浏览器缓存知识小结及应用

2016/01/18 · [HTML5](http://web.jobbole.com/category/html5/) · [2 评论](http://web.jobbole.com/84888/#article-comment)· [浏览器](http://web.jobbole.com/tag/%e6%b5%8f%e8%a7%88%e5%99%a8/), [缓存](http://web.jobbole.com/tag/%e7%bc%93%e5%ad%98/)

分享到：**22**

原文出处： [流云诸葛](http://www.cnblogs.com/lyzg/p/5125934.html)

浏览器缓存，也就是客户端缓存，既是网页性能优化里面静态资源相关优化的一大利器，也是无数web开发人员在工作过程不可避免的一大问题，所以在产品开发的时候我们总是想办法避免缓存产生，而在产品发布之时又在想策略管理缓存提升网页的访问速度。了解浏览器的缓存命中原理，是开发web应用的基础，本文着眼于此，学习浏览器缓存的相关知识，总结缓存避免和缓存管理的方法，结合具体的场景说明缓存的相关问题。希望能对有需要的人有所帮助。

**1. 浏览器缓存基本认识**

它分为强缓存和协商缓存：  
1）浏览器在加载资源时，先根据这个资源的一些http header判断它是否命中强缓存，强缓存如果命中，浏览器直接从自己的缓存中读取资源，不会发请求到服务器。比如某个css文件，如果浏览器在加载它所在的网页时，这个css文件的缓存配置命中了强缓存，浏览器就直接从缓存中加载这个css，连请求都不会发送到网页所在服务器；

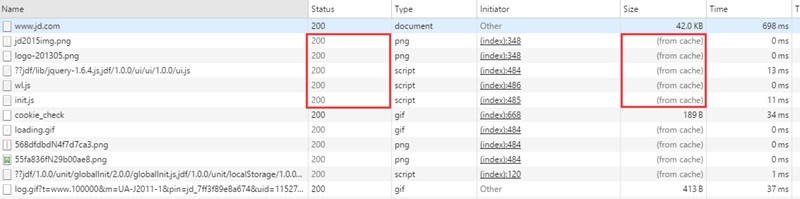
2）当强缓存没有命中的时候，浏览器一定会发送一个请求到服务器，通过服务器端依据资源的另外一些http header验证这个资源是否命中协商缓存，如果协商缓存命中，服务器会将这个请求返回，但是不会返回这个资源的数据，而是告诉客户端可以直接从缓存中加载这个资源，于是浏览器就又会从自己的缓存中去加载这个资源；

3）强缓存与协商缓存的共同点是：如果命中，都是从客户端缓存中加载资源，而不是从服务器加载资源数据；区别是：强缓存不发请求到服务器，协商缓存会发请求到服务器。

4）当协商缓存也没有命中的时候，浏览器直接从服务器加载资源数据。

**2. 强缓存的原理**

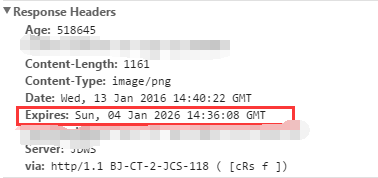
当浏览器对某个资源的请求命中了强缓存时，返回的http状态为200，在chrome的开发者工具的network里面size会显示为from cache，比如京东的首页里就有很多静态资源配置了强缓存，用chrome打开几次，再用f12查看network，可以看到有不少请求就是从缓存中加载的：

[](http://images2015.cnblogs.com/blog/459873/201601/459873-20160113231737569-670985905.png)

强缓存是利用Expires或者Cache-Control这两个http response header实现的，它们都用来表示资源在客户端缓存的有效期。

Expires是http1.0提出的一个表示资源过期时间的header，它描述的是一个绝对时间，由服务器返回，用GMT格式的字符串表示，如：Expires:Thu, 31 Dec 2037 23:55:55 GMT，它的缓存原理是：

1）浏览器第一次跟服务器请求一个资源，服务器在返回这个资源的同时，在respone的header加上Expires的header，如：

[](http://images2015.cnblogs.com/blog/459873/201601/459873-20160113231738710-1642225677.png)

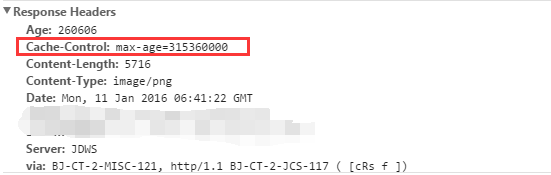
2）浏览器在接收到这个资源后，会把这个资源连同所有response header一起缓存下来（所以缓存命中的请求返回的header并不是来自服务器，而是来自之前缓存的header）；

3）浏览器再请求这个资源时，先从缓存中寻找，找到这个资源后，拿出它的Expires跟当前的请求时间比较，如果请求时间在Expires指定的时间之前，就能命中缓存，否则就不行。

4）如果缓存没有命中，浏览器直接从服务器加载资源时，Expires Header在重新加载的时候会被更新。

Expires是较老的强缓存管理header，由于它是服务器返回的一个绝对时间，在服务器时间与客户端时间相差较大时，缓存管理容易出现问题，比如随意修改下客户端时间，就能影响缓存命中的结果。所以在http1.1的时候，提出了一个新的header，就是Cache-Control，这是一个相对时间，在配置缓存的时候，以秒为单位，用数值表示，如：Cache-Control:max-age=315360000，它的缓存原理是：

1）浏览器第一次跟服务器请求一个资源，服务器在返回这个资源的同时，在respone的header加上Cache-Control的header，如：

[](http://images2015.cnblogs.com/blog/459873/201601/459873-20160114150734725-287071442.png)

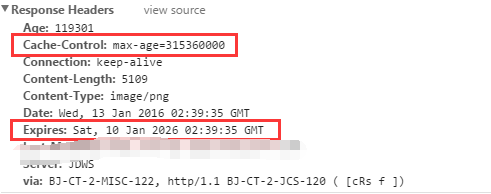
2）浏览器在接收到这个资源后，会把这个资源连同所有response header一起缓存下来；

3）浏览器再请求这个资源时，先从缓存中寻找，找到这个资源后，根据它第一次的请求时间和Cache-Control设定的有效期，计算出一个资源过期时间，再拿这个过期时间跟当前的请求时间比较，如果请求时间在过期时间之前，就能命中缓存，否则就不行。

4）如果缓存没有命中，浏览器直接从服务器加载资源时，Cache-Control Header在重新加载的时候会被更新。

Cache-Control描述的是一个相对时间，在进行缓存命中的时候，都是利用客户端时间进行判断，所以相比较Expires，Cache-Control的缓存管理更有效，安全一些。

这两个header可以只启用一个，也可以同时启用，当response header中，Expires和Cache-Control同时存在时，Cache-Control优先级高于Expires：

[](http://images2015.cnblogs.com/blog/459873/201601/459873-20160114213429663-815418244.png)

**3. 强缓存的管理**

前面介绍的是强缓存的原理，在实际应用中我们会碰到需要强缓存的场景和不需要强缓存的场景，通常有2种方式来设置是否启用强缓存：

1）通过代码的方式，在web服务器返回的响应中添加Expires和Cache-Control Header；

2）通过配置web服务器的方式，让web服务器在响应资源的时候统一添加Expires和Cache-Control Header。

比如在javaweb里面，我们可以使用类似下面的代码设置强缓存：



|  |  |
| --- | --- |
| 1  2  3  4 | java.util.Date date = new java.util.Date();  response.setDateHeader("Expires",date.getTime()+20000); //Expires:过时期限值  response.setHeader("Cache-Control", "public"); //Cache-Control来控制页面的缓存与否,public:浏览器和缓存服务器都可以缓存页面信息；  response.setHeader("Pragma", "Pragma"); //Pragma:设置页面是否缓存，为Pragma则缓存，no-cache则不缓存 |

还可以通过类似下面的java代码设置不启用强缓存：



|  |  |
| --- | --- |
| 1  2  3 | response.setHeader( "Pragma", "no-cache" );  response.setDateHeader("Expires", 0);  response.addHeader( "Cache-Control", "no-cache" );//浏览器和缓存服务器都不应该缓存页面信息 |

tomcat还提供了一个ExpiresFilter专门用来配置强缓存，具体使用的方式可参考tomcat的官方文档：<http://tomcat.apache.org/tomcat-7.0-doc/config/filter.html#Expires_Filter>

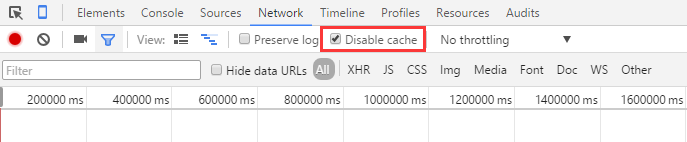
nginx和apache作为专业的web服务器，都有专门的配置文件，可以配置expires和cache-control，这方面的知识，如果你对运维感兴趣的话，可以在百度上搜索“nginx 设置 expires cache-control”或“apache 设置 expires cache-control”都能找到不少相关的文章。

由于在开发的时候不会专门去配置强缓存，而浏览器又默认会缓存图片，css和js等静态资源，所以开发环境下经常会因为强缓存导致资源没有及时更新而看不到最新的效果，解决这个问题的方法有很多，常用的有以下几种：

1）直接ctrl+f5，这个办法能解决页面直接引用的资源更新的问题；

2）使用浏览器的隐私模式开发；

3）如果用的是chrome，可以f12在network那里把缓存给禁掉（这是个非常有效的方法）：

[](http://images2015.cnblogs.com/blog/459873/201601/459873-20160114213430678-777530095.png)

4）在开发阶段，给资源加上一个动态的参数，如css/index.css?v=0.0001，由于每次资源的修改都要更新引用的位置，同时修改参数的值，所以操作起来不是很方便，除非你是在动态页面比如jsp里开发就可以用服务器变量来解决（v=${sysRnd}），或者你能用一些前端的构建工具来处理这个参数修改的问题；

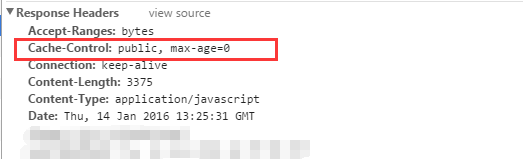
5）如果资源引用的页面，被嵌入到了一个iframe里面，可以在iframe的区域右键单击重新加载该页面，以chrome为例：

[](http://images2015.cnblogs.com/blog/459873/201601/459873-20160114213431538-334819672.png)

6）如果缓存问题出现在ajax请求中，最有效的解决办法就是ajax的请求地址追加随机数；

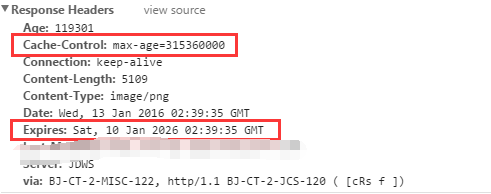
7）还有一种情况就是动态设置iframe的src时，有可能也会因为缓存问题，导致看不到最新的效果，这时候在要设置的src后面添加随机数也能解决问题；

8）如果你用的是grunt和gulp这种前端工具开发，通过它们的插件比如grunt-contrib-connect来启动一个静态服务器，则完全不用担心开发阶段的资源更新问题，因为在这个静态服务器下的所有资源返回的respone header中，cache-control始终被设置为不缓存：

[](http://images2015.cnblogs.com/blog/459873/201601/459873-20160114213432366-1763288997.png)

**4. 强缓存的应用**

强缓存是前端性能优化最有力的工具，没有之一，对于有大量静态资源的网页，一定要利用强缓存，提高响应速度。通常的做法是，为这些静态资源全部配置一个超时时间超长的Expires或Cache-Control，这样用户在访问网页时，只会在第一次加载时从服务器请求静态资源，其它时候只要缓存没有失效并且用户没有强制刷新的条件下都会从自己的缓存中加载，比如前面提到过的京东首页缓存的资源，它的缓存过期时间都设置到了2026年：

[](http://images2015.cnblogs.com/blog/459873/201601/459873-20160114223714288-2031045760.png)

然而这种缓存配置方式会带来一个新的问题，就是发布时资源更新的问题，比如某一张图片，在用户访问第一个版本的时候已经缓存到了用户的电脑上，当网站发布新版本，替换了这个图片时，已经访问过第一个版本的用户由于缓存的设置，导致在默认的情况下不会请求服务器最新的图片资源，除非他清掉或禁用缓存或者强制刷新，否则就看不到最新的图片效果。

这个问题已经有成熟的解决方案，具体内容可阅读知乎这篇文章详细了解：

<http://www.zhihu.com/question/20790576>

文章提到的东西都属于理论上的解决方案，不过现在已经有很多前端工具能够实际地解决这个问题，由于每个工具涉及到的内容细节都有很多，本文没有办法一一深入介绍。有兴趣的可以去了解下grunt gulp webpack fis 还有edp这几个工具，基于这几个工具都能解决这个问题，尤其是fis和edp是百度推出的前端开发平台，有现成的文档可以参考：

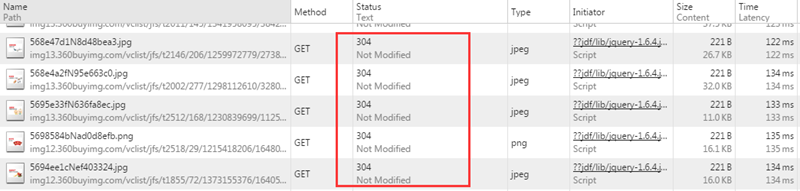
<http://fis.baidu.com/fis3/api/index.html>

<http://ecomfe.github.io/edp/doc/initialization/install/>

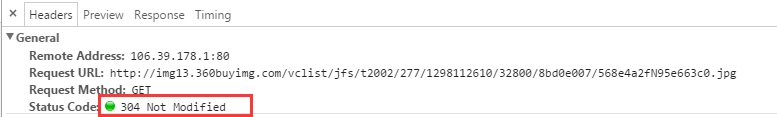
强缓存还有一点需要注意的是，通常都是针对静态资源使用，动态资源需要慎用，除了服务端页面可以看作动态资源外，那些引用静态资源的html也可以看作是动态资源，如果这种html也被缓存，当这些html更新之后，可能就没有机制能够通知浏览器这些html有更新，尤其是前后端分离的应用里，页面都是纯html页面，每个访问地址可能都是直接访问html页面，这些页面通常不加强缓存，以保证浏览器访问这些页面时始终请求服务器最新的资源。

**5. 协商缓存的原理**

当浏览器对某个资源的请求没有命中强缓存，就会发一个请求到服务器，验证协商缓存是否命中，如果协商缓存命中，请求响应返回的http状态为304并且会显示一个Not Modified的字符串，比如你打开京东的首页，按f12打开开发者工具，再按f5刷新页面，查看network，可以看到有不少请求就是命中了协商缓存的：

[](http://images2015.cnblogs.com/blog/459873/201601/459873-20160115111431303-866314106.png)

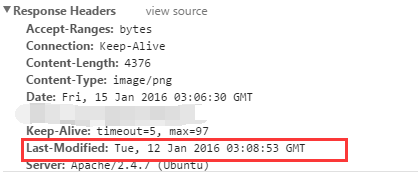
查看单个请求的Response Header，也能看到304的状态码和Not Modified的字符串，只要看到这个就可说明这个资源是命中了协商缓存，然后从客户端缓存中加载的，而不是服务器最新的资源：

[](http://images2015.cnblogs.com/blog/459873/201601/459873-20160115111435100-1664945349.png)

协商缓存是利用的是【Last-Modified，If-Modified-Since】和【ETag、If-None-Match】这两对Header来管理的。

【Last-Modified，If-Modified-Since】的控制缓存的原理是：

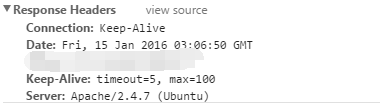
1）浏览器第一次跟服务器请求一个资源，服务器在返回这个资源的同时，在respone的header加上Last-Modified的header，这个header表示这个资源在服务器上的最后修改时间：

[](http://images2015.cnblogs.com/blog/459873/201601/459873-20160115111438210-1551540589.png)

2）浏览器再次跟服务器请求这个资源时，在request的header上加上If-Modified-Since的header，这个header的值就是上一次请求时返回的Last-Modified的值：

[](http://images2015.cnblogs.com/blog/459873/201601/459873-20160115125437460-1062702620.png)

3）服务器再次收到资源请求时，根据浏览器传过来If-Modified-Since和资源在服务器上的最后修改时间判断资源是否有变化，如果没有变化则返回304 Not Modified，但是不会返回资源内容；如果有变化，就正常返回资源内容。当服务器返回304 Not Modified的响应时，response header中不会再添加Last-Modified的header，因为既然资源没有变化，那么Last-Modified也就不会改变，这是服务器返回304时的response header：

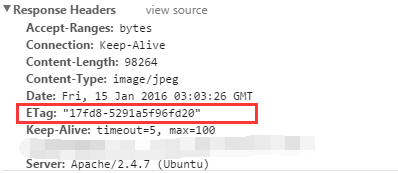
[](http://images2015.cnblogs.com/blog/459873/201601/459873-20160115111450069-558396542.png)

4）浏览器收到304的响应后，就会从缓存中加载资源。

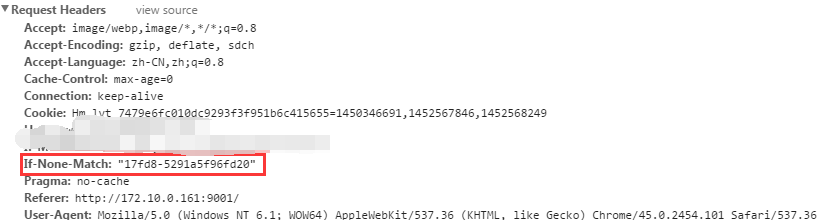
5）如果协商缓存没有命中，浏览器直接从服务器加载资源时，Last-Modified Header在重新加载的时候会被更新，下次请求时，If-Modified-Since会启用上次返回的Last-Modified值。

【Last-Modified，If-Modified-Since】都是根据服务器时间返回的header，一般来说，在没有调整服务器时间和篡改客户端缓存的情况下，这两个header配合起来管理协商缓存是非常可靠的，但是有时候也会服务器上资源其实有变化，但是最后修改时间却没有变化的情况，而这种问题又很不容易被定位出来，而当这种情况出现的时候，就会影响协商缓存的可靠性。所以就有了另外一对header来管理协商缓存，这对header就是【ETag、If-None-Match】。它们的缓存管理的方式是：

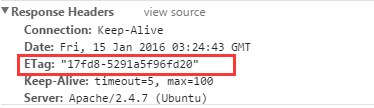
1）浏览器第一次跟服务器请求一个资源，服务器在返回这个资源的同时，在respone的header加上ETag的header，这个header是服务器根据当前请求的资源生成的一个唯一标识，这个唯一标识是一个字符串，只要资源有变化这个串就不同，跟最后修改时间没有关系，所以能很好的补充Last-Modified的问题：

[](http://images2015.cnblogs.com/blog/459873/201601/459873-20160115112627835-440813422.png)

2）浏览器再次跟服务器请求这个资源时，在request的header上加上If-None-Match的header，这个header的值就是上一次请求时返回的ETag的值：

[](http://images2015.cnblogs.com/blog/459873/201601/459873-20160115112635850-559297790.png)

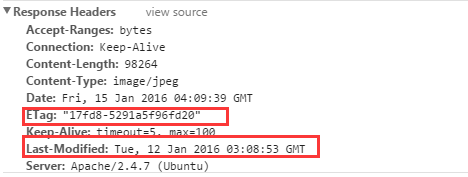
3）服务器再次收到资源请求时，根据浏览器传过来If-None-Match和然后再根据资源生成一个新的ETag，如果这两个值相同就说明资源没有变化，否则就是有变化；如果没有变化则返回304 Not Modified，但是不会返回资源内容；如果有变化，就正常返回资源内容。与Last-Modified不一样的是，当服务器返回304 Not Modified的响应时，由于ETag重新生成过，response header中还会把这个ETag返回，即使这个ETag跟之前的没有变化：

[](http://images2015.cnblogs.com/blog/459873/201601/459873-20160115125438928-1048294855.png)

4）浏览器收到304的响应后，就会从缓存中加载资源。

**6. 协商缓存的管理**

协商缓存跟强缓存不一样，强缓存不发请求到服务器，所以有时候资源更新了浏览器还不知道，但是协商缓存会发请求到服务器，所以资源是否更新，服务器肯定知道。大部分web服务器都默认开启协商缓存，而且是同时启用【Last-Modified，If-Modified-Since】和【ETag、If-None-Match】，比如apache:

[](http://images2015.cnblogs.com/blog/459873/201601/459873-20160115125439975-1724252813.png)

如果没有协商缓存，每个到服务器的请求，就都得返回资源内容，这样服务器的性能会极差。

【Last-Modified，If-Modified-Since】和【ETag、If-None-Match】一般都是同时启用，这是为了处理Last-Modified不可靠的情况。有一种场景需要注意：

分布式系统里多台机器间文件的Last-Modified必须保持一致，以免负载均衡到不同机器导致比对失败；

分布式系统尽量关闭掉ETag(每台机器生成的ETag都会不一样）；

京东页面的资源请求，返回的repsones header就只有Last-Modified，没有ETag：

[](http://images2015.cnblogs.com/blog/459873/201601/459873-20160115125440819-875081491.png)

协商缓存需要配合强缓存使用，你看前面这个截图中，除了Last-Modified这个header，还有强缓存的相关header，因为如果不启用强缓存的话，协商缓存根本没有意义。

**7. 浏览器行为对缓存的影响**

如果资源已经被浏览器缓存下来，在缓存失效之前，再次请求时，默认会先检查是否命中强缓存，如果强缓存命中则直接读取缓存，如果强缓存没有命中则发请求到服务器检查是否命中协商缓存，如果协商缓存命中，则告诉浏览器还是可以从缓存读取，否则才从服务器返回最新的资源。这是默认的处理方式，这个方式可能被浏览器的行为改变：

1）当ctrl+f5强制刷新网页时，直接从服务器加载，跳过强缓存和协商缓存；

2）当f5刷新网页时，跳过强缓存，但是会检查协商缓存；

谢谢阅读：）希望本文的内容能对你有所帮助~