# NSD Operation DAY01

# Nginx扩展20190917：

Nginx 1.14.0版

Ngixn.org 开源版 nginx.com商业版 tengine阿里巴巴修改nginx而来

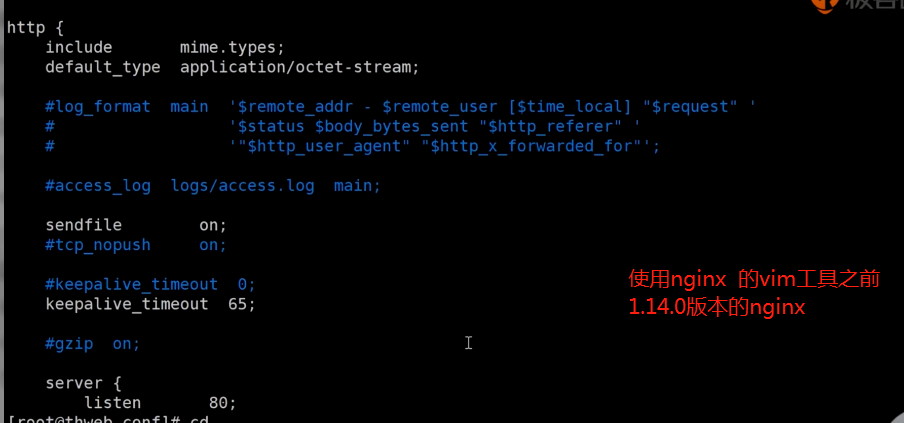


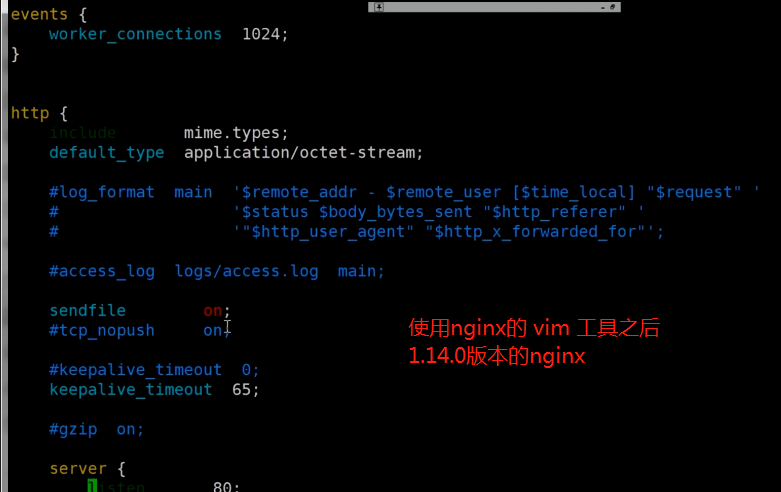


Auto 辅助编译文件 CHANGES 告诉了一些版本的功能和特性 configure脚本用来生成编译文件生成编译前必备 Src 源代码目录 man:帮助man命令

1：nginx自带vim编辑工具，使得编辑nginx配置文件时，nginx语法很清晰的展现，工具在contrib目录中

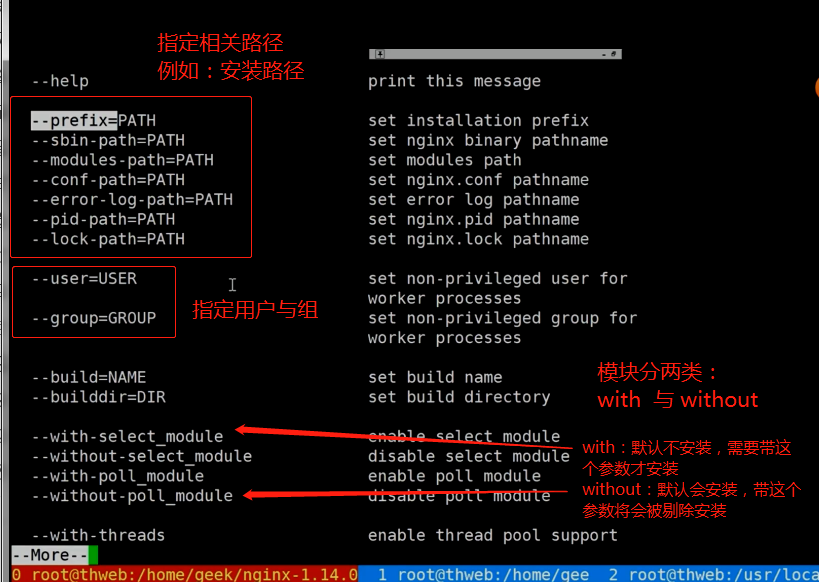




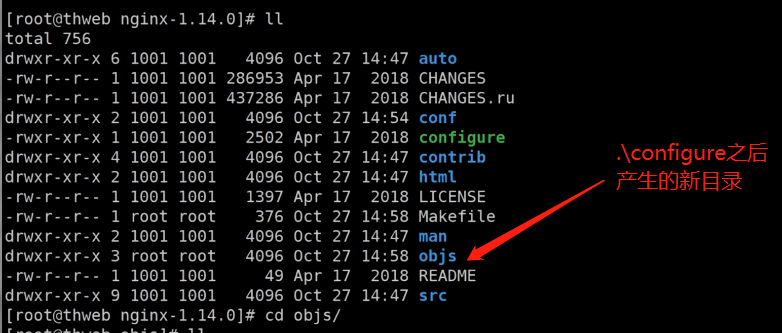


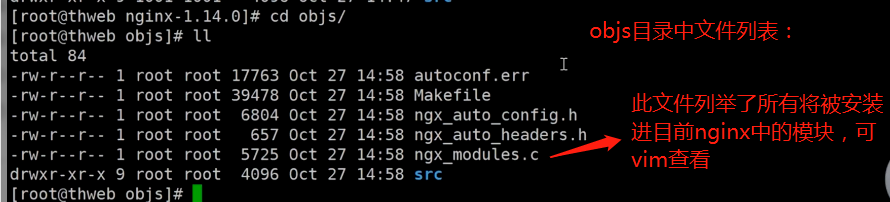
2 nginx 编译前查看相关模块

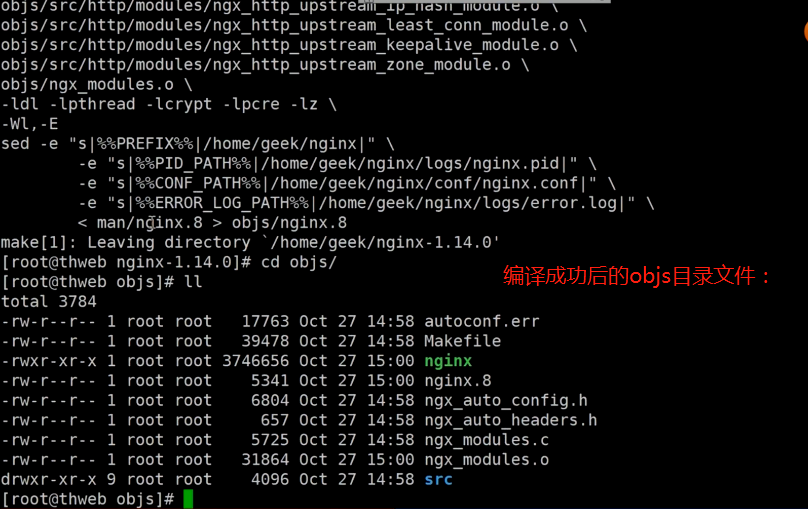












Make 编译之后，如果是首次

[案例1：搭建Nginx服务器](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/OPERATION/DAY01/CASE/01/index.html" \l "case1)

[案例2：用户认证](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/OPERATION/DAY01/CASE/01/index.html" \l "case2)

[案例3：基于域名的虚拟主机](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/OPERATION/DAY01/CASE/01/index.html" \l "case3)

[案例4：SSL虚拟主机](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/OPERATION/DAY01/CASE/01/index.html" \l "case4)

# 1 案例1：搭建Nginx服务器

1.1 问题

在IP地址为192.168.4.5的主机上安装部署Nginx服务，并可以将Nginx服务器，要求编译时启用如下功能：

支持SSL加密功能

设置Nginx账户及组名称均为nginx

Nginx服务器升级到更高版本。

然后客户端访问页面验证Nginx Web服务器：

使用火狐浏览器访问、使用curl访问

1.2 方案

第一天课程需要使用2台RHEL7虚拟机，其中一台作为Nginx服务器（192.168.4.5）、另外一台作为测试用的Linux客户机（192.168.4.10），如图-1所示。

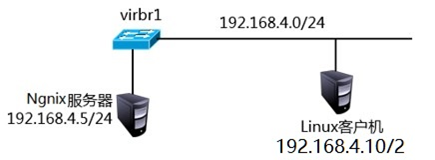


创建:clinet虚拟机 配置eth0 地址

创建:proxy 虚拟机 配置eth0 地址 ,eth1网卡本身存在,添加连接如下:(添加后再配置IP)

[root@proxy ~]# nmcli connection add con-name eth1 ifname eth1 type ethernet #添加eth1,type(类型)为ethernet

参考网卡聚合:nmcli connection add type temo ifname eth0 con-name eth0



nginx 依赖关系是:gcc pcre-devel(支持正则表达式) openssl-devel

nginx的rewrite伪静态匹配规则用到正则，pcre就用来干这个。

用户权限.用户启动的程序,这个程序将有这个用户的相应权限.

linux先创建相应的普通用户,然后配置,指定这个程序的用户是这个普通用户,起到安全作用

Yum安装是有脚本创建用户,并指定软件用户为之前设定的用户.

例如:httpd ps aux | grep httpd

源码安装是没有创建用户的脚本,

## 步骤一：构建Nginx服务器

### 1）使用源码包安装nginx软件包

[root@proxy ~]# yum -y install gcc pcre-devel openssl-devel #依赖包

[root@proxy ~]# useradd -s /sbin/nologin nginx

[root@proxy ~]# tar -xf /nginx-1.10.3.tar.gz #解压包到当前目录/root

[root@proxy ~]# cd nginx-1.10.3/ #进入解压后产生的目录,

[root@proxy nginx-1.10.3]# ./configure \ #运行配置程序

> --prefix=/usr/local/nginx \ #指定安装路径,默认就是安装在/usr/local/nginx

> --user=nginx \ #指定用户

> --group=nginx \ #指定组

> --with-http\_ssl\_module #开启SSL加密功能模块

>--with-stream #TCP/UDP代理(四层)用

>--with-http\_stub\_status\_module #开启状态页面模块

[root@proxy nginx-1.10.3]# make && make install #编译并安装

### 2）nginx命令的用法

[root@proxy ~]# /usr/local/nginx/sbin/nginx //启动服务

[root@proxy ~]# /usr/local/nginx/sbin/nginx -s stop //关闭服务

[root@proxy ~]# /usr/local/nginx/sbin/nginx -s reload //重新加载配置文件

[root@proxy ~]# /usr/local/nginx/sbin/nginx -V //查看软件信息

### 3）ln命令做软链接使用nginx命令

[root@proxy ~]# ln -s /usr/local/nginx/sbin/nginx /sbin/ //做链接方便后期使用

ln 做链接后

[root@proxy ~]# nginx -s stop //关闭服务

[root@proxy ~]# nginx //启动服务

[root@proxy ~]# nginx -s reload //重新加载配置文件

[root@proxy ~]# nginx -V //查看软件信息

[root@proxy ~]# nginx -V

nginx version: nginx/1.10.3

built by gcc 4.8.5 20150623 (Red Hat 4.8.5-16) (GCC)

built with OpenSSL 1.0.2k-fips 26 Jan 2017

TLS SNI support enabled

configure arguments: --prefix=/usr/local/nginx --user=nginx --group=nginx --with-http\_ssl\_module

#第一步给软件指定了”nginx”用户,如无此用户,服务启动不了,

netstat命令可以查看系统中启动的端口信息，该命令常用选项如下：

-a显示所有端口的信息

-n以数字格式显示端口号

-t显示TCP连接的端口

-u显示UDP连接的端口

-l显示服务正在监听的端口信息，如httpd启动后，会一直监听80端口

-p显示监听端口的服务名称是什么（也就是程序名称）

nginx服务默认通过TCP 80端口监听客户端请求：

root@proxy ~]# netstat -anptu | grep nginx

### 设置防火墙与SELinux

（非必须的操作，如果有则关闭）

[root@proxy ~]# firewall-cmd --set-default-zone=trusted

[root@proxy ~]# setenforce 0

## 步骤二 : nginx升级

升级大概思路:

1--准备安装包,解压安装包

2--.configure #根据你的选择将src中要用的包放入objs中

3--make #用gcc编译.则多了nginx目录

4--备份老版本的nginx (/usr/local/nginx/sbin/nginx)

5--更换新产生的nginx:将nginx替换老版本的nginx

### 1) 解压新的安装包 #src目录中有源代码

[root@proxy ~]# cd nginx-1.12.2/ #进入目录

[root@proxy nginx-1.12.2]# ls #刚解压的目录

auto CHANGES CHANGES.ru conf configure contrib html LICENSE man README src

### 2) .configure选择需要的模块

[root@proxy nginx-1.12.2]# ./configure

--prefix=/usr/local/nginx #指定安装路径

--user=nginx

--group=nginx

--with-http\_ssl\_module #开启SSL加密功能模块 SSL虚拟主机用

--with-stream #TCP/UDP代理(四层)用

--with-http\_stub\_status\_module #开启状态页面模块

[root@proxy nginx-1.12.2]# ls #.configure之后多了 “objs”目录

auto CHANGES CHANGES.ru conf configure contrib html LICENSE Makefile man objs README src

### 3) make编译

**切勿make install安装**

[root@proxy nginx-1.12.2]# make

[root@proxy nginx-1.12.2]# cd objs

[root@proxy objs]# pwd

/root/nginx-1.12.2/objs

[root@proxy objs]# ls #在新版本1.12.2目录里面查看

autoconf.err nginx ngx\_auto\_config.h ngx\_modules.c src

Makefile nginx.8 ngx\_auto\_headers.h ngx\_modules.o

[root@proxy objs]# cd /root/nginx-1.10.3/objs/

[root@proxy objs]# pwd

/root/nginx-1.10.3/objs

[root@proxy objs]# ls #在旧版本1.10.3目录里面查看

autoconf.err nginx ngx\_auto\_config.h ngx\_modules.c src

Makefile nginx.8 ngx\_auto\_headers.h ngx\_modules.o

### 4) 备份旧的nginx,更换新的nginx

备份:将 /usr/local/nginx/sbin 中在用的旧的 nginx 备份

[root@proxy objs]# cd /usr/local/nginx/sbin/

[root@proxy sbin]# ls

nginx

[root@proxy sbin]# mv nginx nginx.bak

[root@proxy sbin]# ls

nginx.bak

将 /root/nginx-1.12.2/objs 目录里面的nginx 拷贝放到/usr/local/nginx/sbin/

[root@proxy sbin]# cd /root/nginx-1.12.2

[root@proxy nginx-1.12.2]# cp objs/nginx /usr/local/nginx/sbin/

[root@proxy nginx-1.12.2]# ls /usr/local/nginx/sbin/

nginx nginx.bak

[root@proxy nginx-1.12.2]# cd

[root@proxy ~]# nginx -s stop #关闭服务

或者 killall nginx

[root@proxy ~]# nginx #重启服务

[root@proxy ~]# firefox 192.168.4.5 #测试访问

[root@client ~]# curl http://192.168.4.5

[root@proxy ~]# nginx -V #查看版本,变成1.12.2

nginx version: nginx/1.12.2

built by gcc 4.8.5 20150623 (Red Hat 4.8.5-16) (GCC)

built with OpenSSL 1.0.2k-fips 26 Jan 2017

TLS SNI support enabled

configure arguments: --prefix=/usr/local/nginx --user=nginx --group=nginx --with-http\_ssl\_module

[root@proxy ~]# cd /usr/local/nginx #网页所在目录

[root@proxy nginx]# ls

client\_body\_temp conf fastcgi\_temp html logs proxy\_temp sbin scgi\_temp uwsgi\_temp

[root@proxy nginx]# cd /usr/local/nginx/html/

[root@proxy html]# ls

50x.html index.html

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf 全局配置

.. ..

server {

listen 80;

server\_name localhost;

auth\_basic "Input Password:"; //认证提示符

auth\_basic\_user\_file "/usr/local/nginx/pass"; //认证密码文件

location / {

root html;

index index.html index.htm; #设置默认首页,index.htm为备用

}

}

案例一完成

# 2 案例2：用户认证

沿用练习一，通过调整Nginx服务端配置，实现以下目标：

访问Web页面需要进行用户认证

用户名为：tom，密码为：123456

通过Nginx实现Web页面的认证，需要修改Nginx配置文件，在配置文件中添加auth语句实现用户认证。最后使用htpasswd命令创建用户及密码即可。

## 修改Nginx配置文件

### 1) 修改/usr/local/nginx/conf/nginx.conf

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

.. ..

server {

listen 80;

server\_name localhost;

auth\_basic "hello:";                          //认证提示符,登录框会显示

auth\_basic\_user\_file "/usr/local/nginx/pass";      //认证密码文件

location / {

root html;

ndex index.html index.htm;

}

}

### 2）生成密码文件，创建用户及密码

使用htpasswd命令创建账户文件，需要确保系统中已经安装了httpd-tools(httpd工具)。

[root@proxy html]# yum -y install httpd-tools #安装httpd-tools

[root@proxy html]# htpasswd -c /usr/local/nginx/pass jerry #首次创建要-c

New password:

Re-type new password:

htpasswd: password verification error #未创建成功

[root@proxy html]# htpasswd /usr/local/nginx/pass tom #再次创建用户不要-c

New password:

Re-type new password:

Adding password for user tom #创建

### 3) 重启以及测试

[root@proxy ~]# /usr/local/nginx/sbin/nginx -s reload //重新加载配置文件

#请先确保nginx是启动状态，否则运行该命令会报错,报错信息如下：

#[error] open() "/usr/local/nginx/logs/nginx.pid" failed (2: No such file or directory)

[root@client ~]# firefox http://192.168.4.5 #换客户端访问测试

# 3 案例3：基于域名的虚拟主机

3.1 问题

沿用练习二，配置基于域名的虚拟主机，实现以下目标：

实现两个基于域名的虚拟主机，域名分别为www.a.com和www.b.com

对域名为www.a.com的站点进行用户认证，用户名称为tom，密码为123456

3.2 方案

修改Nginx配置文件，添加server容器实现虚拟主机功能；对于需要进行用户认证的虚拟主机添加auth认证语句。

虚拟主机一般可用分为：基于域名、基于IP和基于端口的虚拟主机

[root@proxy html]# vim /usr/local/nginx/conf/nginx.conf #去掉下一个server注释

1---在末行模式下用sed命令替换一个server段 ---- (:85,94s/#//)

2---并配置listen servername root

3---添加charset utf-8; 用于支持中文,解释ASCll码.Utf-8:万国编码.

85 server {

86 listen 80;

87 server\_name www.b.com;

88 charset utf-8;

89 location / {

90 root www;

91 index index.html index.htm;

92 }

93 }

## 步骤一:编辑配置文件nginx.conf ,

### 1）修改Nginx服务配置，添加相关虚拟主机配置如下

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

.. ..

server {

listen 80; //端口

server\_name www.a.com; //域名

auth\_basic "Input Password:"; //认证提示符

auth\_basic\_user\_file "/usr/local/nginx/pass"; //认证密码文件

location / {

root html; //指定网站根路径

index index.html index.htm;

}

}

… …

server {

listen 80; //端口

server\_name www.b.com; //域名

location / {

root www; //指定网站根路径

index index.html index.htm;

}

}

### 2）创建网站根目录及对应首页文件

[root@proxy ~]# mkdir /usr/local/nginx/www

[root@proxy ~]# echo "www" > /usr/local/nginx/www/index.html

### 3）重启nginx

[root@proxy ~]# /usr/local/nginx/sbin/nginx -s reload

#请先确保nginx是启动状态，否则运行该命令会报错,报错信息如下：

#[error] open() "/usr/local/nginx/logs/nginx.pid" failed (2: No such file or directory)

### 4）本机域名解析添加测试

1）修改客户端主机192.168.4.10的/etc/hosts文件，进行域名解析

[root@client ~]# vim /etc/hosts

192.168.4.5 www.a.com www.b.com

2）登录192.168.4.10客户端主机进行测试

注意：请先关闭真实机的firefox，再SSH -X远程连接调用虚拟机的firefox。

[root@client ~]# firefox http://www.a.com //输入密码后可以访问

[root@client ~]# firefox http://www.b.com //直接访问

## 步骤二：扩展课外实验：其他类型的虚拟主机

### 基于端口的虚拟主机（参考模板）

server {

listen 8080; //端口

server\_name web1.example.com; //域名

......

}

server {

listen 8000; //端口

server\_name web1.example.com; //域名

.......

}

### 基于IP的虚拟主机（参考模板）

server {

listen 192.168.0.1:80; //IP地址与端口

server\_name web1.example.com; //域名

... ...

}

server {

listen 192.168.0.2:80; //IP地址与端口

server\_name web1.example.com;

... ...

}

# 案例4：SSL虚拟主机

4.1 问题

沿用练习三，配置基于加密网站的虚拟主机，实现以下目标：

1. 域名为www.c.com
2. 该站点通过https访问
3. 通过私钥、证书对该站点所有数据加密

4.2 方案

源码安装Nginx时必须使用--with-http\_ssl\_module参数，启用加密模块，对于需要进行SSL加密处理的站点添加ssl相关指令（设置网站需要的私钥和证书）。

加密算法一般分为对称算法、非对称算法、信息摘要。

对称算法有：AES、DES，主要应用在单机数据加密。

非对称算法有：RSA、DSA，主要应用在网络数据加密。公钥,私钥.不可逆

比如:将一串数字组成一个结果,反过来,将结果还原之前那串数字,很难实现

信息摘要：MD5、sha256，主要应用在数据完整性校验。

## MD5 校验文件完整行:

[root@proxy ~]# echo 1>1.txt

[root@proxy ~]# md5sum 1.txt

68b329da9893e34099c7d8ad5cb9c940 1.txt

[root@proxy ~]# mv 1.txt abc.txt

[root@proxy ~]# md5sum abc.txt

68b329da9893e34099c7d8ad5cb9c940 abc.txt #文件名改了内容未改码未变

[root@proxy ~]# echo 2>abc.txt

[root@proxy ~]# echo 2 >abc.txt #更改内容

[root@proxy ~]# md5sum abc.txt #内容改了码也变了

26ab0db90d72e28ad0ba1e22ee510510 abc.txt

## 步骤一：配置SSL虚拟主机

### 1）生成私钥与证书

证书的格式命名 .x509

[root@proxy ~]# cd /usr/local/nginx/conf/ #进入目录

[root@proxy conf]# openssl genrsa > cert.key #生成私钥

Generating RSA private key, 2048 bit long modulus #生成了2048位秘钥

..........................................+++ .......+++

e is 65537 (0x10001)

[root@proxy conf]# openssl req -new -x509 -key cert.key >cert.pem #生成证书(公钥)

You are about to be asked to enter information that will be incorporated

into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

-----

Country Name (2 letter code) [XX]:cn

State or Province Name (full name) []:beijing

Locality Name (eg, city) [Default City]:bejing

Organization Name (eg, company) [Default Company Ltd]:beijing

Organizational Unit Name (eg, section) []:aa@aa.com

Common Name (eg, your name or your server's hostname) []:6

Email Address []:aa@aa.com

[root@proxy conf]#

### 2）修改Nginx配置文件，设置加密网站的虚拟主机

[root@proxy conf]# vim /usr/local/nginx/conf/nginx.conf #修改配置文件

server {

listen 443 ssl;

server\_name www.c.com;

ssl\_certificate cert.pem;

ssl\_certificate\_key cert.key;

ssl\_session\_cache shared:SSL:1m;

ssl\_session\_timeout 5m;

ssl\_ciphers HIGH:!aNULL:!MD5;

ssl\_prefer\_server\_ciphers on;

location / {

root ccc;

index index.html index.htm;

}

}

#批量换掉注释末行模式下 :66,79s/#//

### 3）重启nginx服务与测试

[root@proxy ~]# /usr/local/nginx/sbin/nginx -s reload

#请先确保nginx是启动状态，否则运行该命令会报错,报错信息如下：

#[error] open() "/usr/local/nginx/logs/nginx.pid" failed (2: No such file or directory)

修改客户端主机192.168.4.10的/etc/hosts文件，进行域名解析

[root@client ~]# vim /etc/hosts

192.168.4.5 www.c.com www.a.com www.b.com

登录192.168.4.10客户端主机进行测试

[root@client ~]# firefox https://www.c.com //信任证书后可以访问

# NSD Operation DAY02

案例1：部署LNMP环境

案例2：构建LNMP平台

案例3：地址重写

# 简述什么是LNMP

LNMP是网站运行平台。

* L：操作系统（Linux）
* N：网站服务软件（Nginx）
* M：数据库服务软件（MariaDB）
* P：网站开发语言（PHP、perl、python）

多进程程序

程序:

进程:一个程序可能打开多个进程

线程:

未启动就是程序,启动后进入内存运行,变成进程.进程再开线程

一个程序开启多个进程,一个进程再开多个线程,例如一个进程使用20M内存,开了3个线程,这3个线程共享使用20M.

LNMP是指一组通常一起使用来运行动态网站或者服务器的自由软件名称首字母缩写。L指Linux，N指Nginx，M一般指MySQL，也可以指MariaDB，P一般指PHP，也可以指Perl或Python。

# 1 案例1：部署LNMP环境

1.1 问题

安装部署Nginx、MariaDB、PHP环境

安装部署Nginx、MariaDB、PHP、PHP-FPM；

启动Nginx、MariaDB、FPM服务；

并测试LNMP是否工作正常。

1.2 方案

LNMP（Linux、Nginx、MySQL、PHP）

在RHEL7系统中，源码安装Nginx，使用RPM包安装MariaDB、PHP、PHP-FPM软件。

操作过程中需要安装的软件列表如下：

* nginx
* mariadb、mariadb-server、mariadb-devel
* php、php-fpm、php-mysql

备注：mariadb（数据库客户端软件）、mariadb-server（数据库服务器软件）、mariadb-devel（其他客户端软件的依赖包）、php（解释器）、php-fpm（进程管理器服务）、php-mysql（PHP的数据库扩展包）。

## 步骤一：安装软件

### 1) 使用yum安装基础依赖包

[root@proxy ~]# yum -y install gcc openssl-devel pcre-devel

### 2）安装MariaDB

Mariadb在新版RHEL7光盘中包含有该软件，配置yum源后可以直接使用yum安装：

[root@proxy ~]# yum -y install mariadb mariadb-server mariadb-devel

[root@proxy lnmp\_soft]# systemctl start mariadb

[root@proxy lnmp\_soft]# systemctl enable mariadb

[root@proxy lnmp\_soft]# systemctl status mariadb

● mariadb.service - MariaDB database server

Loaded: loaded (/usr/lib/

### 3）php和php-fpm(该软件包在lnmp\_soft中提供)

[root@proxy ~]# yum -y install php php-mysql

[root@proxy ~]# ls

abc.txt lnmp\_soft.tar.gz nginx.sh 视频 下载

anaconda-ks.cfg nginx-1.10.3 公共 图片 音乐

initial-setup-ks.cfg nginx-1.12.2 模板 文档 桌面

[root@proxy ~]# tar -xf lnmbp\_soft.tar.gz

[root@proxy ~]# cd lnmp\_soft/

[root@proxy lnmp\_soft]# ls

....

[root@proxy lnmp\_soft]# yum -y install php-fpm-5.4.16-42.el7.x86\_64.rpm

#php-fpm-5.4.16-42.el7.x86\_64.rpm,光盘中没有在提供的lnmp\_soft.tar.gz包内

[root@proxy lnmp\_soft]# systemctl start php-fpm

[root@proxy lnmp\_soft]# systemctl enable php-fpm

[root@proxy lnmp\_soft]# vim test.php #创建一个脚本,测试PHP

----脚本内容-----------

t.php

<?php

$i=33;

echo $i;

?>

--------------------------------

[root@proxy lnmp\_soft]# mv test.php /usr/local/nginx/html/

[root@proxy lnmp\_soft]# php /usr/local/nginx/html/test.php #执行测试

t.php

33[root@proxy lnmp\_soft]# #测试成功

[root@proxy ~]# netstat -ntulp | grep 9000

tcp 0 0 127.0.0.1:9000 0.0.0.0:\* LISTEN 5757/php-fpm: maste

[root@proxy ~]# netstat -ntulp | grep 80

[root@proxy ~]# nginx

[root@proxy ~]# netstat -ntulp | grep 80

tcp 0 0 0.0.0.0:80 0.0.0.0:\* LISTEN 5986/nginx: master

[root@proxy ~]# netstat -ntulp | grep 3306

tcp 0 0 0.0.0.0:3306 0.0.0.0:\*

### 4）源码安装Nginx（如果前面课程中已经安装Nginx，则忽略这一步）

[root@proxy ~]# useradd -s /sbin/nologin nginx

[root@proxy ~]# tar -xvf nginx-1.12.2.tar.gz

[root@proxy ~]# cd nginx-1.12.2

[root@proxy nginx-1.12.2]# ./configure \

> --user=nginx \

--group=nginx \

> --with-http\_ssl\_module

[root@proxy ~]# make && make install

修改Nginx配置文件:

第一个server里面有模板,取消#即可 ,为实验方便,将证书与私钥行给注释掉,

如果是动态,转发给9000端口

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

location ~ \.php$ {

root html;

fastcgi\_pass 127.0.0.1:9000; #如果访问的是.php 转发给9000端口

fastcgi\_index index.php;

# fastcgi\_param SCRIPT\_FILENAME /scripts$fastcgi\_script\_name;

include fastcgi.conf

}

:#将 fastcgi\_param 行注释掉

include:是

参数由fastcgi\_params改成 fastcgi.conf ,以前的版本是使用fastcgi\_params,新版本使用fastcgi\_conf

[root@proxy ~]# nginx -s reload #修改后重新加载配置

## 步骤二：启动服务

### 启动Nginx服务

（如果已经启动nginx，则可以忽略这一步）

这里需要注意的是，如果服务器上已经启动了其他监听80端口的服务软件（如httpd），则需要先关闭该服务，否则会出现冲突。

[root@proxy ~]# systemctl stop httpd //如果该服务存在则关闭该服务

[root@proxy ~]# /usr/local/nginx/sbin/nginx //启动Nginx服务

[root@proxy ~]# netstat -utnlp | grep :80

tcp 0 0 0.0.0.0:80 0.0.0.0:\* LISTEN 32428/nginx

### 2）启动MySQL服务

[root@proxy ~]# systemctl start mariadb //启动服务器

[root@proxy ~]# systemctl status mariadb //查看服务状态

[root@proxy ~]# systemctl enable mariadb //设置开机启动

### 3）启动PHP-FPM服务

[root@proxy ~]# systemctl start php-fpm //启动服务

[root@proxy ~]# systemctl status php-fpm //查看服务状态

[root@proxy ~]# systemctl enable php-fpm //设置开机启动

### 4）设置防火墙与SELinux（非必需操作，如果有则关闭）

[root@proxy ~]# firewall-cmd --set-default-zone=trusted

[root@proxy ~]# setenforce 0

# 2 案例2：构建LNMP平台

2.1 问题

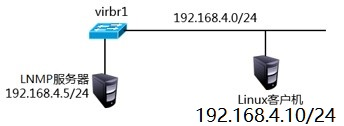
沿用练习一，通过调整Nginx服务端配置，实现以下目标：

配置Fast-CGI支持PHP网页

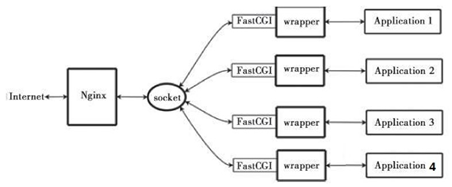
创建PHP测试页面，测试使用PHP连接数据库的效果

2.2 方案

使用2台RHEL7虚拟机，其中一台作为LNMP服务器（192.168.4.5）、另外一台作为测试用的Linux客户机（192.168.4.10），如图-1所示。



Nginx结合FastCGI技术即可支持PHP页面架构，如图-2所示。



因此本案例，需要延续练习一的实验内容，通过修改Nginx及php-fpm配置文件实现对PHP页面的支持。

注意，FastCGI的内存消耗问题，一个PHP-FPM解释器将消耗约25M的内存。

## 步骤一：查看php-fpm配置文件

[root@proxy ~]#vim /etc/php-fpm.d/www.conf #本实验中不需要修改

[www]

listen = 127.0.0.1:9000 //PHP端口号

pm.max\_children = 32 //最大进程数量

pm.start\_servers = 15 //最小进程数量

pm.min\_spare\_servers = 5 //最少需要几个空闲着的进程

pm.max\_spare\_servers = 32 //最多允许几个进程处于空闲状态

[root@proxy ~]# firefox 192.168.4.5/test.php #访问测试

## 步骤二：修改Nginx配置文件并启动服务

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

location / {

root html;

index index.php index.html index.htm;

#设置默认首页为index.php，当用户在浏览器地址栏中只写域名或IP，不说访问什么页面时，服务器会把默认首页index.php返回给用户

}

location ~ \.php$ {

root html;

fastcgi\_pass 127.0.0.1:9000; #将请求转发给本机9000端口，PHP解释器

fastcgi\_index index.php;

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

include fastcgi.conf;

}

[root@proxy ~]# /usr/local/nginx/sbin/nginx -s reload

#请先确保nginx是启动状态，否则运行该命令会报错,报错信息如下：

#[error] open() "/usr/local/nginx/logs/nginx.pid" failed (2: No such file or directory)

## 步骤三：创建PHP页面，测试LNMP架构能否解析PHP页面

### 创建PHP测试页面1，

可以参考lnmp\_soft/php\_scripts/test.php：

[root@proxy ~]# vim /usr/local/nginx/html/test1.php

<?php

$i="This is a test Page";

echo $i;

?>

### 2）创建PHP测试页面,连接并查询MariaDB数据库。

可以参考lnmp\_soft/php\_scripts/mysql.php:

[root@proxy ~]# vim /usr/local/nginx/html/test2.php

<?php

$mysqli = new mysqli('localhost','root','密码','mysql');

//注意：root为mysql账户名称，密码需要修改为实际mysql密码，无密码则留空即可

//localhost是数据库的域名或IP，mysql是数据库的名称

if (mysqli\_connect\_errno()){

die('Unable to connect!'). mysqli\_connect\_error();

}

$sql = "select \* from user";

$result = $mysqli->query($sql);

while($row = $result->fetch\_array()){

printf("Host:%s",$row[0]);

printf("</br>");

printf("Name:%s",$row[1]);

printf("</br>");

}

?>

### 3）客户端使用浏览器访问服务器PHP首页文档，检验是否成功：

[root@client ~]# firefox http://192.168.4.5/test1.php [root@client ~]# firefox http://192.168.4.5/test2.php

### 4）LNMP常见问题

Nginx的默认访问日志文件为/usr/local/nginx/logs/access.log

Nginx的默认错误日志文件为/usr/local/nginx/logs/error.log

PHP默认错误日志文件为/var/log/php-fpm/www-error.log

如果动态网站访问失败，可用参考错误日志，查找错误信息。

### 总结:

1. 安装nginx mariadb： mariadb-server mariadb-devel Php： php-fpm php-mysql
2. 启动nginx systemctl start mariadb systemctl start php-fpm
3. 修改配置nginx.conf
4. 修改配置php-fpm
5. 访问测试
6. 查看报错:用tailf #tailf 动态查看

Nginx默认访问日志:/usr/local/nginx/logs/access.log

Nginx报错日志:/usr/local/nginx/logs/error.log

PHP默认错误日志文件:/var/log/php-fpm/www-error.log

测试数据库

[root@proxy php\_scripts]# cp mysql.php /usr/local/nginx/html/

[root@proxy php\_scripts]# firefox 192.168.4.5/mysql.php

# 3 案例3：地址重写

rewrite 重写

rewrite 访问的地址 跳转到的地址

3.1 问题

沿用练习二，通过调整Nginx服务端配置，实现以下目标：

1. 所有访问a.html的请求，重定向到b.html;
2. 所有访问192.168.4.5的请求重定向至www.tmooc.cn；
3. 所有访问192.168.4.5/下面子页面，重定向至www.tmooc.cn/下相同的页面；
4. 实现firefox与curl访问相同页面文件，返回不同的内容。

3.2 方案

关于Nginx服务器的地址重写，主要用到的配置参数是rewrite：

* rewrite regex replacement flag
* rewrite 旧地址 新地址 [选项]

## 步骤一：修改配置文件(访问a.html重定向到b.html)

### 1）修改Nginx服务配置：

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

.. ..

server {

listen 80;

server\_name localhost;

rewrite /a.html /b.html; #rewrite 访问的地址 跳转到的地址

location / {

root html;

index index.html index.htm;

}

}

[root@proxy ~]# echo "BB" > /usr/local/nginx/html/b.html

### 2）重新加载配置文件

[root@proxy ~]# /usr/local/nginx/sbin/nginx -s reload

### 3）客户端测试

[root@client ~]# firefox http://192.168.4.5/a.html

## 步骤二：访问a.html重定向到b.html（跳转地址栏）

### 1）修改Nginx服务配置：

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

.. ..

server {

listen 80;

server\_name localhost;

rewrite /a.html /b.html redirect; #rewrite 访问的地址 跳转到的地址 重写类型

location / {

root html;

index index.html index.htm;

}

}

访问的地址：可以是字符串或者正则来表示想匹配的目标url

跳转到的地址：表示匹配到规则后要定向的路径，如果规则里有正则，则可以使用$index来表示正则里的捕获分组

重写类型：

* last ：相当于Apache里德(L)标记，表示完成rewrite，浏览器地址栏URL地址不变
* break；本条规则匹配完成后，终止匹配，不再匹配后面的规则，浏览器地址栏URL地址不变
* redirect：返回302临时重定向，浏览器地址会显示跳转后的URL地址
* permanent：返回301永久重定向，浏览器地址栏会显示跳转后的URL地址

简单例子实例一：

server {

rewrite /last.html /index.html last; # 访问 /last.html 的时候，页面内容重写到 /index.html 中

rewrite /break.html /index.html break; #访问/break.html的时候，页面内容重写到/index.html中，并停止后续的匹配

rewrite /redirect.html /index.html redirect; #访问/redirect.html的时候，页面直接302定向到/index.html中

rewrite /permanent.html /index.html permanent; #访问/permanent.html的时候，页面直接301定向到/index.html中

rewrite ^/html/(.+?).html$ /post/$1.html permanent; # 把 /html/\*.html => /post/\*.html ，301定向

rewrite ^/search\/([^\/]+?)(\/|$) /search.html?keyword=$1 permanent; # 把 /search/key => /search.html?keyword=key

}

简单例子实例二：

www.168cd.cn和168cd.cn永久重定向到www.siyuanv.com,包括下面的子页面

server {

listen 80;

server\_name www.168cd.cn 168cd.cn;

return 301 https://www.siyuanv.com$request\_uri;

location / {

root /opt/syqkl/html;

index index.html index.htm;

}

}

### 2）重新加载配置文件

[root@proxy ~]# /usr/local/nginx/sbin/nginx -s reload

#请先确保nginx是启动状态，否则运行该命令会报错,报错信息如下：

#[error] open() "/usr/local/nginx/logs/nginx.pid" failed (2: No such file or directory)

### 3）客户端测试（仔细观察浏览器地址栏的变化）

[root@client ~]# firefox http://192.168.4.5/a.html

## 步骤三：地址重写(正则匹配)

修改配置文件(访问192.168.4.5的请求重定向至www.tmooc.cn)

### 1) 修改Nginx服务配置

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

.. ..

server {

listen 80;

server\_name localhost;

rewrite ^/ http://www.tmooc.cn/;

location / {

root html;

index index.html index.htm;

# rewrite /a.html /b.html redirect;

}

}

### 2）重新加载配置文件

[root@proxy ~]# /usr/local/nginx/sbin/nginx -s reload

#请先确保nginx是启动状态，否则运行该命令会报错,报错信息如下：

#[error] open() "/usr/local/nginx/logs/nginx.pid" failed (2: No such file or directory)

### 3）客户端测试（真实机测试，真实机才可以连接tmooc）

[root@room9pc01 ~]# firefox http://192.168.4.5

## 步骤四：修改配置文件(访问192.168.4.5/下面子页面，重定向至www.tmooc.cn/下相同的页面)

### 1) 修改Nginx服务配置

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

.. ..

server {

listen 80;

server\_name localhost;

rewrite ^/(.\*)$ http://www.tmooc.cn/$1; #(.\*)复制参数， $1粘贴参数

location / {

root html;

index index.html index.htm;

}

}

### 2）重新加载配置文件

[root@proxy ~]# /usr/local/nginx/sbin/nginx -s reload

#请先确保nginx是启动状态，否则运行该命令会报错,报错信息如下：

#[error] open() "/usr/local/nginx/logs/nginx.pid" failed (2: No such file or directory)

### 3）客户端测试（真实机测试，真实机才可以连接tmooc）

[root@room9pc01 ~]# firefox http://192.168.4.5

[root@room9pc01 ~]# firefox http://192.168.4.5/test

## 步骤五：修改配置文件(实现curl和火狐访问相同链接返回的页面不同)

### 1) 创建网页目录以及对应的页面文件：

[root@proxy ~]# echo "I am Normal page" > /usr/local/nginx/html/test.html

[root@proxy ~]# mkdir -p /usr/local/nginx/html/firefox/

[root@proxy ~]# echo "firefox page" > /usr/local/nginx/html/firefox/test.html

### 2) 修改Nginx服务配置

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

.. ..

server {

listen 80;

server\_name localhost;

location / {

root html;

index index.html index.htm;

}

#这里，~符号代表正则匹配，\*符号代表不区分大小写

if ($http\_user\_agent ~\* firefox) { //识别客户端firefox浏览器

rewrite ^(.\*)$ /firefox/$1;

}

}

### 3）重新加载配置文件

[root@proxy ~]# /usr/local/nginx/sbin/nginx -s reload

#请先确保nginx是启动状态，否则运行该命令会报错,报错信息如下：

#[error] open() "/usr/local/nginx/logs/nginx.pid" failed (2: No such file or directory)

### 4）客户端测试

[root@client ~]# firefox http://192.168.4.5/test.html

[root@client ~]# curl http://192.168.4.5/test.html

### 5）地址重写格式【总结】

rewrite 旧地址 新地址 [选项];

last 不再读其他rewrite

break 不再读其他语句，结束请求

redirect 临时重定向

permament 永久重定向

## 地址重写案例1

访问www.a.com/a.html 返回 www.a.com/b.html的内容

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

server {

listen 80;

server\_name www.a.com;

rewrite /a.html /b.html redirect

#在server里面添加rewrite 访问a.html 转到b.html

[root@proxy ~]# firefox www.a.com/a.html

#未加redirect 地址栏显示a.html 访问的是b.html

#加了redirect 地址栏显示b.html 访问的也是b.html

## 地址重写案例2:

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

server {

listen 80;-

server\_name www.a.com;

rewrite ^/ http://www.tmooc.cn;

# ^/ 将所有的www.a.com网站转到www.tmooc.cn

## 地址重写案例3:

访问www.a.com/aa/bb/cc 跳转到www.tmooc.cn/aa/bb/cc

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

server {

listen 80;

server\_name www.a.com;

rewrite ^/(.\*) http://www.tmooc.cn/$1;

# (.\*) 复制 $1粘贴

## 地址重写案例4:

电脑访问:www.a.com/a.html -->/usr/local/nginx/html/a.html

手机访问:www.a.com/a.html -->/usr/local/nginx/phone/html/a.html

可以利用使用浏览器不一样,返回不同的页面,

也可以利用使用操作系统不一样,返回不同的页面

/usr/local/nginx/html/test.html

/usr/local/nginx/html/firefox/test.html

[root@proxy ~]# echo "normal" >/usr/local/nginx/html/test.html

[root@proxy ~]# mkdir /usr/local/nginx/html/firefox

[root@proxy ~]# ls /usr/local/nginx/html/

50x.html b.html firefox index.html mysql.php test.html test.php

[root@proxy ~]# echo "fox" >/usr/local/nginx/html/firefox/test.html

[root@proxy ~]# cat /usr/local/nginx/html/firefox/test.html

fox

[root@proxy ~]# cat /usr/local/nginx/html/test.

test.html test.php

[root@proxy ~]# cat /usr/local/nginx/html/test.html

normal

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf #配置nginx.conf

server {

listen 80;

server\_name www.a.com;

if ($http\_user\_agent ~\* firefox) {

rewrite /(.\*) /firefox/$1;

}

#以上在一个server中加入if 判断,如果是用firefox 访问的将执行rewrite 地址重写

常用正则

. ： 匹配除换行符以外的任意字符

? ： 重复0次或1次

+ ： 重复1次或更多次

\* ： 重复0次或更多次

\d ：匹配数字

^ ： 匹配字符串的开始

$ ： 匹配字符串的介绍

{n} ： 重复n次

{n,} ： 重复n次或更多次

[c] ： 匹配单个字符c

[a-z] ： 匹配a-z小写字母的任意一个

小括号()之间匹配的内容，可以在后面通过$1来引用，$2表示的是前面第二个()里的内容。正则里面容易让人困惑的是\转义特殊字符。

## 地址重写案例5:

http {

# 定义image日志格式

log\_format imagelog '[$time\_local] ' $image\_file ' ' $image\_type ' ' $body\_bytes\_sent ' ' $status;

# 开启重写日志

rewrite\_log on;

server {

root /home/www;

location / {

# 重写规则信息

error\_log logs/rewrite.log notice;

# 注意这里要用‘’单引号引起来，避免{}

rewrite '^/images/([a-z]{2})/([a-z0-9]{5})/(.\*)\.(png|jpg|gif)$' /data?file=$3.$4;

# 注意不能在上面这条规则后面加上“last”参数，否则下面的set指令不会执行

set $image\_file $3;

set $image\_type $4;

}

location /data {

# 指定针对图片的日志格式，来分析图片类型和大小

access\_log logs/images.log mian;

root /data/images;

# 应用前面定义的变量。判断首先文件在不在，不在再判断目录在不在，如果还不在就跳转到最后一个url里

try\_files /$arg\_file /image404.html;

}

location = /image404.html {

# 图片不存在返回特定的信息

return 404 "image not found\n";

}

}

# 日志文件

[root@proxy nginx]# cd logs

[root@proxy logs]# ls

access.log error.log

日志说明:在配置文件中有日志说明

[root@proxy ~]#vim /usr/local/nginx/conf/nginx.conf:

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

22 # '$status $body\_bytes\_sent "$http\_referer" '

23 # '"$http\_user\_agent" "$http\_x\_forwarded\_for"';

24

25 #access\_log logs/access.log main;

$remote\_addr

$remote\_user

[$time\_local]

$request

$status 状态

$body\_bytes\_sent 访问的流量大小

"$http\_referer" 网站是从哪过来的(哪个链接)

'"$http\_user\_agent" 客户端的用户信息

[root@proxy logs]# vim access.log

192.168.4.5 - - [18/Jan/2019:15:34:42 +0800] "GET /favicon.ico HTTP/1.1" 401 195 "-" "Mozilla/5.0 (X11; Linux x86\_64; rv:52.0) Gecko/20100101 Firefox/52.0"

@

"GET /favicon.ico 每个网站默认访问,logo图标.行业约定俗成规定

rewrite /a.html /b.html;

rewrite /b.html /c.html;

访问a,返回c,

-----------------------------------------------------------------

rewrite /a.html /b.html last; #直接结束匹配,不再读其他语句,结束请求

rewrite /b.html /c.html

访问a返回b,不再向下执行,访问b,返回c

-------------------------------------------------------------------------

location /a.html {

rewrite /a.html /b.html last; #还有其他语句

}

location /b.html {

rewrite /b.html /c.html;

}

访问a,返回c,

---------------------------------------------------------------------

location /a.html {ast

rewrite /a.html /b.html break; #直接结束匹配,不再读其他语句,结束请求

}

location /b.html {

rewrite /b.html /c.html;

}

访问a,返回b,

rewrite /a.html /b.html redirect; 临时重定向

rewrite /a.html /b.html permament; 永久重定向

临时重定向\永久重定向

蜘蛛,耙网站(百度,搜狗,google,360等搜索引擎)

# NSD Operation DAY03

案例1：Nginx反向代理(第七层代理)

案例2：Nginx的TCP/UDP调度器(第四层代理)

案例3：Nginx常见问题处理

# 1 案例1：Nginx反向代理(七层)

./configure时候不用安装额外的模块

反向代理（Reverse Proxy）方式是指以代理服务器来接受Internet上的连接请求，然后将请求转发给内部网络上的服务器；并将从服务器上得到的结果返回给Internet上请求连接的客户端，此时代理服务器对外就表现为一个服务器。

通常的代理服务器，只用于代理内部网络对Internet的连接请求，客户机必须指定代理服务器,并将本来要直接发送到Web服务器上的http请求发送到代理服务器中。当一个代理服务器能够代理外部网络上的主机，访问内部网络时，这种代理服务的方式称为反向代理服务。

七层代理(应用层)

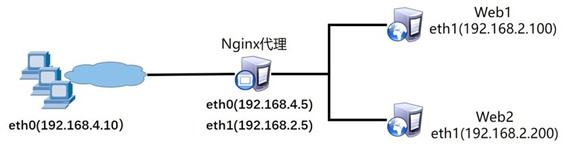
物理层 数据链路层 网络层 传输层 会话层 表示层 应用层

使用Nginx实现Web反向代理功能，实现如下功能：

* 后端Web服务器两台，可以使用httpd实现
* Nginx采用轮询的方式调用后端Web服务器
* 两台Web服务器的权重要求设置为不同的值
* 最大失败次数为1，失败超时时间为30秒

1.2 方案

使用4台RHEL7虚拟机，其中一台作为Nginx代理服务器，该服务器需要配置两块网卡，IP地址分别为192.168.4.5和192.168.2.5，两台Web服务器IP地址分别为192.168.2.100和192.168.2.200。客户端测试主机IP地址为192.168.4.10。如图-1所示。



1.3 步骤

实现此案例需要按照如下步骤进行。

## 步骤一：部署实施后端Web服务器

### 1）部署后端Web1服务器

后端Web服务器可以简单使用yum方式安装httpd实现Web服务，为了可以看出后端服务器的不同，可以将两台后端服务器的首页文档内容设置为不同的内容。

[root@web1 ~]# yum -y install httpd

[root@web1 ~]# echo "192.168.2.100" > /var/www/html/index.html

[root@web1 ~]# systemctl restart httpd

[root@web1 ~]# firewall-cmd --set-default-zone=trusted

[root@web1 ~]# setenforce 0

### 2）部署后端Web2服务器

[root@web2 ~]# yum -y install httpd

[root@web2 ~]# echo "192.168.2.200" > /var/www/html/index.html

[root@web2 ~]# systemctl restart httpd

[root@web2 ~]# firewall-cmd --set-default-zone=trusted

[root@web2 ~]# setenforce 0

## 步骤二：配置Nginx服务器，添加服务器池，实现反向代理功能

### 1）修改/usr/local/nginx/conf/nginx.conf配置文件

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

...

http{

...

upstream webserver { #增加名为”server”的集群,在http内加

server 192.168.2.100:80;

server 192.168.2.200:80;

}

server {

listen 80;

server\_name www.a.com;

if ($http\_user\_agent ~\* firefox) {

rewrite /(.\*) /firefox/$1;

location / {

proxy\_pass http://webserver; #调用函数

root html; #可有可无,无任何作用

index index.html index.htm; #可有可无,无任何作用

}

}

### 2）重启nginx服务

[root@proxy ~]# nginx -s reload

[root@proxy ~]# curl http://192.168.4.5

web11

[root@proxy ~]# curl http://192.168.4.5

web2222

### 3）客户端使用浏览器访问代理服务器测试轮询效果

upstream webserver {

server 192.168.2.100:80 weight=2; #添加权重属性

server 192.168.2.200:80;

}

[root@proxy ~]# nginx -s reload

[root@proxy ~]# curl http://192.168.4.5

web11

[root@proxy ~]# curl http://192.168.4.5

web11

[root@proxy ~]# curl http://192.168.4.5

web11

[root@proxy ~]# curl http://192.168.4.5

web2222

## 步骤三：配置upstream服务器集群池属性

### 1）设置失败次数，超时时间，权重

weight可以设置后台服务器的权重，max\_fails可以设置后台服务器的失败次数，fail\_timeout可以设置后台服务器的失败超时时间。

#weight设置服务器权重值，默认值为1

#max\_fails=2 设置最大失败次数,尝试2次不成功,换下个server

#fail\_timeout=30设置失败超时时间，单位为秒,请求无反应,30s内不在询问

#down标记服务器已关机，不参与集群调度

在定义上新增:

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

.. ..

http {

.. ..

upstream webserver {

server 192.168.2.100 weight=1 max\_fails=1 fail\_timeout=30;

server 192.168.2.200 weight=2 max\_fails=2 fail\_timeout=30;

server 192.168.2.101 down;

}

server {

listen 80;

server\_name localhost;

location / {

proxy\_pass http://webserver;

}

}

### 2）重启nginx服务

[root@proxy ~]# /usr/local/nginx/sbin/nginx -s reload

#请先确保nginx是启动状态，否则运行该命令会报错,报错信息如下：

#[error] open() "/usr/local/nginx/logs/nginx.pid" failed (2: No such file or directory)

### 3）关闭一台后端服务器（如web1）

[root@web1 ~]# systemctl stop httpd

### 4）客户端使用浏览器访问代理服务器测试轮询效果

[root@client ~]# curl http://192.168.4.5 //使用该命令多次访问查看效果

### 5）再次启动后端服务器的httpd（如web1）

[root@web1 ~]# systemctl start httpd

### 6）客户端再次使用浏览器访问代理服务器测试轮询效果

[root@client ~]# curl http://192.168.4.5 //使用该命令多次访问查看效果

## 步骤四：配置upstream服务器集群的调度算法

### 设置相同客户端访问相同Web服务器

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

.. ..

http {

.. ..

#通过ip\_hash设置调度规则为：相同客户端访问相同服务器

在定义上增加:

upstream webserver {

ip\_hash;

server 192.168.2.100:80 max\_fails=2 fail\_timeout=10;

server 192.168.2.200:80;

}

.. ..

server {

listen 80;

server\_name www.tarena.com;

location / {

proxy\_pass http://webserver;

}

}

### 2）重启nginx服务

[root@proxy ~]# /usr/local/nginx/sbin/nginx -s reload

#请先确保nginx是启动状态，否则运行该命令会报错,报错信息如下：

#[error] open() "/usr/local/nginx/logs/nginx.pid" failed (2: No such file or directory)

### 3）客户端使用浏览器访问代理服务器测试轮询效果

[root@client ~]# curl http://192.168.4.5 //使用该命令多次访问查看效果

### 4) ip\_hash原理

1/生成 哈希值加密16进制数,

[root@proxy ~]# echo "192.168.4.10" |md5sum

95e033fe68643fd1e4c11190191baead -

2/将16进制数%2取余,(集群中有2台server)

取余之后要么是0要么是1,,,比如是0,再将0定位给第一个server.

每次来访问都将这个IP换算哈希值,再取余来判断值.最后定位分配server

7层代理,调度(HTTP)

4层代理,调度(tcp/udp)

# 4层与7层代理的区别

所谓四层就是基于IP+端口的负载均衡；七层就是基于URL等应用层信息的负载均衡；同理，还有基于MAC地址的二层负载均衡和基于IP地址的三层负载均衡。 换句换说，二层负载均衡会通过一个虚拟MAC地址接收请求，然后再分配到真实的MAC地址；三层负载均衡会通过一个虚拟IP地址接收请求，然后再分配到真实的IP地址；四层通过虚拟IP+端口接收请求，然后再分配到真实的服务器；七层通过虚拟的URL或主机名接收请求，然后再分配到真实的服务器。

4层是TCP层，使用IP+端口的方式。类似路由器，只是修改下IP地址，然后转发给后端服务器，TCP三次握手是直接和后端连接的。只不过在后端机器上看到的都是与代理机的IP的established而已。

7层代理则必须要先和代理机三次握手后，才能得到7层（HTT层）的具体内容，然后再转发,能写更多的转发规则

七层应用负载的好处，是使得整个网络更智能化。

将对图片类的请求转发到特定的图片服务器并可以使用缓存技术；

将对文字类的请求可以转发到特定的文字服务器并可以使用压缩技术。

这种方式可以对客户端的请求和服务器的响应进行任意意义上的修改，极大的提升了应用系统在网络层的灵活性。很多在后台，例如Nginx或者Apache上部署的功能可以前移到负载均衡设备上，例如客户请求中的Header重写，服务器响应中的关键字过滤或者内容插入等功能。

另外一个功能就是安全性。网络中最常见的SYN Flood攻击，即黑客控制众多源客户端，使用虚假IP地址对同一目标发送SYN攻击，通常这种攻击会大量发送SYN报文，耗尽服务器上的相关资源，以达到Denial of Service(DoS)的目的。从技术原理上也可以看出，四层模式下这些SYN攻击都会被转发到后端的服务器上；而七层模式下这些SYN攻击自然在负载均衡设备上就截止，不会影响后台服务器的正常运营。另外负载均衡设备可以在七层层面设定多种策略，过滤特定报文，例如SQL Injection等应用层面的特定攻击手段，从应用层面进一步提高系统整体安全。

7层负载均衡，主要还是着重于应用HTTP协议，所以其应用范围主要是众多的网站或者内部信息平台等基于B/S开发的系统。

4层负载均衡则对应其他TCP应用，例如基于C/S开发的ERP等系统。

# 2 案例2：Nginx的TCP/UDP调度器(四层代理)

2.1 问题

四层代理 ---实现ssh远程连接

使用Nginx实现TCP/UDP调度器功能，实现如下功能：

* 后端SSH服务器两台
* Nginx编译安装时需要使用--with-stream开启ngx\_stream\_core\_module模块
* Nginx采用轮询的方式调用后端SSH服务器

结果:用client ssh 192.168.4.5,直接轮询192.168.2.100与192.168.2.200

并不是做网站,做第4层代理

nginx从1.9版本才开始支持该功能,

## 步骤一：部署支持4层TCP/UDP代理的Nginx服务器

### 1）部署nginx服务器

编译安装必须要使用--with-stream参数开启4层代理模块。[root@proxy ~]# nginx -s stop #停止nginx

[root@proxy ~]# cp -r /usr/local/nginx /root/nginx #备份原nginx

[root@proxy ~]# rm -rf /usr/local/nginx #源码安装卸载直接删除安装的目录即可

[root@proxy ~]# cd nginx-1.12.2/ #进入 新版neginx目录

[root@proxy nginx-1.12.2]# ./configure \ #配置

> --prefix=/usr/local/nginx \

> --user=nginx \

> --group=nginx \

> --with-http\_ssl\_module \

> --with-stream #配置添加模块

[root@proxy nginx-1.12.2]# make && make install #编译和安装

[root@proxy nginx-1.12.2]# nginx #启动,因为之前做了软连接,否则使用目录启动

## 步骤二：配置Nginx服务器，添加服务器池，实现TCP/UDP反向代理功能

### 1）修改/usr/local/nginx/conf/nginx.conf配置文件

[root@proxy nginx-1.12.2]# vim /usr/local/nginx/conf/ #编辑配置文件

stream {

upstream backend {

server 192.168.2.100:22; #ssh端口为22

server 192.168.2.200:22;

}

server {

listen 12345; #启用12345端口为nginx监听端口

proxy\_connect\_timeout 1s; #连接超时

proxy\_timeout 3s; #访问超时

proxy\_pass backend; #调用集群

}

}

http { #!!!一定要在http之外添加以上内容

include mime.types;

default\_type application/octet-stream;

...

}

### 2）重启nginx服务

[root@proxy nginx-1.12.2]# nginx -s reload

### 3）客户端使用访问代理服务器测试--with-stream轮询效果

**客户端远程测试:**

**远程客户端远程用ssh 方式远程连接nginx调度器4层代理(本实验的192.168.4.5),实际上轮询连接了两个web服务器,192.168.2.100和192.168.2.200**

[root@client ~]# ssh -p 12345 192.168.4.5 #客户端地址:192.168.4.10开始SSH

root@192.168.4.5's password:

Last login: Mon Jan 21 15:55:12 2019 from 192.168.2.5

[root@Web1111 ~]# exit #Web1111地址是:192.168.2.100

登出

Connection to 192.168.4.5 closed.

[root@client ~]# ssh -p 12345 192.168.4.5

root@192.168.4.5's password:

Last login: Mon Jan 21 16:05:34 2019 from 192.168.2.5

[root@Web222 ~]# #Web222地址是:192.168.2.200

----------可多个stream-----------------

stream {

upstream 名称2 {

server 192.168.2.100:22; #ssh端口为22

server 192.168.2.200:22;

}

server {

listen 23456; #启用23456端口为nginx监听端口

proxy\_connect\_timeout 1s; #连接超时

proxy\_timeout 3s; #访问超时

proxy\_pass 名称2;

}

}

# 3 案例3：Nginx常见问题处理

3.1 问题

本案例要求对Nginx服务器进行适当优化，解决如下问题，以提升服务器的处理性能：

1. 如何自定义返回给客户端的404错误页面
2. 如何查看服务器状态信息
3. 如果客户端访问服务器提示“Too many open files”如何解决
4. 如何解决客户端访问头部信息过长的问题
5. 如何让客户端浏览器缓存数据
6. 日志切割
7. 开启gzip压缩功能，提高数据传输效率
8. 开启文件缓存功能

然后客户机访问此Web服务器验证效果：

* 使用ab压力测试软件测试并发量
* 编写测试脚本生成长头部信息的访问请求
* 客户端访问不存在的页面，测试404错误页面是否重定向

## 步骤一：自定义报错页面

### 1)：自定义404报错页面

[root@client ~]# firefox http://192.168.4.5/asdfasdfasdf 　＃访问不存在的网页

404 Not Found

nginx/1.12.2

[root@proxy conf]# vim /usr/local/nginx/conf/nginx.conf

server　｛

　　．．．．

error\_page 404 /404.html;　　＃定义报错返回的页面，事先准备好404.html文件

error\_page 505 /505.html;　　＃定义报错返回的页面，事先准备好505.html文件

charset utf-8;　　　　　　　　　　　　　　　　　　＃字符集支持中文

｝

### 2）修改Nginx配置文件，自定义报错页面

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

.. ..

error\_page 404 /40x.html; //自定义错误页面

.. ..

[root@proxy ~]# vim /usr/local/nginx/html/40x.html //编辑错误页面

Oops,No NO no page …

[root@proxy ~]# nginx -s reload

#请先确保nginx是启动状态，否则运行该命令会报错,报错信息如下：

#[error] open() "/usr/local/nginx/logs/nginx.pid" failed (2: No such file or directory)

### 3）优化后，客户端使用浏览器访问不存在的页面，会提示自己定义的40x.html页面

[root@proxy conf]# echo "<h1>123" >/usr/local/nginx/html/404.html

[root@client ~]# firefox http://192.168.4.5/asdfasdfasdf　　＃访问不存在的网页

123

# 4）nginx优化

## 步骤一 常见http状态码可用参考表-1所示。



## 步骤二：如何查看服务器状态信息（非常重要的功能）

1. 编译安装时使用--with-http\_stub\_status\_module开启状态页面模块

重装nginx

[root@Web1111 nginx-1.12.2]# .\configure \

> --prefix=/usr/local/nginx \

> --user=nginx \

> --group=nginx \

> --with-http\_ssl\_module \

> --with-stream \

> --with-http\_stub\_status\_module #开启状态页面模块

[root@proxy nginx-1.12.2]# vim /usr/local/nginx/conf/nginx.conf

location /suibianqiao {

stub\_status on;

}

[root@proxy nginx-1.12.2]# firefox 192.168.4.5/suibianqiao

Active connections: 1

server accepts handled requests

6 6 102

Reading: 0 Writing: 1 Waiting: 0

#如果前面两个数字大,说明服务器大部分流量去处理;连接/断开请求,数据传输并不多.

nginx 重启,数字重置

HTTP(1.0) 1次链接,1次请求

HTTP(1.1) 1次链接,多次请求

Active connections：当前活动的连接数量。

Accepts：已经接受客户端的连接总数量。 6

Handled：已经处理客户端的连接总数量（一般与accepts一致，除非服务器限制了连接数量）。 6

Requests：客户端发送的请求数量。 102

Reading：当前服务器正在读取客户端请求头的数量。 #数字为0,表示服务器无压力

Writing：当前服务器正在写响应信息的数量。

Waiting：当前多少客户端在等待服务器的响应。

nginx 重启,数字重置

静态页面：主要是消耗内存与网卡流量

动态页面：主要是消耗CPU ,运行脚本

## 步骤三：优化Nginx并发量

1. 优化前使用ab高并发测试

-c并发量 -n 次数 -n必须大于-c

[root@proxy ~]# nginx

[root@proxy ~]# ab -n 100 -c 100 http://192.168.4.5/

[root@proxy ~]# curl http://192.168.4.5/suibianqiao

Active connections: 1

server accepts handled requests

101 101 101

Reading: 0 Writing: 1 Waiting: 0

[root@proxy ~]# ab -n 100 -c 100 http://192.168.4.5/

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

#以下为全局配置：

worker\_processes 1; #启用多少可进程,跟CPU核心数量一致,例如八核就填8

events {

worker\_connections 65536; #每个worker\_能满足多少并发量,默认为1024

use epoll; #epoll： nginx的查询方式,

}

epoll 是nginx的查询方式,

3）优化Linux内核参数（最大文件数量）

[root@proxy ~]# ulimit -a #查看所有属性

core file size (blocks, -c) 0

data seg size (kbytes, -d) unlimited

scheduling priority (-e) 0

file size (blocks, -f) unlimited

pending signals (-i) 3845

max locked memory (kbytes, -l) 64

max memory size (kbytes, -m) unlimited

open files (-n) 1024

pipe size (512 bytes, -p) 8

POSIX message queues (bytes, -q) 819200

real-time priority (-r) 0

stack size (kbytes, -s) 8192

cpu time (seconds, -t) unlimited

max user processes (-u) 3845

virtual memory (kbytes, -v) unlimited

file locks (-x) unlimited

[root@proxy ~]# ulimit -Sn 10000 #设置软限制(临时规则)

[root@proxy ~]# ulimit -Hn 10000 #设置硬限制(临时规则)

[root@proxy ~]# vim /etc/security/limits.conf #永久设置,内核参数,必须重启生效

.. ..

\* soft nofile 100000

\* hard nofile 100000

#该配置文件分4列，分别如下：

#用户或组(\*) 硬限制(hard)或软限制(soft) 需要限制的项目(nofile) 限制的值

## ab工具: 压力测试



[root@proxy ~]#yum –y install http-tools # apache自带的压力测试工具,安装httpd-tools而来

[root@proxy ~]#ab –V #查看版本

[root@proxy ~]# ab -n 2000 -c 2000 http://192.168.4.5/ #-c 并发数 –n请求总数 -n必须>=-c

Percentage of the requests served within a certain time (ms

50% 12

66% 24

75% 32

80% 32

90% 48

95% 48

98% 49

99% 49

100% 202 (longest request)

[root@proxy ~]# curl 192.168.4.5/suibianqiao

Active connections: 1

server accepts handled requests

18104 18104 16649

Reading: 0 Writing: 1 Waiting: 0

**其他压力测试工具(需要额外下载)**：http\_load、webbench、siege

## 步骤四：优化Nginx数据包头缓存（414）

**出现414报错怎么解决**

1）优化前，使用脚本测试长头部请求是否能获得响应

[root@proxy ~]# cat lnmp\_soft/buffer.sh

--------脚本---------------------------------------

#!/bin/bash

URL=http://192.168.4.5/index.html?

for i in {1..5000}

do

URL=${URL}v$i=$i

done

curl $URL #经过5000次循环后，生成一个很长的URL地址栏,并访问

---------------------------------------------

[root@proxy ~]# ./buffer.sh

.. ..

<center><h1>414 Request-URI Too Large</h1></center> #报错:提示头部信息过大

2）修改Nginx配置文件，增加数据包头部缓存大小,解决414报错

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

.. ..

http {

client\_header\_buffer\_size 1k; //手打上去,默认请求包头信息的缓存

large\_client\_header\_buffers 4 4k; //手打上去,大请求包头部信息的缓存个数与容量

.. ..

}

[root@proxy ~]# nginx -s reload

3）优化后，使用脚本测试长头部请求是否能获得响应

## 步骤五：浏览器本地缓存静态数据

(每个浏览器都有,只是命令不一)

1）使用Firefox浏览器查看缓存

以Firefox浏览器为例，在Firefox地址栏内输入about:cache将显示Firefox浏

览器的缓存信息，如图-3所示，点击List Cache Entries可以查看详细信息。

3）修改Nginx配置文件，定义对静态页面的缓存时间(适合静态)

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

server {

listen 80;

server\_name localhost;

location / {

root html;

index index.html index.htm;

}

location ~\* \.(jpg|jpeg|gif|png|css|js|ico|xml)$ { #在server中添加location

expires 30d; //定义客户端缓存时间为30天

}

location ~\* \.(mp4)$ { #可添加多个

expires 30d; //定义客户端缓存时间为30天

}

说明:location ~\* \.(jpg|jpeg|gif|png|css|js|ico|xml)$ ,定义缓存类型,

[root@proxy ~]# cp /usr/share/backgrounds/day.jpg /usr/local/nginx/html

[root@proxy ~]# nginx -s reload

[root@proxy lnmp\_soft]# firefox http://192.168.4.5/day.jpg

访问之后,再去about:cache查看缓存信息

## 步骤六：日志切割

日志文件越来越大怎么办？单个文件10G? 如何切割？（非常常见的面试题）

步骤：1. 把旧的日志重命名 2. kill USR1 PID(nginx的进程PID号)

可查询进程命令: ps pgrep ss cat nginx.pid

[root@proxy ~]# ps aux |grep nginx

[root@proxy ~]# pgrep nginx

[root@proxy ~]# ss -lnp |grep nginx

[root@proxy ~]# pidof nginx

**kill命令**：向进程发送控制信号，以实现对进程管理，不单是杀死进程

显示当前系统可用信号：

# kill -l 列出kill支持能发出的信号

常用信号：1) SIGHUP: 无须关闭进程让其重读配置文件，就是不重启就让修改的配置生效；

2) SIGINT: 中止正在运行的进程；相当于Ctrl+c；

9) SIGKILL: 杀死正在运行的进程；

15) SIGTERM：终止正在运行的进程，温柔的关闭；

18) SIGCONT：进程继续态

19) SIGSTOP：进程暂停态

指定信号的方法：(1) 信号的数字标识；1, 2, 9

(2) 信号完整名称；SIGHUP

(3) 信号的简写名称；HUP，就是把前面的SIG去掉就行

向进程发信号：kill [-SIGNAL] PID...

Kill 发送一个信号给进程,不填参数就是 -15

Kill 9031 === kill -15 9031 #告诉9031自杀

Kill -9 9031 #告诉9031 必须死,

kill -USR2 9031 ==kill -12 9031 #告诉9031进程,日志被删除,等于是-12参数

Kill -19 进程号 暂停进程

kill -18 进程号 继续进程

1）手动执行

备注：/usr/local/nginx/logs/nginx.pid文件中存放的是nginx的进程PID号。

[root@proxy ~]# mv access.log access2.log

[root@proxy ~]# kill -USR1 $(cat /usr/local/nginx/logs/nginx.pid)

2）自动完成

每周5的03点03分自动执行脚本完成日志切割工作。

[root@proxy ~]# vim /usr/local/nginx/logbak.sh

-----脚本----------------

#!/bin/bash

date=`date +%Y%m%d`

logpath=/usr/local/nginx/logs

mv $logpath/access.log $logpath/access-$date.log

mv $logpath/error.log $logpath/error-$date.log

kill -USR1 $(cat $logpath/nginx.pid)

---------------------------------------------------------------------------

[root@proxy ~]# crontab -e

03 03 \* \* 5 /usr/local/nginx/logbak.sh

## 步骤七：对页面进行压缩处理

1）修改Nginx配置文件

[root@proxy ~]# cat /usr/local/nginx/conf/nginx.conf

http {

.. ..

gzip on; //开启压缩

gzip\_min\_length 1000; //小文件不压缩

gzip\_comp\_level 4; //压缩比率

gzip\_types text/plain text/css application/json application/x-javascript text/xml application/xml application/xml+rss text/javascript;

//对特定文件压缩，类型参考mime.types

.. ..

}

MP3本身就是压缩文件,

查看文件类型:[root@proxy ~]# vim /usr/local/nginx/conf/mime.types

所有的浏览器都是自带解压器

## 步骤八：服务器内存缓存

1）如果需要处理大量静态文件，可以将文件缓存在内存，下次访问会更快。

http {

open\_file\_cache max=2000 inactive=20s;

open\_file\_cache\_valid 60s;

open\_file\_cache\_min\_uses 5;

open\_file\_cache\_errors off;

//设置服务器最大缓存2000个文件句柄，关闭20秒内无请求的文件句柄

//文件句柄的有效时间是60秒，60秒后过期

//只有访问次数超过5次会被缓存

//缓存报错关闭,缓存出错不报警

}

如果a文件被缓存到内存,20s内无访问即被删掉,60s后将被删掉,如a又被访问了5次将被重新缓存到内存,

# 已安装好的nginx添加模块

## 1 查看当前nginx编译参数

[root@lnmp01 nginx-1.12.2]# /usr/local/nginx/sbin/nginx -V #查看已有模块

nginx version: nginx/1.12.2

built by gcc 4.8.5 20150623 (Red Hat 4.8.5-16) (GCC)

built with OpenSSL 1.0.2k-fips 26 Jan 2017

TLS SNI support enabled

configure arguments: --with-http\_ssl\_module --with-stream

## 2 添加未编译安装模块（停掉服务）

[root@lnmp01 nginx-1.12.2]# /usr/local/nginx/sbin/nginx -s stop

[root@lnmp01 nginx-1.12.2]# netstatu -unltp|grep 80

重新编译

[root@lnmp01 nginx-1.12.2]# ./configure --with-http\_ssl\_module --with-stream --with-http\_stub\_status\_module

checking for OS

注意：千万不要make install，否则就是覆盖安装

## 3 替换nginx二进制文件

当前nginx文件最好备份下，避免出问题。

停止nginx服务

[root@lnmp01 nginx-1.12.2]# cp /usr/local/nginx/sbin/nginx /usr/local/nginx/sbin/nginx.bak

[root@lnmp01 nginx-1.12.2]# pwd

/root/lnmp\_soft/nginx-1.12.2

[root@lnmp01 nginx-1.12.2]# cp objs/nginx /usr/local/nginx/sbin/

重启下nginx服务，重新查看模块加载情况：

[root@lnmp01 objs]# /usr/local/nginx/sbin/nginx -V

nginx version: nginx/1.12.2

built by gcc 4.8.5 20150623 (Red Hat 4.8.5-16) (GCC)

built with OpenSSL 1.0.2k-fips 26 Jan 2017

TLS SNI support enabled

configure arguments: --with-http\_ssl\_module --with-stream --with-http\_stub\_status\_module

# Nginx的location基础配置

=开头表示精确匹配，优先级最高。如果找到了这个精确匹配，则停止查找

如 A 中只匹配根目录结尾的请求，后面不能带任何字符串。

^~ 开头表示uri以某个常规字符串开头，不是正则匹配

~ 开头表示区分大小写的正则匹配;

~\* 开头表示不区分大小写的正则匹配

/ 通用匹配, 如果没有其它匹配,任何请求都会匹配到

优先级为： = > 完整路径 > ^~ > ~、~\* > 部分起始路径 > /

location = / {

# 精确匹配 / ，主机名后面不能带任何字符串，匹配到后，停止继续匹配

[ configuration A ]

}

location / {

# 因为所有的地址都以 / 开头，所以这条规则将匹配到所有请求

# 但是正则和最长字符串会优先匹配

[ configuration B ]

}

location /documents/ {

# 匹配任何以 /documents/ 开头的地址，匹配符合以后，还要继续往下搜索

# 只有后面的正则表达式没有匹配到时，这一条才会采用这一条

[ configuration C ]

}

location ~ /documents/Abc {

# 正则匹配，区分大小写。

# 匹配任何以 /documents/Abc 开头的地址，匹配符合以后，还要继续往下搜索

# 只有后面的正则表达式没有匹配到时，这一条才会采用这一条

[ configuration CC ]

}

location ^~ /images/ {

# 匹配任何以 /images/ 开头的地址，匹配符合以后，停止往下搜索正则，采用这一条。

[ configuration D ]

}

location ~\* \.(gif|jpg|jpeg)$ {

# 匹配所有以 gif,jpg或jpeg 结尾的请求

# 然而，所有请求 /images/ 下的图片会被 config D 处理，因为 ^~ 到达不了这一条正则

[ configuration E ]

}

location /images/ {

# 字符匹配到 /images/，继续往下，会发现 ^~ 存在

[ configuration F ]

}

location /images/abc {

# 最长字符匹配到 /images/abc，继续往下，会发现 ^~ 存在

# F与G的放置顺序是没有关系的

[ configuration G ]

}

location ~ /images/abc/ {

# 只有去掉 config D 才有效：先最长匹配 config G 开头的地址，继续往下搜索，匹配到这一条正则，采用

[ configuration H ]

}

location ~\* /js/.\*/\.js

按照以上的location配置，如下列举部分实际匹配的情况：

/ -> config A 精确完全匹配，即使/index.html也匹配不了

/downloads/download.html -> config B 匹配B以后，往下没有任何匹配，采用B

/images/1.gif -> configuration D 匹配到F，往下匹配到D，停止往下

/images/abc/def -> config D 最长匹配到G，往下匹配D，停止往下

#你可以看到 任何以/images/开头的都会匹配到D并停止，FG写在这里是没有任何意义的，H是永远轮不到的，这里只是为了说明匹配顺序

/documents/document.html -> config C 匹配到C，往下没有任何匹配，采用C

/documents/1.jpg -> configuration E 匹配到C，往下正则匹配到E

/documents/Abc.jpg -> config CC 最长匹配到C，往下正则顺序匹配到CC，不会往下到E

所以实际使用中，个人觉得至少有三个匹配规则定义，如下：

#直接匹配网站根，通过域名访问网站首页比较频繁，使用这个会加速处理，官网如是说。

#这里是直接转发给后端应用服务器了，也可以是一个静态首页

# 第一个必选规则

location = / {

proxy\_pass http://tomcat:8080/index

}

# 第二个必选规则是处理静态文件请求，这是nginx作为http服务器的强项

# 有两种配置模式，目录匹配或后缀匹配,任选其一或搭配使用

location ^~ /static/ {

root /webroot/static/;

}

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

root /webroot/res/;

}

#第三个规则就是通用规则，用来转发动态请求到后端应用服务器

#非静态文件请求就默认是动态请求，自己根据实际把握

#毕竟目前的一些框架的流行，带.php,.jsp后缀的情况很少了

location / {

proxy\_pass http://tomcat:8080/

}

# Nginx if指令

if判断指令

语法为if(condition){...}，对给定的条件condition进行判断。如果为真，大括号内的rewrite指令将被执行，if条件(conditon)可以是如下任何内容：

当表达式只是一个变量时，如果值为空或任何以0开头的字符串都会当做false

直接比较变量和内容时，使用=或!=

~正则表达式匹配，~\*不区分大小写的匹配，!~区分大小写的不匹配

-f和!-f用来判断是否存在文件

-d和!-d用来判断是否存在目录

-e和!-e用来判断是否存在文件或目录

-x和!-x用来判断文件是否可执行

案例：

if ($http\_user\_agent ~ MSIE) {

rewrite ^(.\*)$ /msie/$1 break; #break : 停止执行当前虚拟主机的后续rewrite指令集

} //如果UA包含"MSIE"，rewrite请求到/msid/目录下

if ($http\_cookie ~\* "id=([^;]+)(?:;|$)") {

set $id $1;

} //如果cookie匹配正则，设置变量$id等于正则引用部分

if ($request\_method = POST) {

return 405;

} //如果提交方法为POST，则返回状态405（Method not allowed）。return不能返回301,302

if ($slow) {

limit\_rate 10k;

} //限速，$slow可以通过 set 指令设置

if (!-f $request\_filename){

break;

proxy\_pass http://127.0.0.1;

} //如果请求的文件名不存在，则反向代理到localhost 。这里的break也是停止rewrite检查

if ($args ~ post=140){

rewrite ^ http://example.com/ permanent;

} //如果query string中包含"post=140"，永久重定向到example.com

location ~\* \.(gif|jpg|png|swf|flv)$ {

valid\_referers none blocked www.jefflei.com www.leizhenfang.com;

if ($invalid\_referer) {

return 404;

} //防盗链

}

# Nginx 配置 location 以及 return、rewrite 和 try\_files 指令

<https://blog.csdn.net/kikajack/article/details/80669260>

# 七财公司实战nginx配置文件解释：

## 案例一：全局配置nginx.confg

user nobody;

worker\_processes 4; #worker角色的工作进程的个数

error\_log logs/error.log; #指定日志文件

pid logs/nginx.pid; #记录pid的文件

events {

use epoll; #nginx默认使用epoll事件模型，因此nginx在Linux操作系统下效率相当高

worker\_connections 51200; #每一个worker进程能并发处理（发起）的最大连接数

}

http {

include mime.types;

default\_type application/octet-stream;

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

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" "$http\_x\_forwarded\_for"';

#access\_log logs/access.log main;

server\_tokens off; #隐藏nginx版本号

sendfile on;

tcp\_nopush on;

server\_names\_hash\_bucket\_size 256;

client\_header\_buffer\_size 256k;

keepalive\_timeout 300s 60s;

large\_client\_header\_buffers 4 256k;

client\_body\_buffer\_size 256k;

client\_header\_timeout 3m;

client\_body\_timeout 3m;

send\_timeout 3m;

client\_max\_body\_size 100m;

gzip on; #开启gzip压缩输出，减少网络传输

gzip\_min\_length 1k; #允许压缩的页面最小字节数，建议大于1k的字节数，小于1k可能会越压越大

gzip\_buffers 4 16k; #安装原始数据大小以16k为单位的4倍申请内存

gzip\_http\_version 1.0; #用于识别 http 协议的版本

gzip\_comp\_level 2; # gzip压缩比，1压缩比最小处理速度最快，9压缩比最大但处理速度最慢 gzip\_types text/plain application/x-javascript text/css application/xml; #这些类型都会被压缩

gzip\_vary on; #在响应头加Vary: Accept-Encoding，让前端的缓存服务器缓存经过gzip压缩的页面

proxy\_connect\_timeout 300s;

proxy\_read\_timeout 300s;

proxy\_send\_timeout 300s;

proxy\_buffer\_size 16k;

proxy\_buffers 4 64k;

proxy\_busy\_buffers\_size 128k;

proxy\_temp\_file\_write\_size 128k;

proxy\_temp\_path /opt/tmp\_dir;

proxy\_cache\_path /opt/cache levels=1:2 keys\_zone=cache\_one:200m inactive=1d max\_size=30g;

include vhost/\*.conf;

}

main全局配置：

main全局配置说明：nginx在运行时与具体业务功能（比如http服务或者email服务代理）无关的一些参数，比如工作进程数，运行的身份等。

* woker\_processes 2：在配置文件的顶级main部分，worker角色的工作进程的个数，master进程是接收并分配请求给worker处理。这个数值简单一点可以设置为cpu的核数grep ^processor /proc/cpuinfo | wc -l，也是 auto 值，如果开启了ssl和gzip更应该设置成与逻辑CPU数量一样甚至为2倍，可以减少I/O操作。如果nginx服务器还有其它服务，可以考虑适当减少。
* worker\_cpu\_affinity：也是写在main部分。在高并发情况下，通过设置cpu粘性来降低由于多CPU核切换造成的寄存器等现场重建带来的性能损耗。如worker\_cpu\_affinity 0001 0010 0100 1000; （四核）。
* worker\_rlimit\_nofile 10240：写在main部分。默认是没有设置，可以限制为操作系统最大的限制65535。

events部分：

events部分说明：

* worker\_connections 2048：写在events部分。每一个worker进程能并发处理（发起）的最大连接数（包含与客户端或后端被代理服务器间等所有连接数）。nginx作为反向代理服务器，计算公式 最大连接数 = worker\_processes \* worker\_connections/4，所以这里客户端最大连接数是1024，这个可以增到到8192都没关系，看情况而定，但不能超过前面的worker\_rlimit\_nofile的参数。当nginx作为http服务器时，计算公式里面是除以2。
* use epoll：写在events部分。在Linux操作系统下，nginx默认使用epoll事件模型，得益于此，nginx在Linux操作系统下效率相当高。同时Nginx在OpenBSD或FreeBSD操作系统上采用类似于epoll的高效事件模型kqueue。在操作系统不支持这些高效模型时才使用select。

http部分：

http部分说明：与提供http服务相关的一些配置参数。例如：是否使用keepalive，是否使用gzip进行压缩等。

* sendfile on：开启高效文件传输模式，sendfile指令指定nginx是否调用sendfile函数来输出文件，减少用户空间到内核空间的上下文切换。对于普通应用设为 on，如果用来进行下载等应用磁盘IO重负载应用，可设置为off，以平衡磁盘与网络I/O处理速度，降低系统的负载。
* keepalive\_timeout 65 : 长连接超时时间，单位是秒，这个参数很敏感，涉及浏览器的种类、后端服务器的超时设置、操作系统的设置，可以另外起一片文章了。长连接请求大量小文件的时候，可以减少重建连接的开销，但假如有大文件上传，65s内没上传完成会导致失败。如果设置时间过长，用户又多，长时间保持连接会占用大量资源。
* send\_timeout : 用于指定响应客户端的超时时间。这个超时仅限于两个连接活动之间的时间，如果超过这个时间，客户端没有任何活动，Nginx将会关闭连接。
* client\_max\_body\_size 10m：允许客户端请求的最大单文件字节数。如果有上传较大文件，请设置它的限制值
* client\_body\_buffer\_size 128k：缓冲区代理缓冲用户端请求的最大字节数

http\_proxy模块部分：

http\_proxy模块说明：这个模块实现的是nginx作为反向代理服务器的功能，包括缓存功能

* proxy\_connect\_timeout 60：nginx跟后端服务器连接超时时间(代理连接超时)
* proxy\_read\_timeout 60：连接成功后，与后端服务器两个成功的响应操作之间超时时间(代理接收超时)
* proxy\_buffer\_size 4k：设置代理服务器（nginx）从后端realserver读取并保存用户头信息的缓冲区大小，默认与proxy\_buffers大小相同，其实可以将这个指令值设的小一点
* proxy\_buffers 4 32k：proxy\_buffers缓冲区，nginx针对单个连接缓存来自后端realserver的响应，网页平均在32k以下的话，这样设置
* proxy\_busy\_buffers\_size 64k：高负荷下缓冲大小（proxy\_buffers\*2）
* proxy\_max\_temp\_file\_size：当 proxy\_buffers 放不下后端服务器的响应内容时，会将一部分保存到硬盘的临时文件中，这个值用来设置最大临时文件大小，默认1024M，它与 proxy\_cache 没有关系。大于这个值，将从upstream服务器传回。设置为0禁用。
* proxy\_temp\_file\_write\_size 64k：当缓存被代理的服务器响应到临时文件时，这个选项限制每次写临时文件的大小。proxy\_temp\_path（可以在编译的时候）：指定写到哪那个目录。
* proxy\_pass， proxy\_redirect见 location 部分。

http\_gzip模块部分：

* gzip on : 开启gzip压缩输出，减少网络传输。
* gzip\_min\_length 1k：设置允许压缩的页面最小字节数，页面字节数从header头得content-length中进行获取。默认值是20。建议设置成大于1k的字节数，小于1k可能会越压越大。
* gzip\_buffers 4 16k：设置系统获取几个单位的缓存用于存储gzip的压缩结果数据流。4 16k代表以16k为单位，安装原始数据大小以16k为单位的4倍申请内存。
* gzip\_http\_version 1.0：用于识别 http 协议的版本，早期的浏览器不支持 Gzip 压缩，用户就会看到乱码，所以为了支持前期版本加上了这个选项，如果你用了 Nginx 的反向代理并期望也启用 Gzip 压缩的话，由于末端通信是 http/1.0，故请设置为 1.0。
* gzip\_comp\_level 6：gzip压缩比，1压缩比最小处理速度最快，9压缩比最大但处理速度最慢(传输快但比较消耗cpu)
* gzip\_types：匹配mime类型进行压缩，无论是否指定,”text/html”类型总是会被压缩的。
* gzip\_proxied any：Nginx作为反向代理的时候启用，决定开启或者关闭后端服务器返回的结果是否压缩，匹配的前提是后端服务器必须要返回包含”Via”的 header头。
* gzip\_vary on：和http头有关系，会在响应头加个 Vary: Accept-Encoding ，可以让前端的缓存服务器缓存经过gzip压缩的页面，例如，用Squid缓存经过Nginx压缩的数据。

## 案例二：proxy\_set\_header 请求转发

admin.kkqb.cn.conf

server {

listen 80;

server\_name admin.kkqb.cn;

location / {

# deny 192.168.1.1; #拒绝此IP访问，可设置为网络段

# allow 222.240.39.48; #允许此IP访问，可设置为网络段

# allow 220.168.85.54; #允许此IP访问，可设置为网络段

# deny all; #拒绝所有，从上到下匹配

proxy\_set\_header Host $http\_host; #请求头重写

proxy\_set\_header X-Real-IP $remote\_addr; #可使web服务器端获得用户的真实ip,$remote\_add客户端地址

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for; #可使web服务器端获得用户的真实ip

proxy\_set\_header X-Forwarded-Proto $scheme; #$scheme变量为请求方案

proxy\_set\_header X-Scheme $scheme;

# if ($remote\_addr ~\* "220.168.85.54|222.240.39.48"){

# proxy\_pass http://172.16.148.28:8787;

# }

auth\_basic "Please input password";

auth\_basic\_user\_file /usr/local/nginx/conf/passwd;

proxy\_pass http://172.16.148.22:8787;

# root /usr/local/nginx/html;

# index index.html;

}

}

**proxy\_set\_header 解释：**

**proxy\_set\_header Host $http\_host;**

nginx为了实现反向代理的需求而增加了一个ngx\_http\_proxy\_module模块。其中proxy\_set\_header指令就是该模块需要读取的配置文件。在这里，所有设置的值的含义和http请求同中的含义完全相同，除了Host外还有X-Forward-For。  
 Host的含义是表明请求的主机名，因为nginx作为反向代理使用，而如果后端真是的服务器设置有类似防盗链或者根据http请求头中的host字段来进行路由或判断功能的话，如果反向代理层的nginx不重写请求头中的host字段，将会导致请求失败【默认反向代理服务器会向后端真实服务器发送请求，并且请求头中的host字段应为proxy\_pass指令设置的服务器】。

**proxy\_set\_header X-Real-IP $remote\_addr;** #可以使web服务器端获得用户的真实ip

“经过反向代理后，由于在客户端和web服务器之间增加了中间层，因此web服务器无法直接拿到客户端的ip，通过$remote\_addr变量拿到的将是反向代理服务器的ip地址”。当你使用了nginx反向服务器后，在web端使用request.getRemoteAddr()（本质上就是获取$remote\_addr），取得的是nginx的地址，即$remote\_addr变量中封装的是nginx的地址，当然是没法获得用户的真实ip的，但是，nginx是可以获得用户的真实ip的，也就是说nginx使用$remote\_addr变量时获得的是用户的真实ip，如果我们想要在web端获得用户的真实ip，就必须在nginx这里作一个赋值操作，如：proxy\_set\_header X-real-ip $remote\_addr;

其中这个X-real-ip是一个自定义的变量名，名字可以随意取，这样做完之后，用户的真实ip就被放在X-real-ip这个变量里了，然后，在web端可以这样获取：request.getHeader("X-real-ip") 这样就明白了吧。

**proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;** #可以使web服务器端获得用户的真实ip

有个X-Forwarded-For变量，这是一个squid开发的，用于识别通过HTTP代理或负载平衡器原始IP一个连接到Web服务器的客户机地址的非rfc标准，如果有做X-Forwarded-For设置的话,每次经过proxy转发都会有记录,格式就是client1, proxy1, proxy2,以逗号隔开各个地址，由于他是非rfc标准，所以默认是没有的，需要强制添加，在默认情况下经过proxy转发的请求，在后端看来远程地址都是proxy端的ip 。也就是说在默认情况下我们使用request.getHeader("X-Forwarded-For")获取不到用户的ip，如果我们想要通过这个变量获得用户的ip，我们需要自己在nginx添加如： proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

意思是增加一个$proxy\_add\_x\_forwarded\_for到X-Forwarded-For里去，注意是增加，而不是覆盖，当然由于默认的X-Forwarded-For值是空的，所以我们总感觉X-Forwarded-For的值就等于$proxy\_add\_x\_forwarded\_for的值，实际上当你搭建两台nginx在不同的ip上，并且都使用了这段配置，那你会发现在web服务器端通过request.getHeader("X-Forwarded-For")获得的将会是客户端ip和第一台nginx的ip。

**proxy\_set\_header X-Forwarded-Proto $scheme;**

**proxy\_set\_header X-Scheme $scheme;**

X-Forwarded-For:简称XFF头，它代表客户端，也就是HTTP的请求端真实的IP，只有在通过了HTTP 代理或者负载均衡服务器时才会添加该项。 它不是RFC中定义的标准请求头信息，在squid缓存代理服务器开发文档中可以找到该项的详细介绍。标准格式如下：X-Forwarded-For: client1, proxy1, proxy2。

这一HTTP头一般格式如下:X-Forwarded-For: client1, proxy1, proxy2

其中的值通过一个 逗号+空格 把多个IP地址区分开, 最左边(client1)是最原始客户端的IP地址, 代理服务器每成功收到一个请求，就把请求来源IP地址添加到右边。 在上面这个例子中，这个请求成功通过了三台代理服务器：proxy1, proxy2 及 proxy3。请求由client1发出，到达了proxy3(proxy3可能是请求的终点)。请求刚从client1中发出时，XFF是空的，请求被发往proxy1；通过proxy1的时候，client1被添加到XFF中，之后请求被发往proxy2;通过proxy2的时候，proxy1被添加到XFF中，之后请求被发往proxy3；通过proxy3时，proxy2被添加到XFF中，之后请求的的去向不明，如果proxy3不是请求终点，请求会被继续转发。

鉴于伪造这一字段非常容易，应该谨慎使用X-Forwarded-For字段。正常情况下XFF中最后一个IP地址是最后一个代理服务器的IP地址, 这通常是一个比较可靠的信息来源。

## 案例三：永久重定向

www.168cd.cn.conf

server {

listen 80;

server\_name www.168cd.cn 168cd.cn;

return 301 https://www.siyuanv.com$request\_uri;

location / {

root /opt/syqkl/html;

index index.html index.htm;

}

}

访问www.168cd.cn 168cd.cn 重定向到www.siyuanv.com $request\_uri:完整的原始请求URI（带参数）

其他案例：http 重定向到 https

server {

listen 80;

server\_name www.168cd.cn 168cd.cn;

return 301 https://$server\_name$request\_uri;

}

## 案例四：跨域add\_header

<https://blog.csdn.net/qq_38128179/article/details/84956552> 跨域概念参考

**一、为什么会出现跨域问题**

出于浏览器的同源策略限制。同源策略（Sameoriginpolicy）是一种约定，它是浏览器最核心也最基本的安全功能，如果缺少了同源策略，则浏览器的正常功能可能都会受到影响。可以说Web是构建在同源策略基础之上的，浏览器只是针对同源策略的一种实现。同源策略会阻止一个域的javascript脚本和另外一个域的内容进行交互。所谓同源（即指在同一个域）就是两个页面具有相同的协议（protocol），主机（host）和端口号（port），所谓同源是指"协议+域名+端口"三者相同，即便两个不同的域名指向同一个ip地址，也非同源。

1. **什么是跨域**

当一个请求url的协议、域名、端口三者之间任意一个与当前页面url不同即为跨域

当前页面url 被请求页面url 是否跨域 原因

http://www.test.com/ http://www.test.com/index.html 否 同源（协议、域名、端口号相同）

http://www.test.com/ https://www.test.com/index.html 跨域 协议不同（http/https）

http://www.test.com/ http://www.baidu.com/ 跨域 主域名不同（test/baidu）

http://www.test.com/ http://blog.test.com/ 跨域 子域名不同（www/blog）

http://www.test.com:8080/ http://www.test.com:7001/ 跨域 端口号不同（8080/7001）

**三、非同源限制**

【1】无法读取非同源网页的 Cookie、LocalStorage 和 IndexedDB

【2】无法接触非同源网页的 DOM 和 JS对象

【3】无法向非同源地址发送 AJAX 请求

**跨域解决方案**

1、 通过jsonp跨域

2、 document.domain + iframe跨域

3、 location.hash + iframe

4、 window.name + iframe跨域

5、 postMessage跨域

6、 跨域资源共享（CORS）：CORS是一个W3C标准，全称是跨域资源共享(Cross-origin resource sharing)。它允许 浏览器向跨源服务器，发出XMLHttpRequest请求，从而克服了AJAX只能同源使用的限制。

7、 nginx代理跨域

8、 nodejs中间件代理跨域

9、 WebSocket协议跨域

**nginx代理跨域**

1、 nginx配置解决iconfont跨域

浏览器跨域访问js、css、img等常规静态资源被同源策略许可，但iconfont字体文件(eot|otf|ttf|woff|svg)例外，此时可在nginx的静态资源服务器中加入以下配置。

location / {

add\_header Access-Control-Allow-Origin \*;

}

2、 nginx反向代理接口跨域

跨域原理： 同源策略是浏览器的安全策略，不是HTTP协议的一部分。服务器端调用HTTP接口只是使用HTTP协议，不会执行JS脚本，不需要同源策略，也就不存在跨越问题。

实现思路：通过nginx配置一个代理服务器（域名与domain1相同，端口不同）做跳板机，反向代理访问domain2接口，并且可以顺便修改cookie中domain信息，方便当前域cookie写入，实现跨域登录。

nginx具体配置：

#proxy服务器

server {

listen 81;

server\_name www.domain1.com;

location / {

proxy\_pass http://www.domain2.com:8080; #反向代理

proxy\_cookie\_domain www.domain2.com www.domain1.com; #修改cookie里域名

index index.html index.htm;

# 当用webpack-dev-server等中间件代理接口访问nignx时，此时无浏览器参与，故没有同源限制，下面的跨域配置可不启用

add\_header Access-Control-Allow-Origin http://www.domain1.com; #当前端只跨域不带cookie时，可为\*

add\_header Access-Control-Allow-Credentials true; #开启多域名配置跨域

}

}

将Access-Control-Allow-Origin添加到某网站下或整个域中的单个资源

Access-Control-Allow-Origin: http://example.com

Access-Control-Allow-Credentials: true (可选) #当该标志为真时，响应于该请求是否可以被暴露(可选)

将允许任何域向您提交请求

Access-Control-Allow-Origin: \*

Access-Control-Allow-Credentials: true (可选) #当该标志为真时，响应于该请求是否可以被暴露(可选)

$ curl -I -X OPTIONS -H "Origin: http://example.com" http://www.example2.com

Nginx配置实例

<https://www.cnblogs.com/sunmmi/articles/5956554.html> 参考资料

实例一：允许任何的应用在gyl.168cd.cn上跨域提取数据

gyl.168cd.cn.conf

server {

listen 80;

server\_name gyl.168cd.cn;

add\_header Access-Control-Allow-Origin \*; #许来自任何域的访问，可以这样配置

add\_header Access-Control-Allow-Headers 'Origin, X-Requested-With, Content-Type, Accept, Authorization, Token'; #允许脚本访问的返回头(可选)

add\_header Access-Control-Allow-Methods GET,POST,OPTIONS;

#指定请求的方法，可以是GET, POST, OPTIONS, PUT, DELETE等(可选)

location =/ {

proxy\_pass\_header Server;

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\_set\_header X-Forwarded-Proto $scheme;

proxy\_set\_header X-Scheme $scheme;

proxy\_pass http://172.16.148.34:8610/static/html/login.html;

}

location / {

proxy\_pass\_header Server;

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\_set\_header X-Forwarded-Proto $scheme;

proxy\_set\_header X-Scheme $scheme;

proxy\_pass http://172.16.148.34:8610;

}

}

**CORS(Cross Orign Resource-Sharing)跨域控制配置**

#是否允许请求带有验证信息

add\_header Access-Control-Allow-Credentials true;

#允许跨域访问的域名,可以是一个域的列表，也可以是通配符\*

add\_header Access-Control-Allow-Origin $allow\_url;

#允许脚本访问的返回头

add\_header Access-Control-Allow-Headers 'x-requested-with,content-type,Cache-Control,Pragma,Date,x-timestamp';

#允许使用的请求方法，以逗号隔开

add\_header Access-Control-Allow-Methods 'POST,GET,OPTIONS,PUT,DELETE';

#允许自定义的头部，以逗号隔开,大小写不敏感

add\_header Access-Control-Expose-Headers 'WWW-Authenticate,Server-Authorization';

#P3P支持跨域cookie操作

add\_header P3P 'policyref="/w3c/p3p.xml", CP="NOI DSP PSAa OUR BUS IND ONL UNI COM NAV INT LOC"';

如需要允许用户请求来自xxx1.example.com或xxx1.example1.com访问xxx.example2.com域名时，返回头Access-Control-Allow-Origin，在Nginx配置文件中xxx.example2.com域名的location /下配置以下内容

location / {

if ( $http\_origin ~ .\*.(example|example1).com ) {

add\_header Access-Control-Allow-Origin $http\_origin;

}

}

**Nginx跨域配置并支持DELETE,PUT请求**

默认Access-Control-Allow-Origin开启跨域请求只支持GET、HEAD、POST、OPTIONS请求，使用DELETE发起跨域请求时，浏览器出于安全考虑会先发起OPTIONS请求，服务器端接收到的请求方式就变成了OPTIONS，所以引起了服务器的405 Method Not Allowed。

解决方法

首先要对OPTIONS请求进行处理

if ($request\_method = 'OPTIONS') {

add\_header Access-Control-Allow-Origin \*;

add\_header Access-Control-Allow-Methods GET,POST,PUT,DELETE,OPTIONS;

#其他头部信息配置，省略...

return 204;

}

当请求方式为OPTIONS时设置Allow的响应头，重新处理这次请求。这样发出请求时第一次是OPTIONS请求，第二次才是DELETE请求。

# 完整配置参考：将配置文件的放到对应的server {}里

add\_header Access-Control-Allow-Origin \*;

location / {

if ($request\_method = 'OPTIONS') {

add\_header Access-Control-Allow-Origin \*;

add\_header Access-Control-Allow-Methods GET,POST,PUT,DELETE,OPTIONS;

return 204;

}

index index.php;

try\_files $uri @rewriteapp;

}

## 案例五：nginx 前后端分离和try\_files

Jsb项目测试环境 20190827笔记

server {

listen 80;

server\_name dkjtym.dakangyihu.com;

add\_header Access-Control-Allow-Origin \*;

add\_header Access-Control-Allow-Headers 'Origin, X-Requested-With, Content-Type, Accept, Token';

add\_header Access-Control-Allow-Methods GET,POST,OPTIONS;

location / {

proxy\_pass\_header Server; #请求头重写

#proxy\_set\_header Host $host:$server\_port;

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\_set\_header X-Forwarded-Proto $scheme;

proxy\_set\_header X-Scheme $scheme;

root jsbbd;

index index.html index.htm;

try\_files $uri $uri/ /index.html; #解决了首页登陆刷新login页面问题

#try\_files $uri $uri/ @req;

}

location /jsb {

proxy\_set\_header Host $http\_host;

#proxy\_set\_header Host $host:$server\_port;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forworded-For $http\_x\_forwarded\_for;

# proxy\_set\_header X-Forwarded-Proto $scheme;

proxy\_set\_header X-Scheme $scheme;

proxy\_pass http://192.168.10.243:8410;

}

项目情况：原jsb项目8410端口为rest ，8400端口为8400，公司决定实行前后端分离，增加一个静态页面放置在nginx目录的jsbbd目录中， dkjtym.dakangyihu.com域名临时调试使用，借宿宝开发目录结构第一级统一使用jsb目录，关于proxy\_pass\_header 请求转发相关配置请看本文上面详细介绍。

try\_files指令说明

在0.7以后的版本中加入了一个try\_files指令，配合命名location，可以部分替代原本常用的rewrite配置方式，提高解析效率。

try\_files指令

语法：try\_files file … uri 或 try\_files file … = code

默认值：无

作用域：server location

其作用是按顺序检查文件是否存在，返回第一个找到的文件或文件夹(结尾加斜线表示为文件夹)，如果所有的文件或文件夹都找不到，会进行一个内部重定向到最后一个参数。

需要注意的是，只有最后一个参数可以引起一个内部重定向，之前的参数只设置内部URI的指向。最后一个参数是回退URI且必须存在，否则会出现内部500错误。命名的location也可以使用在最后一个参数中。与rewrite指令不同，如果回退URI不是命名的location那么$args不会自动保留，如果你想保留$args，则必须明确声明。

## 案例六：SSL

server {

listen 80 ;

listen 443 ssl;

server\_name kkzg.youbasan.com;

add\_header Access-Control-Allow-Origin \*;

add\_header Access-Control-Allow-Headers 'Origin, X-Requested-With, Content-Type, Accept, Authorization, Token';

add\_header Access-Control-Allow-Methods GET,POST,OPTIONS;

ssl on;

ssl\_certificate ./sslkey/kkzg.youbasan.com.pem;

ssl\_certificate\_key ./sslkey/kkzg.youbasan.com.key;

ssl\_protocols TLSv1 TLSv1.1 TLSv1.2;

ssl\_ciphers ECDHE-RSA-AES256-SHA384:AES256-SHA256:RC4:HIGH:!MD5:!aNULL:!eNULL:!NULL:!DH:!EDH:!AESGCM;

ssl\_prefer\_server\_ciphers on;

ssl\_session\_cache shared:SSL:10m;

ssl\_session\_timeout 10m;

location / {

proxy\_pass\_header Server;

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\_set\_header X-Forwarded-Proto $scheme;

proxy\_set\_header X-Scheme $scheme;

proxy\_pass http://172.16.148.33:8910;

}

location = /uVn4zMewKb.txt {

proxy\_pass\_header Server;

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\_set\_header X-Forwarded-Proto $scheme;

proxy\_set\_header X-Scheme $scheme;

proxy\_pass http://172.16.148.33:8910/static/uVn4zMewKb.txt;

}

}

Ssl配置解释：

ssl on; #表示打开SSL支持

ssl\_certificate ./sslkey/kkzg.youbasan.com.pem; #指定秘钥文件

ssl\_certificate\_key ./sslkey/kkzg.youbasan.com.key; #指定公钥文件

ssl\_protocols TLSv1 TLSv1.1 TLSv1.2; #指定SSL协议

ssl\_ciphers ECDHE-RSA-AES256-SHA384:AES256-SHA256:RC4:HIGH:!MD5:!aNULL:!eNULL:!NULL:!DH:!EDH:!AESGCM;

#ssl\_ciphers配置ssl加密算法，多个算法用:分隔，ALL表示全部算法，!表示不启 用该算法，+表示将该算法排到最后面去。

ssl\_prefer\_server\_ciphers on; #如果不指定默认为off，当为on时，在使用SSLv3和TLS协议时，服务器加密算法 将优于客户端加密算法。

ssl\_session\_cache shared:SSL:10m;

ssl\_session\_timeout 10m; #缓存有效期

## 案例七：nginx 405 not allowed问题的解决

Apache、IIS、Nginx等绝大多数web服务器，都不允许静态文件响应POST请求，否则会返回“HTTP/1.1 405 Method not allowed”错误。但在有些应用中，需要使静态文件能够响应POST请求。

对于Nginx，可以修改nginc.conf配置文件，改变“405错误”为“200 ok”，并配置location来解决，方法如下：

解决办法，添加如下配置：

配置nginx将405错误指向成功

------------------

upstream static\_backend {

server localhost:80;

}

server {

listen 80;

# ...

error\_page 405 =200 @405;

location @405 {

root /srv/http;

proxy\_method GET;

proxy\_pass http://static\_backend;

}

}

------------------

实际shou-cloud项目的后端配置

upstream static\_backend {

server localhost:80;

}

server {

listen 80;

server\_name shopadmin.siyuanv.com;

add\_header Access-Control-Allow-Origin \*;

add\_header Access-Control-Allow-Headers 'Origin, X-Requested-With, Content-Type, Accept, Token';

add\_header Access-Control-Allow-Methods GET,POST,OPTIONS,PUT,DELETE;

location / {

proxy\_pass\_header Server;

#proxy\_set\_header Host $host:$server\_port;

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\_set\_header X-Forwarded-Proto $scheme;

proxy\_set\_header X-Scheme $scheme;

root shop-cloud/dist;

index index.html index.htm;

try\_files $uri $uri/ /index.html;

#try\_files $uri $uri/ @req;

}

error\_page 405 =200 @405;

location @405 {

root shop-cloud/dist;

proxy\_method GET;

proxy\_pass http://static\_backend;

}

#以下解决了登陆报token错误

location /siy-admin {

proxy\_set\_header Host $http\_host;

#proxy\_set\_header Host $host:$server\_port;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forworded-For $http\_x\_forwarded\_for;

# proxy\_set\_header X-Forwarded-Proto $scheme;

proxy\_set\_header X-Scheme $scheme;

proxy\_pass http://192.168.10.242:8888;

}

# SSL证书

SSL协议SSL（Secure Sockets Layer）中文名称安全套接字协议，是网络安全通信的重要基石。最初由网景公司（Netscape）设计，主要用来保证网络通信的机密性、认证性以及完整性，在网景公司它经历了SSL 1.0、SSL 2.0、SSL 3.0三个版本的变迁。后来由互联网工程任务组（IETF）接管，并更名为TLS（Transport Layer Security），发展至今已有TLS 1.0、TLS1.1、TLS1.2这几个版本。

互联网上存在着许多权威性的数字证书颁发机构，简称CA机构，他们会根据申请人信息，验证域名的所有权以及公司企业的完整信息，经过复杂的检验手段，再进行发布。

**HTTP为什么不安全：**

那么HTTP协议为什么是不安全的呢？原因是HTTP协议采用的是数据明文传输方式。用户从客户端浏览器提交数据，而数据是以明文的方式通过网络传输到到网站服务器。假如在这个网络传输过程中，有第三者侵入网络，截取到这个数据包，那么用户所传输的数据内容将会被截获者完全得到。假如用户传输的是银行卡、密码等个人私密信息，那对于用户来说是非常不利的。而且网站服务器也无法得知在数据传输过程中，用户数据是否会被篡改，这对网站服务器来说也是一种极大的风险。

解决这个问题的方法是使用HTTPS协议为你的网站加密，以前我们也有说过什么是HTTPS加密。为你的网站服务器安装服务器证书网站即可使用HTTPS协议进行访问。使用HTTPS协议的网站，在数据传输的过程中会对用户数据进行HTTPS加密（即SSL加密），经过SSL加密的数据再进行网络传输，那么即使是被第三者截获也能保证用户数据的安全和数据的完整性。所以HTTPS协议才会被认为是安全的、可靠的。

**SSL为什么能保证安全：**

SSL加密是建立在非对称加密算法的基础上的。非对称加密算法会产生一对长字符串，称为密钥对（公钥、私钥）。数据使用公钥进行加密后，唯一只能使用私钥才能解开。安装了服务器证书的网站，其实是把私钥保存在服务器中，而把公钥连同网站相关信息（如：域名、所有者名称、有效期）制作成一张SSL证书，并把SSL证书公布到互联网上。

当用户访问网站的时候边能得到这张SSL证书，当用户提交数据时，客户端使用保护的公钥的SSL证书对数据进行加密。由于非对称加密必须使用私钥才能解密，那么在网络传输时，即使数据被截获，由于截获者无法得到私钥，那么截获者也就无法破解密文。因此建立在SSL加密的HTTPS协议才会被认为是安全的，HTTPS网站才会被Chrome等主流浏览器认为是安全的网站。

**SSL连接作用：**

1、数据传输的机密性：利用对称密钥算法对传输的数据进行加密。

2、身份验证机制：现在证书基本利用电子签名，对服务器和客户端进行身份验证，值得说明的是客户端的身份验证是可以选的。

## SSL证书类型：

一、域名型证书：简称DV SSL，域名型SSL不会在证书中显示申请的公司单位的名称，只会显示申请的域名。对于DV SSL的申请，只需要1-2个小时左右即可完成对域名的验证并颁发证书，不需要递交纸质验证文件，只需要验证域名的管理权。

一般这种SSL类型适用于中小型企业网站、个人网站等。

二、企业型证书：简称OV SSL，企业型SSL包含了完整的公司信息，因此用户可以在证书中看到申请证书的公司名称。申请的时候不仅仅需要验证域名的管理权，还需要递交公司证明的材料以及纸质的验证文件。

这种SSL类型适用于企业网站、金融机构或是电子商务网站。

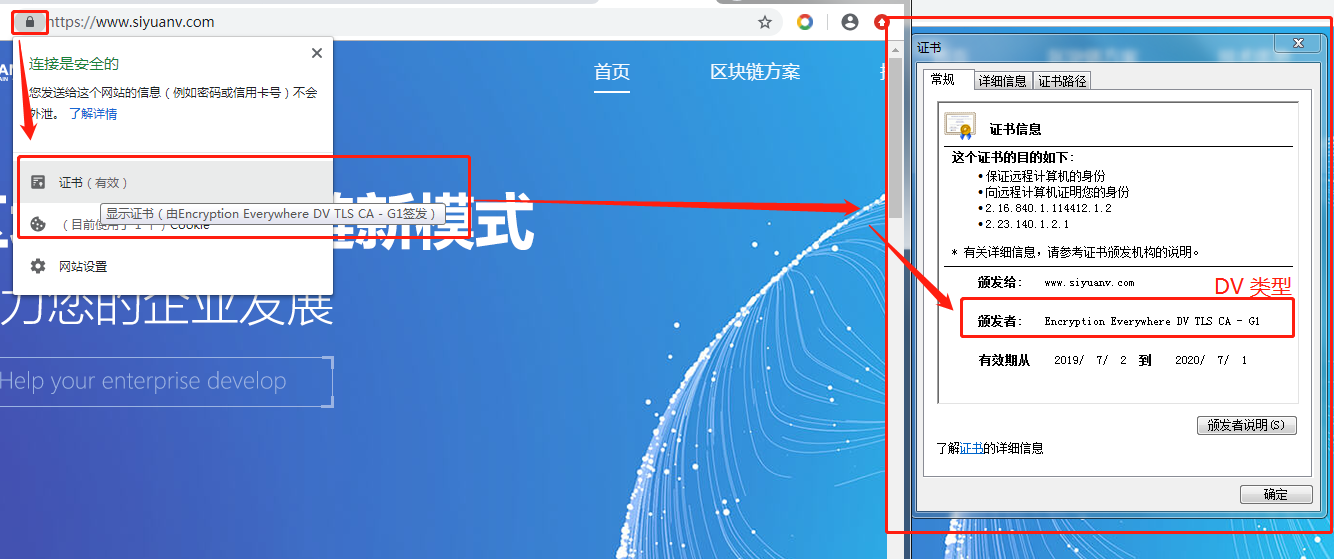
三、增强型证书：简称EV SSL，增强型SSL的申请需要通过极其严格的审查来验证网站企业身份和域名管理权，来确保网站身份的真实性。同时具有超强的加密保护，在地址栏中便显示了公司的名称和证书颁发机构的全称。与前两种SSL证书类型相比，EV SSL具有极高的可信度以及安全性。

这种SSL类型适用于大型的企业，证券、金融平台，政府机构的站点等。

四、代码签名证书：数字证书的其中一种，是提供给计算机软件开发者使用的，对开发的软件代码进行数字签名。签名后的软件，使用者在进行软件安装时，能够验证软件开发者的身份，并能保证软件代码不被篡改。从而保证了用户在软件安装使用的过程中，不会被病毒或恶意代码所侵害，保护了用户的利益的同时也帮助软件产品能在网络上快速发布。

使用代码签名证书可以对各种不同的软件代码文件进行数字签名，其中包括：.sys文件、ActiveX文件、.exe文件、.dll文件、.cab文件、.ocx文件、.xml文件等等。

对于不同企业要求有不同的SSL类型，因此用户不能认为部署了SSL证书的网站就是绝对安全的，他们仅仅对数据的传输进行了加密处理。还需要观察此网站部署了什么SSL证书类型，若只是部署了DV SSL，那么此公司是否真实存在也是需要判断的。因此在这里推荐电子商务网站都部署EV SSL证书 或者是OV SSL证书，绝对不能部署已经被欺诈网站滥用的DV SSL证书。





## 通配符SSL证书：

通配符SSL证书与普通SSL证书最大的不同就在于通配符证书能够保护主域名以及旗下不限数量的子域，这就意味着用户可通过申请单个通配符SSL证书来保护任意数量的子域。可以举个例子，假设用户为\*.example.com申请了通配符证书，那么此证书可以同时保护www.example.com、bolg.example.com等等站点，就不需要为另一个相关的网站申请多一份SSL证书。

尽管通配符证书对于拥有多个子域名的平台相当友好，但其实也有不方便的地方。首先，目前只有DV （域名型）SSL证书以及OV（企业型） SSL证书支持通配符证书的功能，而EV（增强型） SSL证书的用户不能享有通配符SSL证书的功能。另一方面，就是通配符证书不支持多域名，在SSL证书的管理期间，用户可以将通配符SSL域名添加为多域，但是却不能将多域添加为通配符SSL域。

虽然不能排除通配符SSL证书的这些缺点，但通配符SSL证书的出现也极大方便了同时拥有多个子域名的用户。同时，通配符证书也由于可以保护多个子域名，这使得它具有很高的兼容性，能够兼容99%的浏览器。

（一）通配符SSL证书的优点

如果你有多个子域要保护，那么购买通配符SSL证书是个非常好的选择：

1.保护无限子域需要为每个子域购买单个SSL证书，购买并使用一个证书就可以了。

2.便宜实惠。虽然通配符证书比一般的证书成本高很多，但如果使用在多个子域中，就会轻松节省几百甚至上万元。

3.更易于管理。即使使用PKI管理界面，部署30个不同的单独SSL证书也是一项艰巨的任务，更新时更麻烦。但是通配符SSL可以解决上述这些麻烦复杂的问题，而且操作简单方便。

（二）通配符SSL证书的缺点

1.安全性。如果在多个服务器上使用一个证书和私钥，那么当一个服务器受到攻击，其他所有的服务器也会受到攻击。例如，pay.baidu.com子域名遭到破坏，则mail和cloud子域名也会遭到破坏。这样就必须尽快重新替换所有服务器上的证书。

2.移动设备兼容性。某些流行的移动设备操作系统（5）无法识别通配符（\*），因此无法使用通配符证书。如果您需要支持这些设备，则需要使用单个证书。

## SSL证书文件有哪些

SSL是我们广大网友中最为常用的网络系统加密方式，它是由Secure Socket Layer缩写而来。SSL加密认证是包括了使用一个公用密钥和一个私人密钥，我们用户都是使用公用密钥来加密数据的，解密信息就需要使用相对应的私人密钥。这都是相对应的，不仅是信息保密的需要。更是为了最大程度上保护互联网消费者的消费信息，个人信息，保密信息等等。

SSL证书文件一般是在信息终端设置配置的，通常主要有.key、.csr、.crt、.pem等几个不同的后缀，一般专业技术人员一眼就能判断出来，后缀一般是互联网系统用于判断ssl证书文件类型所使用的，Linux是不使用后缀名来判断ssl证书文件名称的，但是由于现在的互联网约定俗成，我们一般都会添加这些后缀。

以下为不同后缀的ssl证书文件的作用:

1、.key文件：属于密钥文件，SSL证书的私人密钥就包括在内。是信息内容的中枢；

2、.csr文件：文件里包括证书的公用密钥和一些公司内部重要秘密信息，要使用者通过请求签名之后才能够直接生出证书；

3、.crt文件：此文件也包含了ssl证书的公用密钥、签字讯息以及根据不同的类型伴随不同认证的信息，通常各类签名签字证书都会在这类文件中，如IP等；

1. .pem文件：该文件较其他后缀文件来说比较少见，里面包含着证书的私人密钥以及其他一部分证书重要的信息。

## SSL证书和ca证书的区别

SSL证书和ca证书是从属关系，因为ssl是众多ca证书中的一种。CA是数字证书管理机构，SSL证书一种数字证书，SSL证书由CA机构签发的。不过CA机构签发的证书有很多种，都叫CA证书，所以SSL证书只是其中一种。

SSL证书就是传统意义的超文本传输协议的安全版本，在原有的基础上添加了https安全协议。浏览器和服务器之间的信息传输通道更加安全，能够保证传输的数据具有稳定性和完整性。而SSL证书是由专业的数字证书签发机构所签发的安全证书。

## 证书标准

X.509 - 这是一种证书标准,主要定义了证书中应该包含哪些内容.其详情可以参考RFC5280,SSL使用的就是这种证书标准.

## SSL中常用证书格式及其应用 |

### 一、各种格式证书介绍：

PKCS 全称是 Public-Key Cryptography Standards ，是由 RSA 实验室与其它安全系统开发商为促进公钥密码的发展而制订的一系列标准，PKCS 目前共发布过 15 个标准。 常用的有：

PKCS#12 Personal Information Exchange: .pfx, .p12

PKCS#10 Certification Request: .p10

PKCS#7 cert request response: .p7r

PKCS#7 binary message: .p7b

PKCS#7 常用的后缀是： .P7B .P7C .SPC

PKCS#12 常用的后缀有： .P12 .PFX

X.509是常见通用的证书格式。所有的证书都符合为Public Key Infrastructure (PKI) 制定的 ITU-T X509 国际标准。

X.509 DER 编码(ASCII)的后缀是： .DER .CER .CRT

X.509 PAM 编码(Base64)的后缀是： .PEM

cer/.crt是用于存放证书，它是2进制形式存放的，不含私钥。

.pem跟crt/cer的区别是它以Ascii来表示。

pfx/p12用于存放个人证书/私钥，他通常包含保护密码，2进制方式

p10是证书请求

p7r是CA对证书请求的回复，只用于导入

p7b以树状展示证书链(certificate chain)，同时也支持单个证书，不含私钥。

注意：其中若要使用浏览器进行双向认证的话，必须在浏览器安装p12格式的证书 ，因为p12格式的数字证书包含私钥，只有包含私钥 才能使双方的加密通信正常进行。

### 二、各种格式证书转换与导入

1、Der/Cer证书导入：

要从某个文件中导入某个证书，使用keytool工具的-import命令：

keytool -import -file mycert.der -keystore mykeystore.jks

如果在 -keystore 选项中指定了一个并不存在的密钥仓库，则该密钥仓库将被创建。

如果不指定 -keystore 选项，则缺省密钥仓库将是宿主目录中名为 .keystore 的文件。如果该文件并不存在，则它将被创建。

创建密钥仓库时会要求输入访问口令，以后需要使用此口令来访问。可使用-list命令来查看密钥仓库里的内容：

keytool -list -rfc -keystore mykeystore.jks

2、P12格式证书导入：

keytool无法直接导入PKCS12文件。

第一种方法是使用IE将pfx证书导入，再导出为cert格式文件。使用上面介绍的方法将其导入到密钥仓库中。这样的话仓库里面只包含了证书信息，没有私钥内容。

第二种方法是将pfx文件导入到IE浏览器中，再导出为pfx文件。

新生成的pfx不能被导入到keystore中，报错：keytool错误： java.lang.Exception: 所输入的不是一个 X.509 认证。新生成的pfx文件可以被当作keystore使用。但会报个错误as unknown attr1.3.6.1.4.1.311.17.1,查了下资料,说IE导出的就会这样,使用Netscape就不会有这个错误.

第三种方法是将pfx文件当作一个keystore使用。但是通过微软的证书管理控制台生成的pfx文件不能直接使用。keytool不认此格式，报keytool错误： java.io.IOException: failed to decrypt safe contents entry。需要通过OpenSSL转换一下：

1）openssl pkcs12 -in mycerts.pfx -out mycerts.pem

2）openssl pkcs12 -export -in mycerts.pem -out mykeystore.p12

通过keytool的-list命令可检查下密钥仓库中的内容：

keytool -rfc -list -keystore mykeystore.p12 -storetype pkcs12

这里需要指明仓库类型为pkcs12，因为缺省的类型为jks。这样此密钥仓库就即包含证书信息也包含私钥信息。

3、P7B格式证书导入：

keytool无法直接导入p7b文件。

需要将证书链RootServer.p7b（包含根证书）导出为根rootca.cer和子rootcaserver.cer 。

将这两个证书导入到可信任的密钥仓库中。

keytool -import -alias rootca -trustcacerts -file rootca.cer -keystore testkeytrust.jks

遇到是否信任该证书提示时，输入y

keytool -import -alias rootcaserver -trustcacerts -file rootcaserver.cer -keystore testkeytrust.jks

总结:

1)P12格式的证书是不能使用keytool工具导入到keystore中的

2)The Sun's PKCS12 Keystore对从IE和其他的windows程序生成的pfx格式的证书支持不太好.

3)P7B证书链不能直接导入到keystore，需要将里面的证书导出成cer格式，再分别导入到keystore。

三、各种证书相互转换

一 用openssl创建CA证书的RSA密钥(PEM格式)：

openssl genrsa -des3 -out ca.key 1024

二用openssl创建CA证书(PEM格式,假如有效期为一年)：

openssl req -new -x509 -days 365 -key ca.key -out ca.crt -config openssl.cnf

openssl是可以生成DER格式的CA证书的，最好用IE将PEM格式的CA证书转换成DER格式的CA证书。

三 x509到pfx

pkcs12 -export –in keys/client1.crt -inkey keys/client1.key -out keys/client1.pfx

四 PEM格式的ca.key转换为Microsoft可以识别的pvk格式。

pvk -in ca.key -out ca.pvk -nocrypt -topvk

五 PKCS#12 到 PEM 的转换

openssl pkcs12 -nocerts -nodes -in cert.p12 -out private.pem

验证 openssl pkcs12 -clcerts -nokeys -in cert.p12 -out cert.pem

六 从 PFX 格式文件中提取私钥格式文件 (.key)

openssl pkcs12 -in mycert.pfx -nocerts -nodes -out mycert.key

七 转换 pem 到到 spc

openssl crl2pkcs7 -nocrl -certfile venus.pem -outform DER -out venus.spc

用 -outform -inform 指定 DER 还是 PAM 格式。例如：

openssl x509 -in Cert.pem -inform PEM -out cert.der -outform DER

八 PEM 到 PKCS#12 的转换，

openssl pkcs12 -export -in Cert.pem -out Cert.p12 -inkey key.pem

九 PER到CER

openssl x509 -in mycerts.pem -out mycerts.cer

# nginx内置变量汇总：

嵌入式变量

该 ngx\_http\_core\_module 模块支持名称与Apache Server变量匹配的嵌入变量。首先，这些是代表客户端请求的报头字段的变量，例如 $http\_user\_agent， $http\_cookie等。还有其他变量：

$arg\_name name 请求行中的参数

$args 请求行中的参数，同$query\_string

$binary\_remote\_addr 客户端地址采用二进制形式，IPv4地址的值长度始终为4个字节，IPv6地址的长度为16个字节

$body\_bytes\_sent 发送到客户端的字节数，不包括响应头; 此变量与 Apache模块的“ %B”参数 兼容mod\_log\_config

$bytes\_sent 发送到客户端的字节数（1.3.8,1.2.5）

$connection 连接序列号（1.3.8,1.2.5）

$connection\_requests 当前通过连接发出的请求数（1.3.8,1.2.5）

$content\_length “Content-Length”请求标头字段

$content\_type “Content-Type”请求标头字段

$cookie\_name 该 name 饼干

$document\_root root 或 alias 指令当前请求的值

$document\_uri 与...一样 $uri， 与$uri相同。

$host 请求主机头字段，否则为服务器名称，按此优先顺序：第1是来自请求行的主机名，第2是 来自“主机”请求标头字段的主机名，第3是与请求匹配的服务器名称

$hostname 主机名

$http\_name 任意请求头字段; 变量名称的最后一部分是字段名称转换为小写，短划线由下划线替换

$https “ on”如果连接在SSL模式下运行，或者否则为空字符串

$http\_user\_agent 客户端agent信息

$http\_cookie 客户端cookie信息

$is\_args “ ?”如果请求行有参数，否则为空字符串

$limit\_rate 设置此变量可启用响应速率限制; 见 limit\_rate

$msec 以毫秒为单位的当前时间（毫秒）（1.3.9,1.2.6）

$nginx\_version nginx版本

$pid 工人流程的PID

$pipe “ p”如果请求是流水线的，“ .”否则（1.3.12,1.2.7）

$proxy\_protocol\_addr 来自PROXY协议头的客户端地址，否则为空字符串（1.5.12）

必须先通过proxy\_protocol 在listen 指令中设置参数 来启用PROXY协议 。

$proxy\_protocol\_port 来自PROXY协议头的客户端端口，否则为空字符串（1.11.0）

必须先通过proxy\_protocol 在listen 指令中设置参数 来启用PROXY协议 。

$query\_string 与...一样 $args

$realpath\_root 与当前请求的根 或 别名指令值对应的绝对路径名 ，所有符号链接都解析为实际路径

$remote\_addr 客户地址

$remote\_port 客户端口

$remote\_user 基本身份验证提供的用户名，已经经过Auth Basic Module验证的用户名。

$request 完整的原始请求行

$request\_body 请求机构当请求体被读取到内存缓冲区时，变量的值在proxy\_pass， fastcgi\_pass，uwsgi\_pass 和 scgi\_pass 指令 处理的位置可用 。

$request\_body\_file 请求正文的临时文件的名称,在处理结束时，需要删除该文件。要始终将请求主体写入文件， 需要启用client\_body\_in\_file\_only。当在代理请求或对FastCGI / uwsgi / SCGI服务器的请求中传递 临时文件的名称时，应分别通过proxy\_pass\_request\_body off， fastcgi\_pass\_request\_body off， uwsgi\_pass\_request\_body off或 scgi\_pass\_request\_body off 指令禁用传递请求体。

$request\_completion “ OK”如果请求已完成，或者否则为空字符串

$request\_filename 基于根 或 别名 指令以及请求URI 的当前请求的文件路径，当前请求的文件路径，由 root或alias指令与URI请求生成。

$request\_id 16个随机字节生成的唯一请求标识符，十六进制（1.11.0）

$request\_length 请求长度（包括请求行，标题和请求体）（1.3.12,1.2.7）

$request\_method 请求方法，通常是“ GET”或“ POST”

$request\_time 请求处理时间，以秒为单位，分辨率为毫秒（1.3.9,1.2.6）; 从客户端读取第一个字节后经过的时间

$request\_uri 完整的原始请求URI（带参数），不包含主机名，如：”/foo/bar.php?arg=baz”。

$scheme 请求方案，“ http”或“ https”

$sent\_http\_name 任意响应头字段; 变量名称的最后一部分是字段名称转换为小写，短划线由下划线替换

$sent\_trailer\_name 在响应结束时发送的任意字段（1.13.2）; 变量名称的最后一部分是字段名称转换为小写， 短划线由下划线替换

$server\_addr 接受请求的服务器的地址,,计算此变量的值通常需要一次系统调用。为避免系统调用， listen指令必须指定地址并使用该 bind 参数。

$server\_name 接受请求的服务器的名称

$server\_port 接受请求的服务器的端口

$server\_protocol 请求协议，通常是“ HTTP/1.0”，“ HTTP/1.1”或“ HTTP / 2.0 ”

$status 回应状态（1.3.2,1.2.2）

$tcpinfo\_rtt, $tcpinfo\_rttvar, $tcpinfo\_snd\_cwnd, $tcpinfo\_rcv\_space 有关客户端TCP连接的信息; 在支持TCP\_INFO 套接字选项的系统上可用

$time\_iso8601 当地时间采用ISO 8601标准格式（1.3.12,1.2.7）

$time\_local 通用日志格式的本地时间（1.3.12,1.2.7）

$uri 请求中的当前URI规范化$uri 在请求处理期间，值可能会发生变化例如在执行 内部重定向或使用索引文件时。不带请求参数的当前URI，$uri不包含主机名，如”/foo/bar.html”。

# 网页报错汇总：

# NSD Operation DAY04

案例1：构建memcached服务

案例2：LNMP+memcached

案例3：PHP的本地Session信息

案例4：PHP实现session共享

本节从案例3开始将,出现关于memcached的问题再讲案例1

# 1 案例1：构建memcached服务

SQL:oracle,mysql,sql\_server 关系型数据库:数据之间有极强的相关联性

NoSQL:memcache,redis,mongodb 非关系型数据库,又叫kv数据库,或者叫缓存数据库\内存数据库(重启或者断电,所有数据将丢失)

Redis :将数据存到内存,会自动同步到硬盘

Memcached:将数据存到内存.数据会丢失,存毫无关系的一些数据,可用作上传中转的网站

Session是记录登录状态,会被定时删除,丢了无所谓,客户重新登录会再产生,因此可用memcahced数据库.

读取速度:CPU缓存>内存>磁盘>数据库

Cpu参数L1:一级缓存 L2:二级缓存,数字越高,性能越好

1.1 问题

本案例要求先快速搭建好一台memcached服务器，并对memcached进行简单的增、删、改、查操作：

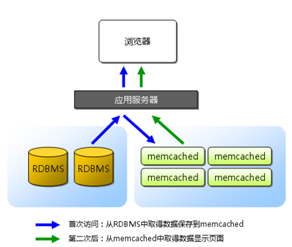
安装memcached软件，并启动服务

使用telnet测试memcached服务

对memcached进行增、删、改、查等操作

1.2 方案

memcached是高性能的分布式缓存服务器，用来集中缓存数据库查询结果，减少数据库访问次数，以提高动态Web应用的响应速度。访问拓扑如图-1所示。



使用1台RHEL7虚拟机作为memcached服务器（192.168.4.5）。

在RHEL7系统光盘中包含有memcached，因此需要提前配置yum源，即可直接使用yum安装，客户端测试时需要提前安装telnet远程工具。

验证时需要客户端主机安装telnet，远程memcached来验证服务器的功能：

* add name 0 180 10 //变量不存在则添加
* set name 0 180 10 //添加或替换变量
* replace name 0 180 10 //替换
* get name //读取变量
* append name 0 180 10 //向变量中追加数据
* delete name //删除变量
* stats //查看状态
* flush\_all //清空所有
* 提示：0表示不压缩，180为数据缓存时间，10为需要存储的数据字节数量。

## 步骤一：构建memcached服务

### 1) 使用yum安装软件包memcached

Systemctl start aaa.service将会到 /usr/lib/systemd/system 目录里面寻找aaa.service 运行这个文本

例如:Systemctl start memcached

### memcached配置文件（查看即可，不需要修改）

本次实验无关,作为了解

[root@proxy ~]# vim /usr/lib/systemd/system/memcached.service

[Unit]

Description=Memcached

Before=httpd.service

After=network.target

[Service]

Type=simple

EnvironmentFile=-/etc/sysconfig/memcached

ExecStart=/usr/bin/memcached -u $USER -p $PORT -m $CACHESIZE -c $MAXCONN $OPTIONS

[Install]

WantedBy=multi-user.target

~

[root@proxy ~]# vim /etc/sysconfig/memcached #查看环境变量

PORT="11211"

USER="memcached"

MAXCONN="1024"

CACHESIZE="64"

OPTIONS=""

~

### 3）启动服务并查看网络连接状态验证是否开启成功：

[root@proxy ~]# ss -nutlp | grep 11211 #查看memcached端口是否起来

[root@proxy ~]# ss -nutlp | grep memcached #查看服务是否起来

[root@proxy ~]# netstat -antulp | grep memcached #查看服务是否起来

tcp 0 0 0.0.0.0:11211 0.0.0.0:\* LISTEN 2839/memcached

tcp 0 0 :::11211 :::\* LISTEN 2839/memcached

udp 0 0 0.0.0.0:11211 0.0.0.0:\* 2839/memcached

udp 0 0 :::11211 :::\* 2839/memcached

[root@proxy ~]# setenforce 0

[root@proxy ~]# firewall-cmd --set-default-zone=trusted

### ４）netstar 命令与ss命令

netstat命令可以查看系统中启动的端口信息，该命令常用选项如下：

-a显示所有端口的信息

-n以数字格式显示端口号

-t显示TCP连接的端口

-u显示UDP连接的端口

-l显示服务正在监听的端口信息，如httpd启动后，会一直监听80端口

-p显示监听端口的服务名称是什么（也就是程序名称）

注意：在RHEL7系统中，使用ss命令可以替代netstat，功能与选项一样。

## 步骤二：使用telnet访问memcached服务器

### 1）使用yum安装telnet

[root@proxy ~]# yum -y install telnet

### ２) 使用telnet连接服务器测试memcached增、删、改、查等操作。

[root@proxy ~]# telnet 192.168.4.5 11211　　#11211端口为memcached

Trying 192.168.4.5...

Connected to 192.168.4.5.

Escape character is '^]'.

ERROR #无命令报错

set name 0 180 3 #定义变量:name 0:不压缩 180s:数据缓存时间 3:数据字节数量

Pil #变量值为:pil

STORED #执行成功

get name #获取name 变量

VALUE name 0 3

pil #输出的结果

END

get name #180s 后再获取变量

END #无输出,结束

##提示：0表示不压缩，180s为数据缓存时间，10为需要存储的数据字节数量。

add myname 0 180 10 //新建，myname不存在则添加，存在则报错

set myname 0 180 10 //添加或替换变量

replace myname 0 180 10 //替换，如果myname不存在则报错

get myname //读取变量

append myname 0 180 10 //向变量中追加数据

delete myname //删除变量

stats //查看状态

flush\_all //清空所有

quit //退出登录

# 2 案例2：LNMP+memcached

2.1 问题

沿用练习一，部署LNMP+memcached网站平台,通过PHP页面实现对memcached服务器的数据操作，实现以下目标：

部署LNMP实现PHP动态网站架构

为PHP安装memcache扩展

创建PHP页面，并编写PHP代码，实现对memcached的数据操作

2.2 方案

使用2台RHEL7虚拟机，其中一台作为memcached及LNMP服务器（192.168.4.5）、另外一台作为测试用的Linux客户机（192.168.4.10），如图-1所示。

在RHEL7系统光盘中包含有我们需要的MariaDB、PHP，我们需要使用源码安装Nginx，使用RPM包安装FPM。另外如果希望使用PHP来操作memcached，注意必须要为PHP安装memcache扩展（php-pecl-memcache），否则PHP无法解析连接memcached的指令。客户端测试时需要提前安装telnet远程工具。

## 步骤一：部署LNMP环境

（如果环境中已经存在LNMP环境本步骤可以忽略）

### 1）使用yum安装基础依赖包

[root@web1 ~]# yum -y install gcc openssl-devel pcre-devel zlib-devel

.. ..

## 2）源码安装Nginx

[root@web1 ~]# tar -xf nginx-1.12.2.tar.gz

[root@web1 ~]# cd nginx-1.12.2

[root@web1 nginx-1.12.2]# ./configure \

> --with-http\_ssl\_module

[root@web1 nginx-1.12.2]# make && make install

### 3）安装MariaDB数据库

[root@web1 ~]# yum -y install mariadb mariadb-server mariadb-devel

### 4）安装PHP

[root@web1 ~]# yum -y install php php-mysql

[root@web1 ~]# yum -y install php-fpm-5.4.16-42.el7.x86\_64.rpm

#php-fpm-5.4.16-42.el7.x86\_64.rpm,光盘中没有在提供的lnmp\_soft.tar.gz包内

### 5）修改Nginx配置文件

[root@web1 ~]# vim /usr/local/nginx/conf/nginx.conf

location / {

root html;

index index.php index.html index.htm;

}

location ~ \.php$ {

root html;

fastcgi\_pass 127.0.0.1:9000;

fastcgi\_index index.php;

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

include fastcgi.conf;

}

## 步骤二：启动服务

（如果所有服务已经启动，也可以忽略这一步骤）

### 1）启动Nginx服务

这里需要注意的是，如果服务器上已经启动了其他监听80端口的服务软件（如httpd），则需要先关闭该服务，否则会出现冲突。

[root@web1 ~]# systemctl stop httpd //如果该服务存在，则关闭该服务

[root@web1 ~]# /usr/local/nginx/sbin/nginx

[root@web1 ~]# netstat -utnlp | grep :80

tcp 0 0 0.0.0.0:80 0.0.0.0:\* LISTEN 32428/nginx

### 2）启动MySQL服务

[root@web1 ~]# systemctl start mariadb

[root@web1 ~]# systemctl status mariadb

### 3）启动PHP-FPM服务

[root@web1 ~]# systemctl start php-fpm

[root@web1 ~]# systemctl status php-fpm

### 4）关闭SELinux、防火墙

[root@web1 ~]# setenforce 0

[root@web1 ~]# firewall-cmd --set-default-zone=trusted

## 步骤三：创建PHP页面，使用PHP语言测试memcached服务

### 1）部署测试页面

创建PHP首页文档/usr/local/nginx/html/index.php，测试页面可以参考lnmp\_soft/php\_scripts/mem.php。

注意：192.168.2.5是memcached数据库。

[root@web1 ~]# vim /usr/local/nginx/html/test.php

<?php

$memcache=new Memcache; //创建memcache对象

$memcache->connect('192.168.2.5',11211) or die ('could not connect!!');

$memcache->set('key','test'); //定义变量

$get\_values=$memcache->get('key'); //获取变量值

echo $get\_values;

?>

### 2）客户端测试（结果会失败）

客户端使用浏览器访问服务器PHP首页文档，检验对memcached的操作是否成功：

[root@web1 ~]# firefox http://192.168.2.100/test.php

注意：这里因为没有给PHP安装扩展包，默认PHP无法连接memcached数据库，需要给PHP安装扩展模块才可以连接memcached数据库。

### 3）为PHP添加安装memcache扩展

[root@web1 ~]# yum -y install php-pecl-memcache

[root@web1 ~]# systemctl restart php-fpm

### 4）客户端再次测试（结果会成功显示数据结果）

[root@web1 ~]# firefox http://192.168.2.100/test.php

# Session与Cookie:

## 防止重复登录

例如访问京东:

客户端:输入账户密码---->

服务器:数据库中验证,通过允许登录并创建一个session文本id.txt,id号随机

服务器:返回登录后的页面,并且在数据包包头返回session文本的id号,

客户端:浏览器记住session文本的id为cookie,

客户端:下次访问,发送cookie可不用验证

如果web集群,采用轮询访问,使用session共享

# 3 案例3：PHP的本地Session信息

Session:在计算机中，尤其是在网络应用中，称为“会话控制”。Session 对象存储特定用户会话所需的属性及配置信息。这样，当用户在应用程序的 Web 页之间跳转时，存储在 Session 对象中的变量将不会丢失，而是在整个用户会话中一直存在下去。当用户请求来自应用程序的 Web 页时，如果该用户还没有会话，则 Web 服务器将自动创建一个 Session 对象。当会话过期或被放弃后，服务器将终止该会话。Session 对象最常见的一个用法就是存储用户的首选项。例如，如果用户指明不喜欢查看图形，就可以将该信息存储在 Session 对象中。

## session与cookie

首先，我大致的知道，session是一次浏览器和服务器的交互的会话，会话是啥呢？就是我问候你好吗？你回恩很好。就是一次会话，那么对话完成后，这次会话就结束了，还有我也知道，我们可以将一个变量存入全部的$\_SESSION['name']中，这样php的各个页面和逻辑都能访问到，所以很轻松的用来判断是否登陆。

这是我之前理解的session，当然也是对的，只是解释的太肤浅，理解的太表面了，面试官如果听到这样的答案其实是不太满意的。我参考了其他的很多资料，彻底理解清楚session。

在说session是啥之前，我们先来说说为什么会出现session会话，它出现的机理是什么？我们知道，我们用浏览器打开一个网页，用到的是HTTP协议，学过计算机的应该都知道这个协议，它是无状态的，什么是无状态呢？就是说这一次请求和上一次请求是没有任何关系的，互不认识的，没有关联的。但是这种无状态的的好处是快速。

所以就会带来一个问题就是，我希望几个请求的页面要有关联，比如：我在www.a.com/login.php里面登陆了，我在www.a.com/index.php 也希望是登陆状态，但是，这是2个不同的页面，也就是2个不同的HTTP请求，这2个HTTP请求是无状态的，也就是无关联的，所以无法单纯的在index.php中读取到它在login.php中已经登陆了！

那咋搞呢？我不可能这2个页面我都去登陆一遍吧。或者用笨方法这2个页面都去查询数据库，如果有登陆状态，就判断是登陆的了。这种查询数据库的方案虽然可行，但是每次都要去查询数据库不是个事，会造成数据库的压力。

所以正是这种诉求，这个时候，一个新的客户端存储数据方式出现了：cookie。cookie是把少量的信息存储在用户自己的电脑上，它在一个域名下是一个全局的，只要设置它的存储路径在域名www.a.com下 ，那么当用户用浏览器访问时，php就可以从这个域名的任意页面读取cookie中的信息。所以就很好的解决了我在www.a.com/login.php页面登陆了，我也可以在www.a.com/index.php获取到这个登陆信息了。同时又不用反复去查询数据库。

虽然这种方案很不错，也很快速方便，但是由于cookie 是存在用户端，而且它本身存储的尺寸大小也有限，最关键是用户可以是可见的，并可以随意的修改，很不安全。那如何又要安全，又可以方便的全局读取信息呢？于是，这个时候，一种新的存储会话机制：session 诞生了。

我擦，终于把session是怎么诞生的给圆清楚了，不容易啊！！！

好，session 诞生了，从上面的描述来讲，它就是在一次会话中解决2次HTTP的请求的关联，让它们产生联系，让2两个页面都能读取到找个这个全局的session信息。session信息存在于服务器端，所以也就很好的解决了安全问题。

3.1 问题

通过Nginx调度器负载后端两台Web服务器，实现以下目标：

部署Nginx为前台调度服务器

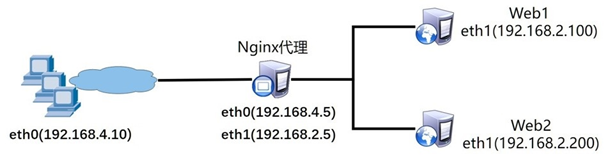
调度算法设置为轮询

后端为两台LNMP服务器

部署测试页面，查看PHP本地的Session信息

3.2 方案

使用4台RHEL7虚拟机，其中一台作为Nginx前端调度器服务器（eth0:192.168.4.5,eth1:192.168.2.5）、两台虚拟机部署为LNMP服务器，分别为Web1服务器（192.168.2.100）和Web2服务器（192.168.2.200），另外一台作为测试用的Linux客户机（192.168.4.10），拓扑如图-2所示。



## 步骤一：部署后端LNMP服务器相关软件

**Web1:192.168.2.100 Web 2:192.168.2.200安装LNMP:**

**注意:以下部署LNMP服务器的操作，需要在两台后端服务器做相同的操作，下面我们以一台Web2服务器（192.168.2.200）为例，对Web1服务器执行相同操作即可。**

### 1）使用yum安装基础依赖包

[root@web2 ~]# yum -y install gcc openssl-devel pcre-devel

### 2）源码安装Nginx

[root@web2 ~]# tar -xf nginx-1.12.2.tar.gz

[root@web2 ~]# cd nginx-1.12.2

[root@web2 nginx-1.12.2]# ./configure --prefix=/usr/local/nginx --user=nginx --group=nginx --with-http\_ssl\_module --with-stream --with-http\_stub\_status\_module

[root@web2 nginx-1.12.2]# make && make install #编译与安装

### 3）安装MariaDB数据库

[root@web2 ~]# yum -y install mariadb mariadb-server mariadb-devel

### 安装PHP

（php-fpm软件包在lnmp\_soft中有提供）

[root@web2 ~]# yum -y install php php-mysql

[root@web2 ~]# cd lnmp\_soft

[root@web2 lnmp\_soft]# yum -y install php-fpm-5.4.16-42.el7.x86\_64.rpm

#非光盘软件包,需要进入有安装包的目录里面安装

### 5）配置Nginx默认首页与动静分离

[root@web2 ~]# vim /usr/local/nginx/conf/nginx.conf

location / {

root html;

index index.php index.html index.htm;

}

#PHP动态网页转至如下 默认网页文件都是index.html,添加index.php为默认

location ~ \.php$ {

root html;

fastcgi\_pass 127.0.0.1:9000;

fastcgi\_index index.php;

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

include fastcgi.conf; #记得修改

}

# :set nu 开启行号,然后替换掉所有指定行的#(起始行,结束行s/#//)

## 步骤二：启动LNMP服务器相关的服务

### 1）启动Nginx服务

这里需要注意的是，如果服务器上已经启动了其他监听80端口的服务软件（如httpd），则需要先关闭该服务，否则会出现冲突。

[root@web2 ~]# systemctl stop httpd //如果该服务存在，则关闭该服务

[root@Web1111 ]# useradd -s /sbin/nologin nginx #创建用户

[root@Web1111 ]# /usr/local/nginx/sbin/nginx #启动nginx

[root@Web1111 ]# ln -s /usr/local/nginx/sbin/nginx /usr/sbin/ #创建软连接

[root@web2 ~]# /usr/local/nginx/sbin/nginx

[root@web2 ~]# netstat -utnlp | grep :80 #查看80端口是否是nginxz在用

tcp 0 0 0.0.0.0:80 0.0.0.0:\* LISTEN 32428/nginx

### 2）启动MySQL服务

[root@web2 ~]# systemctl start mariadb

[root@web2 ~]# systemctl status mariadb #查看是否运行正常端口:3306

### 3）启动PHP-FPM服务

[root@web2 ~]# systemctl start php-fpm

[root@web2 ~]# systemctl status php-fpm #查看php-fpm是否正常

[root@web2 ~]# netstat -utnlp | grep php #查看php 使用的端口

tcp 0 0 127.0.0.1:9000 0.0.0.0:\* LISTEN 998/php-fpm: master

[root@proxy vpn]# ss -utnlp |grep php #查看php 使用的详细情况

tcp LISTEN 0 128 127.0.0.1:9000 \*:\* users:(("php-fpm",pid=1329,fd=0),("php-fpm",pid=1328,fd=0),("php-fpm",pid=1327,fd=0),("php-fpm",pid=1326,fd=0),("php-fpm",pid=1325,fd=0),("php-fpm",pid=998,fd=6))

### 4）关闭SELinux、防火墙

[root@web2 ~]# setenforce 0

[root@web2 ~]# firewall-cmd --set-default-zone=trusted #默认是没开启

## 步骤三：部署前端Nginx调度服务器

(在proxy主机:192.168.4.5上操作)

### 使用源码安装nginx软件选择模块

（如果Nginx软件包已存在可以忽略此步骤）

[root@proxy ~]# yum -y install gcc pcre-devel openssl-devel

[root@proxy ~]# tar -xf nginx-1.12.2.tar.gz

[root@proxy ~]# cd nginx-1.12.2

[root@proxy nginx-1.12.2]# ./configure

--prefix=/usr/local/nginx #指定安装路径

--user=nginx

--group=nginx

--with-http\_ssl\_module #开启SSL加密功能模块 SSL虚拟主机用

--with-stream #TCP/UDP代理(四层)用

--with-http\_stub\_status\_module #开启状态页面模块

[root@proxy nginx-1.12.2]# make && make install

### 2）修改Nginx配置文件

Nginx配置文件中，通过upstream定义后端服务器地址池，默认调度策略为轮询，使用proxy\_pass调用upstream定义的服务器地址池：

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

http{

.. ..

upstream webs {

server 192.168.2.100:80;

server 192.168.2.200:80;

}

server {

listen 80;

server\_name localhost;

location / {

proxy\_pass http://webs; #在此调用定义的webs集群

root html;

index index.php index.html index.htm;

}

}

[root@proxy ~]#ln -s /usr/local/nginx/sbin/nginx /usr/sbin #添加软连接

[root@proxy ~]#nginx

[root@proxy ~]#nginx -s reload #重新加载配置文件

[root@proxy ~]# setenforce 0

[root@proxy ~]# firewall-cmd --set-default-zone=trusted

[root@proxy ~]# firefox 192.168.4.5/index.php #按F5刷新显示不同的页面

### 3）浏览器访问测试页面验证。

[root@client ~]# curl http://192.168.4.5/index.html        //查看是否有数据

为测试成功可先分别在Web1 Web2 中放入不同页面.

## 步骤五：部署测试页面(WEB1/WEB2相同操作)

### 1）部署测试页面(Web1服务器为例）。

测试页面可以参考lnmp\_soft/php\_scripts/php-memcached-demo.tar.gz。

[root@web1 ~]# cd lnmp\_soft/php\_scripts/

[root@web1 php\_scripts]# tar -xf php-memcached-demo.tar.gz

[root@web1 php\_scripts]# cd php-memcached-demo

[root@web1 php-memcached-demo]# cp -a \* /usr/local/nginx/html/

### 2）浏览器直接访问后端服务器的测试页面（Web1服务器为例）

[root@web1 ~]# firefox http://192.168.2.100 //填写账户信息

[root@web1 ~]# cd /var/lib/php/session/ //查看服务器本地的Session信息

[root@web1 ~]# ls

sess\_ahilcq9bguot0vqsjtd84k7244 //注意这里的ID是随机的

[root@web1 ~]# cat sess\_ahilcq9bguot0vqsjtd84k7244

**注意：**可用修改index.php和home.php两个文件的内容，添加页面颜色属性，以区别后端两台不同的服务器:<body bgcolor=blue>。

### 3）浏览器访问前端调度器测试（不同后端服务器Session不一致）。

[root@client ~]# google-chrome http://192.168.4.5　＃推荐使用google测试。

//填写注册信息后，刷新，还需要再次注册，说明两台计算机使用的是本地Session第二台主机并不知道你再第一台主机已经登录，第一台主机的登录信息也没有传递给第二台主机

# 4 案例4：PHP实现session共享

4.1 问题

沿用练习三，通过修改PHP-FPM配置文件，实现session会话共享，本案例需要在练习三的基础上实现：

配置PHP使用memcached服务器共享Session信息

客户端访问两台不同的后端Web服务器时，Session 信息一致

4.2 方案

在练习三拓扑的基础上，Nginx服务器除了承担调度器外，还需要担任memcached数据库的角色，并在两台后端LNMP服务器上实现PHP的session会话共享。拓扑结构如图-4所示。

1. 单服务器有session和cookie 防止重复登录

2.搭建集群服务器默认不共享session

3

## 步骤一：构建memcached服务

### 安装Memcached服务

（如果192.168.4.5上已经有本软件包，此步骤可以忽略）

[root@proxy ~]# yum -y install memcached

### 2）启动服务并查看网络连接状态验证是否开启成功：

[root@proxy ~]# systemctl restart memcached

[root@proxy ~]# netstat -anptu | grep memcached

tcp 0 0 0.0.0.0:11211 0.0.0.0:\* LISTEN 2839/memcached

tcp 0 0 :::11211 :::\* LISTEN 2839/memcached

udp 0 0 0.0.0.0:11211 0.0.0.0:\* 2839/memcached

udp 0 0 :::11211 :::\* 2839/memcached

### 3）关闭SELinux、防火墙

[root@proxy ~]# setenforce 0

[root@proxy ~]# firewall-cmd --set-default-zone=trusted

## 步骤二：在后端LNMP服务器上部署Session共享

注意：这些操作在两台后端Web服务器上均需要执行，以下操作以Web1（192.168.2.100）服务器为例。

### 1）为PHP添加memcache扩展

注意，因为后端两台web服务器(web1,web2)都需要连接memcached数据库，所以两台主机都需要安装PHP扩展模块(下面也web1为例)。

[root@web1 ~]# yum -y install php-pecl-memcache

### 2）修改PHP-FPM配置文件，并重启服务

注意，因为后端两台web服务器(web1,web2)都需要修改配置文件(下面也web1为例)。

[root@web1 ~]# vim /etc/php-fpm.d/www.conf //修改该配置文件的两个参数

//文件的最后2行

修改前效果如下:

php\_value[session.save\_handler] = files

php\_value[session.save\_path] = /var/lib/php/session

//原始文件，默认定义Sessoin会话信息本地计算机（默认在/var/lib/php/session）

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

修改后效果如下:

php\_value[session.save\_handler] = memcache

php\_value[session.save\_path] = "tcp://192.168.2.5:11211"

//定义Session信息存储在公共的memcached服务器上，主机参数中为memcache（没有d）

//通过path参数定义公共的memcached服务器在哪（服务器的IP和端口）

[root@web1 ~]# systemctl restart php-fpm

## 步骤三：客户端测试

客户端使用浏览器访问两台不同的Web服务器。

操作步骤与练习三一致，最终可以获得相关的Session ID信息。

前面已完成:

Web1 Web2部署LNMP 访问测试完成

proxy部署nginx 前端调度,测试完成,

proxy 安装memcache数据库,telnet访问测试成功

给Web1 Web2上安装memcache扩展包:php-pecl-memcache

[root@Web1111 html]# yum -y install install php-pecl-memcache

[root@Web1111 html]# systemctl restart php-fpm.service

**扩展**

隐藏版本号,

打开Nginx主配置文件：nginx.conf，取消注释或添加配置语句。

http {

# ...省略一些配置

　　sendfile on;

　　tcp\_nopush on;

　　keepalive\_timeout 60;

　　tcp\_nodelay on;

server\_tokens off; 　　#隐藏版本号，默认是注释掉的

　　# ...省略一些配置

}

[root@vm7 ~]# curl -I www.tarena.com #测试查看

# NSD Operation DAY05

案例1：安装部署Tomcat服务器

案例2：使用Tomcat部署虚拟主机

案例3：使用Varnish加速Web

Java:跨平台,面向对象的程序设计语言,具有通用性,高效性,平台移植性,安全性

Php只是做网页的,java可做网页,做程序,

Java SE 标准板

Java EE 企业版

Java ME 移动版

JDK (java Development kit):java语言软件开发工具包

JDK 包括java运行环境 java工具 java基础的类库(类似shell函数)

JRE 是JDK的子集

JRE 包括java虚拟机 核心类库和支持文件 不包含工具

Java Servlet

Servlet 是java 扩展Web服务器功能的组件规范

常见的Servlet 容器

IBM公司的 websphere

Oracle公司的 weblogic

Apache 公司的 tomcat

Redhat 公司的 Jboss

PHP ,python + Nginx,httpd

# 1 案例1：安装部署Tomcat服务器

1.1 问题

本案例要求部署Tomcat服务器，具体要求如下：

* 安装部署JDK基础环境
* 安装部署Tomcat服务器
* 创建JSP测试页面，文件名为test.jsp，显示服务器当前时间

然后客户机访问此Web服务器验证效果：

* 使用火狐浏览器访问Tomcat服务器的8080端口，浏览默认首页
* 使用火狐浏览器访问Tomcat服务器的8080端口，浏览默认测试页面

1.2 方案

使用2台RHEL7虚拟机，其中一台作为Tomcat服务器（192.168.2.100）、另外一台作为测试用的Linux客户机（192.168.2.5），如图-1所示。

## 1.3 步骤一

实现此案例需要按照如下步骤进行。

步骤一：部署Tomcat服务器软件(192.168.2.100/24)

### 1）使用RPM安装JDK环境

[root@web1 ~]# yum -y install java-1.8.0-openjdk //安装JDK,实验环境已安装

[root@web1 ~]# yum -y install java-1.8.0-openjdk-headless //安装JDK

[root@web1 ~]# java -version //查看JAVA版本

### 2）安装Tomcat（apache-tomcat-8.0.30.tar.gz软件包，在lnmp\_soft中有提供）

[root@web1 ~]# tar -xf apache-tomcat-8.0.30.tar.gz

[root@web1 ~]# mv apache-tomcat-8.0.30 /usr/local/tomcat #安装完成

[root@web1 ~]# ls /usr/local/tomcat

bin/ //主程序目录

lib/ //库文件目录

logs/ //日志目录

temp/ //临时目录

work/ //自动编译目录jsp代码转换servlet

conf/ //配置文件目录

webapps/ //页面目录

Tomcat软件是java写的脚本.

### 3）启动服务

[root@web1 ~]# /usr/local/tomcat/bin/startup.sh #启动tomcat

[root@web1 ~]# /usr/local/tomcat/bin/shutdown.sh #关闭tomcat

[root@web1 ~]# firewall-cmd --set-default-zone=trusted

[root@web1 ~]# setenforce 0

### 4）服务器验证端口信息

[root@web1 ~]# netstat -nutlp |grep java //查看java监听的端口

tcp 0 0 :::8080 :::\* LISTEN 2778/java

tcp 0 0 ::ffff:127.0.0.1:8005 :::\* LISTEN 2778/java

提示：如果检查端口时，8005端口启动非常慢，可用使用下面的命令用urandom替换random（非必须操作）。 因为:tomcat启动需要取随机数,urandom的随机数,比random效率高很多,删除random.用urandom 改名替换random,以后常用

[root@web1 ~]# mv /dev/random /dev/random.bak

[root@web1 ~]# dom

ln -s /dev/urandom /dev/ran

### 5）客户端浏览测试页面(proxy作为客户端)

[root@proxy ~]# firefox http://192.168.2.100:8080

## 步骤二：修改Tomcat配置文件

### 创建测试JSP页面:首页是在 /usr/local/tomcat/webapps/ROOT/ 目录里面

]# vim /usr/local/tomcat/webapps/ROOT/test.jsp

<html>

<body>

<center>

Now time is: <%=new java.util.Date()%> //显示服务器当前时间

</center>

</body>

</html>

### 重启服务

~]# /usr/local/tomcat/bin/shutdown.sh

~]# /usr/local/tomcat/bin/startup.sh

[root@w0..100:8080

[root@proxy ~]# firefox <http://192.168.2.100:8080/test.jsp>

# 2 案例2：使用Tomcat部署虚拟主机

2.1 问题

沿用练习二，使用Tomcat部署加密虚拟主机，实现以下要求：

* 实现两个基于域名的虚拟主机，域名分别为：www.a.com和 www.b.com
* 使用www.a.com域名访问的页面根路径为/usr/local/tomcat/a/ROOT
* 使用www.b.com域名访问的页面根路径为/usr/local/tomcat/b/base
* 访问www.a.com/test时，页面自动跳转到/var/www/html目录下的页面
* 访问页面时支持SSL加密通讯
* 私钥、证书存储路径为/usr/local/tomcat/conf/cert
* 每个虚拟主机都拥有独立的访问日志文件
* 配置tomcat集群环境

### 步骤一 :创建虚拟主机

修改server.xml配置文件，创建两个域名的虚拟主机，修改如下两个参数块(严格区分大小写)：

# cat /usr/local/tomcat/conf/server.xml

<Server>

<Service>

<Connector port=8080 /> #可起多个端口,只要不重复

<Connector port=8009 />

<Engine name="Catalina" defaultHost="localhost">

<Host name="www.a.com" appBase="a" unpackWARS="true" autoDeploy="true">

</Host>

<Host name="www.b.com" appBase="b" unpackWARS="true" autoDeploy="true">

</Host>

<Host name="localhost" appBase="webapps" unpackWARS="true" autoDeploy="true">

</Host>

… …

<Engine name="Catalina" defaultHost="localhost"> 此行表示用IP访问的时候默认打开一个Host name=localhost的网页

每个Host是一个网站,谁在上面,默认就打开谁

name 为域名

appBase 为网站根目录,Tomcat装在哪根目录就在哪,本实验中a在/usr/local/tomcat/a

unpackWARS=”true”:java打包一般为WAR包,true为自动解压

格式2种风格都是正确的:

格式1 <Connector \*\*\* \*\*\* />

格式2<Host > \*\* \*\* \*\* </Host>

2）创建虚拟主机对应的页面根路径 默认都在ROOT下

[root@web1 ~]# mkdir -p /usr/local/tomcat/{a,b}/ROOT

[root@web1 ~]# echo "AAA" > /usr/local/tomcat/a/ROOT/index.html

[root@web1 ~]# echo "BBB" > /usr/local/tomcat/b/ROOT/index.html

3）重启Tomcat服务器

[root@web1 ~]# /usr/local/tomcat/bin/shutdown.sh

[root@web1 ~]# /usr/local/tomcat/bin/startup.sh

4）客户端设置host文件，并浏览测试页面进行测试(proxy充当客户端角色)

注意：ssh远程连接时使用使用-X参数才可以！！！

[root@proxy ~]# vim /etc/hosts

… …

192.168.2.100 www.a.com www.b.com

[root@proxy ~]# firefox http://www.a.com:8080/ //注意访问的端口为8080

[root@proxy ~]# firefox http://www.b.com:8080/

### 步骤二：修改www.b.com网站的首页目录为base

1. 使用docBase参数可以修改默认网站首页路径

[root@web1 ~]# vim /usr/local/tomcat/conf/server.xml

<Host name="www.b.com" appBase="b" unpackWARS="true" autoDeploy="true">

<Context path="" docBase="base"/> #Host 里面增加此行

</Host>

Path:类似与Nginx的地址跳转

[root@web1 ~]# mkdir /usr/local/tomcat/b/base

[root@web1 ~]# echo "BASE" > /usr/local/tomcat/b/base/index.html

[root@web1 ~]# /usr/local/tomcat/bin/shutdown.sh

[root@web1 ~]# /usr/local/tomcat/bin/startup.sh

[root@proxy ~]# firefox http://www.b.com:8080/ //结果为base目录下的页面内容

### 步骤三：跳转

1）当用户访问http://www.a.com/test打开/var/www/html目录下的页面

[root@web1 ~]# vim /usr/local/tomcat/conf/server.xml

… …

<Host name="www.a.com" appBase="a" unpackWARS="true" autoDeploy="true">

<Context path="/test" docBase="/var/www/html/" />

</Host>

<Host name="www.b.com" appBase="b" unpackWARS="true" autoDeploy="true">

<Context path="" docBase="base" />

</Host>

… …

当访问www.a.com/test 将跳转到/var/www/html

[root@web1 ~]# echo "Test" > /var/www/html/index.html

[root@web1 ~]# /usr/local/tomcat/bin/shutdown.sh

[root@web1 ~]# /usr/local/tomcat/bin/startup.sh

2）测试查看页面是否正确(proxy充当客户端角色)

[root@proxy ~]# firefox http://www.a.com:8080/test

//返回/var/www/html/index.html的内容

//注意，访问的端口为8080

### 步骤四：配置Tomcat支持SSL加密网站

1）创建加密用的私钥和证书文件

只要加密一次,所有的网站都会加密,端口和网站是不绑定的,一个端口可访问所有的网站.

<Engine name="Catalina" defaultHost="localhost">

如果defaultHost 为空的,,,即为没有默认网站,那么访问网页未指定域名.将被拒绝

[root@web1 ~]# keytool -genkeypair -alias tomcat -keyalg RSA -keystore /usr/local/tomcat/keystore //提示输入密码为:123456

//-genkeypair 生成密钥对

//-alias tomcat 密钥别名

//-keyalg RSA 定义密钥算法为RSA算法

//-keystore 定义密钥文件存储在:/usr/local/tomcat/keystore

可最后不输入文件存储路径.将会弹出提示:

[root@Web1111 ~]# keytool -genkeypair -alias tomcat -keyalg RSA -keystore

命令选项-keystore需要一个参数。

keytool -genkeypair [OPTION]...

生成密钥对

选项:

-alias <alias> 要处理的条目的别名

-keyalg <keyalg> 密钥算法名称

-keysize <keysize> 密钥位大小

-sigalg <sigalg> 签名算法名称

-destalias <destalias> 目标别名

-dname <dname> 唯一判别名

-startdate <startdate> 证书有效期开始日期/时间

-ext <value> X.509 扩展

-validity <valDays> 有效天数

-keypass <arg> 密钥口令

-keystore <keystore> 密钥库名称

-storepass <arg> 密钥库口令

-storetype <storetype> 密钥库类型

-providername <providername> 提供方名称

-providerclass <providerclass> 提供方类名

-providerarg <arg> 提供方参数

-providerpath <pathlist> 提供方类路径

-v 详细输出

-protected 通过受保护的机制的口令

[root@Web1111 ~]# keytool -genkeypair -alias tomcat -keyalg RSA -keystore /usr/local/tomcat/keystore

输入密钥库口令: #本实验设置口令为:123456

再次输入新口令:

您的名字与姓氏是什么?

[Unknown]: zdd

您的组织单位名称是什么?

[Unknown]: zdd

您的组织名称是什么?

[Unknown]: zdd

您所在的城市或区域名称是什么?

[Unknown]: zdd

您所在的省/市/自治区名称是什么?

[Unknown]: zdd

该单位的双字母国家/地区代码是什么?

[Unknown]: zdd

CN=zdd, OU=zdd, O=zdd, L=zdd, ST=zdd, C=zdd是否正确?

[否]: y

输入 <tomcat> 的密钥口令 #本实验设置口令为:123456

(如果和密钥库口令相同, 按回车):

再次输入新口令:

[root@Web1111 ~]#

2)再次修改server.xml配置文件，创建支持加密连接的Connector

[root@web1 ~]# vim /usr/local/tomcat/conf/server.xml

… …

<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"

maxThreads="150" SSLEnabled="true" scheme="https" secure="true"

keystoreFile="/usr/local/tomcat/keystore" keystorePass="123456" clientAuth="false" sslProtocol="TLS" />

//备注，默认这段Connector被注释掉了，打开注释，添加密钥信息即可,搜索port="8443"

参数之间无先后顺序,必须有至少一个空格或者回车隔开

需手写秘钥目录:keytoreFile=”/usr/local/tomcat/keystore”

需手写秘钥打开的密码:keystorePass="123456" #密码为”123456”

3）重启Tomcat服务器

[root@web1 ~]# /usr/local/tomcat/bin/shutdown.sh

[root@web1 ~]# /usr/local/tomcat/bin/startup.sh

4）客户端设置host文件，并浏览测试页面进行测试(proxy充当客户端角色)

[root@proxy ~]# vim /etc/hosts

… …

192.168.2.100 www.a.com www.b.com

[root@proxy ~]# firefox https://www.a.com:8443/

[root@proxy ~]# firefox https://www.b.com:8443/

[root@proxy ~]# firefox https://192.168.2.100:8443/

### 步骤五：配置Tomcat日志

1)为每个虚拟主机设置不同的日志文件

[root@web1 ~]# vim /usr/local/tomcat/conf/server.xml

.. ..

<Host name="www.a.com" appBase="a" unpackWARS="true" autoDeploy="true">

<Context path="/test" docBase="/var/www/html/" />

#从默认localhost虚拟主机中把Valve这段复制过来，适当修改下即可

<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"

prefix=" a\_access" suffix=".txt"

pattern="%h %l %u %t &quot;%r&quot; %s %b" />

</Host>/LO

<Host name="www.b.com" appBase="b" unpackWARS="true" autoDeploy="true">

<Context path="" docBase="base" />

<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"

prefix=" b\_access" suffix=".txt"

pattern="%h %l %u %t &quot;%r&quot; %s %b" />

</Host>

#从虚拟机localhost的host里面拷贝Valve模板放到虚拟机”www.a.com”与”www.b.com”中

#directory="logs" #定义产生的日志存放目录

#prefix=" b\_access" #定义产生的日志文件名

suffix=".txt" #定义产生的日志后缀名

#pattern="%h %l %u %t &quot;%r&quot; %s %b" />

### 步骤六：扩展实验(配置Tomcat集群)

1) 在192.168.4.5主机上配置Nginx调度器（具体安装步骤参考前面的章节）

[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf

http{

upstream toms {

server 192.168.2.100:8080;

server 192.168.2.200:8080;

}

server {

listen 80;

server\_name localhost;

location / {

proxy\_pass http://toms;

}

}

}

2) 在192.168.2.100和192.168.2.200主机上配置Tomcat调度器

以下以Web1为例：

[root@web1 ~]# yum -y install java-1.8.0-openjdk //安装JDK

[root@web1 ~]# yum -y install java-1.8.0-openjdk-headless //安装JDK

[root@web1 ~]# tar -xzf apache-tomcat-8.0.30.tar.gz

[root@web1 ~]# mv apache-tomcat-8.0.30 /usr/local/tomcat

3）启动服务

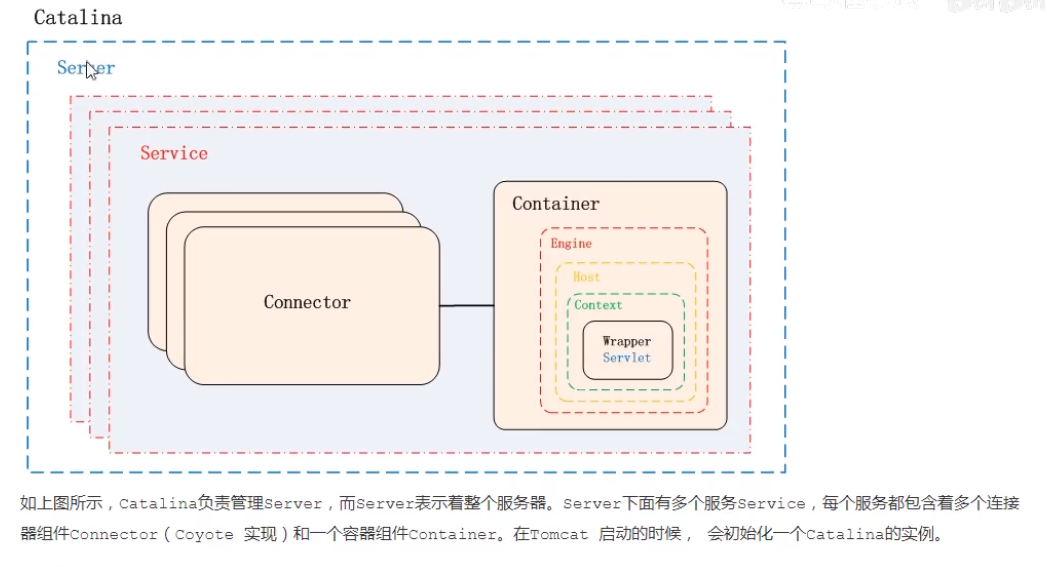
[root@web1 ~]# /usr/local/tomcat/bin/startup.sh

4) 客户端验证

为了防止有数据缓存，可以使用真实主机的google-chrome访问代理服务器，输入Ctrl+F5刷新页面。

Tomcat 结构

容器 catalina





Container 容器





# 3 案例3：使用Varnish加速Web-CDN(内容分发网络)

3.1 问题

通过配置Varnish缓存服务器，实现如下目标：

1. 使用Varnish加速后端Web服务
2. 代理服务器可以将远程的Web服务器页面缓存在本地
3. 远程Web服务器对客户端用户是透明的
4. 利用缓存机制提高网站的响应速度
5. 使用varnishadm命令管理缓存页面
6. 使用varnishstat命令查看Varnish状态

3.2 方案

通过源码编译安装Varnish缓存服务器

编译安装Varnish软件

修改配置文件，缓存代理源Web服务器，实现Web加速功能

使用3台RHEL7虚拟机，其中一台作为Web服务器（192.168.2.100）、一台作为Varnish代理服务器（192.168.4.5,192.168.2.5)，另外一台作为测试用的Linux客户机（192.168.4.10），如图-2所示。

### 步骤一：构建Web服务器

测试web111和web222的web服务器是否都能访问, httpd或者nginx 搭建的都行

### 步骤二：部署Varnish缓存服务器(192.168.4.5)

1. 编译安装软件(python-docutils默认光盘中没有，需要在lnmp\_soft中找)

[root@proxy ~]# yum -y install gcc readline-devel //安装软件依赖包

[root@proxy ~]# yum -y install ncurses-devel //安装软件依赖包

[root@proxy ~]# yum -y install pcre-devel //安装软件依赖包

[root@proxy ~]# yum -y install \

python-docutils-0.11-0.2.20130715svn7687.el7.noarch.rpm //安装软件依赖包

[root@proxy ~]# useradd -s /sbin/nologin varnish //创建账户

[root@proxy ~]# tar -xf varnish-5.2.1.tar.gz

[root@proxy ~]# cd varnish-5.2.1

[root@proxy varnish-5.2.1]# ./configure

[root@proxy varnish-5.2.1]# make && make install

2）复制启动脚本及配置文件

[root@proxy varnish-5.2.1]# cp etc/example.vcl /usr/local/etc/default.vcl

#在varnish-5.2.1目录里面拷贝 etc/example.vcl

3）修改代理配置文件

[root@proxy ~]# vim /usr/local/etc/default.vcl

backend default {

.host = "192.168.2.100";

.port = "80";

}

1. 启动服务

[root@proxy varnish-5.2.1]# varnishd -f /usr/local/etc/default.vcl

Error: Could not get socket :80: Address already in use

(-? gives usage) #80端口被占用,启动不了

[root@proxy varnish-5.2.1]# ss -nlutp | grep :80 #查看80端口

tcp LISTEN 0 128 \*:80 \*:\* users:(("nginx",pid=2585,fd=6),("nginx",pid=2562,fd=6))

[root@proxy varnish-5.2.1]# nginx -s stop #停止占用80端口的nginx

[root@proxy varnish-5.2.1]# varnishd -f /usr/local/etc/default.vcl #重启,成功

Debug: Platform: Linux,3.10.0-693.el7.x86\_64,x86\_64,-junix,-smalloc,-smalloc,-hcritbit

Debug: Child (13552) Started

[root@proxy ~]# varnishd -f /usr/local/etc/default.vcl

//varnishd命令的其他选项说明如下：

//varnishd –s malloc,128M 定义varnish使用内存作为缓存，空间为128M

//varnishd –s file,/var/lib/varnish\_storage.bin,1G 定义varnish使用文件作为缓存

### 步骤三：客户端测试

1）客户端开启浏览器访问

[root@client ~]# curl http://192.168.4.5

### 步骤四：其他操作

1）查看varnish日志

[root@proxy ~]# varnishlog //varnish日志

[root@proxy ~]# varnishncsa //访问日志

2）更新缓存数据，在后台web服务器更新页面内容后，用户访问代理服务器看到的还是之前的数据，说明缓存中的数据过期了需要更新（默认也会自动更新，但非实时更新）。

[root@proxy ~]# varnishadm

varnish> ban req.url ~ .\*

//清空缓存数据，支持正则表达式

# NSD Operation DAY06

什么是Subversion 简称SVN:自由/开源的版本控制系统,

1. 使得文件和目录可以超越时空
2. Subversion允许你数据恢复到早期版本
3. 或者是检查数据修改的历史
4. 允许你和别人协作文档并跟踪所做的修改
5. 许多人将版本控制系统当做一种神奇的”时间机器”

Subversion repository:svn仓库

案例1：Subversion基本操作

案例2：使用Subversion协同工作

案例3：制作nginx的RPM包

# 1 案例1：Subversion基本操作

1.1 问题

本案例要求先快速搭建好一台Subversion服务器，并测试该版本控制软件：

创建版本库

导入初始化数据

检出数据至用户本地副本

对本地副本进行增删改查等操作

1.2 方案

使用YUM安装subversion软件，使用svn客户端工具连接svnserver服务器并测试版本控制软件。

## 步骤一：安装Subversion服务器

Subversion(简称SVN)是近年来崛起的版本管理软件系统，是CVS的接班人。目前，绝大多数开源软件都使用SVN作为代码版本管理软件。

Subversion是一个版本控制系统，相对于的RCS、CVS，采用了分支管理系统，它的设计目标就是取代CVS。互联网上免费的版本控制服务多基于Subversion。

它是一个通用系统，可以管理任何类型的文件集。

### YUM安装subversion软件

[root@web1 ~]# yum -y install subversion

[root@web1 ~]# rpm -q subversion

### 2)创建版本库

[root@web1 ~]# mkdir /var/svn/

[root@web1 ~]# svnadmin create /var/svn/project

[root@web1 ~]# ls /var/svn/project/

conf/ db/ format hooks/ locks/ README.txt

### 3）本地导入初始化数据

[root@web1 ~]# cd /usr/lib/systemd/system/

[root@web1 system]# svn import . file:///var/svn/project/ -m "Init Data"

将system目录里面的 . 导入到 本地/var/svn/project/

#-m “lnit Data” -m后面跟的是本次导入的备注或者注释,

#/usr/lib/systemd/system/目录里面存放有所有systemctl 能启动的程序配置文件

### 4）修改配置文件，创建账户与密码

所有配置文件，要求顶头写，开头不要有空格。有三个文件需要修改

[root@Web1111 ~]# cd /var/svn/project/conf/

[root@Web1111 conf]# ls

authz passwd svnserve.conf

[root@web1 ~]# vim /var/svn/project/conf/svnserve.conf

[general]

### These options control access to the repository for unauthenticated

### and authenticated users. Valid values are "write", "read",

### and "none". The sample settings below are the defaults.

anon-access = none

//19行，匿名无任何权限

auth-access = write

//20行，有效账户可读可写

password-db = passwd

//27行，指定密码文件

authz-db = authz

//34行，ACL访问控制列表文件

[root@web1 ~]# vim /var/svn/project/conf/passwd

… …

[users]

harry = 123456

//用户名和密码

tom = 123456

//用户名和密码

[root@web1 ~]# cat /var/svn/project/conf/authz

[/] //定义ACL访问控制

harry = rw //用户对项目根路径可读可写

tom = rw

### 备注:权限根目录

[/] 根是指 /var/svn/project目录 第3步中,本地导入初始化数据的的目录,

[/abc] 是指/var/svn/project/abc 目录

### 5）启动服务

[root@web1 ~]# svnserve -d -r /var/svn/project

[root@web1 ~]# netstat -nutlp |grep svnserve

tcp 0 0 0.0.0.0:3690 0.0.0.0:\* LISTEN 4043/svnserve

[root@Web1111 conf]# ss -nutlp | grep 3690

tcp LISTEN 0 7 \*:3690 \*:\* users:(("svnserve",pid=9461,fd=3))

svnserve -d -r /var/svn/project

#-d 是放在后台运行

#-r 是要共享的目录  
备注：启动服务也可以使用svnserve -d启动，但客户端访问时需要指定绝对路径（svn://服务器IP/var/svn/project）。

[root@web1 ~]# svnserve -d -r /var/svn/project

[root@web22 ~]# svn co svn://192.168.2.100/ code

[root@web1 ~]# svnserve -d

[root@web22 ~]# svn co svn://192.168.2.100/var/svn/project/ code

## 步骤二：客户端测试(192.168.2.200)

### 1）将服务器上的代码下载到本地

[root@web2 ~]# cd /tmp

[root@web2 ~]# svn --username harry --password 123456 \

co svn://192.168.2.100/ code

#建立本地副本,从服务器192.168.2.100上co下载代码到本地code目录(code自动新建)

//用户名harry,密码123456

Store password unencrypted (yes/no)? yes //提示是否保存密码

### 2) 上传修改过的文件:

[root@web2 ~]# cd /tmp/code

[root@web2 code]# ls

[root@web2 code]# vim user.slice //挑选任意文件修改其内容

[root@web2 code]# svn ci -m "modify user" //将本地修改的数据同步到服务器

### 3) 查看:

[root@web2 code]# svn update //将服务器上新的数据同步到本地

[root@web2 code]# svn info svn://192.168.2.100 //查看版本仓库基本信息

[root@web2 code]# svn log svn://192.168.2.100 //查看版本仓库的日志

### 4) 将新文件加入版本控制:

[root@web2 code]# echo "test" > test.sh //本地新建一个文件

[root@web2 code]# svn ci -m "new file" //提交失败，该文件不被svn管理

[root@web2 code]# svn add test.sh //将文件或目录加入版本控制

[root@web2 code]# svn ci -m "new file" //再次提交，成功

[root@web2 code]# svn mkdir subdir //创建子目录

[root@web2 code]# svn rm timers.target //使用svn删除文件

[root@web2 code]# svn ci -m "xxx" //提交一次代码

[root@web2 code]# vim umount.target //任意修改本地的一个文件

[root@web2 code]# svn diff //查看所有文件的差异

[root@web2 code]# svn diff umount.target //仅查看某一个文件的差异

[root@web2 code]# svn cat svn://192.168.2.100/reboot.target //查看服务器文件的内容

### 5) 对比文件的不同小方法:

[root@Web222 code]# svn diff #查到svn服务器zram.service文件与本机有不同

Index: zram.service

===================================================================

--- zram.service (版本 3)

+++ zram.service (工作副本)

@@ -12,3 +12,4 @@

new 16:40

new 18:04

+18:09

#查看svn服务器zram.service文件具体内容

[root@Web222 code]# svn cat svn://192.168.2.100/zram.service

[Unit]

Description=Service enabling compressing RAM with zRam

ConditionKernelCommandLine=!inst.zram=off

ConditionKernelCommandLine=!inst.zram=0

[Service]

Type=oneshot

ExecStart=/usr/libexec/anaconda/zramswapon

ExecStop=/usr/libexec/anaconda/zramswapoff

RemainAfterExit=yes

new 16:40

new 18:04

[root@Web222 code]# cat zram.service #查看本机zram.service文件具体内容

[Unit]

Description=Service enabling compressing RAM with zRam

ConditionKernelCommandLine=!inst.zram=off

ConditionKernelCommandLine=!inst.zram=0

[Service]

Type=oneshot

ExecStart=/usr/libexec/anaconda/zramswapon

ExecStop=/usr/libexec/anaconda/zramswapoff

RemainAfterExit=yes

new 16:40

new 18:04

18:09 #不同之处

[root@web2 code]# sed -i 'd' tmp.mount

//删除文件所有内容，但未提交

[root@web2 code]# svn revert tmp.mount

已恢复“tmp.mount”

//还原tmp.mount文件

[root@web2 code]# rm -rf \*.target

//任意删除若干文件

[root@web2 code]# svn update

//还原

[root@web2 code]# sed -i '1a #test###' tuned.service

//修改本地副本中的代码文件

[root@web2 code]# svn ci -m "xxx"

//提交代码

[root@web2 code]# svn merge -r7:2 tuned.service

//将文件从版本7还原到版本2

### 6) 使用svn命令测试svnserver服务时可以使用的命令列表如表-1所示。



# 2 案例2：使用Subversion协同工作

### 2.1 问题

沿用练习一，通过svn工具，对subversion版本库进行多人协同工作测试，要求如下：

* 该版本库支持多个账户同时协作编辑文件
* 测试演示多人协作编辑的具体操作
* 手动解决版本冲突问题
* 备份版本库数据
* 注册使用Github

### 2.2 方案

使用svn客户端工具连接subversion服务器并测试多人协同工作以及如何手动解决冲突问题，账户名称分别为harry和tom，最后使用svnadmin dump指令对版本库进行备份工作。

## 步骤一：多人协同工作

### 1)模拟两个终端

下载代码本地副本，注意

web1(192.168.2.100)用tom登录svn

web2（192.168.2.200)用harry登录svn

web1和web2代表了两个不同的主机，看清楚操作是在哪一台计算机上执行！

[root@web1 ~]# cd /tmp

[root@web1 ~]# svn --username tom --password 123456 \

> co svn://192.168.2.100/ code

[root@web2 ~]# cd /tmp

[root@web2 ~]# svn --username harry --password 123456 \

> co svn://192.168.2.100/ code

[root@web1 ~]# cd code

[root@web2 ~]# cd code

### 2) harry和tom修改不同的文件的情况

[root@web1 mycode]# sed -i "3a ###tom modify#####" tmp.mount

[root@web1 mycode]# svn ci -m "has modified"

[root@web2 mycode]# sed -i "3a ###harry modify#####" umount.target

[root@web2 mycode]# svn ci -m "has modified"

[root@web2 mycode]# svn update

[root@web1 mycode]# svn update

### 3) harry和tom修改相同文件的不同行

[root@srv5 ~]# cd harry

[root@web1 mycode]# sed -i "3a ###tom modify#####" user.slice

[root@web1 mycode]# svn ci -m "modified"

[root@web2 mycode]# sed -i "6a ###harry modify#####" user.slice

[root@web2 mycode]# svn ci -m "modified" //提交失败

Sending svnserve

Transmitting file data .svn: Commit failed (details follow):

svn: File '/user.slice' is out of date（过期）

[root@web2 mycode]# svn update //提示失败后，先更新再提交即可

[root@web2 mycode]# svn ci -m "modified" //提交成功

Sending user.slice

Transmitting file data .

### 4) harry和tom修改相同文件的相同行

[root@web1 mycode]# sed -i '1c [UNIT]' tuned.service

[root@web1 mycode]# svn ci -m "modified"

[root@web2 mycode]# sed -i '1c [unit]' tuned.service

[root@web2 mycode]# svn ci -m "modified"

Sending tuned.service

Transmitting file data .svn: Commit failed (details follow):

svn: File '/tuned.service' is out of date(过期)

[root@web2 mycode]# svn update //出现冲突，需要解决

Conflict(冲突) discovered in 'tuned.service'.

Select: (p) postpone, (df) diff-full, (e) edit,

(mc) mine-conflict, (tc) theirs-conflict,

(s) show all options:p //选择先标记p，随后解决

[root@web2 mycode]# ls

tuned.service tuned.service.mine tuned.service.r8 tuned.service.r7

### 5)查看和解决冲突文件:

#tuned.service #什么都没有,只是显示两个冲突的不同

[root@Web222 code]# head -4 tuned.service

<<<<<<< .mine

[unit] #web222修改的内容

=======

[UNIT] #web111修改的内荣

#tuned.service.mine #自己(web222)修改的内容

[root@Web222 code]# head -1 tuned.service.mine

[unit]

#tuned.service.r7 #发生冲突之前的第7个版本内容

[root@Web222 code]# head -1 tuned.service.r7

[Unit]

#tuned.service.r10 #web111修改的内容

[root@Web222 code]# head -1 tuned.service.r8

[UNIT]

经过协商,确定web222的版本为最终版本,在web222上操作,将web222的版本tuned.service.mine 移动覆盖tuned.service,然后删掉tuned.service.r7(冲突之前的版本) 和删除tuned.service.r8(web111修改的版本)

[root@web2 mycode]# mv tuned.service.mine tuned.service

[root@web2 mycode]# rm -rf tuned.service.r10 tuned.service.r9

[root@web2 mycode]# svn ci -m "modified" //解决冲突

## 步骤二：使用dump指令备份版本库数据

[root@web1 ~]# svnadmin dump /var/svn/project > project.bak //备份

\* Dumped revision 0.

\* Dumped revision 1.

\* Dumped revision 2.

\* Dumped revision 3.

\* Dumped revision 4.

[root@web1 ~]# svnadmin create /var/svn/project2 //新建空仓库

[root@web1 ~]# svnadmin load /var/svn/project2 < project.bak //还原

## 步骤三：注册使用Github(非常重要的)

git(分布式版本控制软件)

1.登陆网站https://github.com，点击Sign up（注册），如图-1所示。

2.填写注册信息（用户名，邮箱，密码），如图-2所示。

3. 初始化操作，如图-3和图-4所示。

注意，初始化完成后，到邮箱中去激活Github账户。

4. 创建仓库、使用仓库

点击Start a project（如图-5所示），

填写项目名称（项目名称任意），如图-6所示。

往仓库中上传文件或新建文件，如图-7所示

下载仓库中的代码，如图-8所示。

5. 命令行操作（需要联网的主机，如真实机）

[root@pc001 ~]# yum -y install git

[root@pc001 ~]# git clone https://github.com/账户名称/仓库名称

#clone指令用于将服务器仓库中的资料打包下载到本地

[root@pc001 ~]# cd 仓库名称

[root@pc001 ~]# 任意修改文件，或新建文件

[root@pc001 ~]# git add . #.代表自己识别新文件并全部添加到git

#add添加新文件

[root@pc001 ~]# git commit -m "test" #提交到本机服务器,未提交到github.com

[root@pc001 ~]# git push #提交到github.com

#commit和push实现提交代码的功能

[root@pc001 ~]# git pull

#pull更新，类似于svn update

# 3 案例3：制作nginx的RPM包

### 3.1 问题

本案例使用nginx-1.12.2版本的源码软件，生成对应的RPM包软件，具体要求如下：

软件名称为nginx

软件版本为1.12.2

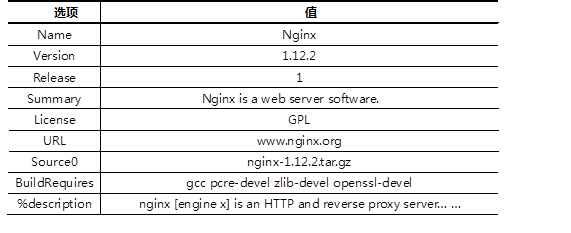
RPM软件包可以查询描述信息

RPM软件包可以安装及卸载

### 3.2 方案

安装rpm-build软件包，编写SPEC配置文件，创建新的RPM软件包。

配置文件中的描述信息如表-2：



## 步骤一：安装rpm-build软件

### 1）安装rpm-build软件包

[root@web1 ~]# yum -y install rpm-build

### 2）生成rpmbuild目录结构

[root@web1 ~]# rpmbuild -ba nginx.spec //会报错，没有文件或目录

错误：stat /root/nginx.spec 失败：没有那个文件或目录

虽然报错,但是会创建rpmbuild目录

[root@Web1111 ~]# ls

anaconda-ks.cfg initial-setup-ks.cfg lnmp\_soft.tar.gz nginx-1.12.2.tar.gz 公共 视频 文档 音乐

code lnmp\_soft nginx-1.12.2 rpmbuild 模板 图片 下载 桌面

[root@web1 ~]# ls /root/rpmbuild //查看自动生成的目录结构

BUILD BUILDROOT RPMS SOURCES SPECS SRPMS

备注:SOURCES 放源码包

RPMS 出成果

### 3）准备工作，将源码软件复制到SOURCES目录

[root@web1 ~]# cp nginx-1.12.2.tar.gz /root/rpmbuild/SOURCES/

### 创建并修改SPEC配置文件

[root@web1 ~]# vim /root/rpmbuild/SPECS/nginx.spec

Name:nginx #软件名:跟刚刚拷贝的,

Version:1.12.2 #软件版本,

Release: 10 #第几次制作RPM包

Summary: Nginx is a web server software. #备注信息

#以上内容就是命令:(yum info 软件名) 能查到的信息

#Group: #软件分组,按需配置可不用填软件分组,可将多个软件包分到一个组,一次性安装所有

License:GPL #软件协议GPL:谁都可以复制,修改.如果有人修改了,允许你重新打标签,放上自己的商标再发布,但是发布的必须开源,否则可起诉,受法律保护

URL: www.-.com #网址随便写

Source0:nginx-1.12.2.tar.gz #刚刚拷贝的源码包名

#BuildRequires:

#Requires: #是否有依赖包,不能解决,但是在rpm安装界面会提示

%description #可写更详细的信息

nginx [engine x] is an HTTP and reverse proxy server.

%post #安装后脚本,安装完之后执行下面命令.可植入恶意命令

useradd -s /sbin/lonogin/ nginx m // %post需要手敲,非必需操作：安装后脚本(创建账户)

%prep

%setup –q //自动解压上面Source0填写的源码包，并cd进入目录

%build

./configure #可添加模块--profile+\*\*\* --user=\*\* 等等

make %{?\_smp\_mflags}

%install

make install DESTDIR=%{buildroot}

%files

%doc

/usr/local/nginx/\* //对哪些文件与目录打包

%changelog

\*.spec 新建spec文件都有以上的内容

Wget <http://wwwf.asdf.co> -o /root/.sh 下载

步骤二：使用配置文件创建RPM包

1）安装依赖软件包

[root@web1 ~]# yum -y install gcc pcre-devel openssl-devel

2）rpmbuild创建RPM软件包

[root@web1 ~]# rpmbuild -ba /root/rpmbuild/SPECS/nginx.spec

[root@web1 ~]# ls /root/rpmbuild/RPMS/x86\_64/nginx-1.12.2-10.x86\_64.rpm

[root@Web1111 SPECS]# rpm -qpi RPMS/x86\_64/nginx-1.12.2-10.x86\_64.rpm

错误：打开 RPMS/x86\_64/nginx-1.12.2-10.x86\_64.rpm 失败： 没有那个文件或目录

[root@Web1111 SPECS]# rpm -qpi /root/rpmbuild/BUILDROOT/RPMS/x86\_64/nginx-1.12.2-10.x86\_64.rpm

错误：打开 /root/rpmbuild/BUILDROOT/RPMS/x86\_64/nginx-1.12.2-10.x86\_64.rpm 失败： 没有那个文件或目录

[root@Web1111 SPECS]# ls

nginx.spec

[root@Web1111 SPECS]# cd ..

[root@Web1111 rpmbuild]# ls #注意此时是在rpmbuild目录里面

BUILD BUILDROOT RPMS SOURCES SPECS SRPMS

[root@Web1111 rpmbuild]# rpm -qpi RPMS/x86\_64/nginx-1.12.2-10.x86\_64.rpm

Name : nginx

Version : 1.12.2

Release : 10

Architecture: x86\_64

Install Date: (not installed)

Group : Unspecified

Size : 909821

License : GPL

Signature : (none)

Source RPM : nginx-1.12.2-10.src.rpm

Build Date : 2019年01月26日 星期六 14时17分07秒

Build Host : Web1111

Relocations : (not relocatable)

URL : www.suibianxie.com

Summary : suibianxie

Description :

Suibianxie

[root@web1 ~]# rpm -qpl nginx-1.12.2-10.x86\_64.rpm

/usr

/usr/local

/usr/local/nginx

/usr/local/nginx/conf

/usr/local/nginx/conf/fastcgi.conf

/usr/local/nginx/conf/fastcgi.conf.default

/usr/local/nginx/conf/fastcgi\_params

/usr/local/nginx/conf/fastcgi\_params.default

/usr/local/nginx/conf/koi-utf

/usr/local/nginx/conf/koi-win

/usr/local/nginx/conf/mime.types

/usr/local/nginx/conf/mime.types.default

/usr/local/nginx/conf/nginx.conf

/usr/local/nginx/conf/nginx.conf.default

/usr/local/nginx/conf/scgi\_params

/usr/local/nginx/conf/scgi\_params.default

/usr/local/nginx/conf/uwsgi\_params

/usr/local/nginx/conf/uwsgi\_params.default

/usr/local/nginx/conf/win-utf

/usr/local/nginx/html

/usr/local/nginx/html/50x.html

/usr/local/nginx/html/index.html

/usr/local/nginx/logs

/usr/local/nginx/sbin

/usr/local/nginx/sbin/nginx

步骤三：安装、卸载软件

[root@web1 ~]# rpm -ivh RPMS/x86\_64/nginx-1.12.2-10.x86\_64.rpm

[root@web1 ~]# rpm -qa |grep nginx

[root@web1 ~]# /usr/local/nginx/sbin/nginx

[root@web1 ~]# curl <http://127.0.0.1/>

rmp 命令的参数:???

# NSD Operation DAY07

案例1：配置GRE VPN

案例2：创建PPTP VPN

案例3：创建L2TP+IPSec VPN

案例4：NTP时间同步

案例5：pssh远程套件工具

# VPN虚拟专用网:

**概述:**

1. 在公网上建立专用私有网络,进行加密通讯
2. 多用于集团公司的各地子公司建立连接
3. 连接上之后,各个地区的子公司可以像局域网一样通讯
4. 偶尔可用于翻墙,
5. 目前主流的VPN技术(GRE(windows不支持), PPTP, L2TP+IPSec, SSL)
6. 国内单位使用VPN需要备案

**主流VPN：**

* GRE:不支持windows系统，
* PPTP VPN:点对点隧道协议（pptp）基于拨号使用的PPP协议使用PAP或CHAP之类的加密算法，或者使用Microsoft的点对点加密算法MPPE。
* L2TP VPN: 第 2 层隧道协议 (L2TP) 是IETF基于L2F (Cisco的第二层转发协议)开发的PPTP的后续版本。是一种工业标准 Internet 隧道协议，其可以为跨越面向数据包的媒体发送点到点协议 (PPP) 框架提供封装。PPTP和L2TP都使用PPP协议对数据进行封装，然后添加附加包头用于数据在互联网络上的传输。PPTP只能在两端点间建立单一隧道。 L2TP支持在两端点间使用多隧道，用户可以针对不同的服务质量创建不同的隧道。L2TP可以提供隧道验证，而PPTP则不支持隧道验证。但是当L2TP 或PPTP与IPSEC共同使用时，可以由IPSEC提供隧道验证，不需要在第2层协议上验证隧道使用L2TP。 PPTP要求互联网络为IP网络。L2TP只要求隧道媒介提供面向数据包的点对点的连接，L2TP可以在IP(使用UDP)，桢中继永久虚拟电路 (PVCs),X.25虚拟电路(VCs)或ATM VCs网络上使用。
* IPSec: IPSec 的隧道是封装、路由与解封装的整个过程。隧道将原始数据包隐藏(或封装)在新的数据包内部。该新的数据包可能会有新的寻址与路由信息，从而使其能够通过网络传输。隧道与数据保密性结合使用时，在网络上窃听通讯的人将无法获取原始数据包数据(以及原始的源和目标)。封装的数据包到达目的地后，会删除封装，原始数据包头用于将数据包路由到最终目的地。
* SSL VPN：SSL协议提供了数据私密性、端点验证、信息完整性等特性。SSL协议由许多子协议组成，其中两个主要的子协议是握手协议和记录协议。握手协议允许服务器 和客户端在应用协议传输第一个数据字节以前，彼此确认，协商一种加密算法和密码钥匙。在数据传输期间，记录协议利用握手协议生成的密钥加密和解密后来交换 的数据。
* Open VPN：典型的SSL VPN应用如OpenVPN，是一个比较好的开源软件。PPTP主要为那些经常外出移动或家庭办公的用户考虑；而OpenVPN主要是针对企业异地两地总分公司之间的VPN不间断按需连接，例如ERP在企业中的应用。OpenVPN 允许参与建立VPN的单点使用预设的私钥，第三方证书，或者用户名/密码来进行身份验证。它大量使用了OpenSSL加密库，以及SSLv3/TLSv1 协议。OpenVPN能在Linux、xBSD、Mac OS X与Windows 2000/XP上运行。它并不是一个基于Web的VPN软件，也不与IPsec及其他VPN软件包兼容。

区别

* 轮使用时速度：几乎没区别
* 轮安全：L2TP高，openvpn更高，PPTP就不怎样了。

例：之前测试过客户端连接PPTP然后在服务端抓包，居然能抓到明文的包，根据别人的包直接就能登录到别人的论坛帐号。不过不需要担心这点，一般服务器管理员不会抓你的包，抓了也要找半天哪个才是论坛登录的包，其次用VPN就会被盗号影响自己声誉，砸自己饭碗。

* 轮连接时速度：
  + PPTP连接速度很快，如果服务器近+给力的情况下，点一下连接就很快就连上
  + L2TP就没那么快了，至于OPENVPN，就算是VPN是局域网的，连接时速度也很慢
* 轮网络环境：
  + pptp，使用率最高的协议，连接速度快而稳定，适合大部分网络
  + l2tp，安全和兼容性好的协议，适合特殊网络和重视安全网络事物
  + openvpn，只要服务端无异常和客户端配置正确，基本上没有连不上。而且不容易被检测出使用了代理

个人总结：

PPTP：适合大部分网络，只要工作事物并不是非常重要，可以用这个。

L2TP：网络环境特殊，如：网吧/教育网/铁通网/公司局域网等，这些对PPTP支持不好甚至不支持，在没有映射端口的情况下可以使用

OPENVPN：个人不太喜欢，连接速度慢，还要创建一个网卡，又不能轻易删掉，看着蛋疼

openvpn并不流行，做这个的人并不是很多，服务端配置蛋疼，客户端配置麻烦。vpn协议中流行的就PPTP和L2TP。

# 1 案例1：配置GRE VPN(linux-linux)

不支持windows系统

1.1 问题

本案例要求搭建一个GRE VPN环境，并测试该VPN网络是否能够正常通讯，要求如下：

启用内核模块ip\_gre

创建一个虚拟VPN隧道(10.10.10.0/24)

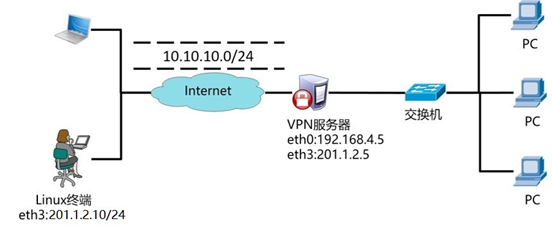
实现两台主机点到点的隧道通讯

1.2 方案

使用lsmod查看当前计算机已经加载的模块，使用modprobe加载Linux内核模块，使用modinfo可以查看内核模块的信息。

准备实验所需的虚拟机环境，实验环境所需要的主机及对应的IP设置列表如表-1所示，正确配置IP地址、主机名称，并且为每台主机配置YUM源。





## 步骤一：启用GRE模块

（client和proxy都需要操作）

### 1）查看计算机当前加载的模块

[root@client ~]# lsmod //显示模块列表

[root@client ~]# lsmod | grep ip\_gre //确定是否加载了gre模块

### 2) 加载模块ip\_gre

[root@client ~]# modprobe ip\_gre

[root@client ~]# lsmod | grep ip\_gre

ip\_gre 22707 0

ip\_tunnel 25163 1 ip\_gre

gre 13144 1 ip\_gre

### 3）查看模块信息

[root@client ~]# modinfo ip\_gre #filename:存放内核的地方

filename: /lib/modules/3.10.0-693.el7.x86\_64/kernel/net/ipv4/ip\_gre.ko.xz

alias: netdev-gretap0

alias: netdev-gre0

alias: rtnl-link-gretap

alias: rtnl-link-gre

license: GPL

rhelversion: 7.4

srcversion: F37A2BF90692F86E3A8BD15

depends: ip\_tunnel,gre

intree: Y

vermagic: 3.10.0-693.el7.x86\_64 SMP mod\_unload modversions

signer: CentOS Linux kernel signing key

sig\_key: DA:18:7D:CA:7D:BE:53:AB:05:BD:13:BD:0C:4E:21:F4:22:B6:A4:9C

sig\_hashalgo: sha256

parm: log\_ecn\_error:Log packets received with corrupted ECN (bool)

[root@client ~]# cd /lib/modules/3.10.0-693.el7.x86\_64/kernel/

[root@client kernel]# ls

arch crypto drivers fs kernel lib mm net sound virt

[root@client kernel]# find ./ -name "\*.ko\*" |wc -l

2406

#总共2406个内核模块

[root@client kernel]# ip help #查看ip 命令的帮助

Usage: ip [ OPTIONS ] OBJECT { COMMAND | help }

ip [ -force ] -batch filename

where OBJECT := { link | address | addrlabel | route | rule | neigh | ntable |

tunnel | tuntap | maddress | mroute | mrule | monitor | xfrm |

netns | l2tp | macsec | tcp\_metrics | token }

OPTIONS := { -V[ersion] | -s[tatistics] | -d[etails] | -r[esolve] |

-h[uman-readable] | -iec |

-f[amily] { inet | inet6 | ipx | dnet | bridge | link } |

-4 | -6 | -I | -D | -B | -0 |

-l[oops] { maximum-addr-flush-attempts } |

-o[neline] | -t[imestamp] | -ts[hort] | -b[atch] [filename] |

-rc[vbuf] [size] | -n[etns] name | -a[ll] }

[root@client kernel]# ip tunnel help #查看ip tunnel 命令的帮助

Usage: ip tunnel { add | change | del | show | prl | 6rd } [ NAME ]

[ mode { ipip | gre | sit | isatap | vti } ] [ remote ADDR ] [ local ADDR ]

[ [i|o]seq ] [ [i|o]key KEY ] [ [i|o]csum ]

[ prl-default ADDR ] [ prl-nodefault ADDR ] [ prl-delete ADDR ]

[ 6rd-prefix ADDR ] [ 6rd-relay\_prefix ADDR ] [ 6rd-reset ]

[ ttl TTL ] [ tos TOS ] [ [no]pmtudisc ] [ dev PHYS\_DEV ]

Where: NAME := STRING

ADDR := { IP\_ADDRESS | any }

TOS := { STRING | 00..ff | inherit | inherit/STRING | inherit/00..ff }

TTL := { 1..255 | inherit }

KEY := { DOTTED\_QUAD | NUMBER }

## 步骤二：Client主机创建VPN隧道

### 1）创建隧道

[root@client ~]# ip tunnel add tun0 mode gre remote 201.1.2.5 local 201.1.2.10

//ip tunnel add创建隧道（隧道名称为tun0），ip tunnel help可以查看帮助

//mode设置隧道使用gre模式

//remote 后面跟对方的IP地址与其他主机建立隧道

//local后面跟本机的IP地址

### 2）启用该隧道（类似与设置网卡up）

[root@client ~]# ip link show #查看

10: tun0@NONE: <POINTOPOINT,NOARP> mtu 1476 qdisc noop state DOWN mode DEFAULT qlen 1

link/gre 201.1.2.10 peer 201.1.2.5

[root@client ~]# ip link set tun0 up //设置UP

[root@client ~]# ip link show

10: tun0@NONE: <POINTOPOINT,NOARP,UP,LOWER\_UP> mtu 1476 qdisc noqueue state UNKNOWN mode DEFAULT qlen 1

link/gre 201.1.2.10 peer 201.1.2.5

### 3）为VPN配置隧道IP地址

[root@client ~]# ip addr add 10.10.10.10/24 peer 10.10.10.5/24 dev tun0

//为隧道tun0设置本地IP地址（10.10.10.10.10/24）

//隧道对面的主机IP的隧道IP为10.10.10.5/24

[root@client ~]# ip a s //查看IP地址

10: tun0@NONE: <POINTOPOINT,NOARP,UP,LOWER\_UP> mtu 1476 qdisc noqueue state UNKNOWN qlen 1

link/gre 201.1.2.10 peer 201.1.2.5

inet 10.10.10.10 peer 10.10.10.5/24 scope global tun0

valid\_lft forever preferred\_lft forever

inet6 fe80::200:5efe:c901:20a/64 scope link

valid\_lft forever preferred\_lft forever

### 4）关闭防火墙

[root@client ~]# firewall-cmd --set-default-zone=trusted

## 步骤三：Proxy主机创建VPN隧道

### 1）查看计算机当前加载的模块

[root@client ~]# lsmod //显示模块列表

[root@client ~]# lsmod | grep ip\_gre //确定是否加载了gre模块

### 2) 加载模块ip\_gre

[root@client ~]# modprobe ip\_gre

### 3）创建隧道

[root@proxy ~]# ~]# ip tunnel add tun0 mode gre \

> remote 201.1.2.10 local 201.1.2.5

//ip tunnel add创建隧道（隧道名称为tun0），ip tunnel help可以查看帮助

//mode设置隧道使用gre模式

//local后面跟本机的IP地址，remote后面是与其他主机建立隧道的对方IP地址

### 4）启用该隧道（类似与设置网卡up）

[root@proxy ~]# ip a s #启用之前用ip a s查看

10: tun0@NONE: <POINTOPOINT,NOARP> mtu 1476 qdisc noop state DOWN qlen 1

link/gre 201.1.2.5 peer 201.1.2.10

[root@proxy ~]# ip link set tun0 up

[root@proxy ~]# ip a s #启用之后用ip a s 查看

10: tun0@NONE: <POINTOPOINT,NOARP,UP,LOWER\_UP> mtu 1476 qdisc noqueue state UNKNOWN qlen 1

link/gre 201.1.2.5 peer 201.1.2.10

inet6 fe80::200:5efe:c901:205/64 scope link

valid\_lft forever preferred\_lft forever

### 5）为VPN配置隧道IP地址

[root@proxy ~]# ip addr add 10.10.10.5/24 peer 10.10.10.10/24 dev tun0

//为隧道tun0设置本地IP地址（10.10.10.10.5/24）

//隧道对面的主机IP的隧道IP为10.10.10.10/24

[root@proxy ~]# ip a s //查看IP地址

10: tun0@NONE: <POINTOPOINT,NOARP,UP,LOWER\_UP> mtu 1476 qdisc noqueue state UNKNOWN qlen 1

link/gre 201.1.2.5 peer 201.1.2.10

inet 10.10.10.5 peer 10.10.10.10/24 scope global tun0

valid\_lft forever preferred\_lft forever

inet6 fe80::200:5efe:c901:205/64 scope link

valid\_lft forever preferred\_lft forever

### 6）开启路由转发、关闭防火墙

[root@proxy ~]# echo "1" > /proc/sys/net/ipv4/ip\_forward

[root@proxy ~]# firewall-cmd --set-default-zone=trusted

### 7)测试连通性

[root@client ~]# ping 10.10.10.5

[root@proxy ~]# ping 10.10.10.10

# 2 案例2：创建PPTP VPN(windows-linux)

支持windows

支持密码身份验证

支持MPPE加密

2.1 问题

本案例要求搭建一个PPTP VPN环境，并测试该VPN网络是否能够正常通讯，要求如下:

使用PPTP协议创建一个支持身份验证的隧道连接

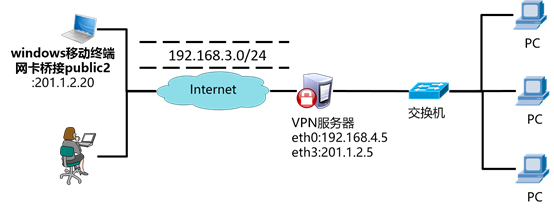
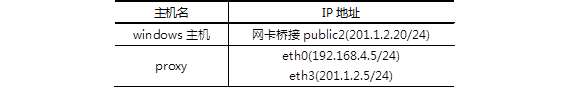
使用MPPE对数据进行加密

为客户端分配192.168.3.0/24的地址池

客户端连接的用户名为jacob，密码为123456

2.2 方案

准备实验所需的虚拟机环境，实验环境所需要的主机及对应的IP设置列表如表-2所示，正确配置IP地址、主机名称，并且为每台主机配置YUM源。



## 步骤一：部署VPN服务器

### 1）安装软件包（软件包参考lnmp\_soft）

[root@proxy ~]# yum localinstall pptpd-1.4.0-2.el7.x86\_64.rpm #本地安装

[root@proxy ~]# rpm -qc pptpd #查看这个软件有没有配置文件

/etc/ppp/options.pptpd

/etc/pptpd.conf

/etc/sysconfig/pptpd

### 2) 修改（三个）配置文件

[root@proxy ~]# vim /etc/pptpd.conf

.. ..#

(Recommended)

#localip 192.168.0.1 #模板:本机IP

#remoteip 192.168.0.234-238,192.168.0.245

#remoteip 分配给客户端的IP地址池,连续的用 - 单个的用 ,

# or

#localip 192.168.0.234-238,192.168.0.245

#remoteip 192.168.1.234-238,192.168.1.245

localip 201.1.2.5 //实际修改服务器本地IP

remoteip 192.168.3.1-50 //实际修改分配给客户端的IP池

[root@proxy ~]# vim /etc/ppp/options.pptpd

name pptpd //vpn给自己取的名字默认为:pptpd

require-mppe-128 //使用MPPE加密数据,将用户认证加密

ms-dns 8.8.8.8 //DNS服务器

[root@proxy ~]# vim /etc/ppp/chap-secrets //修改账户配置文件

zdd \* 123456 \*

//用户名 服务器标记 密码 客户端

#服务器标记可以不写,要写,一定要写前面/etc/ppp/options.pptpd文件里面的name参数相同,本实验为 pptpd

#客户端:是IP地址.\*代表所有IP可用,如果指定单个IP连接,这里填个IP.

### 3) 服务器开启路由转发(Linux软路由)

[root@proxy ~]# echo "1" > /proc/sys/net/ipv4/ip\_forward //开启路由转发

#配置文件ip\_forward 值为1开启路由转发.值为0关闭路由转发

### 4）启动服务

[root@proxy ~]# systemctl start pptpd

[root@proxy ~]# systemctl enable pptpd

[root@proxy ~]# firewall-cmd --set-default-zone=trusted

### 5）翻墙设置（非必需操作）

[root@proxy ~]# iptables -t nat -A POSTROUTING -s 192.168.3.0/24 \

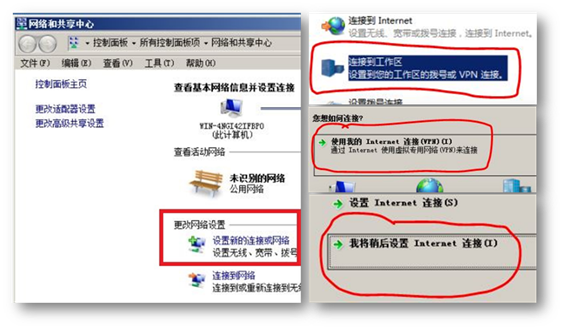
> -j SNAT --to-source 201.1.2.5

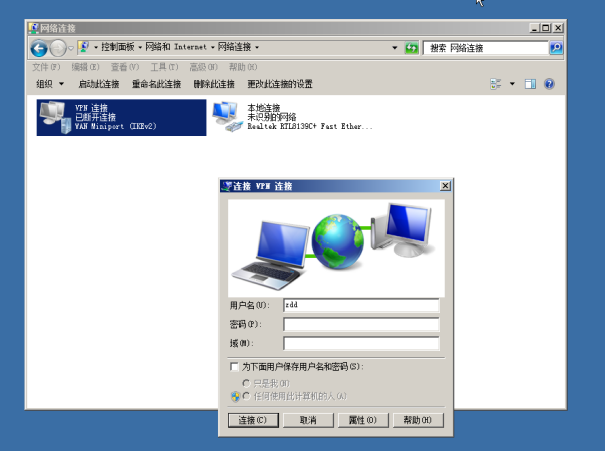
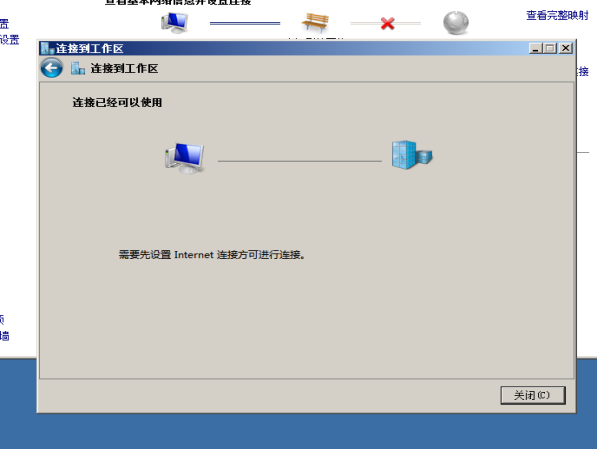
#做nat转换的

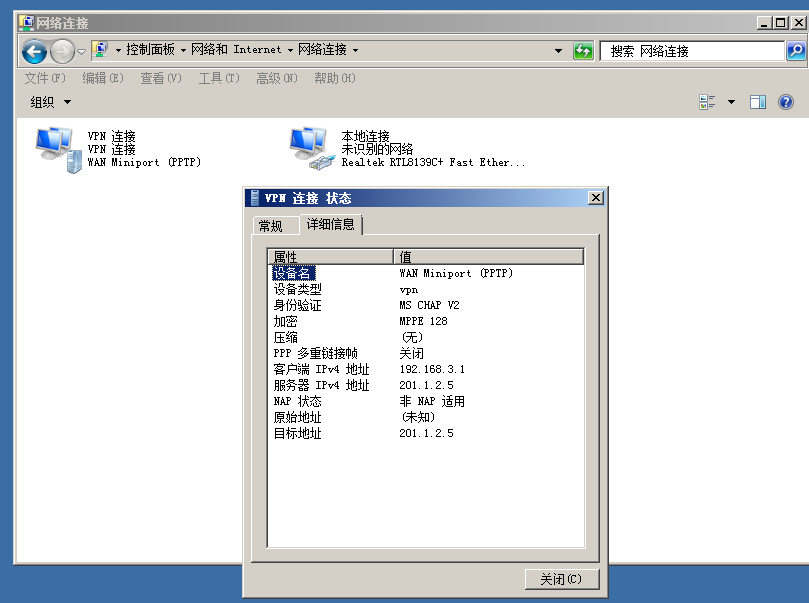
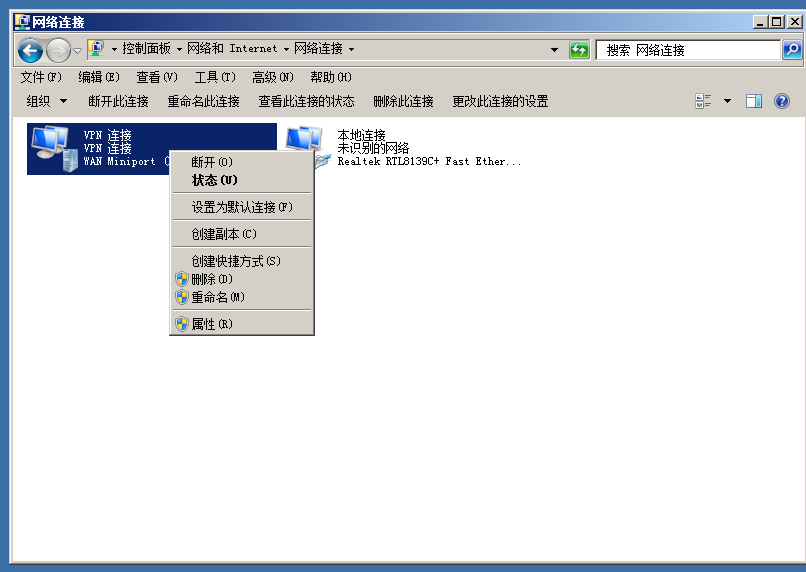
## 步骤二：客户端设置

启动一台Windows虚拟机，将虚拟机网卡桥接到public2，配置IP地址为201.1.2.20。

新建网络连接（具体操作如图-3所示），输入VPN服务器账户与密码（具体操作如图-4所示），连接VPN并测试网络连通性（如图-5所示）。







设置中:域可以

Windows ping vpn的服务器:通了实验成功

Ping 192.168.4.5

Ping 192.168.2.5

# 3 案例3：创建L2TP+IPSec VPN

* L2TP建立主机之间的vpn隧道，压缩、验证
* IPSec提供数据加密，数据校验，访问控制的功能

3.1 问题

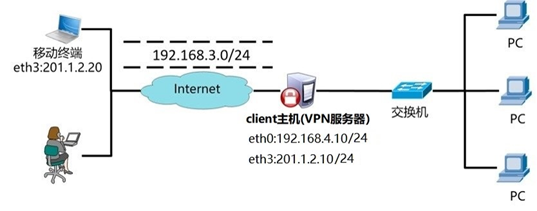
本案例要求搭建一个L2TP+IPSec VPN环境，并测试该VPN网络是否能够正常通讯，具体要求如下：

* 使用L2TP协议创建一个支持身份验证与加密的隧道连接（与pptp类似）
* 使用IPSec对数据进行加密
* 为客户端分配192.168.3.0/24的地址池
* 客户端连接的用户名为：jacob，密码为：123456
* 预共享密钥为：randpass

3.2 方案

准备实验所需的虚拟机环境，实验环境所需要的主机及对应的IP设置列表如表-3所示，正确配置IP地址、主机名称，并且为每台主机配置YUM源。





## 步骤一：部署IPSec服务

### 1）安装软件包

[root@client ~]# yum -y install libreswan (光盘自带的)

### 2)新建IPSec密钥验证配置文件

[root@client ~]# cat /etc/ipsec.conf //仅查看一下该主配置文件

.. ..

include /etc/ipsec.d/\*.conf //加载该目录下的所有配置文件

[root@client ~]# vim /etc/ipsec.d/myipsec.conf

//新建myipsec.conf 文件，参考lnmp\_soft/vpn/myipsec.conf

conn IDC-PSK-NAT

rightsubnet=vhost:%priv //允许建立的VPN虚拟网络

also=IDC-PSK-noNAT

conn IDC-PSK-noNAT

authby=secret //加密认证

ike=3des-sha1;modp1024 //算法

phase2alg=aes256-sha1;modp2048 //算法

pfs=no

auto=add

keyingtries=3

rekey=no

ikelifetime=8h

keylife=3h

type=transport

left=201.1.2.10 //重要，服务器本机的外网IP

leftprotoport=17/1701

right=%any //允许任何客户端连接

rightprotoport=17/%any

### 3)创建IPSec预定义共享密钥(PSK)

[root@client ~]# cat /etc/ipsec.secrets //仅查看，不要修改该文件

include /etc/ipsec.d/\*.secrets

[root@client ~]# vim /etc/ipsec.d/mypass.secrets //需新建该文件

201.1.2.10 %any: PSK "randpass" //randpass为预共享密钥

//201.1.2.10是VPN服务器的IP

### 4)启动IPSec服务

[root@client ~]# systemctl start ipsec

[root@client ~]# netstat -ntulp |grep pluto

udp 0 0 127.0.0.1:4500 0.0.0.0:\* 3148/pluto

udp 0 0 192.168.4.10:4500 0.0.0.0:\* 3148/pluto

udp 0 0 201.1.2.10:4500 0.0.0.0:\* 3148/pluto

udp 0 0 127.0.0.1:500 0.0.0.0:\* 3148/pluto

udp 0 0 192.168.4.10:500 0.0.0.0:\* 3148/pluto

udp 0 0 201.1.2.10:500 0.0.0.0:\* 3148/pluto

udp6 0 0 ::1:500 :::\* 3148/pluto

## 步骤二：部署XL2TP服务

### 1）安装软件包（软件包参考lnmp\_soft）

[root@client ~]# yum localinstall xl2tpd-1.3.8-2.el7.x86\_64.rpm

### 2) 修改xl2tp配置文件（修改3个配置文件的内容）

[root@client ~]# vim /etc/xl2tpd/xl2tpd.conf //修改主配置文件

[global]

.. ..

[lns default]

.. ..

ip range = 192.168.3.128-192.168.3.254 //分配给客户端的IP池

local ip = 201.1.2.10 //VPN服务器的IP地址

.. ..

[root@client ~]# vim /etc/ppp/options.xl2tpd //认证配置

require-mschap-v2 #去掉注释，强制要求认证

#crtscts //注释或删除该行,不注释启动报错

#lock //注释或删除该行,不住事启动将报错

root@client ~]# vim /etc/ppp/chap-secrets //修改密码文件

jacob \* 123456 \* //账户名称 服务器标记 密码 客户端IP

### 3）启动服务

[root@client ~]# systemctl start xl2tpd

[root@client ~]# netstat -ntulp |grep xl2tpd

udp 0 0 0.0.0.0:1701 0.0.0.0:\* 3580/xl2tpd

### 4）设置路由转发，防火墙

[root@client ~]# echo "1" > /proc/sys/net/ipv4/ip\_forward

[root@client ~]# firewall-cmd --set-default-zone=trusted

### 5）翻墙设置（非必需操作）

[root@client ~]# iptables -t nat -A POSTROUTING -s 192.168.3.0/24 -j SNAT --to-source 201.1.2.10

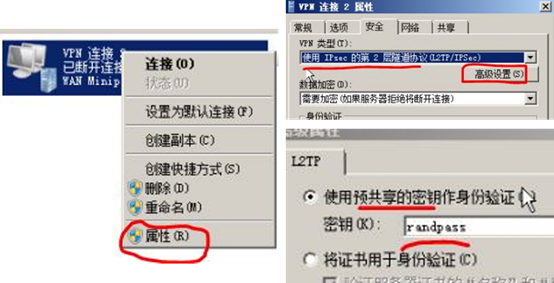
## 步骤二：客户端设置

启动一台Windows虚拟机，将虚拟机网卡桥接到public2，配置IP地址为201.1.2.20。

### 1) 新建网络连接（参考案例2的客户端设置），

输入VPN服务器账户与密码（参考案例2）。

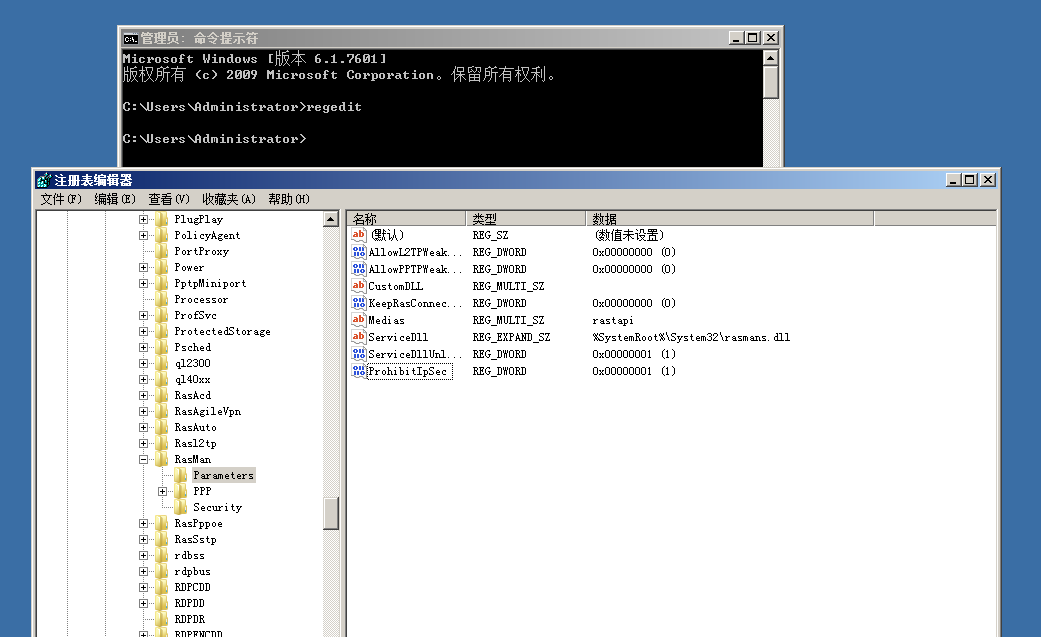
设置VPN连接的属性，预共享密钥是IPSec配置文件中填写的randpass，具体操作如图-7所示。



### 设置Windows注册表

（不修改注册表，连接VPN默认会报789错误），具体操作如下：

* 单击"开始"，单击"运行"，键入"regedit"，然后单击"确定"
* 找到下面的注册表子项，然后单击它(win8以及以上版本不需要修改,BUG被修复)：
* HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\Rasman\Parameters
* 在"编辑"菜单上，单击"新建"->"DWORD值"
* 在"名称"框中，键入"ProhibitIpSec",注意大小写ProhibitIpSec中的IpSec不是L
* 在"数值数据"框中，键入"1"，然后单击"确定"



* 退出注册表编辑器，然后重新启动计算机
* 连接VPN并测试网络连通性（参考案例2）。

# 4 案例4：NTP时间同步

4.1 问题

本案例要求搭建一个NTP服务器，为整个网络环境中的所有主机提供时间校准服务，具体要求如下：

部署一台NTP时间服务器

设置时间服务器上层与0.centos.pool.ntp.org同步

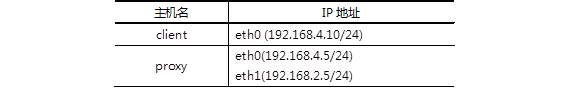
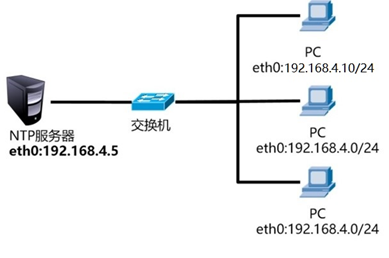
设置本地服务器层级数量为10

允许192.168.4.0/24网络的主机同步时间

客户端验证时间是否同步

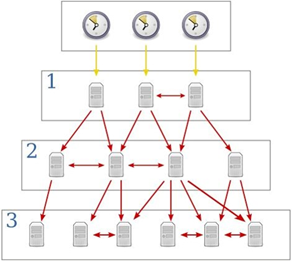
4.2 方案

准备实验所需的虚拟机环境，实验环境所需要的主机及对应的IP设置列表如表-4所示，正确配置IP地址、主机名称，并且为每台主机配置YUM源。

Network Time Protocol（网络时间协议）采用的是分层设计，如图-9所示，Stratum层的总数限制在15以内（包括15）。

原子时间:国家服务器的



## 步骤一：部署NTP服务

### 1）安装软件包

[root@proxy ~]# yum -y install chrony

[root@proxy ~]# rpm -qc chrony //查看配置文件列表

/etc/chrony.conf

/etc/chrony.keys

.. ..

### 2) 修改配置文件

[root@proxy ~]# cat /etc/chrony.conf

.. ..

server 0.centos.pool.ntp.org iburst //server用户客户端指向上层NTP服务器

allow 192.168.4.0/24 //允许那个IP或网络访问NTP

#deny 192.168.4.1 //拒绝那个IP或网络访问NTP

local stratum 10 //设置NTP服务器的层数量

.. ..

### 3) 启动NTP服务

[root@proxy ~]# systemctl restart chronyd

[root@proxy ~]# systemctl enable chronyd

### 4) 设置防火墙

[root@proxy ~]# firewall-cmd --set-default-zone=trusted

## 步骤二：配置客户端

### 1）安装软件包

[root@client ~]# yum -y install chrony

### 2) 修改配置文件

[root@client ~]# vim /etc/chrony.conf

server 192.168.4.5 iburst //设置与哪台服务器同步数据

//iburst参数设置重启服务后尽快同步时间

### 3) 将客户端时间修改为错误的时间

[root@client ~]# date -s "hour:minute" //调整时间（小时：分钟）

[root@client ~]# date //查看修改后的时间

### 4) 重启chrony与服务器同步时间

[root@client ~]# systemctl restart chronyd

### 5) 确认时间是否已经同步

[root@client ~]# date //多执行几次查看结果

# 5 案例5：pssh远程套件工具

先得有ssh,,pssh用Python写的

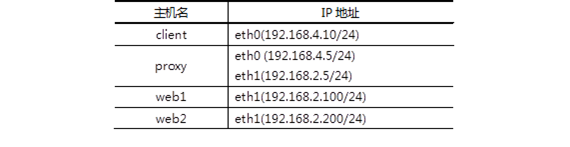
5.1 问题

本案例要求使用pssh套件工具并发远程其他主机，具体要求如下：

* 使用密码批量、多并发远程其他主机
* 使用密钥批量、多并发远程其他主机
* 批量、多并发拷贝数据到其他主机
* 批量、多并发从其他主机下载数据到本机
* 批量、多并发杀死其他主机的进程

5.2 方案

准备实验所需的虚拟机环境，实验环境所需要的主机及对应的IP设置列表如表-5所示，正确配置IP地址、主机名称，并且为每台主机配置YUM源。



安装pssh后会提供如下命令：

/usr/bin/pnuke

/usr/bin/prsync

/usr/bin/pscp.pssh

/usr/bin/pslurp

/usr/bin/pssh

## 步骤一：准备工作

### 1）安装软件包

[root@proxy ~]# rpm -ivh pssh-2.3.1-5.el7.noarch.rpm

### 2) 修改/etc/hosts本地解析文件

[root@proxy ~]# vim /etc/hosts

… …

192.168.2.100 host1

192.168.2.200 host2

192.168.4.10 host3

… …

### 3) 创建主机列表文件

[root@proxy ~]# vim /root/host.txt //每行一个用户名、IP或域名

… …

root@host1

host2

host3

… …

## 步骤二：使用密码批量、多并发远程其他主机

### 1）语法格式

[root@proxy ~]# man pssh //通过man帮助查看工具选项的作用

pssh提供并发远程连接功能

-A 使用密码远程其他主机（默认使用密钥）

-i 将输出显示在屏幕

-H 设置需要连接的主机

-h 设置主机列表文件

-p 设置并发数量

-t 设置超时时间

-o dir 设置标准输出信息保存的目录

-e dir 设置错误输出信息保存的目录

-x 传递参数给ssh

### 2) 使用密码远程多台主机执行命令，屏幕显示标准和错误输出信息

[root@proxy ~]# pssh -i -A -H 'host1 host2 host3' -x '-o StrictHostKeyChecking=no' echo hello

Warning: do not enter your password if anyone else has superuser

privileges or access to your account.

Password:

[1] 14:18:09 [SUCCESS] host3

hello

Stderr: Warning: Permanently added 'host3,192.168.4.10' (ECDSA) to the list of known hosts.

[2] 14:18:09 [SUCCESS] host2

hello

Stderr: Warning: Permanently added 'host2,192.168.2.200' (ECDSA) to the list of known hosts.

[3] 14:18:09 [SUCCESS] host1

hello

Stderr: Warning: Permanently added 'host1,192.168.2.100' (ECDSA) to the list of known hosts.

### 3) 使用密码远程多台主机执行命令不显示标准和错误输出信息，通过读取host.txt读取主机信息(第一次远程连接必须先执行上一步)

[root@proxy ~]# pssh -A -h host.txt echo hello

Warning: do not enter your password if anyone else has superuser

privileges or access to your account.

Password:

[1] 14:18:21 [SUCCESS] host2

[2] 14:18:21 [SUCCESS] root@host1

[3] 14:18:21 [SUCCESS] host3

## 步骤三：使用密钥批量、多并发远程其他主机

### 1）生成密钥并发送密钥到其他主机(无密码验证远程)

[root@proxy ~]# ssh-keygen -N ' ' -f /root/.ssh/id\_rsa //非交互生成密钥文件

-N 设置密码 -f 指定秘钥路径

Generating public/private rsa key pair.

Your identification has been saved in /root/.ssh/id\_rsa.

Your public key has been saved in /root/.ssh/id\_rsa.pub.

The key fingerprint is:

SHA256:Shxehw6Cn0oxlXbSB9FgxX8vtkABb1fzm3oRkcrdeLw root@proxy

The key's randomart image is:

+---[RSA 2048]----+

| .o=Bo. o..|

| o+.o +o. . +.|

| +..oo.o.+.o o+o|

| + = = oo..o..B|

| . o + S. . . =.|

| . . . . . o oE.|

| . . o + . |

| . . |

| |

+----[SHA256]-----+

[root@proxy ~]# ssh-copy-id host1 #发送秘钥至host1

/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id\_rsa.pub"

/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed

/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys

root@host1's password: #输入root 密码

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'host1'"

and check to make sure that only the key(s) you wanted were added.

[root@proxy ~]# ssh-copy-id host2

[root@proxy ~]# ssh-copy-id host3

### 2) 使用密钥远程其他主机

[root@proxy ~]# pssh -h host.txt echo hello

### 3) 使用密钥远程其他主机，将标准输出信息写入到/tmp目录

[root@proxy ~]# pssh -h host.txt -o /tmp/ echo hello

## 步骤四：批量、多并发拷贝数据到其他主机

### 1）语法格式

[root@proxy ~]# man pscp.pssh //通过man帮助查看工具选项的作用

pscp.pssh提供并发拷贝文件功能

-r 递归拷贝目录

其他选项基本与pssh一致

### 2)将本地的/etc/hosts拷贝到远程主机的/tmp目录下

[root@proxy ~]# pscp.pssh -h host.txt /etc/hosts / tmp

### 3)递归将本地的/etc目录拷贝到远程主机的/tmp目录下

[root@proxy ~]# pscp.pssh -r -h host.txt /etc /tmp

## 步骤五：批量、多并发从其他主机下载数据到本机

### 1）语法格式

[root@proxy ~]# man pslurp //通过man帮助查看工具选项的作用

pslurp提供远程下载功能

选项与pscp.pssh基本一致

### 2)将远程主机的/etc/passwd，拷贝到当前目录下，存放在对应IP下的pass文件中

[root@proxy ~]# pslurp -h host.txt /etc/passwd /pass

注意：最后的pass是文件名

### 3)将远程主机的/etc/passwd目录，拷贝到media下，存放在对应IP下的pass文件

[root@proxy ~]# pslurp -h host.txt -L /media /etc/passwd /pas

## 步骤六：批量、多并发杀死其他主机的进程

### 1）语法格式

[root@proxy ~]# man pnuke //通过man帮助查看工具选项的作用

pnuke提供远程杀死进程的功能

选项与pssh基本一致

### 2)将远程主机上的sleep进程杀死

[root@proxy ~]# pnuke -h host.txt sleep

### 3)将远程主机上的test相关脚本都杀死（如：test1,testtt,test2等等）

[root@proxy ~]# pnuke -h host.txt test

### 4)将远程主机上的test.sh脚本杀死

[root@proxy ~]# pnuke -h host.txt test.sh