nginx指令中的优化（配置文件）：

worker\_processes 8;

nginx进程数，建议按照cpu数目来指定，一般为它的倍数。

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。

worker\_rlimit\_nofile 102400;

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。

use epoll; 使用epoll的I/O模型

worker\_connections 102400;

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

keepalive\_timeout 60; keepalive超时时间

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

open\_file\_cache max=102400 inactive=20s;

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

open\_file\_cache\_valid 30s;

这个是指多长时间检查一次缓存的有效信息

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

内核参数的优化：

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

timewait的数量，默认是180000

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

允许系统打开的端口范围

net.ipv4.tcp\_tw\_recycle = 1

启用timewait快速回收

net.ipv4.tcp\_tw\_reuse = 1

开启重用。允许将TIME-WAIT sockets重新用于新的TCP连接

net.ipv4.tcp\_syncookies = 1

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

net.core.somaxconn = 262144

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.netdev\_max\_backlog = 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.ipv4.tcp\_max\_orphans = 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\_syn\_backlog = 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\_timestamps = 0

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。这里需要将其关掉

net.ipv4.tcp\_synack\_retries = 1

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\_syn\_retries = 1

在内核放弃建立连接之前发送SYN包的数量

net.ipv4.tcp\_fin\_timeout = 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\_keepalive\_time = 30

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

关于FastCGI的几个指令：

fastcgi\_cache\_path /usr/local/nginx/fastcgi\_cache levels=1:2 keys\_zone=TEST:10m inactive=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 16k;

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\_buffers 16 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\_busy\_buffers\_size 32k;

默认值是fastcgi\_buffers的两倍

fastcgi\_temp\_file\_write\_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\_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。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\_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

<value name="max\_children">60</value>

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

<value name="rlimit\_files">102400</value>

最多打开文件数

<value name="max\_requests">204800</value>

每个进程在重置之前能够执行的最多请求数

Nginx性能调优：

1. 自定义404错误页面（使客户体验优化）
2. 查看服务器状态信息（开启状态页面模块）
3. 客户端访问服务器提示“too many open files”如何解决（打开的文件太多）测试ab -c 5000 -n 5000 网址提升nginx并发数量vim /usr/local/nginx/nginx.conf 把默认的1024修改成65535 软硬限制连接设为10000 为了以后不在重新再去设置这一块的参数
4. 解决客户端访问头部信息过长的问题

优化前，使用脚本测试长头部请求是否能获得响应

修改nginx配置文件，增加数据包头部缓存大小

优化后，使用脚本测试长头部请求是否能获得响应

1. 让客户端浏览器缓存数据 定时清理

定义对静态页面的缓存时间 vim /usr/local/nginx/conf/nginx.conf

1. 日志切割

把旧的日志重命名

Kill USR1 PID(nginx的进程pid号)并写一个脚本，自动完成日志切割

1. 对页面压缩处理

对页面进行压缩处理 cat /usr/local/nginx/conf/nginx.conf 开启压缩 设定小文件不压缩 压缩比率1-9 选5

1. 开启文件缓存功能

如果需要处理大量静态文件，可以将文件缓存在内存，下次访问会更快设置服务器最大缓存2000个文件句柄 关闭20秒内无情求的文件句柄 //文件句柄的有效时间是60秒，60秒后过期 //只有访问次数超过5次会被缓存

Nginx安全调优：

1. 删除不需要的模块（without）
2. 修改版本信息（修改源码）
3. 隐藏版本号信息（server\_tokens off）
4. 限制并发 ngx\_http\_limit\_req\_module 为默认模块降低DDOS攻击风险
5. 拒绝非法请求：禁用其他请求方法，仅允许（GET|POST）
6. 防止buffer溢出：防止客户端请求数据溢出，有效降低机器Dos攻击风险

Client\_body\_buffer\_size 1K

Client\_header\_buffer\_size 1K

Client\_max\_body\_size 16K

Large\_client\_header\_buffers 4 4K

1. nginx+lua实现waf，需要编译时加载lua，luajit是lua即时编译器

ln -s /usr/local/lib/libluajit -5.1.so.2 /lib64/lib luajit -5.1.so.2(软连接)

修改nginx.conf增加第一个配置

Location /hello {

Default\_type

Content\_by\_lua

Nginx与安全有关的配置：

1. 隐藏版本号

http {

server\_tokens off;

}

1. 开启https
2. 添加黑白名单

Location /admin/ {

allow 192.168.1.0/24

deny

}

只允许192.168.1.0/24网段的主机访问，拒绝其他所有

1. 添加帐号认证，用auth
2. 限制除GET|POST之外的请求方法
3. 拒绝user-agent
4. 图片防盗链
5. 控制并发连接数
6. 缓冲区溢出攻击
7. Header头设置