

day07-nfs-http-nginx-lnmp-proxy

- day07-nfs-http-nginx-lnmp-proxy
 - 01.NFS网络文件系统实战
 - 1.NFS基本概述
 - 2.NFS实现原理
 - 3.NFS服务安装
 - 4.NFS挂载卸载
 - 5.NFS配置详解
 - 6.NFS存储总结
 - 02.Nginx基础Http协议
 - 1.Http协议介绍
 - 1.1 什么是URL
 - 1.2 什么是HTML
 - 1.3 什么是HTTP
 - 1.4 URL、HTML、HTTP之间关系
 - 2.Http工作原理
 - 2.1 图解HTTP工作原理
 - 2.2 抓包分析HTTP原理
 - 2.3 HTTP工作原理总结
 - 3.Http请求Request
 - 3.1 请求Method
 - 3.2 请求Header
 - 3.3 请求Connection
 - 4.Http响应Response
 - 4.1 响应Header
 - 4.2 响应Status
 - 4.3 响应Code
 - 03.Nginx Web快速入门
 - 1.Nginx基本简述
 - 1.1 什么是Nginx
 - 1.2 为什么选择Nginx
 - 1.2.1 高性能、高并发
 - 1.2.2 高扩展性
 - 1.2.3 高可靠性
 - 1.2.4 热部署
 - 1.2.5 应用广泛

- 1.2.6 网络模型
- 1.3 Nginx应用场景
- 1.4 Nginx组成部分
- 2.Nginx快速安装
- 3.Nginx基本配置
- 4.Nginx搭建网站
- 5.Nginx虚拟主机
- 6.Nginx常用模块
 - 6.1 Nginx目录索引
 - 6.2 Nginx访问控制
 - 6.3 Nginx限流限速
 - 6.4 Nginx状态监控
 - 6.5 Nginx Location
- 04.Nginx搭建流行架构
 - 1.LNMP架构基本概述
 - 1.1 什么是LNMP
 - 1.2 LNMP实现过程
 - 1.3 LNMP实现细节
 - 2.LNMP架构环境安装
 - 2.1 Nginx安装
 - 2.2 php安装
 - 2.3 MySQL安装
 - 3.LNMP架构环境配置
 - 3.1 Fastcgi代理语法
 - 3.2 Nginx与PHP集成
 - 3.3 PHP与MySQL集成
 - 4.部署博客产品Wordpress
 - 4.1 配置Nginx
 - 4.2 配置MySQL
 - 4.3 部署Wordpress
 - 5.部署知乎产品Wecenter
 - 5.1 配置Nginx
 - 5.2 配置MySQL
 - 5.3 部署wecenter
 - 6.拆分数据库至独立服务器
 - 6.1 为何要拆分数据库
 - 6.2 数据库拆分架构演变
 - 6.3 数据库拆分环境准备
 - 6.4 数据库拆分实现步骤

- 6.4.1 web服务操作如下
 - 6.4.2 数据库服务操作如下
 - 6.4.3 修改代码指向新数据库
 - 7.扩展多台相同的Web服务器
 - 7.1 为何要扩展多台web节点
 - 7.2 扩展多web节点架构演变
 - 7.3 扩展多web节点环境准备
 - 7.4 扩展多web节点实现步骤
 - 7.4.1 LNP环境安装
 - 7.4.2 LNP配置导入
 - 7.4.3 导入代码文件
 - 7.4.4 启动服务验证
 - 8.拆分静态资源至独立服务器
 - 8.1 为何要拆分静态资源
 - 8.2 拆分静态资源架构演变
 - 8.3 增加共享存储环境准备
 - 8.4 增加共享存储实现步骤
 - 8.4.1 配置NFS存储
 - 8.4.2 导入静态资源至存储
 - 8.4.3 节点1接入共享存储
 - 8.4.4 节点2接入共享存储
 - 9.扩展节点带来的新问题
- 05.Nginx反向代理服务
 - 1.Nginx代理服务基本概述
 - 2.Nginx代理服务常见模式
 - 2.1) 正向代理
 - 1、客户端翻墙
 - 2、客户端提速
 - 3、客户端缓存
 - 4、客户端授权
 - 2.2) 反向代理
 - 1、路由功能
 - 2、负载均衡
 - 3、动静分离
 - 4、数据缓存
 - 2.3) 正向与反向代理区别
 - 3.Nginx代理服务支持协议
 - 4.Nginx反向代理场景实践
 - 06.Nginx七层负载均衡

- 1.Nginx负载均衡基本概述
 - 1.1 什么是负载均衡
 - 1.2 为什么需要负载均衡
 - 1.3 负载均衡与代理区别
- 2.Nginx负载均衡应用场景
 - 2.1 四层负载均衡
 - 2.2 七层负载均衡
 - 2.3 四层与七层区别
- 3.Nginx负载均衡配置场景
 - 3.1 负载均衡场景环境规划
 - 3.2 后端Web节点配置实例
 - 3.3. 前端接入Nginx负载均衡
 - 3.4. 浏览器访问测试负载效果
- 4.Nginx负载均衡调度算法
 - 4.1 轮询调度算法
 - 4.2 加权轮询调度算法
 - 4.3 ip_hash调度算法
 - 4.4 一致性hash调度算法
 - 4.5 url_hash调度算法
 - 4.6 least_conn调度算法
- 5.Nginx负载均衡后端状态
 - 5.1 max_conns限制连接数
 - 5.2 down标识关闭状态
 - 5.3 backup标识备份状态
 - 5.4 maxfails与fail_timeout
- 6.Nginx负载均衡会话共享
 - 6.1 什么是会话保持
 - 6.2 为什么需要会话保持
 - 6.3 如何实现会话保持
 - 6.4 会话保持场景演示
 - 6.4.1 配置web节点
 - 6.4.2 配置负载均衡
 - 6.4.3 配置Redis服务
 - 6.4.4 配置php连接Redis
 - 6.4.5 测试集群会话共享

01.NFS网络文件系统实战

1.NFS基本概述

1.什么是NFS?

NFS是Network File System的缩写及网络文件系统。[通常我们称NFS为共享存储]

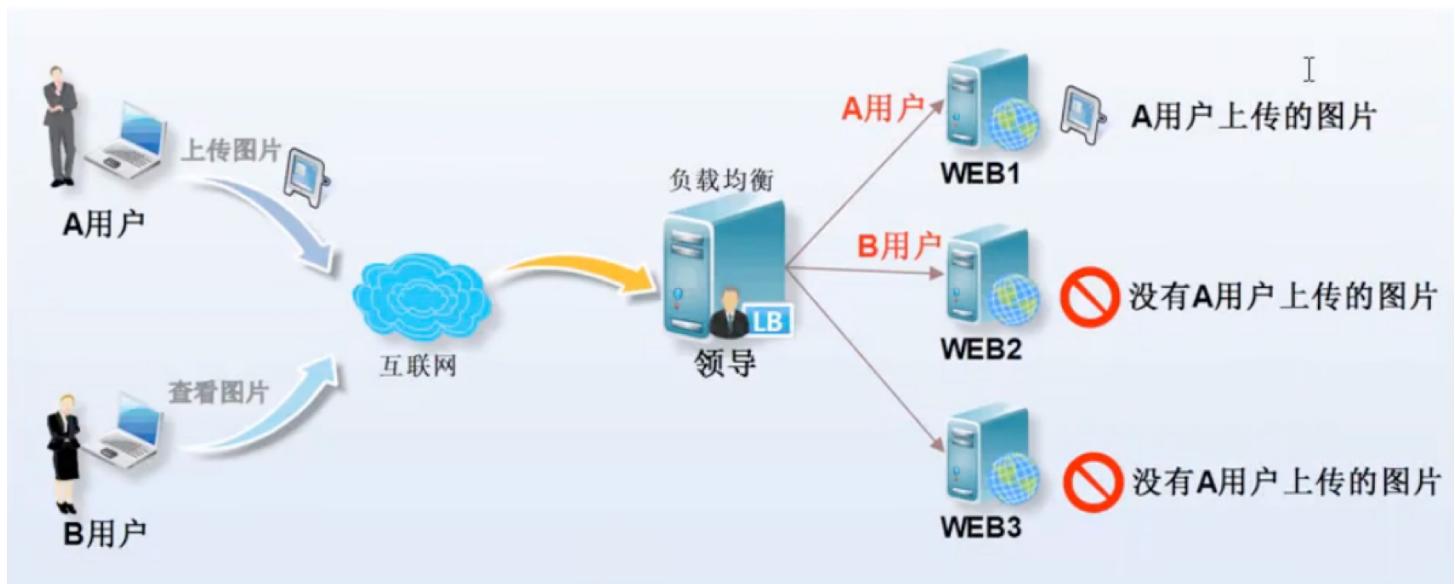
2.NFS能干什么?

NFS的主要功能是通过局域网络让不同主机系统之间可以共享目录。

3.为什么要使用NFS?

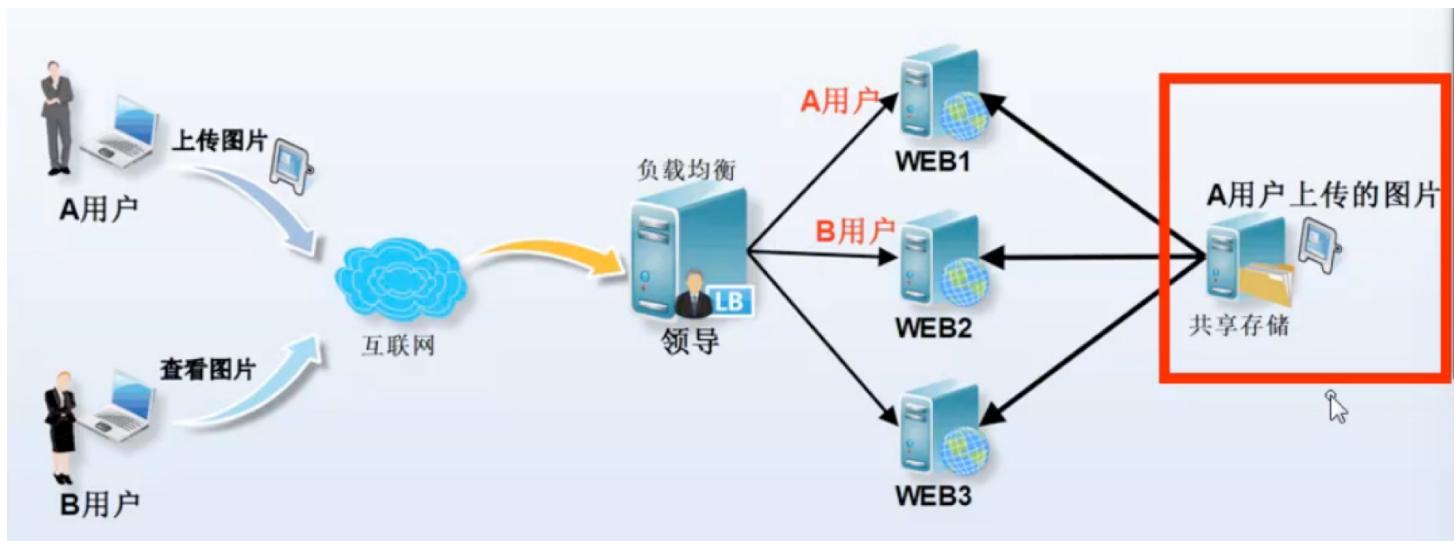
在网站集群架构中如果没有共享存储的情况如下:

- 1) A用户上传图片经过负载均衡, 负载均衡将上传请求调度至WEB1服务器上。
- 2) B用户访问A用户上传的图片, 此时B用户被负载均衡调度至WEB2上, 因为WEB2上没有这张图片, 所以B用户无法看到A用户传的图片。



在网站集群架构中如果有共享存储的情况如下:

- 1) A用户上传图片无论被负载均衡调度至WEB1还是WEB2, 最终数据都被写入至共享存储
- 2) B用户访问A用户上传图片时, 无论调度至WEB1还是WEB2, 最终都会上共享存储访问对应的文件, 这样就可以访问到资源了



4. 使用NFS共享存储能解决集群架构的什么问题?

解决多台web静态资源的共享(所有客户端都挂载服务端, 看到的数据都一样)

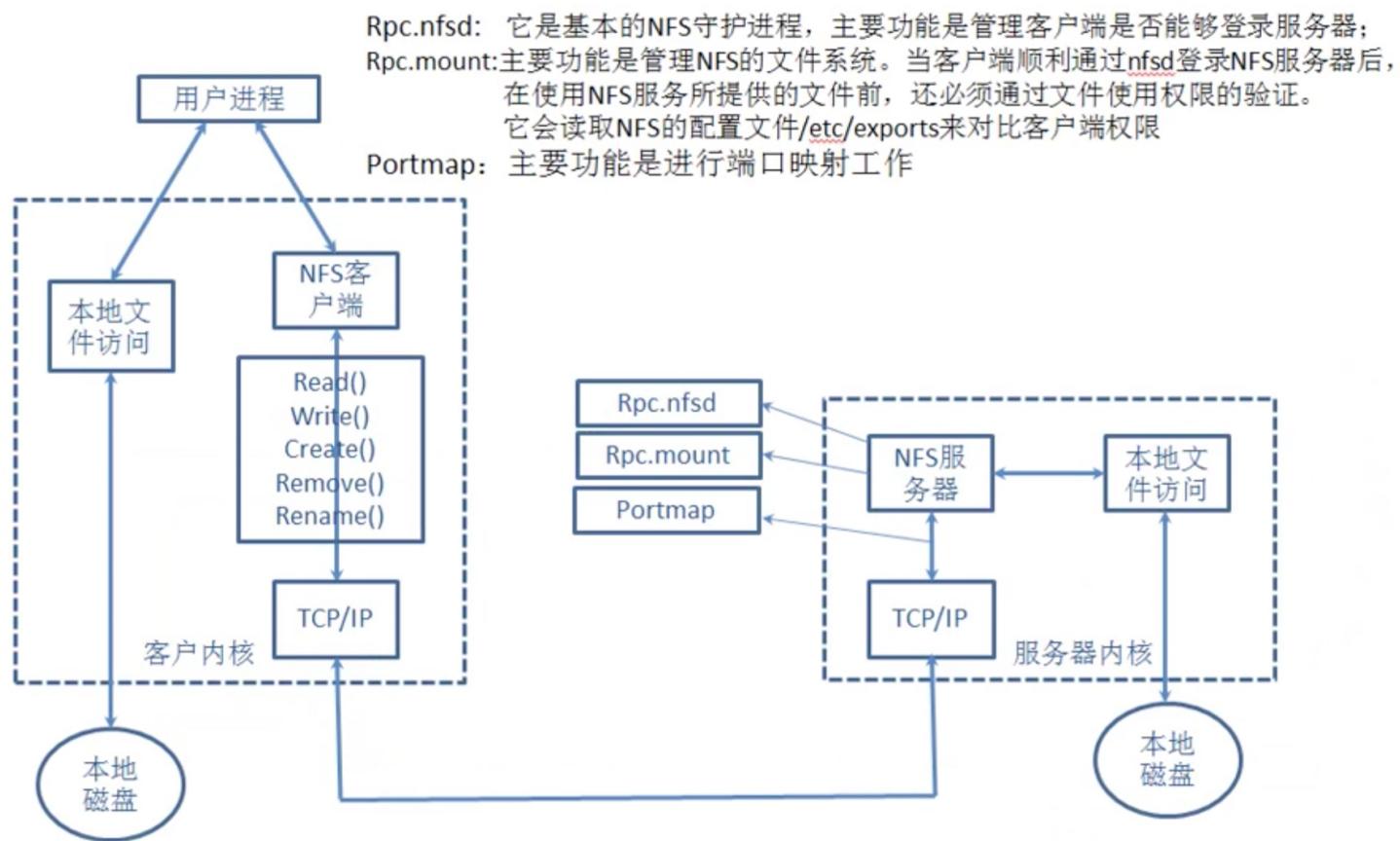
解决多台web静态资源一致性(如果客户端A删除NFS服务上的test文件, 客户端B上也会看不见test文件)

解决多台web磁盘空间的浪费

5. 企业使用NFS注意事项

1. 由于用户请求静态资源每次都需要web连接NFS服务获取, 那么势必会带来一定的网络开销、以及网络延时、所以增加NFS服务并不能给网站带来访问速度的提升。
2. 如果希望对上传的图片、附件等静资源进行加速, 建议将静态资源统一存放至NFS服务端。这样便于后期统一推送至CDN, 以此来实现资源的加速。[CDN???说啥呢? 这个是扩展知识-->别纠结-->纠结打死你]

2. NFS实现原理



本地文件操作方式

1. 当用户执行mkdir命令, BashShell无法完成该命令操作, 会将其翻译给内核。
2. Kernel内核解析完成后会驱动对应的磁盘设备, 完成创建目录的操作。

NFS实现原理(需要先了解[程序|进程|线程])

1. NFS客户端执行增、删等操作, 客户端会使用不同的函数对该操作进行封装。
2. NFS客户端会通过TCP/IP的方式传递给NFS服务端。

3.NFS服务端接收到请求后，会先调用portmap进程进行端口映射。

4.nfsd进程用于判断NFS客户端是否拥有权限连接NFS服务端。

5.Rpc.mount进程判断客户端是否有对应的权限进行验证。

6.idmap进程实现用户映射和压缩。

7.最后NFS服务端会将客户端的函数转换为本地能执行的命令，然后将命令传递至内核，由内核驱动硬件。

注意: rpc是一个远程过程调用，那么使用nfs必须有rpc服务

3.NFS服务安装

0.环境准备

服务器系统	角色	外网IP	内网IP
CentOS 7.6	NFS服务端	eth0:10.0.0.31	eth1:172.16.1.31
CentOS 7.6	NFS客户端	eth0:10.0.0.41	eth1:172.16.1.41

1.关闭防火墙

```
#1. 关闭Firewalld防火墙
```

```
[root@nfs ~]# systemctl disable firewalld  
[root@nfs ~]# systemctl stop firewalld
```

```
#2. 关闭selinux防火墙
```

```
[root@nfs ~]# sed -ri '#^SELINUX=.*' /etc/selinux/config  
[root@nfs ~]# setenforce 0
```

2.安装nfs-server服务

```
[root@nfs ~]# yum -y install nfs-utils
```

3.配置nfs服务，nfs服务程序的配置文件为 /etc/exports，需要严格按照 共享目录的路径 允许访问的NFS客户端（共享权限参数） 格式书写，定义要共享的目录与相应的权限，具体书写方式如下图所示。

xuliangwei.com			
配置语法	/data	172.16.1.0/24	(rw,sync,all_squash)
语法含义	NFS共享目录	NFS客户端地址	(参数1,参数2.....)

4. 配置场景，将nfs服务端的 /data 目录共享给 172.16.1.0/24 网段内的所有主机

- 1) 所有客户端主机都拥有读写权限
- 2) 在将数据写入到NFS服务器的硬盘中后才会结束操作，最大限度保证数据不丢失
- 3) 将所有用户映射为本地的匿名用户(nfsnobody)

#NFS客户端地址与权限之间没有空格

```
[root@nfs ~]# vim /etc/exports
/data  172.16.1.0/24(rw,sync,all_squash)
```

#在NFS服务器上建立用于NFS文件共享的目录，并设置对应权限

```
[root@nfs ~]# mkdir /data
[root@nfs ~]# chown -R nfsnobody.nfsnobody /data
```

#NFS共享目录会记录至/var/lib/nfs/etab,如果该目录不存在共享信息,请检查/etc/exports是否配置错误

4. 在使用 NFS 服务进行文件共享之前，需要使用 RPC (Remote Procedure Call 远程过程调用服务将NFS服务器的IP地址和端口号信息发送给客户端。因此，在启动NFS服务之前，需要先重启并启用rpcbind服务程序,同时都加入开机自启动

#加入开机自启

```
[root@nfs ~]# systemctl enable rpcbind nfs-server
```

#启动服务

```
[root@nfs ~]# systemctl restart rpcbind nfs-server
```

4.NFS挂载卸载

NFS客户端的配置步骤也十分简单。先使用 showmount 命令,查询 NFS 服务器的远程共享信息，其输出格式为“共享的目录名称 允许使用客户端地址”。

1. 安装客户端工具，安装nfs-utils即可，会自动启动rpcbind服务。

```
[root@nfs-client ~]# yum -y install nfs-utils
```

2. 客户端使用 showmount -e 查看远程服务器 rpc 提供的可挂载 nfs 信息

```
[root@nfs-client ~]# showmount -e 172.16.1.31
Export list for 172.16.1.31:
/data 172.16.1.0/24
```

3. 在 NFS 客户端创建一个挂载目录, 使用 mount 命令并结合 -t 参数, 指定要挂载的文件系统的类型, 并在命令后面写上服务器的 IP 地址, 以及服务器上的共享目录, 最后需要写上要挂载到本地系统(客户端)的目录。

```
[root@nfs-client ~]# mkdir /nfsdir
[root@nfs-client ~]# mount -t nfs 172.16.1.31:/data /nfsdir

#查看挂载信息(mount也可以查看)
[root@nfs-client ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3        62G  845M   58G   2% /
tmpfs            244M     0  244M   0% /dev/shm
/dev/sda1        190M   26M  155M  14% /boot
172.16.1.31:/data  62G  880M   58G   2% /nfsdir
```

4. 挂载成功后可以进行增删改操作

```
#使用客户端往nfs存储写入
[root@nfs-client ~]# echo "nfs-client" >> /mnt/test.txt

#检查nfs服务端是否存在客户端创建的新文件
[root@nfs-client ~]# cat /data/test.txt
nfs-client
```

5. 如果希望 NFS 文件共享服务能一直有效, 则需要将其写入到 fstab 文件中

```
[root@nfs-client ~]# vim /etc/fstab
172.16.1.31:/data /nfsdir nfs defaults 0 0
```

6. 如果不希望使用 NFS 共享, 可进行卸载

```
[root@nfs-client ~]# umount /nfsdir

#注意: 卸载的时候如果提示"umount.nfs: /nfsdir: device is busy"
```

```
#1. 切换至其他目录，然后在进行卸载。
```

```
#2. NFS Server宕机，强制卸载umount -lf /nfsdir
```

7. 在企业工作场景，通常情况NFS服务器共享的只是普通静态数据（图片、附件、视频），不需要执行 `suid`、`exec` 等权限，挂载的这个文件系统只能作为数据存取之用，无法执行程序，对于客户端来讲增加了安全性。例如：很多木马篡改站点文件都是由上传入口上传的程序到存储目录。然后执行的。

```
#通过mount -o指定挂载参数，禁止使用suid, exec, 增加安全性能
```

```
[root@nfs-client ~]# mount -t nfs -o nosuid,noexec,nodev 172.16.1.31:/data /mnt
```

8. 有时也需要考虑性能相关参数[可选]

```
#通过mount -o指定挂载参数，禁止更新目录及文件时间戳挂载
```

```
[root@nfs-client ~]# mount -t nfs -o noatime,nodiratime 172.16.1.31:/data /mnt
```

5. NFS配置详解

执行 `man exports` 命令，然后切换到文件结尾，可以快速查看如下样例格式：

nfs共享参数	参数作用
rw*	读写权限
ro	只读权限
root_squash	当NFS客户端以root管理员访问时，映射为NFS服务器的匿名用户(不常用)
no_root_squash	当NFS客户端以root管理员访问时，映射为NFS服务器的root管理员(不常用)
all_squash	无论NFS客户端使用什么账户访问，均映射为NFS服务器的匿名用户(常用)
no_all_squash	无论NFS客户端使用什么账户访问，都不进行压缩
sync*	同时将数据写入到内存与硬盘中，保证不丢失数据
async	优先将数据保存到内存，然后再写入硬盘；这样效率更高，但可能会丢失数据

anonuid*	配置all_squash使用,指定NFS的用户UID,必须存在系统
anongid*	配置all_squash使用,指定NFS的用户UID,必须存在系统

1. 验证ro权限实践

1) 服务端修改rw为ro参数

```
[root@nfs ~]# cat /etc/exports  
/data 172.16.1.0/24(ro,sync,all_squash)  
[root@nfs ~]# systemctl restart nfs-server
```

2) 客户端验证

```
[root@nfs-client ~]# mount -t nfs 172.16.1.31:/data /mnt  
[root@nfs-client ~]# df -h  
Filesystem      Size  Used Avail Use% Mounted on  
172.16.1.31:/data  98G  1.7G   97G   2% /mnt  
  
# 发现无法正常写入文件  
[root@backup mnt]# touch file  
touch: cannot touch 'file': Read-only file system
```

2. 验证all_squash、anonuid、anongid权限

1) NFS服务端配置

```
[root@nfs ~]# cat /etc/exports  
/data 172.16.1.0/24(rw,sync,all_squash,anonuid=666,anongid=666)
```

2) 服务端需要创建对应的用户

```
[root@nfs ~]# groupadd -g 666 www  
[root@nfs ~]# useradd -u 666 -g 666 www  
[root@nfs ~]# id www  
uid=666(www) gid=666(www) groups=666(www)
```

3) 重载nfs-server

```
[root@nfs ~]# systemctl restart nfs-server  
[root@nfs ~]# cat /var/lib/nfs/etab
```

```
/data 172.16.1.0/24(rw,sync,wdelay,hide,nocrossmnt,secure,root_squash,all_squash,no_subtree_check,secure_locks,acl,no_pnfs,anonuid=666,anongid=666,sec=sys,secure,root_squash,all_squash)
```

4) 授权共享目录为www

```
[root@nfs ~]# chown -R www.www /data/
[root@nfs ~]# ll -d /data/
drwxr-xr-x 3 www www 53 Sep 3 02:08 /data/
```

5) 客户端验证

```
[root@backup ~]# umount /mnt/
[root@backup ~]# mount -t nfs 172.16.1.31:/data /mnt
```

6) 客户端查看到的文件，身份是666

```
[root@backup ~]# ll /mnt/
drwxr-xr-x 2 666 666 6 Sep 3 02:08 rsync_dir
-rw-r--r-- 1 666 666 0 Sep 3 02:08 rsync_file
```

7) 客户端依旧能往/mnt目录下写文件

```
[root@backup mnt]# touch fff
[root@backup mnt]# mkdir 111
[root@backup mnt]# ll
drwxr-xr-x 2 666 666 6 Sep 3 03:05 111
-rw-r--r-- 1 666 666 0 Sep 3 03:05 fff
```

8) 建议：将客户端也创建一个uid为666, gid为666, 统一身份，避免后续出现权限不足的情况

```
[root@backup mnt]# groupadd -g 666 www
[root@backup mnt]# useradd -g 666 -u 666 www
[root@backup mnt]# id www
uid=666(www) gid=666(www) groups=666(www)
```

9) 最后检查文件的身份

```
[root@backup mnt]# ll /mnt/
total 4
```

```
drwxr-xr-x 2 www www 6 Sep 3 03:05 111  
-rw-r--r-- 1 www www 0 Sep 3 03:05 fff
```

6.NFS存储总结

1. NFS 存储优点

1.NFS简单易用、方便部署、数据可靠、服务稳定、满足中小企业需求。

2.NFS的数据都在文件系统之上，所有数据都是能看得见。

2. NFS 存储局限

1.存在单点故障, 如果构建高可用维护麻烦 web->nfs()->backup

2.NFS数据都是明文，并不对数据做任何校验，也没有密码验证(强烈建议内网使用)。

3. NFS 应用建议

1.生产场景应将静态数据(jpg\png\mp4\avi\css\js)尽可能放置CDN场景进行环境, 以此来减少后端存储压力

2.如果没有缓存或架构、代码等, 本身历史遗留问题太大, 在多存储也没意义

02.Nginx基础Http协议

1.Http协议介绍

1.1 什么是URL

通常我们在访问一个网站页面时, 请求到的内容通称为"资源"。而"资源"这一概念非常宽泛, 它可以是一份文档, 一张图片, 或所有其他你能够想到的格式。每个资源都由一个 `URI` 来进行标识;

比如: `http://fj.xuliangwei.com/public/tt.jpeg` 这样的资源, 我们会将该其称为 URL 地址;
百度百科解释: `URL` 简称统一资源定位符, 用来唯一地标识万维网中的某一个资源。`URL` 由协议、主机名称、端口以及文件名几部分构成。[深入理解 URL 的组成部分](#)

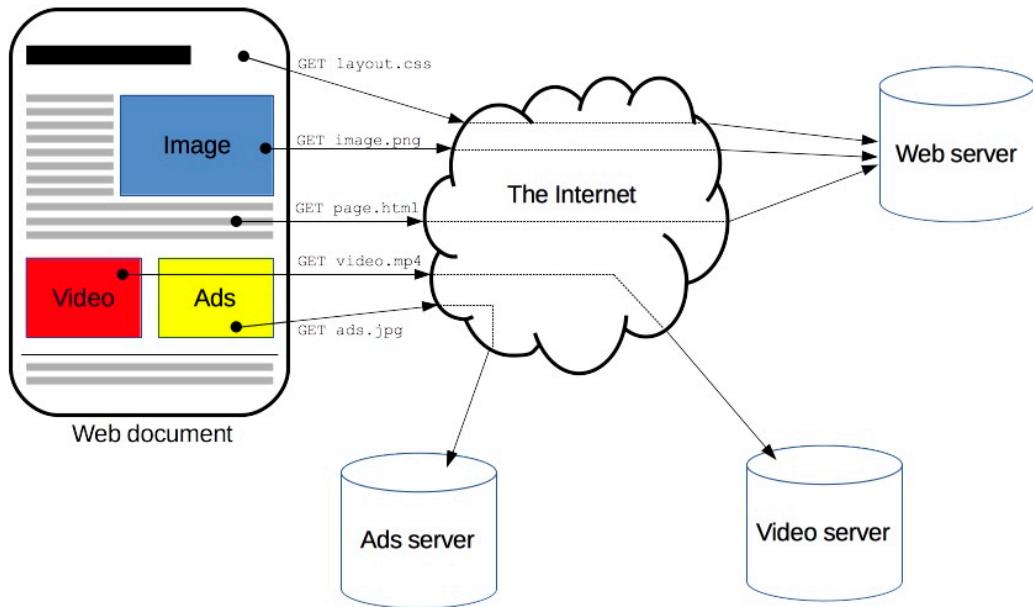
1.2 什么是HTML

Html简称Web Page, 一个完整的Html页面可能会包含很多个URL的资源。(反之: 我们也可以理解一个HTML文件是由多个不同的URL资源拼接而成的。)

1.3 什么是HTTP

`HTTP` (`Hyper Text Transfer Protocol`) 中文名为超文本传输协议。

是一种能够获取如 `HTML` 这样网络资源的通讯协议。它是在 `Web` 上进行数据交换的基础。`HTTP` 的概述参考[URL](#) 简单理解： `HTTP` 协议就是将用户请求的 `HTML` 页面从一台 `Web` 服务器传输到客户端浏览器的一种协议。



1.4 URL、HTML、HTTP之间关系

- 一个完整的 `HTML` 页面是由多个不同的 `URL` 资源组成的；而 `HTTP` 协议主要是用来传输这种 `HTML` 页面的；

2.Http工作原理

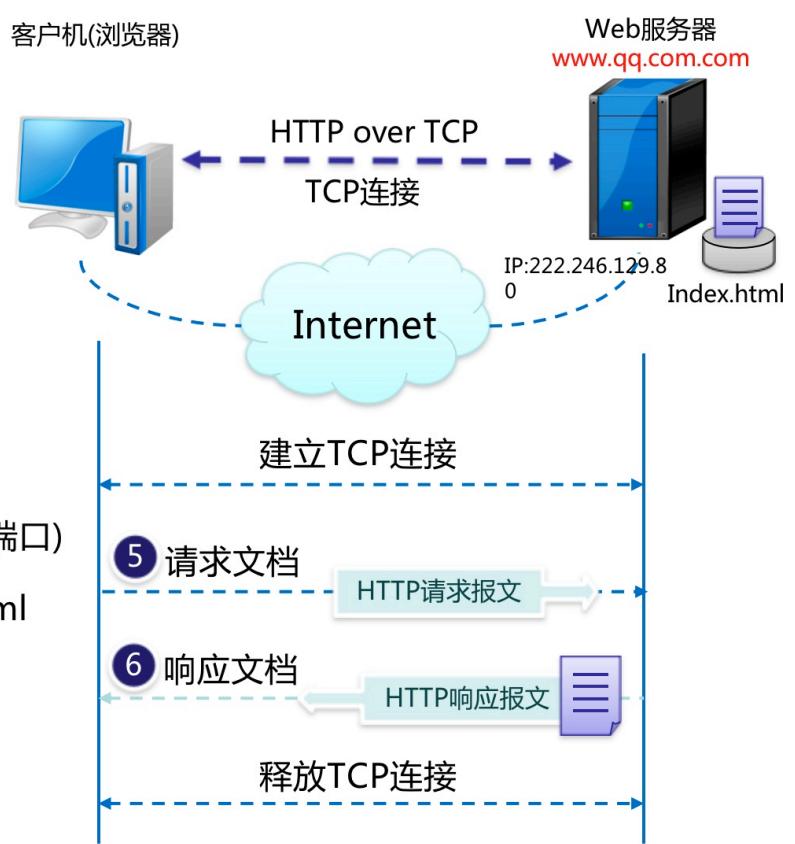
2.1 图解HTTP工作原理

我们详细的了解下HTTP的工作原理，我们到底是如何获取到服务器上的页面。

请求/响应交互模型

在用户点击URL为
<http://www.qq.com/index.html>的链接后，浏览器和Web服务器执行以下动作：

- ① 浏览器分析超链接中的URL
- ② 浏览器向DNS请求解析 www.qq.com 的IP地址
- ③ DNS将解析出的IP地址 222.246.129.80 返回浏览器
- ④ 浏览器与服务器建立TCP连接(80端口)
- ⑤ 浏览器请求文档：GET /index.html
- ⑥ 服务器给出响应，将文档 index.html 发送给浏览器
- ⑦ 释放TCP连接
- ⑧ 浏览器显示index.html中的内容



2.2 抓包分析HTTP原理

第一步：浏览器分析超链接中的URL

第二步：DNS请求：

PC向DNS服务器10.64.0.100发出DNS QUERY请求，请求fj.xuliangwei.com的A记录*

Wi-Fi: en0

No.	Time	Source	Destination	Protocol	Length	Info
77...	3.364251	192.168.1.100	10.64.0.100	DNS	77	Standard query 0xb28f A zhibo.sina.com.cn
78...	3.374387	10.64.0.100	192.168.1.100	DNS	293	Standard query response 0xb28f A zhibo.sina.com.cn
79...	3.377708	192.168.1.100	10.64.0.100	DNS	77	Standard query 0x37e2 A tousu.sina.com.cn
79...	3.385439	10.64.0.100	192.168.1.100	DNS	260	Standard query response 0x37e2 A tousu.sina.com.cn
89...	6.129362	192.168.1.100	10.64.0.100	DNS	77	Standard query 0x1fae A fj.xuliangwei.com
89...	6.144558	10.64.0.100	192.168.1.100	DNS	101	Standard query response 0x1fae A fj.xuliangwei.com

```

▶ Frame 8974: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface 0
▶ Ethernet II, Src: Apple_90:97:72 (38:f9:d3:90:97:72), Dst: Tp-LinkT_b7:f0:36 (bc:46:99:b7:f0:36)
▶ Internet Protocol Version 4, Src: 192.168.1.100, Dst: 10.64.0.100
▶ User Datagram Protocol, Src Port: 53708, Dst Port: 53
▼ Domain Name System (query)
  Transaction ID: 0x1fae
  ▶ Flags: 0x0100 Standard query
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 0
  ▶ Queries
    ▶ fj.xuliangwei.com: type A, class IN
    [Response In: 8975]

```

```

0010 00 3f ef f0 00 00 ff 11 ff 0c c0 a8 01 64 0a 40 .?.....d@
0020 00 64 d1 cc 00 35 00 2b 4b c8 1f ae 01 00 00 01 ..d...5+K.....
0030 00 00 00 00 00 02 66 6a 0a 78 75 6c 69 61 6e .....f j.xulian
0040 67 77 65 69 03 63 6f 6d 00 00 01 00 01 gwei.com .....

```

Text item (text), 23 bytes

Packets: 9008 · Displayed: 82 (0.9%) · Dropped: 0 (0.0%) · Profile: Default

第三步：DNS回复

DNS服务器10.64.0.100回复DNS response, 解析出fj.xuliangwei.com域名对应的一条A记录
39.104.16.126

Wi-Fi: en0

No.	Time	Source	Destination	Protocol	Length	Info
77...	3.364251	192.168.1.100	10.64.0.100	DNS	77	Standard query 0xb28f A zhibo.sina.com.cn
78...	3.374387	10.64.0.100	192.168.1.100	DNS	293	Standard query response 0xb28f A zhibo.sina.com.cn
79...	3.377708	192.168.1.100	10.64.0.100	DNS	77	Standard query 0x37e2 A tousu.sina.com.cn
79...	3.385439	10.64.0.100	192.168.1.100	DNS	260	Standard query response 0x37e2 A tousu.sina.com.cn
89...	6.129362	192.168.1.100	10.64.0.100	DNS	77	Standard query 0x1fae A fj.xuliangwei.com
89...	6.144558	10.64.0.100	192.168.1.100	DNS	101	Standard query response 0x1fae A fj.xuliangwei.com

```

▶ Frame 8975: 101 bytes on wire (808 bits), 101 bytes captured (808 bits) on interface 0
▶ Ethernet II, Src: Tp-LinkT_b7:f0:36 (bc:46:99:b7:f0:36), Dst: Apple_90:97:72 (38:f9:d3:90:97:72)
▶ Internet Protocol Version 4, Src: 10.64.0.100, Dst: 192.168.1.100
▶ User Datagram Protocol, Src Port: 53, Dst Port: 53708
▼ Domain Name System (response)
  Transaction ID: 0x1fae
  ▶ Flags: 0x8180 Standard query response, No error
  Questions: 1
  Answer RRs: 1
  Authority RRs: 0
  Additional RRs: 0
  ▶ Queries
  ▶ Answers
    ▶ fj.xuliangwei.com: type A, class IN, addr 39.104.16.126
    [Request In: 8974]
    [Time: 0.015196000 seconds]

```

```

0030 00 01 00 00 00 00 02 66 6a 0a 78 75 6c 69 61 6e .....f j.xulian
0040 67 77 65 69 03 63 6f 6d 00 00 01 00 01 c0 0c 00 gwei.com .....
0050 01 00 01 00 00 02 58 00 04 27 68 10 7e 3a 6f cc .....X.'h~:o
0060 09 e6 cf 99 47 .....G

```

Text item (text), 16 bytes

Packets: 9008 · Displayed: 82 (0.9%) · Dropped: 0 (0.0%) · Profile: Default

第四步：建立TCP连接

PC向DNS解析fj.xuliangwei.com地址发起tcp三次握手

Source	Destination	Protocol	Length	Info
192.168.1.100	39.104.16.126	TCP	78	49796 → 80 [SYN] Seq=0 Win=65535 Len=0
39.104.16.126	192.168.1.100	TCP	82	80 → 49796 [SYN, ACK] Seq=0 Ack=1 Win=2
192.168.1.100	39.104.16.126	TCP	66	49796 → 80 [ACK] Seq=1 Ack=1 Win=131328

第五步：Http请求

PC向 fj.xuliangwei.com 服务器发出GET请求，请求主页

The screenshot shows the Wireshark interface with the 'tcp' protocol selected. The packet list pane displays four packets: three TCP SYN/ACK/ACK handshake packets and one HTTP GET request packet (packet 8979). The details pane shows the HTTP request headers, including 'Host: fj.xuliangwei.com', 'Upgrade-Insecure-Requests: 1', and 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'. The bytes pane at the bottom shows the raw hex and ASCII data of the request.

Transmission Control Protocol, Src Port: 49796, Dst Port: 80, Seq: 1, Ack: 1, Len: 359

Hypertext Transfer Protocol

GET / HTTP/1.1\r\n

[Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]

Request Method: GET

Request URI: /

Request Version: HTTP/1.1

Host: fj.xuliangwei.com\r\n

Upgrade-Insecure-Requests: 1\r\n

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12

Accept-Language: zh-cn\r\n

Accept-Encoding: gzip, deflate\r\n

Connection: keep-alive\r\n

\r\n

[Full request URI: http://fj.xuliangwei.com/]

[HTTP request 1/1]

0040 02 3e 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31 ->GET / HTTP/1.1

0050 0d 0a 48 6f 73 74 3a 20 66 6a 2e 78 75 6c 69 61 ..Host: fj.xulia

0060 6e 67 77 65 69 2e 63 6f 6d 0d 0a 55 70 67 72 61 ngwei.co m..Upgra

0070 64 65 2d 49 6e 73 65 63 75 72 65 2d 52 65 71 75 de-Insec ure-Requ

Text item (text), 16 bytes

Packets: 9008 · Displayed: 8921 (99.0%) · Dropped: 0 (0.0%) · Profile: Default

第六步：Http响应

fj.xuliangwei.com 服务器回应HTTP/1.1 200 OK，返回主页数据包



```
GET / HTTP/1.1
Host: fj.xuliangwei.com
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/605.1.15 (KHTML,
like Gecko) Version/12.0.3 Safari/605.1.15
Accept-Language: zh-cn
Accept-Encoding: gzip, deflate
Connection: keep-alive

HTTP/1.1 200 OK
Server: nginx
Date: Sat, 25 May 2019 10:01:43 GMT
Content-Type: text/html; charset=utf8,gbk
Transfer-Encoding: chunked
Connection: keep-alive
Vary: Accept-Encoding
Content-Encoding: gzip

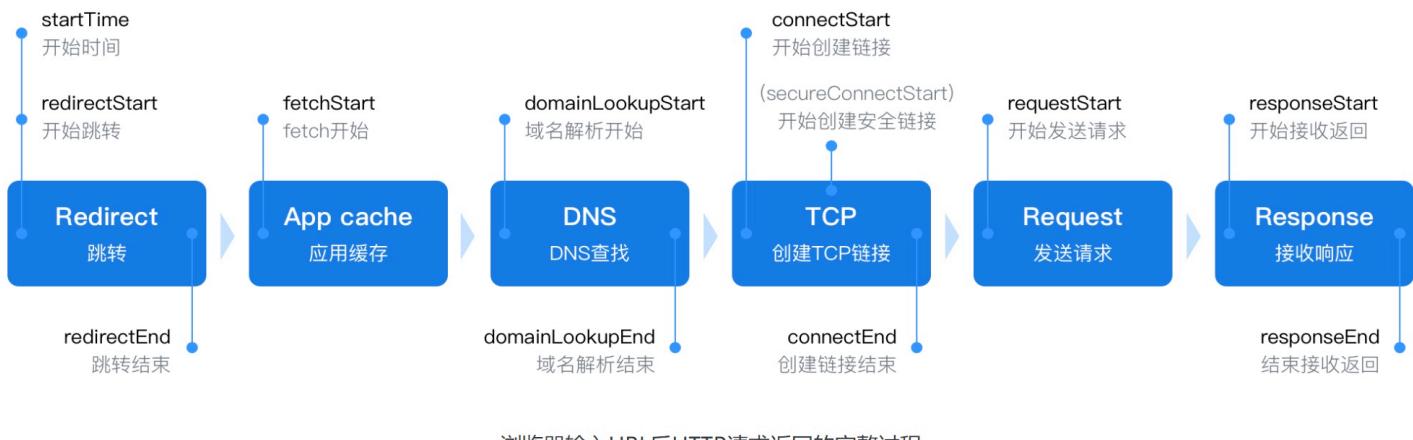
b7
.....P... ....{...D.....F..`....5i....3;Lf...U....]w
..V.g.....~.B.i.q..5..Y.+>.!h..r%...W8z...x.W<..+*.u.....%j~D.m8.....
!OL..C9...cr.N....?9...8.. E.Y!.&..y._.X.[...
0
```

第七步：连接断开

完成数据交互过程，四次挥手断开连接

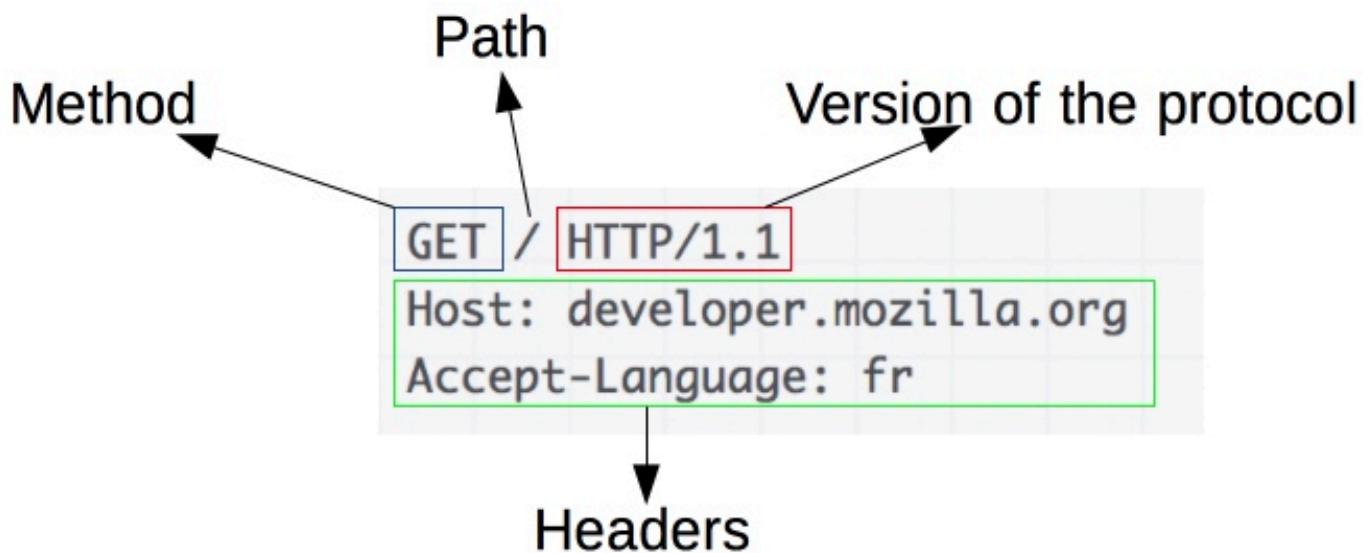
2.3 HTTP工作原理总结

- 整个用户访问网站过程就是DNS-TCP-HTTP（面试必须的协议）



3.Http请求Request

- HTTP 请求的一个例子：



3.1 请求Method

- 客户端向服务端发送请求时，会根据不同的资源发送不同的请求方法 Method：
 - GET：用于获取URI对应的资源；（比如看朋友圈）
 - POST：用于提交请求，可以更新或者创建资源，是非幂等的；（发布朋友圈）
 - PUT：用于向指定的URI传送更新资源，是幂等的；（更新朋友圈）
 - DELETE：用于向指定的URI删除资源；（比如删朋友圈）
 - HEAD：用于检查
- 一般创建对象时用 POST，更新对象时用 PUT；
 - PUT 是幂等的， POST 是非幂等的；
 - 幂等：对于相同的输入，每次得到的结果都是相等的；

3.2 请求Header

```

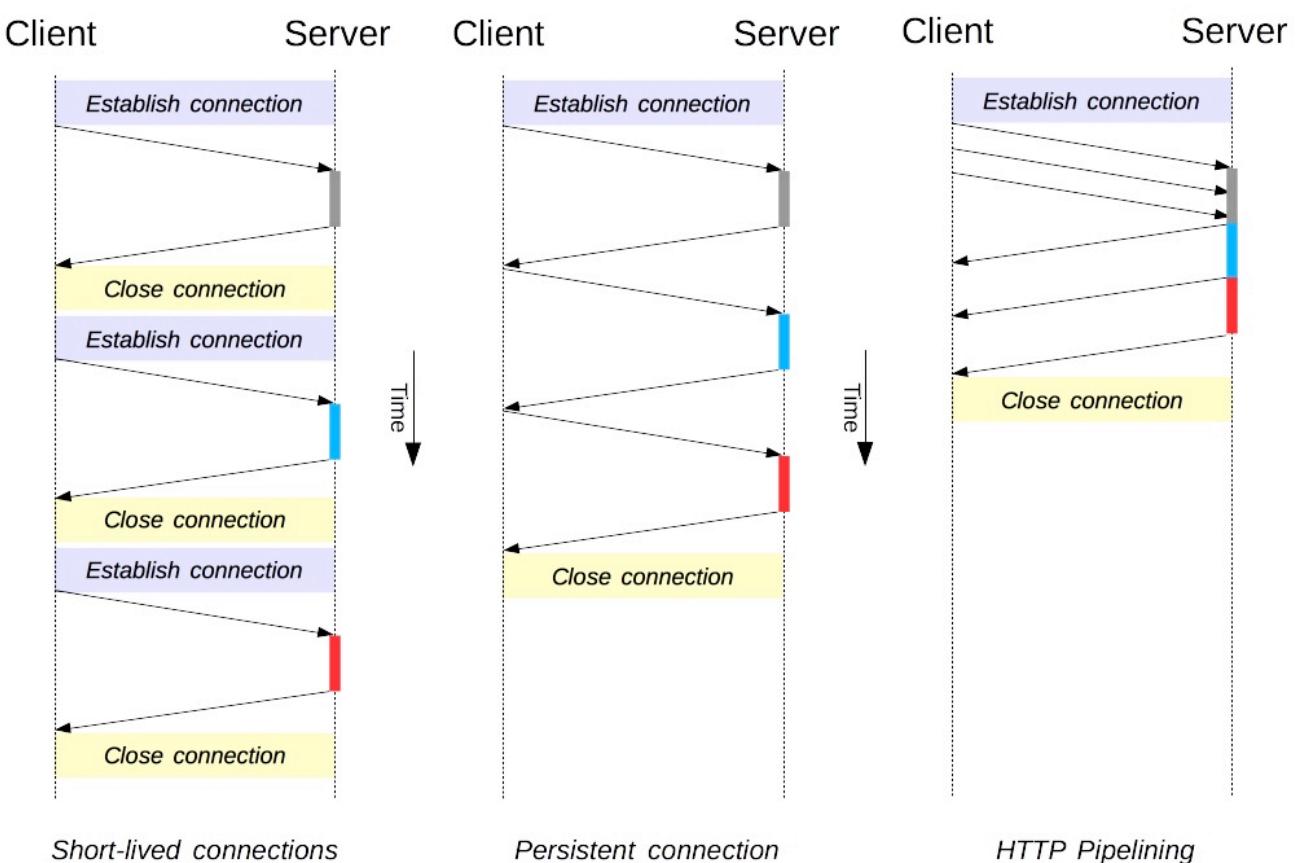
:authority: www.oldxu.com
:method: GET
:path: /
:scheme: https
Accept: text/html, # 请求的类型
Accept-Encoding: gzip, deflate # 是否进行压缩
Accept-Language: zh-CN,zh;q=0.9 # 请求的语言
Cache-Control: max-age=0 # 缓存
Connection: keep-alive # TCP长连接
Host: www.oldxu.com # 请求的域名
If-Modified-Since: Fri, 04 May 2018 08:13:44 GMT # 修改的时间
User-Agent: Mozilla/5.0 # 请求浏览器的工具
"== 请求一个空行 =="

```

"==== 请求内容主体 ==="

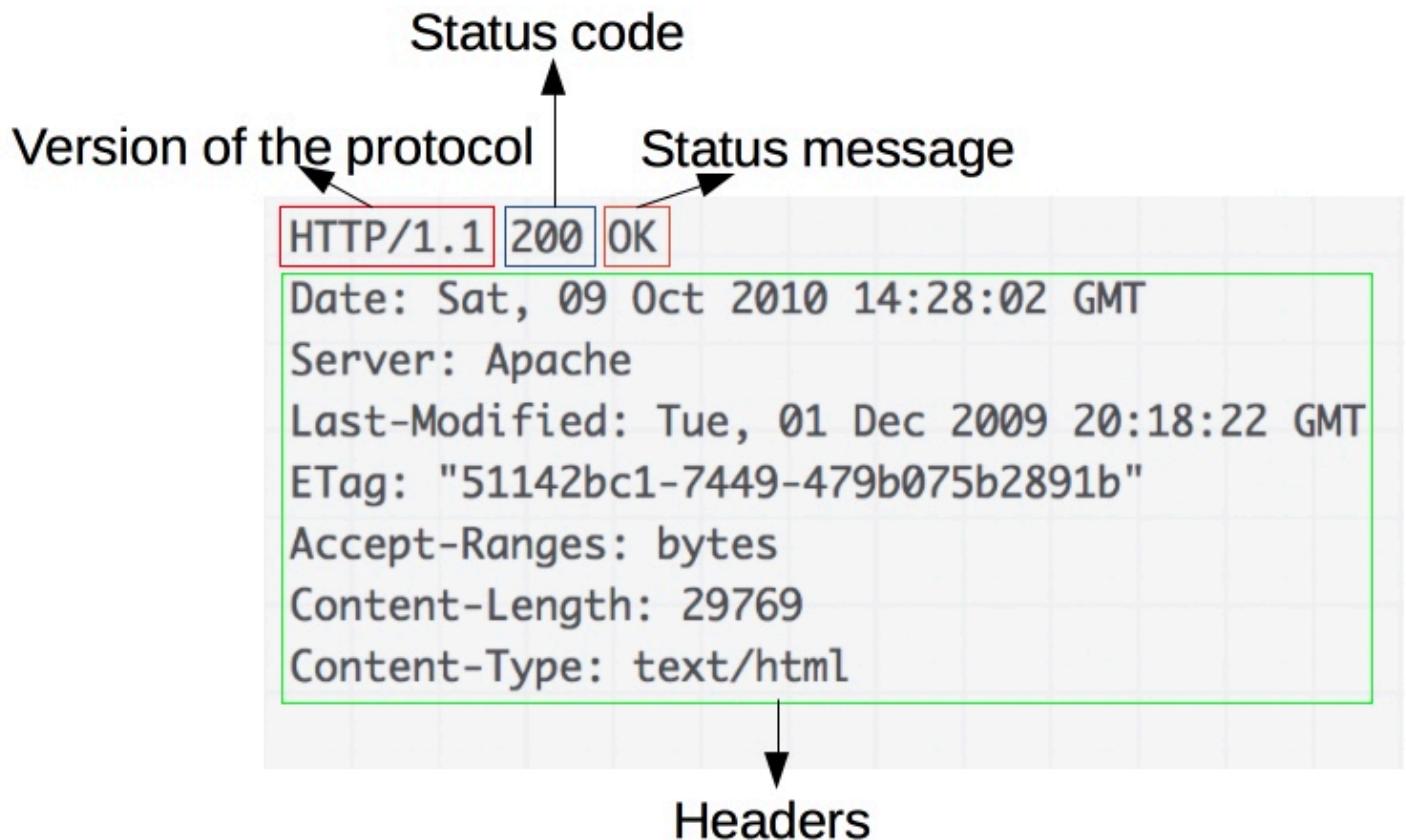
3.3 请求Connection

- `Http` 请求中的长连接与短连接是什么：
 - `http1.0` 协议使用的是短连接：建立一次 `tcp` 的连接，发起一次 `http` 的请求，结束，`tcp` 断开。
 - `http1.1` 协议使用的是长连接：建立一次 `tcp` 的连接，发起多次 `http` 的请求，结束，`tcp` 断开。
 - [HTTP协议版本参考URL、HTTP1.1与HTTP2.0速度对比](#)



4.Http响应Response

- `HTTP` 响应的一个例子：



4.1 响应Header

```

#3. 服务端响应的头部信息
HTTP/1.1 200 OK                                # 返回服务器的http协议, 状态码
Date: Fri, 14 Sep 2018 09:14:28 GMT              # 返回服务器的时间
Server: Apache/2.4.6                            # 返回服务器使用的软件Apache
Connection: Keep-Alive                          # TCP长连接
Keep-Alive: timeout=5, max=100                  # 长连接的超时时间
"--- 返回一个空行 ---"
"--- 返回内容主体 ---"

```

4.2 响应Status

http 响应状态码 Status-Code 以3位数字组成，用来标识该请求是否成功，比如是正常还是错误等，HTTP/1.1 中状态码可以分为五大类。

状态码	说明
1xx	信息，服务器收到请求，需要请求者继续执行操作
2xx	成功，操作被成功接收并处理
3xx	重定向，需要进一步的操作以完成请求

4xx	客户端错误，请求包含语法错误或无法完成请求
5xx	服务器错误，服务器在处理请求的过程中发生了错误

4.3 响应Code

- 以下是常见状态码

状态码	说明
200	表示成功客户端成功接收到了服务端返回的数据，这是最常见的状态码
206	客户端发完请求后，服务端只是返回了部分数据，就会出现该状态码，例如当下载一个很大的文件时，在没有下载完成前就会出现该状态码
301	永久重定向(redirect)
302	临时重定向(redirect)
400	客户端请求语法错误，服务端无法理解
401	服务端开启了用户认证，而客户端没有提供正确的验证信息
403	服务端不允许客户端访问，或者没有找到默认返回页面
404	客户端请求的资源不存在
413	客户端向服务端上传一个比较大的文件，并且文件大小超过了服务端的限制
500	服务端出现了内部错误，需要进行人为排查故障
502	服务器充当代理角色时，后端被代理的服务器不可用或者没有正常回应
503	服务当前不可用，由于超载或系统维护，服务器暂时的无法处理客户端请求
504	服务器充当代理角色时，后端的服务端没有按时返回数据，超时了

03.Nginx Web快速入门

1.Nginx基本简介

1.1 什么是Nginx

Nginx是一个开源且高性能、可靠的Http Web服务、代理服务。

开源，体现在直接获取Nginx的源代码。

高性能，体现在支持海量的并发。

高可靠，体现在服务稳定。

1.2 为什么选择Nginx

1.2.1 高性能、高并发

通常正常情况下，单次请求会得到更快的响应。另一方面在高峰期（如有数以万计的并发请求），Nginx可以比其他Web服务器更快地响应请求。

1.2.2 高扩展性

Nginx功能模块化。Nginx官方提供了非常多的优秀模块提供使用。这些模块都可以实现快速增加和减少。

1.2.3 高可靠性

所谓高可靠性，是指Nginx可以在服务器上持续不间断的运行，而很多web服务器往往运行几周或几个月就需要进行一次重启。对于nginx这样的一个高并发、高性能的反向代理服务器而言，他往往运行网站架构的最前端，那么此时如果我们企业如果想提供9999、99999，对于nginx持续运行能够宕机的时间，一年可能只能以秒来计算，所以在这样的一个角色中，nginx的高可靠性为我们提供了非常好的保证。^{*}

1.2.4 热部署

热部署是指在不停服务的情况下升级nginx，这个功能非常的重要。对于普通的服务，只需要kill掉进程再启动，但对于Nginx而言，如果Nginx有很多的客户端连接，那么kill掉Nginx。Nginx会像客户端发送tcp reset复位包，但很多客户端无法很好的理解reset包，就会造成异常。由于Nginx的master管理进程与worker工作进程的分离设计，使得Nginx能够在7×24小时不间断服务的前提下，升级Nginx的可执行文件。当然，也支持不停止服务更新配置、更换日志文件等功能。

1.2.5 应用广泛

首先Nginx技术成熟，具备企业最常使用的功能，如代理、代理缓存、负载均衡、静态资源、动静分离、Https、Inmp、Inmt等等

其次使用Nginx统一技术栈，降低维护成本，同时降低技术更新成本。

1.2.6 网络模型

6.Nginx使用Epoll网络模型，而常听到Apache采用的是Select网络模型。

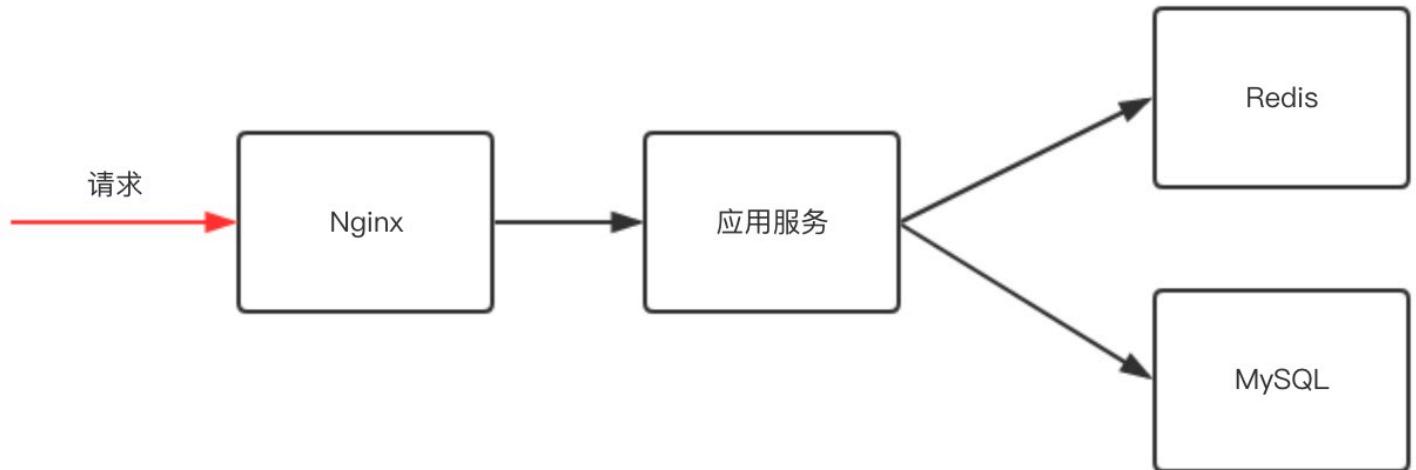
Select: 当用户发起一次请求，select模型就会进行一次遍历扫描，从而导致性能低下。

Epoll: 当用户发起请求，epoll模型会直接进行处理，效率高效。

1.3 Nginx应用场景

Nginx的主要使用场景我归纳为三个，分为是静态资源服务、代理资源服务、安全服务，场景详细介绍如下

如下图是一个网站的基本架构，首先用户请求先到达nginx，然后再到tomcat或php这样的应用服务器，然后应用服务器再去访问redis、mysql这样的数据库，提供基本的数据功能。



那么这里有一个问题，我们的程序代码要求开发效率高，所以他的运行效率是很低的，或者说它并发是受限，所以我们需要很多应用服务组成一个集群，为更多用户提供访问。

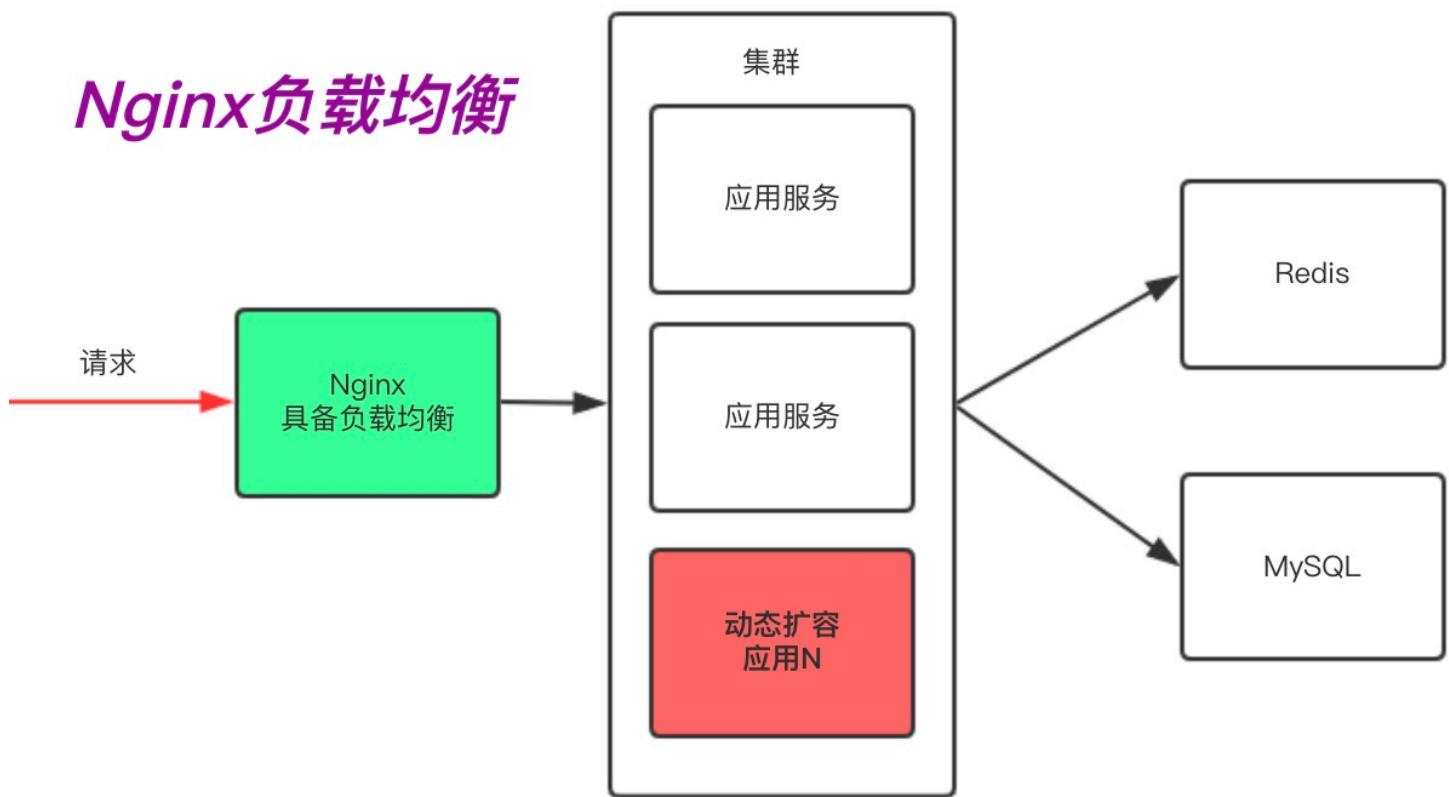
而应用服务一但构成集群，则需要我们的nginx具有反向代理功能，这样可以将动态请求传倒给集群服务。

但很多应用构成集群，那么一定会带来两个需求。

- 1、应用服务器需要动态扩展。
- 2、有些服务出问题需要做容灾。

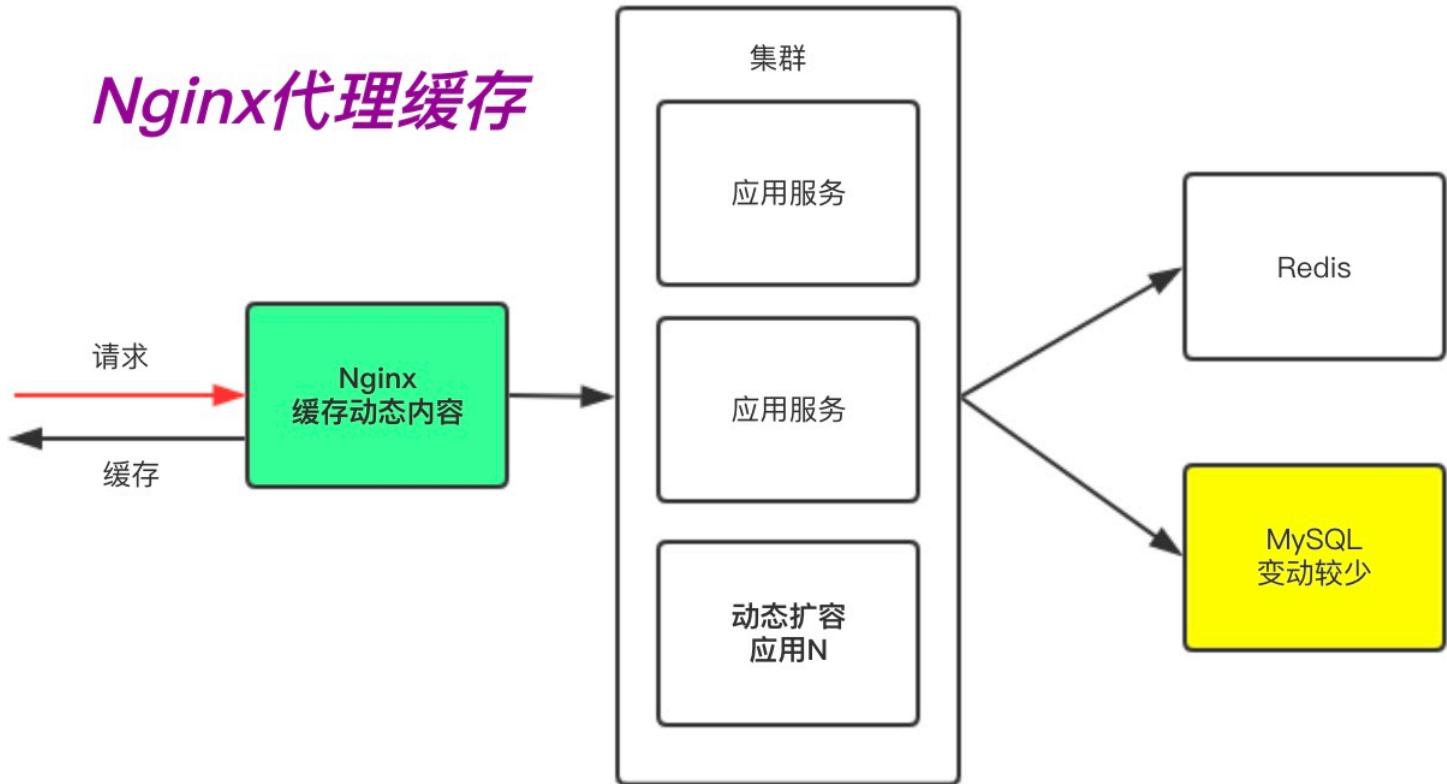
那么我们的反向代理必须具备负载均衡功能。

Nginx负载均衡



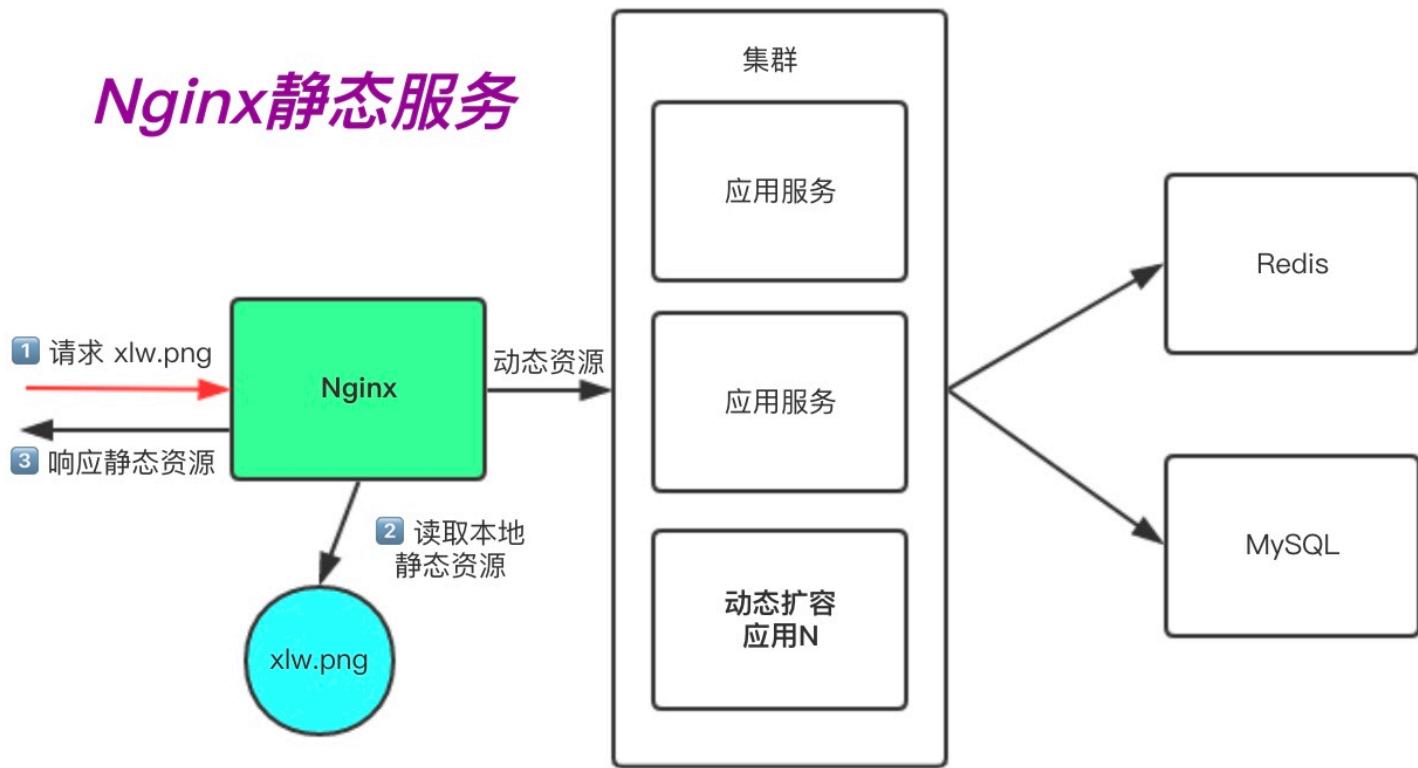
其次，随着我们网络链路的增长，用户体验到的时延则会增加。如果我们能把一段时间内不会发生变化的“动态”内容，缓存在Nginx，由Nginx直接向用户提供访问，那么这样用户请求的时延就会减少很多，所以在这里反向代理会演生出另外一个功能“缓存”，因为它能加速我们的访问。

Nginx代理缓存



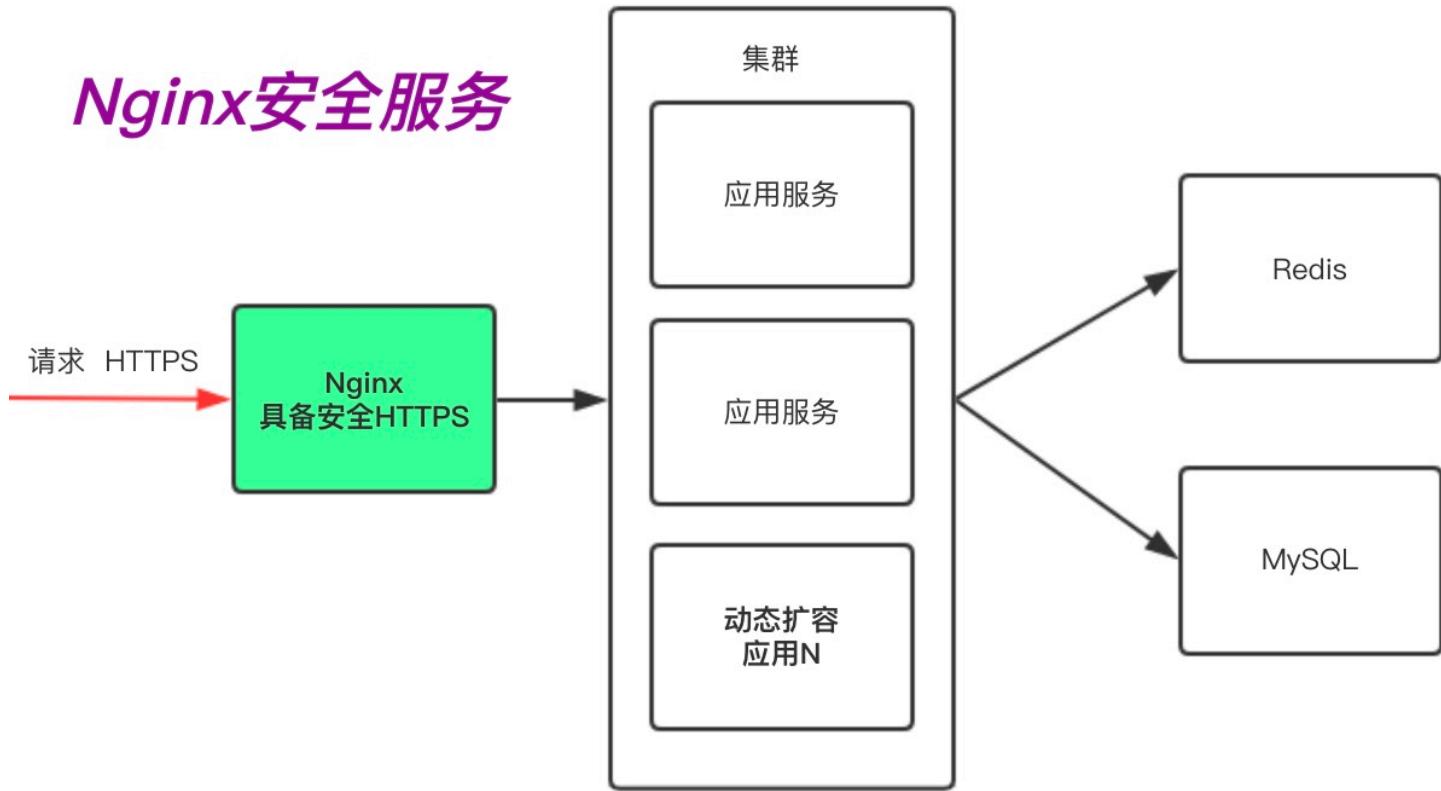
在很多时候我们访问`docs`、`pdf`、`mp4`、`png`等这样的静态资源时，是没有必要将这些请求通过Nginx交给后端的应用服务，我们只需要通过Nginx直接处理“静态资源”即可。这是Nginx的静态资源功能。

Nginx静态服务



当我们使用http网站时，可能会遭到劫持和篡改，如果使用https安全通讯协议，那么数据在传输过程中是加密的，从而能有效的避免黑客窃取或者篡改数据信息，同时也能避免网站在传输过程中的信息泄露。大大的提升我们网站安全。

Nginx安全服务



PS: Nginx的功能远不止上面列出一角，随着我们后面课程的不断深入，会发现Nginx的功能非常的强大。

1.4 Nginx组成部分

在这里我们将Nginx的组成架构比喻为一辆汽车：

这个汽车提供了基本的驾驶功能，但是还需要一个驾驶员控制这辆汽车开往哪个方向，同时该汽车行驶过的地方还会形成GPS轨迹，如果汽车在行驶的过程中出现了任何问题，我们需要一个黑匣子，分析是汽车本身的问题，还是驾驶人员的操作出现了问题。

Nginx二进制可执行文件

由各模块源码编译出的一个文件



Nginx配置文件

控制Nginx的行为

access.log访问日志

记录每一条 http 请求信息

error.log日志

定位问题

1.第一个组成部分Nginx二进制可执行文件：它是Nginx本身框架以及相关模块等构建的一个二进制文件，这个文件就相当于汽车本身，所有的功能都由它提供。

2.第二个组成部分Nginx.conf文件：它相当于驾驶人员，虽然二进制可执行文件已经提供了许多的功能，但是这些功能究竟有没有开启，或者开启后定义怎样的行为去处理请求，都是由nginx.conf这个文件决定的，所以他就相当于这个汽车的驾驶员，控制这个汽车的行为。

3.第三个组成部分access.log：它相当于这辆汽车经过所有地方形成的GPS轨迹，access.log会记录Nginx处理过的每一条HTTP的请求信息、响应信息。

4.第四个组成部分error.log：它相当于黑匣子，当出现了一些不可预期的问题时，可以通过error.log将问题定位出来。

Nginx组成部分小结：

Nginx的组成部分是相辅相成，Nginx二进制可执行文件和Nginx.conf文件，它定义了Nginx处理请求的方式。

而如果我们想对nginx服务做一些web的运营和运维，需要对access.log做进一步分析。

而如果出现了任何未知的错误，或者预期的行为不一致时，应该通过error.log去定位根本性的问题。

2.Nginx快速安装

安装Nginx软件的方式有很多种，分为如下几种

1. 源码编译=>Nginx (1.版本随意 2.安装复杂 3.升级繁琐)
- 2.epel仓库=>Nginx (1.版本较低 2.安装简单 3.配置不易读)
3. 官方仓库=>Nginx (1.版本较新 2.安装简单 3.配置易读，推荐)

1. 安装Nginx软件所需依赖包

```
[root@web ~]# yum install -y gcc gcc-c++ autoconf pcre pcre-devel make automake httpd-tools
```

2. 配置nginx官方yum源

```
[nginx-stable]
name=nginx stable repo
baseurl=http://nginx.org/packages/centos/$releasever/$basearch/
gpgcheck=1
enabled=1
gpgkey=https://nginx.org/keys/nginx_signing.key
```

3. 安装Nginx服务，启动并加入开机自启

```
[root@web ~]# yum install nginx -y
[root@web ~]# systemctl enable nginx
[root@web ~]# systemctl start nginx
```

4. 通过浏览器访问该服务器ip地址获取资源



5. 检查Nginx软件版本以及编译参数

```
[root@web ~]# nginx -v
nginx version: nginx/1.16.0
[root@web ~]# nginx -V
```

6.为了让大家更清晰的了解Nginx软件的全貌，可使用`rpm -ql nginx`查看整体的目录结构及对应的功能，如下表格整理了Nginx比较重要的配置文件

1.Nginx主配置文件

路径	类型	作用
/etc/nginx/nginx.conf	配置文件	nginx主配置文件
/etc/nginx/conf.d/default.conf	配置文件	默认网站配置文件

2.Nginx代理相关参数文件

路径	类型	作用
/etc/nginx/fastcgi_params	配置文件	Fastcgi代理配置文件
/etc/nginx/scgi_params	配置文件	scgi代理配置文件
/etc/nginx/uwsgi_params	配置文件	uwsgi代理配置文件

3.Nginx编码相关配置文件

路径	类型	作用
/etc/nginx/win-utf	配置文件	Nginx编码转换映射文件
/etc/nginx/koi-utf	配置文件	Nginx编码转换映射文件
/etc/nginx/koi-win	配置文件	Nginx编码转换映射文件
/etc/nginx/mime.types	配置文件	Content-Type与扩展名

4.Nginx管理相关命令

路径	类型	作用
/usr/sbin/nginx	命令	Nginx命令行管理终端工具
/usr/sbin/nginx-debug	命令	Nginx命令行与终端调试工具

4.Nginx日志相关目录与文件

路径	类型	作用

/var/log/nginx	目录	Nginx默认存放日志目录
/etc/logrotate.d/nginx	配置文件	Nginx默认的日志切割

3.Nginx基本配置

Nginx主配置文件 `/etc/nginx/nginx.conf` 是一个纯文本类型的文件，整个配置文件是以区块的形式组织的。一般，每个区块以一对大括号 {} 来表示开始与结束。

Nginx主配置文件整体分为三块进行学习，分别是CoreModule(核心模块)，EventModule(事件驱动模块)，HttpCoreModule(http内核模块)

CoreModule 核心模块

```
user www;                                #Nginx进程所使用的用户
worker_processes 1;                      #Nginx运行的work进程数量(建议与CPU数量一致或auto)
error_log /log/nginx/error.log;          #Nginx错误日志存放路径
pid /var/run/nginx.pid;                  #Nginx服务运行后产生的pid进程号
```

events 事件模块

```
events {
    worker_connections 25535;  #每个worker进程支持的最大连接数
    use epoll;                #事件驱动模型, epoll默认
}
```

http 内核模块

```
http {  #http层开始
...
#使用Server配置网站， 每个Server{}代表一个网站(简称虚拟主机)
'server' {
    listen      80;          #监听端口， 默认80
    server_name oldxu.com;   #提供的域名
    access_log  access.log;  #该网站的访问日志
    #控制网站访问的路径
    'location' / {
        root   /usr/share/nginx/html;  #存放网站源代码的位置
        index  index.html index.htm;  #默认返回网站的文件
    }
}
...
#第二个虚拟主机配置
'server' {
```

```
...
}

include /etc/nginx/conf.d/*.conf; #包含/etc/nginx/conf.d/目录下所有以.conf结尾的文件
#include作用是：简化主配置文件写太多造成臃肿，这样会让整体的配置文件更加的清晰。

} #http层结束
```

PS: Nginx中的http、server、location之间的关系是？

http 标签主要用来解决用户的请求与响应。

server 标签主要用来响应具体的某一个网站。

location 标签主要用于匹配网站具体URL路径。

http{} 层下允许有多个Server{}, 一个Server{} 下又允许有多个location{}

4.Nginx搭建网站

1.新增 nginx 配置文件

```
[root@web01 ~]# cat /etc/nginx/conf.d/game.oldxu.com.conf
server {
    listen 80;
    server_name game.oldxu.com;

    location / {
        root /code;
        index index.html;
    }
}
```

2.放置游戏源代码文件至 nginx 配置文件 root 指定的目录

```
[root@web01 conf.d]# mkdir /code && cd /code
[root@web01 code]# wget http://fj.xuliangwei.com/public/html5.zip
[root@web01 code]# unzip html5.zip
[root@web01 code]# ls
ceshi  game  html5.zip  img  index.html  readme.txt
```

3.检查 nginx 的语法是否存在错误

```
[root@web01 code]# nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
```

```
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

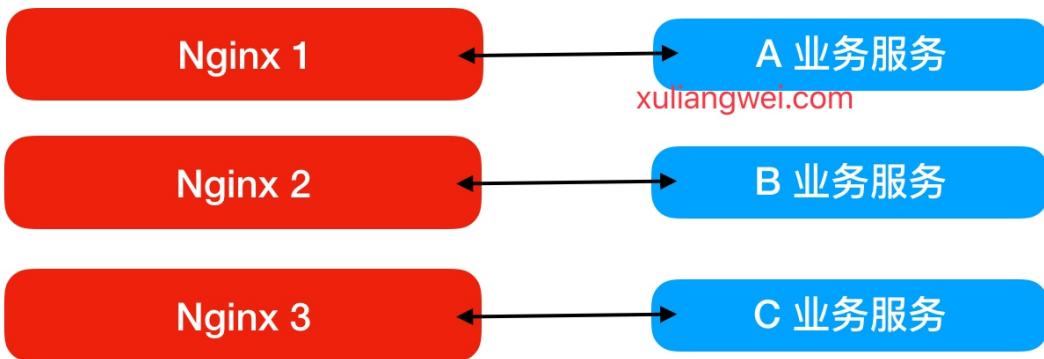
4. 重载 Nginx [reload|restart] 服务

```
[root@web01 code]# systemctl reload nginx
```

5.Nginx虚拟主机

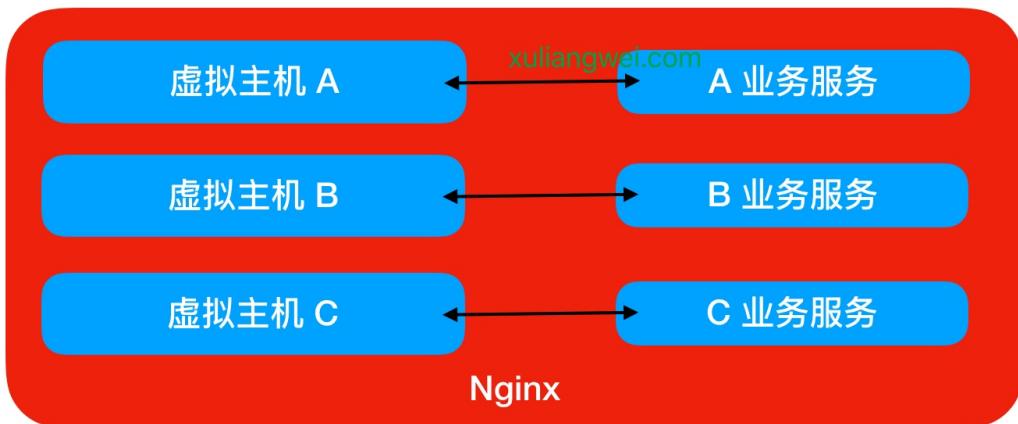
通常在企业中可能会有很多业务系统，那么多套业务服务如何使用Nginx配置？

多套业务服务如何配置？



如果使用如上方式部署，则需要多台服务器配置Nginx，但如果使用虚拟主机方式，则在同一个Nginx上运行多套单独服务，这些服务是相互独立的。简单来说，看似多套业务系统，实则可以运行在一台Nginx服务上

Nginx虚拟主机配置方式



Nginx配置虚拟主机有如下三种方式：

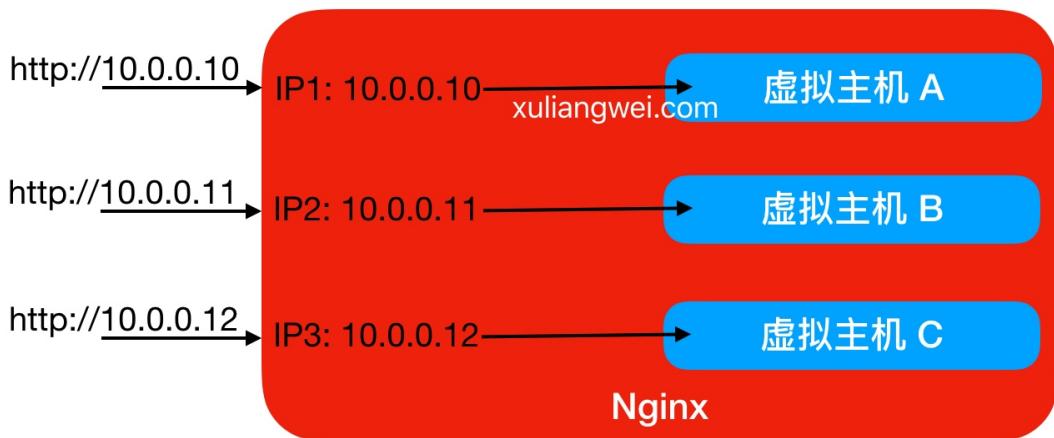
方式一、基于主机多IP方式

方式二、基于端口的配置方式

方式三、基于多个hosts名称方式(多域名方式)

1. 基于多IP的虚拟主机配置实战

基于多IP的虚拟主机方式

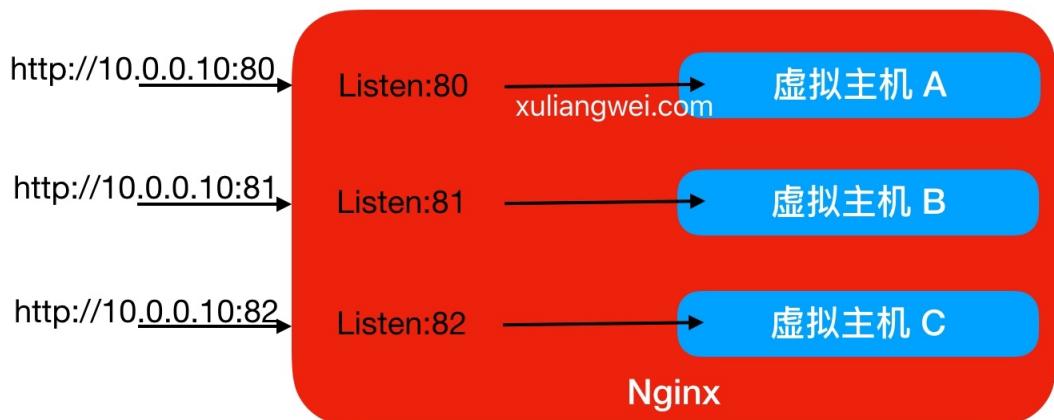


1. 配置多网卡多IP的方式

```
server {  
    ...  
    listen 10.0.0.10:80;  
    ...  
}  
  
server {  
    ...  
    listen 10.0.0.11:80;  
    ...  
}
```

2. 基于端口虚拟主机配置实战

基于多端口的方式



Nginx多端口虚拟主机方式，具体配置如下

```
#仅修改listen监听端口即可，但不能和系统端口出现冲突
```

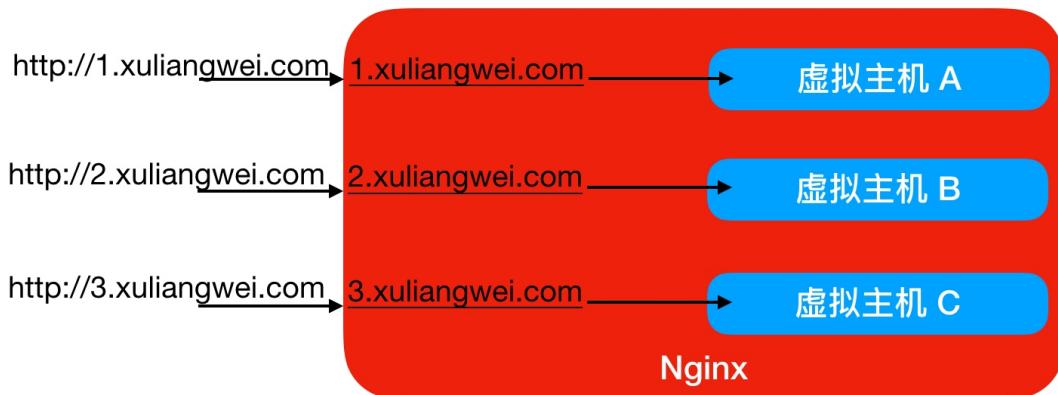
```
[root@web01 ~]# cat /etc/nginx/conf.d/port1.conf
server {
    ...
    listen 80;
    ...
}

[root@web01 ~]# cat /etc/nginx/conf.d/port2.conf
server {
    ...
    listen 81;
    ...
}

[root@web01 ~]# cat /etc/nginx/conf.d/port3.conf
server {
    ...
    listen 82;
    ...
}
```

3. 基于host名称的虚拟主机方式配置实战

基于host名称的虚拟主机方式



1. 创建对应的web站点目录以及程序代码

```
[root@web01 ~]# mkdir /code/{server1,server2}
[root@web01 ~]# echo "server1" > /code/server1/index.html
[root@web01 ~]# echo "server2" > /code/server2/index.html
```

2. 配置不同域名的虚拟主机

```
[root@web02 ~]# cat /etc/nginx/conf.d/server1.conf
server {
    listen      80;
```

```
server_name 1.xuliangwei.com;
root /code/server1;
index index.html;
...
}
[root@web01 ~]# cat /etc/nginx/conf.d/server2.conf
server {
...
listen      80;
server_name 2.xuliangwei.com;
root /code/server2;
index index.html;
}
```

6.Nginx常用模块

6.1 Nginx目录索引

当`ngx_http_index_module`模块找不到索引文件时，通常会将请求传递给`ngx_http_autoindex_module`模块。

`ngx_http_autoindex_module`模块处理以斜杠字符(')结尾的请求，并生成目录列表。

1. 指令

#启用或禁用目录列表输出, `on`开启, `off`关闭。

Syntax: `autoindex on | off;`

Default: `autoindex off;`

Context: `http, server, location`

#指定是否应在目录列表中输出确切的文件大小, `on`显示字节, `off`显示大概单位。

Syntax: `autoindex_exact_size on | off;`

Default: `autoindex_exact_size on;`

Context: `http, server, location`

#指定目录列表中的时间是应以本地时区还是UTC输出。`on`本地时区, `off` UTC时间。

Syntax: `autoindex_localtime on | off;`

Default: `autoindex_localtime off;`

Context: `http, server, location`

2. 场景示例

```
[root@web ~]# cat /etc/nginx/conf.d/mirror.oldxu.com.conf
server {
listen 80;
```

```
server_name mirror.oldxu.com;
charset utf-8;          #设定字符集，防止中文字符乱码显示。
autoindex on;
autoindex_exact_size off;
autoindex_localtime on;

location / {
    root /code/;
}
}
```

3.场景示例：模拟搭建企业内部yum仓库

```
[root@web01 mirror]# cat /etc/nginx/conf.d/mirror.oldxu.com.conf
server {
    listen 80;
    server_name mirror.oldxu.com;
    charset utf-8;
    root /mirror;

    location / {
        index index.html;
    }

    #提供yum仓库目录
    location /repo {
        autoindex on;
        autoindex_exact_size off;
        autoindex_localtime on;
    }
}

#使用rsync同步
[root@web01 mirror]# rsync -avz rsync://rsync.mirrors.ustc.edu.cn/repo/centos/ /mirror/repo/
```

6.2 Nginx访问控制

*ngx_http_access_module*模块允许限制对某些客户端地址的访问。

1.指令

```
#允许配置语法
Syntax: allow address | CIDR | unix: | all;
Default: -
```

```
Context:    http, server, location, limit_except
```

#拒绝配置语法

Syntax: deny address | CIDR | unix: | all;

Default: -

```
Context:    http, server, location, limit_except
```

2. 场景示例，只允许指定的来源IP能访问/centos，其它网段全部拒绝。

```
[root@web ~]# cat /etc/nginx/conf.d/mirror.oldxu.com.conf
server {
    listen 80;
    server_name mirror.oldxu.com;
    charset utf-8;          #设定字符集，防止中文字符乱码显示。
    autoindex on;
    autoindex_exact_size off;
    autoindex_locatime on;

    location / {
        index index.html;
    }

    location /centos {
        allow 127.0.0.1;
        allow 10.0.0.1/32;  #允许地址或地址段
        deny all;           #拒绝所有人
    }
}
```

3. 场景示例，拒绝指定的IP访问该网站的/centos，其他IP全部允许访问。

```
[root@web ~]# cat /etc/nginx/conf.d/mirror.oldxu.com.conf
server {
    listen 80;
    server_name mirror.oldxu.com;
    charset utf-8;          #设定字符集，防止中文字符乱码显示。
    autoindex on;
    autoindex_exact_size off;
    autoindex_locatime on;

    location / {
        index index.html;
    }

    location /centos {
```

```
    deny 10.0.0.1/32;      #拒绝指定的地址或地址段
    allow all;            #允许所有的地址
}
}
```

注意:deny和allow的顺序是有影响的

默认情况下，从第一条规则进行匹配

如果匹配成功，则不继续匹配下面的内容。

如果匹配不成功，则继续往下寻找能匹配成功的内容。

`ngx_http_auth_basic_module`模块允许使用HTTP基本身份验证，验证用户名和密码来限制对资源的访问。

1. 指令

```
#使用HTTP基本身份验证协议启用用户名和密码验证。
Syntax: auth_basic string| off;
Default: auth_basic off;
Context: http, server, location, limit_except
```

```
#指定保存用户名和密码的文件
Syntax: auth_basic_user_file file;
Default: -
Context: http, server, location, limit_except
```

2. 指定保存用户名和密码的文件，格式如下：

```
#可以使用htpasswd程序或"openssl passwd"命令生成对应的密码;
name1: passwd1
name2: passwd2

#使用htpasswd创建新的密码文件， -c 创建新文件 -b 允许命令行输入密码
[root@xuliangwei ~]# yum install httpd-tools
[root@xuliangwei ~]# htpasswd -b -c /etc/nginx/auth_conf xuliangwei 123456
```

3. 场景示例，基于用户名和密码认证实践。

```
[root@web ~]# cat /etc/nginx/conf.d/mirror.oldxu.com.conf
server {
    listen 80;
    server_name mirror.oldxu.com;
    charset utf-8;          #设定字符集，防止中文字符乱码显示。
    autoindex on;
```

```
autoindex_exact_size off;
autoindex_locatime on;

location / {
    index index.html;
}

location /centos {
    auth_basic "Auth access Blog Input your Passwd!";
    auth_basic_user_file /etc/nginx/auth_conf;
}
}
```

6.3 Nginx限流限速

1.为什么要限速?

限制某个用户在一定时间内能够产生的Http请求。或者说限制某个用户的下载速度。

2.限速应用场景?

下载限速：限制用户下载资源的速度，使用Nginx `ngx_http_core_module`。

请求限制：限制用户单位时间内所产生的Http请求数，使用 Nginx `ngx_http_limit_req_module`。

连接限制：限制同一时间的连接数，及并发数限制。使用Nginx `ngx_http_limit_conn_module`

3.请求频率限速实现的原理?



水（请求）从上方倒入水桶，从水桶下方流出（被处理）；

如果说水（请求）流入的过快，水桶流出（被处理）的过慢，来不及流出的水存在水桶中（缓存），然后以固定速率流出，水桶满后则水溢出（丢弃）。

简单来说就是：当处理速度，达不到请求的速度，则会将请求放置缓存，然后持续处理。当缓存

被沾满，如果还有大量的请求，则会被丢弃。

4. 场景实践一、限制单位时间内所产生的Http请求数。

1. 指令

Syntax: `limit_req_zone key zone=name:size rate=rate;`

Default: -

Context: http

Syntax: `limit_conn zone number [burst=number] [nodelay];`

Default: -

Context: http, server, location

2. 基于来源IP对下载速率限制，限制每秒处理1次请求，但可以突发超过5个请求放入缓存区

```
# http标签段定义请求限制, rate限制速率, 限制一秒钟最多一个IP请求
http {
    limit_req_zone $binary_remote_addr zone=req_one:10m rate=1r/s;
}
```

```
server {
    listen 80;
    server_name mirror.oldxu.com;
    # 请求超过1r/s, 剩下的将被延迟处理, 请求数超过burst定义的数量, 则返回503
    limit_req zone=req_one burst=3 nodelay;
    location / {
        root /code;
        index index.html;
    }
}
```

```
limit_req_zone $binary_remote_addr zone=req_one:10m rate=1r/s;
```

#第一个参数: \$binary_remote_addr表示通过这个标识来做限制, 限制同一客户端ip地址。

#第二个参数: zone=req_one:10m表示生成一个大小为10M, 名为req_one的内存区域, 用来存储访问的频次信息。

#第三个参数: rate=1r/s表示允许相同标识的客户端的访问频次, 这里限制的是每秒1次。

```
limit_req zone=req_one burst=3 nodelay;
```

#第一个参数: zone=req_one 设置使用哪个配置区域来做限制, 与上面limit_req_zone 里的name对应。

#第二个参数: burst=3, 设置一个大小为3的缓冲区, 当有大量请求过来时, 超过了访问频次限制的请求可以先放到这个缓冲区内。

#第三个参数: nodelay, 超过访问频次并且缓冲区也满了的时候, 则会返回503, 如果没有设置, 则所有请求会等待排队。

5. 场景实践二、限制客户端同一时刻的并发连接数。

1. 指令

```
Syntax: limit_conn_zone key zone=name:size;
Default: -
Context: http
```

```
Syntax: limit_conn zone number;
Default: -
Context: http, server, location
```

2. 设置共享内存区域和给定键值的最大允许个连接数。超过此限制时，服务器将返回503错误以回复请求

```
[root@web01 ~]# cat /etc/nginx/conf.d/mirror.oldxu.com.conf
limit_conn_zone $binary_remote_addr zone=conn_mg:10m;

server {
    listen 80;
    server_name mirror.oldxu.com;
    root /code;
    charset utf8;
    autoindex on;
    autoindex_exact_size off;
    autoindex_localtime on;

    limit_conn conn_mg 2;

    location / {
        index index.html;
    }
}
```

6. 场景实践三、限制下载速度。

```
[root@web01 ~]# cat /etc/nginx/conf.d/mirror.oldxu.com.conf
server {
    listen 80;
    server_name mirror.oldxu.com;
    root /code;
    charset utf8;
    autoindex on;
    autoindex_exact_size off;
```

```

autoindex_localtime on;

limit_rate_after 100m; #达到100m开始限速
limit_rate 100k;

location / {
    index index.html;
}
}

```

7.综合案例、限制web服务器请求数处理为1秒一个，触发值为5、限制用户仅可同时下载一个文件。当下载超过100M则限制下载速度为500k。如果同时下载超过2个视频，则返回提示 "请联系oldxu进行会员充值"。

```

[root@web01 conf.d]# cat mirror.oldxu.com.conf
limit_req_zone $binary_remote_addr zone=req_mg:10m rate=1r/s;
limit_conn_zone $binary_remote_addr zone=conn_mg:10m;

server {
    listen 80;
    server_name mirror.oldxu.com;
    root /code;
    charset utf8;
    autoindex on;
    autoindex_exact_size off;
    autoindex_localtime on;

    limit_req zone=req_mg burst=5 nodelay;
    limit_conn conn_mg 1;
    limit_rate_after 100m;
    limit_rate 500k;

    error_page 503 @errpage;
    location @errpage {
        default_type text/html;
        return 200 'Oldxu提示-->请联系QQ: 552408925 进行会员充值';
    }
    location / {
        index index.html;
    }
}

```

6.4 Nginx状态监控

*ngx_http_stub_status_module*模块提供对基本状态信息的访问。

默认情况下不集成该模块，需要使用--with-http_stub_status_module 集成。

1. 指令

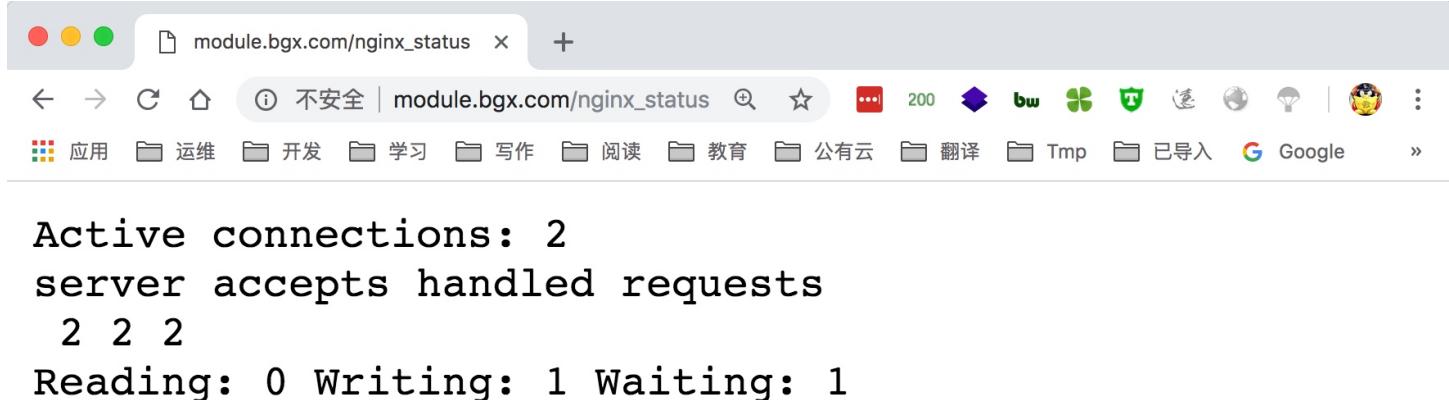
```
Syntax: stub_status;
Default: -
Context: server, location
```

2. 示例配置

```
[root@web ~]# cat /etc/nginx/conf.d/module.conf
server {
    listen 80;
    server_name mirror.oldxu.com;
    access_log off;

    location /nginx_status {
        stub_status;
    }
}
```

3. 此配置创建一个简单的网页，其基本状态数据可能如下所示：



Active connections: 2
server accepts handled requests
2 2 2
Reading: 0 Writing: 1 Waiting: 1

4. 提供以下状态信息

状态	含义
Active connections	当前活跃连接数，包括Waiting等待连接数。
accepts	已接收的总TCP连接数量。
handled	已处理的TCP连接数量。
requests	当前总http请求数量。

Reading	当前读取的请求头数量。
Writing	当前响应的请求头数量。
Waiting	当前等待请求的空闲客户端连接数

如何简单理解Reading、Writing、Waiting。

假设现在有两条船分别为：C、S。C船需要S船的1个物品，那么此时C船就要给S船发送一个消息。

1、S船收到这个消息时就是reading。

2、S船将物资发送给C船，这个时候就是writing。

3、如果C船需要S船很多个物品，那么需要C船和S船建立起一个物资传送管道，不断的传送物资。这个管道建立起来的时候，就是waiting状态了。

6.5 Nginx Location

1.什么是Location

Location用来控制访问网站的uri路径

2.Location语法

```
location [ = | ~ | ~* | ^~ ] uri { ... }
location @name { ... }
```

# 匹配符	匹配规则	优先级
# =	精确匹配	1
# ^~	以某个字符串开头	2
# ~	区分大小写的正则匹配	3
# ~*	不区分大小写的正则匹配	4
# /	通用匹配，任何请求都会匹配到	5

3.location优先级语法示例

```
[root@web01 conf.d]# cat location.oldxu.com.conf
server {
    listen 80;
    server_name location.oldxu.com;

    location = / {
        default_type text/html;
        return 200 'location = /';
    }

    location / {
```

```

    default_type text/html;
    return 200 'location /';
}

location /documents/ {
    default_type text/html;
    return 200 'location /documents/';
}

location ^~ /images/ {
    default_type text/html;
    return 200 'location ^~ /images/';
}

location ~* \.(gif|jpg|jpeg)$ {
    default_type text/html;
    return 200 'location ~* \.(gif|jpg|jpeg)';
}
}

```

#测试结果如下(建议是curl测试)

#1. 请求 `http://location.oldxu.com/`
#2. 请求 `http://location.oldxu.com/index.html`
#3. 请求 `http://location.oldxu.com/documents/1.html`
匹配
#4. 请求 `http://location.oldxu.com/images/1.gif`
匹配
#5. 请求 `http://location.oldxu.com/documents/1.jpg`
`jpeg)$`匹配

会被	<code>Location =/</code>	匹配
会被	<code>Location /</code>	匹配
会被	<code>Location /documents/</code>	
会被	<code>Location ^~ /images/</code>	
会被	<code>Location ~* \.(gif jpg j</code>	

4.Location规则配置应用场景，可进行验证测试

```
[root@web01 conf.d]# cat location2.oldxu.com.conf
server {
    listen 80;
    server_name location2.oldxu.com;

    # 通用匹配，任何请求都会匹配到
    location / {
        root html;
        index index.html;
    }

    # 精准匹配，必须请求的uri是/nginx_status
    location = /nginx_status {
        stub_status;
    }
}
```

```
}

# 严格区分大小写, 匹配以.php结尾的都走这个location
location ~ \.php$ {
    default_type text/html;
    return 200 'php访问成功';
}

# 严格区分大小写, 匹配以.jsp结尾的都走这个location
location ~ \.jsp$ {
    default_type text/html;
    return 200 'jsp访问成功';
}

# 不区分大小写匹配, 只要用户访问.jpg,.gif,.png,.js,.css 都走这条location
location ~* \.(jpg|gif|png|js|css)$ {
    return 403;
}

# 不区分大小写匹配
location ~* \.(sql|bak|tgz|tar.gz|git)$ {
    deny all;
}
}
```

04.Nginx搭建流行架构

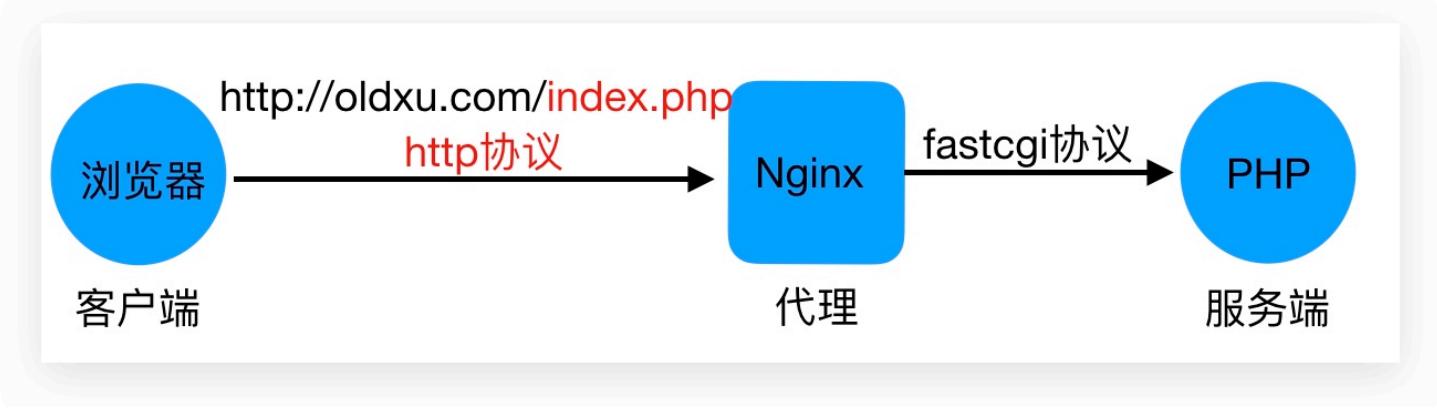
1.LNMP架构基本概述

1.1 什么是LNMP

- LNMP 是一套技术的组合, L=Linux、N=Nginx、M=[MySQL | Mariadb]、P=[PHP | Python]

1.2 LNMP实现过程

- 用户请求 `http://oldxu.com/index.php`, 对于Nginx服务而言, 是无法处理 `index.php` 这样的动态脚本的, 那么 Nginx 该如何配置, 才能支持这样的动态请求呢?
 - 第一步: 当用户发起 `HTTP` 请求, 请求首先被 `Nginx` 接收;
 - 第二步: `Nginx` 通过预先定义好的 `location` 规则进行匹配;
 - 第三步: `Nginx` 将匹配到的动态内容, 通过 `fastcgi` 协议传到给后端的 `php` 应用服务器处理



1.3 LNMP实现细节

- Nginx、PHP、MySQL 之间是如何工作的
 - 1. 用户首先通过 http 协议发起请求，请求会先抵达 Nginx；
 - 2. Nginx 根据用户的请求进行 Location 规则匹配；
 - 3. Location 如果匹配到请求是静态，则由 Nginx 读取本地直接返回；
 - 4. Location 如果匹配到请求是动态，则由 Nginx 将请求转发给 fastcgi 协议；
 - 5. fastcgi 收到后会将请求交给 php-fpm 管理进程；
 - 6. php-fpm 管理进程接收到后会调用具体的工作进程 warrap
 - 6. warrap 进程会调用 php 解析器解析代码，php 解析后直接返回
 - 7. 如果有查询数据库操作，则由 php 连接数据库(用户 密码 IP)发起查询的操作
 - 8. 最终数据由 mysql->php->php-fpm->fastcgi->nginx->http->user

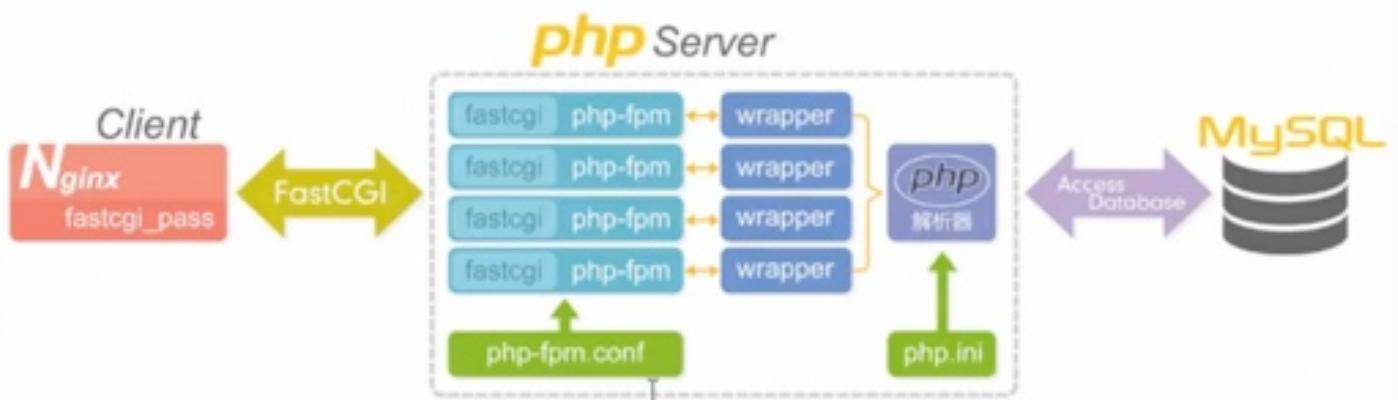


图 6-3 Nginx 结合 PHP FastCGI 运行原理图

2.LNMP架构环境安装

2.1 Nginx安装

1. 使用官方仓库安装 Nginx

```
[root@nginx ~]# cat /etc/yum.repos.d/nginx.repo
[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/centos/7/$basearch/
gpgcheck=0
enabled=1

#安装Nginx
[root@nginx ~]# yum install nginx -y
```

2.配置 Nginx 进程运行用户

```
[root@nginx ~]# groupadd -g666 www
[root@nginx ~]# useradd -u666 -g666 www
[root@nginx ~]# sed -i '/^user/c user    www;' /etc/nginx/nginx.conf
```

3.启动 Nginx，并将 Nginx 加入开机自启

```
[root@nginx ~]# systemctl start nginx
[root@nginx ~]# systemctl enable nginx
```

2.2 php安装

- 使用第三方扩展源安装 [php7.1 本地安装方式点击此链接](#)

1.安装 rpm 生成 repo 文件，或手动新增 repo 文件；

```
# rpm -Uvh https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
# rpm -Uvh https://mirror.webtatic.com/yum/el7/webtatic-release.rpm

# 手动配置yum源
[root@nginx ~]# cat /etc/yum.repos.d/php.repo
[webtatic-php]
name = php Repository
baseurl = http://us-east.repo.webtatic.com/yum/el7/x86_64/
gpgcheck = 0
```

2.卸载低版本的 php

```
[root@nginx ~]# yum remove php-mysql-5.4 php php-fpm php-common
```

3. 安装高版本的 PHP

```
[root@nginx ~]# yum -y install php71w php71w-cli \
php71w-common php71w-devel php71w-embedded php71w-gd \
php71w-mcrypt php71w-mbstring php71w-pdo php71w-xml \
php71w-fpm php71w-mysqlnd php71w-opcache \
php71w-pecl-memcached php71w-pecl-redis php71w-pecl-mongodb
```

4. 配置 PHP-FPM 用户与 Nginx 的运行用户保持一致

```
[root@nginx ~]# sed -i '/^user/c user = www' /etc/php-fpm.d/www.conf
[root@nginx ~]# sed -i '/^group/c group = www' /etc/php-fpm.d/www.conf
```

5. 启动 PHP-FPM 并将其加入开机自启

```
[root@nginx ~]# systemctl start php-fpm
[root@nginx ~]# systemctl enable php-fpm
```

2.3 MySQL 安装

1. 安装 Mariadb 数据库

```
[root@nginx ~]# yum install mariadb-server mariadb -y
```

2. 启动 Mariadb 数据库，并加入开机自动

```
[root@nginx ~]# systemctl start mariadb
[root@nginx ~]# systemctl enable mariadb
```

3. 给 Mariadb 配置登陆密码，并用新密码进行登录数据库

```
[root@nginx ~]# mysqladmin password 'oLdxu123.com'
[root@nginx ~]# mysql -uroot -poLdxu123.com
```

3. LNMP 架构环境配置

- 在配置 Nginx 与 PHP 集成之前，我们需要先了解 Nginx 的 FastCGI 代理配置语法

3.1 Fastcgi代理语法

1.设置 fastcgi 服务器的地址，该地址可以指定为域名或IP地址，以及端口

```
Syntax: fastcgi_pass address;  
Default: -  
Context: location, if in location
```

```
#语法示例  
fastcgi_pass localhost:9000;
```

2.设置 fastcgi 默认的首页文件，需要结合 fastcgi_param 一起设置

```
Syntax: fastcgi_index name;  
Default: -  
Context: http, server, location
```

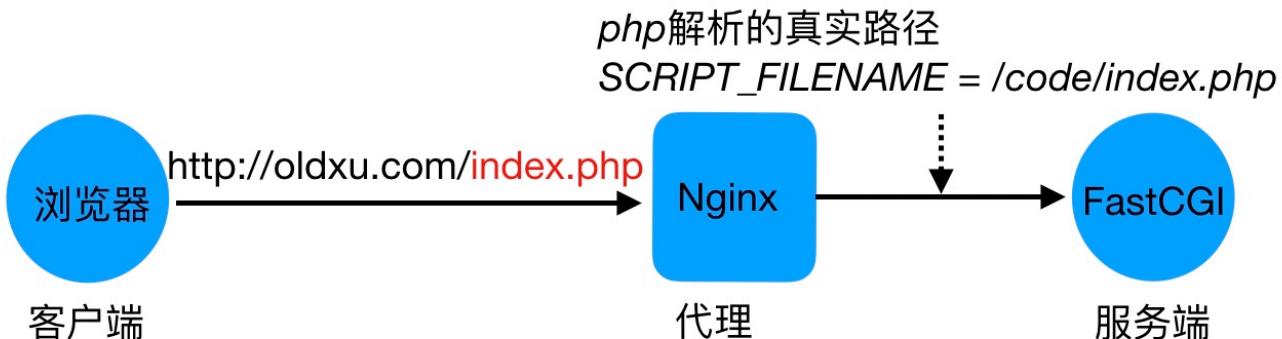
3.通过 fastcgi_param 设置变量，并将设置的变量传递到后端的 fastcgi 服务

```
Syntax: fastcgi_param parameter value [if_not_empty];  
Default: -  
Context: http, server, location  
  
#语法示例  
fastcgi_index index.php;  
fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
```

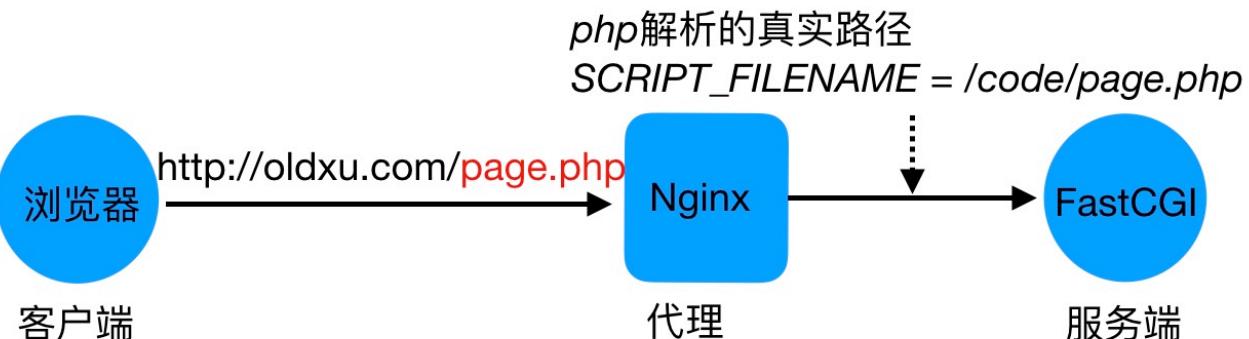
4.通过图形方式展示 fastcgi_index 与 fastcgi_param 作用。

Fastcgi代理配置

```
root /code;
fastcgi_index index.php;
fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
```



```
root /code;
fastcgi_index index.php;
fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
```



3.2 Nginx与PHP集成

1. 编写Nginx配置文件

```
[root@nginx ~]# cat /etc/nginx/conf.d/php.conf
*server {
    server_name php.oldxu.com;
    listen 80;
    root /code;
    index index.php index.html;

    location ~ \.php$ {
        root /code;
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }
}
```

```
    }  
}*  
}
```

2. 在 /code 目录下创建 info.php 文件

```
[root@nginx ~]# cat /code/info.php  
<?php  
    phpinfo();  
?>
```

3. 通过浏览器访问 /info.php , 返回如下页面表示 nginx 与 php 配置成功;

PHP Version 7.1.24	
System	Linux web01 3.10.0-862.el7.x86_64 #1 SMP Fri Apr 20 16:44:24 UTC 2018 x86_64
Build Date	Nov 11 2018 08:06:08
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc
Loaded Configuration File	/etc/php.ini
Scan this dir for additional .ini files	/etc/php.d
Additional .ini files parsed	/etc/php.d/bz2.ini, /etc/php.d/calendar.ini, /etc/php.d/ctype.ini, /etc/php.d/curl.ini, /etc/php.d/dom.ini,

3.3 PHP与MySQL集成

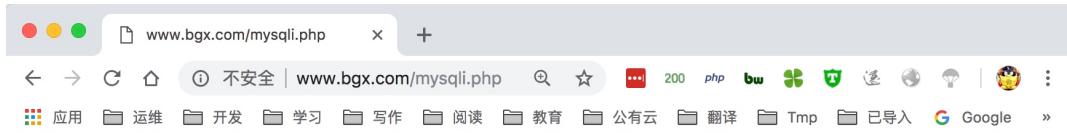
1. 在 /code 目录下创建 mysqli.php 文件，填入对应的数据库IP、用户名、密码*

```
[root@nginx ~]# cat /code(mysqli.php  
<?php  
    $servername = "localhost";  
    $username = "root";  
    $password = "oldxu123.com";  
  
    // 创建连接  
    $conn = mysqli_connect($servername, $username, $password);  
  
    // 检测连接  
    if (!$conn) {  
        die("Connection failed: " . mysqli_connect_error());  
    }  
    echo "php连接MySQL数据库成功";  
?>
```

2. 使用 php 命令直接解析文件

```
[root@nginx ~]# php /code/mysqli.php  
php连接MySQL数据库成功
```

3.也可以通过浏览器访问 `/mysqli.php` 文件，获取解析结果



PHP连接MySQL数据库成功

4.部署博客产品Wordpress

4.1 配置Nginx

1.配置 Nginx 虚拟主机站点，域名为 `blog.oldxu.com`

```
[root@nginx ~]# cat /etc/nginx/conf.d/wordpress.conf  
server {  
    listen 80;  
    server_name blog.oldxu.com;  
    root /code/wordpress;  
    index index.php index.html;  
  
    location ~ \.php$ {  
        fastcgi_pass 127.0.0.1:9000;  
        fastcgi_index index.php;  
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;  
        include fastcgi_params;  
    }  
}
```

2.检测语法，并重启 nginx 服务

```
[root@nginx ~]# nginx -t  
[root@nginx ~]# systemctl restart nginx
```

4.2 配置MySQL

由于 wordpress 产品需要依赖数据库， 所以需要手动建立数据库

```
[root@nginx ~]# mysql -uroot -poldxu123.com
mysql> create database wordpress;
mysql> exit
```

4.3 部署Wordpress

1.获取 wordpress 产品，解压并部署 wordpress

```
[root@nginx ~]# mkdir /code
[root@nginx code]# wget https://cn.wordpress.org/wordpress-4.9.4-zh_CN.tar.gz
[root@nginx ~]# tar xf wordpress-4.9.4-zh_CN.tar.gz -C /code
```

2.授权目前的权限为进程运行的用户身份；

```
[root@nginx ~]# chown -R www.www /code/wordpress/
```

5.部署知乎产品WeCenter

5.1 配置Nginx

1.配置 Nginx 虚拟主机站点，域名为 zh.oldxu.com

```
[root@nginx ~]# cat /etc/nginx/conf.d/zh.conf
server {
    listen 80;
    server_name zh.oldxu.com;
    root /code/zh;
    index index.php index.html;

    location ~ \.php$ {
        root /code/zh;
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }
}
```

2.检查语法，并重启 nginx 服务

```
[root@nginx ~]# systemctl restart nginx
```

5.2 配置MySQL

由于 wecenter 产品需要依赖数据库，所以需要手动建立数据库

```
[root@nginx ~]# mysql -uroot -poldxu123.com
MariaDB [(none)]> create database zh;
MariaDB [(none)]> exit
```

5.3 部署wecenter

1.获取 Wecenter 产品，解压并部署 Wecenter

```
[root@nginx ~]# wget http://ahdx.down.chinaz.com/201605/WeCenter_v3.2.1.zip
[root@nginx ~]# unzip WeCenter_v3.1.9.zip
[root@nginx ~]# mv UPLOAD/ /code/zh
```

2.授权目前的权限为进程运行的用户身份；

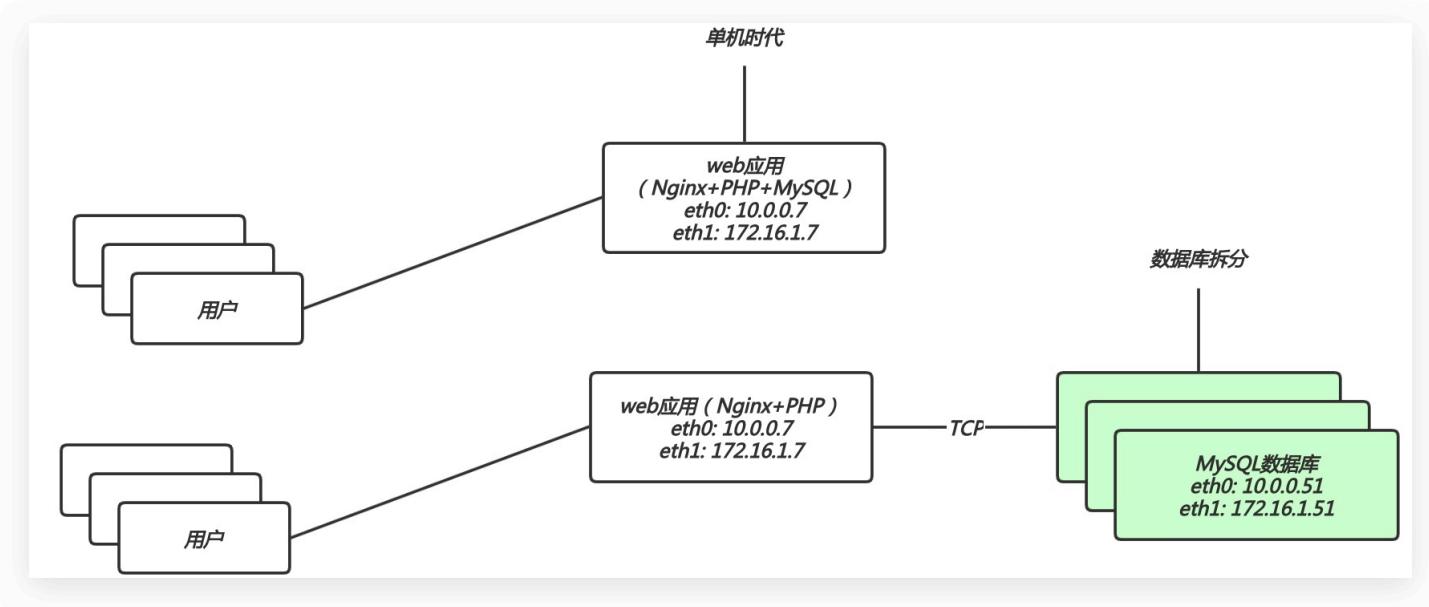
```
[root@nginx ~]# chown -R www.www /code/zh/
```

6.拆分数据库至独立服务器

6.1 为何要拆分数据库

- 由于单台服务器运行 LNMP 架构会导致网站访问缓慢，当系统内存被吃满时，很容易导致系统出现 oom，从而 kill 掉 MySQL 数据库，所以需要将 web 和数据库进行独立部署。
- 拆分数据库能解决什么问题
 - 1.缓解 web 网站的压力；
 - 2.增强数据库读写性能；
 - 3.提高用户访问的速度；

6.2 数据库拆分架构演变



6.3 数据库拆分环境准备

主机名称	应用环境	外网地址	内网地址
web01	nginx+php	10.0.0.7	172.16.1.7
db01	mysql		172.16.1.51

6.4 数据库拆分实现步骤

6.4.1 web服务操作如下

1. 备份 web01 上的数据库

```
[root@web01 ~]# mysql dump -uroot -p'oldxu123.com' --all-databases > mysql-all.sql
```

2. 将 web01 上备份的数据库拷贝至 db01 服务器上

```
[root@web01 ~]# scp mysql-all.sql root@172.16.1.51:/tmp
```

6.4.2 数据库服务操作如下

1. 将 web01 服务器上推送的数据库备份文件恢复至 db01 服务器新数据库中

```
[root@db01 ~]# yum install mariadb mariadb-server -y
[root@db01 ~]# systemctl start mariadb
[root@db01 ~]# systemctl enable mariadb
[root@db01 ~]# mysql -uroot < /tmp/mysql-all.sql
```

2. 数据库导入完成后，重启数据库，使用新密码进行登录，并检查数据库已被导入成功

```
[root@db01 ~]# systemctl restart mariadb
[root@db01 ~]# mysql -uroot -poldxu123.com
mysql> show databases;
```

3. 在新数据库上授权，允许所有网段，通过 all 账户连接并操作该数据库

授权所有权限 grant all privileges

授权所有库所有表 *.*

将授权赋予给哪个用户，这个用户只能通过哪个网段过来，% 表示所有；'all'@'%'

授权该用户登录的密码 identified by；

```
mysql> grant all on *.* to all@'%' identified by 'oldxu123.com';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

6.4.3 修改代码指向新数据库

1. 修改 Wordpress 产品代码连接数据库的配置文件

```
[root@web01 ~]# vim /code/wordpress/wp-config.php
# 数据库名称
define('DB_NAME', 'wordpress');
# 数据库用户
define('DB_USER', 'all');
# 数据库密码
define('DB_PASSWORD', 'oldxu123.com');
# 数据库地址
define('DB_HOST', '172.16.1.51');
```

2. 修改 wecenter 产品代码连接数据库的配置文件

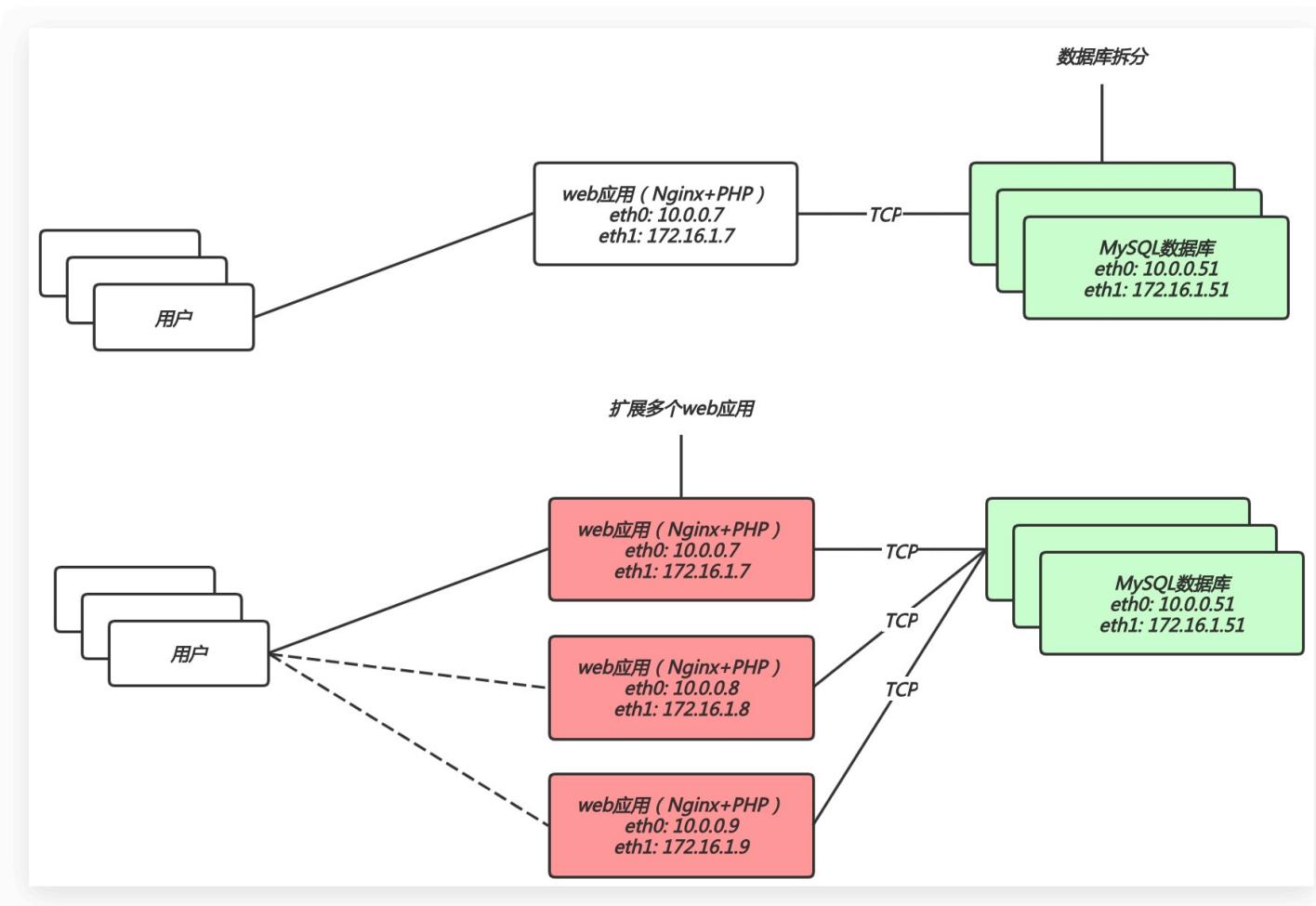
```
[root@web01 zh]# grep -iR "oldxu123.com" | grep -v cache
system/config/database.php: 'password' => 'oldxu123.com',
[root@web01 zh]# vim /code/zh/system/config/database.php
'host' => '172.16.1.51',
'username' => 'all',
'password' => 'oldxu123.com',
'dbname' => 'zh',
```

7. 扩展多台相同的Web服务器

7.1 为何要扩展多台web节点

- 单台 web 服务器能抗拒的访问量是有限的，配置多台 web 服务器能提升更高的访问速度。
- 扩展多台节点解决什么问题
 - 1.单台 web 节点如果故障，会导致业务 down 机；
 - 2.多台 web 节点能保证业务的持续稳定，扩展性高；
 - 3.多台 web 节点能有效的提升用户访问网站的速度；

7.2 扩展多web节点架构演变



7.3 扩展多web节点环境准备

主机名称	应用环境	外网地址	内网地址
web01	nginx+php	10.0.0.7	172.16.1.7
web02	nginx+php	10.0.0.8	172.16.1.8

db01

mysql

172.16.1.51

7.4 扩展多web节点实现步骤

- 通过 web01 现有环境快速的扩展一台 web02 的服务器，数据库统一使用 db01

7.4.1 LNP环境安装

1. 创建 www 用户

```
[root@web02 ~]# groupadd -g666 www  
[root@web02 ~]# useradd -u666 -g666 www
```

2. 安装 LNP

```
[root@web02 ~]# scp -rp root@172.16.1.7:/etc/yum.repos.d/* /etc/yum.repos.d/  
[root@web02 ~]# scp -rp root@172.16.1.7:/etc/pki/rpm-gpg/* /etc/pki/rpm-gpg/  
  
[root@web02 ~]# yum install nginx -y  
[root@web02 ~]# yum -y install php71w php71w-cli \  
php71w-common php71w-devel php71w-embedded php71w-gd \  
php71w-mcrypt php71w-mbstring php71w-pdo php71w-xml \  
php71w-fpm php71w-mysqlnd php71w-opcache \  
php71w-pecl-memcached php71w-pecl-redis php71w-pecl-mongodb
```

7.4.2 LNP配置导入

1. 将 web01 的 nginx 配置文件导入到 web02

```
[root@web02 ~]# scp -rp root@172.16.1.7:/etc/nginx /etc/
```

2. 将 web01 的 php 配置文件导入到 web02

```
[root@web02 ~]# scp -rp root@172.16.1.7:/etc/php-fpm.d /etc/
```

7.4.3 导入代码文件

1. 将 web01 的代码打包传输到 web02 服务器上，在 web1 上线进行打包操作

```
[root@web01 ~]# tar czf code.tar.gz /code  
[root@web01 ~]# scp code.tar.gz root@172.16.1.8:/tmp
```

2. 在 web02 服务器上进行解压

```
[root@web02 ~]# tar xf /tmp/code.tar.gz -C /
```

7.4.4 启动服务验证

启动 nginx 与 php-fpm 并加入开机自启

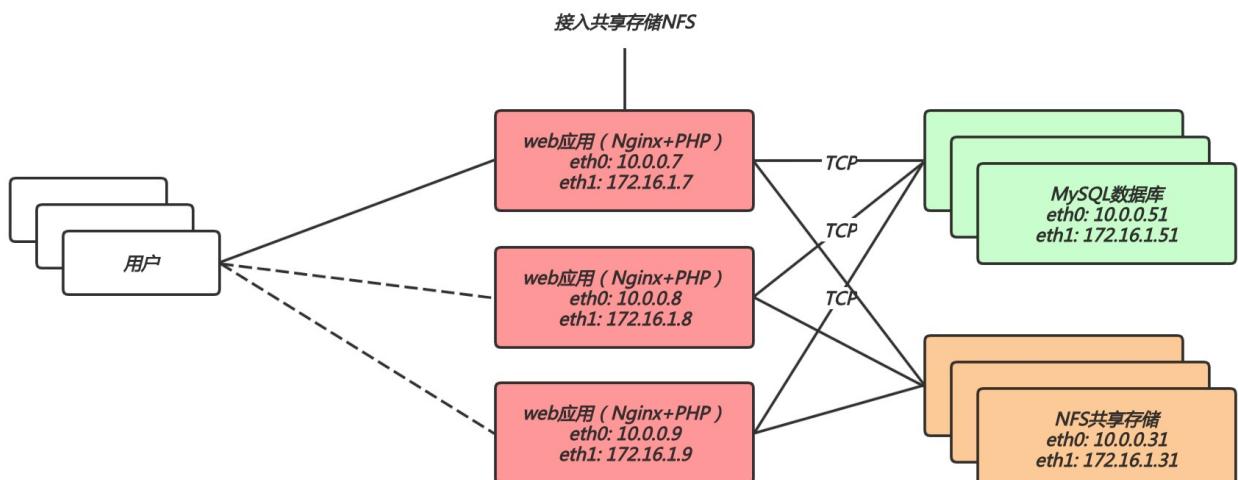
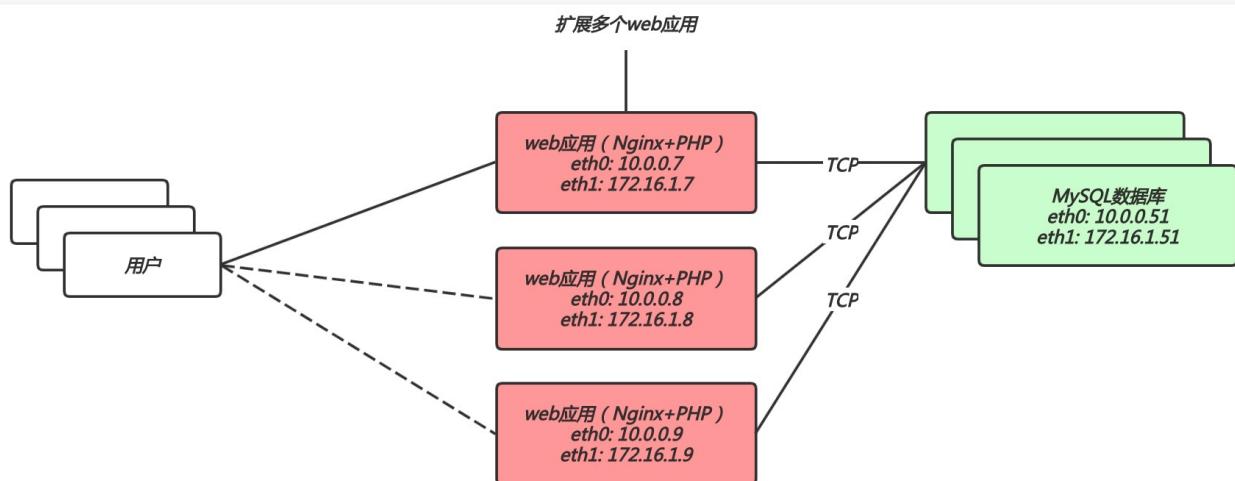
```
[root@web03 ~]# systemctl start nginx php-fpm  
[root@web03 ~]# systemctl enable nginx php-fpm
```

8. 拆分静态资源至独立服务器

8.1 为什么要拆分静态资源

- 当后端的 web 节点出现多台时，会导致用户上传的图片、视频附件等内容仅上传至一台 web 服务器，那么其他的 web 服务器则无法访问到该图片。
- 如果增加一台共享存储能解决什么问题
 - 1. 保证了多台 web 节点静态资源一致。
 - 2. 有效节省多台 web 节点的存储空间。
 - 3. 统一管理静态资源，便于后期推送至 CDN 进行静态资源加速

8.2 拆分静态资源架构演变



8.3 增加共享存储环境准备

主机名称	应用环境	外网地址	内网地址
web01	nginx+php	10.0.0.7	172.16.1.7
web02	nginx+php	10.0.0.8	172.16.1.8
nfs	nfs		172.16.1.31
db01	mysql		172.16.1.51

8.4 增加共享存储实现步骤

8.4.1 配置NFS存储

1. 安装并配置 nfs

```
[root@nfs ~]# yum install nfs-utils -y
[root@nfs ~]# cat /etc/exports
```

```
/data/blog 172.16.1.0/24(rw,sync,all_squash,anonuid=666,anongid=666)
/data/zh 172.16.1.0/24(rw,sync,all_squash,anonuid=666,anongid=666)
```

2. 创建共享目录，并进行授权*

```
[root@nfs01 ~]# mkdir /data/{blog,zh} -p
[root@nfs01 ~]# chown -R www.www /data/
```

3. 启动 nfs 服务，并加入开机自启*

```
[root@nfs01 ~]# systemctl restart nfs-server
```

8.4.2 导入静态资源至存储

1. web01 节点安装 nfs，然后使用 showmount 查看服务端共享的资源；

```
[root@web01 ~]# yum install nfs-utils -y
[root@web01 ~]# showmount -e 172.16.1.31
Export list for 172.16.1.31:
/data/zh    172.16.1.0/24
/data/blog  172.16.1.0/24
```

2. 查找 Wordpress 静态资源，然后；

```
# 首先打开浏览器->右键->检查->Network->
# 然后点击左上角的Select按钮->点击对应的图片
# 最后提取站点中对应的url地址->
# http://blog.oldxu.com/wp-content/uploads/2018/11/timg.gif
```

3. 拷贝静态资源至 nfs 共享存储

```
[root@web01 ~]# cd /code/wordpress/wp-content/
[root@web01 wp-content]# scp -rp uploads/* root@172.16.1.31:/data/blog
```

8.4.3 节点1接入共享存储

1. web01 客户端执行挂载操作

```
[root@web01 wp-content]# mount -t nfs 172.16.1.31:/data/blog /code/wordpress/wp-content/uploads/
```

2.将挂载信息加入开机自启

```
[root@web01 wp-content]# tail -1 /etc/fstab  
172.16.1.31:/data/blog /code/wordpress/wp-content/uploads nfs defaults 0 0  
[root@web01 wp-content]# mount -a
```

8.4.4 节点2接入共享存储

1. web02 客户端直接挂载 nfs 即可

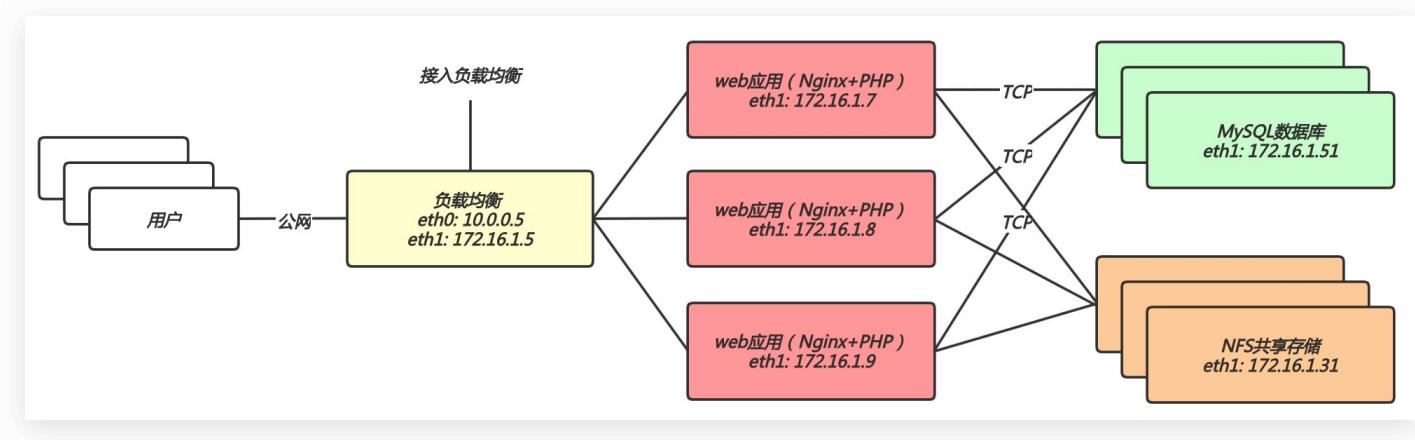
```
[root@web02 ~]# mount -t nfs 172.16.1.31:/data/blog /code/wordpress/wp-content/uploads/
```

2.将挂载信息加入开机自启

```
[root@web02 ~]# tail -1 /etc/fstab  
172.16.1.31:/data/blog /code/wordpress/wp-content/uploads nfs defaults 0 0  
[root@web02 ~]# mount -a
```

9.扩展节点带来的新问题

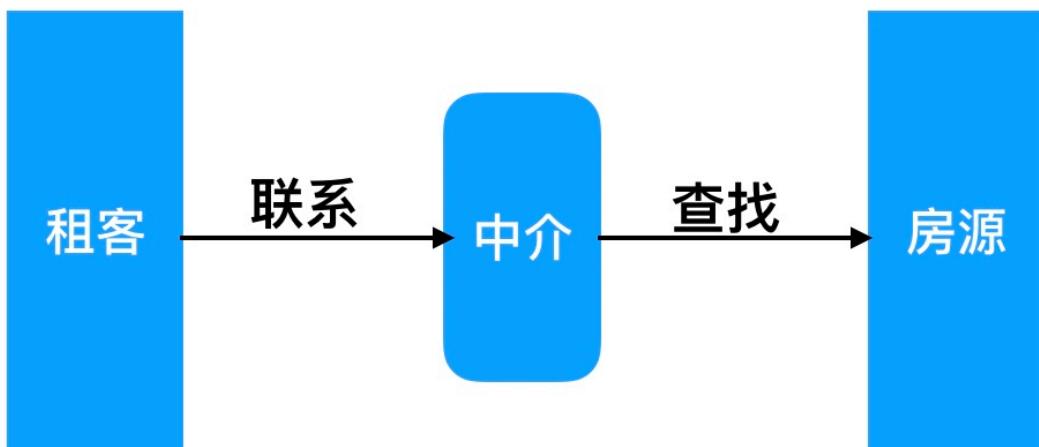
- F1：如果我们添加了一台C应用服务器，如何能实现快速扩展?
 - 1.准备LNP环境
 - 2.拷贝任意A或B上的配置文件,代码
 - 3.挂载NFS存储
- F2：现在有多个WEB服务器，该如何进行访问?
 - 1.DNS轮询
 - (1) 需要所有的web节点具备公网IP地址
 - (2) 公网独立IP需要费用，而且不便宜。
 - (3) 所有的web节点有公网IP，不安全。
 - (4) DNS只有轮询机制，没有健康检查功能。
 - 2.负载均衡
 - (1) 所有的web节点不需要有公网IP，能节省成本、并保证安全
 - (2) 能够对后端的web节点进行健康检查机制；
 - (3) 负载均衡有多种调度算法来满足企业不同需求；



05.Nginx反向代理服务

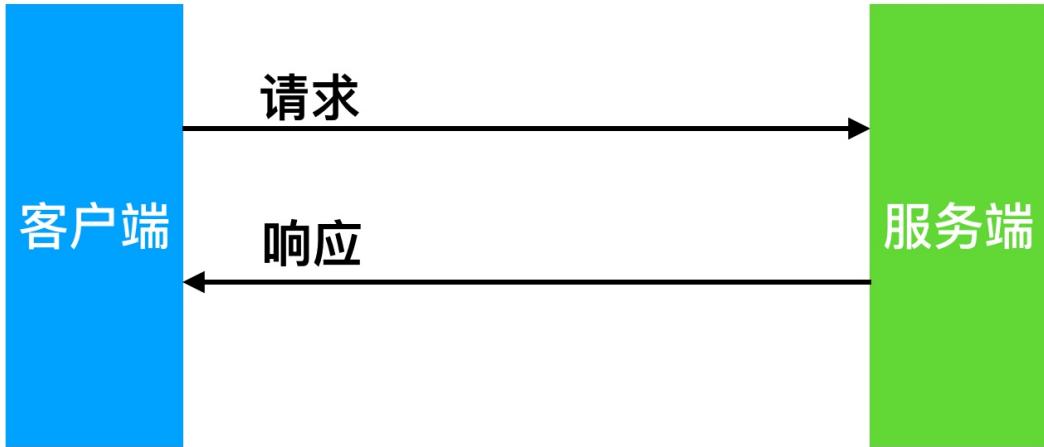
1.Nginx代理服务基本概述

1. 代理一词往往我们并不陌生，该服务我们常常用到。(比如：代理理财、代理租房、代理收货等等)

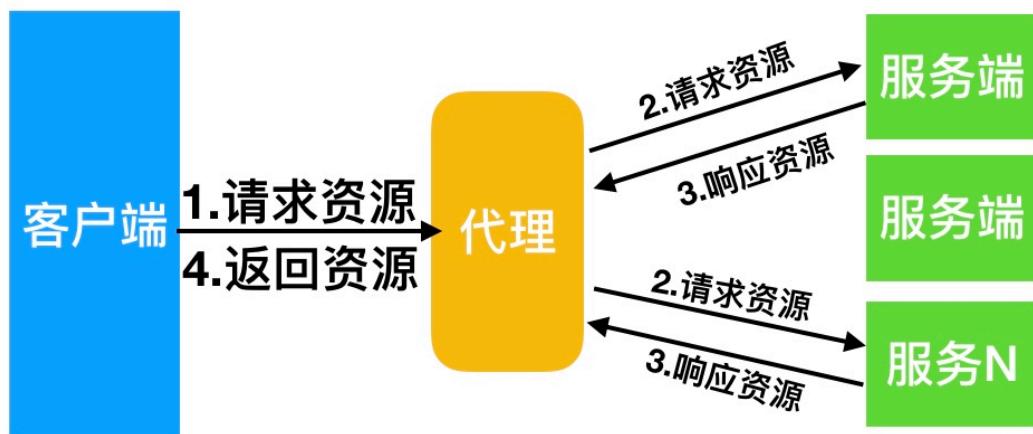


2. 在没有代理模式的情况下，都是客户端直接请求服务端，服务端直接响应客户端。

比如：抖音在初创阶段时没太多人关注，单台服务器足以支撑业务运行。但随着事件推移，xx门事件的发生，引得抖音迅速蹿红，那么此时单台服务器难以支撑海量的用户请求，甚至一度造成服务瘫痪。



3. 在有代理模式的情况下，客户端往往无法直接向服务端发起请求，而是需要使用到代理服务，来实现客户端和服务通信。

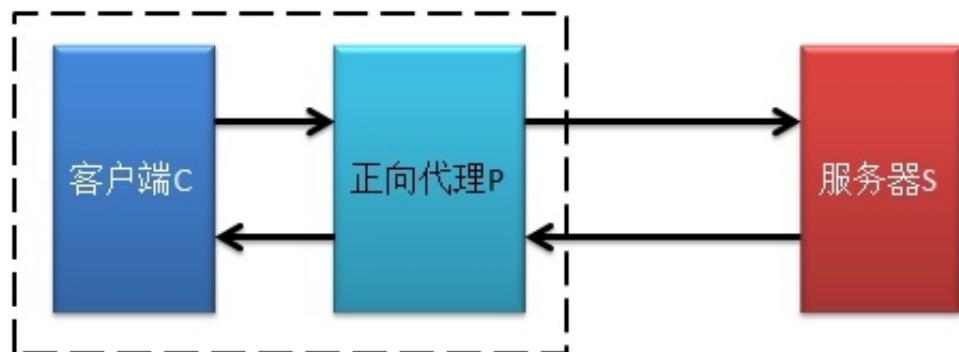


2.Nginx代理服务常见模式

那Nginx作为代理服务，按照应用场景模式进行总结，代理分为正向代理、反向代理

2.1) 正向代理

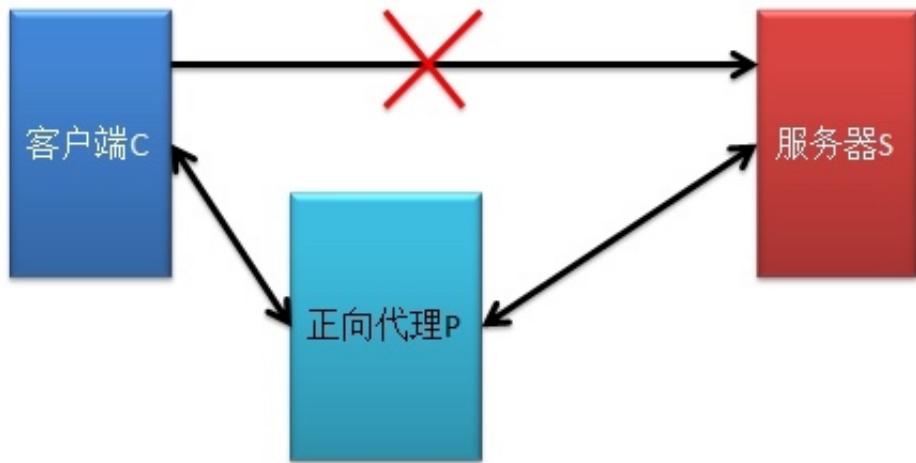
正向代理，(内部上网) 客户端<-->代理->服务端



正向代理服务器功能示意图

1、客户端翻墙

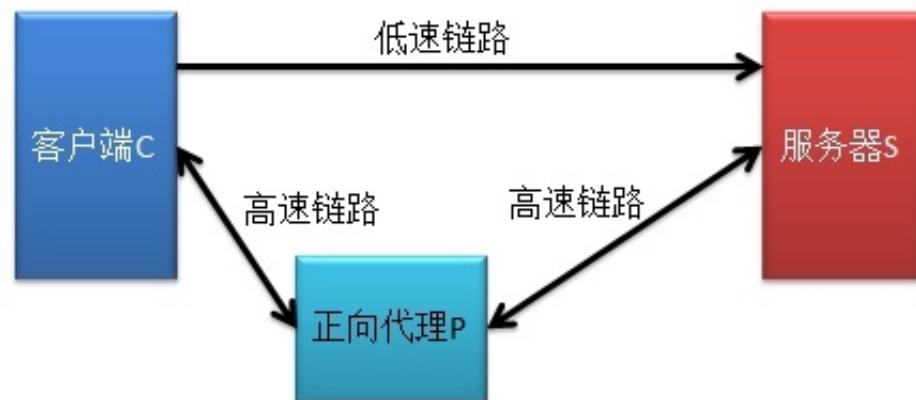
比如：科学的方式访问Google



正向代理服务器“翻墙”功能示意图

2、客户端提速

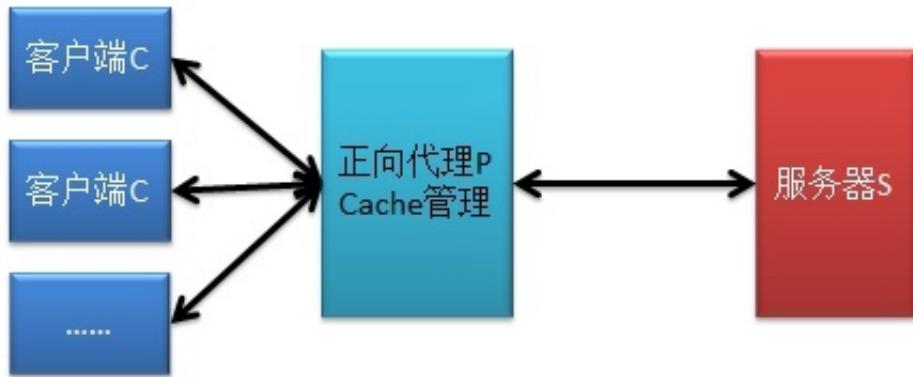
比如：游戏加速器。



正向代理服务器“提速”功能示意图

3、客户端缓存

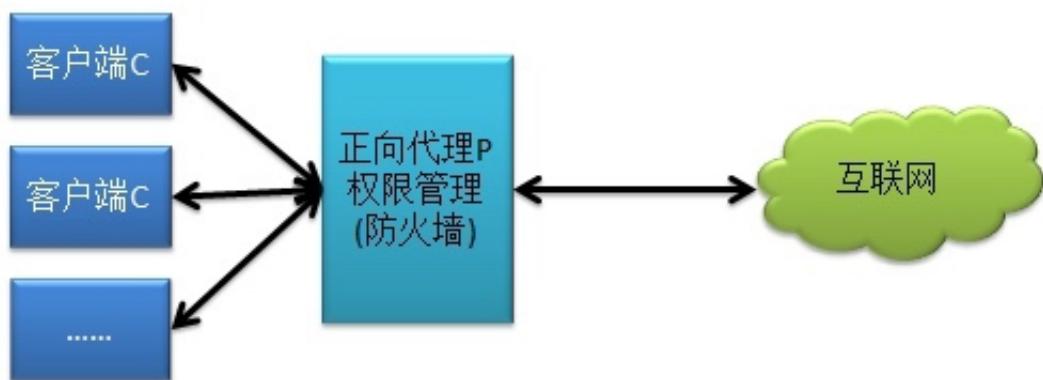
比如：下载资源，可以先查看代理服务是否有，如果有直接通过代理获取。



正向代理服务器“缓存”功能示意图

4、客户端授权

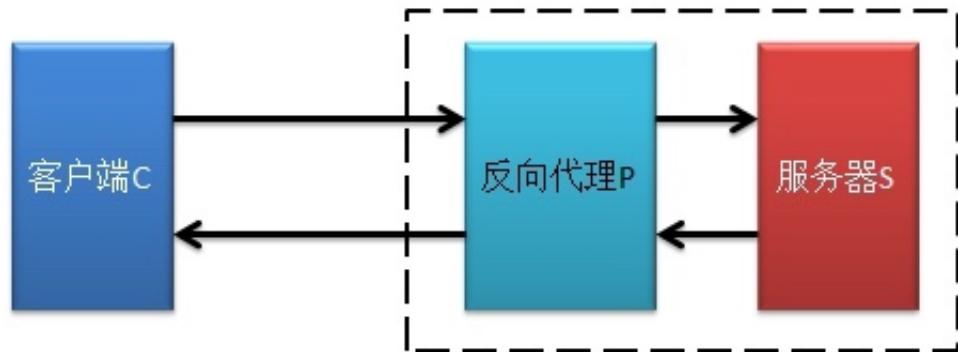
很多公司为了安全，连接外网需要通过防火墙，防火墙可以配置规则，允许谁可以上外网，谁不可以以上外网。



正向代理服务器“授权”功能示意图

2.2) 反向代理

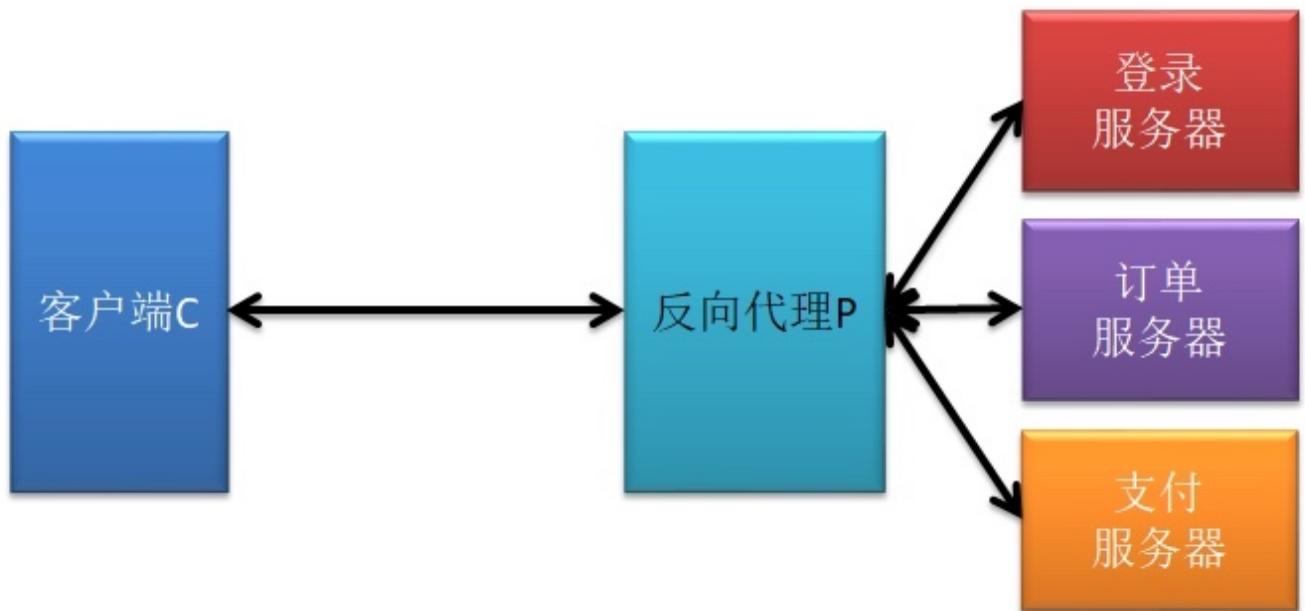
反向代理，用于公司集群架构中，客户端->代理<-->服务端



反向代理服务器功能示意图

1、路由功能

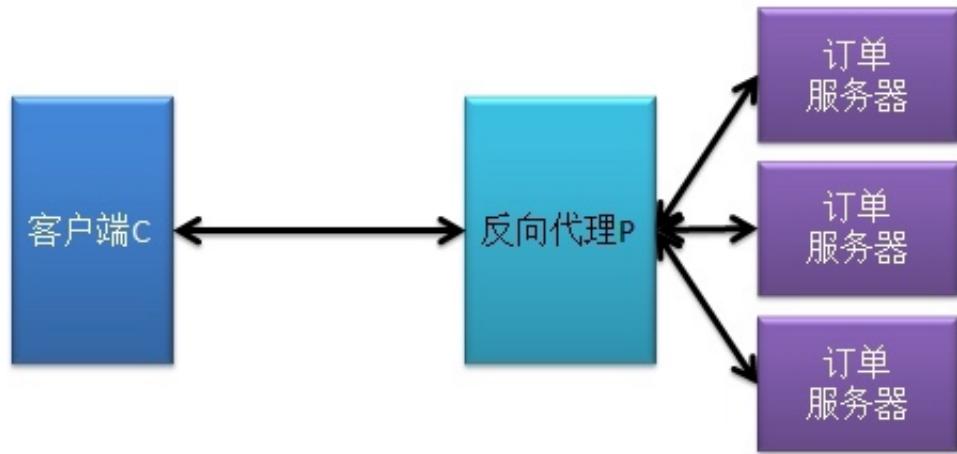
根据用户请求的URI调度到不同的功能的服务器进行处理。



反向代理服务器“分布式路由”示意图

2、负载均衡

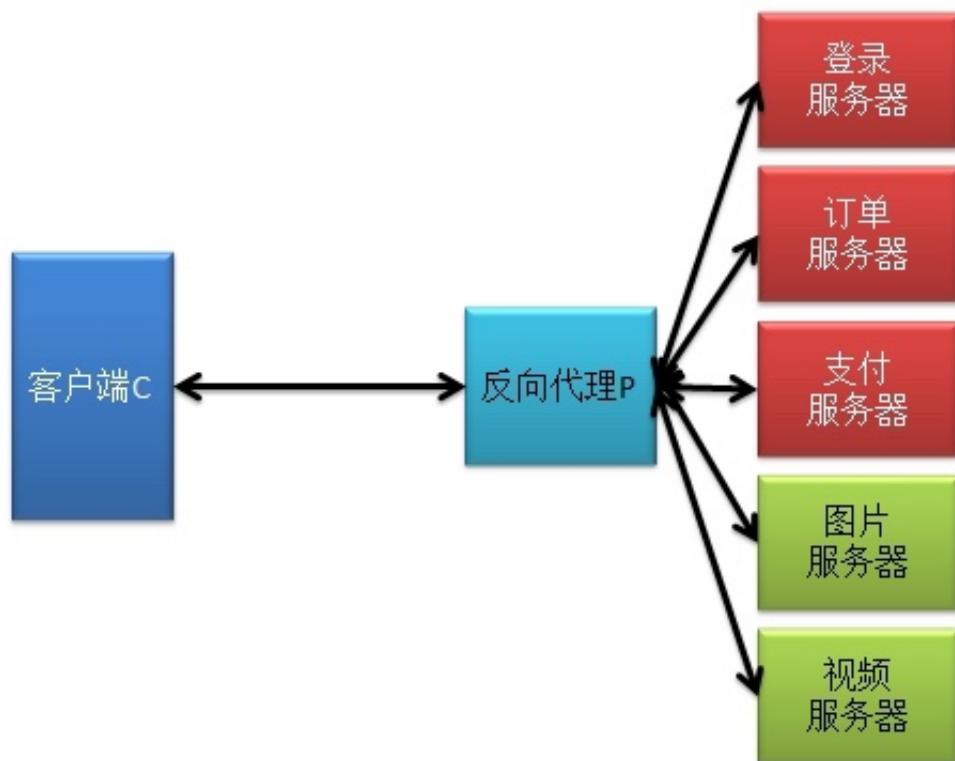
将用户发送的请求，通过负载均衡调度算法挑选一台合适的节点进行请求处理。



反向代理服务器“集群负载均衡”示意图

3、动静分离

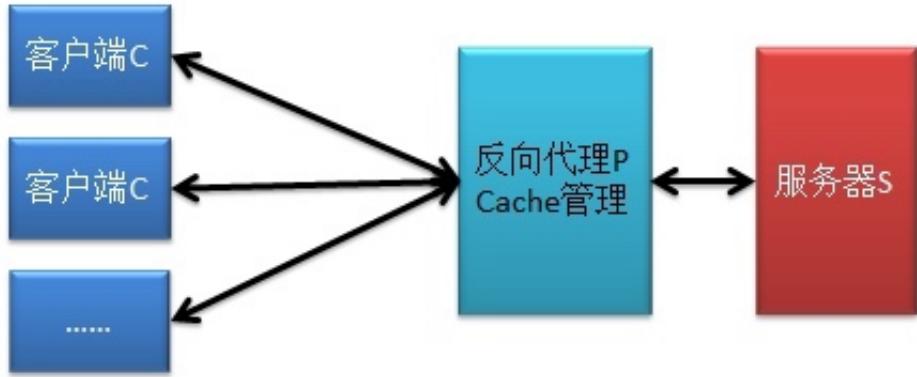
根据用户请求的URI进行区分，将动态资源调度至应用服务器处理，将静态资源调度至静态资源服务器处理。



反向代理服务器“动静分离”示意图

4、数据缓存

将后端查询的数据存储至反向代理上缓存，可以加速用户获取资源。



反向代理服务器“缓存”功能示意图

2.3) 正向与反向代理区别

区别在于形式上服务的“对象”不一样、其次架设的位置点不一样

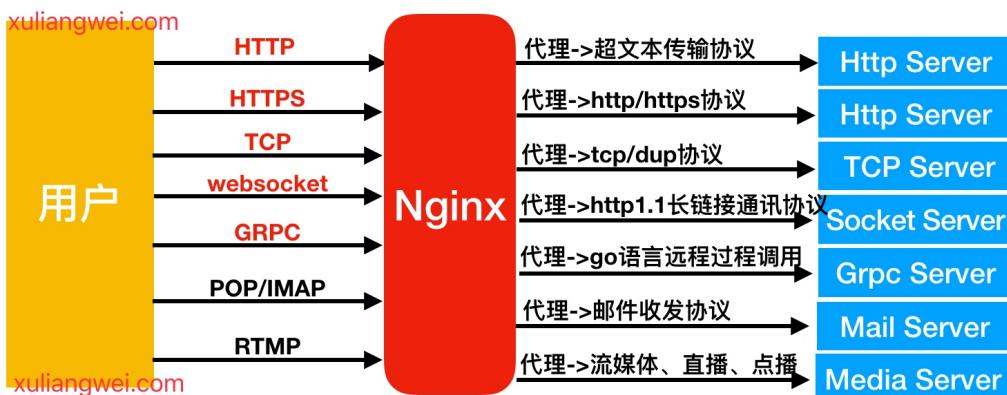
正向代理代理的对象是客户端，为客户端服务

反向代理代理的对象是服务端，为服务端服务

3.Nginx代理服务支持协议

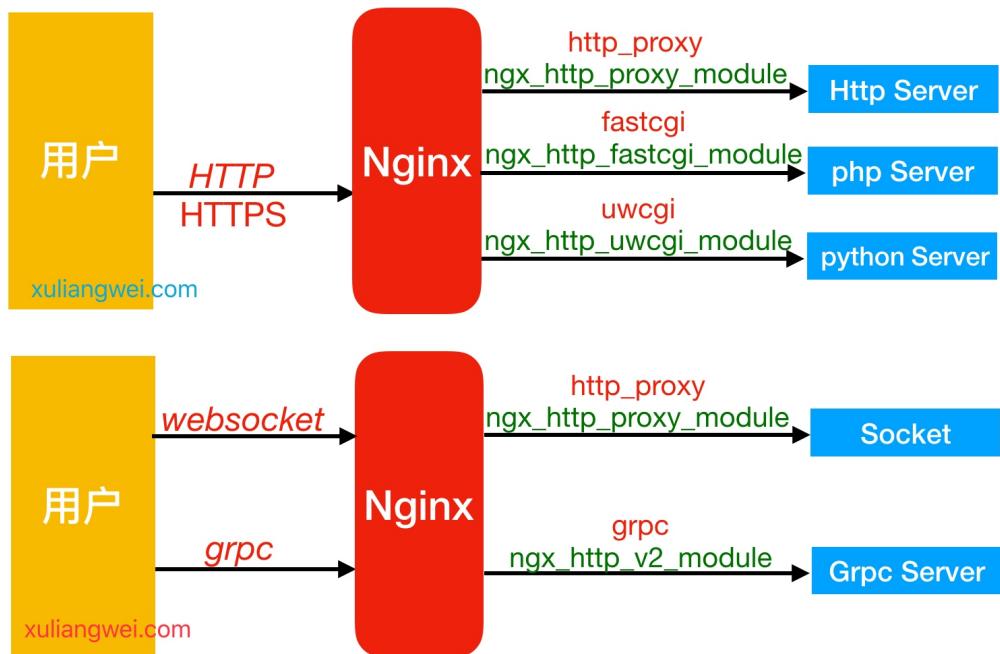
1.Nginx作为代理服务，支持的代理协议非常的多，具体如下图

Nginx代理可支持的代理协议



2.但通常情况下，我们将Nginx作为反向代理，常常会用到如下几种代理协议，如下图所示

Nginx作为反向代理常用代理协议



4.Nginx反向代理场景实践

1.Nginx反向代理配置语法示例

```
Syntax: proxy_pass URL;  
Default: -  
Context: location, if in location, limit_except
```

```
http://localhost:8000/uri/  
http://192.168.56.11:8000/uri/  
http://unix:/tmp/backend.socket/:uri/
```

2.Nginx反向代理配置实例



角色	外网IP(NAT)	内网IP(LAN)	主机名
Proxy	eth0:10.0.0.5	eth1:172.16.1.5	lb01
web01		eth1:172.16.1.7	web01

3.web01服务器, 配置一个网站, 监听在8080, 此时网站仅172网段的用户能访问

```
[root@web01 ~]# cd /etc/nginx/conf.d/web.oldxu.com.conf
server {
    listen 8080;
    server_name web.oldxu.com;

    location / {
        root /code_8080;
        index index.html;
    }
}

[root@web01 conf.d]# mkdir /code_8080
[root@web01 conf.d]# echo "web01-7..." >/code_8080/index.html
[root@web01 conf.d]# systemctl restart nginx
```

4.proxy代理服务器, 配置监听eth0的80端口, 使10.0.0.0网段的用户, 能够通过代理服务器访问到后端的172.16.1.7的8080端口站点内容

```
[root@lb01 conf.d]# cat /etc/nginx/conf.d/proxy_web.oldxu.com.conf
server {
    listen 80;
    server_name web.oldxu.com;

    location / {
        proxy_pass http://172.16.1.7:8080;
    }
}

[root@lb01 conf.d]# systemctl enable nginx
[root@lb01 conf.d]# systemctl start nginx
```

4. 抓包分析Nginx代理处理整个请求的过程

5. 添加发往后端服务器的请求头信息, 如图

```
Syntax: proxy_set_header field value;
Default:   proxy_set_header Host $proxy_host;
           proxy_set_header Connection close;
Context:   http, server, location
```

```
# 客户端请求Host的值是www.oldxu.com, 那么代理服务会像后端请求时携带Host变量为www.oldxu.com
```

```
proxy_set_header Host $http_host;
# 将$remote_addr的值放进变量X-Real-IP中, $remote_addr的值为客户端的ip
proxy_set_header X-Real-IP $remote_addr;
# 客户端通过代理服务访问后端服务, 后端服务会通过该变量会记录真实客户端地址
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

6. 代理的向后端请求时使用的HTTP协议版本。默认1.0版本。如果使用长连接，建议调整为1.1版本协议。

```
Syntax: proxy_http_version 1.0 | 1.1;
Default: proxy_http_version 1.0;
Context: http, server, location
This directive appeared in version 1.1.4.
```

7. 代理到后端的TCP连接、响应、返回等超时时间, 如图

```
#nginx代理与后端服务器连接超时时间(代理连接超时)
Syntax: proxy_connect_timeout time;
Default: proxy_connect_timeout 60s;
Context: http, server, location

#nginx代理等待后端服务器响应 (Header) 的超时时间
Syntax: proxy_read_timeout time;
Default: proxy_read_timeout 60s;
Context: http, server, location

#后端服务器数据 (Data) 回传给nginx代理超时时间
Syntax: proxy_send_timeout time;
Default: proxy_send_timeout 60s;
Context: http, server, location
```

8.proxy_buffer代理缓冲区

1) 启用缓冲时, nginx代理服务器将尽快的接收响应Header以及响应报文，并将其保存到proxy_buffer_size(Headers) 和 proxy_buffers(data) 设置的缓冲区中。

```
Syntax: proxy_buffering on | off;
Default: proxy_buffering on;
Context: http, server, location
```

如果响应报文过大无法存储至内存，则会将其中部分保存到磁盘上的临时文件中。写入临时文件由proxy_temp_path (控制临时存储目录) proxy_max_temp_file_size (控制临时存储目录大

小) 和 proxy_temp_file_write_size (控制一次写入临时文件的数据大小)，临时文件最大大小由 proxy_buffer_size 和 proxy_buffers 限制。但当禁用缓冲时，nginx代理服务器会在接收到响应时立即同步传递给客户端。nginx代理服务器不会读取整个响应。

2) proxy_buffer_size 用于控制代理服务读取后端第一部分响应Header的缓冲区大小。

```
Syntax: proxy_buffer_size size;
Default: proxy_buffer_size 4k|8k;
Context: http, server, location

#proxy_buffer_size 64k;
```

3) proxy_buffers 是代理服务器为单个连接设置响应缓冲区“数量”和“大小”。

如果一个后端服务所返回的页面大小为256KB，那么会为其分配4个64KB的缓冲区来缓存，如果页面大小大于256KB，那么大于256KB的部分会缓存到 proxy_temp_path 指定的路径中。但是这并不是好方法，因为内存中的数据处理速度要快于硬盘。所以这个值一般建议设置为站点响应所产生的页面大小中间值，如果站点大部分脚本所产生的页面大小为256KB，那么可以把这个值设置为“16 16k”、“4 64k”等。

```
Syntax: proxy_buffers number size;
Default: proxy_buffers 8 4k|8k;
Context: http, server, location

#proxy_buffers 4 64k;
```

9. 代理网站常用优化配置如下，将配置写入新文件，调用时使用include引用即可

```
[root@Nginx ~]# vim /etc/nginx/proxy_params
proxy_http_version 1.1;
proxy_set_header Host $http_host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

proxy_connect_timeout 30;
proxy_send_timeout 60;
proxy_read_timeout 60;

proxy_buffering on;
proxy_buffer_size 64k;
proxy_buffers 4 64k;
```

10. 代理配置location时调用，方便后续多个Location重复使用。

```
location / {  
    proxy_pass http://127.0.0.1:8080;  
    include proxy_params;  
}
```

06.Nginx七层负载均衡

1.Nginx负载均衡基本概述

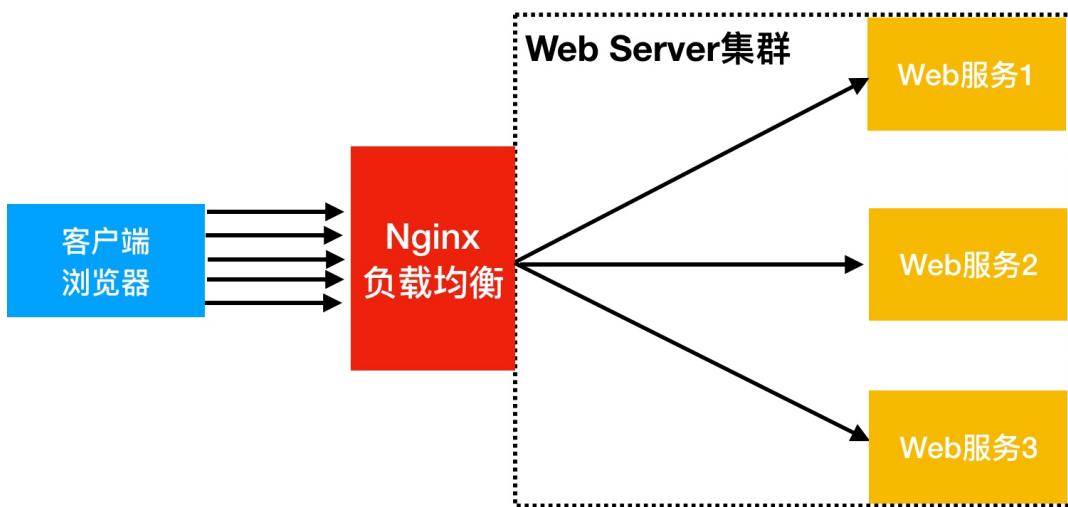
1.1 什么是负载均衡

负载均衡 Load Balance , 指的是将用户访问请求所产生的流量，进行平衡，分摊到多个应用节点处理。

负载均衡扩展了应用的服务能力，增强了应用的可用性。

1.2 为什么需要负载均衡

当我们的 Web 服务器直接面向用户，往往要承载大量并发请求，单台服务器难以负荷，我使用多台 WEB 服务器组成集群，前端使用 Nginx 负载均衡，将请求分散的打到我们的后端服务器集群中，实现负载的流量分发。从而提升整体性能、以及系统的容灾能力。



1.3 负载均衡与代理区别

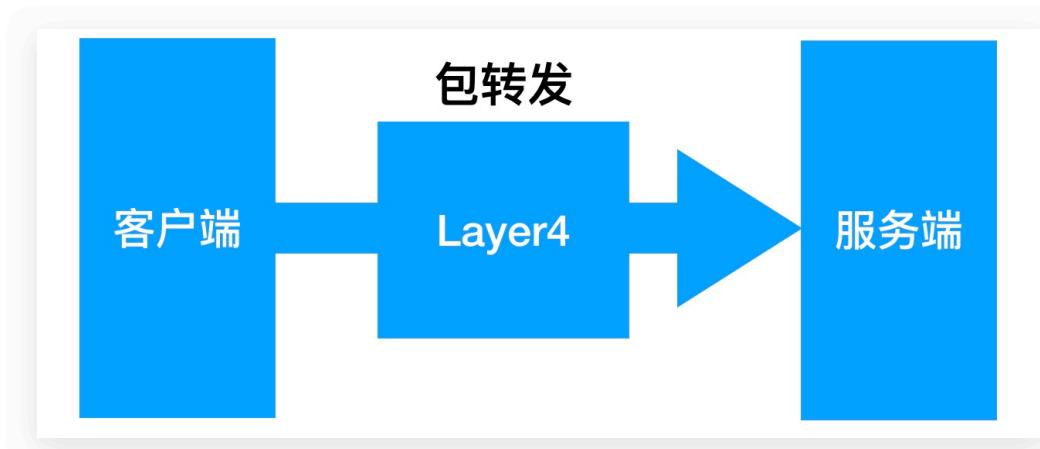
- Nginx 负载均衡与 Nginx 反向代理不同地方在于：
 - Nginx 代理仅代理一台服务器。
 - Nginx 负载均衡则是将客户端请求通过 proxy_pass 代理至一组 upstream 资源池。

2.Nginx负载均衡应用场景

2.1 四层负载均衡

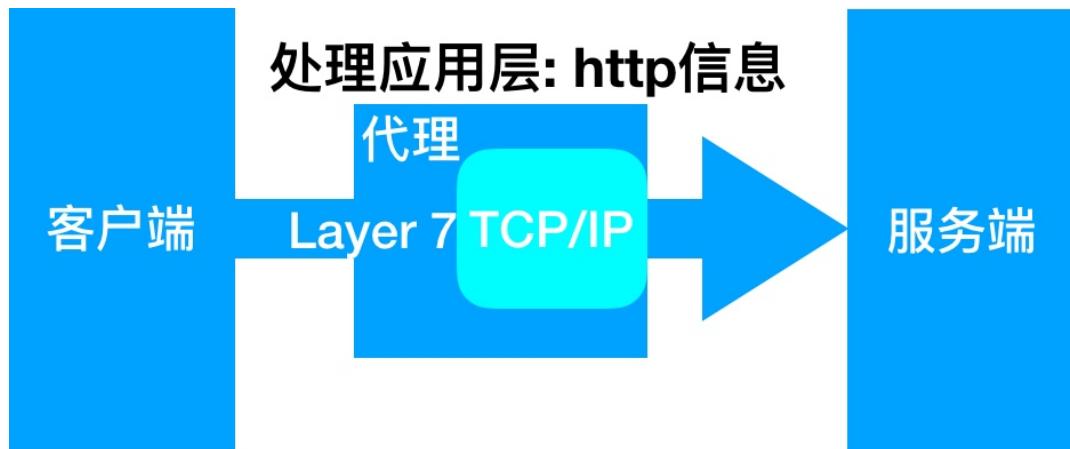
四层负载均衡指的是 OSI 七层模型中的传输层，四层仅需要对客户端的请求进行 TCP/IP 协议的包转发就可以实现负载均衡。

四层负载均衡的性能极好、因为只需要底层进行转发处理，而不需要进行一些复杂的逻辑。



2.2 七层负载均衡

七层负载均衡工作在应用层，它可以完成很多应用方面的协议请求，比如我们说的 http 应用负载均衡，它可以实现 http 头信息的改写、安全应用规则控制、URI 匹配规则控制、及 rewrite 等功能，所以在应用层里面可以做的内容就更多了。



2.3 四层与七层区别

- 四层负载均衡：传输层
 - 优点：性能高，数据包在底层就进行了转发
 - 缺点：仅支持 ip:prot 转发，无法完成复杂的业务逻辑应用。
- 七层负载均衡：应用层
 - 优点：贴近业务，支持 URI 路径匹配、Header 改写、Rewrite 等
 - 缺点：性能低，数据包需要拆解到顶层才进行转发

3.Nginx负载均衡配置场景

- Nginx 实现负载均衡需要两个模块：
 - proxy_pass 代理模块
 - upstream 虚拟资源池模块

Syntax: `upstream name { ... }`

Default: -

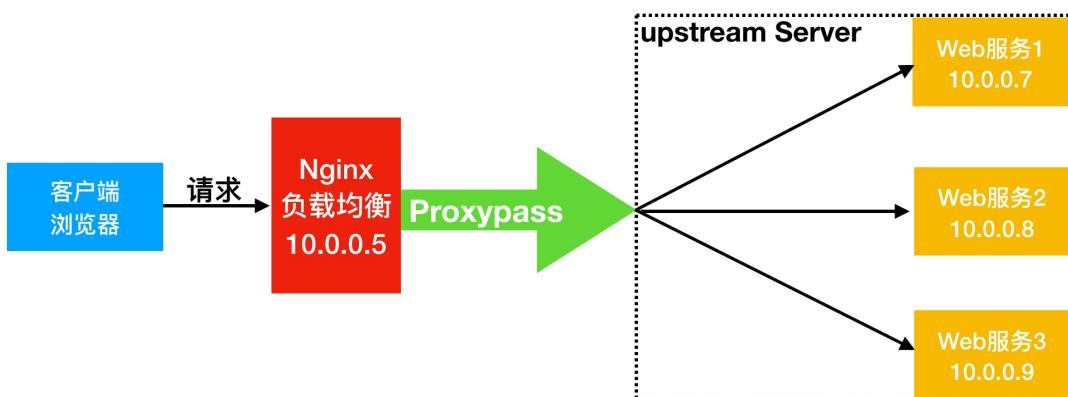
Context: http

#upstream例

```
upstream backend {
    server backend1.example.com      weight=5;
    server backend2.example.com:8080;
    server unix:/tmp/backend3;
    server backup1.example.com:8080  backup;
}
server {
    location / {
        proxy_pass http://backend;
    }
}
```

3.1 负载均衡场景环境规划

- 负载均衡场景架构图规划



- 负载均衡场景地址规划

角色	外网IP(NAT)	内网IP(LAN)	主机名
LB01	eth0:10.0.0.5	eth1:172.16.1.5	lb01
web01	eth0:10.0.0.7	eth1:172.16.1.7	web01

```
web02 | eth0:10.0.0.8 | eth1:172.16.1.8 | web02
```

3.2 后端Web节点配置实例

1. Web01 服务器上配置为应用服务节点, 创建对应 html 文件

```
[root@web01 ~]# cd /etc/nginx/conf.d/web.oldxu.com.conf
server {
    listen 80;
    server_name web.oldxu.com;
    root /web;

    location / {
        index index.html;
    }
}

[root@web01 conf.d]# mkdir /web
[root@web01 conf.d]# echo "Web01..." > /node/index.html
[root@web01 conf.d]# systemctl restart nginx
```

2. Web02 服务器上配置为应用服务节点, 创建对应 html 文件

```
[root@web02 ~]# cat /etc/nginx/conf.d/web.oldxu.com.conf
server {
    listen 80;
    server_name web.oldxu.com;
    root /web;

    location / {
        index index.html;
    }
}

[root@web02 conf.d]# mkdir /web
[root@web02 conf.d]# echo "Web02..." > /node/index.html
[root@web02 conf.d]# systemctl restart nginx
```

3.3. 前端接入Nginx负载均衡

1. 将 lb01 配置为负载均衡, 将所有请求代理至虚拟资源池

```
[root@lb01 ~]# cat /etc/nginx/conf.d/proxy_web.oldxu.com.conf
upstream web {
```

```
server 172.16.1.7:80;
server 172.16.1.8:80;
}
server {
    listen 80;
    server_name web.oldxu.com;

    location / {
        proxy_pass http://web;
        include proxy_params;
    }
}
[root@lb01 conf.d]# systemctl restart nginx
```

2.准备 Nginx 负载均衡需要使用的 proxy_params 文件

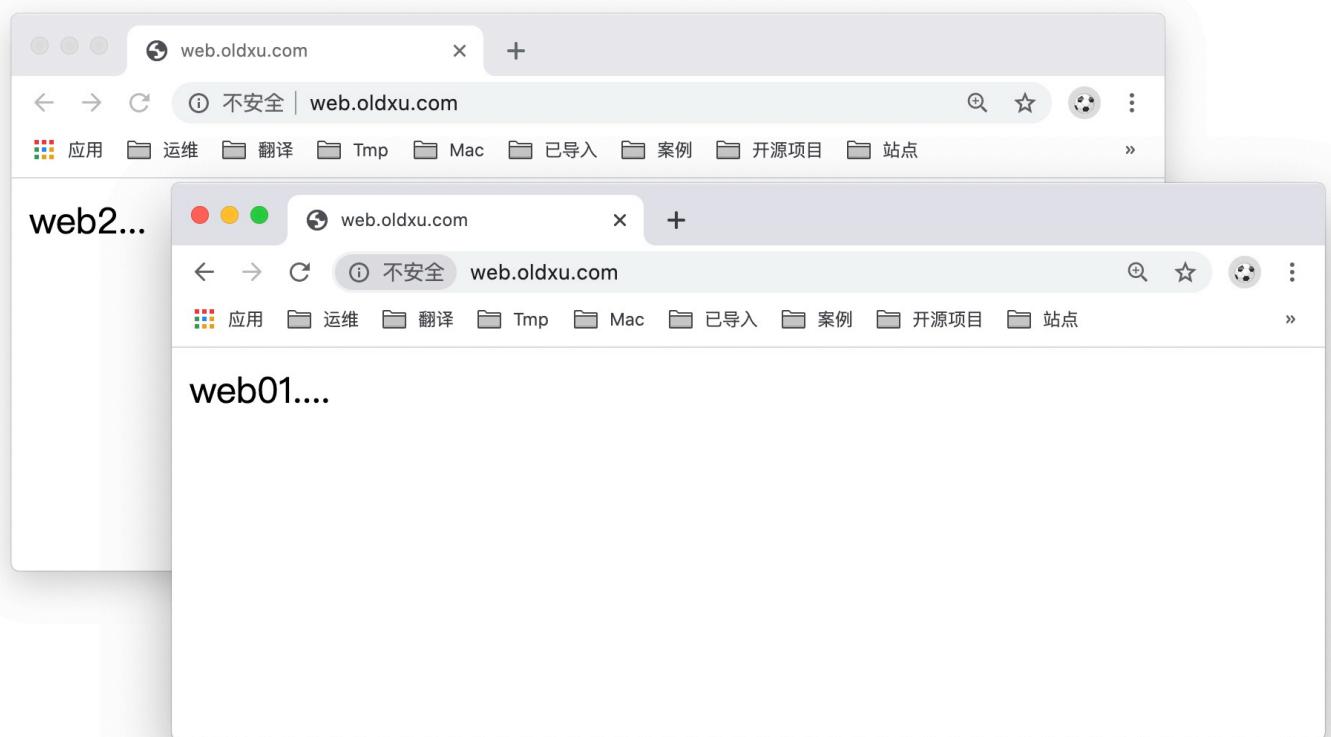
```
[root@Nginx ~]# vim /etc/nginx/proxy_params
proxy_set_header Host $http_host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

proxy_connect_timeout 30;
proxy_send_timeout 60;
proxy_read_timeout 60;

proxy_buffering on;
proxy_buffer_size 64k;
proxy_buffers 4 64k;
```

3.4.浏览器访问测试负载效果

使用浏览器访问 web.oldxu.com，然后进行刷新测试。



4.Nginx负载均衡调度算法

调度算法	概述
轮询	按时间顺序逐一分配到不同的后端服务器(默认)
weight	加权轮询,weight值越大,分配到的访问几率越高
ip_hash	每个请求按访问IP的hash结果分配,这样来自同一IP的固定访问一个后端服务器
least_conn	将请求传递到活动连接数最少的服务器。

4.1 轮询调度算法

轮询调度算法的原理是将每一次用户的请求，轮流分配给内部中的服务器。

轮询算法的优点是其简洁性，它无需记录当前所有连接的状态，所以它是一种无状态调度。

```
upstream load_pass {
    server 172.16.1.7:80;
    server 172.16.1.8:80;
}
server {
```

```
listen 80;
server_name web.oldxu.com;

location / {
    proxy_pass http://load_pass;
    include proxy_params;
}
```

4.2 加权轮询调度算法

轮询调度算法没有考虑每台服务器的处理能力，在实际情况中，由于每台服务器的配置、安装的业务应用等不同，其处理能力会不一样。所以，我们根据服务器的不同处理能力，给每个服务器分配不同的权值，使其能够接受相应权值数的服务请求。

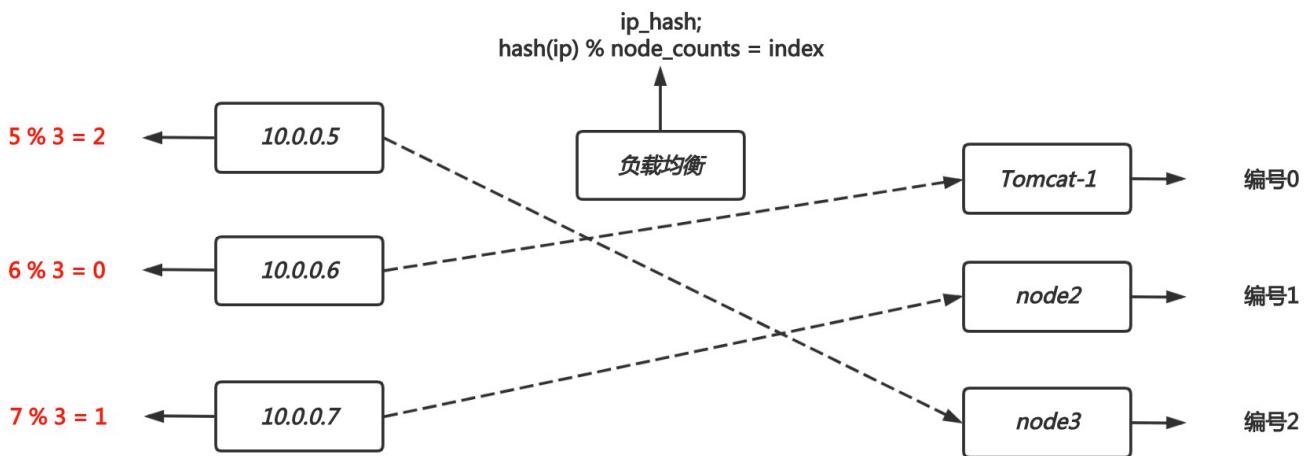
```
upstream load_pass {
    server 172.16.1.7:80 weight=5;
    server 172.16.1.8:80 weight=1;
}
server {
    listen 80;
    server_name web.oldxu.com;

    location / {
        proxy_pass http://load_pass;
        include proxy_params;
    }
}
```

4.3 ip_hash调度算法

`ip_hash` 是基于用户请求的 `IP`，对该 `IP` 进行 `hash` 运算，根据 `hash` 运算的值，将请求分配到后端特定的一台节点进行处理。`ip_hash` 算法实现公式：`hash(ip) % node_counts = index`

Hash算法实现原理

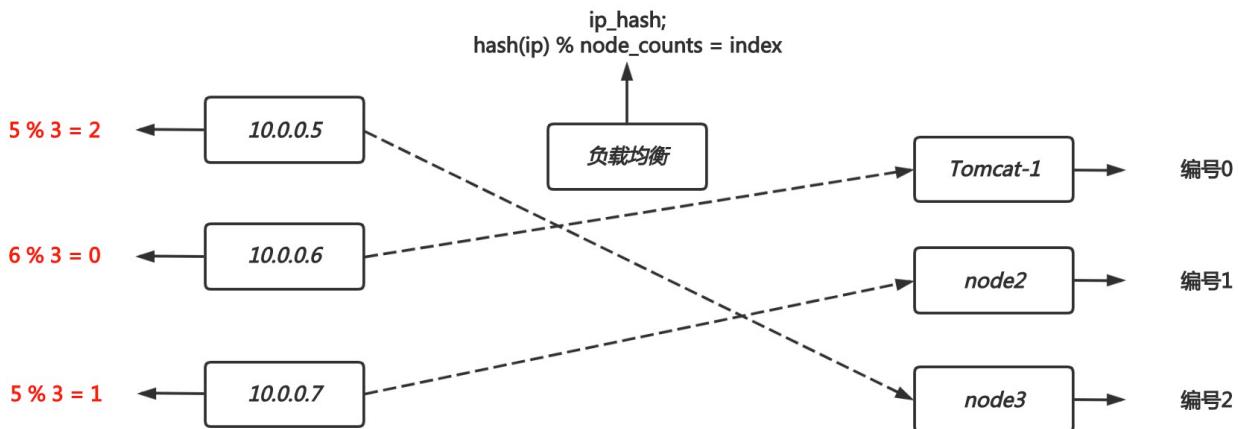


如何配置 ip_hash 调度算法

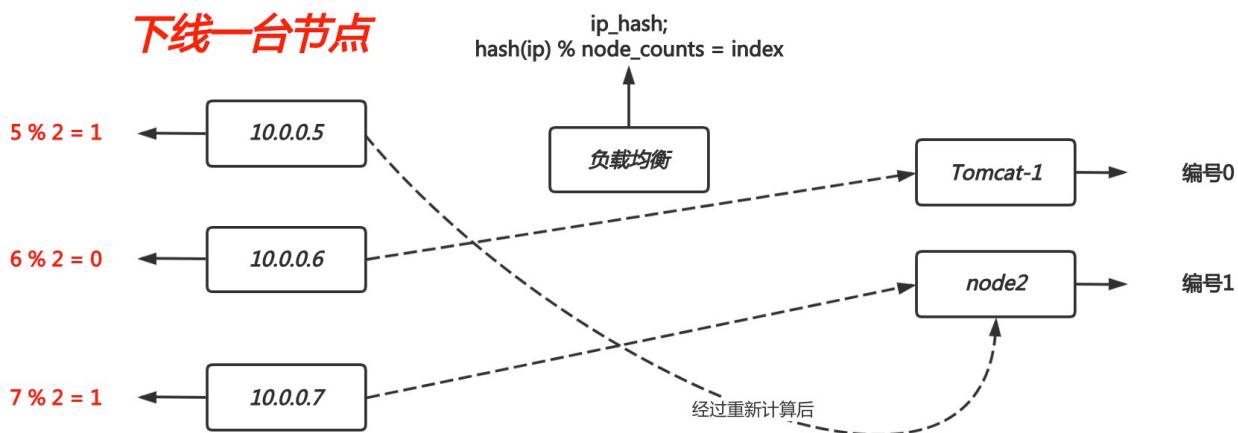
```
upstream load_pass {  
    ip_hash;  
    server 172.16.1.7:80;  
    server 172.16.1.8:80;  
}  
server {  
    listen 80;  
    server_name web.oldxu.com;  
  
    location / {  
        proxy_pass http://load_pass;  
        include proxy_params;  
    }  
}
```

- `ip_hash` 调度算法会带来两个问题
 - 1.如果有大量来自于同一IP的请求会造成某个后端节点流量过大，而其他节点无流量
 - 2.如果临时下线一台节点，会出现重新计算 `hash` 值，官方建议将下线节点标记为 `down` 状态，以保留客户端 IP 地址的当前哈希值。 (如下图所示：)

Hash算法问题



下线一台节点



如果有大量的用户调度到某一节点，而该节点刚好故障，则该算法会重新计算结果，从而造成大量的用户被转移其他节点处理，而需要重新建立会话。

4.4 一致性hash调度算法

为了规避上述 hash 情况，一致性 hash 算法就诞生，一致性 Hash 算法也是使用取模的方法，但不是对服务器节点数量进行取模，而是对 2^{32} 方取模。即，一致性 Hash 算法将整个 Hash 空间组织成一个虚拟的圆环，Hash 函数值的空间为 $0 \sim 2^{32} - 1$ ，整个哈希环如下：

- [一致性Hash文档](#)
- [一致性Hash参考地址](#)
 - Hash算法原理
 - Hash算法增加节点
 - Hash算法减少节点
 - Hash算法数据倾斜问题

4.5 url_hash调度算法

根据用户请求的 URL 进行 hash 取模，根据 hash 运算的值，将请求分配到后端特定的一台节

点进行处理。`URL` 算法使用场景如下：`client-->nginx-->url_hash-->cache1-->app`

- 1. 用户请求 `nginx` 负载均衡器，通过 `url` 调度算法，将请求调度至 `Cache1`
- 2. 由于 `Cache1` 节点没有对应的缓存数据，则会请求后端获取，然后返回数据，并将数据缓存起来；
- 3. 当其他用户再次请求此前相同的 `URL` 时，此时调度器依然会调度至 `Cache1` 节点处理；
- 5. 由于 `Cache1` 节点已存在该 `URL` 资源缓存，所以直接将缓存数据进行返回；能大幅提升网站的响应；

1. 配置后端节点

```
# web1 节点
[root@web01 ~]# echo "web1 Url1" > /web/url1.html
[root@web01 ~]# echo "web1 Url2" > /web/url2.html

# web2 节点
[root@web02 ~]# echo "web2 Url1" > /web/url1.html
[root@web02 ~]# echo "web2 Url2" > /web/url2.html
```

2. 负载均衡配置 `url_hash` 调度算法

```
upstream load_pass {
    # 请求同一个url，会始终定向到同一个服务器节点，consistent表示使用一致性hash算法
    hash $request_uri consistent;
    server 172.16.1.7:80;
    server 172.16.1.8:80;
}
server {
    listen 80;
    server_name web.oldxu.com;

    location / {
        proxy_pass http://load_pass;
        include proxy_params;
    }
}
```

3. Client 测试，会发现请求相同的 `URL`，始终会被定向至某特定后端节点。

```
[root@client ~]# curl -HHost:web.oldxu.com http://10.0.0.5/url1.html
web2 Url1
[root@client ~]# curl -HHost:web.oldxu.com http://10.0.0.5/url1.html
web2 Url1
[root@client ~]# curl -HHost:web.oldxu.com http://10.0.0.5/url1.html
```

4.6 least_conn调度算法

least_conn调度算法实现原理，哪台节点连接数少，则将请求调度至哪台节点。

假设：A节点有1000个连接、b节点有500连接，如果此时新的连接进入会分发给b节点

```
upstream load_pass {
    least_conn;
    server 172.16.1.7:80;
    server 172.16.1.8:80;
}
server {
    listen 80;
    server_name web.oldxu.com;

    location / {
        proxy_pass http://load_pass;
        include proxy_params;
    }
}
```

5.Nginx负载均衡后端状态

后端 Web 节点在前端 Nginx 负载均衡调度中的状态

状态	概述
down	当前的server暂时不参与负载均衡
backup	预留的备份服务器
max_fails	允许请求失败的次数
fail_timeout	经过max_fails失败后，服务暂停时间
max_conns	限制最大的接收连接数

5.1 max_conns限制连接数

max_conns 用来限制每个后端节点能够接收的最大TCP连接数，如果超过此连接则会抛出错误。

```
[root@lb01 ~]# cat proxy_web.oldxu.com.conf
upstream load_pass {
```

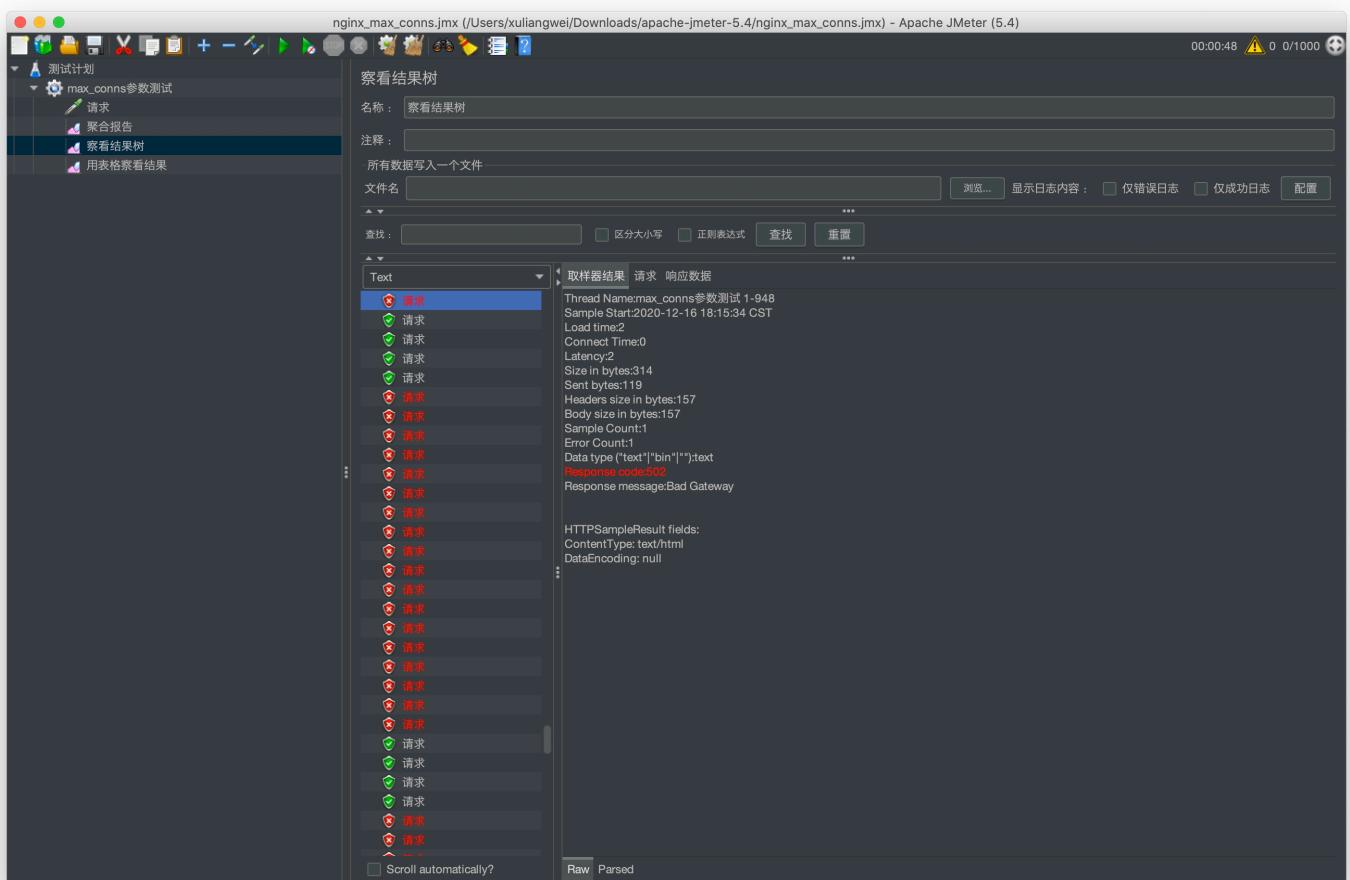
```

server 172.16.1.7:80 max_conns=2;
server 172.16.1.8:80 max_conns=2;
}
server {
    listen 80;
    server_name web.oldxu.com;

    location / {
        proxy_pass http://load_pass;
        include proxy_params;
    }
}

```

通过 `jmeter` 压力测试发现，当后端节点处理的连接数非常多的时候，当大于4个连接，其余的连接则会抛出异常，也就是每次仅能满足4个连接。[jmter-max_conns测试用例](#)



5.2 down标识关闭状态

`down` 将服务器标记为不可用状态。

```
[root@lb01 ~]# cat proxy_web.oldxu.com.conf
upstream load_pass {
    server 172.16.1.7:80 down;  #一般用于停机维护
    server 172.16.1.8:80;
```

```
}

server {
    listen 80;
    server_name web.oldxu.com;

    location / {
        proxy_pass http://load_pass;
        include proxy_params;
    }
}
```

5.3 backup标识备份状态

`backup` 将服务器标记为备份服务器。当主服务器不可用时，将请求传递至备份服务器处理。

```
[root@lb01 ~]# cat proxy_web.oldxu.com.conf
upstream load_pass {
    server 172.16.1.7:80 backup;
    server 172.16.1.8:80;
    server 172.16.1.9:80;
}
server {
    listen 80;
    server_name web.oldxu.com;

    location / {
        proxy_pass http://load_pass;
        include proxy_params;
    }
}
```

5.4 max_fails与fail_timeout

`max_fails=2` 服务器通信失败尝试2次，任然失败，认为服务器不可用；

`fail_timeout=5s` 服务器通信失败后，每5s探测一次节点是否恢复可用；

在 `fail_timeout` 设定的时间内，与服务器连接失败达到 `max_fails` 则认为服务器不可用；

```
[root@lb01 ~]# cat proxy_web.oldxu.com.conf
upstream load_pass {
    server 172.16.1.7:80 max_fails=2 fail_timeout=5s;
    server 172.16.1.8:80 max_fails=2 fail_timeout=5s;
}
server {
    listen 80;
    server_name web.oldxu.com;
```

```
location / {  
    proxy_pass http://load_pass;  
    include proxy_params;  
}
```

6.Nginx负载均衡会话共享

6.1 什么是会话保持

当用户登陆一个网站服务器，网站服务器会将用户的登陆信息存储下来（存储下来的内容叫Session），以保证我们能够一直处于“登陆在线”状态。

6.2 为什么需要会话保持

由于我们使用的是负载均衡轮询机制，会导致用户请求分散在不同的节点，从而造成会话无法保持。

假设用户A，通过负载均衡登陆了网站，此时会话信息存储在A节点，那么当它一刷新，负载均衡会将请求分发给B节点，那么B节点没有用户A的登陆信息，就会提示用户A登陆，当A用户点击登陆时又会将请求分发给C节点，从而造成用户A无法实现会话保持。

6.3 如何实现会话保持

- 1.粘性session：指Nginx每次都将同一用户的所有请求转发至同一台服务器上，及Nginx的IP_hash。
- 2.session复制：每次session发生变化，就广播给集群中的服务器，使所有的服务器上的session相同。
- 3.session共享：缓存session至内存数据库中，使用redis, memcached实现。
- 4.session持久化：将session存储至数据库中，像操作数据一样操作session。

6.4 会话保持场景演示

6.4.1 配置web节点

1.首先安装并配置 phpmyadmin

```
[root@web01 conf.d]# cd /code  
[root@web01 code]# wget https://files.phpmyadmin.net/phpMyAdmin/4.8.4/phpMyAdmin-4.  
8.4-all-Languages.zip  
[root@web01 code]# unzip phpMyAdmin-4.8.4-all-Languages.zip
```

2.修改 phpmyadmin 连接远程的数据库

```
[root@web01 code]# cd phpMyAdmin-4.8.4-all-languages/
[root@web01 phpMyAdmin-4.8.4-all-languages]# cp config.sample.inc.php config.inc.php
[root@web01 phpMyAdmin-4.8.4-all-languages]# vim config.inc.php
/* Server parameters */
$cfg['Servers'][$i]['host'] = '172.16.1.51';
```

3.在多台 web 上准备 phpmyadmin 的 nginx 配置文件*

```
[root@web01 ~]# cat /etc/nginx/conf.d/php.conf
server {
    listen 80;
    server_name php.oldxu.com;
    root /code/phpMyAdmin-4.8.4-all-languages;

    location / {
        index index.php index.html;
    }

    location ~ \.php$ {
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }
}

#重启Nginx服务
[root@web01 ~]# systemctl restart nginx
```

6.4.2 配置负载均衡

1.编写一份 proxy 负载均衡的配置文件，将请求调度到后端 web 节点

```
[root@lb01 ~]# cat /etc/nginx/conf.d/proxy_php.conf
upstream php {
    server 172.16.1.7:80;
    server 172.16.1.8:80;
}
server {
    listen 80;
    server_name php.oldxu.com;
    location / {
        proxy_pass http://php;
```

```
    include proxy_params;
}
}
```

2.检查语法并重载 nginx

```
[root@lb01 conf.d]# nginx -t
[root@lb01 conf.d]# systemctl restart nginx
```

6.4.3 配置Redis服务

1.安装 redis 内存数据库

```
[root@db01 ~]# yum install redis -y
```

2.配置 redis 监听在本地的内网网卡上

```
[root@db01 ~]# sed -i '/^bind/c bind 127.0.0.1 172.16.1.51' /etc/redis.conf
```

3.启动 redis

```
[root@db01 ~]# systemctl start redis
[root@db01 ~]# systemctl enable redis
```

6.4.4 配置php连接Redis

1.修改 /etc/php.ini 文件。[所有节点都需要操作]

```
[root@web ~]# vim /etc/php.ini
session.save_handler = redis
session.save_path = "tcp://172.16.1.51:6379"
;session.save_path = "tcp://172.16.1.51:6379?auth=123" #如果redis存在密码，则使用该方式
```

2.注释 php-fpm.d/www.conf 里面的两条内容，否则 session 内容会一直写入 /var/lib/php/session 目录中，从而造成会话共享失败。[所有节点都需要操作]

```
[root@web ~]# vim /etc/php-fpm.d/www.conf
;php_value[session.save_handler] = files
;php_value[session.save_path]      = /var/lib/php/session
```

3.重启 php-fpm 服务。[所有节点都需要操作]

```
[root@web ~]# php-fpm -t  
[root@web ~]# systemctl restart php-fpm
```

6.4.5 测试集群会话共享

1.使用浏览器登陆网站，获取对应的 cookie 信息

The screenshot shows a browser window with the URL `php.bgx.com / 172.16.1.51 | ph...`. The main content is the PHPMyAdmin login screen. On the left, there's a sidebar with database links like information_schema, jpress, mysql, performance_schema, test, wecenter, wordpress, and zabbix. The main area has tabs for常规设置 (General Settings) and 外观设置 (Appearance Settings). The right side has sections for 数据库服务器 (Database Server) and 网站服务器 (Website Server), both listing details about the MySQL connection. Below the main content, the browser's developer tools are open, specifically the Network tab. It shows a table of session cookies for the domain `http://php.bgx.com`. One cookie, `PHPSESSID_xlw`, is highlighted.

Name	Value	Domain	Path	Expires / Max-Age	Size	HTTP	Secure	SameSite
PHPSESSID_xlw	13f6c312e8c0bb1d6789bacf7c58bfe	php.bg...	/	1969-12-31T23:59:59.000Z	44			
phpMyAdmin	393ff522ed2a7e26ba44f6d925f991f2	php.bg...	/	1969-12-31T23:59:59.000Z	42	✓		
pmaAuth-1	967B%22iv%22%3A%22bctVog3OEbwSWGK3uM...	php.bg...	/	1969-12-31T23:59:59.000Z	192	✓		
pmaUser-1	%67B%22iv%22%3A%2261%2B4nFm%5C%2FmwbX...	php.bg...	/	2019-01-25T09:48:06.855Z	181	✓		
pma_lang	zh_CN	php.bg...	/	2019-01-25T07:44:34.337Z	13	✓		

2.检查 redis 中是否存在 cookie 对应的 session 信息

```
[root@db01 ~]# redis-cli  
172.16.1.51:6379> keys *  
1) "PHPREDIS_SESSION:393ff522ed2a7e26ba44f6d925f991f2"  
172.16.1.51:6379>
```

3.此时用户的 cookie 始终都不会发生任何变化，无论请求被负载调度到那一台后端 web 节点服务器都不会出现没有登陆情况。

php.bgx.com / 172.16.1.51 / my +

① 不安全 | php.bgx.com/sql.php?server=1&db=mysql&table=help_topic&pos=0

应用 运维 开发 学习 写作 阅读 教育 公有云 翻译 Tmp 已导入 Google Linux运维 Linux架构 微信视频在线视频...

phpMyAdmin

近期访问 表收藏夹

服务器: 172.16.1.51 » 数据库: mysql » 表: help_topic "help topics"

浏览 结构 SQL 搜索 插入 导出 导入 权限 操作 触发器

行数: 25 过滤行: 在表中搜索 按索引排序: 无

+ 选项

help_topic_id	name	help_category_id	description	example	url
0	MIN	16	Syntax: MIN([DISTINCT] expr)	mysql> SELECT student_name, MIN (test_score), MAX (t...)	http://dev.mysql.com/doc/refman/8.0/en/mathematical-functions.html
1	JOIN	27	MySQL supports the following JOIN syntaxes for the...	SELECT left_tbl.* FROM left_tbl LEFT JOIN right_...	http://dev.mysql.com/doc/refman/8.0/en/join.html

控制台

Syntax: mysql>

Elements Console Sources Network Performance Memory Application Security Audits Adblock Plus

Session Storage IndexedDB Web SQL Cookies http://php.bgx.com

Cache Cache Storage Application Cache

Console What's New