**Nginx**

**内容概览**

1、nginx 简介

* 介绍 nginx 的应用场景和具体可以做什么事情
* 介绍什么是反向代理
* 介绍什么是负载均衡
* 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

* 介绍 nginx 启动、关闭、重新加载命令
* 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. **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

**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>

**正向代理**

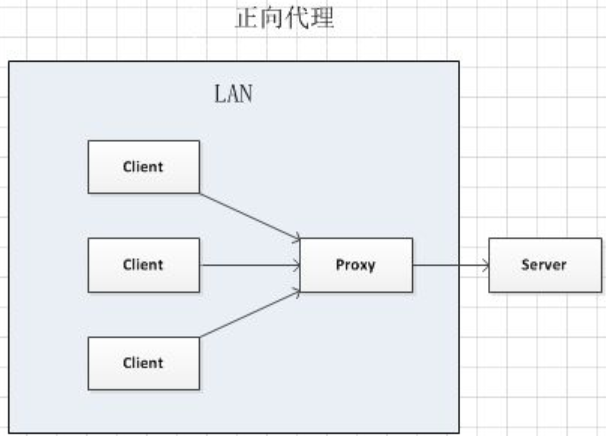
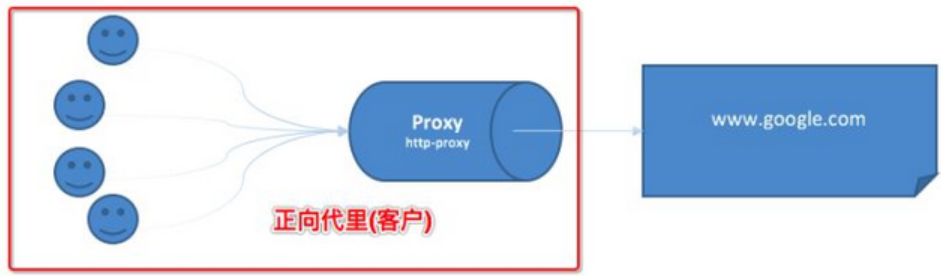
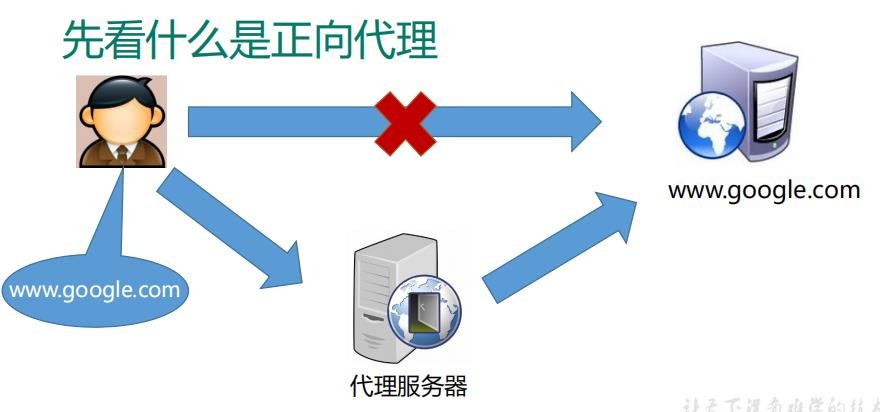
刚开始的时候，代理多数是帮助内网client访问外网server用的，由内到外。后来出现了反向代理，"反向"这个词在这儿的意思其实是指方向相反，即代理将来自外网客户端的请求转发到内网服务器，从外到内 。

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。

正向代理:需要在客户端配置代理服务器进行指定网站访问

比如我们国内访问谷歌，直接访问访问不到，我们可以通过一个正向代理服务器，请求发到代理服务器，代理服务器能够访问谷歌，这样由代理去谷歌取到返回数据，再返回给我们，这样我们就能访问谷歌了，而国内就相当于内网，国外就是外网。



正向代理的用途：

　　（1）访问原来无法访问的资源，如google

（2） 可以做缓存，加速访问资源

　　（3）对客户端访问授权，上网进行认证

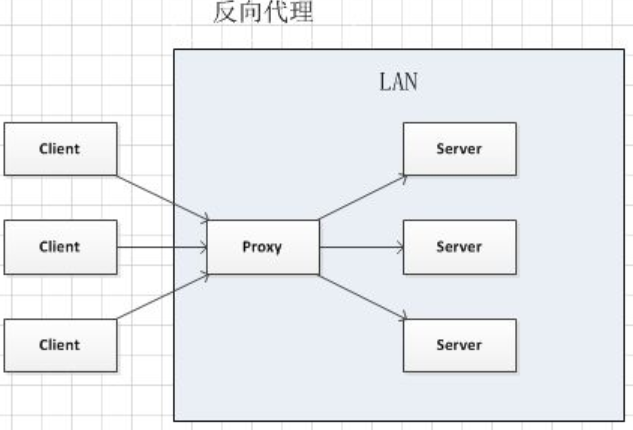
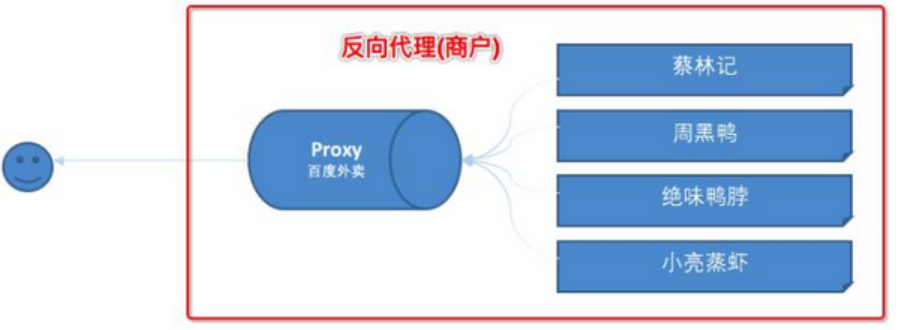
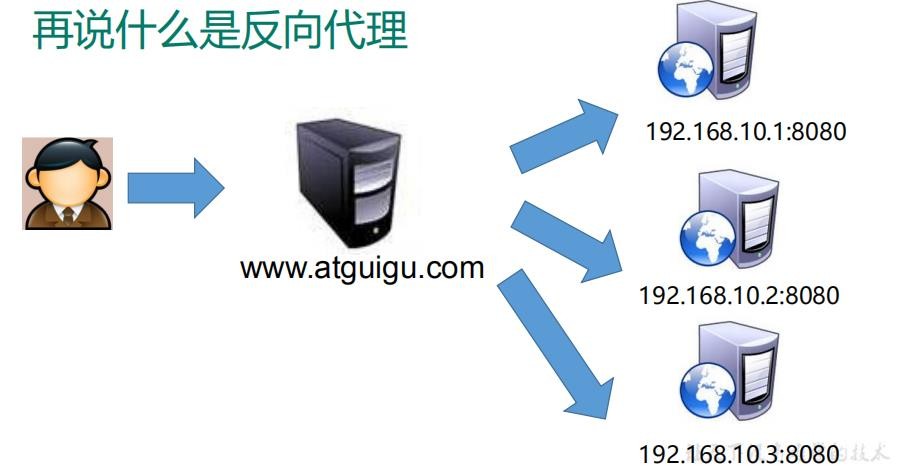
　　（4）代理可以记录用户访问记录（上网行为管理），对外隐藏用户信息

**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。

反向代理暴露的是代理服务器地址，隐藏了真实服务器 IP 地址。

反向代理（Reverse Proxy）实际运行方式是指以代理服务器接受来自外网internet上的连接请求，然后将请求转发给内部网络上的服务器，并将从内网服务器上得到的结果返回给外网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服务器



反向代理的作用：

（1）保证内网的安全，阻止web攻击，大型网站，通常将反向代理作为公网访问地址，Web服务器是内网

（2）负载均衡，通过反向代理服务器来优化网站的负载

正向代理中，proxy和client同属一个LAN，对server透明；  
反向代理中，proxy和server同属一个LAN，对client透明。  
实际上proxy在两种代理中做的事都是代为收发请求和响应，不过从结构上来看正好左右互换了下，所以把后出现的那种代理方式叫成了反向代理c

虽然正向代理服务器和反向代理服务器所处的位置都是客户端和真实服务器之间，所做的事情也都是把客户端的请求转发给服务器，再把服务器的响应转发给客户端，但是二者之间还是有一定的差异的。

1、正向代理其实是客户端的代理，帮助客户端访问其无法访问的服务器资源。反向代理则是服务器的代理，帮助服务器做负载均衡，安全防护等。

2、正向代理一般是客户端架设的，比如在自己的机器上安装一个代理软件。而反向代理一般是服务器架设的，比如在自己的机器集群中部署一个反向代理服务器。

3、正向代理中，服务器不知道真正的客户端到底是谁，以为访问自己的就是真实的客户端。而在反向代理中，客户端不知道真正的服务器是谁，以为自己访问的就是真实的服务器。

4、正向代理和反向代理的作用和目的不同。正向代理主要是用来解决访问限制问题。而反向代理则是提供负载均衡、安全防护等作用。二者均能提高访问速度。

**总结：**

正向代理: 买票的黄牛  
反向代理: 租房的代理

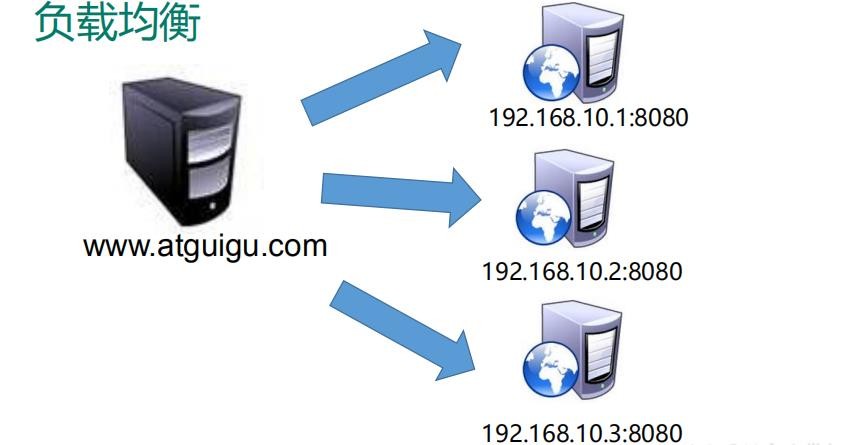
**负载均衡**

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响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](https://www.cnblogs.com/ysocean/p/7641540.html)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**



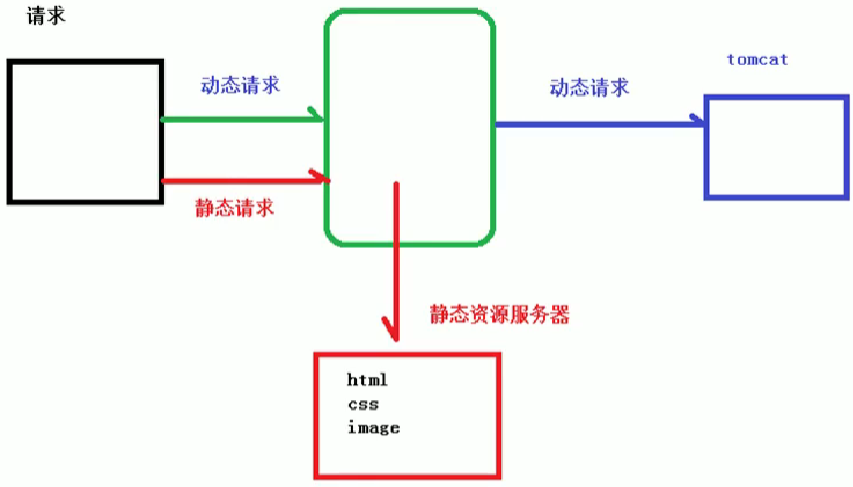
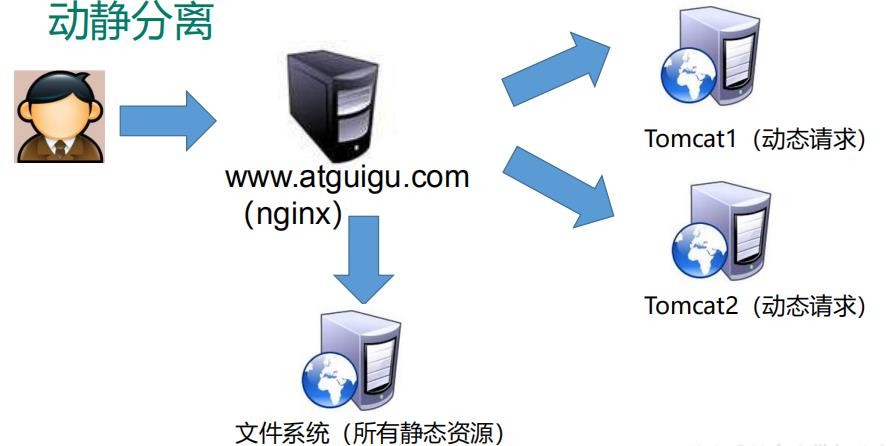
**动静分离**

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 处理静态页面如加载某些图片，Tomcat 处理动态页面如请求数据库。动静分离从目前实现角度来讲大致分为两种，

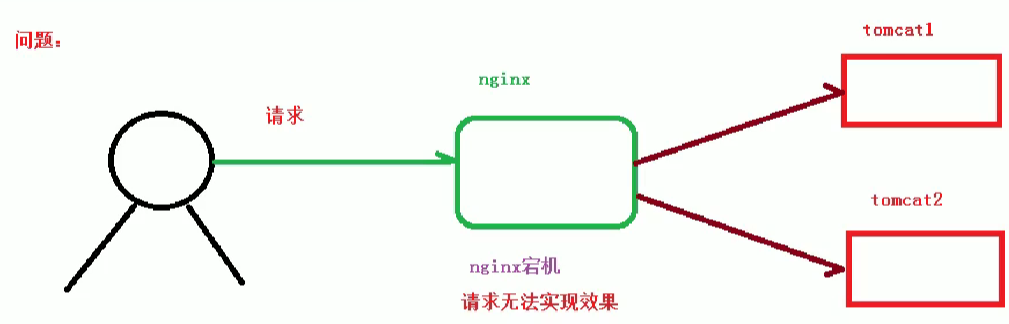
一种是纯粹把静态文件独立成单独的域名，放在独立的服务器上，也是目前主流推崇的方案；

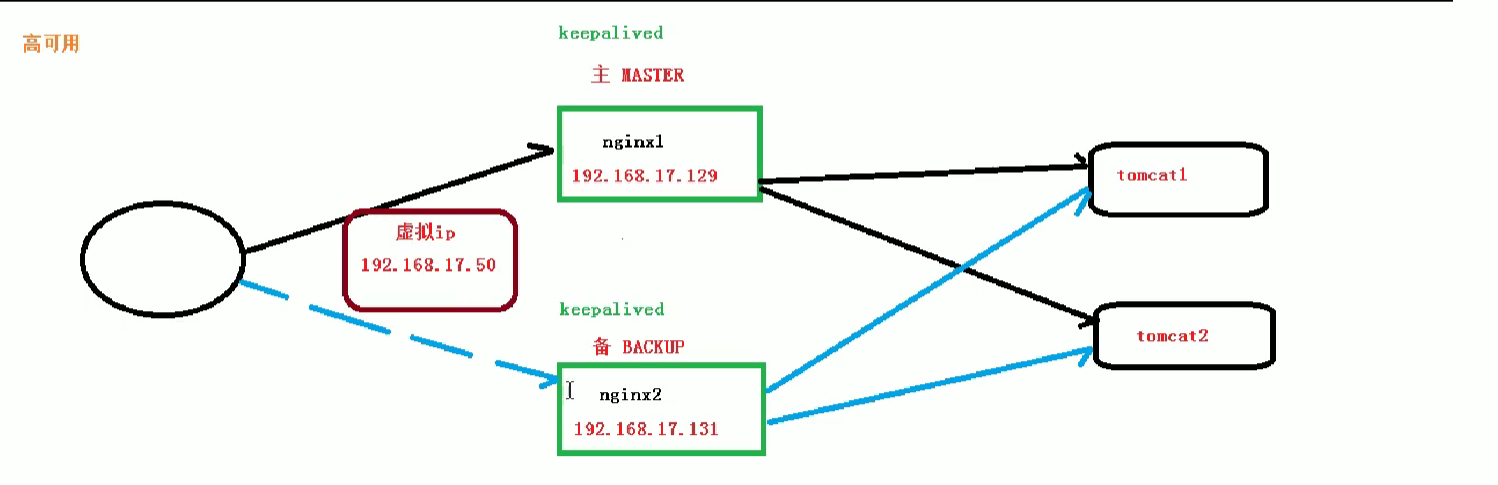
另外一种方法就是动态跟静态文件混合在一起发布，通过 nginx 来分开。

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。



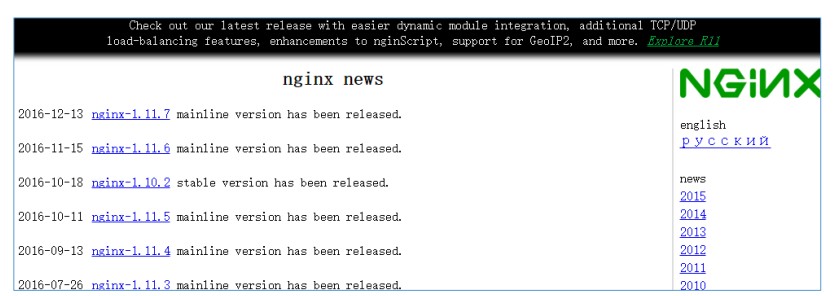
**高可用**



当Nginx宕机后则，用户的请求就不能再分配到后面的服务器了，为了保证较高的可用性，可以准备两台Nginx分别为master、backup，且用keepalived软件绑定一个虚拟ip分配给两台Nginx，keepalived作用类似于一个脚本当检测到作为master的Nginx宕机就会自动启用backup的Nginx，这样就保证了高可用性。

**第 2 章 Nginx 安装**

**进入 nginx 官网，下载**

<http://nginx.org/>



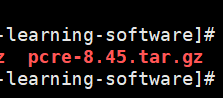
**安装 nginx**

需要pcre、openssl、zlib、gcc依赖

**第一步，安装 pcre依赖**

**联网手动下载 pcre 压缩文件依赖安装**

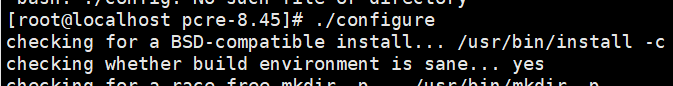
wget <http://downloads.sourceforge.net/project/pcre/pcre/8.37/pcre-8.37.tar.gz>



**解压压缩文件使用命令 tar –xvf pcre-8.37.tar.gz**



**进入**cd pcre-8.45**后输入./configure 完成后，**



若末尾有则需要先安装G++再./configure

**回到 pcre 目录下执行 make，最后执行 make install或者直接两者一起**

**make && make install**

**查看版本pcre-config --version**

****

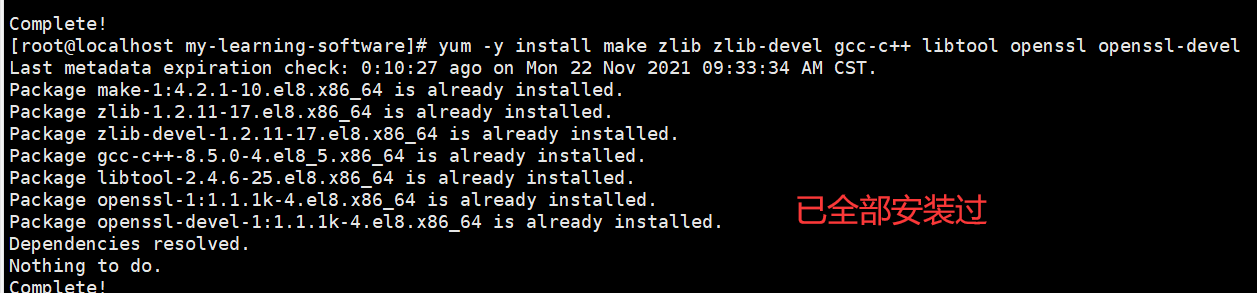
**第二步，安装 openssl**

**第三步，安装 zlib**

**使用yum命令进行一键安装 openssl 、zlib 、 gcc 依赖**

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





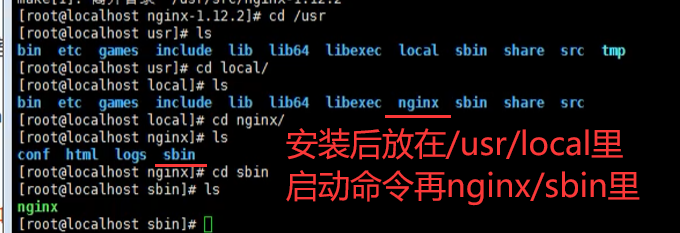
**第四步，安装 nginx**

1、 解压缩 nginx-xx.tar.gz 包。

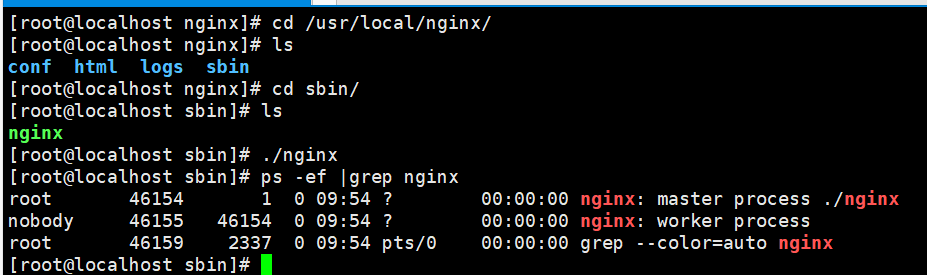
2、 进入解压缩目录，执行./configure。

3 、 make && make install

**安装完成能后会自动放在/usr/loacl里有nginx文件夹，里面的sbin有启动命令**

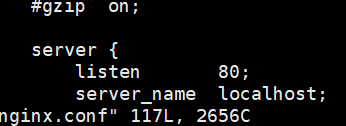


**进入目录 /usr/local/nginx/sbin/nginx/sbin 后输入./nginx启动服务**

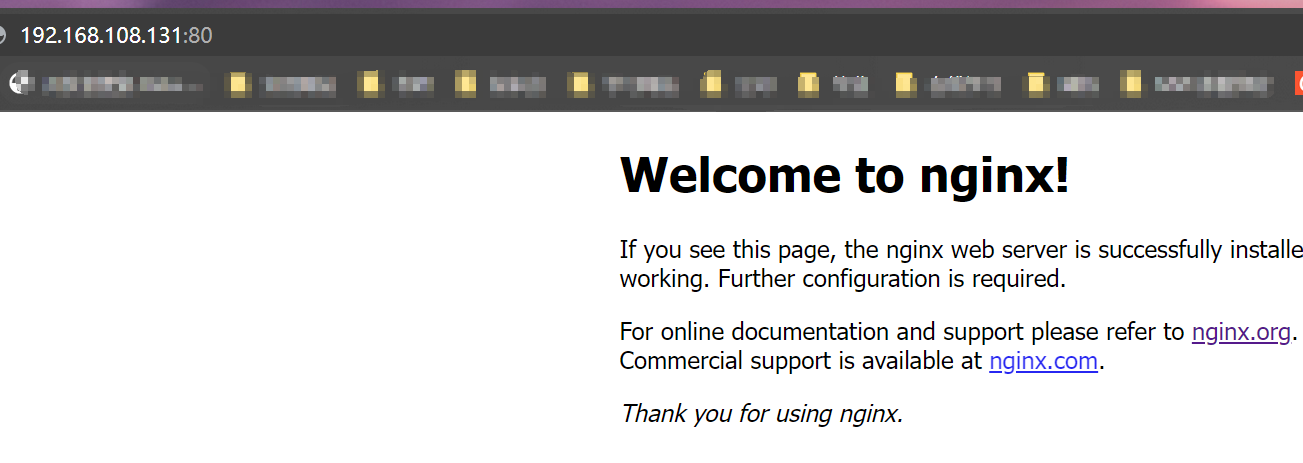


**第五步，访问nginx**

可以看到Nginx默认配置端口号是80



访问虚拟机IP:80即可看到Nginx，若不能访问则查看防火墙是否开启80端口号或者直接关闭防火墙



**查看开放的80端口号**

firewall-cmd --list-all

**设置开放的80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**

**nginx 常用的命令：**

1. **启动命令**

**在/usr/local/nginx/sbin 目录下执行 ./nginx**

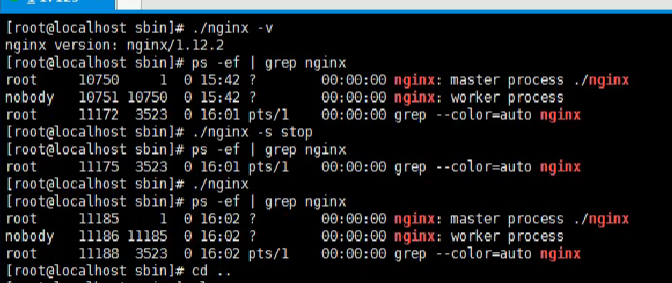
**查看 nginx 版本号**

**./nginx -v**



**2.关闭命令**

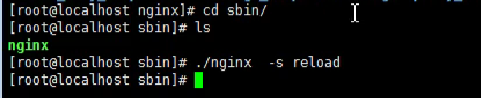
**在/usr/local/nginx/sbin 目录下执行 ./nginx -s stop**



**3.重新加载命令**

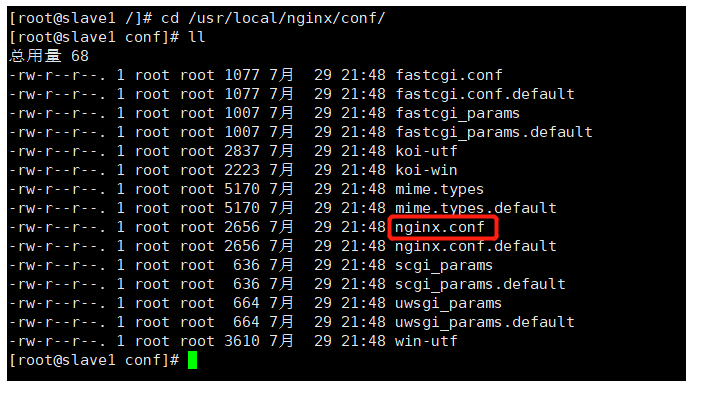
**在更改了配置文件后可以进行重新加载配置文件，不用整个Nginx重启**

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



**nginx.conf 配置文件**

配置文件位置：



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

#user nobody;

worker\_processes 1;

#error\_log logs/error.log;

#error\_log logs/error.log notice;

#error\_log logs/error.log info;

#pid logs/nginx.pid;

events {

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;

keepalive\_timeout 65;

#gzip on;

server {

listen 80;

server\_name localhost;

"nginx.conf" 117L, 2656C

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;

"nginx.conf" 117L, 2656C

**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**

#user nobody;

worker\_processes 1;

#error\_log logs/error.log;

#error\_log logs/error.log notice;

#error\_log logs/error.log info;

#pid logs/nginx.pid;

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：

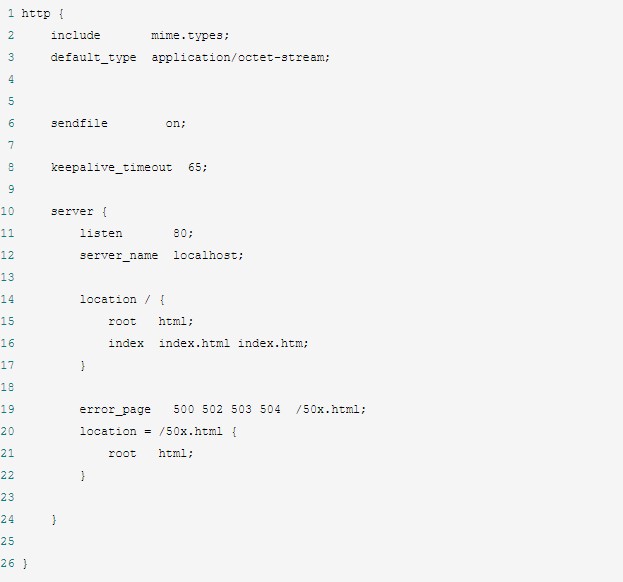


events 块涉及的指令主要影响 Nginx 服务器与用户的网络连接，常用的设置包括是否开启对多 work process 下的网络连接进行序列化、是否允许同时接收多个网络连接、选取哪种事件驱动模型来处理连接请求、每个 word process 可以同时支持的最大连接数等。

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**



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。

如server\_name localhost;

设定服务名称是localhost。

nginx允许多个Server服务在一个nginx实例中。

并允许Server有一个或多个名称，同时支持通配符等。具体参见Docs

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**

**反向代理实例一**

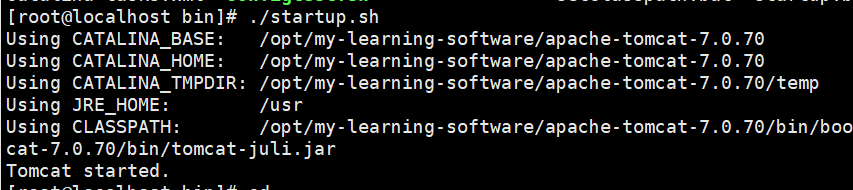
实现效果： **打开浏览器，使用nginx 反向代理，在浏览器地址栏输入地址** [***www.123.com***](http://www.123.com/)**，跳转到** 127.0.0.1:8080**即liunx 系统开启的tomcat 主页面中**

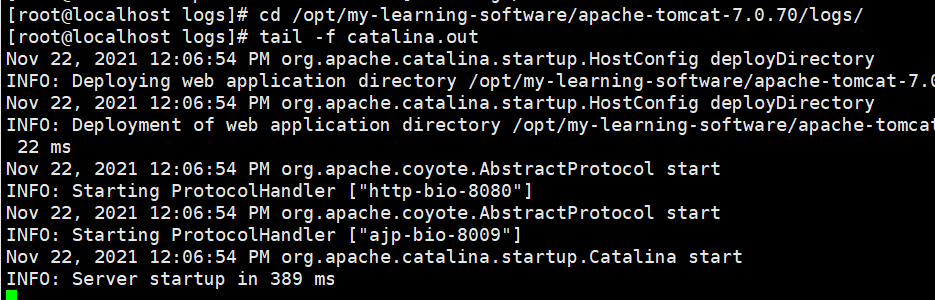
1. **实验代码**

需要先在linux安装JDK，<https://www.cnblogs.com/zyh-smile/p/12353123.html>

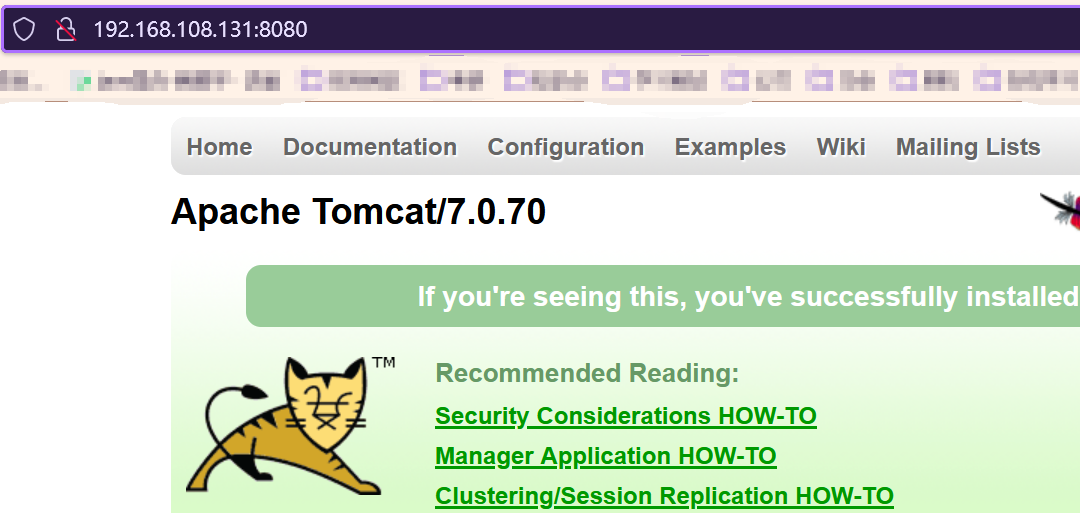
**在 liunx 系统安装 tomcat，使用默认端口 8080**

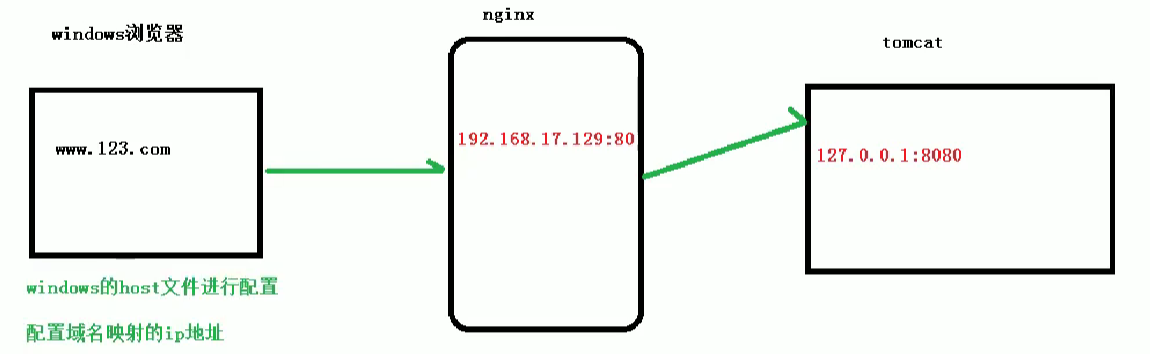
* **tomcat 安装文件放到 liunx 系统中，解压**
* **进入 tomcat 的 bin 目录中，./startup.sh 启动 tomcat 服务器或者使用**systemctl start tomcat**命令，可以在tomcat 的logs目录中** tail -f catalina.out**看日志知道tomcat已经启动**



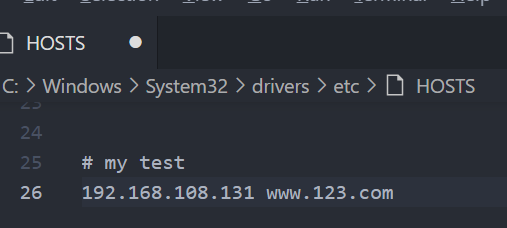
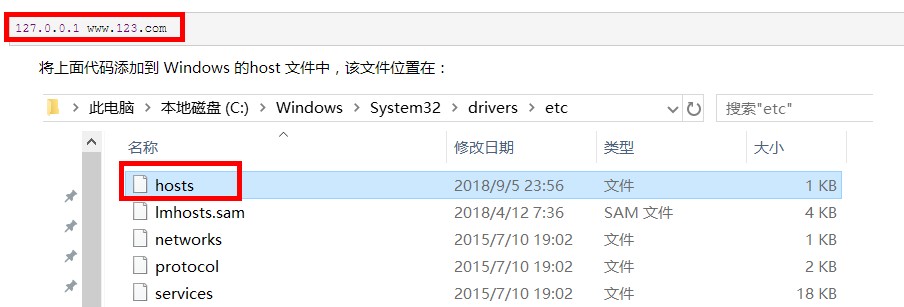


浏览器地址栏输入 虚拟机IP:8080，出现如下界面，可以现在修改windows本地的host文件，因为浏览器访问地址是先找host文件里的地址，若没有则再去网络上找





1. 通过修改本地 host 文件，将 [www.123.com](http://www.123.com/) 映射到 虚拟机IP



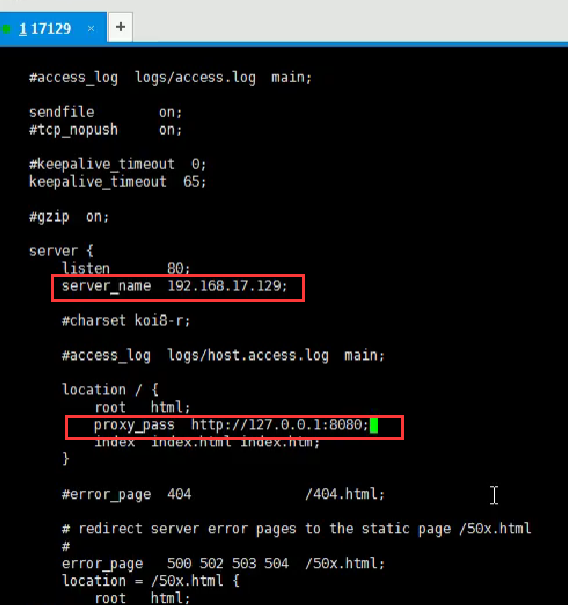
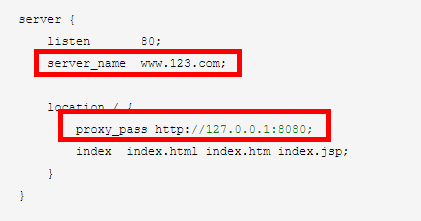
配置完成之后，我们便可以通过 www.123.com:8080 访问到第一步出现的 Tomcat 初始界面。

那么如何只需要输入 [www.123.com](http://www.123.com/) 便可以跳转到 Tomcat 初始界面呢？

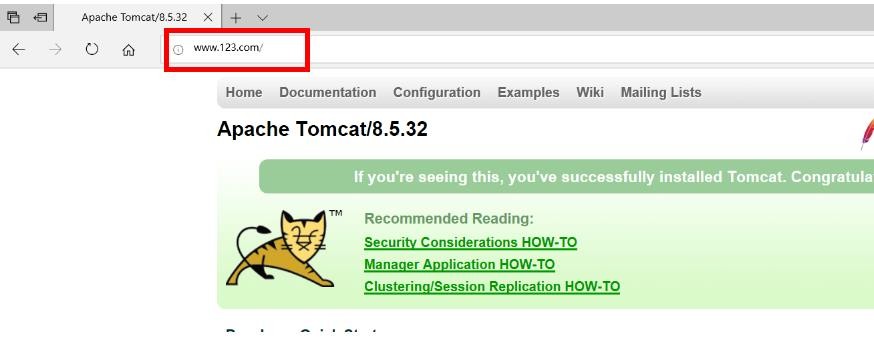
1. **在 nginx.conf 配置文件中增加如下配置**

便用到 nginx 的反向代理，在/usr/local/nginx/conf/nginx.conf配置文件中(**修改配置文件前记得先备份**)，修改完配置文件后再用./nginx -s reload命令让Nginx重加载配置文件

备份



如上配置，我们监听 80 端口，访问域名为 [www.123.com](http://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(127.0.0.1是本机)。在浏览器端输入 [www.123.com](http://www.123.com/) 结果如下：



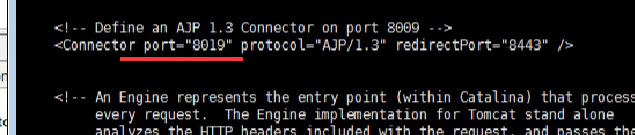
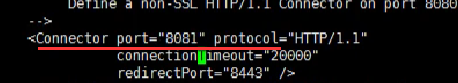
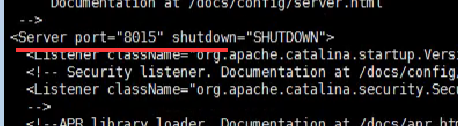
**反向代理实例二**

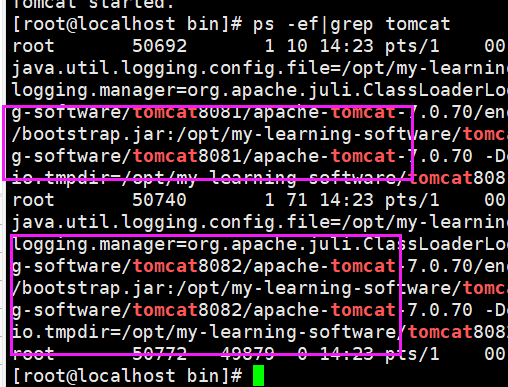
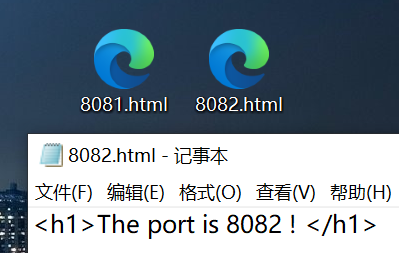
实现效果：使用 nginx 反向代理，根据访问的路径跳转到不同端口的服务中nginx 监听端口为 9001，

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

1. **实验代码**

第一步，Linux创建两个文件夹分别准备两个 tomcat，在Tomcat的conf目录vi server.xml一个 8001 端口，一个 8002 端口，且修改其中一个的启动端口号防止两个启动时冲突，且分别在两个tomcat的webapps目录中建立edu、vod文件夹，并准备好测试的页面放入表示的html文件，再启动两个tomcat



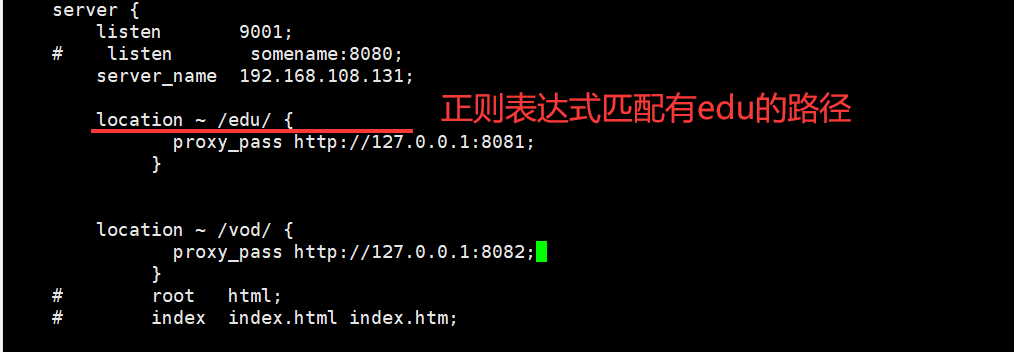


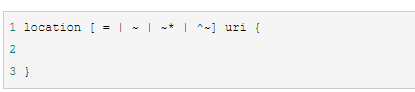
**找到 nginx 配置文件nginx.conf，进行反向代理配置**

修改nginx 的配置文件

在 http 块中添加 server{}







**location 指令说明**

该指令用于匹配 URL。语法如下：

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，不停访问同一个地址会分配到不同的Tomcat服务器**

1.按照第四章实例一样准备两台 tomcat 服务器，一台 8081，一台 8082

2.在两台 tomcat 里面 webapps 目录中同时创建名称是 edu 文件夹，在 edu 文件夹中创建页面 index.html，用于测试

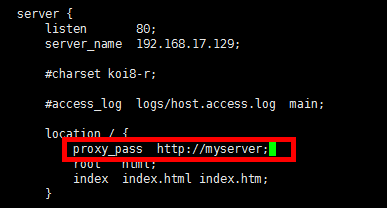
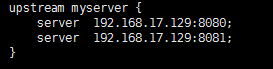
3.在 nginx 的配置文件中进行负载均衡的配置

* 1. **实验代码**

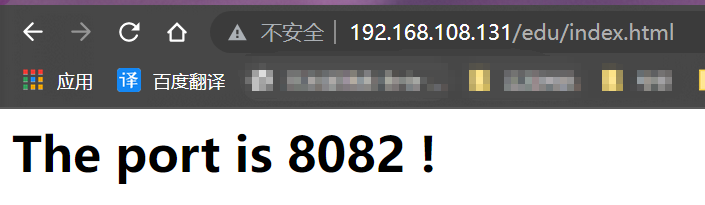
首先准备两个同时启动的Tomcat



**在 nginx.conf 中进行配置**



结果：不停访问同一个地址，会将请求分发到不同的tomcat里



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。

Nginx 提供了几种负载均衡的分配方式(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，即会轮流分配服务器。如果后端服务器 down 掉能自动剔除宕机服务器，如上面的8081宕机了，则只会将请求分配到8082。

**2、weight**

weight 代表权重默认为 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;

# 192.168.5.22 性能不好则可以少分配权重

server 192.168.5.22 weight=4;

}

**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共享 的问题，原理是通过hash的散列特性把访问分配开，缺点是如果宕机，不会自动剔除当前宕机服务器。

看官网，现在最少连接用了新的单词，least—conn。

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。

例如：

upstream server\_pool{

server 192.168.5.21:80;

server 192.168.5.22:80;

fair;

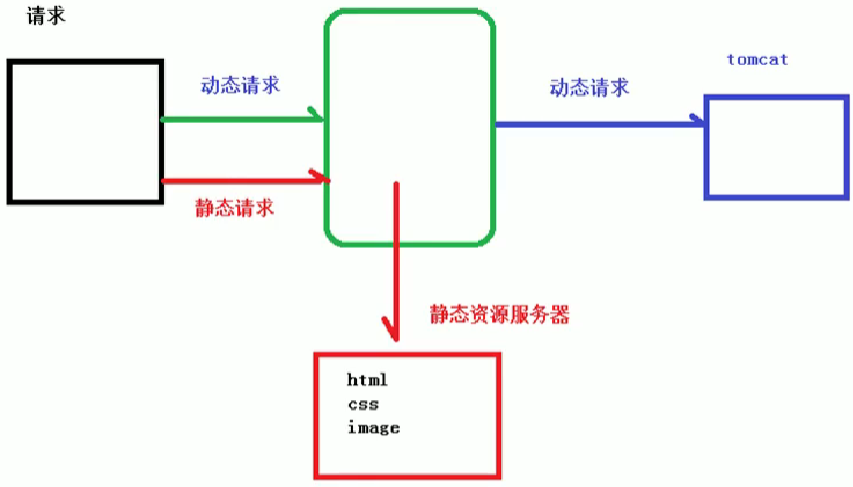
}

**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 处理静态页面如加载某些图片，Tomcat 处理动态页面如请求数据库。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；

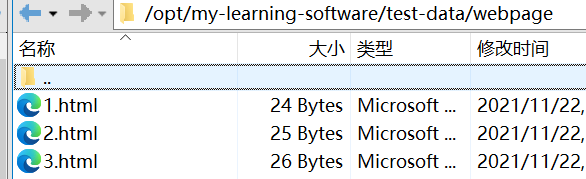
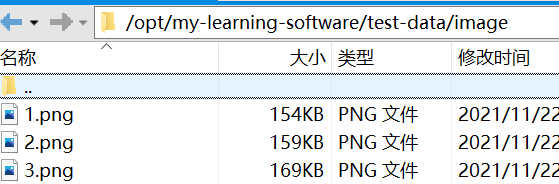
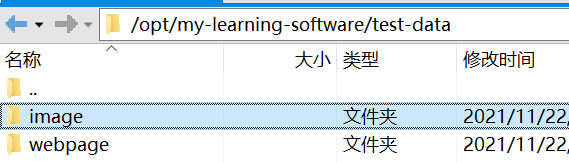
另外一种方法就是动态跟静态文件混合在一起发布，通过 nginx 来分开。



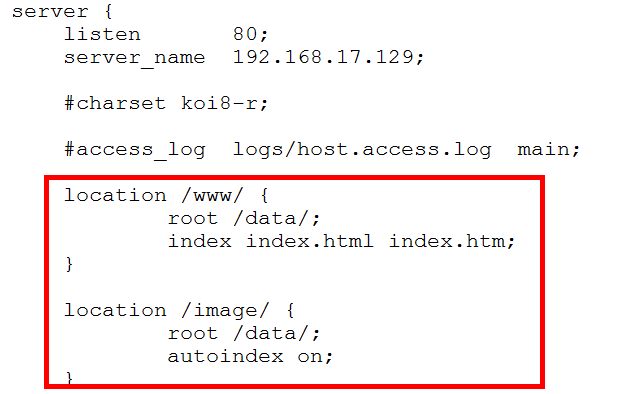
通过 location 指定不同的后缀名实现不同的请求转发。通过 expires 参数设置浏览器缓存过期时间，减少与服务器之前的请求和流量。具体 Expires 定义：是给一个资源设定一个过期时间，也就是说无需去服务端验证，直接通过浏览器自身确认是否过期即可， 所以不会产生额外的流量。此种方法非常适合不经常变动的资源。（如果经常更新的文件， 不建议使用 Expires 来缓存），如我这里设置 3d，表示在这 3 天之内访问这个 URL，发送一个请求会比对服务器该文件最后更新时间，若没有变化则不会从服务器抓取，返回状态码304，如果有修改，则直接从服务器重新下载，返回状态码 200。

**6.1 实验代码**

1.在 liunx 系统中准备静态资源，用于进行访问

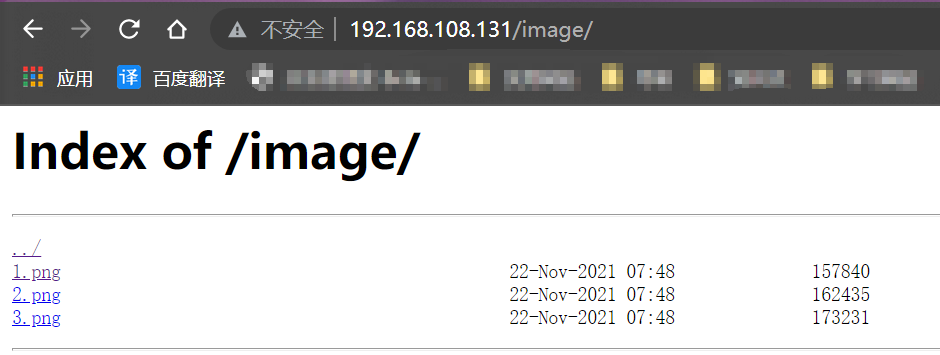


2.进行 nginx 配置

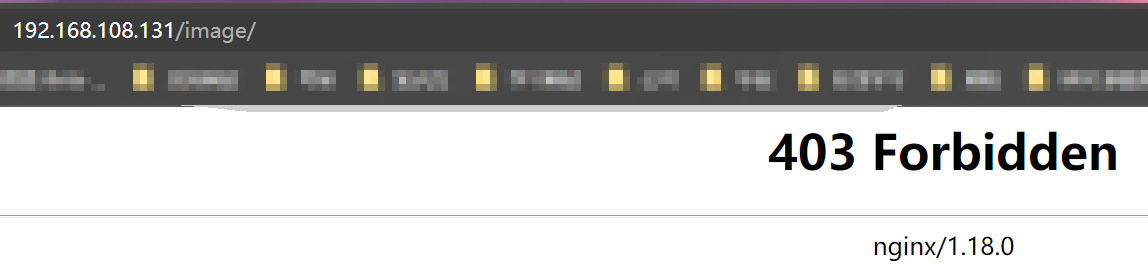


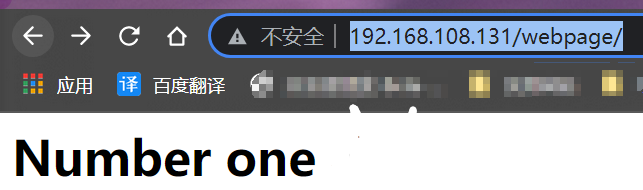


结果:

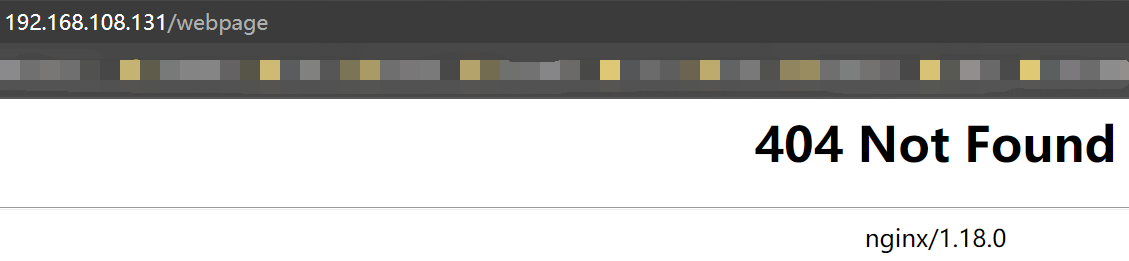
开启了autoindex on后可以列出该文件夹下所以资源，如下图，若不开启则只能访问固定资源

不开启autoindex 如下





不开启默认的index

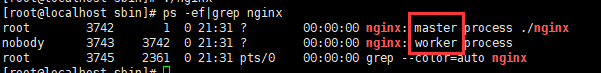
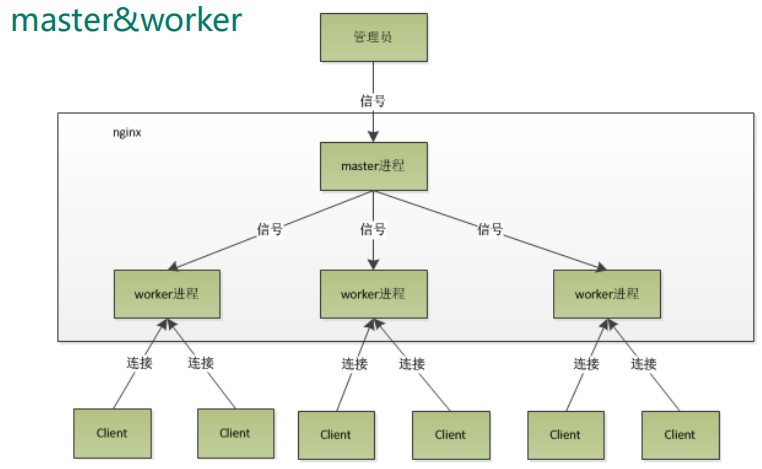


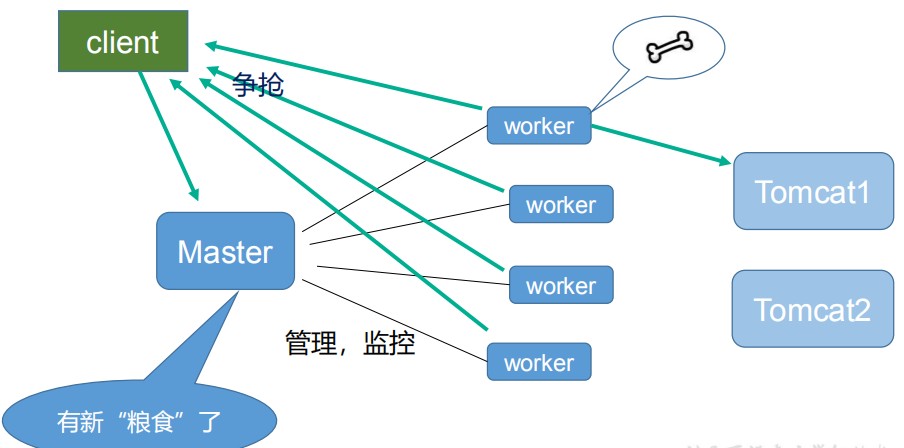
找到 nginx 安装目录，打开/conf/nginx.conf 配置文件，

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-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进程互相之间不会影响，一个进程退出后，其它进程还在工作，服务不会中断，master 进程则可以很快启动新的worker 进程。多个workers利于Nginx进行热部署如./nginx -s reload。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，不过不会影响到其它workder的所有请求，其他 woker由于是 独立的会继续进行争抢，实现请求过程而不会造成服务中断，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，还会占用两个连接。

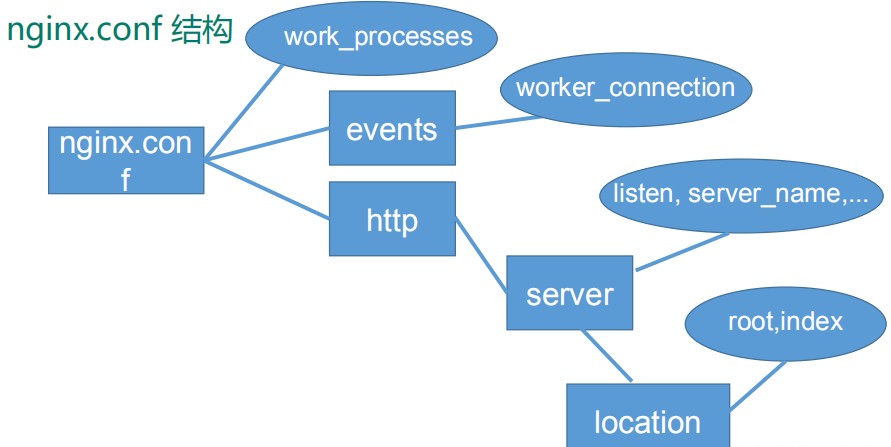
**第一个：发送请求，占用了 woker 的几个连接数？**

**答案：2 或者 4 个。若访问的是静态资源则Worker的连接一个用来接受请求，一个用来返回数据，若是访问动态数据如除了上面两个连接还需要转到数据库或Tomcat等，再用一个连接用来接受返回的数据，所以共4个。**

**第二个：nginx 有一个 master，有四个 woker，每个woker 支持最大的连接数 1024，支持的最大并发数是多少？**

**4个worker共支持的连接数最大为4\*1024**

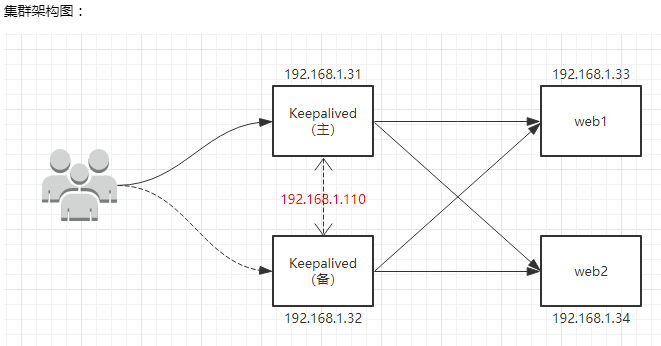
* **普通的静态访问最大并发数是： worker\_connections \* worker\_processes /2，**
* **而如果是 HTTP 作 为反向代理来说，最大并发数量应该是 worker\_connections \* worker\_processes/4。**



**详情见配置文件 nginx.conf**

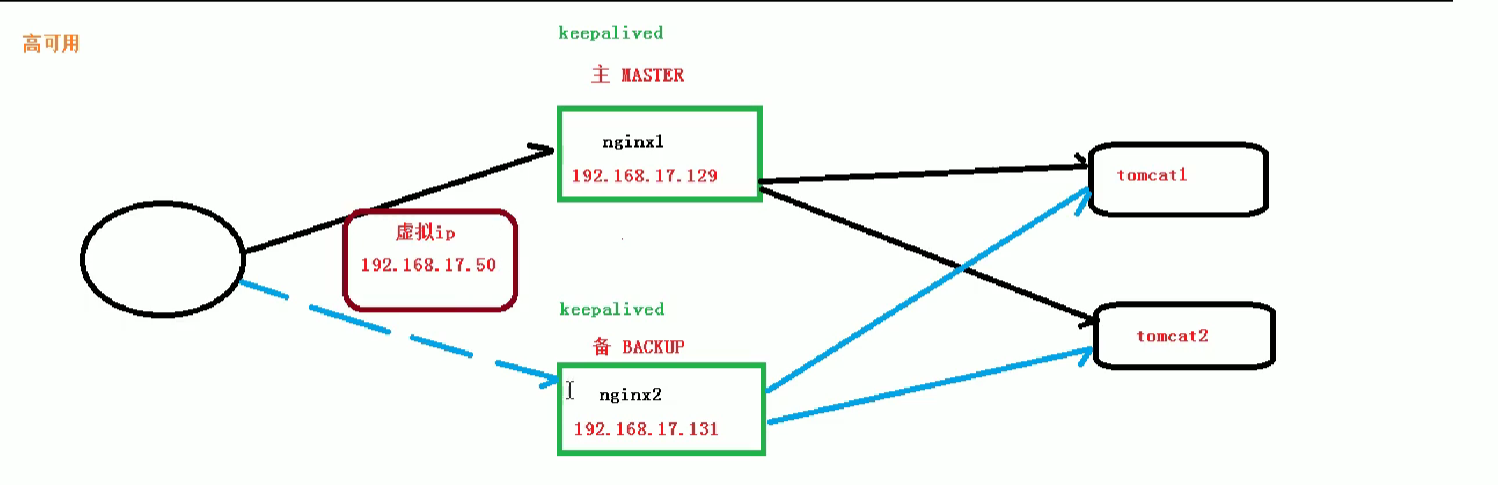
**第 8 章 nginx 搭建高可用集群**

**Keepalived+Nginx 高可用集群（主从模式）**



**1、什么是 nginx 高可用**

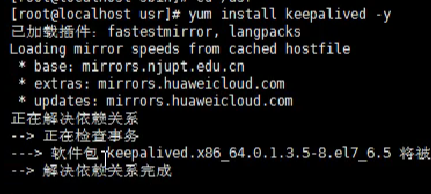
当Nginx宕机后则，用户的请求就不能再分配到后面的服务器了，为了保证较高的可用性，可以准备两台Nginx分别为master、backup，且用keepalived软件绑定一个虚拟ip分配给两台Nginx，keepalived作用类似于一个脚本当检测到作为master的Nginx宕机就会自动启用backup的Nginx，这样就保证了高可用性。



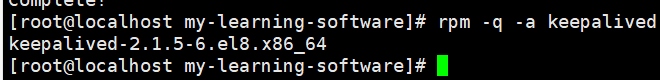
1. **需要两台 nginx 服务器**
2. **需要 keepalived**

先安装keepalived

yum install keepalived -y



查看keepalived命令，rpm -q -a keepalived



**安装之后，在 etc 里面生成目录 keepalived，有配置文件 keepalived.conf**

1. **需要虚拟 ip**

2、配置高可用的准备工作

4、完成高可用配置（主从配置）

keepalived安装后会

（1）修改/etc/keepalived/keepalivec.conf 配置文件(记得先备份)

! Configuration File for keepalived

# 全局定义

global\_defs {

notification\_email {

#指定keepalived在发生切换时需要发送email到的对象，一行一个

acassen@firewall.loc

failover@firewall.loc

sysadmin@firewall.loc

}

#指定发件人

notification\_email\_from Alexandre.Cassen@firewall.loc

#指定发件人

smtp\_server 192.168.17.129

#指定smtp连接超时时间

smtp\_connect\_timeout 30

#运行keepalived机器的一个标识,重要的是下面的值要取唯一值，也可取ip,cent7在/etc/host里

router\_id LVS\_DEVEL

}

# 检测脚本和权重参数配置

vrrp\_script chk\_http\_port {

# 监控脚本目录

script "/opt/my-learning-software/test-data/nginx\_check.sh"

# 检测脚本执行的间隔，时间间隔，2秒

interval 2

# 权重,当前脚本成立则主机权重+2

weight 2

}

# 虚拟IP的相关配置

vrrp\_instance VI\_1 {

# 备份服务器上将 MASTER 改为 BACKUP

state BACKUP

#设置实例绑定的网卡

interface ens33

# 主、备机的 virtual\_router\_id 必须相同，一个标识

virtual\_router\_id 51

# 主、备机取不同的优先级，主机值较大，备份机值较小

priority 100

#MASTER与BACKUP负载均衡器之间同步检查的时间间隔，单位是秒

advert\_int 1

#设置认证

authentication {

#主从服务器验证方式

auth\_type PASS

#密码

auth\_pass 1111

}

#设置vip虚拟IP

virtual\_ipaddress {

# 自定义虚拟IP，注意：该虚拟ip要和这两个虚拟机在同一个网段，即前三个要一致

192.168.108.134

}

}

检查Nginx是否宕机和重启备份Nginx的check\_nginx.sh脚本

#!/bin/bash

A=`ps -C nginx --no-header |wc -l`

if [ $A -eq 0 ];then

/usr/local/nginx/sbin/nginx #启动Nginx的位置

sleep 2

if [ `ps -C nginx --no-header |wc -l` -eq 0 ];then

killall keepalived

fi

fi

或者脚本写为

#将nginx停止，将keepalived启动，执行脚本：sh/etc/keepalived/check\_nginx.sh

#从执行可以看出自动将keepalived进程kill掉了。

#!/bin/bash

# 如果进程中没有nginx则将keepalived进程kill掉

A=`ps -C nginx--no-header |wc -l` ## 查看是否有 nginx进程 把值赋给变量A

if [ $A -eq 0];then ## 如果没有进程值得为零

service keepalived stop ## 则结束 keepalived 进程

fi

两台虚拟机分别启动Nginx和keepalived

**启动Nginx**

**./nginx**

**停止Nginx**

**./nginx -s stop**

**启动 keepalived：**

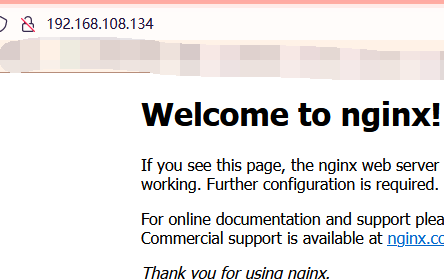
**systemctl start keepalived.service**

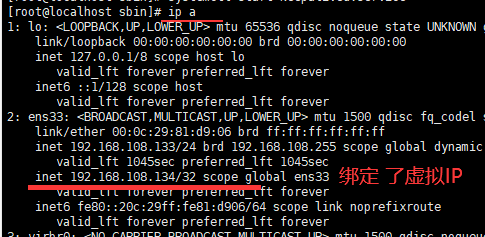
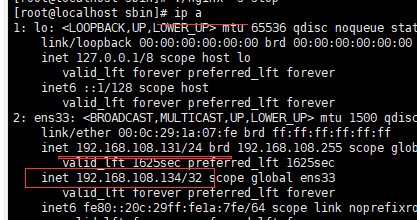
**停止 keepalived:**

**systemctl stop keepalived.service**

结果:

直接访问虚拟IP可以进入Nginx默认欢迎页面，关闭Master的Nginx服务和keepalived发现仍可以访问，是因为转到了BACKUP





# 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

**（1）在所有节点上面进行配置**

1. **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

面中

# vim /etc/nginx/nginx.conf

//准备测试文件，此处是将主机名和 ip 写到 index.html 页

//编辑配置文件

1. **配置 LB 服务器（两台都一样）**



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

# 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

1. **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](http://www.mtian.org/)

192.168.1.31 [www.mtian.org](http://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](http://www.mtian.org/)

web01 192.168.1.33

[root@node01 ~]# curl [www.mtian.org](http://www.mtian.org/) web02 192.168.1.34

[root@node01 ~]# curl [www.mtian.org](http://www.mtian.org/) web01 192.168.1.33

[root@node01 ~]# curl [www.mtian.org](http://www.mtian.org/) web02 192.168.1.34

[root@node01 ~]# curl [www.mtian.org](http://www.mtian.org/) web01 192.168.1.33

[root@node01 ~]# curl [www.mtian.org](http://www.mtian.org/)

web02 192.168.1.34

1. **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

1. **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](mailto: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

......

1. **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](mailto: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

......

1. **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 备节点上面的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

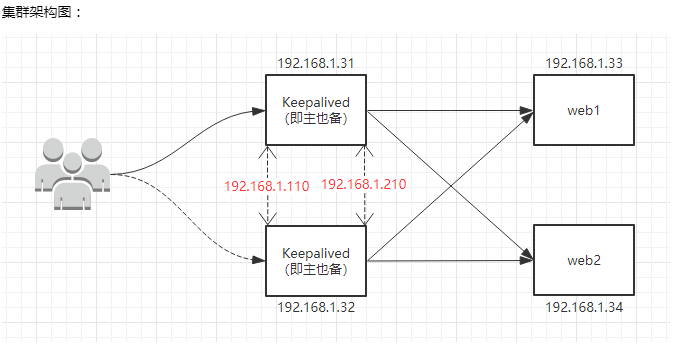
ens33:1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

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。**

**Keepalived+Nginx 高可用集群（双主模式）**



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）

**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](mailto: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

1. **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](mailto: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

**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。