# 1、nginx的安装与集群搭建

## 1.1 nginx的安装

安装环境准备：

1、linux内核2.6及以上版本

只有2.6之后才支持epool，在此之前使用select或pool多路复用的IO模型，无法解决高并发压力的问题。通过命令uname –a 即可查看linux内核

2、GCC编译期

GCC（GNU Compiler Collection）可用来编译C语言程序。Nginx不会直接提供二进制可执行程序,只能下载源码进行编译。

3、PCRE库

PCRE（Perl Compatible Regular Expressions，Perl兼容正则表达式）是由Philip Hazel开发的函数库，目前为很多软件所使用，该库支持正则表达式。

4、zlib库

zlib库用于对HTTP包的内容做gzip格式的压缩，如果我们在nginx.conf里配置了gzip on，并指定对于某些类型（content-type）的HTTP响应使用gzip来进行压缩以减少网络传输量

5、OpenSSL开发库

如果我们的服务器不只是要支持HTTP，还需要在更安全的SSL协议上传输HTTP，那么就需要拥有OpenSSL了。另外，如果我们想使用MD5、SHA1等散列函数，那么也需要安装它。

上面几个库都是Nginx 基础功能所必需的，为简单起见我们可以通过yum 命令统一安装。

#yum 安装nginx 环境

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

**获取nginx源码：**

nginx 下载页：http://nginx.org/en/download.html 。

# 下载nginx 最新稳定版本

wget http://nginx.org/download/nginx-1.14.0.tar.gz

#解压

cd /usr/local/nginx

tar -zxvf nginx-1.14.0.tar.gz

最简单的安装：

# 全部采用默认安装

./configure & make & make install

make & make install

执行完成之后 nginx 运行文件 就会被安装在 /usr/local/nginx 下。

**模块更新：**

# 添加状态查查看模块

./configure --with-http\_stub\_status\_module

# 重新创建主文件

make

# 将新生成的nginx 文件覆盖 旧文件。

cp objs/nginx /usr/local/nginx/sbin/

# 查看是否更新成功 显示了 configure 构建参数表示成功

/usr/local/nginx/sbin/nginx –V

**控制命令：**

#查看命令帮助

./sbin/nginx -?

#默认方式启动：

./sbin/nginx

#指定配置文件启动

./sbin/nginx -c /tmp/nginx.conf

#指定nginx程序目录启动

./sbin/nginx -p /usr/local/nginx/

#快速停止

./sbin/nginx -s stop

#优雅停止

./sbin/nginx -s quit

# 热装载配置文件

./sbin/nginx -s reload

# 重新打开日志文件

./sbin/nginx -s reopen

# 设置全局命令，如下表示设置启动用户为root

./sbin/nginx -g "user root;"

## 1.2 nginx的集群搭建

Nginx集群搭建的方式：

1、每个节点上安装一台nginx与一台Keepalived，即一对一监控

2、nginx与Keepalived不存放于同一节点上，分开部署，每一台Keepalived均监控所有的nginx

### 1.2.1 基于一对一监控的nginx集群

访问虚拟ip时，只访问master Keepalived，只转发到master机器上的nginx，无法对nginx负载

Nginx 默认端口号80，与http协议端口号相同

nginx/sbin/nginx 根目录，直接从根目录启动nginx，ps aux|grep nginx命令查看nginx是否启动，tail –f 动态的查看文件内容，vim编辑文件

keepalived，nginx监听软件，每个nginx服务器需要单独配一个keepalived，nginx的主从由keepalived进行管理，在其config文件中配置state属性，主节点为master，从节点为backup

keepalived中配置一个虚拟ip，集群内所有nginx都使用相同的虚拟ip，访问时由虚拟ip转移到对应的master节点，如果主节点挂掉可以直接使用从节点而不用手动改写ip地址

keepalived的心跳配置，通过脚本检测nginx的进程个数，如果进程数为零则nginx宕机，会自动启动nginx，然后重新检测nginx的进程数，如果启动失败，则关闭keepalived

1、安装Keepalived

进入放置压缩包目录，如我的为 cd /alidata/resource

wget http://www.keepalived.org/software/keepalived-1.2.22.tar.gz 开始下载

创建目录： mkdir -p /alidata/server/keepalived

tar -zxvf keepalived-1.2.22.tar.gz

cd keepalived-1.2.22

./configure –prefix=/alidata/server/keepalived

make && make install

cp keepalived/etc/rc.d/init.d/keepalived /etc/init.d/

cp /alidata/server/keepalived/etc/sysconfig/keepalived /etc/sysconfig/

ln -s /alidata/server/keepalived/sbin/keepalived /usr/sbin/

./keepalived/etc/rc.d/init.d/keepalived start #启动服务

2、修改Keepalived配置

global\_defs {

# keepalived 自带的邮件提醒需要开启 sendmail 服务。建议用独立的监控或第三方 SMTP,也可配置邮件发送

#三方的smtp

notification\_email {  #指定keepalived在发生切换时需要发送email到的对象，一行一个  
    [sysadmin@fire.loc](mailto:sysadmin@fire.loc) 收件人  
    }

notification\_email\_from Alexandre.Cassen@firewall.loc #指定发件人  
   smtp\_server localhost #指定smtp服务器地址  
   smtp\_connect\_timeout 30 #指定smtp连接超时时间

router\_id 192.168.0.121 #运行Keepalived的一个机器标识

}

vrrp\_script chk\_nginx {

script "/alidata/server/keepalived/nginx\_check.sh" #运行脚本，脚本内容下面有，就是起到一个nginx宕机以后，自动开启服务，vrrp\_script使用详解在下面

interval 2 #检测时间间隔

weight -20 #如果条件成立的话，则权重 -20

}

# 定义虚拟路由，VI\_1 为虚拟路由的标示符，自己定义名称

vrrp\_instance VI\_1 {

state MASTER/BACKUP #来决定主从

interface eno16777736 # 绑定虚拟 IP 的网络接口，根据自己的机器填写

virtual\_router\_id 121 # 虚拟路由的 ID 号， 两个节点设置必须一样

mcast\_src\_ip 192.168.0.121 #填写本机ip

priority 100 # 节点优先级,主要比从节点优先级高

nopreempt # 优先级高的设置 nopreempt 解决异常恢复后再次抢占的问题

advert\_int 1 # 组播信息发送间隔，两个节点设置必须一样，默认 1s

authentication {

auth\_type PASS

auth\_pass 1111

}

# 将 track\_script 块加入 instance 配置块

track\_script {

chk\_nginx #执行 Nginx 监控的服务

}

virtual\_ipaddress {

192.168.0.50 # 虚拟ip,也就是解决写死程序的ip怎么能切换的ip,也可扩展，用途广泛。可配置多个。

}

}

监控脚本

#!/bin/bash

A=`ps -C nginx –no-header |wc -l`

if [ $A -eq 0 ];then

/alidata/server/nginx/sbin/nginx

sleep 2

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

killall keepalived

fi

fi

**vrrp\_script使用（用于一对一的情况）**

在一个一主多备的Keepalived集群中，“priority”值最大的将成为集群中的Master节点，而其他都是Backup节点。在Master节点发生故障后，Backup节点之间将进行“民主选举”，通过对节点优先级值“priority”和““weight”的计算，选出新的Master节点接管集群服务。

    在vrrp\_script模块中，如果不设置“weight”选项值，那么集群优先级的选择将由Keepalived配置文件中的“priority”值决定，而在需要对集群中优先级进行灵活控制时，可以通过在vrrp\_script模块中设置“weight”值来实现。下面列举一个实例来具体说明。

    假定有A和B两节点组成的Keepalived集群，在A节点keepalived.conf文件中，设置“priority”值为100，而在B节点keepalived.conf文件中，设置“priority”值为80，并且A、B两个节点都使用了“vrrp\_script”模块来监控mysql服务，同时都设置“weight”值为10，那么将会发生如下情况。

    在两节点都启动Keepalived服务后，正常情况是A节点将成为集群中的Master节点，而B自动成为Backup节点，此时将A节点的mysql服务关闭，通过查看日志发现，并没有出现B节点接管A节点的日志，B节点仍然处于Backup状态，而A节点依旧是Master状态，在这种情况下整个HA集群将失去意义。

下面就分析一下产生这种情况的原因，这也就是Keepalived集群中主、备角色选举策略的问题。下面总结了在Keepalived中使用vrrp\_script模块时整个集群角色的选举算法，由于“weight”值可以是正数也可以是负数，因此，要分两种情况进行说明。

    1.“weight”值为正数时

    在vrrp\_script中指定的脚本如果检测成功，那么Master节点的权值将是“weight值与”priority“值之和，如果脚本检测失败，那么Master节点的权值保持为“priority”值，因此切换策略为：

Master节点“vrrp\_script”脚本检测失败时，如果Master节点“priority”值小于Backup节点“weight值与”priority“值之和，将发生主、备切换。

Master节点“vrrp\_script”脚本检测成功时，如果Master节点“weight”值与“priority”值之和大于Backup节点“weight”值与“priority”值之和，主节点依然为主节点，不发生切换。

    2.“weight”值为负数时

    在“vrrp\_script”中指定的脚本如果检测成功，那么Master节点的权值仍为“priority”值，当脚本检测失败时，Master节点的权值将是“priority“值与“weight”值之差，因此切换策略为：

Master节点“vrrp\_script”脚本检测失败时，如果Master节点“priority”值与“weight”值之差小于Backup节点“priority”值，将发生主、备切换。

    Master节点“vrrp\_script”脚本检测成功时，如果Master节点“priority”值大于Backup节点“priority”值时，主节点依然为主节点，不发生切换。

    在熟悉了Keepalived主、备角色的选举策略后，再来分析一下刚才实例，由于A、B两个节点设置的“weight”值都为10，因此符合选举策略的第一种，在A节点停止Mysql服务后，A节点的脚本检测将失败，此时A节点的权值将保持为A节点上设置的“priority”值，即为100，而B节点的权值将变为“weight”值与“priority”值之和，也就是90（10+80），这样就出现了A节点权值仍然大于B节点权值的情况，因此不会发生主、备切换。

对于“weight”值的设置，有一个简单的标准，即“weight”值的绝对值要大于Master和Backup节点“priority”值之差。对于上面A、B两个节点的例子，只要设置“weight”值大于20即可保证集群正常运行和切换。由此可见，对于“weight值的设置，要非常谨慎，如果设置不好，将导致集群角色选举失败，使集群陷于瘫痪状态。

### 1.2.2 Keepalived监控所有的nginx节点

加入lvs，由lvs实现对nginx负载

global\_defs {

notification\_email {

edisonchou@hotmail.com

}

notification\_email\_from sns-lvs@gmail.com

smtp\_server 192.168.80.1

smtp\_connection\_timeout 30

router\_id LVS\_DEVEL # 设置lvs的id，在一个网络内应该是唯一的

}

vrrp\_instance VI\_1 {

state MASTER #指定Keepalived的角色，MASTER为主，BACKUP为备 记得大写

interface eno16777736 #网卡id 不同的电脑网卡id会有区别 可以使用:ip a查看

virtual\_router\_id 51 #虚拟路由编号，主备要一致取值在0-255之间，用来区分多个instance的VRRP组播

priority 100 #定义优先级，数字越大，优先级越高，主DR必须大于备用DR

advert\_int 1 #检查间隔，默认为1s

authentication { #这里配置的密码最多为8位，主备要一致，否则无法正常通讯

auth\_type PASS

auth\_pass 1111

}

virtual\_ipaddress {

192.168.1.200 #定义虚拟IP(VIP)为192.168.1.200，可多设，每行一个

}

}

# 定义对外提供服务的LVS的VIP以及port

virtual\_server 192.168.1.200 80 {

delay\_loop 6 # 设置健康检查时间，单位是秒即每隔6秒realserver的状态

lb\_algo rr # 设置负载调度的算法为轮循 **算法与负载机制在下面**

lb\_kind DR # 设置LVS实现负载的机制，有NAT、TUN、DR三个模式

nat\_mask 255.255.255.0

persistence\_timeout 60 在60秒内同一ip的访问会被分配到同一台realserver上，即会话保持时间

protocol TCP 用tcp协议检查realserver的状态

#virtualhost string：检查的web服务器的虚拟主机（host：头）

sorry\_server ip port:备用机，所有realserver失效后启用

real\_server 192.168.1.130 80 { # 指定real server1的IP地址

weight 3 # 配置节点权值，数字越大权重越高，默认为1,0为失效

TCP\_CHECK {

connect\_timeout 10 10秒无响应超时

nb\_get\_retry 3 重连次数

delay\_before\_retry 3 重连时间间隔

connect\_port 80 健康检查，如果端口通则认为服务正常此处80为nginx端口，如果是MySQL此处为3306

}

}

real\_server 192.168.1.131 80 { # 指定real server2的IP地址

weight 3 # 配置节点权值，数字越大权重越高

TCP\_CHECK {

connect\_timeout 10

nb\_get\_retry 3

delay\_before\_retry 3

connect\_port 80

}

}

}

lb\_algo：lvs负载调度算法

固定调度算法：rr、wrr、dh、sh

rr：轮循调度

wrr：轮循加权重调度，权重越高分发请求数越多，权重取值范围0-100

dh：目的地址哈希调度，以目的地址获取hash来确定目标服务器

sh：源地址哈希调度，以源地址为关键字获取hash确定目标服务器

动态调度算法：

LC：最少连接 least connection，这个算法把请求分发给连接最少的服务器

WLC：权重最少连接

LBLC：最小连接数调度，IPVS表存储了所有活动的连接，lb会比较将连接请求发送到当前连接最少的机器

LBLCR：带复制的基于本地最少连接，时lblc的改进

**\*\*\*\*\*\*\*\*下面为完整配制，生产不需要，根据需要配置**

global\_defs {

notification\_email {

edisonchou@hotmail.com

}

notification\_email\_from sns-lvs@gmail.com

smtp\_server 192.168.80.1

smtp\_connection\_timeout 30

router\_id LVS\_DEVEL # 设置lvs的id，在一个网络内应该是唯一的

}

vrrp\_instance VI\_1 {

state MASTER #指定Keepalived的角色，MASTER为主，BACKUP为备 记得大写

interface eno16777736 #网卡id 不同的电脑网卡id会有区别 可以使用:ip a查看

virtual\_router\_id 51 #虚拟路由编号，主备要一致取值在0-255之间，用来区分多个instance的VRRP组播

**nopreempt #当主down时，备接管，主恢复不自动接管**

priority 100 #定义优先级，数字越大，优先级越高，主DR必须大于备用DR

advert\_int 1 #检查间隔，默认为1s

authentication { #这里配置的密码最多为8位，主备要一致，否则无法正常通讯

auth\_type PASS

auth\_pass 1111

}

virtual\_ipaddress {

192.168.1.200 #定义虚拟IP(VIP)为192.168.1.200，可多设，每行一个

}

notify\_master /etc/keepalived/to\_master.sh:节点变为master时执行

notify\_backup:节点变为backup时执行

notify-fault：节点变为故障时执行

}

# 定义对外提供服务的LVS的VIP以及port

virtual\_server 192.168.1.200 80 {

delay\_loop 6 # 设置健康检查时间，单位是秒即每隔6秒realserver的状态

lb\_algo rr # 设置负载调度的算法为轮循 **算法与负载机制在下面**

lb\_kind DR # 设置LVS实现负载的机制，有NAT、TUN、DR三个模式

nat\_mask 255.255.255.0

persistence\_timeout 60 在60秒内同一ip的访问会被分配到同一台realserver上，即会话保持时间

protocol TCP 用tcp协议检查realserver的状态

#virtualhost string：检查的web服务器的虚拟主机（host：头）

sorry\_server ip port:备用机，所有realserver失效后启用

**real\_server 192.168.1.130 80 { # 指定real server1的IP地址**

weight 3 # 配置节点权值，数字越大权重越高，默认为1,0为失效

notify\_up：虚拟服务器启动时执行

notify\_down /etc/keepalived/kill\_keepalived.sh  #检测3306端口为down状态就执行此脚本（只有keepalived关闭，VIP才漂移，此种情况用于lvs只绑定一台机器时）

TCP\_CHECK {

connect\_timeout 10 10秒无响应超时

nb\_get\_retry 3 重连次数

delay\_before\_retry 3 重连时间间隔

connect\_port 80 健康检查，如果端口通则认为服务正常此处80为nginx端口，如果是MySQL此处为3306

}

HTTP\_GET|SSL\_GET {  http服务状态健康检测

      url {

        path /index.html        #检测url，可写多个

        digest  24326582a86bee478bac72d5af25089e    #检测效验码

        genhash –s IP –p 80 –u http://IP/index.html #digest效验码获取方法

        status\_code 200    #检测返回http状态码

      }

      connect\_port 80 #连接端口

      connect\_timeout 3  #连接超时时间

      nb\_get\_retry 3  #重试次数

      delay\_before\_retry 2 #连接间隔时间

   }

SMTP\_CHECK {  #健康检测邮件服务器smtp

      host {

        connect\_ip

        connect\_port

      }

      connect\_timeout 5

      retry 2

      delay\_before\_retry 3

      hello\_name "mail.domain.com" SMTP hello请求命令参数，可选

   }

MISC\_CHECK{

     misc\_path <string> | <quoted-string> #外部脚本路径

    misc\_timeout #脚本执行超时时间

    misc\_dynamic #如设置该项，则退出状态码会用来动态调整服务器的权重，返回0 正常，不修改；返回1，检查失败，权重改为0；返回2-255，正常，权重设置为：返回状态码-2

}

}

real\_server 192.168.1.131 80 { # 指定real server2的IP地址

weight 3 # 配置节点权值，数字越大权重越高

TCP\_CHECK {

connect\_timeout 10

nb\_get\_retry 3

delay\_before\_retry 3

connect\_port 80

}

}

}

1、当服务器变为master时执行此脚本

# cat to\_master.sh

#!/bin/bash

Date=$(date +%F" "%T)

IP=$(ifconfig eth0 |grep "inet addr" |cut -d":" -f2 |awk '{print $1}')

Mail="baojingtongzhi@163.com"

echo "$Date $IP change to master." |mail -s "Master-Backup Change Status" $Mail

2、当服务器改变为备时执行此脚本

# cat to\_backup.sh

#!/bin/bash

Date=$(date +%F" "%T)

IP=$(ifconfig eth0 |grep "inet addr" |cut -d":" -f2 |awk '{print $1}')

Mail="baojingtongzhi@163.com"

echo "$Date $IP change to backup." |mail -s "Master-Backup Change Status" $Mail

3、当服务器改变为故障时执行此脚本

# cat to\_fault.sh

#!/bin/bash

Date=$(date +%F" "%T)

IP=$(ifconfig eth0 |grep "inet addr" |cut -d":" -f2 |awk '{print $1}')

Mail="baojingtongzhi@163.com"

echo "$Date $IP change to fault." |mail -s "Master-Backup Change Status" $Mail

4、当检测TCP端口3306为不可用时，执行此脚本，杀死keepalived，实现切换

# cat mysql\_down.sh

#!/bin/bash

Date=$(date +%F" "%T)

IP=$(ifconfig eth0 |grep "inet addr" |cut -d":" -f2 |awk '{print $1}')

Mail="baojingtongzhi@163.com"

pkill keepalived

echo "$Date $IP The mysql service failure,kill keepalived." |mail -s "Master-Backup MySQL Monitor" $Mail

5、当检测TCP端口3306可用时，执行此脚本

# cat mysql\_up.sh

#!/bin/bash

Date=$(date +%F" "%T)

IP=$(ifconfig eth0 |grep "inet addr" |cut -d":" -f2 |awk '{print $1}')

Mail="baojingtongzhi@163.com"

echo "$Date $IP The mysql service is recovery." |mail -s "Master-Backup MySQL Monitor" $Mail

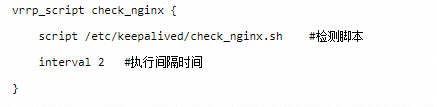
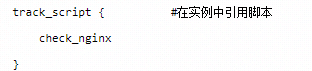
### 1.2.3 Keepalived的使用总结

当使用一对一部署时，如果服务器宕机，则需要将对应Keepalived关闭，切换到备用服务器，否则当前的Keepalived转发请求到宕机服务器，如果使用了lvs对服务器（nginx）进行负载，则无需关闭Keepalived

Layer3：工作在三层时，keepalived会定期向热备组中的服务器发送一个ICMP数据包，来判断某台服务器是否故障，如果故障则将这台服务器从热备组移除。

Layer4：工作在四层时，keepalived以TCP端口的状态判断服务器是否故障，比如检测mysql 3306端口，如果故障则将这台服务器从热备组移除。

**Layer7：**工作在七层时，keepalived根据用户设定的策略判断服务器上的程序是否正常运行，如果故障则将这台服务器从热备组移除，一对一中的脚本。

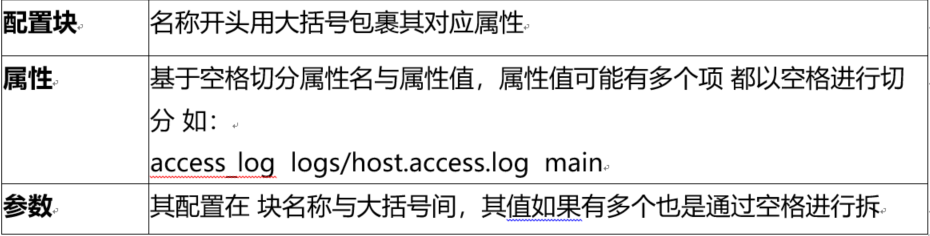
 

# 2、nginx的使用

## 2.1 nginx的配置

Nginx可实现的功能：动静分离、防盗链、多域名站点、下载限速、IP黑名单、基于user-agent分流

配置中events、http、server、location、upstream等属于配置块。而worker\_processes、worker\_connections、include、listen属于配置块中的属性。其中server块嵌套于http块，其可以直接访问继承http块当中的参数



注意 如果配置项值中包括语法符号，比如空格符，那么需要使用单引号或双引号括住配置项值，否则Nginx会报语法错误。例如：

    log\_format  main  '$remote\_addr - $remote\_user [$time\_local] "$request" '

                     '$status $body\_bytes\_sent "$http\_referer" ' '"$http\_user\_agent" "$http\_x\_forwarded\_for"';

基本配置介绍：

1、监听端口

语法：listen address

默认：listen 80

配置块：server

2、主机名称

语法：server\_name name[……]

默认：server\_name “”;

Server\_name [www.testweb.com](http://www.testweb.com) download.testweb.com;支持通配符与正则

配置块：server

3、location

语法：[=|~|~\*|^~|@]/uri/{……}

配置块：server

1. =表示把URI作为字符串，以便与参数中的uri做完全匹配

2. /基于uri目录匹配

3. ~表示正则匹配URI时字母大小写敏感

4. ~\*表示正则匹配URI时忽略大小写问题

5. ^~表示匹配URI时只需要前半部分与uri参数匹配即可

匹配的优先规则：

a、精确匹配优先= b、正则匹配优先^~ c、前缀最大匹配优先 d、配置靠前优化

4、root指定站点目录

可配置在server与location中，基于root路径+url中路径去寻找指定文件

5、alias指定站点别名

只能配置于location中，基于alias路径+URL移除location前缀后的路径来寻找文件

如下示例：

location /V1 {

      alias  /www/old\_site;

      index  index.html index.htm;

}

#防问规则如下

URL：http://xxx:xx/V1/a.html

最终寻址：/www/old\_site/a.thml

6、**动静分离实现**

A、创建静态站点 b、配置location/static c、配置~\*\.(gif|png|css|js)$

server {

        listen 80;

        server\_name \*.luban.com;

        root /usr/www/luban;

        location / {

                index luban.html;

        }

        location /static {

         alias /usr/www/static;

        }

 }

基于正则动静分离

location ~\* \.(gif|jpg|png|css|js)$ {

      root /usr/www/static;

}

7、防盗链配置（浏览器另存也会被屏蔽）

# 加入至指定location 即可实现（需要保护的资源只需在其location加入，代表只有符合\*.luban.com的访问路径才能获取资源）

valid\_referers none blocked \*.luban.com;

 if ($invalid\_referer) {

       return 403;

}

8、**下载限速**

location /download {

    limit\_rate 1m; //限制每S下载速度

limit\_rate\_after 30m; // 超过30 m 后下载限速

}

9、创建IP黑名单

#封禁指定IP

deny 192.168.0.1;

allow 192.168.0.1;

#开放指定IP 段

allow 192.168.0.0/24;

#封禁所有

deny all;

#开放所有

allow all;

# 创建黑名单文件

echo 'deny 192.168.0.132;' >> black.ip

#http 配置块中引入 黑名单文件

include       black.ip;

以deny all 结尾代表除了上面allow的其余全部被禁止

deny一定要加一个ip，否则直接跳转到403，不往下执行了，如果403默认页是同一域名下，会造成死循环访问

24代表子网掩码：255.255.255.0

16代表子网掩码：255.255.0.0

8代表子网掩码：255.0.0.0

3、日志配置：

日志格式（log\_format 日志格式，main日志格式名称 具体的日志格式参数）：

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;

$remote\_addr：客户端ip地址

$remote\_user：客户端用户名

$request：记录请求的URL与http协议（get，post，del，put等）

$status：记录请求状态（如200）

$body\_bytes\_sent：发送给客户端的字节数，不包括响应头的大小

$ bytes\_sent：发送给客户端的总字节数

$connection:连接的序列号

$connection\_requests:当前通过一个连接获得的请求数量

$msec：日志写入时间。单位为s，精度是毫秒

$pipe：如果请求是通过HTTP流水线（pipelined）发送，pipe值为“p”，否则为“.”

$http\_referer:记录从哪个页面连接访问过来的

$http\_user\_agent:记录客户端浏览器相关信息

$request\_uri：包含请求参数的原始uri，不包含主机名

$request\_length 请求的长度（包括请求行，请求头和请求正文）

$request\_time:请求处理时间，单位秒，精度毫秒；从读入客户端的第一个字节开始，直到把最后一个字符发送给客户端后进行日志写入为止

$time\_iso8601 ISO8601 标准格式下的本地时间

$time\_local 通用日志格式下的本地时间

$http\_host：带有端口号的ip

$host:不带端口号的ip

#基于域名打印日志

access\_log logs/$host.access.log main;

error日志的设置

语法：error\_log /path/file level;

默认：error\_log logs/error.log error;

level是日志的输出级别，取值范围是debug、info、notice、warn、error、crit、alert、emerg，

针对指定的客户端输出debug级别的日志

语法：debug\_connection[IP|CIDR]

events {

debug\_connection 192.168.0.147;

debug\_connection 10.224.57.0/200;

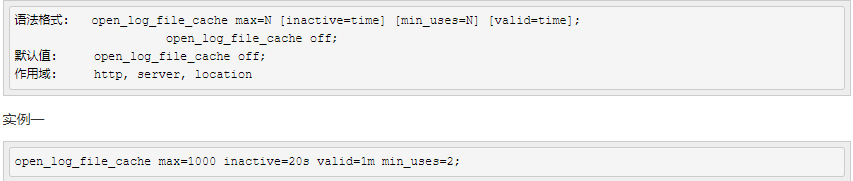
}

注意：debug 日志开启 必须在安装时 添加 --with-debug (允许debug)

#### open\_log\_file\_cache

使用open\_log\_file\_cache来设置日志文件缓存(默认是off)。

* max:设置缓存中的最大文件描述符数量，如果缓存被占满，采用LRU算法将描述符关闭。
* inactive:设置存活时间，默认是10s
* min\_uses:设置在inactive时间段内，日志文件最少使用多少次后，该日志文件描述符记入缓存中，默认是1次
* valid:设置检查频率，默认60s
* off：禁用缓存



## 2.2 nginx的方向代理与负载均衡

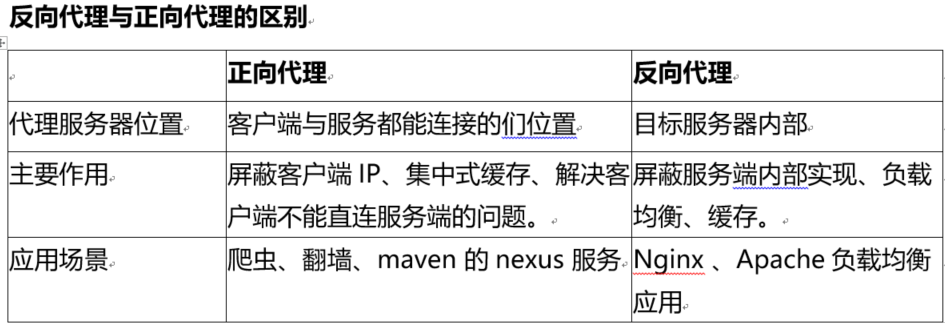
### 2.2.1 nginx的方向代理

**正向代理：**

指客户端与目标服务器之间增加一个代理服务器，客户端直接访问代理服务器，在代服务器访问目标服务器并返回客户端。这个过程当中客户端需要知道代理服务器的地址，并配置连接

**反向代理：**

指客户端访问目标服务器，在目标服务器内部有一个统一接入网关请求转发至后端真正处理的服务器并返回结果。这个过程中客户端不需要知道代理服务器地址，代理对客户端而言是透明的



**Nginx代理基本配置**

Nginx 代理只需要配置 location 中配置proxy\_pass 属性即可。其指向代理的服务器地址。

**正向代理与方向代理本质相同，只是看代理服务器由谁控制，客户端ip等信息是否传递到服务端，代理服务器转发的路径由谁来写来控制**

# 正向代理到baidu 服务

location = /baidu.html {

         proxy\_pass http://www.baidu.com;

}

# 反向代理至 本机的8010服务

location /luban/ {

     proxy\_pass http://127.0.0.1:8010;

}

**代理相关参数：**

proxy\_pass           # 代理服务（进行转发，类似于forward）

proxy\_redirect off;   #不允许重定向，默认default，还有redirect replacement替换为参数即（a b 将location路径中的a替换为b）（重定向，将路径返回给客户端，由客户端重新发送request请求）

proxy\_set\_header Host $host; # 传 header 参数至后端服务

proxy\_set\_header X-Forwarded-For $remote\_addr; # 设置request header 即客户端IP 地址

keepalive\_timeout 65:参数是一个请求完成之后还要保持连接多久，即长连接，减少创建连接过程给系统内带来的的性能开销

proxy\_connect\_timeout 90; # 连接代理服务超时时间

proxy\_send\_timeout 90; # 请求发送最大时间

proxy\_read\_timeout 90;  # 读取最大时间

proxy\_buffer\_size 4k;

proxy\_buffers 4 32k;

proxy\_busy\_buffers\_size 64k;

proxy\_temp\_file\_write\_size 64k;

sendfile on/off；开启文件高效传输模式(避免了数据在用户态与内核态之间的拷贝)，同时与tcp\_nopush  和tcp\_nodelay  两个指令配合使用，{http，server，location，if in location}

tcp\_nopush on/off： 当有数据时，先别着急发送, 确保数据包已经装满数据, 避免了网络拥塞，默认off{ http，server，location }

tcp\_nodelay on/off：有时要抓紧发货, 确保数据尽快发送, 提高可数据传输效率，默认on，{ http，server，location }

### 2.2.2 nginx负载均衡

**1、Nginx使用upstream模块实现负载均衡**

upstream backend {

server 127.0.0.1:8010 weight=1;

   server 127.0.0.1:8080 weight=2;

server 127.0.0.1:8030 weight=1 backup;

}

location / {

    proxy\_pass http://backend;

}

**2、upstream 相关参数:**

server 反向服务地址 加端口

weight 权重

max\_fails在fail\_timeout时间内失败多少次 认为主机已挂掉则，踢出

fail\_timeout服务器被认为停机的时间长度，默认为10s

backup备用服务（当所有服务挂掉之后启用）

max\_conns允许最大连接数

slow\_start当节点恢复，不立即加入,而是等待 slow\_start 后加入服务对列（防止启动后就有大量请求压垮服务器或者处理不及时被标记为down）。

* down 当前节点不可用

3、upstream 负载均衡算法介绍

1、ll+weight： 轮询加权重 (默认)

upstream backserver {   
server 192.168.0.14;   
server 192.168.0.15;   
}

upstream backserver {   
server 192.168.0.14 weight=8;   
server 192.168.0.15 weight=10;

server 192.168.0.16 backup  
}

2、ip\_hash : 基于Hash 计算 ,用于保持session 一至性，同一客户端每次使用同一服务器的服务

upstream backserver {   
ip\_hash;   
server 192.168.0.14:88;   
server 192.168.0.15:80;   
}

3、least\_conn ：最少链接

upstream dynamic\_zuoyu {

least\_conn; #把请求转发给连接数较少的后端服务器

server localhost:8080 weight=2; #tomcat 7.0

server localhost:8081; #tomcat 8.0

server localhost:8082 backup; #tomcat 8.5

server localhost:8083 max\_fails=3 fail\_timeout=20s; #tomcat 9.0

}

4、url\_hash: 静态资源缓存,节约存储，加快速度（第三方），每一个URL定向到同一台后端服务器，后端服务器有缓存时比较有效

upstream backserver {   
server squid1:3128;   
server squid2:3128;   
hash $request\_uri;   
hash\_method crc32;   
}

5、least\_time  ：最小的响应时间,计算节点平均响应时间，然后取响应最快的那个，分配更高权重（第三方）

upstream backserver {   
server server1;   
server server2;   
fair;   
}

6、**随机random**每个请求将传递到随机选择的服务器。如果two指定了参数，首先，NGINX考虑服务器权重随机选择两个服务器，然后使用指定的方法选择这些服务器之一：

least\_conn –活动连接最少

least\_time=header（NGINX Plus）–从服务器接收响应标头的最短平均时间（[$upstream\_header\_time](https://nginx.org/en/docs/http/ngx_http_upstream_module.html#var_upstream_header_time)）

least\_time=last\_byte（NGINX Plus）–从服务器接收完整响应的最短平均时间（[$upstream\_response\_time](https://nginx.org/en/docs/http/ngx_http_upstream_module.html#var_upstream_response_time)）

upstream backend {

random two least\_time=last\_byte;

server backend1.example.com;

server backend2.example.com;

server backend3.example.com;

server backend4.example.com;

}

### 2.2.3 nginx的缓冲控制

Nginx如果禁止缓冲，那么当nginx一收到后端的反馈就同时传给客户端

Nginx不会从被代理的服务器读取整个反馈信息

1、proxy\_buffering

语法：proxy\_buffering on|off

默认值：proxy\_buffering on 打开

上下文：http，server，location

此参数用来控制是否打开后端响应内容的缓冲区，如果值为off，那么proxy\_buffers与proxy\_buffers\_size这两个指令将会失效，但无论proxy\_buffering 是否开启，proxy\_buffer\_size都生效。在proxy\_buffering开启的情况下，nginx会把后端返回的内容先放到缓冲区当中，然后再返回给客户端（边收边传，不是全部接受完再传给客户端），临时文件由proxy\_max\_file\_temp\_size与proxy\_temp\_file\_write\_size这两个指令决定的。如果响应内用无法放在内存里边，那么部分内容会写到磁盘上，如果proxy\_buffering关闭，则nginx会立即把后端收到的响应内容传递给客户端，每次取的大小为proxy\_buffer\_size的大小，这样效率较低

注意：proxy\_buffering启用时，要提防使用的缓冲区太大，消耗太多内存，从而影响nginx能支持的最大并发连接数；对于基于长轮询（long-polling）的comet应用来说，关闭proxy\_buffering是重要的，不然异步响应将被缓存导致comet无法工作（即时通讯交互的程序会使用到）

2、**proxy\_buffer\_size**

语法：proxy\_buffer\_size size

默认值：proxy\_buffer\_size 4k/8k

上下文：http，server，location

该指令设置缓冲区大小，从代理的后端服务器取得的第一部分响应内容会放到这里，小的响应header通常位于这部分响应内容里边，默认来说，该缓冲区的大小等于proxy\_buffers所设置的，可以把它设置设置的更小

3、**proxy\_buffers**

语法：proxy\_buffers number size

默认值：proxy\_buffers 8 4k/8k

上下文：http，server，location

该指令设置缓冲区的大小和数量，从被代理的后端服务器取得响应内容会被放置到这里，默认情况下一个缓冲区的大小等于内存页面的大小，可能是4k也可能是8k，取决于平台

proxy\_buffers由缓冲区的数量与大小组成，总大小为number\*size，若某些请求的响应过大，则超过proxy\_buffers的部分将被缓冲到硬盘（缓冲目录由proxy\_temp\_path指令指定），当然这将会使读取响应速度减慢，影响用户体验，可以使用proxy\_max\_temp\_file\_size指令关闭磁盘缓冲

4、**proxy\_busy\_buffers\_size**

语法：proxy\_busy\_buffers\_size size

默认值：proxy\_busy\_buffers\_size proxy\_buffer\_size\*2

上下文：http，server，location，if

proxy\_busy\_buffers\_size不是独立的空间，它是proxy\_buffers与proxy\_buffer\_size的一部分。Nginx会在没有完全读完后端响应的时候就开始向客户端传送数据，所以它会划出一部分缓冲区来专门向客户端传送数据，（这部分的大小建议为proxy\_buffers中单个缓冲区大小的2倍），然后继续从后端读取数据，缓冲区满了之后就写到磁盘的临时文件中。

5、**proxy\_max\_temp\_file\_size和proxy\_temp\_file\_write\_size**

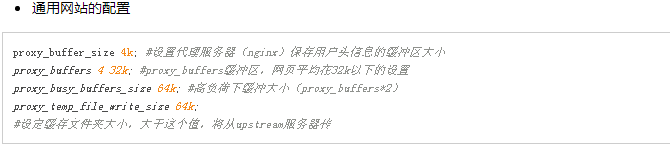
临时文件由proxy\_max\_temp\_file\_size和proxy\_temp\_file\_write\_size这两个指令决定。 proxy\_temp\_file\_write\_size是一次访问能写入的临时文件的大小，默认是proxy\_buffer\_size和proxy\_buffers中设置的缓冲区大小的2倍，Linux下一般是8k。  
proxy\_max\_temp\_file\_size指定当响应内容大于proxy\_buffers指定的缓冲区时, 写入硬盘的临时文件的大小. 如果超过了这个值, Nginx将与Proxy服务器同步的传递内容, 而不再缓冲到硬盘. 设置为0时, 则直接关闭硬盘缓冲。

总结：

1、所有的proxy\_buffer参数是作用到每一个请求，即每一个请求私有的

2、无论proxy\_buffering是否开启，proxy\_buffer\_size（main buffer）都是工作的，proxy\_buffer\_size所设置的buffer\_size的作用是用来存储upstream端response的header。

3、一旦proxy\_buffers设置的buffer被写入，直到buffer里面的数据被完整的传输完（传输到客户端），这个buffer将会一直处在busy状态，我们不能对这个buffer进行任何别的操作。所有处在busy状态的buffer size加起来不能超过proxy\_busy\_buffers\_size，所以proxy\_busy\_buffers\_size是用来控制同时传输到客户端的buffer数量的。



### 2.2.4 nginx静态资源缓存

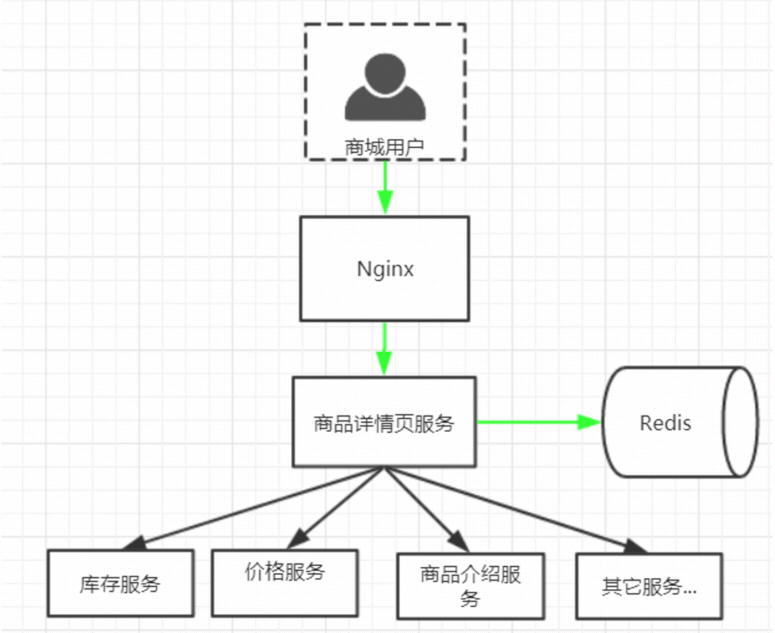
1、电商案例

一个商品的详情页主要涉及了以下服务，商品详情页HTML页面渲染，价格服务，促销服务，库存状态/配送服务，广告词服务，预售/秒杀服务，评价服务，试用服务，推荐服务，商品介绍服务，其它。

解决方案：

\* 采用Ajax动态加载价格、广告、库存的等服务

\* 采用key value 缓存详情页主题



问题：当达到500QPS时很难继续压测上去

原因：一个详情页HTML主体平均150kb，那么500qps已接近千M局域网宽带极限，必须减少内网通讯

解决：使用nginx静态缓存将静态页面缓存在nginx本地

**2、nginx静态缓存基本配置**

proxy\_cache\_path ：缓存路径

levels： 缓存层级及目录位数

keys\_zone： 缓存区内存大小

inactive ：有效期

max\_size： 缓存区硬盘最大值，超出后，闲置数据将会被删除

expires：浏览器中缓存过期时间（当缓存已过期或者缓存的文件已经被修改则浏览器会更新缓存）

 proxy\_cache\_valid：有效期，其优先级高于inactive，用于给不同的响应状态码设置缓存时间

proxy\_cache\_valid 200 302 24h;如果静态资源的响应状态码为200、302(暂时性重定向)缓存文件有效期为一天

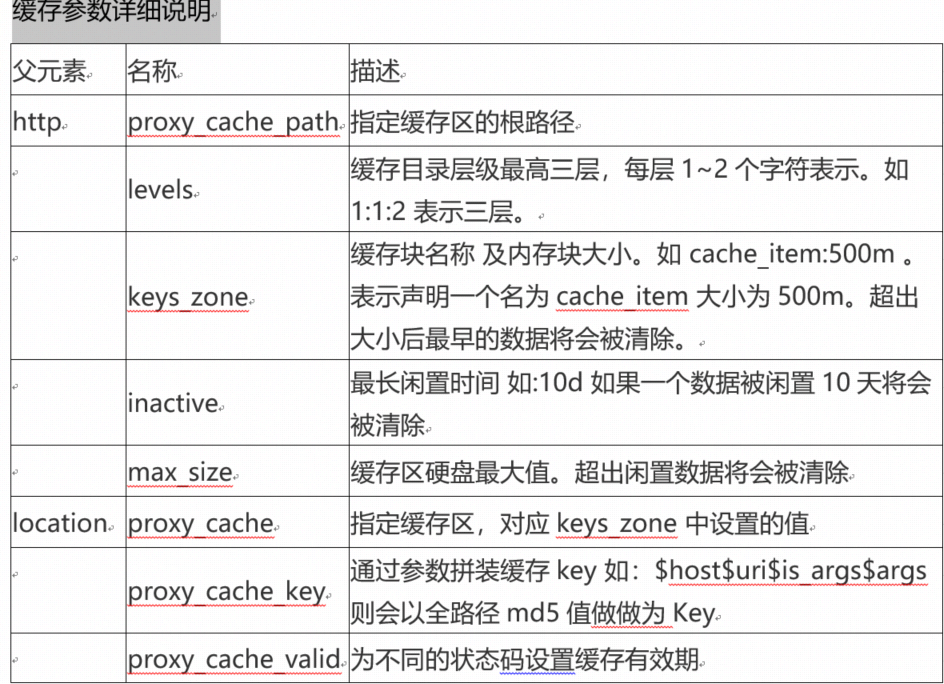
proxy\_cache\_valid any 5m：any代表其他情况

#以全路径md5值做做为Key

proxy\_cache\_key $host$uri$is\_args$args;

proxy\_cache\_path /data/nginx/cache\_luban levels=1:2

keys\_zone=cache\_luban:500m inactive=20d max\_size=1g;



**3、nginx压缩配置（http，server，location）**

gzip on/off :开启/关闭压缩

gzip\_min\_length 1k :设置允许压缩的页面最小字节（从header头的content-length中获取），建议大于1k

gzip\_buffers 4 16k :以16k为单位（如果数据大于16k），按照原始数据大小以16k为单位的4倍申情内存

gzip\_http\_version1.1 :识别http协议的版本，之前的浏览器可能不支持gzip自解压，用户会看到乱码

gzip\_comp\_level 6: 压缩级别1-9，比如level为1的话，压缩的比例比较低，但是效率比较高，比如100K的文件压缩之后还剩40K或者50K，但是处理的时间很短；如果level为9的话，压缩的效果最好，效率会低一点，比如还是100K的文件，压缩的会更小，甚至20K ，这样对cpu消耗会高点，一般设置中间差不多

gzip\_types: 匹配压缩类型定义了压缩的类型，比如文本的，js、css等文本文件压缩，像是图片啊，就没必要定义了，本身就支持，这里就不需要定义了，默认压缩text/html 不需要指定，否则报错text/plain application/x-javascript text/css application/xml

gzip\_vary on/off : 定义是否在发送客户端的响应头部插入“Vary：Accept-Encoding”响应信息，用于客户端浏览器识别内容是否已经进行压缩；默认：gzip\_vary off；

gzip\_proxied（默认值off，在做反向代理是使用，也可以不用不配置）

on/off:启用/关闭所有代理结果的数据压缩

expired:如果header头包含“Expires”头信息则启用压缩

no-cache: 如果header头包含“Cache-Control：no-cache”头信息则启用压缩(HTTP设置代表不使用缓存，向原服务器发起请求)

no-store：如果header头包含“Cache-Control：no-store”头信息则启用压缩（http设置所有内容不会保存到缓存或者internet临时文件中）

private: 如果header头包含“Cache-Control：private”头信息则启用压缩

no\_last\_modefied: 如果header头不包含“Last-Modefied”头信息则启用压缩

no\_etag: 如果header头不包含“Etag”头信息则启用压缩

auth：如果header头包含“Authorization”头信息则启用压缩

any：全部请求都做压缩

gzip\_disable：“MSIE[1-6]”IE1-6版本不支持gzip压缩

**4、缓存的清除**

该功能采用第三方模块ngx\_cache\_purge实现

为nginx添加ngx\_cache\_purge模块

#下载ngx\_cache\_purge 模块包 ,这里nginx 版本为1.6.2 purge 对应2.0版

wget http://labs.frickle.com/files/ngx\_cache\_purge-2.3.tar.gz

#查看已安装模块

./sbin/nginx -V

#进入nginx安装包目录 重新安装 --add-module为模块解压的全路径

./configure --prefix=/root/svr/nginx --with-http\_stub\_status\_module --with-http\_ssl\_module  --add-module=/root/svr/packages/ngx\_cache\_purge-2.3

#重新编译

make

#拷贝 安装目录/objs/nginx 文件用于替换原nginx 文件

#检测查看安装是否成功

nginx -t

**清除配置：**

location~/clear(/.\*){

允许访问的ip

allow 127.0.0.1

allow 192.168.0.15

其余ip禁止

deny all

#配置清除指定缓存区和路径（与proxy\_cache\_key一至）

proxy\_cache\_purge cache\_kevin$host$1$is\_args$args

配置好以后 直接访问 ：

# 访问生成缓存文件

http://www.luban.com/?a=1

# 清除生成的缓存,如果指定缓存不存在 则会报404 错误。

http://www.luban.com/clear/?a=1

# 3、nginx面试题

1、为什么nginx不能使用iphash存储session？

1、nginx是不是最前端的服务器。ip\_hash要求nginx一定是最前端的服务器，否则nginx得不到正确ip，就不能根据ip作hash。譬如使用的是squid为最前端，那么nginx取ip时只能得到squid的服务器ip地址，用这个地址来作分流是肯定错乱的。

2、nginx的后端还有其它方式的负载均衡。假如nginx后端又有其它负载均衡，将请求又通过另外的方式分流了，那么某个客户端的请求肯定不能定位到同一台session应用服务器上。这么算起来，nginx后端只能直接指向应用服务器，或者再搭一个squid，然后指向应用服务器。最好的办法是用location作一次分流，将需要session的部分请求通过ip\_hash分流，剩下的走其它后端去。