一、nigxin配置文件优化

设置nginx进程数，推荐按照cpu数目来指定，一般跟cpu核数相同。

worker\_processes 8;

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。

worker\_cpu\_affinity 0001 0010 0011 0100 0101 0110 0111 1000;

配置nginx进程打开的最多文件数目，理论值应该是系统的最多打开文件数（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。

worker\_rlimit\_nofile 65535;

使用epoll的I/O模型，用这个模型来高效处理异步事件

use epoll;

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。

worker\_connections 65535;

http连接超时时间，默认是60s，功能是使客户端到服务器端的连接在设定的时间内持续有效，当出现对服务器的后继请求时，该功能避免了建立或者重新建立连接。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，一般一个请求的头部大小不会超过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。

client\_header\_buffer\_size 4k;

下面这个参数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=102400 inactive=20s;

下面这个是指多长时间检查一次缓存的有效信息。

open\_file\_cache\_valid 30s;

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。

open\_file\_cache\_min\_uses 1;

隐藏响应头中的有关操作系统和web server（Nginx）版本号的信息，这样对于安全性是有好处的。

server\_tokens off;

可以让sendfile()发挥作用。sendfile()可以在磁盘和TCP socket之间互相拷贝数据(或任意两个文件描述符)。Pre-sendfile是传送数据之前在用户空间申请数据缓冲区。之后用read()将数据从文件拷贝到这个缓冲区，write()将缓冲区数据写入网络。sendfile()是立即将数据从磁盘读到OS缓存。因为这种拷贝是在内核完成的，sendfile()要比组合read()和write()以及打开关闭丢弃缓冲更加有效(更多有关于sendfile)。

sendfile on;

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;

比如：

http {

server\_tokens off;

sendfile on;

tcp\_nopush on;

tcp\_nodelay on;

…

}

客户端请求头部的缓冲区大小，这个可以根据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。

client\_header\_buffer\_size 4k;

客户端请求头部的缓冲区大小，这个可以根据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@test-huanqiu ~]# 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。

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;

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。

open\_file\_cache\_min\_uses 1;

指定多长时间检查一次缓存的有效信息。

open\_file\_cache\_valid 80s;

========================================================================

如下一个运维日常用的nginx标准配置文件

[root@dev-huanqiu ~]# cat /usr/local/nginx/conf/nginx.conf

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 65535;

events

{

use epoll;

worker\_connections 65535;

}

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 16k;

fastcgi\_buffers 16 16k;

fastcgi\_busy\_buffers\_size 16k;

fastcgi\_temp\_file\_write\_size 16k;

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 huan.wangshibo.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;

}

}

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

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

二、关于FastCGI的几个指令

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;

指定连接到后端FastCGI的超时时间。

fastcgi\_connect\_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\_send\_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\_read\_timeout 300;

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 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。

fastcgi\_buffer\_size 16k;

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 16 16k;

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 32k;

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 32k;

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。

fastcgi\_cache TEST

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\_valid 200 302 1h;

fastcgi\_cache\_valid 301 1d;

fastcgi\_cache\_valid any 1m;

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\_min\_uses 1;

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。

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

注：，\*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：

同时处理的并发请求数，即它将开启最多60个子线程来处理并发连接。

60

最多打开文件数。

65535

每个进程在重置之前能够执行的最多请求数。

65535\*

三、关于内核参数的优化，在/etc/sysctl.conf文件内

timewait的数量，默认是180000。(Deven:因此如果想把timewait降下了就要把tcp\_max\_tw\_buckets值减小)

net.ipv4.tcp\_max\_tw\_buckets = 6000

允许系统打开的端口范围。

net.ipv4.ip\_local\_port\_range = 1024 65000

启用TIME-WAIT状态sockets快速回收功能;用于快速减少在TIME-WAIT状态TCP连接数。1表示启用;0表示关闭。但是要特别留意的是：这个选项一般不推荐启用，因为在NAT(Network Address Translation)网络下，会导致大量的TCP连接建立错误，从而引起网站访问故障。

net.ipv4.tcp\_tw\_recycle = 0

实际上，net.ipv4.tcp\_tw\_recycle功能的开启，一般需要net.ipv4.tcp\_timestamps（一般系统默认是开启这个功能的）这个开关开启后才有效果；

当tcp\_tw\_recycle 开启时（tcp\_timestamps 同时开启，快速回收 socket 的效果达到），对于位于NAT设备后面的 Client来说，是一场灾难！！会导致到NAT设备后面的Client连接Server不稳定（有的 Client 能连接 server，有的 Client 不能连接 server）。

tcp\_tw\_recycle这个功能，其实是为内部网络（网络环境自己可控 ” -不存在NAT 的情况）设计的，对于公网环境下，不宜使用。通常来说，回收TIME\_WAIT状态的socket是因为“无法主动连接远端”，因为无可用的端口，而不应该是要回收内存（没有必要）。也就是说，需求是Client的需求，Server会有“端口不够用”的问题吗？除非是前端机，需要大量的连接后端服务，也就是充当着Client的角色。

正确的解决这个总是办法应该是：

net.ipv4.ip\_local\_port\_range = 9000 6553 #默认值范围较小

net.ipv4.tcp\_max\_tw\_buckets = 10000 #默认值较小，还可适当调小

net.ipv4.tcp\_tw\_reuse = 1

net.ipv4.tcp\_fin\_timeout = 10

1

2

3

4

开启重用功能，允许将TIME-WAIT状态的sockets重新用于新的TCP连接。这个功能启用是安全的，一般不要去改动！

net.ipv4.tcp\_tw\_reuse = 1

开启SYN Cookies，当出现SYN等待队列溢出时，启用cookies来处理。

net.ipv4.tcp\_syncookies = 1

web应用中listen函数的backlog默认会给我们内核参数的net.core.somaxconn限制到128，而nginx定义的NGX\_LISTEN\_BACKLOG默认为511，所以有必要调整这个值。

net.core.somaxconn = 262144

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，允许送到队列的数据包的最大数目。

net.core.netdev\_max\_backlog = 262144

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，孤儿连接将即刻被复位并打印出警告信息。这个限制仅仅是为了防止简单的DoS攻击，不能过分依靠它或者人为地减小这个值，更应该增加这个值(如果增加了内存之后)。

net.ipv4.tcp\_max\_orphans = 262144

记录的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128M内存的系统而言，缺省值是1024，小内存的系统则是128。

net.ipv4.tcp\_max\_syn\_backlog = 262144

时间戳可以避免序列号的卷绕。一个1Gbps的链路肯定会遇到以前用过的序列号。时间戳能够让内核接受这种“异常”的数据包。

net.ipv4.tcp\_timestamps = 1

有不少服务器为了提高性能，开启net.ipv4.tcp\_tw\_recycle选项，在NAT网络环境下，容易导致网站访问出现了一些connect失败的问题。

个人建议：

关闭net.ipv4.tcp\_tw\_recycle选项，而不是net.ipv4.tcp\_timestamps；

因为在net.ipv4.tcp\_timestamps关闭的条件下，开启net.ipv4.tcp\_tw\_recycle是不起作用的；而net.ipv4.tcp\_timestamps可以独立开启并起作用。

=============================================================================

为了打开对端的连接，内核需要发送一个SYN并附带一个回应前面一个SYN的ACK。也就是所谓三次握手中的第二次握手。这个设置决定了内核放弃连接之前发送SYN+ACK包的数量。

net.ipv4.tcp\_synack\_retries = 1

在内核放弃建立连接之前发送SYN包的数量。

net.ipv4.tcp\_syn\_retries = 1

如果套接字由本端要求关闭，这个参数 决定了它保持在FIN-WAIT-2状态的时间。对端可以出错并永远不关闭连接，甚至意外当机。缺省值是60秒。2.2 内核的通常值是180秒，你可以按这个设置，但要记住的是，即使你的机器是一个轻载的WEB服务器，也有因为大量的死套接字而内存溢出的风险，FIN- WAIT-2的危险性比FIN-WAIT-1要小，因为它最多只能吃掉1.5K内存，但是它们的生存期长些。

net.ipv4.tcp\_fin\_timeout = 30

当keepalive起用的时候，TCP发送keepalive消息的频度。缺省是2小时。

net.ipv4.tcp\_keepalive\_time = 30

以下是一个常用的内核参数的标准配置

[root@dev-huanqiu ~]# cat /etc/sysctl.conf

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 //这四行标红内容，一般是发现大量TIME\_WAIT时的解决办法

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 = 1 //在net.ipv4.tcp\_tw\_recycle设置为1的时候，这个选择最好加上

net.ipv4.tcp\_synack\_retries = 1

net.ipv4.tcp\_syn\_retries = 1

net.ipv4.tcp\_tw\_recycle = 1 //开启此功能可以减少TIME-WAIT状态，但是NAT网络模式下打开有可能会导致tcp连接错误，慎重。

net.ipv4.tcp\_tw\_reuse = 1

net.ipv4.tcp\_mem = 94500000 915000000 927000000

net.ipv4.tcp\_fin\_timeout = 30

net.ipv4.tcp\_keepalive\_time = 30

net.ipv4.ip\_local\_port\_range = 1024 65000

net.ipv4.ip\_conntrack\_max = 6553500

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

31

32

33

34

注：

net.ipv4.tcp\_tw\_recycle = 1 这个功能打开后，确实能减少TIME-WAIT状态，但是打开这个参数后，会导致大量的TCP连接建立错误，从而引起网站访问故障。

Nginx安全配置小提示

下面是一个常见安全陷阱和解决方案的列表，它可以辅助来确保你的Nginx部署是安全的。

禁用autoindex模块。这个可能在你使用的Nginx版本中已经更改了，如果没有的话只需在配置文件的location块中增加autoindex off;声明即可。

禁用服务器上的ssi (服务器端引用)。这个可以通过在location块中添加ssi off; 。

关闭服务器标记。如果开启的话（默认情况下）所有的错误页面都会显示服务器的版本和信息。将server\_tokens off;声明添加到Nginx配置文件来解决这个问题。

在配置文件中设置自定义缓存以限制缓冲区溢出攻击的可能性。

client\_body\_buffer\_size 1K;

client\_header\_buffer\_size 1k;

client\_max\_body\_size 1k;

large\_client\_header\_buffers 2 1k;

1

2

3

4

将timeout设低来防止DOS攻击。所有这些声明都可以放到主配置文件中。

client\_body\_timeout 10;

client\_header\_timeout 10;

keepalive\_timeout 65;

send\_timeout 10;

1

2

3

4

限制用户连接数来预防DOS攻击。

limit\_zone slimits $binary\_remote\_addr 5m;

limit\_conn slimits 5;

1

2

试着避免使用HTTP认证。HTTP认证默认使用crypt，它的哈希并不安全。如果你要用的话就用MD5（这也不是个好选择但负载方面比crypt好） 。