## 一、为什么使用nginx？

Nginx是一个高性能、可靠性高、扩展性好的Web服务器和反向代理服务器。以下是使用Nginx的一些好处：

1. **高性能**：Nginx采用异步I/O和事件驱动模型，以及非阻塞I/O技术来处理高并发和高性能的网络请求。这使得Nginx在处理大量请求时能够保持高性能和低延迟。
2. **可靠性高**：Nginx具有很高的稳定性和可靠性，因为它可以处理大量的并发连接和请求，而不会出现资源耗尽或崩溃的情况。
3. **扩展性好**：Nginx可以通过插件和模块来扩展其功能，例如添加缓存、负载均衡、安全认证等模块，以满足不同的需求。
4. **简单易用**：Nginx的配置非常简单易用，可以通过修改配置文件来快速地配置和部署服务器。
5. **负载均衡**：Nginx可以作为反向代理服务器进行负载均衡，将请求分发到多个后端服务器，从而提高系统的可伸缩性和稳定性。
6. **静态资源缓存**：Nginx可以对静态资源进行缓存，从而提高网站的访问速度和用户体验。
7. **安全性高**：Nginx具有很强的安全性，可以防止常见的攻击，如DDoS攻击、SQL注入、跨站脚本等。

视频教程：https://www.bilibili.com/video/BV1yS4y1N76R

### 安装

### Windows

1. 下载 Nginx 的 Windows 版本压缩包，可以从 Nginx 的官方网站 <https://nginx.org/en/download.html> 下载到最新版本的 Windows 版本。
2. 解压缩 Nginx 压缩包到您想要安装的目录下，例如 C:\nginx。
3. 在解压缩后的目录下找到 nginx.exe 可执行文件，并且双击运行该文件。
4. 打开浏览器，输入 http://localhost 或者 http://127.0.0.1，如果您看到了 Nginx 的欢迎页面，说明 Nginx 已经成功安装。

### Linux系统（centos）

1. 使用以下命令更新系统的软件包列表：

sudo yum update

2.使用以下命令安装 Nginx：

sudo yum install nginx

3.安装完成后，使用以下命令启动 Nginx 服务：

sudo systemctl start nginx

4.使用以下命令将 Nginx 服务设置为在系统启动时自动启动：

sudo systemctl enable nginx

5.打开浏览器，输入 http://localhost 或者 http://127.0.0.1，如果您看到了 Nginx 的欢迎页面，说明 Nginx 已经成功安装。

### mac系统

1. 打开终端应用程序，使用以下命令安装 Homebrew：

/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

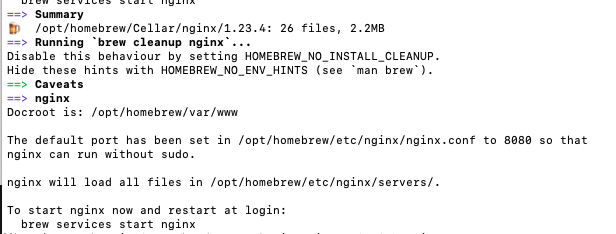
2.安装完成后，使用以下命令检查 Homebrew 是否安装成功：

brew doctor

如果没有显示任何错误信息，说明 Homebrew 已经成功安装。

3.使用 Homebrew 来安装nginx

brew install nginx



启动nginx

nginx //启动

启动成功后访问路径：<http://localhost:8080/>



关闭nginx

nginx -s stop

修改配置文件nginx.conf后，重新加载命令

无需重启服务器，自动编译

nginx -s reload

nginx.conf的内容

#user nobody;  
worker\_processes 1; #要启动的工作进程数，默认为1.通常设为cpu核心数的2倍  
  
#error\_log logs/error.log;  
#error\_log logs/error.log notice;  
#error\_log logs/error.log info;  
  
#pid logs/nginx.pid;  
  
  
events {  
 #指定每个工作进程的最大客户端连接数，每个工作进程最多可以处理1024个客户连接  
 worker\_connections 1024;   
}  
  
  
http {  
 include mime.types;  
 default\_type application/octet-stream;  
  
 #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;  
  
 sendfile on;  
 #tcp\_nopush on;  
  
 #keepalive\_timeout 0;  
 #HTTP keep-alive 连接的超时时间，表示客户端在 65 秒内没有发起新的请求，Nginx 将关闭该连接  
 keepalive\_timeout 65;   
  
 #gzip on;  
  
 server {  
 listen 8088; #服务器端口  
 server\_name localhost; #服务器名称  
  
 #charset koi8-r;  
  
 #access\_log logs/host.access.log main;  
 #匹配规则  
 location / {  
 root html;  
 index index.html index.htm;  
 }  
 #对 URI 为 localhost:8080/test的请求映射  
 # location /test {  
 # root html;  
 # index test.html;  
 # }  
   
 #error\_page 404 /404.html;  
  
 # redirect server error pages to the static page /50x.html  
 #  
 error\_page 500 502 503 504 /50x.html;  
 location = /50x.html {  
 root html;  
 }  
  
 # proxy the PHP scripts to Apache listening on 127.0.0.1:80  
 #  
 #location ~ \.php$ {  
 # proxy\_pass http://127.0.0.1;  
 #}  
  
 # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000  
 #  
 #location ~ \.php$ {  
 # root html;  
 # fastcgi\_pass 127.0.0.1:9000;  
 # fastcgi\_index index.php;  
 # fastcgi\_param SCRIPT\_FILENAME /scripts$fastcgi\_script\_name;  
 # include fastcgi\_params;  
 #}  
  
 # deny access to .htaccess files, if Apache's document root  
 # concurs with nginx's one  
 #  
 #location ~ /\.ht {  
 # deny all;  
 #}  
 }  
  
  
 # another virtual host using mix of IP-, name-, and port-based configuration  
 #  
 #server {  
 # listen 8000;  
 # listen somename:8080;  
 # server\_name somename alias another.alias;  
  
 # location / {  
 # root html;  
 # index index.html index.htm;  
 # }  
 #}  
  
  
 # HTTPS server  
 #  
 #server {  
 # listen 443 ssl;  
 # server\_name localhost;  
  
 # ssl\_certificate cert.pem;  
 # ssl\_certificate\_key cert.key;  
  
 # ssl\_session\_cache shared:SSL:1m;  
 # ssl\_session\_timeout 5m;  
  
 # ssl\_ciphers HIGH:!aNULL:!MD5;  
 # ssl\_prefer\_server\_ciphers on;  
  
 # location / {  
 # root html;  
 # index index.html index.htm;  
 # }  
 #}  
 include servers/\*;  
}

## 二、Tomcat下载与安装

Tomcat 是一个基于 Java 技术的 Web 应用服务器，它需要依赖于 Java 运行环境才能正常运行。因此，在安装 Tomcat 之前，需要先安装 JDK<https://www.oracle.com/java/technologies/javase-downloads.html>

官网下载<https://tomcat.apache.org/download-90.cgi>

### 安装

### windows

1. 解压 Tomcat：将下载的 Tomcat 压缩包解压到任意文件夹中，例如 C:\Program Files\Apache Software Foundation\Tomcat。
2. 配置环境变量：在系统环境变量中添加 CATALINA\_HOME 变量，值为 Tomcat 的安装路径，例如 C:\Program Files\Apache Software Foundation\Tomcat。
3. 启动 Tomcat：进入 Tomcat 的 bin 目录，双击 startup.bat 启动 Tomcat。如果一切正常，应该能看到命令行窗口中输出 Tomcat started. 的信息。
4. 访问 Tomcat：在浏览器中访问 http://localhost:8080，如果出现 Tomcat 的欢迎页面，则说明 Tomcat 已经成功启动。
5. 停止 Tomcat：进入 Tomcat 的 bin 目录，双击 shutdown.bat 停止 Tomcat。

### LInux/Mac

1. 解压 Tomcat：在 Linux 终端中解压下载的 Tomcat 压缩包，例如：

* tar -xvf apache-tomcat-9.0.50.tar.gz  
    
  解压完成后，将解压出来的文件夹移动到合适的位置，例如 `/opt` 目录：  
  sudo mv apache-tomcat-9.0.50 /opt/tomcat  
  授权/opt/tomcat/bin目录下的所有操作  
  sudo chmod 755 \*.sh

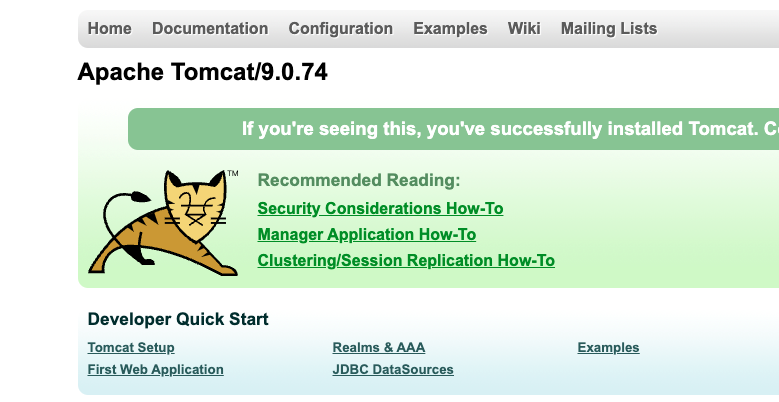
1. 配置环境变量：在 /etc/profile 文件中添加以下两行：

* export CATALINA\_HOME=/opt/tomcat  
  export PATH=$CATALINA\_HOME/bin:$PATH  
    
  然后执行以下命令使配置生效：  
  source /etc/profile

1. 启动 Tomcat：在终端中进入 Tomcat 的 bin 目录，执行以下命令启动 Tomcat：

* sudo sh ./startup.sh  
  如果一切正常，应该能看到终端中输出 `Tomcat started.` 的信息。

1. 访问 Tomcat：在浏览器中访问 http://localhost:8080，如果出现 Tomcat 的欢迎页面，则说明 Tomcat 已经成功启动



1. 停止 Tomcat：在终端中进入 Tomcat 的 bin 目录，执行以下命令停止 Tomcat：

sudo sh ./shutdown.sh

## 三、Nginx 反向代理

**正向代理与反向代理的区别**：**正向代理**相对于目标服务器而言隐藏了客户端的真实IP地址，因为对于目标服务器而言所有请求都是从正向代理服务器发出的，正向代理主要是为了突破网络访问限制，比如科学上网，还有就是隐藏客户端IP地址。

**反向代理**对于客户端而言隐藏了目标服务器IP地址，只需要知道代理服务器地址就能访问到目标服务器的资源。其主功能是可以做负载均衡和安全防护。不过，不管正向代理还是反向代理，都能加快客户端的访问速度，因为nginx服务器是一个高性能的http web服务器，其能够对代理中的数据作缓冲。

### 配置实例1

**实现效果**

打开浏览器，输入<http://localhost:9001/>，可以跳转到tomcat欢迎页面

**访问过程**

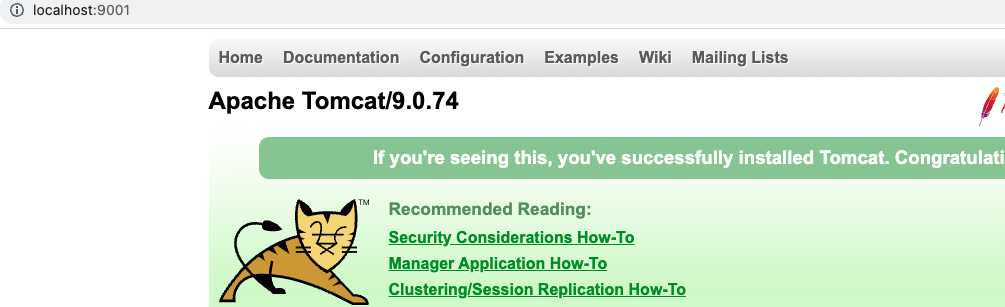


**配置**

nginx.conf文件中新增server块

server {  
 listen 9001;  
 server\_name localhost;  
  
 #charset koi8-r;  
  
 #access\_log logs/host.access.log main;  
  
 location / {  
 proxy\_pass http://127.0.0.1:8080/; #请求转发地址  
 }  
  
 #error\_page 404 /404.html;  
  
 # redirect server error pages to the static page /50x.html  
 #  
 error\_page 500 502 503 504 /50x.html;  
 location = /50x.html {  
 root html;  
 }  
}

**最终实现**



### 配置实例2

**实现效果**

访问 <http://127.0.0.1:9001/hit/> 直接跳转到 127.0.0.1:8080

访问 <http://127.0.0.1:9001/sdu/> 直接跳转到 127.0.0.1:8081

**准备工作**

1.两个tomcat和两个测试页面

新建一个文件夹存放重新解压后的tomcat文件

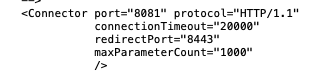
mkdir tomcat1  
tar -zxvf apache-tomcat-9.0.74.tar.gz -C ./tomcat1

修改tomcat1文件夹下的/conf/server.xml 文件

server port由8005改为8006

fig:

http port由8080改为8081



准备好测试页面

a.html，存放在tomcat目录下的/webapps/hit下，内容为：

<h1>hit-nginx-8080!!!</h1>



b.html，存放在tomcat1目录下的/webapps/sdu下，内容为：

<h1>sdu-nginx-8081!!!</h1>

启动tomcat1

进入tomcat1的/bin目录下

sudo sh ./startup.sh

输入网址测试

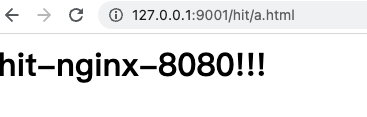


修改nginx配置文件：

server {  
 listen 9001;  
 server\_name localhost;  
  
 #charset koi8-r;  
  
 #access\_log logs/host.access.log main;  
  
  
 location ~ /hit/ {  
 proxy\_pass http://127.0.0.1:8080;  
}  
 location ~ /sdu/ {  
 proxy\_pass http://127.0.0.1:8081;  
}

**测试**

输入<http://127.0.0.1:9001/hit/a.html>



输入<http://127.0.0.1:9001/sdu/b.html>



## 四、nginx负载均衡 配置实例

**实现效果**

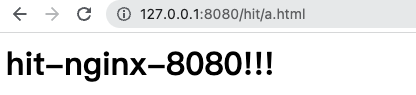
浏览器地址栏输入地址 <http://127.0.0.1/hit/a.html>，负载均衡效果，平均 8081 和 8082 端口对应的a.html页面。

**准备工作**

进入tomcat1目录中/webapps下创建hit文件夹，在hit文件夹下新建a.html

<h1>hit-nginx-8080!!! from Tomcat1</h1>

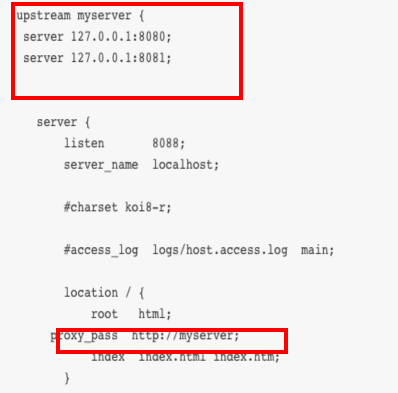
**测试**





进入nginx的配置文件，进行修改

vim /opt/homebrew/etc/nginx/nginx.conf



**最终测试**

浏览器输入<http://127.0.0.1:8088/hit/a.html>, 不断刷新，可以发现请求会被分配给不同的tomcat 服务器





### nginx 分配服务器策略

#### （1）轮询（default）

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

weight 代表权重, 默认为 1,权重越高被分配的客户端越多

upstream myserver {  
 server 127.0.0.1:8080 weight=10; # 在这儿  
 server 127.0.0.1:8081 weight=10;  
}   
  
 server {  
 listen 8088;  
 server\_name localhost;  
  
 #charset koi8-r;  
  
 #access\_log logs/host.access.log main;  
  
 location / {  
 root html;  
 proxy\_pass http://myserver;  
 index index.html index.htm;  
 }

#### （3）ip\_hash

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。

upstream myserver {  
 ip\_hash; # 在这儿  
 server 127.0.0.1:8080;  
 server 127.0.0.1:8081;  
}   
  
 server {  
 listen 8088;  
 server\_name localhost;  
  
 #charset koi8-r;  
  
 #access\_log logs/host.access.log main;  
  
 location / {  
 root html;  
 proxy\_pass http://myserver;  
 index index.html index.htm;  
 }

#### （4）fair（第三方）

upstream myserver {  
 server 127.0.0.1:8080;  
 server 127.0.0.1:8081;  
 fair; # 在这儿  
}   
  
 server {  
 listen 8088;  
 server\_name localhost;  
  
 #charset koi8-r;  
  
 #access\_log logs/host.access.log main;  
  
 location / {  
 root html;  
 proxy\_pass http://myserver;  
 index index.html index.htm;  
 }

### Nginx原理

nginx启动后，由两个进程组成，master进程和worker进程。

fig:

在Nginx中，当一个请求发送到主进程（master process）时，主进程会将请求分配给一个或多个工作进程（worker process）进行处理。

### 具体步骤：

1. **接收请求**：当主进程接收到一个请求时，它会将请求放入请求队列中。
2. **分配工作进程**：主进程会选择一个或多个空闲的工作进程来处理请求。Nginx默认使用轮询算法来分配请求，即将请求均匀地分配给所有的工作进程。
3. **处理请求**：当工作进程接收到请求时，会与客户端建立TCP连接，它会使用Nginx的事件驱动模型来处理请求。工作进程会读取请求并将其解析，然后将请求转发给后端服务器进行处理。
4. **处理响应**：当后端服务器返回响应时，工作进程会将响应发送回客户端。如果响应过大，工作进程也可以将响应分成多个块进行发送。
5. **关闭连接**：当请求处理完毕后，工作进程会关闭连接（ keepalive\_timeout 65; nginx.conf中配置，默认65秒关闭）并等待下一个请求。

### master-worker进程模型主要好处：

1. **提高稳定性**：Nginx的master-worker进程模型可以使得多个worker进程互相独立，当一个worker进程崩溃时，其他worker进程可以继续处理请求，从而提高了系统的稳定性和可靠性。
2. **提高性能**：使用多个worker进程可以充分利用多核CPU的性能，同时避免了I/O阻塞对性能的影响。Nginx的事件驱动模型可以在处理请求时避免了阻塞，从而最大限度地提高了系统的并发处理能力和吞吐量。
3. 简化开发：使用master-worker进程模型可以有效地隔离不同模块之间的代码，从而简化了系统的开发和维护。
4. **灵活性**：Nginx的master-worker进程模型可以通过增加或减少worker进程来适应不同的负载需求，从而提高了系统的灵活性和可扩展性。

总之，Nginx的master-worker进程模型可以在提高系统性能和稳定性的同时，简化开发和提高系统的灵活性，因此在Web服务器领域得到广泛应用。