Operation的复习

Fastcgi 用于php上

CGI的解释

比如，如果请求/index.html，那么web server会去文件系统中找到这个文件，发送给浏览器，这里分发的是静态数据。 如果现在请求的是/index.php，根据配置文件，nginx知道这个不是静态文件，需要去找PHP解析器来处理，那么他会把这个请求简单处理后交给PHP解析器。Nginx会传哪些数据给PHP解析器呢？url要有吧，查询字符串也得有吧，POST数据也要有，HTTP header不能少吧…… CGI就是规定要传哪些数据、以什么样的格式传递给后方处理这个请求的协议。 当web **server收到/index.php这个请求后，会启动对应的CGI程序，这里就是PHP的解析器。**接下来PHP解析器会解析php.ini文件，初始化执行环境，然后处理请求，再以规定CGI规定的格式返回处理后的结果，退出进程。web server再把结果返回给浏览器。 3、CGI是个协议，跟进程什么的没关系。

**cgi 负责web server发过来的php解释工作**

Fastcgi的解释

1**、提高性能**，那么CGI程序的性能问题在哪呢？”PHP解析器会解析php.ini文件，初始化执行环境”，就是这里了。标准的CGI对每个请求都会执行这些步骤，所以处理每个时间的时间会比较长。这明显不合理嘛！

2、Fastcgi是怎么做的呢？首先，Fastcgi会先启一个master，解析配置文件，初始化执行环境，然后再启动多个worker。当请求过来时，master会传递给一个worker，然后立即可以接受下一个请求。这样就避免了重复的劳动，效率自然是高。而且当worker不够用时，master可以根据配置预先启动几个worker等着；当然空闲worker太多时，也会停掉一些，这样就提高了性能，也节约了资源。这就是fastcgi的对进程的管理。

http、nginx、tomcat 该如何选择

Apache一个连接对应一个进程，apache非常的稳定，对php的支持简单，nginx需要配合其他后端用

Nginx可以多个进程对应一个进程，并发亮大，处理静态文件好，本身不具备动态解析的功能，需要配置其他插件或通过其他软件协同才具备动态功能。（基础并发几万）

Tomcat需要配置JDK的支持（基础并发100）tomcat更多用来做做一个应用容器，让java web app跑在里面的东西（java和php类似都是语言），通常用在内网和不需要流控等小型服务的场景。

**nginx的模块**

--with-http\_ssl\_module //开启SSL加密功能

--with-stream                                 //开启4层反向代理功能

--with-http\_stub\_status\_module                //开启status状态页面

Nginx用户的认证 装包httpd-tools

location / {

auth\_basic “auth-domain”

Auth\_basic\_user\_client\_file /usr/local/nginx/pass

Htpasswd -c /usr/local/nginx/pass

**Nginx-http\_ssl\_module 加密的时候密钥文件要放在conf配置文件下**

Lnmp的数据库安装yum -y install mariadb mariadb-server mariadb-devel

开启php 功能 php、php-mysql、php-fpm

启动php的功能要开启fastcgi\_pass 127.0.0.1：9000 //php-fpm解析的IP

fastcgi\_index index.php;

include fastcgi index.cgi;

Fsatcgi的内存消耗是nginx的一倍.

FastCGI是用来提高CGI性能的，FastCGI每次处理完请求之后不会关闭掉进程。而是保留这个进程，使这个进程可以处理多个请求。这样的话每个请求都不用再重新创建一个进程了。大大提升了处理效率。

地址重写

rewrite /a.html /b.html redirect;    转换地址,本来地址栏是不会变的

rewrite ^/ http://www.tmooc.cn/; http的前面是没有/号的哟.

rewrite \.(gif|jpg)$ /logo.png

rewrite ^/(.\*) http://bbs.tarena.com/$1;

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 转换地址

permament 永久重定向

**Nginx 4层，后面少写了分号，改了半天！！**

upstream wenserver {

server 192.168.1.12:80;

server 192.168.1.13:80;

}

nginx的反向代理(7层)

http {

upstream webserver {

server 192.168.2.100:80;

server 192.168.2.200:80;

}

....

location / {

#通过proxy\_pass将用户的请求转发给webserver集群

proxy\_pass http://webserver;

}

}

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

#weight设置服务器权重值，默认值为1

#max\_fails设置最大失败次数

#fail\_timeout设置失败超时时间，单位为秒

#down标记服务器已关机，不参与集群调度

设置相同客户端访问相同Web服务器

http {

  ip\_hash;

server 192.168.2.100 weight=1 max\_fails=2 fail\_timeout=10;

server 192.168.2.200 weight=2 max\_fails=2 fail\_timeout=10;

Server {

}

}

nginx的4层解析首先需要的是--with-stream模块

stream {

        upstream backend {

    server 192.168.2.100:22;            //后端SSH服务器的IP和端口

    server 192.168.2.200:22;

}

Server {

..}

http {

1.优化nginx高并发测试

ab -n 2000 -c 2000 http://192.168.4.5 测试并发

@1.修改nginx配置文件，增加并发量

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

worker\_processes 2;                    //与CPU核心数量一致,进程数量

worker\_connections 65535;        //每个worker最大并发连接数

@2.优化linux内存参数

[root@proxy ~]# ulimit -a                        //查看所有属性值

[root@proxy ~]# ulimit -Hn 100000                //设置硬限制（临时规则）

[root@proxy ~]# ulimit -Sn 100000                //设置软限制（临时规则）

[root@proxy ~]# vim /etc/security/limits.conf //永久设置

    .. ..

\* soft nofile 100000

\* hard nofile 100000

测试优化 ab -n 2000 -c 2000 http://192.168.4.5 xxxx

[root@proxy ~]# ab -n 2000 -c 2000 http://192.168.4.5/ 有/号才对

n 总的次数 c 并发数

1. 自定义报错信息

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

error\_page 404 /40x.html;    //自定义错误页面



1. 如何查看服务器状态信息（非常重要的功能）

需要安装--with-http\_stub\_status\_module开启状态页面模块

location /status {

stub\_status on;

                 #allow IP地址;

Active connections: 1 实时的并发量

server accepts handled requests

52 52 60 //总共52次连接，创建了52次握手（**处理了多少，设置了并发量安全才会和总连接不一样。**），总共处理了60个请求（**keepalive，一次连接多次请求**）

Reading: 0 Writing: 1 Waiting: 0

　　每个连接有三种状态waiting、reading、writing

　　reading —读取客户端的Header信息数.这个操作只是读取头部信息，读取完后马上进入writing状态，因此时间很短。

　　writing — 响应数据到客户端的Header信息数.这个操作不仅读取头部，还要等待服务响应，因此时间比较长。

　　waiting — 开启keep-alive后等候下一次请求指令的驻留连接.

　　正常情况下waiting数量是比较多的，并不能说明性能差。反而如果reading+writing数量比较多说明服务并发有问题。

1. 优化nginx数据的包头缓存

http {

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

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

1. 浏览器本地缓存静态数据

Server {

}

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

expires        30d;            //定义客户端缓存时间为30天

6.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）

kill USR1 PID(nginx的进程PID号)

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

#!/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

1. 对页面进行压缩

http {

gzip on;                            //开启压缩

gzip\_min\_length 1000;                //小文件不压缩，压缩反而会变大

gzip\_comp\_level 4;                //压缩比率，数值越大越慢，多媒体本身就是压缩格式。特别大的文件不适合压缩，特别满。

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

8.服务器内存缓存

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

}



　　每个连接有三种状态waiting、reading、writing

　　reading —读取客户端的Header信息数.这个操作只是读取头部信息，读取完后马上进入writing状态，因此时间很短。

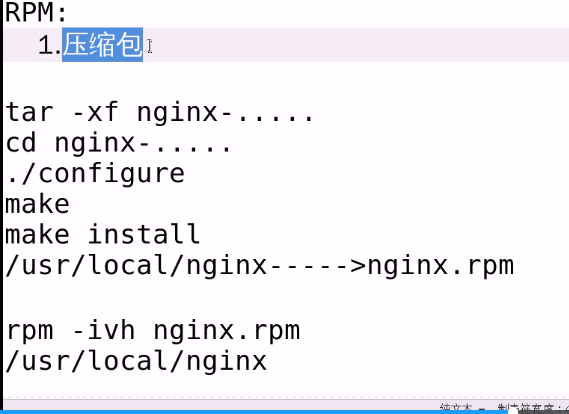
　　writing — 响应数据到客户端的Header信息数.这个操作不仅读取头部，还要等待服务响应，因此时间比较长。

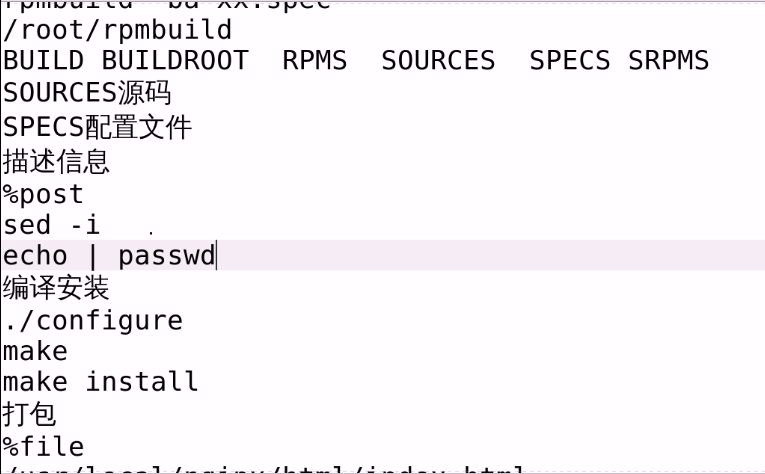
　　waiting — 开启keep-alive后等候下一次请求指令的驻留连接.

　　正常情况下waiting数量是比较多的，并不能说明性能差。反而如果reading+writing数量比较多说明服务并发有问题。

**RPM是一个压缩包！！! Rpm2cpio解压（）。**

**什么文件移动到什么目录早就在压缩源码包的时候就做完了.**





rpm -ql httpd 看所有的文件

rpm -ql httpd 不安装就可以看到里面有什么内容

rpm -qi(info) 简介

1.装包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

这个文件写的是什么，rpm -qi显示的就是什么。

[root@web1 ~]# ls /root/rpmbuild                    //自动生成的目录结构

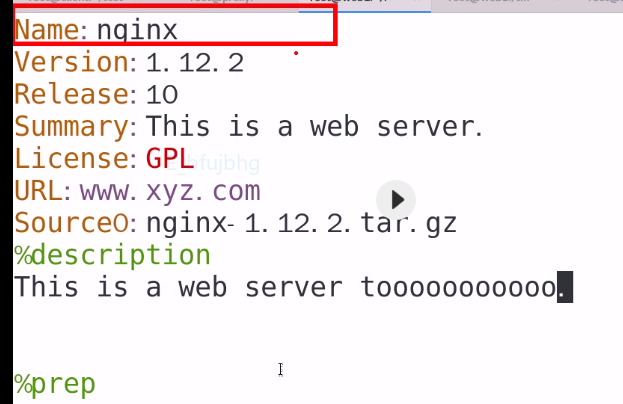
BUILD BUILDROOT RPMS SOURCES SPECS SRPMS

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/

1. 创建并修改SPEC配置文件

**描述信息**



**操作信息**

%prep

%setup -q **%prep 自动解包！！！！,这行不能改**

%build

./configure .....(with-http\_ssl\_module --with-stream --prefix=/usr/local/..)

Make %{ ?\_smp\_mflags}

%install

make install DESTDIR=%{buildroot}

//本机的RPM包在/root/rpmbuild/buildroot下

%files

%doc

/usr/local/nginx/\*

%changelog

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 zlib-devel openssl-devel

2）rpmbuild创建RPM软件包

[root@web1 ~]# rpmbuild -ba /root/rpmbuild/SPECS/nginx.spec

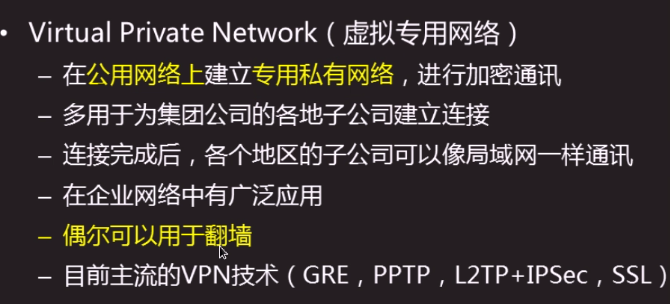
SOURCES 源码的位置

SPECS 配置路径

BUILDROOT 本机安装的路径

RPMS 打好包放置的位置

VPN  **借助于公有IP才能实现！@！！！！！**



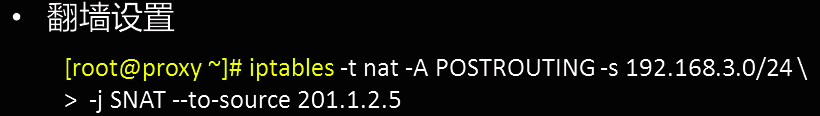
**GRE VPN WINDOW不支持**

**配置PPTP**

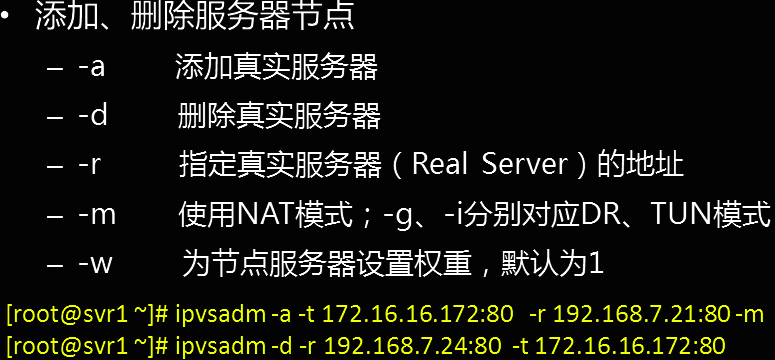


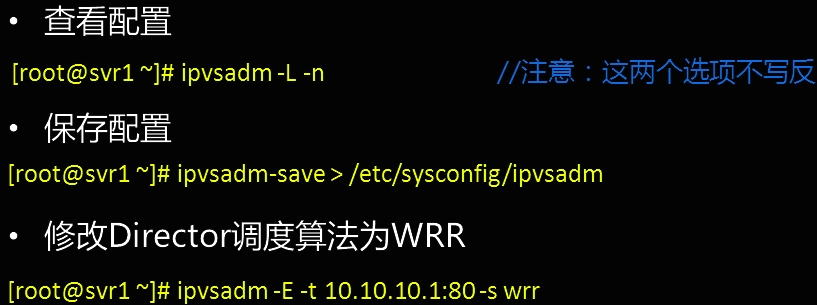


systemctl restart pptpd



LVS

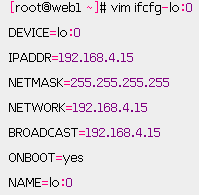


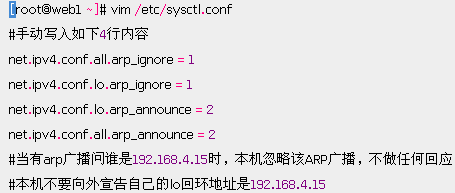


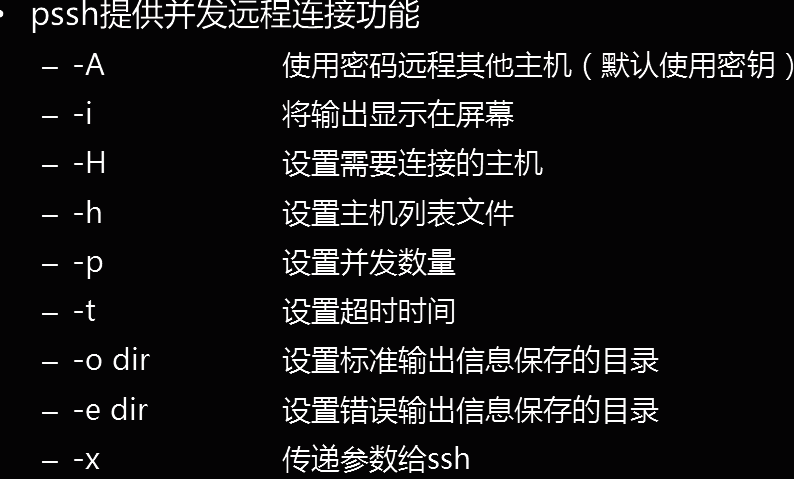
**Ipvsadm -S 保存**

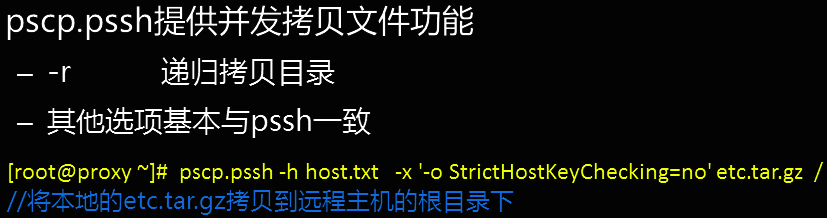
**NAT模式需要在真实服务器上绑定网关，开启路由转发功能**

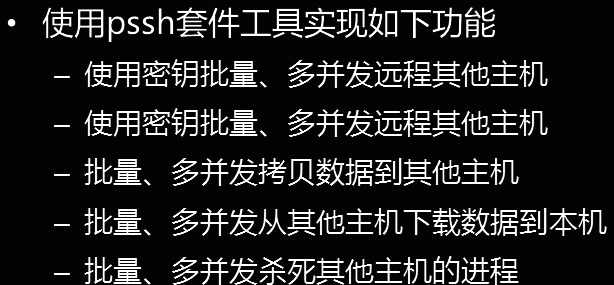
**DR模式需要在真实服务器上搭建虚拟IP，开启路由转发并防止网络地址起冲突。**











1． Ehcache

在java项目中应用非常广泛, 主要面向通用缓存,J2EE和轻量级容器。一级缓存在内存，二级缓存在磁盘。

优点：

1. 直接在jvm虚拟机中缓存，速度快，效率高；

2. 小巧，使用简单；

缺点：

缓存共享麻烦，不易维护；集群分布式应用不方便。

应用场景：

单个应用或者对缓存访问要求很高的应用。

2． Varnish

Directors:

varnish支持负载均衡，其中有random,client.hash,round-robin,dns,fallback等算法

有电镀的功能

高性能、开源的反向代理服务器和内存缓存服务器。

优点：

1. 高性能；

2. 多核支持；

3. 支持0-60秒的精确缓存时间。

缺点：

1. 不具备自动容错和恢复功能，重启后数据丢失；

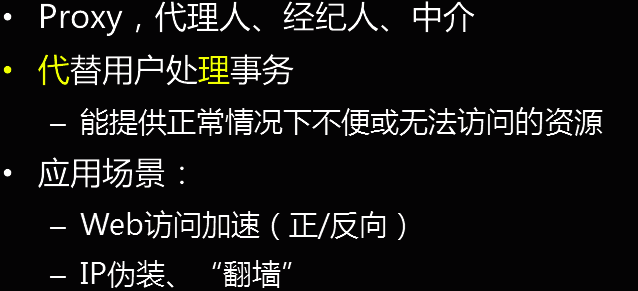
2. 在线扩容比较难。

3. 32位机器上缓存文件大小为最大2GB；

4. 不支持集群。

应用场景：

并发要求不是很大的小型系统和应用。



3． Memcache

一个高性能的分布式内存对象缓存系统。通过key-value形式缓存。

优点：

1. 高性能；

2. 多线程支持，安装简单；

3. 部分容灾；

4. 均衡请求。

缺点：

1. 不能持久化存储；

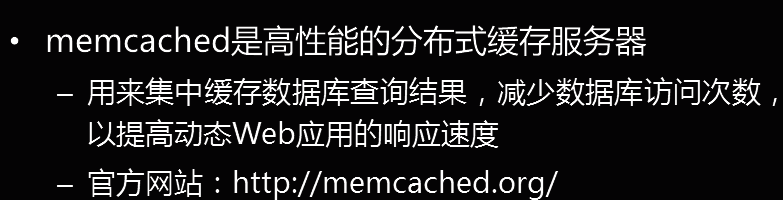
2. 存储数据有限；

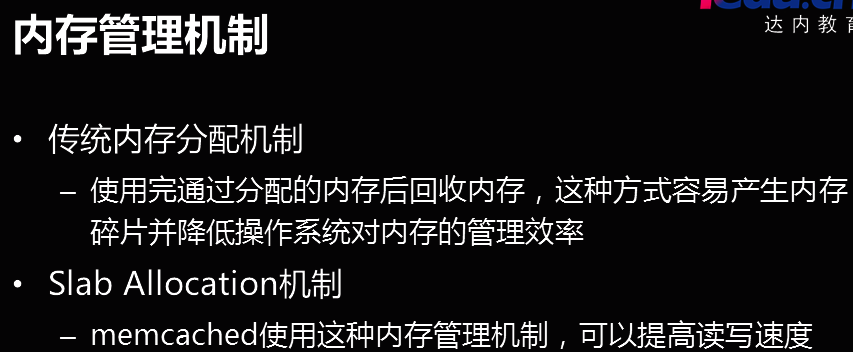
3. 集群数据没有复制和同步机制；

4. 内存回收不能及时。

应用场景：

大型系统，缓存共享、分布式部署和集群应用。





4． Redis

基于内存的键-值存储数据库。

优点：

1. 高性能；

2. 丰富的数据类型；

3. 支持持久化；

4. 支持主从复制。

缺点：

1. 不具备自动容错和恢复功能；

2. 在线扩容比较难；

3．不支持多核多线程处理。

应用场景：

大型系统，缓存共享、分布式部署和集群应用。

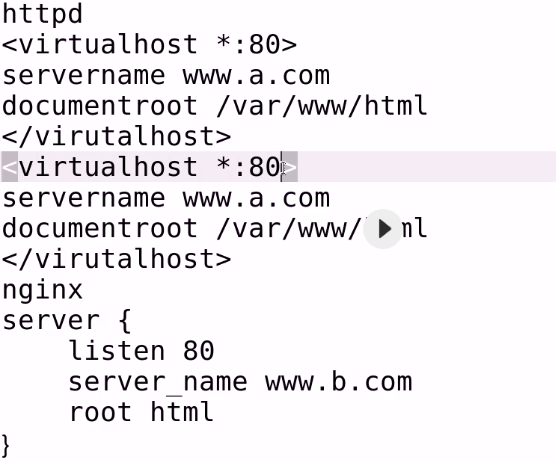


**redis加强版memcached，本来缓存数据库都存在内存里，但是redis隔几秒钟会将内存里的东西存到硬盘中！！！！**

百度搜索网站演变

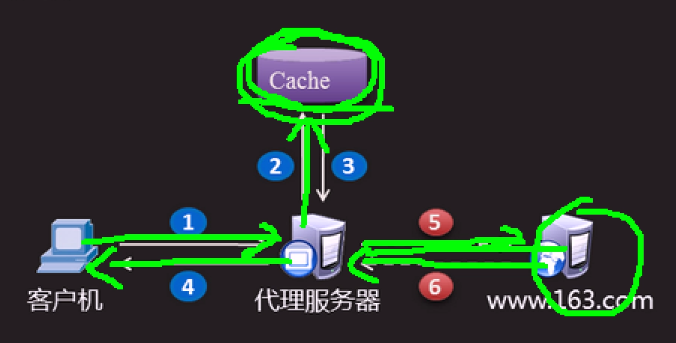
数据的垂直拆分水平拆分。

**例如点击京东，每个按钮就是一个集群（域名变了），不会有单点故障。**



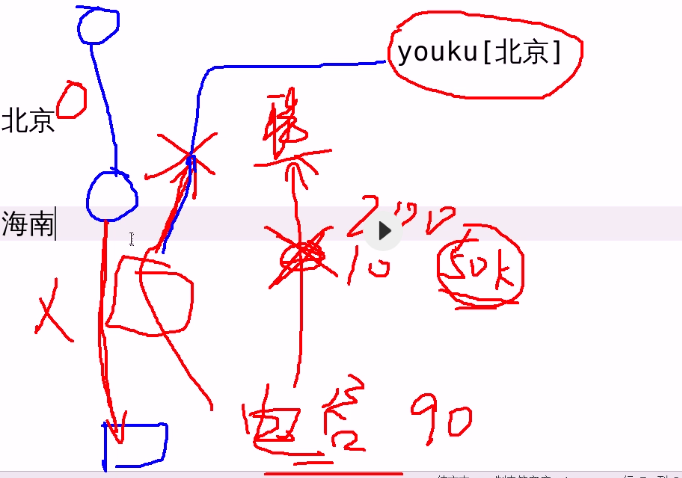
**http、nginx和tomcat有区别，nginx和httpd的端口和域名都是绑定的！！！Tomcat只是相当于一个大门，你可以开多个门，由engine进行请求的分配。**

**varnish**



第一次速度一样，但是后面的速度非常的快。

南北通信图

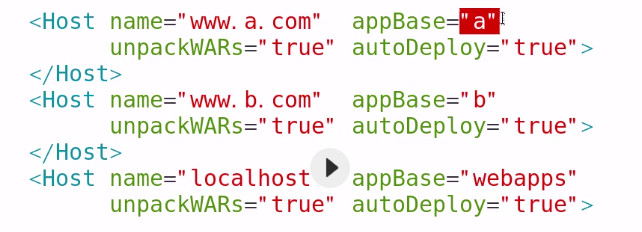


**varnish就是cnd的一部分阿，就是cdn中的代理服务器**

memcache只是在机器上的，不是代理服务器阿，区别很大

Tomcat只是java的一个脚本

虚拟主机



mkdir -p /usr/local/tomcat/{a,b}/ROOT //目录下还应该有个ROOT

创建加密用的私钥和证书文件

[root@web1 ~]# keytool -genkeypair -alias tomcat -keyalg RSA -keystore /usr/local/tomcat/keystore                //提示输入密码为:123456

==精确比较

～正则模糊比较