

公告

昵称：为了这有限的生命
园龄：4年8个月
粉丝：39
关注：7
[+加关注](#)

<	2016年4月						>
日	一	二	三	四	五	六	
27	28	29	30	31	1	2	
3	4	5	6	7	8	9	
10	11	12	13	14	15	16	
17	18	19	20	21	22	23	
24	25	26	27	28	29	30	
1	2	3	4	5	6	7	

搜索

找找看

谷歌搜索

常用链接

- 我的随笔
- 我的评论
- 我的参与
- 最新评论
- 我的标签

我的标签

jquery(1)

随笔分类

- ajax
- G(6)
- G++(1)
- oi(1)
- GSS(3)
- GSS hack(1)
- drupal(28)
- drupal 模块(2)
- ecshop(2)
- java(15)
- java_web(1)
- javascript(24)

Nginx优化（十七）

【教程主题】：[Nginx优化](#)

【课程录制】：[创E](#)

【主要内容】

Nginx 优化

nginx介绍

Nginx是俄罗斯人编写的十分轻量级的HTTP服务器,Nginx，它的发音为“engine X”，是一个高性能的HTTP和反向代理服务器，同时也是一个IMAP/POP3/SMTP 代理服务器．Nginx是由俄罗斯人 Igor Sysoev为俄罗斯访问量第二的 Rambler.ru站点开发.

Nginx以事件驱动（epoll）的方式编写，所以有非常好的性能，同时也是一个非常高效的反向代理、负载平衡。其拥有匹配 Lighttpd的性能，同时还没有Lighttpd的内存泄漏问题，而且Lighttpd的mod_proxy也有一些问题并且很久没有更新。但是Nginx并不支持cgi方式运行，原因是可以减少因此带来的一些程序上的漏洞。所以必须使用FastCGI方式来执行PHP程序。

nginx做为HTTP服务器，有以下几项基本特性：

处理静态文件，索引文件以及自动索引；打开文件描述符缓冲。

无缓存的反向代理加速，简单的负载均衡和容错。

FastCGI，简单的负载均衡和容错。

模块化的结构。包括gziping, byte ranges, chunked responses,以及 SSI-filter等filter。如果由FastCGI或其它代理服务器处理单页中存在的多个SSI，则这项处理可以并行运行，而不需要相互等待。

Nginx专为性能优化而开发，性能是其最重要的考量,实现上非常注重效率。它支持内核ePoll模型，能经受高负载的考验,有报告表明能支持高达 50,000个并发连接数。

Nginx具有很高的稳定性。其它HTTP服务器，当遇到访问的峰值，或者有人恶意发起慢速连接时，也很可能会导致服务器物理内存耗尽频繁交换，失去响应，只能重启服务器。例如当前apache一旦上到200个以上进程，web响应速度就明显非常缓慢了。而Nginx采取了分阶段资源分配技术，使得它的CPU与内存占用率非常低。nginx官方表示保持10,000个没有活动的连接，它只占2.5M内存，所以类似DOS这样的攻击对nginx来说基本上是毫无用处的。就稳定性而言,nginx比lighthttpd更胜一筹。

Nginx支持热部署。它的启动特别容易,并且几乎可以做到7*24不间断运行，即使运行数个月也不需要重新启动。你还能够在不间断服务的情况下，对软件版本进行进行升级。

ab的使用

- java编程思想(1)
- java基础
- linux(27)
- linux 服务器安全(16)
- mysql(11)
- new_java(9)
- php(34)
- PHP 全局函数(1)
- phpcms(15)
- php核心(2)
- php模块小结(9)
- symfony(1)
- thinkphp(3)
- zend(1)
- 大型网站核心技术笔记(6)
- 电脑基础知识(1)
- 服务器(30)
- 工具方案(1)
- 算法、数据结构、设计模式(2)
- 碎片(17)
- 网络基础
- 闲言碎语(7)
- 一段段代码(2)

随笔档案

- 2015年6月 (5)
- 2015年5月 (4)
- 2015年4月 (5)
- 2015年3月 (29)
- 2015年2月 (1)
- 2014年12月 (5)
- 2014年11月 (2)
- 2014年10月 (1)
- 2014年9月 (1)
- 2014年8月 (3)
- 2014年7月 (2)
- 2014年6月 (19)
- 2014年5月 (9)
- 2014年4月 (4)
- 2014年3月 (7)
- 2014年2月 (8)
- 2014年1月 (11)
- 2013年12月 (12)
- 2013年11月 (5)
- 2013年10月 (3)
- 2013年9月 (5)
- 2013年8月 (28)
- 2013年7月 (20)
- 2013年6月 (12)
- 2013年5月 (5)
- 2013年4月 (6)
- 2013年3月 (3)
- 2013年2月 (2)
- 2013年1月 (4)
- 2012年12月 (3)
- 2012年11月 (3)
- 2012年10月 (2)
- 2012年9月 (21)

[root@localhost bin]# ab -n 10 -c 10 <http://opslinux.com/>

意思是这样的：

-n表示发送多少个请求，

-c表示一次发送多少个（实际上就是把-n分批发送），

后面跟地址， 注意后的斜杠。

返回信息如下（红色部分为我的注释）：

This is ApacheBench, Version 2.0.40-dev <\$Revision: 1.146 \$> apache-2.0
Copyright 1996 Adam Twiss, Zeus Technology Ltd, <http://www.zeustech.net/>
Copyright 2006 The Apache Software Foundation, <http://www.apache.org/>
Benchmarking http://www.yi1.com.cm/ (be patient).....done
Server Software: Apache/2.2.4
Server Hostname: <http://opslinux.com/>
Server Port: 80
Document Path: /
Document Length: 31848 bytes
Concurrency Level: 10
Time taken for tests: 1.722254 seconds/*测试持续时间*/
Complete requests: 10/*完成请求数量*/
Failed requests: 0/*失败请求数量*/
Write errors: 0
Total transferred: 323490 bytes/*总流量*/
HTML transferred: 318480 bytes/*HTML传输量*/
Requests per second: 5.81 [#/sec] (mean)/*每秒事务数*/
Time per request: 1722.254 [ms] (mean)/*平均响应时间*/
Time per request: 172.225 [ms] (mean, across all concurrent requests)/*每个请求响应时间（平均）*/
Transfer rate: 182.90 [Kbytes/sec] received/*传输效率*/
Connection Times (ms)
min mean[+/-sd] median max
Connect: 165 166 1.2 167 168
Processing: 1300 1418 91.5 1427 1554
Waiting: 803 925 92.9 929 1064
Total: 1465 1585 92.2 1595 1721
Percentage of the requests served within a certain time (ms)
50% 1595/*50%的请求响应时间小于1595*/
66% 1620/*66%的请求响应时间小于1620*/
75% 1668
80% 1706
90% 1721
95% 1721
98% 1721
99% 1721
100% 1721 (longest request)/*最长响应时间1721*/

编译安装过程优化

在编译Nginx时，默认以debug模式进行，而在debug模式下会插入很多跟踪和ASSERT之类的信息，编译完成后，一个Nginx要有好几兆字节。在编译前取消Nginx的debug模式，编译完成后Nginx只有几百千字节，因此可以在编译之前，修改相关源码，取消debug模式，具体方法如下： 在Nginx源码文件被解压后，找到源码目录下的auto/cc/gcc文件，在其中找到如下几行:

```
# debug
```

```
CFLAGS="$CFLAGS -g"
```

注释掉或删除

为特定的CPU指定CPU类型编译优化

- 2012年8月 (19)
- 2012年7月 (8)
- 2012年6月 (19)
- 2011年12月 (1)

文章分类

ecshop 源码分析
linux(1)

相册

one(2)

最新评论

1. Re:linux rsync介绍 （八）
感谢分享。请问uid和gid和后面auth
user什么关系？

--itfanr
2. Re:在ubuntu中出现Call to
undefined function:
mysql_connect()
谢谢，有用~

--yuanww
3. Re:浏览器检测
如何判断本地是否安装某某exe？

--Mask-male
4. Re:nginx 编译安装
@keep--movingzlib-devel 是开发包
zlib 是程序包...

--HumpSmart
5. Re:addslashes与
mysql_real_escape_string的区别
<?php echo
mysql_real_escape_string("fdsafd
a'fdsa"); 这个我执行并没有报错，
这个函数不需要先连接数据库啊...

--[] 神、风

阅读排行榜

1. nginx 编译安装(17369)
2. addslashes与
mysql_real_escape_string的区别
(10568)
3. phpcms数据库操作(7967)
4. JAVA基本类库介绍(5866)
5. linux 压缩 解压缩 详解(5636)

评论排行榜

1. nginx 编译安装(2)
2. addslashes与
mysql_real_escape_string的区别
(2)
3. 浏览器检测(1)
4. linux rsync介绍 （八） (1)
5. linux 压缩 解压缩 详解(1)

推荐排行榜

1. addslashes与
mysql_real_escape_string的区别
(2)
2. JS 高级总结(2)
3. PHP无限级分类的实现 （不使用递
归） (1)
4. php核心知识要点(1)
5. linux服务器的性能分析与优化 （十
三） (1)

在编译Nginx时，默认的GCC编译参数是“-O”，要优化GCC编译，可以使用以下两个参数:

```
--with-cc-opt='-O3'
```

```
--with-cpu-opt=CPU    #为特定的 CPU  编译，有效的值包
```

括: pentium, pentiumpro, pentium3, pentium4, athlon, opteron, amd64, sparc32, sparc64, ppc64

要确定CPU类型，可以通过如下命令:

```
[root@localhost home]#cat /proc/cpuinfo | grep "model name"
```

利用TCMalloc优化Nginx的性能

TCMalloc的全称为Thread-Caching Malloc，是谷歌开发的开源工具“google-perftools”中的一个成员。与标准的glibc库的malloc相比，TCMalloc库在内存分配效率和速度上要高很多，这在很大程度上提高了服务器在高并发情况下的性能，从而降低系统负载。下面简单介绍如何为Nginx添加TCMalloc库支持。 要安装TCMalloc库，需要安装libunwind（32位操作系统不需要安装）和google-perftools两个软件包，libunwind库为基于64位CPU和操作系统的程序提供了基本函数调用链和函数调用寄存器功能。下面介绍利用TCMalloc优化Nginx的具体操作过程:

1.安装libunwind库

可以从http://download.savannah.gnu.org/releases/libunwind下载相应的libunwind版本，这里下载的是libunwind-0.99-alpha.tar.gz，安装过程如下

```
[root@localhost home]#tar zxvf libunwind-0.99-alpha.tar.gz
```

```
[root@localhost home]# cd libunwind-0.99-alpha/
```

```
[root@localhost libunwind-0.99-alpha]#CFLAGS=-fPIC ./configure
```

```
[root@localhost libunwind-0.99-alpha]#make CFLAGS=-fPIC
```

```
[root@localhost libunwind-0.99-alpha]#make CFLAGS=-fPIC install
```

2.安装google-perftools

可以从http://google-perftools.googlecode.com下载相应的google-perftools版本，这里下载的是google-perftools-1.8.tar.gz，安装过程如下:

```
[root@localhost home]# tar zxvf google-perftools-1.8.tar.gz
```

```
[root@localhost home]# cd google-perftools-1.8/
```

```
[root@localhost google-perftools-1.8]# ./configure
```

```
[root@localhost google-perftools-1.8]# make && make install
```

```
[root@localhost google-perftools-1.8]# echo "/usr/local/lib" > /etc/ld.so.conf.d/usr_local_lib.conf
```

```
[root@localhost google-perftools-1.8]# ldconfig
```

至此，google-perftools安装完成。

3.重新编译Nginx

为了使Nginx支持google-perftools，需要在安装过程中添加“- with-google_perftools_module”选项重新编译Nginx，安装代码如下:

```
[root@localhostnginx-0.7.65]#
```

```
./configure --user=www --group=www --prefix=/usr/local/nginx --with-http_stub_status_module --with-http_ssl_module --with-http_gzip_static_module --with-ipv6 --with-google_perftools_module
```

```
[root@localhost nginx-0.7.65]#make
```



```
[root@localhost nginx-0.7.65]#make install
```

到这里Nginx安装完成。

4.为google-perftools添加线程目录

创建一个线程目录，这里将文件放在/tmp/tcmalloc下，操作如下:

```
[root@localhost home]#mkdir /tmp/tcmalloc
```

```
[root@localhost home]#chmod 0777 /tmp/tcmalloc
```

5.修改Nginx主配置文件

修改nginx.conf文件，在pid这行的下面添加如下代码:

```
#pid          logs/nginx.pid;
```

```
google_perftools_profiles /tmp/tcmalloc;
```

接着，重启Nginx，完成google-perftools的加载。

6.验证运行状态

为了验证google-perftools已经正常加载，通过如下命令查看:

```
[root@ localhost home]# lsof -n | grep tcmalloc
```

```
nginx      2395  nobody   9w  REG      8,8        0    1599440 /tmp/tcmalloc.2395
```

```
nginx      2396  nobody  11w  REG      8,8        0    1599443 /tmp/tcmalloc.2396
```

```
nginx      2397  nobody  13w  REG      8,8        0    1599441 /tmp/tcmalloc.2397
```

```
nginx      2398  nobody  15w  REG      8,8        0    1599442 /tmp/tcmalloc.2398
```

由于在Nginx配置文件中，设置worker_processes的值为4，因此开启了4个Nginx线程，每个线程会有一行记录。每个线程文件后面的数字值就是启动的Nginx的PID值。至此，利用TCMalloc优化Nginx的操作完成。

Nginx内核参数优化

内核参数的优化，主要是在Linux系统中针对Nginx应用而进行的系统内核参数优化，常见的优化参数值如下。

下面给出一个优化实例以供参考:

```
net.ipv4.tcp_max_tw_buckets = 6000
```

```
net.ipv4.ip_local_port_range = 1024 65000
```

```
net.ipv4.tcp_tw_recycle = 1
```

```
net.ipv4.tcp_tw_reuse = 1
```

```
net.ipv4.tcp_syncookies = 1
```

```
net.core.somaxconn = 262144
```

```
net.core.netdev_max_backlog = 262144
```

```
net.ipv4.tcp_max_orphans = 262144
```

```
net.ipv4.tcp_max_syn_backlog = 262144
```

```
net.ipv4.tcp_synack_retries = 1
```

```
net.ipv4.tcp_syn_retries = 1
```

```
net.ipv4.tcp_fin_timeout = 1
```

```
net.ipv4.tcp_keepalive_time = 30
```

将上面的内核参数值加入/etc/sysctl.conf文件中，然后执行如下命令使之生效:

```
[root@ localhost home]#/sbin/sysctl -p
```

下面是对实例中选项的含义进行介绍:

net.ipv4.tcp_max_tw_buckets参数用来设定timewait的数量，默认是180000，这里设为6000。

net.ipv4.ip_local_port_range选项用来设定允许系统打开的端口范围。

net.ipv4.tcp_tw_recycle选项用于设置启用timewait快速回收。

net.ipv4.tcp_tw_reuse选项用于设置开启重用，允许将TIME-WAIT sockets重新用于新的TCP连接。

net.ipv4.tcp_syncookies选项用于设置开启SYN Cookies，当出现SYN等待队列溢出时，启用cookies进行处理。

net.core.somaxconn选项默认值是128，这个参数用于调节系统同时发起的tcp连接数，在高并发的请求中，默认的值可能会导致链接超时或者重传，因此，需要结合并发请求数来调节此值。

net.core.netdev_max_backlog选项表示当每个网络接口接收数据包的速率比内核处理这些包的速率快时，允许发送到队列的数据包的最大数目。

net.ipv4.tcp_max_orphans选项用于设定系统中最多有多少个TCP套接字不被关联到任何一个用户文件句柄上。如果超过这个数字，孤立连接将立即被复位并打印出警告信息。这个限制只是为了防止简单的DoS攻击。不能过分依靠这个限制甚至人为减小这个值，更多的情况是增加这个值。

net.ipv4.tcp_max_syn_backlog选项用于记录那些尚未收到客户端确认信息的连接请求的最大值。对于有128MB内存的系统而言，此参数的默认值是1024，对小内存的系统则是128。

net.ipv4.tcp_synack_retries参数的值决定了内核放弃连接之前发送SYN+ACK包的数量。

net.ipv4.tcp_syn_retries选项表示在内核放弃建立连接之前发送SYN包的数量。

net.ipv4.tcp_fin_timeout选项决定了套接字保持在FIN-WAIT-2状态的时间。默认值是60秒。正确设置这个值非常重要，有时候即使一个负载很小的Web服务器，也会出现因为大量的死套接字而产生内存溢出的风险。

net.ipv4.tcp_keepalive_time选项表示当keepalive启用的时候，TCP发送keepalive消息的频度。默认值是2（单位是小时）。

下面贴一个完整的内核优化设置:

vi /etc/sysctl.conf CentOS5.5中可以将所有内容清空直接替换为如下内容:

```
net.ipv4.ip_forward = 0
```

```
net.ipv4.conf.default.rp_filter = 1
```

```
net.ipv4.conf.default.accept_source_route = 0
```

```
kernel.sysrq = 0
```

```
kernel.core_uses_pid = 1
```

```
net.ipv4.tcp_syncookies = 1
```

```
kernel.msgmnb = 65536
```

```
kernel.msgmax = 65536
```

```
kernel.shmmax = 68719476736
```



```
kernel.shmall = 4294967296

net.ipv4.tcp_max_tw_buckets = 6000

net.ipv4.tcp_sack = 1

net.ipv4.tcp_window_scaling = 1

net.ipv4.tcp_rmem = 4096 87380 4194304

net.ipv4.tcp_wmem = 4096 16384 4194304

net.core.wmem_default = 8388608

net.core.rmem_default = 8388608

net.core.rmem_max = 16777216

net.core.wmem_max = 16777216

net.core.netdev_max_backlog = 262144

net.core.somaxconn = 262144

net.ipv4.tcp_max_orphans = 3276800

net.ipv4.tcp_max_syn_backlog = 262144

net.ipv4.tcp_timestamps = 0

net.ipv4.tcp_synack_retries = 1

net.ipv4.tcp_syn_retries = 1

net.ipv4.tcp_tw_recycle = 1

net.ipv4.tcp_tw_reuse = 1

net.ipv4.tcp_mem = 94500000 915000000 927000000

net.ipv4.tcp_fin_timeout = 1

net.ipv4.tcp_keepalive_time = 30

net.ipv4.ip_local_port_range = 1024 65000
```

配置文件优化

基本优化

一般来说nginx 配置文件中对优化比较有作用的为以下几项：

1. worker_processes 8;

nginx 进程数，建议按照cpu 数目来指定，一般为它的倍数 (如,2个四核的cpu计为8)。
2. worker_cpu_affinity 00000001 00000010 00000100 00001000 00010000 00100000 01000000 10000000;

为每个进程分配cpu，上例中将8 个进程分配到8 个cpu，当然可以写多个，或者将一个进程分配到多个cpu。
3. worker_rlimit_nofile 65535;

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。详见[ulimit关于系统连接数的优化](#)

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。

查看linux系统文件描述符的方法:

```
[root@web001 ~]# sysctl -a | grep fs.file
```



```
fs.file-max = 789972
```

```
fs.file-nr = 510 0 789972
```

```
4. use epoll;
```

使用epoll 的I/O 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。

)

```
5. worker_connections 65535;
```

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_processes*worker_connections。

```
6. keepalive_timeout 60;
```

keepalive 超时时间。

```
7. client_header_buffer_size 4k;
```

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。

```
8. open_file_cache max=65535 inactive=60s;
```

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。

```
9. open_file_cache_valid 80s;
```

这个是指多长时间检查一次缓存的有效信息。

```
10. open_file_cache_min_uses 1;
```

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。

简单配置文件

下面是一个简单的nginx 配置文件:

```
user www www;
```

```
worker_processes 8;

worker_cpu_affinity 00000001 00000010 00000100 00001000 00010000 00100000

01000000;

error_log /www/log/nginx_error.log crit;

pid /usr/local/nginx/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;

server

{

listen 8080;

server_name backup.aiju.com;

index index.php index.htm;

root /www/html/;

location /status

{

stub_status on;

}

location ~ .*\. (php|php5)?$

{

fastcgi_pass 127.0.0.1:9000;

fastcgi_index index.php;

include fcgi.conf;

}

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

{

expires 30d;

}

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

'$status $body_bytes_sent "$http_referer" '

'"$http_user_agent" $http_x_forwarded_for';

access_log /www/log/access.log access;

}

}
```

关于FastCGI 的几个指令:

fastcgi_cache_path /usr/local/nginx/fastcgi_cache levels=1:2 keys_zone=TEST:10minactive=5m;

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_connect_timeout 300;
```

指定连接到后端FastCGI 的超时时间。

```
fastcgi_send_timeout 300;
```

向FastCGI 传送请求的超时时间，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_read_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_buffer_size 4k;
```

指定读取FastCGI 应答第一部分需要用多大的缓冲区，一般第一部分应答不会超过1k，由于页面大小为4k，所以这里设置为4k。

```
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。

```
fastcgi_busy_buffers_size 8k;
```

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_temp_file_write_size 8k;
```

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_cache TEST
```

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。

```
fastcgi_cache_valid 200 302 1h; fastcgi_cache_valid 301 1d; fastcgi_cache_valid any 1m;
```

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_min_uses 1;
```

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_use_stale error timeout invalid_header http_500;
```

不知道这个参数的作用，猜想应该是让nginx 知道哪些类型的缓存是没用的。以上为nginx 中FastCGI 相关参数，另外，FastCGI 自身也有一些配置需要进行优化，如果你使用php-fpm 来管理FastCGI，可以修改配置文件中的以下值：

再来优化PHP-FPM，打开/usr/local/php/etc/php-fpm.conf，这个文件和PHP的语法很相似，凡是需要激活的配置，直接删掉前面的分号（;）即可：

```
1. [global]
2.
3. pid = run/php-fpm.pid
4.
5. process_control_timeout=5
6.
7. [www]
8.
9. listen.allowed_clients = 127.0.0.1
```



```
10.
11. user=www-data
12.
13. group=www-data
14.
15. pm=dynamic
16.
17. pm.max_children=20（这个配置决定了php-fpm的总进程数，内存小的少设点）
18.
19. pm.max_requests=10000（并发数越大，此请求数应越大）
20.
21. pm.start_servers =10（初始php-fpm进程数）
22.
23. emergency_restart_threshold = 60
24.
25. emergency_restart_interval = 60s
```

上边这两个，表示在emergency_restart_interval所设值内出现SIGSEGV或者SIGBUS错误的php-cgi进程数如果超过 emergency_restart_threshold个，php-fpm就会优雅重启。这两个选项一般保持默认值

ulimit关于系统连接数的优化

linux 默认值 open files 和 max user processes 为 1024

```
#ulimit -n
```

```
1024
```

```
#ulimit -u
```

```
1024
```

问题描述： 说明 server 只允许同时打开 1024 个文件，处理 1024 个用户进程

使用ulimit -a 可以查看当前系统的所有限制值，使用ulimit -n 可以查看当前的最大打开文件数。

新装的linux 默认只有1024 ，当作负载较大的服务器时，很容易遇到error: too many open files 。因此，需要将其改大。

解决方法：

使用 ulimit -n 65535 可即时修改，但重启后就无效了。（注ulimit -SHn 65535 等效 ulimit -n 65535 ， -S 指soft ， -H 指hard)

修改方式

有如下三种修改方式：

1. 在/etc/rc.local 中增加一行 ulimit -SHn 65535

2. 在/etc/profile 中增加一行 ulimit -SHn 65535

3. 在/etc/security/limits.conf 最后增加:

```
4. * soft nofile 65535
```

```
5. * hard nofile 65535
```

```
6. * soft nproc 65535
```

```
7. * hard nproc 65535
```

具体使用哪种，在 CentOS 中使用第1 种方式无效果，使用第3 种方式有效果，而在Debian 中使用第2 种有效果


```
# ulimit -n
```

```
65535
```

```
# ulimit -u
```

```
65535
```

备注：ulimit 命令本身就有分软硬设置，加-H 就是硬，加-S 就是软默认显示的是软限制

soft 限制指的是当前系统生效的设置值。 hard 限制值可以被普通用户降低。但是不能增加。 soft 限制不能设置的比 hard 限制更高。 只有 root 用户才能够增加 hard 限制值。

分类: [服务器](#)

好文要顶

关注我

收藏该文







[为了这有限的生命](#)
[关注 - 7](#)
[粉丝 - 39](#)


[+加关注](#)

« 上一篇: [Nginx+php （十六）](#)
» 下一篇: [Drupal Nginx伪静态设置方法](#)

posted @ 2014-06-20 16:44 为了这有限的生命 阅读(3269) 评论(0) 编辑 收藏

努力加载评论中...

[刷新评论](#) [刷新页面](#) [返回顶部](#)

 注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

- 【推荐】50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库
- 【推荐】融云即时通讯云－豆果美食、Faceu等亿级APP都在用



最新IT新闻

-
- 96万！特斯拉Model X国内首次亮相
- Windows 10 Build 14328正式推送 带来一大波更新
- Ubuntu 16.04 LTS现已正式支持IBM LinuxONE与z Systems
- GitHub正经历一场全面的动荡，我们有高管和员工离开的完整内幕
- 中国火星探测任务立项：迈向火星之路有多难？
- » 更多新闻...



最新知识库文章

·

- 架构漫谈（九）：理清技术、业务和架构的关系
 - 架构漫谈（八）：从架构的角度看如何写好代码
 - 架构漫谈（七）：不要空设架构师这个职位，给他实权
 - 架构漫谈（六）：软件架构到底是要解决什么问题？
 - 架构漫谈（五）：什么是软件
- » 更多知识库文章...

历史上的今天:

- 2013-06-20 抽象类
- 2012-06-20 bind9 详细解析
- 2012-06-20 DNS和DHCP服务器