[【协议分析】HTTP响应头中的2种编码方式介绍](http://blog.csdn.net/jiayanhui2877/article/details/39501425)

2014-09-23 17:45 7159人阅读 [评论](http://blog.csdn.net/jiayanhui2877/article/details/39501425#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/jiayanhui2877/article/details/39501425#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

Protocol Identifying（52） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

HTTP 1.1中有两个实体头(Entity-Header)直接与编码相关,分别为Content-Encoding和Transfer-Encoding。  
    先说Content-Encoding, 该头表示实体已经采用了的编码方式.Content-Encoding是请求URL对应实体(Entity)本身的一部分.比如请求URL为 http://host/image.png.gz时,可能会得到的Content-Encoding为gzip.Content-Encoding的值是不区分大小写的,目前HTTP1.1标准中已包括的有gzip/compress/deflate/identity等.  
   与Content-Encoding头对应,HTTP请求中包含了一个Accept-Encoding头,该头用来说明用户代理(User- Agent,一般也就是浏览器)能接受哪些类型的编码. 如果HTTP请求中不存在该头,服务器可以认为用户代理能接受任何编码类型.  
    接下来重点描述Transfer-Encoding, 该头表示为了达到安全传输或者数据压缩等目的而对实体进行的编码. Transfer-Encoding与Content-Encoding的不同之处在于:  
        1, Transfer-Encoding只是在传输过程中才有的,并非请求URL对应实体的本身特性.   
        2, Transfer-Encoding是一个"跳到跳"头,而Content-Encoding是"端到端"头.   
    该头的用途举例如,请求URL为http://host/abc.txt,服务器发送数据时认为该文件可用gzip方式压缩以节省带宽,接收端看到Transfer-Encoding为gzip首先进行解码然后才能得到请求实体.  
    此外多个编码可能同时对同一实体使用,所以Transfer-Encoding头中编码顺序相当重要,它代表了解码的顺序过程.同样,Transfer- Encoding的值也是不区分大小写的,目前HTTP1.1标准中已包括的有gzip/compress/deflate/identity /chunked等。

报文举例：

Server: Apache-Coyote/1.1  
  
Cache-Control: no-store  
  
Pragma: no-cache  
  
Expires: Thu, 01 Jan 1970 00:00:00 GMT  
  
Content-Type: text/html;charset=GBK  
  
Transfer-Encoding: chunked  
  
Content-Encoding: gzip  
  
Vary: Accept-Encoding  
  
Date: Mon, 01 Jul 2013 02:37:55 GMT  
  
a  
.........  
200  
.=ks.....U..v..f......(..l....lCl..5.#5.t..{$d../X[......c.c@<,.$..^..n..7...q...v%...G....F~.T..P?.=............?.-......J.tU-..z.I.m[......h[...3K..1..U.k\3.K...<..........Oo....o.^......v).#.....(c...  
b..(.......3.I....R'......\*...%o...9...(c....5.....V...4......NW.. .m./...]..}..L..Z.X=\*.>.$=....{G7y....[f.(..M.........e..........Nh`.UU.n.....|ZE....,=.>l.JZ...v..y$5....ho.c....NB...  
..\m.p..[J...A .I....6..RsL.q......6>.h.]Y....J.1.F...e......&Z....w...p...P..^.z+..H..SmS..i...q.m.TS.....(..K....U.0>.k  
200  
..d)M19..}-.{....I.~mui...N....+k...j#..qdq.....x.7MaI3..K..Z....`...j...)4...^...=......B..~(.  
..]...S........>=]9`...C:....|F+K........^.hiUGD.X.T.SY..bA...v..........O..S....f.P...IY;.oI  
........FD...3.Q....e..........dL...T..M.<`Z...Kf.."pR.....Y6..+.f..e..Lw&.m..t...Vt..1..].'..3.Z...'.RI5..j..;.:...J..:.~...>i.V\.v..wum....aM..V...&c+....<  
Sf.F|.........I...Q.Q.3.....U..F...O.....!.R.E.....X...k.....z.tf.Xz....$.>)R.2..6... f.........KP7P...92.c..e......&.[.&yS.P.S.  
....4.  
dn....p.^.N.@..{T7.Mf  
..jUT.  
200

一、Transfer-Encoding含义介绍

有时候，Web服务器生成HTTP Response是无法在Header就确定消息大小的，这时一般来说服务器将不会提供Content-Length的头信息，而采用Chunked编码动态的提供body内容的长度。

进行Chunked编码传输的HTTP Response会在消息头部设置：

Transfer-Encoding: chunked

表示Content Body将用Chunked编码传输内容。

Chunked编码使用若干个Chunk串连而成，由一个标明**长度为0**的chunk标示结束。每个Chunk分为头部和正文两部分，头部内容指定下一段正文的字符总数（**十六进制的数字**）和数量单位（一般不写），正文部分就是指定长度的实际内容，两部分之间用**回车换行(CRLF)**隔开。在最后一个长度为0的Chunk中的内容是称为footer的内容，是一些附加的Header信息（通常可以直接忽略）。具体的Chunk编码格式如下：

　　Chunked-Body = \*chunk  
　　　　　　　　　"0" CRLF  
　　　　　　　　　footer  
　　　　　　　　　CRLF   
　　chunk = chunk-size [ chunk-ext ] CRLF  
　　　　　　 chunk-data CRLF

　　hex-no-zero = <HEX excluding "0">

　　chunk-size = hex-no-zero \*HEX  
　　chunk-ext = \*( ";" chunk-ext-name [ "=" chunk-ext-value ] )  
　　chunk-ext-name = token  
　　chunk-ext-val = token | quoted-string  
　　chunk-data = chunk-size(OCTET)

　　footer = \*entity-header

RFC文档中的Chunked解码过程如下：  
　　length := 0  
　　read chunk-size, chunk-ext (if any) and CRLF  
　　while (chunk-size > 0) {  
　　read chunk-data and CRLF  
　　append chunk-data to entity-body  
　　length := length + chunk-size  
　　read chunk-size and CRLF  
　　}  
　　read entity-header  
　　while (entity-header not empty) {  
　　append entity-header to existing header fields  
　　read entity-header  
　　}  
　　Content-Length := length  
　　Remove "chunked" from Transfer-Encoding

最后提供一段[**PHP**](http://lib.csdn.net/base/php)版本的chunked解码代码：

$chunk\_size = (integer)hexdec(fgets( $socket\_fd, 4096 ) );  
while(!feof($socket\_fd) && $chunk\_size > 0) {  
    $bodyContent .= fread( $socket\_fd, $chunk\_size );  
    fread( $socket\_fd, 2 ); // skip /r/n  
    $chunk\_size = (integer)hexdec(fgets( $socket\_fd, 4096 ) );  
}

二、Content-Encoding含义介绍

Content-Encoding是HTTP协议的响应报文头，一般形式如：

Content-Encoding：gzip,deflate,compress

Content-Encoding的说明中指出deflate指的是在RFC1950说明的zlib格式。也就是说当Content-Encoding为deflate时，内容应该为zlib格式。

compress具说chrome支持，但还没见到哪个web服务器支持

gzip,deflate,zlib的关系：  
  
deflate(RFC1951):一种压缩[**算法**](http://lib.csdn.net/base/datastructure)，使用LZ77和哈弗曼进行编码；    
zlib(RFC1950):一种格式，是对deflate进行了简单的封装；    
gzip(RFC1952):一种格式，也是对deflate进行的封装.

可以看出deflate是最核心的算法，而zlib和gzip格式的区别仅仅是头部和尾部不一样，而实际的内容都是deflate编码的，即：  
gzip = gzip头(10字节) + deflate编码的实际内容 + gzip尾(8字节)

[GZIP的实现可参考GzipOutputStream.[**Java**](http://lib.csdn.net/base/javase)]  
zlib = zlib头 + deflate编码的实际内容 + zlib尾

访问[www.163.com](http://www.163.com/). 响应报文含有gzip头，而[www.baidu.com](http://www.baidu.com/)的响应报文没有gzip头。

看到gzip大家都很好的支持，有无gzip头都没有问题。

(以下内容本人未做验证)

对deflate即zlib格式：

那么在IE上面是打不开页面的，包括IE6，IE7，IE8，提示为一片空白或者出错。但是在其他的浏览器如Firefox，Chrome，Opera等上面都能正常打开。要让IE能够正常打开页面，内容必须是deflate原始格式的数据，即去掉zlib头和zlib尾。不知道IE为什么不修改这个 Bug，按理说在IE6就出现的这种很简单的问题，IE8不应该出现才对。  
为了照顾IE，只好在压缩deflate的时候去掉zlib头和zlib尾，还好其他的浏览器也都能正常处理这种原始的deflate格式。