**http协议学习笔记**

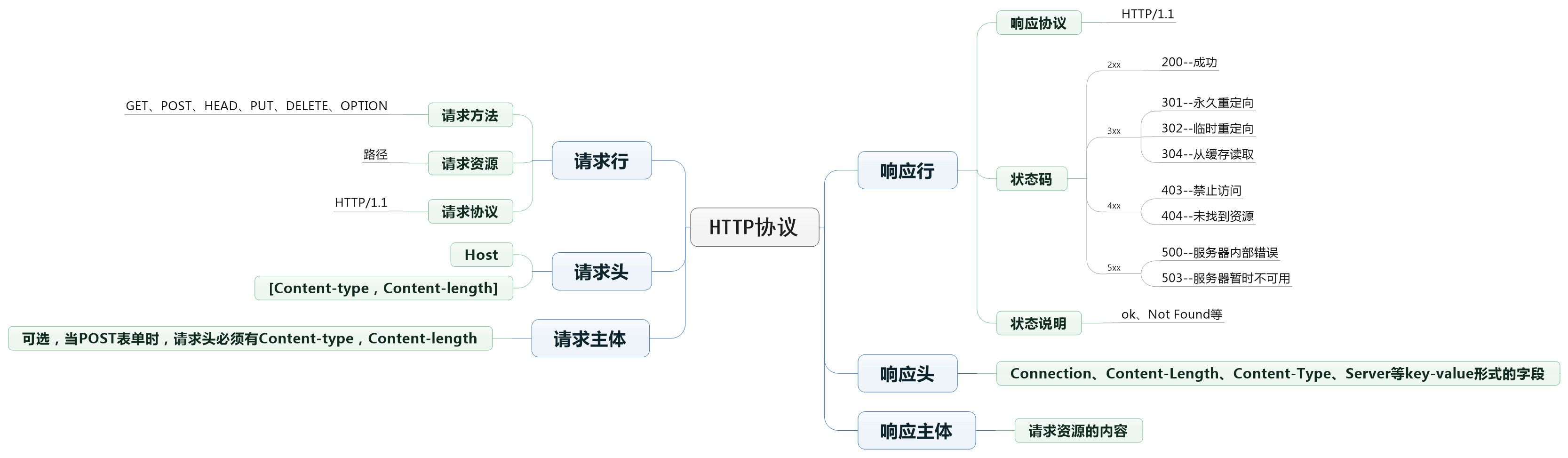
# 1 http协议概念

http协议是超文本传输协议，http协议是封装在TCP/IP协议之上，http没有状态。通俗的说，客户端按一定规则向服务器请求或提交数据，服务器按一定规则响应数据。

在浏览器输入<http://localhost/hello.php>，查看到客户端和服务器的http协议内容，如下图所示：



# 2 http请求信息和响应信息格式



## 2.1 请求信息

请求信息：包括请求行、请求头信息、请求主体信息

注意：请求头和请求主体之间用空行来区分，即使没有请求主体，空行也不能少。

### 2.1.1 请求行

请求行：包括请求方法、请求资源、请求协议版本。

请求方法：GET、POST、HEAD、PUT、DELETE、OPTION，前3种方法比较常用，其他方法服务器未必让你使用。OPTION方法可以验证服务支持哪些方法。

请求资源：请求url的一部分，如index.html、index.php

请求协议版本：HTTP/1.1



### 2.1.2 请求头信息

请求头信息非常丰富。

## 2.2 响应信息

响应信息：包括响应行、响应头信息、响应主体信息。

注意：响应头和响应主体之间用空行来区分，即使没有响应主体，空行也不能少。

### 2.2.1 响应行

包括协议版本、状态码、状态文字



状态码：用来反映服务器响应情况

常用状态码：

200 -- 服务器成功返回网页

301/302 -- 永久/临时重定向

|  |
| --- |
| php的重定向  header('Location: <http://www.baidu.com>'); // 默认是302（临时）  header('Location: <http://www.baidu.com>', true, 301); // 永久重定向，true参数表示用301信息替换原来头信息。 |

304 -- 未修改（说明当前取的是缓存数据，不是从服务器那里取的）

307 -- 重定向中保持原有的请求数据

|  |
| --- |
| 例如user.html提交数据到user.php，user.php又重定向到user2.php  user.php重定向：header('Location: <http://www.baidu.com>', true, 307);  注意：用其他重定向会造成数据丢失。 |

403 -- 禁止访问

404 -- 访问的资源不存在

500 -- 服务器内部错误

503 -- 服务器暂时不可用



### 2.2.2 响应头信息

响应头信息很丰富。格式为一行一个的key: value形式。

## 2.3 用Telnet模拟浏览器的发送请求

(1) GET请求

打开终端，输入命令telnet 127.0.0.1 80，回车，进入清空的终端后按下键盘ctrl+]组合键，然后再按下回车键，打开了回显功能，输入请求信息：

|  |
| --- |
| GET /hello.php HTTP/1.1  Host: 127.0.0.1 |

连续按下两次回车（请求头和请求主体需要一个空行）开始发出请求，GET请求结果如下图所示:

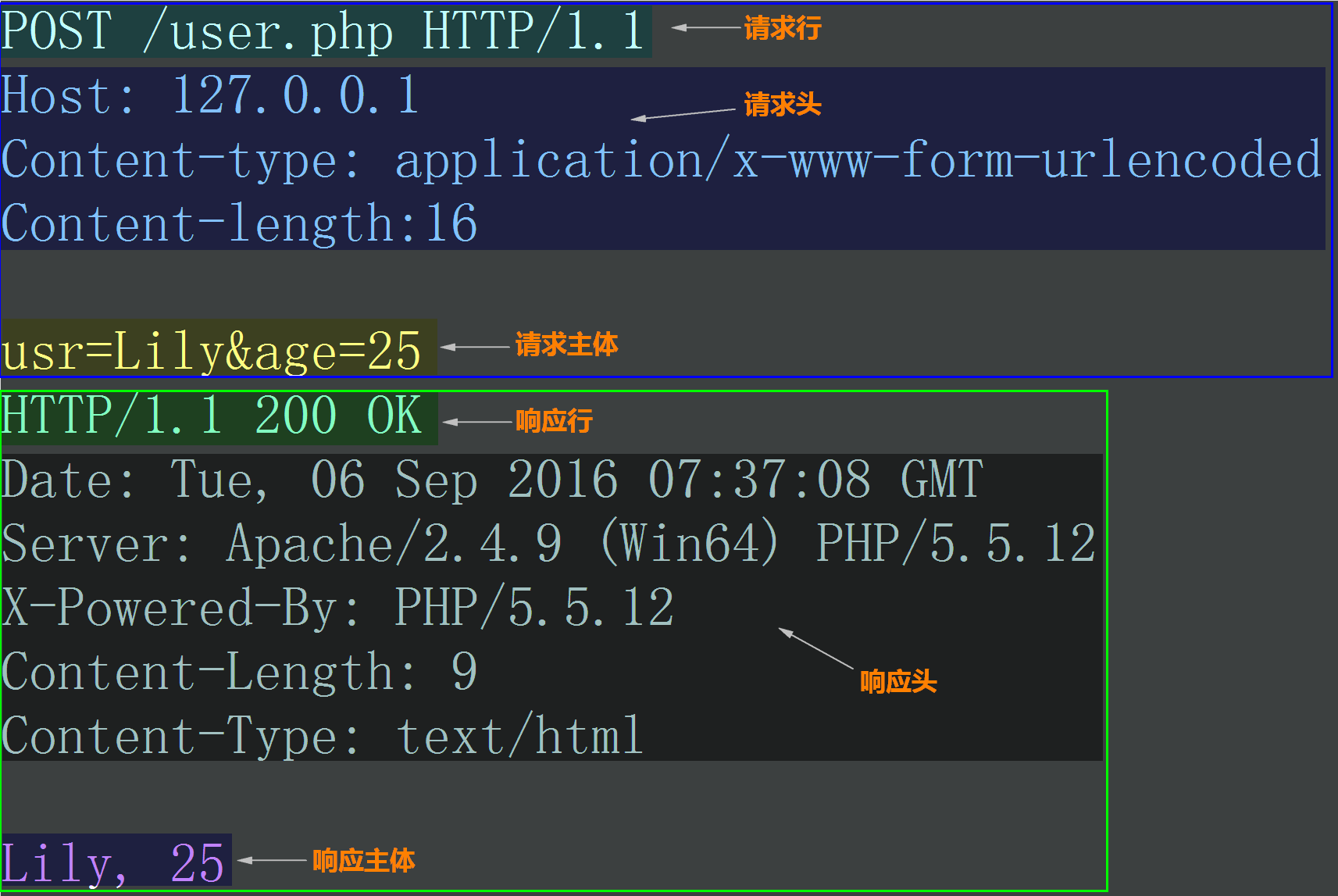


(2) POST请求

在终端输入命令telnet 127.0.0.1 80，输入请求

|  |
| --- |
| POST /user.php HTTP/1.1  Host: 127.0.0.1  Content-type: application/x-www-form-urlencoded  Content-length: 16  usr=Lily&age=25 |

POST请求结果如下图所示:



# 3 cookie

php设置cookie：setcookie('name','zhangsan');

php获取cookie：$\_COOKIE['name'];

通过cookie和请求头，可以模拟登录发帖。

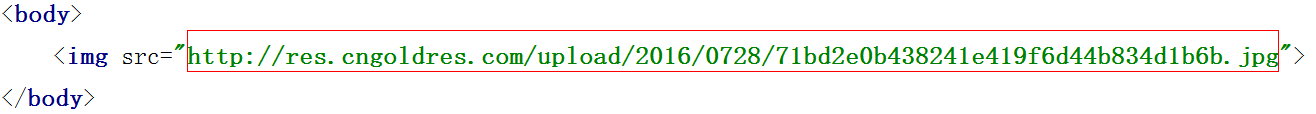
# 4 Referer

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。

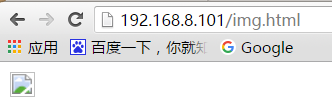
Referer的主要两个应用：图片防盗链和网站入口统计。

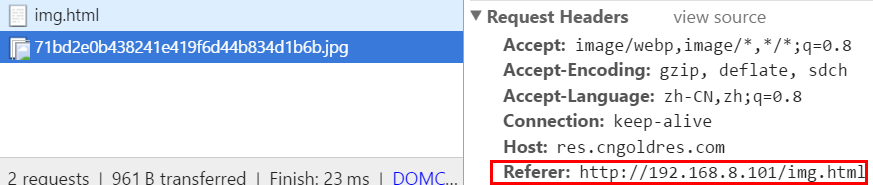
一般大网站后台通过请求头的Referer值统计网站入口或来源。

## 4.1 图片防盗链



用浏览器打开该html文件，



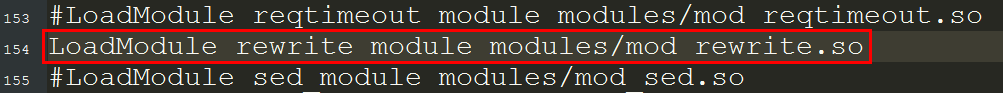


查看请求头的Referer信息，图片服务器发现访问的Referer来源不是指定的主机或域名，禁止访问，达到外站防盗链的效果，避免服务器附加的压力。

## 4.2 apache防盗链设置

(1) 打开apache重写模块mod\_rewrite

打开httpd.conf文件，删除mod\_rewrite.so前面的#号



(2) 在防盗链的网站或目录写.htaccess文件，指定防盗链规则

规则如下：

|  |
| --- |
| RewriteEngine On  RewriteCond %{REQUEST\_FILENAME} .\*\.(jpg|jpeg|gif|png|bmp)[NC]  RewriteCond %{HTTP\_REFERER} !localhost[NC]  RewriteRule .\* fb.png |

(3) 测试是否防盗链生效

分别访问

<http://localhost/img.html>

<http://192.168.8.101/img.html>

如果<http://192.168.8.101/img.html>被禁止访问说明防盗链成功。

## 4.3 突破防盗链图片下载图片

php可以伪造一个Referer就可以下载防盗链的图片了，实现过程如下：

|  |
| --- |
| **<?php require** (**'./http\_class.php'**);  *// get请求图片* $http = **new** Http(**'http://localhost/bird.jpg'**); $http->setHeader(**'Referer: http://localhost'**); *// 伪造一个referer* $res = $http->get();  *// 获取文件格式 preg\_match*(**"/Content-Type.\*\r\n/"**,$res,$out); $format = *trim*(*strrchr*($out[0], **'/'**),**'/'**); $format = *trim*($format);  *// 保存文件 file\_put\_contents*(**'./a.'**.$format, *substr*(*strstr*($res,**"\r\n\r\n"**),4)); **echo 'ok'**; |

请求类文件：http\_class.php

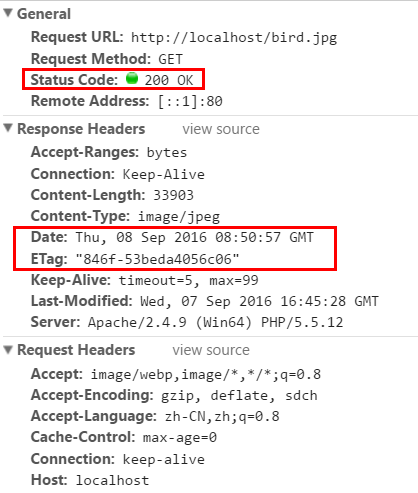
|  |
| --- |
| **<?php** *// http协议接口* **interface** HttpProto{  **function** connect($url); *// socket连接* **function** get(); *// get请求* **function** post($content); *// post请求* **function** close(); *// 关闭连接* }  **class** Http **implements** HttpProto{  **protected $reqLine** = **array**(); *// 存放请求行* **protected $reqHeader** = **array**(); *// 存放请求头* **protected $reqBody** = **array**(); *// 存放请求主体* **protected $url** = **array**(); *// 请求的url信息* **protected $fp** = **null**; *// 连接资源* **protected $errno** = 0;  **protected $errstr** = **''**;  **protected $timeout** = 10;   **public function** \_\_construct($url){ *// 构造一个函数* $this->**url** = *parse\_url*($url); *// 解析url* $this->connect($this->**url**); *// 连接* }   *// 设置请求行* **protected function** setLine($method){  $query = **''**;  **if** (**isset**($this->**url**[**'query'**])){ *// 判断是否有查询参数，如果有则和?号拼接* $query = **'?'**.$this->**url**[**'query'**];  }  **if** (!**isset**($this->**url**[**'path'**])){ *// 判断是否设置path，默认为根目录* $this->**url**[**'path'**] = **'/'**;  }  $this->**reqLine**[] = $method.**' '**.$this->**url**[**'path'**].$query.**' '**.**'HTTP/1.1'**;  }   *// 设置请求头* **public function** setHeader($headerLine){  $this->**reqHeader**[] = $headerLine;  }   *// 设置请求主体* **protected function** setBody($body){  $this->**reqBody**[] = $body;  }   *// socket连接* **public function** connect($url){  **if** (!**isset**($this->**url**[**'port'**])){ *// 判断是否指定端口，没指定默认为80* $this->**url**[**'port'**] = 80;  }  $this->**fp** = *fsockopen*($this->**url**[**'host'**], $this->**url**[**'port'**], $this->**errno**,$this->**errstr**,$this->**timeout**);  **if** (!$this->**fp**) {  **echo "**$this->**errstr (**$this->**errno)<br />\n"**;  **exit**(-1); *// 打开失败则退出* }  }   *// 发送和接收数据* **protected function** sr(){  $merge=*array\_merge*($this->**reqLine**,$this->**reqHeader**,**array**(**''**),$this->**reqBody**,**array**(**''**)); *// 合并数组* $req = *implode*(**"\r\n"**,$merge); *// 把数组个元素拼接成字符串 // echo $req;exit(0);   fwrite*($this->**fp**, $req); *// 发送数据给服务器* $response = **''**;  **while**(!*feof*($this->**fp**)){ *// 读取响应信息* $response .= *fread*($this->**fp**, 1000);  }   $this->close(); *// 关闭连接* **return** $response;  }   *// get请求* **public function** get(){  $this->setLine(**'GET'**);  $this->setHeader(**'Host: '**.$this->**url**[**'host'**]);   **return** $this->sr(); *// 发送和接收响应数据* }   *// post请求* **public function** post($content){  $this->setLine(**'POST'**);  $this->setBody(*http\_build\_query*($content));  $this->setHeader(**'Host: '**.$this->**url**[**'host'**]);  $this->setHeader(**'Content-type: application/x-www-form-urlencoded'**);  $this->setHeader(**'Content-length: '**.*strlen*(*implode*(**''**,$this->**reqBody**)));   **return** $this->sr(); *// 发送和接收响应数据* }   *// 关闭连接* **public function** close(){  *fclose*($this->**fp**);  } } |

# 5 缓存

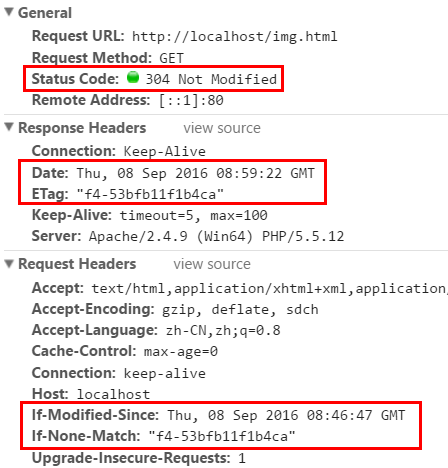
## 5.1 缓存原理

首先观察第一次和第二次请求的请求头和响应头信息。

第一次请求图片



第一次之后的请求



从两张图发现第一次请求的状态码是200，第二次请求状态码是304，说明第一次请求的图片是从服务器读取的，第二次图片是从浏览器缓存读取的。

原理：

第一次请求图片时，服务器响应头带有Date和ETag字段返回给客户端，

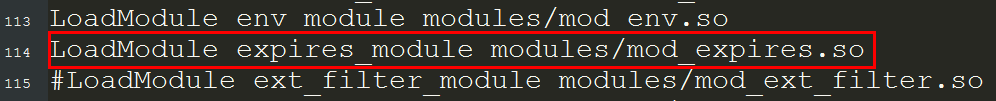
第二次请求时，客户端的请求头会带着这两个字段发给服务器，告诉服务器我有这个图片，请验证下是否有修改，如果有修改则重新请求，如果没没修改客户端就直接读取本地缓存的图片，节省了图片传输时间，减轻服务器压力

注：Date字段表示从该时刻以后，如果文件有修改则重新请求；

ETag字段表示图片的签名(即指纹，图片修改则也会跟着改变)。

## 5.2 apache设置缓存

(1)打开httpd.conf文件，开启mod\_expires模块，去掉前面#号



(2) 在网站或目录写.htaccess文件，填写缓存参数

语法：ExpiresByType type/encoding "<base> [plus] {<num> <type>}\*"

type/encoding：缓存文件类型，通过mime.types查看

base：基于哪个时间点来计算缓存有效期，Access/now/Modification：基于请求/响应/的那一瞬间，被请求文件的最后修改日期。

num：缓存时间大小

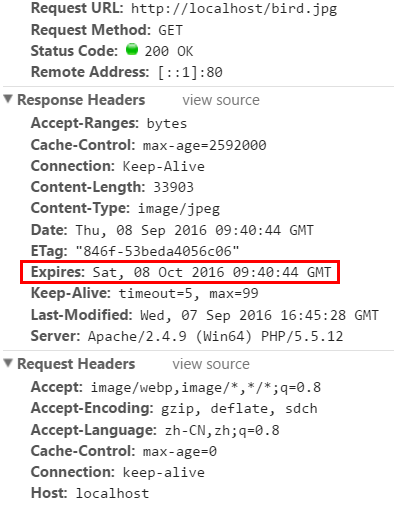
type：缓存时间单位

下面是设置图片缓存30天

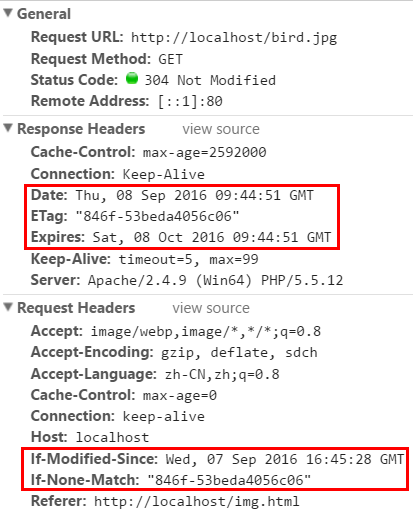
|  |
| --- |
| ExpiresActive On  ExpiresByType image/jpeg "access plus 30 days"  ExpiresByType image/gif "access plus 30 days"  ExpiresByType image/png "access plus 30 days"  ExpiresByType image/bmp "access plus 30 days" |

设置后重启apache。

第一次访问：



第一次之后的访问：



从两张图看出设置缓存成功。

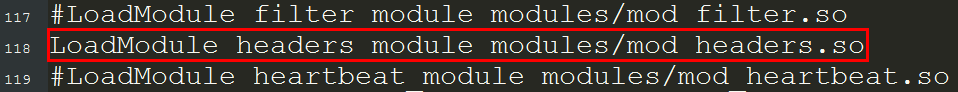
在集群环境里，缓存服务器将缓存该图片30天，在有效期内访问都是从缓存服务器内读取，从而减轻主服务器负担。

## 5.3 apache禁止缓存设置

有些重要信息(校验码、个人信息)不允许缓存服务器缓存，必须到主服务器请求。

apache设置禁止缓存

(1) 打开httpd.conf文件，开启mod\_headers模块，去掉前面#号



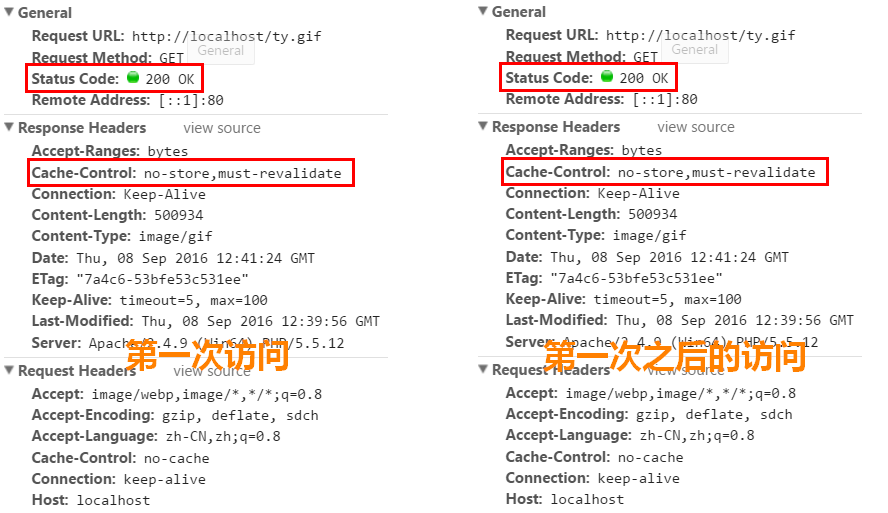
(2) 在网站或目录写.htaccess文件，禁止缓存设置

|  |
| --- |
| <FilesMatch "\.(gif)$">  header set Cache-Control "no-store,must-revalidate"  </FilesMatch> |

must-revalidate:作用与no-cache相同，但更严格，强制意味更明显。但这只是理论上的描述，根据我在ff6上的测试，它几乎不起作用：只要请求的频率加快到一定程度，服务器就接收不到请求。

no-store:缓存将不存储response,包括header和body。测试结果表明，除每次请求都必发送到服务器外，响应代码均是200，且request并没有发送"If-Modified-Since"和"If-None-Match"头，这意味着缓存的确没有存储response。

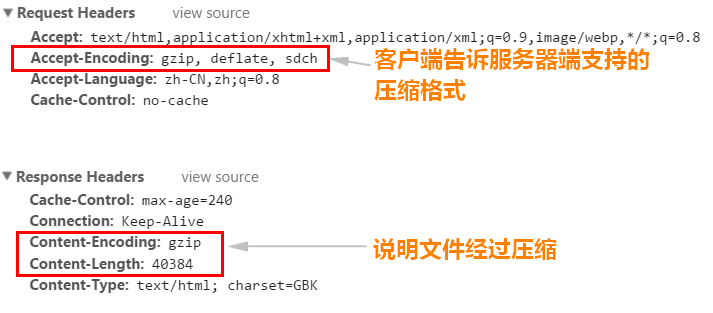
第一次访问和之后的访问对比如下图，没变化，每次都到主服务器读取数据。



# 6 压缩

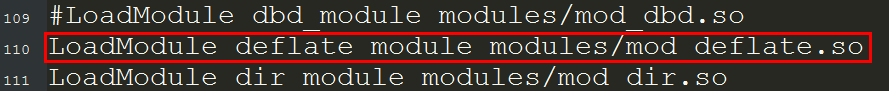
为了提高网页在网络上的传输速度，服务器对主体信息进行压缩，当然压缩是会消耗CPU资源的，一般压缩的文件是文本格式文件，压缩率高，而图片、音频和视频一般不做压缩，因为压缩效果不好又占CPU资源。

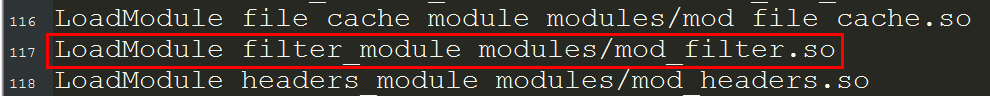
下图说明浏览器接收到压缩文件后经过解压和渲染的。



在apache服务器启用压缩

(1) 打开httpd.conf文件，开启mod\_deflate和mod\_filter模块，去掉前面#号。





(2) 在httpd.conf文件中添加下面内容：

|  |
| --- |
| <ifmodule mod\_deflate.c>  DeflateCompressionLevel 6 #压缩率, 6是建议值.  AddOutputFilterByType DEFLATE text/plain  AddOutputFilterByType DEFLATE text/html  AddOutputFilterByType DEFLATE text/xml  AddOutputFilterByType DEFLATE text/css  AddOutputFilterByType DEFLATE text/javascript  AddOutputFilterByType DEFLATE application/xhtml+xml  AddOutputFilterByType DEFLATE application/xml  AddOutputFilterByType DEFLATE application/rss+xml  AddOutputFilterByType DEFLATE application/atom\_xml  AddOutputFilterByType DEFLATE application/x-javascript  AddOutputFilterByType DEFLATE application/x-httpd-php  AddOutputFilterByType DEFLATE image/svg+xml  </ifmodule> |

(3)对比压缩前和压缩后的效果



从上图看，纯文本的压缩率是非常高的。

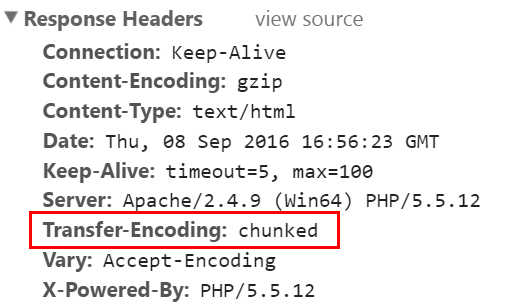
# 7 comet反向ajax

HTTP协议+持久链接+分块传输 -----> 反向ajax

反向ajax又叫comet，server push服务器推送技术

应用范例：网页聊天服务器、新浪微博在线聊天、google mail等

原理：通常HTTP协议的特点是连接--断开，客户端收到指定length长度的内容时就和服务器断开了，在HTTP1.1协议中允许服务器不返回content-length，例如发送的内容长度不知道时，这时需要一个特殊的content-type：chunked。响应头如下图所示：



分场传输原理：

|  |
| --- |
| 20H\r\n (服务器传送20H长度的内容传输给客户端)  ......浏览器接收20H长度内容  10H\r\n (服务器传送100H长度的内容传输给客户端)  ......浏览器接收10H长度内容  0\r\n (服务器发送完了，可以断开连接) |

简易即时通信

(1)创建一个msg.php，内容如下：

|  |
| --- |
| **<?php** *set\_time\_limit*(0); *// 取消超时 ob\_start*(); *// 打开输出控制缓冲* $pad = *str\_repeat*(**' '**,5000); **echo** $pad,**'<br>'**;  *ob\_flush*(); *// 冲刷出（送出）输出缓冲区中的内容 flush*(); *// flush不会对服务器或客户端浏览器的缓存模式产生影响。因此，必须同时使用 ob\_flush() 函数来刷新输出缓冲。  // 连接数据库* $mysqli = **new** mysqli(**'localhost'**,**'root'**,**'123456'**,**'test'**); **if** (*mysqli\_connect\_errno*()){  **die**(**'Unable to connect!'**). *mysqli\_connect\_error*();*//注意mysqli\_connect\_error()新特性* **return**; }  **while**(1){  *// 判断数据库是否有数据* **if** ($result = $mysqli->query(**'SELECT** *\** **FROM msg WHERE tick=1'**)){  **while**($row = $result->fetch\_assoc()){  **echo** $pad,**'<br>'**;  **echo** $row[**'content'**],**'<br>'**;  $mysqli->query(**'UPDATE msg SET tick=0 WHERE id='**.$row[**'id'**]);  }  $result->free();  }   *ob\_flush*();  *flush*();  *sleep*(1); } |

(2)在test库下创建一个msg表，内容如下：

|  |
| --- |
| **USE** test;  **CREATE TABLE** msg(  **id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY NOT NULL** ,  **content VARCHAR**(200) **NOT NULL DEFAULT ''**,  **tick TINYINT NOT NULL DEFAULT** 1 ); |

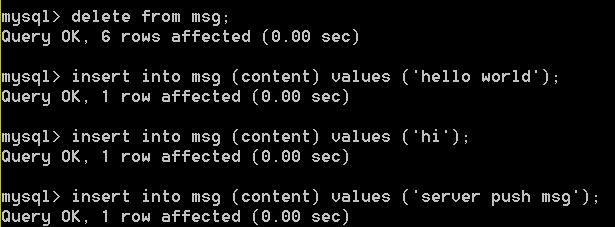
(3)使用浏览器访问msg.php，在终端输入mysql命令

**INSERT INTO** msg (**content**) **VALUES** (**'hello world'**);

每添加一个信息就会在浏览器显示该信息。

实验结果如下：

mysql命令：



浏览器输出结果:

