

# Nginx

## 内容概览

### 1、nginx 简介

- (1) 介绍 nginx 的应用场景和具体可以做什么事情
- (2) 介绍什么是反向代理
- (3) 介绍什么是负载均衡
- (4) 介绍什么是动静分离

### 2、nginx 安装

- (1) 介绍 nginx 在 linux 系统中如何进行安装

### 3、nginx 常用的命令和配置文件

- (1) 介绍 nginx 启动、关闭、重新加载命令
- (2) 介绍 nginx 的配置文件

### 4、nginx 配置实例-反向代理

### 5、nginx 配置实例-负载均衡

### 6、nginx 配置实例-动静分离

### 7、nginx 原理与优化参数配置

### 8、搭建 nginx 高可用集群

- (1) 搭建 nginx 高可用集群（主从模式）
- (2) 搭建 nginx 高可用集群（双主模式）

# 第 1 章 Nginx 简介

## 1.1 Nginx 概述

Nginx (“engine x”) 是一个高性能的 HTTP 和反向代理服务器,特点是占有内存少, 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 nginx 网站用户有: 百度、京东、新浪、网易、腾讯、淘宝等

## 1.2 Nginx 作为 web 服务器

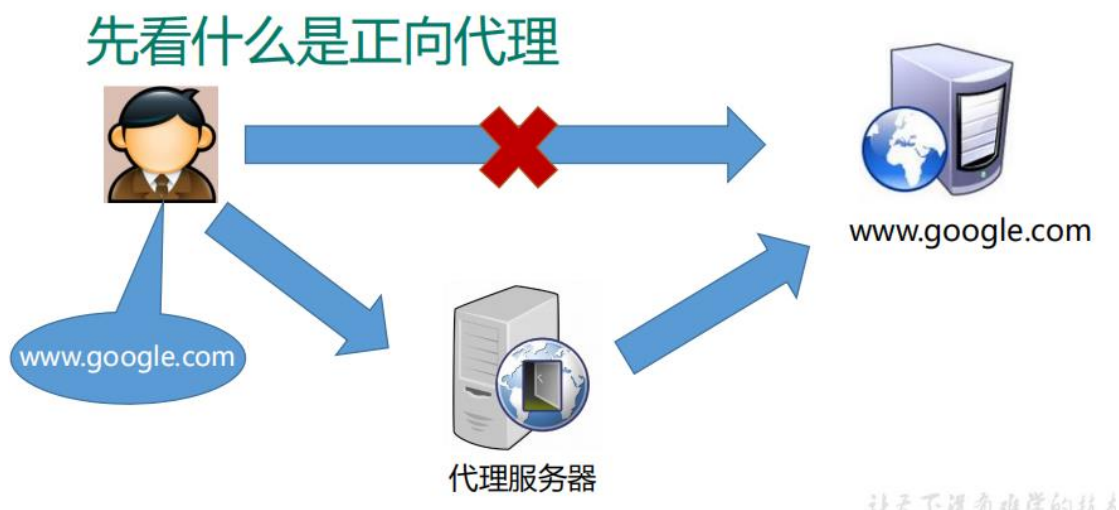
Nginx 可以作为静态页面的 web 服务器,同时还支持 CGI 协议的动态语言, 比如 perl、php 等。但是不支持 java。Java 程序只能通过与 tomcat 配合完成。Nginx 专为性能优化而开发,性能是其最重要的考量,实现上非常注重效率, 能经受高负载的考验,有报告表明能支持高达 50,000 个并发连接数。

<https://lnmp.org/nginx.html>

## 1.3 正向代理

Nginx 不仅可以做反向代理, 实现负载均衡。还能用作正向代理来进行上网等功能。

正向代理: 如果把局域网外的 Internet 想象成一个巨大的资源库, 则局域网中的客户端要访问 Internet, 则需要通过代理服务器来访问, 这种代理服务就称为正向代理。



## 1.4 反向代理

反向代理，其实客户端对代理是无感知的，因为客户端不需要任何配置就可以访问，我们只需要将请求发送到反向代理服务器，由反向代理服务器去选择目标服务器获取数据后，在返回给客户端，此时反向代理服务器和目标服务器对外就是一个服务器，暴露的是代理服务器地址，隐藏了真实服务器 IP 地址。

### 再说什么是反向代理



## 1.5 负载均衡

客户端发送多个请求到服务器，服务器处理请求，有一些可能要与数据库进行交互，服务器处理完毕后，再将结果返回给客户端。

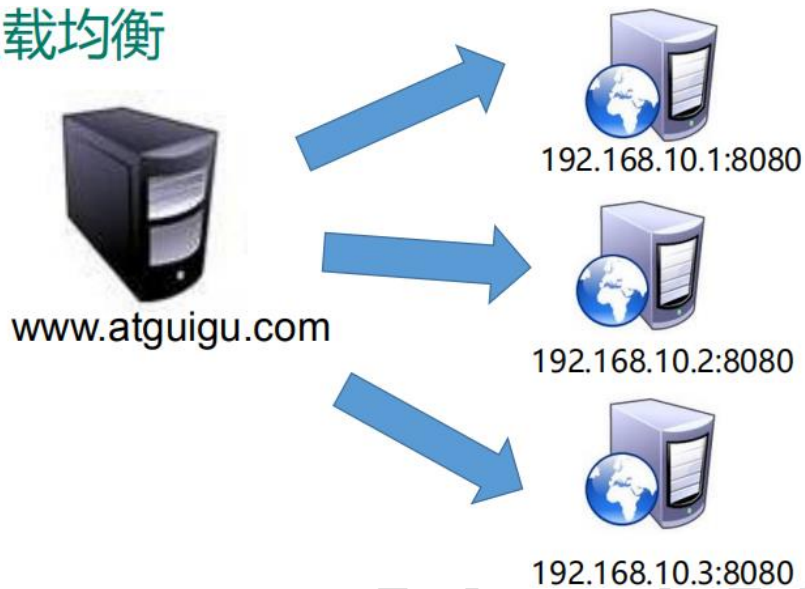
这种架构模式对于早期的系统相对单一，并发请求相对较少的情况下是比较适合的，成本也低。但是随着信息数量的不断增长，访问量和数据量的飞速增长，以及系统业务的复杂度增加，这种架构会造成服务器相应客户端的请求日益缓慢，并发量特别大的时候，还容易造成服务器直接崩溃。很明显这是由于服务器性能的瓶颈造成的问题，那么如何解决这种情况呢？

我们首先想到的可能是升级服务器的配置，比如提高 CPU 执行频率，加大内存等提高机器的物理性能来解决此问题，但是我们知道摩尔定律的日益失效，硬件的性能提升已经不能满足日益提升的需求了。最明显的一个例子，天猫双十一当天，某个热销商品的瞬时访问量是极其庞大的，那么类似上面的系统架构，将机器都增加到现有的顶级物理配置，都是不能够满足需求的。那么怎么办呢？

上面的分析我们去掉了增加服务器物理配置来解决问题的办法，也就是说纵向解决问题的办法行不通了，那么横向增加服务器的数量呢？这时候集群的概念产生了，单个服务器解决不了，我们增加服务器的数量，然后将请求分发到各个服务器上，将原先请求集中到单个

服务器上的情况改为将请求分发到多个服务器上，将负载分发到不同的服务器，也就是我们所说的负载均衡

## 负载均衡



## 1.6 动静分离

为了加快网站的解析速度，可以把动态页面和静态页面由不同的服务器来解析，加快解析速度。降低原来单个服务器的压力。

## 动静分离



## 第 2 章 Nginx 安装

### 2.1 进入 nginx 官网，下载

<http://nginx.org/>



## 需要的素材

pcr-8.37.tar.gz

openssl-1.0.1t.tar.gz

zlib-1.2.8.tar.gz

nginx-1.11.1.tar.gz

### 2.2 安装 nginx

第一步，安装 pcr

wget <http://downloads.sourceforge.net/project/pcr/pcr/8.37/pcr-8.37.tar.gz>

5

更多 Java - 大数据 - 前端 - python 人工智能资料下载，可访问百度：尚硅谷官网

解压文件，

./configure 完成后，回到 pcre 目录下执行 make，

再执行 make install

**第二步，安装 openssl**

**第三步，安装 zlib**

yum -y install make zlib zlib-devel gcc-c++ libtool openssl openssl-devel

```
// 一键安装上面四个依赖  
yum -y install gcc zlib zlib-devel pcre-devel openssl openssl-devel
```

**第四步，安装 nginx**

1、解压缩 nginx-xx.tar.gz 包。

2、进入解压缩目录，执行./configure。

3、make && make install

**查看开放的端口号**

firewall-cmd --list-all

**设置开放的端口号**

firewall-cmd --add-service=http --permanent

sudo firewall-cmd --add-port=80/tcp --permanent

**重启防火墙**

firewall-cmd --reload

## 第 3 章 nginx 常用的命令和配置文件

### 3.1 nginx 常用的命令：

(1) 启动命令

在/usr/local/nginx/sbin 目录下执行 ./nginx

## (2) 关闭命令

在/usr/local/nginx/sbin 目录下执行 ./nginx -s stop

## (3) 重新加载命令

在/usr/local/nginx/sbin 目录下执行 ./nginx -s reload

## 3.2 nginx.conf 配置文件

nginx 安装目录下，其默认的配置文​​件都放在这个目录的 conf 目录下，而主配置文件 nginx.conf 也在其中，后续对 nginx 的使用基本上都是对此配置文件进行相应的修改

```
[root@slave1 ~]# cd /usr/local/nginx/conf/
[root@slave1 conf]# ll
总用量 68
-rw-r--r--. 1 root root 1077 7月 29 21:48 fastcgi.conf
-rw-r--r--. 1 root root 1077 7月 29 21:48 fastcgi.conf.default
-rw-r--r--. 1 root root 1007 7月 29 21:48 fastcgi_params
-rw-r--r--. 1 root root 1007 7月 29 21:48 fastcgi_params.default
-rw-r--r--. 1 root root 2837 7月 29 21:48 koi-utf
-rw-r--r--. 1 root root 2223 7月 29 21:48 koi-win
-rw-r--r--. 1 root root 5170 7月 29 21:48 mime.types
-rw-r--r--. 1 root root 5170 7月 29 21:48 mime.types.default
-rw-r--r--. 1 root root 2656 7月 29 21:48 nginx.conf
-rw-r--r--. 1 root root 2656 7月 29 21:48 nginx.conf.default
-rw-r--r--. 1 root root 636 7月 29 21:48 scgi_params
-rw-r--r--. 1 root root 636 7月 29 21:48 scgi_params.default
-rw-r--r--. 1 root root 664 7月 29 21:48 uwsgi_params
-rw-r--r--. 1 root root 664 7月 29 21:48 uwsgi_params.default
-rw-r--r--. 1 root root 3610 7月 29 21:48 win-utf
[root@slave1 conf]#
```

配置文件中有很多#，开头的表示注释内容，我们去掉所有以 # 开头的段落，精简之后的内容如下：



```
worker_processes 1;

events {
    worker_connections 1024;
}

http {
    include mime.types;
    default_type application/octet-stream;

    sendfile on;

    keepalive_timeout 65;

    server {
        listen 80;
        server_name localhost;

        location / {
            root html;
            index index.html index.htm;
```

根据上述文件，我们可以很明显的将 nginx.conf 配置文件分为三部分：

### 第一部分：全局块

从配置文件开始到 events 块之间的内容，主要会设置一些影响 nginx 服务器整体运行的配置指令，主要包括配置运行 Nginx 服务器的用户（组）、允许生成的 worker process 数，进程 PID 存放路径、日志存放路径和类型以及配置文件的引入等。

比如上面第一行配置的：

```
worker_processes 1;
```

这是 Nginx 服务器并发处理服务的关键配置，worker\_processes 值越大，可以支持的并发处理量也越多，但是会受到硬件、软件等设备的制约

### 第二部分：events 块

比如上面的配置：

```
events {
    worker_connections 1024;
}
```



events 块涉及的指令主要影响 Nginx 服务器与用户的网络连接，常用的设置包括是否开启对多 work process 下的网络连接进行序列化，是否允许同时接收多个网络连接，选取哪种事件驱动模型来处理连接请求，每个 word process 可以同时支持的最大连接数等。

上述例子就表示每个 work process 支持的最大连接数为 1024。

这部分的配置对 Nginx 的性能影响较大，在实际中应该灵活配置。

### 第三部分：http 块

```
1 http {
2     include      mime.types;
3     default_type  application/octet-stream;
4
5
6     sendfile      on;
7
8     keepalive_timeout 65;
9
10    server {
11        listen      80;
12        server_name localhost;
13
14        location / {
15            root      html;
16            index      index.html index.htm;
17        }
18
19        error_page   500 502 503 504 /50x.html;
20        location = /50x.html {
21            root      html;
22        }
23
24    }
25
26 }
```

这算是 Nginx 服务器配置中最频繁的部分，代理、缓存和日志定义等绝大多数功能和第三方模块的配置都在这里。

需要注意的是：http 块也可以包括 **http 全局块**、**server 块**。

#### ①、http 全局块

http 全局块配置的指令包括文件引入、MIME-TYPE 定义、日志自定义、连接超时时间、单链接请求数上限等。

## ②、server 块

这块和虚拟主机有密切关系，虚拟主机从用户角度看，和一台独立的硬件主机是完全一样的，该技术的产生是为了节省互联网服务器硬件成本。

每个 http 块可以包括多个 server 块，而每个 server 块就相当于一个虚拟主机。

而每个 server 块也分为全局 server 块，以及可以同时包含多个 locaton 块。

### 1、全局 server 块

最常见的配置是本虚拟机主机的监听配置和本虚拟主机的名称或 IP 配置。

### 2、location 块

一个 server 块可以配置多个 location 块。

这块的主要作用是基于 Nginx 服务器接收到的请求字符串（例如 server\_name/uri-string），对虚拟主机名称（也可以是 IP 别名）之外的字符串（例如 前面的 /uri-string）进行匹配，对特定的请求进行处理。地址定向、数据缓存和应答控制等功能，还有许多第三方模块的配置也在这里进行。

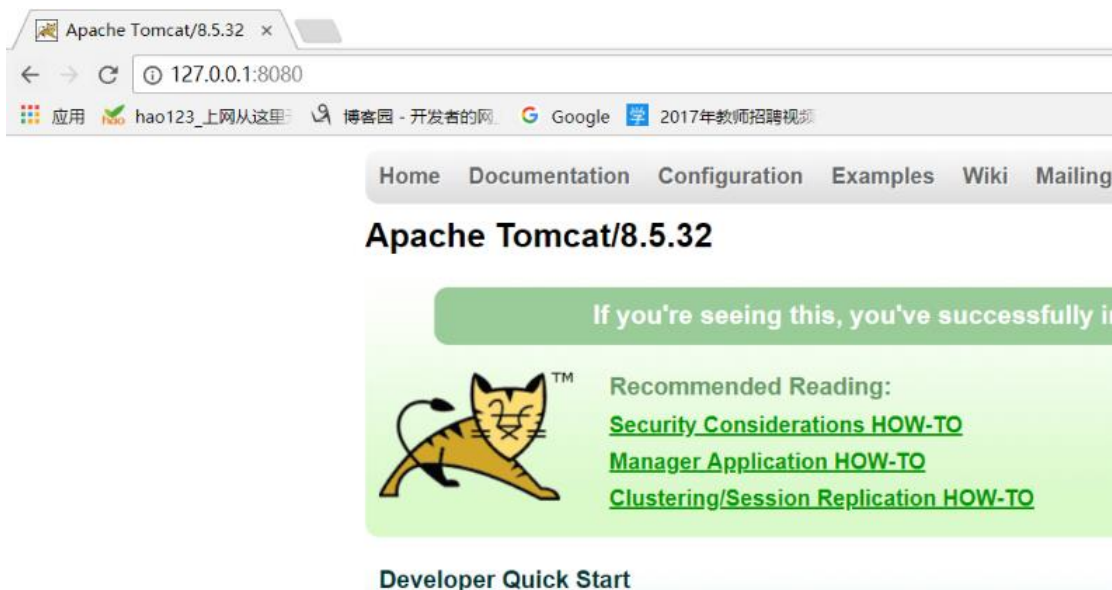
## 第 4 章 nginx 配置实例-反向代理

### 4.1 反向代理实例一

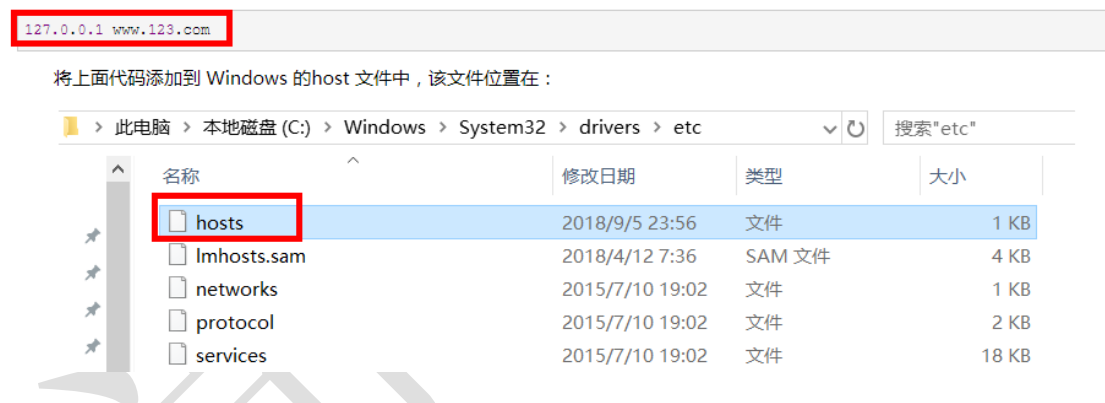
实现效果：使用 nginx 反向代理，访问 [www.123.com](http://www.123.com) 直接跳转到 127.0.0.1:8080

#### 4.1.1 实验代码

- 1) 启动一个 tomcat，浏览器地址栏输入 127.0.0.1:8080，出现如下界面



2) 通过修改本地 host 文件, 将 `www.123.com` 映射到 `127.0.0.1`

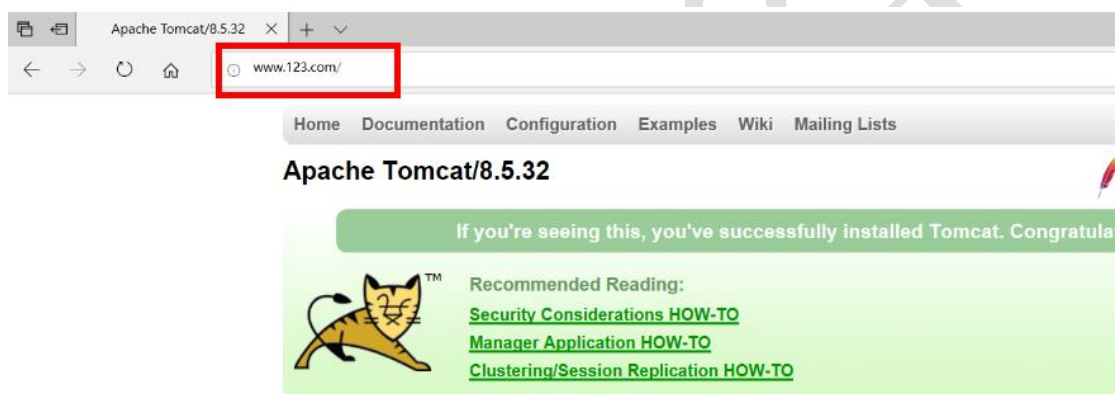


配置完成之后, 我们便可以通过 `www.123.com:8080` 访问到第一步出现的 Tomcat 初始界面。那么如何只需要输入 `www.123.com` 便可以跳转到 Tomcat 初始界面呢? 便用到 nginx 的反向代理。

3) 在 `nginx.conf` 配置文件中增加如下配置

```
server {  
    listen      80;  
    server_name www.123.com;  
  
    location / {  
        proxy_pass http://127.0.0.1:8080;  
        index index.html index.htm index.jsp;  
    }  
}
```

如上配置，我们监听 80 端口，访问域名为 www.123.com，不加端口号时默认为 80 端口，故访问该域名时会跳转到 127.0.0.1:8080 路径上。在浏览器端输入 www.123.com 结果如下：



## 4.3 反向代理实例二

实现效果：使用 nginx 反向代理，根据访问的路径跳转到不同端口的服务中

nginx 监听端口为 9001，

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

访问 <http://127.0.0.1:9001/vod/> 直接跳转到 127.0.0.1:8082

### 4.3.1 实验代码

第一步，准备两个 tomcat，一个 8001 端口，一个 8002 端口，并准备好测试的页面

第二步，修改 nginx 的配置文件

在 http 块中添加 server {}

```
server {  
    listen 9001;  
    server_name localhost;  
  
    location ~ /edu/ {  
        proxy_pass http://localhost:8001;  
    }  
  
    location ~ /vod/ {  
        proxy_pass http://localhost:8002;  
    }  
}
```



### location 指令说明

该指令用于匹配 URL。

语法如下：

```
1 location [ = | ~ | ~* | ^~ ] uri {  
2  
3 }
```

1、=：用于不含正则表达式的 uri 前，要求请求字符串与 uri 严格匹配，如果匹配成功，就停止继续向下搜索并立即处理该请求。

2、~：用于表示 uri 包含正则表达式，并且区分大小写。

3、~\*：用于表示 uri 包含正则表达式，并且不区分大小写。

4、^~：用于不含正则表达式的 uri 前，要求 Nginx 服务器找到标识 uri 和请求字符串匹配度最高的 location 后，立即使用此 location 处理请求，而不再使用 location 块中的正则 uri 和请求字符串做匹配。

注意：如果 uri 包含正则表达式，则必须要有 ~ 或者 ~\* 标识。

## 第 5 章 nginx 配置实例-负载均衡

实现效果：配置负载均衡

## 5.1 实验代码

1) 首先准备两个同时启动的 Tomcat

2) 在 nginx.conf 中进行配置

```
http {  
.....  
    upstream myserver{  
        ip_hash;  
        server 115.28.52.63:8080 weight=1;  
        server 115.28.52.63:8180 weight=1;  
    }  
.....  
    server{  
        location / {  
            .....  
            proxy_pass http://myserver;  
            proxy_connect_timeout 10;  
        }  
        .....  
    }  
}
```

随着互联网信息的爆炸性增长，负载均衡（load balance）已经不再是一个很陌生的话题，顾名思义，负载均衡即是将负载分摊到不同的服务单元，既保证服务的可用性，又保证响应足够快，给用户很好的体验。快速增长的访问量和数据流量催生了各式各样的负载均衡产品，很多专业的负载均衡硬件提供了很好的功能，但却价格不菲，这使得负载均衡软件大受欢迎，nginx 就是其中的一个，在 linux 下有 Nginx、LVS、Haproxy 等等服务可以提供负载均衡服务，而且 Nginx 提供了几种分配方式（策略）：

### 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

weight 代表权重，默认认为 1，权重越高被分配的客户端越多

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 server_pool{
server 192.168.5.21 weight=10;
server 192.168.5.22 weight=10;
}
```

### 3、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 server_pool{
ip_hash;
server 192.168.5.21:80;
server 192.168.5.22:80;
}
```

### 4、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 server_pool{
server 192.168.5.21:80;
server 192.168.5.22:80;
fair;
}
```

## 第 6 章 nginx 配置实例-动静分离

Nginx 动静分离简单来说就是把动态跟静态请求分开，不能理解成只是单纯的把动态页面和静态页面物理分离。严格意义上说应该是动态请求跟静态请求分开，可以理解成使用 Nginx 处理静态页面，Tomcat 处理动态页面。动静分离从目前实现角度来讲大致分为两种，

一种是纯粹把静态文件独立成单独的域名，放在独立的服务器上，也是目前主流推崇的方案；另外一种方法就是动态跟静态文件混合在一起发布，通过 nginx 来分开。

通过 location 指定不同的后缀名实现不同的请求转发。通过 expires 参数设置，可以使浏览器缓存过期时间，减少与服务器之前的请求和流量。具体 Expires 定义：是给一个资源设定一个过期时间，也就是说无需去服务端验证，直接通过浏览器自身确认是否过期即可，所以不会产生额外的流量。此种方法非常适合不经常变动的资源。（如果经常更新的文件，不建议使用 Expires 来缓存），我这里设置 3d，表示在这 3 天之内访问这个 URL，发送一个请求，比对服务器该文件最后更新时间没有变化，则不会从服务器抓取，返回状态码 304，如果有修改，则直接从服务器重新下载，返回状态码 200。



## 6.1 实验代码

1.项目资源准备

2.进行 nginx 配置

找到 nginx 安装目录，打开/conf/nginx.conf 配置文件，

```
server {  
    listen      80;  
    server_name 192.168.17.129;  
  
    #charset koi8-r;  
  
    #access_log logs/host.access.log main;  
  
    location /www/ {  
        root /data/;  
        index index.html index.htm;  
    }  
  
    location /image/ {  
        root /data/;  
        autoindex on;  
    }  
}
```

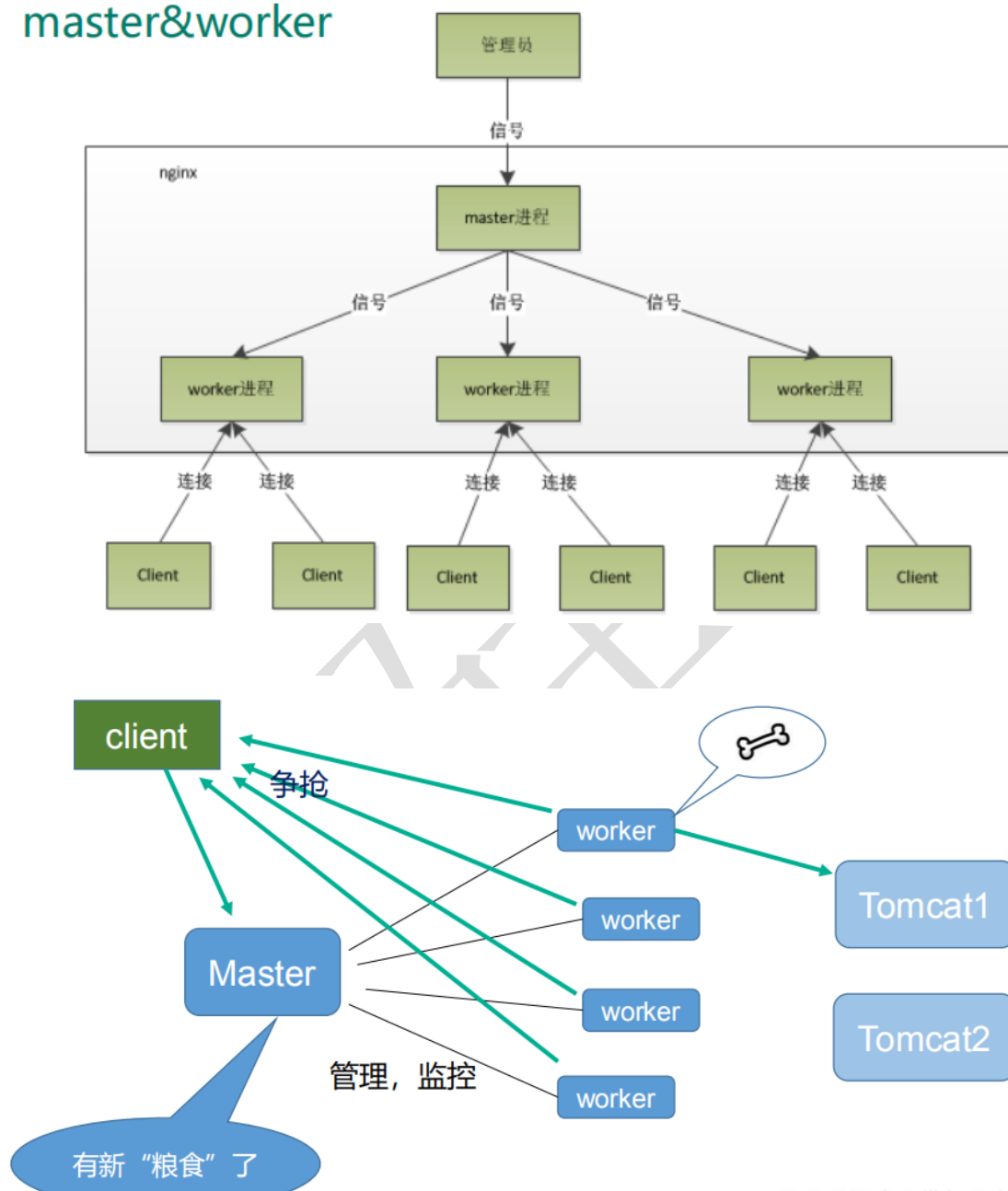
添加监听端口、访问名字

重点是添加 location，

最后检查 Nginx 配置是否正确即可，然后测试动静分离是否成功，之需要删除后端 tomcat 服务器上的某个静态文件，查看是否能访问，如果可以访问说明静态资源 nginx 直接返回了，不走后端 tomcat 服务器

## 第 7 章 nginx 原理与优化参数配置

### master&worker



#### master-workers 的机制的好处

首先，对于每个 **worker** 进程来说，独立的进程，不需要加锁，所以省掉了锁带来的开销，同时在编程以及问题查找时，也会方便很多。其次，采用独立的进程，可以让互相之间不会互相影响，一个进程退出后，其它进程还在工作，服务不会中断，**master** 进程则很快启动新的 **worker** 进程。当然，**worker** 进程的异常退出，肯定是程序有 **bug** 了，异常退出，会导致当前 **worker** 上的所有请求失败，不过不会影响到所有请求，所以降低了风险。

### 需要设置多少个 worker

Nginx 同 redis 类似都采用了 io 多路复用机制，每个 worker 都是一个独立的进程，但每个进程里只有一个主线程，通过异步非阻塞的方式来处理请求，即使是千上万个请求也不在话下。每个 worker 的线程可以把一个 cpu 的性能发挥到极致。所以 worker 数和服务器的 cpu 数相等是最为适宜的。设少了会浪费 cpu，设多了会造成 cpu 频繁切换上下文带来的损耗。

### #设置 worker 数量。

```
worker_processes 4
```

#work 绑定 cpu(4 work 绑定 4cpu)。

```
worker_cpu_affinity 0001 0010 0100 1000
```

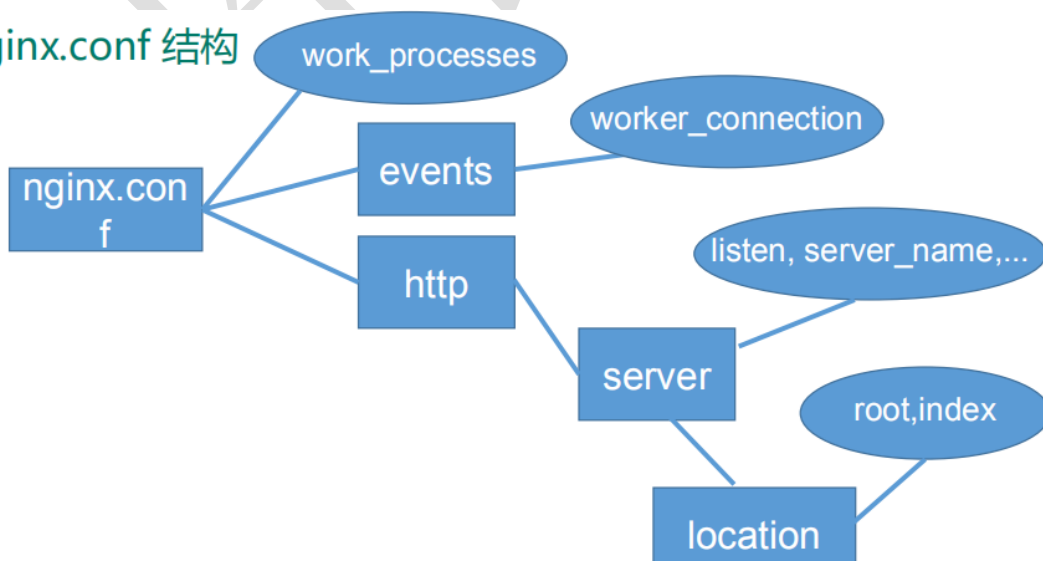
#work 绑定 cpu (4 work 绑定 8cpu 中的 4 个) 。

```
worker_cpu_affinity 0000001 00000010 00000100 00001000
```

### 连接数 worker\_connection

这个值是表示每个 worker 进程所能建立连接的最大值，所以，一个 nginx 能建立的最大连接数，应该是  $worker\_connections * worker\_processes$ 。当然，这里说的是最大连接数，对于 HTTP 请求本地资源来说，能够支持的最大并发数量是  $worker\_connections * worker\_processes$ ，如果是支持 http1.1 的浏览器每次访问要占两个连接，所以普通的静态访问最大并发数是： $worker\_connections * worker\_processes / 2$ ，而如果是 HTTP 作为反向代理来说，最大并发数量应该是  $worker\_connections * worker\_processes / 4$ 。因为作为反向代理服务器，每个并发会建立与客户端的连接和与后端服务的连接，会占用两个连接。

### nginx.conf 结构

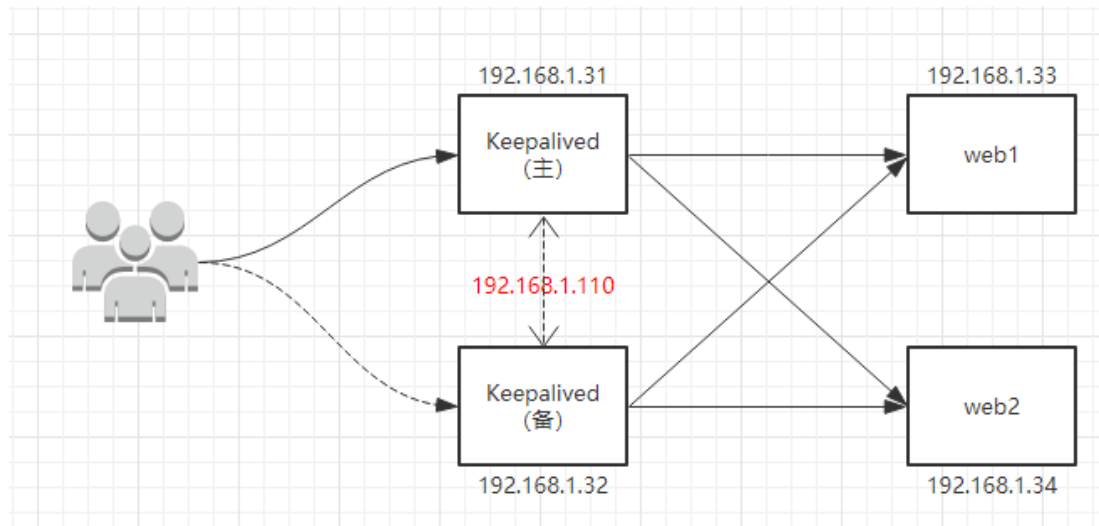


详情见配置文件 nginx.conf

## 第 8 章 nginx 搭建高可用集群

### 8.1 Keepalived+Nginx 高可用集群（主从模式）

集群架构图：



```
global_defs {
    notification_email {
        acassen@firewall.loc
        failover@firewall.loc
        sysadmin@firewall.loc
    }
    notification_email_from Alexandre.Cassen@firewall.loc
    smtp_server 192.168.17.129
    smtp_connect_timeout 30
    router_id LVS_DEVEL
}
```

```
vrrp_script chk_http_port {

    script "/usr/local/src/nginx_check.sh"

    interval 2      #（检测脚本执行的间隔）

    weight 2
```

```
}

vrrp_instance VI_1 {
    state BACKUP # 备份服务器上将 MASTER 改为 BACKUP
    interface ens33 //网卡
    virtual_router_id 51 # 主、备机的 virtual_router_id 必须相同
    priority 100 # 主、备机取不同的优先级，主机值较大，备份机值较小
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.17.50 // VRRP H 虚拟地址
    }
}

#!/bin/bash
A=`ps -C nginx -no-header |wc -l`
if [ $A -eq 0 ];then
    /usr/local/nginx/sbin/nginx
    sleep 2
    if [ `ps -C nginx --no-header |wc -l` -eq 0 ];then
        killall keepalived
    fi
fi
```

#### (1) 在所有节点上面进行配置

```
# systemctl stop firewalld //关闭防火墙
# sed -i 's/^SELINUX=.*/SELINUX=disabled/' /etc/sysconfig/selinux //关闭 selinux，重启生效
# setenforce 0 //关闭 selinux，临时生效
# ntpdate 0.centos.pool.ntp.org //时间同步
# yum install nginx -y //安装 nginx
```

#### (2) 配置后端 web 服务器（两台一样）

```
# echo ``hostname` `ifconfig ens33 |sed -n 's#.*inet \(.*\)netmask.*#\1#p``" >
/usr/share/nginx/html/index.html //准备测试文件，此处是将主机名和 ip 写到 index.html 页面中
# vim /etc/nginx/nginx.conf //编辑配置文件
```

```
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;
include /usr/share/nginx/modules/*.conf;
events {
    worker_connections 1024;
}
http {
    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 /var/log/nginx/access.log main;
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    include /etc/nginx/conf.d/*.conf;
    server {
        listen 80;
        server_name www.mtian.org;
        location / {
            root /usr/share/nginx/html;
        }
        access_log /var/log/nginx/access.log main;
    }
}
# systemctl start nginx //启动 nginx
# systemctl enable nginx //加入开机启动
```

### (3) 配置 LB 服务器（两台都一样）

```
# vim /etc/nginx/nginx.conf
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;
include /usr/share/nginx/modules/*.conf;
events {
```

```
worker_connections 1024;
}
http {
    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 /var/log/nginx/access.log main;
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    include /etc/nginx/conf.d/*.conf;
    upstream backend {
        server 192.168.1.33:80 weight=1 max_fails=3 fail_timeout=20s;
        server 192.168.1.34:80 weight=1 max_fails=3 fail_timeout=20s;
    }
    server {
        listen 80;
        server_name www.mtian.org;
        location / {
            proxy_pass http://backend;
            proxy_set_header Host $host:$proxy_port;
            proxy_set_header X-Forwarded-For $remote_addr;
        }
    }
}
# systemctl start nginx //启动 nginx
# systemctl enable nginx //加入开机自启动
```

(4) 在测试机 (192.168.1.35) 上面添加 host 解析, 并测试 lb 集群是否正常。(测试机任意都可以, 只要能访问 lb 节点。)

```
[root@node01 ~]# cat /etc/hosts
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.1.32 www.mtian.org
192.168.1.31 www.mtian.org
// 测试时候轮流关闭 lb1 和 lb2 节点, 关闭后还是能够访问并看到轮循效果即表示 nginx lb 集群搭建成功。
[root@node01 ~]# curl www.mtian.org
```



```
web01 192.168.1.33
[root@node01 ~]# curl www.mtian.org
web02 192.168.1.34
[root@node01 ~]# curl www.mtian.org
web01 192.168.1.33
[root@node01 ~]# curl www.mtian.org
web02 192.168.1.34
[root@node01 ~]# curl www.mtian.org
web01 192.168.1.33
[root@node01 ~]# curl www.mtian.org
web02 192.168.1.34
```

(5) 上面步骤成功后, 开始搭建 keepalived, 在两台 lb 节点上面安装 keepalived (也可以源码编译安装、此处直接使用 yum 安装)

```
# yum install keepalived -y
```

#### (6) 配置 LB-01 节点

```
[root@LB-01 ~]# vim /etc/keepalived/keepalived.conf
! Configuration File for keepalived

global_defs {
    notification_email {
        381347268@qq.com
    }
    smtp_server 192.168.200.1
    smtp_connect_timeout 30
    router_id LVS_DEVEL
}

vrrp_instance VI_1 {
    state MASTER
    interface ens33
    virtual_router_id 51
    priority 150
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.1.110/24 dev ens33 label ens33:1
    }
}
```

```
}
[root@LB-01 ~]# systemctl start keepalived      //启动 keepalived
[root@LB-01 ~]# systemctl enable keepalived     //加入开机自启动

[root@LB-01 ~]# ip a      //查看 IP，会发现多出了 VIP 192.168.1.110
.....
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:94:17:44 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.31/24 brd 192.168.1.255 scope global ens33
        valid_lft forever preferred_lft forever
    inet 192.168.1.110/24 scope global secondary ens33:1
        valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe94:1744/64 scope link
        valid_lft forever preferred_lft forever
.....
```

### (7) 配置 LB-02 节点

```
[root@LB-02 ~]# vim /etc/keepalived/keepalived.conf
! Configuration File for keepalived

global_defs {
    notification_email {
        381347268@qq.com
    }
    smtp_server 192.168.200.1
    smtp_connect_timeout 30
    router_id LVS_DEVEL
}

vrrp_instance VI_1 {
    state BACKUP
    interface ens33
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.1.110/24 dev ens33 label ens33:1
    }
}
```

```
}

[root@LB-02 ~]# systemctl start keepalived          //启动 keepalived
[root@LB-02 ~]# systemctl enable keepalived         //加入开机自启动

[root@LB-02 ~]# ifconfig    //查看 IP，此时备节点不会有 VIP（只有当主挂了的时候，VIP 才会飘到备节点）
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 192.168.1.32  netmask 255.255.255.0  broadcast 192.168.1.255
    inet6 fe80::20c:29ff:feab:6532  prefixlen 64  scopeid 0x20<link>
    ether 00:0c:29:ab:65:32  txqueuelen 1000  (Ethernet)
    RX packets 43752  bytes 17739987 (16.9 MiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 4177  bytes 415805 (406.0 KiB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
.....
```

#### （8）在测试机器上面访问 Keepalived 上面配置的 VIP 192.168.1.110

```
[root@node01 ~]# curl 192.168.1.110
web01 192.168.1.33
[root@node01 ~]# curl 192.168.1.110
web02 192.168.1.34
[root@node01 ~]# curl 192.168.1.110
web01 192.168.1.33
[root@node01 ~]# curl 192.168.1.110
web02 192.168.1.34

//关闭 LB-01 节点上面 keepalived 主节点。再次访问
[root@LB-01 ~]# systemctl stop keepalived
[root@node01 ~]#
[root@node01 ~]# curl 192.168.1.110
web01 192.168.1.33
[root@node01 ~]# curl 192.168.1.110
web02 192.168.1.34
[root@node01 ~]# curl 192.168.1.110
web01 192.168.1.33
[root@node01 ~]# curl 192.168.1.110
web02 192.168.1.34

//此时查看 LB-01 主节点上面的 IP，发现已经没有了 VIP
[root@LB-01 ~]# ifconfig
```

```
ens33: flags=4163<UP, BROADCAST, RUNNING, MULTICAST> mtu 1500
    inet 192.168.1.31 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::20c:29ff:fe94:1744 prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:94:17:44 txqueuelen 1000 (Ethernet)
    RX packets 46813 bytes 18033403 (17.1 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 9350 bytes 1040882 (1016.4 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

...

//查看 LB-02 各节点上面的 IP，发现 VIP 已经成功飘过来了

[root@LB-02 ~]# ifconfig

```
ens33: flags=4163<UP, BROADCAST, RUNNING, MULTICAST> mtu 1500
    inet 192.168.1.32 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::20c:29ff:feab:6532 prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:ab:65:32 txqueuelen 1000 (Ethernet)
    RX packets 44023 bytes 17760070 (16.9 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 4333 bytes 430037 (419.9 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

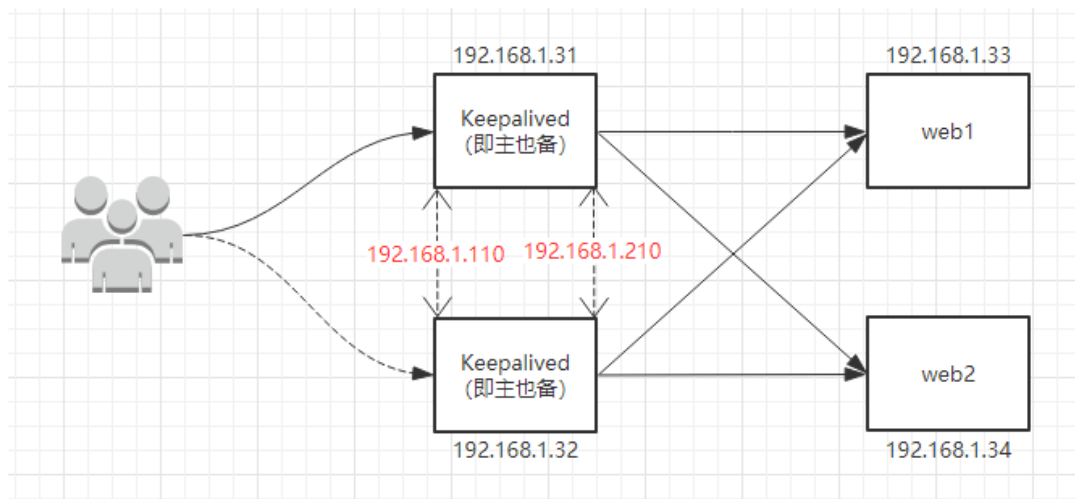
```
ens33:1: flags=4163<UP, BROADCAST, RUNNING, MULTICAST> mtu 1500
    inet 192.168.1.110 netmask 255.255.255.0 broadcast 0.0.0.0
    ether 00:0c:29:ab:65:32 txqueuelen 1000 (Ethernet)
```

...

到此，Keepalived+Nginx 高可用集群就搭建完成了。

## 8.2 Keepalived+Nginx 高可用集群（双主模式）

集群架构图：



说明：还是按照上面的环境继续做实验，只是修改 LB 节点上面的 keepalived 服务的配置文件即可。此时 LB-01 节点即为 Keepalived 的主节点也为备节点，LB-02 节点同样即为 Keepalived 的主节点也为备节点。LB-01 节点默认的主节点 VIP（192.168.1.110），LB-02 节点默认的主节点 VIP（192.168.1.210）

### （1）配置 LB-01 节点

[root@LB-01 ~]# vim /etc/keepalived/keepalived.conf //编辑配置文件，增加一段新的 vrrp\_instance 规则

! Configuration File for keepalived

```
global_defs {
    notification_email {
        381347268@qq.com
    }
    smtp_server 192.168.200.1
    smtp_connect_timeout 30
    router_id LVS_DEVEL
}

vrrp_instance VI_1 {
    state MASTER
    interface ens33
    virtual_router_id 51
    priority 150
    advert_int 1
    authentication {
```

```
    auth_type PASS
    auth_pass 1111
}
virtual_ipaddress {
    192.168.1.110/24 dev ens33 label ens33:1
}
}

vrrp_instance VI_2 {
    state BACKUP
    interface ens33
    virtual_router_id 52
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 2222
    }
    virtual_ipaddress {
        192.168.1.210/24 dev ens33 label ens33:2
    }
}

[root@LB-01 ~]# systemctl restart keepalived    //重新启动 keepalived

// 查看 LB-01 节点的 IP 地址，发现 VIP (192.168.1.110) 同样还是默认在该节点
[root@LB-01 ~]# ip a
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:94:17:44 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.31/24 brd 192.168.1.255 scope global ens33
        valid_lft forever preferred_lft forever
    inet 192.168.1.110/24 scope global secondary ens33:1
        valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe94:1744/64 scope link
        valid_lft forever preferred_lft forever
```

## (2) 配置 LB-02 节点

```
[root@LB-02 ~]# vim /etc/keepalived/keepalived.conf    //编辑配置文件，增加一段新的
vrrp_instance 规则
! Configuration File for keepalived

global_defs {
```

```
notification_email {
    381347268@qq.com
}

smtp_server 192.168.200.1
smtp_connect_timeout 30
router_id LVS_DEVEL
}

vrrp_instance VI_1 {
    state BACKUP
    interface ens33
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.1.110/24 dev ens33 label ens33:1
    }
}

vrrp_instance VI_2 {
    state MASTER
    interface ens33
    virtual_router_id 52
    priority 150
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 2222
    }
    virtual_ipaddress {
        192.168.1.210/24 dev ens33 label ens33:2
    }
}

[root@LB-02 ~]# systemctl restart keepalived    //重新启动 keepalived
// 查看 LB-02 节点 IP，会发现也多了一个 VIP（192.168.1.210），此时该节点也就是一个主了。
[root@LB-02 ~]# ip a
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
```



```
link/ether 00:0c:29:ab:65:32 brd ff:ff:ff:ff:ff:ff
inet 192.168.1.32/24 brd 192.168.1.255 scope global ens33
    valid_lft forever preferred_lft forever
inet 192.168.1.210/24 scope global secondary ens33:2
    valid_lft forever preferred_lft forever
inet6 fe80::20c:29ff:feab:6532/64 scope link
    valid_lft forever preferred_lft forever
```

### (3) 测试

```
[root@node01 ~]# curl 192.168.1.110
web01 192.168.1.33
[root@node01 ~]# curl 192.168.1.110
web02 192.168.1.34
[root@node01 ~]# curl 192.168.1.210
web01 192.168.1.33
[root@node01 ~]# curl 192.168.1.210
web02 192.168.1.34

// 停止 LB-01 节点的 keepalived 再次测试
[root@LB-01 ~]# systemctl stop keepalived
[root@node01 ~]# curl 192.168.1.110
web01 192.168.1.33
[root@node01 ~]# curl 192.168.1.110
web02 192.168.1.34
[root@node01 ~]# curl 192.168.1.210
web01 192.168.1.33
[root@node01 ~]# curl 192.168.1.210
web02 192.168.1.34
```

测试可以发现我们访问 keepalived 中配置的两个 VIP 都可以正常调度等,当我们停止任意一台 keepalived 节点, 同样还是正常访问; 到此, keepalived+nginx 高可用集群 (双主模式) 就搭建完成了。