# CentOS7.2 – Nginx 搭建服务

## Nginx在线安装与配置:

### 下载Nginx服务:

http://nginx.org

nginx-1.12.2.tar.gz

### 下载pcre依赖:

https://sourceforge.net/projects/pcre/files/pcre/

pcre-8.37.tar.gz

### 安装pcre依赖:

解压压缩文件执行: tar –xvf pcre-8.37.tar.gz

进入解压目录执行: ./configure

进入解压目录执行: make && make install

### 安装 openssl 、zlib 、 gcc 依赖:

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

注: 直接安装 yum -y install gcc zlib zlib-devel pcre-devel openssl openssl-devel

### 安装Nginx服务:

解压压缩文件执行: tar –xvf nginx-1.12.2.tar.gz

进入解压目录执行: ./configure

进入解压目录执行: make && make install

### Nginx安装位置: /usr/local/nginx/sbin/nginx

cd /usr/local/nginx/sbin/

./nginx

查看nginx版本: ./nginx -v

启动nginx服务: ./nginx

停止nginx服务: ./nginx -s stop

重新加载nginx服务: ./nginx -s reload

### 验证是否开启: <http://127.0.0.1> (默认80端口)

### Nginx配置文件: /usr/local/nginx/conf/nginx.conf

## Nginx配置文件详解:

**配置文件nginx.conf包含三部分内容:**

### (1)全局块: 影响 nginx 服务器整体运行的配置指令

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

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

如 worker\_processes 1; 数值越大,可以支持的并发处理数量也越多

### (2)events块: 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 wordprocess 可以同时支持的最大连接数等;下列例子就表示每个 work process 支持的最大连接数为 1024

如 worker\_connections 1024; 支持的最大连接数为 1024

### (3)http块: 包含http全局块与server块

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

#### (3.1) http全局快:

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

#### (3.2) server块:

server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

每个 server块 也分为全局 server块,以及可以同时包含多个 location块

##### (3.2.1) 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

##### (3.2.2) 多个location块:

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

location 指令说明: 该指令用于匹配 URL

语法: location [= | ~ | ~\* | ^~] uri {

}

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

## Nginx请求分配策略:

### 第一种轮询(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

upstream server\_pool{

server 192.168.5.21;

server 192.168.5.22;

}

### 第二种weight:

weight代表权重,默认为1,权重越高被分配的客户端越多

upstream server\_pool{

server 192.168.5.21 weight=10;

server 192.168.5.22 weight=10;

}

### 第三种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 server\_pool{

ip\_hash;

server 192.168.5.21:80;

server 192.168.5.22:80;

}

### 第四种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;

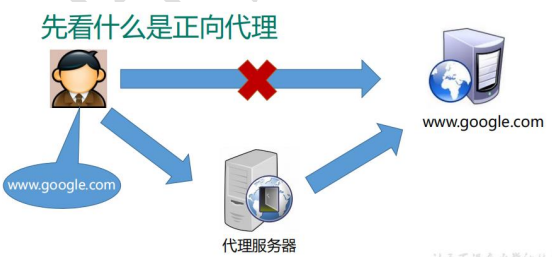
server 192.168.5.22:80;

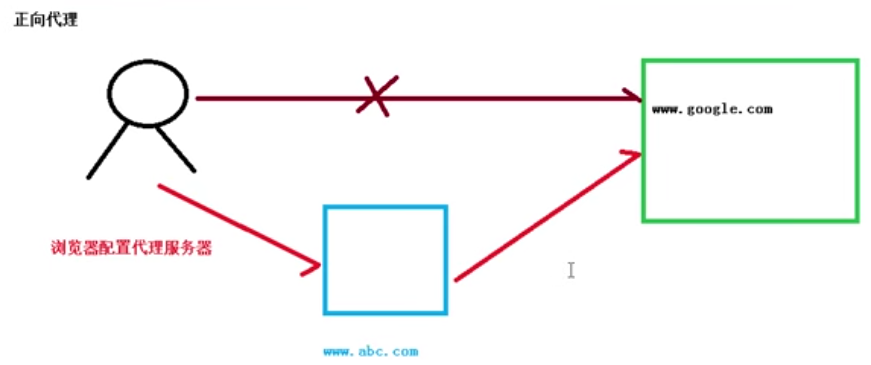
fair;

}

## Nginx – 正向代理:

在客户端(浏览器)配置代理服务器,通过代理服务器进行互联网访问

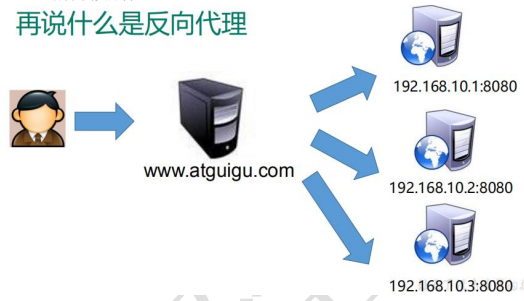


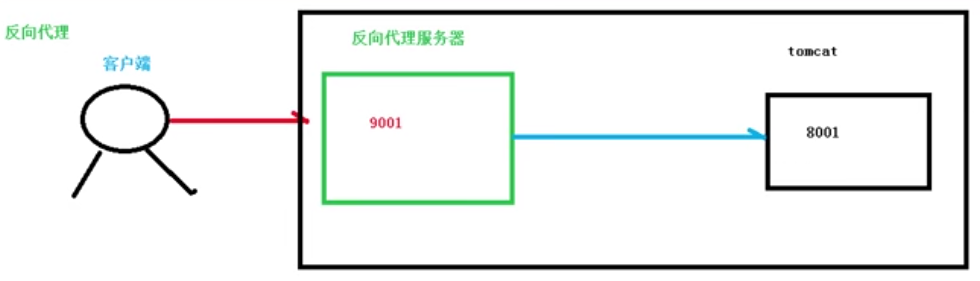


## Nginx – 反向代理:

客户端(浏览器)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(浏览器)

此时反向代理服务器和目标服务器对外就一个服务器,暴露的是代理服务器地址,隐藏了真实的服务器IP地址





### 反向代理示例:

**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://192.168.208.129:9001/edu/ 直接跳转到 127.0.0.1:8081

访问 http://192.168.208.129:9001/vod/ 直接跳转到 127.0.0.1:8082

**实现代码:**

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

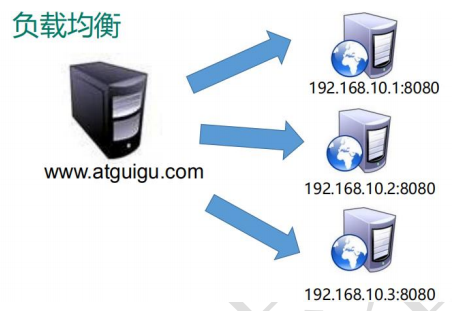
第二步: 修改Nginx的配置文件nginx.conf,在 http块 中添加 server{}块,内容如下

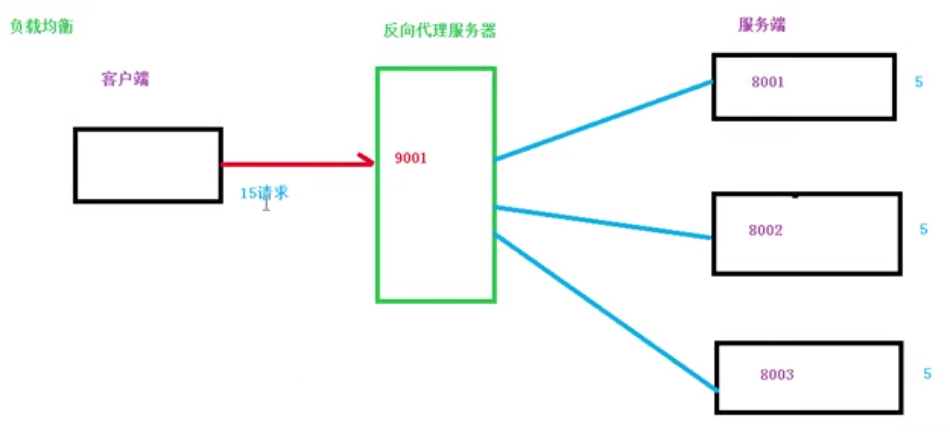
|  |
| --- |
| server {  listen 9001;  server\_name 192.168.208.129; # IP地址,如 127.0.0.1,localhost  location ~ /edu/ {  proxy\_pass http://127.0.0.1:8001;  }  location ~ /vod/ {  proxy\_pass http://127.0.0.1:8002;  }  } |

## Nginx – 负载均衡:

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





### 负载均衡示例:

**实现效果:**

浏览器地址栏输入地址 http://192.168.208.129/edu/a.html,负载均衡效果,平均 8080 和 8081 端口中

**实现代码:**

1. 准备两台tomcat服务器,一台8001,一台8002,分别在两台tomcat中的webapps目录中创建edu文件夹并创建a.html文件

2. 修改Nginx的配置文件nginx.conf进行配置: http块进行负载均衡的配置

|  |
| --- |
| http {  #...  upstream myserver{  ip\_hash;  server 192.168.209.129:8080;  server 192.168.209.129:8081;  }  server {  #...  location / {  #...  proxy\_pass http://myserver;  proxy\_connect\_timeout 10;  }  }  } |

### 请求分配策略:

第一种轮询(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

upstream server\_pool{

server 192.168.5.21;

server 192.168.5.22;

}

第二种weight: weight代表权重,默认为1,权重越高被分配的客户端越多

upstream server\_pool{

server 192.168.5.21 weight=10;

server 192.168.5.22 weight=10;

}

第三种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

upstream server\_pool{

ip\_hash;

server 192.168.5.21:80;

server 192.168.5.22:80;

}

第四种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;

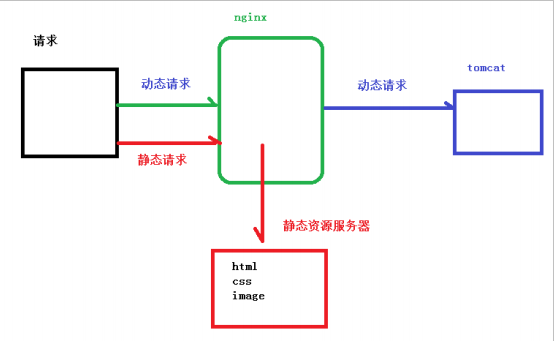
fair;

}

## Nginx – 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**

**一种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。

### 动静分离示例:

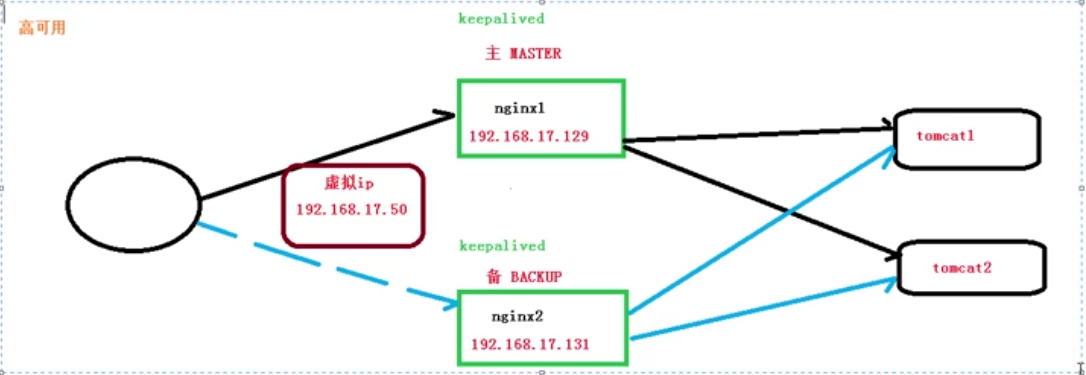
**实现代码:**

修改Nginx的配置文件nginx.conf进行配置

|  |
| --- |
| **资源目录:**  - data:  - image:  - 1.jpg  - www:  - a.html  **nginx.conf配置:**  http {  # ...  server {  listen 9001;  server\_name 192.168.208.129;  location /www/ {  root /data/;  index index.html index.htm;  }  location /image/ {  root /data/;  autoindex on;  }  }  } |

## Nginx – 高可用集群:

Keepalived+Nginx 高可用集群(主从模式)



### 主从模式示例:

**配置准备工作:**

1. 准备两台服务器(192.168.129.201与192.168.129.202),每台服务器都安装nginx和keepalived服务

2. 安装keepalived服务:

安装命令: yum install keepalived –y

安装位置: /etc/keepalived/

配置文件: /etc/keepalived/keepalivec.conf

3. 完成高可用配置(主从配置):

3.1 修改keepalived服务配置文件: vi /etc/keepalived/keepalivec.conf 两台服务器

|  |
| --- |
| 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 # 访问到主机**  }  vrrp\_script chk\_http\_port {  **script "/usr/local/src/nginx\_check.sh" # 脚本位置路径**  interval 2 # 检测脚本执行的间隔,单位秒  weight 2 # 权重  }  vrrp\_instance VI\_1 {  **state MASTER # 备份服务器上将 MASTER 改为 BACKUP**  **interface ens33 # 网卡名称**  **virtual\_router\_id 51 # 主、备机的 virtual\_router\_id 必须相同**  **priority 90 # 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**  }  } |

**3.2 在/usr/local/src 添加检测脚本:** 两台服务器

|  |
| --- |
| #!/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 |

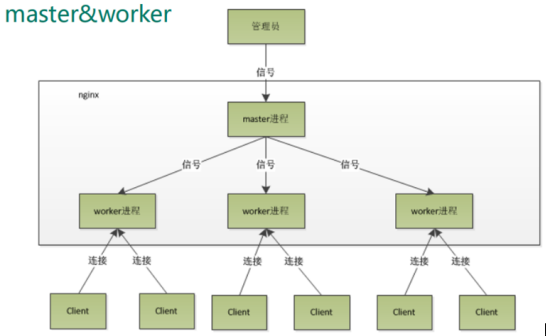
3.3 启动两台服务器上的 nginx 和 keepalived 服务

启动 nginx：./nginx

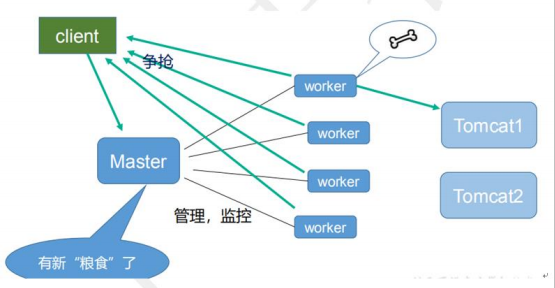
启动 keepalived：systemctl start keepalived.service

4. 主从测试应用: 略

## Nginx – worker机制:



### worker是怎样进行工作的:



### 一个 master 和多个 woker 有好处:

（1）可以使用 nginx –s reload 热部署,利用 nginx 进行热部署操作

（2）每个woker是独立进程,如果有其中一个 woker 出现问题,其他 woker 独立的,继续进行争抢,实现请求过程,不会造成服务中断

### 设置多少个worker合适:

worker 数和服务器的 cpu 数相等是最为适宜的

### 连接数: worker\_connection

第一个：发送请求，占用了 woker 的几个连接数？

答案：2 或者 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?

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