**Nginx服务详细部署及部署架构**

目录

[一、Nginx服务详细部署 2](#_Toc2151475)

[CentOS 6.6系统部署方案： 2](#_Toc2151476)

[二、Nginx双机主备架构 18](#_Toc2151477)

[双机主备原理 19](#_Toc2151478)

[高可用高可用负载均衡(双主模式) 21](#_Toc2151479)

[VIP概念普及 22](#_Toc2151480)

# 一、Nginx服务详细部署

### CentOS 6.6系统部署方案：

#### Nginx部署

|  |
| --- |
| <https://www.cnblogs.com/taiyonghai/p/6728707.html> |
| 注意：Windows的txt编码方式与Linux不一样，Linux的运行脚本要直接在Linux系统下编写。  wget http://nginx.org/download/nginx-1.10.2.tar.gz  wget http://www.openssl.org/source/openssl-fips-2.0.16.tar.gz  wget http://zlib.net/zlib-1.2.11.tar.gz  wget ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/pcre-8.40.tar.gz  #!/bin/bash  cd /usr/local/src  yum install gcc-c++  tar zxvf openssl-fips-2.0.16.tar.gz  cd openssl-fips-2.0.16  ./config && make && make install  cd ..  tar zxvf pcre-8.40.tar.gz  cd pcre-8.40  ./configure && make && make install  cd ..  tar zxvf zlib-1.2.11.tar.gz  cd zlib-1.2.11  ./configure && make && make install  cd ..  tar zxvf nginx-1.10.2.tar.gz  cd nginx-1.10.2  ./configure && make && make install  cd ..  cd /usr/local/nginx  ln -s /usr/local/lib/libpcre.so.1 /lib64  cd ~  /usr/local/nginx/sbin/nginx |
| /usr/local/nginx/sbin/nginx //启动nginx  /usr/local/nginx/sbin/nginx -s stop(quit、reload) // 停止/重启nginx  /usr/local/nginx/sbin/nginx –h //命令帮助  /usr/local/nginx/sbin/nginx –t //验证配置文件  vim /usr/local/nginx/conf/nginx.conf //配置文件 |
| 当运行nginx报错: error while loading shared libraries: libpcre.so.1: cannot open shared object file: No such file or directory  解决办法：  whereis libpcre.so.1 //查找libpcre.so.1的位置  ln -s /usr/local/lib/libpcre.so.1 /lib64 //做软连接 |
| service iptables stop //关闭防火墙  chkconfig iptables off //关闭开机自启动防火墙  chkconfig --list|grep ipt //查看 |
|  |
| echo "/usr/local/nginx/sbin/nginx" >> /etc/rc.local //将nginx加入开机自启 |

#### nginx 负载均衡

|  |
| --- |
| 基于ip\_hash方式，注意ip\_hash使用的是源IP的前24位地址  vim /usr/local/nginx/conf/nginx.conf //打开nginx配置文件，添加这些代码    注意：假如server的监听端口修改为81，则浏览器输入时，后面要接端口81才能访问 |

nginx的负载均衡有4种模式：

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

      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。

2)、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。

3)、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。

4)、url\_hash（第三方）

通常大家比较懒，用前三种的多。

配置方法：

打开nginx.cnf文件

在http节点下添加upstream节点：

upstream webname {

server 192.168.0.1:8080;

server 192.168.0.2:8080;

}

其中webname是自己取的名字，最后会通过这个名字在url里访问的，像上面这个例子一样什么都不加就是默认的轮询，第一个请求过来访问第一个server，第二个请求来访问第二个server。依次轮着来。

upstream webname {

server 192.168.0.1:8080 weight 2;

server 192.168.0.2:8080 weight 1;

}

这个weight也很好理解，权重大的被访问的概率就大，上面这个例子的话，访问2次server1，访问一次server2

upstream webname {

ip\_hash;

server 192.168.0.1:8080;

server 192.168.0.2:8080;

}

ip\_hash的配置也很简单，直接加一行就可以了，这样只要是同一个ip过来的都会到同一台server上

然后在server节点下进行配置：

location /name {

proxy\_pass http://webname/name/;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection "upgrade";

}

proxy\_pass里面用上面配的webname代替了原来的ip地址。

这样就基本完成了负载均衡的配置。

下面是主备的配置：

还是在upstream里面

upstream webname {

server 192.168.0.1:8080;

server 192.168.0.2:8080 backup;

}

设置某一个节点为backup，那么一般情况下所有请求都访问server1，当server1挂掉或者忙的的时候才会访问server2

upstream webname {

server 192.168.0.1:8080;

server 192.168.0.2:8080 down;

}

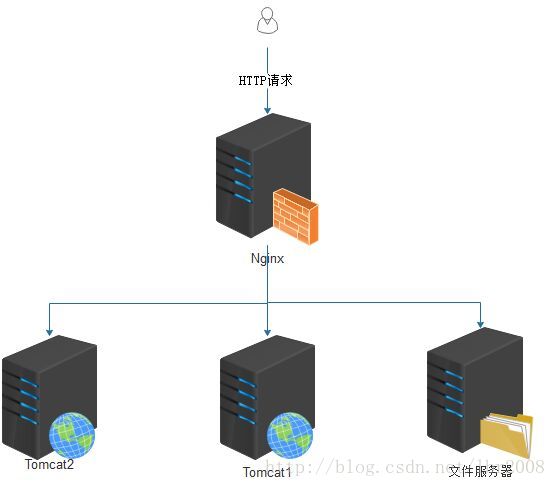
设置某个节点为down，那么这个server不参与负载。

#### nginx 主备

|  |
| --- |
| **Keepalived实现nginx主备：**  环境：openssl-devel、popt-devel  yum –y install openssl-devel  yum –y install popt-devel  安装Keepalived  cd /user/local/src  wget <http://www.keepalived.org/software/keepalived-1.2.2.tar.gz>  tar -zxvf keepalived-1.2.2.tar.gz  cd keepalived-1.2.2  ./configure  --prefix=/usr/local/keepalived  make  make install  <---拷贝keepalived相关启动命令--->  cp /usr/local/sbin/keepalived /usr/sbin/  cp /usr/local/etc/sysconfig/keepalived /etc/sysconfig/  <---将keepalived启动脚本添加到系统服务--->  cp /usr/local /etc/rc.d/init.d/keepalived /etc/ init.d/  chkconfig --add keepalived  chkconfig --level 2345 keepalived on  <---创建keepalived相关配置文件--->  mkdir -p /etc/keepalived  cp /usr/local/etc/keepalived/keepalived.conf /etc/keepalived/  service keepalived start //启动服务  ps aux | grep keepalived //查看服务启动情况  echo "/etc/init.d/keepalived start" >> /etc/rc.local //将之加入开机自启 |
| **Keepalived配置（nginx-master）主:**  vim /etc/keepalived/keepalived.conf |
| 使用ip addr查看：  C:\Users\jianfan\Desktop\无标题.png  C:\Users\jianfan\Desktop\无标题.png |
| **Keepalived配置（nginx-backup）备:**  vim /etc/keepalived/keepalived.conf |
| 使用ip addr查看：  当主存在时：  **C:\Users\jianfan\Desktop\无标题.png**  当主挂掉后：  **C:\Users\jianfan\Desktop\无标题.png**  当主有恢复后：  **C:\Users\jianfan\Desktop\无标题.png** |
| **chk-nginx脚本:**  该脚本检测ngnix的运行状态，并在nginx进程不存在时尝试重新启动ngnix，如果启动失败则停止keepalived，准备让其它机器接管。  vim /opt/chk\_nginx.sh |
|  |
| **Keepalived配置解释：**  ! Configuration File for keepalived     #全局定义  global\_defs {  notification\_email {     #指定keepalived在发生事件时(比如切换)发送通知邮件的邮箱  ops@wangshibo.cn   #设置报警邮件地址，可以设置多个，每行一个。 需开启本机的sendmail服务  tech@wangshibo.cn  }    notification\_email\_from ops@wangshibo.cn   #keepalived在发生诸如切换操作时需要发送email通知地址  smtp\_server 127.0.0.1      #指定发送email的smtp服务器  smtp\_connect\_timeout 30    #设置连接smtp server的超时时间  router\_id master-node     #运行keepalived的机器的一个标识，通常可设为hostname。故障发生时，发邮件时显示在邮件主题中的信息。  }    vrrp\_script chk\_http\_port {      #检测nginx服务是否在运行。有很多方式，比如进程，用脚本检测等等      script "/opt/chk\_nginx.sh"   #这里通过脚本监测      interval 2                   #脚本执行间隔，每2s检测一次      weight -5                    #脚本结果导致的优先级变更，检测失败（脚本返回非0）则优先级 -5      fall 2                    #检测连续2次失败才算确定是真失败。会用weight减少优先级（1-255之间）      rise 1                    #检测1次成功就算成功。但不修改优先级  }    vrrp\_instance VI\_1 {    #keepalived在同一virtual\_router\_id中priority（0-255）最大的会成为master，也就是接管VIP，当priority最大的主机发生故障后次priority将会接管      state MASTER    #指定keepalived的角色，MASTER表示此主机是主服务器，BACKUP表示此主机是备用服务器。注意这里的state指定instance(Initial)的初始状态，就是说在配置好后，这台服务器的初始状态就是这里指定的，但这里指定的不算，还是得要通过竞选通过优先级来确定。如果这里设置为MASTER，但如若他的优先级不及另外一台，那么这台在发送通告时，会发送自己的优先级，另外一台发现优先级不如自己的高，那么他会就回抢占为MASTER      interface em1          #指定HA监测网络的接口。实例绑定的网卡，因为在配置虚拟IP的时候必须是在已有的网卡上添加的      mcast\_src\_ip 103.110.98.14  # 发送多播数据包时的源IP地址，这里注意了，这里实际上就是在哪个地址上发送VRRP通告，这个非常重要，一定要选择稳定的网卡端口来发送，这里相当于heartbeat的心跳端口，如果没有设置那么就用默认的绑定的网卡的IP，也就是interface指定的IP地址      virtual\_router\_id 51         #虚拟路由标识，这个标识是一个数字，同一个vrrp实例使用唯一的标识。即同一vrrp\_instance下，MASTER和BACKUP必须是一致的      priority 101                 #定义优先级，数字越大，优先级越高，在同一个vrrp\_instance下，MASTER的优先级必须大于BACKUP的优先级      advert\_int 1                 #设定MASTER与BACKUP负载均衡器之间同步检查的时间间隔，单位是秒      authentication {             #设置验证类型和密码。主从必须一样          auth\_type PASS           #设置vrrp验证类型，主要有PASS和AH两种          auth\_pass 1111           #设置vrrp验证密码，在同一个vrrp\_instance下，MASTER与BACKUP必须使用相同的密码才能正常通信      }      virtual\_ipaddress {          #VRRP HA 虚拟地址 如果有多个VIP，继续换行填写          103.110.98.20      }    track\_script {                      #执行监控的服务。注意这个设置不能紧挨着写在vrrp\_script配置块的后面（实验中碰过的坑），否则nginx监控失效！！     chk\_http\_port                    #引用VRRP脚本，即在 vrrp\_script 部分指定的名字。定期运行它们来改变优先级，并最终引发主备切换。  }  } |

# 二、Nginx双机主备架构

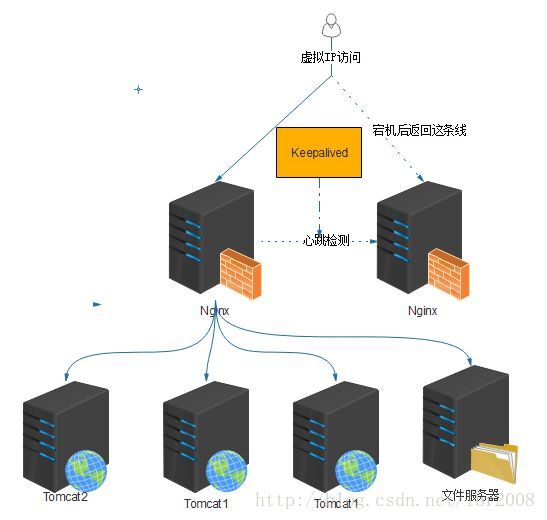
最基础的Nginx架构，如下：



如果说Nginx服务器出现宕机，那整个网站都会处于瘫痪状态，所以我们想到是否可以使用多台Nginx实现主备切换呢？如何实现？使用Keepalive实现[心跳检测](https://www.baidu.com/s?wd=%E5%BF%83%E8%B7%B3%E6%A3%80%E6%B5%8B&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)。

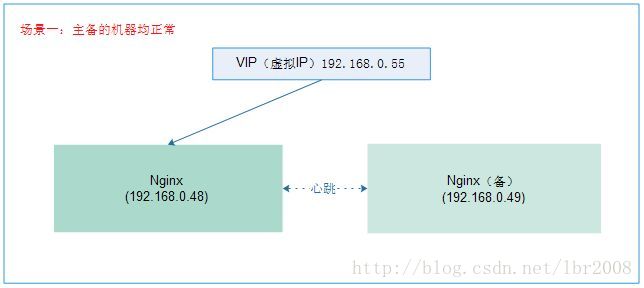
keepalived是VRRP协议的完美实现，主要是用来避免单点故障实现心跳检测。

主备模式整体架构图：

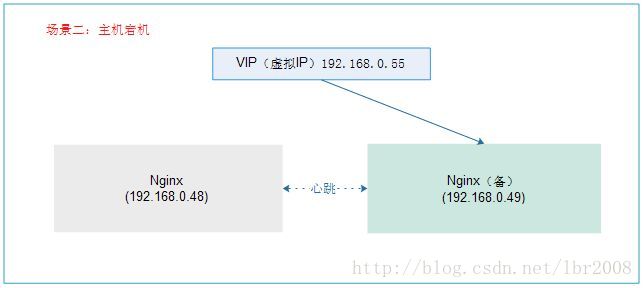


### 双机主备原理

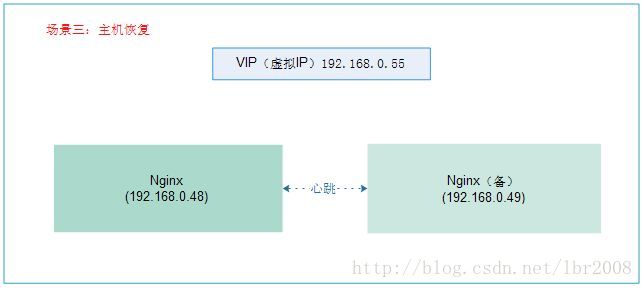
场景1



场景2



场景3



好处：

通过Keepalived解决Nginx单点故障的问题，实现主从切换

问题：

1、备机由于如果在未发生故障的情况下，一直处于闲置状态，造成资源浪费   
2、主机在高负载的情况下运行，有一定的上限。   
3、如果主备机都出现宕机。如何处理？

### 高可用高可用负载均衡(双主模式)

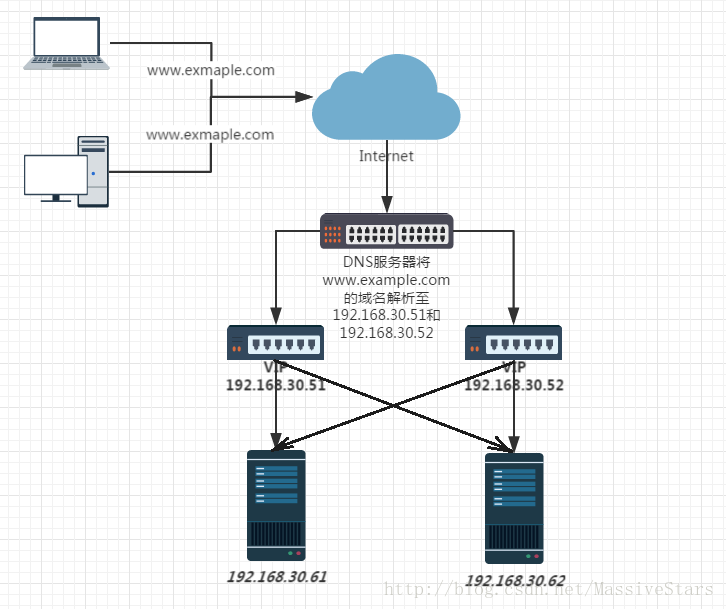
高可用分为以下两种方式：

1、Nginx+keepalived 主从配置

这种方案，使用一个vip地址，前端使用2台机器，一台做主，一台做备，但同时只有一台机器工作，另一台备份机器在主机器不出现故障的时候，永远处于浪费状态，对于服务器不多的网站，该方案不经济实惠。

2、Nginx+keepalived 双主配置

这种方案，使用两个vip地址，前端使用2台机器，互为主备，同时有两台机器工作，当其中一台机器出现故障，两台机器的请求转移到一台机器负担，非常适合于当前架构环境。架构图如下：



### VIP概念普及

VIP是指虚拟IP，何为虚拟IP，就是一个未分配给真实主机的IP，也就是说对外提供数据库服务器的主机除了有一个真实IP外还有一个虚IP，使用这两个IP中的 任意一个都可以连接到这台主机，所有项目[中数据](https://www.baidu.com/s?wd=%E4%B8%AD%E6%95%B0%E6%8D%AE&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)库链接一项配置的都是这个虚IP，当服务器发生故障无法对外提供服务时，动态将这个虚IP切换到备用主机。

如何实现：

开始我也不明白这是怎么实现的，以为是软件动态改IP地址，其实不是这样，其实现原理主要是靠TCP/IP的ARP协议。因为ip地址只是一个逻辑 地址，在以太网中MAC地址才是真正用来进行数据传输的物理地址，每台主机中都有一个ARP高速缓存，存储同一个网络内的IP地址与MAC地址的对应关 系，以太网中的主机发送数据时会先从这个缓存中查询目标IP对应的MAC地址，会向这个MAC地址发送数据。操作系统会自动维护这个缓存。这就是整个实现 的关键。

下边就是我电脑上的arp缓存的内容。

(192.168.1.219) at 00:21:5A:DB:68:[E8](https://www.baidu.com/s?wd=E8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd) [ether] on bond0  
(192.168.1.217) at 00:21:5A:DB:68:E8 [ether] on bond0  
(192.168.1.218) at 00:21:5A:DB:7F:C2 [ether] on bond0

192.168.1.217、192.168.1.218是两台真实的电脑，

192.168.1.217为对外提供数据库服务的主机。

192.168.1.218为热备的机器。

192.168.1.219为虚IP。

大家注意红字部分，219、217的MAC地址是相同的。

再看看那217宕机后的arp缓存

(192.168.1.219) at 00:21:5A:DB:7F:C2 [ether] on bond0  
(192.168.1.217) at 00:21:5A:DB:68:E8 [ether] on bond0  
(192.168.1.218) at 00:21:5A:DB:7F:C2 [ether] on bond0

这就是奥妙所在。当218 发现217宕机后会向网络发送一个ARP数据包，告诉所有主机192.168.1.219这个IP对应的MAC地址是00:21:5A:DB:7F:C2，这样所有发送到219的数据包都会发送到mac地址为00:21:5A:DB:7F:C2的机器，也就是218的机器。