nginx笔记

一、简介

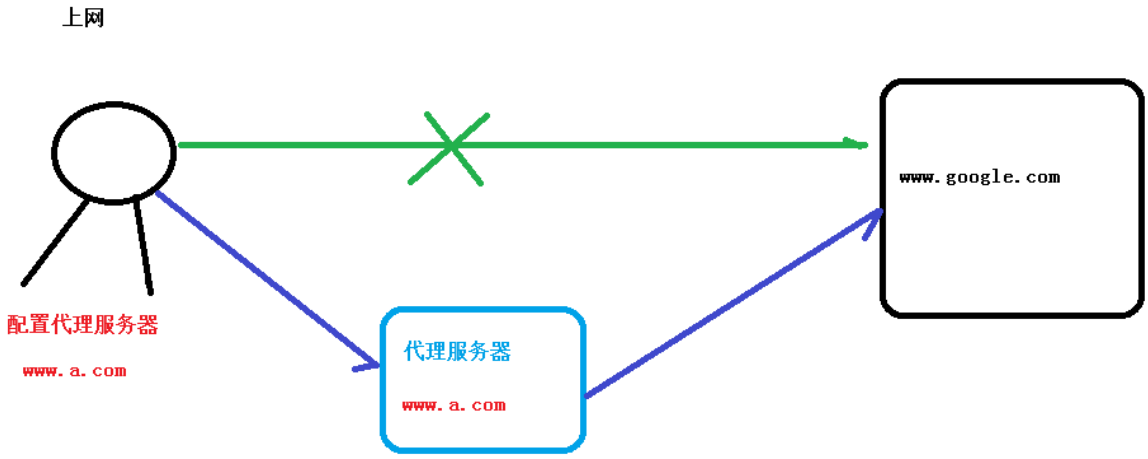
1. 什么是 nginx ?

nginx 是高性能的 HTTP 和反向代理的服务器，特点是占用内存少，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。

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。

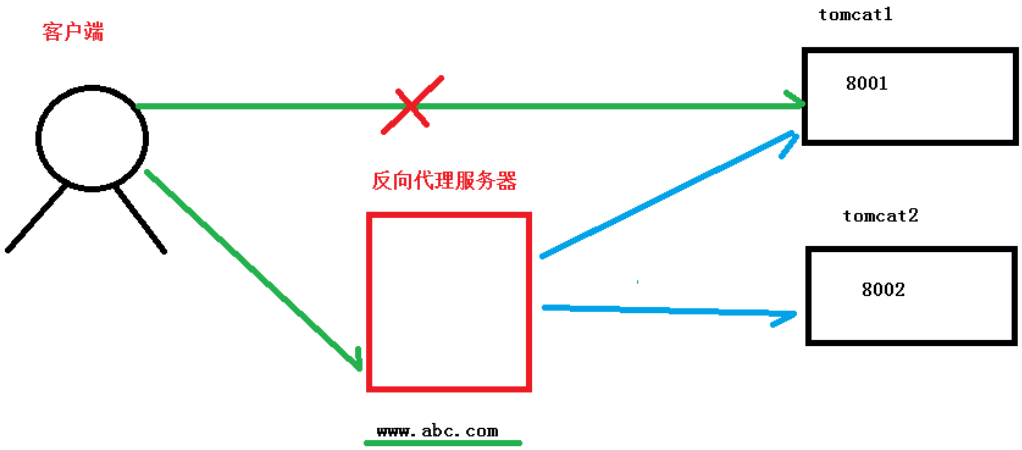
2. 正向代理

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。



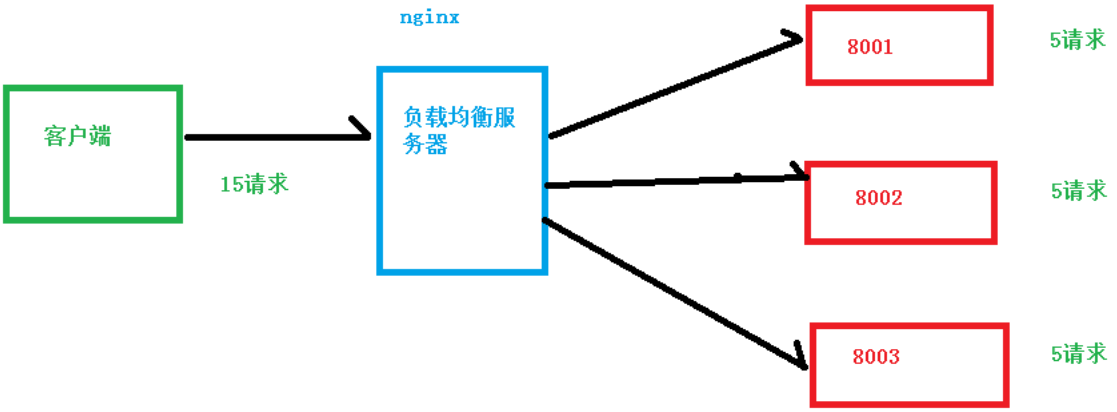
3. 反向代理

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。**



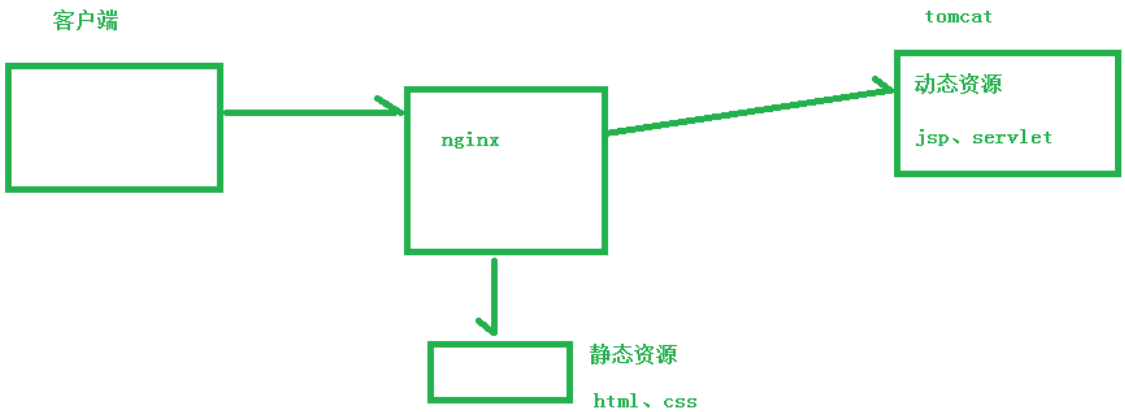
4. 负载均衡

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。



5. 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。



二、安装和启动

1. 安装

到 nginx 官网下载软件 http://nginx.org/

1. 安装依赖

```
sudo apt-get install gcc
sudo apt-get install libpcre3 libpcre3-dev
sudo apt-get install zlib1g-dev
sudo apt-get install openssl
```

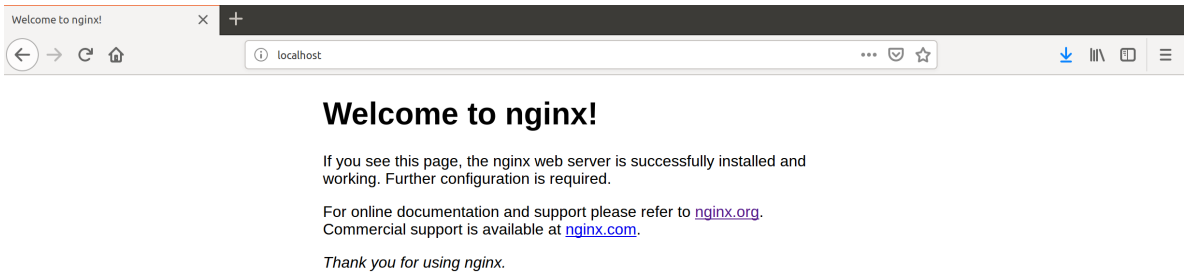
2. 安装 nginx

```
# 进入nginx目录
cd /usr/local/nginx-1.18.0
# 执行命令
./configure
# 执行make命令
make
# 执行make install命令
make install
```

2. 启动

```
cd /usr/local/nginx/sbin
./nginx
```

nginx启动成功：



3. 常用命令

进入 nginx目录中

```
cd /usr/local/nginx/sbin
```

1. 查看 nginx版本号

   ```
   ./nginx -v
   ```

2. 启动 nginx

   ```
   ./nginx
   ```

3. 停止 nginx

   ```
   ./nginx -s stop
   ```

4. 重新加载 nginx

   ```
   ./nginx -s reload
   ```

4. 配置文件

nginx 配置文件位置

```
vim /usr/local/nginx/conf/nginx.conf
```

配置文件中的内容，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
   比如 `worker_processes 1;` 处理并发数
2. events 块 ： 影响 nginx 服务器与用户的网络连接
   比如 `worker_connections 1024;` 支持的最大连接数为 1024
3. http 块：配置最频繁的部分。代理、缓存、日志定义都在这里。

   还包含两部分：http 全局块，server 块

三、配置实例

1. 反向代理实例1

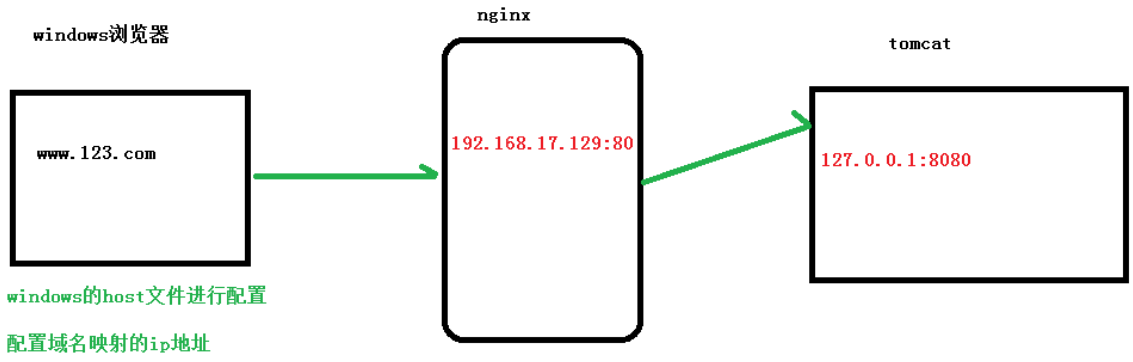
实现效果：打开浏览器，在浏览器地址栏输入地址 www.123.com，跳转到 linux 系统 tomcat 主页面中。

准备工作：

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

访问过程分析：



具体配置：

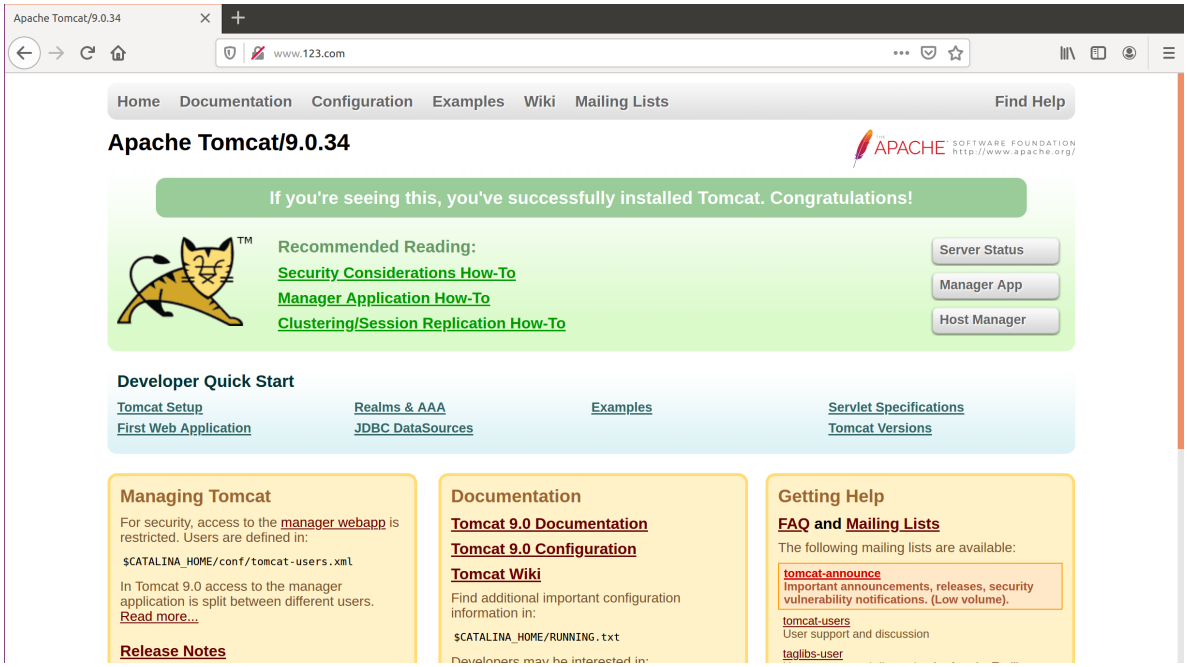
1. 配置 /etc/hosts 文件，添加下面内容：

   ```
   # nginx
   127.0.0.1 www.123.com
   ```

2. 在 nginx.conf 进行请求转发的配置（反向代理配置）

   ```
   location / {
       # 添加下面一行
       proxy_pass http://127.0.0.1:8080;
       root   html;
       index  index.html index.htm;
   }
   ```

最终测试：



2. 反向代理实例2

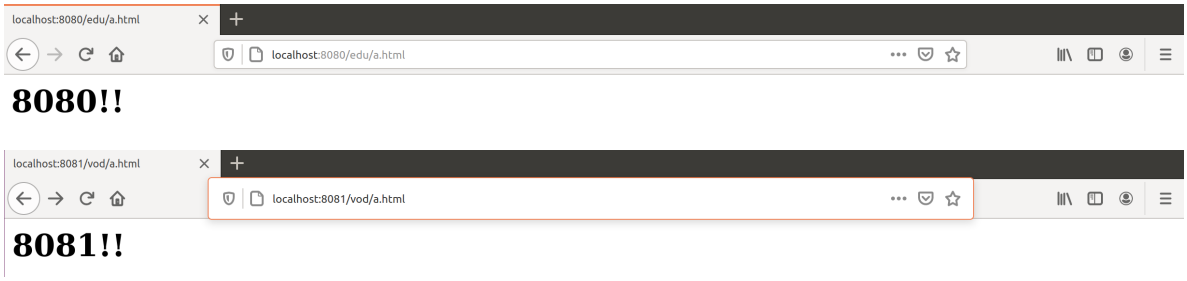
实现效果:

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 127.0.0.1:8080
访问 http://127.0.0.1:9001/vod/ 直接跳转到 127.0.0.1:8081

准备工作:

1 ）准备两个 tomcat 服务器，一个 8080 端口，一个 8081 端口
2 ）创建文件夹和测试页面



具体配置:

找到/usr/local/nginx/nginx.conf，进行反向代理配置

```
# nginx experiment
server {
    listen       9001;
    server_name  localhost;

    location ~ /edu/ {
        proxy_pass http://127.0.0.1:8080;
    }

    location ~ /vod/ {
        proxy_pass http://127.0.0.1:8081;
    }
}
```

重新加载 nginx 配置：

```
sudo ./nginx -s reload
```

最终测试：

3. 负载均衡

实现效果：浏览器地址栏输入地址 http://127.0.0.1/edu/a.html ，负载均衡效果，平均到8080和8081端口中。

准备工作：

1 ）准备两台 tomcat 服务器，一台 8080 ，一台 8081
2 ）在两台 tomcat 里面 webapps 目录中，创建名称是 edu 文件夹，在 edu 文件夹中创建页面 a.html ，用于测试

开始配置：

在 nginx.conf 进行请求转发的配置（反向代理配置）

```
# nginx experiment
upstream myserver {
    server 127.0.0.1:8080;
    server 127.0.0.1:8081;
}

server {
    listen       80;
    server_name  localhost;

    #charset koi8-r;

    #access_log  logs/host.access.log  main;
```

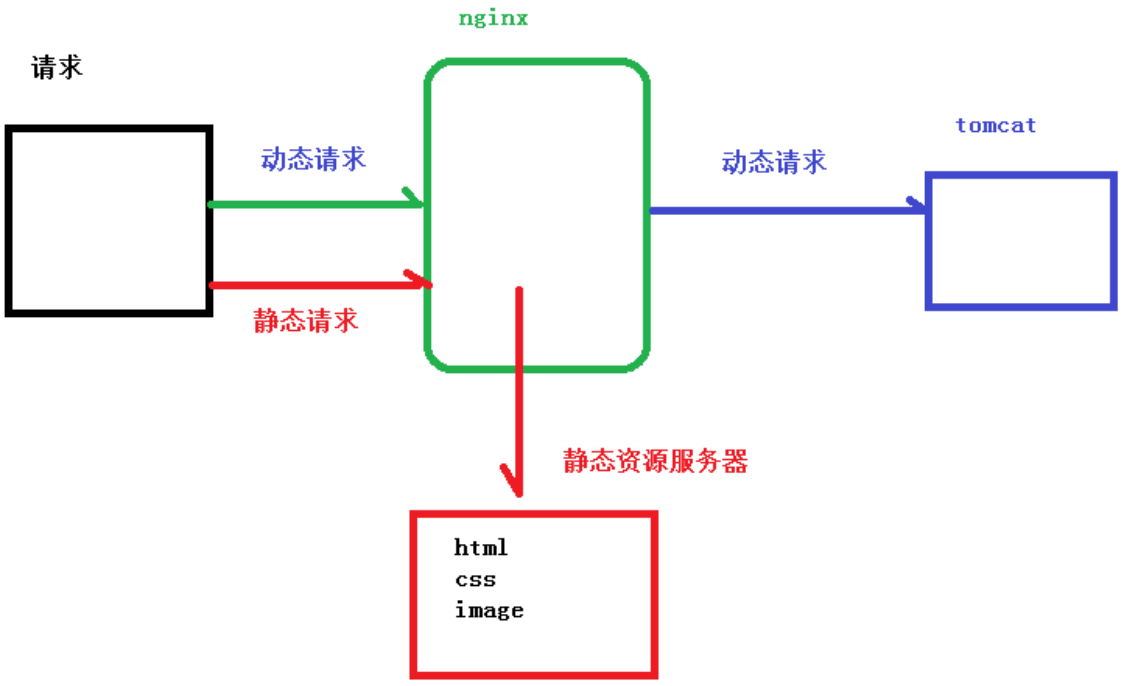
```
    location / {
        proxy_pass http://myserver;
        root   html;
        index  index.html index.htm;
    }
```

最终测试：

nginx分配服务器策略：

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**
   weigh t 代表权重默认为 1, 权重越高被分配的客户端越多
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
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。

4. 动静分离



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，减少与服务器之间的请求和流量。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 。

准备工作:

在 linux 系统中准备静态资源，用于进行访问。

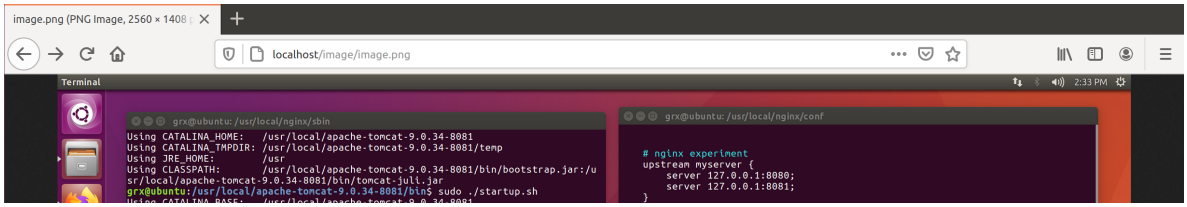
```
grx@ubuntu:/data$ ls
applogs  image  www
```

具体配置:

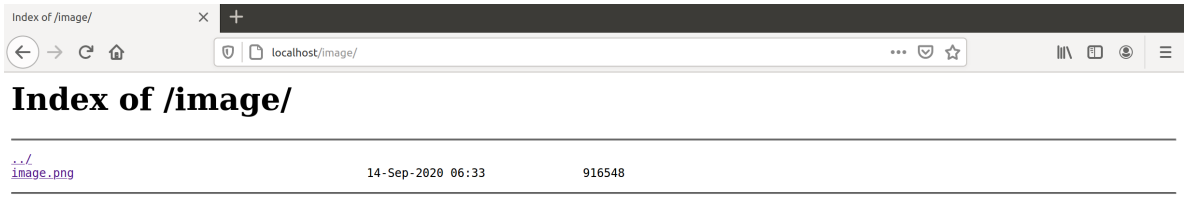
```
# nginx experiment
location /www/ {
    root /data/;
    index index.html index.htm;
}

location /image/ {
    root /data/;
    autoindex on;
}
```

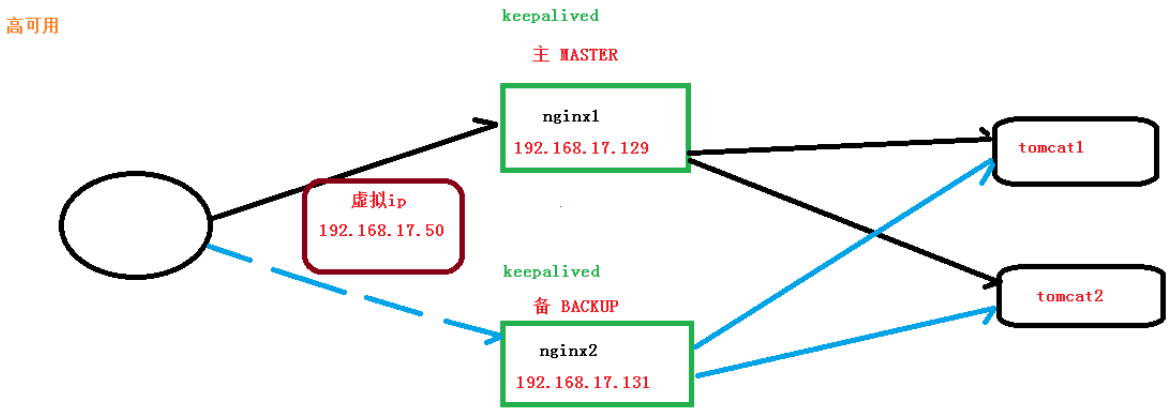
最终测试:



autoindex生效:



5. 配置高可用集群



准备工作:

1. 需要两台服务器 192.168.1.133 和 192.168.1.134。
2. 在两台服务器上安装nginx。

3. 在两台服务器上安装keepalived。

```
# 安装依赖
sudo apt-get install libssl-dev
sudo apt-get install openssl
sudo apt-get install libpopt-dev
# 安装keepalived
sudo apt-get install keepalived
```

主从配置:

修改 /etc/keepalived/keepalivec.conf 配置文件

```
global_defs {
        notification_email {
                acassen@firewall.loc
                failover@firewall.loc
                sysadmin@firewall.loc
        }
        notification_email_from Alexandre.Cassen@firewall.loc
        smtp_server 192.168.1.133
        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 MASTER # 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 {
                192.168.1.50 # 虚拟地址
        }
}
```

在 /usr/local/src 添加检测脚本 nginx_check.sh

```
#!/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
```

把两台服务器上 nginx 和 keepalived 启动
启动 nginx

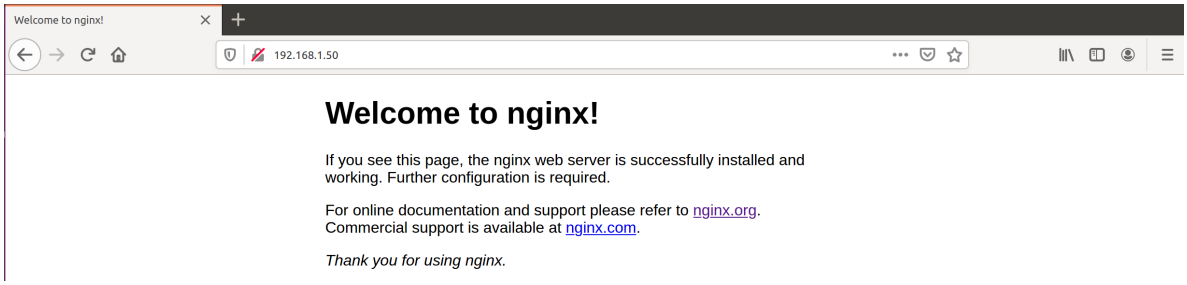
```
./nginx
```

启动 keepalived

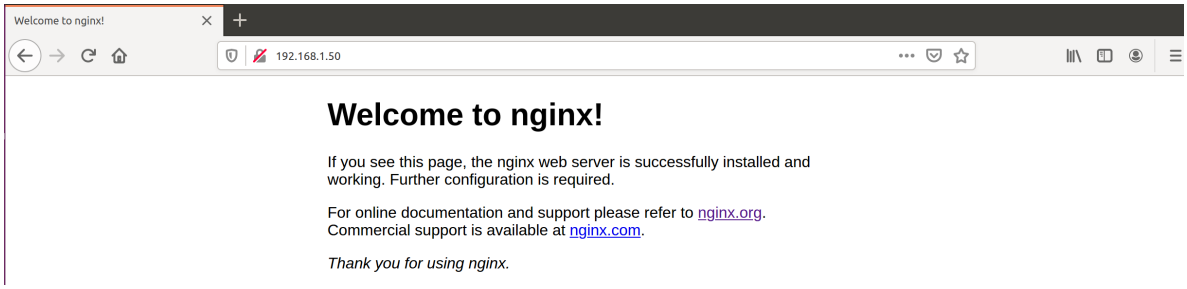
```
systemctl start keepalived.service
```

最终测试:

通过虚拟地址访问成功

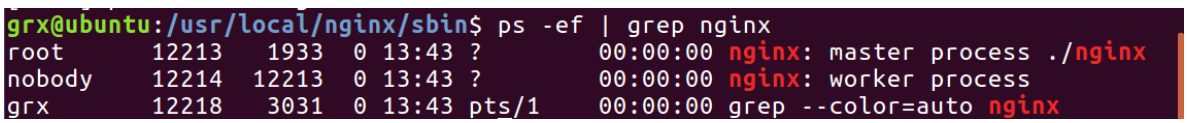


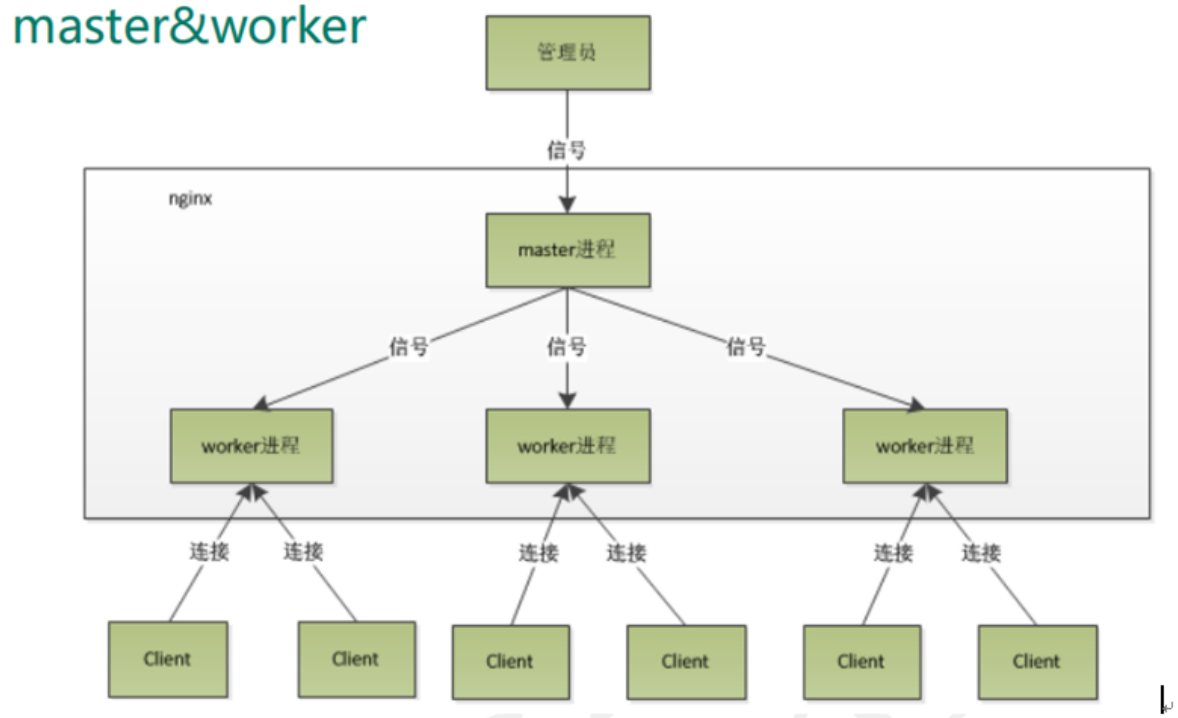
停止掉主机的 keepalived 和 nginx，仍然可以访问



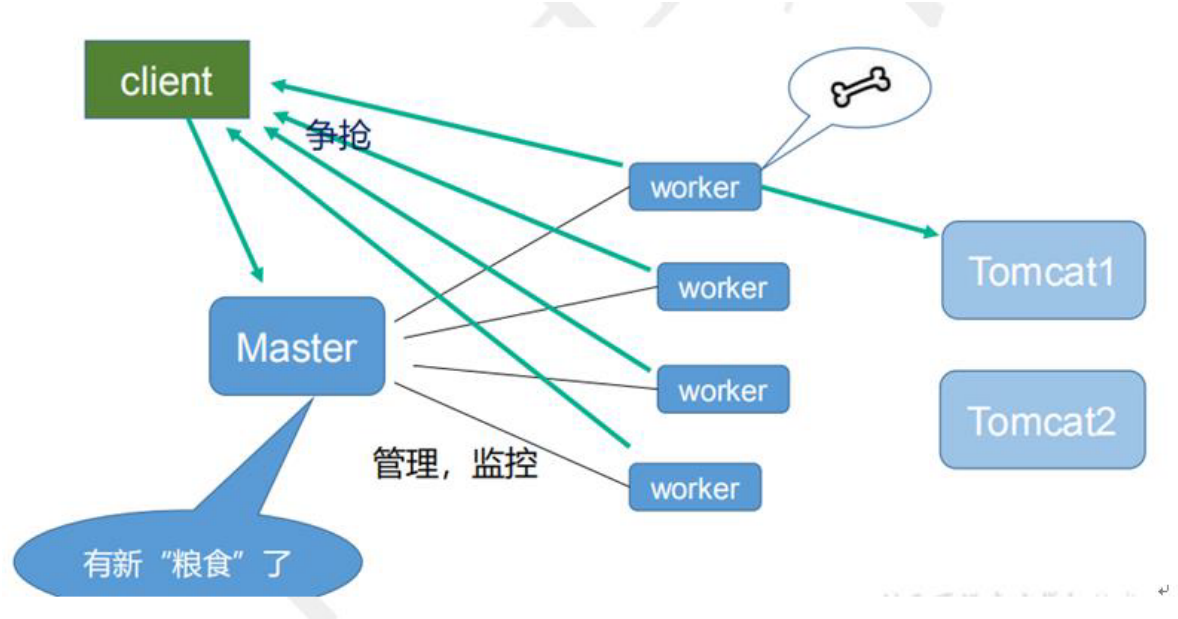
四、原理解析

Master和Worker





worker是如何工作的



一个master和多个worker的好处：

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 是独立的进程，如果有其中的一个 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

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?

2 （访问静态资源）或者 4 个（反向代理）

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