基础知识

**一.304静态资源离线缓存**

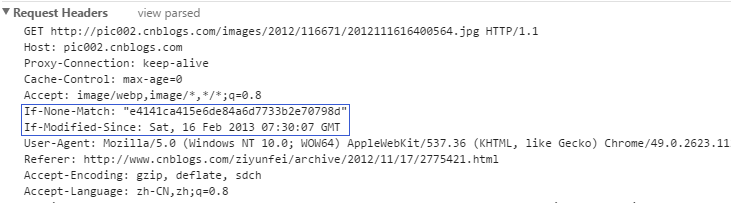
**1)关于 Last-Modified**

　　在浏览器第一次请求某一个URL时，服务器端的返回状态会是200，资源响应头有一个Last-Modified的属性标记此文件在服务期端最后被修改的时间，另外一半也有个Etag，格式类似这样：

Last-Modified:Fri, 15 Feb 2013 03:06:18 GMT

ETag:"be15b26c29bce1:0" #可选，这里为了准确确认资源是否变化

举例：



客户端第二次请求此URL时，根据 HTTP 协议的规定，浏览器会向服务器传送 If-Modified-Since和If-None-Match(可选报头，值Etag的值) 报头，询问该时间之后文件是否有被修改过：

If-Modified-Since:Sat, 16 Feb 2013 07:30:07 GMT

If-None-Match:"be15b26c29bce1:0" #可选，这里为了准确确认资源是否变化

举例：



如果服务器端的资源没有变化，则自动返回 HTTP 304 （Not Changed.）状态码，内容为空，否则重新发起请求，请求下载资源这样就节省了传输数据量。当服务器端代码发生改变或者重启服务器时，则重新发出资源，返回和第一次请求时类似。从而保证不向客户端重复发出资源，也保证当服务器有变化时，客户端能够得到最新的资源。

**2) 什么是”Etag”?**

　　HTTP 协议规格说明定义ETag为“被请求变量的实体值” 。 另一种说法是，ETag是一个可以与Web资源关联的记号（token）。典型的Web资源可以一个Web页，但也可能是JSON或XML文档。服务器单独负责判断记号是什么及其含义，并在HTTP响应头中将其传送到客户端，以下是服务器端返回的格式：  
　　ETag: "50b1c1d4f775c61:df3"  
　　客户端的查询更新格式是这样的：  
　　If-None-Match: W/"50b1c1d4f775c61:df3"  
　　如果ETag没改变，则返回状态304然后不返回，这也和Last-Modified一样。本人测试Etag主要在断点下载时比较有用。

**ETag出现的历史原因是Last-Modified只做到了 “秒级别”的验证，无法辨识毫秒，微妙级别的校验，因此才出现了ETag。**

**ETag的出现，意味着客户端需要需求的升级，因此If-None-Match，If-Match,If-Range随之用来验证文件变化。**

3) **Last-Modified和Etags如何帮助提高性能?**  
　　聪明的开发者会把Last-Modified 和ETags请求的http报头一起使用，这样可利用客户端（例如浏览器）的缓存。因为服务器首先产生 Last-Modified/Etag标记，服务器可在稍后使用它来判断页面是否已经被修改。本质上，客户端通过将该记号传回服务器要求服务器验证其（客户端）缓存。  
　　过程如下:

1. 客户端请求一个页面（A）。
2. 服务器返回页面A，并在给A加上一个Last-Modified/ETag。
3. 客户端展现该页面，并将页面连同Last-Modified/ETag一起缓存。
4. 客户再次请求页面A，并将上次请求时服务器返回的Last-Modified/ETag一起传递给服务器。
5. 服务器检查该Last-Modified或ETag，并判断出该页面自上次客户端请求之后还未被修改，直接返回响应304和一个空的响应体。
6. 此外，如果缓存服务器版本，类型不同，建议使用Last-Modified，Etag可能造成缓存无法验证的问题

**二.缓存有效期期的实现（Cache-Control与Expires）**

　　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。

　　HTTP 1.1为了修正这个问题，引入了Cache-Control:max-age首部，这个首部使用相对时间来控制保质期，让一切变得更加合理。举个例子，我们买了一瓶汽水，如果使用Expires首部来标注保质期，就会这么写：饮料过期时间：2012年12月21日，如果某个2货不知道今天多少号，他还真不知道这饮料过期没，我小时候饮料都这么写。后来，有个挺有名的卖牛奶的，大概就叫蒙牛，他发明了一种标注保质期的方法，他怎么搞了？他这么写：保质期：12个月，行，牛逼了，我牛奶一年前就生产出来的牛奶，今天要发给厂家，发之前，先往包装上印上生产日期（当然是印发货那天），然后告诉你，明年才过期，这多聪明，搞成相对的，毒死你。也许HTTP 1.1借鉴了这个伟大的发明，于是就有了Cache-Control:max-age首部。

**1) Last-Modified&Cache-Control 与 Expire比较**

《1》Last-Modified & Cache-Control  
设置 header("Last-Modified: ".gmdate("D, d M Y H:i:s", time() )." GMT");   
Last-Modified虽然使用了缓存，但是每次打开页面依然需要向服务器发起http请求，浏览器根据用户的$\_SERVER['HTTP\_IF\_MODIFIED\_SINCE']来判断浏览器的内容是否过期，没过期的话返回304状态，浏览器内容从缓存中读取。

此外 Cache-Control也很重要，如果他的值是max-age=0，max-stable,min-refresh等于0或no-store之类的，缓存是不会被缓存的,每次都会请求服务器。如果是max-age，max-stable,min-refresh[大于0]或only-if-cached，immutable，那么很可能出现 from cache现象。  
  
《2》Expires缓存控制  
设置 header("Expires: ".gmdate("D, d M Y H:i:s", time()+$cache\_time )." GMT");   
状态码依然是200，时间依然是旧的时间，Size栏目显示为from cache，表示内容是直接从浏览器读取，浏览器根本就没有向服务器发起http请求。



**2）200 OK (from cache)  实现方案**

Status 200 Ok (from cache)出现的条件是Cache-Control或者Expires满足一定的条件。

**注意：缓存控制是服务器进行报头建议，以下报头是响应头，不是请求头**

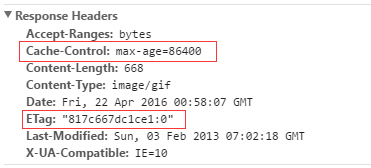
2.1 使用 Cache-Control

1.[(max-age|max-stable|min-refresh) = 缓存创建时间 < 当前系统时间][immutable][only-if-cached]

2.缓存必须带有ETag或者Last-Modified

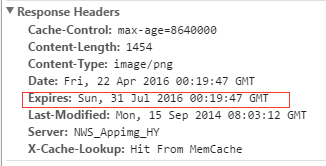
Cache-Control：public,Max-Age=84800

ETag:"f6c01531e9c65fa96f3d40409fd030f1"



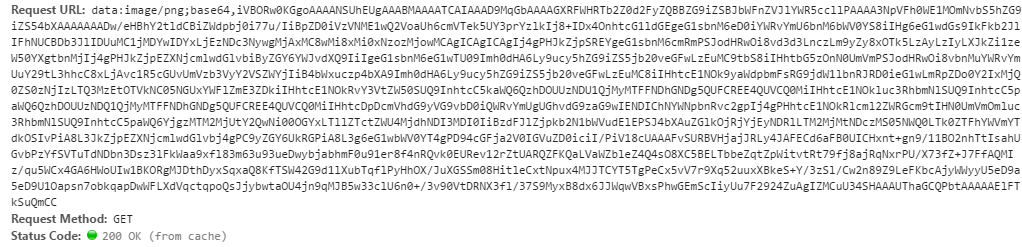
2.2 Expires不能过期

Expires：Sun, 31 Jul 2016 00:19:47 GMT



以上2种方案只要实现一种即可实现资源from cache

**对于浏览器而言，还有一种数据是 DataURL协议的数据，这种数据也会从缓存读取，实现from cache ,但是，如果将所有数据转码成DataURL,会出现性能问题。**



**三.缓存使用技巧**

①.无论是开发阶段还是生产阶段，建议使用Cache-Control + Last-Modified或Tag控制缓存

②.开发阶段， 建议使用Cache-Control:[no-cache,no-store|Max-Age=0]这样可以阻止浏览器使用缓存

③无论是开发阶段还是生产阶段，如果是永远不会被更改的资源，那么建议使用缓存Cache-Control:[Max-Age=3600][only-if-cached][immutable] 从而实现from cache，减少http请求。

④.如果是生产阶段建议使用Cache-Control:Max-Age=3600[no-cache|] ，缓存1小时，每次必须到服务器进行校验

⑤禁止缓存

Cache-Control: no-cache, no-store, must-revalidate

⑥缓存静态资源也可以加上public，实现跨域缓存共享

Cache-Control:public, max-age=31536000

 ⑦ must-revalidate，校验本地缓存是否过期，过期了才去请求服务器更新缓存

 ⑧ 默认情况下，Cache-Control:public缓存都回去和服务器校验的