什么叫阻塞？一个进程发出的IO请求没有得到响应 ----- 向内核发出系统调用的请求 ---- 然后内核把数据复制到自己的缓冲区 ---- 然后内核把数据从自己的缓冲区复制到用户的内存空间 ---- 在数据顺备好的过程中，这个请求IO的进程必须等待 ---- 这个就叫做阻塞

什么叫非阻塞？向内核发出IO系统调用之后 不在那等待 而是一遍遍过来轮询 ---- 非阻塞性能很差

什么叫同步？ 什么叫异步？

通信中的同步：指的是 发一个数据 必须等到对方确认 并且完整保存 然后接收到对方的反馈信号 这个叫同步

异步就是把数据丢出去就不管了 这个叫异步

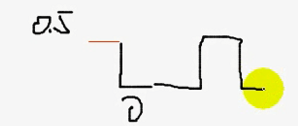
---- 对MySQL复制 也有同步和异步

Mysql的复制异步 ----- 只是把数据存储到二进制 什么时候 发到节点不知道

硬件方面 同步 就是双方必须同步时钟信号 异步就是双方不必同步时钟信号



计算机通信无非就是发的是高电平还是低电平



问后面是几个1？----- 所以 就是必须要在时钟频率上面取得同步 才知道后面有几个1 才能理解通信数据的含义

比如时钟频率是1GHz 这就意味着的长度的信号是一个单位

这样 有多少个 就知道多少个1

IP报文：为了实现异步通信 前面就要加一个协议报文 这样就可以异步通信 告诉这个报文怎么解析

同步 异步主要来判定 通信双方在通信的过程中 是否要等待对方完完整整把数据接受下来 而等待对方向应的一种机制 ---这个是同步

异步就是把数据报文数据发出去 对方什么时候接收到 什么时候解析 都不管

同步和阻塞是两码事

同步阻塞

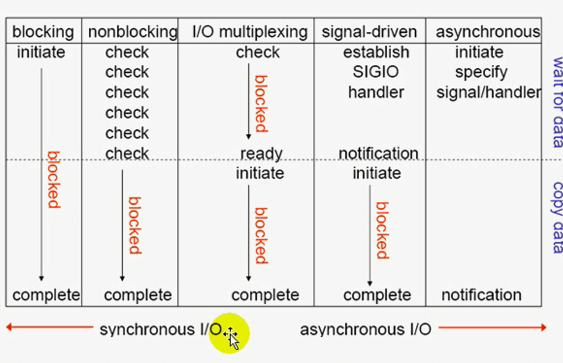
同步非阻塞性能太差 很少用到

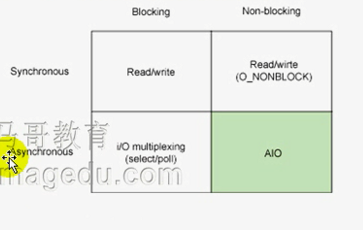
异步阻塞 --- IO复用

异步阻塞 ----

基于事件驱动的IO --- event-driven ---- 异步阻塞的一种 前半段阻塞 后半段非阻塞

异步非阻塞 ---- -AIO





nginx 几个重要特性 支持mmap event-driven 可以实现一个线程/进程响应多个请求

一个进程是单线程进程 ----- 所以对内存的占用量小的多得多

但是 apache是一个主进程 多个子进程

支持aio

Nginx和php结合 要给予 fpm的方式 --- 只能通过fastcgi与php通信

开始站点访问量小

用户对web的请求都到nginx

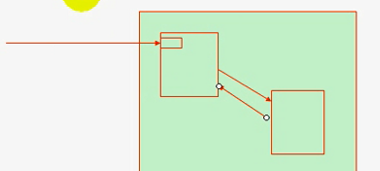
如果是php页面 都转发到php-fpm 协议是fastcgi

当php服务器处理完 把结果返回给nginx 最终有nginx吧数据封装成响应报文 返回给用户

每一个请求 进来了 nginx发现处理不了 交给php

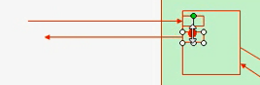
但是 当前的这个请求不能断开 意味这个nginx要维持一段内存空间 来保存这个链接请求的相关信息

很显然 用户请求发进来 是有请求报文 ---- 所以 这个报文也要暂时在报文中保存起来



Tcp链接 对方发 我们收 另一个对方收 我们发 ---- 这个两个通道 –一个是读管道 另一个写管道

当php响应完成之后 会把结果返回给nginx nginx会把数据封装成响应报文 这个过程 也会有一个缓冲区 ---- 这是两个缓冲区

 这都是需要内存来维持这些会话信息

当一个客户端连接进来 nginx发现是动态的 不是本地能处理的 ---- 向后转发 我们的客户端要处于等待状态

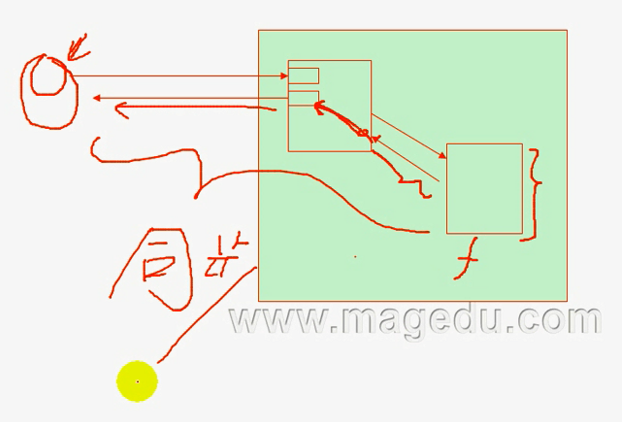
----- php将结果会送过来的时候 有两种情况

(1). Nginx是立即把fpm的报文响应给用户呢？----- 同步响应

(2). nginx会立刻把数据缓存到本地 然后把结果 封装成http响应报文 再响应给客户端？

如果是第一种(1) 也就是 你给我一位 我就立即放到发送缓冲区

第二种 就是 后端的的结果 通过fastcgi协议 --- 如果客户端可以理解fastcgi协议的话 ---- 后端php一旦和前端的nginx建立连接的时候 就发一位 传一位 意味着 我们客户端收到的报文 直接来自于fastcgi ----异步响应

 ---- 我们的nginx仅仅实现了透明转发

很显然 客户端很难理解fastcgi协议 ---- 所以 fpm把结果发送给nginx 然后nginx把这结果转换成客户端可以理解的http响应 ----- 客户端可以理解为http

所以 这个要暂存fpm的结果 这样 nginx和php这两台机器 就是异步

我把你的请求 接过来 发现不能处理 再给fpm 这个就是异步   
-------

如果php处理的结果 可以缓存在本地

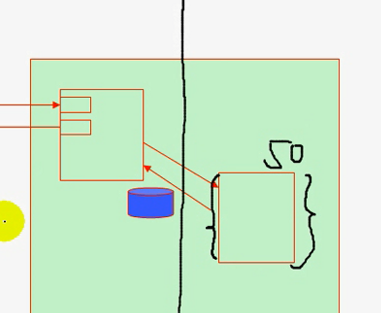
第一个用户请求的内容 对大多数用户来讲 都是一样的

第一个用户请求进来的时候 不是将用户请求转发到后端 而是检查本地缓存是否有？没有 再把请求报文 --- 把请求报文发送给后端php

第二个请求过来的时候 先查缓存 并且没过期 这个时候直接返回

通过异步模型+本地缓存

比如nginx有1K个链接 但是 fpm仅仅有50个

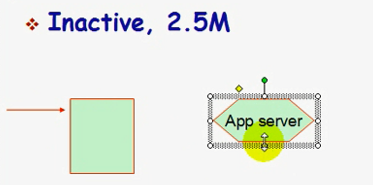
 ----让某一个模块仅仅做最擅长的事情

这样前端做nginx 后端叫app server

用户的请求到达nginx请求 ---- 所有请求都由nginx建立链接 ---- ngixn使用自己的内存来维持 --- nginx维持1w个内存 仅仅需要2.5M ----- 如果把请求直接交给app server 响应动态请求 ---- 这样每一个进程的开销是非常大的

这样就各司其职

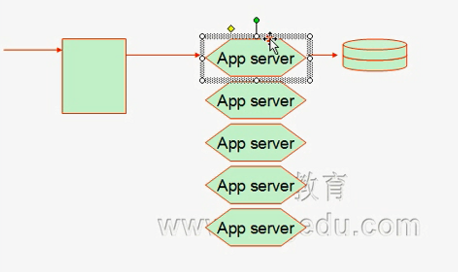
前端1K个请求 后端的每一个Server可能就是50K的链接



App Server中 莫非每一个程序都要处理数据么？

程序是指令+数据 ----- 后端还要依赖于数据库 用数据库来数据存储

App Server 和 数据库软件都是CPU密集型的 所以 随着链接的增长 5K --- > 分发到APP Server 有100个 ---- 还要查数据库 这个时候 扛不住 ----- 所以 App Server 和 数据库要进行分离 ----- 如果App Server和数据库在一台主机上面 这样网络开销小 传输速度快 ----- 如果在两台机器上面 传输速度会变慢 ----- 但是 前端5K请求 ---- App Server只能承受100个 但是 过来是 500个 ---- 所以 一台服务器不能满足500个请求 ---- 所以 App Server就要变成集群 ---- 所以 共享MySQL服务器



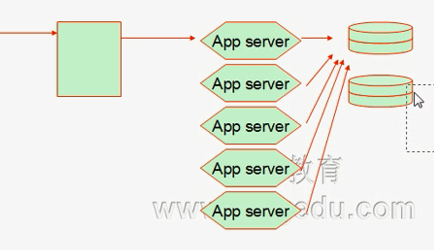
100个请求 30个需要数据库查询 70个需要本地执行逻辑

5台服务器要150个

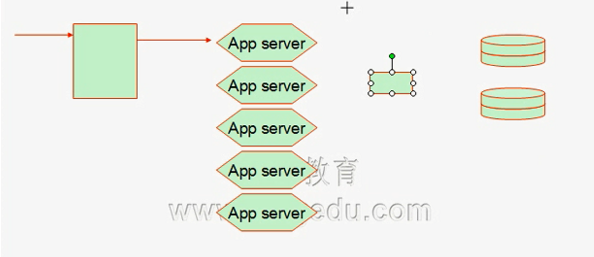
一个MySQL服务器 面临大量读写请求 50个每秒

所以 MySQL读写分离

一台服务器负责读 一台负责写



现在有一个交换机 所有App Server都练到这个交换机上面

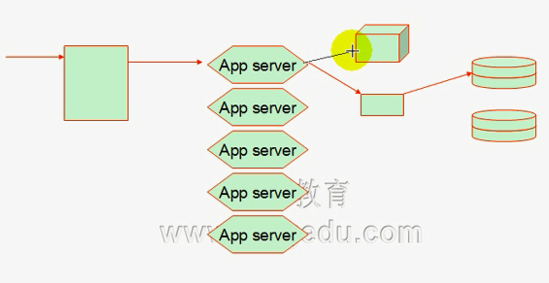


很多用户 进来之后 访问的页面都是差不多一样 很多数据 反反复复都要进行查询 可以将查询的数据 进行缓存 --- 可以使用memcached服务器

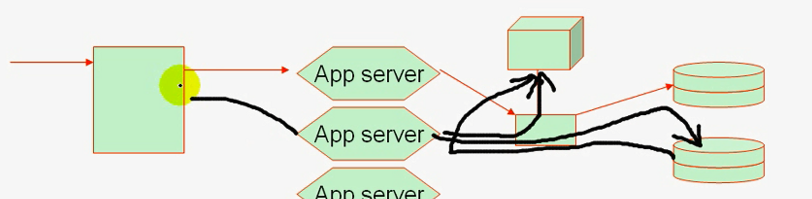
Memcached仅仅是提供缓存功能的服务器

但是 memcached到底缓不缓存 谁来缓存 怎么缓存 缓存什么时候过期 什么时候清除？

所以App Server 通过MySQL查询出来的数据 自身的程序决定是否要放到Memcached服务器中



查询的时候 首先到memcacehd中找



这就是一个公共缓存 很多节点都可以使用这个缓存

其实memcached这个组件可以大大提高我们的系统性能

----- **互联网时代 缓存为王**

如果命中率达到40% ---- 这样对整体性能的提升都是非常明显的

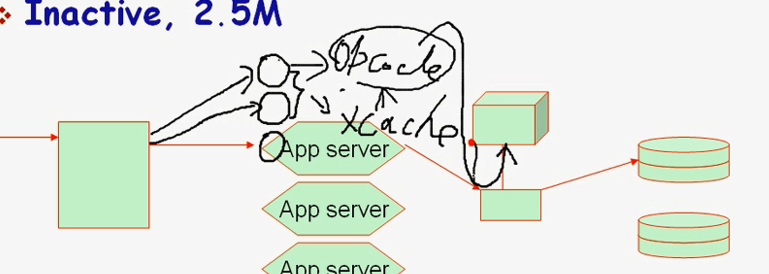
有了memcached 缓存的数据 都可以被多个app server使用

第一个请求 发到第一个app server --- 是php服务器 要编译成opcode 之后才能执行 -----第二个请求要是要编译成相同的opcode 然后才能执行 ---- app server 以fastcgi工作的时候 是为每一个链接生成一个fastcgi进程

因此第一请求用第一个进程响应 第二个请求 使用第二个fastcgi进程来响应 但是 不能使用第一个编译生成的opcode ----- 所以opcode无法再多进程之间缓存 ----- 所以使用xcache

编译过程最消耗时间 ---- 所以 xcache又提高了服务器的性能

代码编译成opcode之后 才需要和后端的memcached进行交互



这就是xcache和Memcached的区别

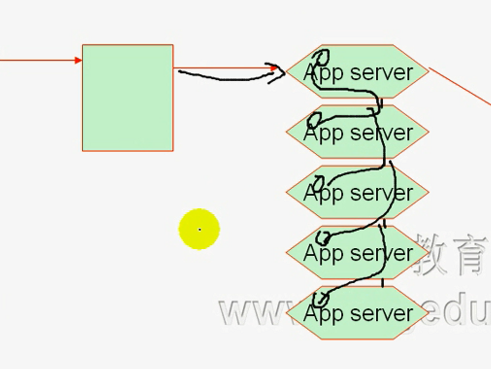
很显然 前端用户的请求比较多 但是 本地没有 都要给到appserver 这样用户的链接被发到别的appserver ---- 但是对于电商网站 每一个客户端连接都要有自己的购物车 什么之类的 这样每一个客户端链接都要有一个session ----- 很显然 同一个请求 多次转发 session就不能用了 ------ 所以 可以使用持久链接 ---- 让来自于一个客户端请求持续的发送到同一个app server ----这样 session就有效了

但是 这个服务器挂了 里面的会话就没了 这时候购物车中的商品没有了

------ 由于主机出现故障 会丢失很多用户的会话信息 非常可怕

----- 所以 让各个APPserver共享会话信息就没有问题

这样 就是 一个请求过来之后 对应的session就马上同步到所有的appsever中

 ---- 这样就会带来额外的开销 如过appserver太多 这样同步就会出现巨大的问题 i----- 大规模集群中 不甚理想

所以 使用一个缓存机制 让请求到这个公共机制中 读或者写这个session

本地就是进程自己的内存 ------ 所以 只能被当前的app server所使用

为了多个服务器共享 ----- 所以 使用Memcached

所以 php 通常是保存到本地文件系统的某个目录下 + 内存中

----- 后期 同一个服务器 app server 会到 Memcached中 根据用户的cookie中找到session 就可以了

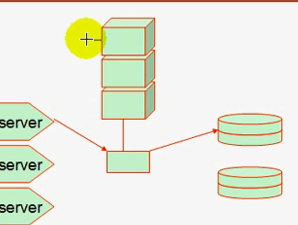
但是风险大了 所以 如果 Memcached宕机了 就完蛋了---- 所以 Memcached可以高可用

但是 缓存的信息都不是重要的信息 ----- 所以 Memcached可以这样假设 保存在我这的信息都是非关键的信息 ---- 丢了无非就是重建就可以了 ---Memcached不支持高可用 -----

所以 Memcached可以建设成高可用 但是 意义不大----- Memcached对高可用的支持不理想 但是 可以很好地实现分布式功能

所以 如果一个Memcached不够 可以用多个Memcached

Appsever 挑选同一个服务器来存储



【CPU密集型 叫CPU bound】

每一个用户建立一个session ---- 都有自己的cookie ----做hash运算 ---- 之后 到图示中的3台Memcache中进行查找 ------ /3取余数就能找到了

所以memcached既可以存放mysql查询出来的数据 还可以缓存客户端的session 所以Memcache叫做万金油 或者银弹

----- 注意 memcache缓存的一定是可序列化数据 ------ 就都能实现缓存

什么是可序列化？------ 无论什么类型的数据 都会转化成一根线 ---- 传输到另一方 ---- 另一方还可以把数据进行还原 ---- 这样的数据就是可序列化数据

不能序列化 就不能再内存中存储 就更不能传输 --- 一般来讲 memcache可以序列化的数据 -----

比如 String 各种Object

缓存的方法 就是在内存中开辟一个很小的区域

将来缓存1kw个对象 怎么在缓存中查找？----- 一个个找 就太慢了 ---- 为了快速 就是键值存储 ---- 存储的时候 要先建立key

Key-value存储 都是基于hash的的存储 ----- 没存储一个key key也需要空间存储 ---- hash要存放在桶 bucket中 

多个桶 每一个key存放到某一个桶 ---- 存放的路由规则就是 我这个key通过hash函数 存放到某一个桶中 ----- 然后 某一个key过来之后 hansh运算找到某一个桶 ---然后在桶里面进行字符串匹配 就快了

基于这种方式查找 就非常快 --- 1000万 和 1000个键 通过hash函数都能直接定位到某一个桶上面 ---- 这样 无论key有多少 查询的速度是一样的 所以 算法的时间复杂度就是O(1)

----- hash算法后面会将MySQL的索引的

所以hash索引查找速度非常快

所以 查找都是非常快的

太多也不好 因为每存储一个对象 都要开辟内存 对象用户用完 要清除缓存 时间长了 内存就碎片化 ---- 这个就要内存管理策略 ----- 这个就要消耗CPU时间

所以 每一次6要参与执行 就要有额外消耗 ----- 10w个的对象创建和回收 开销比较大

Memcache最大支持1M的缓存对象 memcache本身有一个最小的空间限定 ---- 48B

Memcache是CS架构

应用程序通过协议传给Memcache Memcached就对数据进行缓存

有些数据不仅仅是键值对

比如存储一个用户信息 把用户名 年龄 浏览的帖子 等等

这些信息 该怎么记录？

用户名 key

仅有键值对的数据 并不能无法应付

如果我们可以把几个键值组合到一起 建立自己的数据结构

放在一个大的独享 ---- 年龄-v 性别-v 这样存放在Memcache就方便一些

对于列表数据的存储 redis就好很多 ----可以提供类似于数据库的功能

就相当于一行数据 但是这一行数据 是独立存储的

Mysql需要每一个字段事先建立好 ---- 浪费内存

Redis可以存储在内存 也可以持久化到硬盘上 但是 memcache做不到这些

所以 需要用到持久数据的时候 需要为一个对象存储复杂的键值信息的时候 我们就使用redis替换Memcache

----- 到底用Memcache 还是redis

如果仅仅是kv存储 那就Memcache 否则就用redis

Redis可以使用持久存储 可以保存对象 不像Memcache ----- 不支持SQL语句查找 ----- 但是 是一种内存数据库 有自己的查询命令

Memcache本身不是nosql 本身仅仅是键值存储

用了redis 我们服务器的功能得到增强 ---- 但是也是非结构化数据 --- 应用起来也是很不方便

通常需要持久存储 或者 数据挖掘 还需要 关系型数据

NOSQL是一类技术 ---- 仅仅解决特定领域的问题 应用场景各有不同 找到时间 NoSQL的适用场景

键值库可以存储session 可以存储数据

Redis可以存储计数器 ---- 这种更新特别频繁

所以 各有各的适用领域

所以 适用哪一种应用 这种应用使用哪一种硬件？

这些要进行选型

必须要了解每一种材料的特性

----- 既能够存储session 又能够存储计数器

----

Nginx可以将动态内存转发到app server中

----- 人员增多 网站不停向前发展   
 ---功能模块不停加上来 ---- 可能服务器会扛不住

--- 各种复杂 应用程序会产生各种bug 难以追踪 ---- 这个时候可以将网站分区

--- 比如论坛是一组服务器 首页是另一组服务器 但是页面上面 是同一个页面可以看到  
----- 只不过 各种应用服务器处理不同的业务逻辑

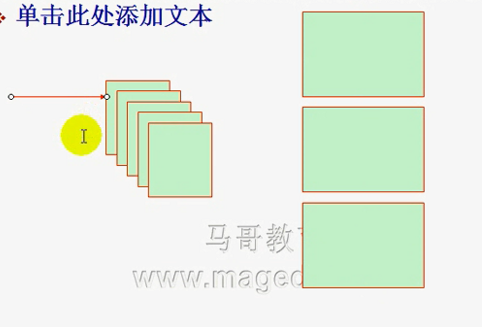
**---- 设计高性能网站 因为首页访问量最大 所以 首页一定要静态化**

---- 只是某些子内容 可能是缓存的

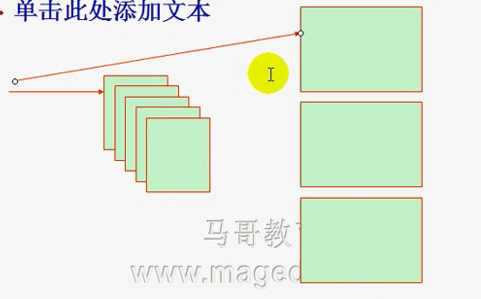
由此 可以看到这样的结果

一堆服务器组 搜索服务器组 抢购服务器组 还有论坛服务器组 前端服务器组 ---- 这样就是很多组服务器

前端服务器组接受用户对主页的访问 ----



20w用户 每一个服务器可以承受2w个请求 10个服务器即可 如果用户点击了某一个功能 通过url重定向 就定向到另一个服务器组上面



左边的是前端组 右边的是其他服务器组

这时候 我们的网站就开始分区了

回过头来看

总有一天 你的某一个服务器故障了 ---- 此时要进行高可用

这个是一个分发的服务器

实现分发的服务器有哪些？lvs nginx(方向代理服务) ha-proxy

Lvs是四层的

Nginx和ha-proxy是7层的 可以理解各种协议 而lvs是在内核中工作的 但是后面两种是在用户空间工作的

4层的性能好 ---- 因为就没有到用户的内存空间 数据就完成了转发

Lvs经过优化 转发数百万个请求 没有问题

但是nginx ha-proxy几万个就够了

但是 不仅仅是转发请求 还要建厂应用服务器的健康状况

所以 还要有Lvs-director 另外 使用dr模型 lvs配置比较复杂 但是性能是最好的

网站 开始可以使用意配置的

同等条件下 nginx转发能力不如ha-proxy 但是 差别不大 都是基于线程

Nginx自身有缓存功能

有必要 可以借助于专业的缓存服务器来实现

这种转发 直接转发到用户服务器 每一个转发

Nginx如何降低每一个app server的服务器压力？

nginx本地可以缓存opcode执行的结果的 ---- 就可以使用nginx本地缓存

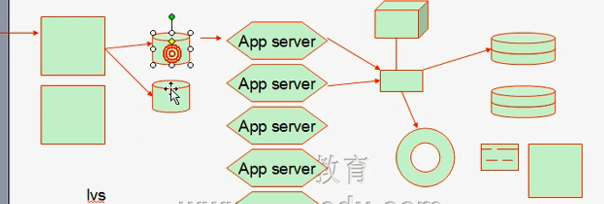
Memcached xcache nginx本地缓存

Memcached缓存mysql查询到的数据 需要应用程序写入Memcached中

Xcache缓存php编译的结果 可以在多进程之间共享opcode

转发服务器本身就很忙 再负责缓存 压力更大 可以进一步把缓存独立出来

独立出来一个缓存服务器 直接面对的是 转发服务器和 后端的业务 app server

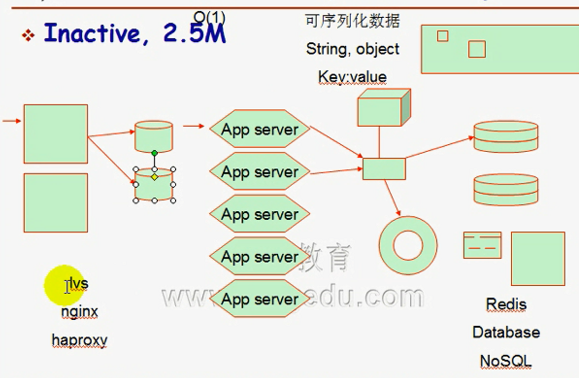


所以请求 由请求服务器转发的缓存之后（这个缓存缓存的是最终的请求和响应） 缓存本地查找 有 就把结果交给转发服务器 转发服务器进行响应 没有 走后面的App server

缓存服务器没有 就走app server

存在一个问题：前端转发的时候 第一次请求转发到第一个缓存 第二次请求转发到了第二个缓存 这样前一次的缓存姐估计就不能使用了 所以 要么使用长连接 要么使用缓存数据同步方式来解决这个问题

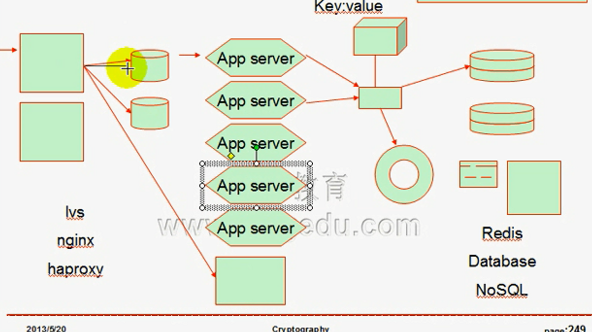
持久链接会破坏负载均衡效果



这里面就是4层

淘宝站点图片量非常大 所以 有一组服务器 做图片内存

所以动态内容先转发给memcache/varnish 静态内容直接转发给 静态服务器组



还可以找到两台服务器 为图片服务器进行缓存

Nginx作为前端 nginx可以对php转发到fastcgi服务器 --- 如何配置？

做正则表达式匹配？

所有以.php的请求 使用fastcgi\_proxy 127.0.0.1:9000

Fast\_proxy这个本身就是反向代理 --- 代理 --- 就是 你到我这里查询 如果没有 可以替提供帮助 到代理的对象中 做真正的查找

期望nginx 对任何内容都反向代理 而不仅仅是对fastcgi做反向代理

 ---- fastcgi的代理

这次本地不提供任何内容---- 配置nginx的本地 就是nginx本地没有资源可以提供 ---- 没有root ---- 全部转发到后端服务器



如何定义这个后端服务器？使用proxy\_pass 明确说明转发的对象 ---使用proxy\_pass进行反向代理

nginx配置是分段的

main段 events http server location main

这些都是分段的配置

Server只能放在http中

Nginx每一个指令 必须使用;进行结尾

Nginx配置中每一个指令 必须有很多参数

Server用来定义一个虚拟主机

----- nginx必须定义成虚拟主机 哪怕只有一个

http ---- web服务相关 都放在这里面

events ---- nginx是事件驱动的 epoll 或者 /dev/poll ----- 定义事件驱动内容

main端主要定义work进程数量 错误日志error log user group



location 是用户访问的URI ------ 还有网页在本地存放位置 ----- 还有访问控制规则 就是 这个URI对于别人来讲 谁能访问 谁不能访问

由此 location主要来定义谁能来干什么的 ---- 我们uri及其对应的文件操作属性

Location的语法：~ ~\* = ^~

=路径精确匹配 通常匹配一个文件

Location {opt} URI{

}

Location内部 应以了URI ----- 要把root 修改为proxy\_pass 就实现了把用户的请求 反向代理到后端服务器上面去 【从web请求 ---- 实现反向代理】

Proxy\_pass的格式 ---- 同上是一个URI

 会把请求的URI 定义到后面的





假设访问的是[www.magedu.com](http://www.magedu.com)

http:// [www.magedu.com/forum/的时候](http://www.magedu.com/forum/的时候) 就会反向代理 或者 转发到后面的

是这样转发的

如果在URI前面使用了模式匹配 ~ 那么转发的一定是不能指定路径

 ---- 多了就是语法错误 具体如下：

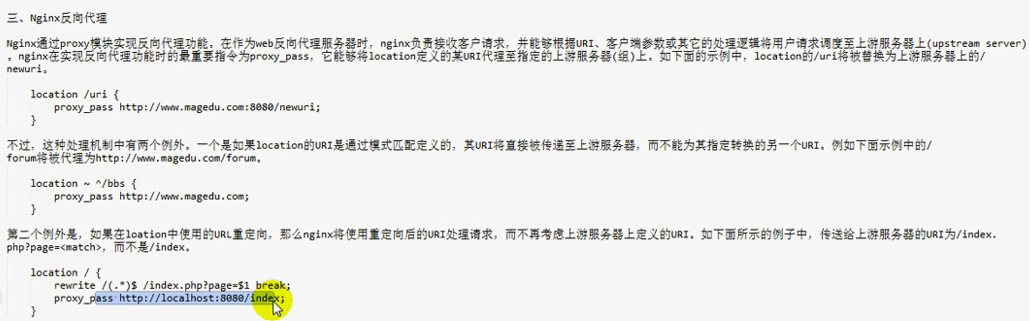
比如

 这个是以forum开头的



对这样的一路径的访问 要转发到

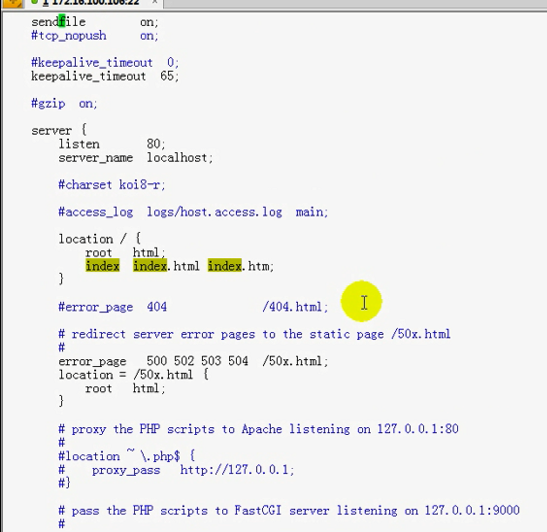
如果是这样的  那么就是转发到/bbs/forum ---- 这是一种特殊情况 ---老师给的文档有特殊的说明【nginx反向代理的特殊说明】



【upstream叫做上游】

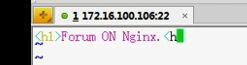
【老师写的文档要看】

如何将请求转发到后端



现在期望对本地/forum的访问 都转发到后端服务器





很显然 此时是可以访问到的



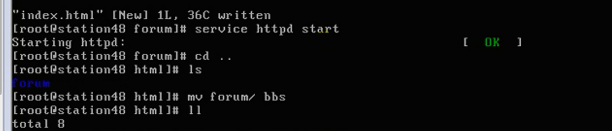
现在期望对forum的访问 都代理到另一台服务器上面做访问

安装一个httpd

默认 /var/www/html/下面 创建一个/forum







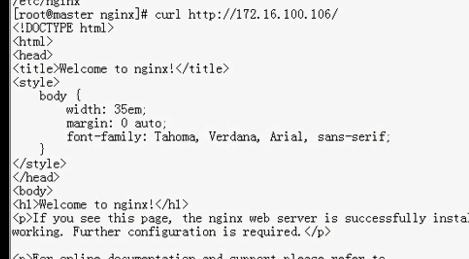
Nginx请求代理到后端

在nginx上面重新定义



本地测试

使用Curl命令





前面写的是document moved



说你生成了过多的重定向

重新编辑一下

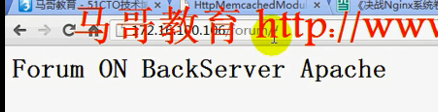
去掉这个/

Curl一下 就没有问题了

这个/是这样的 就是 /forum后面要/ 那么proxy\_pass后面的URI也要有/

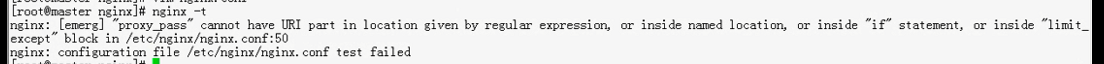
 【这个地方要注意】

要不就都没有 ---- 要对应起来



----- 如果使用正则表达式匹配 发生：



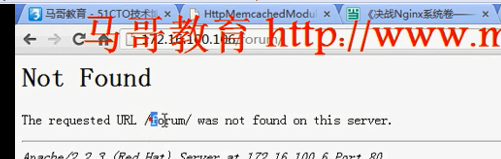


语法错误

说明 不允许uri方式

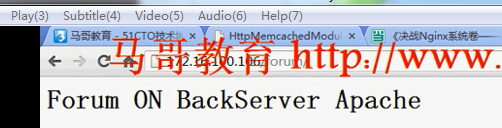




 这时候 后端服务器的目录是/bbs 所以 就访问不到了

现在 把/bbs修改为/forum



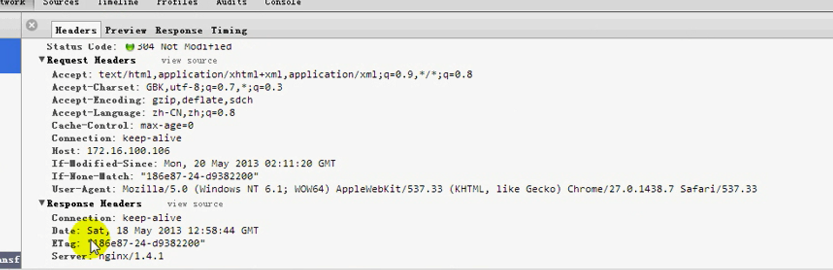


所以反向代理用到了location的正则匹配 那么 这个URI就会自动提交到proxy\_pass后面的主机名:port

这就是反向代理

浏览的时候 分析整个执行过程 F12





这里面就有请求首部 响应首部 响应headers中 还有Server 给出的是nginx的版本

使用curl –I xxxx 就可以仅仅给出首部信息



有一个问题 对应的web服务器上面



Log访问的是：

---- 浏览器中的强行刷新 --- Ctrl+F5 ---- 不能使用缓存 必须送到服务器中

后端服务器的日志中 仍然是 nginx的服务器的日志



为什么？

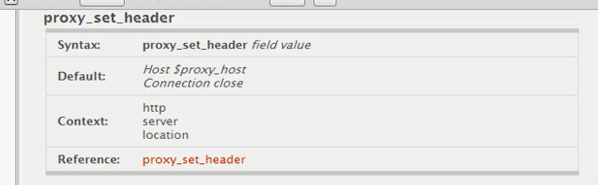


反向代理的过程 实际上是前端机Nginx创建的request 很显然 记录的都是nginx访问的 所以 这样 就是没有意义

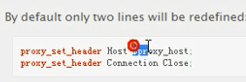
这个该怎么办？

我们的nginx 提供代理功能是依靠代理模块提供的

这个模块中 有一个指令 就是proxy\_set\_header指令



客户端传递到前端机nginx ---> 向后转发报文 在这个nginx中的请求报文中添加一个特殊的key 使用proxy\_set\_header ---- 在转发到后端服务器 或者上游服务器的时候 就可以在请求报文中使用proxy\_set\_header进行添加



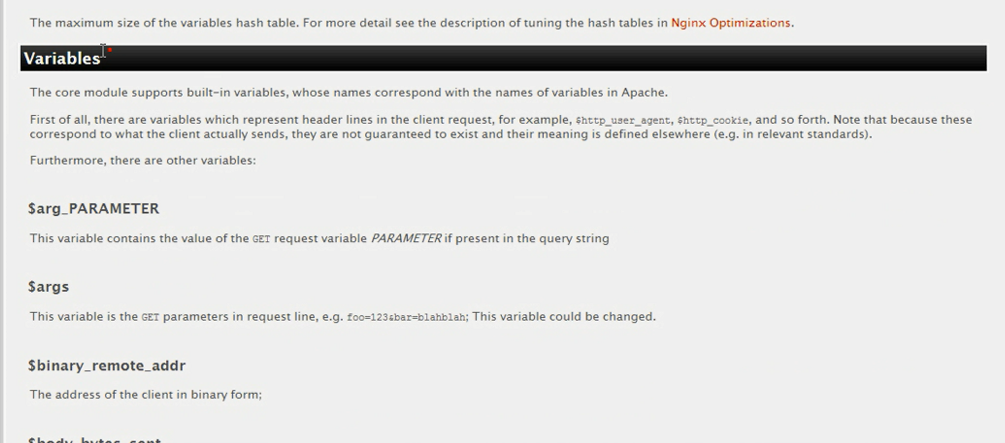
加一个特有的首部 x-real-ip



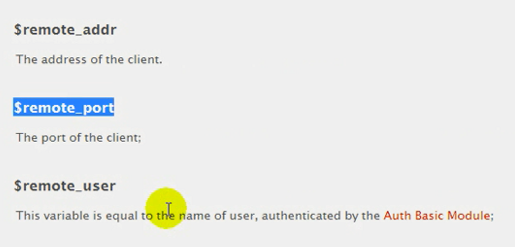
这样你的首部就有X\_Real\_IP 这个值是什么？

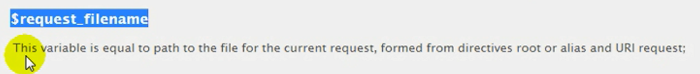
Nginx在官方 核心模块中http的core 模块中 提供了一大堆变量 可以在Nginx配置中直接访问

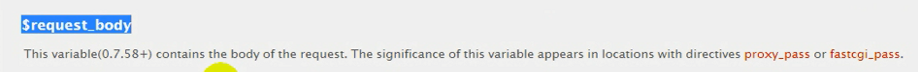


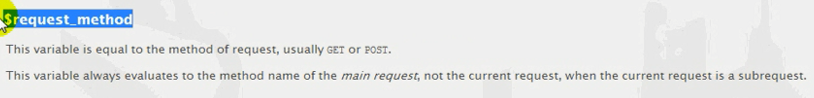












HTTP请求中有很多方法 ---8个

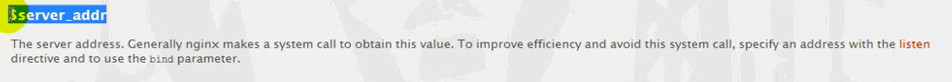
get post head put trace options connection delete

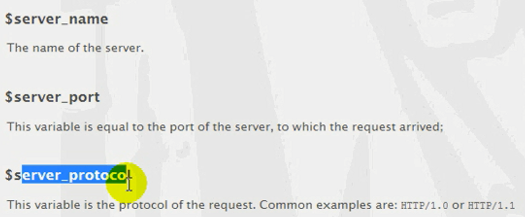
最常用的是get post head

request\_uri 是什么？选中部分

还有就是

 这里面就是http还是https

这个一般是反向代理服务器端的地址

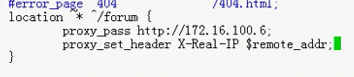


所以我们直接传递

这样 把客户端的地址 独特的首部传递给后端服务器

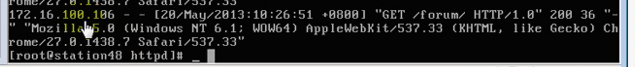
编辑nginx.conf

 也可以叫x-forwarded-for



这样转发到后端的请求的首部报文 就会有一个X-Real-IP来传递客户端真正的请求地址

传送给后端 还是无法记录下来

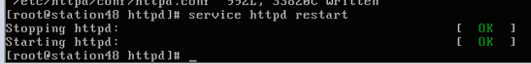


后端如何修改呢？

编辑后端的日志地址



修改成%{X-Real-IP}i ---- 这个%{}i 就表示引用请求首部的某一个header的值

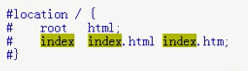




这个就是整个向后端转发地址

---- 这里面引入了一个新的指令 set\_proxy\_header

现在要把整个网站的访问 统统代理到一个后端 怎嘛办？





(还是要注意 因为我的URI只有一个/ 所以 我的反向代理的结果 也是要有/作为配对)



这样所有请求都转发到这个后端

如果有多个后端服务器 扛不住了 该怎么办？

Nginx可以代理用户请求至多个服务器 ---可以实现负载均衡的效果 还可以实现每一个后端服务器的健康检查 ---- 这个时候已经不再是反向代理了 而是upstream

上游服务器upstream ---- 用来把多个服务器归并为一个服务器组 并且还可以定义调度算法



点开



Upstream不能定义在Server中 要定义在server之外

Upstream模块中的server指令 用来定义后端服务器



没有权重 说明权重为0 说明不使用负载均衡

定义server的时候 不能使用http 只能有服务器名和端口号



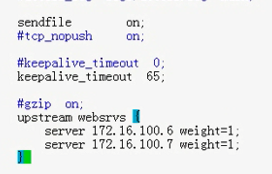
复制到另外一个接节点上去



修改nginx.conf

在server之外定义

可以定义多个upstream



仍然使用反向代理 只不过 代理的不是某一个主机 是一个集群







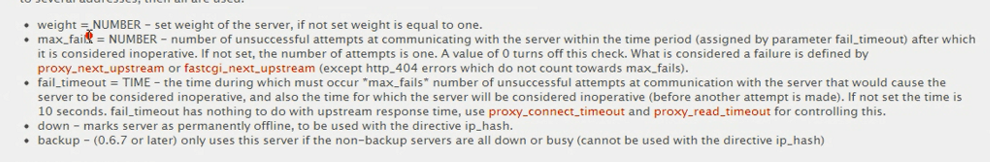
这样就实现了负载均衡的效果

如果某一个real server挂了怎么办？

做 健康状况检查

在upstream





 ---- 最多错误此时

 ---- 每次检查要持续的时间

所以 修改如下



此时1和2正常

2停机

刷新 始终是1

2重新启动 就是1 2 负载轮转

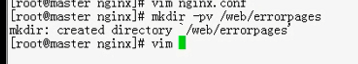
还有一个健康检查 就是所有的服务器都宕机了

----

Nginx也能

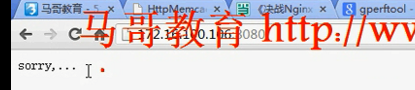
本机定义一个虚拟主机





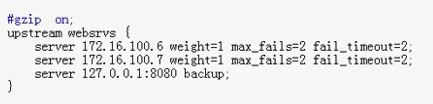
创建一个index.html



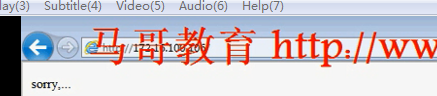


在upstream中





这样两个服务器都宕机



此时 又重新上线



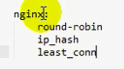
这就是nginx的负载均衡

这个是轮询的

有时候 要把同一个客户端请求始终定向到同一个server

新版的nginx支持一个ip\_hash 根据客户端请求的原地址 进行分发

Nginx支持三种附在均衡的算法



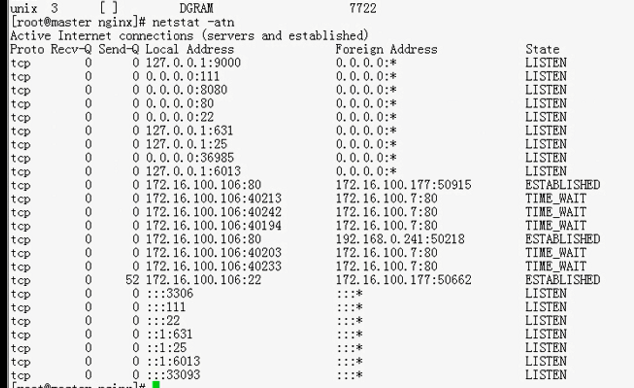
默认是round-robin ---- 没有定义权重 表示这个服务器不参加负载均衡

Ip\_hash根据ip做hash 一直访问到固定的服务器 保证缓存可用

Least\_conn ----- 看哪个服务器的请求少

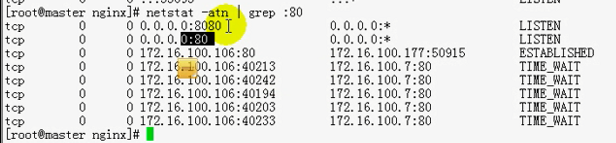
在ip\_hash方式下 不让使用backup server



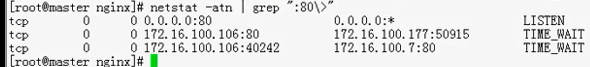


当前客户端的链接状况

找到80



再限制 “:80\>”



这样可以看到web服务器的相关状态

使用awk来统计各类的服务器的状态

Awk中 引用变量值 不需要加$



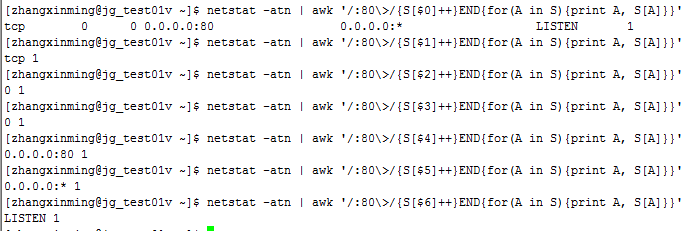
Awk中使用引用变量的值 不需要加$

NF假设就是4 ----- $4表示第四个字段

这个$NF ----就是表示第四个字段

实际上：





所以 实际上 NF的值是6 $0表示整行的字符串

S是数组

Awk的数组可以是任意字符串 可以不是0 1 2

评估一个服务器当前有多少链接的时候 这个命令是非常重要的

数组中的元素 是随时使用 随时定义

 这个就是存储的是 [“LISTEN”]这个++ 值++

默认是0

 A in S是A存储的都是S的一个下标

【NF ---- number of field ---表示字段数 这里面有5个字段 所以 NF=4

这里面的代码就是

/:80\>/

{S[$NF]++} ---- 每一行的最后一个部分作为index进行累计 ---- 定义的数组名就是S

END{

for( A in S)

{

Print A, S[A]

}

}

】