# Nginx学习文档

# 什么是Nginx

Nginx是一个高性能的HTTP和反向代理web服务器

Nginx是一款轻量级的Web服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器，在BSD-like协议下发行，其特点是占有内存小，并发能力强，事实上nginx的并发能力确实在同类型的网页服务器中表现较好，中国大陆使用的nginx网站用户有：百度、京东、新浪、网易、腾讯、淘宝等。

# 为什么使用Nginx

在传统Web项目中，并发量小，用户使用的少，所以在低并发的情况下，用户可以直接访问tomcat服务器，然后tomcat服务器返回消息给用户。

用户访问<--->Tomcat服务器

而在互联网项目下，因单个tomcat默认并发量有限制。如果请求量过大，会产生如下问题：

## 高并发（High Concurrency）

是互联网分布式系统结构设计中必须考虑的因素之一，它通常是指，通过设计保证系统能够同时并行处理很多请求。

高并发相关常用的一些指标有响应时间（Response Time），吞吐量（Throughput），每秒查询率QPS（Query Per Second），并发用户数等。

**响应时间：**系统对请求做出响应的时间。例如系统处理一个HTTP请求需要200ms，这个200ms就是系统的响应时间。

**吞吐量：**单位时间内处理的请求数量。

**QPS：**每秒响应请求数。在互联网领域，在这个指标和吞吐量区分的没有那么明显。

**并发用户数：**同时承载正常使用系统功能的用户数量。

## 高可用（High Availability）

通常来描述一个系统经过专门的设计，从而减少停工时间，而保持其服务的高度可用性。

## 高性能

是指服务响应时间快，（CPU/处理器/内存）特别始在高并发响应时间下不会急剧增加。

## Nginx特点

高并发、高性能

可扩展性好

高可靠性

热部署

BSD许可证

# 简介

## 正向代理和反向代理区别区别？

**反向代理：**

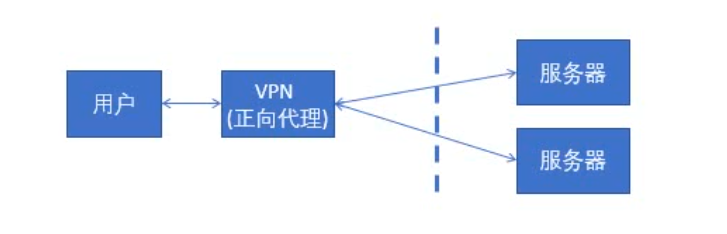
反向代理（Reverse Proxy）方式是指以代理服务器来接收internet上的连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给internet上请求连接的客户端，此时代理服务器对外就表现为一个反向代理服务器。

**正向代理：**

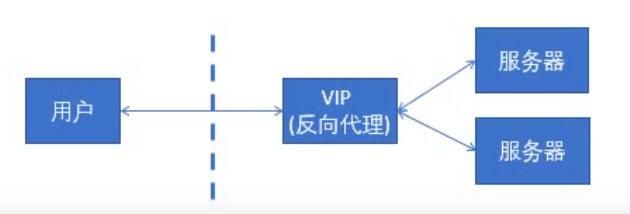
是一个位于客户端和原始服务器（origin server）之间的服务器，为了从原始服务器取得内容，客户端向代理发送一个请求并指定目标（原始服务器），然后代理向原始服务器转发请求并将获得的内容返回给客户端。客户端才能使用正向代理。

**正向代理和反向代理的区别**

**正向代理**，是在客户端的。比如需要访问某个国外网站，我们可能需要购买vpn。并且vpn是在我们的用户浏览器端设置的（并不是在远端的服务器设置的）。浏览器先访问vpn地址，vpn地址转发请求，并最后将请求结果原路返回来。



**反向代理是作用在服务器端的，**是一个虚拟的ip(VIP)。对于用户的一个请求，会转发到多个后端处理器中的一台来处理该具体请求。



# Nginx的下载和安装

nginx下载地址：<http://nginx.org/en/download.html>

## windows

****

### 安装

1. 解压文件夹
2. 双击nginx.exe，一闪而过，nginx启动成功，通过访问：<http://localhost/>

（端口默认80）

### 使用

nginx的使用比较简单，就是几条命令。

常用到的命令如下：

start nginx 启动nginx

nginx -s stop 快速关闭nginx，可能不保存相关信息，并迅速中止web服务

nginx -s quit 平稳的关闭nginx，保存相关信息，有安排的结束web服务

nginx -s reload 因改变了Nginx相关配置，需要重新加载配置而重载

nginx -s reopen 重新打开日志文件

nginx -c filename 为Nginx指定一个配置文件，来代替缺省的

nginx -t 不运行，而仅仅测试配置文件。nginx将检查配置文件的语法的正确性，并尝试打开配置文件中所引用到得到文件

nginx -v 显示nginx的版本

nginx -V 显示nginx的版本，编译器版本和配置参数

Windows下执行命令需要带上exe ./nginx.exe -s stop

提示：

如果不想每次都敲命令，可以在nginx安装目录下新添一个启动批处理文件startup.bat，双击即可运行。内容如下：

@echo off

rem 如果启动前已经启动nginx并记录下pid文件，会kill指定进程

nginx.exe -s stop

rem 测试配置文件语法正确性

nginx.exe -t -c conf/nginx.conf

rem 显示版本信息

nginx.exe -v

rem 按照指定配置去启动nginx

nginx.exe -c conf/nginx.conf



## Linux

linux安装和启动参考博客：<https://blog.csdn.net/xnlay/article/details/79062270>

## 配置文件描述

### nginx.conf配置文件描述：

**基本配置**

全局块配置：配置影响nginx全局的指令。一搬有运行nginx服务器的用户组，nginx进程pid存放路径，日志存放路径，配置文件引入，允许生成worker process数等。

#user nobody; #配置用户或者组，默认为nobody

worker\_processes 1; #允许生成的进程数，默认为1

#error\_log logs/error.log; #制定日志路径，级别，这个设置可以放入全局块，http块，server #块，级别以此为：debug|info|notice|warn|error|crit|ert|emery

#error\_log logs/error.log notice;

#error\_log logs/error.log info;

#pid logs/nginx.pid; #指定nginx进程运行文件存放地址

**events块配置**

evevts块：配置影响nginx服务器或与用户的网络连接。有每个进程的最大连接数，选取哪种事件驱动模型处理连接请求，是否允许同时接受多个网络连接，开启多个网络连接序列化等。

accept\_mutex on; #设置网络连接序列化，防止惊群现象发生，默认为on

multi\_accept on; #设置一个进程是否同时接受多个网络连接，默认为off

#use epoll; #事件驱动模型，select|poll|kqueue|epoll|resig|/dev/poll|exentport

worker\_connections 1024; #最大连接数，默认为1024（早期是512）

**max\_client:**

nginx作为http服务器的时候：

max\_clients = worker\_processes \* worker\_connections

由HTTP客户端发起一个请求，创建一个到服务器指定端口（默认是80端口）的TCP连接。HTTP服务器则在那个端口监听客户端的请求。一旦收到请求，服务器会像客户端返回一个状态，比如”HTTP/1.1 200 OK”，以及返回的内容，如请求的文件、错误消息、或者其他信息。同一时刻nginx在处理客户端发送的http请求应该只是一个connection，由此可知理论上作为http web服务器角色的nginx能够处理的最大连接数就是最大客户端连接数。

nginx作为反向代理服务器的时候：

max\_clients = worker\_processes \* worker\_connections/4

如果作为反向代理，因为浏览器默认会开启2个连接到server，而且nginx还会使用fds(file descriptor)从同一个连接池建立连接到upstream后端。则最大连接数的计算公式需要除4

**http块配置**

可以嵌套多个server，配置代理，缓存，日志定义等绝大多数功能和第三方模块的配置。如文件引入，mime-type定义，日志自定义，是否使用sendfile传输文件，连接超时时间，单连接请求数等。

#配置nginx支持哪些文件扩展名与文件类型映射表。在conf/mime.type查看支持哪些类型

include mime.types;

#默认文件类型（流）类型，支持很多文件、图片、js/css等

default\_type application/octet-stream;

#自定义格式

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

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

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

access\_log log/access.log myFormat; #combined为日志格式的默认值

#优化参数 允许sendfile方式传输文件，开启高效传输模式

sendfile on;

#tcp\_nopush on; #防止网络阻塞

#keepalive\_timeout 0;

keepalive\_timeout 65; #长连接超时时间（单位秒）

#gzip on; #开启gzip压缩

# 应用场景

## 需求1：静态资源

**静态配置文件处理**

由于Nginx性能很高，对于常用的静态资源，可直接交有Nginx进行访问处理

示例：

location / {

root F:/nginx-1.18.0/static/exam;

index index.html index.htm;

}

配置多个静态资源示例：

location / {

root F:/nginx-1.18.0/static/exam;

index index.html index.htm;

}

location /test {

alias F:/nginx-1.18.0/test/exam;

autoindex on;

charset utf-8;

}

在访问/test配置的静态资源时，在访问路径最后加上/test

## 需求2：反向代理

**让nginx进行转发 ，即所谓的反向代理访问localhost时转到tomcat**

修改nginx.conf文件，查看server节点，相当于一个代理服务器，可以配置多个。

listen: 表示当前的代理服务器监听的端口，默认的是监听80端口

server\_name: 表示服务名称

location: 表示匹配的路径，表示所有的请求都被匹配到这里

root: 里面配置了root这时表示当匹配这个请求的路径时，将会在这个文件夹内寻找相应的文件。

index: 当没有指定主页时，默认会选择这个指定的文件，它可以有多个，并按顺序来加载，如果第一个不存在，则找第二个，依此类推下面的 error\_page 配置的错误页面

error\_page: 代表错误的页面

server {

listen 80;

server\_name localhost;

#charset koi8-r;

#access\_log logs/host.access.log main;

location / {

root F:/nginx-1.18.0/static/exam;

index index.html index.htm;

}

#error\_page 404 /404.html;

localhost跳转到tomcat时，修改两个地方：

location / {

#root F:/nginx-1.18.0/static/exam;

#index index.html index.htm;

proxy\_pass http://localhost:8080/;

}

（上面两行注释的可以删掉）

## 需求3：动静分离

我们需要分文件类型来进行过滤，比如jsp直接给tomcat处理，因为nginx并不是servlet容器，没办法处理jsp，而html,js,css这些不需要处理的，直接给nginx进行缓存即可。

动态资源：如jsp由Tomcat或其他WEB服务器处理。

静态资源：如图片、css、js等由nginx服务器完成。

（动静分离充分利用各自的优势完成高性能访问）

让JSP页面直接给tomcat，而html,png等一些图片和JS等直接给nginx进行缓存

location ~ \.jsp$ {

proxy\_pass http://127.0.0.1:8080;

}

location ~ \.(html|js|css|image|png|gif)$ {

root F:/nginx-1.18.0/static/exam;

}

同时，页面上方要进行静态资源的变更：（以绝对路径方式进行访问）

<script type="text/javascript" src=”/js/jquery-3.1.1.js”></script>

<script src=”/js/test.js”></script>

**结束：nginx对location访问优先是以精确优先为原则，故将精确细的请求放在前面。这样就可以完成基本的动静分离配置。**

location ~ \.(html|js|css|png|gif)$ {

root F:/nginx-1.18.0/static/exam;

}

location /{

proxy\_pass http://127.0.0.1:8080;

}

# 集群

## 集群

同一个业务，部署在多个服务器上

将一个应用程序，部署到多台服务器上面，然后在这些服务器的前面通过负载均衡服务器来择优选择哪一台服务器去执行；（保证高可用）

1. 进入nginx的conf路径，修改配置文件nginx.xml

配置服务器组，在http{}节点之间添加upstream配置。（注意不要写localhost，不然访问速度会很慢）

upstream nginxCluster{

server 127.0.0.1:8080; #服务器8080

server 127.0.0.1:8081; #服务器8081

}

1. 在location{}中，利用proxy\_pass配置反向代理地址：

此处”http://”不能少，后面的地址要和第一步upstream定义的名称保持一致

location /{

proxy\_pass http://nginxCluster;

}

注意：

upstream中的server元素必须要注意，不能加http://，但proxy\_pass必须加

proxy\_pass，它表示代理路径，相当于转发，而不像之前说的root必须指定一个文件夹

完整如下：

注：tomcatCluster 为自己起的名字

upstream tomcatCluster{

server 127.0.0.1:8080;

server 127.0.0.1:8081;

}

server {

listen 80;

server\_name localhost;

#charset koi8-r;

#access\_log logs/host.access.log main;

location / {

#root F:/nginx-1.18.0/static/exam;

#index index.html index.htm;

proxy\_pass <http://tomcatCluster/;> #集群配置

}

# Nginx负载均衡策略

## 负载均衡

将服务器接收到的请求按照规则分发的过程，称为负载均衡

### 轮询（默认）

每个请求按照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

### 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默认选项，当weight不能指定时，各服务器weight相同，（weight=1）

upstream tomcatCluster{

server 127.0.0.1:8080 wetight=1;

server 127.0.0.1:8081 wetight=1;

}

数字越大、表明请求到的机会越大

### ip\_hash

每个请求按照访问ip的hash值分配，这样同一客户端连续的web请求都会被分发到同一服务器进行处理，可以解决session的问题。当后台服务器宕机时，会自动跳转到其他服务器。

upstream tomcatCluster{

ip\_hash;

server 127.0.0.1:8080 weight=1;

server 127.0.0.1:8081 weight=2;

}

基于weight的负载均衡和基于ip\_hash的负载均衡可以组合在一起使用，也可以删除weight，不在一起使用

### ur\_hash（第三方）

url\_hash是nginx的第三方模块，nginx本身不支持，需要补丁

nginx按访问的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，后端服务器为缓存服务器、文件服务器、静态服务器时比较有效。缺点是当后端服务器宕机的时候，url\_hash不会自动跳转到其他缓存服务器，而是返回给用户一个503错误。

### 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

在实际开发过程中使用最多的负载均衡策略是轮询策略，如果部署的服务器性能不一样，可以指定轮询几率来使用。

# 集群/分布式环境下session解决方案

**集群是个物理形态，分布式是个工作方式**

分布式：一个业务分拆多个子业务，部署在不同的服务器上

集群：同一个业务，部署在多个服务器上

**为什么要处理session?**

这个问题想必大多数朋友都知道，在搭建完集群或者分布式环境之后，如果不做任何处理的话，网站将频繁的出现用户登陆的现象。比如：集群中有A、B两台服务器，用户第一次访问网络时，nginx将用户请求分发到A服务器，这时A服务器给用户创建了一个Session，当用户第二次访问网站时，假设Nginx将用户请求分发到了B服务器上，而这时B服务器并不存在用户的session，所以就会出现用户未登录的情况，这对用户来说是不可忍受的。

所以我们在搭建分布式/集群环境之后，必须考虑的一个问题就是用户访问产生的session如何处理，即session的共享机制

**解决方案**

**我们将处理Session的方式大致分为三种：**

Session保持（也有人叫黏性Session）

Session复制

Session共享

Session保持（会话保持）就是将用户锁定到某一个服务器上。比如上面说的例子，用户第一次请求时，负载均衡器（Nginx）将用户的请求分发到了A服务器上，如果负载均衡器（Nginx）设置了Session保持的话，那么用户以后每次请求都会分发到A服务器上，相当于把用户和A服务器粘到了一起，这就是Session保持的原理。Session保持方案在所有的负载均衡器都有对应的实现。而且这是在负载均衡这一层就可以解决Session问题。

优点：非常简单，不需要对Session做任何处理。

缺点：1、负载不均衡了：由于使用了Session保持，很显然就无法保持负载的均衡。

1. 缺乏容错性：如果后端某台服务器宕机，那么这台服务器的Session丢失，被分配到这台服务器请求的用户还是需要重新登陆，所有没有彻底解决问题。

实现方式：以Nginx为例，在upstream模块配置ip\_hash属性即可实现粘性Session