目录

[1正向代理 1](#_Toc3841)

[1.1服务对象是客户端（vpn等） 1](#_Toc16749)

[1.2正向代理支持协议 1](#_Toc15527)

[2反向代理 2](#_Toc2283)

[2.1服务对象是服务端 2](#_Toc18931)

[2.2反向代理支持协议 2](#_Toc13906)

[2.3长见的反向代理模式与nginx代理模块 2](#_Toc16387)

[3 proxy\_pass用法 3](#_Toc24305)

[3.1 用于location 3](#_Toc5330)

[3.2 proxy\_pass url（url类型） 3](#_Toc10017)

[3.3 反向代理例子 3](#_Toc20624)

[3.4 正向代理例子 3](#_Toc32413)

[3.5 其他重要语法（proxy\_buffering,默认on，减少磁盘io？） 4](#_Toc21268)

[4 代理缓存（cdn?proxy\_cache） 4](#_Toc24776)

[4.1 实际语法 4](#_Toc24977)

[4.2 缓存命中分析upstream\_cache\_status 6](#_Toc16377)

[5 websocket代理(通过proxy\_pass) 6](#_Toc10029)

[5.1 协议定义及应用场景 6](#_Toc15421)

[5.2 协议原理 7](#_Toc26423)

[5.3 ws协议安装及运行 7](#_Toc2364)

[5.4 ws协议nginx配置 8](#_Toc22018)

[6 fastcgi代理 8](#_Toc17436)

[6.1 协议原理 8](#_Toc30137)

[6.2 配置语法 9](#_Toc16788)

[6.3 配置详解 9](#_Toc32417)

[6.4 fastcgi代理缓存（参考代理缓存） 10](#_Toc23954)

# 1正向代理

## 1.1服务对象是客户端（vpn等）

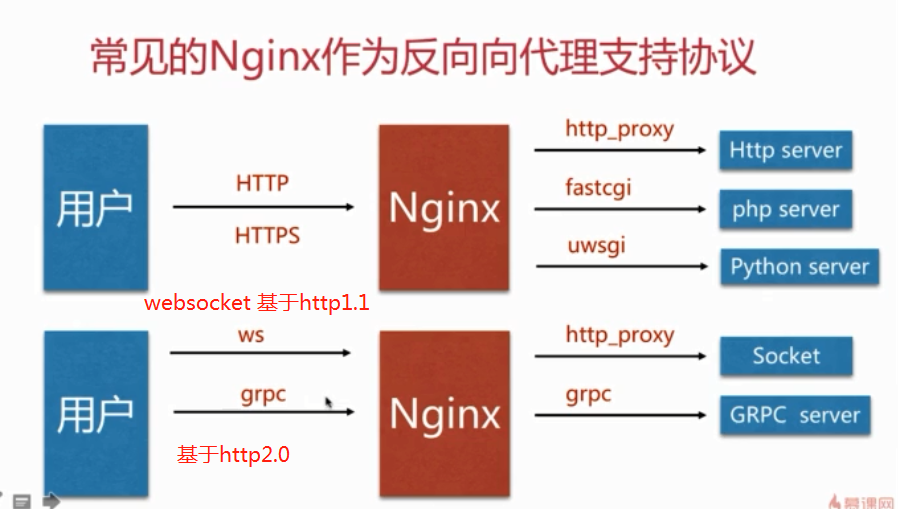
## 1.2正向代理支持协议

不支持https，只支持http

# 2反向代理

## 2.1服务对象是服务端

## 2.2反向代理支持协议



## 2.3长见的反向代理模式与nginx代理模块



# 3 proxy\_pass用法

## 3.1 用于location

## 3.2 proxy\_pass url（url类型）

<http://location:8080/uri>

[https://192.168.1.1:8080/uri](http://location:8080/uri)

<http://unix:/tmp/backend.socket:/uri/;>

## 3.3 反向代理例子

Location ~ /test\_proxy.html${

Proxy\_pass http://127.0.0.1:8080

}

## 3.4 正向代理例子

后端服务器配置（基于访问控制规则http\_x\_forwarded\_for）：

location /{

if($http\_x\_forwarded\_for !~\* "^116\.62\.103\.228"){

return 403;

}

root /opt/app/code;

index index.html index.htm;

}

正向代理配置（116.62.103.228）：

Server{

...

Resolver 8.8.8.8；#nginx用到的dns解析服务器

Location / {

proxy\_pass http://$http\_host$request\_uri;

}

...

}

## 3.5 其他重要语法（proxy\_buffering,默认on，减少磁盘io？）

**企业常用的做法是，将配置写入proxy\_params，在proxy\_pass后面加入include proxy\_params**

//proxy\_redirect default;//如果后端服务器返回301的时候，需要重写的时候，就需要用到这个参数，默认是default

***proxy\_set\_header Host $http\_host;（发给后端服务器的头）***

***//基于域名的虚拟主机必配***

***proxy\_set\_header X-Real-Ip $remote\_addr;（发给后端服务器的头）***

***//获取真实的ip***

//proxy\_connect\_timeout 30;//与后端服务器建立tcp请求的超时，默认60s

//proxy\_send\_timeout 60; //请求完成后，发送给客户端的超时时间

//proxy\_read\_timeout 60; //建立连接后，真实服务器去处理这个请求的时间

//proxy\_buffer\_size 32k;//指定后端 response 的 buffer 的大小。它是来自后端 response 的一部分，它包含 Headers，从 response 分离出来。它仅用于限定 headers 的 buffer 区，所以它的值比 proxy\_buffers 更低。proxy\_buffer\_size 有一点特殊在于，无论 proxy\_buffering 是否开启，proxy\_buffer\_size 都会起作用。

//proxy\_buffering on;//默认打开，是后端读取信息存放到buffer，然后再发给前端（边送边发）在 proxy\_buffering 开启的时候，proxy\_buffers 和 proxy\_busy\_buffers\_size 才会起作用。

//proxy\_buffers 4 128k;//指定一个 request 的 buffer 的数量和大小。

//proxy\_busy\_buffers\_size 256k;//忙时 buffer 的最大值。一个客户端一次只能从一个 buffer 中读取数据的同时，剩下的 buffer 会被放到队列中，等待发送到客户端，这个 directive 指定在这个状态下的 buffer 的大小。

//proxy\_max\_temp\_file\_size 256k;

# 4 代理缓存（cdn?proxy\_cache）

## 4.1 实际语法

upstream imooc{

server 116.12.12.12:8001;

server 116.12.12.12:8002;

server 116.12.12.12:8003;

}

proxy\_cache\_path /opt/app/cache levels=1:2 keys\_zone=***imooc\_cache***:10m max\_size=10g inactive=60m use\_temp\_path=off;

#levels 缓存目录的层级

#keys\_zone 开劈空间的名字，一般（$host$uri$is\_args$args）一兆可以存放8000个key

#max\_size 控制磁盘目录的最大值，当达到最大空间时候，就会触发淘汰规则。

#Inactive = 60，60分钟后如果没被人访问到就触发淘汰规则（清理掉缓存文件）

#use\_temp\_path，当我们自定义目录存放位置时候，就可以关闭它，因为如果两个目录都有，容易会引发一些性能问题。

server{

...

if($request\_uri ~ ^/(login|register|password)){

set $cookie\_nocache 1; #设置不缓存的名字

}

location /{

proxy\_cache ***imooc\_cache***;#应用名字，关闭缓存是设为off

proxy\_pass http://imooc;#基于代理开启缓存

proxy\_cache\_valid 200 304 12h;#200,304设置为12小时

proxy\_cache\_valid any 10m;# 其他10分钟过期时间

proxy\_cache\_key $host$uri$is\_args$args;#定义缓存key的维度

***proxy\_no\_cache $cookie\_nocache***;#应用不缓存的名字

add\_header Nginx-Cache "$upstream\_cache\_status";# 加到response头信息给客户(Nginx-Cache可以自定义名字)

proxy\_next\_upstream error timeout invalid\_header http\_500 http\_502 http\_503 http\_504;#当服务器返回错误，超时，不正确的头，还有500,502,503,504的时候，自动下一台服务器

include proxy\_params;

}

...

}

## 4.2 缓存命中分析upstream\_cache\_status

方式一 通过设置response的头信Nginx-Cache（设置服务端给客户端的头信息）

add\_header Nginx-Cache “$upstream\_cache\_status”;

方式二 通过设置log\_format,打印日志分析

```

http{

```

log format main ‘$remote\_addr - $remote\_user [$time\_local] “$request”’

```

增加这一行 ，双引号加个空格，好看些

‘ ”$upstream\_cache\_status”’;

```

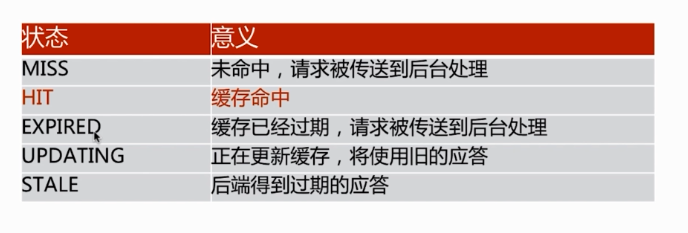
}

```

然后再

Awk ‘{if ($NF==”\”HIT\””){hit++}}END{printf “%.2f”,hit/NR}’ /var/log/nignx/test.access.log

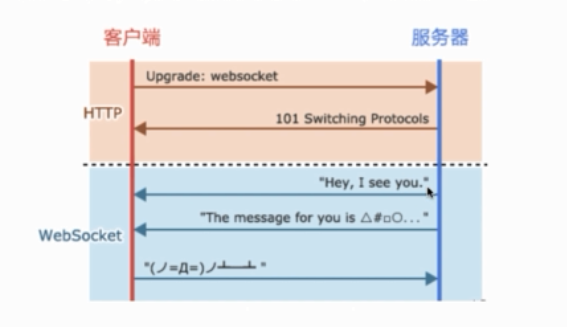
$upstream\_cache\_statusd的状态



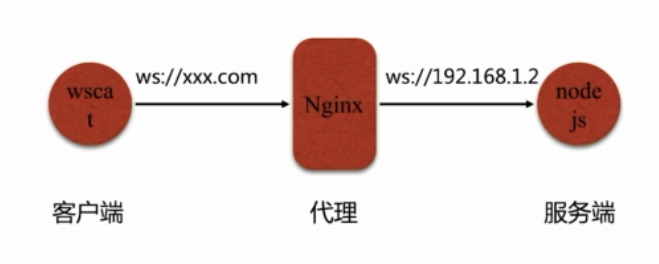
# 5 websocket代理(通过proxy\_pass)

## 5.1 协议定义及应用场景

Websocket实现在http连接的基础上，客户端通过发送http中的*upgrade*协议头将连接从http升级到websocket，这样就可以实现多次双休通讯，直到连接被关闭。  
广泛应用于聊天室和app信息推送



## 5.2 协议原理



## 5.3 ws协议安装及运行

Yun install nodejs -y;(安装服务端)

Yum install npm;（nodejs的模块工具）

Npm install ws;（利用npm 安装websocket，要在js文件中的目录中安装，启用服务node ./server.js）

Npm install wscat;（ws协议的客户端测试工具,利用wscat来发起ws协议的连接

wscat --connect ws://127.0.0.1:8020

）

## 5.4 ws协议nginx配置

# 判断并映射

Map $http\_upgrade $connection\_upgrade{

Default upgrade;#如果没有值，connecton\_upgrade=upgrade；

‘ ’ close; # 如果值为空，connecton\_upgrade=close；

}

Upstream websocket{

Server 127.0.0.1:8010;#nodejs 提供后端服务

}

Server{

Listen 8020;

Aceess\_log /var/log/ngixn/tesr\_websocket.access.log main;

Location /{

Proxy\_pass <http://websocket;>

*Proxy\_http\_version 1.1;*

*Proxy\_set\_header Upgrade $http\_upgrade;（nginx发给后端服务器的头）*

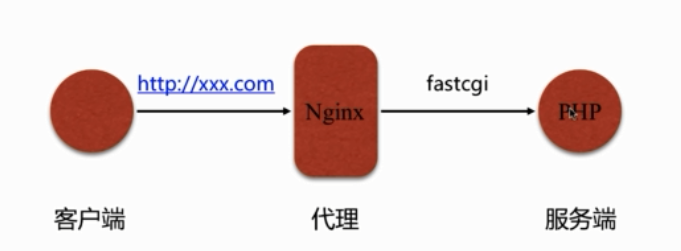
*Proxy\_set\_header Connection $connection\_upgrade;（nginx发给后端服务器的头）*

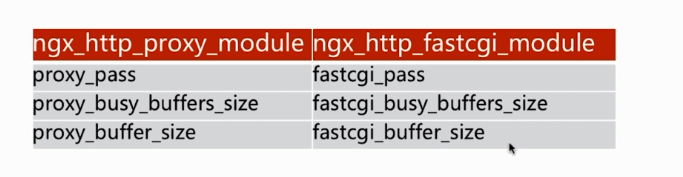
}

}

# 6 fastcgi代理

## 6.1 协议原理





## 6.2 配置语法

fastcgi\_index index.php;#首页配置

fastcgi\_param SCRIPT\_FILENAME /home/www/scripts/php$fastcgi\_script\_name;

# fastcgi\_param用于设置变量。传递给后端fastcgi服务器

# 前端访问http：//xxx.com时候，$fastcgi\_script\_name = index.php SCRIPT\_FILENAME=/home/www/scripts/php/index.php

# 前端访问http：//xxx.com/page.php时候。$fastcgi\_script\_name = page.php SCRIPT\_FILENAME=/home/www/scripts/php/page.php

## 6.3 配置详解

Server{

···

root /opt/app/code5;

Location / {

index index.php

}

Location ~ \.php${

fastcgi\_pass 127.0.0.1:9000;

fastcgi\_index index.php;

Include fastcgi\_param;

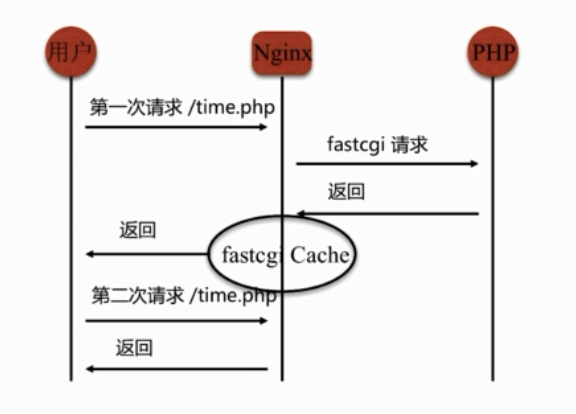
Fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;

*#$document\_root= /opt/app/code5*

}

}

## 6.4 fastcgi代理缓存（参考代理缓存）



fastcgi\_cache\_path /opt/app/cache levels=1:2 keys\_zone=imooc:100m max\_size=1g inactive=60m ;

#levels 缓存目录的层级 keys\_zone 开劈空间的名字，一般一兆可以存放8000个key

max\_size 控制磁盘目录的最大值，当达到最大空间时候，就会触发淘汰规则。inactive = 60，60分钟后就触发淘汰规则。

server{

...

location ~ \.php${

fastcgi\_pass 127.0.0.1:9000;

fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;

fastcgi\_index index.php;

Include fastcgi\_param;

#fastcgi\_cache\_key $scheme$host;#设置缓存的维度

fastcgi\_cache\_key $scheme$request\_method$host$request\_uri$is\_args$args;

fastcgi\_cache imooc;#应用名字，关闭缓存是设为off，默认关闭

fastcgi\_cache\_valid 200 60m;#200设置为60分钟过期

add\_header X-Cache-Source $upstream\_cache\_status;# 加到response头信息给客户，方便前端查看是否缓存页面

***#fastcgi\_ignore\_headers Cache-Control Expires***

# 即使后端服务告诉代理nocache，代理也会忽略掉，相当于代理强制缓存内容

}

...

}