# Nginx学习

相关链接：

https://segmentfault.com/a/1190000015006377?utm\_source=index-hottest --未完成

## 概述

Nginx（engine x）是一款轻量级的高性能Http服务器、反向代理服务器及电子邮件（IMAP/POP3）代理服务器。

Nginx有一个主进程和多个工作进程。主进程的作用只有一个，即通过读取配置文件（nginx.conf）来维持工作进程正常工作；工作进程对应的才是实际的请求（request），nginx会自动将客户端的请求高效地分发给工作进程。工作进程的数量在配置文件中定义，或者通过cpu的核心数自动调整。

## 应用场景

-1）http服务器：

Nginx是一个轻量级的高性能http服务器，可以将服务器上的静态文件（如html、img）通过http协议展现给客户端。

-2）虚拟主机：

可以实现在一台服务器虚拟出多个网站。例如个人网站使用的虚拟主机。

例如将www.aaa.com和www.bbb.com两个网站部署在同一台服务器上，两个域名解析到同一个IP地址，但是用户通过两个域名却可以打开两个完全不同的网站，互相不影响，就像访问两个服务器一样，所以叫两个虚拟主机。

虚拟主机的原理是通过HTTP请求头中的Host是否匹配server\_name来实现的

-3）反向代理，负载均衡：

当网站的访问量达到一定程度后，单台服务器不能满足用户的请求时，需要用多台服务器集群，可以使用nginx做反向代理，并且多台服务器可以平均分担负载，不会因为某台服务器负载高宕机而某台服务器闲置的情况。

-4）容错技术：

容错就是当由于种种原因在系统中出现了数据、文件损坏或丢失时，系统能够自动将这些损坏或丢失的文件和数据恢复到发生事故前的状态，是系统能够连续正常运行的技术。

## 安装教程

-1）下载并解压nginx.rar

-2）启动：

双击nginx.exe（方式一）：

直接双击nginx.exe出现屏幕一闪而过，证明已经启动成功了。任务管理器会有两个nginx.exe进程，其中1个是守护进程，1个工作进程。

测试：浏览器输入http://127.0.0.1后，出现welcome to nginx！

命令行启动（方式二）：

打开：（呼出cmd输入）

> cd d:/nginx

> start nginx

关闭：> nginx –s stop

修改端口号：

打开nginx目录下的conf/nginx.conf文件，然后找到listen 80,修改为9090，关闭nginx再启动，就可以通过如下地址访问了:http://127.0.0.1:9090

常用命令：

nginx -s stop :关闭nginx

nginx -s reload :重新加载

nginx -s reopen :重新打开日志文件

## 配置文件（nginx.conf）

描述：

打开配置文件：vim /usr/local/nginx/conf/nginx.conf

server相当于一个服务器，你可以配置多个server模块来虚拟多个服务器。listen表示该服务器监听的端口，默认80端口。server\_name表示匹配主机名，比如www.yjy.com。那么当客户端通过域名www.yjy.com来访问的时候，就是匹配到这个服务器。

location就是匹配的路由，root是返回给客户端资源文件的根目录，它相对nginx路径的，也可以指定绝对路径(比如:/usr/local/nginx/www)。

index就是默认返回的文件名（比如index.html，index.php，index.asp等）。

配置简单的反向代理服务器，只需匹配到路由的location模块中，使用proxy\_pass来转到真实的服务器。如：location / { proxy\_pass://www.baidu.com; }

## 反向代理

### 相关概念

-1）正向代理

指我们通过中间代理去访问另一个服务器。比如我们一般都通过连接一个代理服务器，通过代理服务器连接要访问的国外某网站，这网络访问形式就是正向代理。

比如说要访问youtube，但是不能直接访问，只能先找个翻墙软件，通过翻墙软件才能访问youtube。该翻墙软件就叫正向代理

-2）反向代理

在nginx部署的负载均衡系统中，用户访问某个网址，负载均衡根据一定的规则，将该访问分发到具体某一服务器（多个服务器运行着同一个项目），最后用户得到执行结果，这网络访问形式就是反向代理。

比如说用户要访问youtube，但是youtube悄悄把这个请求交给bilibili来做。该bilibili就是反向代理。

再比如说我们访问nginx，但nginx把请求交给tomcat来做。

### 使用步骤

-1）启动tomcat

-2）修改nginx.conf，主要在30~42行

location / { #表示处理所有请求

proxy\_pass http://127.0.0.1:8111; #表示把请求交给它处理

}

-3）重启nginx并访问

呼出cmd: > nginx –s reload

浏览器访问: http://127.0.0.1，此时它会反向代理到tomcat中...

-4）意义：

既然直接通过127.0.0.1:8111/login.jsp 可以访问，那么干嘛要通过127.0.0.1/login.jsp 去反向代理到tomcat呢？ 为什么不直接使用tomcat呢？

因为nginx在处理静态文件的吞吐量上面比tomcat好很多，通常他们俩配合，不会把所有的请求都如本例所示的交给tomcat, 而是把静态请求交给nginx，动态请求，如jsp, servlet,ssm， struts等请求交给tomcat. 从而达到动静分离的效果。

配置反向代理：

-1）我们能看到其工作进程配置为1，每个工作进程可以创建的连接数为1024

work\_processes 1;

events {

work\_connections 1024;

}

-2）server配置项

--监听端口为80，服务器名为localhost，在location选项里可以配置我们的代理

server {

listen 80;

server\_name localhost;

... //省略

location / {

root html;

index index.html index.htm;

}

}

-3）配置反向代理

--将我们的代理配置加入到location里

location / {

root html;

index index.html index.htm;

proxy\_pass http://localhost:8088;

}

-4）启动tomcat服务器 和 nginx服务器

-5）测试

在浏览器输入 http://localhost ,发现访问的就是tomcat的主页

### 动静分离

描述：

动静分离，即css、js、images等都交给nginx来处理，nginx处理不了的，如jsp就交给tomcat来处理。

好处：

nginx处理静态内容的吞吐量很高，比tomcat高多了，这样无形中提升了性能。

修改nginx.conf：

--在location下面添加一个新的location：

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

root C:/Users/tomcat/webapps/ROOT; #注意是正斜杠"/"

}

--这表示所有的css、js、png访问都由nginx来做。

### 多个tomcat使用

描述：

nginx是一个web服务器，很快速，但是不能作为Servlet容器独立运行，所以通常的工作方式是nginx配合tomcat来协同工作。

提供多个tomcat原因：因为在后续的负载均衡，session共享教程里需要多个tomcat配合演示。

启动tomcat：

解压tomcat.rar，并运行其中bin/startup.bat文件。在conf/server.xml文件中配置端口

在浏览器输入：http://127.0.0.1:8080

## 负载均衡

### 相关概念

定义：

就是一个业务请求可以交给多个服务器来处理。但要根据每个服务器的负载情况，很忙的服务器不去处理该业务请求，仅交给比较闲的服务器去处理。

当访问量很大时，一个tomcat吃不消，这时候准备多个tomcat，由nginx按照权重来对请求进行分配，从而缓解单独一个tomcat受到的压力。

先到ngix，然后根据，tomcat的负载情况，去访问某个tomcat。

问题描述：

互联网早期，业务流量比较小并且业务逻辑比较简单，单台服务器便可以满足基本的需求。但随着发展，业务流量越来越大并且业务逻辑也越来越复杂，因此需要多台服务器进行性能的水平扩展以及避免单点故障。但如何将不同用户的流量分发到不同的服务器上面呢？

早期的方法：

使用DNS做负载，通过给客户端解析不同的IP地址，让客户端的流量直接到达各个服务器。缺点是延时性问题。

负载均衡原理：

客户端的流量首先会到达负载均衡服务器（nginx），由负载均衡服务器通过一定的调度算法将流量分发到不同的应用服务器上面，同时负载均衡服务器也会对应用服务器做周期性的健康检查，当发现故障节点时会将它从服务器集群中剔除，以此来保证应用的高可用。

负载均衡分类：四层负载均衡、七层负载均衡

1）四层负载均衡工作在OSI模型的传输层，主要工作是转发，它在接收到客户端的流量以后通过修改数据包的地址信息将流量转发到应用服务器。

2）七层负载均衡工作在OSI模型的应用层。

负载均衡策略：（3种）

1）轮询：每种请求逐个分发到后端服务器

2）加权轮询：按照分配的权重将请求分发到后端服务器

3）ip hash：轮询的基础上，保持一个客户端多次请求分发到一台后端服务器上

OSI模型(七层)：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层

1）物理层：比特流传输。将数据转换为可通过物理介质传送的电子信号，相当于邮局的搬运工人

2）数据链路层：提供介质访问、链路管理等。决定访问网络介质的方式。在此层将数据分帧，并处理流控制。本层指定拓扑结构并提供硬件寻址，相当于邮局的装拆箱工人

3）网络层：寻址和路由选择。使用权数据路由经过大型网络，相当于邮局的排序工人

4）传输层：建立主机端到端连接。提供终端到终端的可靠连接，相当于公司的跑邮局的送信职员

5）会话层：建立、维护和管理会话。允许用户使用简单的名称建立连接，相当于公司中收寄信、写信封与拆信封的秘书

6）表示层：处理数据格式、数据加密等。协商数据交换格式，相当于公司中简报老板、替老板写信的助理

7）应用层：提供应用程序间通信。用户的应用程序和网络之间的接口老板

### 实现

--1）修改nginx.conf配置

http {

...

upstream tomcats {

server 192.168.0.100:8080 weight=2; # 2/6次

server 192.168.0.101:8080 weight=3; # 3/6次

server 192.168.0.102:8080 weight=1; # 1/6次

}

server {

listen 80;

location / {

proxy\_pass http://tomcats;

}

}

...

}

--2）在上游配置代理服务器组，并定位集proxy\_pass上游组名。当请求到来时，Nginx会轮询服务器在指定的上游过程。

步骤1：首先增加一个upstream，用来指向这两个tomcat

-- -weight表示权重，值越大，被分配到的几率越大。

upstream tomcat\_8111\_8222 {

server 127.0.0.1:8111 weight=1;

server 127.0.0.1:8222 weight=2;

}

步骤2：然后修改location，反向代理到上述配置

location / {

proxy\_pass http://tomcat\_8111\_8222;

}

## Session共享

问题描述：

通过负载均衡，我们可以把请求分发到不同的tomcat来缓解服务器的压力。但存在一个问题：当同一个用户第一次访问tomcat\_8111并登录成功，而第二次却被分配到tomcat\_8222，这里并没有记录他的登录状态，那么就会呈现未登录状态。

解决方法：

用Redis来存取session.当tomcat1需要保存session值的时候，就可以把它放在Redis上，需要取的时候，也从Redis上取。（借助Jedis对tomcat做session共享）