1.nginx是什么

nginx是一个开源的，支持高性能，高并发的www服务和代理服务软件。它是一个俄罗斯人lgor sysoev开发的，作者将源代码开源出来供全球使用。

nginx比它大哥apache性能改进许多，nginx占用的系统资源更少，支持更高的并发连接，有更高的访问效率。

nginx不但是一个优秀的web服务软件，还可以作为反向代理，负载均衡，以及缓存服务使用。

安装更为简单，方便，灵活

2.nginx能干什么

1）反向代理

       正向代理：特定情况下，代理用户访问服务器，需要用户手动的设置代理服务器的ip和端口号。

       反向代理：是用来代理服务器，代理用户要访问的目标服务器。代理服务器接受请求，然后将请求转发给内部网络的服务器(服务集群模式) ，并将从服务器上得到的结果返回给客户端，此时代理服务器对外就表现为一个服务器。

        Nginx在反向代理上，提供灵活的功能，可以根据不同的正则采用不同的转发策略，如图设置好后不同的请求就可以走不同的服务器。

2）负载均衡

       负载均衡：多在高并发情况下需要使用。其原理就是将数据流量分摊到多个服务器执行，减轻每台服务器的压力，多台服务器(集群)共同完成工作任务，从而提高了数据的吞吐量。

       Nginx可使用的负载均衡策略有：轮询（默认）、权重、ip\_hash、url\_hash(第三方)、fair(第三方)。

3）动静分离

       常用于前后端分离，Nginx提供的动静分离是指把动态请求和静态请求分离开，合适的服务器处理相应的请求，使整个服务器系统的性能、效率更高。

       Nginx可以根据配置对不同的请求做不同转发，这是动态分离的基础。静态请求对应的静态资源可以直接放在Nginx上做缓冲，更好的做法是放在相应的缓冲服务器上。动态请求由相应的后端服务器处理。

3.nginx的组成

Nginx二进制可执行文件（由各模块原码编译出的一个文件）

Nginx.conf配置文件（控制Nginx的行为）

access.log（记录每一条http请求信息）

error.log（记录错误日志，定位问题）

4.nginx语法介绍

配置文件由指令与指令块构成

指令块以｛｝大括号将多条指令组织在一起

使用#符号添加注释，提高可读性

include语句允许组合多个配置文件以提升可维护性

使用$符号使用变量

部分指令的参数支持正则表达式

5.nginx命令行简介

nginx -s reload 重新加载配置文件

nginx -s stop 立刻停止服务

nginx -s quit 友好的停止服务，也就是等nginx处理完请求后停止

nignx -s reopen 重新开始记录日志

nginx -t 检测配置文件语法是否有问题

nginx -v 打印nginx 版本，编译信息

6.安装,启动nginx

1.下载源码包

cd /opt

wget -c https://nginx.org/download/nginx-1.12.0.tar.gz

2.解压缩源码

tar -zxvf nginx-1.12.0.tar.gz

3.配置，编译安装 开启nginx状态监测功能

mkdir nginx112

cd/nginx-1.12.0

./configure --prefix=/opt/nginx112/ --with-http\_ssl\_module --with-http\_stub\_status\_module

make && make install

4.启动nginx，进入sbin目录,找到nginx启动命令

cd /opt/nginx112/sbin

./nginx #启动

./nginx -s stop #关闭

./nginx -s reload #平滑重启 ，修改了nginx.conf之后，可以不重启服务，加载新的配置

5.分析nginx的工作目录，内容

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

　　conf 存放nginx的配置文件的

　　nginx.conf 这里是控制nginx所有功能的文件

　　html 存放网页html的目录

　　index.html

　　logs 存放log日志文件

　　sbin 存放nginx可执行命令的,nginx脚本命令

7:Nginx主配置文件解析

整个配置文件是以区块的形式组织的。一般，每个区块以一对大括号{}来表示开始与结束。

#定义Nginx运行的用户和用户组

user www;

#nginx进程数，建议设置为等于CPU总核心数。

worker\_processes 8;

#全局错误日志定义类型，[ debug | info | notice | warn | error | crit ]

error\_log /usr/local/nginx/logs/error.log info;

#进程pid文件

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

#指定进程可以打开的最大描述符：数目

#工作模式与连接数上限

#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。

#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。

worker\_rlimit\_nofile 65535;

events

{

#参考事件模型，use [ kqueue | rtsig | epoll | /dev/poll | select | poll ]; epoll模型

#是Linux 2.6以上版本内核中的高性能网络I/O模型，linux建议epoll，如果跑在FreeBSD上面，就用kqueue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。

use epoll;

#单个进程最大连接数（最大连接数=连接数\*进程数）

#根据硬件调整，和前面工作进程配合起来用，尽量大，但是别把cpu跑到100%就行。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\_connections 65535;

#keepalive超时时间。

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。

client\_header\_buffer\_size 4k;

#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 max=65535 inactive=60s;

#这个是指多长时间检查一次缓存的有效信息。

#语法:open\_file\_cache\_valid time 默认值:open\_file\_cache\_valid 60 使用字段:http, server, location 这个指令指定了何时需要检查open\_file\_cache中缓存项目的有效信息.

open\_file\_cache\_valid 80s;

#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。

#语法:open\_file\_cache\_min\_uses number 默认值:open\_file\_cache\_min\_uses 1 使用字段:http, server, location 这个指令指定了在open\_file\_cache指令无效的参数中一定的时间范围内可以使用的最小文件数,如果使用更大的值,文件描述符在cache中总是打开状态.

open\_file\_cache\_min\_uses 1;

#语法:open\_file\_cache\_errors on | off 默认值:open\_file\_cache\_errors off 使用字段:http, server, location 这个指令指定是否在搜索一个文件是记录cache错误.

open\_file\_cache\_errors on;

}

#设定http服务器，利用它的反向代理功能提供负载均衡支持

http

{

#文件扩展名与文件类型映射表

include mime.types;

#默认文件类型

default\_type application/octet-stream;

#默认编码

#charset utf-8;

#服务器名字的hash表大小

#保存服务器名字的hash表是由指令server\_names\_hash\_max\_size 和server\_names\_hash\_bucket\_size所控制的。参数hash bucket size总是等于hash表的大小，并且是一路处理器缓存大小的倍数。在减少了在内存中的存取次数后，使在处理器中加速查找hash表键值成为可能。如果hash bucket size等于一路处理器缓存的大小，那么在查找键的时候，最坏的情况下在内存中查找的次数为2。第一次是确定存储单元的地址，第二次是在存储单元中查找键 值。因此，如果Nginx给出需要增大hash max size 或 hash bucket size的提示，那么首要的是增大前一个参数的大小.

server\_names\_hash\_bucket\_size 128;

#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 32k;

#客户请求头缓冲大小。nginx默认会用client\_header\_buffer\_size这个buffer来读取header值，如果header过大，它会使用large\_client\_header\_buffers来读取。

large\_client\_header\_buffers 4 64k;

#设定通过nginx上传文件的大小

client\_max\_body\_size 8m;

#开启高效文件传输模式，sendfile指令指定nginx是否调用sendfile函数来输出文件，对于普通应用设为 on，如果用来进行下载等应用磁盘IO重负载应用，可设置为off，以平衡磁盘与网络I/O处理速度，降低系统的负载。注意：如果图片显示不正常把这个改成off。

#sendfile指令指定 nginx 是否调用sendfile 函数（zero copy 方式）来输出文件，对于普通应用，必须设为on。如果用来进行下载等应用磁盘IO重负载应用，可设置为off，以平衡磁盘与网络IO处理速度，降低系统uptime。

sendfile on;

#开启目录列表访问，合适下载服务器，默认关闭。

autoindex on;

#此选项允许或禁止使用socke的TCP\_CORK的选项，此选项仅在使用sendfile的时候使用

tcp\_nopush on;

tcp\_nodelay on;

#长连接超时时间，单位是秒

keepalive\_timeout 120;

#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。下面参数看字面意思都能理解。

fastcgi\_connect\_timeout 300;

fastcgi\_send\_timeout 300;

fastcgi\_read\_timeout 300;

fastcgi\_buffer\_size 64k;

fastcgi\_buffers 4 64k;

fastcgi\_busy\_buffers\_size 128k;

fastcgi\_temp\_file\_write\_size 128k;

#gzip模块设置

gzip on; #开启gzip压缩输出

gzip\_min\_length 1k; #最小压缩文件大小

gzip\_buffers 4 16k; #压缩缓冲区

gzip\_http\_version 1.0; #压缩版本（默认1.1，前端如果是squid2.5请使用1.0）

gzip\_comp\_level 2; #压缩等级

gzip\_types text/plain application/x-javascript text/css application/xml; #压缩类型，默认就已经包含textml，所以下面就不用再写了，写上去也不会有问题，但是会有一个warn。

gzip\_vary on;

#开启限制IP连接数的时候需要使用

#limit\_zone crawler $binary\_remote\_addr 10m;

#负载均衡配置

upstream wwwXXX {

#upstream的负载均衡，weight是权重，可以根据机器配置定义权重。weigth参数表示权值，权值越高被分配到的几率越大。

server 127.0.0.1:9090 weight=3;

server 127.0.0.1:8080 weight=2;

server 127.0.0.1:6060 weight=3;

#nginx的upstream目前支持4种方式的分配

#1、轮询（默认）

#每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，能自动剔除。

#2、weight

#指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。

#例如：

#upstream bakend {

# server 127.0.0.1:9090 weight=10;

# server 127.0.0.1:8080 weight=10;

#}

#2、ip\_hash

#每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。

#例如：

#upstream bakend {

# ip\_hash;

# server 127.0.0.1:9090;

# server 127.0.0.1:8080;

#}

#3、fair（第三方）

#按后端服务器的响应时间来分配请求，响应时间短的优先分配。

#upstream backend {

# server server1;

# server server2;

# fair;

#}

#4、url\_hash（第三方）

#按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，后端服务器为缓存时比较有效。

#例：在upstream中加入hash语句，server语句中不能写入weight等其他的参数，hash\_method是使用的hash算法

#upstream backend {

# server squid1:3128;

# server squid2:3128;

# hash $request\_uri;

# hash\_method crc32;

#}

#tips:

#upstream bakend{#定义负载均衡设备的Ip及设备状态}{

# ip\_hash;

# server 127.0.0.1:9090 down;

# server 127.0.0.1:8080 weight=2;

# server 127.0.0.1:6060;

# server 127.0.0.1:7070 backup;

#}

#在需要使用负载均衡的server中增加 proxy\_pass http://bakend/;

#每个设备的状态设置为:

#1.down表示单前的server暂时不参与负载

#2.weight为weight越大，负载的权重就越大。

#3.max\_fails：允许请求失败的次数默认为1.当超过最大次数时，返回proxy\_next\_upstream模块定义的错误

#4.fail\_timeout:max\_fails次失败后，暂停的时间。

#5.backup： 其它所有的非backup机器down或者忙的时候，请求backup机器。所以这台机器压力会最轻。

#nginx支持同时设置多组的负载均衡，用来给不用的server来使用。

#client\_body\_in\_file\_only设置为On 可以讲client post过来的数据记录到文件中用来做debug

#client\_body\_temp\_path设置记录文件的目录 可以设置最多3层目录

#location对URL进行匹配.可以进行重定向或者进行新的代理 负载均衡

}

#虚拟主机的配置

server

{

#监听端口

listen 80;

#域名可以有多个，用空格隔开

server\_name www.w3cschool.cn w3cschool.cn;

index index.html index.htm index.php;

root /data/www/w3cschool;

#对\*\*\*\*\*\*进行负载均衡

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

{

fastcgi\_pass 127.0.0.1:9000;

fastcgi\_index index.php;

include fastcgi.conf;

}

#图片缓存时间设置

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

{

expires 10d;

}

#JS和CSS缓存时间设置

location ~ .\*.(js|css)?$

{

expires 1h;

}

#日志格式设定

#$remote\_addr与$http\_x\_forwarded\_for用以记录客户端的ip地址；

#$remote\_user：用来记录客户端用户名称；

#$time\_local： 用来记录访问时间与时区；

#$request： 用来记录请求的url与http协议；

#$status： 用来记录请求状态；成功是200，

#$body\_bytes\_sent ：记录发送给客户端文件主体内容大小；

#$http\_referer：用来记录从那个页面链接访问过来的；

#$http\_user\_agent：记录客户浏览器的相关信息；

#通常web服务器放在反向代理的后面，这样就不能获取到客户的IP地址了，通过$remote\_add拿到的IP地址是反向代理服务器的iP地址。反向代理服务器在转发请求的http头信息中，可以增加x\_forwarded\_for信息，用以记录原有客户端的IP地址和原来客户端的请求的服务器地址。

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

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

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

#定义本虚拟主机的访问日志

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

access\_log /usr/local/nginx/logs/host.access.404.log log404;

#对 "/" 启用反向代理

location / {

proxy\_pass http://127.0.0.1:88;

proxy\_redirect off;

proxy\_set\_header X-Real-IP $remote\_addr;

#后端的Web服务器可以通过X-Forwarded-For获取用户真实IP

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

#以下是一些反向代理的配置，可选。

proxy\_set\_header Host $host;

#允许客户端请求的最大单文件字节数

client\_max\_body\_size 10m;

#缓冲区代理缓冲用户端请求的最大字节数，

#如果把它设置为比较大的数值，例如256k，那么，无论使用firefox还是IE浏览器，来提交任意小于256k的图片，都很正常。如果注释该指令，使用默认的client\_body\_buffer\_size设置，也就是操作系统页面大小的两倍，8k或者16k，问题就出现了。

#无论使用firefox4.0还是IE8.0，提交一个比较大，200k左右的图片，都返回500 Internal Server Error错误

client\_body\_buffer\_size 128k;

#表示使nginx阻止HTTP应答代码为400或者更高的应答。

proxy\_intercept\_errors on;

#后端服务器连接的超时时间\_发起握手等候响应超时时间

#nginx跟后端服务器连接超时时间(代理连接超时)

proxy\_connect\_timeout 90;

#后端服务器数据回传时间(代理发送超时)

#后端服务器数据回传时间\_就是在规定时间之内后端服务器必须传完所有的数据

proxy\_send\_timeout 90;

#连接成功后，后端服务器响应时间(代理接收超时)

#连接成功后\_等候后端服务器响应时间\_其实已经进入后端的排队之中等候处理（也可以说是后端服务器处理请求的时间）

proxy\_read\_timeout 90;

#设置代理服务器（nginx）保存用户头信息的缓冲区大小

#设置从被代理服务器读取的第一部分应答的缓冲区大小，通常情况下这部分应答中包含一个小的应答头，默认情况下这个值的大小为指令proxy\_buffers中指定的一个缓冲区的大小，不过可以将其设置为更小

proxy\_buffer\_size 4k;

#proxy\_buffers缓冲区，网页平均在32k以下的设置

#设置用于读取应答（来自被代理服务器）的缓冲区数目和大小，默认情况也为分页大小，根据操作系统的不同可能是4k或者8k

proxy\_buffers 4 32k;

#高负荷下缓冲大小（proxy\_buffers\*2）

proxy\_busy\_buffers\_size 64k;

#设置在写入proxy\_temp\_path时数据的大小，预防一个工作进程在传递文件时阻塞太长

#设定缓存文件夹大小，大于这个值，将从upstream服务器传

proxy\_temp\_file\_write\_size 64k;

}

#设定查看Nginx状态的地址

location /NginxStatus {

stub\_status on;

access\_log on;

auth\_basic "NginxStatus";

auth\_basic\_user\_file confpasswd;

#htpasswd文件的内容可以用apache提供的htpasswd工具来产生。

}

#本地动静分离反向代理配置

#所有jsp的页面均交由tomcat或resin处理

location ~ .(jsp|jspx|do)?$ {

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_pass http://127.0.0.1:8080;

}

#所有静态文件由nginx直接读取不经过tomcat或resin

location ~ .\*.(htm|html|gif|jpg|jpeg|png|bmp|swf|ioc|rar|zip|txt|flv|mid|doc|ppt|

pdf|xls|mp3|wma)$

{

expires 15d;

}

location ~ .\*.(js|css)?$

{

expires 1h;

}

}

}

8. location 匹配规则：

location [=|~|~\*|^~|@] /uri/ {

...

}

= : 表示精确匹配后面的url

~ : 表示正则匹配，但是区分大小写

~\* : 正则匹配，不区分大小写

^~ : 表示普通字符匹配，如果该选项匹配，只匹配该选项，不匹配别的选项，一般用来匹配目录

@ : "@" 定义一个命名的 location，使用在内部定向时，例如 error\_page

上述匹配规则的优先匹配顺序：

= 前缀的指令严格匹配这个查询。如果找到，停止搜索；

所有剩下的常规字符串，最长的匹配。如果这个匹配使用 ^~ 前缀，搜索停止；

正则表达式，在配置文件中定义的顺序；

如果第 3 条规则产生匹配的话，结果被使用。否则，使用第 2 条规则的结果。

9.Rewrite规则

rewrite功能就是，使用nginx提供的全局变量或自己设置的变量，结合正则表达式和标志位实现url重写以及重定向。

rewrite只能放在server{},location{},if{}中，并且只能对域名后边的除去传递的参数外的字符串起作用，

例如 http://seanlook.com/a/we/index.php?id=1&u=str 只对/a/we/index.php重写。语法rewrite regex replacement [flag];

flag标志位

last : 相当于Apache的[L]标记，表示完成rewrite

break : 停止执行当前虚拟主机的后续rewrite指令集

redirect : 返回302临时重定向，地址栏会显示跳转后的地址

permanent : 返回301永久重定向，地址栏会显示跳转后的地址

10.Nginx日志格式:

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

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

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

#access\_log logs/access.log main;

语法:

# 设置访问日志

access\_log path [format [buffer=size] [gzip[=level]] [flush=time][if=condition]];

# 关闭访问日志

access\_log off;

1.path 指定日志的存放位置。

2.format 指定日志的格式。默认使用预定义的 combined 。

3.buﬀer 用来指定日志写入时的缓存大小。默认是64k。

4.gzip 日志写入前先进行压缩。压缩率可以指定，从1到9数值越大压缩比越高，同时压缩的速度也越慢。默认 是1。

5.ﬂush 设置缓存的有效时间。如果超过ﬂush指定的时间，缓存中的内容将被清空。

6.if 条件判断。如果指定的条件计算为0或空字符串，那么该请求不会写入日志。