log /usr/local/nginx/logs/nginx\_error.log crit;

pid /usr/local/nginx/logs/nginx.pid;

worker\_rlimit\_nofile 204800;

events {

use epoll;

worker\_connections 204800;

}

http {

include mime.types;

default\_type application/octet-stream;

charset utf-8;

server\_names\_hash\_bucket\_size 128;

client\_header\_buffer\_size 2k;

large\_client\_header\_buffers 4 4k;

client\_max\_body\_size 8m;

sendfile on;

tcp\_nopush on;

keepalive\_timeout 60;

fastcgi\_cache\_path /usr/local/nginx/fastcgi\_cache levels=1:2

keys\_zone=TEST:10m inactive=5m;

fastcgi\_connect\_timeout 300;

fastcgi\_send\_timeout 300;

fastcgi\_read\_timeout 300;

fastcgi\_buffer\_size 4k;

fastcgi\_buffers 8 4k;

fastcgi\_busy\_buffers\_size 8k;

fastcgi\_temp\_file\_write\_size 8k;

fastcgi\_cache TEST;

fastcgi\_cache\_valid 200 302 1h;

fastcgi\_cache\_valid 301 1d;

fastcgi\_cache\_valid any 1m;

fastcgi\_cache\_min\_uses 1;

fastcgi\_cache\_use\_stale error timeout invalid\_header http\_500;

open\_file\_cache max=204800 inactive=20s;

open\_file\_cache\_min\_uses 1;

open\_file\_cache\_valid 30s;

tcp\_nodelay on;

gzip on;

gzip\_min\_length 1k;

gzip\_buffers 4 16k;

gzip\_http\_version 1.0;

gzip\_comp\_level 2;

gzip\_types text/plain application/x-javascript text/css application/xml;

gzip\_vary on;

log\_format access '$remote\_addr - $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" $http\_x\_forwarded\_for';

server {

listen 80;

server\_name www.linuxfan.cn;

location / {

root /usr/local/nginx/html/;

index index.html index.htm;

}

location /status {

stub\_status on;

access\_log off;

}

location ~ .\*\.(gif|jpg|jpeg|png|bmp|swf|js|css)$ {

expires 30d;

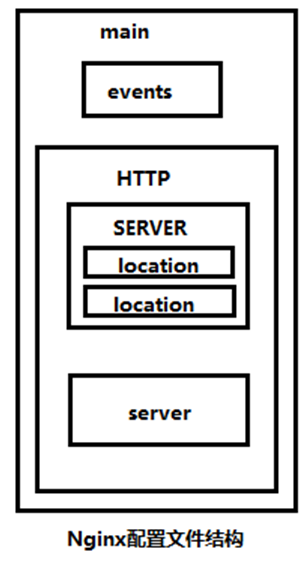
}

access\_log /usr/local/nginx/logs/access.log access;

}

}

注解：



worker\_processes 8; ##设置worker进程数量

worker\_cpu\_affinity 00000001 00000010 00000100 00001000 00010000 00100000 01000000 10000000; ##设置每个worker进程对应一个cpu的核心

error\_log /usr/local/nginx/logs/nginx\_error.log crit; ##指定错误日志

pid /usr/local/nginx/logs/nginx.pid; ##指定运行时产生的pid文件

worker\_rlimit\_nofile 204800; ##指定nginx进程最多能够打开多少个文件描述符，通常与系统中的ulimit -n保持一致；

events { ##事件区域配置

use epoll; ##指定处理模型为epoll

worker\_connections 4096; ##每个进程最多能够处理多少个连接，进程数量\*此参数=并发数量

}

http { ##http服务配置区域

include mime.types; ##指定文件扩展名和文件类型映射表

default\_type application/octet-stream; ##指定文件类型

charset utf-8; ##指定字符集

server\_names\_hash\_bucket\_size 128; ##服务器名字的hash表大小

client\_header\_buffer\_size 2k; ##客户端请求头部buffer大小

large\_client\_header\_buffers 4 4k; ##指定客户端请求中较大的消息头的缓存数量和大小

client\_max\_body\_size 8m; ##指定客户端请求的单个文件的最大字节数

sendfile on; ##开启高效传输模式

tcp\_nopush on; ##防止网络阻塞

keepalive\_timeout 60; ##客户端连接超时时间

#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。

fastcgi\_cache\_path /usr/local/nginx/fastcgi\_cache levels=1:2 ##配置fastcgi缓存路径和目录结构等级

keys\_zone=TEST:10m inactive=5m; ##关键字区域存储时间和非活动删除时间

fastcgi\_connect\_timeout 300; ##连接到后端FastCGI的超时时间

fastcgi\_send\_timeout 300; ##向FastCGI传送请求的超时时间

fastcgi\_read\_timeout 300; ##接收FastCGI应答的超时时间

fastcgi\_buffer\_size 4k; ##指定读取FastCGI应答第一部分需要多大的缓冲区

fastcgi\_buffers 8 4k; ##指定本地需要用多少和多大的缓冲区来缓冲FastCGI的应答请求

fastcgi\_busy\_buffers\_size 8k; ##通常为fastcgi\_buffer\_size大小的两倍

fastcgi\_temp\_file\_write\_size 8k; ##写入缓存文件时使用多大的数据块，大小同上

fastcgi\_cache TEST; ##开启Fastcgi的缓存并且为其指定一个名称

fastcgi\_cache\_valid 200 302 1h; ##指定不同的状态码，其缓存的时间

fastcgi\_cache\_valid 301 1d;

fastcgi\_cache\_valid any 1m;

fastcgi\_cache\_min\_uses 1; ##URL经过被访问多少次将被缓存

fastcgi\_cache\_use\_stale error timeout invalid\_header http\_500; ##指定什么情况下不进行缓存

open\_file\_cache max=204800 inactive=20s; ##指定缓存文件最大数量，经过多长时间文件没有被请求后则删除缓存，

open\_file\_cache\_min\_uses 1; ##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，文件更改信息一直是在缓存中打开的；

open\_file\_cache\_valid 30s; ##指定多长时间检查一次缓存的有效信息，检查该缓存的源文件是否发生变化修改等；

tcp\_nodelay on; ## nagle算法，有需要发送的就立即发送，连接转换为长连接时使用；

gzip on; ##开启gzip压缩

gzip\_min\_length 1k; ##指定最小压缩文件的大小

gzip\_buffers 4 16k; ##指定压缩缓冲区的个数和大小

gzip\_http\_version 1.0; ##指定压缩版本

gzip\_comp\_level 2; ##指定压缩等级1-9，9等级最高

gzip\_types text/plain application/x-javascript text/css application/xml; ##指定压缩文件类型

gzip\_vary on; ##前端缓存服务器缓存经过压缩的页面

log\_format access '$remote\_addr - $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" $http\_x\_forwarded\_for';

##配置日志格式，具体变量表示请结合百度，日志格式为access

server {

listen 80;

server\_name www.linuxfan.cn;

location / {

root /usr/local/nginx/html/;

index index.html index.htm;

}

location /status {

stub\_status on;

access\_log off;

}

location ~ .\*\.(gif|jpg|jpeg|png|bmp|swf|js|css)$ {

expires 30d; ##指定以上格式的文件将进行缓存

}

access\_log /usr/local/nginx/logs/access.log access;

}

}

注解：location语法讲解

语法规则： location [=|~|~\*|^~] /uri/ { … }  
=：表示精确匹配,这个优先级也是最高的

^~：表示url以某个常规字符串开头，理解为匹配 url路径即可。

~ ：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(和上面的唯一区别就是大小写)

!~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

/ ：通用匹配，任何请求都会匹配到，默认匹配.

优先级排序： = > ^~ >

常用的正则表达式：

\* ##重复0次或者多次

? ##重复0次或者1次

+ ##重复1次或多次

. ##匹配除换行符号之外的所有字符

(a|b) ##匹配a或b

^ ##以...开头

$ ##以...结尾

{n} ##重复n次

{n,} ##重复最少n次

{n,m} ##重复n到m次

\W ##匹配任意不是字母，数字，下划线，汉字的字符

\S ##匹配任意不是空白的字符

\D ##匹配任意非数字的字 符

\B ##匹配不是单词开头或结束的位置

[^x] ##匹配除了x以外的所有字符

[^abcd] ##匹配除了a或b或c或d以外的所有字符

^[abcd] ##匹配除了字符串abcd以外的所有字符

[root@www~]# /etc/init.d/nginx start

nginx: [warn] no "fastcgi\_cache\_key" for "fastcgi\_cache" in /usr/local/nginx/conf/nginx.conf:75

nginx: [warn] no "fastcgi\_cache\_key" for "fastcgi\_cache" in /usr/local/nginx/conf/nginx.conf:75

nginx: [warn] no "fastcgi\_cache\_key" for "fastcgi\_cache" in /usr/local/nginx/conf/nginx.conf:75

nginx is starting!!

[root@www ~]# netstat -utpln |grep nginx

tcp 0 0 0.0.0.0:80 0.0.0.0:\* LISTEN 3627/nginx

[root@www ~]# ps aux |grep nginx |grep -v grep

root 3627 0.0 0.0 30708 376 ? Ss 18:59 0:00 nginx: master process /usr/local/nginx/sbin/nginx

nginx 3628 0.1 11.6 113920 57600 ? S 18:59 0:00 nginx: worker process

nginx 3629 0.0 11.1 113920 54932 ? S 18:59 0:00 nginx: worker process

nginx 3630 0.0 9.7 113920 48336 ? S 18:59 0:00 nginx: worker process

nginx 3631 0.0 9.6 113920 47768 ? S 18:59 0:00 nginx: worker process

nginx 3632 0.0 9.8 113920 48516 ? S 18:59 0:00 nginx: worker process

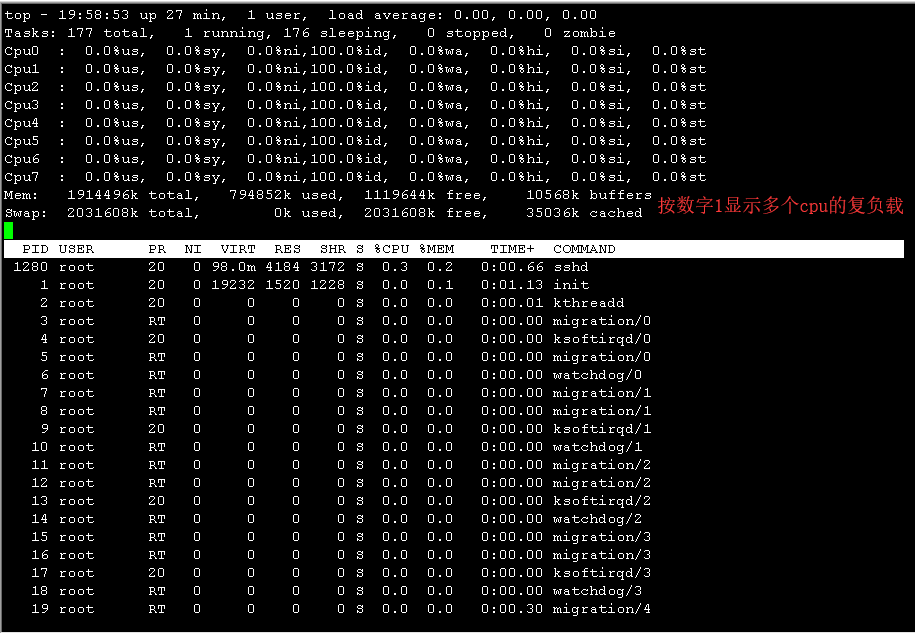
nginx 3633 0.0 10.5 113920 52084 ? S 18:59 0:00 nginx: worker process

nginx 3634 0.0 11.5 113920 57156 ? S 18:59 0:00 nginx: worker process

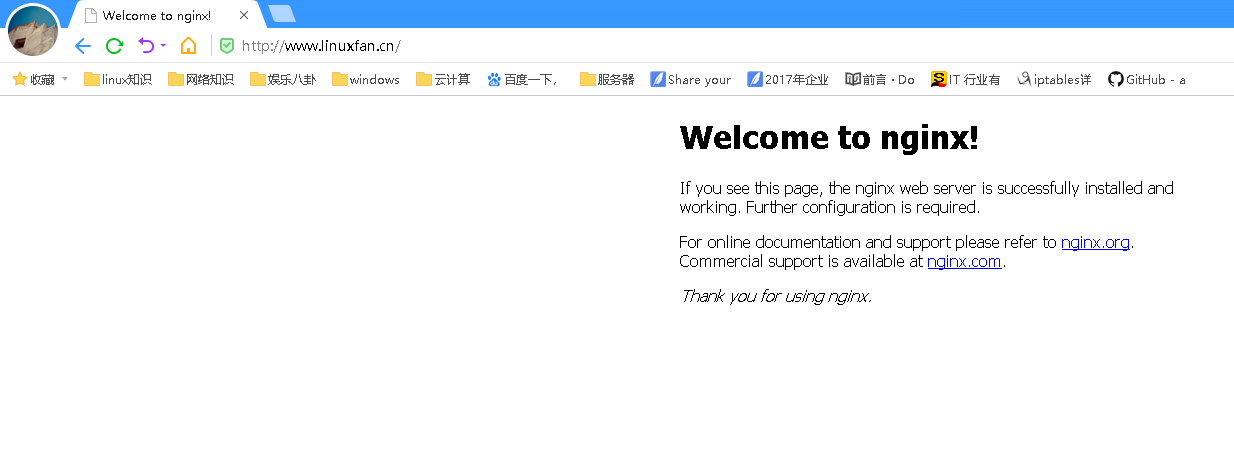
nginx 3635 0.0 9.4 113920 46892 ? S 18:59 0:00 nginx: worker process

nginx 3636 0.0 0.0 30844 392 ? S 18:59 0:00 nginx: cache manager process

[root@www ~]# top



* **访问测试优化后nginx服务；**



* **安装webbench压力测试工具，进行测试nginx性能；**

[root@www ~]# yum -y install gcc ctags

[root@www ~]# wget http://home.tiscali.cz/~cz210552/distfiles/webbench-1.5.tar.gz

[root@www ~]# tar zxvf webbench-1.5.tar.gz -C /usr/src/

[root@www ~]# cd /usr/src/webbench-1.5/

[root@www webbench-1.5]# mkdir /usr/local/man

[root@www webbench-1.5]# make && make install

[root@www webbench-1.5]# cd

[root@www~]# webbench -c 10000 -t 5 http://www.linuxfan.cn:80/index.html ##并发数为10000，时间为5秒

Webbench - Simple Web Benchmark 1.5

Copyright (c) Radim Kolar 1997-2004, GPL Open Source Software.

Benchmarking: GET http://www.linuxfan.cn:80/index.html

10000 clients, running 5 sec.

Speed=147744 pages/min, 1397500 bytes/sec.

Requests: 12312 susceed, 0 failed.

[root@www ~]# netstat -nat | awk '/^tcp/ {++S[$NF]} END {for(key in S) print key,"\t",S[key]}' ##查看数据包状态

TIME\_WAIT 15961 ##表示收到了对方的FIN报文，并发送出了ACK报文

FIN\_WAIT1 166 ##已发送FIN报文，等待对方的ACK

SYN\_SENT 189 ##这个状态与SYN\_RCVD遥相呼应，用于建立连接

FIN\_WAIT2 1 ##半关闭状态

ESTABLISHED 1343 ##已经建立连接

SYN\_RECV 256 ##已经接收到对方的SYN报文，等待ACK报文

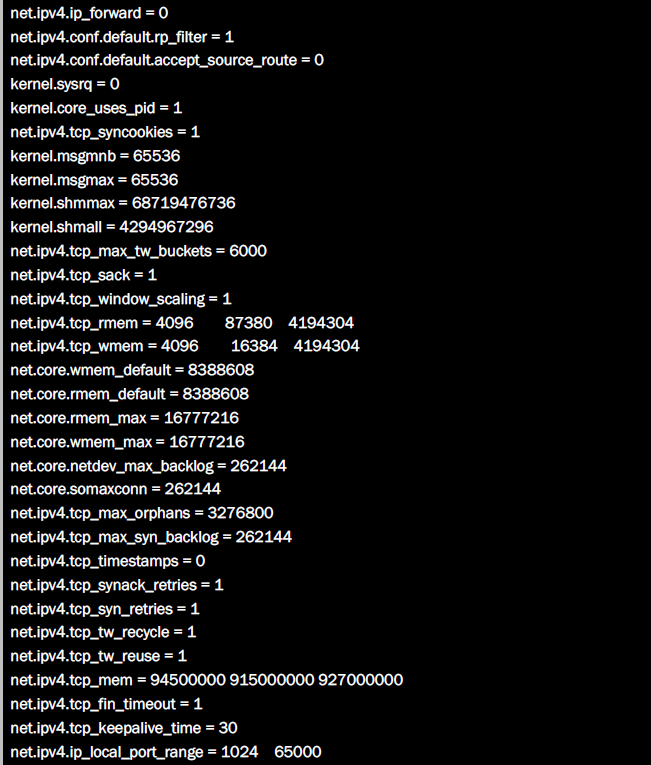
LISTEN 9 ##监听状态

浏览器访问监控界面刷新测试：



* **自主学习：Nginx服务器内核优化；**

[root@www ~]# vi /etc/sysctl.conf



[root@www ~]# sysctl -p