# 高并发和大流量解决方案

**PHP如何解决网站大流量与高并发的问题？**

**高并发架构相关概念**

**高并发解决方案案例**

**并发**

并发，在操作系统中，是指一个时间段中有几个程序都处于已启动运行到运行完毕之间，且这几个程序都是在同一个处理机上运行，但任何一个时刻点上只有一个程序在处理机上运行。

**我们说的高并发是什么？**

上面的定义明显不是我们通常所言的并发，在互联网时代，所讲的并发，高并发，通常是指并发访问。也就是在某个时间点，有多少个访问同时到来。

通常如果一个系统的日PV在千万以上，有可能是一个高并发的系统，有的公司完全不走技术路线，全靠机器堆，这不在我们的讨论范围。

**高并发的问题，我们具体该关系什么？**

QPS：每秒钟请求或者查询的数量，在互联网领域，指每秒响应请求数(指HTTP请求)；

**吞吐量：**单位时间内处理的请求数量（通常由QPS与并发数决定）

**响应时间：**从请求出发到收到响应花费的时间。例如系统处理一个HTTP请求需要100ms，这个100ms就是系统的响应时间。

**PV:**综合浏览量(Page View),即页面浏览或者点击量，一个访客在24小时内访问的页面数量（从零点到24点访问的页面数量叫pv）

同一个人浏览你的网站同一个页面，只记作一次PV

**UV:**独立访客（UniQue Visitor）,即一定时间范围内相同访客多次访问网站，只计算为1个独立访客。

**带宽：**计算带宽大小需关注两个指标，峰值流量和页面的平均大小

**日网站带宽=PV/统计时间（换算到秒）\*平均页面大小（单位KB）\* 8**

**峰值一般是平均值的倍数，根据实际情况来定**

**QPS不等于 并发连接数**

**QPS是每秒HTTP请求数量，并发连接数是系统同时处理的请求数量**

**(总PV数 \* 80%)/(6小时秒数 \* 20%)=峰值每秒请求数(QPS)**

**80%的访问量集中在20%的时间**

**压力测试**

**测试能承受最大并发**

**测试最大承受的QPS**

**常用性能测试工具**

**Ab、wrk、http\_load、Web Bench、Siege、Apache JMeter**

**AB**

全称是apache benchmark、是apache官方推出的工具创建多个并发访问线程，模拟多个访问者同时堆某一URL地址进行访问。它的测试目标是基于URL的，因此，它即可以用来测试apache的负载压力，也可以测试nginx、lighthttp、tomcat、IIS等其它Web服务器的压力。

**ab的使用**

模拟并发请求100次，总共请求5000次

ab -c 100 -n 5000 待测试网站

yum -y install httpd-tools #ab的安装

ab -c 100 -n 5000 <http://121.42.60.3/phpinfo.php> #ab工具的用法

ab测试出来的比较重要的值

Complete requests: 5000 #请求了5000次

Failed requests: 2946 #失败了2946次

(Connect: 0,Receive: 0,Length: 2946,Exceptions: 0)

Requests per Second : 505.36 [#/sec] (mean) #QPS值

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

50% 197 #百分之五十的页面平均响应时间是 197 毫秒

100% 241 （longest request） 最长响应时间是241毫秒

**注意事项**

测试机器与被测试机器分开

不用对线上服务做压力测试

观察测试工具ab所在机器，以及被测试的前端机的CPU、内存、网络等都不超过最高限度的75%

**QPS达到极限**

随着QPS的增长，每个阶段需要根据实际情况来进行优化，优化的方案也与硬件条件，网络带宽息息相关。

**QPS达到50**

可以称之为小型网站，一般的服务器就可以应付

**QPS达到100**

假设关系型数据库的每次请求在0.01秒完成

假设单页面只有一个SQL查询，那么100QPS意味着1秒钟完成100次请求，但是此时我们并不能保证数据库查询能完成100次

方案：数据库缓存层，数据库的负载均衡

**QPS达到800**

假设我们使用百兆带宽，意味着网站出口的实际带宽是8M左右假设每个页面只有10K，在这个并发条件下，百兆带宽已经吃完

方案：CDN加速，负载均衡

**QPS达到1000**

假设使用Memcache缓存数据库查询数据，每个页面对Memcache的请求大于直接对DB的请求

Memcache的悲观并发数在2W左右，但有可能在之前内网带宽已经吃光，表现出不稳定。

方案：静态HTML缓存

**QPS达到2000**

这个级别下，文件系统访问锁都成为了灾难

方案：做业务分离，分布式存储

**流量优化**

防盗链处理

就是把一些恶意的请求拒之门外，例如有A,B两个站A站想用B站的资源它直接在页面里面去嵌入了B站的一些图片啊js啊或者是css啊，本身来说A站并不关心它会消耗什么样的流量，但是对于B站来说如果我们调用了B站的一些图片或js或者css是不是会对它做一些HTTP请求？这样就会走它的流量和带宽。所以本身来说对B站是会有一些不好的影响。所以我们要做一些防盗链的一些处理，这是我们流量的优化。

#### 前端优化

**减少HTTP请求**

比如把js进行合并，css进行合并，图片进行合并这样就减少了http请求，这样来是虽然我们的文件会大一些，但总体我们只会请求一次。

**添加异步请求**

比如说一些不是很重要的数据头一次用户过来在进行请求界面的时候我先不给他展示他需要的时候呢，比如说我们旁边给他放一些事件，通过我们的JS，jquery等待一些前端的第三方库可以去做一些我们的ajax相关的异步的请求这样的话其实对于我们的这个http请求来说就会有一个大幅度的一个性能上的提升。

**启用浏览器缓存和文件压缩**

比如我们可以去启用咱们浏览器去缓存咱们html文件给他设定一些我们的过期时间设定一些我们的这个缓存文件的相关的指纹啊等等一些内容都可以去做！然后还可以将我们的静态资源文件比如说js，比如swf,比如image等等等等一些相关的前端的都可以去做一些过期时间的缓存。我们可以指定一个过期时间的然后把他缓存在我们的浏览器当中，这样的话当我们在进行下一次请求访问的时候，我们就不需要在去请求服务端。我们直接就通过浏览器把我们的缓存读取出来。

对于文件压缩来说我们可以通过一些压缩方式比如说我们把图片压的小一些，展示的时候我们就会下载的更快一些，响应速度也会加快。并且也减少了流量的消耗减少了带宽的消耗。而且我们还可以启用nginx的gcp压缩将我们的文件整体来说给它压的比较小一些。

**CDN加速**

我 们可以把一些前端的文件，前端的资源全部到放到我们的CDN当中然后用户过来访问的时候我们可以就近来进行访问从而提高我们的访问速度并且从一定意义来说也解决了我们带宽不够用的问题，我们可以把数缓存到我们的cdn的节点当中然后我们在用户访问的时候都会去选择就近的一个cdn节点在去访问。从一定意义上来说是不是就不会在去访问我们的真实的服务器。

**建立独立图片服务器**

因为本身来说我们的图片服务器是比较吃I/0的那怎么办呢？我们为了解决它对I/O的损耗我们可以把他根我们的web服务器完全分离开，这样的话web服务器本身I/O呢不会被我们的图片来进行损耗然后我们还可以针对性的对我们的图片服务器来做一些优化。比如说去对我们的的硬盘转速来进行提高我们可以把CPU的计算能力稍微降下来一点。不需要去增强计算这么一些相关的概念。而且我们还可以把图片服务器来做一个集群。

#### 服务端的优化

**页面静态化**

我们可以把现有的服务端的逻辑比如PHP，我们刚刚可以看到咱们去请求html的时候根我们去请求PHP文件的时候是不是完成不一样的QPS。所以我们可以把咱们PHP的一些逻辑把我们PHP里面的一些数据把我们PHP要生成的要显示给用户的一些html的一些内容是不要把它缓存起来！直接缓存成html代码是不是更快！并且对我们的CPU负载对我们的服务器的压力都会减少的很多我们刚刚可以看到，是不是QPS就能看得出来。

**并发处理**

然后我们如果说用户穿透了页面静态化，比如说对于页面来说我慢做了一些静态化但是页面静态化我们是不是要有一个过期时间？对吧！你不可能永远只显示我们的静态数据对不对，那这样的话是不是创建一些动态的内容就没有意义了！但是对于一些实时性要求比较高的来说我们可能在做一些静态化的时候不是特别合理那这时候怎么办呢？我们就要穿透静态化可能我们就绕过静态化来直接访问我们真实的数据了！访问真实数据的时候呢我们可能就需要做一些我们程序上面的并发的处理比如说我们去做一个多线程多进程的一个异步的处理比如说我们去做一个队列的处理对吧！等等一些内容都可以去异步完成我们的数据的处理从而提升的请求的一个响应的速度并且也提升了我们的并发数。所以在这的话对于我们的并发数处理来说，我们在后面的课程当中也会给大家进行讲解，我们会给大家简单的去看一下我们的swoole看一下咱们的队列

**队列处理**

**数据库优化**

**数据库缓存**

**如:**memcached、redis

**分库分表，分区操作**

然后我们如果说绕过了数据库缓存我们还需要在去访问真实的mysql数据的时候这个时候访问量还是比较大怎么办我们可以去做什么呀！分库分表对我们的数据表来做一个什么！拆分我们可做垂直拆分，也可以去做水平拆分这个呢！我们在MYSQL的高可以里面是不是给大家讲过了对不对！我们还可以做什么呀！还可以去做分区操作，那对于这样的内容来说呢如果大家不知道该这么样来操作我们是不是可以借助一些工具啊！对吧比如说mycat啊比如说adetionby啊的一些内容如果大家还是不了解这样的内容该怎么样去做在我的yii2高组建 优化电商优化咱们的这个京东平台的第二个课程里面有咱们详细的讲解包括咱们mysql的主从复制包括我们mysql双主热备包括我们mysql的分库分表啊包括我们mysql的负载均衡啊都有讲到所以在这大家可以去看一下具体的一个实现方式

**读写分离**

读写分离我们可以把一些数据呢！把我们的一些服务器完全分库一些服务器去进行一些mysql的读操作另外一些服务器做一个写的操作。

**负载均衡**

#### Web服务器的优化

负载均衡，可以用nginx的反响代理来实现负载均衡还可以使用4层的LVS软件去实现一个负载均衡都是可以。所以在这里面对于我们的web服务器的负载均衡来说我们在刚才的那门可里面就是我们的yii2高级组件电商平台的这们课里面也有咱们的LVS相关的实现大家也可以去进行一个学习。

# web资源防盗链

什么是防盗链

防盗链的工作原理

防盗链的实现方法

**盗链概念**

盗链是指自己的页面上展示一些并不在自己服务器上的内容

获得他人服务器上的资源地址，绕过别人的资源展示页面，直接在自己的页面上向最终用户提供此内容。

常见的是小站盗用大站的图片、音乐、视频、软件等资源

通过盗链的方法可以减轻自己服务器的负担，因为真实的空间和流量均是来自别人的服务器。

如果我们那一天发现我们的流量慢慢慢慢在消失，但是我们的PV的统计IP的统计很少那这个时候我们就应该去考虑被别人盗链了。

**防盗链概念**

防止别人通过一些技术手段绕过本站的资源展示页面，盗用本站的资源，让绕开本站资源展示页面的资源连接失败。

可以大大减轻服务器及带宽的压力

**工作原理**

通过Referer或者签名，网站可以检测目标网站访问的来源网页、如果是资源文件，则可以跟踪到显示它的网页地址。

一旦检测到来源不是本站即进行阻止或者返回指定的页面

通过计算签名的方式，判断请求是否合法、如果合法则显示，否则返回错误信息。

**Referer**

Nginx 模块ngx\_http\_referer\_module 用于阻挡来源非法的域名请求

Nginx 指令 valid\_referers , 全局变量$invalid\_referer

Valid\_referers none | blocked | server\_names | string....;

**none :**代表referers为空的时候也代表是合法的

**blocked ：**Referer 来源头部不为空，但是里面的值被代理或者防火墙删除了，这些值都不以http://或者https://开头。

**server\_names：**“Referer”来源头部包含当前的server\_names

//当我去访问以 gif,jpg,png,flv,swf,ara,zip结尾的文件的时候，就会做防盗链的处理

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

{

valid\_referers none blocked imooc.com \*.imooc.com;// imooc.com \*.imooc.com 为被允许的源地址

//valid\_referers 有时候需要用nginx -V 查看一下有没有安装 valid\_referers 模块

if($invalid\_referer){//如果来源不合法就显示403页面

rewrite ^/http://www.imooc.com/403.jpg;

}

}

//针对目录的防盗链

location /images/

{

valid\_referers none blocked imooc.com \*.imooc.com;

if($invalid\_referer){

rewrite ^/http://www.imooc.com/403.jpg;

}

}

#### 传统防盗链遇到的问题

伪造referer

可以使用加密签名解决

**加密签名**

使用第三方模块HttpAccessKeyModule实现Nginx防盗链

Accesskey on|off 模块关闭

Accesskey\_hashmethod md5|sha-1签名加密方式

Accesskey\_arg GET参数名称

Accesskey\_signature 加密规则

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

accesskey on;

accesskey\_hashmethod md5;

accesskey\_arg "key";

accesskey\_signature "mypass$remote\_addr";//mypass这样的字符串加上客户端ip通过md5加密然后得到签名

}

**PHP端显示图片时的签名规则**

$sign = md5('mypass'.$\_SERVER['REMOTE\_ADDR']);

echo '<img src="./log\_new.png?sign='.$sign.'">';

# 减少HTTP请求次数

为什么要减少HTTP请求

减少HTTP请求的方式

**性能黄金法则**

只有10%-20%的最终用户响应时间花在接收请求HTML文档上，剩下的80%-90%时间花在HTML文档所引用的所有组件（图片、script、css、flash等等）进行的http请求上。

**如何改善**

改善响应时间的最简单途径就是减少组件的数量，并由此减少HTTP请求的数量。

**HTTP连接产生的开销**

域名解析--TCP连接--发送请求--等待--下载资源--解析时间

**疑问？**

DNS缓存

Keep-Alive

打破谣言

查找DNS缓存也需要时间，多个缓存就要查找多次有可能缓存会被清除

HTTP1.1协议规定请求只能串行发送，也就是说一百个请求必须一次逐个发送，前面的一个请求完成才能开始下个请求

**图片地图**

图片地图允许你在一个图片上关联多个URL。目标URL的选择取决于用户单击了图片的哪个位置。

我们可以通过使用五个分开的图片，然后每个图片对应一个超链接

图片地图允许你在一个图片上关联多个URL。目标URL的选择取决于用户单击了图片上的哪个位置。

我们可以通过使用五个分开的图片，然后每个图片对应一个超链接产生5个HTTP请求，我们的目标是要减少HTTP请求。

将五个图片合并为一张图片，然后以位置信息定位超链接

图片地图允许你在一个图片上关联多个URL。目标URL的选择取决于用户单击了图片上的哪个位置。

把HTTP请求减少为一个，可以保证设计的完整性和功能的齐全性

使用<map><area></area></map>标签

未使用图像地图的例子

<http://stevesouders.com/hpws/imagemap-no.php>

使用了图像地图的例子：

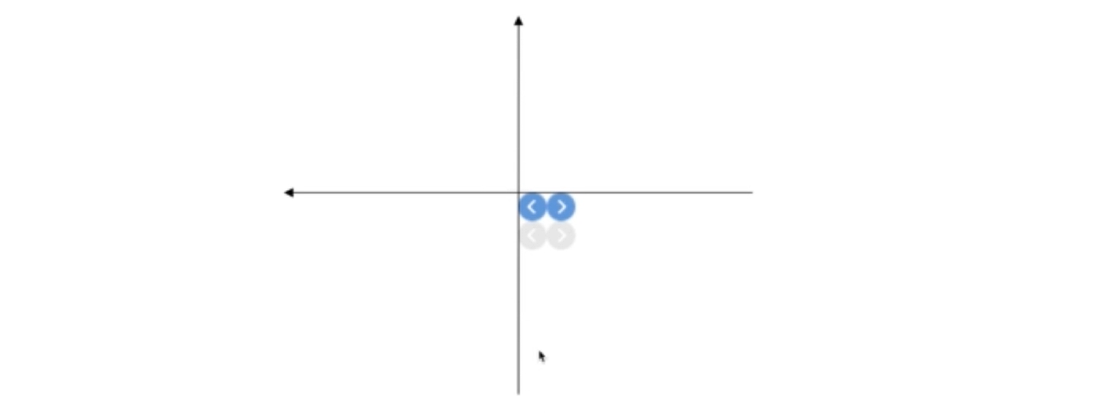
<http://stevesouders.com/hpws/imagemap.php>

**CSS sprites**

CSS Sprites 中文翻译为CSS精灵，通过使用合并图片，通过指定css的backgroud-image和background-position来显示元素。

Background-position:x y;x和y可以写负值也可以写正值，我们可以想象图片的左上方为(0,0)，以(0,0)坐标向右是为负数的x轴，以(0,0)坐标向下是为负数y轴。

**Background-position属性**



**性能影响**

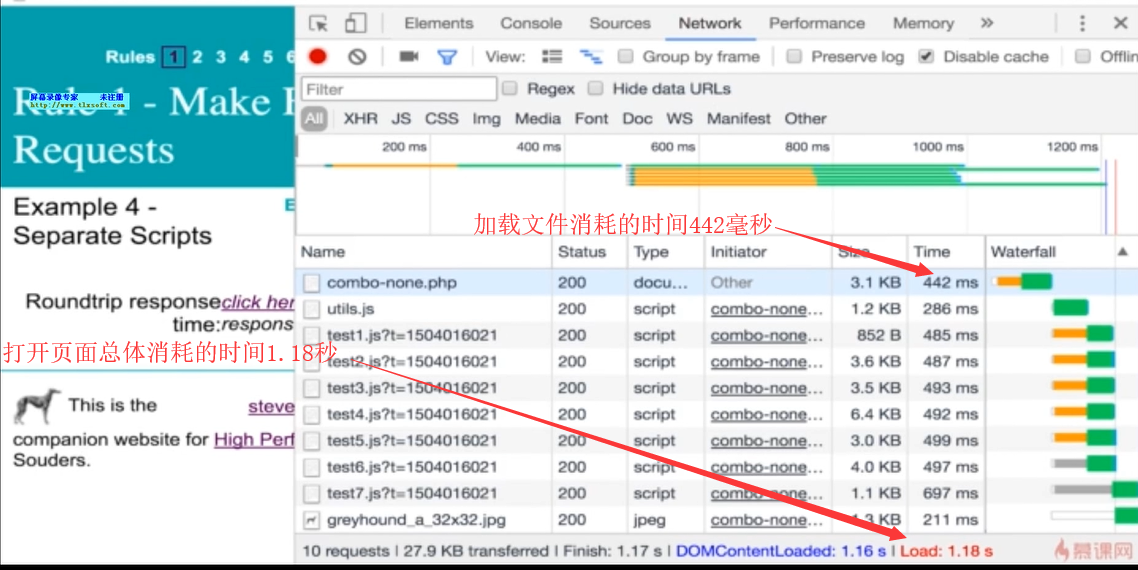
图片地图于CSS精灵的响应时间基本上相同，但是比使用各自独立图片的方式要快50%以上

**合并脚本和样式表适**

使用外部的js和css文件引用的方式，因为这要比直接写在页面中性能要更好一点

独立的一个js比用多个js文件组成的页面载入要快38%

把多个脚本合并为一个脚本，把多个样式表合并为一个样式表



**图片使用Base64编码减少页面请求数**

采用Base64的编码方式将图片直接嵌入到网页中，而不是从外部载入

<img src="data:image/gif;base64./9j/4AAQSkZJ...." >

**浏览器缓存和数据压缩**

**http缓存机制**

**Nginx配置缓存策略**

**前端代码和资源的压缩**

**高并发下只能通过提升服务器负载解决？**

**缓存只能做数据库缓存吗？**

**启用浏览器缓存**

**缓存分类**

HTTP缓存模型中，如果请求成功会有三种情况

200 from cache：直接从本地缓存中获取响应，最快速、最省流量、因为根本没有向服务器发送请求

304 Not Modified:协商缓存，浏览器在本地没有命中的情况下请求头中发送一定的校验数据到服务端，如果服务端数据没有改变浏览器从本地缓存响应，返回304

快速，发送的数据很少，只返回一些基本的响应头信息，数据量很小，部发送实际响应体。

200 ok以上两种缓存全部失败，服务器返回完整响应。没有用到缓存，相对最慢。

浏览器认为本地缓存可以使用，部会器请求服务端

**相关的Header**

Pragma:HTTP1.0时代的遗留物，该字段被设置为no-cache时，会告知浏览器禁用本地缓存，即每次都向服务器发送请求。

Expires:HTTP1.0时代用来启用本地缓存的字段，expires值对应一个形如Thu，31 Dec 3237 23:55:55 GMT的格林威治时间，告诉浏览器缓存实现的时刻，如果还没到该时该刻，标明缓存有效，无需发送请求。

浏览器与服务器的时间无法保持一致，如果时间差距大，就会影响缓存结果。

Cache-Control:HTTP1.1针对Expires时间部一致的解决方案，运用Cache-Control告知浏览器缓存过期的时间间隔而部是时刻，即使具体时间部一致，也不影响缓存的管理。

Cache-Control

On-store:禁止浏览器缓存响应

On-cache:不允许直接使用本地缓存，先发起请求和服务器协商

Max-age=delta-seconds:告知浏览器响应本地缓存有效的最长期限，以秒为单位

**优先级**

依次从左到右

Pragma > Cache-Control > Expires

**协商缓存**

当浏览器没有命中本地缓存，如本地缓存过期或者响应中声明不允许直接使用本地缓存，那么浏览器肯定会发起服务端请求

服务端会验证数据是否修改，如果没有通知浏览器使用本地缓存

**相关Header**

Last-Modified:通知浏览器资源的最后修改时间

Last-Modified:Mon,28 Sep 2015 08:06:43 GMT

If-Modified-Since:得到资源的最后修改时间后，会将这个信息通过If-Modified-Since提交到服务器做检查，如果没有修改，返回304状态码。

If-Modified-Since:Mon,28 Sep 2015 08:06:43

ETag:HTTP1.1推出，文件的指纹标识符，如果文件内容修改，指纹会改变

ETag:”78437822c-6739”

If-None-Match:本地缓存失效，会携带此值请求服务端，服务端判断该资源是否改变，如果没有改变，直接使用本地缓存，返回304

不变的图像，如logo,图标等

Js,css静态文件

可下载的内容，媒体文件

HTML文件

经常替换的图片

经常修改的js、css文件

**建议使用协商缓存**

Js、css文件加载可以入文件的签名来拒绝缓存

Index.css？签名

Index.签名.js

**不建议缓存的内容**

用户隐私等敏感数据

经常改变api数据接口

**本地缓存配置**

Add\_header指令：添加状态码为2xx和3xx的响应头信息

Add\_header name value [always];

可以设置Pragma/Expires/Cache-Control，可以继承

expires指令：通知浏览器过期时长

Expires time;

为负值时表示Cache-Control:no-cache;

当为正或者0时，就表示Cache-Control:max-age=指定的时间

当为max时，会把Expires设置为"Thu 31 Dec 2037 23:55:55",Cache-Control 设置到10年

Etag指令：指定签名

etag on | off; 默认是on

PHP代码

header('Last-Modified:'.gmdate('D, d M Y H:i:s', time()).' GMT');

//获取时间戳

$since=$\_SERVER['HTTP\_IF\_MODIFIED\_SINCE'];//此函数有问题，原因不明

$lifetime = 3600;//当前文件的保存过期时间

//如果当前时间小于获取的时间加3600秒，则认为是没有过期

if(strtotime($since) + $lifetime > time()){

//通知浏览器当前页面使用本地缓存

header('HTTP/1.1 304 Not Modified');

exit;

}

echo time();

**前端代码和资源的压缩**

**优势**

让资源文件更小，加快文件在网络中的传输，让网页更快展现，降低带宽和流量开销。

**压缩方式**

Js,css,图片,html代码的压缩

gzip压缩

**javaScript代码压缩**

javaScript压缩的原理一般是去掉多余的空格和回车，替换长变量名、简化一些代码写法等。

JavaScript代码压缩工具很多，有在线工具，有应用程序，有编辑器插件。

YUI Compressor:来自Yahoo,只有压缩功能

Closure Compiler:来自Google、功能和UglifyJS类似、压缩的方式不一样

**css代码压缩**

原理根javaScript压缩原理类似，同样是去除空白符，注释并且优化一些CSS语义规则等

常用压缩工具：YUI Compressor、CSS Compressor

CSS Compressor:压缩时可以选择模式

**HTML代码压缩**

不建议使用代码压缩，有时会破坏代码结构，可以使用Gzip压缩，当然也可以使用htmlcompressor工具，不果转换后一定检查代码结构。

**图片压缩**

除了代码的压缩外，有时对图片的压缩也是很有必要的，一般情况下图片在web系统的比重都比较大。

**压缩工具：**tinypng、jpegMini、ImageOptim

**Gzip压缩**

配置Nginx服务

Nginx配置

Gzip on|off; #是否开启gzip

Gzip\_buffers 32 4k|16 8k #缓冲(在内存中缓冲几块？每块多大)

Gzip\_comp\_level[1-9]#推荐6压缩级别(级别越高，压的越小，越浪费CPU计算资源)

Gzip\_disable #正则匹配UA什么样的Uri不进行gzip

Gzip\_min\_length 200 #开始压缩的最小长度

Gzip\_http\_version 1.0|1.1 #开始压缩的http协议版本

Gzip\_proxied #设置请求者代理服务器，该如何缓存内容

Gzip\_types text/plain application/xml #对哪些类型的文件用压缩如txt,xml,html,css

Gzip\_vary on|off #是否传输gzip压缩标志

**其他工具**

自动化构建工具Grunt

**CDN加速**

**什么是CDN**

**使用CDN的优势**

**CDN的工作原理**

**CDN的使用场景**

**CDN的实现**

**什么是CDN**

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

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，时内容传输的更快、更稳定

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系统能够实时的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。

本地Cache加速、提高了企业站点(尤其含有大量图片和静态页面站点)的访问速度。

跨运营的网络加速，保证不同网络的用户都得到良好的访问质量

远程访问用户根据DNS负载均衡技术智能自动选择Cache服务器

自动生成服务器的远程Mirror(镜像)cache服务器、远程用户访问从cache服务器上读取数据，减少远程访问的带宽、分担网络流量，减轻源站点WEB服务器负载等功能。

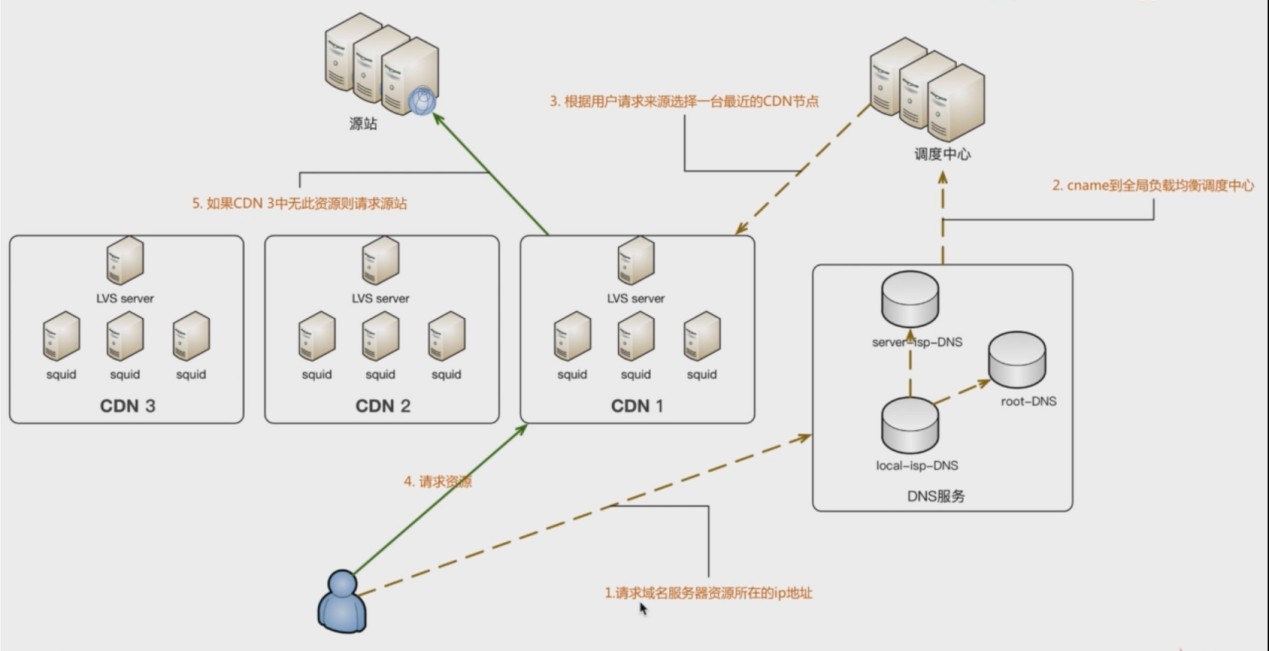
广泛分布的CDN节点加上节点之间的只能冗余机制，可以有效的预防黑客入侵

**传统访问**

用户在浏览器输入域名发起请求-->解析域名获取服务器IP地址-->根据IP地址找到对应的服务器-->服务器响应并返回数据

**使用CDN访问**

**用户发起请求-->只能DNS解析(根据IP判断地理位置，接入网类型，选择路由最短和负载最轻的服务器)-->取得缓存服务器IP-->把内容返回给用户(如果缓存中有)-->向源站发起请求-->将结果返回给用户-->将结果存入缓存服务器**



**适用场景**

站点或者应用中大量静态资源的加速分发、例如css、js图片和HTML

大文件下载

直播网站等

**实现**

BAT等都有提供CDN服务

可用LVS做4层负载均衡

可用Nginx、Varnish、Squid、Apache TrafficServer做7层负载均衡和cache

使用squid反响代理，或者Nginx等的反响代理

# 独立图片服务器的部署

**独立的必要性**

**采用独立域名**

**独立后的问题**

**独立的必要性：**

分担Web服务器的I/O负载-将耗费资源的图片服务分离出来、提高服务器的性能和稳定性

能够专门对图片服务器进行优化-为图片服务器设置有针对性的缓存方案，减少带宽成本，提高访问速度

提高网站的可扩展性-通过增加图片服务器、提高图片吞吐能力

**采用独立域名**

原因：

同一域名下浏览器的并发连接数有限制，突破浏览器连接数的限制

由于cookie的原因、对缓存不利、大部分Web cache都只缓存不带cookie的请求，导致每次的图片请求都不能命中cache

**如何进行图片上传和图片同步**

**NFS共享方式**

**利用FTP同步**

# 动态语言层的并发处理视频

**相关概念**

**什么是动态语言静态化**

**为什么要静态化**

**静态化的实现方式**

**什么是动态语言静态化**

将现有PHP等动态语言逻辑代码生成为静态HTML文件，用户访问动态脚本重定向到静态HTML文件的过程。

对实时性要求不高的页面

**原因：**

动态脚本通常化做逻辑计算和数据查询，访问量越大、服务器压力越大

访问量大时可能会造成CPU负载过高，数据库服务器压力过大静态化可以减低逻辑处理压力，降低数据库服务器查询压力

**使用模板引擎**

可以使用Smarty的缓存机制生成静态HTML缓存文件

$smarty->cache\_dir=$\_ROOT."/cache"; //缓存目录

$smarty->caching = true;//是否开启缓存

$smarty->cache\_lifetime = "3600"//缓存时间

$smarty->display(string template[,string cache\_id[,string compile\_id]]);

$smarty->clear\_all\_cache();//清除所有缓存

$smarty->clear\_cache('file.html');//清除指定的缓存

$smarty->clear\_cache('article.html',$art\_id);//清除同一个模板下的指定缓存号的缓存

**利用ob系列的函数**

Ob\_start():打开输出控制缓冲

Ob\_get\_contents():返回输出缓冲区内容

Ob\_clean():清空输出缓冲区

Ob\_end\_flush():冲刷出(送出)输出缓冲区内容并关闭缓冲

Ob\_start();

输出到页面的HTML代码...

....

Ob\_get\_contents();

Ob\_end\_flush();

Fopen()写入

**利用ob系列的函数**

可以判断文件的inode修改时间、判断是否过期使用filectime函数

**迷你版的模板引擎**

<?php

$cache\_name = md5(\_\_FILE\_\_).'.html'; //生成的文件名

$cache\_lifetime = 3600;//过期时长

//filectime 该函数返回文件上次 inode 被修改的时间

//filectime(\_\_FILE\_\_) <= filectime($cache\_name) 判断文件是否被修改过，如果没有被修改就加载include

//file\_exists($cache\_name) 判断$file\_exists($cache\_name)文件是否存在

//filectime($cache\_name) + $cache\_lifetime > time() 判断文件是否过期

if(filectime(\_\_FILE\_\_) <= filectime($cache\_name) && file\_exists($cache\_name) && filectime($cache\_name) + $cache\_lifetime > time()){

include $cache\_name;

exit;

}

ob\_start(); //开启缓冲区

?>

<b>This is My Script</b>

<?php

$content=ob\_get\_contents(); //获取缓冲区内容

var\_dump($content);

ob\_end\_flush();//关闭缓冲区

$handle=fopen($cache\_name,'w'); //打开文件

fwrite($handle,$content); //把内容写入文件

fclose($handle);

?>