

目錄

Introduction	1.1
1.千万pv级web架构设计	1.2
2.HTTP协议分析	1.3
3.apache 优化	1.4
4.lamp-优化	1.5
5.nginx 反向代理&&负载均衡&&缓存	1.6
6.nginx-优化	1.7
7-varnish	1.8
8-memcache	1.9
9-redis	1.10
10-zabbix	1.11
邮件报警	1.11.1

web 架构设计

内容包括

1. 千万 pv 级架构设计
2. HTTP 协议分析
3. apache 优化
4. lamp 优化
5. nginx 发现代理，缓存，负载均衡
6. nginx 优化
7. varnish
8. memcache
9. redis

千万级PV规模高性能高并发网站架构

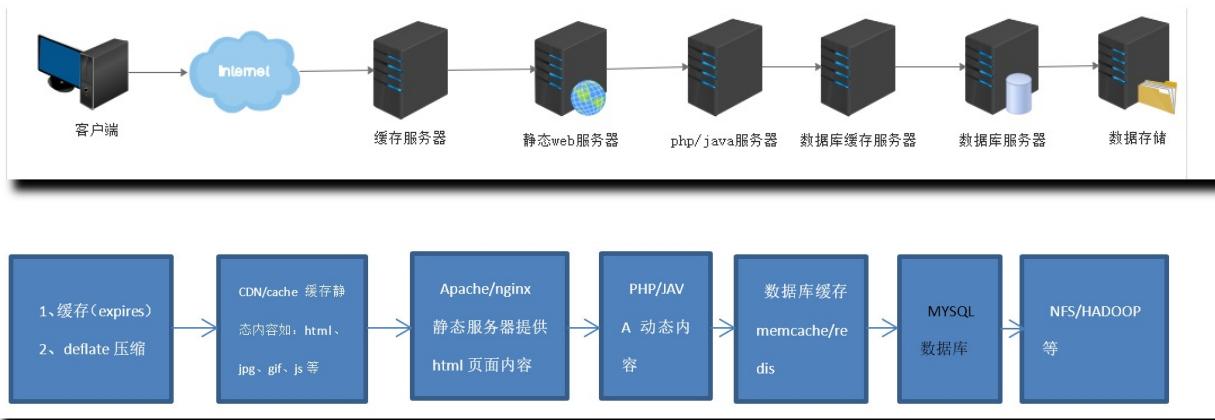
目录

- I. 高性能高并发高可扩展网站架构访问的几个层次：
 - 第一层：首先在用户浏览器端
 - 第二层：静态页面内容缓存
 - CDN
 - 小型CDN
 - 第三层：静态服务器层
 - 第四层：动态服务器层：`php`，`java` 等，
 - 第五层：数据库cache层，
 - 第六层：数据库层，
 - 第七层：
- II. 扩展知识点1：CDN
 - 1.CDN含义理解
 - 2.例子
 - 3.未使用CDN
 - 4.使用CDN
 - CDN网络实现的具体操作过程
 - CDN的典型拓扑图如下
 - CDN对网络的优化作用
- III. 扩展知识点2 `pv` `uv` `ip`
 - 1. `pv`
 - 2. `uv`
 - 通过 `cookie` 是判断 `UV` 值的方式
 - 3. `ip`
 - 查询方法
 - 1. 使用alexa统计
 - 2. 一般大型网站都有自己的一套流量统计系统，可以到自己的后台查看。
 - 3. 如果没有的话，可以借助 `GoogleAnalytics`、`cnzz`、`51.la` 等统计平台查看数据。
 - IP、PV、UV的计算

- 对IP计算
- 对PV的计算
- 对UV的计算

- 每秒并发数预估：

- **IV. 千万 PV 级别 WEB 站点架构设计**



高并发访问的核心原则其实就一句话“把所有的用户访问请求都尽量往前推”。

如果把来访用户比作来犯的“敌人”，我们一定要把他们挡在800里地以外，即不能让他们的请求一下打到我们的指挥部（指挥部就是数据库及分布式存储）。

如：能缓存在用户电脑本地的，就不要让他去访问CDN/cache。能缓存CDN/cache服务器上的，就不要让CDN/cache去访问源（静态web服务器）了。能访问静态web服务器的，就不要去访问动态服务器。以此类推：能不访问数据库和存储就一定不要去访问数据库和存储。

I. 高性能高并发高可扩展网站架构访问的几个层次：

[回目录](#)

第一层：首先在用户浏览器端

[回目录](#)

使用Apache的 `mod_deflate` 压缩传输，再比如：`expires` 功能，`deflate` 和 `expires` 功能利用的好，就会大大提升用户体验效果及减少网站带宽，减少后端服务器的压力。

提示：有关压缩传输及 `expires` 功能 `nginx / lighttpd` 等软件同样也有。

第二层：静态页面内容缓存

[回目录](#)

CDN

[回目录](#)

如图片/`js / css` 等或静态数据 `html`，这个层面是网页缓存层，比如CDN（效果比公司自己署 `squid / nginx / varnish` 要好，他们更专业，价格低廉，比如快网/`CC` 等，而且覆盖的城市节点更多）。

小型CDN

[回目录](#)

自己架设 `squid / nginx / varnish` 来做小型CDN是次选(超大规模的公司可能会考虑风险问题实行自建加购买服务结合)，为前端的CDN提供数据源服务，以减轻后端我们的服务器数据及存储压力，而不是直接提供cache服务给最终用户。淘宝的CDN曾经因为一部分图片的尺寸大而导致CDN压力大的情况，甚至对图片尺寸大的来改小，以达到降低流量及带宽的作用。

提示：我们也可以自己架设一层cache层，对我们购买的CDN提供数据源服务，可用的软件有`varnish/nginx/squid` 等cache，以减轻第三层静态数据层的压力。在这层的前端我们也可以架设DNS服务器，来达到跨机房业务拓展及智能解析的目的。

第三层：静态服务器层

[回目录](#)

一般为

- 图片服务器
- 视频服务器
- 静态HTML服务器

这一层是前面缓存层和后面动态服务器层的连接纽带。

第四层：动态服务器层：**php**，**java** 等，

[回目录](#)

只有通过了前面3层后的访问请求才会到这个层，才可能会访问数据库及存储设备。经过前三层的访问过滤能到这层访问请求一般来说已非常少了，一般都是新发布的内容和新发布内容第一次浏览如；博文（包括微博等），BBS帖子。

特别提示：此层可以在程序上多做文章，比如向下访问**cache**

层，**memcache**，**redis**，**mysql**，**oracle**，在程序级别实现分布式访问，分布式读写分离，而程序级别分布式访问的每个 **db cache** 节点，又可以是一组业务或者一组业务拆分开来的多台服务器的负载均衡。这样的架构会为后面的数据库和存储层大大的减少压力，那么这里呢，相当于指挥部的外层了。

第五层：数据库**cache**层，

[回目录](#)

比如：**memcache**，**redis** 等等。

根据不同的业务需求，选择适合具体业务的数据库。对于 **memcache**、**redis**，可以在第四层通过程序来实现对本层实现分布式访问，每个分布式访问的节点都可能是一组负载均衡（数十台机器）。

第六层：数据库层，

[回目录](#)

一般的不是超大站点都会用**mysql**主从结构，程序层做分布式数据库读写分离，一主（或双主）多从的方式，访问大了，可以做级连的主从及环状的多主多从，然后，实现多组负载均衡，供前端的分布式程序调用，如果访问量再大，就需要拆业务了，比如：公司把 **www** 服务，**blog** 服务，**bbs** 服务都放一个服务器上，然后

做主从。这种情况，当业务访问量大了，可以简单的把 www , blog , bbs 服务分别各用一组服务器拆分开。当然访问量再大了，可以继续针对某一个服务拆分如： www 库拆分，每个库做一组负载均衡，还可以对库里的表拆分。需要高可用可以通过MHA等工具做成高可用方式。对于写大的，可以做主主或多主的MYSQL REP方式。

像百度等巨型公司除了会采用常规的 mysql 及 oracle 数据库库外，会在性能要求更高的领域，大量的使用 nosql 数据库（非关系型的数据库），然后前端在加 DNS ，负载均衡，分布式的读写分离，最后依然是拆业务，拆库。逐步细化，然后每个点又可以是一组或多组机器。

特别提示：数据库层的硬件好坏也会决定访问量的多少，尤其是要考虑磁盘 IO 的问题，大公司往往在性价比上做文章，比如核心业务采用硬件 netapp / emc 及 san 光纤架构，对于资源数据存储，如图片视频，会采用 sas 或固态 ssd 盘，如果数据超大，可以采取热点分取分存的方法：如：最常访问的10-20%使用 ssd 存储，中间的20-30%采用 sas 盘，最后的40-50%可以采用廉价的 sata 。

第七层：

[回目录](#)

千万级PV的站如果设计的合理一些，1，2个 NFS SERVER 就足够了。当然可以做成 drbd + heartbeat + nfs + a/a 的方式。

以上1-7层，如果都搭好了，这样漏网到第四层动态服务器层的访问，就不多了。一般的中等站点，绝对不会对数据库造成太大的压力。程序层的分布式访问是从千万及PV向亿级PV的发展，当然特殊的业务还需要特殊架构，来合理利用数据库和存储。

II. 扩展知识点1：CDN

[回目录](#)

1. CDN含义理解

[回目录](#)

CDN的全称是 Content Delivery Network，即内容分发网络。

其基本思路是尽可能避开互联网上有可能影响数据传输速度和稳定性的瓶颈和环节，使内容传输的更快、更稳定。通过在网络各处放置 节点服务器 所构成的在现有的互联网基础之上的一层智能 虚拟网络 ，CDN系统能够实时地根据 网络流量 和各节点的连接、负载状况以及到用户的距离和响应时间等综合信息将用户的请求重新导向离用户最近的服务节点上。其目的是使用户可就近取得所需内容，解决Internet 网络拥挤 的状况，提高用户访问网站的响应速度。

2. 例子

[回目录](#)

举个通俗的例子：

谈到CDN的作用，可以用早些年买火车票来比喻：在没有火车票代售点和 12306.cn 之前。那时候火车票还只能在火车站的售票大厅购买，而小县城并不通火车，火车票都要去市里的火车站购买，而从县城到市里，来回就得n个小时车程。

到后来，小县城里出现了火车票代售点，可以直接在代售点购买火车，方便了不少，人们再也不用在一个点排队买票了。

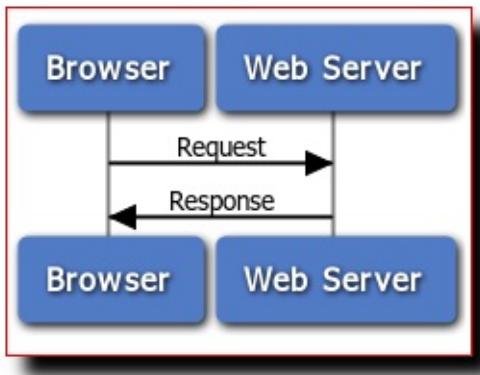
CDN就可以理解为分布在每个县城的火车票代售点，用户在浏览网站的时候，CDN会选择一个离用户最近的CDN节点来响应用户的请求，这样海南移动用户的请求就不会千里迢迢跑到北京电信机房的服务器（假设源站部署在北京电信机房）上了。

CDN的基本原理为反向代理，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节点服务器。通过部署更多的反向代理服务器，来达到实现多节点CDN的效果。

3. 未使用CDN

[回目录](#)

首先，让我们先看传统的未加缓存服务的访问过程，以便了解CDN缓存访问方式与未加缓存访问方式的差别：



用户提交域名→浏览器对域名进行解析→得到目的主机的IP地址→根据IP地址访问发出请求→得到请求数据并回复

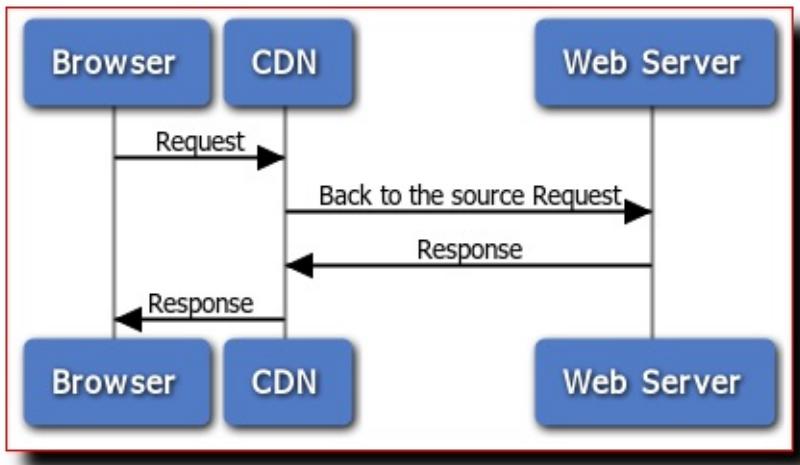
```
st=>start: 用户提交域名  
op=>operation: 浏览器对域名进行解析  
op1=>operation: 得到目的主机的IP地址  
op2=>operation: 根据IP地址访问发出请求  
op3=>operation: 得到请求数据并回复  
e=>end
```

```
st->op->op1->op2->op3->end
```

由上可见，用户访问未使用CDN缓存网站的过程为：

- 1)、用户向浏览器提供要访问的域名；
- 2)、浏览器调用[域名解析](#)函数库对域名进行解析，以得到此域名对应的IP地址；
- 3)、浏览器使用所得到的IP地址，向域名的服务主机发出数据访问请求；
- 4)、浏览器根据域名主机返回的数据显示网页的内容。

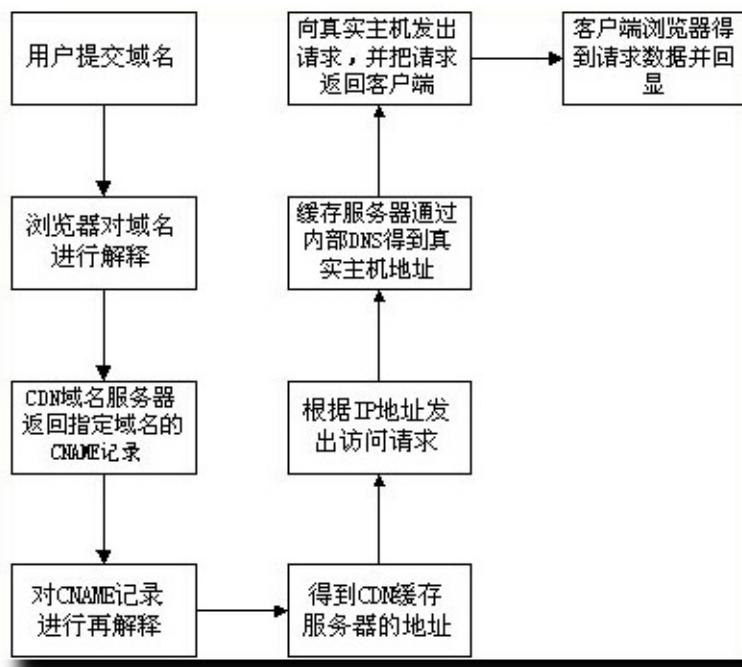
通过以上四个步骤，浏览器完成从用户处接收用户要访问的域名到从域名服务主机处获取数据的整个过程。CDN网络是在用户和服务器之间增加Cache层



4. 使用CDN

[回目录](#)

如何将用户的请求引导到Cache上获得源服务器的数据，主要是通过接管DNS实现，下面让我们看看访问使用CDN缓存后的网站的过程：



通过上图，我们可以了解到，使用了CDN缓存后的网站的访问过程变为：

- 1)、用户向浏览器提供要访问的域名；

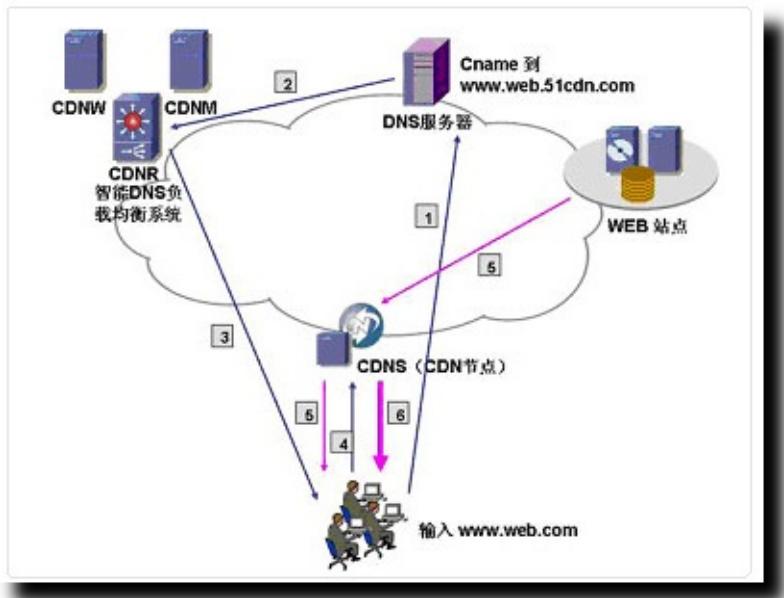
- 2)、浏览器调用域名解析库对域名进行解析，由于CDN对域名解析过程进行了调整，所以解析函数库一般得到的是该域名对应的 CNAME 记录，为了得到实际 IP地址，浏览器需要再次对获得的 CNAME 域名进行解析以得到实际的 IP地址；在此过程中，使用的全局负载均衡DNS解析，如根据地理位置信息解析对应的IP地址，使得用户能就近访问。
- 3)、此次解析得到CDN缓存服务器的IP地址，浏览器在得到实际的IP地址以后，向缓存服务器发出访问请求；
- 4)、缓存服务器根据浏览器提供的要访问的域名，通过Cache内部专用DNS解析得到此域名的实际IP地址，再由缓存服务器向此实际IP地址提交访问请求；
- 5)、缓存服务器从实际IP地址得得到内容以后，一方面在本地进行保存，以备以后使用，另一方面把获取的数据返回给客户端，完成数据服务过程；
- 6)、客户端得到由缓存服务器返回的数据以后显示出来并完成整个浏览的数据请求过程。

通过以上的分析我们可以得到，为了实现既要对普通用户透明(即加入缓存以后用户客户端无需进行任何设置，直接使用被加速网站原有的域名即可访问，只要修改整个访问过程中的域名解析部分，以实现透明的加速服务。

CDN网络实现的具体操作过程

[回目录](#)

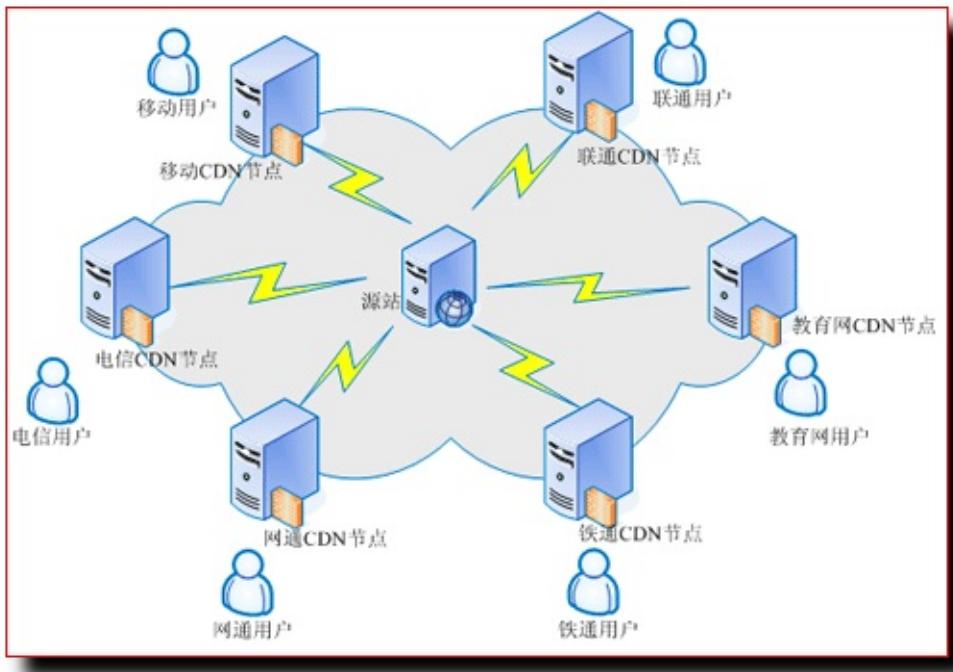
使用了CDN服务后，用户的访问流程如下图所示：



通过以上的分析我们可以看出，CDN服务对网站访问提供加速的同时，可以实现对普通访问用户透明，即加入缓存以后用户客户端无需进行任何设置，直接输入被加速网站原有的域名即可访问。

CDN的典型拓扑图如下

[回目录](#)



CDN和反向代理的基本原理都是缓存数据，区别就在于CDN部署在网络提供商的机房，使用户在请求网站服务时，可以从距离自己最近的网络提供商机房获取数据。

CDN对网络的优化作用

[回目录](#)

CDN系统通过在网络各处放置节点服务器，从而将网站的内容放置到离用户最近的地方，避免了影响互联网传输性能的“第一公里”和“网间互联瓶颈”等各个环节，为改善互联网环境、解决网站的服务质量和提高用户的上网速度提供了有效的解决方案。

CDN对网络的优化作用主要体现在如下几个方面：

- 解决服务器端的“第一公里”问题
- 缓解甚至消除了不同运营商之间互联的瓶颈造成的影响
- 减轻了各省的出口带宽压力
- 缓解了骨干网的压力

提起CDN，一般人都会望而止步，因为CDN太贵，都是大企业才能用得起的贵族式服务，而如今面对中小企业的CDN技术开发已经实现，并进入市场开始运营。

现在市面上CDN提供商计费方式多样，有按每月最低消费的，有按带宽收费的，有按请求数收费的，有包月包季包年限制的，还有些大多人看不懂的技术指标收费的，总之比较复杂，CDN服务在所有计费方式中，中小企业一致认为按流量收费最为合理，另外大多按流量计费方式中会有时间限制，规定时间内用不完就会全部作废，对于流量把握不好的中小企业，存在相当一部分浪费。所以企业自己也可以使用 squid / varnish / nginx 等构建缓存服务器。

III. 扩展知识点2 **pv** **uv** **ip**

[回目录](#)

1. **pv**

[回目录](#)

PV(page view)：即页面浏览量，或点击量，PV是网站分析的一个术语，用以衡量网站用户访问的网页的数量。一般来说，PV与来访者的数量成正比，但是PV并不直接决定页面的真实来访者数量，如同一个来访者通过不断的刷新页面，也可以制造出非常高的PV。

2. **uv**

[回目录](#)

UV (uniquevisitor) 即独立访客数：指访问某个站点或点击某个网页的**不同 IP地址**的人数。在同一天内，**UV** 只记录第一次进入网站的具有独立**IP**的访问者，在同一天内再次访问该网站则不计数。**UV**提供了一定时间内不同观众数量的统计指标，而没有反应出网站的全面活动。

通过 **cookie** 是判断 **UV** 值的方式

用 **Cookie** 分析 **UV** 值：当客户端第一次访问某个网站服务器的时候，网站服务器会给这个客户端的电脑发出一个 **Cookie**，通常放在这个客户端电脑的 C 盘当中。在这个 **Cookie** 中会分配一个独一无二的编号，这其中会记录一些访问服务器的信息，如访问时间，访问了哪些页面等等。当你下次再访问这个服务器的时候，

服务器就可以直接从你的电脑中找到上一次放进去的 `Cookie` 文件，并且对其进行一些更新，但那个独一无二的编号是不会变的。所以当客户端再次使用 `cookie` 访问网站时，会附带此 `Cookie`，那么此时服务器就会认为是同一个客户端，那么只会记录一次的 `UV`

使用 `Cookie` 方法比分析客户端 `HTTP` 请求头部信息更为精准，但是会有缺点，那就是用户可能会关闭了 `Cookie` 功能。或者自动删除了 `cookie` 等操作，所以获取的指标也不能说是完全准确。

3.ip

[回目录](#)

IP即独立IP数：

`IP` 可以理解为独立IP的访问用户，指1天内使用不同 `IP` 地址的用户访问网站的数量，同一 `IP` 无论访问了几个页面，独立 `IP` 数均为1。但是假如说两台机器访问而使用的是同一个 `IP`，那么只能算是一个IP的访问。

`IP` 和 `UV` 之间的数据不会有太大的差异，通常 `UV` 量和比 `IP` 量高出一点，每个 `UV` 相对于每个 `IP` 更准确地对应一个实际的浏览者。

① UV 大于 IP

这种情况就是在网吧、学校、公司等，公用相同IP的场所中不同的用户，或者多种不同浏览器访问您网站，那么UV数会大于IP数。

② UV 小于 IP

在家庭中大多数电脑使用 `ADSL` 拨号上网，所以同一个用户在家里不同时间访问您网站时，`IP` 可能会不同，因为它会根据时间变动 `IP`，即动态的 `IP地址`，但是实际访客数唯一，便会出现 `UV` 数小于 `IP` 数。

`PV` 和 `UV` 是衡量一个网站流量好坏的一个重要指标，对于网站的 `PV` 和 `UV` 的统计，可使用第三方统计工具进行统计，只需要将第三方统计工具的 `JS` 代码放置于网站需要统计 `PV` 和 `UV` 的页面即可，然后登录统计工具后台查询网站的 `PV` 和 `UV` 量（如可使用的第三方统计工具为百度统计）；

查询方法

[回目录](#)

1. 使用[alexa](#)统计

英文站：<http://www.alexa.com/>

中文站：<http://alexa.chinaz.com/>

2. 一般大型网站都有自己的一套流量统计系统，可以到自己的后台查看。

3. 如果没有的话，可以借助 **GoogleAnalytics** 、 **cnzz** 、 **51.la** 等统计平台查看数据。

IP、PV、UV的计算

对IP计算

[回目录](#)

1. 分析网站的访问日志，去除相同的IP地址

2. 使用第三方统计工具

3. 在网页后添加多一个程序代码统计字段，然后使用日志分析工具对程序代码字段进行统计。

对PV的计算

[回目录](#)

1. 分析网站的访问日志，计算HTML及动态语言等网页的数量

2. 使用第三方统计工具

3. 在网页后添加多一个程序代码统计字段，然后使用日志分析工具对程序代码字段进行统计。

对UV的计算

[回目录](#)

1. 分析客户端的HTTP请求报文，将客户端特有的信息记录下来进行分析。若能满足共同的特征将会被认为是同一个客户端，那么此时将记录为一个UV。

2. 通过cookie

当客户端访问一个网站时，服务器会向该客户端发送一个Cookie，Cookie具有唯一性，所以当客户端再次使用cookie访问网站时，会附带此Cookie，那么此时服务器就会认为是同一个客户端，那么只会记录一次的UV

缺点：使用Cookie方法比分析客户端HTTP请求头部信息更为精准，但是会有缺点，那就是用户可能会关闭了Cookie功能。或者自动删除了cookie等操作，所以获取的指标也不能说是完全准确。

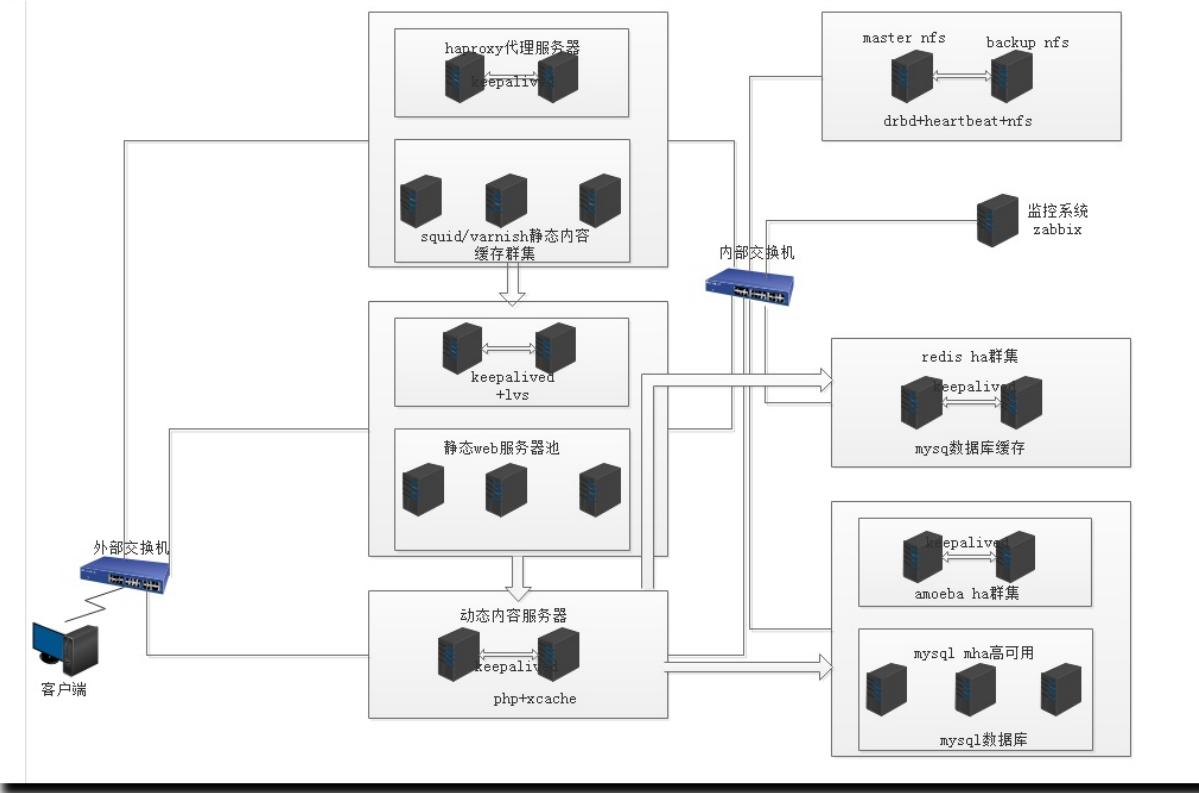
每秒并发数预估：

[回目录](#)

1. 假如每天的 pv 为 6000 万；
2. 集中访问量： $24 * 0.2 = 4.8$ 小时，会有 $6000 \text{ 万} * 0.8 = 4800 \text{ 万}$ (二八原则)；
3. 每分并发量： $4.8 * 60 = 288$ 分钟，每分钟访问 $4800 / 288 = 16.7$ 万 (约等于)；
4. 每秒并发量： $16.7 \text{ 万} / 60 = 2780$ (约等于)；
5. 假设：高峰期为平常值的二到三倍，则每秒的并发数可以达到 $5560 \sim 8340$ 次。

IV. 千万 PV 级别 WEB 站点架构设计

[回目录](#)



1、代理层可以使用 `Haproxy` 或 `nginx`，`Haproxy / nginx` 是非常优秀的反向代理软件，十分高效、稳定。可以考虑用 `F5-LTM` 或成熟的开源解决方案 `LVS` 实现代理层负载均衡方案。

2、缓存层可以使用 `Squid` 或 `Varnish`，缓存服务器作为网页服务器的前置 `cache` 服务器，可以代理用户向 `web` 服务器请求数据并进行缓存。

3、静态 `web` 服务器（`apache / nginx`）提供静态内容访问，实现静动分离；通过相关工具（`lvs / haproxy / nginx`）做负载均衡（`Load Balancer`）

4、动态内容服务器（`php / java`）通过相关工具如 `xcache` 缓存解析过的动态内容。

5、数据库缓存（`memcache / redis`）作为数据库缓存都非常理想。

6、数据库层主流开源解决方案 `Mysql` 是首选，主从复制（一主对多从或多主多从）是目前比较靠谱的模式。

7、存储层作为数据的存储可以考虑 `nfs`、分布式文件系统（如 `mfs`）、`hadoop`（`hadoop` 适合海量数据的存储与处理，如做网站日志分析、用户数据挖掘等）

当用户请求的是静态资源(图片/视频/ html 等),不需要计算处理时,在CDN或缓存层就结束了,当缓存不能命中时,就会去web server中取相应的数据。只有当用户请求动态资源时,才会到动态内容服务器。动态内容服务器可以从数据库缓存或者MySQL中获得数据。

此外,如果前端的程序和数据的存取不同步,是需要异步访问的。这就需要使用一些消息队列,如 rabbitmq ,这在后面 openstack 相关学习做进一步的讲解。同时Apache的开源项目中的 activemq 也能提供相关的功能。

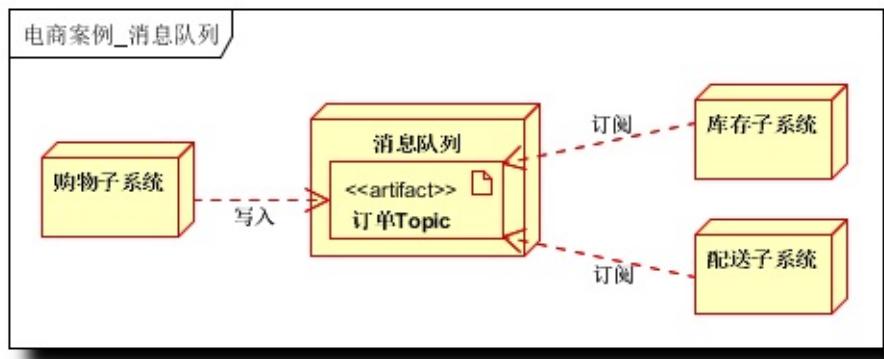
注:消息队列可以解决子系统/模块之间的耦合,实现异步,高可用,高性能的系统。是分布式系统的标准配置。

例如消息队列在购物,配送环节的应用。

用户下单后,写入消息队列,后直接返回客户端;

库存子系统:读取消息队列信息,完成减库存;

配送子系统:读取消息队列信息,进行配送;



[回目录](#)

http协议介绍

目录

- I.http协议的版本
 - http 0.9 : 仅于用户传输html文档
 - http 1.0
 - http 1.1
 - http 2.0
- II.html文本介绍
 - html文本架构
 - html文档的生成方式
 - 静态
 - 动态
 - 静态和动态的方式 ~ 静态 ~ 动态
- III.http协议
 - http协议的报文
 - 请求报文格式介绍
 - 1.请求行
 - 2.请求首部
 - 3.空白行
 - 4.请求实体
 - 响应报文格式介绍
 - 1.起始行
 - 2.响应首部
 - 3.空白行
 - 4.响应实体
 - HTTP请求方法
 - HTTP的状态码
 - 常用状态码说明
- IV.HTTP首部介绍
 - 通用首部
 - 请求首部
 - 响应首部

- 实体首部
 - HTTP最常见的请求头如下：
 - Cookie相关的HTTP扩展头
- HTTP最常见的实体头
- V.HTTP的事务
 - HTTP资源
 - 常用的MIME类型
 - URI和URL
 - CGI
- VI.其他需要了解的知识
 - 一次Web资源请求的具体过程
 - http如何并发的接收多个用户请求
- VII.扩展知识点1：利用wireshark分析HTTP协议
 - 实验步骤
 - 1.清空缓存
 - 2.启动wireshark
 - 3.开始俘获
 - 4.分析数据
- VIII.扩展知识点2：curl查看HTTP响应头信息
 - HTTP响应头的信息
 - 缓存分类

http : Hyper Text Transfer Protocol超文本传输协议，是互联网应用最为广泛的一种网络协议，主要用于Web服务。通过计算机处理文本信息，格式为HTML（Hyper Text Mark Language）超文本标记语言来实现。

I.http协议的版本

[回目录](#)

http 0.9：仅于用户传输html文档

http 1.0

-1.引入了MIME(Multipurpose Internet Mail Extensions)机制：多用途互联网邮件扩展，引入这个技术之后，http可以发送多媒体（比如视频、音频等）信息。此机制让http不再单单只支持html格式，还可以支持其他格式来进行发送了。

2.引入了keep-alive机制，支持持久连接的功能（但这个keep-alive原理是在头部添加了某个字段而形成的，并非原生就支持此功能）

3.引入支持缓存功能

http 1.1

支持更多的请求方法，更加精细的缓存控制，原生直接支持持久连接功能（persistent）。

http 2.0

提供了HTTP语义优化的传输，spdy :google引入了一个技术，能够加速http数据交互，尤其是使用ssl加速机制，但是spdy现在用的还不多。

目前常用的版本就是http 1.0版本和http 1.1版本。

II.html文本介绍

html文本架构

[回目录](#)

```
<html>
  <head>
    <title>TITLE</title>
  </head>
  <body>
    <h1>H1</h1>
    <p></p>
    <h2>H2</h2>
    <p><a href="admin.html">ToGoogle</a> </p>
  </body>
</html>
```

html文档的生成方式

[回目录](#)

静态

事先就编辑并定义完成的

动态

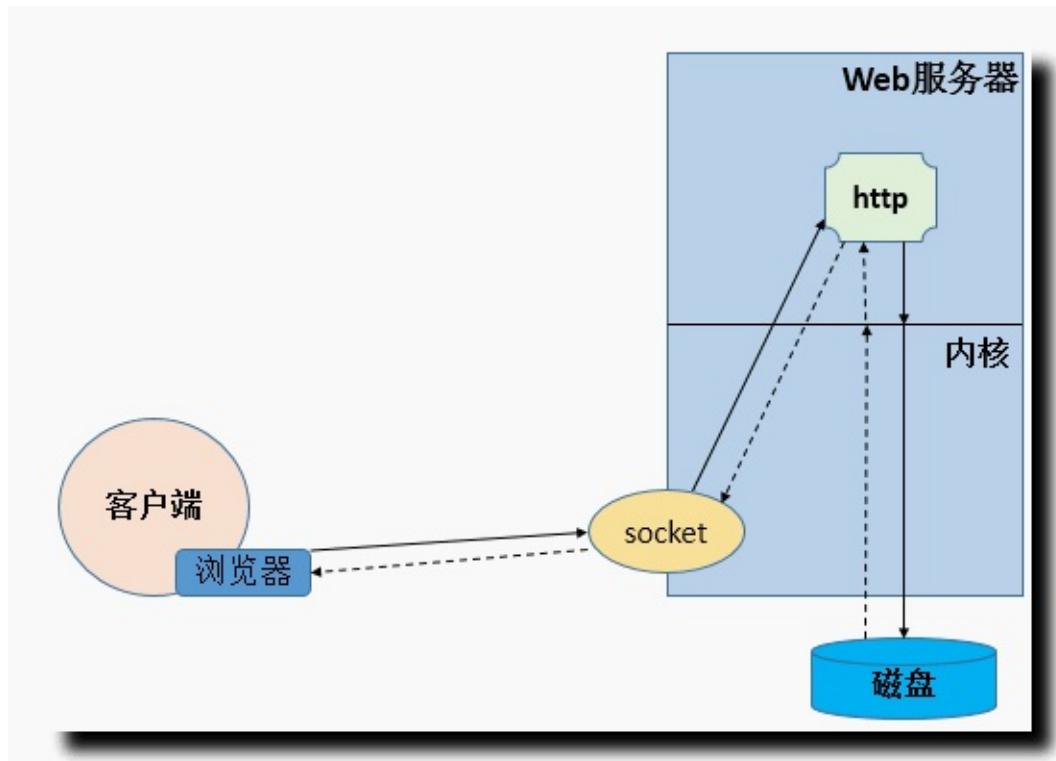
通过编译语言编写的程序后输出html格式的结果

动态语言有：php，jsp，asp，.net

备注：这些脚本都必须有相应的解释器，比如说 php需要有php解释器等等

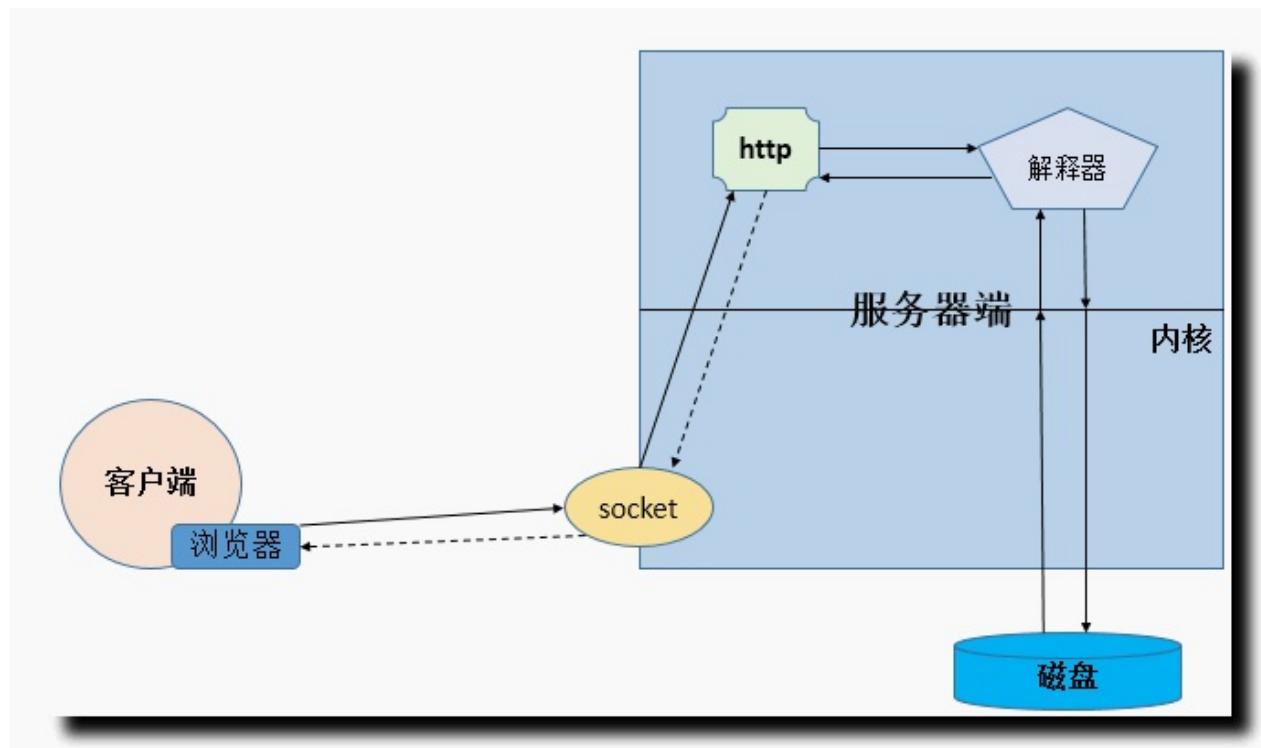
静态和动态的方式

静态



- 1、Web服务器向内核注册socket
- 2、客户端通过浏览器，向Web服务器发起request请求
- 3、Web服务器收到客户端的request信息
- 4、如果用户请求的资源在服务器本地的话，http服务会向系统内核申请调用
- 5、内核调用本地磁盘里的数据，并将数据发给http服务
- 6、http将用户请求的资源通过response报文，最终响应给客户端

动态



与静态不同的是，如果用户请求的是动态内容，那么此时http服务会调用后端的解析器，由动态语言去处理用户的请求，如果需要请求数据的时候，会向内核申请调用，从而向磁盘中获取用户指定的数据，通过解释器运行，运行的结果通常会生成html格式的文件。然后构建成响应报文，最终发回给客户端。

III. http协议

http协议的报文

[回目录](#)

HTTP报文中存在着很多行的内容，一般是由ASCII码串组成，各字段长度是不确定的。HTTP的报文可分为两种：请求报文与响应报文

1.request Message(请求报文)

客户端 → 服务器端

由客户端向服务器端发出请求，不同的网站用于请求不同的资源（html文档）

2.response Message(响应报文)

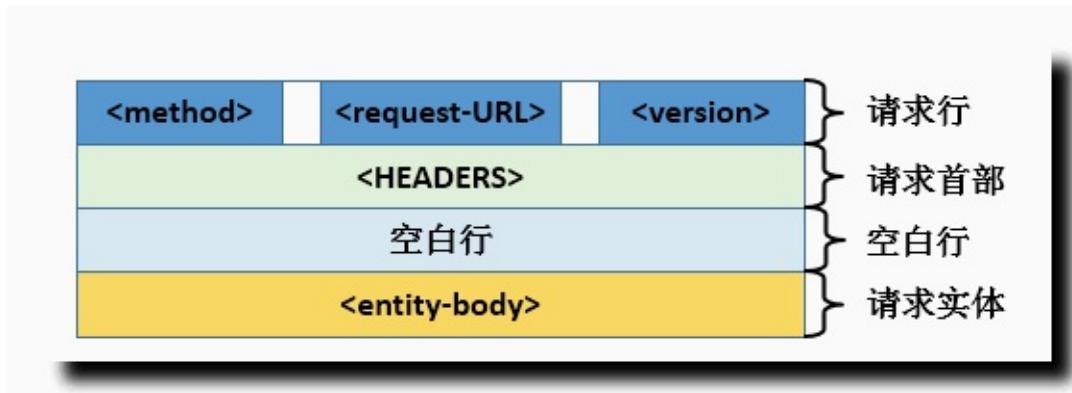
服务器端 → 客户端

是服务器予以响应客户端的请求

请求报文格式介绍

[回目录](#)

请求行 + 请求首部 + 空白行 + 请求实体



例如：

```

<method> <request-URL> <version>
<HEADERS>
# 这里一定要是一个空白行 <entity-body>
    
```

1. 请求行

[回目录](#)

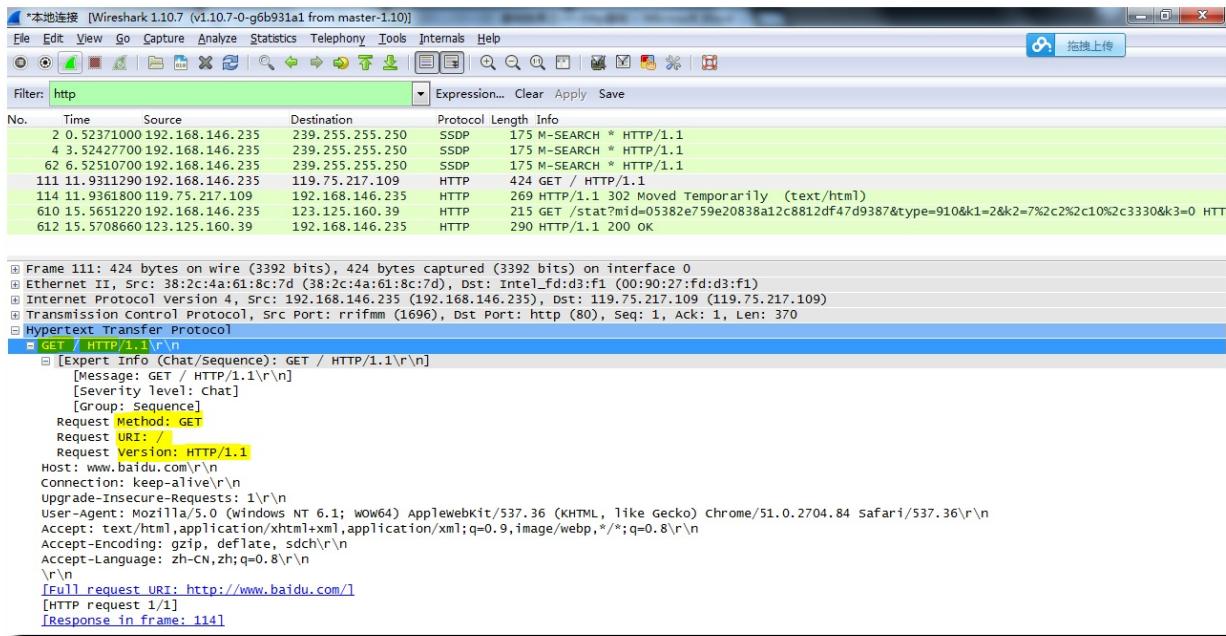
由请求方法字段**<method>**+请求URL字段**<request-URL>**+HTTP协议版本
<version>组成，用来标识客户端请求的资源时使用的请求方法，请求的资源，请求的协议版本是什么，它们直接使用“空格”进行分隔！

<method> 这次请求的方式是什么，也就是请求方法

<request-URL> 请求的是哪个资源，哪个URL。可以是相对路径，
如/images/log.jpg，也可以是绝对路径，如
<http://www.baidu.com/images.banner.jpg>

<version> 请求的协议版本是什么，http协议版本，格式HTTP/<major>.<minor>，
例如：HTTP/1.0，HTTP/1.1

2.HTTP协议分析



上图是用wireshark工具抓取http请求报文的显示结果。在首部后的“\r\n”表示一个回车和换行，以此将该首部与下一个首部隔开。

或者用curl命令获取http请求报文

```
[root@cent01 ~]# curl -I --verbose -X GET "http://www.baidu.com/"
* About to connect() to www.baidu.com port 80 (#0)
*   Trying 119.75.217.109... connected
* Connected to www.baidu.com (119.75.217.109) port 80 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.19.7 (x86_64-redhat-linux-gnu) libcurl/7.19.7 NSS/3.15.3 zlib/1.2.3 libidn/1.18 libssh2/1.4.2
> Host: www.baidu.com
> Accept: */*
```

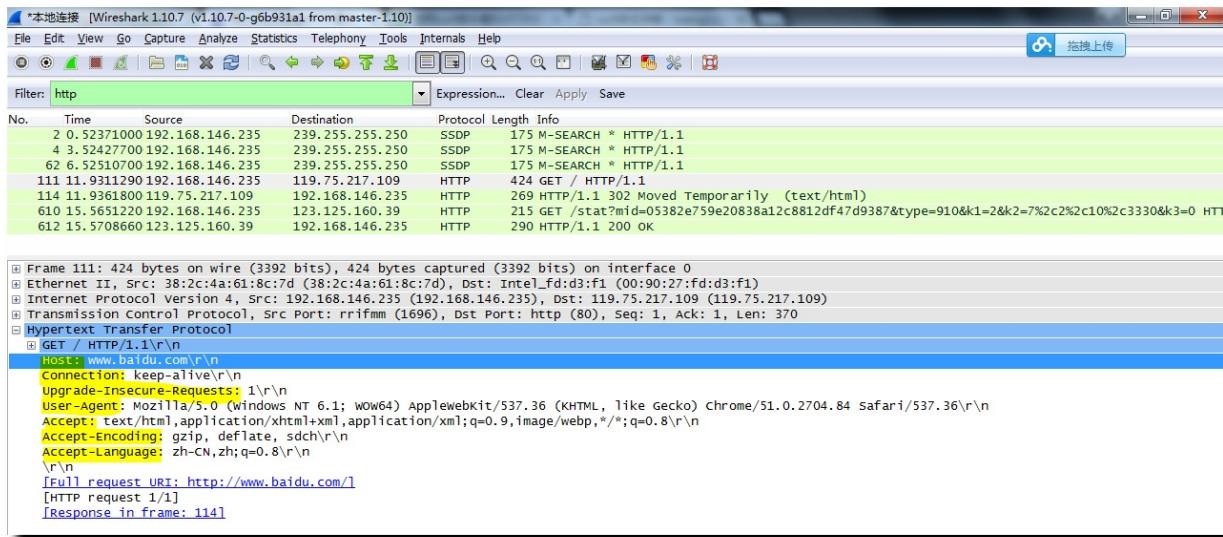
2. 请求首部

[回目录](#)

由关键字+关键字的值组成，之间使用“：“进行分隔，格式Name : Value，请求首部的作用是通过客户端将请求的相关内容告知服务器端，首部可以不止一个。

<HEADERS> 首部，首部可能不止一个。各种所可以使用的首部信息

2.HTTP协议分析



3. 空白行

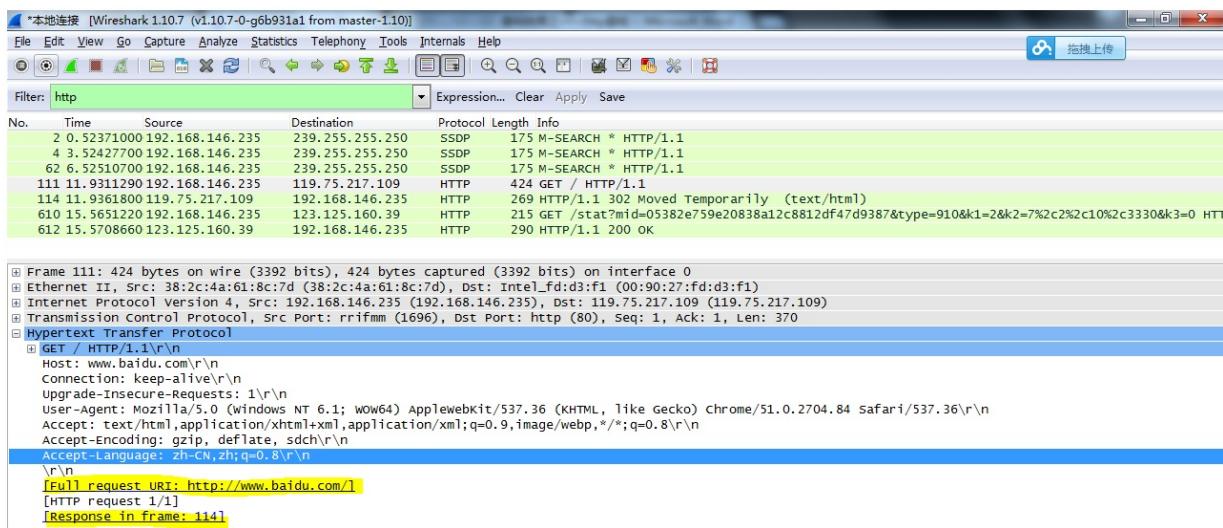
回目录

请求首部之后会有一个空白行，通过发送回车字符和换行符，用于通知服务器端以下的内容将不会再出现请求首部的信息。

4. 请求实体

你需要请求的内容到底是什么

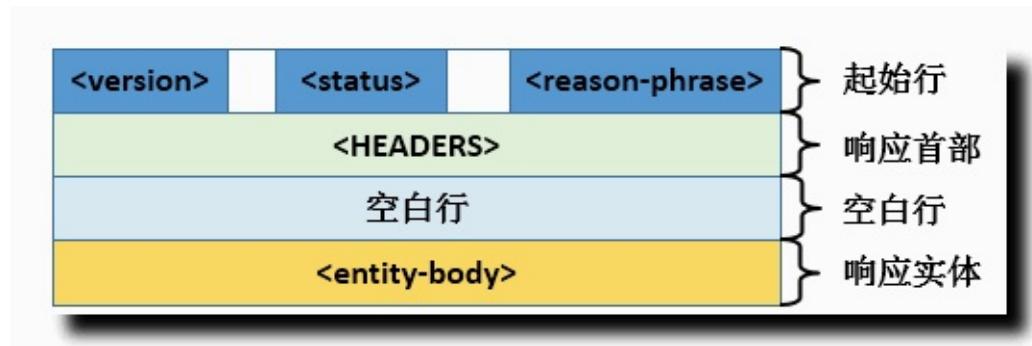
<entity-body> 请求实体，你到底请求的内容是什么



响应报文格式介绍

[回目录](#)

起始行 + 响应首部 + 空白行 + 响应实体



例如：

```
<version> <status> <reason-phrase>

<HEADERS>
# 这里一定要是一个空白行 <entity-body>
```

1. 起始行

[回目录](#)

也称之为状态行，用于服务器端响应客户端请求的状态信息，由版本号<version>+状态码<status> + 原因短语<reason-phrase>组成，例如“HTTP/1.1 200 OK”

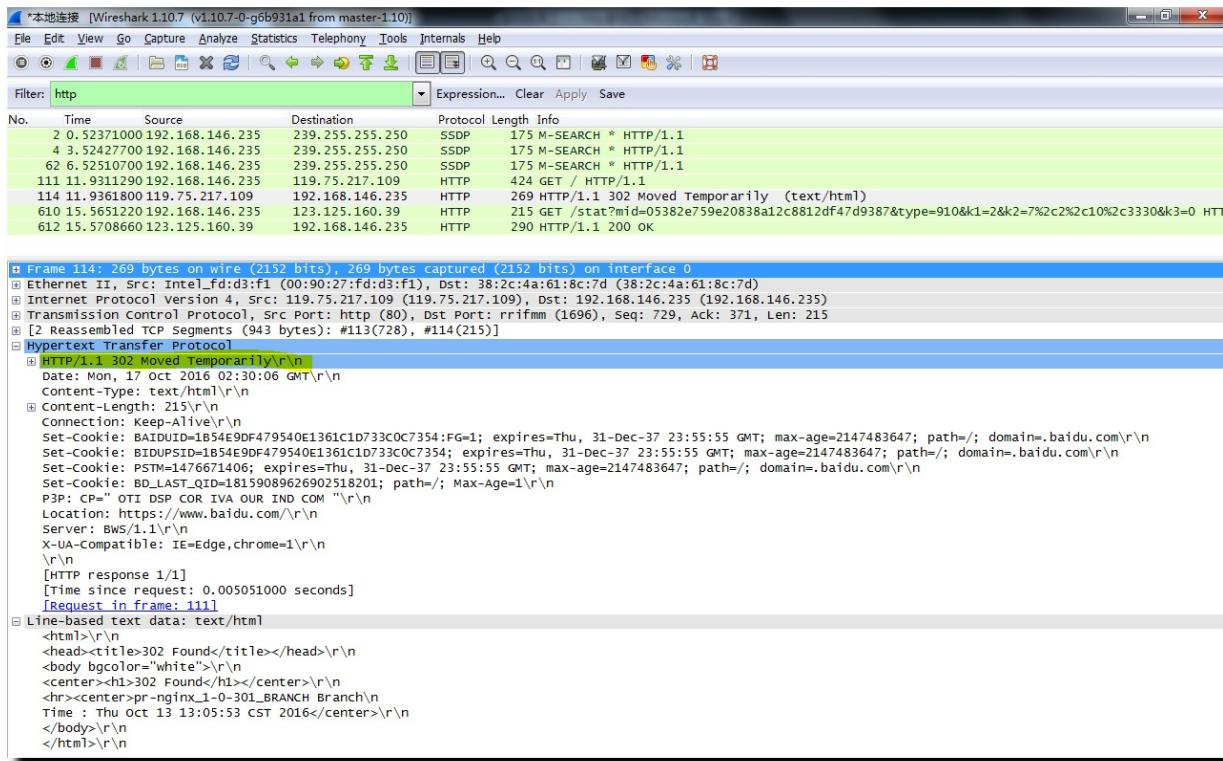
<version> 响应时客户端请求的是什么版本，服务器端就需要响应什么版本

<status> 请求的状态码是什么 202，403等

<reason-phrase> 响应的状态码的信息是什么，原因短语，这个状态码所响应的意义，易读信息

<HEADERS> 一大堆的响应首部

<entity-body> 响应体



2. 响应首部

目录

<version> 响应时客户端请求的是什么版本，服务器端就需要响应什么版本

<status> 请求的状态码是什么 202，403等

<reason-phrase> 响应的状态码的信息是什么，原因短语，这个状态码所响应的意义，易读信息

<HEADERS> 一大堆的响应首部

<entity-body> 响应体

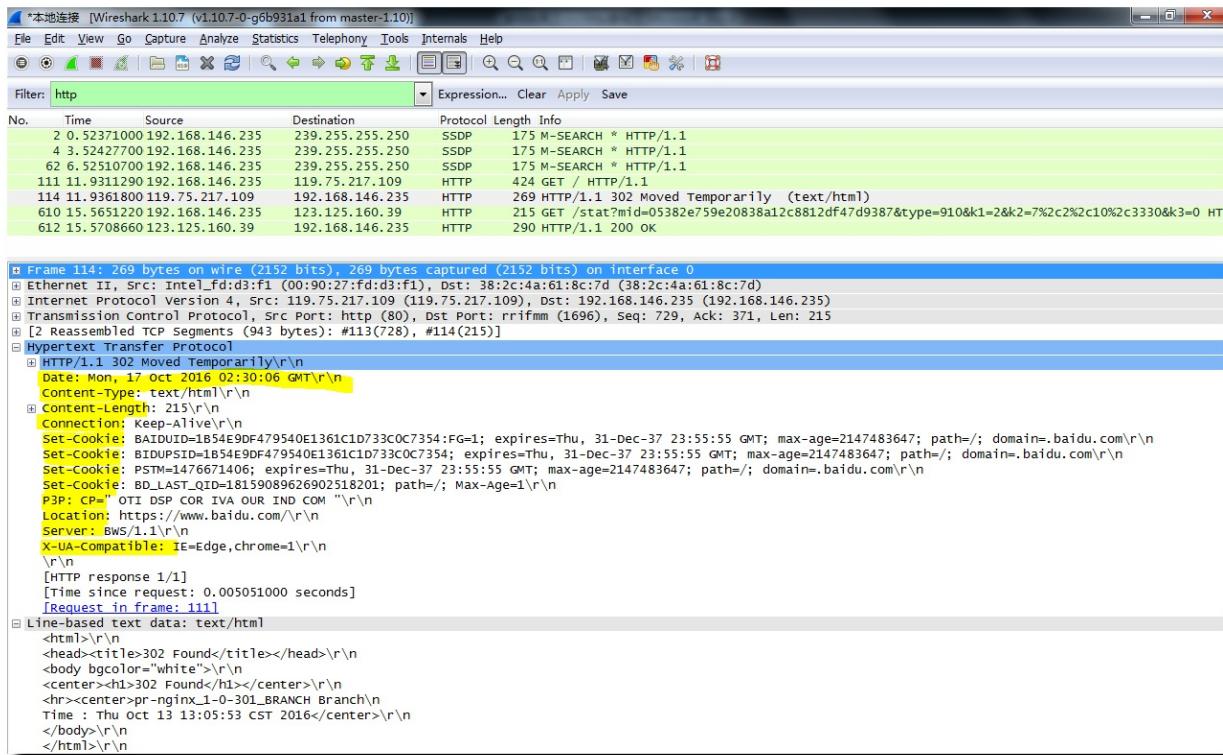
类似请求报文，起始行后面一般有若干个头部字段。每个头部字段都包含一个名字和一个值，两者之间用冒号分割。格式Name : Value。

例如：

Content-Type: test/html; charset=utf-8

Content-Length: 78

2.HTTP协议分析



3. 空白行

[回目录](#)

最后一个响应首部信息之后就是一个空行，通过发送回车符和换行符，通知客户端空行下无首部信息

4. 响应实体

[回目录](#)

响应实体中装载了要返回给客户端的数据。这些数据可以是文本，也可以是二进制（例如图片，视频）

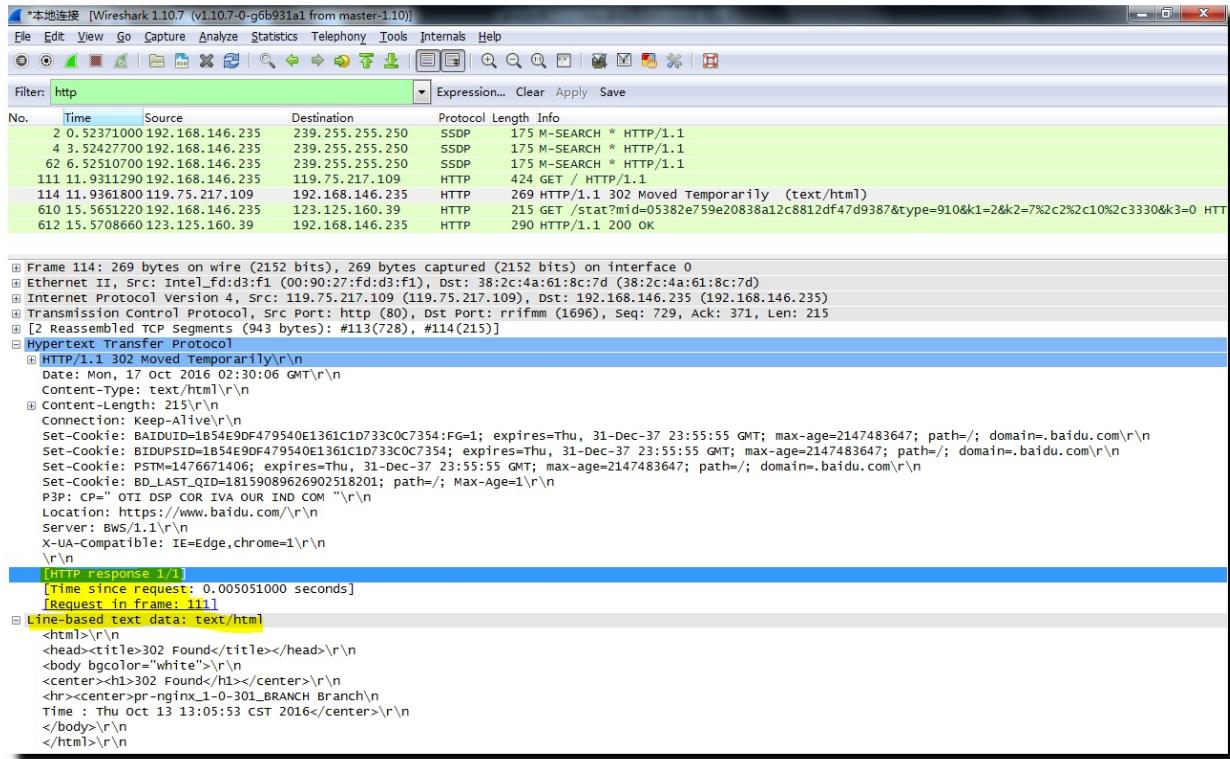
<version> 响应时客户端请求的是什么版本，服务器端就需要响应什么版本

<status> 请求的状态码是什么 202，403等

<reason-phrase> 响应的状态码的信息是什么，原因短语，这个状态码所响应的意义，易读信息

<HEADERS> 一大堆的响应首部

<entity-body> 响应体



HTTP请求方法

回目录

在HTTP通信过程中，每个HTTP请求报文中都会包含一个HTTP请求方法，用于告知客户端向服务器端请求执行某些具体的操作，下面列举几项常用的HTTP请求方法。

HTTP请求方法	描述
GET	用于客户端请求指定资源信息，并返回指定资源实体
HEAD	跟GET相似，但其不需要服务器响应请求的资源，而返回响应首部（只需要响应首部即可，就是告诉我有或者没有，不需要缓存界面给我）
POST	基于HTML表单向服务器提交数据，服务器通常需要存储此数据，通常存放在mysql这种关系型数据库中
PUT	与GET相反，是向服务器发送资源的，服务器通常需要存储此资源（存放的位置通常是文件系统）
DELETE	请求服务器端删除URL指定的资源
MOVE	请求服务器将指定的页面移至另一个网络地址
OPTIONS	探测服务器端对请求的URL所支持使用的请求方法
TRACE	跟一次请求中间所经历的代理服务器、防火墙或网关等。

常用的HTTP请求方式是**GET, POST, HEAD**

HTTP的状态码

[回目录](#)

状态码	说明
1XX	信息性状态码，用于指定客户端相应的某些操作
2XX	成功状态码，我请求一个资源，这个资源在，这就表示请求成功了。
3XX	重定向的状态码，有时会返回的是一个新地址，而非结果
4XX	客户端类错误，你请求的资源不存在，或者你请求的时候，我们这个资源拒绝你访问，你没有权限
5XX	服务器类的错误信息。向服务器发起请求，服务器发现需要运行一个脚本，从而调用解析库。如果在调用过程中出错就会出现这种情况。或者你的脚本有语法错误，也可能导致这个问题。

常用状态码说明

[回目录](#)

状态码	说明
200	服务器成功返回网页，这是成功的HTTP请求返回的标准状态码
201	CREATED 上传文件成功后显示
301	Move Permanently,永久重定向，会返回一个新地址，并告诉我们你所请求的地址将永久挪到那个新地址去了
302	Found,临时重定向，临时放到某个地方，会在响应报文中使用“Location：新位置”；
304	Not Modified，资源没有做任何修改
403	Forbidden 请求被拒绝
404	Not Found 请求的资源不存在
405	Method Not Allowed 你使用的方法不被允许，不支持
500	Internal Server Error：服务器内部错误
502	Bad Gateway，代理服务器从上游服务器收到一条伪响应；上一层服务器返回了一个无法理解的报文，所以代理服务器就会表示错误。
503	Service Unavailable，服务暂时不可用

IV. HTTP首部介绍

[回目录](#)

- 通用首部
- 请求首部
- 响应首部
- 实体首部：专门用来表示实体中资源内部的类型、长度、编码格式等
- 扩展首部：非标准首部，可有程序员自行创建

通用首部

[回目录](#)

- **Connection**：定义C/S之间关于请求、响应的有关选项
在http1.0 的时候，如果他想使用持久连接，那么他所设置的选项即为 Connection : keep-alive
- **Cache-Control**：缓存控制，实现更精细的缓存控制方式。在http 1.1上比较常见

请求头部

[回目录](#)

- **Client-IP** : 客户端 IP地址
- **Host** : 请求的主机，这在实现基于主机名的虚拟主机时很有用
- **Referer** : 指明了请求当前资源原始资源的URL，使用referer是可以防盗链
- **User-Agent** : 用户代理，一般而言是浏览器
- **Accept首部** : 指客户端可以接受哪些编码的类型
 - **Accept** : 服务端能够发送的媒体的类型
 - **Accept-Charset** : 接收的字符集
 - **Accept-Encoding** : 编码格式
 - **Accept-Language** : 所能接受的语言编码格式
- 条件式请求头部：(在http1.1中才会用到)
当发送请求时，先问问对方是否满足条件，如果满足条件就请求，不满足就不请求
- 跟安全相关的请求：
 - **Authorization**
 - **Cookie**

响应头部

[回目录](#)

- **Age** : 资源响应给你之后可以使用的时长
- **Server** : 向客户端说明自己用到的程序名称和版本
- 协商类的首部：
 - **Vary** : 首部列表，服务器会根据此列表挑选最适合的版本发给客户端
- 跟安全相关：
 - **WWW-Authentication**
 - **Set-Cookie**

实体首部

[回目录](#)

- **Location** : 指明资源的新位置，实现302响应码时通常会用到
- **Allow** : 允许对此资源使用的请求方法
- 内容相关的首部
 - **Content-Encoding**
 - **Content-Language**
 - **Content-Length**
 - **Content-Location** : 内容所在的位置
 - **Content-Type**
- 缓存相关：
 - **ETag** : 扩展标签/标记
 - **Expires** : 过期时间
 - **Last-Modified** : 最后修改时间

附：

HTTP最常见的请求头如下：

[回目录](#)

Accept：浏览器可接受的MIME类型；

Accept-Charset：浏览器可接受的字符集；

Accept-Encoding：浏览器能够进行解码的数据编码方式，比如gzip。

Accept-Language：浏览器所希望的语言种类

Authorization：授权信息，通常出现在对服务器发送的WWW-Authenticate头的应答中；

Connection：表示是否需要持久连接。值为“Keep-Alive”，或者看到请求使用的是HTTP 1.1（HTTP1.1默认进行持久连接），它就可以利用持久连接的优点，当页面包含多个元素时（例如Applet，图片），显著地减少下载所需要的时间。

Content-Length：表示请求消息正文的长度；

Cookie：这是最重要的请求头信息之一；

Cookie相关的HTTP扩展头

[回目录](#)

1) Cookie：客户端将服务器设置的Cookie返回到服务器；

2) Set-Cookie：服务器向客户端设置Cookie；

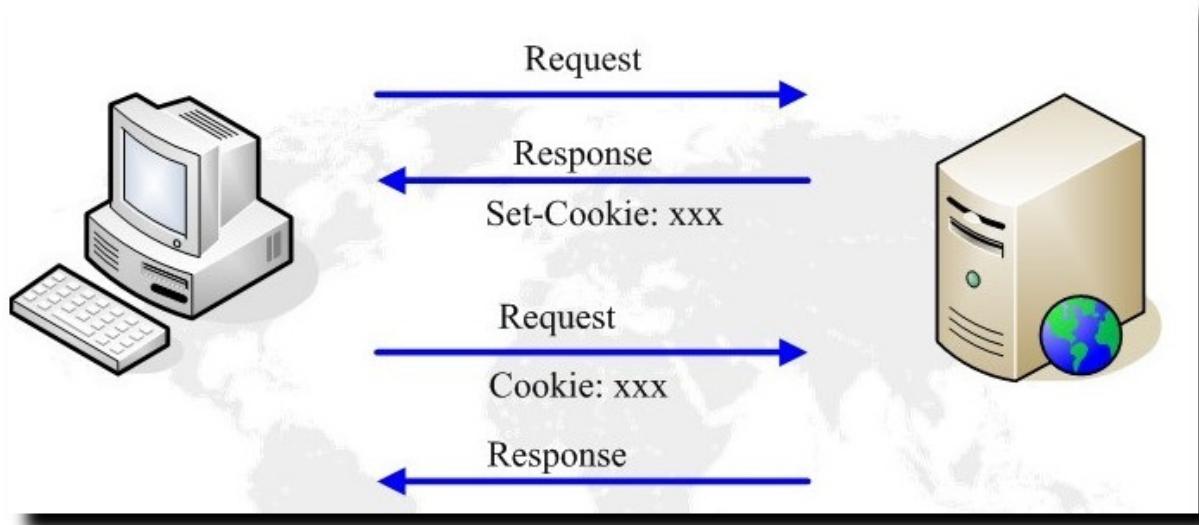
3) Cookie2 (RFC2965)：客户端指示服务器支持Cookie的版本；

4) Set-Cookie2 (RFC2965)：服务器向客户端设置Cookie。

/Cookie的流程

服务器在响应消息中用Set-Cookie头将Cookie的内容回送给客户端，客户端在新的请求中将相同的内容携带在Cookie头中发送给服务器。从而实现会话的保持。

流程如下图所示：



Host：初始URL中的主机和端口；

If-Modified-Since：只有当所请求的内容在指定的日期之后又经过修改才返回它，否则返回304“Not Modified”应答；

Referer：包含一个URL，用户从该URL代表的页面出发访问当前请求的页面。

User-Agent：浏览器类型

HTTP最常见的响应头

HTTP最常见的响应头如下所示：

Allow：服务器支持哪些请求方法（如GET、POST等）；

Content-Encoding：文档的编码（Encode）方法。

Content-Length：表示内容长度。只有当浏览器使用持久HTTP连接时才需要这个数据。

Content-Type：表示后面的文档属于什么MIME类型。

Accept-Ranges: bytes 该响应头表明服务器支持Range请求,以及服务器所支持的单位是字节(这也是唯一的单位).我们还能知道:服务器支持断点续传,以及支持同时下载文件的多个部分,也就是说下载工具可以利用范围请求加速下载该文件.Accept-Ranges: none 响应头表示服务器不支持范围请求.

Date：当前的GMT时间。

Expires：指明应该在什么时候认为文档已经过期，从而不再缓存它。

Last-Modified：文档的最后改动时间。

Location：表示客户应当到哪里去提取文档。

Refresh：表示浏览器应该在多少时间之后刷新文档，以秒计。

HTTP最常见的实体头

[回目录](#)

实体头用作实体内容的元信息，描述了实体内容的属性，包括实体信息类型，长度，压缩方法，最后一次修改时间，数据有效性等。

Allow：GET,POST

Content-Encoding：文档的编码（Encode）方法，例如：gzip，

Content-Language：内容的语言类型，例如：zh-cn；

Content-Length：表示内容长度，eg：80，可参考“2.5响应头”；

Content-Location：表示客户应当到哪里去提取文档，例如：

<<http://www.dfd.org/dfdf.html>，

Content-MD5：MD5 实体的一种MD5摘要，用作校验和。发送方和接受方都计算 MD5摘要，接受方将其计算的值与此头标中传递的值进行比较。

Content-Type：标明发送或者接收的实体的MIME类型。Eg：text/html; charset=GB2312 主类型/子类型；

V.HTTP的事务

[回目录](#)

包含了一个HTTP请求，和对应请求的响应就叫做一个http事务，也可以理解http事务就是一个完整的HTTP请求和HTTP响应的过程。

http协议默认情况下每个事务都会打开和关闭一个新的连接，所以会相当耗费时间和带宽，由于TCP慢启动特性，所以每条新的连接的性能本身就会有所降低，所以

可打开的并行连接的数量上限是有限的。所以使用持久连接这种模式比默认情况下不使用持久连接的方式会好一点，他的好处表现在其请求和tcp断开的过程所消耗的时间会被减少。

HTTP 资源

[回目录](#)

资源就是通过HTTP协议可以让用户通过浏览器或用户代理能够通过基于http协议向服务器端请求并获取的内容，像html文档，一张图片等等。

资源类型：是通过MIME进行标记

格式：major/minor 主标记和次标记

常用的MIME类型

[回目录](#)

MIME类型	文件类型
text/html	html、htm文本类型
text/plain	text文本类型
image/jpeg	jpeg图像类型
image/gif	gif图像类型
video/mpeg4	音频标记类型
application/vnd.ms-powerpoint	动态资源的标记方式

URI和URL

[uri,url区别](#)

[回目录](#)

- URI(Uniform Resource Identifier) 同一资源标示符

用于标识某一互联网资源名称的字符串，通过这种标识来允许你用户对资源可通过特定的协议进行交互操作。在Web上可用的每种资源，包括HTML文档、

图像、视频片段、程序等，由一个通用资源标识符进行定位。所以我们可以使用URI来标识每个资源的名称

- URL (Uniform Resource Locator) (统一资源定位符)

用于描述一个特定服务器上某资源的特定位置。

例如：<http://www.baidu.com:80/download/bash-4.3.1-1.rpm>

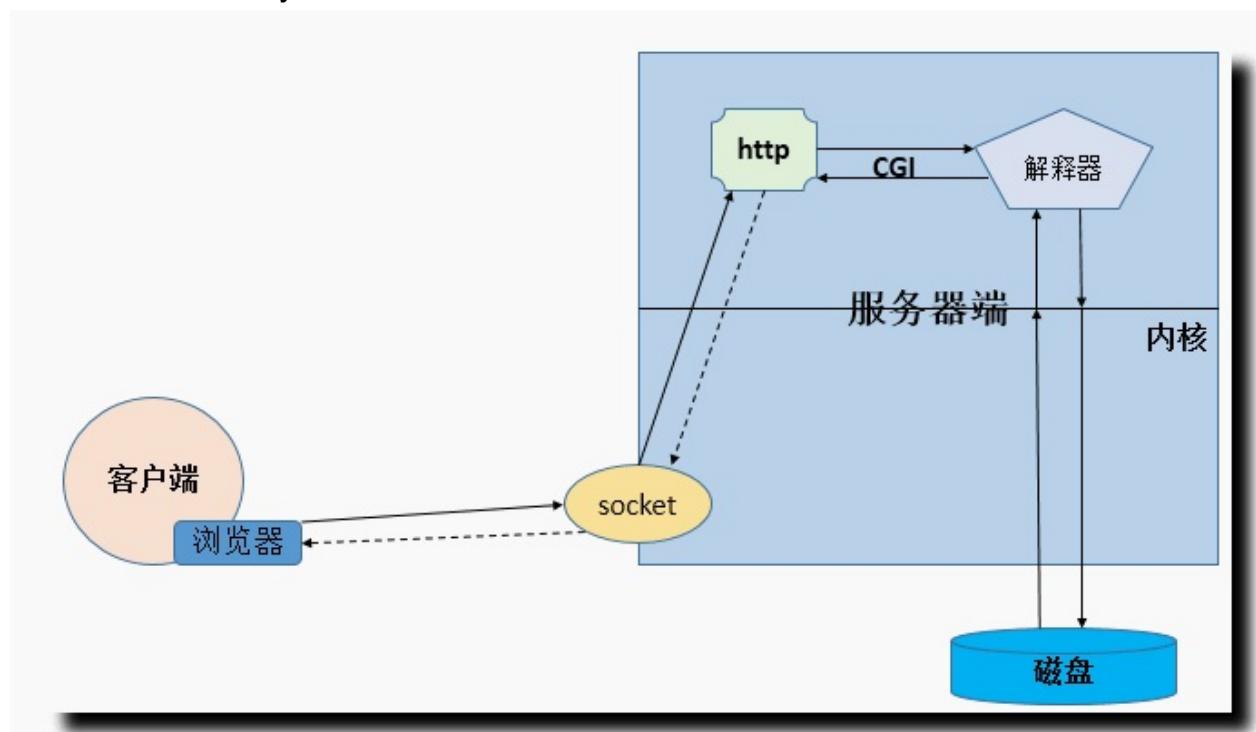
URL的格式分为三个部分

1. scheme (方案) (也叫协议) : http : //
2. Internet地址：一般这个地址指的是服务器:www.baidu.com:8080
3. 特定服务器上的资源 : download/bash-4.3.1-1.rpm

CGI

[回目录](#)

Common Gateway Interface 通用网关接口



web服务器发现需要执行脚本了，就通过CGI协议跟后端的应用程序打交道，把用户的请求动态交给服务器，这个服务器的结果通过CGI协议返回给http服务器。

VI. 其他需要了解的知识

[回目录](#)

一次Web资源请求的具体过程

[回目录](#)

1. 客户端在Web浏览器输入需要访问的地址
2. Web浏览器会请求DNS服务器，查询解析到指定域名和Web服务器的地址
3. 客户端与请求的Web服务器端建立连接（TCP三次握手）
4. TCP建立成功之后，发起HTTP请求
5. 服务器端收到客户端HTTP请求之后，会处理该请求
6. 处理客户端指定请求的资源
7. 服务器构建响应报文，响应给客户端
8. 服务器端将此信息记录到日志中

http如何并发的接收多个用户请求

[回目录](#)

因为http默认是工作在阻塞模型下的，默认一次只接收一个请求，处理完请求后再去接收下一个请求，所以只能一个一个来。

所以我们希望并发响应用户请求，需要多进程模型。web服务器自己会生成多个子进程响应用户请求，也就是说，当一个用户请求发到Web服务器，Web主进程不会直接响应用户请求，而是生成一个子进程响应这个用户请求，这样当子进程和此用户建立连接之后。Web的主进程就会再等待另一个用户的请求，当第二个用户请求过来之后，再生成一个子进程响应第二个用户请求。以此类推。所以每一个用户请求都由一个子进程来处理。

VII. 扩展知识点1：利用wireshark分析HTTP协议

实验步骤

[回目录](#)

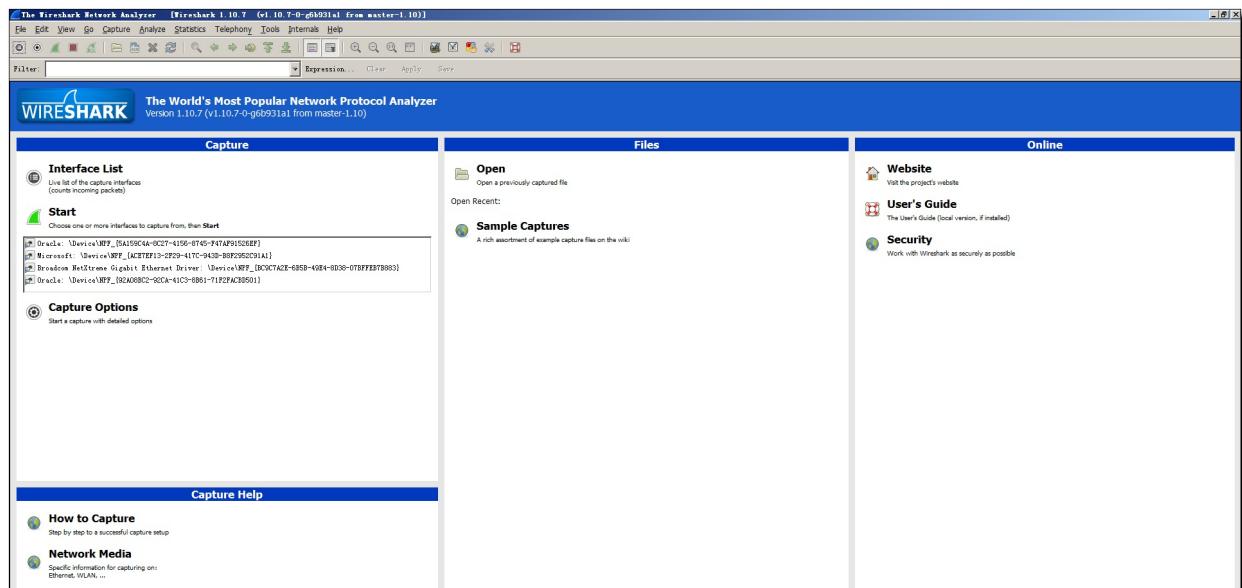
1.清空缓存

[回目录](#)

在进行跟踪之前，我们首先清空Web浏览器的高速缓存来确保Web网页是从网络中获取的，而不是从高速缓冲中取得的。之后，还要在客户端清空DNS高速缓存，来确保Web服务器域名到IP地址的映射是从网络中请求。

2.启动wireshare

[回目录](#)

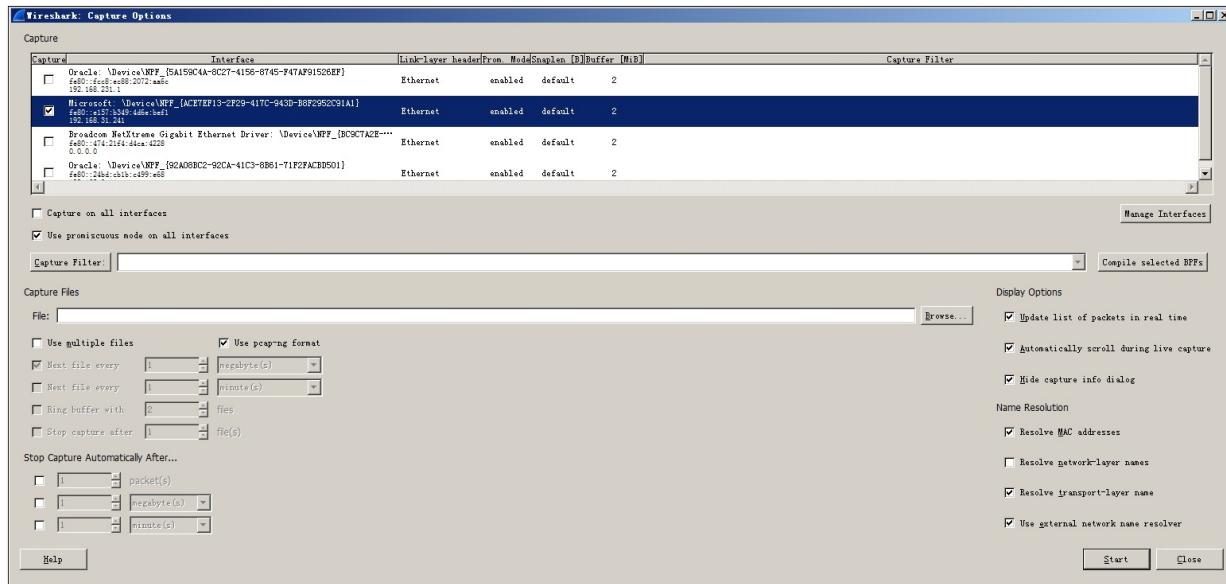


3.开始俘获

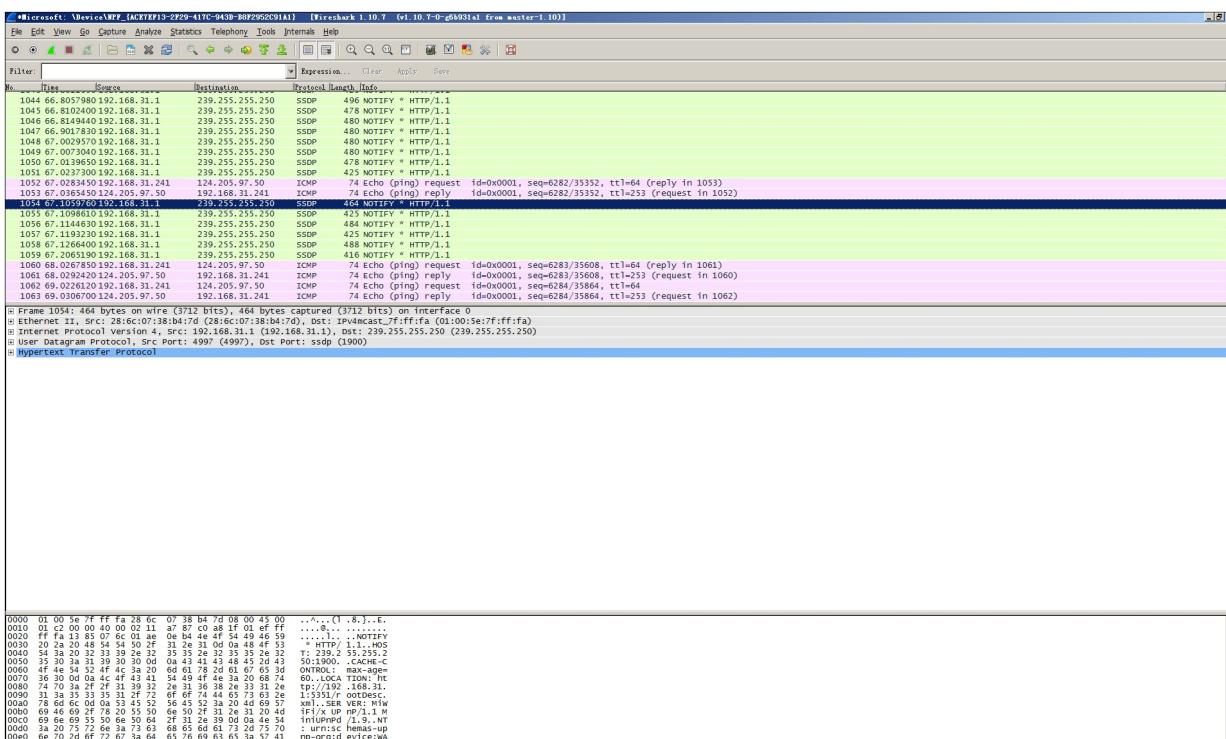
[回目录](#)

1) 在菜单中选择capture-options，选择网络，打开start。如下图：

2.HTTP协议分析

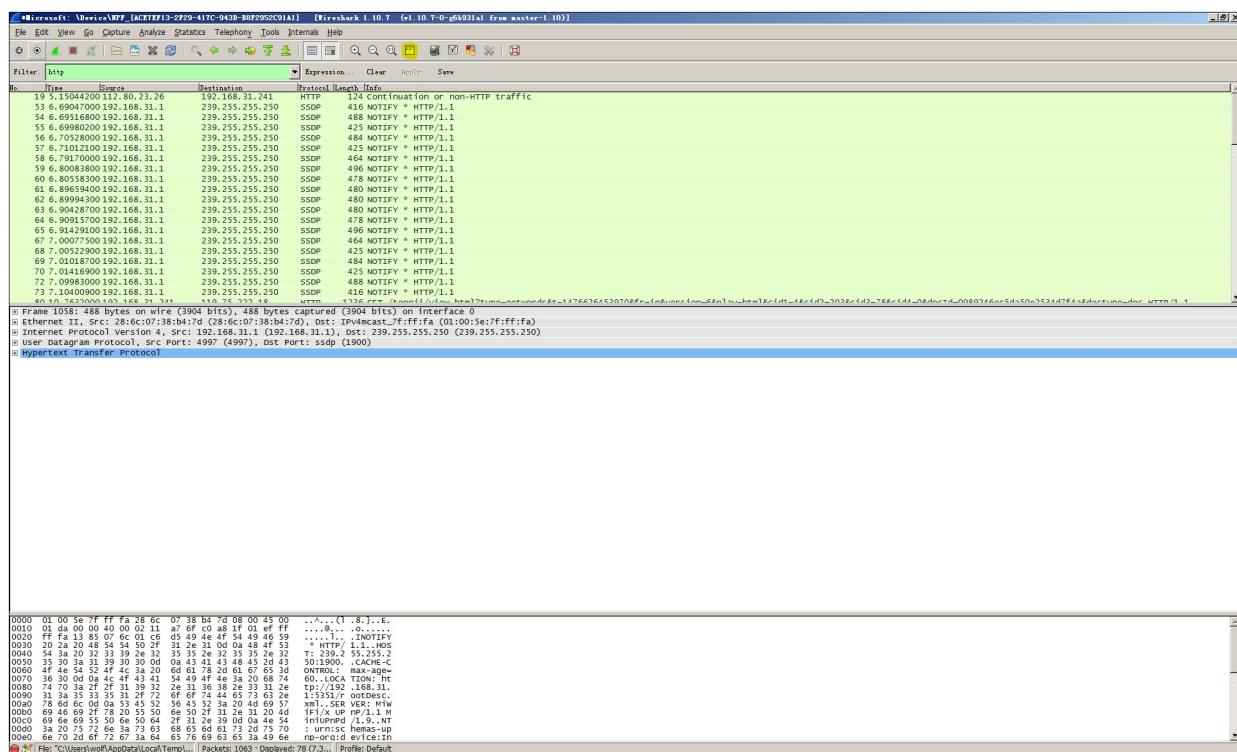


2) 在浏览器地址栏中输入www.baidu.com，然后结束俘获，得到如下结果：



3) 在过滤器中选择HTTP，点击apply，得到如下结果：

2.HTTP协议分析



4. 分析数据

回目录

在协议框中选择“GET/HTTP/1.1”所在的分组会看到这个基本请求行后跟随着一系列额外的请求首部。在首部后的“\r\n”表示一个回车和换行，以此将该首部与下一个首部隔开。

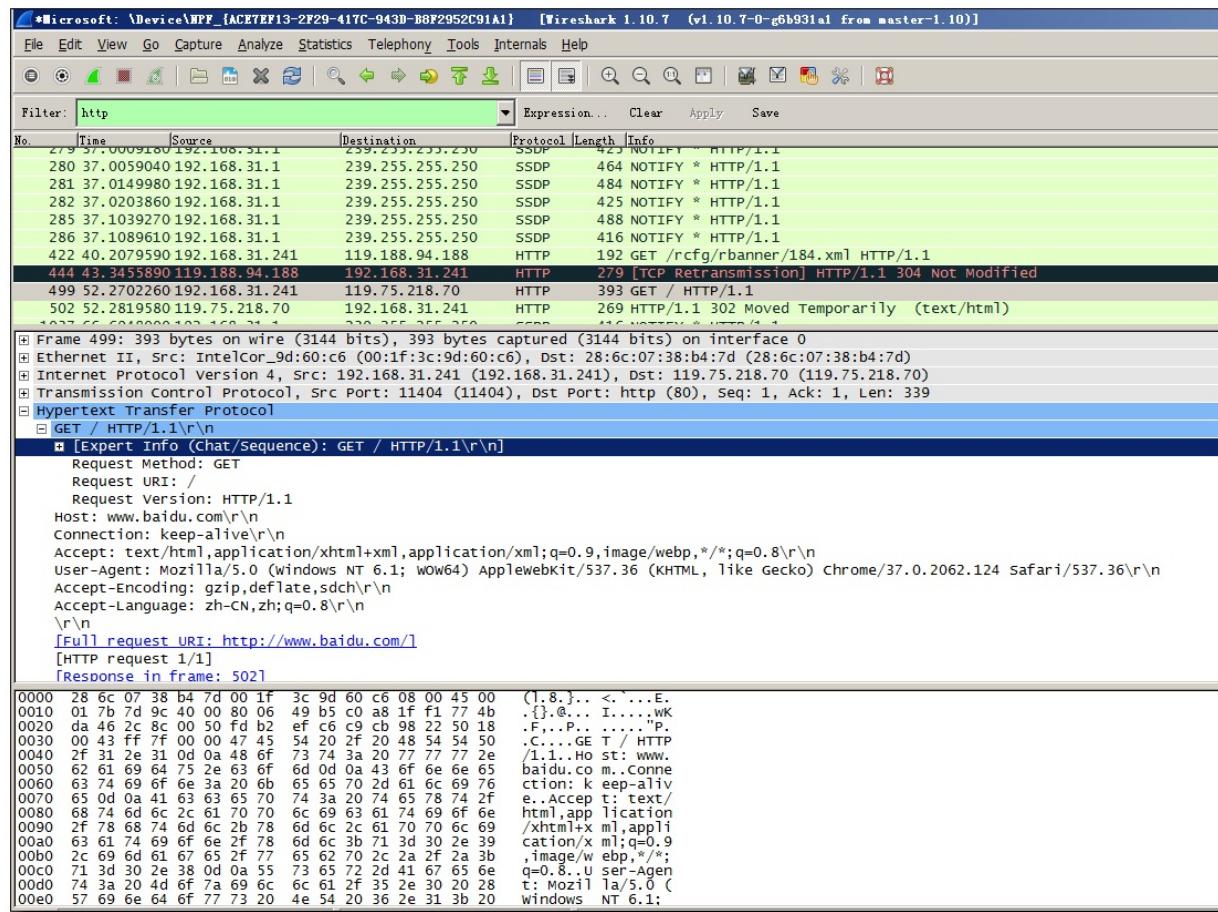
“Host”首部在HTTP1.1版本中是必须的，它描述了URL中机器的域名，本测试中是www.baidu.com。这就允许了一个Web服务器在同一时间支持许多不同的域名。有了这个首部，Web服务器就可以区别客户试图连接哪一个Web服务器，并对每个客户响应不同的内容。

User-Agent首部描述了提出请求的Web浏览器及客户机器。

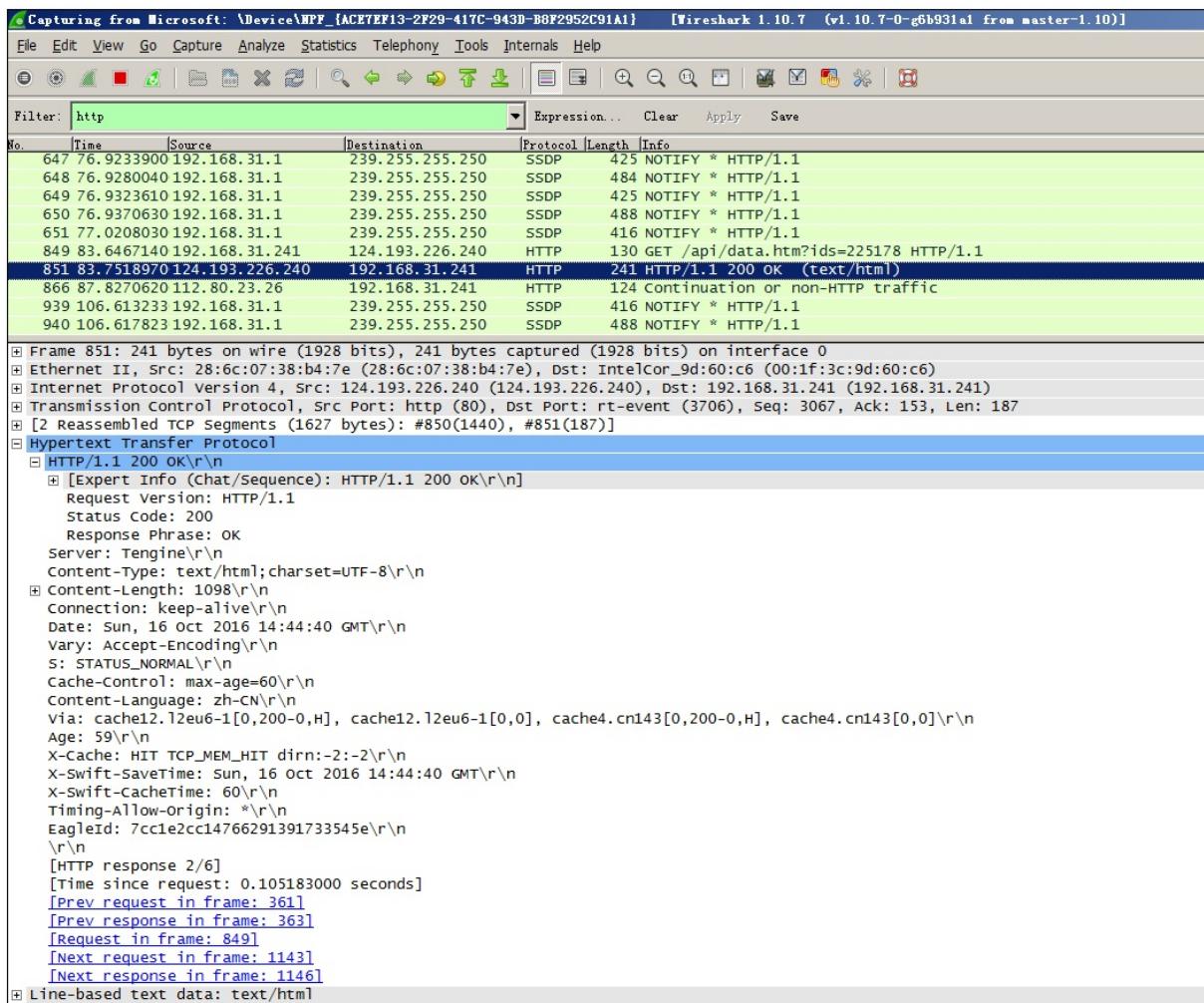
接下来是一系列的Accpet首部，包括Accept（接受）、Accept-Language（接受语言）、Accept-Encoding（接受编码）、Accept-Charset（接受字符集）。它们告诉Web服务器客户Web浏览器准备处理的数据类型。Web服务器可以将数据转变为不同的语言和格式。这些首部表明了客户的能力和偏好。

Keep-Alive及Connection首部描述了有关TCP连接的信息，通过此连接发送HTTP请求和响应。它表明在发送请求之后连接是否保持活动状态及保持多久。大多数HTTP1.1连接是持久的（persistent），意思是在每次请求后不关闭TCP连接，而是

保持该连接以接受从同一台服务器发来的多个请求。



我们已经察看了由Web浏览器发送的请求，现在我们来观察Web服务器的应答。响应首先发送“HTTP/1.1 200 ok”，指明它开始使用HTTP1.1版本来发送网页。同样，在响应分组中，它后面也跟随着一些首部。最后，被请求的实际数据被发送。第一个Cache-control首部，用于描述是否将数据的副本存储或高速缓存起来，以便将来引用。一般个人的Web浏览器会高速缓存一些本机最近访问过的网页，随后对同一页面再次进行访问时，如果该网页仍存储于高速缓存中，则不再向服务器请求数据。在HTTP请求中，Web服务器列出内容类型及可接受的内容编码。此例中Web服务器选择发送内容的类型是text/html



VIII. 扩展知识点2：curl查看HTTP响应头信息

回目录

先看看客户端（浏览器）从服务器请求数据经历如下基本步骤：

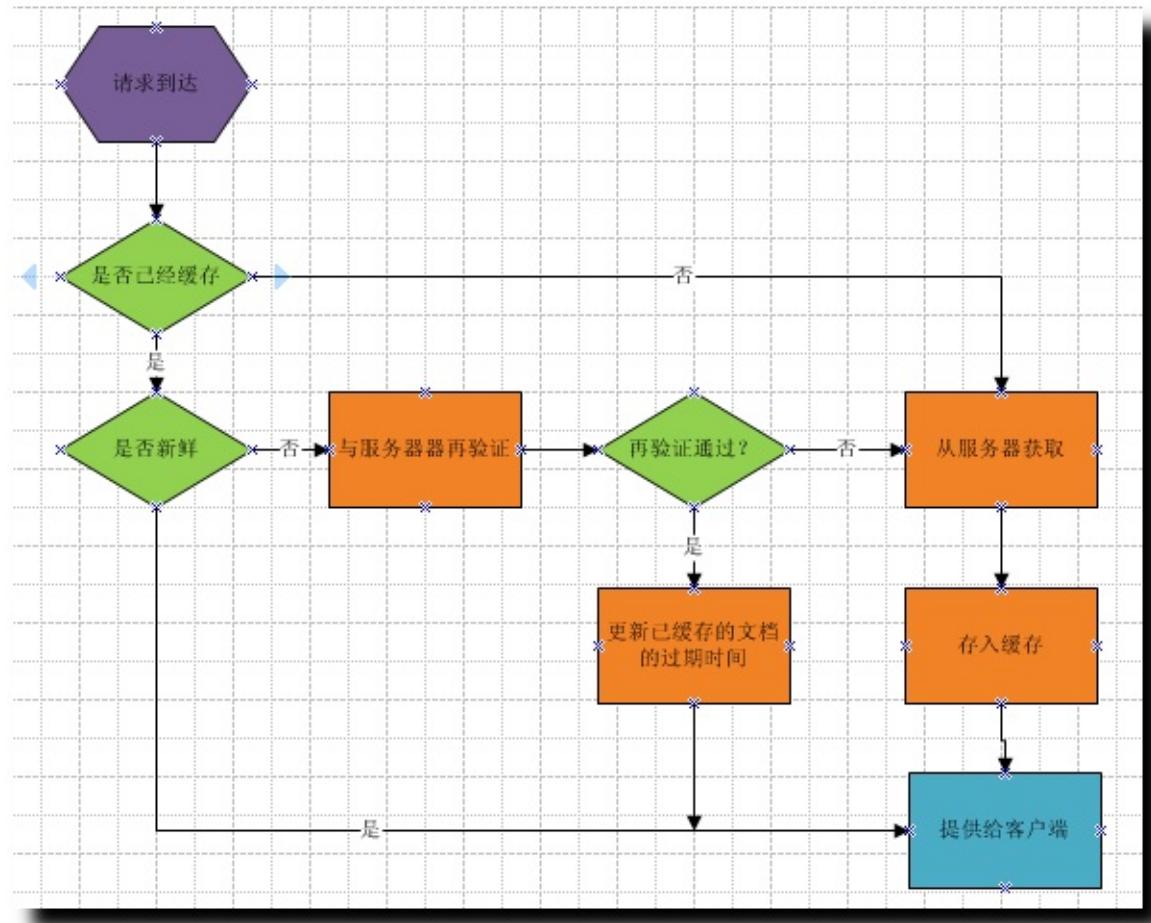
1. 用户发起一个http请求，缓存获取到URL，根据URL查找是否有匹配的副本，这个副本可能在内存中，也可能在本地磁盘。

2、如果请求命中本地缓存则从本地缓存中获取一个对应资源的"copy"；

3、检查这个"copy"是否过期，否则直接返回，是则继续向服务器转发请求。HTTP中，通过Cache-Control首部和Expires首部为文档指定了过期时间，通过对过期时间的判断，缓存就可以知道文档是不是在保质期内。Expires首部和Cache-Control:max-age首部都是来告诉缓存文档有没有过期，为什么需要两个响应首部来做这件简单的事情了？其实这一切都是历史原因，Expires首部是HTTP 1.0中提出来的，因为他使用的是绝对日期，如果服务端和客户端时钟不同步的话（实际上这

种情况非常常见），缓存可能就会认为文档已经过了保质期。

4、服务器接收到请求，然后判断资源是否变更，是则返回新内容，否则返回304，未变更，更新过期时间。



```
[root@docker ~]# curl -I http://www.baidu.com/
HTTP/1.1 200 OK
Server: bfe/1.0.8.18
Date: Sun, 16 Oct 2016 15:07:17 GMT
Content-Type: text/html
Content-Length: 277
Last-Modified: Mon, 13 Jun 2016 02:50:09 GMT
Connection: Keep-Alive
ETag: "575e1f61-115"
Cache-Control: private, no-cache, no-store, proxy-revalidate, no-transform
Pragma: no-cache
Accept-Ranges: bytes
```

HTTP 响应头的信息

[回目录](#)

/span(1). HTTP 返回码：

1xx : client的请求server已经接收，正在处理

2xx : 成功 表示 client请求，server端已经接收、理解并处理

3xx : client 请求被重定向其他的server【其他的URL】

4xx : 表示client请求不正确，server不能识别

5xx : server端服务不正常

(2). Cache-Control:

web 站点对缓存的设置： Cache-Control指定请求和响应遵循的缓存机制

缓存分类

[回目录](#)

1) 私有缓存：常见就是我们的浏览器里内置的缓存。

2) 公有缓存：常见的就是代理缓存

先看**Cache-Control**可选的参数:private、public、no-cache、max-age、must-revalidate等/span

no-cache : 响应不会被缓存,而是实时向服务器端请求资源

no-store : 在任何条件下，响应都不会被缓存，并且不会被写入到客户端的磁盘里，这也是基于安全考虑的某些敏感的响应才会使用这个。

Private : 指示对于单个用户的整个或部分响应消息，不能被共享缓存处理。这允许服务器仅仅描述当前用户的部分响应消息，此响应消息对于其他用户的请求无效。不能再用户间共享。

Public : 响应会被缓存，并且在多用户间共享。正常情况，如果要求 [HTTP 认证](#), 响应会自动设置为 private。

max-age : 指示客户机可以接收生存期不大于指定时间（以秒为单位）的响应，例如： Cache-control: max-age=5 表示当访问此网页后的5秒内再次访问不会去服务器。

must-revalidate : 响应在特定条件下会被重用，以满足接下来的请求，但是它必须到服务器端去验证它是不是仍然是最新的（强制所有缓存都验证响应）。

proxy-revalidate : 类似于 **must-revalidate**, 它要求对公共缓存进行验证

/

(3). Connection:

server 是否支持长连接；如果**keep-alive** 说明web的server 支持长连接。

但是TCP 的长连接是双向的；必须是**client**和**server** 都支持长连接；才可以建立长连接。

一般**client**【浏览器】都是默认支持长连接；所以只要**server**端支持长连接；就可以建立长连接。通过curl的-w参数我们可以自定义curl的输出，%{http_code}代表http状态码

```
[root@docker ~]# curl -I -o /dev/null -s -w %{http_code}"\n" www.baidu.com
200
```

[回目录](#)

apache 优化

目录

- I. 安装 apache2.4.23
 - i. 新版本的 httpd-2.4 新增以下特性
 - 1. 新增模块
 - 2. 新特性
 - ii. 安装 apache2.4.23
 - 1. 检查是否安装 httpd
 - 2. 安装 apr 和 apr-util
 - 3. 安装 zlib
 - 4. 安装 pcre
 - 5. 安装 openssl
 - 6. 安装 apache2.4.23
 - 6.1 相关参数解释
 - 7. 优化 http 程序执行路径
 - 8. 开启 apache 服务器
 - 8.1 apache 优雅启动
 - 9. 开机后自动启动
 - 10. 将 Apache 加入开机自动启动
 - 11. 启动编译好的 Apache 2.4.23
- II. Apache 的优化配置
 - i. apache 的工作模式
 - 1. Prefork 模式（一个非线程型的）：
 - 1.1 Apache 在 prefork 工作模式下影响性能的重要参数说明
 - 1.2 apache 查看编译参数
 - 2. worker 模式（多线程多进程）：
 - 2.1 Apache 在 worker 工作模式下影响性能的重要参数说明
 - 2.2 线程与进程的区别
 - 3. Event 模式：
 - ii. apache 配置参数
 - 1. KeepAlive On/Off
 - 2. KeepAliveTimeOut number

- 3. MaxKeepAliveRequests 100
 - 4. HostnameLookups on|off|double
 - 5. timeout 5
- iii. MPM 这个比较关键是影响并发效率的主要因素
 - 1. StartServers 10
 - 2. MinSpareServers 10
 - 3. MaxSpareThreads 75
 - 4. ServerLimit 2000
 - 5. MaxClients / MaxRequestWorkers 256
 - 6. MaxRequestsPerChild / MaxConnectionsPerChild 0
 - iv. 开启 apache 的 Gzip (deflate) 功能
 - 1. 未使用 Gzip
 - 2. 开启使用 Gzip
 - 2.1 如果要开启 deflate 的话,一定要打开下面二个模块
 - 2.2 mod_deflate 模块检查及安装
 - 3. 编译时安装方法
 - 4. DSO 方式安装(适用于已经编译安装完成的场景)
 - 5. 修改 Apache 配置文件开启 gzip 压缩传输
 - 5.1 http.conf 修改、增加配置参数
 - 6. 实际测试结果图
 - 7. 查看 deflate_log 日志
 - v. 配置 mod_expires 模块
 - 1. 未启用 expire 的效果
 - 2. 启用 expire 缓存
 - 2.1 启用 expires_module
 - 2.2 然后添加 Expires 配置规则
 - 3. 验证
 - vi. Apache 禁止目录遍历
 - vii. apache 隐藏版本信息
 - 1、主配置中启用 httpd-default.conf
 - 2、修改 httpd-default.conf
 - viii. Apache 日志切割
 - 1. 为什么要分割日志
 - 2. rotatelogs 使用方法
 - 3. cronolog 使用方法
 - 4. 扩展

- ix. 配置防盗链

- 1. rewrite 实现防盗
- 1.1 相关选项的解释
- 2. 防盗链配置的说明
- 2.1 超链接测试
- 2.2 超链接语法
- 3. SetEnvIfNoCase 和 access 实现防盗
- 4. 2.4 版本以下
- 5. 2.4 版本以上

I. 安装 apache2.4.23

i. 新版本的 httpd-2.4 新增以下特性

1. 新增模块

[回目录](#)

- mod_proxy_fcgi (可提供fcgi代理)
- mod_ratelimit (限制用户带宽)
- mod_request (请求模块，对请求做过滤)
- mod_remoteip (匹配客户端的IP地址)

对于基于IP的访问控制做了修改，不再支持 allow , deny , order 机制，而是统一使用 require 进行

2. 新特性

[回目录](#)

- 1、MPM 支持在运行时装载;不过要开启这种特性，在编译安装要启用这三种功能； --enable-mpms-shared=all --with-mpm=event
- 2、支持 event
- 3、支持异步读写
- 4、在每个模块及每个目录上指定日志级别

- 5、增强版的表达式分析器
- 6、每请求配置：`<If>`，`<Elseif>`
- 7、毫秒级别的 `keepalive timeout`
- 8、基于 `FQDN` 的虚拟主机不再需要 `Name VirtualHost` 指令
- 9、支持使用自定义变量

ii. 安装 apache2.4.23

[回目录](#)

安装环境：操作系统：`Centos7.2`，关闭 `selinux`

[回目录](#)

1. 检查是否安装 httpd

[回目录](#)

检查 `httpd` 包是否安装，如查安装则卸载

```
[root@www ~]# rpm -q httpd
```

下载源码包：

```
httpd-2.4.23.tar.gz  
apr-1.5.2.tar.gz  
apr-util-1.5.4.tar.gz  
zlib-1.2.8.tar.gz  
pcre-8.39.tar.gz
```

注：`apr`(Apache Portable Runtime)Apache可移植运行库，它是一个对操作系统调用的抽象库，用来实现Apache内部组件对操作系统的使用，提高系统的可移植性。

2. 安装 apr 和 apr-util

[回目录](#)

```
[root@www ~]# tar zxf apr-1.5.2.tar.gz
[root@www ~]# cd apr-1.5.2/
[root@www apr-1.5.2]# ./configure --prefix=/usr/local/apr
[root@www apr-1.5.2]# make && make install
[root@www ~]# tar zxf apr-util-1.5.4.tar.gz
[root@www ~]# cd apr-util-1.5.4/
[root@www apr-util-1.5.4]# ./configure --prefix=/usr/local/apr-u
til --with-apr=/usr/local/apr
[root@www apr-util-1.5.4]# make && make install
```

3. 安装 zlib

[回目录](#)

```
[root@www ~]# tar zxf zlib-1.2.8.tar.gz
[root@www ~]# cd zlib-1.2.8/
[root@www zlib-1.2.8]# ./configure --prefix=/usr/local/zlib
[root@www zlib-1.2.8]# make && make install
```

4. 安装 pcre

[回目录](#)

```
[root@www ~]# tar zxf pcre-8.39.tar.gz
[root@www ~]# cd pcre-8.39/
[root@www pcre-8.39]# ./configure --prefix=/usr/local/pcre
[root@www pcre-8.39]# make && make install
```

5. 安装 openssl

安装 apache2.4.23 时提示 openssl 版本过低，centos7自带版本 openssl-1.0.1e 下载 openssl

```
# wget https://www.openssl.org/source/openssl-1.0.1u.tar.gz
```

```
[root@www ~]# tar zxf openssl-1.0.1u.tar.gz
[root@www ~]# cd openssl-1.0.1u/
[root@www openssl-1.0.1u]# ./config -fPIC --prefix=/usr/local/openssl enable-shared
[root@www openssl-1.0.1u]# make && make install
[root@www ~]# mv /usr/bin/openssl /usr/bin/openssl.1.0.1e
[root@www ~]# ln -s /usr/local/openssl/bin/openssl /usr/bin/openssl
```

6. 安装 apache2.4.23

[回目录](#)

```
[root@www ~]# tar zxf httpd-2.4.23.tar.gz
[root@www ~]# cd httpd-2.4.23/
[root@www httpd-2.4.23]# ./configure --prefix=/usr/local/httpd-2.4.23 --enable-so --enable-cgi --enable-cgid --enable-ssl --with-ssl=/usr/local/openssl --enable-rewrite --with-pcre=/usr/local/pcre --with-z=/usr/local/zlib --with-apr=/usr/local/apr --with-apr-util=/usr/local/apr-util --enable-modules=most --enable-mods-shared=most --enable-mpms-shared=all --with-mpm=event --enable-proxy --enable-proxy-fcgi --enable-expires --enable-deflate
```

6.1 相关参数解释

[回目录](#)

[安装拓展](#)

参数	作用
--enable-so :	支持动态共享模块（即打开DSO支持）
--enable-rewrite :	支持url重写
--enable-ssl :	支持ssl
--with-ssl=/usr/local/openssl :	指定ssl安装位置
--enable-cgi :	启用cgi

--enable-cgid:MPM	使用的是event或worker要启用cgid
--enable-modules=most :	明确指明要静态编译到httpd二进制文件的模块，<MODULE-LIST> 为空格分隔的模块名列表、all或者most，all表示包含所有模块，most表示包含大部分常用模块
--enable-mods-shared=most :	明确指明要以DSO方式编译的模块，<MODULE-LIST> 为空格分隔的模块名列表、all或者most，all表示包含所有模块，most表示包含大部分模块
--enable-mpms-shared=all :	启用MPM所有支持的模式，这样event、worker、prefork就会以模块化的方式安装，要用哪个就在httpd.conf里配置就好了。 多处理动态模块
--with-mpm=event :	指定启用的mpm模式，默认使用event模式，在apache的早期版本2.0默认prefork,2.2版本是worker，2.4版本是event.
--with-pcre=/usr/local/pcre :	支持pcre
--with-z=/usr/local/zlib :	使用 zlib压缩库
--with-apr=/usr/local/apr :	指定apr的安装路径
--with-apr-util=/usr/local/apr-util :	指定apr-util的安装路径
--enable-expire :	激活或通过配置文件控制HTTP的“Expires:”和“Cache-Control:”头内容，即对网站图片、js、css等内容，提供客户端浏览器缓存的设置。这是apache调优的一个重要选项之一。
--enable-deflate :	提供对内容的压缩传输编码支持，一般是html、js、css等内容的站点。使用此参数会大大提高传输速度，提升访问者访问的体验。在生产环境中，这是apache调优的一个重要选项之一。

--enable-mpms-shared=all : This switch ensures that all MPM (Multi Processing Modules) are built as Dynamic Shared Objects (DSOs), so the user can choose which one to use at runtime.

Dynamic Shared Object (DSO) Support

```
[root@www httpd-2.4.23]# make && make install
```

7.优化 http 程序执行路径

[回目录](#)

```
[root@www httpd-2.4.23]# ln -s /usr/local/http-2.4.23/bin/* /usr/local/bin/
```

修改配置文件 `httpd.conf`，设置其中的 `ServerName` 值 例如：`ServerName www.benet.com`

8.开启apache服务器

[回目录](#)

```
# /usr/local/http-2.4.23/bin/apachectl start
```

8.1 apache 优雅启动

```
[root@nfs ~]# apachectl stop
[root@nfs ~]# netstat -anptlu | grep 80
[root@nfs ~]# apachectl graceful
httpd not running, trying to start
[root@nfs ~]# apachectl graceful
```

9.开机后自动启动

[回目录](#)

```
[root@www httpd-2.4.23]# cp /usr/local/http-2.4.23/bin/apachectl
/etc/init.d/httpd
```

编辑 `/etc/init.d/httpd` 文件，在首行 `#!/bin/sh` 下面加入两行：

```
[root@www httpd-2.4.23]# vi /etc/init.d/httpd
# chkconfig: 35 85 15
# description: apache 2.4.23
```

10.将 **Apache** 加入开机自动启动

[回目录](#)

```
[root@www httpd-2.4.23]# chkconfig --add httpd
[root@www httpd-2.4.23]# chkconfig httpd on
```

11.启动编译好的 **Apache 2.4.23**

[回目录](#)

```
[root@www httpd-2.4.23]# service httpd start
[root@www httpd-2.4.23]# netstat -anplt | grep 80
tcp6       0      0 ::::80          ::::*           LISTEN
4807/httpd
```

客户端测试访问（注意防火墙）

II. **Apache** 的优化配置

[回目录](#)

apache所运行的硬件环境都是对性能影响最大的因素，即使不能对硬件进行升级，也最好给apache一个单独的主机以免受到其他应用的干扰。各个硬件指标中，对性能影响最大的是**内存**，对于静态内容（图片、javascript文件、css文件等），它决定了apache可以缓存多少内容，它缓存的内容越多，在硬盘上读取内容的机会就越少，大内存可以极大提高静态站点的速度；对动态高负载站点来说，每个请求保存

的时间更多一些，apache的 mpm 模块会为每个请求派生出相应的进程或线程分别处理，而进程或线程的数量与内存的消耗近似成正比，因此增大内存对提高动态站点的负载和运行速度也极为有利

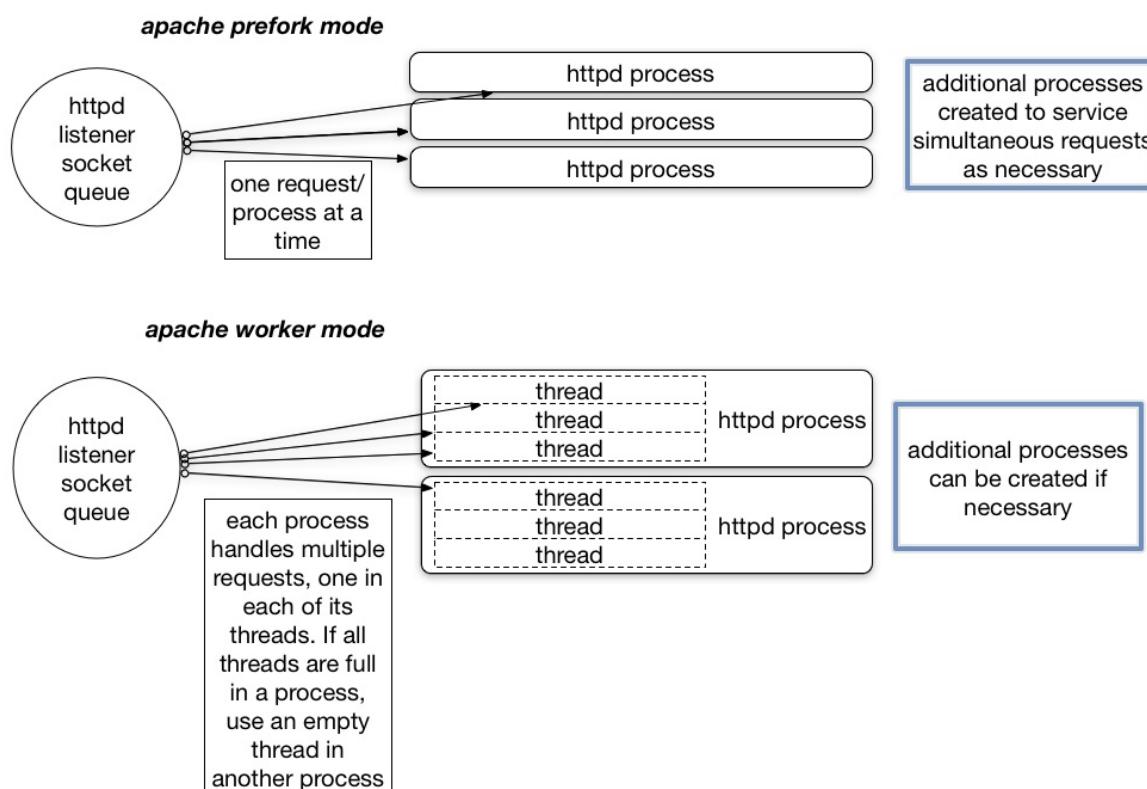
其次是硬盘的速度，静态站点尤为突出，apache不断的在读取文件并发送给相应的请求，硬盘的读写是极其频繁的；动态站点也要不断的加载web程序（php等），一个请求甚至要读取十几个文件才能处理完成，因此尽可能的提高硬盘速度和质量对提高apache的性能是有积极意义的。

最后是cpu和网络，cpu影响的是web程序执行速度，网络影响流量大小。

内存-->硬盘--> cpu ， 网络

i. apache 的工作模式

[回目录](#)



Apache HTTP服务器被设计为一个强大的、灵活的能够在多种平台以及不同环境下工作的服务器。这种模块化的设计就叫做“多进程处理模块”（ Multi-Processing Module ， MPM ），也叫做工作模式。

1. Prefork 模式（一个非线程型的）：

[回目录](#)

其主要工作方式是：当Apache服务器启动后，`mpm_prefork` 模块会预先创建多个子进程(默认为 5 个)，每个子进程只有一个线程，当接收到客户端的请求后，`mpm_prefork` 模块再将请求转交给子进程处理，并且每个子进程同时只能用于处理单个请求。如果当前的请求数将超过预先创建的子进程数时，`mpm_prefork` 模块就会创建新的子进程来处理额外的请求。Apache总是试图保持一些备用的或者是空闲的子进程用于迎接即将到来的请求。这样客户端的请求就不需要在接收后等候子进程的产生。

由于在 `mpm_prefork` 模块中，每个请求对应一个子进程，因此其占用的系统资源相对其他两种模块而言较多。不过 `mpm_prefork` 模块的优点在于它的每个子进程都会独立处理对应的单个请求，这样，如果其中一个请求出现问题就不会影响到其他请求。`Prefork` 在效率上要比 `Worker` 要高，但是内存使用大得多不擅长处理高并发的场景。

1.1 Apache 在 `prefork` 工作模式下影响性能的重要参数说明

[回目录](#)

```
# prefork MPM
<IfModule mpm_prefork_module>
    StartServers          5
    #apache启动时候默认开始的子进程数
    MinSpareServers       5
    #最小的闲置子进程数
    MaxSpareServers       10
    #最大的闲置子进程数
    MaxRequestWorkers      250
    #MaxRequestWorkers设置了允许同时的最大接入请求数。任何超过MaxRequestWorkers限制的请求将进入等候队列，在apache2.3.1以前的版本MaxRequestWorkers被称为MaxClients，旧的名字仍旧被支持。
    MaxConnectionsPerChild 500
    #设置的是每个子进程可处理的请求数。每个子进程在处理了“MaxConnectionsPerChild”个请求后将自动销毁。0意味着无限，即子进程永不销毁。虽然缺省设为0可以使每个子进程处理更多的请求，但如果设成非零值也有两点重要的好处：1、可防止意外的内存泄漏。2、在服务器负载下降的时候会自动减少子进程数。因此，可根据服务器的负载来调整这个值。在Apache2.3.9之前称之为MaxRequestsPerChild。
</IfModule>
```

注1： MaxRequestWorkers 是这些指令中最为重要的一个，设定的是 Apache 可以同时处理的请求，是对 Apache 性能影响最大的参数。如果请求数已达到这个值（可通过 `ps -ef|grep http|wc -l` 来确认），那么后面的请求就要排队，直到某个已处理请求完毕。这就是系统资源还剩下很多而 HTTP 访问却很慢的主要原因。虽然理论上这个值越大，可以处理的请求就越多，建议将初始值设为(以Mb为单位的最大物理内存/2),然后根据负载情况进行动态调整。比如一台4G内存的机器，那么初始值就是 $4000/2=2000$ 。

3.apache 优化

注2： `prefork` 控制进程在最初建立 `StartServers` 个子进程后，为了满足 `MinSpareServers` 设置的需要创建一个进程，等待一秒钟，继续创建两个，再等待一秒钟，继续创建四个……如此按指数级增加创建的进程数，最多达到每秒32个，直到满足 `MinSpareServers` 设置的值为止。这种模式可以不必在请求到来时再产生新的进程，从而减小了系统开销以增加性能。`MaxSpareServers` 设置了最大的空闲进程数，如果空闲进程数大于这个值，Apache会自动kill掉一些多余进程。这个值不要设得过大，但如果设的值比 `MinSpareServers` 小，Apache会自动把其调整为 `MinSpareServers+1`。如果站点负载较大，可考虑同时加大 `MinSpareServers` 和 `MaxSpareServers`。

注3： `ServerLimit` 和 `MaxClients`（`MaxRequestWorkers`）有什么区别呢？是因为在apache1时代，控制最大进程数只有 `MaxClients` 这个参数，并且这个参数最大值为256，并且是写死了的，试图设置为超过256是无效的，这是由于apache1时代的服务器硬件限制的。但是apache2时代由于服务器硬件的升级，硬件已经不再是限制，所以使用 `ServerLimit` 这个参数来控制最大进程数，`ServerLimit` 值 \geq `MaxClient` 值才有效。`ServerLimit` 要放在 `MaxClients` 之前，值要不小于 `MaxClients`。

注4：查看Apache加载的模块

```
[root@www ~]# apachectl -t -D DUMP_MODULES
```

或

```
[root@www ~]# apachectl -M
```

或

```
[root@www ~]# apachectl -l
```

如何查看Apache的工作模式呢？可以使用 `httpd -V` 命令查看，另外使用 `httpd -l` 也可以查看到

1.2 apache 查看编译参数

[回目录](#)

```
[root@nfs ~]# cat /usr/local/http-2.4.23/build/config.nice
#!/bin/sh
#
# Created by configure

"./configure" \
"--prefix=/usr/local/http-2.4.23" \
"--enable-so" \
"--enable-cgi" \
"--enable-cgid" \
"--enable-ssl" \
"--with-ssl=/usr/local/openssl" \
"--enable-rewrite" \
"--with-pcre=/usr/local/pcre" \
"--with-z=/usr/local/zlib" \
"--with-apr=/usr/local/apr" \
"--with-apr-util=/usr/local/apr-util" \
"--enable-modules=most" \
"--enable-mods-shared=most" \
"--enable-mpms-shared=all" \
"--with-mpm=event" \
"--enable-proxy" \
"--enable-proxy-fcgi" \
"--enable-expires" \
"--enable-deflate" \
"$@"
```

注5：如何修改 prefork 参数和启用 prefork 模式

```
[root@www ~]# vi /usr/local/http-2.4.23/conf/extra/httpd-mpm.conf
[root@www ~]# vi /usr/local/http-2.4.23/conf/httpd.conf
LoadModule mpm_prefork_module modules/mod_mpm_prefork.so
Include conf/extra/httpd-mpm.conf
```

3.重启 httpd 服务

2. Worker 模式(多线程多进程)：

[回目录](#)

和 `prefork` 模式相比，`worker` 使用了多进程和多线程的混合模式，`worker` 模式也同样会先预派生一些子进程，然后每个子进程创建一些线程，同时包括一个监听线程，每个请求过来会被分配到一个线程来服务。线程比起进程会更轻量，因为线程是通过共享父进程的内存空间，因此，内存的占用会减少一些，在高并发的场景下会比 `prefork` 有更多可用的线程，表现会更优秀一些；另外，如果一个线程出现了问题也会导致同一进程下的线程出现问题，如果是多个线程出现问题，也只是影响Apache的一部分，而不是全部。由于用到多进程多线程，需要考虑到线程的安全了，在使用 `keep-alive` 长连接的时候，某个线程会一直被占用，即使中间没有请求，需要等待到超时才会被释放（该问题在 `prefork` 模式下也存在）总的来说，`prefork` 方式速度要稍高于 `worker`，然而它需要的cpu和memory资源也稍多于woker。

2.1 Apache 在 `worker` 工作模式下影响性能的重要参数说明

[回目录](#)

```
# worker MPM
<IfModule mpm_worker_module>
    StartServers            3
        #apache启动时候默认开始的子进程数
    MinSpareThreads         75
        #最小空闲数量的工作线程
    MaxSpareThreads         250
        #最大空闲数量的工作线程
    ThreadsPerChild          25
        #每个子进程产生的线程数量
    MaxRequestWorkers        400
        #与prefork模式相同
    MaxConnectionsPerChild   0
        #与prefork模式相同
</IfModule>
```

注1：Worker 由主控制进程生成 StartServers 个子进程，每个子进程中包含固定的 ThreadsPerChild 线程数，各个线程独立地处理请求。同样，为了不在请求到来时再生成线程， MinSpareThreads 和 MaxSpareThreads 设置了最少和最多的空闲线程数；而 MaxRequestWorkers 设置了同时连入的 clients 最大总数。如果现有子进程中的线程总数不能满足负载，控制进程将派生新的子进程 MinSpareThreads 和 MaxSpareThreads 的最大缺省值分别是 75 和 250 。这两个参数对 Apache 的性能影响并不大，可以按照实际情况相应调节。

注2： ThreadsPerChild 是 worker MPM 中与性能相关最密切的指令。 ThreadsPerChild 的最大缺省值是 64 ，如果负载较大， 64 也是不够的。这时要显式使用 ThreadLimit 指令，它的最大缺省值是 20000 。

注3：Worker模式下所能同时处理的请求总数是由子进程总数乘以 `ThreadsPerChild` 值决定的，应该大于等于 `MaxRequestWorkers`。如果负载很大，现有的子进程数不能满足时，控制进程会派生新的子进程。默认最大的子进程总数是 16，加大时也需要显式声明 `ServerLimit`（系统配置的最大进程数量，最大值是 20000）。需要注意的是，如果显式声明了 `ServerLimit`，那么它乘以 `ThreadsPerChild` 的值必须大于等于 `MaxRequestWorkers`，而且 `MaxRequestWorkers` 必须是 `ThreadsPerChild` 的整数倍，否则 Apache 将会自动调节到一个相应值。

注4：进程与线程的区别 线程是指进程内的一个执行单元,也是进程内的可调度实体.

2.2 线程与进程的区别

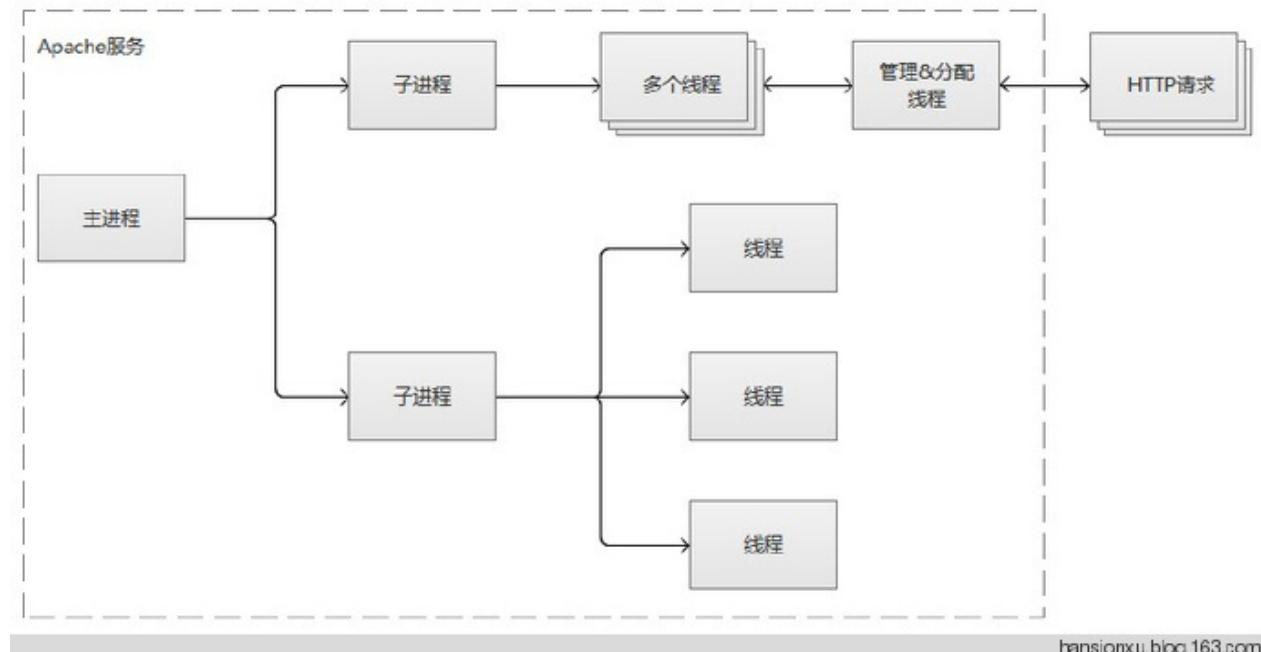
[回目录](#)

- (1) 地址空间: 进程内的一个执行单元; 进程至少有一个线程; 它们共享进程的地址空间; 而进程有自己独立的地址空间;
- (2) 资源拥有: 进程是资源分配和拥有的单位, 同一个进程内的线程共享进程的资源
- (3) 线程是处理器调度的基本单位, 但进程不是.
- (4) 二者均可并发执行.

进程和线程都是由操作系统所体会的程序运行的基本单元，系统利用该基本单元实现系统对应用的并发性。进程和线程的区别在于：简而言之，一个程序至少有一个进程，一个进程至少有一个线程。线程的划分尺度小于进程，使得多线程程序的并发性高。另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。

3. Event 模式：

[回目录](#)



这是Apache最新的工作模式，是 `worker` 模式的变种，它把服务进程从连接中分离出来，一 `worker` 模式不同的是在于它解决了 `keep-alive` 长连接的时候占用线程资源被浪费的问题，在 `event` 工作模式中，会有一些专门的线程用来管理这些 `keep-alive` 类型的线程，当有真实请求过来的时候，将请求传递给服务器的线程，执行完毕后，又允许它释放。这增强了在高并发场景下的请求处理。`event` 模式不能很好的支持 `https` 的访问（HTTP认证相关的问题）。

ii. apache 配置参数

[回目录](#)

1. KeepAlive On/Off

[回目录](#)

`KeepAlive`指的是保持连接活跃，换一句话说，如果将 `KeepAlive` 设置为 `On`，那么来自同一客户端的请求就不需要再一次连接，避免每次请求都要新建一个连接而加重服务器的负担。一般情况下，图片较多的网站应该把 `KeepAlive` 设为 `On`。

2. KeepAliveTimeOut number

[回目录](#)

如果第二次请求和第一次请求之间超过 `KeepAliveTimeOut` 的时间的话，第一次连接就会中断，再新建第二个连接。它的设置一般考虑图片或者JS等文件两次请求间隔，一般设置为 `3-5` 秒。

3. `MaxKeepAliveRequests 100`

[回目录](#)

一次连接可以进行的HTTP请求的最大请求次数。将其值设为0将支持在一次连接内进行无限次的传输请求。事实上没有客户程序在一次连接中请求太多的页面，通常达不到这个上限就完成连接了。

4. `HostnameLookups on|off|double`

[回目录](#)

如果是使用 `on`，那么只有进行一次反查，如果用 `double`，那么进行反查之后还要进行一次正向解析，只有两次的结果互相对应才行，而 `off` 就是不进行域名验证。如果为了安全，建议使用 `double`；为了加快访问速度，建议使用 `off`。域名查找开启这个会增加apache的负担，减慢访问速度建议关闭

5. `timeout 5`

[回目录](#)

推荐5 这个是 apache接受请求或者发出相应的时间超过这个时间 断开 注：以上配置项可在 `/usr/local/http-2.4.23/conf/extra/httpd-default.conf` 设置并在 `httpd.conf` 文件中通过 `include` 选项引用

iii. MPM 这个比较关键是影响并发效率的主要因素

[回目录](#)

1. `StartServers 10`

[回目录](#)

设置服务器启动时建立的子进程数量。因为子进程数量动态的取决于负载的轻重,所以一般没有必要调整这个参数。

2. MinSpareServers 10

[回目录](#)

设置空闲子进程的最小数量。所谓空闲子进程是指没有正在处理请求的子进程。如果当前空闲子进程数少于 `MinSpareServers`,那么Apache将以最大每秒一个的速度产生新的子进程。只有在非常繁忙机器上才需要调整这个参数。将此参数设的太大通常是一个坏主意。

3. MaxSpareThreads 75

[回目录](#)

设置空闲子进程的最大数量。如果当前有超过 `MaxSpareServers` 数量的空闲子进程,那么父进程将杀死多余的子进程。只有在非常繁忙机器上才需要调整这个参数。将此参数设的太大通常是一个坏主意。如果你将该指令的值设置为比 `MinSpareServers` 小,Apache将会自动将其修改成 `MinSpareServers+1`。

4. ServerLimit 2000

[回目录](#)

服务器允许配置的进程数上限。只有在你需要将 `MaxClients` 设置成高于默认值 256 的时候才需要使用。要将此指令的值保持和 `MaxClients` 一样。修改此指令的值必须完全停止服务后再启动才能生效,以`restart`方式重启动将不会生效。

5. MaxClients / MaxRequestWorkers 256

[回目录](#)

用于客户端请求的最大请求数量（最大子进程数），任何超过 `MaxClients` 限制的请求都将进入等候队列。默认值是 256，如果要提高这个值必须同时提高`ServerLimit`的值。建议将初始值设为(以Mb为单位的最大物理内存/2),然后根据负载情况进行动态调整。比如一台4G内存的机器，那么初始值就是 $4000/2=2000$ 。

6. MaxRequestsPerChild / MaxConnectionsPerChild 0

[回目录](#)

设置的是每个子进程可处理的请求数。每个子进程在处理了 `MaxRequestsPerChild` 个请求后将自动销毁。`0` 意味着无限，即子进程永不销毁。内存较大的服务器可以设置为 `0` 或较大的数字。内存较小的服务器不妨设置成30、50、100。所以一般情况下，如果你发现服务器的内存直线上升，建议修改该参数试试。注：以上配置项可在 `/usr/local/httpd-2.4.23/conf/extra/httpd-mpm.conf` 设置并在 `httpd.conf` 文件中通过 `include` 选项引用

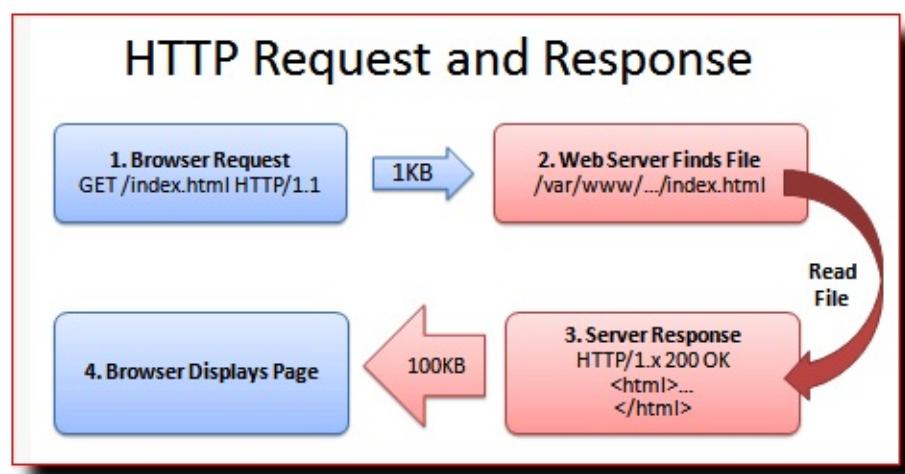
iv. 开启 apache 的 Gzip (deflate) 功能

[回目录](#)

`gzip` 可以极大的加速网站，有时压缩比率高到80%，最少都有40%以上，还是相当不错的。在Apache2之后的版本，模块名不叫 `gzip`，而叫 `mod_deflate`

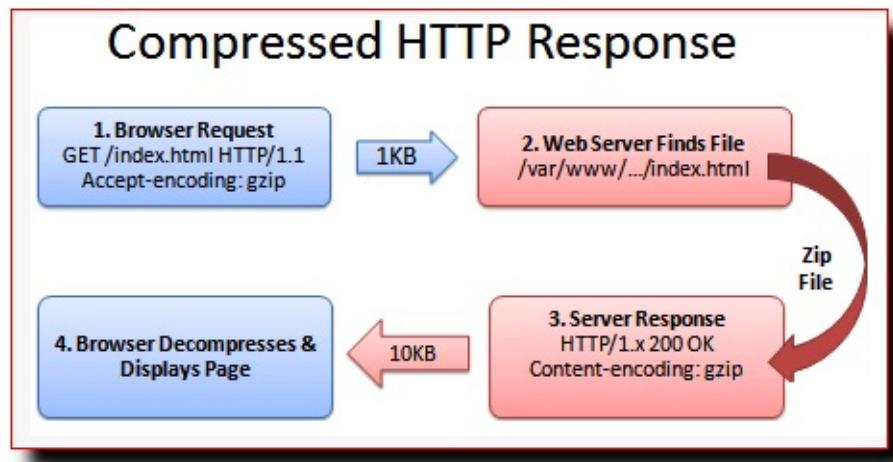
1. 未使用 Gzip

[回目录](#)



2. 开启使用 Gzip

[回目录](#)



2.1 如果要开启 **deflate** 的话,一定要打开下面二个模块

[回目录](#)

```
LoadModule deflate_module modules/mod_deflate.so
LoadModule headers_module modules/mod_headers.so
```

设置压缩比率,取值范围在 1 (最低) 到 9 (最高)之间,不建议设置太高,虽然有很高的压缩率,但是占用更多的CPU资源.

2.2 mod_deflate 模块检查及安装

[回目录](#)

```
[root@www ~]# /usr/local/http-2.4.23/bin/apachectl -M | grep deflate
deflate_module (shared)
```

如果没有安装：

3. 编译时安装方法

[回目录](#)

编译的时候跟上 `--enable-deflate` 即可实现安装

4. DSO 方式安装(适用于已经编译安装完成的场景)

[回目录](#)

`apxs`

```
[root@www ~]# cd /root/httpd-2.4.23/modules/filters/ #切到apache  
源码包mod_deflate所在的目录下  
[root@www filters]# /usr/local/http-2.4.23/bin/apxs -c -i -a mod  
_deflate.c #以dso的方式编译安装到apache中  
[root@www filters]# ll /usr/local/http-2.4.23/modules/mod_deflat  
e.so #检查mod_deflate是否安装，成功安装这里会显示出该文件  
-rwxr-xr-x. 1 root root 98144 Oct 22 23:14 /usr/local/http-2.4.2  
3/modules/mod_deflate.so
```

`apxs` 命令参数说明：

`-i` 此选项表示需要执行安装操作，以安装一个或多个动态共享对象到服务器的 `modules` 目录中。 `-a` 此选项自动增加一个 `LoadModule` 行到 `httpd.conf` 文件中，以启用此模块，或者，如果此行已经存在，则启用之。 `-c` 此选项表示需要执行编译操作。

如果重启的时候出现错误

引用

```
Cannot load /usr/local/apache/modules/mod_deflate.so into server  
: /usr/local/apache/modules/mod_deflate.so: undefined symbol: in  
flateEnd
```

3.apache 优化

```
[root@localhost filters]# apxs -c -i -a mod_deflate.c
/usr/local/apr/build-1/libtool --silent --mode=compile gcc -std=
gnu99 -prefer-pic -DLINUX -D_REENTRANT -D_GNU_SOURCE -g -O2 -p
thread -I/usr/local/http-2.4.23/include -I/usr/local/apr/include/apr-1 -c -o mod_deflat
e.lo mod_deflate.c && touch mod_deflate.slo
mod_deflate.c:51:18: fatal error: zlib.h: No such file or direct
ory
#include "zlib.h"
^
compilation terminated.

apxs:Error: Command failed with rc=65536
```

添加这一行 LoadFile /usr/lib/libz.so 到 httpd.conf 中

```
[root@localhost filters]# apachectl graceful
httpd: Syntax error on line 108 of /usr/local/http-2.4.23/conf/h
ttpd.conf: Cannot load /usr/lib/libz.so into server: /usr/lib/li
bz.so: cannot open shared object file: No such file or directory
[root@localhost filters]# grep libz.so /usr/local/http-2.4.23/co
nf/httpd.conf
LoadFile /usr/lib/libz.so
[root@localhost filters]#
```

需要在 LoadModule deflate_module modules/mod_deflate.so 的前面加
载 zlib.so 这里需要注意的是 LoadModule deflate_module 需要放
在 LoadModule php5_module 之后 引用 LoadFile /usr/lib/libz.so (x64系
统中该库文件位于 /usr/lib64 目录下，可以软链接到 /usr/lib 下
LoadModule deflate_module modules/mod_deflate.so

```
[root@localhost ~]# ln -sv /usr/lib64/libz.so /usr/lib/libz.so
'/usr/lib/libz.so' -> '/usr/lib64/libz.so'
[root@localhost ~]# apachectl graceful
httpd: Syntax error on line 108 of /usr/local/http-2.4.23/conf/h
ttpd.conf: Cannot load /usr/lib/libz.so into server: /usr/lib/li
bz.so: cannot open shared object file: No such file or directory
```

```
[root@localhost lib]# ln -sv /usr/lib64/libz
libzapojit-0.0.so.0      libzapojit-0.0.so.0.0.0  libz.so.1
                           libz.so.1.2.7
[root@localhost lib]# ln -sv /usr/lib64/libz.so.1 /usr/lib/libz.
so
'/usr/lib/libz.so' -> '/usr/lib64/libz.so.1'
[root@localhost lib]# apachectl stop
AH00558: httpd: Could not reliably determine the server's fully
qualified domain name, using ::1. Set the 'ServerName' directive
globally to suppress this message
httpd (no pid file) not running
[root@localhost lib]# apachectl start
AH00558: httpd: Could not reliably determine the server's fully
qualified domain name, using ::1. Set the 'ServerName' directive
globally to suppress this message
[root@localhost lib]# apachectl -M | grep deflate
AH00558: httpd: Could not reliably determine the server's fully
qualified domain name, using ::1. Set the 'ServerName' directive
globally to suppress this message
deflate_module (shared)
```

这里要注意的是 `ln -sv /usr/lib64/libz.so /usr/lib/libz.so` 这种方式是无效的，因为 `/usr/lib64/` 下面并没有 `libz.so` 这个文件，而当使用 `ln -sv /usr/lib64/libz.so.1 /usr/lib/libz.so` 的时候是好使的。

这里的报错是正确的报错，的确是沒有这个文件。

重新启动httpd：

```
# /usr/local/http2.4.23/bin/apachectl graceful #优雅启动httpd服务
```

5.修改 Apache 配置文件开启 gzip 压缩传输

[回目录](#)

5.1 http.conf 修改、增加配置参数

[回目录](#)

```
LoadModule deflate_module      modules/mod_deflate.so  
LoadModule headers_module     modules/mod_headers.so
```

打开 `httpd.conf` 后，先将上面两行配置前面的#号去掉，这样apache就会启用这两个模块，其中 `mod_deflate` 是压缩模块，就是对要传输到客户端的代码进行 gzip压缩； `mod_headers` 模块的作用是告诉浏览器页面使用了gzip压缩，如果不开启 `mod_headers` 那么浏览器就会对gzip压缩过的页面进行下载，而无法正常显示。

在 `httpd.conf` 中加入以下代码，可以加到任何空白地方，不了解apache的话，如果担心加错地方，就放到`http.conf`文件的最后一行

注：在添加代码前最好先查一查要添加的代码是否存在

[Apache Module mod_filter 官网](#)

```
<IfModule mod_deflate.c>
    DeflateCompressionLevel 9      # 压缩程度的等级，预设可以采用
6 这个数值，以维持耗用处理器效能与网页压缩质量的平衡。
    SetOutputFilter DEFLATE      #设置输出过滤器，对输出启用压缩，必
须的，就像一个开关一样，告诉apache对传输到浏览器的内容进行压缩
    #AddOutputFilterByType DEFLATE text/html text/plain text
/xml application/x-javascript application/x-httpd-php
    #AddOutputFilterByType DEFLATE image/*
    AddOutputFilterByType DEFLATE text/* #设置对文件是文本的
内容进行压缩，例如text/html text/css text/plain等.
    AddOutputFilterByType DEFLATE application/ms* applicatio
n/vnd* application/postscript application/javascript application
/x-javascript #对javascript文件进行压缩
    AddOutputFilterByType DEFLATE application/x-httpd-php ap
plication/x-httpd-fastphp #对php类型的文件进行压缩.
    SetEnvIfNoCase Request_URI .(?:gif|jpe?g|png)$ no-gzip d
ont-vary #设置不对后缀gif, jpg, jpeg, png的图片文件进行压缩。注：?:表示
不会捕获( )里内容了

    SetEnvIfNoCase Request_URI .(?:exe|t?gz|zip|bz2|sit|rar)
$ no-gzip dont-vary #同上，就是设置不对exe, tgz, gz等的文件进行压缩
    SetEnvIfNoCase Request_URI .(?:pdf|mov|avi|mp3|mp4|rm)$
no-gzip dont-vary #同上就是设置不对pdf, avi, mp3等的文件进行压缩
</IfModule>
```

#设置日志输出！

```
DeflateFilterNote Input input_info #声明输入流的byte数量
DeflateFilterNote Output output_info #声明输出流的byte数量
DeflateFilterNote Ratio ratio_info #声明压缩的百分比
LogFormat '"%r" %{output_info}n %{input_info}n (%{ratio_info}n%
)' deflate #声明日志格式
CustomLog logs/deflate_log.log deflate
```

修改完成后保存退出并重启 httpd 服务 使用谷歌浏览器测试访问，如下图显示结果：（提示：在访问测试页之前按 F12 键）

3.apache 优化

The screenshot shows the Chrome DevTools Network tab. The 'Headers' tab is selected. A request for 'test.html' is listed. The 'Response Headers' section is expanded, showing the following content:

```
Accept-Ranges: bytes
Connection: Keep-Alive
Content-Encoding: gzip
Content-Length: 6439
Content-Type: text/html
Date: Tue, 25 Oct 2016 14:05:44 GMT
ETag: "4ded-53fb0de3b2c9b-gzip"
Keep-Alive: timeout=5, max=100
Last-Modified: Tue, 25 Oct 2016 13:55:58 GMT
Server: Apache/2.4.23 (Unix)
Vary: Accept-Encoding

The 'Request Headers' section is also expanded, showing the following content:
```

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8
Connection: keep-alive
Host: 192.168.31.83
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36
```

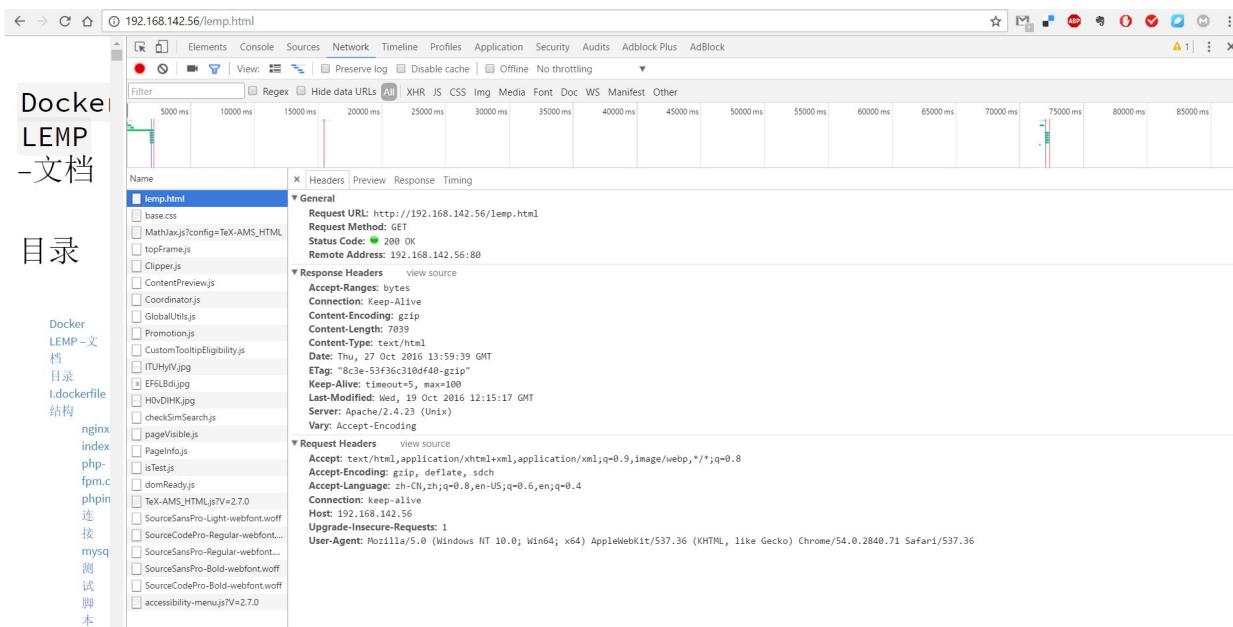
实际测试

```
[root@localhost ~]# tail -11 /usr/local/http-2.4.23/conf/httpd.conf
<IfModule mod_deflate.c>
    DeflateCompressionLevel 9
    SetOutputFilter DEFLATE
    AddOutputFilterByType DEFLATE text/*
    DeflateFilterNote Input input_info
    DeflateFilterNote Output output_info
    DeflateFilterNote Ratio ratio_info
    LogFormat '"%r" %{output_info}n %{input_info}n (%{ratio_info}n%)' deflate
    CustomLog logs/deflate_log.log deflate
</IfModule>
```

6. 实际测试结果图

回目录

chrome



3.apache 优化

Headers Preview Response Timing

General

Request URL: http://192.168.142.56/1emp.html
Request Method: GET
Status Code: 200 OK
Remote Address: 192.168.142.56:80

Response Headers view source

Accept-Ranges: bytes
Connection: Keep-Alive
Content-Encoding: gzip **(highlighted)**
Content-Length: 7039
Content-Type: text/html
Date: Thu, 27 Oct 2016 13:59:39 GMT
ETag: "8c3e-53f36c310df40-gzip"
Keep-Alive: timeout=5, max=100
Last-Modified: Wed, 19 Oct 2016 12:15:17 GMT
Server: Apache/2.4.23 (Unix)
Vary: Accept-Encoding

Request Headers view source

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.6,en;q=0.4
Connection: keep-alive
Host: 192.168.142.56
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36

firefox

apache 优化

目录

apache 优化
目录
一、安装apache2.4.23
新版本的httpd-2.4新增以下特性；
新增模块；
还新增以下几条新特性；
安装apache2.4.23
安装
安装apr和apr-util
安装 zlib
安装pcre
安装openssl
安装apache2.4.23
相关参数解释：
优化http程序执行路径
开启apache服务器：
开机后自动启动
将 Apache 加入开机自动启动：
启动编译好的 Apache 2.4.23：
二、Apache的优化配置：
1、apache的工作模式：
A.Prefork模式（一个非线程型的）：
Apache对prefork工作模式下影响性能的重要参数说明
apache查看编译参数
B.Worker模式(多线程多进程)：
Apache在worker工作模式下影响性能的重要参数说明

请求网址: https://stackedit.io/res-min/themes/base.css
请求方法: GET
状态码: 200 OK
版本: HTTP/1.1
响应头 (0 KB)
Content-Encoding: gzip **(highlighted)**
Content-Type: text/css; charset=UTF-8
Date: Thu, 27 Oct 2016 13:57:54 GMT
Etag: "W/1aa63-1568400724"
Last-Modified: "Sat, 13 Aug 2016 13:03:57 GMT"
Vary: Accept-Encoding
X-Powered-By: "Express"
请求头 (0 KB)
Host: "stackedit.io"
User-Agent: "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:49.0) Gecko/20100101 Firefox/49.0"
Accept: "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
Accept-Language: "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3"
Accept-Encoding: "gzip, deflate, br"
Connection: "keep-alive"

消息头	Cookie	参数	响应	耗时	安全性
请求网址: https://stackedit.io/res-min/themes/base.css					
请求方法: GET					
状态码: <input checked="" type="radio"/> 200 OK					
版本: HTTP/1.1					
<input type="button"/> 过滤消息头					
▼ 响应头 (0 KB)					
Accept-Ranges: "bytes"					
Cache-Control: "public, max-age=0"					
Content-Encoding: "gzip"					
Content-Type: "text/css; charset=UTF-8"					
Date: "Thu, 27 Oct 2016 13:57:54 GMT"					
Etag: "W/"1aa63-15684007248"					
Last-Modified: "Sat, 13 Aug 2016 13:03:57 GMT"					
Vary: "Accept-Encoding"					
X-Powered-By: "Express"					
▼ 请求头 (0 KB)					
Host: "stackedit.io"					
User-Agent: "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:49.0) Gecko/20100101 Firefox/49.0"					
Accept: "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"					
Accept-Language: "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3"					
Accept-Encoding: "gzip, deflate, br"					
Connection: "keep-alive"					

消息头	Cookie	参数	响应	耗时
请求网址: http://192.168.142.56/Learning_Python.txt				
请求方法: GET				
远程地址: 192.168.142.56:80				
状态码: ● 200 OK				
版本: HTTP/1.1				
请求头:			响应头:	
Host: 192.168.142.56 User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:49.0) Gecko/20100101 Firefox/49.0 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3 Accept-Encoding: gzip, deflate Connection: keep-alive Upgrade-Insecure-Requests: 1			Accept-Ranges: bytes Connection: Keep-Alive Content-Encoding: gzip Content-Type: text/plain Date: Thu, 27 Oct 2016 14:12:30 GMT Etag: "438b44-53fd8e6673706-gzip" Keep-Alive: timeout=5, max=100 Last-Modified: Thu, 27 Oct 2016 13:41:34 GMT Server: Apache/2.4.23 (Unix) Transfer-Encoding: chunked Vary: Accept-Encoding	

7. 查看 `deflate_log` 日志

[回目录](#)

```
[root@localhost logs]# cat deflate_log.log
"GET /apache.html HTTP/1.1" -/- (-%)
"GET /apache.html HTTP/1.1" 12493/43670 (28%)
"GET /favicon.ico HTTP/1.1" -/- (-%)
"GET /favicon.ico HTTP/1.1" -/- (-%)
"GET /apache.html HTTP/1.1" -/- (-%)
"GET /lemp.html HTTP/1.1" 7021/35902 (19%)
"GET /Learning_Python.txt HTTP/1.1" 1242271/4426564 (28%)
```

```
[root@localhost logs]# grep deflate_module /usr/local/http-2.4.2
3/conf/httpd.conf
#LoadModule deflate_module modules/mod_deflate.so
[root@localhost logs]# tail -11 /usr/local/http-2.4.23/conf/http
d.conf
#<IfModule mod_deflate.c>
#    DeflateCompressionLevel 9
#    SetOutputFilter DEFLATE
#    AddOutputFilterByType DEFLATE text/*
#    DeflateFilterNote Input input_info
#    DeflateFilterNote Output output_info
#    DeflateFilterNote Ratio ratio_info
#    LogFormat '"%r" %{output_info}n/%{input_info}n (%{ratio_inf
o}n%)' deflate
#    CustomLog logs/deflate_log.log deflate
#</IfModule>

[root@localhost logs]# apachectl graceful
```

V. 配置 **mod_expires** 模块

[回目录](#)

[Apache Module mod_expires 官网](#)

这个非常有用的优化，`mod_expires` 可以减少 20-30% 左右的重复请求，让重复的用户对指定的页面请求结果都 `CACHE` 在本地，根本不向服务器发出请求。但要注意更新快的文件不要这么做。

这个模块控制服务器应答时的 `Expires` 头内容和 `Cache-Control` 头的 `max-age` 指令。有效期(`expiration date`)可以设置为相对于源文件的最后修改时刻或者客户端的访问时刻。

1. 未启用 **expire** 的效果

[回目录](#)

```
[root@www htdocs]# curl -I 192.168.31.83/12.png
HTTP/1.1 200 OK
Date: Tue, 25 Oct 2016 15:52:37 GMT
Server: Apache/2.4.23 (Unix)
Last-Modified: Sun, 23 Oct 2016 15:34:10 GMT
ETag: "8c9f-53f8a01b18080"
Accept-Ranges: bytes
Content-Length: 35999
Vary: Accept-Encoding
Content-Type: image/png
```

2.启用 **expire** 缓存

[回目录](#)

`mod_expires` 的安装配置：

2.1启用 **expires_module**

[回目录](#)

```
LoadModule expires_module modules/mod_expires.so
```

2.2然后添加 **Expires** 配置规则

[回目录](#)

3.apache 优化

```
<IfModule mod_expires.c>
    ExpiresActive On
    ExpiresByType text/css "now plus 1 month"
    ExpiresByType application/x-javascript "now plus 5 day"
    ExpiresByType image/jpeg "access plus 1 month"
    ExpiresByType image/gif "access plus 1 month"
    ExpiresByType image/bmp "access plus 1 month"
    ExpiresByType image/x-icon "access plus 1 month"
    ExpiresByType image/png "access plus 1 minutes"
    ExpiresByType application/x-shockwave-flash "access plus 1 month"
    ExpiresDefault "now plus 0 minutes"
</IfModule>
```

```
[root@apache ~]# tail -7 /usr/local/http-2.4.23/conf/httpd.conf
<IfModule mod_expires.c>
    ExpiresActive On
    ExpiresByType text/css "now plus 1 month"
    ExpiresByType image/png "access plus 1minutes"
    ExpiresDefault "now plus 0 minutes"
</IfModule>
```

```
[root@apache ~]# grep expires_module /usr/local/http-2.4.23/conf
/httpd.conf
LoadModule expires_module modules/mod_expires.so
```

```
[root@apache ~]# apachectl graceful
AH00526: Syntax error on line 531 of /usr/local/http-2.4.23/conf
/httpd.conf:
'ExpiresByType image/png access plus 1minutes': bad expires code
, missing <type>
```

```
[root@apache ~]# tail -7 /usr/local/http-2.4.23/conf/httpd.conf
<IfModule mod_expires.c>
    ExpiresActive On
    ExpiresByType text/css "now plus 1 month"
    ExpiresByType image/png "access plus 1 minutes"
    ExpiresDefault "now plus 0 minutes"
</IfModule>

[root@apache ~]# apachectl graceful
```

1 和 minutes 之间是有空格的

3. 验证

[回目录](#)

```
[root@www htdocs]# curl -I 192.168.31.83/12.png
HTTP/1.1 200 OK
Date: Tue, 25 Oct 2016 16:00:57 GMT
Server: Apache/2.4.23 (Unix)
Last-Modified: Sun, 23 Oct 2016 15:34:10 GMT
ETag: "8c9f-53f8a01b18080"
Accept-Ranges: bytes
Content-Length: 35999
Cache-Control: max-age=60
Expires: Tue, 25 Oct 2016 16:01:57 GMT
Vary: Accept-Encoding
Content-Type: image/png
```

```
[root@apache ~]# curl -I 192.168.142.56/test.png
HTTP/1.1 200 OK
Date: Fri, 28 Oct 2016 13:08:36 GMT
Server: Apache/2.4.23 (Unix)
Last-Modified: Sun, 11 Sep 2016 13:21:03 GMT
ETag: "f4b3-53c3b404bcd0"
Accept-Ranges: bytes
Content-Length: 62643
Cache-Control: max-age=60
Expires: Fri, 28 Oct 2016 13:09:36 GMT
Content-Type: image/png
```

`ExpiresDefault` 和 `ExpiresByType` 指令同样能够用易懂的语法格式进行定义：

```
ExpiresDefault "<base> [plus] {<num> <type>}"
ExpiresByType type/encoding "<base> [plus] {<num> <type>}"
```

其中 `<base>` 是下列之一：

- `access`
- `now` (等价于 `access '`)
- `modification`

`plus` 关键字是可选的。 `<num>` 必须是整数， `<type>` 是下列之一：

- `years`
- `months`
- `weeks`
- `days`
- `hours`
- `minutes`
- `seconds`

例如，下列3个指令都表示文档默认的有效期是一个月：

```
ExpiresDefault "access plus 1 month"
ExpiresDefault "access plus 4 weeks"
ExpiresDefault "access plus 30 days"
```

有效期可以通过增加 `<num> <type>` 子句进一步调整：

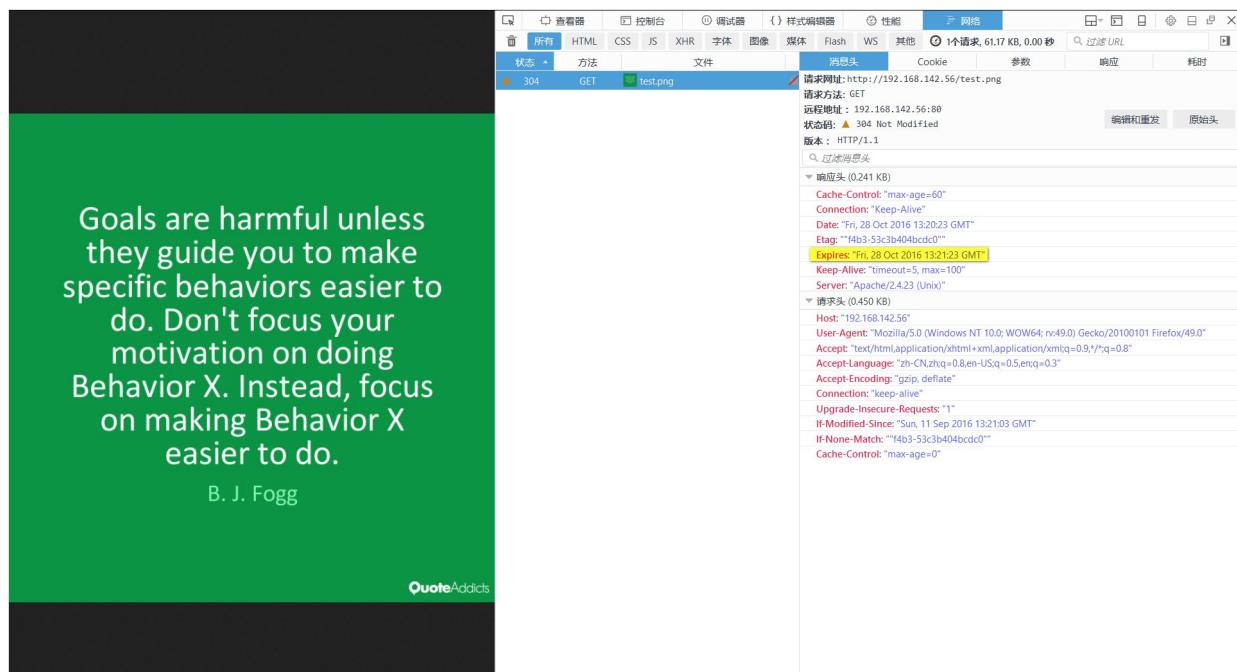
```
ExpiresByType text/html "access plus 1 month 15 days 2 hours"
ExpiresByType image/gif "modification plus 5 hours 3 minutes"
```

注意，如果你使用基于最后修改日期的设置，`Expires:` 头将不会被添加到那些并非来自于磁盘文件的内容。这是因为这些内容并不存在"最后修改时间"的属性。

```
# GIF有效期为1个月（秒数）
ExpiresByType image/gif A2592000
ExpiresByType image/jpeg A2592000
ExpiresByType image/png A2592000
ExpiresByType image/x-icon A2592000
ExpiresByType application/x-javascript A604800
ExpiresByType text/plain A604800
```

```
# HTML文档的有效期是最后修改时刻后的一星期
ExpiresByType text/html M604800
</IfModule>
```

"M"表示源文件的最后修改时刻，"A"表示客户端对源文件的访问时刻。后面的时间则以秒计算。有关 Apache Expires Module 的介绍，可以参阅其[官方文档](#)



Goals are harmful unless they guide you to make specific behaviors easier to do. Don't focus your motivation on doing Behavior X. Instead, focus on making Behavior X easier to do.

B. J. Fogg

QuoteAddicts

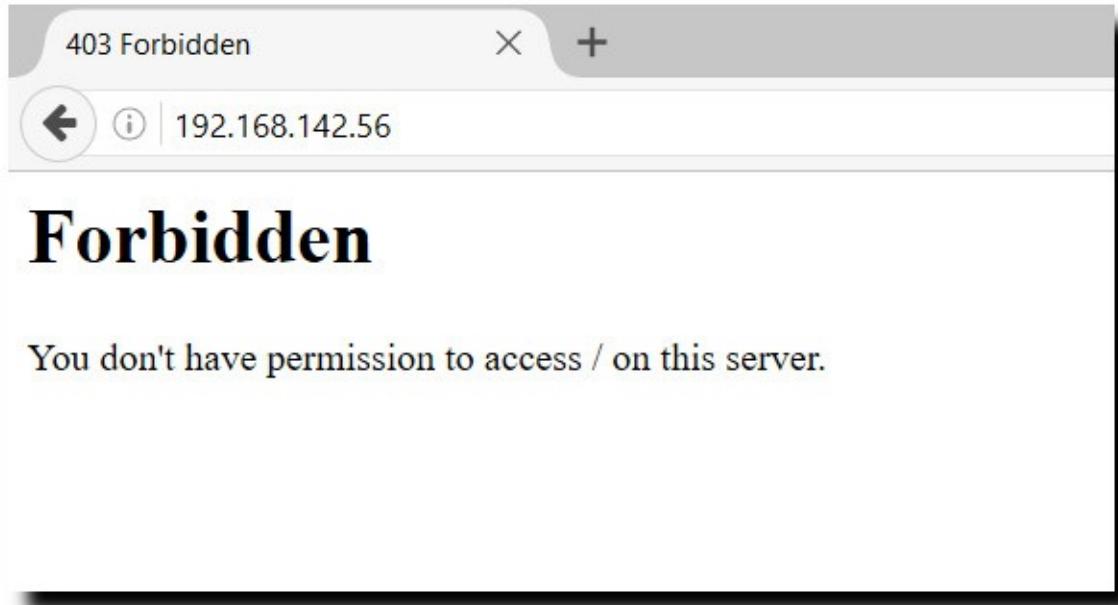
vi. Apache 禁止目录遍历

回目录

将 Options Indexes FollowSymLinks 中的 Indexes 去掉，就可以禁止 Apache 显示该目录结构。Indexes 的作用就是当该目录下没有 index.html 文件时，就显示目录结构。

```
[root@apache ~]# grep FollowSymLinks /usr/local/http-2.4.23/conf/httpd.conf
#   Indexes Includes FollowSymLinks SymLinksIfOwnerMatch ExecCGI MultiViews
# Options Indexes FollowSymLinks
Options FollowSymLinks
```

```
[root@apache ~]# cd /usr/local/http-2.4.23/htdocs/
[root@apache htdocs]# ls
apache.html  index.html  Learning_Python.txt  temp.html  test.png
[root@apache htdocs]# mv index.html index.jsp
```



```
[root@apache ~]# grep FollowSymLinks /usr/local/http-2.4.23/conf/httpd.conf
    #   Indexes Includes FollowSymLinks SymLinksIfOwnerMatch ExecCGI MultiViews
    # Options Indexes FollowSymLinks
    Options Indexes FollowSymLinks
[root@apache ~]# apachectl graceful
```

The screenshot shows a browser developer tools Network tab with a red box highlighting the 'temp.html' entry. The status is 403 Forbidden. The response headers are:

- Cache-Control: "max-age=0"
- Connection: "Keep-Alive"
- Content-Length: "403"
- Content-Type: "text/html; charset=ISO-8859-1"
- Date: "Fri, 28 Oct 2016 13:35:44 GMT"
- Expires: "Fri, 28 Oct 2016 13:35:44 GMT"
- Keep-Alive: "timeout=5, max=100"
- Server: "Apache/2.4.23 (Unix)"

The request headers from the browser are:

- Host: "192.168.142.56"
- User-Agent: "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:49.0) Gecko/20100101 Firefox/49.0"
- Accept: "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
- Accept-Language: "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3"
- Accept-Encoding: "gzip, deflate"
- Connection: "keep-alive"
- Upgrade-Insecure-Requests: "1"
- Cache-Control: "max-age=0"

vii. apache 隐藏版本信息

[回目录](#)

测试默认 apache 的状态信息

```
[root@www htdocs]# curl -I 192.168.31.83
HTTP/1.1 200 OK
Date: Tue, 25 Oct 2016 16:09:09 GMT
Server: Apache/2.4.23 (Unix)
Last-Modified: Mon, 11 Jun 2007 18:53:14 GMT
ETag: "2d-432a5e4a73a80"
Accept-Ranges: bytes
Content-Length: 45
Cache-Control: max-age=0
Expires: Tue, 25 Oct 2016 16:09:09 GMT
Content-Type: text/html
```

1、主配置中启用 `httpd-default.conf`

[回目录](#)

```
Include conf/extra/httpd-default.conf
```

```
[root@apache ~]# grep httpd-default.conf /usr/local/http-2.4.23/
conf/httpd.conf
Include conf/extra/httpd-default.conf
```

2、修改 `httpd-default.conf`

[回目录](#)

文件： `/usr/local/http-2.4.23/conf/extra/httpd-default.conf`

找到

3.apache 优化

```
ServerTokens Full  
ServerSignature On
```

改成

```
ServerTokens Prod  
ServerSignature off
```

```
[root@apache ~]# grep ServerTokens /usr/local/http-2.4.23/conf/e  
xtra/httpd-default.conf  
# ServerTokens  
# ServerTokens Full  
ServerTokens Prod  
[root@apache ~]# grep ServerSignature /usr/local/http-2.4.23/con  
f/extrra/httpd-default.conf  
# ServerSignature Off  
ServerSignature Off
```

重启 apache 测试

```
[root@apache ~]# apachectl graceful
```

测试隐藏版本号后 apache 的状态信息

3.apache 优化

```
[root@www htdocs]# curl -I 192.168.31.83
HTTP/1.1 200 OK
Date: Tue, 25 Oct 2016 16:14:32 GMT
Server: Apache
Last-Modified: Mon, 11 Jun 2007 18:53:14 GMT
ETag: "2d-432a5e4a73a80"
Accept-Ranges: bytes
Content-Length: 45
Cache-Control: max-age=0
Expires: Tue, 25 Oct 2016 16:14:32 GMT
Content-Type: text/html
```

```
[root@apache ~]# curl -I 192.168.142.56
HTTP/1.1 200 OK
Date: Fri, 28 Oct 2016 13:42:12 GMT
Server: Apache
Cache-Control: max-age=0
Expires: Fri, 28 Oct 2016 13:42:12 GMT
Content-Type: text/html; charset=ISO-8859-1
```

如果你需要彻底将版本之类的信息进行改头换面，你就需要在编译之前做准备或者进行从新编译了。在重新编译时，修改源码包下 `include` 目录下的 `ap_release.h` 文件

```
#define AP_SERVER_BASEVENDOR "Apache Software Foundation" #服务的
供应商名称
#define AP_SERVER_BASEPROJECT "Apache HTTP Server" #服务的项目名
#define AP_SERVER_BASEPRODUCT "Apache" #服务的产品名
#define AP_SERVER_MAJORVERSION_NUMBER 2 #主要版本号
#define AP_SERVER_MINORVERSION_NUMBER 4 #小版本号
#define AP_SERVER_PATCHLEVEL_NUMBER 23 #补丁级别
#define AP_SERVER_DEVBUILD_BOOLEAN 0 #
```

上述列出的行，已经给出了注释，大家可以修改成自己想要的，然后编译安装之后，对方就彻底不知道你的版本号了。

viii. Apache 日志切割

[回目录](#)

1.为什么要分割日志

[回目录](#)

随着网站的访问越来越大，WebServer产生的日志文件也会越来越大，如果不对日志进行分割，那么只能一次将大的日志(如Apache的日志)整个删除，这样也丢失了很多对网站比较宝贵的信息，因为这些日志可以用来进行访问分析、网络安全监察、网络运行状况监控等，因此管理好这些海量的日志对网站的意义是很大的。

2. rotatelogs 使用方法

[回目录](#)

方法1:使用 `rotatelogs` (apache自带的工具) 每隔一天记录一个日志

辑Apache的主配置文件，更改内容如下：

注释掉如下两行

```
ErrorLog logs/error_log  
CustomLog logs/access_log common
```

然后添加如下两行

```
ErrorLog "|/usr/local/http-2.4.23/bin/rotatelogs -l logs/error_%  
Y%m%d.log 86400"  
CustomLog "|/usr/local/http-2.4.23/bin/rotatelogs -l logs/access  
_%Y%m%d.log 86400" combined
```

3.apache 优化

```
[root@apache ~]# grep ErrorLog /usr/local/http-2.4.23/conf/httpd.conf
# ErrorLog: The location of the error log file.
# If you do not specify an ErrorLog directive within a <VirtualHost>
# ErrorLog "logs/error_log"
ErrorLog "|/usr/local/http-2.4.23/bin/rotatelogs -l logs/error_%Y%m%d.log 86400"
[root@apache ~]# grep CustomLog /usr/local/http-2.4.23/conf/httpd.conf
# a CustomLog directive (see below).
# CustomLog "logs/access_log" common
CustomLog "|/usr/local/http-2.4.23/bin/rotatelogs -l logs/aces_%Y%m%d.log 86400" combined
#CustomLog "logs/access_log" combined
#      CustomLog logs/deflate_log.log deflate
```

注：其中 86400 为轮转的时间单位为秒

验证：查看 logs 目录下的日志文件

```
[root@www ~]# ls /usr/local/http-2.4.23/logs/
access_20161026.log  access_log  deflate_log.log  error_20161026.log  error_log  httpd.pid
```

访问以下网站

```
[root@apache ~]# apachectl graceful
[root@apache ~]# ls /usr/local/http-2.4.23/logs/
acces_20161028.log  access_log  deflate_log.log  error_20161028.log  error_log  httpd.pid
[root@apache logs]# tail acces_20161028.log
192.168.142.1 - - [28/Oct/2016:21:58:15 +0800] "GET / HTTP/1.1"
200 403 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:49.0) Gecko/20100101 Firefox/49.0"
```

由于 apache 自带的日志轮询工具 rotatelogs，据说在进行日志切割时容易丢日志，因此我们通常使用 cronolog 进行日志轮询。

3. cronolog 使用方法

[回目录](#)

方法2、使用 cronolog 为每一天建立一个新的日志

安装 cronolog 程序 下载 cronolog

```
[root@www ~]# tar zxf cronolog-1.6.2.tar.gz
[root@www ~]# cd cronolog-1.6.2/
[root@www cronolog-1.6.2]# ./configure && make && make install
```

主配置文件中的使用方法

```
ErrorLog "|/usr/local/sbin/cronolog logs/error-%Y%m%d.log"
CustomLog "|/usr/local/sbin/cronolog logs/access-%Y%m%d.log" combined
```

```
[root@apache ~]# grep ErrorLog /usr/local/http-2.4.23/conf/httpd.conf
# ErrorLog: The location of the error log file.
# If you do not specify an ErrorLog directive within a <VirtualHost>
# ErrorLog "logs/error_log"
ErrorLog "|/usr/local/sbin/cronolog logs/error_cronolog_%Y%m%d.log"
[root@apache ~]# grep CustomLog /usr/local/http-2.4.23/conf/httpd.conf
# a CustomLog directive (see below).
# CustomLog "logs/access_log" common
CustomLog "|/usr/local/sbin/cronolog logs /%Y%m%d/access_log.%H" combined
#CustomLog "logs/access_log" combined
# CustomLog logs/deflate_log.log deflate
[root@apache ~]# apachectl graceful
```

3.apache 优化

```
[root@apache logs]# ls
acces_20161028.log  cronolog-1.6.2           deflate_log.log    e
rror_cronolog_20161028.log  httpd.pid
access_log          cronolog-1.6.2.tar.gz  error_20161028.log  e
rror_log
```

```
[root@apache logs]# grep CustomLog /usr/local/http-2.4.23/conf/
httpd.conf
    # a CustomLog directive (see below).
    # CustomLog "logs/access_log" common
    CustomLog "|/usr/local/sbin/cronolog logs /acces_cronolog_%
Y%m%d.log" combined
    #CustomLog "logs/access_log" combined
#    CustomLog logs/deflate_log.log deflate
[root@apache logs]# apachectl graceful
```

```
"/usr/local/http-2.4.23/conf/httpd.conf" 536L, 19840C written

[root@apache logs]#
[root@apache logs]# grep CustomLog /usr/local/http-2.4.23/conf/httpd.conf
    # a CustomLog directive (see below).
    # CustomLog "logs/access_log" common
    # CustomLog "|/usr/local/sbin/cronolog logs /access_cronolog
_%Y%m%d.log" combined
    CustomLog "|/usr/local/sbin/cronolog logs/access-%Y%m%d.log"
combined
    #CustomLog "logs/access_log" combined
#    CustomLog logs/deflate_log.log deflate
[root@apache logs]# apachectl graceful
[root@apache logs]# ls
acces_20161028.log  cronolog-1.6.2          deflate_log.log      e
rror_cronolog_20161028.log  httpd.pid
access_log           cronolog-1.6.2.tar.gz   error_20161028.log  e
rror_log
[root@apache logs]# ls
acces_20161028.log  access_log       cronolog-1.6.2.tar.gz  erro
r_20161028.log        error_log
access-20161028.log  cronolog-1.6.2  deflate_log.log      erro
r_cronolog_20161028.log  httpd.pid
[root@apache logs]# tail access-20161028.log
192.168.142.1 - - [28/Oct/2016:22:14:14 +0800] "GET /test.png HT
TP/1.1" 304 - "-" "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:49.0)
Gecko/20100101 Firefox/49.0"
```

logs/access 关键就是这里你上面的 logs

/acces_cronolog_%Y%m%d.log 在 logs 和 /access 之间有空格，这两个中间不能有空格。

如果Apache中有多个虚拟主机，最好每个虚拟主机中放置一个这样的代码，并将日志文件名改成不同的名字。

4.扩展

[回目录](#)

这个保证了每天一个文件夹 文件夹下 每个小时产生一个 log

```
CustomLog "|/usr/local/sbin/cronolog logs /%Y%m%d/access_log.%H"
combined
```

按 天 轮询（生产环境常见用法，推荐使用）：

```
CustomLog "|/usr/local/sbin/cronolog logs/access_www_%Y%m%d.log"
combined
```

按 小时 轮询（生产环境较常见用法）：

```
CustomLog "|/usr/local/sbin/cronolog logs /access_www_ %Y%m%d%H.
log" combined
```

验证：查看 logs 目录下的日志文件

```
[root@www ~]# ls /usr/local/http-2.4.23/logs/
access-20161026.log  access_log  deflate_log.log  error-20161026
.log  error_log  httpd.pid
```

注意：这两个管道日志文件程序还有一点不同之处是使用 cronolog 时如果日志是放在某个不存在的路径则会自动创建目录，而使用 rotatelogs 时不能自动创建，这一点要特别注意

ix. 配置防盗链

[回目录](#)

有时候，你的网站莫名其妙的访问量变大，不要高兴的太早，有可能是被别人盗链了。

举个例子：比如你搭了个discuz论坛，里面有些热点图片、视频；然后别人将他网站上访问图片的地址重定向到你的discuz上，这样他的服务器就可以空闲出来了；也就是说别人访问他网站的图片视频，消耗的确是你服务器的资源；解决这个问题的方法是配置下防盗链，让外来的盗不了链；

1. rewrite 实现防盗

[回目录](#)

方法1：Apache 防盗链的第一种实现方法，可以用 `rewrite` 实现。

首先要确认 Apache 的 `rewrite module` 可用：

```
[root@www ~]# apachectl -M | grep rewrite
rewrite_module (shared)
```

打开 `httpd.conf`，确保有这么一行配置：

```
LoadModule rewrite_module modules/mod_rewrite.so
```

然后在找到自己网站对应的配置的地方（如在主配置文件中或虚拟主机中），加入下列代码：

```
ServerName www.benet.com
```

```
#防盗链配置
RewriteEngine On
RewriteCond %{HTTP_REFERER} !^http://benet.com/.*$ [NC]
RewriteCond %{HTTP_REFERER} !^http://benet.com$ [NC]
RewriteCond %{HTTP_REFERER} !^http://www.benet.com/.*$ [NC]
RewriteCond %{HTTP_REFERER} !^http://www.benet.com$ [NC]
RewriteRule \.(gif|jpg|swf)$ http://www.benet.com/about/nolink
.png [R,NC]
```

1.1 相关选项的解释

[回目录](#)

参数	作用
RewriteEngine On	#启用 rewrite , 要想rewrite起作用，必须要写上
RewriteCond test-string condPattern	#写在RewriteRule之前，可以有一或N条，用于测试 rewrite的匹配条件，具体怎么写，后面会详细说到。
RewriteRule Pattern Substitution	#规则
% {HTTP_REFERER} :	服务器变量， HTTP Referer 是header的一部分，当浏览器向web服务器发送请求的时候，一般会带上 Referer，告诉服务器我是从哪个页面链接过来的，服务器藉此可以获得一些信息用于处理。比如从我主页上链接到一个朋友那里，他的服务器就能够从 HTTP Referer 中统计出每天有多少用户点击我主页上的链接访问他的网站。
[NC]	指的是不区分大小写
[R]	强制重定向 redirect

2. 防盗链配置的说明

[回目录](#)

```

RewriteEngine On
RewriteCond %{HTTP_REFERER} !^http://benet.com/.*$ [NC]
RewriteCond %{HTTP_REFERER} !^http://benet.com$ [NC]
RewriteCond %{HTTP_REFERER} !^http://www.benet.com/.*$ [NC]
RewriteCond %{HTTP_REFERER} !^http://www.benet.com$ [NC]
RewriteRule .*\.(gif|jpg|swf)$ http://www.benet.com/about/nolink.png [R, NC]

```

颜色	解释
红色部分	表示自己的信任站点。对我的站点来说，设置为 <code>http://www.benet.com</code> 和 <code>http://benet.com</code>
绿色部分	要保护文件的扩展名(以 \ 分开)。以这些为扩展名的文件，必须通过红色标注的网址引用，才可以访问。
蓝色部分	定义被盗链时替代的图片，让所有盗链 <code>jpg</code> 、 <code>gif</code> 、 <code>swf</code> 等文件的网页，显示网页文档根目录下的 <code>about/nolink.png</code> 文件。

注意：替换显示的图片不要放在设置防盗链的目录中，并且该图片文件体积越小越好。当然你也可以不设置替换图片，而是使用下面的语句即可：

```
RewriteRule .*\.(gif\|jpg\|png)$ - [F]
```

注： `[F]` (强制 URL 为被禁止的 `forbidden`), 强制当前 URL 为被禁止的，即，立即反馈一个 HTTP 响应代码 `403` (被禁止的)。

2.1 超链接测试

[回目录](#)

```
[root@nfs htdocs]# cat index.html
<html>
<title>Jason</title>
<body><h1>This is jason!</h1></body>
<a href='http://192.168.142.56/test.png'><font color=red>Rewrite Test</font></a>
</html>
```

The screenshot shows a browser's developer tools Network tab. A failed request for 'test.png' is highlighted with a yellow background. The details are as follows:

- Request URL: <http://192.168.142.59/192.168.142.56/test.png>
- Request Method: GET
- Status Code: 404 404 Not Found
- Remote Address: 192.168.142.59:80

General

- Connection: Keep-Alive
- Content-Length: 221
- Content-Type: text/html; charset=iso-8859-1
- Date: Fri, 28 Oct 2016 23:13:07 GMT
- Keep-Alive: timeout=5, max=99
- Server: Apache/2.4.23 (Unix)

Response Headers

- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
- Accept-Encoding: gzip, deflate, sdch
- Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.6,en;q=0.4
- Cache-Control: max-age=0
- Connection: keep-alive
- Host: 192.168.142.59
- Referer: <http://192.168.142.59/>
- Upgrade-Insecure-Requests: 1
- User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36

2.2超链接语法

[回目录](#)

HTML <a> href Attribute

< HTML <a> tag

Example

The href attribute specifies the link's destination:

```
<a href="http://www.w3schools.com">Visit W3Schools</a>
```

[Try it Yourself »](#)

3.apache 优化

```
[root@nfs htdocs]# cat index.html
<html>
<title>Jason</title>
<body><h1>This is jason!</h1></body>
<a href="http://192.168.142.56/test.png"><font color=red>Rewrite
Test</font></a>
</html>
```

将 ' 变成 "

The screenshot shows a network request details panel. At the top, there are tabs for Headers, Preview, Response, and Timing. The Headers tab is selected. Below the tabs, there are sections for General, Response Headers, and Request Headers.

General:

- Request URL: http://192.168.142.56/test.png
- Request Method: GET
- Status Code: 200 OK (from disk cache)
- Remote Address: 192.168.142.56:80

Response Headers:

- Accept-Ranges: bytes
- Cache-Control: max-age=60
- Content-Length: 62643
- Content-Type: image/png
- Date: Fri, 28 Oct 2016 23:16:52 GMT
- ETag: "f4b3-53c3b404bcd0"
- Expires: Fri, 28 Oct 2016 23:17:52 GMT
- Last-Modified: Sun, 11 Sep 2016 13:21:03 GMT
- Server: Apache

Request Headers:

⚠ Provisional headers are shown

- Referer: http://192.168.142.59/
- Upgrade-Insecure-Requests: 1
- User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36

General

Request URL:`http://192.168.142.56/test.png`

Request Method:`GET`

Status Code:`200 OK (from disk cache)`

Remote Address:`192.168.142.56:80`

Response Headers

`Accept-Ranges:bytes`

`Cache-Control:max-age=60`

`Content-Length:62643`

`Content-Type:image/png`

`Date:Fri, 28 Oct 2016 23:16:52 GMT`

`ETag:"f4b3-53c3b404bcd0"`

`Expires:Fri, 28 Oct 2016 23:17:52 GMT`

`Last-Modified:Sun, 11 Sep 2016 13:21:03 GMT`

`Server:Apache`

Request Headers

Provisional headers are shown

`Referer:http://192.168.142.59/`

`Upgrade-Insecure-Requests:1`

`User-Agent:Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36`

测试成功

而且可以看到这里是 `from disk cache` 还有 `Referer` 是链接到这里的地址- `http://192.168.142.59/`

```
[root@apache ~]# apachectl -M | grep rewrite
rewrite_module (shared)
```

3.apache 优化

```
[root@apache ~]# grep rewrite_module /usr/local/http-2.4.23/conf/httpd.conf
LoadModule rewrite_module modules/mod_rewrite.so
```

```
[root@apache ~]# grep Rewrite /usr/local/http-2.4.23/conf/httpd.conf
RewriteEngine On
RewriteCond %{HTTP_PREFERER} !^http://192.168.142.56/.*$ [NC]
RewriteRule .*\.png$ http://192.168.142.56/about/noti.html [N,
RC]
[root@apache ~]# cd /usr/local/http-2.4.23/htdocs/
[root@apache htdocs]# mkdir about
[root@apache htdocs]# cd about/
[root@apache about]# cat noti.html
<h1><font color=red>Warning !</font></h1>
```

```
[root@apache about]# apachectl graceful
AH00526: Syntax error on line 218 of /usr/local/http-2.4.23/conf/httpd.conf:
/RewriteRule: unknown flag 'RC'
```

```
[root@apache ~]# grep Rewrite /usr/local/http-2.4.23/conf/httpd.conf
RewriteEngine On
RewriteCond %{HTTP_PREFERER} !^http://192.168.142.56/.*$ [NC]
RewriteRule .*\.png$ http://192.168.142.56/about/noti.html [R,
NC]
[root@apache ~]# apachectl graceful
```

3.apache 优化

The screenshot shows the Network tab in Chrome DevTools. A request to `http://192.168.142.59/` is selected. The General section shows:

- Request URL: `http://192.168.142.59/`
- Request Method: GET
- Status Code: 200 OK (from disk cache)
- Remote Address: 192.168.142.59:80

The Response Headers section includes:

- Accept-Ranges: bytes
- Content-Length: 154
- Content-Type: text/html
- Date: Fri, 28 Oct 2016 23:38:40 GMT
- ETag: "9a-53ff50c2d037f"
- Last-Modified: Fri, 28 Oct 2016 23:16:27 GMT
- Server: Apache/2.4.23 (Unix)

The Request Headers section shows:

- ⚠ Provisional headers are shown
- Upgrade-Insecure-Requests: 1
- User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.143 Safari/537.36

The screenshot shows the Network tab in Chrome DevTools. A request to `http://192.168.142.56/test.png` is selected. The General section shows:

- Request URL: `http://192.168.142.56/test.png`
- Request Method: GET
- Status Code: 302 Found
- Remote Address: 192.168.142.56:80

The Response Headers section includes:

- Cache-Control: max-age=0
- Connection: Keep-Alive
- Content-Length: 221
- Content-Type: text/html; charset=iso-8859-1
- Date: Fri, 28 Oct 2016 23:40:41 GMT
- Expires: Fri, 28 Oct 2016 23:40:41 GMT
- Keep-Alive: timeout=5, max=100
- Location: `http://192.168.142.56/about/noti.html`
- Server: Apache

The Request Headers section shows:

- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
- Accept-Encoding: gzip, deflate, sdch
- Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.6,en;q=0.4
- Connection: keep-alive
- Host: 192.168.142.56
- Referer: `http://192.168.142.59/`
- Upgrade-Insecure-Requests: 1
- User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.143 Safari/537.36

A yellow callout box points to the `Location` header with the text: "Found,临时重定向，临时放到某个地方，会在响应报文中使用“Location：新位置”；".

3. SetEnvIfNoCase 和 access 实现防盗

[回目录](#)

方法2：通过判断浏览器头信息来阻止某些请求，即利用 `SetEnvIfNoCase` 和 `access`。

这个方法可以通过阻止某些机器人或蜘蛛爬虫抓取你的网站来节省你的带宽流量。

语法：

```
SetEnvIfNoCase attribute regex [!]env-variable[=value] [[!]env-v  
ariable[=value]] ...
```

`SetEnvIfNoCase` 当满足某个条件时，为变量赋值，即根据客户端请求属性设置环境变量。

注：`Referer`：指明了请求当前资源原始资源的 `URL`，使用 `referer` 是可以防盗链

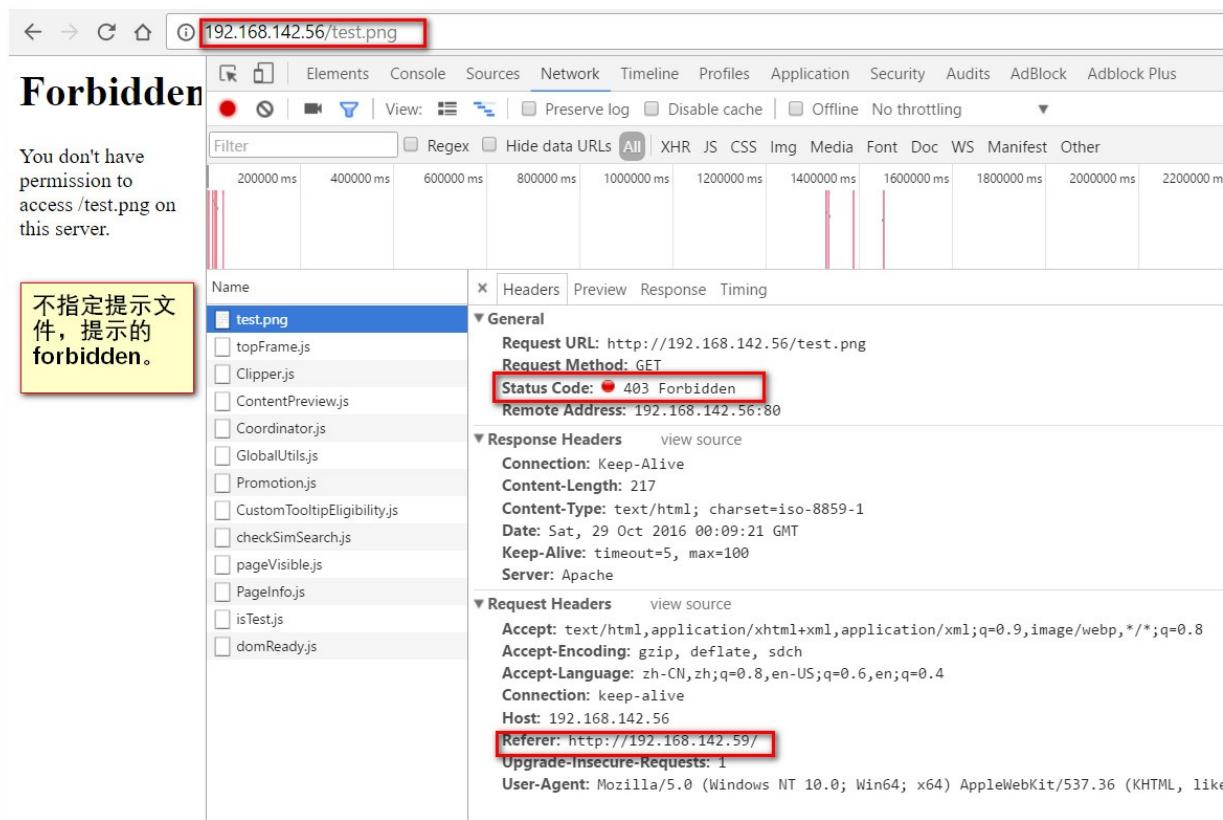
然后在找到自己网站对应的配置的地方（如在主配置文件中或虚拟主机中），加入下列代码：

```
SetEnvIfNoCase Referer "^http://www.benet.com" local_ref  
SetEnvIfNoCase Referer "^http://benet.com" local_ref  
<filesmatch "\.(mp3|mp4|zip|rar|jpg|gif|png)">
```

```
SetEnvIfNoCase Referer "^http://192.168.142.56" local_ref  
<filesmatch "\.(png)">  
    Require all denied  
    Require env local_ref  
</filesmatch>
```

```
[root@apache ~]# apachectl graceful
```

3.apache 优化



4. 2.4版本以下

回目录

方法一：回目录

```
Order Deny,Allow  
Allow from env=local_ref  
Deny from all
```

方法二：回目录

```
Order Allow,Deny  
Allow from env=local_ref
```

5. 2.4版本以上

回目录

```
Require all denied  
Require env local_ref  
</filesmatch>
```

[回目录](#)

lamp-优化

目录

- I. php 的工作模式
 - FastCGI 工作机制
- II. 编译环境及各软件版本
 - i. 编译安装 LAMP
 - ii. 解决依赖关系
 - iii. 编译安装 php
 - 相关选项的解释
 - iv. 提供 php 配置文件
 - v. 为 php-fpm 提供脚本
 - vi. 提供 php-fpm 配置文件并编辑
 - vii. 配置 apache (切换到 apache 主机上操作)
- III. 测试 LAMP 环境
 - i. 压力测试
 - 1. ab 的原理
 - 2. ab 的安装
 - 3. ab 参数说明
 - 4. ab 性能指标
 - 4.1 吞吐率 (Requests per second)
 - 4.2 并发连接数 (The number of concurrent connections)
 - 4.3 并发用户数 (Concurrency Level)
 - 4.4 用户平均请求等待时间 (Time per request)
 - 4.5 服务器平均请求等待时间 (Time per request:across all concurrent requests)
 - ii. CentOS7.2 下安装 php 加速软件 Xcache (在 php 主机上完成下面的操作)
 - 1. 安装 xcache
 - 2. 创建 xcache 缓存文件
 - 3. 拷贝 xcache 后台管理程序到网站根目录
 - 4. 配置 php 支持 xcache

■ 5. 测试

I. php 的工作模式

[回目录](#)

php 在 lamp 环境下共有三种工作模式：

- CGI 模式
- apache 模块
- FastCGI 模式

CGI 模式下运行 PHP ，性能不是很好。作为 apache 的模块方式运行，在以前的课程中编译安装 lamp 已经介绍过了。 FastCGI 的方式和 apache 模块的不同点在于： FastCGI 方式 PHP 是一处独立的进程，所有 PHP 子进程都由 PHP 的一个叫作 php-fpm 的组件负责管理；而 apache 模块化方式运行的 PHP ，则是 apache 负责调用 PHP 完成工作。 PHP 的 FastCGI 方式性能要比 apache 模块化方式强很多，今天我们以 FastCGI 方式编译安装 lamp 。

FastCGI 工作机制

[回目录](#)

首先客户端发起请求，请求分为2种，一种是静态请求它可以直接由Apache直接响应返回；另一种是动态的请求，如其中包含中 php 或者 Perl 这种脚本解释性语言，则由Apache服务器通过 fastcgi 协议调用php服务器执行并返回给Apache由Apache返回解释执行后的结果，如果这个过程中涉及到对数据的操作，此时php服务器还会通过mysql协议调用mysql服务器。

Linux下LAMP分主机部署



II. 编译环境及各软件版本

[回目录](#)

Linux	Web服务器	Php	Mysql数据库	xcache
Centos7.2	Httpd-2.4.23	php-5.4.26	Mysql5.7	xcache-3.1.0

[主机规划](#)

至少3台主机，操作系统都是 centos7.2。网段在 192.168.197.0/24 网关 192.168.197.2 分配如下：

- 1台 httpd 服务器 (192.168.31.83)
- 1台 php 服务器 (192.168.31.141)
- 1台 mysql 服务器 (192.168.31.225)

i. 编译安装 LAMP

[回目录](#)

编译安装 apache (请参考前面apache的安装) 编译安装 mysql (请参考mysql安装) FastCGI 方式安装 php

ii.解决依赖关系

[回目录](#)

```
[root@phpserver ~]# yum -y install libxml2-devel lzip2-devel libcurl-devel libmcrypt-devel openssl-devel bzip2-devel
```

安装 libmcrypt

```
[root@phpserver ~]# tar zxf libmcrypt-2.5.7.tar.gz
[root@phpserver ~]# cd libmcrypt-2.5.7/
[root@phpserver libmcrypt-2.5.7]# ./configure --prefix=/usr/local/libmcrypt && make && make install
```

iii.编译安装 php

[回目录](#)

```
[root@phpserver ~]# tar zxf php-5.6.27.tar.gz
[root@phpserver ~]# cd php-5.6.27/
[root@phpserver php-5.6.27]# ./configure --prefix=/usr/local/php
5.6 --with-mysql=mysqlnd --with-pdo-mysql=mysqlnd --with-mysqli=
mysqlnd --with-openssl --enable-fpm --enable-sockets --enable-sysvshm --enable-mbstring --with-freetype-dir --with-jpeg-dir --with-png-dir --with-zlib --with-libxml-dir=/usr --enable-xml --with-mhash --with-mcrypt=/usr/local/libmcrypt --with-config-file-path=/etc --with-config-file-scan-dir=/etc/php.d --with-bz2 --enable-maintainer-zts
[root@phpserver php-5.6.27]# make && make install
```

相关选项的解释

回目录

- --prefix=/usr/local/php5.6 //安装位置
- --with-mysql=mysqlnd //支持mysql
- --with-pdo-mysql=mysqlnd //支持pdo模块
- --with-mysqli=mysqlnd //支持mysqli模块 注：上面的三选项的作用：数据库与php不在一个服务器上，指定此种方式，安装数据库连接驱动
- --with-openssl //支持openssl模块
- --enable-fpm //支持fpm模式
- --enable-sockets //启用socket支持
- --enable-sysvshm //启用系统共享内存支持
- --enable-mbstring //多字节字串、像我们的中文就是多字节字串
- --with-freetype-dir //支持freetype、就要装freetype-devel、跟字体相关的、字体解析工具
- --with-jpeg-dir
- --with-png-dir 注：上面的二选项的作用：处理jpeg、png图片的、php可以动态生成jpeg图片
- --with-zlib //是个压缩库、在互联网传输时用来压缩传输的
- --with-libxml-dir=/usr //这个libxml是用来解析xml的、指定/usr下
- --enable-xml //支持xml的
- --with-mhash //支持mhash
- --with-mcrypt=/usr/local/libmcrypt //libmcrypt-devel这个程序包所指定的
- --with-config-file-path=/etc //指定配置文件的存放路径的
- --with-config-file-scan-dir=/etc/php.d //配置文件扫描路径
- --with-bz2 //支持BZip2

为了支持apache的 worker 或 event 这两个 MPM ，编译时使用了 --enable-maintainer-zts 选项如果使用 PHP5.3 以上版本，为了链接MySQL数据库，可以指定 mysqlnd ，这样在本机就不需要先安装MySQL或MySQL开发包了。 mysqlnd 从 php 5.3 开始可用，可以编译时绑定到它（而不用和具体的 MySQL客户端库绑定形成依赖），但从 PHP 5.4 开始它就是默认设置了。

iv. 提供 php 配置文件

[回目录](#)

```
[root@phpserver php-5.6.27]# cp php.ini-production /etc /php.ini
```

v.为 **php-fpm** 提供脚本

[回目录](#)

```
[root@phpserver php-5.6.27]# cp sapi/fpm/init.d.php-fpm /etc/init.d/php-fpm  
[root@phpserver php-5.6.27]# chmod +x /etc/init.d/php-fpm  
[root@phpserver php-5.6.27]# chkconfig --add php-fpm  
[root@phpserver php-5.6.27]# chkconfig php-fpm on
```

vi.提供 **php-fpm** 配置文件并编辑

[回目录](#)

```
# cp /usr/local/php5.6/etc/php-fpm.conf.default /usr/local/php5.  
6/etc/php-fpm.conf  
[root@phpserver ~]# vi /usr/local/php5.6/etc/php-fpm.conf
```

修改内容如下

```
pid = run/php-fpm.pid  
listen = 192.168.31.141:9000  
pm.max_children = 50  
pm.start_servers = 5  
pm.min_spare_servers = 5  
pm.max_spare_servers = 35
```

启动 **php-fpm** 服务

```
[root@phpserver ~]# service php-fpm start
Starting php-fpm done
[root@phpserver ~]# netstat -anpt | grep php-fpm
tcp        0      0 0.0.0.0:9000          0.0.0.0:*
              LISTEN      25456/php-fpm: mast
[root@phpserver ~]# firewall-cmd --permanent --add-port=9000/tcp
success
[root@phpserver ~]# firewall-cmd --reload
Success
```

在该主机上新建虚拟主机目录用于存放网页文件

```
[root@phpserver ~]# mkdir -p /var/www/benet
```

至此php安装配置完毕，下面配置apache通过 fastcgi 协议调用php

vii. 配置 apache (切换到 apache 主机上操作)

[回目录](#)

在Apache2.4以后已经专门有一个模块针对 FastCGI 的实现，此模块为 mod_proxy_fcgi.so ，它其实是作为 mod_proxy.so 模块的扩充，因此，这两个模块都要加载

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_fcgi_module modules/mod_proxy_fcgi.so
```

```
[root@www ~]# apachectl -M | grep proxy
proxy_module (shared)
proxy_fcgi_module (shared)
```

建立一个目录作为虚拟主机的家目录

```
[root@www ~]# mkdir -p /var/www/benet
```

4.lamp-优化

编辑主配置文件 `httpd.conf`，开启虚拟主机

启用 `Include conf/extra/httpd-vhosts.conf` 同时定位 `AddType`；添加下面两行：让apache能识别php格式的页面

```
AddType application/x-httpd-php .php  
AddType application/x-httpd-php-source .phps
```

并且定位至 `DirectoryIndex`：支持php格式的主页 `DirectoryIndex index.php index.html` #添加 `index.php`（最好添加在最前面）

配置虚拟主机支持使用 `fcgi`

```
[root@www ~]# vi /usr/local/http-2.4.23/conf/extra/httpd-vhosts.conf  
<VirtualHost *:80>  
    ServerAdmin webmaster@benet.com  
    DocumentRoot "/var/www/benet"  
    ServerName www.benet.com  
    ServerAlias benet.com  
    ErrorLog "logs/benet.com-error_log"  
    CustomLog "logs/benet.com-access_log" common  
    ProxyRequests Off  
    ProxyPassMatch ^/(.*\.php(.*))$ fcgi://192.168.31.141:9000/var/www/benet/$1  
    #<LocationMatch "^(.*\.php(.*))$">  
    #    ProxyPass fcgi://192.168.31.141:9000/var/www/benet  
    #</LocationMatch>  
    <Directory "/var/www/benet">  
        Options FollowSymLinks  
        AllowOverride None  
        Require all granted  
    </Directory>  
</VirtualHost>
```

其中：

- `ProxyRequests off` #关闭正向代理
- `ProxyPassMatch`：把以 `.php` 结尾的文件请求发送到 `php-fpm` 进

程，`php-fpm` 至少需要知道运行的目录和 `URI`，所以这里直接在 `fcgi://192.168.31.141:9000` 后指明了这两个参数，其它的参数的传递已经被 `mod_proxy_fcgi.so` 进行了封装，不需要手动指定。

特别注意的是，`/var/www/benet` 部分需要与 `<VirtualHost>` 中的 `DocumentRoot` 后的路径一致 `ProxyPassMatch` 只有满足特定正则模式的内容才会匹配并执行此规则，这里的模式是，`^/(.*\.\php(/.*))?$` 从网站（虚拟主机 `<VirtualHost>`）的根目录开始，匹配任何以 `.php` 结尾，或者在 `.php` 之后紧跟一个 `/` 再跟别的内容的路径。`^ (caret)` 和 `$ (dollar)` 标志要匹配的路径的开始和结束（）括号里的内容可以用 `$1` 来表示，以方便后面引用它。

`fcgi:// 192.168.31.141:9000` 通过 `mod_proxy_fcgi` 来转发的代理，使用 `fastCGI` 协议，转到 `PHP-FPM` 监听的端口。

`/path/to/your/documentroot/` 非常重要！必须与虚拟主机的路径匹配，且必须是对应 `php` 文件在操作系统中的绝对路径。否则会找不到文件。

`$1` 可以从原始请求扩展成整个请求路径的变量，这里指代前面（）里面匹配的那个路径（uri）

补充：`Apache httpd 2.4` 以前的版本中，要么把 `PHP` 作为 `Apache` 的模块运行，要么添加一个第三方模块支持 `PHP-FPM` 实现。

III. 测试 LAMP 环境

[回目录](#)

在 `mysql` 主机上创建用于 `php` 服务器连接的 `mysql` 账户

```
mysql> grant all on *.* to testuser@'%' identified by '123456';
```

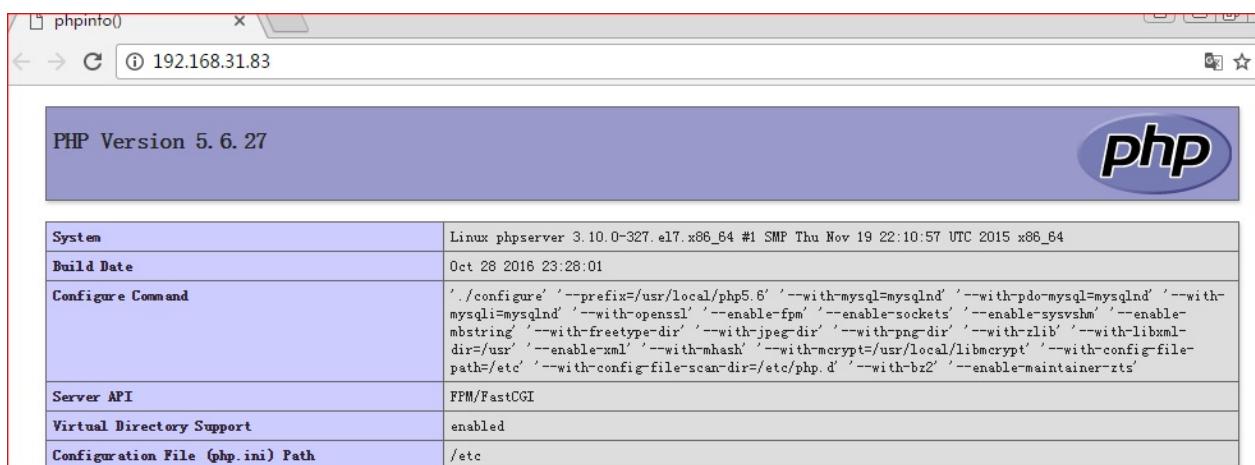
注意防火墙要允许 `mysql` 连接。

在 `php` 服务器上的 `/var/www/benet` 目录下创建 `.php` 的测试页：

```
[root@phpserver ~]# cat /var/www/benet/index.php
<?php
phpinfo();
?>

[root@phpserver ~]# cat /var/www/benet/test1.php
<?php
$link=mysql_connect('192.168.31.225','testuser','123456');
if ($link)echo "connection success.....";
mysql_close();
?>
```

测试访问 php 测试页：



看到上面两个测试页说明 apache 、 php 、 mysql 之间可以协同工作了。

I. 压力测试

[回目录](#)

网站性能压力测试是服务器网站性能调优过程中必不可少的一环。只有让服务器处在高压情况下，才能真正体现出软件、硬件等各种设置不当所暴露出的问题。

性能测试工具目前最常见的有以下几种：`ab`、`http_load`、`webbench`、`siege`。今天我们专门来介绍 `ab`。
`ab` 是 apache 自带的压力测试工具。`ab` 非常实用，它不仅可以对 apache 服务器进行网站访问压力测试，也可以对或其它类型的服务器进行压力测试。比如 `nginx`、`tomcat`、`IIS` 等。

下面我们开始介绍有关 `ab` 命令的使用：

- 1 · `ab` 的原理
- 2 · `ab` 的安装
- 3 · `ab` 参数说明
- 4 · `ab` 性能指标
- 5 · `ab` 实际使用
- 6 · 测试 `nginx` 性能

1. `ab` 的原理

[回目录](#)

`ab` 是 `apachebench` 命令的缩写。

`ab` 的原理：`ab` 命令会创建多个并发访问线程，模拟多个访问者同时对某一 URL 地址进行访问。它的测试目标是基于 URL 的，因此，它既可以用来测试 apache 的负载压力，也可以测试 `nginx`、`lighttp`、`tomcat`、`IIS` 等其它 Web 服务器的压力。

`ab` 命令对发出负载的计算机要求很低，它既不会占用很高 CPU，也不会占用很多内存。但却会给目标服务器造成巨大的负载，其原理类似 CC 攻击。自己测试使用也需要注意，否则一次上太多的负载。可能造成目标服务器资源耗完，严重时甚至导致死机。

2. `ab` 的安装

[回目录](#)

`ab` 的安装非常简单，如果是源码安装 apache 的话，那就更简单了。apache 安装完毕后 `ab` 命令存放在 apache 安装目录的 bin 目录下。如下：

```
/usr/local/http2.4.23/bin/ab
```

如果 apache 是通过 yum 的 RPM 包方式安装的话， ab 命令默认存放
在 /usr/bin 目录下。如下：

```
which ab
```

注意：如果不装 apache 但是又想使用 ab 命令的话，我们可以直接安装
apache的工具包 httpd-tools 。如下：

```
yum -y install httpd-tools
```

查看ab是否安装成功，可以切换到上述目录下，使用 ab -V 命令进行检测。如
下：

```
[root@www ~]# /usr/local/http-2.4.23/bin/ab -V
This is ApacheBench, Version 2.3 <$Revision: 1748469 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

3. ab 参数说明

[回目录](#)

有关ab命令的使用，我们可以通过帮助命令进行查看。如下：

```
[root@cent05 ~]# ab --help
```

下面我们将对这些参数，进行相关说明。如下：

- -n : 在测试会话中所执行的请求个数(即总请求数)。
- -c : 一次产生的请求个数（即并发用户数）。

```
[root@www ~]# ab -c 500 -n 10000 http://192.168.31.83/index.html
This is ApacheBench, Version 2.3 <$Revision: 1748469 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
```

```

ech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
Benchmarking 192.168.31.83 (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests

Server Software:          Apache
Server Hostname:         192.168.31.83
Server Port:              80

Document Path:            /index.html      #请求的资源
Document Length:          312 bytes       #HTTP响应数据的正文长度

Concurrency Level:        500      #并发个数 (并发用户数)
Time taken for tests:    1.452 seconds   #所有这些请求处理完成所花费的时间
Complete requests:        10000     #完成请求数
Failed requests:           0          #失败的请求数
Non-2xx responses:        10000
Total transferred:         4760000 bytes  #表示所有请求的响应数据长度总和，包括每个HTTP响应数据的头信息和正文数据的长度。注意这里不包括HTTP请求数据的长度，仅为web服务器流向用户PC的应用层数据总长度。
HTML transferred:          3120000 bytes  # 表示所有请求的响应数据中正文数据的总和，也就是减去了Total transferred中HTTP响应数据中的头信息的长度。

Requests per second:      7530.93 [#/sec] (mean)      #吞吐量-每秒请求数。计算公式：Complete requests/Time taken for tests
Time per request:          66.393 [ms] (mean)      #用户平均请求等待时间，计

```

算公式：Time token for tests / (Complete requests / Concurrency Level)。

Time per request: 0.133 [ms] (mean, across all concurrent requests) #服务器平均请求等待时间，计算公式：Time taken for tests / Complete requests。

Transfer rate: 3500.71 [Kbytes/sec] received #表示这些请求在单位时间内从服务器获取的数据长度，计算公式：Total transferred / Time taken for tests，这个统计很好的说明服务器的处理能力达到极限时，其出口宽带的需求量。（即平均每秒网络上的流量）

Connection Times (ms)

	min	mean [+/-sd]	median	max
Connect:	10	27	7.3	27
Processing:	4	37	36.5	32
Waiting:	2	27	37.0	21
Total:	30	64	37.6	470

Percentage of the requests served within a certain time (ms)

50%	60
66%	63
75%	64
80%	66
90%	71
95%	76
98%	89
99%	261
100%	470 (longest request)

这部分数据用于描述每个请求处理时间的分布情况，比如以上测试，80%的请求处理时间都不超过66ms，这个处理时间是指前面的Time per request，即对于单个用户而言，平均每个请求的处理时间。

继续压力测试 我们再来进行一次压力测试，此时并发用户数为1000，其他条件不变，查看两次测试结果的吞吐量差别

4. ab 性能指标

[回目录](#)

在进行性能测试过程中有几个指标比较重要：

4.1 吞吐率（ Requests per second ）

[回目录](#)

服务器并发处理能力的量化描述，单位是reqs/s，指的是在某个并发用户数下单位时间内处理的请求数。某个并发用户数下单位时间内能处理的最大请求数，称之为最大吞吐率。记住：吞吐率是基于并发用户数的。这句话代表了两个含义：

- a、吞吐率和并发用户数相关
- b、不同的并发用户数下，吞吐率一般是不同的

计算公式：总请求数/处理完成这些请求数所花费的时间，即

Request per second=Complete requests/Time taken for tests

必须要说明的是，这个数值表示当前机器的整体性能，值越大越好。

4.2 并发连接数（ The number of concurrent connections ）

[回目录](#)

并发连接数指的是某个时刻服务器所接受的请求数目，简单的讲，就是一个会话。

4.3 并发用户数（ Concurrency Level ）

[回目录](#)

要注意区分这个概念和并发连接数之间的区别，一个用户可能同时会产生多个会话，也即连接数。

4.4 用户平均请求等待时间（ Time per request ）

[回目录](#)

计算公式：处理完成所有请求数所花费的时间/（总请求数/并发用户数），即：

Time per request=Time taken for tests/ (Complete requests/Concurrency Level)

4.5 服务器平均请求等待时间 (Time per request:across all concurrent requests)

[回目录](#)

计算公式：处理完成所有请求数所花费的时间/总请求数，即：

Time taken for/testsComplete requests

可以看到，它是吞吐率的倒数。同时，它也等于用户平均请求等待时间/并发用户数，即

Time per request/Concurrency Level

ii. CentOS7.2 下安装 php 加速软件 Xcache (在 php 主机上完成下面的操作)

[回目录](#)

说明：

- php 安装目录： /usr/local/php5.6
- php.ini 配置文件路径： /etc/php.ini
- php 网页根目录： /var/www/benet

1. 安装 xcache

[回目录](#)

wget <http://xcache.lighttpd.net/pub/Releases/3.2.0/xcache-3.2.0.tar.gz> #下载

```
[root@phpserver ~]# tar zxf xcache-3.2.0.tar.gz      #解压  
[root@phpserver ~]# cd xcache-3.2.0/      #进入安装目录  
[root@phpserver xcache-3.2.0]# /usr/local/php5.6/bin/phpize #用p  
hpize生成configure配置文件  
[root@phpserver xcache-3.2.0]# ./configure --enable-xcache --ena  
ble-xcache-coverager --enable-xcache-optimizer --with-php-config  
=/usr/local/php5.6/bin/php-config #配置  
[root@phpserver xcache-3.2.0]# make && make install #编译、安装  
  
Installing shared extensions:      /usr/local/php5.6/lib/php/exte  
nsions/no-debug-zts-20131226/
```

安装完成之后，出现下面的界面，记住以下路径，后面会用到

```
/usr/local/php5.6/lib/php/extensions/no-debug-zts-20131226/
```

2.创建 xcache 缓存文件

[回目录](#)

```
# touch /tmp/xcache  
# chmod 777 /tmp/xcache
```

3.拷贝 xcache 后台管理程序到网站根目录

[回目录](#)

```
[root@phpserver xcache-3.2.0]# cp -r htdocs/ /var/www/benet/xcac  
he
```

4.配置 php 支持 xcache

[回目录](#)

```
vi /etc/php.ini #编辑配置文件，在最后一行添加以下内容
```

```
[xcache-common]
extension = /usr/local/php5.6/lib/php/extensions/no-debug-zts-20
131226/xcache.so
[xcache.admin]
xcache.admin.enable_auth = Off
[xcache]
xcache.shm_scheme ="mmap"
xcache.size=60M
xcache.count =1
xcache.slots =8K
xcache.ttl=0
xcache.gc_interval =0
xcache.var_size=64M
xcache.var_count =1
xcache.var_slots =8K
xcache.var_ttl=0
xcache.var_maxttl=0
xcache.var_gc_interval =300
xcache.test =Off
xcache.readonly_protection = Off
xcache.mmap_path ="/tmp/xcache"
xcache.coredump_directory =""
xcache.cacher =On
xcache.stat=On
xcache.optimizer =Off
[xcache.coverager]
xcache.coverager =On
xcache.coveragedump_directory =""

```

将 `xcache` 目录拷贝到apache主机的网页文档目录下

```
[root@phpserver ~]# scp -r /var/www/benet/xcache/ root@192.168.3
1.83:/var/www/benet/
```

5. 测试

[回目录](#)

```
service php-fpm restart #重启php-fpm
```

浏览器打开网站根目录下面的 xcache

```
http:// http://192.168.31.83/xcache
```

可以看到如下页面：

The screenshot shows the XCache Cacher interface at <http://192.168.31.83/xcache/cacher/>. The top navigation bar includes links for XCache, 帮助文档 (Help), INI 参考 (INI Reference), 获得支持 (Get Support), 讨论 (Discussion), PHP, Lighttpd, 缓存器 (Cache), 代码覆盖查看器 (Code Coverage Viewer), and 诊断 (Diagnosis). The main content area has tabs for 摘要信息 (Summary Information), 列出PHP (List PHP), and 列变量数据 (List Variable Data). The "摘要信息" tab is active, displaying the "缓存区" (Cache Zone) table. The table has columns for 缓存 (Cache), 槽 (Slot), 大小 (Size), 剩余 (Remaining), 百分比图 (Percentage Graph), 操作 (Operation), 状态 (Status), 命中 (Hit), 24H 分布 (24H Distribution), 命中/H (Hit/H), 命中/S (Hit/S), 更新 (Update), 跳过 (Skip), 内存不足 (Memory Shortage), 错误 (Error), 保护 (Protection), 缓存 (Cache), 待删 (Delete), and GC. It shows two entries: 'php#0' and 'var#0'. Below the table is a legend: 已用 (Used) represented by green, 已用块 (Used Block) by blue, and 命中 (Hit) by red.

至此，Linux下安装 php 加速软件 xcache 教程完成

执行 ab 压力测试

执行第一次压力测试：

```
[root@www ~]# ab -c 100 -n 1000 http://192.168.31.83/index.php
This is ApacheBench, Version 2.3 <$Revision: 1748469 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.31.83 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
```

```

Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests

```

```

Server Software:          Apache
Server Hostname:         192.168.31.83
Server Port:              80

```

```

Document Path:            /index.php
Document Length:          85006 bytes

```

```

Concurrency Level:        100
Time taken for tests:    1.773 seconds
Complete requests:        1000
Failed requests:          368
    (Connect: 0, Receive: 0, Length: 368, Exceptions: 0)
Total transferred:        85259504 bytes
HTML transferred:         85013504 bytes
Requests per second:      563.88 [#/sec] (mean)
Time per request:         177.344 [ms] (mean)
Time per request:         1.773 [ms] (mean, across all concurrent
                           requests)
Transfer rate:             46948.95 [Kbytes/sec] received

```

Connection Times (ms)

	min	mean	[+/-sd]	median	max
Connect:	0	1	2.7	0	16
Processing:	31	167	73.7	140	406
Waiting:	23	162	73.1	136	401
Total:	40	168	73.0	141	406

Percentage of the requests served within a certain time (ms)

50%	141
66%	152
75%	176

80%	188
90%	294
95%	389
98%	396
99%	400
100%	406 (longest request)

执行第二次压力测试

```
[root@www ~]# ab -c 100 -n 1000 http://192.168.31.83/index.php
This is ApacheBench, Version 2.3 <$Revision: 1748469 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.31.83 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests

Server Software:        Apache
Server Hostname:       192.168.31.83
Server Port:           80

Document Path:          /index.php
Document Length:        85006 bytes

Concurrency Level:      100
Time taken for tests:   1.264 seconds
```

```

Complete requests:      1000
Failed requests:       316
          (Connect: 0, Receive: 0, Length: 316, Exceptions: 0)
Total transferred:    85257983 bytes
HTML transferred:     85011983 bytes
Requests per second:   790.93 [#/sec] (mean)
Time per request:     126.434 [ms] (mean)
Time per request:     1.264 [ms] (mean, across all concurrent
requests)
Transfer rate:        65852.24 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   2.7      0     11
Processing:    33  120  20.6    120    180
Waiting:       28  115  20.0    115    179
Total:         44  121  19.2    121    180

Percentage of the requests served within a certain time (ms)
 50%    121
 66%    124
 75%    127
 80%    128
 90%    147
 95%    159
 98%    166
 99%    169
100%   180 (longest request)

```

查看 xcache 的命中率：

4.lamp-优化

① 192.168.31.83/xcache/cacher/?do=listphp																	拖拽上传
php#0 1	{DOCROOT}/xcache/cacher/summary.tpl.php	9	2.84 K	0	1	0	0	245.00 b	1h	64768	72275940	1499	5m	13m			
php#0 2	{DOCROOT}/xcache/coverager/coverager.tpl.php	0	43.78 K	0	1	0	8	4.72 K	1h	64768	135302848	2950	6m	6m			
php#0 3	{DOCROOT}/xcache/coverager/index.php	0	76.24 K	0	1	1	2	7.76 K	1h	64768	135302849	2951	6m	6m			
php#0 4	{DOCROOT}/xcache/cacher/lang/zh-simplified.php	17	12.13 K	1	1	0	0	3.88 K	1h	64768	135302839	3057	0s	13m			
php#0 5	{DOCROOT}/xcache/coverager/common.php	0	1.73 K	0	1	0	0	106.00 b	1h	64768	135302844	3086	6m	6m			
php#0 6	{DOCROOT}/xcache/coverager/config.default.php	0	1.25 K	0	1	0	0	363.00 b	1h	64768	135302845	3087	6m	6m			
php#0 7	{DOCROOT}/xcache/cacher/sub/entrylist.tpl.php	13	36.52 K	0	1	0	0	3.38 K	1h	64768	203079887	3489	4m	12m			
php#0 8	{DOCROOT}/xcache/coverager/lang/zh-simplified.php	0	1.89 K	0	1	0	0	250.00 b	1h	64768	203079894	3512	6m	6m			
php#0 9	{DOCROOT}/xcache/cacher/sub/moduleinfo.tpl.php	9	2.16 K	0	1	0	0	100.00 b	1h	64768	203079888	3518	5m	13m			
php#0 10	{DOCROOT}/xcache/cacher/sub/summary.tpl.php	17	46.36 K	1	1	0	0	4.49 K	1h	64768	203079889	3519	0s	13m			
php#0 11	{DOCROOT}/xcache/common/lang/zh-simplified.php	24	1.95 K	1	1	0	0	303.00 b	1h	64768	71304276	4176	0s	13m			
php#0 12	{DOCROOT}/xcache/cacher/listentries.tpl.php	7	7.53 K	1	1	0	0	656.00 b	1h	64768	72279612	4611	0s	12m			
php#0 13	{DOCROOT}/xcache/cacher/index.php	17	73.64 K	1	1	0	9	7.22 K	1h	64768	72279611	4612	0s	13m			
php#0 14	{DOCROOT}/xcache/cacher/config.default.php	17	1.50 K	1	1	0	0	461.00 b	1h	64768	72279607	4616	0s	13m			
php#0 15	{DOCROOT}/xcache/cacher/common.php	17	1.72 K	1	1	0	0	103.00 b	1h	64768	72279606	4617	0s	13m			
php#0 16	{DOCROOT}/xcache/common/header.tpl.php	19	7.44 K	1	1	0	0	1.23 K	1h	64768	4748800	4645	0s	13m			
php#0 17	{DOCROOT}/xcache/config.default.php	24	1.50 K	1	1	0	0	425.00 b	1h	64768	4748803	4646	0s	13m			
php#0 18	{DOCROOT}/index.php	1998	1,002.00 b	0	1	1	0	20.00 b	2d	64768	139982419	4656	3m	3m			

回目录

nginx反向代理&&负载均衡&&缓存

目录

- I. nginx 反向代理：Web服务器的调度器
 - i. 反向代理的作用
 - ii. 什么是 nginx
 - iii. Nginx 的核心特点
- II. nginx 配置反向代理
 - 环境
 - i. 安装 nginx
 - ii. 编写 nginx 服务脚本
 - iii. 查看 nginx 加载的模块
- III. nginx-sticky-module 模块
- IV. load-balance 其它调度方案
- V. 负载均衡与健康检查
- VI. nginx 的 proxy 缓存使用
- VII. 常用指令说明
 - i. main 全局配置
 - 附- CPU 工作状况--输入 top 后，按1 查看
 - ii. 模块 http_gzip
 - iii. 模块 http_stream
 - iv. nginx 修改版本等信息

代理服务可简单的分为 正向代理 和 反向代理：

正向代理

用于代理内部网络对Internet的连接请求(如 VPN / NAT),客户端指定代理服务器,并将本来要直接发送给目标Web服务器的HTTP请求先发送到代理服务器上,然后由代理服务器去访问Web服务器,并将Web服务器的Response回传给客户端:

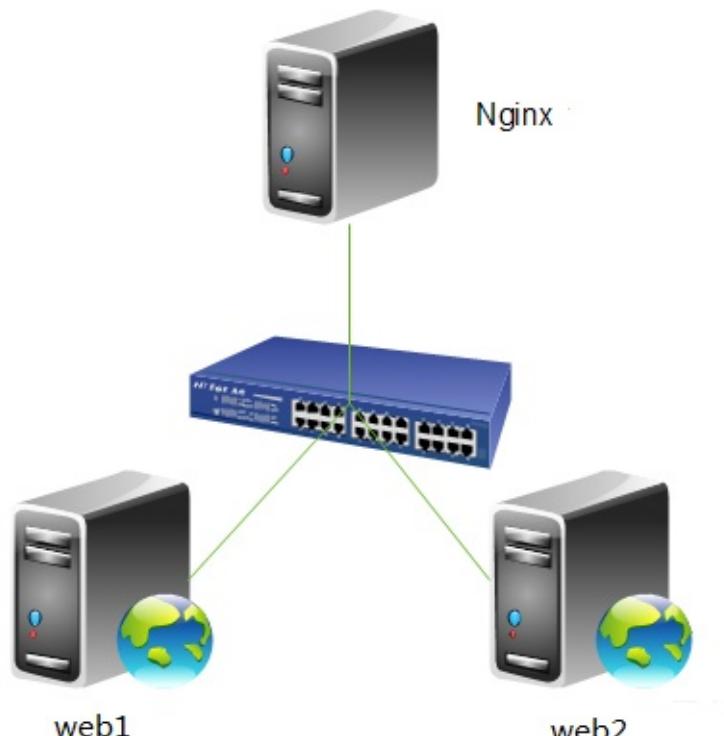
反向代理

与正向代理相反,如果局域网向Internet提供资源,并让Internet上的其他用户可以访问局域网内资源,也可以设置一个代理服务器,它提供的服务就是反向代理. 反向代理服务器接受来自Internet的连接,然后将请求转发给内部网络上的服务器,并将Response回传给Internet上请求连接的客户端:

I. nginx 反向代理 : Web服务器的调度器

[回目录](#)

反向代理（ Reverse Proxy ）方式是指以代理服务器来接受客户端的连接请求，然后将请求转发给网络上的web服务器（可能是 apache 、 nginx 、 tomcat 、 iis 等），并将从web服务器上得到的结果返回给请求连接的客户端，此时代理服务器对外就表现为一个服务器。

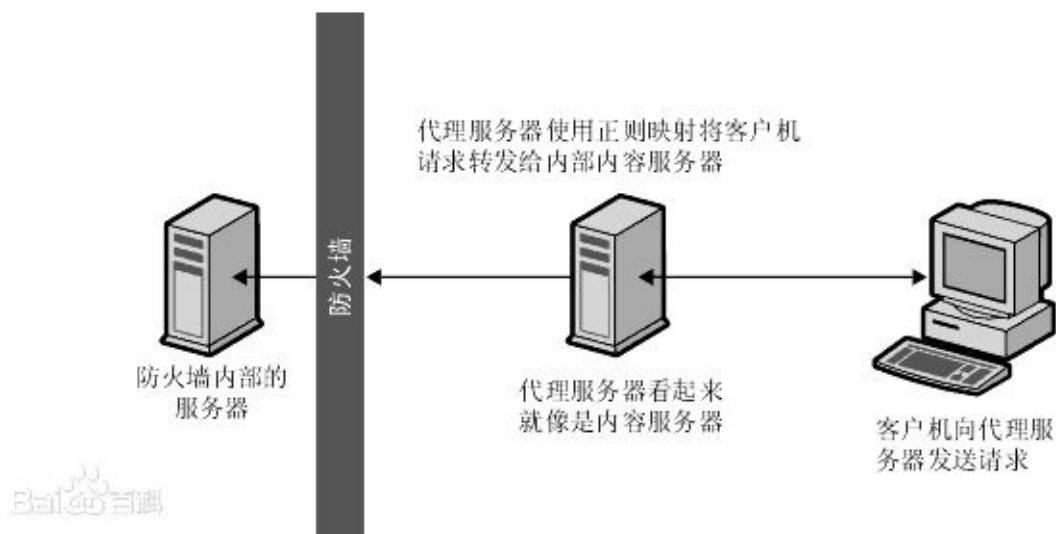


从上图可以看出：反向代理服务器代理网站Web服务器接收Http请求，对请求进行转发。

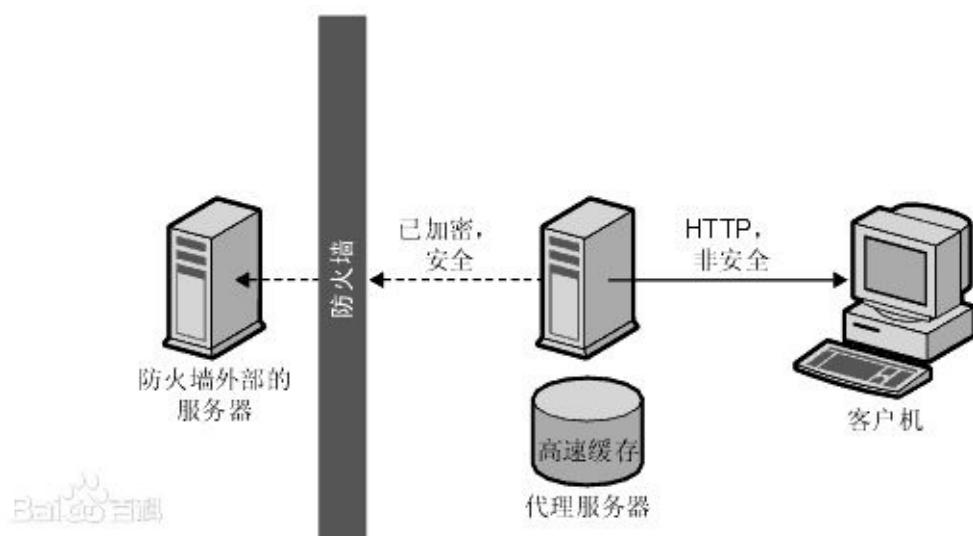
I. 反向代理的作用

[回目录](#)

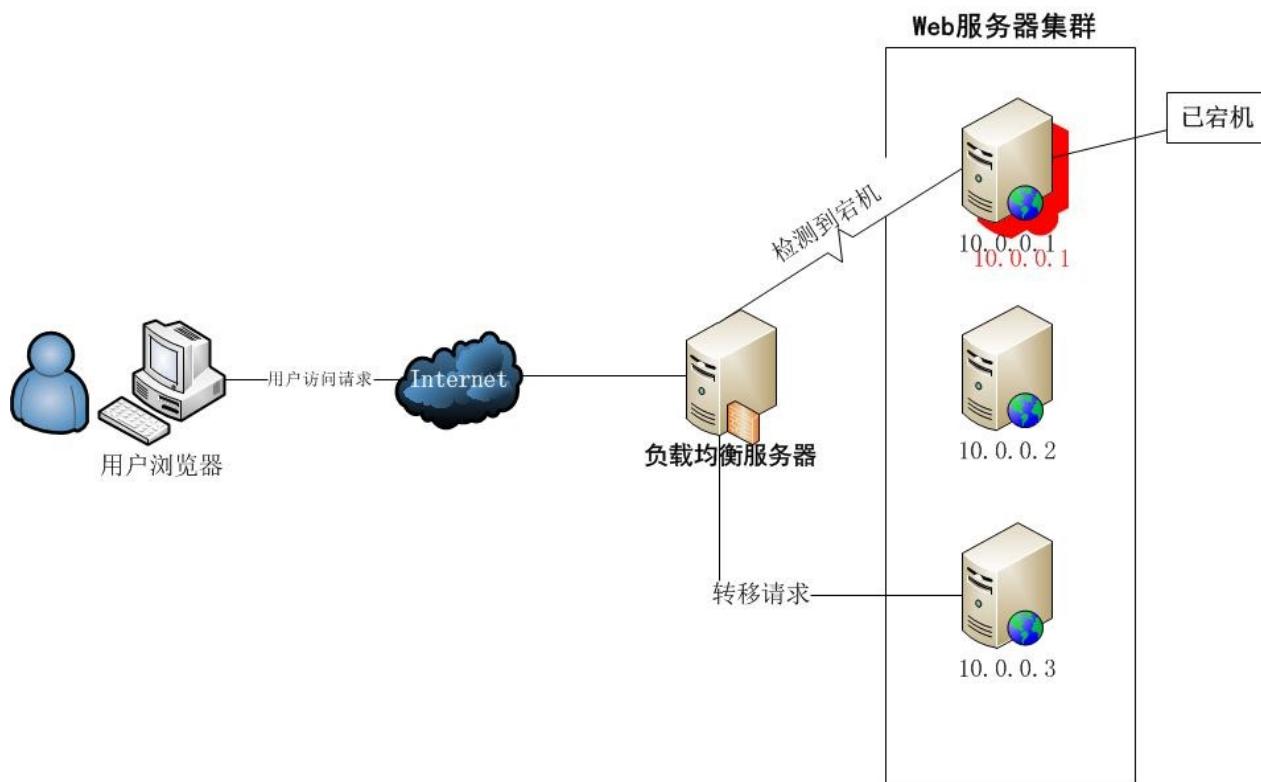
1. 保护网站安全：任何来自Internet的请求都必须先经过代理服务器；



1. 通过配置缓存功能加速Web请求：可以缓存真实Web服务器上的某些静态资源，减轻真实Web服务器的负载压力；



1. 实现负载均衡：充当负载均衡服务器均衡地分发请求，平衡集群中各个服务器的负载压力；



ii. 什么是 nginx

回目录

Nginx 是一款轻量级的网页服务器、反向代理器以及电子邮件代理服务器。因它的稳定性、丰富的功能集、示例配置文件和低系统资源的消耗而闻名。Nginx（发音同engine x），它是由俄罗斯程序员 Igor Sysoev 所开发的。起初是供俄国大型的门户网站及搜索引擎 Rambler（俄语：Рамблер）使用。此软件 BSD-like 协议下发行，可以在 UNIX、GNU/Linux、BSD、Mac OS X、Solaris，以及 Microsoft Windows 等操作系统中运行。

Nginx 的应用现状

Nginx 已经在俄罗斯最大的门户网站——Rambler Media (www.rambler.ru) 上运行，同时俄罗斯超过 20% 的虚拟主机平台采用 Nginx 作为反向代理服务器。在国内，已经有 淘宝、新浪博客、新浪播客、网易新闻、六间房、56.com、Discuz!、水木社区、豆瓣、YUPOO、海内、迅雷在线 等多家网站使用 Nginx 作为 Web 服务器或反向代理服务器。

iii. Nginx 的核心特点

回目录

(1) 跨平台：Nginx 可以在大多数OS编译运行，而且也有Windows的版本；

(2) 配置异常简单：非常容易上手。

(3) 非阻塞、高并发连接：官方测试能够支撑5万并发连接，在实际生产环境中跑到2~3万并发连接数。（这得益于 Nginx 使用了最新的 epoll 模型）；

注：对于一个Web服务器来说，首先看一个请求的基本过程：建立连接—接收数据—发送数据，在系统底层看来：上述过程（建立连接—接收数据—发送数据）在系统底层就是读写事件。

如果采用阻塞调用的方式，当读写事件没有准备好时，那么就只能等待，当前线程被挂起，等事件准备好了，才能进行读写事件。

非阻塞方式就是：事件马上返回，告诉你事件还没准备好呢，过会再来吧。过一会，再来检查一下事件，直到事件准备好了为止，在这期间，你就可以先去做其它事情，然后再来看看事件好了没。虽然不阻塞了，但你得不时地过来检查一下事件的状态，你可以做更多的事情了，但带来的开销也是不小的。非阻塞调用指在不能立刻得到结果之前，该调用不会阻塞当前线程

(4) 事件驱动：通信机制采用epoll模型，支持更大的并发连接。

非阻塞通过不断检查事件的状态来判断是否进行读写操作，这样带来的开销很大，因此就有了异步非阻塞的事件处理机制。这种机制让你可以同时监控多个事件，调用他们是非阻塞的，但可以设置超时时间，在超时时间之内，如果有事件准备好了，就返回。这种机制解决了上面阻塞调用与非阻塞调用的两个问题。

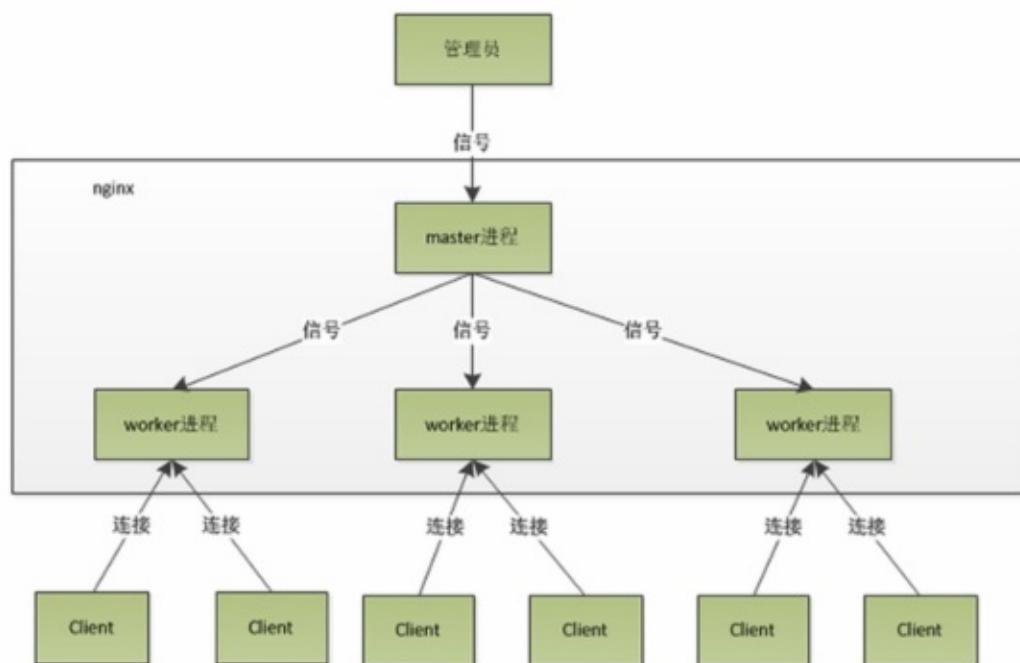
以 epoll 模型为例：当事件没有准备好时，就放入 epoll (队列)里面。如果有事件准备好了，那么就去处理；当事件没有准备好时，才在 epoll 里面等着。这样，我们就可以并发处理大量的并发了，当然，这里的并发请求，是指未处理完的

请求，线程只有一个，所以同时能处理的请求当然只有一个了，只是在请求之间进行不断地切换而已，切换也是因为异步事件未准备好，而主动让出的。这里的切换是没有任何代价，你可以理解为循环处理多个准备好的事件。

多线程方式相比，这种事件处理方式是有很大的优势的，不需要创建线程，每个请求占用的内存也很少，没有上下文切换，事件处理非常的轻量级，并发数再多也不会导致无谓的资源浪费（上下文切换）。对于apache服务器，每个请求会独占一个工作线程，当并发数上到几千时，就同时有几千的线程在处理请求了。这对操作系统来说，是个不小的挑战：因为线程带来的内存占用非常大，线程的上下文切换带来的cpu开销很大，自然性能就上不去，从而导致在高并发场景下性能下降严重。

总结：通过异步非阻塞的事件处理机制，Nginx实现由进程循环处理多个准备好的事件，从而实现高并发和轻量级。

（5）Master/Worker结构：一个master进程，生成一个或多个worker进程。



注：Master-Worker设计模式核心思想是将原来串行的逻辑并行化，并将逻辑拆分成很多独立模块并行执行。其中主要包含两个主要组件Master和Worker，Master主要将逻辑进行拆分，拆分为互相独立的部分，同时维护了Worker队列，将每个独立部分下发到多个Worker并行执行，Worker主要进行实际逻辑计算，并将结果返回给Master。

nginx采用这种进程模型有什么好处？采用独立的进程，可以让互相之间不会影响，一个进程退出后，其它进程还在工作，服务不会中断，Master 进程则很快重新启动新的Worker进程。当然，Worker进程的异常退出，肯定是程序有bug了，异常退出，会导致当前Worker上的所有请求失败，不过不会影响到所有请求，所以降低了风险。

(6) 内存消耗小：处理大并发的请求内存消耗非常小。在3万并发连接下，开启的10个Nginx 进程才消耗150M内存（ $15M \times 10 = 150M$ ）。

(7) 内置的健康检查功能：如果 Nginx 代理的后端的某台 Web 服务器宕机了，不会影响前端访问。

(8) 节省带宽：支持 GZIP 压缩，可以添加浏览器本地缓存的 Header 头。

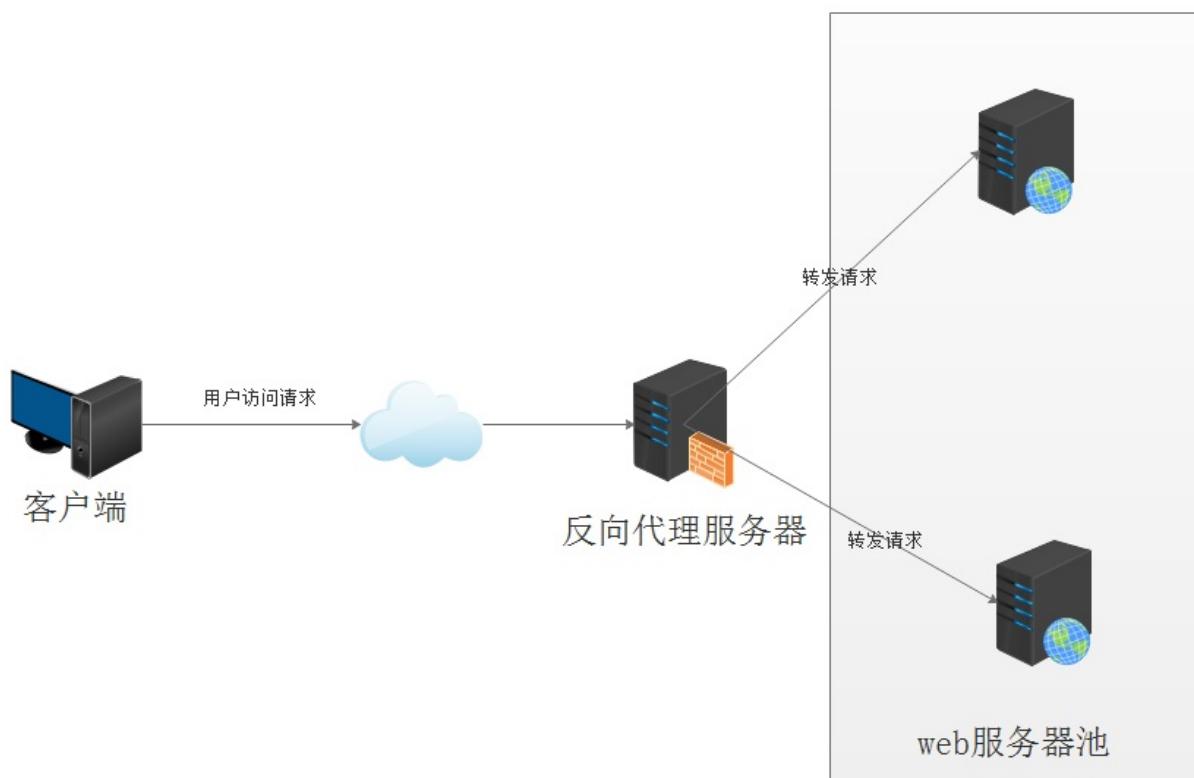
(9) 稳定性高：用于反向代理，宕机的概率微乎其微。

Nginx+apache构筑Web服务器集群的负载均衡

II. nginx 配置反向代理

[回目录](#)

配置nginx作为反向代理和负载均衡，同时利用其缓存功能，将静态页面在nginx缓存，以达到降低后端服务器连接数的目的并检查后端web服务器的健康状况。



环境

[回目录](#)

```
OS : centos7.2  nginx : 192.168.31.83  apache1 : 192.168.31.141  
apache2 : 192.168.31.250
```

安装 `zlib-devel` 、 `pcre-devel` 等依赖包

```
[root@www ~]# yum -y install gcc gcc-c++ make libtool zlib zlib-devel pcre pcre-devel openssl openssl-devel
```

注：结合 `proxy` 和 `upstream` 模块实现后端 web 负载均衡 使用 `proxy` 模块实现静态文件缓存 结合 nginx 默认自带的 `ngx_http_proxy_module` 模块和 `ngx_http_upstream_module` 模块实现后端服务器的健康检查，也可以使用第三方模块 `nginx_upstream_check_module` 使用 `nginx-sticky-module` 扩展模块实现 `Cookie` 会话黏贴（保持会话） 使用 `ngx_cache_purge` 实现更强大的缓存清除功能 上面提到的2个模块都属于第三方扩展模块，需要提前下好源码，然后编译时通过`-- add-module=src_path` 一起安装。

i. 安装 nginx

[回目录](#)

```
[root@www ~]# groupadd www      #添加www组
[root@www ~]# useradd -g www www -s /sbin/nologin      #创建nginx运行账户www并加入到www组，不允许www用户直接登录系统

#tar zxf nginx-1.10.2.tar.gz
#tar zxf ngx_cache_purge-2.3.tar.gz
#tar zxf master.tar.gz
# cd nginx-1.10.2/
[root@www nginx-1.10.2]# ./configure --prefix=/usr/local/nginx1.10 --user=www --group=www --with-http_stub_status_module --with-http_realip_module --with-http_ssl_module --with-http_gzip_static_module --http-client-body-temp-path=/var/tmp/nginx/client --http-proxy-temp-path=/var/tmp/nginx/proxy --http-fastcgi-temp-path=/var/tmp/nginx/fcgi --with-pcre --add-module=../ngx_cache_purge-2.3 --with-http_flv_module --add-module=../nginx-goodies-nginx-sticky-module-ng-08a395c66e42
[root@www nginx-1.10.2]# make && make install
```

注： nginx 的所有模块必须在编译的时候添加，不能再运行的时候动态加载。

优化 nginx 程序的执行路径

```
[root@www nginx-1.10.2]# ln -s /usr/local/nginx1.10/sbin/nginx /usr/local/sbin/
[root@www nginx-1.10.2]# nginx -t
[root@www nginx-1.10.2]# mkdir -p /var/tmp/nginx/client
[root@www nginx-1.10.2]# chown -R www:www /var/tmp/nginx/
[root@www nginx-1.10.2]# nginx -t
nginx: the configuration file /usr/local/nginx1.10/conf/nginx.conf syntax is ok
nginx: configuration file /usr/local/nginx1.10/conf/nginx.conf test is successful
```

ii. 编写 nginx 服务脚本

[回目录](#)

```
[root@www ~]# vi /etc/init.d/nginx 内容如下：
#!/bin/bash
# chkconfig: 2345 99 20
# description: Nginx Service Control Script
PROG="/usr/local/nginx1.10/sbin/nginx"
PIDF="/usr/local/nginx1.10/logs/nginx.pid"
case "$1" in
    start)
        netstat -anplt |grep ":80" &> /dev/null && pgrep "nginx" &> /dev/null
        if [ $? -eq 0 ]
        then
            echo "Nginx service already running."
        else
            $PROG -t &> /dev/null
            if [ $? -eq 0 ] ; then
                $PROG
                echo "Nginx service start success."
            else
                $PROG -t
            fi
        fi
    ;;
    stop)
        netstat -anplt |grep ":80" &> /dev/null && pgrep "nginx" &> /dev/null
        if [ $? -eq 0 ]
        then
            kill -s QUIT $(cat $PIDF)
            echo "Nginx service stop success."
        else
            echo "Nginx service already stop"
        fi
    ;;
    restart)
```

```

$0 stop
$0 start
;;
status)
netstat -anplt |grep ":80" &> /dev/null && pgrep "nginx" &> /
dev/null
if [ $? -eq 0 ]
then
    echo "Nginx service is running."
else
    echo "Nginx is stop."
fi
;;
reload)
netstat -anplt |grep ":80" &> /dev/null && pgrep "nginx" &> /
dev/null
if [ $? -eq 0 ]
then
    $PROG -t &> /dev/null
    if [ $? -eq 0 ] ; then
        kill -s HUP $(cat $PIDF)
        echo "reload Nginx config success."
    else
        $PROG -t
    fi
else
    echo "Nginx service is not run."
fi
;;
*)
echo "Usage: $0 {start|stop|restart|reload}"
exit 1
esac

```

```
[root@www ~]# chmod +x /etc/init.d/nginx
[root@www ~]# chkconfig --add nginx
[root@www ~]# chkconfig nginx on
[root@www ~]# service nginx start
Nginx service start success.
[root@www ~]# service nginx status
Nginx service is running.
[root@www ~]# netstat -anpt | grep nginx
tcp      0      0 0.0.0.0:80          0.0.0.0:*
      LISTEN      3977/nginx: master
[root@www ~]# firewall-cmd --permanent --add-port=80/tcp
success
[root@www ~]# firewall-cmd --reload
Success
```

注：如果你想在已安装好的 nginx 上添加第三方模块，依然需要重新编译，但为了不覆盖你原有的配置，请不要 `make install`，而是直接拷贝可执行文件：

```
# nginx -v
```

```
[root@www nginx-1.10.2]# ./configure --add-module=..... #你的第三方
模块
[root@www nginx-1.10.2] #make后不要make install, 改为手动拷贝，先备份
[root@www nginx-1.10.2] #cp /usr/local/nginx1.10/sbin/nginx /usr
/local/nginx1.10/sbin/nginx.bak
[root@www nginx-1.10.2] #cp objs/nginx /usr/local/nginx1.10/sbin
/nginx
```

iii. 查看 nginx 加载的模块

[回目录](#)

```
[root@www ~]## nginx -V
nginx version: nginx/1.10.2
built by gcc 4.8.5 20150623 (Red Hat 4.8.5-4) (GCC)
built with OpenSSL 1.0.1e-fips 11 Feb 2013
TLS SNI support enabled
configure arguments: --prefix=/usr/local/nginx1.10 --user=www --
group=www --with-http_stub_status_module --with-http_realip_modu
le --with-http_ssl_module --with-http_gzip_static_module --http-
client-body-temp-path=/var/tmp/nginx/client --http-proxy-temp-pa
th=/var/tmp/nginx/proxy --http-fastcgi-temp-path=/var/tmp/nginx/
fcgi --with-pcre --add-module=../ngx_cache_purge-2.3 --with-http
_flv_module --add-module=../nginx-goodies-nginx-sticky-module-ng
-08a395c66e42
```

nginx 的所有模块必须在编译的时候添加，不能再运行的时候动态加载。

III. nginx-sticky-module 模块

回目录

这个模块的作用是通过 `cookie` 黏贴的方式将来自同一个客户端（浏览器）的请求发送到同一个后端服务器上处理，这样一定程度上可以解决多个 `backend servers` 的 `session` 同步的问题——因为不再需要同步，而 `RR` 轮询 模式必须要运维人员自己考虑 `session` 同步的实现。

另外内置的 `ip_hash` 也可以实现根据客户端 `IP` 来分发请求，但它很容易造成负载不均衡的情况，而如果 `nginx` 前面有 `CDN` 网络或者来自同一局域网的访问，它接收的客户端 `IP` 是一样的，容易造成负载不均衡现象。`nginx-sticky-module` 的 `cookie` 过期时间，默认浏览器关闭就过期。

这个模块并不适合不支持 `Cookie` 或手动禁用了 `cookie` 的浏览器，此时默认 `sticky` 就会切换成 `RR`。它不能与 `ip_hash` 同时使用。

```
upstream backend {
    server 192.168.31.141:80 weight=1;
    server 192.168.31.250:80 weight=1;
    sticky;
}
```

配置起来超级简单，一般来说一个 `sticky` 指令就够了。相关信息可以查看[官方文档](#)

IV. load-balance 其它调度方案

[回目录](#)

这里顺带介绍一下 `nginx` 的负载均衡模块支持的其它调度算法：

- 轮询（默认）：每个请求按时间顺序逐一分配到不同的后端服务器，如果后端某台服务器宕机，故障系统被自动剔除，使用户访问不受影响。`Weight` 指定轮询权值，`Weight` 值越大，分配到的访问机率越高，主要用于后端每个服务器性能不均的情况下。
- `ip_hash`：每个请求按访问 `IP` 的 `hash` 结果分配，这样来自同一个 `IP` 的访客固定访问一个后端服务器，有效解决了动态网页存在的 `session` 共享问题。当然如果这个节点不可用了，会发到下个节点，而此时没有 `session` 同步的话就注销掉了。
- `least_conn`：请求被发送到当前活跃连接最少的 `realserver` 上。会考虑 `weight` 的值。
- `url_hash`：此方法按访问 `url` 的 `hash` 结果来分配请求，使每个 `url` 定向到同一个后端服务器，可以进一步提高后端缓存服务器的效率。`Nginx` 本身是不支持 `url_hash` 的，如果需要使用这种调度算法，必须安装 `Nginx` 的 `hash` 软件包 `nginx_upstream_hash`。
- `fair`：这是比上面两个更加智能的负载均衡算法。此种算法可以依据页面大小和加载时间长短智能地进行负载均衡，也就是根据后端服务器的响应时间来分配请求，响应时间短的优先分配。`Nginx` 本身是不支持 `fair` 的，如果需要使用这种调度算法，必须下载 `Nginx` 的 `upstream_fair` 模块。

V. 负载均衡与健康检查

[回目录](#)

严格来说，nginx自带是没有针对负载均衡后端节点的健康检查的，但是可以通过默认自带的 `ngx_http_proxy_module` 模块和 `ngx_http_upstream_module` 模块中的相关指令来完成当后端节点出现故障时，自动切换到下一个节点来提供访问。

```
upstream backend {
    sticky;
    server 192.168.31.141:80 weight=1 max_fails=2 fail_timeout=10s;
    server 192.168.31.250:80 weight=1 max_fails=2 fail_timeout=10s;
}
server {
    ...
location / {
    proxy_pass http://backend;
}
...
}
```

- `weight` : 轮询权值也是可以用在 `ip_hash` 的，默认值为1
- `max_fails` : 允许请求失败的次数，默认为1。当超过最大次数时，返回 `proxy_next_upstream` 模块定义的错误。
- `fail_timeout` : 有两层含义，一是在10s 时间内最多容许2次失败；二是在经历了2次失败以后，10s时间内不分配请求到这台服务器。

VI. nginx 的 proxy 缓存使用

[回目录](#)

缓存也就是将 `js` 、 `css` 、 `image` 等静态文件从后端服务器缓存到nginx指定的缓存目录下，既可以减轻后端服务器负担，也可以加快访问速度，但这样缓存及时清理成为了一个问题，所以需要 `ngx_cache_purge` 这个模块来在过期时间未到

之前，手动清理缓存。

proxy 模块中常用的指令时 proxy_pass 和 proxy_cache . nginx的web缓存功能主要是由 proxy_cache 、 fastcgi_cache 指令集和相关指令集完成， proxy_cache 指令负责反向代理缓存后端服务器的静态内容， fastcgi_cache 主要用来处理 FastCGI 动态进程缓存。

```

http {
    #$upstream_cache_status记录缓存命中率
    log_format main '$remote_addr - $remote_user [$time_local] "$request"
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"
                      "'"$upstream_cache_status"'';

    access_log logs/access.log main;
    proxy_buffering on;      #代理的时候，开启或关闭缓冲后端服务器的响应
    proxy_temp_path /usr/local/nginx1.10/proxy_temp;
        proxy_cache_path /usr/local/nginx1.10/proxy_cache levels=1:2
        keys_zone=my-cache:100m
        max_size=1000m inactive=600m max_size=2g;
    server {
        listen      80;
        server_name localhost;
        root      html;
        index     index.php index.html index.htm;
        location  ~/purge(/.*) {
            allow 127.0.0.1;
            allow 192.168.31.0/24;
            deny all;
            proxy_cache_purge my-cache $host$is_args$args;
        }
        location ~ .*\.(gif|jpg|png|html|htm|css|js|ico|swf|pdf)(.*)
        {
            proxy_pass  http://backend;
            proxy_redirect off;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forward
        }
    }
}

```

```
ed_for;
    proxy_next_upstream error timeout invalid_header http
_500 http_502 http_503 http_504;
    proxy_cache my-cache;
    add_header Nginx-Cache $upstream_cache_status;
    proxy_cache_valid 200 304 301 302 8h;
    proxy_cache_valid 404 1m;
    proxy_cache_valid any 1d;
    proxy_cache_key $host$uri$is_args$args;
    expires 30d;
}
```

注：

- `proxy_buffering on` ; 代理的时候，开启或关闭缓冲后端服务器的响应。
当开启缓冲时，nginx尽可能快地从被代理的服务器接收响应，再将它存入缓冲区中。
- `proxy_temp_path` : 缓存临时目录。后端的响应并不直接返回客户端，而是先写到一个临时文件中，然后被`rename`一下当做缓存放在`proxy_cache_path`。0.8.9版本以后允许`temp`和`cache`两个目录在不同文件系统上（分区），然而为了减少性能损失还是建议把它们设成一个文件系统上。
- `proxy_cache_path` : 设置缓存目录，目录里的文件名是`cache_key`的MD5值。
- `levels=1:2 keys_zone=my-cache:50m` 表示采用2级目录结构，第一层目录只有一个字符，是由`levels=1:2`设置，总共二层目录，子目录名字由二个字符组成。Web缓存区名称为`my-cache`，内存缓存空间大小为`100MB`，这个缓冲`zone`可以被多次使用。文件系统上看到的缓存文件名类似于`/usr/local/nginx1.10/proxy_cache/c/29/b7f54b2df7773722d382f4809d65029c`。`inactive=600 max_size=2g` 表示 600分钟 没有被访问的内容自动清除，硬盘最大缓存空间为`2GB`，超过这个大学将清除最近最少使用的数据。
- `proxy_cache` : 引用前面定义的缓存区`my-cache`
- `proxy_cache_key` : 定义`cache_key`，设置web缓存的key值，nginx根据key值md5哈希存储缓存
- `proxy_cache_valid` : 为不同的响应状态码设置不同的缓存时间，比如

200、302等正常结果可以缓存的时间长点，而404、500等缓存时间设置短一些，这个时间到了文件就会过期，而不论是否刚被访问过。`expires`：在响应头里设置 `Expires:` 或 `Cache-Control:max-age`，返回给客户端的浏览器缓存失效时间。

下面的 `nginx.conf` 简单的实现 `nginx` 在前端做反向代理服务器的例子，处理 `js`、`png` 等静态文件，`jsp/php` 等动态请求转发到其它服务器 `tomcat/apache`

```

user    www www;
worker_processes  4;
worker_cpu_affinity 0001 0010 0100 1000;
error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;
worker_rlimit_nofile 10240;
pid        logs/nginx.pid;
events {
    use epoll;
    worker_connections  4096;
}
http {
    include      mime.types;
    default_type application/octet-stream;
    log_format  main  '$remote_addr - $remote_user [$time_local]'
"$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for'
";
    "$upstream_cache_status";
    access_log  logs/access.log  main;
    server_tokens off;
    sendfile      on;
    #tcp_nopush    on;
    #keepalive_timeout  0;
    keepalive_timeout  65;
    #Compression Settings
    gzip on;
}

```

```

gzip_comp_level 6;
gzip_http_version 1.1;
gzip_proxied any;
gzip_min_length 1k;
gzip_buffers 16 8k;
gzip_types text/plain text/css text/javascript application/json application/javascript application/x-javascript application/xml;
gzip_vary on;
#end gzip
# http_proxy Settings
client_max_body_size    10m;
client_body_buffer_size  128k;
proxy_connect_timeout   75;
proxy_send_timeout      75;
proxy_read_timeout       75;
proxy_buffer_size        4k;
proxy_buffers            4 32k;
proxy_busy_buffers_size  64k;
proxy_temp_file_write_size 64k;
proxy_buffering on;
proxy_temp_path /usr/local/nginx1.10/proxy_temp;
proxy_cache_path /usr/local/nginx1.10/proxy_cache levels=1:2
keys_zone=my-cache:100m max_size=1000m inactive=600m max_size=2
g;
#load balance Settings
upstream backend {
    sticky;
    server 192.168.31.141:80 weight=1 max_fails=2 fail_timeo
ut=10s;
    server 192.168.31.250:80 weight=1 max_fails=2 fail_timeo
ut=10s;
}
#virtual host Settings
server {
    listen      80;
    server_name localhost;
    charset utf-8;
    location ~/purge(/.*) {
        allow 127.0.0.1;
    }
}

```

```

        allow 192.168.31.0/24;
        deny all;
        proxy_cache_purge my-cache $host$is_args$args;
    }
    location / {
        index index.php index.html index.htm;
        proxy_pass http://backend;
        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_next_upstream error timeout invalid_header http_500 http_502 http_503 http_504;
    }
    location ~ \.(gif|jpg|png|html|htm|css|js|ico|swf|pdf) (.*) {
        proxy_pass http://backend;
        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_next_upstream error timeout invalid_header http_500 http_502 http_503 http_504;
        proxy_cache my-cache;
        add_header Nginx-Cache $upstream_cache_status;
        proxy_cache_valid 200 304 301 302 8h;
        proxy_cache_valid 404 1m;
        proxy_cache_valid any 1d;
        proxy_cache_key $host$uri$is_args$args;
        expires 30d;
    }
    location /nginx_status {
        stub_status on;
        access_log off;
        allow 192.168.31.0/24;
        deny all;
    }
}

```



VII. 常用指令说明

[回目录](#)

i. main 全局配置

[回目录](#)

woker_processes 4

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) 。

附- CPU 工作状况--输入 top 后，按1 查看

```
[root@www ~]# top
top - 22:56:36 up 2:31, 2 users, load average: 0.00, 0.01, 0.05
Tasks: 443 total, 2 running, 441 sleeping, 0 stopped, 0 zombie
%Cpu0 : 0.0 us, 0.0 sy, 0.0 ni, 100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu1 : 0.0 us, 0.0 sy, 0.0 ni, 100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu2 : 0.0 us, 0.3 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu3 : 0.0 us, 0.0 sy, 0.0 ni, 100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 3866948 total, 2754648 free, 555544 used, 556756 buff/cache
KiB Swap: 4063228 total, 4063228 free, 0 used. 3031288 avail Mem
```

上面的配置表示：4核CPU，开启4个进程。 0001 表示开启第一个cpu内核， 0010 表示开启第二个cpu内核，依次类推；有多少个核，就有几位数， 1 表示该内核开启， 0 表示该内核关闭。例如：

1、2核CPU，开启2个进程

```
worker_processes 2;
worker_cpu_affinity 01 10;
```

2、2核CPU，开启4进程

```
worker_processes 4;
worker_cpu_affinity 01 10 01 10;
```

3、2核CPU，开启8进程

```
worker_processes 8;
worker_cpu_affinity 01 10 01 10 01 10 01 10;
```

4、8核CPU，开启2进程

```
worker_processes 2;
worker_cpu_affinity 10101010 01010101;
```

说明： 10101010 表示开启了第2,4,6,8内核， 01010101 表示开始了1,3,5,7内核
通过 apache 的ab测试查看nginx对CPU的使用状况：

```
top - 23:04:04 up 2:39, 3 users, load average: 0.94, 0.22, 0.11
Tasks: 450 total, 8 running, 442 sleeping, 0 stopped, 0 zombie
%Cpu0 : 1.2 us, 22.4 sy, 0.0 ni, 70.1 id, 0.0 wa, 0.0 hi, 6.2 si, 0.0 st
%Cpu1 : 1.6 us, 11.6 sy, 0.0 ni, 81.7 id, 0.0 wa, 0.0 hi, 5.2 si, 0.0 st
%Cpu2 : 1.6 us, 15.7 sy, 0.0 ni, 79.5 id, 0.0 wa, 0.0 hi, 3.2 si, 0.0 st
%Cpu3 : 1.3 us, 11.3 sy, 0.0 ni, 66.2 id, 0.0 wa, 0.0 hi, 21.2 si, 0.0 st
KiB Mem : 3866948 total, 2659984 free, 606244 used, 600720 buff/cache
KiB Swap: 4063228 total, 4063228 free, 0 used. 2942672 avail Mem
```

如果多个CPU内核的利用率都相差不多，证明nginx已经成功的利用了多核CPU。
测试结束后，CPU内核的负载应该都同时降低。

`worker_connections 4096`

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，但不能超过后面的 `worker_rlimit_nofile`。当nginx作为http服务器时，计算公式里面是除以2。

注意：为什么除以2：该公式基于http 1.1协议，一次请求大多数浏览器发送两次连接，并不是request和response响应占用两个线程（很多人也是这么认为，实际情况：请求是双向的，连接是没有方向的，由上面的图可以看出来）

为什么除以4：因nginx作为方向代理，客户端和nginx建立连接，nginx和后端服务器也要建立连接

```
worker_rlimit_nofile 10240
```

写在main部分。`worker`进程的最大打开文件数限制。默认是没有设置，如果没设置的话，这个值为操作系统的限制(`ulimit -n`)。可以限制为操作系统最大的限制65535。把这个值设高，这样nginx就不会有“too many open files”问题了。

```
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。

`http`服务器: 与提供http服务相关的一些配置参数。例如：是否使用`keepalive`啊，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I/O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，长连接请求大量小文件的时候，可以减少重建连接的开销，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。

```
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

```
server_tokens off;
```

隐藏nginx的版本号

模块 http_proxy :

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
```

语法: proxy_buffers the_number is_size; 设置缓冲区的大小和数量,从被代理的后端服务器取得的响应内容,会放置到这里. 默认情况下,一个缓冲区的大小等于内存页面大小,可能是4K也可能是8K,这取决于平台。附：查看Linux内存页大小

```
[root@www ~]# getconf PAGESIZE  
4096
```

或

```
[root@www ~]# getconf PAGE_SIZE  
4096
```

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。

`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。

ii. 模块 `http_gzip`

[回目录](#)

- `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，按照原始数据大小以 `16k` 为单位的 `4` 倍申请内存。
- `gzip_http_version 1.1` : 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/1.1`，故请设置为 `1.1`。
- `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。
- `off` - 关闭所有的代理结果数据的压缩
- `expired` - 启用压缩，如果 `header` 头中包含 “`Expires`” 头信息
- `no-cache` - 启用压缩，如果 `header` 头中包含 “`Cache-Control:no-cache`” 头信息
- `no-store` - 启用压缩，如果 `header` 头中包含 “`Cache-Control:no-store`” 头信息
- `private` - 启用压缩，如果 `header` 头中包含 “`Cache-Control:private`” 头信息
- `no_last_modified` - 启用压缩，如果 `header` 头中不包含 “`Last-Modified`” 头信息
- `no_etag` - 启用压缩，如果 `header` 头中不包含 “`ETag`” 头信息
- `auth` - 启用压缩，如果 `header` 头中包含 “`Authorization`” 头信息

- `any` – 无条件启用压缩
- `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。

III. 模块 `http_stream`

[回目录](#)

这个模块通过一个简单的调度算法来实现客户端IP到后端服务器的负载均衡，`upstream` 后接负载均衡器的名字，后端`realserver`以 `host:port` `options;` 方式组织在 `{}` 中。如果后端被代理的只有一台，也可以直接写在 `proxy_pass` 。

```
Location:  
root /var/www/html
```

定义服务器的默认网站根目录位置。如果 `locationURL` 匹配的是子目录或文件，`root`没什么作用，一般放在`server`指令里面或 `/` 下。

```
index index.jsp index.html index.htm
```

定义路径下默认访问的文件名，一般跟着 `root` 放

```
proxy_pass http://backend
```

请求转向 `backend` 定义的服务器列表，即反向代理，对应 `upstream` 负载均衡器。也可以

```
proxy_pass http://ip:port。  
proxy_redirect off;
```

指定是否修改被代理服务器返回的响应头中的 `location` 头域跟 `refresh` 头域数值

```
proxy_set_header Host $host;
```

Host的含义是表明请求的主机名，nginx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字段重写为 proxy_pass 指令设置的服务器。因为nginx作为反向代理使用，而如果后端真实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。

```
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

后端的Web服务器可以通过 X-Forwarded-For 获取用户真实IP

X_Forward_For 字段表示该条http请求是有谁发起的？如果反向代理服务器不重写该请求头的话，那么后端真实服务器在处理时会认为所有的请求都来自反向代理服务器，如果后端有防攻击策略的话，那么机器就被封掉了。因此，在配置用作反向代理的nginx中一般会增加两条配置，修改http的请求头：

```
proxy_set_header Host $host;
proxy_set_header X-Forward-For $remote_addr;
proxy_next_upstream error timeout invalid_header http_500 http_502 http_503 http_504;
```

增加故障转移，如果后端的服务器返回502、504、执行超时等错误，自动将请求转发到upstream负载均衡池中的另一台服务器，实现故障转移。

proxy_set_header X-Real-IP \$remote_addr; web服务器端获得用户的真实ip但是，实际上要获得用户的真实ip，也可以通过 X-Forward-For 下面我们来测试一下缓存功能 如果在缓存时间之内需要更新被缓存的静态文件怎么办呢，这时候就需要手动来清除缓存了。 ngx_cache_pure 清除缓存模块使用说明 用谷歌浏览器测试的时候，可以按F12调用开发工具，选择Network选项，我们可以看到，Response Headers，在这里我们可以看到，我们请求的是否是缓存

5.nginx 反向代理&&负载均衡&&缓存

The screenshot shows the Network tab in Chrome DevTools. A request for '11.gif' is selected. The 'General' section shows the Request URL as `http://192.168.31.83/11.gif`, Request Method as GET, Status Code as 304 Not Modified, and Remote Address as 192.168.31.83:80. The 'Response Headers' section includes Cache-Control: max-age=2592000, Connection: keep-alive, Date: Thu, 03 Nov 2016 13:33:42 GMT, ETag: "87e-5403666e13480", Expires: Sat, 03 Dec 2016 13:33:42 GMT, Last-Modified: Tue, 01 Nov 2016 05:14:42 GMT, Nginx-Cache: HIT, and Server: nginx/1.10.2.

从图中我们可以看到，我们访问的服务器是 192.168.31.83，缓存命中。

注：

- MISS 未命中
- EXPIRED - expired。请求被传送到后端。
- UPDATING - expired。由于 proxy/fastcgi_cache_use_stale 正在更新，将使用旧的应答。
- STALE - expired。由于 proxy/fastcgi_cache_use_stale，后端将得到过期的应答。
- HIT 命中

清除缓存：

上述配置的 `proxy_cache_purge` 指令用于方便的清除缓存，但必须按照第三方的 `ngx_cache_purge` 模块才能使用 使用 `ngx_cache_purge` 模块清除缓存（直接删除缓存目录下的文件也算一种办法）：GET方式请求URL 即使用配置文件中的 `location ~ /purge(/.*)`

浏览器访问 `http://192.168.31.83/purge/your/may/path` 来清除缓存

The screenshot shows a browser window with the URL `http://192.168.31.83/purge/11.gif`. The page content displays 'Successful purge' and provides details about the purged item: Key : 192.168.31.83/11.gif and Path: //usr/local/nginx1.10/proxy_cache/8/d4/c2f0980efb24f033e07704f2ff762d48.

备注：

- 1、`purge` 是 `ngx_cache_pure` 模块指令
- 2、`your/may/path` 是要清除的缓存文件URL路径缓存清除成功。

若要验证负载均衡和健康检查可以先关掉缓存功能

```
#proxy_buffering off;  
#sticky
```

测试过程略

扩展知识1：

iv. nginx 修改版本等信息

[回目录](#)

1、`vi /usr/local/src/nginx-1.0.12/src/core/nginx.h #编译前编辑`

```
#define nginx_version  
#define NGINX_VERSION  
#define NGINX_VER  
#define NGINX_VAR
```

修改上面的信息，即可更改 `nginx` 显示版本。

2、`vi /usr/local/src/nginx-1.0.12/src/http/ngx_http_special_response.c #编译前编辑`

```
static u_char ngx_http_error_full_tail[] =  
static u_char ngx_http_error_tail[] =
```

修改上面的信息为自己的。

3、`vi /usr/local/src/nginx-1.0.12/src/http/ngx_http_header_filter_module.c #编译前编辑`

```
static char ngx_http_server_string[] =
```

修改上面的信息为自己的。

4、编译完成之后，修改 /usr/local/nginx/conf 目录下面

```
fastcgi.conf、fastcgi.conf.default、fastcgi_params、fastcgi_params.default
```

这四个文件里面的版本名称

```
/usr/local/nginx/sbin/nginx -V #查看nginx版本号
```

[回目录](#)

nginx-优化

目录

- 一、Nginx的优化
 - 1、编译安装前优化
 - i. 隐藏软件名称和版本号
 - ii. 修改 HTTP 头信息中的 connection 字段，防止回显具体版本号
 - iii. 定义了 http 错误码的返回
 - 2、安装 nginx
 - i. 相关选项说明
 - ii. 测试是否隐藏了版本和软件名
 - 3、nginx配置项优化
 - (1) Nginx运行工作进程个数，一般我们设置CPU的核心或者核心数×2 ~ i. Nginx运行 CPU亲和力 ~ ii. Nginx最多可以打开文件数
 - (2) Nginx事件处理模型
 - (3) 开启高效传输模式
 - (4) 连接超时时间
 - (5) fastcgi 调优 ~ 总结
 - (6) gzip 调优
 - (7) expires 缓存调优 ~ i. expire 功能优点 ~ ii. expire 功能缺点
 - (8) 防盗链
 - (9) 内核参数优化
 - (10) 关于系统连接数的优化
- 二、部署 LNMP
 - 1、安装 php
 - (1) 解决依赖关系 ~ 安装 libmcrypt
 - (2) 编译安装 php
 - (3) 提供 php 配置文件
 - (4) 为 php-fpm 提供脚本
 - (5) 提供 php-fpm 配置文件并编辑 ~ i. 修改内容如下 ~ ii. 启动 php-fpm 服务 ~ iii. 在 nginx.conf 文件的 server 中添加下面内容支持 php ~ iv. 重载 nginx 服

务

● 三、验证、压力测试

- (1) 验证防盗链
- (2) 验证 gzip 功能
- (3) 压力测试
 - i. 安装 httpd-tools 软件包
 - ii. 第二次压力测试，比较两次的差异
- (5) xcache 加速 php
 - 1) 安装 xcache
 - 2) 创建 xcache 缓存文件
 - 3) 拷贝 xcache 后台管理程序到网站根目录
 - 4) 配置 php 支持 xcache ~ i. 测试 ~ ii. 浏览器打开网站根目录下面的 xcache ~ iii. 测试对 php 动态页面的压力测试

Nginx是俄罗斯人编写的十分轻量级的HTTP服务器,Nginx，它的发音为 engine X ，是一个高性能的HTTP和反向代理服务器，同时也是一个 IMAP/POP3/SMTP 代理服务器。Nginx是由俄罗斯人 Igor Sysoev 为俄罗斯访问量第二的 Rambler.ru 站点开发。

Nginx以事件驱动（ epoll ）的方式编写，所以有非常好的性能，同时也是一个非常高效的反向代理、负载平衡。但是Nginx并不支持 cgi 方式运行，原因是可以减少因此带来的一些程序上的漏洞。所以必须使用 FastCGI 方式来执行 PHP 程序。

由于Nginx本身的一些优点，轻量，开源，易用，越来越多的公司使用 nginx 作为自己公司的 web 应用服务器，本文详细介绍 nginx 源码安装的同时并对 nginx 进行优化配置。

一、Nginx的优化

[回目录](#)

1、编译安装前优化

[回目录](#)

编译前的优化主要是用来修改程序名等等，目的更改源码隐藏软件名称和版本号

安装 `zlib-devel` 、 `pcre-devel` 等依赖包

```
[root@www ~]# yum -y install gcc gcc-c++ make libtool zlib zlib-devel pcre pcre-devel openssl openssl-devel
```

下载nginx的源码包：<http://nginx.org/download>

解压源码包：

```
[root@www ~]# tar zxf nginx-1.10.2.tar.gz
[root@www ~]# cd nginx-1.10.2/
```

i. 隐藏软件名称和版本号

[回目录](#)

```
[root@www nginx-1.10.2]# vim src/core/nginx.h
//此行修改的是你想要的版本
#define NGINX_VERSION      "1.10.2"          //第13行
//此行修改的是你想修改的软件名称
#define NGINX_VER           "nginx/" NGINX_VERSION //第14行
修改上面的信息，即可更改nginx显示版本。例如：
#define NGINX_VERSION      "7.0"
#define NGINX_VER           "IIS/" NGINX_VERSION
```

ii. 修改 HTTP 头信息中的 `connection` 字段，防止回显具体版本号

[回目录](#)

拓展：通用 `http` 头，通用头包含请求和响应消息都支持的头，通用头包含 `Cache-Control` 、 `Connection` 、 `Date` 、 `Pragma` 、 `Transfer-Encoding` 、 `Upgrade` 、 `Via` 。对通用头的扩展要求通讯双方都支持此扩展，如

果存在不支持的通用头，一般将会作为实体头处理。那么也就是说有部分设备，或者是软件，能获取到 connection ，部分不能，要隐藏就要彻底！

```
[root@www nginx-1.10.2]# vi src/http/ngx_http_header_filter_modu  
le.c
```

修改前：

```
static char ngx_http_server_string[] = "Server: nginx" CRLF; //  
第49行
```

修改后：

```
static char ngx_http_server_string[] = "Server: IIS" CRLF;
```

iii. 定义了 http 错误码的返回

[回目录](#)

有时候我们页面程序出现错误， Nginx 会代我们返回相应的错误代码，回显的时候，会带上 nginx 和版本号，我们把他隐藏起来

```
[root@www nginx-1.10.2]# vi src/http/ngx_http_special_response.c
```

修改前

```
static u_char ngx_http_error_tail[] =      //第29行  
"<hr><center>nginx</center>" CRLF  
"</body>" CRLF  
"</html>" CRLF  
;
```

修改后

```
static u_char ngx_http_error_tail[] =
"<hr><center>IIS</center>" CRLF
"</body>" CRLF
"</html>" CRLF
;
```

2、安装 nginx

回目录

```
[root@www ~]# groupadd www      #添加www组
[root@www ~]# useradd -g www www -s /sbin/nologin      #创建nginx运行账户www并加入到www组，不允许www用户直接登录系统
[root@www nginx-1.10.2]# ./configure --prefix=/usr/local/nginx1.10 --with-http_dav_module --with-http_stub_status_module --with-http_addition_module --with-http_sub_module --with-http_flv_module --with-http_mp4_module --with-pcre --with-http_ssl_module --with-http_gzip_static_module
--user=www --group=www
[root@www nginx-1.10.2]# make && make install
```

i. 相关选项说明

回目录

- `--with-http_dav_module` #增加 PUT , DELETE , MKCOL : 创建集合， COPY 和 MOVE 方法
- `--with-http_stub_status_module` #获取Nginx的状态统计信息
- `--with-http_addition_module` #作为一个输出过滤器，支持不完全缓冲，分部分相应请求
- `--with-http_sub_module` #允许一些其他文本替换Nginx相应中的一些文本
- `--with-http_flv_module` #提供支持flv视频文件支持
- `--with-http_mp4_module` #提供支持mp4视频文件支持，提供伪流媒体服务端支持
- `--with-http_ssl_module` #启用 `ngx_http_ssl_module`

如果pcre是通过编译安装的话，例如

```
# tar zxvf /usr/local/src/pcre-8.36.tar.gz -C /usr/local/src/
# cd /usr/local/src/pcre-8.36
# ./configure && make && make install
```

则 `--with-pcre=/usr/local/src/pcre-8.36` #需要注意，这里指的是源码，
用 `./configure --help | grep pcre` 查看帮助

```
[root@www nginx-1.10.2]# ln -s /usr/local/nginx1.10/sbin/nginx /usr/local/sbin/
[root@www nginx-1.10.2]# nginx -t
启动nginx
[root@www nginx-1.10.2]# nginx
[root@www nginx-1.10.2]# netstat -anpt | grep nginx
tcp      0      0 0.0.0.0:80          0.0.0.0:*        LISTEN      9834/nginx:
master
```

ii. 测试是否隐藏了版本和软件名

[回目录](#)

```
[root@www ~]# curl -I http://127.0.0.1
HTTP/1.1 200 OK
Server: IIS/7.0
Date: Sat, 05 Nov 2016 14:38:21 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Sat, 05 Nov 2016 14:19:47 GMT
Connection: keep-alive
ETag: "581dea83-264"
Accept-Ranges: bytes
[root@www ~]# nginx -h
nginx version: IIS/7.0
Usage: nginx [-?hvVtTq] [-s signal] [-c filename] [-p prefix] [-g directives]
Options:
  -v          : show version and exit
  -V          : show version and configure options then exit
  -t          : test configuration and exit
  -T          : test configuration, dump it and exit
  -q          : suppress non-error messages during configuration
on testing
  -s signal    : send signal to a master process: stop, quit, r
eopen, reload
  -p prefix    : set prefix path (default: /usr/local/nginx1.10
/)
  -c filename   : set configuration file (default: conf/nginx.co
nf)
  -g directives : set global directives out of configuration fil
e
```

3、nginx配置项优化

[回目录](#)

```
[root@www ~]# ps -ef | grep nginx
root      9834      1  0 22:36 ?          00:00:00 nginx: master
process
www       9953    9834  0 22:43 ?          00:00:00 nginx: worker
process
```

在这里我们还可以看到在查看的时候，`work` 进程是 `nginx` 程序用户，但是 `master` 进程还是 `root`，其中，`master` 是监控进程，也叫主进程，`work` 是工作进程，部分还有 `cache` 相关进程，关系如图：

可以直接理解为 `master` 是管理员，`work` 进程才是为用户提供服务的！

(1)Nginx运行工作进程个数，一般我们设置**CPU**的核心或者核心数x2

[回目录](#)

如果不了解cpu的核数，可以top命令之后按1也可以看出来，也可以查

```
看 /proc/cpuinfo文件 #grep ^processor /proc/cpuinfo | wc -l
```

```
[root@www ~]# vi /usr/local/nginx1.10/conf/nginx.conf
worker_processes 4;
[root@www ~]# /usr/local/nginx1.10/sbin/nginx -s reload
[root@www ~]# ps -aux | grep nginx | grep -v grep
root  9834  0.0  0.0  47556  1948 ?          Ss   22:36   0:00 nginx: master process nginx
www   10135  0.0  0.0  50088  2004 ?          S    22:58   0:00 nginx: worker process
www   10136  0.0  0.0  50088  2004 ?          S    22:58   0:00 nginx: worker process
www   10137  0.0  0.0  50088  2004 ?          S    22:58   0:00 nginx: worker process
www   10138  0.0  0.0  50088  2004 ?          S    22:58   0:00 nginx: worker process
```

i.Nginx运行 **CPU**亲和力

[回目录](#)

比如 4 核配置

```
worker_processes 4;
worker_cpu_affinity 0001 0010 0100 1000
```

比如 8 核配置

```
worker_processes 8;
worker_cpu_affinity 00000001 00000010 00000100 00001000 00010000
00100000 01000000 10000000;
```

`worker_processes` 最多开启 8 个， 8 个以上性能提升不会再提升了，而且稳定性变得更低，所以8个进程够用了。

ii.Nginx最多可以打开文件数

[回目录](#)

```
worker_rlimit_nofile 65535;
```

这个指令是指当一个nginx进程打开的最多文件描述符数目，理论值应该是最多打开文件数（`ulimit -n`）与nginx进程数相除，但是nginx分配请求并不是那么均匀，所以最好与 `ulimit -n` 的值保持一致。

注：文件资源限制的配置可以在 `/etc/security/limits.conf` 设置，针对 root/user等各个用户或者*代表所有用户来设置。

```
*      soft    nofile    65535
*      hard    nofile    65535
```

用户重新登录生效（`ulimit -n`）

(2)Nginx事件处理模型

[回目录](#)

```

events {
    use epoll;
    worker_connections 65535;
    multi_accept on;
}

```

nginx 采用 epoll 事件模型，处理效率高 worker_connections 是单个worker进程允许客户端最大连接数，这个数值一般根据服务器性能和内存来制定，实际最大值就是worker进程数乘以 work_connections 实际我们填入一个65535，足够了，这些都算并发值，一个网站的并发达到这么大的数量，也算一个大站了！

multi_accept 告诉nginx收到一个新连接通知后接受尽可能多的连接

(3)开启高效传输模式

[回目录](#)

```

http {
    include mime.types;
    default_type application/octet-stream;
    .....
    sendfile on;
    tcp_nopush on;
    .....
}

```

- `include mime.types;` //媒体类型， `include` 只是一个在当前文件中包含另一个文件内容的指令
- `default_type application/octet-stream;` //默认媒体类型足够
- `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，对于普通应用设为 `on`，如果用来进行下载等应用磁盘 I/O 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。

注意：如果图片显示不正常把这个改成 `off`。

- `tcp_nopush on;` 必须在 `sendfile` 开启模式才有效，防止网路阻塞，积极的减少网络报文段的数量（告诉 nginx 在一个数据包里发送所有头文件，

而不一个接一个的发送。)

(4)连接超时时间

[回目录](#)

主要目的是保护服务器资源，CPU，内存，控制连接数，因为建立连接也是需要消耗资源的

```
keepalive_timeout 60;
tcp_nodelay on;
client_header_buffer_size 4k;
open_file_cache max=102400 inactive=20s;
open_file_cache_valid 30s;
open_file_cache_min_uses 1;
client_header_timeout 15;
client_body_timeout 15;
reset_timedout_connection on;
send_timeout 15;
server_tokens off;
client_max_body_size 10m;
```

- `keepalive_timeout` 客户端连接保持会话超时时间，超过这个时间，服务器断开这个链接
- `tcp_nodelay`； 也是防止网络阻塞，不过要包涵在`keepalive`参数才有效
- `client_header_buffer_size 4k`； 客户端请求头部的缓冲区大小，这个可以根据你的系统分页大小来设置，一般一个请求头的大小不会超过 1k，不过由于一般系统分页都要大于1k，所以这里设置为分页大小。分页大小可以用命令 `getconf PAGESIZE` 取得。
- `open_file_cache max=102400 inactive=20s`； 这个将为打开文件指定缓存，默认是没有启用的，`max`指定缓存数量，建议和打开文件 数一致，`inactive` 是指经过多长时间文件没被请求后删除缓存。
- `open_file_cache_valid 30s`； 这个是指多长时间检查一次缓存的有效信息。
- `open_file_cache_min_uses 1`； Sets the minimum number of file accesses during the period configured by the `inactive` parameter of the `open_file_cache` directive, required for a file descriptor to remain open in the

`cache`.一般是热点数据。

- `open_file_cache` 指令中的`inactive`参数时间内文件的最少使用次数，如果超过这个数字，文件描述符一直是在缓存中打开的，如上例，如果有一个文件在`inactive`时间内一次没被使用，它将被移除。
- `client_header_timeout` 设置请求头的超时时间。我们也可以把这个设置低些，如果超过这个时间没有发送任何数据，nginx将返回request time out的错误
- `client_body_timeout` 设置请求体的超时时间。我们也可以把这个设置低些，超过这个时间没有发送任何数据，和上面一样的错误提示
- `reset_timeout_connection` 告诉nginx关闭不响应的客户端连接。这将会释放那个客户端所占有的内存空间。
- `send_timeout` 响应客户端超时时间，这个超时时间仅限于两个活动之间的时间，如果超过这个时间，客户端没有任何活动，nginx关闭连接
- `server_tokens` 并不会让nginx执行的速度更快，但它可以关闭在错误页面中的nginx版本数字，这样对于安全性是有好处的。
- `client_max_body_size` 上传文件大小限制

(5) fastcgi 调优

[回目录](#)

```
fastcgi_connect_timeout      600;
fastcgi_send_timeout 600;
fastcgi_read_timeout 600;
fastcgi_buffer_size 64k;
fastcgi_buffers 4 64k;
fastcgi_busy_buffers_size 128k;
fastcgi_temp_file_write_size 128k;
fastcgi_temp_path /usr/local/nginx1.10/nginx_tmp;
fastcgi_intercept_errors on;
fastcgi_cache_path /usr/local/nginx1.10/fastcgi_cache levels=1:2
    keys_zone=cache_fastcgi:128m inactive=1d max_size=10g;
```

- Cache：写入缓存区
- Buffer：读取缓存区
- Fastcgi 是静态服务和动态服务的一个接口

- `fastcgi_connect_timeout 600;` #指定连接到后端FastCGI的超时时间。
- `fastcgi_send_timeout 600;` #向FastCGI传送请求的超时时间。
- `fastcgi_read_timeout 600;` #指定接收FastCGI应答的超时时间。
- `fastcgi_buffer_size 64k;` 指定读取FastCGI应答第一部分需要用多大的缓冲区，默认的缓冲区大小为 `fastcgi_buffers` 指令中的每块大小，可以将这个值设置更小。
- `fastcgi_buffers 4 64k;` 指定本地需要用多少和多大的缓冲区来缓冲 FastCGI 的应答请求，如果一个php脚本所产生的页面大小为256KB，那么会分配4个64KB的缓冲区来缓存，如果页面大小大于256KB，那么大于256KB的部分会缓存到 `fastcgi_temp_path` 指定的路径中，但是这并不是好方法，因为内存中的数据处理速度要快于磁盘。一般这个值应该为站点中php脚本所产生的页面大小的 **中间值**，如果站点大部分脚本所产生的页面大小为256KB，那么可以把这个值设置为“8 32K”、“4 64K”等。
- `fastcgi_busy_buffers_size 128k;` #建议设置为 `fastcgi_buffers` 的两倍，繁忙时候的buffer
- `fastcgi_temp_file_write_size 128k;` #在写入 `fastcgi_temp_path` 时将用多大的数据块，默认值是 `fastcgi_buffers` 的两倍，该数值设置小时若负载上来时可能报 502 Bad Gateway
- `fastcgi_temp_path` #缓存临时目录
- `fastcgi_intercept_errors on;` #这个指令指定是否传递 4xx 和 5xx 错误信息到客户端，或者允许nginx使用 `error_page` 处理错误信息。

注：静态文件不存在会返回 404 页面，但是php页面则返回空白页！！

- `fastcgi_cache_path /usr/local/nginx1.10/fastcgi_cache levels=1:2 keys_zone=cache_fastcgi:128m inactive=1d max_size=10g;`
 - `keys_zone=cache_fastcgi` 后面的 `cache_fastcgi` 为名字，是自定义的

`fastcgi_cache` 缓存目录，可以设置目录层级，比如 `1:2` 会生成 `16*256` (`1:2`的含义即16的一次方乘以16的二次方)个子目录，`cache_fastcgi` 是这个缓存空间的名字，`cache`是用多少内存（这样热门的内容nginx直接放内存，提高访问速度），`inactive`表示默认失效时间，如果缓存数据在失效时间内没有被访问，将被删除，`max_size`表示最多用多少硬盘空间。

- `fastcgi_cache cache_fastcgi;` 表示开启 FastCGI 缓存并为其指定一个名称。开启缓存非常有用，可以有效降低CPU的负载，并且防止 502 的错误发生。`cache_fastcgi` 为 `proxy_cache_path` 指令创建的缓存区名称
- `fastcgi_cache_valid 200 302 1h;` #用来指定应答代码的缓存时间，实例中的值表示将200和302应答缓存一小时，要和 `fastcgi_cache` 配合使用
- `fastcgi_cache_valid 301 1d;` #将301应答缓存一天
- `fastcgi_cache_valid any 1m;` #将其他应答缓存为1分钟
- `fastcgi_cache_min_uses 1;` #该指令用于设置经过多少次请求的相同URL将被缓存。
- `fastcgi_cache_key http://$host$request_uri;` #该指令用来设置web缓存的Key值,nginx根据Key值md5哈希存储.一般根据 \$host(域名) 、 \$request_uri(请求的路径) 等变量组合成 `proxy_cache_key` 。
- `fastcgi_pass` #指定 FastCGI 服务器监听端口与地址，可以是本机或者其它

总结

回目录

- nginx 的缓存功能有：`proxy_cache / fastcgi_cache`
- `proxy_cache` 的作用是缓存后端服务器的内容，可能是任何内容，包括静态的和动态的。
- `fastcgi_cache` 的作用是缓存fastcgi生成的内容，很多情况是php生成的动态的内容。
- `proxy_cache` 缓存减少了nginx与后端通信的次数，节省了传输时间和后端宽带。
- `fastcgi_cache` 缓存减少了nginx与php的通信的次数，更减轻了php和数据库(mysql)的压力。

(6) gzip 调优

回目录

使用 `gzip` 压缩功能，可能为我们节约带宽，加快传输速度，有更好的体验，也为我们节约成本，所以说这是一个重点。

Nginx 启用压缩功能需要你来 `ngx_http_gzip_module` 模块， apache 使用的是 `mod_deflate` 一般我们需要压缩的内容有：文本， js ， html ， css ，对于图片，视频，flash什么的不压缩，同时也要注意，我们使用gzip的功能是需要消耗CPU的！

```
gzip on;
gzip_min_length 2k;
gzip_buffers     4 32k;
gzip_http_version 1.1;
gzip_comp_level 6;
gzip_types text/plain text/css text/javascript application/json
application/javascript application/x-javascript application/xml;
gzip_vary on;
gzip_proxied any;
```

- `gzip on;` #开启压缩功能
- `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头的Content-Length中获取，默认值是0，不管页面多大都进行压缩，建议设置成大于1K，如果小与1K可能会越压越大。
- `gzip_buffers 4 32k;` #压缩缓冲区大小，表示申请 4 个单位为 32K 的内存作为压缩结果流缓存，默认值是申请与原始数据大小相同的内存空间来存储 gzip 压缩结果。
- `gzip_http_version 1.1;` #压缩版本，用于设置识别HTTP协议版本，默认是1.1，目前大部分浏览器已经支持GZIP解压，使用默认即可
- `gzip_comp_level 6;` #压缩比例，用来指定 GZIP 压缩比， 1 压缩比最小，处理速度最快， 9 压缩比最大，传输速度快，但是处理慢，也比较消耗CPU资源。
- `gzip_types text/css text/xml application/javascript;` #用来指定压缩的类型， text/html 类型总是会被压缩。

默认值: `gzip_types text/html` (默认不对 js/css 文件进行压缩)

```
# 压缩类型，匹配MIME类型进行压缩
# 不能用通配符 text/*
# (无论是否指定)text/html默认已经压缩
# 设置哪压缩种文本文件可参考 conf/mime.types
```

`gzip_vary on; # vary header` 支持，改选项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

(7) `expires` 缓存调优

[回目录](#)

缓存，主要针对于图片，`css`，`js`等元素更改机会比较少的情况下使用，特别是图片，占用带宽大，我们完全可以设置图片在浏览器本地缓存`365d`，`css`，`js`，`html`可以缓存个10来天，这样用户第一次打开加载慢一点，第二次，就非常快了！缓存的时候，我们需要将需要缓存的拓展名列出来，`expires` 缓存配置在`server` 字段里面

```
location ~* \.(ico|jpe?g|gif|png|bmp|swf|flv)$ {  
    expires 30d;  
    # log_not_found off;  
    access_log off;  
}  
  
location ~* \.(js|css)$ {  
    expires 7d;  
    log_not_found off;  
    access_log off;  
}
```

- `~` 区分大小写匹配
- `~*` 不区分大小写匹配

注：`log_not_found off;` 是否在`error_log` 中记录不存在的错误。默认是。

i. `expire` 功能优点

[回目录](#)

- (1) `expires` 可以降低网站购买的带宽，节约成本
- (2) 同时提升用户访问体验
- (3) 减轻服务的压力，节约服务器成本，是web服务非常重要的功能。

ii. `expire` 功能缺点

[回目录](#)

被缓存的页面或数据更新了，用户看到的可能还是旧的内容，反而影响用户体验。

解决办法：

- 第一个缩短缓存时间，例如：1天，但不彻底，除非更新频率大于1天；
- 第二个对缓存的对象改名。

网站不希望被缓存的内容

- 1) 网站流量统计工具
- 2) 更新频繁的文件（google 的 logo）

(8) 防盗链

[回目录](#)

防止别人直接从你网站引用图片等链接，消耗了你的资源和网络流量，那么我们的解决办法由几种：

1. 水印，品牌宣传，你的带宽，服务器足够
2. 防火墙，直接控制，前提是你知道IP来源
3. 防盗链策略 下面的方法是直接给予 404 的错误提示

```
location ~* ^.+\.(jpg|gif|png|swf|flv|wma|wmv|ASF|mp3|mmf|zip|rar)$ {
    valid_referers none blocked www.benet.com benet.com;
    if ($invalid_referer) {
        # return 302 http://www.benet.com/img/nolink.jpg;
        return 404;
        break;
    }
    access_log off;
}
```

- 这段代码要放在 `server` 中的最前面，这样就是第一个匹配。
- 加一个 !

- location !~* ^.+\.
(jpg|gif|png|swf|flv|wma|wmv|ASF|mp3|mmf|zip|rar)\$ 这个时候是对匹配的否定，而不是对 区分大小写 的否定。

(9) 内核参数优化

[回目录](#)

- `fs.file-max = 999999` : 这个参数表示进程（比如一个 worker 进程）可以同时打开的最大句柄数，这个参数直线限制最大并发连接数，需根据实际情况配置。
- `net.ipv4.tcp_max_tw_buckets = 6000` 这个参数表示操作系统允许 `TIME_WAIT` 套接字数量的最大值，如果超过这个数字，`TIME_WAIT` 套接字将立刻被清除并打印警告信息。该参数默认为 `180000`，过多的 `TIME_WAIT` 套接字会使Web服务器变慢。注：**主动关闭连接**的服务端会产生 `TIME_WAIT` 状态的连接
- `net.ipv4.ip_local_port_range = 1024 65000` #允许系统打开的端口范围。
- `net.ipv4.tcp_tw_recycle = 1` #启用 `timewait` 快速回收。
- `net.ipv4.tcp_tw_reuse = 1` #开启重用。允许将 `TIME-WAIT sockets` 重新用于新的 TCP 连接。这对于服务器来说很有意义，因为服务器上总会有大量 `TIME-WAIT` 状态的连接。
- `net.ipv4.tcp_keepalive_time = 30` : 这个参数表示当 `keepalive` 启用时，`TCP` 发送 `keepalive` 消息的频度。默认是 `2小时`，若将其设置的小一些，可以更快地清理无效的连接。
- `net.ipv4.tcp_syncookies = 1` #开启 `SYN Cookies`，当出现 `SYN` 等待队列溢出时，启用 `cookies` 来处理。
- `net.core.somaxconn = 40960` #web 应用中 `listen` 函数的 `backlog` 默认会给我们内核参数的 `net.core.somaxconn` 限制到128，而nginx定义的 `NGX_LISTEN_BACKLOG` 默认为511，所以有必要调整这个值。

注：对于一个 `TCP` 连接，`Server`与`Client`需要通过三次握手来建立网络连接.当三次握手成功后,我们可以看到端口的状态由`LISTEN`转变为`ESTABLISHED`,接着这条链路上就可以开始传送数据了.每一个处于监听(`Listen`)状态的端口,都有自己的监听队列.监听队列的长度与如`somaxconn`参数和使用该端口的程序中`listen()`函数有关

- `somaxconn`参数 :定义了系统中每一个端口最大的监听队列的长度,这是个全局的参数,默认值为128,对于一个经常处理新连接的高负载 web服务环境来说,默认的128太小了。大多数环境这个值建议增加到1024或者更多。大的侦听队列对防止拒绝服务 DoS 攻击也会有所帮助。
- `net.core.netdev_max_backlog = 262144` #每个网络接口接收数据包的速度比内核处理这些包的速率快时,允许送到队列的数据包的最大数目。
- `net.ipv4.tcp_max_syn_backlog = 262144` #这个参数表示TCP三次握手建立阶段接受SYN请求队列的最大长度,默认为1024,将其设置得大一些可以使出现Nginx繁忙来不及accept新连接的情况时,Linux不至于丢失客户端发起的连接请求。
- `net.ipv4.tcp_rmem = 10240 87380 12582912` #这个参数定义了TCP接受缓存(用于TCP接受滑动窗口)的最小值、默认值、最大值。
- `net.ipv4.tcp_wmem = 10240 87380 12582912`: 这个参数定义了TCP发送缓存(用于TCP发送滑动窗口)的最小值、默认值、最大值。
- `net.core.rmem_default = 6291456` :这个参数表示内核套接字接受缓存区默认的大小。`net.core.wmem_default = 6291456` :这个参数表示内核套接字发送缓存区默认的大小。
- `net.core.rmem_max = 12582912` :这个参数表示内核套接字接受缓存区的最大大小。
- `net.core.wmem_max = 12582912` :这个参数表示内核套接字发送缓存区的最大大小。
- `net.ipv4.tcp_syncookies = 1` :该参数与性能无关,用于解决TCP的SYN攻击。

下面贴一个完整的内核优化设置：

```
fs.file-max = 999999
net.ipv4.ip_forward = 0
net.ipv4.conf.default.rp_filter = 1
net.ipv4.conf.default.accept_source_route = 0
kernel.sysrq = 0
kernel.core_uses_pid = 1
net.ipv4.tcp_syncookies = 1
kernel.msgmnb = 65536
kernel.msgmax = 65536
kernel.shmmax = 68719476736
kernel.shmall = 4294967296
net.ipv4.tcp_max_tw_buckets = 6000
net.ipv4.tcp_sack = 1
net.ipv4.tcp_window_scaling = 1
net.ipv4.tcp_rmem = 10240 87380 12582912
net.ipv4.tcp_wmem = 10240 87380 12582912
net.core.wmem_default = 8388608
net.core.rmem_default = 8388608
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
net.core.netdev_max_backlog = 262144
net.core.somaxconn = 40960
net.ipv4.tcp_max_orphans = 3276800
net.ipv4.tcp_max_syn_backlog = 262144
net.ipv4.tcp_timestamps = 0
net.ipv4.tcp_synack_retries = 1
net.ipv4.tcp_syn_retries = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_mem = 94500000 915000000 927000000
net.ipv4.tcp_fin_timeout = 1
net.ipv4.tcp_keepalive_time = 30
net.ipv4.ip_local_port_range = 1024 65000
```

执行 `sysctl -p` 使内核修改生效

(10) 关于系统连接数的优化

[回目录](#)

linux 默认值 open files 为 1024

```
#ulimit -n  
1024
```

说明 server 只允许同时打开 1024 个文件 使用 ulimit -a 可以查看当前系统的所有限制值，使用 ulimit -n 可以查看当前的最大打开文件数。新装的linux 默认只有 1024 ，当作负载较大的服务器时，很容易遇到 error: too many open files 。因此，需要将其改大

在 /etc/security/limits.conf 最后增加：

```
* soft    nofile 65535  
* hard    nofile 65535  
* soft    noproc 65535  
* hard    noproc 65535
```

二、部署 LNMP

1、安装 php

(1)解决依赖关系

[回目录](#)

```
[root@www ~]# yum -y install libxml2-devel libcurl-devel openssl-devel bzip2-devel
```

安装 libmcrypt

```
[root@www ~]# tar zxf libmcrypt-2.5.7.tar.gz  
[root@wwwr ~]# cd libmcrypt-2.5.7/  
[root@www libmcrypt-2.5.7]# ./configure --prefix=/usr/local/libmcrypt && make && make install
```

(2) 编译安装 php

[回目录](#)

```
[root@www ~]# tar zxf php-5.6.27.tar.gz  
[root@www ~]# cd php-5.6.27/  
[root@www php-5.6.27]# ./configure --prefix=/usr/local/php5.6 --with-mysql=mysqlnd --with-pdo-mysql=mysqlnd --with-mysqli=mysqlnd --with-openssl --enable-fpm --enable-sockets --enable-sysvshm --enable-mbstring --with-freetype-dir --with-jpeg-dir --with-png-dir --with-zlib --with-libxml-dir=/usr --enable-xml --with-mhash --with-mcrypt=/usr/local/libmcrypt --with-config-file-path=/etc --with-config-file-scan-dir=/etc/php.d --with-bz2 --enable-maintainer-zts  
[root@www php-5.6.27]# make && make install
```

(3) 提供 php 配置文件

[回目录](#)

```
[root@www php-5.6.27]# cp php.ini-production /etc/php.ini
```

(4) 为 php-fpm 提供脚本

[回目录](#)

```
[root@www php-5.6.27]# cp sapi/fpm/init.d.php-fpm /etc/init.d/php-fpm
[root@www php-5.6.27]# chmod +x /etc/init.d/php-fpm
[root@www php-5.6.27]# chkconfig --add php-fpm
[root@www php-5.6.27]# chkconfig php-fpm on
```

(5) 提供 **php-fpm** 配置文件并编辑

[回目录](#)

```
# cp /usr/local/php5.6/etc/php-fpm.conf.default /usr/local/php5.6/etc/php-fpm.conf
[root@www ~]# vi /usr/local/php5.6/etc/php-fpm.conf
```

i. 修改内容如下

[回目录](#)

```
pid = run/php-fpm.pid
listen = 0.0.0.0:9000
pm.max_children = 50
pm.start_servers = 5
pm.min_spare_servers = 5
pm.max_spare_servers = 35
```

ii. 启动 **php-fpm** 服务

[回目录](#)

```
[root@www ~]# service php-fpm start
Starting php-fpm  done
[root@www ~]# netstat -anpt | grep php-fpm
tcp        0      0 0.0.0.0:9000          0.0.0.0:*
              LISTEN      25456/php-fpm: mast
[root@www ~]# firewall-cmd --permanent --add-port=9000/tcp
success
[root@www ~]# firewall-cmd --reload
Success
```

iii. 在 `nginx.conf` 文件的 `server` 中添加下面内容支持 `php`

[回目录](#)

```
location ~ .*\.(php|php5)?$ {
    root html;
    fastcgi_pass 127.0.0.1:9000;
    fastcgi_index index.php;
    include fastcgi.conf;
    fastcgi_cache cache_fastcgi;
    fastcgi_cache_valid 200 302 1h;
    fastcgi_cache_valid 301 1d;
    fastcgi_cache_valid any 1m;
    fastcgi_cache_min_uses 1;
    fastcgi_cache_use_stale error timeout invalid_header
http_500;
    fastcgi_cache_key http://$host$request_uri;
}
```

iii. `nginx.conf` 的一个完整配置文件

[回目录](#)

```
user www www;
worker_processes 4;
worker_cpu_affinity 0001 0010 0100 1000;
error_log logs/error.log;
#error_log logs/error.log notice;
```

```
#error_log  logs/error.log  info;

pid        logs/nginx.pid;

events {
    use epoll;
    worker_connections  65535;
    multi_accept on;
}

http {
    include      mime.types;
    default_type application/octet-stream;

    #log_format  main  '$remote_addr - $remote_user [$time_local]
] "$request" '
    #                                '$status $body_bytes_sent "$http_referer"
'
    #                                '"$http_user_agent" "$http_x_forwarded_for
r"';;

#access_log  logs/access.log  main;

    sendfile        on;
    tcp_nopush      on;
    keepalive_timeout  65;
    tcp_nodelay     on;
    client_header_buffer_size 4k;
    open_file_cache max=102400 inactive=20s;
    open_file_cache_valid 30s;
    open_file_cache_min_uses 1;
    client_header_timeout 15;
    client_body_timeout 15;
    reset_timedout_connection on;
    send_timeout 15;
    server_tokens off;
    client_max_body_size 10m;
```

```

fastcgi_connect_timeout      600;
fastcgi_send_timeout        600;
fastcgi_read_timeout        600;
fastcgi_buffer_size         64k;
fastcgi_buffers             4 64k;
fastcgi_busy_buffers_size   128k;
fastcgi_temp_file_write_size 128k;
fastcgi_temp_path           /usr/local/nginx1.10/nginx_tmp;
fastcgi_intercept_errors    on;
fastcgi_cache_path          /usr/local/nginx1.10/fastcgi_cache levels
=1:2 keys_zone=cache_fastcgi:128m inactive=1d max_size=10g;

gzip on;
gzip_min_length 2k;
gzip_buffers     4 32k;
gzip_http_version 1.1;
gzip_comp_level 6;
gzip_types       text/plain text/css text/javascript application/
json application/javascript application/x-javascript application
/xml;
gzip_vary on;
gzip_proxied any;
server {
    listen      80;
    server_name www.benet.com;

    #charset koi8-r;

    #access_log  logs/host.access.log  main;

    location ~* ^\.(jpg|gif|png|swf|flv|wma|wmv|asf|mp3|mm
f|zip|rar)$ {
        valid_referers none blocked www.benet.com benet.co
m;
        if ($invalid_referer) {
            #return 302 http://www.benet.com/img/nolink.jpg;

            return 404;
            break;
        }
    }
}

```

```
        access_log off;
    }
    location / {
        root    html;
        index   index.php index.html index.htm;
    }
    location ~* \.(ico|jpe?g|gif|png|bmp|swf|flv)$ {
        expires 30d;
        log_not_found off;
        access_log off;
    }

    location ~* \.(js|css)$ {
        expires 7d;
        log_not_found off;
        access_log off;
    }

    location = /(favicon.ico|robots.txt) {
        access_log off;
        log_not_found off;
    }
    location /status {
        stub_status on;
    }
    location ~ .*\.(php|php5)?$ {
        root html;
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_index index.php;
        include fastcgi.conf;
        fastcgi_cache cache_fastcgi;
        fastcgi_cache_valid 200 302 1h;
        fastcgi_cache_valid 301 1d;
        fastcgi_cache_valid any 1m;
        fastcgi_cache_min_uses 1;
        fastcgi_cache_use_stale error timeout invalid_header
http_500;
        fastcgi_cache_key http://$host$request_uri;
    }
    #error_page 404          /404.html;
```

```
# redirect server error pages to the static page /50x.html
error_page 500 502 503 504 /50x.html;
location = /50x.html {
    root   html;
}
}
```

iv. 重载 nginx 服务

[回目录](#)

```
[root@www ~]# /usr/local/nginx1.10/sbin/nginx -s reload
```

三、验证、压力测试

[回目录](#)

(1) 验证防盗链

[回目录](#)

使用apache做一个测试站点，域名为 `www.test.com`，在测试页上做一个超链接，链接 `nginx` 站点的一张图片

```
[root@centos1 ~]# cat /var/www/html/index.html
<a href="http://www.benet.com/11.gif">lianjie</a>
```

Nginx站点的网页目录结如下

```
tree /usr/local/nginx1.10/html/
/usr/local/nginx1.10/html/
├── 11.gif
├── 50x.html
├── img
│   └── nolink.jpg
└── index.html
└── test.php
```

在客户端浏览器中输入 `www.test.com` 点击页面链接

从上图可以看到防盗链设置生效了

(2)验证gzip功能

[回目录](#)

使用谷歌浏览器测试访问，如下图显示结果：（提示：在访问测试页之前按F12键）

用户访问 `test.php` 文件，在上图中 `content-encoding:gzip` 表明响应给用户的数据是压缩传输。

(3)压力测试

[回目录](#)

i.安装 `httpd-tools` 软件包

[回目录](#)

```
[root@www ~]# yum -y install httpd-tools
```

```
[root@www ~]# ab -c 500 -n 50000 http://www.benet.com/index.html
This is ApacheBench, Version 2.3 <$Revision: 1430300 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeust
```

```
ech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/  
  
Benchmarking www.benet.com (be patient)
Completed 5000 requests
Completed 10000 requests
Completed 15000 requests
Completed 20000 requests
Completed 25000 requests
Completed 30000 requests
Completed 35000 requests
Completed 40000 requests
Completed 45000 requests
Completed 50000 requests
Finished 50000 requests  
  
  
Server Software:           IIS
Server Hostname:          www.benet.com
Server Port:               80  
  
Document Path:            /index.html
Document Length:           612 bytes  
  
Concurrency Level:        500
Time taken for tests:     5.734 seconds
Complete requests:         50000
Failed requests:           0
Write errors:              0
Total transferred:         41800000 bytes
HTML transferred:          30600000 bytes
Requests per second:       8719.82 [#/sec] (mean)
Time per request:          57.341 [ms] (mean)
Time per request:          0.115 [ms] (mean, across all concurrent
                           requests)
Transfer rate:             7118.92 [Kbytes/sec] received  
  
Connection Times (ms)
                           min  mean[+/-sd] median   max
```

Connect:	1	25	4.2	25	38
Processing:	7	32	5.5	31	47
Waiting:	4	24	6.8	21	39
Total:	40	57	3.9	57	71

Percentage of the requests served within a certain time (ms)

50%	57
66%	59
75%	59
80%	60
90%	61
95%	62
98%	63
99%	64
100%	71 (longest request)

ii. 第二次压力测试，比较两次的差异

[回目录](#)

```
[root@www ~]# ab -c 1000 -n 100000 http://www.benet.com/index.html
This is ApacheBench, Version 2.3 <$Revision: 1430300 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking www.benet.com (be patient)
Completed 10000 requests
Completed 20000 requests
Completed 30000 requests
Completed 40000 requests
Completed 50000 requests
Completed 60000 requests
Completed 70000 requests
Completed 80000 requests
Completed 90000 requests
Completed 100000 requests
```

```
Finished 100000 requests
```

```

Server Software:           IIS
Server Hostname:          www.benet.com
Server Port:              80

Document Path:            /index.html
Document Length:          612 bytes

Concurrency Level:        1000
Time taken for tests:    12.010 seconds
Complete requests:        100000
Failed requests:          0
Write errors:             0
Total transferred:        83600000 bytes
HTML transferred:         61200000 bytes
Requests per second:      8326.49 [#/sec] (mean)
Time per request:         120.099 [ms] (mean)
Time per request:         0.120 [ms] (mean, across all concurrent
                           requests)
Transfer rate:            6797.80 [Kbytes/sec] received

```

Connection Times (ms)

	min	mean	[+/-sd]	median	max
Connect:	1	53	8.9	53	82
Processing:	17	67	11.4	66	98
Waiting:	0	49	14.3	43	84
Total:	70	119	6.5	120	140

Percentage of the requests served within a certain time (ms)

50%	120
66%	122
75%	123
80%	124
90%	126
95%	128
98%	129
99%	130
100%	140 (longest request)

(5) xcache 加速 php

[回目录](#)

1) 安装 xcache

[回目录](#)

```
wget http://xcache.lighttpd.net/pub/Releases/3.2.0/xcache-3.2.0.tar.gz #下载
```

```
[root@www ~]# tar zxf xcache-3.2.0.tar.gz      #解压  
[root@www ~]# cd xcache-3.2.0/      #进入安装目录  
[root@www xcache-3.2.0]# /usr/local/php5.6/bin/phpize #用phpize生成configure配置文件  
[root@www xcache-3.2.0]# ./configure --enable-xcache --enable-xcache-coverager --enable-xcache-optimizer --with-php-config=/usr/local/php5.6/bin/php-config #配置  
[root@www xcache-3.2.0]# make && make install #编译、安装
```

Installing shared extensions:

```
/usr/local/php5.6/lib/php/extensions/no-debug-zts-20131226/
```

安装完成之后，出现下面的界面，记住以下路径，后面会用到

```
/usr/local/php5.6/lib/php/extensions/no-debug-zts-20131226/
```

2) 创建 xcache 缓存文件

[回目录](#)

```
# touch /tmp/xcache  
# chmod 777 /tmp/xcache
```

3) 拷贝 **xcache** 后台管理程序到网站根目录

[回目录](#)

```
[root@www xcache-3.2.0]# cp -r htdocs/ /usr/local/nginx1.10/html  
/xcache
```

4) 配置 **php** 支持 **xcache**

[回目录](#)

```
vi /etc/php.ini #编辑配置文件，在最后一行添加以下内容
```

```
[xcache-common]
extension = /usr/local/php5.6/lib/php/extensions/no-debug-zts-20
131226/xcache.so
[xcache.admin]
xcache.admin.enable_auth = off
[xcache]
xcache.shm_scheme ="mmap"
xcache.size=60M
xcache.count =1
xcache.slots =8K
xcache.ttl=0
xcache.gc_interval =0
xcache.var_size=64M
xcache.var_count =1
xcache.var_slots =8K
xcache.var_ttl=0
xcache.var_maxttl=0
xcache.var_gc_interval =300
xcache.test =off
xcache.readonly_protection = Off
xcache.mmap_path ="/tmp/xcache"
xcache.coredump_directory =""
xcache.cacher =on
xcache.stat=on
xcache.optimizer =off
[xcache.coverager]
xcache.coverager =on
xcache.coveragedump_directory =""
```

i. 测试

[回目录](#)

```
service php-fpm restart #重启php-fpm
```

ii. 浏览器打开网站根目录下面的 **xcache**

[回目录](#)

http:// http://www.benet.com/xcache 可以看到如下页面：

iii. 测试对 php 动态页面的压力测试

[回目录](#)

```
[root@www ~]# ab -c 1000 -n 100000 http://www.benet.com/test.php
This is ApacheBench, Version 2.3 <$Revision: 1430300 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

```
Benchmarking www.benet.com (be patient)
Completed 10000 requests
Completed 20000 requests
Completed 30000 requests
Completed 40000 requests
Completed 50000 requests
Completed 60000 requests
Completed 70000 requests
Completed 80000 requests
Completed 90000 requests
Completed 100000 requests
Finished 100000 requests
```

```
Server Software:           IIS
Server Hostname:          www.benet.com
Server Port:               80
```

```
Document Path:            /test.php
Document Length:          85102 bytes
```

```
Concurrency Level:        1000
Time taken for tests:    13.686 seconds
Complete requests:        100000
Failed requests:          0
Write errors:             0
```

```
Total transferred:          8527900000 bytes
HTML transferred:          8510200000 bytes
Requests per second:       7306.71 [#/sec] (mean)
Time per request:          136.861 [ms] (mean)
Time per request:          0.137 [ms] (mean, across all concurrent
                           requests)
Transfer rate:             608504.46 [Kbytes/sec] received
```

Connection Times (ms)

	min	mean	[+/-sd]	median	max
Connect:	0	17	5.5	17	81
Processing:	21	119	10.8	121	140
Waiting:	1	17	6.7	16	68
Total:	50	136	8.1	137	151

Percentage of the requests served within a certain time (ms)

50%	137
66%	139
75%	140
80%	141
90%	143
95%	144
98%	146
99%	148
100%	151 (longest request)

[回目录](#)

varnish

目录

- I. varnish 原理
 - i. Varnish 简介
 - ii. Varnish 与 Squid 的对比
 - iii. Varnish 的劣势
 - iv. Varnish 劣势的解决方案
- II. 缓存原理
 - i. 使用 varnish 作为 web 代理缓存的原理
 - ii. 简单架构
 - iii. varnish 主要配置部分
 - iv. VCL 中内置预设变量:变量(也叫 object)
 - v. 特定功能性语句
 - vii. return 语句
 - viii. varnish 中内置于程序
 - IX. 优雅模式(Grace mode)
- III. 安装 varnish
 - i. 安装依赖关系的软件包(注:使用 centos 在线 yum 源)
 - ii. 安装 varnish
- IV. varnish 实例解析
 - i. 后端服务器地址池配置及后端服务器健康检查
 - 1. 后端服务器定义
 - 2. 监视器定义
 - 3. 集群负载均衡 directors
 - 4. 访问控制列表(ACL)
 - 5. 缓存规则配置
- V. varnish 完整配置实例
 - i. 拓扑环境
 - ii. vcl 文件配置内容
 - iii. 启动 varnish
 - 1. -a
 - 2. -f VCL-file or -b backend

- 3. 访问测试 varnish
 - 4. varnish4 配置手动清除缓存
-

I. varnish 原理

[回目录](#)

i. Varnish 简介

[回目录](#)

varnish 缓存是 web 应用加速器，同时也作为 http 反向缓存代理。你可以安装 varnish 在任何 http 的前端，同时配置它缓存内容。与传统的 squid 相比， varnish 具有性能更高、速度更快、管理更加方便等诸多优点。有一部分企业已经在生产环境中使用其作为旧版本的 squid 的替代方案，以在相同的服务器成本下提供更好的缓存效果， varnish 更是作为 CDN 缓存服务器的可选服务之一。根据官网的介绍， varnish 的主要特性如下：<https://www.varnish-cache.org/>

- 1. 缓存位置：可以使用内存也可以使用磁盘。如果要使用磁盘的话推荐 SSD 做 RAID1
- 2. 日志存储：日志也存储在内存中。存储策略：固定大小，循环使用
- 3. 支持虚拟内存的使用。
- 4. 有精确的时间管理机制，即缓存的时间属性控制。
- 5. 状态引擎架构：在不同的引擎上完成对不同的缓存和代理数据进行处理。可以通过特定的配置语言设计不同的控制语句，以决定数据在不同位置以不同方式缓存，在特定的地方对经过的报文进行特定规则的处理。
- 6. 缓存管理：以二叉堆格式管理缓存数据，做到数据的及时清理。

ii. Varnish 与 Squid 的对比

[回目录](#)

相同点

- 都是一个反向代理服务器；
- 都是开源软件；

Varnish 的优势

- 1、 Varnish 的稳定性很高，两者在完成相同负荷的工作时， Squid 服务器发生故障的几率要高于 Varnish ，因为使用 Squid 要经常重启；
- 2、 Varnish 访问速度更快，因为采用了“ Visual Page Cache ”技术，所有缓存数据都直接从内存读取，而 squid 是从硬盘读取，因而 Varnish 在访问速度方面会更快；
- 3、 Varnish 可以支持更多的并发连接，因为 Varnish 的 TCP 连接释放要比 Squid 快，因而在高并发连接情况下可以支持更多 TCP 连接；
- 4、 Varnish 可以通过管理端口，使用正则表达式批量的清除部分缓存，而 Squid 是做不到的； squid 属于是单进程使用单核 CPU ，但 Varnish 是通过 fork 形式打开多进程来做处理，所以可以合理的使用所有核来处理相应的请求；

iii. Varnish 的劣势

回目录

1、 varnish 进程一旦 Crash 或者重启，缓存数据都会从内存中完全释放，此时所有请求都会发送到后端服务器，在高并发情况下，会给后端服务器造成很大压力；

2、在 varnish 使用中如果单个 url 的请求通过 HA/F5 等负载均衡，则每次请求落在不同的 varnish 服务器中，造成请求都会被穿透到后端；而且同样的请求在多台服务器上缓存，也会造成 varnish 的缓存的资源浪费，造成性能下降；

iv. Varnish 劣势的解决方案

回目录

针对劣势一：在访问量很大的情况下推荐使用 varnish 的内存缓存方式启动，而且后面需要跟多台 squid/nginx 服务器。主要为了防止前面的 varnish 服务、服务器被重启的情况下，大量请求穿透 varnish ，这样 squid/nginx 可以就担当第二层 CACHE ，而且也弥补了 varnish 缓存在内存中重启都会释放的问题；

针对劣势二：可以在负载均衡上做 url 哈希，让单个 url 请求固定请求到一台 varnish 服务器上；

II. 缓存原理

[回目录](#)

i. 使用 varnish 作为 web 代理缓存的原理

[回目录](#)

varnish是一个http反向代理的缓存。它从客户端接收请求然后尝试从缓存中获取数据来响应客户端的请求，如果varnish不能从缓存中获得数据来响应客户端，它将转发请求到后端（backend servers），获取响应同时存储，最后交付给客户端。

如果varnish已经缓存了某个响应，它比你传统的后端服务器的响应要快很多，所以你需要尽可能多的请求直接从varnish的缓存中获取响应。

varnish决定是缓存内容或者是从后端服务器获取响应。后端服务器能通过http响应头中的Cache-Control来同步varnish缓存内容。在某些条件下varnish将不缓存内容，最常见的是使用cookie。当一个被标记有cookie的客户端web请求，varnish默认是不缓存。这些众多的varnish功能特点都是可以通过写vcl来改变的。

ii. 简单架构

[回目录](#)

Varnish 分为 management 进程和 child 进程；

- Management 进程：对子进程进行管理，同时对 VCL 配置进行编译，并应用到不同的状态引擎。
- Child 进程：生成线程池，负责对用户请求进行处理，并通过 hash 查找返回用户结果。

iii. varnish 主要配置部分

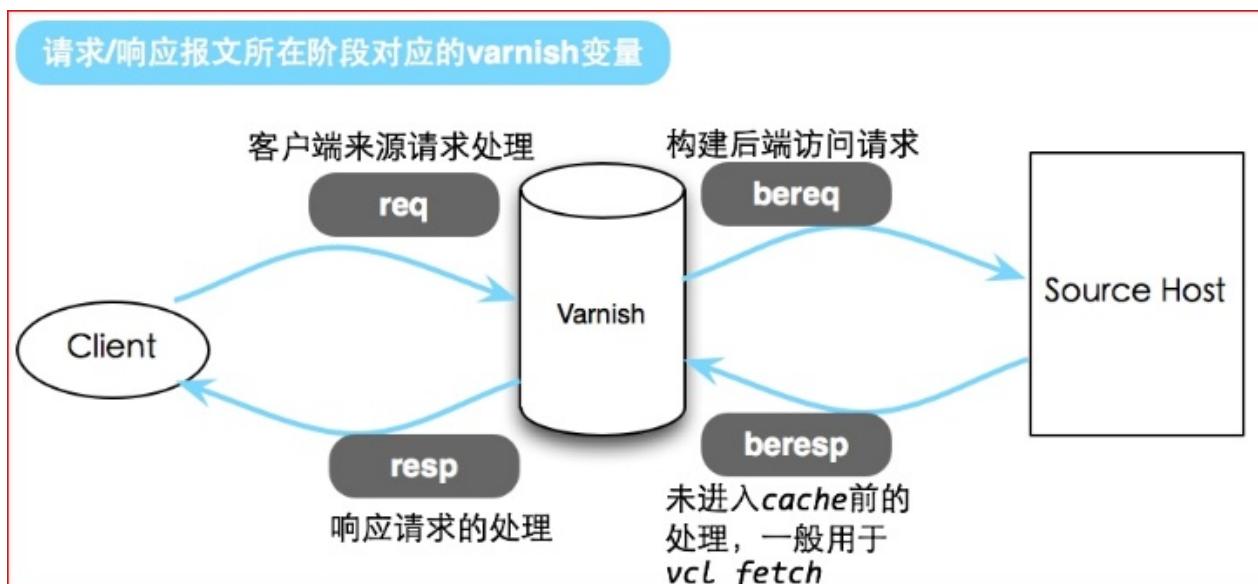
[回目录](#)

varnish 配置主要分为：

- 后端配置
- ACL 配置
- probes 配置
- directors 配置，核心子程序配置几大块。其中后端配置是必要的，在多台服务器中还会用到 directors 配置，核心子程序配置。
- 后端配置：即给varnish添加反代服务器节点，最少配置一个。
- ACL 配置：即给varnish添加访问控制列表，可以指定这些列表访问或禁止访问。
- probes 配置：即给varnish添加探测后端服务器是否正常的规则，方便切换或禁止对应后端服务器。
- directors 配置：即给varnish添加负载均衡模式管理多个后端服务器。核心子程序配置：即给varnish添加后端服务器切换，请求缓存，访问控制，错误处理等规则。

iv. VCL 中内置预设变量: 变量(也叫 object)

[回目录](#)



- **req** : The request object，请求到达时可用的变量(客户端发送的请求对象)
- **bereq** : The backend request object，向后端主机请求时可用的变量
- **beresp** : The backend response object，从后端主机获取内容时可用的变量(后端响应请求对象)
- **resp** : The HTTP response object，对客户端响应时可用的变量(返回给客户端的响应对象)
- **obj** : 存储在内存中时对象属性相关的可用的变量(高速缓存对象，缓存后端

响应请求内容)

预设变量是系统固定的，请求进入对应的 `vc1` 子程序后便生成，这些变量可以方便子程序提取，当然也可以自定义一些全局变量。

当前时间

`now` :作用：返回当前时间戳。

客户端：（客户端基本信息）

`client.ip` : 返回客户端IP地址。

注：原 `client.port` 已经弃用，如果要取客户端请求端口号使用
`std.port(client.ip)`，需要 `import std;` 才可以使用 `std`
`client.identity` : 用于装载客户端标识码。

服务器：（服务器基本信息）

注：原 `server.port` 已经弃用，如果要取服务器端口号使
用 `std.port(server.ip)`，需要 `import std;` 才可以使用 `std`

- `server.hostname` : 服务器主机名。
- `server.identity` : 服务器身份标识。
- `server.ip` : 返回服务器端IP地址。

`req`

- `req` : (客户端发送的请求对象)
- `req` : 整个HTTP请求数据结构

<code>req</code> 选项	作用
<code>req.backend_hint</code>	指定请求后端节点，设置后 <code>bereq.backend</code> 才能获取后端节点配置数据
<code>req.can_gzip</code>	客户端是否接受 GZIP 传输编码。
<code>req.hash_always_miss</code>	是否强制不命中高速缓存，如果设置为 <code>true</code> ，则高速缓存不会命中，一直会从后端获取新数据。
<code>req.hash_ignore_busy</code>	忽略缓存中忙碌的对象，多台缓存时可以避免死锁。
<code>req.http</code>	对应请求HTTP的header。
<code>req.method</code>	请求类型（如 <code>GET</code> ， <code>POST</code> ）。
<code>req.proto</code>	客户端使用的HTTP协议版本。
<code>req.restarts</code>	重新启动次数。默认最大值是4
<code>req.ttl</code>	缓存有剩余时间。
<code>req.url</code>	请求的 URL。
<code>req.xid</code>	唯一 ID。

`bereq`

- `bereq` : (发送到后端的请求对象，基于 `req` 对象)
- `bereq` : 整个后端请求后数据结构。

bereq 选项	解释
bereq.backend	所请求后端节点配置。
bereq.between_bytes_timeout	从后端每接收一个字节之间的等待时间（秒）。
bereq.connect_timeout	连接后端等待时间（秒），最大等待时间。
bereq.first_byte_timeout	等待后端第一个字节时间（秒），最大等待时间。
bereq.http	对应发送到后端 HTTP 的 header 信息。
bereq.method	发送到后端的请求类型（如： GET , POST ）。
bereq.proto	发送到后端的请求的 HTTP 版本。
bereq.retries	相同请求重试计数。
bereq.uncacheable	无缓存这个请求。
bereq.url	发送到后端请求的 URL 。
bereq.xid	请求唯一 ID 。

beresp

- beresp : (后端响应请求对象)
- beresp : 整个后端响应HTTP数据结构。

bereqsp 选项	解释
beresp.backend.ip	后端响应的 IP。
beresp.backend.name	响应后端配置节点的 name。
beresp.do_gunzip	默认为 false。缓存前解压该对象
beresp.do_gzip	默认为 false。缓存前压缩该对象
beresp.grace	设置当前对象缓存过期后可额外宽限时间，用于特殊请求加大缓存时间，当并发量巨大时，不易设置过大否则会堵塞缓存，一般可设置 1m 左右，当 beresp.ttl=0s 时该值无效。
beresp.http	对应的 HTTP 请求 header
beresp.keep	对象缓存后带保持时间
beresp.proto	响应的 HTTP 版本
beresp.reason	由服务器返回的 HTTP 状态信息
beresp.status	由服务器返回的状态码
beresp.storage_hint	指定保存的特定存储器
beresp.ttl	该对象缓存的剩余时间，指定统一缓存剩余时间。
beresp.uncacheable	继承 bereq.uncacheable，是否不缓存

OBJ

- **OBJ** : (高速缓存对象，缓存后端响应请求内容)

obj 选项	解释
obj.grace	该对象额外宽限时间
obj.hits	缓存命中次数，计数器从1开始，当对象缓存该值为 1，一般可以用于判断是否有缓存，当前该值大于 0 时则为有缓存。
obj.http	对应 HTTP 的 header
obj.proto	HTTP 版本
obj.reason	服务器返回的 HTTP 状态信息
obj.status	服务器返回的状态码
obj.ttl	该对象缓存剩余时间 (秒)
obj.uncacheable	不缓存对象

resp

- **resp** : (返回给客户端的响应对象)
- **resp** : 整个响应HTTP数据结构。

resp 选项	解释
resp.http	对应 HTTP 的 header 。
resp.proto	编辑响应的 HTTP 协议版本。
resp.reason	将要返回的 HTTP 状态信息。
resq.status	将要返回的 HTTP 状态码。

存储

- **storage.<name>.free_space** : 存储可用空间 (字节数) 。
- **storage.<name>.used_space** : 存储已经使用空间 (字节数) 。
- **storage.<name>.happy** : 存储健康状态。

V. 特定功能性语句

[回目录](#)

- **ban(expression)** : 清除指定对象缓存
- **call(subroutine)** : 调用子程序，如：**call(name)** ;
- **hash_data(input)** : 生成 hash 键，用于制定 hash 键值生成结构，只能在 **vcl_hash** 子程序中使用。调用 **hash_data(input)** 后，即这个 hash 为当前页面的缓存 hash 键值，无需其它获取或操作，如：

```
sub vcl_hash{
    hash_data(client.ip);
    return(lookup);
}
```

注意：**return(lookup);** 是默认返回值，所以可以不写。

- **new()** : 创建一个 **vcl** 对象，只能在 **vcl_init** 子程序中使用。
- **return()** : 结束当前子程序，并指定继续下一步动作，如：**return (ok);** 每个子程序可指定的动作均有不同。

- `rollback()` : 恢复HTTP头到原来状态，已经弃用，使用 `std.rollback()` 代替。
- `synthetic(STRING)` : 合成器，用于自定义一个响应内容，比如当请求出错时，可以返回自定义 `404` 内容，而不只是默认头信息，只能在 `vcl_synth` 与 `vcl_backend_error` 子程序中使用，如：

```
sub vcl_synth {
    //自定义内容
    synthetic ({""
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html lang="zh-cn">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <title>error</title>
    </head>
    <body>
        <h1>Error</h1>
        <h3>这是一个测试自定义响应异常内容</h3>
    </body>
</html>
"});
    //只交付自定义内容
    return(deliver);
}
```

- `regsub(str, regex, sub)` : 使用正则替换第一次出现的字符串，第一个参数为待处理字符串，第二个参数为正则表达式，第三个为替换为字符串。
- `regsuball(str, regex, sub)` : 使用正则替换所有匹配字符串。参数与 `regsuball` 相同。具体变量详见：<https://www.varnish-cache.org/docs/4.0/reference/vcl.html#reference-vcl>

vii. `return` 语句

[回目录](#)

`return` 语句是终止子程序并返回动作，所有动作都根据不同的VCL子程序限定来选用。<https://www.varnish-cache.org/docs/4.0/users-guide/vcl-built-in-subs.html>

语法：

```
return(action);
```

常用的动作：

- `abandon` 放弃处理，并生成一个错误。
- `deliver` 交付处理
- `fetch` 从后端取出响应对象
- `hash` 哈希缓存处理
- `lookup` 查找缓存对象
- `ok` 继续执行
- `pass` 进入`pass`非缓存模式
- `pipe` 进入`pipe`非缓存模式
- `purge` 清除缓存对象，构建响应
- `restart` 重新开始
- `retry` 重试后端处理
- `synth(status code, reason)` 合成返回客户端状态信息

viii. varnish 中内置子程序

[回目录](#)

注：varnish内置子程序均有自己限定的返回动作 `return (动作)`；不同的动作将调用对应下一个子程序。

vcl_recv 子程序

开始处理请求，通过 `return (动作)`；选择varnish处理模式，默认进入 `hash` 缓存模式（即 `return(hash);`），缓存时间为配置项 `default_ttl`（默认为 `120秒`）过期保持时间 `default_grace`（默认为 `10秒`）。该子程序一般用于模式选择，请求对象缓存及信息修改，后端节点修改，终止请求等操作。

- 可操作对象：（部分或全部值）
- 读：`client`，`server`，`req`，`storage`
- 写：`client`，`req`
- 返回值：
 - `synth(status code, reason);` 定义响应内容。

- `pass` 进入 `pass` 模式，并进入 `vcl_pass` 子程序。
- `pipe` 进入 `pipe` 模式，并进入 `vcl_pipe` 子程序。
- `hash` 进入 `hash` 缓存模式，并进入 `vcl_hash` 子程序，默认返回值。
- `purge` 清除缓存等数据，子程序先从 `vcl_hash` 再到 `vcl_purge`。

`vcl_pipe` 子程序

`pipe` 模式处理，该模式主要用于直接取后端响应内容返回客户端，可定义响应内容返回客户端。该子程序一般用于需要及时且不作处理的后端信息，取出后端响应内容后直接交付到客户端不进入 `vcl_deliver` 子程序处理。

- 可操作对象：（部分或全部值）
- 读：`client`，`server`，`bereq`，`req`，`storage`
- 写：`client`，`bereq`，`req`
- 返回值：
 - `synth(status code, reason);` 定义响应内容。
 - `pipe` 继续 `pipe` 模式，进入后端 `vcl_backend_fetch` 子程序，默认返回值。

`vcl_pass` 子程序

`pass` 模式处理，该模式类似 `hash` 缓存模式，仅不做缓存处理。

- 可操作对象：（部分或全部值）
- 读：`client`，`server`，`req`，`storage`
- 写：`client`，`req`
- 返回值：
 - `synth(status code, reason);` 定义响应内容。
 - `fetch` 继续 `pass` 模式，进入后端 `vcl_backend_fetch` 子程序，默认返回值。

`vcl_hit` 子程序

`hash` 缓存模式时，存在 `hash` 缓存时调用，用于缓存处理，可放弃或修改缓存。

- 可操作对象：（部分或全部值）
- 读：`client`，`server`，`obj`，`req`，`storage`
- 写：`client`，`req`
- 返回值：

- `restart` 重启请求。
- `deliver` 交付缓存内容，进入 `vc1_deliver` 子程序处理，默认返回值。
- `synth(status code, reason);` 定义响应内容。

`vc1_miss` 子程序

`hash` 缓存模式时，不存在 `hash` 缓存时调用，用于判断性的选择进入后端取响应内容，可以修改为 `pass` 模式。

- 可操作对象：（部分或全部值）
- 读：`client`，`server`，`req`，`storage`
- 写：`client`，`req`
- 返回值：
 - `restart` 重启请求。
 - `synth(status code, reason);` 定义响应内容。
 - `pass` 切换到 `pass` 模式，进入 `vc1_pass` 子程序。
 - `fetch` 正常取后端内容再缓存，进入 `vc1_backend_fetch` 子程序，默认返回值。

`vc1_hash` 子程序

`hash` 缓存模式，生成 `hash` 值作为缓存查找键名提取缓存内容，主要用于缓存 `hash` 键值处理，可使用 `hash_data(string)` 指定键值组成结构，可在同一个页面通过 `IP` 或 `cookie` 生成不同的缓存键值。

- 可操作对象：（部分或全部值）
- 读：`client`，`server`，`req`，`storage`
- 写：`client`，`req`
- 返回值：
 - `lookup` 查找缓存对象，存在缓存进入 `vc1_hit` 子程序，不存在缓存进入 `vc1_miss` 子程序，当使用了 `purge` 清理模式时会进入 `vc1_purge` 子程序，默认返回值。

`vc1_purge` 子程序

清理模式，当查找到对应的缓存时清除并调用，用于请求方法清除缓存，并报告。

- 可操作对象：（部分或全部值）
- 读：`client`，`server`，`req`，`storage`

- 写： `client` , `req`
- 返回值：
 - `synth(status code, reason);` 定义响应内容。
 - `restart` 重启请求。

`vcl_deliver` 子程序

客户端交付子程序，在 `vcl_backend_response` 子程序后调用（非 `pipe` 模式），或 `vcl_hit` 子程序后调用，可用于追加响应头信息，`cookie` 等内容。

- 可操作对象：（部分或全部值）
- 读： `client` , `server` , `req` , `resp` , `obj` , `storage`
- 写： `client` , `req` , `resp`
- 返回值：
 - `deliver` 正常交付后端或缓存响应内容，默认返回值。- `restart` 重启请求。

`vcl_backend_fetch` 子程序

发送后端请求之前调用，可用于改变请求地址或其它信息，或放弃请求。

- 可操作对象：（部分或全部值）
- 读： `server` , `bereq` , `storage`
- 写： `bereq`
- 返回值：
 - `fetch` 正常发送请求到后端取出响应内容，进入 `vcl_backend_response` 子程序，默认返回值。
 - `abandon` 放弃后端请求，并生成一个错误，进入 `vcl_backend_error` 子程序。

`vcl_backend_response` 子程序

后端响应后调用，可用于修改缓存时间及缓存相关信息。

- 可操作对象：（部分或全部值）
- 读： `server` , `bereq` , `beresp` , `storage`
- 写： `bereq` , `beresp`
- 返回值：
 - `deliver` 正常交付后端响应内容，进入 `vcl_deliver` 子程序，默认返回值。

- `abandon` 放弃后端请求，并生成一个错误，进入 `vcl_backend_error` 子程序。
- `retry` 重试后端请求，重试计数器加1，当超过配置中 `max_retries` 值时会报错并进入 `vcl_backend_error` 子程序。

`vcl_backend_error` 子程序

后端处理失败调用，异常页面展示效果处理，可自定义错误响应内容，或修改 `beresp.status` 与 `beresp.http.Location` 重定向等。

- 可操作对象：（部分或全部值）
- 读：`server`，`bereq`，`beresp`，`storage`
- 写：`bereq`，`beresp`
- 返回值：
 - `deliver` 只交付 `sysnthetic(string)` 自定义内容，默认返回后端异常标准错误内容。
 - `retry` 重试后端请求，重试计数器加 1，当超过配置中 `max_retries` 值时会报错并进入 `vcl_backend_error` 子程序。

`vcl_synth` 子程序

自定义响应内容。可以通过 `synthetic()` 和返回值 `synth` 调用，这里可以自定义异常显示内容，也可以修改 `resp.status` 与 `resp.http.Location` 重定向。

- 可操作对象：（部分或全部值）
- 读：`client`，`server`，`req`，`resp`，`storage`
- 写：`req`，`resp`
- 返回值：
 - `deliver` 只交付 `sysnthetic(string)` 自定义内容，默认返回 `sysnth` 异常指定状态码与错误内容。
 - `restart` 重启请求。

`vcl_init` 子程序

加载 `vcl` 时最先调用，用于初始化 VMODs，该子程序不参与请求处理，仅在 `vcl` 加载时调用一次。

- 可操作对象：（部分或全部值）
- 读：`server`

- 写：无
- 返回值：
 - `ok` 正常返回，进入 `vcl_recv` 子程序，默认返回值。

`vcl_fini` 子程序

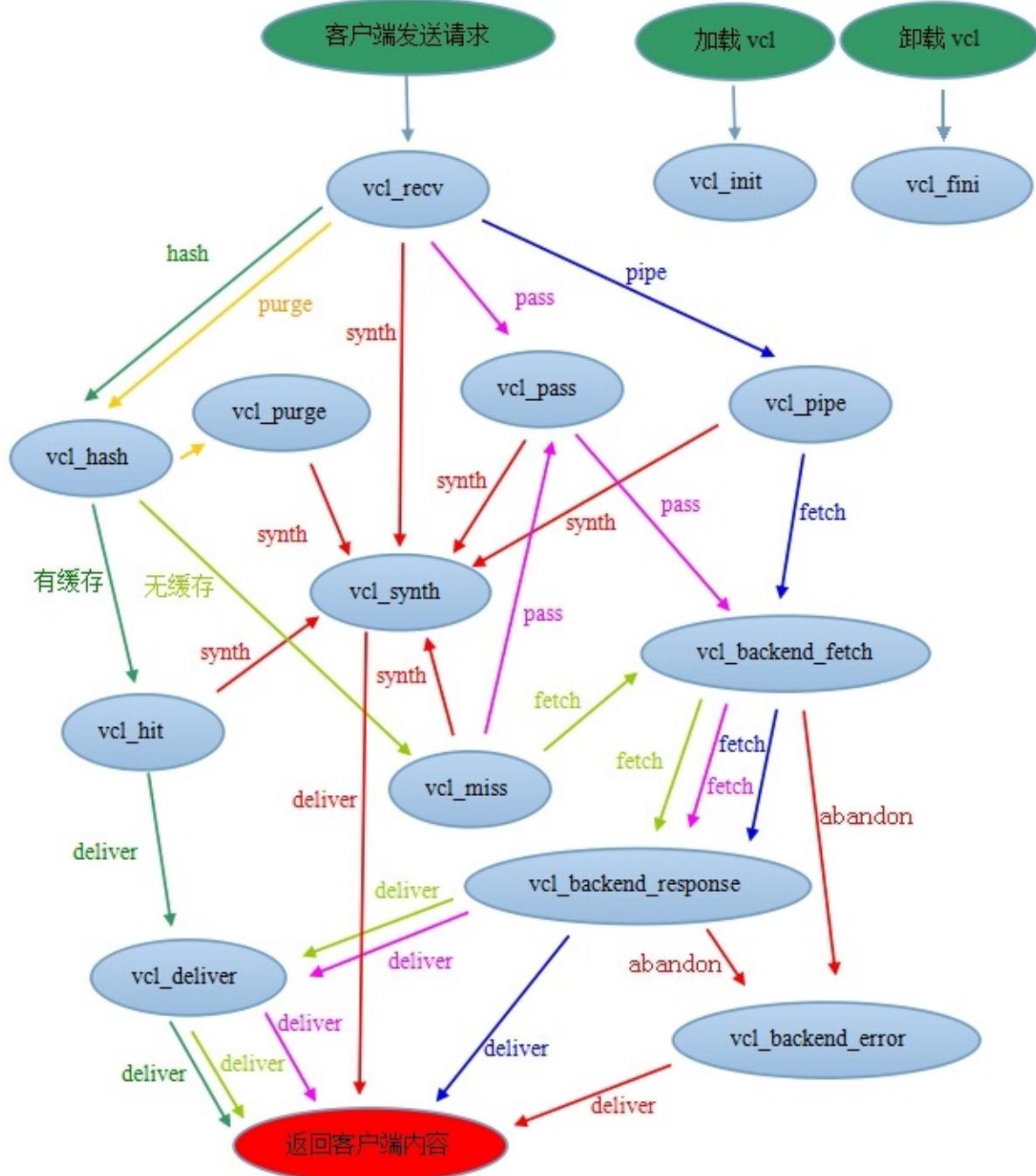
卸载当前 `vcl` 配置时调用，用于清理 `VMODs`，该子程序不参与请求处理，仅在 `vcl` 正常丢弃后调用。

- 可操作对象：（部分或全部值）
- 读：`server`
- 写：无
- 返回值：- `ok` 正常返回，本次 `vcl` 将释放，默认返回值。

varnish子程序调用流程图，通过大部分子程序的return返回值进入下一步行动：

IX. 优雅模式(Grace mode)

[回目录](#)



Varnish中的请求合并当几个客户端请求同一个页面的时候，varnish只发送一个请求到后端服务器，然后让其他几个请求挂起并等待返回结果；获得结果后，其它请求再复制后端的结果发送给客户端；

但如果同时有数以千计的请求，那么这个等待队列将变得庞大，这将导致2类潜在问题：惊群问题(thundering herd problem)，即突然释放大量的线程去复制后端返回的结果，将导致负载急速上升；没有用户喜欢等待；

故为了解决这类问题，可以配置varnish在缓存对象因超时失效后再保留一段时间，以给那些等待的请求返回过去的文件内容(stale content)，配置案例如下：

```

sub vcl_recv {
  if (! req.backend.healthy) {
    set req.grace = 5m;
  } else {
    set req.grace = 15s;
  }
}
sub vcl_fetch {
  set beresp.grace = 30m;
}

```

以上配置表示varnish将会将失效的缓存对象再多保留 30分钟，此值等于最大的 `req.grace` 值即可；

而根据后端主机的健康状况，varnish可向前端请求分别提供 5分钟 内或 15秒 内的过期内容

III. 安装 varnish

[回目录](#)

i. 安装依赖关系的软件包(注:使用 centos 在线 yum 源)

[回目录](#)

```
[root@varnish ~]# yum -y install autoconf automake jemalloc-devel libedit-devel libtool ncurses-devel pcre-devel pkgconfig python-docutils python-sphinx
```

ii. 安装 varnish

[回目录](#)

首先需要建立 varnish 用户以及用户组来运行Varnish，并且创建Varnish缓存目录和日志目录。

```
[root@varnish ~]# useradd -s /sbin/nologin varnish  
[root@varnish ~]# mkdir -p /data/varnish/{cache,log}  
[root@varnish ~]# chown -R varnish:varnish /data/varnish
```

Varnish的 [官方网址](#)，可以在这里下载最新版本的软件。[下载地址](#) 注意：

Varnish网站有时会被墙。 Git 下载：git clone

```
https://github.com/varnish/Varnish-Cache /var/tmp/
```

解压，进入解压目录编译安装

```
[root@varnish ~]# tar zxf varnish-4.0.3.tar.gz  
[root@varnish ~]# cd varnish-4.0.3/  
[root@varnish varnish-4.0.3]# export PKG_CONFIG_PATH=/usr/local/  
lib/pkgconfig
```

注：

```
./autogen.sh
```

如果从Git库下载的安装包时才需要运行，用于生成 configure 编译文件。配置：

```
[root@varnish varnish-4.0.3]# ./configure
```

注：不指定安装路径，默认是安装在/usr/local目录下

编译、安装

```
[root@varnish varnish-4.0.3]# make && make install
```

复制 vcl 文件（在编译安装目录下），如果安装目录里没有 default.vcl 文件。复制到安装目录的 /usr/local/var/varnish/ 目录下（当然并无必需要求在哪个目录，因为正式启动时还得指定这个文件的目录）

```
[root@varnish varnish-4.0.3]# cp etc/example.vcl /usr/local/var  
/varnish/default.vcl
```

IV. varnish 实例解析

[回目录](#)

varnish 配置基本上是编辑 VCL (Varnish Configuration Language) 文件, varnish 有一套自定义 VCL 语法, 启动时, 会将配置文件编译为 C 语言, 再执行。

varnish 4.0 开始, 每个VCL文件必须在开始行声明它的版本 vcl 4.0; 块 (子程序) 由大括号分隔, 语句用分号结束。所有的关键字及预设子程序名都是全小写。注释: 支持 // 或 # 多行时还可以使用 /* .. */

i. 后端服务器地址池配置及后端服务器健康检查

[回目录](#)

varnish 有"后端"或者"源"服务器的概念。 backend server 提供给 varnish 加速的内容。实际上就是给 varnish 添加可供访问的 web 服务器, 如果有多台 web 服务器时, 可添加多个 backend 块。

1. 后端服务器定义

[回目录](#)

命令: backend

这个定义为最基本的反向入口定义, 用于 varnish 连接对应的服务器, 如果没有定义或定义错误则用户无法访问正常页面。

语法格式:

```
backend name{
    .attribute = "value";
}
```

说明： `backend` 是定义后端关键字， `name` 是当前后端节点的 别名 ，多个后端节点时， `name` 名不能重复，否则覆盖。花括号里面定义当前节点相关的属性（ 键=值 ）。除默认节点外其它节点定义后必需有调用，否则 varnish 无法启动。后端是否正常可以通过 `std.healthy(backend)` 判断。

支持运算符：

`=` （赋值运算） `==` （相等比较） `~` （匹配，可以使用正则表达式，或访问控制列表） `!~` （不匹配，可以使用正则表达式，或访问控制列表） `!` （非）
`&&` （逻辑与） `||` （逻辑或）

属性列表：

- `.host="xxx.xxx.xxx.xxx";` //要转向主机（即后端主机）的IP或域名，必填键/值对。
- `.port="8080";` //主机连接端口号或协议名（HTTP等），默认80
- `.host_header='';` //请示主机头追加内容
- `.connect_timeout=1s;` //连接后端的超时时间
- `.first_byte_timeout=5s;` //等待从后端返回的第一个字节时间
- `.between_bytes_timeout=2s;` //每接收一个字节之间等待时间
- `.probe=probe_name;` //监控后端主机的状态,指定外部监控name或者内部直接添加
- `.max_connections=200;` //设置最大并发连接数，超过这个数后连接就会失败

例：（下面两个例子结果是一样的，但第二个例子中更适用于集群，可以方便批量修改）

```
backend web{
    .host="192.168.31.83";
    .port="80";
    .probe={           //直接追加监控块.probe是一个的参数
        .url="/";
        .timeout=2s;
    }
}
```

或 如果有多个web需要 .probe 属性，这个时候可以这样做。

```
probe web_probe{ //监控必需定义在前面，否则后端调用找不到监控块。
    .url="/";
    .timeout=2s;
}

backend web{
    .host="192.168.31.83";
    .port="80";
    .probe=web_probe; //调用外部共用监控块
}
```

2. 监视器定义

回目录

命令： probe

监控可以循环访问指定的地址，通过响应时间判定服务器是否空闲或正常。这类命令非常适用于集群中某些节点服务器崩溃或负载过重，而禁止访问这台节点服务器。

语法格式：

```
probe name{
    .attribute = "value";
}
```

说明：`probe` 是定义监控关键字，`name` 是当前监控点的别名，多个监控节点时，`name` 名不能重复，否则覆盖。花括号里面定义当前节点相关的属性（键=值）。没有必填属性，因为默认值就可以正常执行操作。

属性列表：

- `.url="/";` //指定监控入口URL地址，默认为 /
- `.request="";` //指定监控请求入口地址，比 `.url` 优先级高。
- `.expected_response="200";` //请求响应代码，默认是 200
- `.timeout=2s;` //请求超时时间。
- `.interval=5s;` //每次轮询请求间隔时间，默认为 5s。
- `.initial=-1;` //初始启动时以 `.window` 轮询次数中几次良好后续才能使用这个后端服务器节点，默认为 -1，则轮询完 `.window` 所有次数良好判定为正常。
- `.window=8;` //指定多少轮询次数，用于判定服务器正常，默认是 8。
- `.threshold=3;` //必须多少次轮询正常才算该后端节点服务器正常，默认是 3。

例：创建健康监测，定义健康检查名称为 `backend_healthcheck`

```
probe backend_healthcheck {
    .url = "/";
    .timeout = 1s;
    .interval = 5s;
    .window = 5;
    .threshold = 3;
}
```

在上面的例子中 `varnish` 将每 5s 检测后端，超时设为 1s。每个检测将会发送 `get /` 的请求。如果 5 个检测中大于 3 个是成功，`varnish` 就认为后端是健康的，反之，后端就有问题了。

3. 集群负载均衡 `directors`

[回目录](#)

varnish 可以定义多个后端，也可以将几个后端放在一个后端集群里面已达到负载均衡的目的。你也可以将几个后端组成一组后端。这个组被叫做 `Directors` 。可以提高性能和弹性。

`directors` 是 varnish 负载均衡模块，使用前必需引入 `directors` 模块，`directors` 模块主要包含：`round_robin`，`random`，`hash`，`fallback` 负载均衡模式。

- `round_robin`：循环依次逐个选择后端服务器。
- `random`：随机选择后端服务器，可设置每个后端权重增加随机率。
- `hash`：通过散列随机选择对应的后端服务器且保持选择对应关系，下次则直接找对应的后端服务器。
- `Fallback`: 后备

注意：`random`，`hash` 有权重值设置，用于提高随机率。每个后端最好都配置监控器（后端服务器正常监测）以便 `directors` 自动屏蔽不正常后端而不进入均衡队列中。

这些操作需要你载入 VMOD (varnish module)，然后在 `vcl_init` 中调用这个 VMOD 。

```

import directors;                      # load the directors
backend web1 {
    .host = "192.168.0.10";
    .port = "80";
    .probe = backend_healthcheck;
}
backend web2 {
    .host = "192.168.0.11";
    .port = "80";
    .probe = backend_healthcheck;
}
//初始化处理
sub vcl_init {                         //调用vcl_init初始化子程序创建后端主机组，即
    directors
        new web_cluster = directors.round_robin(); //使用new关键字创
建director对象，使用round_robin算法
        web_cluster.add_backend(web1);      //添加后端服务器节点
    web_cluster.add_backend(web2);
}
//开始处理请求
sub vcl_recv {                          //调用vcl_recv子程序，用于接收和
    处理请求
        set req.backend_hint = web_cluster.backend();      //选取后端
}

```

说明：

- `set` 命令是设置变量
- `unset` 命令是删除变量
- `web_cluster.add_backend(backend, real);` 添加后端服务器节
点，`backend` 为后端配置别名，`real` 为权重值，随机率计算公式： $100 * (当前权重 / 总权重)$ 。
- `req.backend_hint` 是 varnish 的预定义变量，作用是指定请求后端节点
vcl 对象需要使用 `new` 关键字创建，所有可创建对象都是内定的，使用前必
需 `import`，所有 `new` 操作只能在 `vcl_init` 子程序中。

扩展： varnish 将不同的 url 发送到不同的后端 server

```

import directors;                      # load the directors
backend web1 {
    .host = "192.168.0.10";
    .port = "80";
    .probe = backend_healthcheck;
}
backend web2 {
    .host = "192.168.0.11";
    .port = "80";
    .probe = backend_healthcheck;
}
backend img1 {
    .host = "img1.lnmmp.com";
    .port = "80";
    .probe = backend_healthcheck;
}
backend img2 {
    .host = "img2.lnmmp.com";
    .port = "80";
    .probe = backend_healthcheck;
}

```

//初始化处理

```

sub vcl_init {                  //调用vcl_init初始化子程序创建后端主机组，即
    directors
    new web_cluster = directors.round_robin(); //使用new关键字创建director对象，使用round_robin算法
    web_cluster.add_backend(web1);   //添加后端服务器节点
    web_cluster.add_backend(web2);
    new img_cluster = directors.random();
    img_cluster.add_backend(img1,2); //添加后端服务器节点，并且设置权重值
    img_cluster.add_backend(img2,5);
}

```

//根据不同的访问域名，分发至不同的后端主机组

```

sub vcl_recv {
    if (req.http.host ~ "(?i)^www\.\?benet\.com\$") {
        set req.backend_hint = web_cluster.backend(); //选取后端
    } elseif (req.http.host ~ "(?i)^images\.benet\.com\$") {
        set req.backend_hint = img_cluster.backend();
    }
}

```

说明：中的*i*就是忽略大小写的意思。`(?i)` 表示开启忽略大小写，而 `(?-i)` 表示关闭忽略大小写

4. 访问控制列表(ACL)

[回目录](#)

创建一个地址列表，用于后面的判断，可以是 域名 或 IP 集合。这个可以用于指定某些地址请求入口，防止恶意请求等。

语法格式：

```

acl purgers {
    "127.0.0.1";
    "localhost";
    "192.168.134.0/24"
    !"192.168.134.1";
}

```

说明：`acl` 是访问列表关键字（**必需小写**），`name` 是该列表的别名用于调用，花括号内部是地址集。

注意：如果列表中包含了无法解析的主机地址，它会匹配任何地址。如果不希望让它匹配可以在前添加一个 `!` 符号，如上面 `!"192.168.134.1";`

使用 `ACL` 只需要用 匹配运算符 `~` 或 `!~` 如：

```

sub vcl_recv {
    if (req.method == "PURGE") { //PURGE请求的处理
        if (client.ip ~ purgers) {
            return(purge);
        } else {
            return(synth(403, "Access denied."));
        }
    }
}

```

5.缓存规则配置

[回目录](#)

此处是完整的配置文件，为了视觉效果将整个配置文件拆开了。保留了配置文件中的注释 // .

```

sub vcl_recv {
    // PURGE请求的处理
    if (req.method == "PURGE") {
        if (!client.ip ~ purgers) {
            return (synth(405, "Not Allowed."));
        }
        return (purge);
    }

    set req.backend_hint = web.backend();
}

```

//将 php 、 asp 等动态内容访问请求直接发给后端服务器，不缓存。

```

if (req.url ~ "\.(php|asp|aspx|jsp|do|ashx|shtml)(\$|\?)" ) {
    return (pass);
}

```

//将非 GET 和 HEAD 访问请求直接发给后端服务器，不缓存。例如 POST 请求。

```

if (req.method != "GET" && req.method != "HEAD") {
    return (pass);
}

```

//如果 varnish 看到 header 中有 Authorization 头，它将 pass 请求。

```

if (req.http.Authorization) {
    return (pass);
}

```

//带 cookie 首部的 GET 请求也缓存

```

if (req.url ~ "\.(css|js|html|htm|bmp|png|gif|jpg|jpeg|ico|gz|
tgz|bz2|tbz|zip|rar|mp3|mp4|ogg|swf|flv)($|\?)") {
    unset req.http.cookie;
    return (hash);
}

```

说明：默认情况， varnish 不缓存从后端响应的 http 头中带有 Set-Cookie 的对象。如果客户端发送的请求带有 Cookie header ， varnish 将忽略缓存，直接将请求传递到后端。

//为发往后端主机的请求添加 X-Forward-For 首部,首次访问增加 X-Forwarded-For 头信息,方便后端程序获取 客户端ip ，而不是 varnish 地址

```

if (req.restarts == 0) {
    if (req.http.x-forwarded-for) {//如果设置过此header则要再次
        //加上逗号隔开
        set req.http.X-Forwarded-For = req.http.X-Forwarded-
        For + ", " + client.ip;
    } else {//如果只有一层代理的话,就无需设置了
        set req.http.X-Forwarded-For = client.ip;
    }
}

```

- req.restarts == 0 代表客户端首次访问

说明：`X-Forwarded-For` 是用来识别通过 `HTTP` 代理或负载均衡方式连接到 `Web` 服务器的客户端最原始的 `IP地址` 的 `HTTP` 请求头字段子程序：

子程序是一种类似 `C` 的函数，但是程序没有调用参数，子程序以 `sub` 关键字定义。在 `VCL` 里子程序是用于管理程序。

注意：所有 `VCL` 内置的程序都是以 `vcl_` 开头，并已经预置好，在 `VCL` 文件中只要声明对应的内置子程序，都会在对应的流程中调用。

V. varnish 完整配置实例

[回目录](#)

i. 拓扑环境

[回目录](#)

主机	地址
Varnish	:192.168.31.250
Web01	:192.168.31.83
Web02	:192.168.31.141

配置 `web01`、`web02` 做为后端服务器（过程略）确保 `varnish` 服务器能正常访问 `web01`、`web02` `Varnish` 缓存代理服务器配置：

ii. vcl 文件配置内容

[回目录](#)

```
[root@varnish ~]# cat /usr/local/var/varnish/default.vcl

#使用varnish版本4的格式.
vcl 4.0;
#加载后端负载均衡模块
import directors;
```

```

#加载std模块
import std;
#创建名为backend_healthcheck的健康检查策略
probe backend_healthcheck {
    .url="/";
    .interval = 5s;
    .timeout = 1s;
    .window = 5;
    .threshold = 3;
}

#定义后端服务器
backend web_app_01 {
    .host = "192.168.31.83";
    .port = "80";
    .first_byte_timeout = 9s;
    .connect_timeout = 3s;
    .between_bytes_timeout = 1s;
    .probe = backend_healthcheck;
}

backend web_app_02 {
    .host = "192.168.31.141";
    .port = "80";
    .first_byte_timeout = 9s;
    .connect_timeout = 3s;
    .between_bytes_timeout = 1s;
    .probe = backend_healthcheck;
}

#定义允许清理缓存的IP
acl purgers {
    "127.0.0.1";
    "localhost";
    "192.168.31.0/24";
}

#vcl_init初始化子程序创建后端主机组
sub vcl_init {
    new web = directors.round_robin();
    web.add_backend(web_app_01);
    web.add_backend(web_app_02);
}

#请求入口，用于接收和处理请求。这里一般用作路由处理，判断是否读取缓存和指定该

```

```

请求使用哪个后端
sub vcl_recv {
    #将请求指定使用web后端集群，在集群名后加上 .backend()
    set req.backend_hint = web.backend();

    # 匹配清理缓存的请求
    if (req.method == "PURGE") {
        if (!client.ip ~ purgers) {
            return (synth(405, "Not Allowed."));
        }
        # 是的话就执行清理
        return (purge);
    }

    # 如果不是正常请求 就直接穿透没商量
    if (req.method != "GET" &&
        req.method != "HEAD" &&
        req.method != "PUT" &&
        req.method != "POST" &&
        req.method != "TRACE" &&
        req.method != "OPTIONS" &&
        req.method != "PATCH" &&
        req.method != "DELETE") {
        return (pipe);
    }

    # 如果不是GET和HEAD就跳到pass
    if (req.method != "GET" && req.method != "HEAD") {
        return (pass);
    }

    #如果匹配动态内容访问请求就跳到pass
    if (req.url ~ "\.(php|asp|aspx|jsp|do|ashx|shtml)(\$|\?)") {
        return (pass);
    }

    #具有身份验证的请求跳到pass
    if (req.http.Authorization) {
        return (pass);
    }

    if (req.http.Accept-Encoding) {
        if (req.url ~ "\.(bmp|png|gif|jpg|jpeg|ico|gz|tgz|bz2|tb
z|zip|rar|mp3|mp4|ogg|swf|flv)$") {
            unset req.http.Accept-Encoding;
    }
}

```

```

        } elseif (req.http.Accept-Encoding ~ "gzip") {
            set req.http.Accept-Encoding = "gzip";
        } elseif (req.http.Accept-Encoding ~ "deflate") {
            set req.http.Accept-Encoding = "deflate";
        } else {
            unset req.http.Accept-Encoding;
        }
    }

    if (req.url ~ "\.(css|js|html|htm|bmp|png|gif|jpg|jpeg|ico|gz|tgz|bz2|tbz|zip|rar|mp3|mp4|ogg|swf|flv)($|\?)") {
        unset req.http.cookie;
        return (hash);
    }

# 把真实客户端IP传递给后端服务器 后端服务器日志使用X-Forwarded-For来接收
    if (req.restarts == 0) {
        if (req.http.X-Forwarded-For) {
            set req.http.X-Forwarded-For = req.http.X-Forwarded-For + ", " + client.ip;
        } else {
            set req.http.X-Forwarded-For = client.ip;
        }
    }
    return (hash);
}

# hash事件(缓存事件)
sub vcl_hash {
    hash_data(req.url);
    if (req.http.host) {
        hash_data(req.http.host);
    } else {
        hash_data(server.ip);
    }
    return (lookup);
}

# 缓存命中事件
sub vcl_hit {
    if (req.method == "PURGE") {
        return (synth(200, "Purged."));
    }
}

```

```

        return (deliver);
    }
# 缓存不命中事件
sub vcl_miss {
    if (req.method == "PURGE") {
        return (synth(404, "Purged."));
    }
    return (fetch);
}
# 返回给用户的前一个事件 通常用于添加或删除header头
sub vcl_deliver {
    if (obj.hits > 0) {
        set resp.http.X-Cache = "HIT";
        set resp.http.X-Cache-Hits = obj.hits;
    } else {
        set resp.http.X-Cache = "MISS";
    }
#取消显示php框架版本的header头
unset resp.http.X-Powered-By;
#取消显示web软件版本、Via(来自varnish)等header头 为了安全
unset resp.http.Server;
unset resp.http.X-Drupal-Cache;
unset resp.http.Via;
unset resp.http.Link;
unset resp.http.X-Varnish;
#显示请求经历restarts事件的次数
set resp.http.xx_restarts_count = req.restarts;
#显示该资源缓存的时间单位秒
set resp.http.xx_Age = resp.http.Age;
#显示该资源命中的次数
set resp.http.hit_count = obj.hits;
#取消显示Age 为了不和CDN冲突
unset resp.http.Age;
#返回给用户
        return (deliver);
}
# pass事件
sub vcl_pass {
    return (fetch);
}

```

```

#处理对后端返回结果的事件(设置缓存、移除cookie信息、设置header头等) 在fet
ch事件后自动调用
sub vcl_backend_response {
    #开启grace模式 表示当后端全挂掉后 即使缓存资源已过期(超过缓存时间) 也
    //会把该资源返回给用户 资源最大有效时间为5分钟
    set beresp.grace = 5m;
    #后端返回如下错误状态码 则不缓存
    if (beresp.status == 499 || beresp.status == 404 || beresp.s
tatus == 502) {
        set beresp.uncacheable = true;
    }
    #如请求php或jsp 则不缓存
    if (bereq.url ~ "\.(php|jsp)(\?|\$)") {
        set beresp.uncacheable = true;
    } else { //自定义缓存文件的缓存时长, 即TTL值
        if (bereq.url ~ "\.(css|js|html|htm|bmp|png|gif|jpg|jpeg|
ico)($|\?)") {
            set beresp.ttl = 15m;
            unset beresp.http.Set-Cookie;
        } elseif (bereq.url ~ "\.(gz|tgz|bz2|tbz|zip|rar|mp3|mp4|
ogg|swf|flv)($|\?)") {
            set beresp.ttl = 30m;
            unset beresp.http.Set-Cookie;
        } else {
            set beresp.ttl = 10m;
            unset beresp.http.Set-Cookie;
        }
    }
    #返回给用户
    return (deliver);
}
sub vcl_purge {
    return (synth(200, "success"));
}

sub vcl_backend_error {
    if (beresp.status == 500 ||
        beresp.status == 501 ||
        beresp.status == 502 ||
        beresp.status == 503 ||

```

```

        beresp.status == 504) {
            return (retry);
        }
    }

sub vcl_fini {
    return (ok);
}

```

iii. 启动 varnish

[回目录](#)

当启动varnish时有两个重要的参数你必须设置: 一个是处理 http 请求的 tcp 监听端口, 另一个是处理真实请求的后端 server

注: 如果你使用操作系统自带的包管理工具安装的varnish, 你将在下面的文件找到启动参数: Red Hat , Centos : /etc/sysconfig/varnish

1. - a

[回目录](#)

- a 参数定义了varnish监听在哪个地址, 并用该地址处理 http 请求, 你可能想设置这个参数在众所周知的 http 80端口 . 例子:

- -a :80
- -a localhost:80
- -a 192.168.1.100:8080
- -a '[fe80::1]:80'
- -a '0.0.0.0:8080,[::]:8081'

如果你的 webserver 和 varnish 运行在同一台机器, 你必须换一个监听地址.

2. - f VCL-file or -b backend

[回目录](#)

-f 添加 vcl文件 , - b 定义后端 server varnish需要知道从哪里找到这个需要缓存的 http server .你可以用 -b 参数指定,或者帮把它放在 vcl文件 中,然后使用 -f 参数指定. 在启动的时候使用 -b 是一个快捷的方式. -b 192.168.1.2:80

注意:如果你指定的是 name ,这个 name 必须能解析成一个 IPv4 或者 IPv6 的地址 如果你使用 -f 参数,你启动的时候可以在 -f 指定 vcl文件 。

默认的varnish使用 100M 的内存来缓存对象,如果你想缓存更多,可以使用 -s 参数.
注 : Varnish拥有大量的有用的命令行参数,建议查看其帮助

```
[root@varnish ~]# /usr/local/sbin/varnishd -h
```

启动 varnish

```
[root@varnish ~]# /usr/local/sbin/varnishd -f /usr/local/var/varnish/default.vcl -s malloc,200M -a 0.0.0.0:80  
[root@varnish ~]# netstat -anpt | grep 80  
tcp      0      0 0.0.0.0:80          0.0.0.0:*        LISTEN      6173/varnishd
```

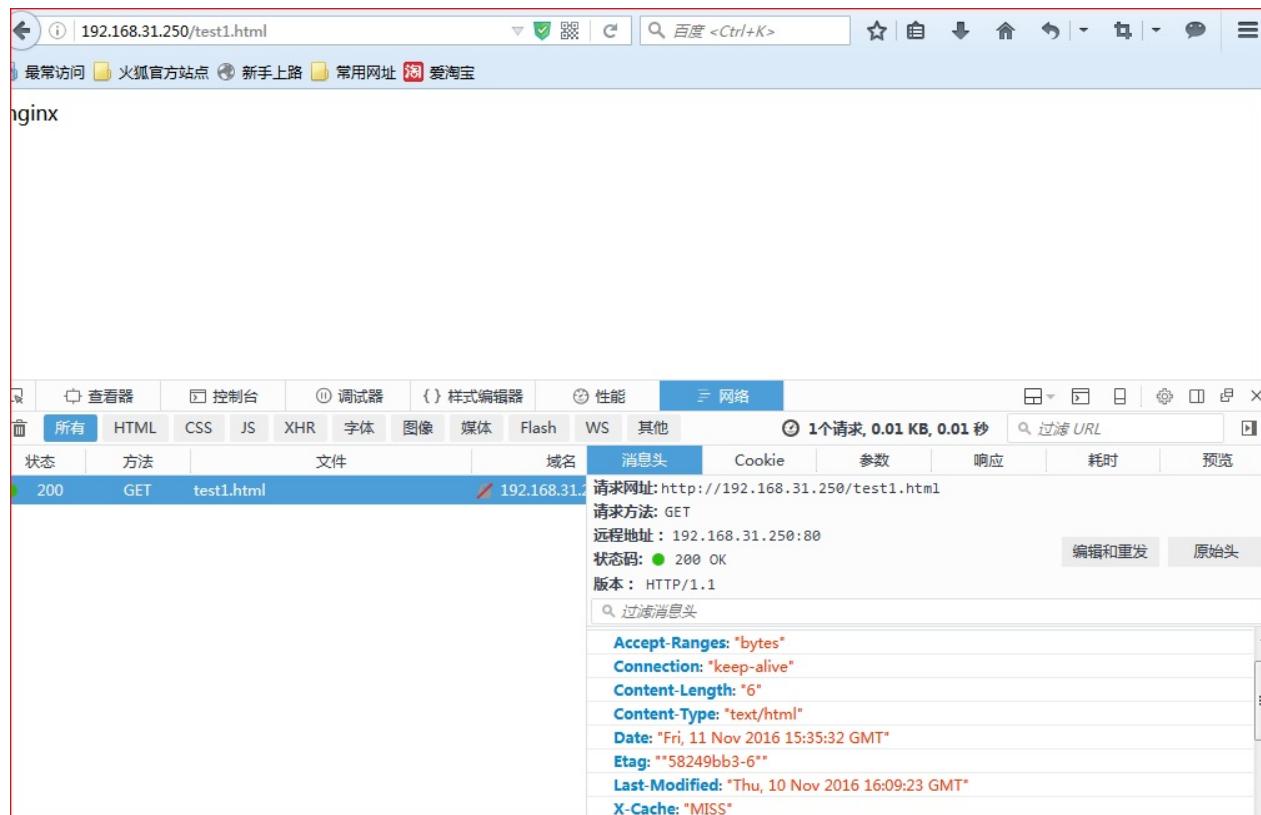
3. 访问测试 varnish

[回目录](#)

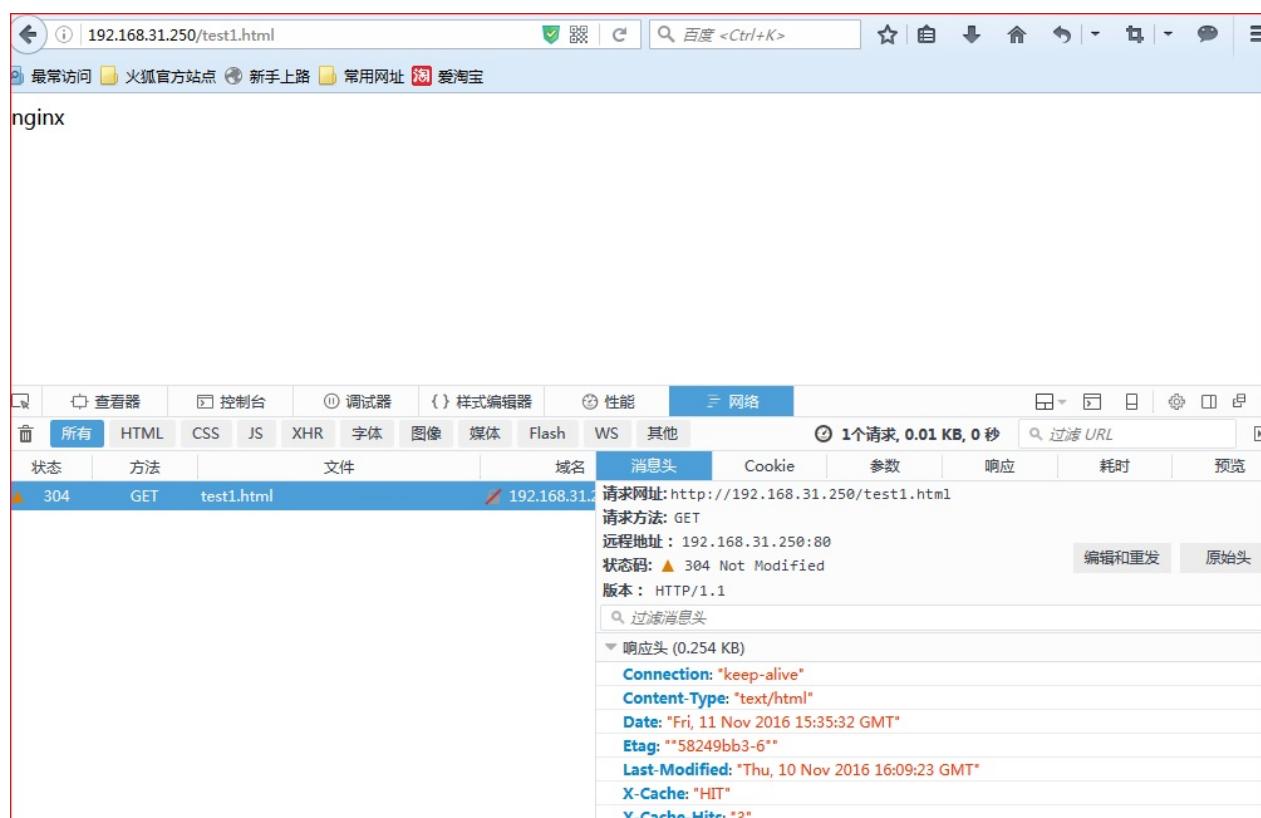
现在, varnish已经启动和运行, 你可以通过varnish访问您的Web应用程序。

打开火狐浏览器

第一次访问



第二次访问



4. varnish4 配置手动清除缓存

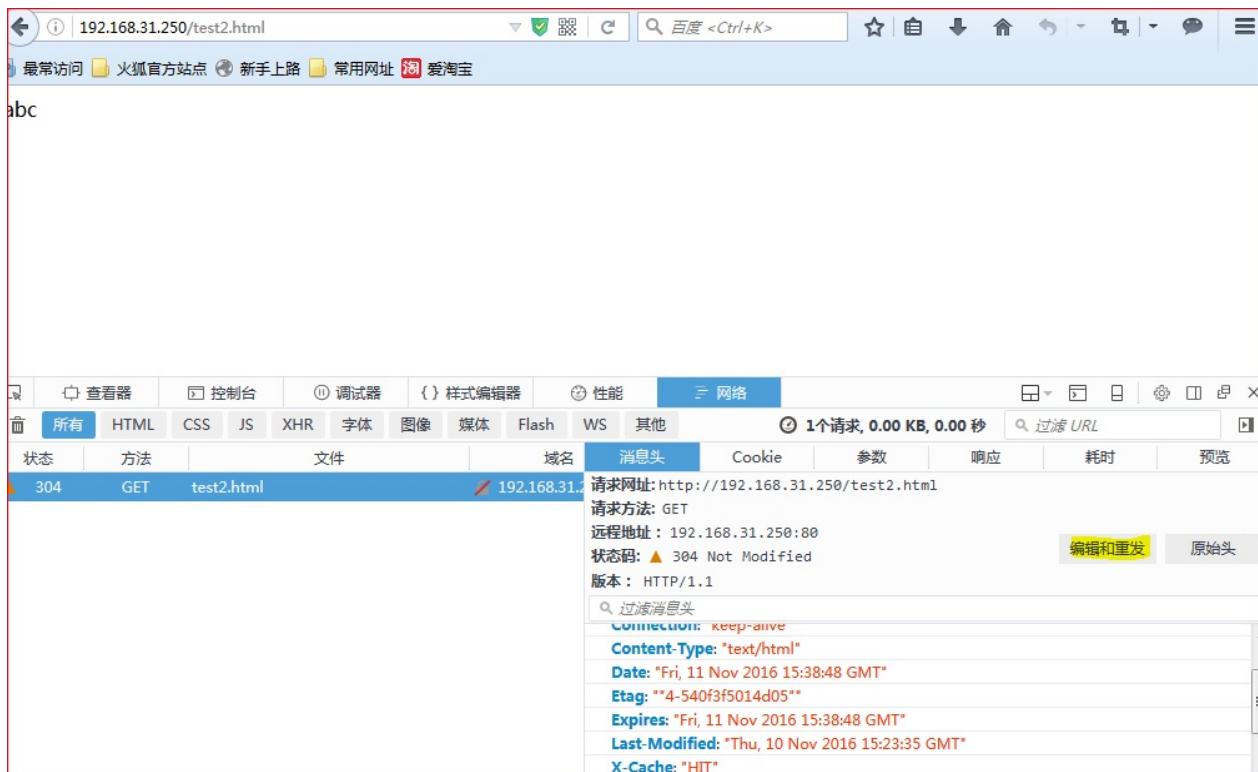
[回目录](#)

varnish4 通过 vcl 配置清楚缓存 通过 vcl 配置可以让客户端手动请求清楚缓存，以保证局部数据及时更新，而不用重启varnish服务器。

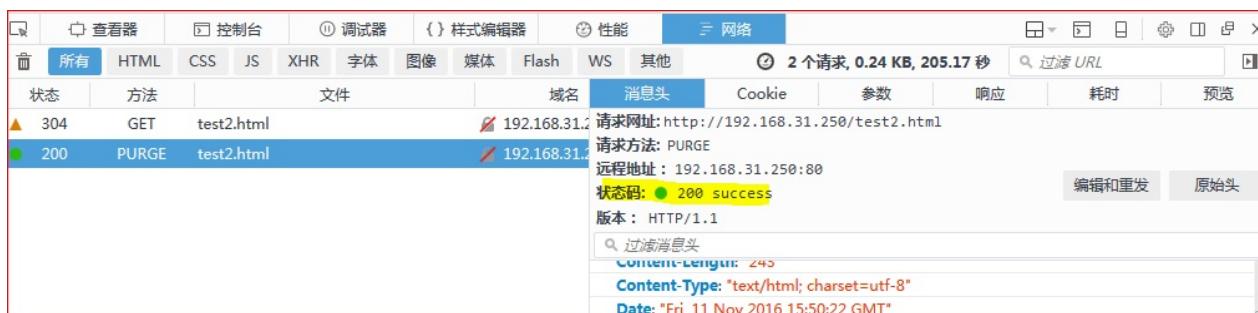
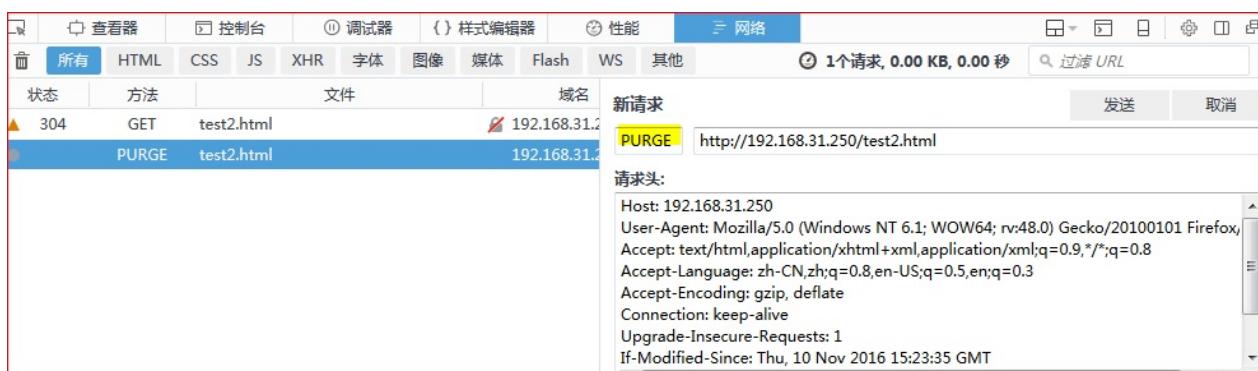
配置方法

```
#允许清除缓存IP集
acl purgers {
    "127.0.0.1";
    "localhost";
    "192.168.31.0/24";
}
sub vcl_recv {
    .....
    if (req.method == "PURGE") {
        if (!client.ip ~ purgers) {
            return (synth(405, "Not Allowed."));
        }
        return (purge); //清除缓存
    }
    .....
}
sub vcl_purge {
    return (synth(200, "success"));
}
```

打开火狐浏览器，随便进入一个缓存页面,如下图所示。



点击编辑和重发，修改请求类型为 PURGE 再点击发送



查看返回状态，如果成功则成功清除缓存，可以按 F5 刷新页面，查看新内容。

[回目录](#)

Memcache

目录

- I. MemCache 简
 - i. MemCache 访问模型
 - ii. Hash 算法
 - 1.余数 Hash
 - 2.一致性 Hash 算法
 - iii. MemCache 实现原理
 - iv. Memcache 的工作流程
 - v. Memcached 特征
- II. centos7.2 + nginx + php + memcache + mysql
 - 环境描述
 - i. 安装 nginx (在 192.168.31.141 主机操作)
 - 1.解压 zlib
 - 2.解压 pcre
 - 3.解压源码包
 - 4.启动 nginx
 - ii. 安装php
 - 1.安装 libmcrypt
 - 2.创建 php-fpm 服务启动脚本
 - iii. 安装 mysql (在 192.168.31.225 主机操作)
 - 1.查看是否安装 mariadb
 - 2.安装依赖包
 - 3.编译安装 mysql
 - 4.设置权限并初始化 MySQL 系统授权表
 - 5.创建配置文件
 - 6.配置 mysql 自动启动
 - iv. 安装 memcached 服务端 (在 192.168.31.250 主机操作)
 - 1.首先先安装 memcached 依赖库 libevent
 - 2.安装 memcached
 - 3.检测是否成功安装
 - 4.设置防火墙

- 5. 刷新用户环境变量
- 6. 编写 memcached 服务启停脚本
- 7. 配置 nginx.conf 文件 (在 nginx 主机操作)
- v. memcache 客户端 (在 php 服务器操作) :
 - 1. 安装 PHP Memcache 扩展
 - 2. 安装 memcache 扩展库
 - 3. 测试 memcache 可用性
- vi. 测试 memcache 缓存数据库数据
 - 1. 在 Mysql 服务器上创建测试表
 - 2. 在 web 服务器上创建测试脚本内容如下
 - 3. 访问页面测试
 - 4. 查看 Memcached 缓存情况

I. MemCache 简

[回目录](#)

session

MemCache 是一个自由、源码开放、高性能、分布式的分布式内存对象缓存系统，用于动态 Web 应用以减轻数据库的负载。它通过在内存中缓存数据和对象来减少读取数据库的次数，从而提高了网站访问的速度。 MemCache 是一个存储键值对的 HashMap，在内存中对任意的数据（比如字符串、对象等）所使用的 key-value 存储，数据可以来自数据库调用、 API 调用，或者页面渲染的结果。

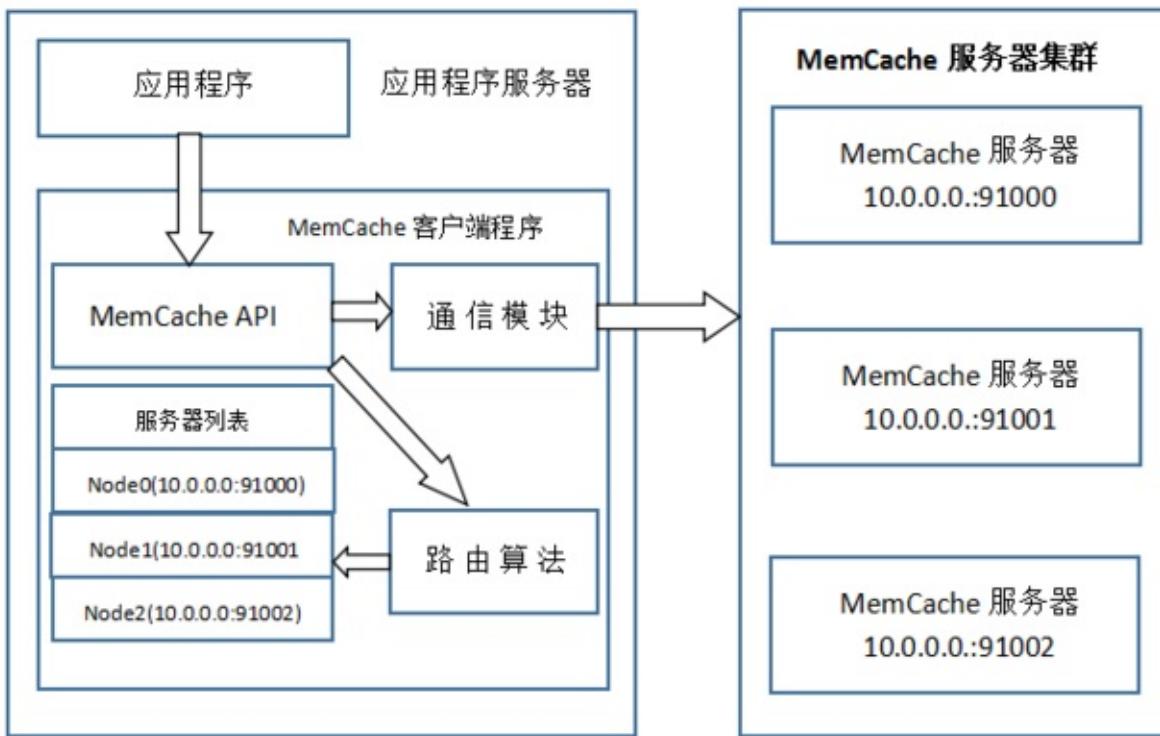
MemCache 设计理念就是小而强大，它简单的设计促进了快速部署、易于开发并解决面对大规模的数据缓存的许多难题，而所开放的 API 使得 MemCache 能用于 Java 、 C/C++/C# 、 Perl 、 Python 、 PHP 、 Ruby 等大部分流行的程序语言。

另外，说一下为什么会有 Memcache 和 memcached 两种名称？其实 Memcache 是这个项目的名称，而 memcached 是它服务器端的主程序文件名
MemCache 的官方网站为 <http://memcached.org/>

I. MemCache 访问模型

[回目录](#)

为了加深对 memcache 的理解，以 memcache 为代表的分布式缓存，访问模型如下：



特别澄清一个问题，MemCache 虽然被称为“分布式缓存”，但是 MemCache 本身完全不具备分布式的功能，MemCache 集群之间不会相互通信（与之形成对比的，比如JBoss Cache，某台服务器有缓存数据更新时，会通知集群中其他机器更新缓存或清除缓存数据），所谓的“分布式”，完全依赖于客户端程序的实现，就像上面这张图的流程一样。

同时基于这张图，理一下 MemCache 一次写缓存的流程：

- 1、应用程序输入需要写缓存的数据
 - 2、API将Key输入路由算法模块，路由算法根据 Key 和 MemCache 集群服务器列表得到一台服务器编号
 - 3、由服务器编号得到 MemCache 及其的 ip地址 和 端口号
 - 4、API 调用通信模块和指定编号的服务器通信，将数据写入该服务器，完成一次分布式缓存的写操作
- 读缓存和写缓存一样，只要使用相同的路由算法和服务端列表，只要应用程序查询的是相同的 Key，MemCache 客户端总是访问相同的客户端去读取数据，只要服务器中还缓存着该数据，就能保证缓存命中。

这种 MemCache 集群的方式也是从分区容错性的方面考虑的，假如 Node2 宅机了，那么 Node2 上面存储的数据都不可用了，此时由于集群中 Node0 和 Node1 还存在，下一次请求 Node2 中存储的 Key 值的时候，肯定是没有命中的，这时先从数据库中拿到要缓存的数据，然后路由算法模块根据 Key 值在 Node0 和 Node1 中选取一个节点，把对应的数据放进去，这样下一次就又可以走缓存了，这种集群的做法很好，但是缺点是成本比较大。

ii. Hash 算法

[回目录](#)

从上面的图中，可以看出一个很重要的问题，就是对服务器集群的管理，路由算法至关重要，就和负载均衡算法一样，路由算法决定着究竟该访问集群中的哪台服务器，先看一个简单的路由算法。

1. 余数 Hash

[回目录](#)

简单的路由算法可以使用 余数Hash：用服务器数目和缓存数据 KEY 的 hash 值相除，余数为服务器列表下标编号，假如某个 str 对应的HashCode 是 52、服务器的数目是 3，取余数得到 1，str 对应节点 Node1，所以路由算法把 str 路由到 Node1 服务器上。由于HashCode 随机性比较强，所以使用 余数 Hash 路由算法就可以保证缓存数据在整个 MemCache 服务器集群中有比较均衡的分布。

如果不考虑服务器集群的伸缩性，那么 余数Hash 算法几乎可以满足绝大多数的缓存路由需求，但是当分布式缓存集群需要扩容的时候，就难办了。

就假设 MemCache 服务器集群由 3 台变为 4 台 吧，更改服务器列表，仍然使用 余数Hash，52 对 4 的余数是 0，对应 Node0，但是 str 原来是存在 Node1 上的，这就导致了缓存没有命中。再举个例子，原来有HashCode 为 0~19 的 20 个数据，

那么不妨举个例子，原来有HashCode 为 0~19 的 20 个数据，那么：

HashCode	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
路由到的服务器	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1

现在扩容到 4 台，加粗标红的表示命中：

HashCode	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
路由到的服务器	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3

如果扩容到 20+ 的台数，只有前三个 `HashCode` 对应的 `Key` 是命中的，也就是 15%。当然现实情况肯定比这个复杂得多，不过足以说明，使用 余数Hash 的路由算法，在扩容的时候会造成大量的数据无法正确命中（其实不仅仅是无法命中，那些大量的无法命中的数据还在原缓存中在被移除前占据着内存）。在网站业务中，大部分的业务数据度操作请求上事实上是通过缓存获取的，只有少量读操作会访问数据库，因此数据库的负载能力是以有缓存为前提而设计的。当大部分被缓存了的数据因为服务器扩容而不能正确读取时，这些数据访问的压力就落在了数据库的身上，这将大大超过数据库的负载能力，严重的可能会导致数据库宕机。

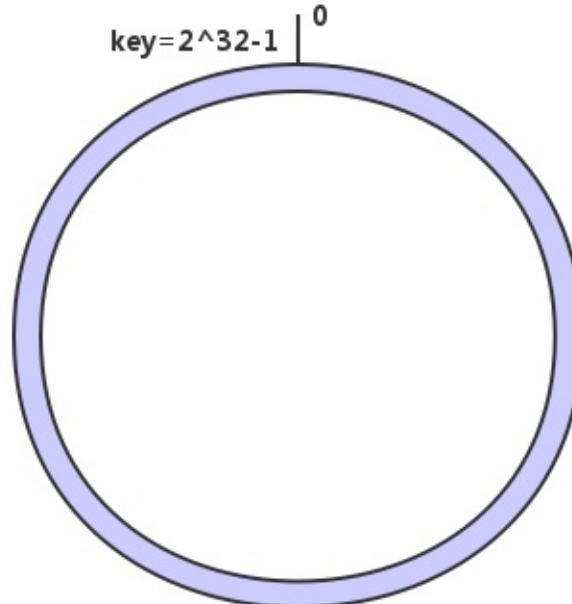
这个问题有解决方案，解决步骤为：

- (1) 在网站访问量低谷，通常是深夜，技术团队加班，扩容、重启服务器
- (2) 通过模拟请求的方式逐渐预热缓存，使缓存服务器中的数据重新分布

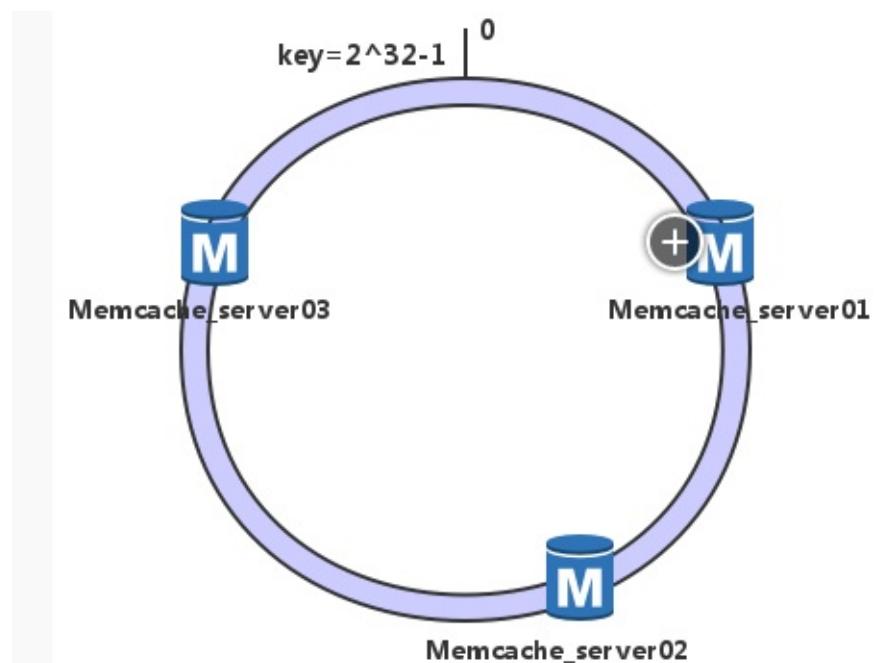
2.一致性 Hash 算法

[回目录](#)

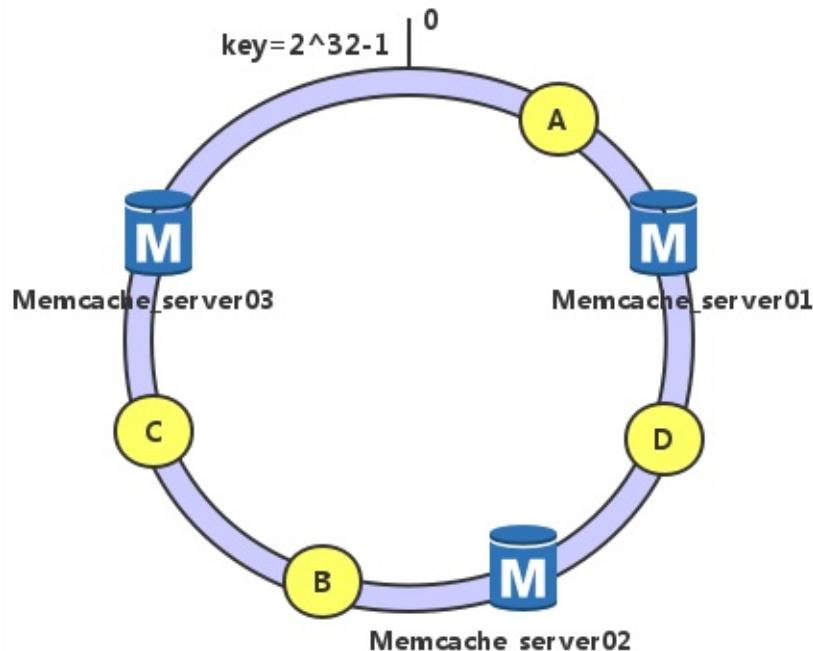
一致性Hash算法 通过一个叫做 一致性Hash环的数据结构 实现 `Key` 到缓存服务器的 `Hash` 映射。简单地说，一致性哈希将整个哈希值空间组织成一个虚拟的圆环（这个环被称为 一致性Hash环），如假设某空间哈希函数H的值空间是 0~2^32-1（即哈希值是一个 32 位无符号整形），整个哈希空间如下：



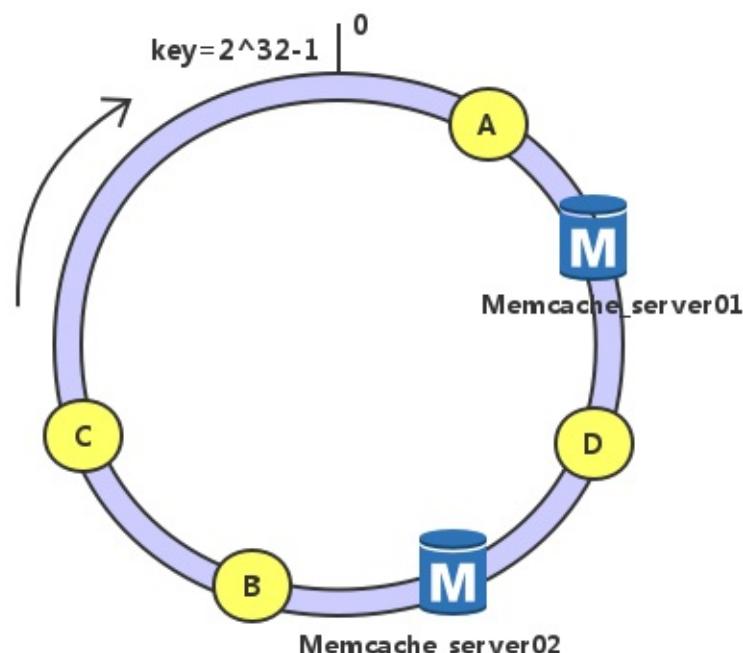
下一步将各个服务器使用H进行一个哈希计算，具体可以使用服务器的IP地址或者主机名作为关键字，这样每台机器能确定其在上面的哈希环上的位置了，并且是按照顺时针排列，这里我们假设三台节点 memcache 经计算后位置如下



接下来使用相同算法计算出数据的哈希值h,并由此确定数据在此哈希环上的位置 假如我们有数据 A 、 B 、 C 、 D 、 4 个对象，经过哈希计算后位置如下：

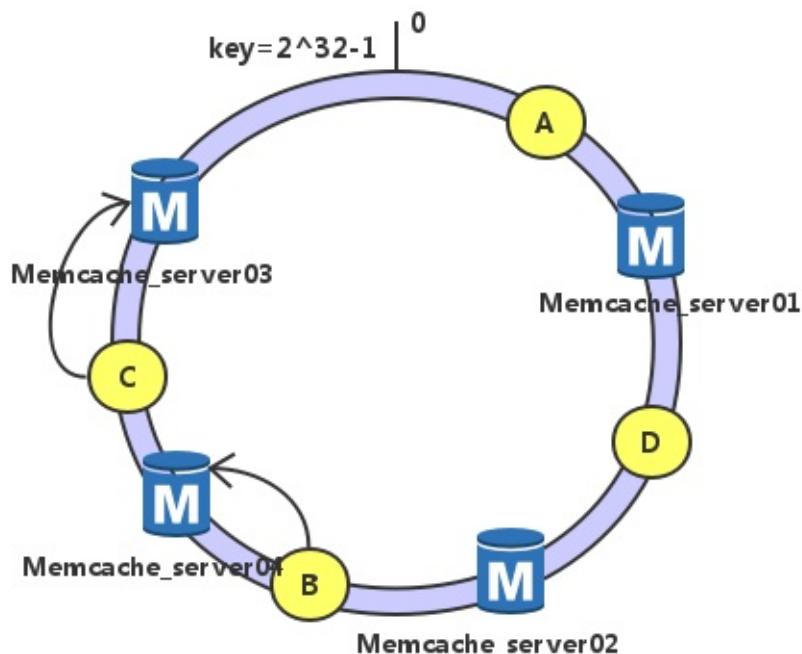


根据一致性哈希算法，数据A就被绑定到了 server01 上，D 被绑定到了 server02 上，B 、C 在 server03 上，是按照顺时针找最近服务节点方法这样得到的哈希环调度方法，有很高的容错性和可扩展性：假设 server03 宕机



可以看到此时 C 、 B 会受到影响，将 B 、 C 节点被重定位到 Server01 。一般的，在一致性哈希算法中，如果一台服务器不可用，则受影响的数据仅仅是此服务器到其环空间中前一台服务器（即顺着逆时针方向行走遇到的第一台服务器）之间数据，其它不会受到影响。

考虑另外一种情况，如果我们在系统中增加一台服务器 Memcached Server 04 :



此时 A 、 D 、 C 不受影响，只有 B 需要重定位到新的 Server04 。一般的，在一致性哈希算法中，如果增加一台服务器，则受影响的数据仅仅是新服务器到其环空间中前一台服务器（即顺着逆时针方向行走遇到的第一台服务器）之间数据，其它不会受到影响。

综上所述，一致性哈希算法对于节点的增减都只需重定位环空间中的一小部分数据，具有较好的容错性和可扩展性。

一致性哈希的缺点：

在服务节点太少时，容易因为节点分布不均匀而造成数据倾斜问题。我们可以采用增加虚拟节点的方式解决。

更重要的是，集群中缓存服务器节点越多，增加/减少节点带来的影响越小，很好理解。换句话说，随着集群规模的增大，继续命中原有缓存数据的概率会越来越大，虽然仍然有小部分数据缓存在服务器中不能被读到，但是这个比例足够小，即使访问数据库，也不会对数据库造成致命的负载压力。

iii. MemCache 实现原理

[回目录](#)

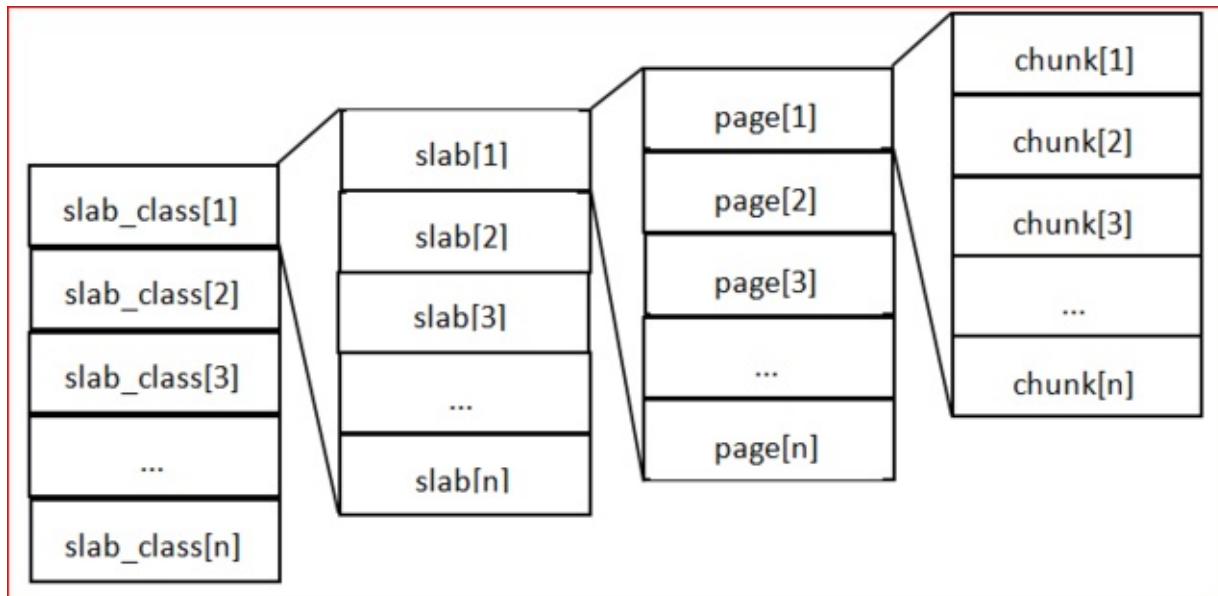
首先要说明一点， MemCache 的数据存放在内存中

- 1、访问数据的速度比传统的关系型数据库要快，因为 Oracle 、 MySQL 这些传统的关系型数据库为了保持数据的持久性，数据存放在硬盘中， IO 操作

速度慢

- 2、MemCache 的数据存放在内存中同时意味着只要 MemCache 重启了，数据就会消失
- 3、既然 MemCache 的数据存放在内存中，那么势必受到机器位数的限制，32 位机器最多只能使用 2GB 的内存空间，64 位机器可以认为没有上限

然后我们来看一下 MemCache 的原理，MemCache 最重要的是内存如何分配的，MemCache 采用的内存分配方式是固定空间分配，如下图所示：



这张图片里面涉及了 slab_class 、 slab 、 page 、 chunk 四个概念，它们之间的关系是：

- 1、MemCache 将内存空间分为一组 slab
 - 2、每个 slab 下又有若干个 page ，每个 page 默认是 1M ，如果一个 slab 占用 100M 内存的话，那么这个 slab 下应该有 100 个 page
 - 3、每个 page 里面包含一组 chunk ， chunk 是真正存放数据的地方，同一个 slab 里面的 chunk 的大小是固定的
 - 4、有相同大小 chunk 的 slab 被组织在一起，称为 slab_class
- MemCache 内存分配的方式称为 allocator (分配运算)，slab 的数量是有限的，几个、十几个或者几十个，这个和启动参数的配置相关。

MemCache 中的 value 存放的地方是由 value 的大小决定的， value 总是会被存放到与 chunk 大小最接近的一个 slab 中，比如

- slab[1] 的 chunk 大小为 80 字节
- slab[2] 的 chunk 大小为 100 字节
- slab[3] 的 chunk 大小为 125 字节 (相邻 slab 内的 chunk 基本

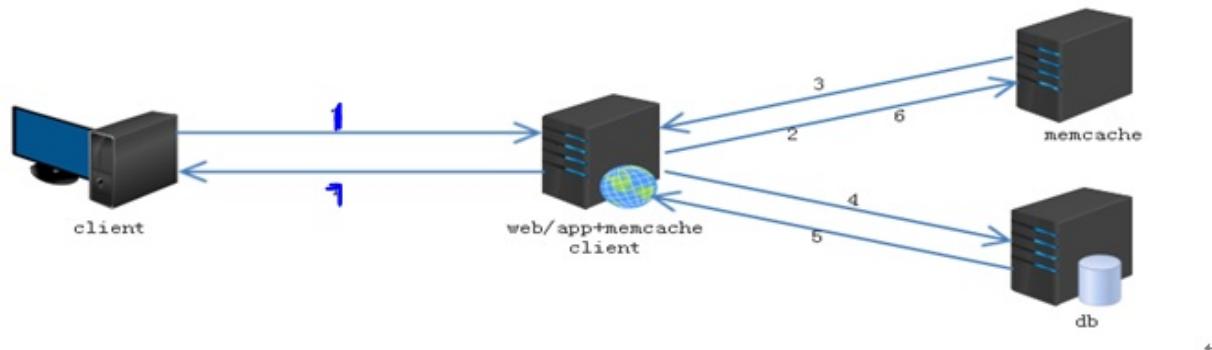
以 1.25 为比例进行增长， MemCache 启动时可以用 -f 指定这个比例）

那么过来一个 88 字节的 value ，这个 value 将被放到 2 号 slab 中。放 slab 的时候，首先 slab 要申请内存，申请内存是以 page 为单位的，所以在放入第一个数据的时候，无论大小为多少，都会有 1M 大小的 page 被分配给该 slab 。申请到 page 后， slab 会将这个 page 的内存按 chunk 的大小进行切分，这样就变成了一个 chunk 数组，最后从这个 chunk 数组中选择一个用于存储数据。

如果这个 slab 中没有 chunk 可以分配了怎么办，如果 MemCache 启动没有追加 -M （禁止 LRU ，这种情况下内存不够会报 Out Of Memory 错误），那么 MemCache 会把这个 slab 中最近最少使用的 chunk 中的数据清理掉，然后放上最新的数据。

iv. Memcache 的工作流程

回目录



- 1、检查客户端的请求数据是否在 memcached 中，如果有，直接把请求数据返回，不再对数据库进行任何操作，路径操作为①②③⑦。
- 2、如果请求的数据不在 memcached 中，就去查数据库，把从数据库中获取的数据返回给客户端，同时把数据缓存一份到 memcached 中（ memcached 客户端不负责，需要程序明确实现），路径操作为①②④⑤⑦⑥。
- 3、每次更新数据库的同时更新 memcached 中的数据，保证一致性。
- 4、当分配给 memcached 内存空间用完之后，会使用 LRU （ Least Recently Used ，最近最少使用）策略加上到期失效策略，失效数据首先被替换，然后再替换掉最近未使用的数据。

V. Memcached 特征

[回目录](#)

协议简单：

它是基于文本行的协议，直接通过 telnet 在 memcached 服务器上可进行存取数据操作
注：文本行的协议：指的是信息以文本传送，一个信息单元传递完毕后要传送换行。比如对于 HTTP 的 GET 请求来说， GET /index.html HTTP/1.1 是一行，接下去每个头部信息各占一行。一个空行表示整个请求结束

基于libevent事件处理：

Libevent 是一套利用 C 开发的程序库，它将 BSD 系统的 kqueue，Linux 系统的 epoll 等事件处理功能封装成一个接口，与传统的 select 相比，提高了性能。

内置的内存管理方式：

所有数据都保存在内存中，存取数据比硬盘快，当内存满后，通过 LRU 算法自动删除不使用的缓存，但没有考虑数据的容灾问题，重启服务，所有数据会丢失。

分布式

各个 memcached 服务器之间互不通信，各自独立存取数据，不共享任何信息。服务器并不具有分布式功能，分布式部署取决于 memcache 客户端。

Memcache的安装

分为两个过程：

- memcache 服务器端的安装
- memcached 客户端的安装。

所谓服务器端的安装就是在服务器（一般都是linux系统）上安装 Memcache 实现数据的存储。所谓客户端的安装就是指 php （或者其他程序， Memcache 还有其他的不错的 api 接口提供）去使用服务器端的 Memcache 提供的函数，需要 php 添加扩展。 PHP 的 Memcache

II. centos7.2 + nginx + php + memcache

+ mysql

[回目录](#)

环境描述

[回目录](#)

OS

```
[root@www ~]# cat /etc/redhat-release
CentOS Linux release 7.2.1511 (Core)
```

nginx和php

- nginx-1.10.2.tar.gz
- php-5.6.27.tar.gz
- ip地址 : 192.168.31.141/24

memcache

- memcached-1.4.33.tar.gz
- ip地址 : 192.168.31.250/24

mysql

- mysql-5.7.13.tar.gz
- ip地址 : 192.168.31.225/24

i. 安装 nginx (在 192.168.31.141 主机操作)

[回目录](#)

1. 解压 zlib

[回目录](#)

```
[root@www ~]# tar zxf zlib-1.2.8.tar.gz
```

说明：不需要编译，只需要解压就行。

2. 解压 pcre

[回目录](#)

```
[root@www ~]# tar zxf pcre-8.39.tar.gz
```

说明：不需要编译，只需要解压就行。

```
[root@www ~]# yum -y install gcc gcc-c++ make libtool openssl openssl-devel
```

下载 nginx 的源码包：<http://nginx.org/download>

3. 解压源码包

[回目录](#)

```
[root@www ~]# tar zxf nginx-1.10.2.tar.gz
[root@www ~]# cd nginx-1.10.2/
[root@www ~]# groupadd www #添加www组
[root@www ~]# useradd -g www www -s /sbin/nologin #创建nginx运行
账户www并加入到www组，不允许www用户直接登录系统
[root@www nginx-1.10.2]# ./configure --prefix=/usr/local/nginx1.
10 --with-http_dav_module --with-http_stub_status_module --with-
http_addition_module --with-http_sub_module --with-http_flv_mod
ule --with-http_mp4_module --with-pcre=/root/pcre-8.39 --with-zl
ib=/root/zlib-1.2.8 --with-http_ssl_module --with-http_gzip_stat
ic_module --user=www --group=www
[root@www nginx-1.10.2]# make&& make install
```

注：

- `--with-pcre` : 用来设置 pcre 的源码目录。
- `--with-zlib` : 用来设置 zlib 的源码目录。

因为编译 nginx 需要用到这两个库的源码。

```
[root@www nginx-1.10.2]# ln -s /usr/local/nginx1.10/sbin/nginx /usr/local/sbin/
[root@www nginx-1.10.2]# nginx -t
```

4. 启动 nginx

[回目录](#)

```
[root@www nginx-1.10.2]# nginx
[root@www nginx-1.10.2]# netstat -anpt | grep nginx
tcp      0      0 0.0.0.0:80          0.0.0.0:*      LISTEN      9834/nginx:
master
[root@www nginx-1.10.2]# firewall-cmd --permanent --add-port=80/tcp
success
[root@www nginx-1.10.2]# firewall-cmd --reload
success
```

启动后可以在浏览器中打开页面，会显示 nginx 默认页面。



ii. 安装 php

[回目录](#)

1. 安装 libmcrypt

[回目录](#)

```
[root@www ~]# tar zxf libmcrypt-2.5.7.tar.gz  
[root@www ~]# cd libmcrypt-2.5.7/  
[root@www libmcrypt-2.5.7]# ./configure --prefix=/usr/local/libmcrypt && make && make install
```

```
[root@www ~]# yum -y install libxml2-devel libcurl-devel openssl-devel bzip2-devel
```

```
[root@www ~]# tar zxf php-5.6.27.tar.gz  
[root@www ~]# cd php-5.6.27/  
[root@www php-5.6.27]# ./configure --prefix=/usr/local/php5.6 --with-mysql=mysqlnd --with-pdo-mysql=mysqlnd --with-mysqli=mysqlnd --with-openssl --enable-fpm --enable-sockets --enable-sysvshm --enable-mbstring --with-freetype-dir --with-jpeg-dir --with-png-dir --with-zlib --with-libxml-dir=/usr --enable-xml --with-mhash --with-mcrypt=/usr/local/libmcrypt --with-config-file-path=/etc --with-config-file-scan-dir=/etc/php.d --with-bz2 --enable-main-tainer-zts  
[root@www php-5.6.27]# make&& make install
```

```
[root@www php-5.6.27]# cp php.ini-production /etc/php.ini  
修改/etc/php.ini文件，将short_open_tag修改为on，修改后的内容如下：  
short_open_tag = On //支持php短标签
```

2. 创建 php-fpm 服务启动脚本

[回目录](#)

```
[root@www php-5.6.27]# cp sapi/fpm/init.d.php-fpm /etc/init.d/php-fpm
[root@www php-5.6.27]# chmod +x /etc/init.d/php-fpm
[root@www php-5.6.27]# chkconfig --add php-fpm
[root@www php-5.6.27]# chkconfig php-fpm on
```

提供 `php-fpm` 配置文件并编辑

```
# cp /usr/local/php5.6/etc/php-fpm.conf.default /usr/local/php5.6/etc/php-fpm.conf
[root@www php-5.6.27]# vi /usr/local/php5.6/etc/php-fpm.conf
```

修改内容如下

```
pid = run/php-fpm.pid
listen =127.0.0.1:9000
pm.max_children = 300
pm.start_servers = 10
pm.min_spare_servers = 10
pm.max_spare_servers =50
```

启动 `php-fpm` 服务

```
[root@phpserver ~]# service php-fpm start
Starting php-fpm done
[root@www php-5.6.27]# netstat -anpt | grep php-fpm
tcp      0      0 127.0.0.1:9000      0.0.0.0:*      LISTEN      10937/php-fpm: mast
```

iii. 安装 `mysql` (在 **192.168.31.225** 主机操作)

[回目录](#)

因为 `centos7.2` 默认安装了 `mariadb-libs` , 所以先要卸载掉

1. 查看是否安装 mariadb

[回目录](#)

```
#rpm -qa | grep mariadb
```

卸载 mariadb

```
rpm -e --nodeps mariadb-libs
```

```
[root@localhost ~]# rpm -qa | grep mariadb
mariadb-libs-5.5.47-1.el7_2.x86_64
[root@localhost ~]# rpm -e mariadb-libs --nodeps
[root@localhost ~]#
```

2. 安装依赖包

[回目录](#)

注：相关依赖包的作用

- `cmake`：由于从 MySQL5.5 版本开始弃用了常规的 `configure` 编译方法，所以需要 CMake 编译器，用于设置 mysql 的编译参数。如：安装目录、数据存放目录、字符编码、排序规则等。
- `Boost` #从 MySQL 5.7.5 开始 Boost 库是必需的，mysql 源码中用到了 C++ 的 Boost 库，要求必须安装 boost1.59.0 或以上版本
- `GCC` 是Linux下的 C语言 编译工具，mysql 源码编译完全由 C 和 C++ 编写，要求必须安装 GCC
- `bison` : Linux下 C/C++ 语法分析器
- `ncurses` : 字符终端处理库

1) 安装文件准备

- 下载 `cmake-3.5.tar.gz` <http://www.cmake.org/download/>
- 下载 `ncurses-5.9.tar.gz` <ftp://ftp.gnu.org/gnu/ncurses/>
- 下载 `bison-3.0.4.tar.gz` <http://ftp.gnu.org/gnu/bison/>
- 下载 `mysql-5.7.13.tar.gz` `wget` <http://cdn.mysql.com/Downloads/MySQL-5.7/mysql-5.7.13.tar.gz>
- 下载 `Boost_1_59_0.tar.gz` `wget`

http://nchc.dl.sourceforge.net/project/boost/boost/1.59.0/boost_1_59_0.tar.gz

2) 安装 CMAKE 及必要的软件

安装 cmake

```
[root@mysqla ~]# tar zxf cmake-3.5.2.tar.gz  
[root@mysqla ~]# cd cmake-3.5.2/  
[root@mysqla cmake-3.5.2]# ./bootstrap
```

```
[root@mysqla cmake-3.5.2]# gmake && gmake install
```

cmake -version ---查看 cmake 版本

```
[root@mysqla cmake-3.5.2]# cmake -version  
cmake version 3.5.2  
  
CMake suite maintained and supported by Kitware (kitware.com/cmake) .
```

安装 ncurses

```
[root@mysqla ~]# tar zxf ncurses-5.9.tar.gz  
[root@mysqla ~]# cd ncurses-5.9/  
[root@mysqla ncurses-5.9]# ./configure && make && make install
```

安装 bison

安装 bootst

```
tar zxf boost_1_59_0.tar.gz  
mv boost_1_59_0 /usr/local/boost
```

3) 创建mysql用户和用户组及目录

```
# groupadd -r mysql && useradd -r -g mysql -s /bin/false -M mysql  
1---新建mysql组和mysql用户禁止登录shell  
# mkdir /usr/local/mysql ---创建目录  
# mkdir /usr/local/mysql/data ---数据库目录
```

3. 编译安装 mysql

[回目录](#)

解压 mysql 源码包

```
[root@mysqla ~]# tar zxf mysql-5.7.13.tar.gz
[root@mysqla ~]# cd mysql-5.7.13
```

执行 `cmake` 命令进行编译前的配置

```
[root@mysqla mysql-5.7.13]# cmake -DCMAKE_INSTALL_PREFIX=/usr/local/mysql -DMYSQL_DATADIR=/usr/local/mysql/data -DSYSCONFDIR=/etc -DDEFAULT_CHARSET=utf8 -DDEFAULT_COLLATION=utf8_general_ci -DEXTRA_CHARSETS=all -DMYSQL_UNIX_ADDR=/usr/local/mysql/mysql.sock -DWITH_MYISAM_STORAGE_ENGINE=1 -DWITH_INNODB_STORAGE_ENGINE=1 -DWITH_ARCHIVE_STORAGE_ENGINE=1 -DWITH_PARTITION_STORAGE_ENGINE=1 -DWITH_SYSTEMD=1 -DWITH_BOOST=/usr/local/boost
```

开始编译、编译安装：

```
[root@mysqla mysql-5.7.13]# make && make install
```

注1：配置解释：

- - DCMAKE_INSTALL_PREFIX=/usr/local/mysql [MySQL 安装的根目录]
- - DMYSQL_DATADIR=/usr/local/mysql /data [MySQL 数据库文件存放目录]
- - DSYSCONFDIR=/etc [MySQL 配置文件所在目录]
- - DWITH_MYISAM_STORAGE_ENGINE=1 [添加 MYISAM 引擎支持]
- - DWITH_INNODB_STORAGE_ENGINE=1 [添加 InnoDB 引擎支持]
- - DWITH_ARCHIVE_STORAGE_ENGINE=1 [添加 ARCHIVE 引擎支持]
- - DMYSQL_UNIX_ADDR=/usr/local/mysql /mysql.sock [指定 mysql.sock 位置]
- - DWITH_PARTITION_STORAGE_ENGINE=1 [安装支持数据库分区]
- - DEXTRA_CHARSETS=all [使 MySQL 支持所有的扩展字符]
- - DDEFAULT_CHARSET=utf8 [设置 MySQL 的默认字符集为 utf8]
- - DDEFAULT_COLLATION=utf8_general_ci [设置默认字符集校对规则]
- - DWITH_SYSTEMD=1 [可以使用 systemd 控制 mysql 服务]
- - DWITH_BOOST=/usr/local/boost [指向 boost 库所在目录]

更多参数执行

```
[root@localhost mysql-5.7.13]# cmake . -LH
```

注2：为了加快编译速度可以按下面的方式编译安装

```
[root@localhost mysql-5.7.13]# make -j $(grep processor /proc/cpuinfo | wc -l) && make install
```

```
make -j $(grep processor /proc/cpuinfo | wc -l)
```

-j 参数表示根据 CPU 核数指定编译时的线程数，可以加快编译速度。默认为 1 个线程编译。注3：若要重新运行 cmake 配置，需要删除 CMakeCache.txt 文件

```
# make clean  
# rm -f CMakeCache.txt
```

优化 Mysql 的执行路径

```
[root@mysqla mysql-5.7.13]# tail -1 /etc/profile  
export PATH=$PATH:/usr/local/mysql/bin  
[root@mysqla mysql-5.7.13]# source /etc/profile  
[root@mysqla mysql-5.7.13]#
```

4. 设置权限并初始化 MySQL 系统授权表

[回目录](#)

```
# cd/usr/local/mysql  
# chown -R mysql:mysql .           ---更改所有者,属组,注意是mysql .  
#bin/mysqld --initialize--user=mysql --basedir=/usr/local/mysql  
--datadir=/usr/local/mysql/data
```

注1：以root初始化操作时要加 --user=mysql 参数，生成一个随机密码（注意保存登录时用）注2： MySQL 5.7.6 之前的版本执行这个脚本初始化系统数据库

```
/usr/local/mysql/bin/mysql_install_db --user=mysql --basedir=/us  
r/local/mysql --datadir=/usr/local/mysql/data
```

5.7.6 之后版本初始系统数据库脚本（本文使用此方式初始化）

```
#!/usr/local/mysql/bin/mysqld --initialize-insecure--user=mysql -  
-basedir=/usr/local/mysql --datadir=/usr/local/mysql/data
```

```
[root@mysqla mysql]# bin/mysqlld --initialize --user=mysql --basedir=/usr/local/mysql --datadir=/usr/local/mysql/data
2016-08-27T15:42:47.961744Z 0 [Warning] TIMESTAMP with implicit DEFAULT value is deprecated. Please use --explicit_defaults_for_timestamp server option (see documentation for more details).
2016-08-27T15:42:49.288990Z 0 [Warning] InnoDB: New log files created, LSN=45790
2016-08-27T15:42:49.428124Z 0 [Warning] InnoDB: Creating foreign key constraint system tables.
2016-08-27T15:42:49.488286Z 0 [Warning] No existing UUID has been found, so we assume that this is the first time that this server has been started. Generating a new UUID: e8823be-6c6c-11e6-8254-000c29e2c7fa.
2016-08-27T15:42:49.622253Z 0 [Warning] Gtid table is not ready to be used. Table 'mysql.gtid_executed' cannot be opened.
2016-08-27T15:42:49.623091Z 1 [Note] A temporary password is generated for root@localhost: I27rY/<BLM8/
```

注意：如果使用 `--initialize` 参数初始化系统数据库之后，会生成 `root` 用户的一个临时密码，如上图高亮中所示。

```
# chown -Rmysql:mysql .           ---改所有者，注意是root .
```

5. 创建配置文件

[回目录](#)

```
# cd/usr/local/mysql/support-files      ---进入MySQL安装目录支持文件
目录
# cp my-default.cnf /etc/my.cnf       ---复制模板为新的配置文件，
```

```
[root@mysqla mysql]# cd /root/mysql-5.7.13/support-files/
[root@mysqla support-files]# cp my-default.cnf /etc/my.cnf
```

修改文件中配置选项，如下图所示，添加如下配置项

```
#vi /etc/my.cnf
```

```
basedir = /usr/local/mysql
datadir = /usr/local/mysql/data
port = 3306
server_id = 1
socket = /usr/local/mysql/mysql.sock
log-error=/usr/local/mysql/data/mysqld.err
```

6. 配置 mysql 自动启动

回目录

```
[root@mysqla ~]# cp /usr/local/mysql/usr/lib/systemd/system/mysqld.service /usr/lib/systemd/system/
[root@mysqla ~]# systemctl enable mysqld.service
Created symlink from /etc/systemd/system/multi-user.target.wants/mysqld.service to /usr/lib/systemd/system/mysqld.service.
```

```
[root@mysqla mysql]# systemctl start mysqld.service
Job for mysqld.service failed because the control process exited with error code. See "systemctl status mysqld.service" and "journalctl -xe" for details.
```

服务启动失败，查看错误日志文件

```
[root@mysqla ~]# tail /usr/local/mysql/data/mysqld.err
2016-08-27T16:29:14.663708Z 0 [Note] InnoDB: Loading buffer pool(s) from /usr/local/mysql/data/ib_buffer_pool
2016-08-27T16:29:14.664316Z 0 [Note] Plugin 'FEDERATED' is disabled.
2016-08-27T16:29:14.666380Z 0 [Note] InnoDB: Buffer pool(s) load completed at 160828 0 :29:14
2016-08-27T16:29:14.666805Z 0 [Warning] Failed to set up SSL because of the following S
SL library error: SSL context is not usable without certificate and private key
2016-08-27T16:29:14.666837Z 0 [Note] Server hostname (bind-address): '*' ; port: 3306
2016-08-27T16:29:14.667031Z 0 [Note] IPv6 is available.
2016-08-27T16:29:14.667052Z 0 [Note] - '::' resolves to '::';
2016-08-27T16:29:14.667088Z 0 [Note] Server socket created on IP: '::'.
2016-08-27T16:29:14.668973Z 0 [ERROR] /usr/local/mysql/bin/mysqld: Can't create/write t
o file '/var/run/mysqld/mysqld.pid' (Errcode: 2 - No such file or directory)
2016-08-27T16:29:14.668998Z 0 [ERROR] Can't start server: can't create PID file: No suc
h file or directory
```

在 `mysqld.service`，把默认的 `pid` 文件指定到了 `/var/run/mysqld/` 目录，而并没有事先建立该目录，因此要手动建立该目录并把权限赋给 `mysql` 用户。

```
[root@mysqla ~]# mkdir /var/run/mysqld
[root@mysqla ~]# chown -R mysql:mysql /var/run/mysqld/
```

或者修改 `/usr/lib/systemd/system/mysqld.service`，修改内容如下：

```
#PIDFile=/var/run/mysqld/mysqld.pid
PIDFile=/usr/local/mysql/mysqld.pid
# Disable service start and stop timeout logic of systemd for mysqld service.
TimeoutSec=0

# Execute pre and post scripts as root
PermissionsStartOnly=true

# Needed to create system tables
ExecStartPre=/usr/local/mysql/bin/mysqld_pre_systemd

# Start main service
#ExecStart=/usr/local/mysql/bin/mysqld --daemonize --pid-file=/var/run/mysqld/mysqld.pi
d $MYSQLD_OPTS
ExecStart=/usr/local/mysql/bin/mysqld --daemonize --pid-file=/usr/local/mysql/mysqld.pi
d $MYSQLD_OPTS
```

```
#systemctl daemon-reload
```

再次启动 mysql 服务

```
[root@mysqla ~]# systemctl start mysqld.service
[root@mysqla ~]# systemctl status mysqld.service
● mysqld.service - MySQL Server
   Loaded: loaded (/usr/lib/systemd/system/mysqld.service; enabled; vendor preset: disabled)
     Active: active (running) since Sun 2016-08-28 00:36:45 CST; 6s ago
       Process: 100069 ExecStart=/usr/local/mysql/bin/mysqld --daemonize --pid-file=/var/run/mysqld/mysqld.pid $MYSQLD_OPTS (code=exited, status=0/SUCCESS)
      Process: 100051 ExecStartPre=/usr/local/mysql/bin/mysqld_pre_systemd (code=exited, status=0/SUCCESS)
     Main PID: 100072 (mysqld)
        CGroup: /system.slice/mysqld.service
                  └─100072 /usr/local/mysql/bin/mysqld --daemonize --pid-file=/var/run/mysq...

Aug 28 00:36:45 mysqla systemd[1]: Starting MySQL Server...
Aug 28 00:36:45 mysqla systemd[1]: Started MySQL Server.
```

查看端口号

```
[root@mysqla ~]# netstat -anpt | grep mysqld
tcp6      0      0 :::3306          ::::*                      LISTEN      100072
```

服务启动成功

访问 MySQL 数据库

```
# mysql -u root -h 127.0.0.1 -p      --- 连接mysql，输入初始化时生成的随机密码
```

```
[root@mysqla ~]# mysql -u root -h 127.0.0.1 -p'I27rY/<BLM8/'
```

设置数据库管理员用户 root 的密码

```
[root@mysqla ~]# mysqladmin -u root -p password '123.abc'
Enter password:
mysqladmin: [Warning] Using a password on the command line interface can be insecure.
Warning: Since password will be sent to server in plain text, use ssl connection to ensure password safety.
```

iv. 安装 memcached 服务端 (在 192.168.31.250 主机操作)

回目录

memcached 是基于 libevent 的事件处理。libevent 是个程序库，它将 Linux 的 epoll 、BSD 类操作系统的 kqueue 等事件处理功能封装成统一的接口。即使对服务器的连接数增加，也能发挥 I/O 的性能。memcached 使用这

个 libevent 库，因此能在 Linux 、 BSD 、 Solaris 等操作系统上发挥其高性能。

1.首先先安装 memcached 依赖库 libevent

[回目录](#)

```
[root@memcache ~]# tar zxf libevent-2.0.22-stable.tar.gz  
[root@memcache ~]# cd libevent-2.0.22-stable/  
[root@memcache libevent-2.0.22-stable]# ./configure  
[root@memcache libevent-2.0.22-stable]# make&& make install
```

编译安装的时候没有指定路径，那么默认路径就是 /usr/local

2.安装 memcached

[回目录](#)

```
[root@memcache ~]# tar zxf memcached-1.4.33.tar.gz  
[root@memcache ~]# cd memcached-1.4.33/  
[root@memcache memcached-1.4.33]# ./configure --prefix=/usr/local  
memcached --with-libevent=/usr/local  
[root@memcache memcached-1.4.33]# make&& make install
```

libevent 默认的安装路径是在 /usr/local ，所以指定此路径

3.检测是否成功安装

[回目录](#)

```
[root@memcache ~]# ls /usr/local/memcached/bin/memcached  
/usr/local/memcached/bin/memcached
```

通过以上操作就很简单的把 memcached 服务端编译好了。这时候就可以打开服务端进行工作了。配置环境变量：

进入用户宿主目录，编辑 `.bash_profile`，为系统环境变量 `LD_LIBRARY_PATH` 增加新的目录，需要增加的内容如下：

```
[root@memcache ~]# vi ~/.bash_profile
MEMCACHED_HOME=/usr/local/memcached
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MEMCACHED_HOME/lib
[root@memcache ~]# /usr/local/memcached/bin/memcached -d -m 2048
-l 192.168.31.250 -p 11211 -u root -c 10240 -P /usr/local/memca
ched/memcached.pid
```

启动参数说明

参数	作用
-d	选项是启动一个守护进程。
-m	分配给 Memcache 使用的内存数量，单位是 MB，默认 64MB。
-l	监听的 IP地址。 (默认： INADDR_ANY，所有地址)
-p	设置 Memcache 的 TCP 监听的端口，最好是 1024 以上的端口。
-u	运行 Memcache 的用户，如果当前为 root 的话，需要使用此参数指定用户。
-c	选项是最大运行的并发连接数，默认是 1024。
-P	设置保存 Memcache 的 pid 文件。
-M	内存耗尽时返回错误，而不是删除项
-f	块大小增长因子，默认是 1.25
-n	最小分配空间，key + value + flags 默认是 48
-h	显示帮助

```
[root@memcache ~]# netstat -anpt |grep memcached
tcp    0    0 192.168.31.250:11211      0.0.0.0:*      LISTEN      12840/
memcached
```

4. 设置防火墙

[回目录](#)

```
[root@memcache ~]# firewall-cmd --permanent --add-port=11211/tcp  
success  
[root@memcache ~]# firewall-cmd --reload  
success
```

5.刷新用户环境变量

[回目录](#)

```
[root@memcache ~]# source ~/.bash_profile
```

6.编写 memcached 服务启停脚本

[回目录](#)

```
[root@memcache ~]# vi /etc/init.d/memcached
```

脚本内容如下

```
[root@memcache ~]# cat /etc/init.d/memcached  
#!/bin/sh  
  
#  
# pidfile: /usr/local/memcached/memcached.pid  
# memcached_home: /usr/local/memcached  
# chkconfig: 35 21 79  
# description: Start and stop memcached Service  
  
# Source function library  
. /etc/rc.d/init.d/functions  
  
RETVAL=0  
  
prog="memcached"  
basedir=/usr/local/memcached
```

```

cmd=${basedir}/bin/memcached
pidfile="$basedir/${prog}.pid"

#interface to listen on (default: INADDR_ANY, all addresses)
ipaddr="192.168.31.250"
#listen port
port=11211
#username for memcached
username="root"
#max memory for memcached, default is 64M
max_memory=2048
#max connections for memcached
max_simul_conn=10240
start() {
echo -n $"Starting service: $prog"
$cmd -d -m $max_memory -u $username -l $ipaddr -p $port -c $max_
simul_conn -P $pidfile
RETVAL=$?
echo
[ $RETVAL -eq 0 ] && touch /var/lock/subsys/$prog
}

stop() {
echo -n $"Stopping service: $prog"
run_user=$(whoami)
pidlist=$(ps -ef | grep $run_user | grep memcached | grep -v gre
p | awk '{print($2)}')
for pid in $pidlist
do
kill -9 $pid
if [ $? -ne 0 ]; then
return 1
fi
done
RETVAL=$?
echo
[ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/$prog
}

# See how we were called.

```

```
case "$1" in
start)
start
;;
stop)
stop
;;
restart)
stop
start
;;
*)
echo "Usage: $0 {start|stop|restart|status}"
exit 1
esac
exit $RETVAL
```

设置脚本可被执行

```
[root@memcache ~]# chmod +x /etc/init.d/memcached
[root@memcache ~]# chkconfig --add memcached
[root@memcache ~]# chkconfig memcached on
```

说明：

shell 脚本中 return 的作用

- 1) 终止一个函数.
- 2) return 命令允许带一个整型参数, 这个整数将作为函数的"退出状态码"返回给调用这个函数的脚本, 并且这个整数也被赋值给变量 \$? .
- 3) 命令格式： return value

7. 配置 `nginx.conf` 文件 (在 nginx 主机操作)

[回目录](#)

配置内容如下：

```
user www www;
```

```
worker_processes 4;
worker_cpu_affinity 0001 0010 0100 1000;
error_log logs/error.log;
#error_log logs/error.log notice;
#error_log logs/error.log info;

pid        logs/nginx.pid;

events {
    use epoll;
    worker_connections 65535;
    multi_accept on;
}

http {
    include      mime.types;
    default_type application/octet-stream;

    #log_format main '$remote_addr - $remote_user [$time_local]
] "$request"
#                                '$status $body_bytes_sent "$http_referer"
'
#                                '"$http_user_agent" "$http_x_forwarded_for"';
    access_log  logs/access.log  main;

    sendfile      on;
    tcp_nopush    on;
    keepalive_timeout 65;
    tcp_nodelay on;
    client_header_buffer_size 4k;
    open_file_cache max=102400 inactive=20s;
    open_file_cache_valid 30s;
    open_file_cache_min_uses 1;
    client_header_timeout 15;
    client_body_timeout 15;
    reset_timedout_connection on;
    send_timeout 15;
    server_tokens off;
```

```
client_max_body_size 10m;

fastcgi_connect_timeout      600;
fastcgi_send_timeout 600;
fastcgi_read_timeout 600;
fastcgi_buffer_size 64k;
fastcgi_buffers    4 64k;
fastcgi_busy_buffers_size 128k;
fastcgi_temp_file_write_size 128k;
fastcgi_temp_path /usr/local/nginx1.10/nginx_tmp;
fastcgi_intercept_errors on;
fastcgi_cache_path /usr/local/nginx1.10/fastcgi_cache levels
=1:2
keys_zone=cache_fastcgi:128m inactive=1d max_size=10g;

gzip on;
gzip_min_length  2k;
gzip_buffers     4 32k;
gzip_http_version 1.1;
gzip_comp_level 6;
gzip_types   text/plain text/css text/javascript application/
json
application/javascript application/x-javascript application/
xml;
gzip_vary on;
gzip_proxied any;

server {
    listen      80;
    server_name www.benet.com;

#charset koi8-r;

#access_log  logs/host.access.log  main;

location ~* ^.+\.(jpg|gif|png|swf|flv|wma|wmv|ASF|mp3|mmf|zip|r
r)$ {
valid_referers none blocked www.benet.com benet.com;
if ($invalid_referer) {
#return 302 http://www.benet.com/img/nolink.jpg;
```

```
    return 404;
    break;
}
access_log off;
}
location / {
    root    html;
    index   index.php index.html index.htm;
}
location ~* \.(ico|jpe?g|gif|png|bmp|swf|flv)$ {
    expires 30d;
    #log_not_found off;
    access_log off;
}

location ~* \.(js|css)$ {
    expires 7d;
    log_not_found off;
    access_log off;
}

location = /(favicon.ico|robots.txt) {
    access_log off;
    log_not_found off;
}
location /status {
    stub_status on;
}
location ~ .*\.(php|php5)?$ {
    root html;
    fastcgi_pass 127.0.0.1:9000;
    fastcgi_index index.php;
    include fastcgi.conf;
    fastcgi_cache cache_fastcgi;
    fastcgi_cache_valid 200 302 1h;
    fastcgi_cache_valid 301 1d;
    fastcgi_cache_valid any 1m;
    fastcgi_cache_min_uses 1;
    fastcgi_cache_use_stale error timeout invalid_header
http_500;
```

```

        fastcgi_cache_key http://$host$request_uri;
    }
    #error_page 404 /404.html;
    # redirect server error pages to the static page /50x.htm
m1
#
    error_page 500 502 503 504 /50x.html;
location = /50x.html {
    root html;
}
}
}
}

```

重启 nginx 服务生成一个 php 测试页

```
[root@www memcache-3.0.8]# cat /usr/local/nginx1.10/html/test1.php
<?php
phpinfo();
?>
```

使用浏览器访问 test1.php 测试页

Directive	Local Value	Master Value
date.default_latitude	31.7667	31.7667
date.default_longitude	35.2333	35.2333
date.sunrise_zenith	90.583333	90.583333
date.sunset_zenith	90.583333	90.583333
date.timezone	no value	no value

v. memcache 客户端（在 php 服务器操作）：

[回目录](#)

memcache 分为服务端和客户端。服务端用来存放缓存，客户端用来操作缓存。
安装php扩展库（ phpmemcache ）。

1. 安装 PHP Memcache 扩展

[回目录](#)

可以使用 `php` 自带的 `pecl` 安装程序

```
# /usr/local/servers/php/bin/pecl install memcache
```

也可以从源码安装,他是生成 `php` 的扩展库文件 `memcache.so` 。

2. 安装 `memcache` 扩展库

[回目录](#)

```
[root@www ~]# tar zxf memcache-3.0.8.tgz
[root@www ~]# cd memcache-3.0.8/
[root@www memcache-3.0.8]# /usr/local/php5.6/bin/phpize
[root@www memcache-3.0.8]# ./configure --enable-memcache --with-php-config=/usr/local/php5.6/bin/php-config
[root@www memcache-3.0.8]# make&& make install
```

安装完后会有类似这样的提示：

```
Installing shared extensions:      /usr/local/php5.6/lib/php/extensions/no-debug-zts-20131226/
```

把这个记住，然后修改 `php.ini`

添加一行

```
extension=/usr/local/php5.6/lib/php/extensions/no-debug-zts-20131226/memcache.so
```

重启 `php-fpm` 服务

```
[root@www memcache-3.0.8]# service php-fpm restart
Gracefully shutting down php-fpm .done
Starting php-fpm done
```

测试：

检查 php 扩展是否正确安装

1、 [root@www html]# /usr/local/php5.6/bin/php -m 命令行执行 php -m 查询结果中是否有 memcache 项 2、创建 phpinfo() 页面，查询 session 项下面的 Registered save handlers 值中是否有 memcache 项

浏览器访问 test1.php

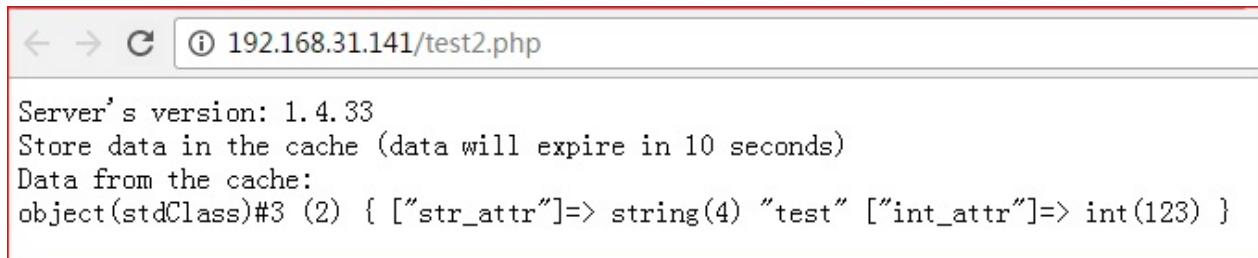
memcache	
memcache support	enabled
Version	3.0.8
Revision	\$Revision: 329835 \$

session	
Session Support	enabled
Registered save handlers	files user memcache
Registered serializer handlers	php_serialize php php_binary

测试代码

```
[root@www ~]# cat /usr/local/nginx1.10/html/test2.php
<?php
$memcache = new Memcache;
$memcache->connect('192.168.31.250', 11211) or die ("Could not connect");
$version = $memcache->getVersion();
echo "Server's version: ".$version."<br/>";
$tmp_object = new stdClass;
$tmp_object->str_attr = 'test';
$tmp_object->int_attr = 123;
$memcache->set('key', $tmp_object, false, 10) or die ("Failed to save data at the server");
echo "Store data in the cache (data will expire in 10 seconds)<br/>";
$get_result = $memcache->get('key');
echo "Data from the cache:<br/>";
var_dump($get_result);
?>
```

浏览器访问 test2.php



The screenshot shows a browser window with the URL 192.168.31.141/test2.php. The page content displays the following output:

```
Server's version: 1.4.33
Store data in the cache (data will expire in 10 seconds)
Data from the cache:
object(stdClass)#3 (2) { ["str_attr"]=> string(4) "test" ["int_attr"]=> int(123) }
```

使用 memcache 实现 session 共享

配置 `php.ini` 中的 `Session` 为 `memcache` 方式。

```
session.save_handler = memcache
session.save_path = "tcp://192.168.31.250:11211?persistent=1
&weight=1&timeout=1&retry_interval=15"
```

注：

- `session.save_handler`：设置 `session` 的储存方式为 `memcache`。默认以文件方式存取 `session` 数据，如果想要使用自定义的处理来存取 `session` 数据，比如 `memcache` 方式则修为 `session.save_handler = memcache`
- `session.save_path`：设置 `session` 储存的位置，多台 `memcache` 用逗号隔开 使用多个 `memcached server` 时用逗号，隔开，可以带额外的参数 `persistent`、`weight`、`timeout`、`retry_interval` 等等，类似这样的：`tcp://host:port?persistent=1&weight=2,tcp://host2:port2`。

memcache 实现 session 共享也可以在某个一个应用中设置：

```
ini_set ("session.save_handler", "memcache");
ini_set ("session.save_path", "tcp://192.168.0.9:11211"); ini_set() 只对当前php页面有效，并且不会去修改 php.ini 文件本身，也不会影响其他php页面。
```

3. 测试 memcache 可用性

[回目录](#)

在 web 服务器上新建 `/usr/local/nginx1.10/html/memcache.php` 文件。

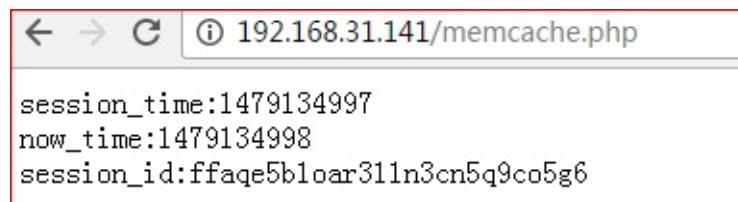
内容如

```

<?php
session_start();
if (!isset($_SESSION['session_time']))
{
    $_SESSION['session_time'] = time();
}
echo "session_time:". $_SESSION['session_time']. "<br />";
echo "now_time:". time(). "<br />";
echo "session_id:". session_id(). "<br />";
?>

```

访问网址 `http://192.168.31.141/memcache.php` 可以查看 `session_time` 是否都是为 `memcache` 中的 `Session`，同时可以在不同的服务器上修改不同的标识查看是否为不同的服务器上的。



可以直接用 `sessionid` 去 `memcached` 里查询一下

```

[root@www html]# telnet 192.168.31.250 11211
Trying 192.168.31.250...
Connected to 192.168.31.250.
Escape character is '^].
get ffaqe5b1oar311n3cn5q9co5g6
VALUE ffaqe5b1oar311n3cn5q9co5g6 0 26
session_time|i:1479134997;

```

得到 `session_time|i : 1479134997` ;这样的结果，说明 `session` 正常工作，默认 `memcache` 会监听 `11221` 端口，如果想清空服务器上 `memecache` 的缓存，一般使用的是：

```
[root@memcache ~]# telnet 192.168.31.250 11211
Trying 192.168.31.250...
Connected to 192.168.31.250.
Escape character is '^]'.
flush_all
OK
```

同样也可以使用

```
[root@memcache ~]# echo "flush_all" | nc 192.168.31.250 11211
OK
```

使用 `flush_all` 后并不是删除 memcache 上的 key ，而是置为过期 memcache 安全配置因为 memcache 不进行权限控制，因此需要通过 iptables 将 memcache 仅开放给 web 服务器。

vi. 测试 memcache 缓存数据库数据

[回目录](#)

1. 在 Mysql 服务器上创建测试表

[回目录](#)

```

mysql> create database testdb1;
Query OK, 1 row affected (0.00 sec)

mysql> use testdb1;
Database changed
mysql> create table test1(id int not null auto_increment,name va
rchar(20) default null,primary key (id)) engine=innodb auto_incr
ement=1 default charset=utf8;
Query OK, 0 rows affected (0.03 sec)

mysql> insert into test1(name) values ('tom1'),('tom2'),('tom3')
,('tom4'),('tom5');
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from test1;
+----+----+
| id | name |
+----+----+
| 1 | tom1 |
| 2 | tom2 |
| 3 | tom3 |
| 4 | tom4 |
| 5 | tom5 |
+----+----+
5 rows in set (0.00 sec)

```

测试

下面就是测试的工作了，这里有个 php 脚本，用于测试 memcache 是否缓存数据成功

需要为这个脚本添加一个只读的数据库用户，命令格式

```

mysql> grant select on testdb1.* to user@'%' identified by '1234
56';
Query OK, 0 rows affected, 1 warning (0.00 sec)

```

2. 在**web**服务器上创建测试脚本内容如下

[回目录](#)

```
[root@www html]# cat /usr/local/nginx1.10/html/test_db.php
<?php
$memcachehost = '192.168.31.250';
$memcacheport = 11211;
$memcachelife = 60;
$memcache = new Memcache;
$memcache->connect($memcachehost,$memcacheport) or die ("Could n
ot connect");
$query="select * from test1 limit 10";
$key=md5($query);
if(! $memcache->get($key))
{
    $conn=mysql_connect("192.168.31.225","user","123
456");
    mysql_select_db(testdb1);
    $result=mysql_query($query);
    while ($row=mysql_fetch_assoc($result))
    {
        $arr[]=$row;
    }
    $f = 'mysql';
    $memcache->add($key,serialize($arr),0,30);
    $data = $arr ;
}
else{
    $f = 'memcache';
    $data_mem=$memcache->get($key);
    $data = unserialize($data_mem);
}
echo $f;
echo "<br>";
echo "$key";
echo "<br>";
//print_r($data);
foreach($data as $a)
{
```

```

echo "number is <b><font color=#FF0000>$a[id]</font></b>" ;
echo "<br>";
echo "name is <b><font color=#FF0000>$a[name]</font></b>" ;
echo "<br>";
}
?>

```

3. 访问页面测试

[回目录](#)

```

mysql
d8c961e9895ba4b463841924dbcefc2b
number is 1
name is tom1
number is 2
name is tom2
number is 3
name is tom3
number is 4
name is tom4
number is 5
name is tom5

```

如果出现 `mysql` 表示 memcached 中没有内容，需要 memcached 从数据库中取得再刷新页面，如果有 `memcache` 标志表示这次的数据是从 memcached 中取得的。 memcached 有个缓存时间默认是1分钟，过了一分钟后， memcached 需要重新从数据库中取得数据

```

memcache
d8c961e9895ba4b463841924dbcefc2b
number is 1
name is tom1
number is 2
name is tom2
number is 3
name is tom3
number is 4
name is tom4
number is 5
name is tom5

```

4. 查看 Memcached 缓存情况

[回目录](#)

需要使用 telnet 命令查看

```
[root@memcache ~]# telnet 192.168.31.250 11211
Trying 192.168.31.250...
Connected to 192.168.31.250.
Escape character is '^]'.
stats
STAT pid 1681                                //Memcached 进程的ID
STAT uptime 8429                               //进程运行时间
STAT time 1479142306                          //当前时间
STAT version 1.4.33                            // Memcached 版本
STAT libevent 2.0.22-stable
STAT pointer_size 64
STAT rusage_user 1.218430
STAT rusage_system 1.449512
STAT curr_connections 5
STAT total_connections 32
STAT connection_structures 10
STAT reserved_fds 20
STAT cmd_get 25//总共获取数据的次数(等于 get_hits + get_misses )
STAT cmd_set 19 //总共设置数据的次数
STAT cmd_flush 4
STAT cmd_touch 0
STAT get_hits 15//命中了多少次数据，也就是从 Memcached 缓存中成功获取数据的次数
STAT get_misses 10//没有命中的次数
STAT get_expired 3
STAT get_flushed 1
STAT delete_misses 0
STAT delete_hits 0
STAT incr_misses 2
STAT incr_hits 2
STAT decr_misses 0
STAT decr_hits 0
STAT cas_misses 0
STAT cas_hits 0
```

```
STAT cas_badval 0
STAT touch_hits 0
STAT touch_misses 0
STAT auth_cmds 0
STAT auth_errors 0
STAT bytes_read 3370
STAT bytes_written 15710
STAT limit_maxbytes 2147483648//总的存储大小，默认为 64M
STAT accepting_conns 1
STAT listen_disabled_num 0
STAT time_in_listen_disabled_us 0
STAT threads 4
STAT conn_yields 0
STAT hash_power_level 16
STAT hash_bytes 524288
STAT hash_is_expanding 0
STAT malloc_fails 0
STAT log_worker_dropped 0
STAT log_worker_written 0
STAT log_watcher_skipped 0
STAT log_watcher_sent 0
STAT bytes 584//当前所用存储大小
STAT curr_items 3
STAT total_items 17
STAT expired_unfetched 2
STAT evicted_unfetched 0
STAT evictions 0
STAT reclaimed 4
STAT crawler_reclaimed 0
STAT crawler_items_checked 0
STAT lrutail_reflocked 0
END
```

命中率= get_hits/ cmd_get

[回目录](#)

redis

目录

- I.redis介绍
 - i. 单地比较 Redis 与 Memcached 的区别
 - ii. memcached 和 redis 的比较
 - 1. 网络 IO 模型
 - 2. 内存管理方面
 - 3. 存储方式及其它方面
- II. 如何保持 session 会话
 - i. 请求精确定位
 - ii. session 复制共享
 - iii. 基于 cache DB 缓存的 session 共享
- III. nginx + tomcat + redis 实现负载均衡、 session 共享
 - i. 实验环境
 - ii. 实验拓扑
 - iii. nginx 安装配置
 - 1. 安装 nginx
 - 2. 编写 nginx 服务脚本
 - 3. 配置 nginx 反向代理：反向代理+负载均衡+健康探测
 - iv. 安装部署 tomcat 应用程序服务器
 - 1. 安装 JDK，配置 java 环境
 - 1.2 tomcat 环境变量
 - 1.3 java version
 - 2. 在 tomcat-1 和 tomcat-2 节点安装配置 tomcat
 - 2.1 配置 tomcat 环境变量
 - 2.2 防火墙规则配置
 - v. 安装 redis
 - 1. 解压安装 redis
 - 2. 初始化 redis
 - 3. systemd
 - 4. 启动 redis

- 5. 防火墙规则设置
- 6. redis version
- 7. 最终 redis 配置文件如下
- 8. 重新启动 redis 服务
- vi. 配置 tomcat session redis 同步
 - 1. 下载 tomcat-redis-session-manager 相应的 jar 包，主要有三个：
 - 2. 修改 tomcat 的 context.xml
 - 3. 重启 tomcat 服务
- vii. tomcat 连接数据库
 - 1. 192.168.31.225 作为 mysql 数据库服务器
 - 2. 插入些数据
 - 3. mysql 防火墙配置
 - 4. tomcat 连接 mysql
 - 5. 在 <Context> 中添加如下内容
 - 6. test.jsp

I.redis 介绍

[回目录](#)

redis 是一个 key-value 存储系统。和 Memcached 类似，它支持存储的 value 类型相对更多，包括 string (字符串)、 list (链表)、 set (集合)、 zset (sorted set --有序集合) 和 hash (哈希类型)。与 memcached 一样，为了保证效率，数据都是缓存在内存中。区别的是 redis 会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现 master-slave (主从) 同步。

Redis 是一个高性能的 key-value 数据库。 redis 的出现，很大程度补偿了 memcached 这类 key/value 存储的不足，在部分场合可以对关系数据库起到很好的补充作用。它提供

了 Java , C/C++ , C# , PHP , JavaScript , Perl , Object-C , Python , Ruby 等客户端，使用很方便。

i. 单地比较 **Redis** 与 **Memcached** 的区别

[回目录](#)

基本上有以下3点

- 1、Redis不仅仅支持简单的 k/v 类型的数据，同时还提供 list , set , zset , hash 等数据结构的存储。
- 2、Redis支持数据的备份，即 master-slave 模式的数据备份。
- 3、Redis支持数据的持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用。

在Redis中，并不是所有的数据都一直存储在内存中的。这是和Memcached相比一个最大的区别。Redis只会缓存所有的key的信息，如果Redis发现内存的使用量超过了某一个阀值，将触发 swap 的操作，Redis根据 swappability = age*log(size_in_memory) 计算出哪些 key 对应的 value 需要swap到磁盘。然后再将这些key对应的value持久化到磁盘中，同时在内存中清除。这种特性使得Redis可以保持超过其机器本身内存大小的数据。当然，机器本身的内存必须要能够保持所有的key，因为这些数据是不会进行swap操作的。

当从Redis中读取数据的时候，如果读取的 key 对应的 value 不在内存中，那么 Redis 就需要从 swap 文件中加载相应数据，然后再返回给请求方。

ii. memcached 和 redis 的比较

[回目录](#)

1. 网络 IO 模型

[回目录](#)

Memcached是多线程，非阻塞 IO 复用的网络模型，分为监听主线程和 worker 子线程，监听线程监听网络连接，接受请求后，将连接描述字 pipe 传递给 worker 线程，进行读写 IO，网络层使用 libevent 封装的事件库，多线程模型可以发挥多核作用。

Redis 使用单线程的 IO 复用模型，自己封装了一个简单的 AeEvent 事件处理框架，主要实现了 epoll 、 kqueue 和 select ，对于单纯只有 IO 操作来说，单线程可以将速度优势发挥到最大，但是 Redis 也提供了一些简单的计算功能，比如排序、聚合等，对于这些操作，单线程模型实际会严重影响整体吞吐量，CPU 计算过程中，整个 IO 调度都是被阻塞住的。

2. 内存管理方面

[回目录](#)

Memcached 使用预分配的内存池的方式，使用 slab 和大小不同的 chunk 来管理内存， value 根据大小选择合适的 chunk 存储。Redis 使用现场申请内存的方式来存储数据。

3. 存储方式及其它方面

[回目录](#)

Memcached 基本只支持简单的 key-value 存储，不支持持久化和复制等功能，Redis 除 key/value 之外，还支持 list , set , sorted set , hash 等众多数据结构

II. 如何保持 session 会话

[回目录](#)

目前，为了使 web 能适应大规模的访问，需要实现应用的集群部署。集群最有效的方案就是负载均衡，而实现负载均衡用户每一个请求都有可能被分配到不固定的服务器上，这样我们首先要解决 session 的统一来保证无论用户的请求被转发到哪个服务器上都能保证用户的正常使用，即需要实现 session 的共享机制。

在集群系统下实现 session 统一的有如下几种方案

i. 请求精确定位

[回目录](#)

`session sticky`，例如基于访问 ip 的 hash 策略，即当前用户的请求都集中定位到一台服务器中，这样单台服务器保存了用户的 session 登录信息，如果宕机，则等同于单点部署，会丢失，会话不复制。

ii. session 复制共享

[回目录](#)

`session replication`，如 tomcat 自带 session 共享，主要是指集群环境下，多台应用服务器之间同步 session，使 session 保持一致，对外透明。如果其中一台服务器发生故障，根据负载均衡的原理，调度器会遍历寻找可用节点，分发请求，由于 session 已同步，故能保证用户的 session 信息不会丢失，会话复制。

此方案的不足之处

必须在同一种中间件之间完成(如: tomcat-tomcat 之间). session 复制带来的性能损失会快速增加.特别是当 session 中保存了较大的对象,而且对象变化较快时,性能下降更加显著,会消耗系统性能。这种特性使得 web 应用的水平扩展受到了限制。Session 内容通过广播同步给成员,会造成网络流量瓶颈,即便是内网瓶颈。在大并发下表现并不好

iii. 基于 cache DB 缓存的 session 共享

[回目录](#)

基于 memcache/redis 缓存的 session 共享

即使使用 cacheDB 存取 session 信息，应用服务器接受新请求将 session 信息保存在 cache DB 中，当应用服务器发生故障时，调度器会遍历寻找可用节点，分发请求，当应用服务器发现 session 不在本机内存时，则去 cache DB 中查找，如果找到则复制到本机，这样实现 session 共享和高可用。

III. nginx + tomcat + redis 实现负载均

衡、 session 共享

[回目录](#)

i. 实验环境

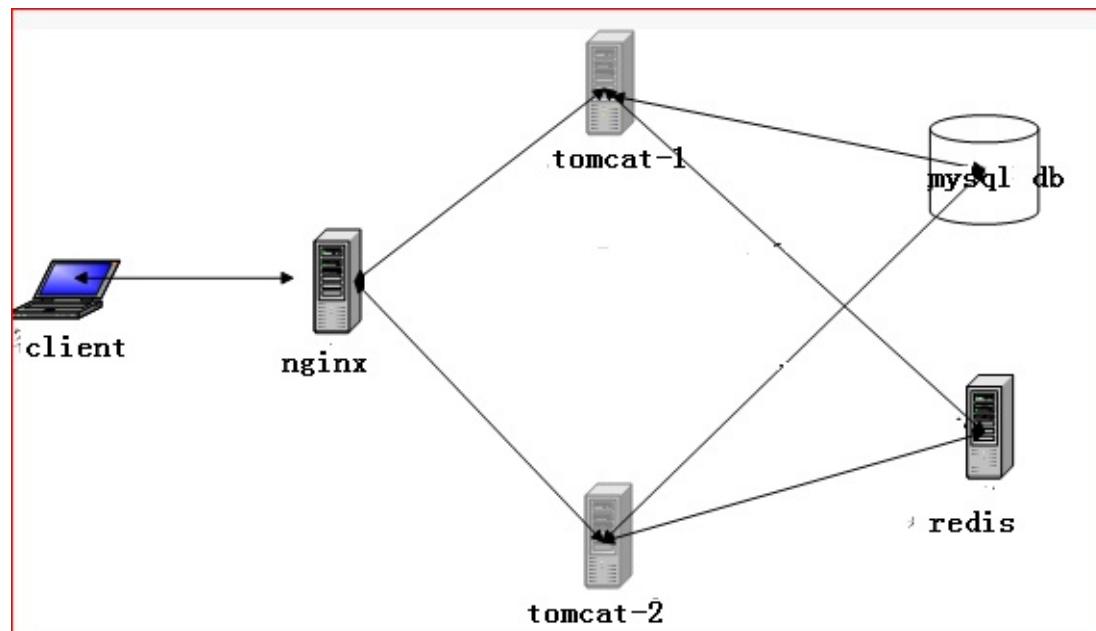
[回目录](#)

主机 操作系统 IP地址

- Nginx Centos7.2 192.168.31.141
- Tomcat-1 192.168.31.83
- Tomcat-2 192.168.31.250
- Mysql 192.168.31.225
- Redis 192.168.31.106

ii. 实验拓扑

[回目录](#)



在这个图中， nginx 做为反向代理，实现动静分离，将客户动态请求根据权重随机分配给两台 tomcat 服务器， redis 做为两台 tomcat 的共享 session 数据服务器， mysql 做为两台 tomcat 的后端数据库。

iii. nginx 安装配置

[回目录](#)

使用 Nginx 作为 Tomcat 的负载平衡器，Tomcat 的会话 Session 数据存储在 Redis，能够实现零宕机的 7x24 效果。因为将会话存储在 Redis 中，因此 Nginx 就不必配置成 stick 粘贴某个 Tomcat 方式，这样才能真正实现后台多个 Tomcat 负载平衡。

1. 安装 nginx

[回目录](#)

安装 zlib-devel 、 pcre-devel 等依赖包

```
[root@www ~]# yum -y install gcc gcc-c++ make libtool zlib zlib-devel pcre pcre-devel openssl openssl-devel
```

注：结合 proxy 和 upstream 模块实现后端 web 负载均衡 结合 nginx 默认自带的 ngx_http_proxy_module 模块和 ngx_http_upstream_module 模块实现后端服务器的健康检查

创建 nginx 程序用户

```
[root@www ~]# useradd -s /sbin/nologin www
```

编译安装 nginx

```
[root@www ~]# tar zxf nginx-1.10.2.tar.gz
[root@www ~]# cd nginx-1.10.2/
[root@www nginx-1.10.2]# ./configure --prefix=/usr/local/nginx1.10 --user=www --group=www --with-http_stub_status_module --with-http_realip_module --with-http_ssl_module --with-http_gzip_static_module --with-pcre --with-http_flv_module
[root@www nginx-1.10.2]# make && make install
```

优化 nginx 程序的执行路径

```
[root@www nginx-1.10.2]# ln -s /usr/local/nginx1.10/sbin/nginx /usr/local/sbin/
[root@www nginx-1.10.2]# nginx -t
nginx: the configuration file /usr/local/nginx1.10/conf/nginx.conf syntax is ok
nginx: configuration file /usr/local/nginx1.10/conf/nginx.conf test is successful
```

2. 编写 nginx 服务脚本

[回目录](#)

脚本内容如下

```
[root@www ~]# cat /etc/init.d/nginx
#!/bin/bash
# nginx Startup script for the Nginx HTTP Server
# chkconfig: - 85 15
# pidfile: /usr/local/nginx1.10/logs/nginx.pid
# config: /usr/local/nginx1.10/conf/nginx.conf
nginxd=/usr/local/nginx1.10/sbin/nginx
nginx_config=/usr/local/nginx1.10/conf/nginx.conf
nginx_pid=/usr/local/nginx1.10/logs/nginx.pid
RETVAL=0
prog="nginx"
# Source function library.
. /etc/rc.d/init.d/functions
# Start nginx daemons functions.
start() {
if [ -f $nginx_pid ] ; then
    echo "nginx already running...."
    exit 1
fi
echo -n "Starting $prog: "
$nginxd -c ${nginx_config}
RETVAL=$?
[ $RETVAL = 0 ] && touch /var/lock/subsys/nginx
}
```

```
# Stop nginx daemons functions.
stop() {
    echo -n "Stopping $prog: "
    $nginxd -s stop
    RETVAL=$?
    [ $RETVAL = 0 ] && rm -f /var/lock/subsys/nginx
}

# reload nginx service functions.
reload() {
    echo -n "Reloading $prog: "
    $nginxd -s reload
}

# status ngnx service functions
status() {
if [ -f $nginx_pid ] ; then
    echo "$prog is running"
else
    echo "$prog is stop"
fi
}
case "$1" in
start)
    start
    ;;
stop)
    stop
    ;;
reload)
    reload
    ;;
restart)
    stop
    start
    ;;
status)
    status
    ;;
*)
    echo "Usage: $prog {start|stop|restart|reload|status}"
    exit 1

```

```
;;
esac
```

```
[root@www ~]# chmod +x /etc/init.d/nginx
[root@www ~]# chkconfig --add nginx
[root@www ~]# chkconfig nginx on
```

3. 配置 nginx 反向代理：反向代理+负载均衡+健康探测

[回目录](#)

nginx.conf 文件内容

```
[root@www ~]# cat /usr/local/nginx1.10/conf/nginx.conf
user    www www;
worker_processes  4;
worker_cpu_affinity 0001 0010 0100 1000;
error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;
worker_rlimit_nofile 10240;
pid        logs/nginx.pid;
events {
    use epoll;
    worker_connections  4096;
}
http {
    include      mime.types;
    default_type application/octet-stream;
    log_format  main  '$remote_addr - $remote_user [$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;
```

```
sendfile          on;
tcp_nopush       on;
#keepalive_timeout  0;
keepalive_timeout  65;
#Compression Settings
gzip on;
gzip_comp_level 6;
gzip_http_version 1.1;
gzip_proxied any;
gzip_min_length 1k;
gzip_buffers 16 8k;
gzip_types text/plain text/css text/javascript application/json application/javascript application/x-javascript application/xml;
gzip_vary on;
#end gzip
# http_proxy Settings
client_max_body_size   10m;
client_body_buffer_size 128k;
proxy_connect_timeout   75;
proxy_send_timeout      75;
proxy_read_timeout       75;
proxy_buffer_size        4k;
proxy_buffers            4 32k;
proxy_busy_buffers_size  64k;
proxy_temp_file_write_size 64k;
#load balance Settings
upstream backend_tomcat {
    server 192.168.31.83:8080 weight=1 max_fails=2 fail_time_out=10s;
    server 192.168.31.250:8080 weight=1 max_fails=2 fail_time_out=10s;
}
#virtual host Settings
server {
    listen      80;
    server_name www.benet.com;
    charset utf-8;
    location / {
        root html;
```

```

        index  index.jsp index.html index.htm;
    }
    location ~* \.(jsp|do)$ {
        proxy_pass  http://backend_tomcat;
        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forward
ed_for;
        proxy_next_upstream error timeout invalid_header http
_500 http_502 http_503 http_504;
    }
    location /nginx_status {
        stub_status on;
        access_log off;
        allow 192.168.31.0/24;
        deny all;
    }
}
}

```

重启 nginx 服务，使修改生效

```
[root@www ~]# service nginx restart
```

配置防火墙规则

```
[root@www ~]# firewall-cmd --permanent --add-port=80/tcp
success
[root@www ~]# firewall-cmd --reload
success
```

iv. 安装部署 tomcat 应用程序服务器

[回目录](#)

在 tomcat-1 和 tomcat-2 节点上安装 JDK

在安装 tomcat 之前必须先安装 JDK , JDK 的全称是 java development kit ,是 sun 公司免费提供的 java 语言的软件开发工具包，其中包含 java 虚拟机 (JVM) ，编写好的 java 源程序经过编译可形成 java 字节码，只要安装了 JDK ，就可以利用 JVM 解释这些字节码文件，从而保证了 java 的跨平台性。

1. 安装 JDK ，配置 java 环境

[回目录](#)

将 jdk-7u65-linux-x64.gz 解压

```
[root@tomcat-1 ~]# tar zxf jdk-7u65-linux-x64.gz
```

将解压的 jdk1.7.0_65 目录移致动到 /usr/local/ 下并重命名为 java

```
[root@tomcat-1 ~]# mv jdk1.7.0_65/ /usr/local/java
```

1.2 tomcat 环境变量

[回目录](#)

在 /etc/profile 文件中添加内容如下

```
export JAVA_HOME=/usr/local/java
export PATH=$JAVA_HOME/bin:$PATH
```

通过 source 命令执行 profile 文件，使其生效。

```
[root@tomcat-1 ~]# source /etc/profile
[root@tomcat-1 ~]# echo $PATH
/usr/local/java/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
```

按照相同方法在`tomcat-2`也安装`JDK`

1.3 java version

[回目录](#)

分别在在 tomcat-1 和 tomcat-2 节点运行 `java -version` 命令查看 java 版本是否和之前安装的一致。

```
[root@tomcat-1 ~]# java -version
java version "1.7.0_65"
Java(TM) SE Runtime Environment (build 1.7.0_65-b17)
Java HotSpot(TM) 64-Bit Server VM (build 24.65-b04, mixed mode)
```

至此java环境已经配置完成

2. 在 tomcat-1 和 tomcat-2 节点安装配置 tomcat

[回目录](#)

解压 `apache-tomcat-7.0.54.tar.gz` 包

```
[root@tomcat-1 ~]# tar zxf apache-tomcat-7.0.54.tar.gz
```

将解压生成的文件夹移动到 `/usr/local/` 下，并改名为 `tomcat7`

```
[root@tomcat-1 ~]# mv apache-tomcat-7.0.54 /usr/local/tomcat7
```

2.1 配置 tomcat 环境变量

[回目录](#)

`/etc/profile` 文件内容如下

```
export JAVA_HOME=/usr/local/java
export CATALINA_HOME=/usr/local/tomcat7
export PATH=$JAVA_HOME/bin:$CATALINA_HOME/bin:$PATH
```

通过 `source` 命令执行 `profile` 文件，使其生效

```
[root@tomcat-1 ~]# source /etc/profile
[root@tomcat-1 ~]# echo $PATH
/usr/local/java/bin:/usr/local/tomcat7/bin:/usr/local/java/bin:/
usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
```

查看 `tomcat` 的版本信息

```
[root@tomcat-1 ~]# catalina.sh version
Using CATALINA_BASE:      /usr/local/tomcat7
Using CATALINA_HOME:      /usr/local/tomcat7
Using CATALINA_TMPDIR:   /usr/local/tomcat7/temp
Using JRE_HOME:          /usr/local/java
Using CLASSPATH:         /usr/local/tomcat7/bin/bootstrap.jar:/usr/
local/tomcat7/bin/tomcat-juli.jar
Server version: Apache Tomcat/7.0.54
Server built:  May 19 2014 10:26:15
Server number: 7.0.54.0
OS Name:        Linux
OS Version:    3.10.0-327.el7.x86_64
Architecture:  amd64
JVM Version:   1.7.0_65-b17
JVM Vendor:    Oracle Corporation
```

启动tomcat

```
[root@tomcat-1 ~]# /usr/local/tomcat7/bin/startup.sh
Using CATALINA_BASE:      /usr/local/tomcat7
Using CATALINA_HOME:      /usr/local/tomcat7
Using CATALINA_TMPDIR:   /usr/local/tomcat7/temp
Using JRE_HOME:          /usr/local/java
Using CLASSPATH:         /usr/local/tomcat7/bin/bootstrap.jar:/usr/
local/tomcat7/bin/tomcat-juli.jar
Tomcat started.
```

Tomcat默认运行在8080端口，运行netstat命令查看8080端口监听的信息

```
[root@tomcat-1 ~]# netstat -anpt | grep java
tcp6      0      0 :::8009      :::*          LISTEN
        42330/java
tcp6      0      0 :::8080      :::*          LISTEN
        42330/java
```

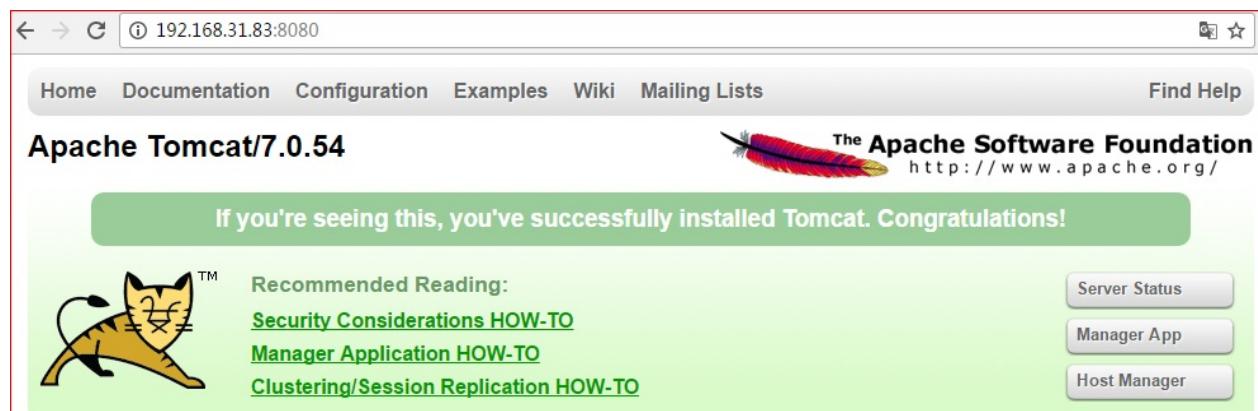
2.2 防火墙规则配置

[回目录](#)

```
[root@tomcat-1 ~]# firewall-cmd --permanent --add-port=8080/tcp
success
[root@tomcat-1 ~]# firewall-cmd --reload
success
```

按照相同方法在`tomcat-2`也安装

打开浏览器分别对 tomcat-1 和 tomcat-2 访问测试



如果想关闭 tomcat 则运行 /usr/local/tomcat7/bin/shutdown.sh 命令好了，大家可以看到访成功。说明我们的 tomcat 安装完成，下面我们就来修改配置文件

```
[root@tomcat-1 ~]# vim /usr/local/tomcat7/conf/server.xml
```

设置默认虚拟主机，并增加 jvmRoute

```
<Engine name="Catalina" defaultHost="localhost" jvmRoute="tomcat-1">
```

修改默认虚拟主机，并将网站文件路径指向 /web/webapp1，在 host 段增加 context 段

```
<Host name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true">
<Context docBase="/web/webapp1" path="" reloadable="true"/>
</Host>
```

增加文档目录与测试文件

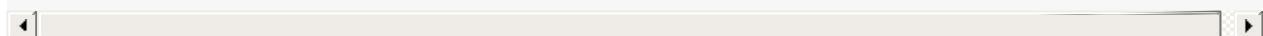
```
[root@tomcat-1 ~]# mkdir -p /web/webapp1
[root@tomcat-1 ~]# cd /web/webapp1/
[root@ tomcat-1 webapp1]# vi index.jsp
```

index.jsp 内容如下

```
<%@page language="java" import="java.util.*" pageEncoding="UTF-8"
%>
<html>
  <head>
    <title>tomcat-1</title>
  </head>
  <body>
    <h1><font color="red">Session serviced by tomcat</font></h1>
    <table align="center" border="1">
      <tr>
        <td>Session ID</td>
        <td><%=session.getId() %></td>
      <% session.setAttribute("abc", "abc");%>
      </tr>
      <tr>
        <td>Created on</td>
        <td><%= session.getCreationTime() %></td>
      </tr>
    </table>
  </body>
<html>
```

停止 tomcat 运行，检查配置文件并启动 tomcat

```
[root@tomcat-1 ~]# shutdown.sh
[root@tomcat-1 ~]# netstat -anpt | grep java
[root@tomcat-1 ~]# catalina.sh configtest
Using CATALINA_BASE:      /usr/local/tomcat7
Using CATALINA_HOME:      /usr/local/tomcat7
Using CATALINA_TMPDIR:   /usr/local/tomcat7/temp
Using JRE_HOME:          /usr/local/java
Using CLASSPATH:         /usr/local/tomcat7/bin/bootstrap.jar:/usr/
local/tomcat7/bin/tomcat-juli.jar
Nov 16, 2016 1:04:05 AM org.apache.catalina.core.AprLifecycleLi
tener init
INFO: The APR based Apache Tomcat Native library which allows op
timal performance in production environments was not found on th
e java.library.path: /usr/java/packages/lib/amd64:/usr/lib64:/li
b64:/lib:/usr/lib
Nov 16, 2016 1:04:05 AM org.apache.coyote.AbstractProtocol init
INFO: Initializing ProtocolHandler ["http-bio-8080"]
Nov 16, 2016 1:04:05 AM org.apache.coyote.AbstractProtocol init
INFO: Initializing ProtocolHandler ["ajp-bio-8009"]
Nov 16, 2016 1:04:05 AM org.apache.catalina.startup.Catalina loa
d
INFO: Initialization processed in 534 ms
[root@tomcat-1 ~]# startup.sh
Using CATALINA_BASE:      /usr/local/tomcat7
Using CATALINA_HOME:      /usr/local/tomcat7
Using CATALINA_TMPDIR:   /usr/local/tomcat7/temp
Using JRE_HOME:          /usr/local/java
Using CLASSPATH:         /usr/local/tomcat7/bin/bootstrap.jar:/usr/
local/tomcat7/bin/tomcat-juli.jar
Tomcat started.
```



```
[root@tomcat-1 ~]# netstat -anpt | grep java
tcp6      0      0 :::8009      :::*          LISTEN
  8180/java
tcp6      0      0 :::8080      :::*          LISTEN
  8180/java
```

Tomcat-2 节点与 tomcat-1 节点配置基本类似，只是 jvmRoute 不同，另外为了区分由哪个节点提供访问，测试页标题也不同（生产环境两个 tomcat 服务器提供的网页内容是相同的）。其他的配置都相同。

用浏览器访问 nginx 主机，验证负载均衡

第一次访问的结果



第二次访问的结果



验证健康检查的方法可以关掉一台 tomcat 主机，用客户端浏览器测试访问。从上面的结果能看出两次访问，nginx 把访问请求分别分发给了后端的 tomcat-1 和 tomcat-2，客户端的访问请求实现了负载均衡，但 sessionid 并不一样。所以，到这里我们准备工作就全部完成了，下面我们来配置 tomcat 通过 redis 实现会话保持。

v. 安装 redis

[回目录](#)

下载 redis 源码，并进行相关操作，如下：

wget <http://download.redis.io/releases/redis-3.2.3.tar.gz>

1. 解压安装 redis

[回目录](#)

```
[root@redis ~]# tar zxf redis-3.2.3.tar.gz
```

解压完毕后，现在开始安装，如下：

```
[root@redis ~]# cd redis-3.2.3/
[root@redis redis-3.2.3]# make && make install
```

```
XSLTPROC      : /usr/bin/xsltproc
XSLROOT       :

PREFIX        : /usr/local
BINDIR        : /usr/local/bin
DATADIR       : /usr/local/share
INCLUDEDIR    : /usr/local/include
LIBDIR        : /usr/local/lib
MANDIR        : /usr/local/share/man

Hint: It's a good idea to run 'make test' ;)

make[1]: Leaving directory `/root/redis-3.2.3/src'
cd src && make install
make[1]: Entering directory `/root/redis-3.2.3/src'

Hint: It's a good idea to run 'make test' ;)

INSTALL
INSTALL
INSTALL
INSTALL
INSTALL
```

通过上图，我们可以很容易的看出， redis 安装

到 /usr/local,/usr/local/bin,/usr/local/share,/usr/local/include,/usr/local/lib,/usr/local/share/man 目录下。

2. 初始化 redis

[回目录](#)

然后再切换到 utils 目录下，执行 redis 初始化脚本 `install_server.sh` ，如下：

```
[root@redis redis-3.2.3]# cd utils/
[root@redis utils]# ./install_server.sh
Welcome to the redis service installer
This script will help you easily set up a running redis server

Please select the redis port for this instance: [6379]
Selecting default: 6379
Please select the redis config file name [/etc/redis/6379.conf]
Selected default - /etc/redis/6379.conf
Please select the redis log file name [/var/log/redis_6379.log]
Selected default - /var/log/redis_6379.log
Please select the data directory for this instance [/var/lib/redis/6379]
Selected default - /var/lib/redis/6379
Please select the redis executable path [/usr/local/bin/redis-server]
Selected config:
Port          : 6379
Config file    : /etc/redis/6379.conf
Log file       : /var/log/redis_6379.log
Data dir       : /var/lib/redis/6379
Executable     : /usr/local/bin/redis-server
Cli Executable : /usr/local/bin/redis-cli
Is this ok? Then press ENTER to go on or Ctrl-C to abort.
Copied /tmp/6379.conf => /etc/init.d/redis_6379
Installing service...
Successfully added to chkconfig!
Successfully added to runlevels 345!
Starting Redis server...
Installation successful!
```

通过上面的安装过程，我们可以看出 redis 初始化后 redis 配置文件为 `/etc/redis/6379.conf`，日志文件为 `/var/log/redis_6379.log`，数据文件 `dump.rdb` 存放到 `/var/lib/redis/6379` 目录下，启动脚本为 `/etc/init.d/redis_6379`。

3. systemd

[回目录](#)

现在我们要使用 `systemd`，所以在 `/etc/systemd/system` 下创建一个单位文件名字为 `redis_6379.service`。

```
[root@redis utils]# vi /etc/systemd/system/redis_6379.service
```

内容如下

```
[Unit]
Description=Redis on port 6379
[Service]
Type=forking
ExecStart=/etc/init.d/redis_6379 start
ExecStop=/etc/init.d/redis_6379 stop
[Install]
WantedBy=multi-user.target
```

注：这里 `Type = forking` 是后台运行的形式

4. 启动 redis

[回目录](#)

```
[root@redis utils]# systemctl daemon-reload
[root@redis utils]# systemctl enable redis_6379.service
[root@redis utils]# systemctl start redis_6379.service
[root@redis utils]# systemctl status redis_6379.service
● redis_6379.service - Redis on port 6379
   Loaded: loaded (/etc/systemd/system/redis_6379.service; enabled; vendor preset: disabled)
     Active: active (running) since Wed 2016-11-16 21:07:26 CST; 4 min 25s ago
       Process: 7732 ExecStart=/etc/init.d/redis_6379 start (code=exited, status=0/SUCCESS)
      Main PID: 7734 (redis-server)
         CGroup: /system.slice/redis_6379.service
                   └─7734 /usr/local/bin/redis-server 127.0.0.1:6379

Nov 16 21:07:26 redis systemd[1]: Starting Redis on port 6379...
Nov 16 21:07:26 redis redis_6379[7732]: Starting Redis server...
Nov 16 21:07:26 redis systemd[1]: Started Redis on port 6379.
[root@redis utils]# netstat -anpt | grep 6379
tcp        0      0 127.0.0.1:6379        0.0.0.0:*      LISTEN
7734/redis-server 1
```

从显示结果可以看到 redis 默认监听的是 127.0.0.1 的 6379 端口

5. 防火墙规则设置

[回目录](#)

```
[root@redis utils]# firewall-cmd --permanent --add-port=6379/tcp
success
[root@redis utils]# firewall-cmd --reload
success
```

6. redis version

[回目录](#)

现在来查看 redis 版本使用 `redis-cli --version` 命令，如下

```
[root@redis utils]# redis-cli --version
redis-cli 3.2.3
```

通过显示结果，我们可以看到 redis 版本是 3.2.3。到此源码方式安装 redis 就介绍完毕。

redis 安装完毕之后，我们再来配置 redis 设置 redis 监听的地址，添加监听 redis 主机的 ip 考虑到安全性，我们需要启用 redis 的密码验证功能 `requirepass` 参数

7. 最终 redis 配置文件如下

[回目录](#)

```
[root@redis ~]# grep -Ev '^#|^$' /etc/redis/6379.conf
bind 127.0.0.1 192.168.31.106
protected-mode yes
port 6379
tcp-backlog 511
timeout 0
tcp-keepalive 300
daemonize yes
supervised no
pidfile /var/run/redis_6379.pid
loglevel notice
logfile /var/log/redis_6379.log
databases 16
save 900 1
save 300 10
save 60 10000
stop-writes-on-bgsave-error yes
rdbcompression yes
rdbchecksum yes
dbfilename dump.rdb
dir /var/lib/redis/6379
slave-serve-stale-data yes
```

```
slave-read-only yes
repl-diskless-sync no
repl-diskless-sync-delay 5
repl-disable-tcp-nodelay no
slave-priority 100
requirepass pwd@123
appendonly no
appendfilename "appendonly.aof"
appendfsync everysec
no-appendfsync-on-rewrite no
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
aof-load-truncated yes
lua-time-limit 5000
slowlog-log-slower-than 10000
slowlog-max-len 128
latency-monitor-threshold 0
notify-keyspace-events "$"
hash-max-ziplist-entries 512
hash-max-ziplist-value 64
list-max-ziplist-size -2
list-compress-depth 0
set-max-intset-entries 512
zset-max-ziplist-entries 128
zset-max-ziplist-value 64
hll-sparse-max-bytes 3000
activerehashing yes
client-output-buffer-limit normal 0 0 0
client-output-buffer-limit slave 256mb 64mb 60
client-output-buffer-limit pubsub 32mb 8mb 60
hz 10
aof-rewrite-incremental-fsync yes
```

8. 重新启动 **redis** 服务

[回目录](#)

```
[root@redis ~]# systemctl restart redis_6379.service
[root@redis ~]# netstat -anpt | grep redis
tcp    0    0 192.168.31.106:6379      0.0.0.0:*      LISTEN      8418
/redis-server 1
```

redis 配置文件配置完毕后，我们来启动 redis 并进行简单的操作。如下：

```
[root@redis ~]# redis-cli -h 192.168.31.106 -p 6379 -a pwd@123
192.168.31.106:6379> keys *
(empty list or set)
192.168.31.106:6379> set name lisi
OK
192.168.31.106:6379> get name
"lisi"
192.168.31.106:6379>
```

说明：

关于 redis-cli -h 192.168.31.106 -p 6379 -a pwd@123 的参数解释

这条命令是说要连接 redis 服务器，

- IP 是 192.168.31.106
- 端口是 6379
- 密码是 pwd@123
- keys * 是查看 redis 所有的键值对
- set name lisi 添加一个键值 name ，内容为 lisi
- get name 查看 name 这个键值的内容

redis 的命令使用暂时我们就介绍这么多

vi. 配置 tomcat session redis 同步

[回目录](#)

1. 下载 tomcat-redis-session-manager 相应的 jar 包，主要有三个：

[回目录](#)

- tomcat-redis-session-manage-tomcat7.jar
- jedis-2.5.2.jar
- commons-pool2-2.2.jar

下载完成后拷贝到 \$TOMCAT_HOME/lib 中

```
[root@tomcat-1 ~]# cp tomcat-redis-session-manage-tomcat7.jar jedis-2.5.2.jar commons-pool2-2.2.jar /usr/local/tomcat7/lib/
```

2.修改 tomcat 的 context.xml

[回目录](#)

```
[root@tomcat-1 ~]# cat /usr/local/tomcat7/conf/context.xml
<?xml version='1.0' encoding='utf-8'?>
<!--
    Licensed to the Apache Software Foundation (ASF) under one or
    more
    contributor license agreements. See the NOTICE file distributed
    with
    this work for additional information regarding copyright owner
    ship.
    The ASF licenses this file to You under the Apache License, Version
    2.0
    (the "License"); you may not use this file except in compliance
    with
    the License. You may obtain a copy of the License at
        http://www.apache.org/licenses/LICENSE-2.0
    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied.
    See the License for the specific language governing permission
    s and
    limitations under the License.
```

```
-->
<!-- The contents of this file will be loaded for each web application -->
<Context>
    <!-- Default set of monitored resources -->
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <!-- Uncomment this to disable session persistence across Tomcat restarts -->
    <!--
        <Manager pathname="" />
    -->
    <!-- Uncomment this to enable Comet connection tracking (provides events
        on session expiration as well as webapp lifecycle) -->
    <!--
        <Valve className="org.apache.catalina.valves.CometConnection
ManagerValve" />
    -->
    <Valve className="com.orangefunction.tomcat.redissessions.Re
disSessionHandlerValve" />
    <Manager className="com.orangefunction.tomcat.redissessions.
RedisSessionManager"
host="192.168.31.106"
password="pwd@123"
port="6379"
database="0"
maxInactiveInterval="60" />
</Context>
```

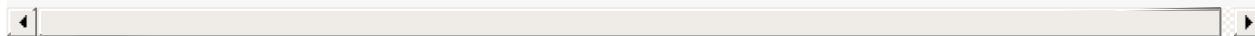
3. 重启 tomcat 服务

[回目录](#)

说明：

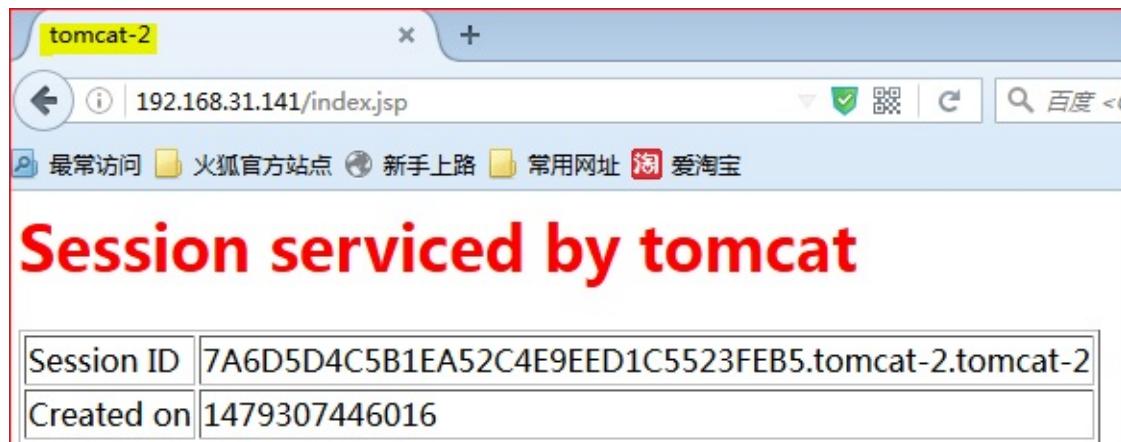
maxInactiveInterval="60" : session 的失效时间

```
[root@tomcat-1 ~]# shutdown.sh
Using CATALINA_BASE:      /usr/local/tomcat7
Using CATALINA_HOME:      /usr/local/tomcat7
Using CATALINA_TMPDIR:   /usr/local/tomcat7/temp
Using JRE_HOME:          /usr/local/java
Using CLASSPATH:         /usr/local/tomcat7/bin/bootstrap.jar:/usr/
local/tomcat7/bin/tomcat-juli.jar
[root@tomcat-1 ~]# startup.sh
Using CATALINA_BASE:      /usr/local/tomcat7
Using CATALINA_HOME:      /usr/local/tomcat7
Using CATALINA_TMPDIR:   /usr/local/tomcat7/temp
Using JRE_HOME:          /usr/local/java
Using CLASSPATH:         /usr/local/tomcat7/bin/bootstrap.jar:/usr/
local/tomcat7/bin/tomcat-juli.jar
Tomcat started.
```

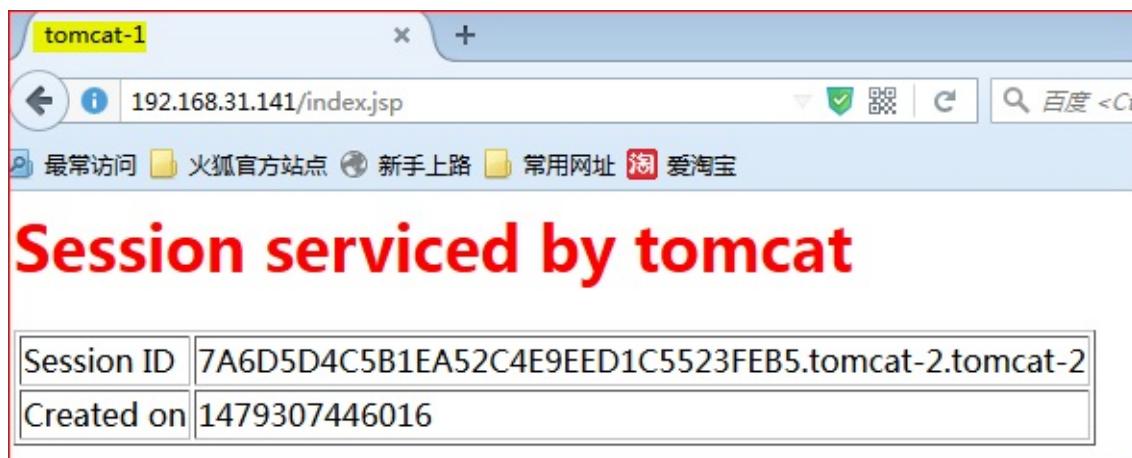


`tomcat-2` 执行和 `tomcat-1` 相同的操作

通过浏览器访问 <http://192.168.31.141/index.jsp> 测试页



刷新页面



可以看出，分别访问了不同的 tomcat ，但是得到的 session 却是相同的，说明达到了集群的目的。

注：从 Tomcat6 开始默认开启了 Session 持久化设置，测试时可以关闭本地 Session 持久化，其实也很简单，在 Tomcat 的 conf 目录下的 context.xml 文件中，取消注释下面那段配置即可：

修改前：

```
<!-- Uncomment this to disable session persistence across Tomcat
     restarts -->
<!--
<Manager pathname="" />
-->
```

修改后：

```
<!-- Uncomment this to disable session persistence across Tomcat
     restarts -->
<Manager pathname="" />
```

重启 tomcat 服务

查看 redis

```
[root@redis ~]# redis-cli -h 192.168.31.106 -p 6379 -a pwd@123
192.168.31.106:6379> keys *
1) "6C3F950BE6413AD2E0EF00F930881224.tomcat-1.tomcat-1"
2) "name"
3) "7A6D5D4C5B1EA52C4E9EED1C5523FEB5.tomcat-2.tomcat-2"
4) "32C35EEA064884F065E93CB00C690662.tomcat-1.tomcat-1"
```

vii. tomcat 连接数据库

[回目录](#)

1. 192.168.31.225 作为 mysql 数据库服务器

[回目录](#)

```
[root@db ~]# mysql -uroot -p123.abc
mysql> grant all on *.* to javauser@'192.168.31.%' identified by
'javapasswd';
Query OK, 0 rows affected, 1 warning (0.02 sec)

mysql> create database javatest;
Query OK, 1 row affected (0.01 sec)

mysql> use javatest;
Database changed

mysql> create table testdata(id int not null auto_increment primary key,foo varchar(25),bar int);
Query OK, 0 rows affected (0.02 sec)
```

2. 插入些数据

[回目录](#)

```

mysql> insert into testdata(foo,bar) values ('hello','123456'),(
'ok','654321');
Query OK, 2 rows affected (0.04 sec)
Records: 2  Duplicates: 0  Warnings: 0
mysql> select * from testdata;
+---+---+---+
| id | foo | bar |
+---+---+---+
| 1 | hello | 123456 |
| 2 | ok | 654321 |
+---+---+---+
2 rows in set (0.00 sec)

```

3. mysql 防火墙配置

[回目录](#)

配置 tomcat 服务器连接 mysql 数据库

4. tomcat 连接 mysql

[回目录](#)

下载 mysql-connector-java-5.1.22-bin.jar 并复制
到 \$CATALINA_HOME/lib 目录下

```

[root@tomcat-1 ~]# cp mysql-connector-java-5.1.22-bin.jar /usr/local/tomcat7/lib/
[root@tomcat-1 ~]# ls /usr/local/tomcat7/lib/mysql-connector-jav
a-5.1.22-bin.jar
/usr/local/tomcat7/lib/mysql-connector-java-5.1.22-bin.jar

context configuration
configure the JNDI datasource in tomcat by adding a declaration
for your resource to your context
[root@tomcat-1 ~]# vi /usr/local/tomcat7/conf/context.xml

```

5. 在 <Context> 中添加如下内容

[回目录](#)

```
<Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
    maxActive="100" maxIdle="30" maxWait="10000"
    username="javauser" password="javapass" driverClassName="com.mysql.jdbc.Driver"
    url="jdbc:mysql://192.168.31.225:3306/javatest"/>
```

保存修改并退出

[web.xml configuration](#)

```
[root@tomcat-1 ~]# mkdir /web/webapp1/WEB-INF
[root@tomcat-1 ~]# vi /web/webapp1/WEB-INF/web.xml
```

添加内容如下

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">
    <description>MySQL Test App</description>
    <resource-ref>
        <description>DB Connection</description>
        <res-ref-name>jdbc/TestDB</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>
</web-app>
```

保存修改并退出，重启 tomcat 服务

```
[root@tomcat-1 ~]# shutdown.sh
Using CATALINA_BASE:      /usr/local/tomcat7
Using CATALINA_HOME:      /usr/local/tomcat7
Using CATALINA_TMPDIR:   /usr/local/tomcat7/temp
Using JRE_HOME:          /usr/local/java
Using CLASSPATH:         /usr/local/tomcat7/bin/bootstrap.jar:/usr/
local/tomcat7/bin/tomcat-juli.jar
[root@tomcat-1 ~]# startup.sh
Using CATALINA_BASE:      /usr/local/tomcat7
Using CATALINA_HOME:      /usr/local/tomcat7
Using CATALINA_TMPDIR:   /usr/local/tomcat7/temp
Using JRE_HOME:          /usr/local/java
Using CLASSPATH:         /usr/local/tomcat7/bin/bootstrap.jar:/usr/
local/tomcat7/bin/tomcat-juli.jar
Tomcat started.
```

‘tomcat-2’进行和‘tomcat-1’相同的操用

Test code

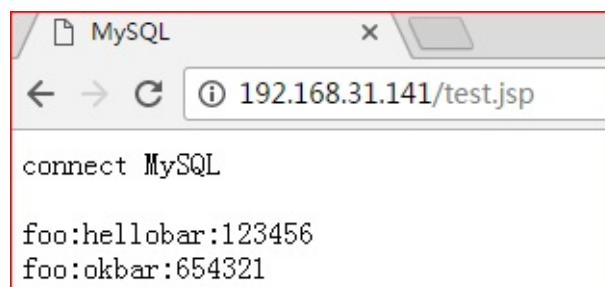
Now create a simple `test.jsp` page, 内容如下：

6. test.jsp

[回目录](#)

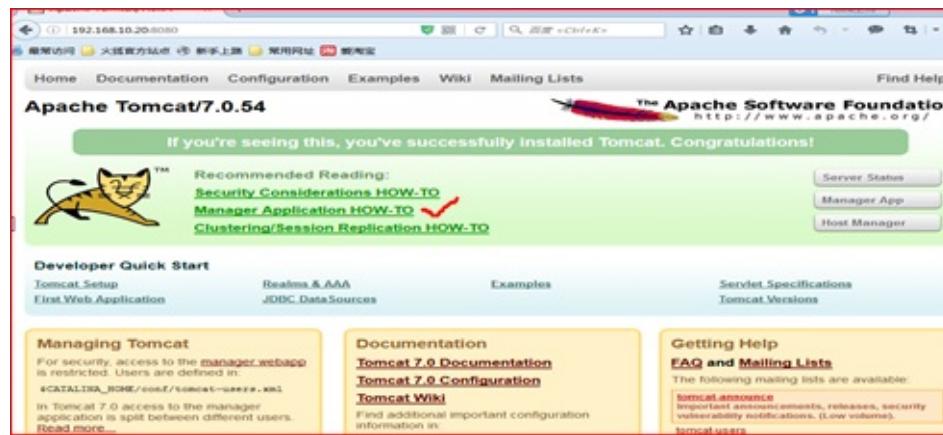
```
[root@tomcat-2 ~]# vi /web/webapp1/test.jsp
[root@tomcat-2 webapp1]# cat test.jsp
<%@ page language="java" import="java.sql.*" pageEncoding="GB231
2"%>
<html>
  <head>
    <title>MySQL</title>
  </head>
<body>
connect MySQL<br>
<%
String driverClass="com.mysql.jdbc.Driver";
String url="jdbc:mysql://192.168.31.225:3306/javatest";
String username = "javauser";
String password = "javapasswd";
Class.forName(driverClass);
Connection conn=DriverManager.getConnection(url, username, passw
ord);
Statement stmt=conn.createStatement();
ResultSet rs = stmt.executeQuery("select * from testdata");
while(rs.next()){
  out.println("<br>foo:"+rs.getString(2)+"bar:"+rs.getString(3));
}
rs.close();
stmt.close();
conn.close();
%>
</body></html>
```

通过浏览器访问 <http://192.168.31.141/test.jsp> 测试页



注：

以上配置可以参考 [tomcat docs](#)



User Guide

- [1\) Introduction](#)
- [2\) Setup](#)
- [3\) First webapp](#)
- [4\) Deployer](#)
- [5\) Manager](#)
- [6\) Realms and AAA](#)
- [7\) Security Manager](#)
- [8\) JNDI Resources](#)
- [9\) JDBC DataSources](#) ✓
- [10\) Classloading](#)
- [11\) ISPs](#)
- [12\) SSL](#)
- [13\) SSI](#)

- [Introduction](#)
- [Configuring Manager Application Access](#)

Supported Manager Commands

1. [Deploy A New Application Remotely](#)
2. [Deploy A New Application from a Local Pa](#)
3. [List Currently Deployed Applications](#)
4. [Reload An Existing Application](#)
5. [List OS and JVM Properties](#)
6. [List Available Global JNDI Resources](#)
7. [Session Statistics](#)
8. [Start an Existing Application](#)
9. [Stop an Existing Application](#)
10. [Undeploy an Existing Application](#)
11. [Finding memory leaks](#)
12. [Server Status](#)

Links

- [Docs Home](#)
- [FAQ](#)
- [User Comments](#)

JNDI Datasource HOW-TO

Table of Contents

- [Introduction](#)
- [DriverManager, the service provider mechanism](#)
- [Database Connection Pool \(DBCP\) Configuration](#)
 1. [Installation](#) ✓
 2. [Preventing database connection pool leaks](#)
 3. [MySQL DBCP Example](#)
 4. [Oracle 8i, 9i & 10g](#)
 5. [PostgreSQL](#)
- [Non-DBCP Solutions](#)
- [Oracle 8i with OCI client](#)
 1. [Introduction](#)
 2. [Putting it all together](#)

[回目录](#)

zabbix

目录

- I. zabbix 简介
 - i. Zabbix 的主要功能
 - ii. 服务器/代理和客户端模式
 - iii. zabbix 重要组件说明
- II. Zabbix 部署前的 LNMP 环境的搭建过程
 - i. MySQL 安装参见 Memcache 文档
 - ii. 安装 nginx
 - iii. 安装 php
 - iv. 测试 LNMP
- III. 监控系统 Zabbix-3.0.3 的安装
 - i. Zabbix Server 配置与启动
 - 1. 创建 Zabbix 数据库和 MySQL 用户
 - 2. 导入 Zabbix 初始数据
 - 3. 编辑 /usr/local/zabbix/etc/zabbix_server.conf
 - 4. 启动 Zabbix Server
 - 5. 添加开机启动脚本
 - 6. 设置防火墙规则
 - ii. 配置 zabbix web 页面
 - 1. 在安装目录将 frontends 拷贝到指定的 web root
 - 2. 安装 web
 - 3. 显示简体中文界面
 - 4. 解决 zabbix 绘图中出现中文乱码问题
- IV. zabbix 客户端的安装
 - i. 安装 Zabbix，配置 Zabbix agent
 - 1. 修改 zabbix 开机启动脚本中的 zabbix 安装目录
 - 2. 编辑 zabbix_agentd.conf
 - 3. 启动 zabbix_agentd
 - ii. 实现 zabbix 添加监测项
 - 1. 登录 zabbix，先在“配置”-“主机”里单击“创建主机”

- 2. 点击“主机”标签，填写相关信息
- 3. 切换到模板
- 4. 监控结果
- iii. Zabbix_agent 客户端操作
 - 1. mysql 授权
 - 2. 在 zabbix_agent 服务目录下创建 .my.cnf 连接文件 zabbix 安装目录是 /usr/local/zabbix
 - 3. 配置 MySQL 的 key 文件
 - 4. 重启 zabbix_agent
 - 5. 登录 zabbix 监控界面，在“配置”-“模板”里可以看到自带的mysql监控模板

I. zabbix 简介

[回目录](#)

zabbix 是完全开源的工具，整合了 cacti 和 nagios 等特性
 附：SNMP (udp 161 udp 162) 众多网络工具都支持此协议，比如常见路由交换，常见 OS 其既可以做管理端也可以做被管理端 snmp 协议大致有3个版本分别是 v1 v2 v3 无论是 v1 和 v2 的安全性是比较差的，因为传输是明文的，v3 的认证密码用 MD5 / SHA 摘要算法加密

很多工具支持网络管理的功能，而对于非网络设备（操作系统），可以完全抛开 snmp 这种不安全的架构来实现监控的。所以很多工具都是控制端和 agent 架构，他们有专属的 agent

i. Zabbix 的主要功能

[回目录](#)

- 具备常见的商业监控软件所具备的功能（主机的性能监控、网络设备性能监控、数据库、FTP 等通用协议监控、多种告警方式、详细的报表图表绘制）
- 支持自动发现网络设备和服务器（可以通过配置自动发现服务器规则来实现）
- 支持分布式，能集中展示、管理分布式的监控点，扩展性强
- server 提供通用接口（api 功能），可以自己开发完善各类监控（根据

相关接口编写程序实现），编写插件容易，可以自定义监控项，报警级别的设置。

- 数据收集，支持 `snmp` (包括 `trapping and polling`)，`IPMI`，`JMX`，`SSH`，`TELNET`；
- 自定义的检测；自定义收集数据的频率；

ii. 服务器/代理和客户端模式

[回目录](#)

- 灵活的触发器；可以定义非常灵活的问题阈值，称为触发器；
- 高可定制的报警；发送通知，可定制的报警升级，收件人，媒体类型。
`CPU` 负荷、内存使用、磁盘使用、网络状况、端口监视、日志监视等等。
- 硬件监控：`Zabbix IPMI Interface`
- 系统监控：`ZabbixAgent Interface`
- Java 监控：`Zabbix JMX Interface`
- 网络设备监控：`Zabbix SNMP Interface`
- 应用服务监控：`Zabbix Agent UserParameter`
- MySQL 数据库监控：`percona-monitoring-plulgins`
- URL 监控：`Zabbix Web 监控`

iii. zabbix 重要组件说明

[回目录](#)

- 1) `zabbix server` : 负责接收agent发送的报告信息的核心组件，所有配置、统计数据及操作数据都由它组织进行；
- 2) `database storage` : 专用于存储所有配置信息，以及由 `zabbix` 收集的数据；
- 3) `web interface` : `zabbix` 的 GUI 接口；
- 4) `proxy` : 可选组件，常用于监控节点很多的分布式环境中，代理 `server` 收集部分数据转发到 `server`，可以减轻 `server` 的压力；
- 5) `agent` : 部署在被监控的主机上，负责收集主机本地数据如 `cpu`、`内存`、`数据库` 等数据发往 `server` 端或 `proxy` 端；

另外，`zabbix server`、`proxy`、`agent` 都有自己的配置文件以及log文件，重要的参数需要在这里配置，后面会详细说明。

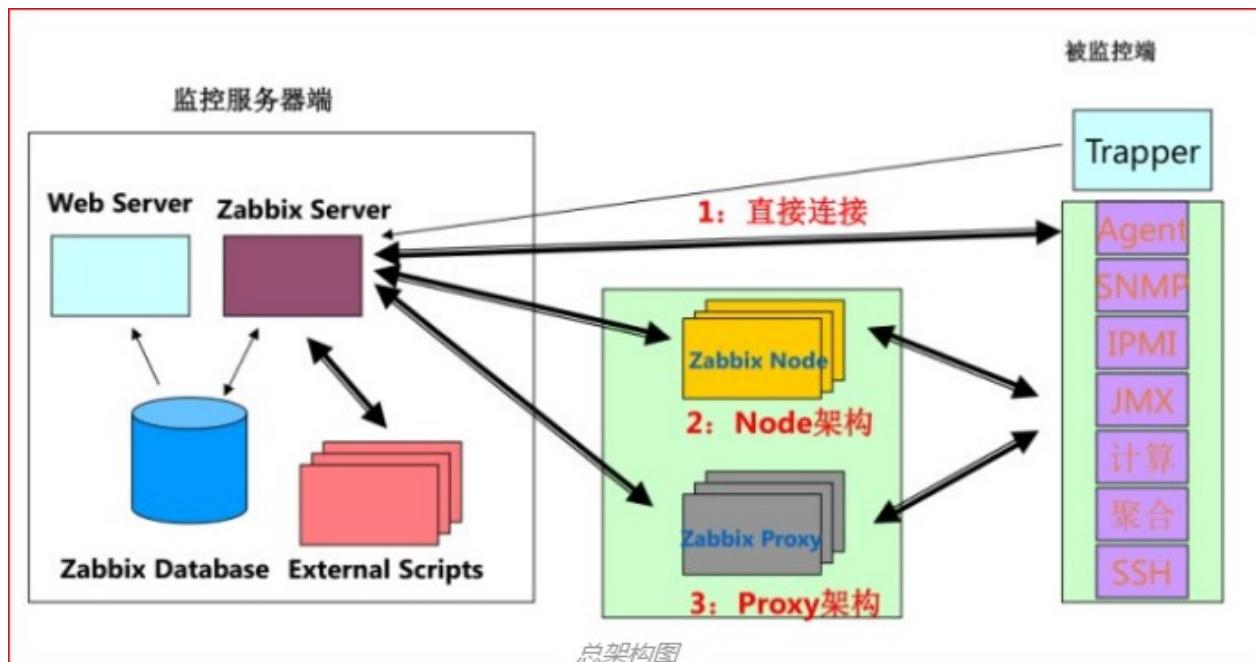
一个监控系统运行的大概的流程是这样的：

`agentd` 需要安装到被监控的主机上，它负责定期收集各项数据，并发送到 `zabbix server` 端，`zabbix server` 将数据存储到数据库中，`zabbix web` 根据数据在前端进行展现和绘图。这里 `agentd` 收集数据分为主动和被动两种模式：

- 主动：`agent` 请求 `server` 获取主动的监控项列表，并主动将监控项内需要检测的数据提交给 `server / proxy`
- 被动：`server` 向 `agent` 请求获取监控项的数据，`agent` 返回数据。

zabbix 常用的监控架构平台

1、`server-agentd` 模式：这个是最简单的架构了，常用于监控主机比较少的情况下。
 2、`server-proxy-agentd` 模式：这个常用于比较多的机器，使用 `proxy` 进行分布式监控，有效的减轻 `server` 端的压力。`zabbix` 的系统架构：



Zabbix 是一个基于 Web 界面的提供分布式系统监视以及网络监视功能的企业级开源解决方案。借助 Zabbix，可以很轻松地减轻运维人员们繁重的服务器管理任务，实现业务系统的持续运行。下面会逐步介绍 Zabbix 分布式监控系统的部署及使用

配置好 IP 、 DNS 、 网关 ，确保使用远程连接工具能够连接服务器

- zabbix 监控服务器：192.168.130.76 #zabbix的服务端（若要监控本机，则需要配置本机的 zabbix agent）
- Zabbix agent 被监控主机：192.168.130.78 #zabbix的客户端（被监控

端，需要配置 Zabbix agent

II. Zabbix 部署前的 LNMP 环境的搭建过程

[回目录](#)

安装编译工具及库文件(zabbix server)

```
[root@mysqla ~]# yum -y install make apr* autoconf automake curl-devel gcc gcc-c++ openssl openssl-devel gd kernel keyutils patch perl kernel-headers compat* mpfr cpp glibc libgomp libstdc++-devel keyutils-libs-devel libcom_err-devel libsepol-devel libselinux-devel krb5-devel zlib-devel libXpm* freetype libjpeg* libpng* libtool* libxml2 libxml2-devel patch libcurl-devel bzip2-devel freetype-devel
```

i. MySQL 安装参见 Memcache 文档

[回目录](#)

ii. 安装 nginx

[回目录](#)

解压 zlib

```
[root@mysqla ~]# tar zxf zlib-1.2.8.tar.gz
```

说明：不需要编译，只需要解压就行。

解压 pcre

```
[root@mysqla ~]# tar zxf pcre-8.39.tar.gz
```

说明：不需要编译，只需要解压就行。

```
[root@mysqla ~]# groupadd www
[root@mysqla ~]# useradd -g www www -s /sbin/nologin
```

下载nginx的源码包：<http://nginx.org/download>

解压源码包

```
[root@mysqla ~]# tar zxf nginx-1.10.2.tar.gz
[root@mysqla ~]# cd nginx-1.10.2/
[root@mysqla nginx-1.10.2]# ./configure --prefix=/usr/local/nginx1.10 --with-http_dav_module --with-http_stub_status_module --with-http_addition_module --with-http_sub_module --with-http_flv_module --with-http_mp4_module --with-pcre=/root/pcre-8.39 --with-zlib=/root/zlib-1.2.8 --with-http_ssl_module --with-http_gzip_static_module --user=www --group=www
[root@mysqla nginx-1.10.2]# make&& make install
[root@mysqla ~]# ln -s /usr/local/nginx1.10/sbin/nginx /usr/local/sbin/
[root@mysqla ~]# nginx
[root@mysqla ~]# netstat -anpt | grep nginx
tcp      0      0 0.0.0.0:80          0.0.0.0:*        LISTEN      11598/
nginx: master
[root@mysqla ~]# firewall-cmd --permanent --add-port=80/tcp
success
[root@mysqla ~]# firewall-cmd --reload
success
```

启动后可以在浏览器中打开页面，会显示nginx默认页面。



iii. 安装 php

[回目录](#)

```
[root@mysqla ~]# tar zxf libmcrypt-2.5.7.tar.gz
[root@mysqla ~]# cd libmcrypt-2.5.7/
[root@mysqla libmcrypt-2.5.7]# ./configure && make && make install
# ln -s /usr/local/mysql/lib/libmysqlclient.so.20.3.0 /usr/local/mysql/lib/libmysqlclient_r.so
[root@mysqla ~]# tar zxf php-5.6.27.tar.gz
[root@mysqla php-5.6.27]# ./configure --prefix=/usr/local/php5.6
--with-config-file-path=/etc --with-mysql=/usr/local/mysql --with-mysqli=/usr/local/mysql/bin/mysql_config --with-mysql-sock=/usr/local/mysql/mysql.sock --with-gd --with-iconv --with-libxml-dir=/usr --with-mhash --with-mcrypt --with-config-file-scan-dir=/etc/php.d --with-bz2 --with-zlib --with-freetype-dir --with-png-dir --with-jpeg-dir --enable-xml --enable-bcmath --enable-shmop --enable-sysvsem --enable-inline-optimization --enable-mbregex --enable-fpm --enable-mbstring --enable-ftp --enable-gd-native-ttf --with-openssl --enable-pcntl --enable-sockets --with-xmlrpc --enable-zip --enable-soap --without-pear --with-gettext --enable-session --with-mcrypt --with-curl
[root@mysqla php-5.6.27]# make&& make install
[root@mysqla php-5.6.27]# cp php.ini-production /etc/php.ini
```

编辑配置文件 `/etc/php.ini`，修改后的内容如下：

找到：

```
;date.timezone =
```

修改为：

```
date.timezone = PRC #设置时区
```

找到：

```
expose_php = On
```

修改为：

```
expose_php = Off #禁止显示php版本的信息
```

找到：

```
short_open_tag = Off
```

修改为：

```
short_open_tag = On //支持php短标签
```

找到：

```
post_max_size = 8M
```

修改为：

```
post_max_size = 16M //上传文件大小
```

找到：

```
max_execution_time = 30
```

修改为：

```
max_execution_time = 300 //php脚本最大执行时间
```

找到：

```
max_input_time = 60
```

修改为：

- max_input_time = 300 //以秒为单位对通过POST、GET以及PUT方式接收数据时间进行限制

- always_populate_raw_post_data = -1
- mbstring.func_overload = 0

创建 php-fpm 服务启动脚本

```
[root@mysqla php-5.6.27]# cp sapi/fpm/init.d.php-fpm /etc/init.d/php-fpm
[root@mysqla php-5.6.27]# chmod +x /etc/init.d/php-fpm
[root@mysqla php-5.6.27]# chkconfig --add php-fpm
[root@mysqla php-5.6.27]# chkconfig php-fpm on
```

提供 php-fpm 配置文件并编辑

```
#cp /usr/local/php5.6/etc/php-fpm.conf.default /usr/local/php5.6
/etc/php-fpm.conf
[root@mysqla php-5.6.27]# vi /usr/local/php5.6/etc/php-fpm.conf
```

修改内容如下

```
pid = run/php-fpm.pid
user = www
group = www
listen =127.0.0.1:9000
pm.max_children = 300
pm.start_servers = 10
pm.min_spare_servers = 10
pm.max_spare_servers =50
```

启动 php-fpm 服务

```
[root@phpserver ~]# service php-fpm start
Starting php-fpm done
[root@mysqla php-5.6.27]# netstat -anpt | grep php-fpm
tcp      0      0 127.0.0.1:9000      0.0.0.0:*      LISTEN      10937/ph
p-fpm: mast
```

配置 nginx 支持 php

```
[root@mysqla ~]# cat /usr/local/nginx1.10/conf/nginx.conf
user    www www;
worker_processes  4;
#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;
#pid        logs/nginx.pid;
events {
    use epoll;
    worker_connections  1024;
}
http {
    include       mime.types;
    default_type  application/octet-stream;
    #log_format  main  '$remote_addr - $remote_user [$time_local]
] "$request" '
    #                                '$status $body_bytes_sent "$http_referer"
'
    #                                '"$http_user_agent" "$http_x_forwarded_for"';
    #access_log  logs/access.log  main;
    sendfile       on;
    #tcp_nopush    on;
    #keepalive_timeout  0;
    keepalive_timeout  65;
    #gzip  on;
    server {
        listen      80;
        server_name  localhost;
        charset utf-8;
        #access_log  logs/host.access.log  main;
    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;
include fastcgi.conf;
}
#error_page 404 /404.html;
# redirect server error pages to the static page /50x.htm
ml
#
error_page 500 502 503 504 /50x.html;
location = /50x.html {
root html;
}
location /status {
stub_status on;
}
}
[root@mysqla conf]# nginx -t
nginx: the configuration file /usr/local/nginx1.10/conf/nginx.co
nf syntax is ok
nginx: configuration file /usr/local/nginx1.10/conf/nginx.conf t
est is successful
[root@mysqla conf]# killall -s HUP nginx
```

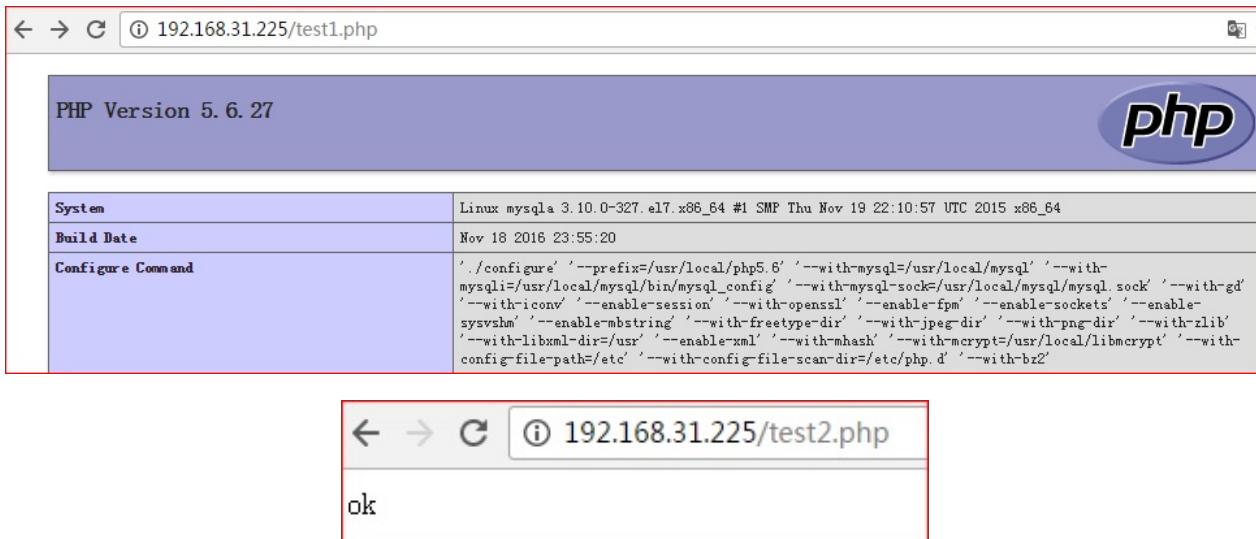
IV. 测试 LNMP

回目录

进入 nginx 默认的网页根目录，创建 .php 的测试页

```
[root@mysqla ~]# cat /usr/local/nginx1.10/html/test1.php
<?php
phpinfo()
?>
[root@mysqla ~]# cat /usr/local/nginx1.10/html/test2.php
<?php
$link=mysql_connect('localhost','root','123.abc');
if($link) echo "ok";
mysql_close();
?>
```

访问结果：



至此，LNMP 部署完毕。

III. 监控系统 Zabbix-3.0.3 的安装

[回目录](#)

zabbix-server端的操作

zabbix服务器端要提前安装好LNMP环境（mysql，nginx，php5的安装目录均是/usr/local）

创建Zabbix运行的用户：

```
[root@mysqla ~]#groupadd zabbix  
[root@mysqla ~]#useradd -g zabbix zabbix
```

安装libcurl和net-snmp:

```
[root@mysqla ~]# yum -y install net-snmp net-snmp-devel curl-devel  
java-1.8.0-openjdk java-1.8.0-openjdk-devel OpenIPMI-devel libssh2-devel
```

注：`OpenIPMI-devel` 和 `libssh2-devel` 软件包使用centos在线yum软件源安装

安装Fping:

```
[root@mysqla ~]# tar zxf fping-3.12.tar.gz  
[root@mysqla ~]# cd fping-3.12/  
[root@mysqla fping-3.12]# ./configure && make && make install  
[root@mysqla fping-3.12]# chown root:zabbix /usr/local/sbin/fping  
[root@mysqla fping-3.12]# chmod 4710 /usr/local/sbin/fping
```

安装Zabbix Server:

```
[root@mysqla ~]# tar zxf zabbix-3.2.1.tar.gz  
[root@mysqla ~]# cd zabbix-3.2.1/  
[root@mysqla zabbix-3.2.1]# ./configure --prefix=/usr/local/zabbix  
--enable-server --enable-agent --enable-javascript --with-mysql=/usr/  
local/mysql/bin/mysql_config --with-net-snmp --with-libcurl --  
with-openipmi
```

注意：编译时最好带上 `--enable-javascript` 这个参数，方便后续监控 `tomcat` 程序所用。`--with-ssh2` 是不需要在客户端服务器上面安装 Zabbix agent，如果需要使用 `ssh` 检查，需要在编译的时候加上这项，最低需要 `libssh2 1.0.0` 版本，需要安装 `ssh` 开发包 `--with-openipmi` 用户可以利用 `IPMI` 监视服务器的物理特征，如温度、电压、电扇工作状态、电源供应等。

如果添加了 `--enable-proxy`，那么会生成 `get` 和 `sender` 两条命令。如下，用于接收 `agent` 发送过来的信息，同时发送给 `server`。

```
[root@mysqla zabbix-3.2.1]#make && make install
```

添加系统软连接

```
[root@mysqla ~]# ln -s /usr/local/zabbix/bin/* /usr/local/bin/  
[root@mysqla ~]# ln -s /usr/local/zabbix/sbin/* /usr/local/sbin/
```

i. Zabbix Server 配置与启动

[回目录](#)

1. 创建 Zabbix 数据库和 MySQL 用户

[回目录](#)

```
mysql> create database zabbix character set utf8;  
Query OK, 1 row affected (0.01 sec)  
  
mysql> grant all privileges on zabbix.* to zabbix@localhost identified by 'zabbix';  
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

2. 导入 Zabbix 初始数据

[回目录](#)

切换到 zabbix 的解压目录下

```
[root@mysqla zabbix-3.2.1]# cd database/mysql/  
[root@mysqla mysql]# pwd  
/root/zabbix-3.2.1/database/mysql  
[root@mysqla mysql]# ls  
data.sql  images.sql  schema.sql
```

进行 zabbix 初始数据导入

```
[root@mysqla mysql]# mysql -uzabbix -pzabbix -hlocalhost zabbix  
< schema.sql  
[root@mysqla mysql]# mysql -uzabbix -pzabbix -hlocalhost zabbix  
< images.sql  
[root@mysqla mysql]# mysql -uzabbix -pzabbix -hlocalhost zabbix  
< data.sql
```

3. 编辑 /usr/local/zabbix/etc/zabbix_server.conf

[回目录](#)

```
root@mysqla ~]# grep -v "^#" /usr/local/zabbix/etc/zabbix_server  
.conf | grep -v "^\$"  
LogFile=/usr/local/zabbix/logs/zabbix_server.log  
PidFile=/usr/local/zabbix/logs/zabbix_server.pid  
DBHost=localhost  
DBName=zabbix  
DBUser=zabbix  
DBPassword=zabbix  
DBPort=3306  
FpingLocation=/usr/local/sbin/fping  
#mkdir -p /usr/local/zabbix/logs  
#chown -R zabbix:zabbix /usr/local/zabbix
```

4. 启动 Zabbix Server

[回目录](#)

```
# /usr/local/zabbix/sbin/zabbix_server -c /usr/local/zabbix/etc/zabbix_server.conf
/usr/local/zabbix/sbin/zabbix_server: error while loading shared libraries: libmysqlclient.so.20: cannot open shared object file: No such file or directory
```

解决方法

在 `ld.so.conf` 中加入 `/usr/local/mysql/lib`

```
[root@mysqla zabbix-3.2.1]# cat /etc/ld.so.conf
include ld.so.conf.d/*.conf
/usr/local/mysql/lib
/usr/local/lib
[root@mysqla zabbix-3.2.1]# ldconfig
```

再次执行 `zabbix_server` 启动

```
# /usr/local/zabbix/sbin/zabbix_server -c /usr/local/zabbix/etc/zabbix_server.conf
[root@mysqla init.d]# netstat -anpt |grep zabbix_server
tcp      0  0 0.0.0.0:10051    0.0.0.0:*      LISTEN      27199/zabbix_server
```

5.添加开机启动脚本

[回目录](#)

```
[root@mysqla ~]# cd zabbix-3.2.1/
# cp misc/init.d/fedora/core/zabbix_server /etc/rc.d/init.d/zabbix_server
# cp misc/init.d/fedora/core/zabbix_agentd /etc/rc.d/init.d/zabbix_agentd
[root@mysqla zabbix-3.2.1]# chmod +x /etc/rc.d/init.d/zabbix_server
[root@mysqla zabbix-3.2.1]# chmod +x /etc/rc.d/init.d/zabbix_agentd
[root@mysqla zabbix-3.2.1]# chkconfig --add zabbix_server
[root@mysqla zabbix-3.2.1]# chkconfig --add zabbix_agentd
[root@mysqla zabbix-3.2.1]# chkconfig zabbix_server on
[root@mysqla zabbix-3.2.1]# chkconfig zabbix_agentd on
```

修改 zabbix 开机启动脚本中的 zabbix 安装目录

```
vi /etc/rc.d/init.d/zabbix_server #编辑服务端配置文件
BASEDIR=/usr/local/zabbix/ #zabbix安装目录
PIDFILE=/usr/local/zabbix/logs/$BINARY_NAME.pid #pid文件路径
:wq! #保存退出
vi /etc/rc.d/init.d/zabbix_agentd #编辑客户端配置文件
BASEDIR=/usr/local/zabbix/ #zabbix安装目录
PIDFILE=/usr/local/zabbix/logs/$BINARY_NAME.pid #pid文件路径
:wq! #保存退出
```

```
[root@mysqla zabbix-3.2.1]# systemctl daemon-reload
[root@mysqla zabbix-3.2.1]# /etc/init.d/zabbix_server stop
Stopping zabbix_server (via systemctl): [ OK ]
[root@mysqla zabbix-3.2.1]# netstat -anpt | grep zabbix
[root@mysqla zabbix-3.2.1]# /etc/init.d/zabbix_server start
Starting zabbix_server (via systemctl): [ OK ]
[root@mysqla zabbix-3.2.1]# netstat -anpt | grep zabbix
tcp 0 0 0.0.0.0:10051 0.0.0.0:* LISTEN 28106/zabbix_server
```

6. 设置防火墙规则

[回目录](#)

```
[root@mysqla zabbix-3.2.1]# firewall-cmd --permanent --add-port=10051/tcp
success
[root@mysqla zabbix-3.2.1]# firewall-cmd --reload
success
```

ii. 配置 zabbix web 页面

[回目录](#)

1. 在安装目录将 **frontends** 拷贝到指定的 **web root**

[回目录](#)

```
[root@mysqla ~]# cd /root/zabbix-3.2.1/
[root@mysqla zabbix-3.2.1]# cp -r frontends/php/ /usr/local/nginx1.10/html/zabbix
[root@mysqla zabbix-3.2.1]# chown -R www:www /usr/local/nginx1.10/html/zabbix/
```

注： /usr/local/nginx/html 为 Nginx 默认站点目录 www 为 Nginx 运行账户
注： PHP 需要至少开启扩展：

- gd
- bcmath
- ctype
- libXML
- xmlreader
- xmlwriter
- session
- sockets
- mbstring

- gettext
- mysql

如下，查看是否包括了上面所提到的扩展模块

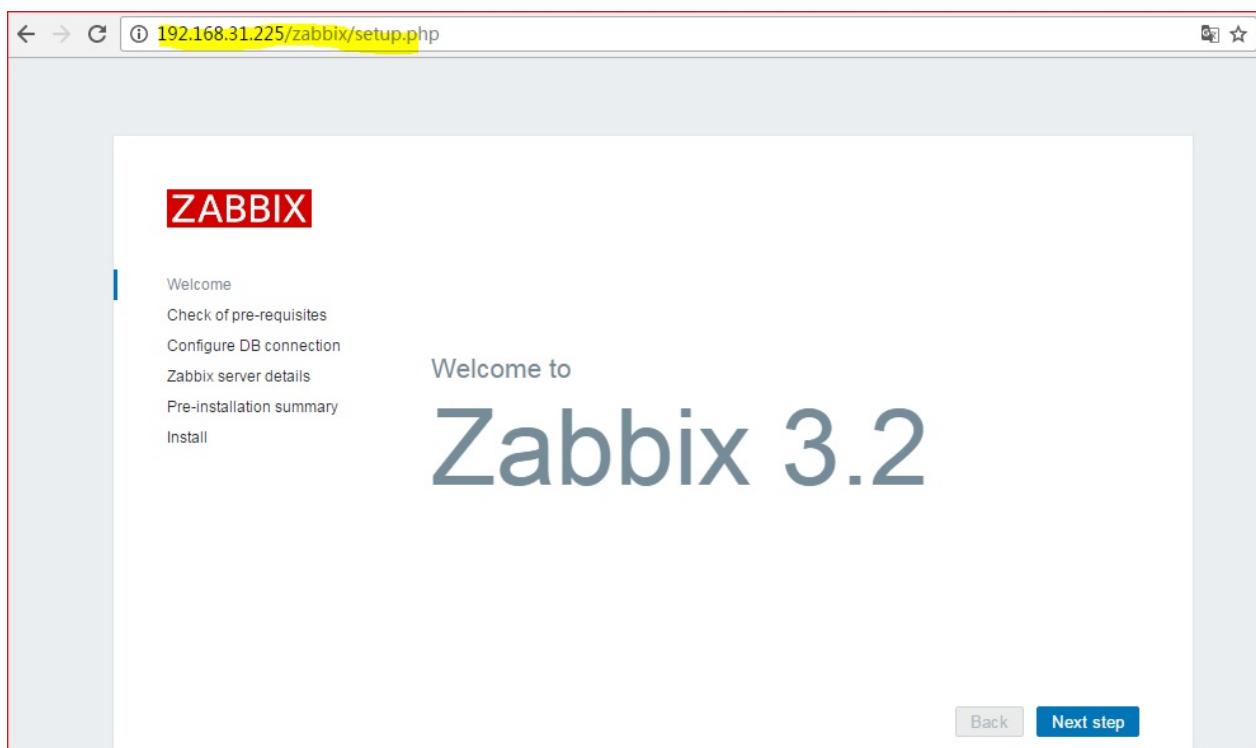
```
[root@mysqla ~]# /usr/local/php5.6/bin/php -m
[PHP Modules]
bcmath
bz2
Core
ctype
curl
date
dom
ereg
fileinfo
filter
ftp
gd
gettext
hash
iconv
json
libxml
mbstring
mcrypt
mhash
mysql
mysqli
openssl
pcntl
pcre
PDO
pdo_sqlite
Phar
posix
Reflection
session
shmop
SimpleXML
```

```
soap  
sockets  
SPL  
sqlite3  
standard  
sysvsem  
tokenizer  
xml  
xmlreader  
xmlrpc  
xmlwriter  
zip  
zlib
```

2. 安装 web

回目录

访问Web界面 <http://192.168.31.225/zabbix> ,进行相关 web 配置,配置完成后使用默认用户 admin (密码: zabbix)登陆即可进入 ZABBIX WEB 安装画面



进入 ZABBIX 检测画面

The screenshot shows the Zabbix setup interface at the 'Check of pre-requisites' step. On the left, a sidebar lists navigation options: Welcome, Check of pre-requisites (which is selected), Configure DB connection, Zabbix server details, Pre-installation summary, and Install. The main content area is titled 'Check of pre-requisites' and displays a table of PHP configuration settings against their required values. Most entries show 'OK' status, except for 'PHP version' which is '5.6.27' (Required: 5.4.0). Below the table are 'Back' and 'Next step' buttons.

	Current value	Required
PHP version	5.6.27	5.4.0 OK
PHP option "memory_limit"	128M	128M OK
PHP option "post_max_size"	16M	16M OK
PHP option "upload_max_filesize"	2M	2M OK
PHP option "max_execution_time"	300	300 OK
PHP option "max_input_time"	300	300 OK
PHP option "date.timezone"	PRC	OK
PHP databases support	MySQL SQLite3	OK
PHP bcmath	on	OK
PHP mbstring	on	OK

对数据库进行设置，这里如果数据库在本地的话端口可以使用 0

The screenshot shows the Zabbix setup interface at the 'Configure DB connection' step. The sidebar remains the same. The main content area is titled 'Configure DB connection' and contains instructions: 'Please create database manually, and set the configuration parameters for connection to this database. Press "Next step" button when done.' It includes fields for Database type (set to MySQL), Database host (localhost), Database port (3306), Database name (zabbix), User (zabbix), and Password (*****). Below the form are 'Back' and 'Next step' buttons.

点击 Next step 一下步进行zabbix server 细节的设置 这一步可以默认

ZABBIX

Zabbix server details

Please enter the host name or host IP address and port number of the Zabbix server, as well as the name of the installation (optional).

Welcome	
Check of pre-requisites	Host <input type="text" value="localhost"/>
Configure DB connection	Port <input type="text" value="10051"/>
Zabbix server details	Name <input type="text"/>
Pre-installation summary	
Install	

[Back](#) [Next step](#)

点击安装，安装完成后 如果没有错误就会进入完成画面 完成画面有设置文件放在服务器的位置，请记下来

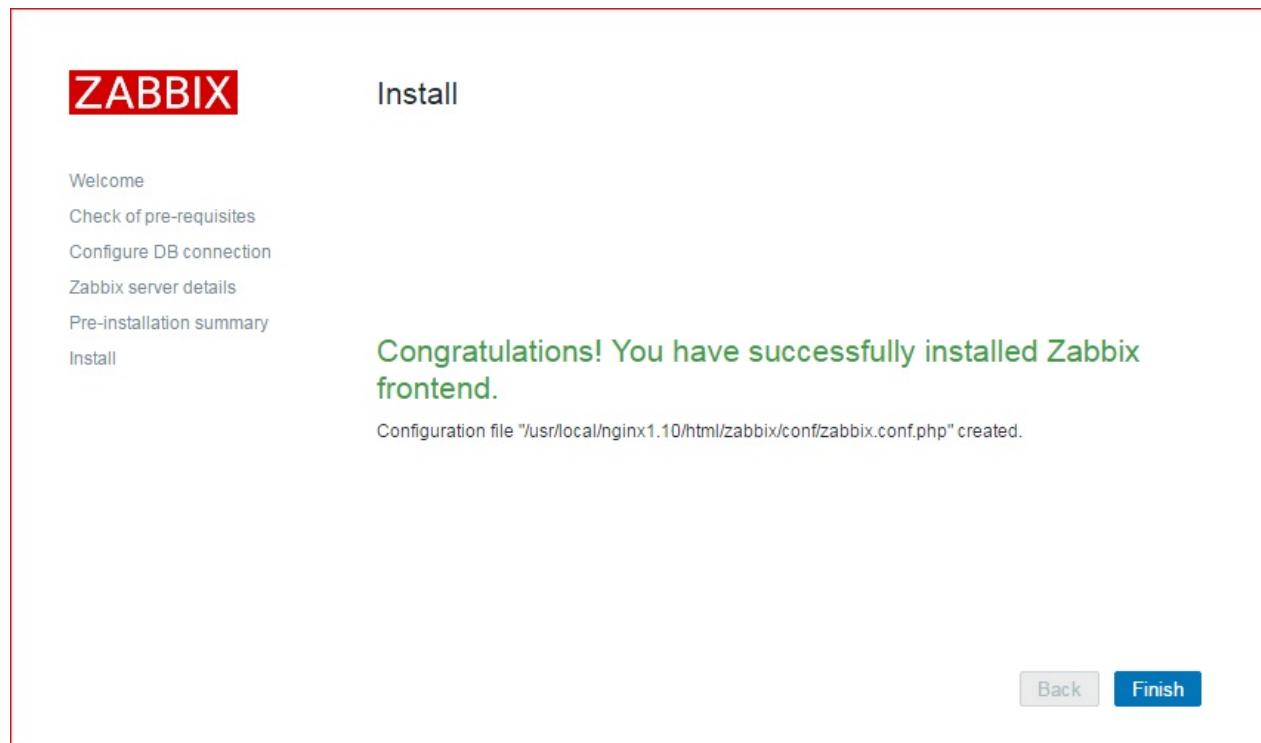
ZABBIX

Pre-installation summary

Please check configuration parameters. If all is correct, press "Next step" button, or "Back" button to change configuration parameters.

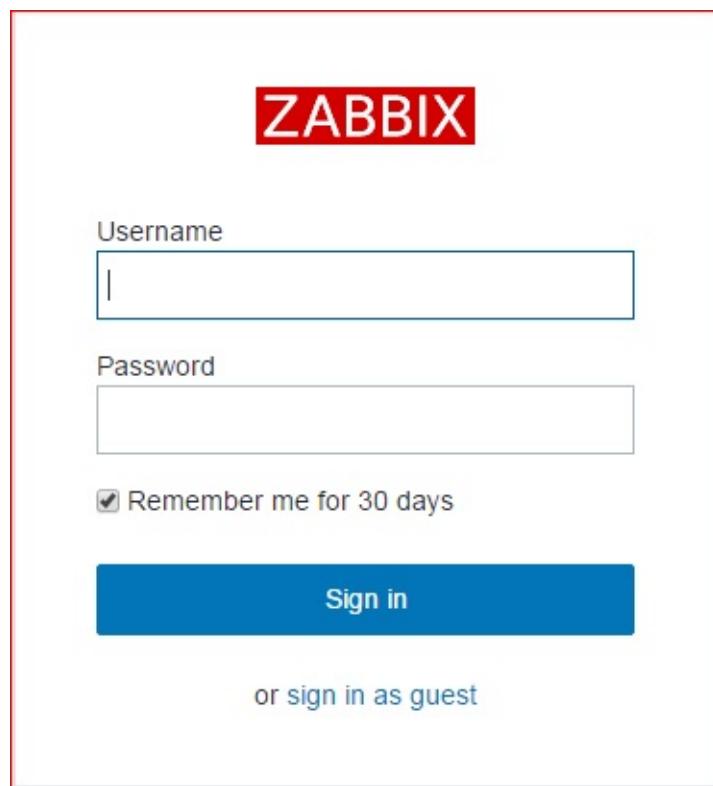
Welcome	Database type <input type="text" value="MySQL"/>
Check of pre-requisites	Database server <input type="text" value="localhost"/>
Configure DB connection	Database port <input type="text" value="3306"/>
Zabbix server details	Database name <input type="text" value="zabbix"/>
Pre-installation summary	Database user <input type="text" value="zabbix"/>
Install	Database password <input type="text" value="*****"/>
	Zabbix server <input type="text" value="localhost"/>
	Zabbix server port <input type="text" value="10051"/>
	Zabbix server name <input type="text"/>

[Back](#) [Next step](#)



点击 Finish 完成

使用 Admin 用默认密码 zabbix 登录



登录后画面如下

至此 ZABBIX 的基础安装完成

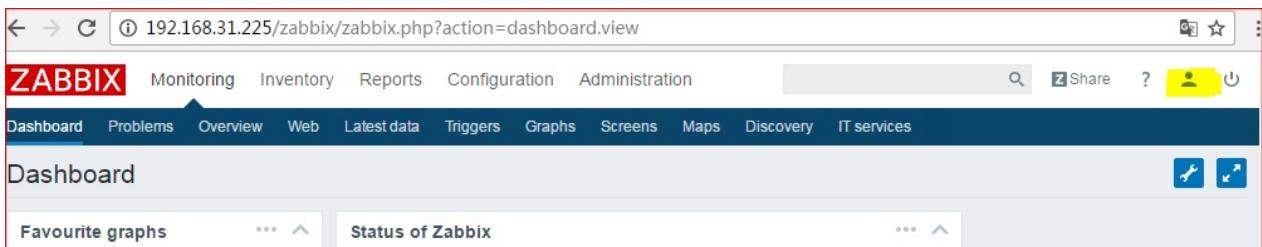
3. 显示简体中文界面

[回目录](#)

在安装数据库时已经将 zabbix 库设置了 utf-8 字符首先确定 zabbix 开启了中文支持功能：登录到 zabbix 服务器的数据目录下（前面部署的 zabbix 数据目录是 /usr/local/nginx1.10/html/zabbix/），打开 locales.inc.php 文件

```
[root@mysqla include]# pwd
/usr/local/nginx1.10/html/zabbix/include
[root@mysqla include]# vi locales.inc.php
function getLocales() {
    return [
        'en_GB' => ['name' =>_('English (en_GB)'), 'display' => true],
        'en_US' => ['name' =>_('English (en_US)'), 'display' => true],
        'bg_BG' => ['name' =>_('Bulgarian (bg_BG)'), 'display' => false],
        'zh_CN' => ['name' =>_('Chinese (zh_CN)'), 'display' => true],
        'zh_TW' => ['name' =>_('Chinese (zh_TW)'), 'display' => false],
        'cs_CZ' => ['name' =>_('Czech (cs_CZ)'), 'display' => true],
        'nl_NL' => ['name' =>_('Dutch (nl_NL)'), 'display' => false],
    ];
}
```

登陆 zabbix 后，点击右上角的“用户”图标，将语言设置为“中文”：



按照如图选择 Chinese (zh_CN)，点击 update

User profile: Zabbix Administrator

User Media Messaging

Password	<input type="button" value="Change password"/>
Language	Chinese (zh_CN) ▾
Theme	System default ▾
Auto-login	<input type="checkbox"/>
Auto-logout (min 90 seconds)	<input type="text"/> 900
Refresh (in seconds)	<input type="text"/> 30
Rows per page	<input type="text"/> 50
URL (after login)	<input type="text"/>
<input type="button" value="Update"/> <input type="button" value="Cancel"/>	

4. 解决 zabbix 绘图中出现中文乱码问题

回目录

a. 从 windows 下控制面板->字体->选择一种中文字库例如 楷体



b. 把它拷贝到 zabbix 的 web 端的 fonts 目录下例

如： /usr/local/nginx1.10/html/zabbix/fonts/，并且把 TTF 后缀改为 ttf

```
[root@mysqla fonts]# pwd  
/usr/local/nginx1.10/html/zabbix/fonts  
[root@mysqla fonts]# ls  
DejaVuSans.ttf  simkai.ttf
```

并且将之前的字体文件 DejaVuSans.ttf 移动到别处

c.然后，接着修改代码 include/defines.inc.php 文件中的字体配置，将里面关于字体设置从 DejaVuSans 替换成 simkai

vi 替换技巧： %s/DejaVuSans/simkai 其中： simkai 为字库名字,不包含 ttf 后缀

```
# cd/usr/local/nginx1.10/html/zabbix  
# vi include/defines.inc.php
```

在 vi 编辑器的末行模式下输入 %s/DejaVuSans/simkai 这样，修改后， zabbix 监控图形中的中文字就不会出现乱码了

IV. zabbix 客户端的安装

[回目录](#)

既然要监控我们就要添加要监控的主机，在添加主机之前我们首先要在被检测主机上面安装 agent，安装 agent 比较简单，我们也是按照安装 server 的流程，下载软件包，在编译的时候，我们只选择 agent

192.168.31.250 作为 zabbix 的被监控端，提供 web 和 mysql 应用

i. 安装 Zabbix，配置 Zabbix agent

[回目录](#)

```
[root@server1 ~]# tar zxf zabbix-3.2.1.tar.gz
[root@server1 ~]# cd zabbix-3.2.1/
[root@server1 zabbix-3.2.1]# ./configure --prefix=/usr/local/zabbix --enable-agent
[root@server1 zabbix-3.2.1]# make&& make install
[root@server1 zabbix-3.2.1]# cp misc/init.d/fedora/core/zabbix_agentd /etc/init.d/
[root@server1 zabbix-3.2.1]# mkdir -p /usr/local/zabbix/logs
[root@server1 zabbix-3.2.1]# groupadd zabbix
[root@server1 zabbix-3.2.1]# useradd -g zabbix zabbix
[root@server1 zabbix-3.2.1]# chown -R zabbix:zabbix /usr/local/zabbix/
```

配置开机自动启动

```
[root@server1 ~]# chkconfig --add zabbix_agentd
[root@server1 ~]# chkconfig zabbix_agentd on
```

1.修改 zabbix 开机启动脚本中的 zabbix 安装目录

[回目录](#)

```
vi /etc/rc.d/init.d/zabbix_agentd #编辑客户端配置文件
BASEDIR=/usr/local/zabbix/ #zabbix安装目录
PIDFILE=/usr/local/zabbix/logs/$BINARY_NAME.pid #pid文件路径
:wq! #保存退出
[root@server1 zabbix-3.2.1]# systemctl daemon-reload
```

2.编辑 zabbix_agentd.conf

[回目录](#)

```
[root@server1 zabbix-3.2.1]# vi /usr/local/zabbix/etc/zabbix_agentd.conf
```

内容如下

```
[root@server1 zabbix-3.2.1]# grep -v "^#" /usr/local/zabbix/etc/zabbix_agentd.conf | grep -v "^\$"
PidFile=/usr/local/zabbix/logs/zabbix_agentd.pid
LogFile=/usr/local/zabbix/logs/zabbix_agentd.log
Server=192.168.31.225
ListenPort=10050
ServerActive=192.168.31.225
Hostname=192.168.31.250
Timeout=15
Include=/usr/local/zabbix/etc/zabbix_agentd.conf.d/
```

注：其中 Server 和 ServerActive 都指定 zabbixserver 的 IP地址，不同的是，前者是被动后者是主动。也就是说 Server 这个配置是用来允许 192.168.31.225 这个 ip 来我这取数据。而 serverActive 的 192.168.31.225 的意思是，客户端主动提交数据给他。Hostname=XXX ,这个定义的名字必须和 web 页面里面 host 的名字一样。

3. 启动 zabbix_agentd

```
[root@server1 ~]# /etc/init.d/zabbix_agentd start
Starting zabbix_agentd (via systemctl): [OK]
[1]
[root@server1 ~]# netstat -anpt | grep zabbix_agentd
tcp        0      0 0.0.0.0:10050          0.0.0.0:*
      LISTEN      12926/zabbix_agentd
[root@server1 ~]# firewall-cmd --permanent --add-port=10050/tcp
success
[root@server1 ~]# firewall-cmd --reload
success
```

到此， zabbix3.2.1 监控系统的基本环境安装完成。

ii. 实现 zabbix 添加监测项

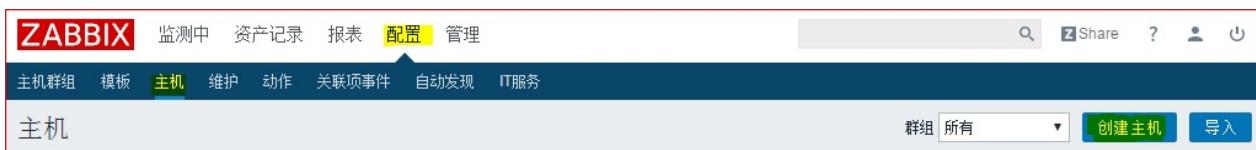
回目录

添加对Linux主机的监控

在浏览器中打开：`http://192.168.31.225/zabbix` 登录 zabbix，先在“配置”-“主机”里添加主机监控，监控 os 资源：内存，cpu，io，负载，带宽等。

1. 登录 zabbix，先在“配置”-“主机”里单击“创建主机”

回目录



2. 点击“主机”标签，填写相关信息

回目录

主机名称	192.168.31.250
可见的名称	server1-192.168.31.250
群组 在...群组之中	Linux servers
其它群组	Discovered hosts Hypervisors Templates Virtual machines Zabbix servers
新的群组	<input type="text"/>
agent代理程序的接口	IP地址: <input type="text" value="192.168.31.250"/> DNS名称: <input type="text"/> 连接到: <input type="radio"/> IP地址 <input type="radio"/> DNS 端口: 10050 <input checked="" type="radio"/> 移除
SNMP接口	<input type="button" value="添加"/>
JMX接口	<input type="button" value="添加"/>
IPMI接口	<input type="button" value="添加"/>

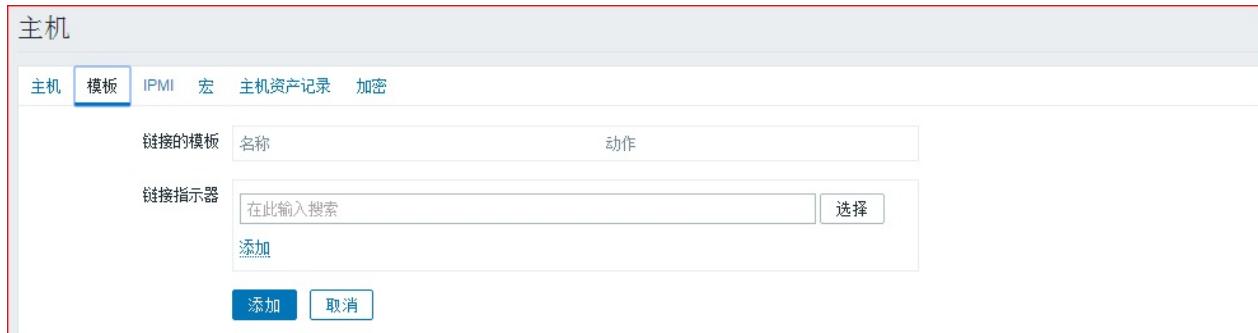


注意：下图中的“主机名称”要和 `zabbix_agentd.conf` 文件中设置的 `Hostname` 后面的名称一致！

- 主机名称： 192.168.31.250
- 群组： Linux servers
- agent 代理程序接口 ip : 192.168.21.128
- 已启用：勾选
- 其它选项默认即可

3. 切换到模板

[回目录](#)



选择

选择： Template OS Linux

点添加

主机

主机 模板 IPMI 宏 主机资产记录 加密

链接的模板 名称 动作
Template OS Linux 取消链接

链接指示器 在此输入搜索 选择
添加

添加 取消

添加

主机

主机 模板 IPMI 宏 主机资产记录 加密

链接的模板 名称 动作
Template OS Linux 取消链接

链接指示器 在此输入搜索 选择
添加

添加 取消

至此，Zabbix 监控 Linux 主机设置完成。

ZABBIX 监测中 资产记录 报表 配置 管理

主机群组 模板 主机 维护 动作 关联项事件 自动发现 IT服务

细节 已添加主机

主机 群组 所有 创建主机 导入

名称 DNS IP地址 端口 过滤器 ▲

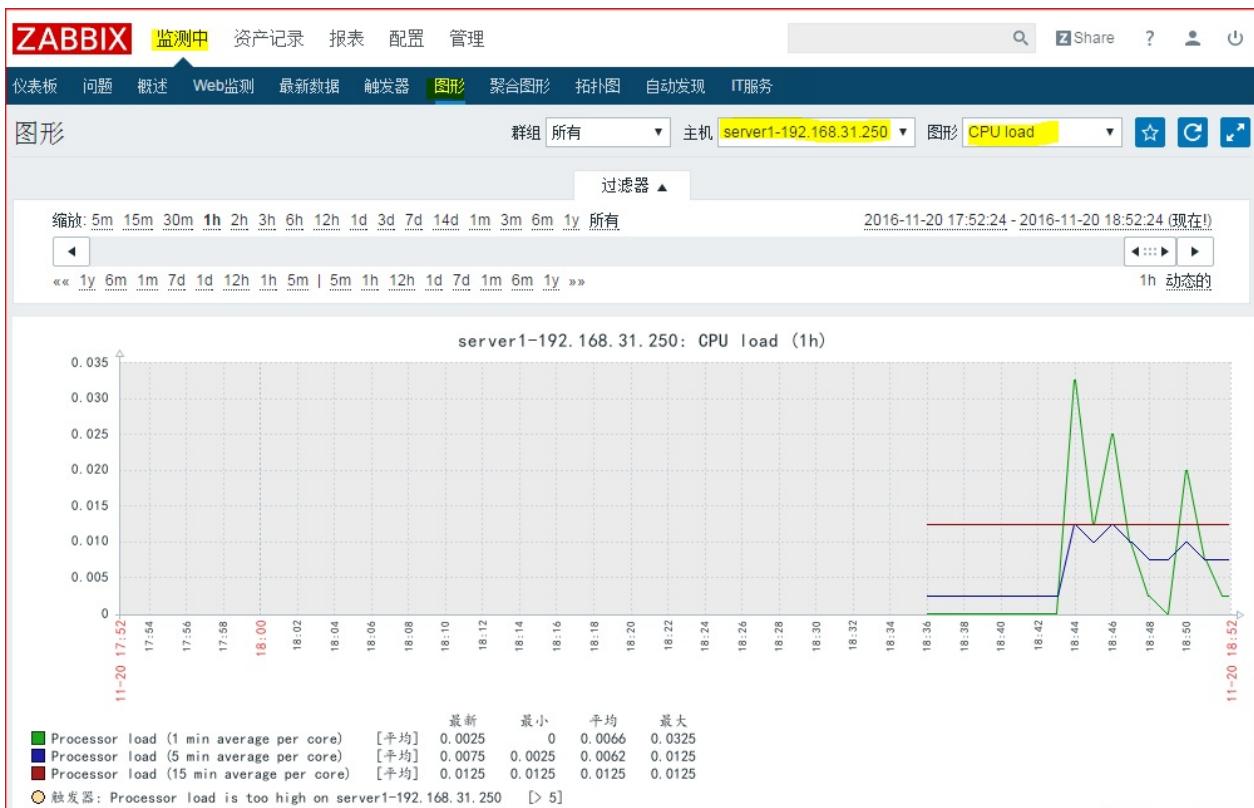
应用 重设

名称	应用集	监控项	触发器	图形	自动发现	Web监测	接口	模板	状态	可用性	agent加密	信息
server1-192.168.31.250	应用集 10	监控项 32	触发器 15	图形 5	自动发现 2	Web监测 192.168.31.250:10050	Template OS Linux (Template App Zabbix Agent)	已启用	ZBX SNMP JMX IPMI	无		

4. 监控结果

回目录

配置过一段时间后，观察下监控图效果出来了没



zabbix3.0 server 已自带 mysql 的模板了，只需配置好 agent 客户端，然后在 web 端给主机增加模板就行了。

iii. Zabbix_agent 客户端操作

[回目录](#)

1. mysql 授权

[回目录](#)

首先在客户端的 mysql 里添加权限，即本机使用 zabbix 账号连接本地的 mysql

```
mysql> grant all on *.* to zabbix@'localhost' identified by "123456";
mysql> flush privileges;
```

2. 在 `zabbix_agent` 服务目录下创建 `.my.cnf` 连接文件 `zabbix` 安装目录是 `/usr/local/zabbix`

[回目录](#)

```
[root@server1 ~]# cd /usr/local/zabbix/etc/
[root@server1 ~]# cat .my.cnf
[client]
user=zabbix
password=123456
```

注意：

如果在数据库 `grant` 授权时，针对的是 `localhost`，这个 `.my.cnf` 里面就不用加 `host` 参数了【如上配置】

但如果 `grant` 授权时针对的是本机的 `ip`（如 `192.168.31.250`），那么在 `.my.cnf` 文件里就要加上 `host` 参数进行指定了

即在 `.my.cnf` 文件就要加上

```
host=192.168.31.250
socket= /usr/local/mysql/mysql.sock
user=zabbix
password=123456
```

3. 配置 MySQL 的 key 文件

[回目录](#)

这个可以从 `zabbix3.2` 安装时的解压包里拷贝过来

从 `zabbix_server` 服务端安装时解压目录 `/root/zabbix-3.2.1/conf/zabbix_agentd/userparameter_mysql.conf` 拷贝到 `zabbix_agent` 客户端上
的 `/usr/local/zabbix/etc/zabbix_agentd.conf.d/` 目录下

```
[root@server1 ~]# cd /usr/local/zabbix/etc/zabbix_agentd.conf.d/  
[root@server1 zabbix_agentd.conf.d]# pwd  
/usr/local/zabbix/etc/zabbix_agentd.conf.d  
# cp /root/zabbix-3.2.1/conf/zabbix_agentd/userparameter_mysql.cnf ./  
[root@server1 zabbix_agentd.conf.d]# ls userparameter_mysql.conf  
  
userparameter_mysql.conf
```

然后查看 `userparameter_mysql.conf` 文件，看到类似
`HOME=/var/lib/zabbix` 的路径设置，把路径全都替换
为 `/usr/local/zabbix/etc/`，也就是上面的 `my.cnf` 文件所在的目录路径。

另外，注意 `userparameter_mysql.conf` 文件里的 `mysql` 命令路径（提前做好 `mysql` 的系统环境变量，以防 `mysql` 命令不被系统识别）如下：

```
[root@server1 zabbix_agentd.conf.d]# cat userparameter_mysql.conf
#
# For all the following commands HOME should be set to the directory that has .my.cnf file with password information.

# Flexible parameter to grab global variables. On the frontend side, use keys like mysql.status[Com_insert].
# Key syntax is mysql.status[variable].
UserParameter=mysql.status[*],echo "show global status where Variable_name='$1';" | HOME=/usr/local/zabbix/etc/ mysql -N | awk '{print $$2}'

# Flexible parameter to determine database or table size. On the frontend side, use keys like mysql.size[zabbix,history,data].
# Key syntax is mysql.size[<database>,<table>,<type>].
# Database may be a database name or "all". Default is "all".
# Table may be a table name or "all". Default is "all".
# Type may be "data", "index", "free" or "both". Both is a sum of data and index. Default is "both".
# Database is mandatory if a table is specified. Type may be specified always.
# Returns value in bytes.
# 'sum' on data_length or index_length alone needed when we are getting this information for whole database instead of a single table
UserParameter=mysql.size[*],bash -c 'echo "select sum($(case \"$3\" in both|\"\") echo \"data_length+index_length\";; data|index) echo \"$3_length\";; free) echo \"data_free\";; esac)) from information_schema.tables$([[ \"$1\" = "all" || ! \"$1\" ]] || echo " where table_schema=\"$1\"")$([[ \"$2\" = "all" || ! \"$2\" ]] || echo "and table_name=\"$2\"");" | HOME=/usr/local/zabbix/etc/ mysql -N'
UserParameter=mysql.ping,HOME=/usr/local/zabbix/etc/ mysqladmin ping | grep -c alive
UserParameter=mysql.version,mysql -V
```

修改内容

- HOME=/usr/local/zabbix/etc/ mysql -N | awk '{print \$\$2}'
- HOME=/usr/local/zabbix/etc/ mysql -N'

- HOME=/usr/local/zabbix/etc/ mysqladmin ping | grep -c alive

4. 重启 zabbix_agent

[回目录](#)

按照上面修改完之后检查一下，然后重启 zabbix_agent

```
[root@server1 ~]# pkill -9 zabbix_agent
[root@server1 ~]# netstat -anpt | grep zabbix_agent
[root@server1 ~]# /usr/local/zabbix/sbin/zabbix_agentd
[root@server1 ~]# netstat -anpt | grep zabbix_agent
tcp      0        0 0.0.0.0:10050    0.0.0.0:*      LISTEN      15400/za
bbix_agentd
```

接着在 zabbix_server 服务端进行命令行测试 [下面的 192.168.31.250 是客户端的 ip]

```
[root@mysqla ~]#/usr/local/zabbix/bin/zabbix_get -s 192.168.31.2
50 -p 10050 -k "mysql.status[Uptime]"
12593
[root@mysqla ~]#
```

如果出现类似这一串 key 的数字，就说明配置 ok ，服务端能监控到客户端的 mysql 数据了！成功啦，之后在监控界面增加主机对应的 MySQL 模板就 ok 了。

注： zabbix_get 命令介绍 参数说明：

- -s --host : 指定客户端主机名或者 IP
- -p --port : 客户端端口，默认 10050
- -I --source-address : 指定源 IP ，写上 zabbix server 的 ip 地址即可，一般留空，服务器如果有多 ip 的时候，你指定一个。
- -k --key : 你想获取的 key

zabbix_get 获取数据

获取负载

```
# zabbix_get -s 192.168.31.250 -p 10050 -k "system.cpu.load[all,avg15]"
```

获取主机名

```
# zabbix_get -s 192.168.31.250 -p 10050 -k "system.hostname"
```

5. 登录zabbix监控界面，在“配置”-“模板”里可以看到自带的mysql监控模板

[回目录](#)

名称	应用集	监控项	触发器	图形	聚合图形	自动发现	Web监测	链接的模板	已链接到
Template App FTP Service	应用集 1	监控项 1	触发器 1	图形	聚合图形	自动发现	Web监测		
Template App HTTP Service	应用集 1	监控项 1	触发器 1	图形	聚合图形	自动发现	Web监测		
Template App HTTPS Service	应用集 1	监控项 1	触发器 1	图形	聚合图形	自动发现	Web监测		
Template App IMAP Service	应用集 1	监控项 1	触发器 1	图形	聚合图形	自动发现	Web监测		
Template App LDAP Service	应用集 1	监控项 1	触发器 1	图形	聚合图形	自动发现	Web监测		
Template App MySQL	应用集 1	监控项 14	触发器 1	图形 2	聚合图形 1	自动发现	Web监测		

在相应主机监控配置里添加 mysql 的监控模版（ zabbix 自带的）即可。

ZABBIX 监测中 资产记录 报表 配置 管理

所有主机 / server1-192.168.31.250 已启用 ZBX SNMP JMX IPMI 应用集 11 监控项 62 触发器 20 图形 12 自动发现规则 2 Web 场景

主机 模板 IPMI 宏 主机资产记录 加密

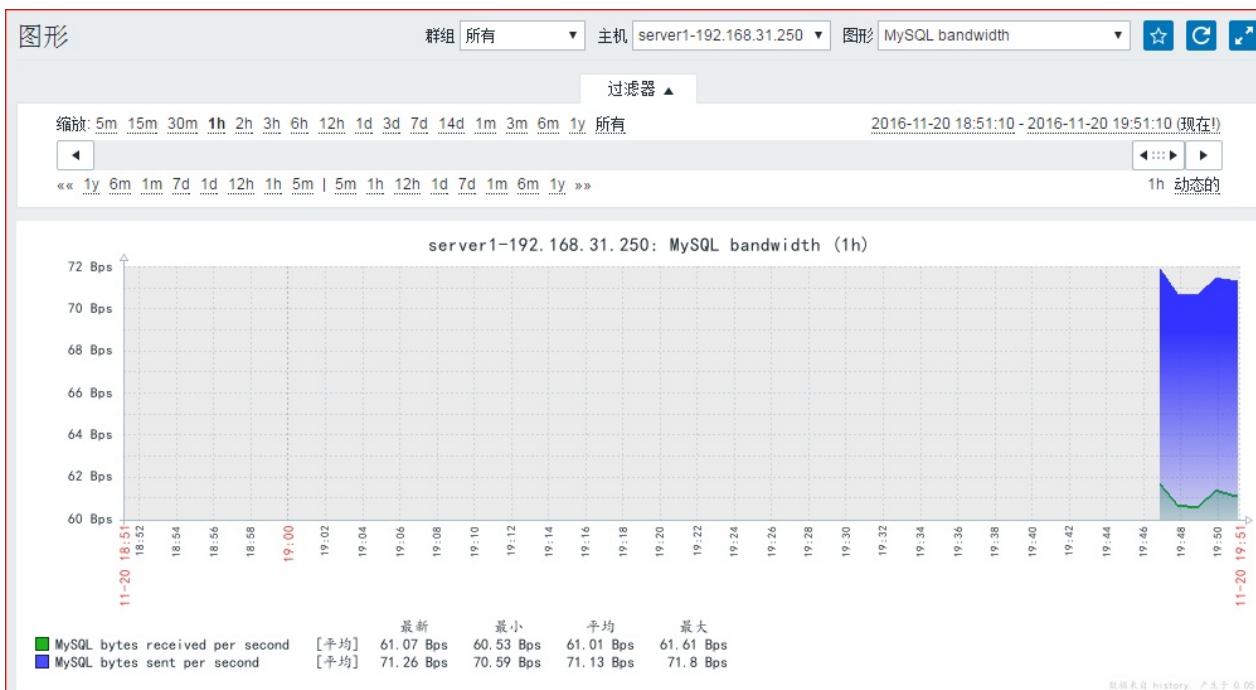
链接的模板	名称	动作
	Template App MySQL	取消链接 取消链接并清理
	Template OS Linux	取消链接 取消链接并清理

链接指示器 在此输入搜索 [选择] 添加

更新 克隆 全克隆 删除 取消

下面是 mysql 监控项的效果图

[1] 监控 mysql 的带宽：在 zabbix 前端可以实时查看 mysql 发送接收的字节数。其中 bytes received 表示从所有客户端接收到的字节数，bytes sent 表示发送给所有客户端的字节数。



[2] 监控 mysql 的操作状态：在 zabbix 前端可以实时查看各种 SQL 语句每秒钟的操作次数。



邮件报警