大多数的Nginx安装指南告诉你如下基础知识——通过apt-get安装，修改这里或那里的几行配置，好了，你已经有了一个Web服务器了！而且，在大多数情况下，一个常规安装的nginx对你的网站来说已经能很好地工作了。然而，如果你真的想挤压出nginx的性能，你必须更深入一些。在本指南中，我将解释Nginx的那些设置可以微调，以优化处理大量客户端时的性能。需要注意一点，这不是一个全面的微调指南。这是一个简单的预览——那些可以通过微调来提高性能设置的概述。你的情况可能不同。

基本的 (优化过的)配置

我们将修改的唯一文件是**nginx.conf**，其中包含Nginx不同模块的所有设置。你应该能够在服务器的**/etc/nginx**目录中找到nginx.conf。首先，我们将谈论一些全局设置，然后按文件中的模块挨个来，谈一下哪些设置能够让你在大量客户端访问时拥有良好的性能，为什么它们会提高性能。本文的结尾有一个完整的配置文件。

高层的配置

nginx.conf文件中，Nginx中有少数的几个高级配置在模块部分之上。

```
user www-data;
pid /var/run/nginx.pid;

worker_processes auto;

worker_rlimit_nofile 100000;
```

user和**pid**应该按默认设置 - 我们不会更改这些内容，因为更改与否没有什么不同。

worker_processes 定义了nginx对外提供web服务时的worder进程数。最优值取决于许多因素，包括（但不限于）CPU核的数量、存储数据的硬盘数量及负载模式。不能确定的时候，将其设置为可用的CPU内核数将是一个好的开始（设置为“auto”将尝试自动检测它）。

worker_rlimit_nofile 更改worker进程的最大打开文件数限制。如果没设置的话，这个值为操作系统的限制。设置后你的操作系统和Nginx可以处理比“ulimit -a”更多的文件，所以把这个值设高，这样nginx就不会有“too many open files”问题了。

Events模块

events模块中包含nginx中所有处理连接的设置。

```
events {
    worker_connections 2048;
    multi_accept on;
```

```
    use epoll;
}
```

worker_connections设置可由一个worker进程同时打开的最大连接数。如果设置了上面提到的worker_rlimit_nofile，我们可以将这个值设得很高。
记住，最大客户数也由系统的可用socket连接数限制（~ 64K），所以设置不切实际的高没什么好处。

multi_accept 告诉nginx收到一个新连接通知后接受尽可能多的连接。

use 设置用于复用客户端线程的轮询方法。如果你使用Linux 2.6+，你应该使用epoll。如果你使用*BSD，你应该使用kqueue。想知道更多有关事件轮询？看下维基百科吧（注意，想了解一切的话可能需要neckbeard和操作系统的课程基础）

（值得注意的是如果你不知道Nginx该使用哪种轮询方法的话，它会选择一个最适合你操作系统的）

HTTP 模块

HTTP模块控制着nginx http处理的所有核心特性。因为这里只有很少的配置，所以我们只节选配置的一小部分。所有这些设置都应该在http模块中，甚至你不会特别的注意到这段设置。

```
http {

    server_tokens off;

    sendfile on;

    tcp_nopush on;

    tcp_nodelay on;

    ...
}
```

server_tokens 并不会让nginx执行的速度更快，但它可以关闭在错误页面中的nginx版本数字，这样对于安全性是有好处的。
sendfile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)

tcp_nopush 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

tcp_nodelay 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。

```
access_log off;
error_log /var/log/nginx/error.log crit;
```

access_log设置nginx是否将存储访问日志。关闭这个选项可以让读取磁盘IO操作更快(aka,YOLO)
error_log 告诉nginx只能记录严重的错误

```
keepalive_timeout 10;

client_header_timeout 10;
client_body_timeout 10;

reset_timedout_connection on;
send_timeout 10;
```

keepalive_timeout 给客户端分配keep-alive链接超时时间。服务器将在这个超时时间过后关闭链接。我们将它设置低些可以让ngnix持续工作的时间更长。
client_header_timeout 和**client_body_timeout** 设置请求头和请求体(各自)的超时时间。我们也可以把这个设置低些。

reset_timeout_connection告诉nginx关闭不响应的客户端连接。这将会释放那个客户端所占有的内存空间。

send_timeout 指定客户端的响应超时时间。这个设置不会用于整个转发器，而是在两次客户端读取操作之间。如果在这段时间内，客户端没有读取任何数据，nginx就会关闭连接。

```
limit_conn_zone $binary_remote_addr zone=addr:5m;
limit_conn addr 100;
```

limit_conn_zone设置用于保存各种key（比如当前连接数）的共享内存的参数。5m就是5兆字节，这个值应该被设置的足够大以存储（32K*5）32byte状态或者（16K*5）64byte状态。
limit_conn为给定的key设置最大连接数。这里key是addr，我们设置的值是100，也就是说我们允许每一个IP地址最多同时打开有100个连接。

```
include /etc/nginx/mime.types;
default_type text/html;
charset UTF-8;
```

include只是一个在当前文件中包含另一个文件内容的指令。这里我们使用它来加载稍后会用到的一系列的MIME类型。default_type设置文件使用的默认的MIME-type。

charset设置我们的头文件中的默认的字符集

以下两点对于性能的提升在伟大的WebMasters StackExchange中有解释。

```
gzip on;
gzip_disable "msie6";

# gzip_static on;
gzip_proxied any;
gzip_min_length 1000;
gzip_comp_level 4;

gzip_types text/plain text/css application/json application/x-javascript text/xml application/xml
```

gzip是告诉nginx采用gzip压缩的形式发送数据。这将会减少我们发送的数据量。

gzip_disable为指定的客户端禁用gzip功能。我们设置成IE6或者更低版本以使我们的方案能够广泛兼容。

gzip_static告诉nginx在压缩资源之前，先查找是否有预先gzip处理过的资源。这要求你预先压缩你的文件（在这个例子中被注释掉了），从而允许你使用最高压缩比，这样nginx就不用再压缩这些文件了（想要更详尽的gzip_static的信息，请点击这里）。

gzip_proxied允许或者禁止压缩基于请求和响应的响应流。我们设置为any，意味着将会压缩所有的请求。

gzip_min_length设置对数据启用压缩的最少字节数。如果一个请求小于1000字节，我们最好不要压缩它，因为压缩这些小的数据会降低处理此请求的所有进程的速度。

gzip_comp_level设置数据的压缩等级。这个等级可以是1-9之间的任意数值，9是最慢但是压缩比最大的。我们设置为4，这是一个比较折中的设置。

gzip_type设置需要压缩的数据格式。上面例子中已经有一些了，你也可以再添加更多的格式。

```
# cache informations about file descriptors, frequently accessed files
# can boost performance, but you need to test those values
open_file_cache max=100000 inactive=20s;
open_file_cache_valid 30s;
open_file_cache_min_uses 2;
open_file_cache_errors on;
```

```
##
# Virtual Host Configs
# aka our settings for specific servers
##

include /etc/nginx/conf.d/*.conf;
include /etc/nginx/sites-enabled/*;
```

open_file_cache打开缓存的同时也指定了缓存最大数目，以及缓存的时间。我们可以设置一个相对高的最大时间，这样我们可以在它们不活动超过20秒后清除掉。

open_file_cache_valid 在open_file_cache中指定检测正确信息的间隔时间。

open_file_cache_min_uses 定义了open_file_cache中指令参数不活动时间期间里最小的文件数。

open_file_cache_errors指定了当搜索一个文件时是否缓存错误信息，也包括再次给配置中添加文件。我们也包括了服务器模块，这些是在不同文件中定义的。如果你的服务器模块不在这些位置，你就得修改这一行来指定正确的位置。

一个完整的配置

```
user www-data;
pid /var/run/nginx.pid;
worker_processes auto;
worker_rlimit_nofile 100000;

events {
    worker_connections 2048;
    multi_accept on;
    use epoll;
}

```

```
http {
    server_tokens off;
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;

    access_log off;
    error_log /var/log/nginx/error.log crit;

    keepalive_timeout 10;
    client_header_timeout 10;
    client_body_timeout 10;
    reset_timedout_connection on;
    send_timeout 10;

    limit_conn_zone $binary_remote_addr zone=addr:5m;
    limit_conn addr 100;

    include /etc/nginx/mime.types;
    default_type text/html;
    charset UTF-8;

    gzip on;
    gzip_disable "msie6";
    gzip_proxied any;
    gzip_min_length 1000;
    gzip_comp_level 6;
```

```
    gzip_types text/plain text/css application/json application/x-javascript text/xml application
    
    open_file_cache max=100000 inactive=20s;
    open_file_cache_valid 30s;
    open_file_cache_min_uses 2;
    open_file_cache_errors on;

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}
```

编辑完配置后，确认重启nginx使设置生效。

```
sudo service nginx restart
```