重用已获取的资源能够有效的提升网站与应用的性能。Web 缓存能够减少延迟与网络阻塞，进而减少显示某个资源所用的时间。借助 HTTP 缓存，Web 站点变得更具有响应性。

**各种类型的缓存**[**Edit**](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Caching_FAQ$edit#%E5%90%84%E7%A7%8D%E7%B1%BB%E5%9E%8B%E7%9A%84%E7%BC%93%E5%AD%98)

缓存是一种保存资源副本并在下次请求时直接使用该副本的技术。当 web 缓存发现请求的资源已经被存储，它会拦截请求，返回该资源的拷贝，而不会去源服务器重新下载。这样带来的好处有：缓解服务器端压力，提升性能(获取资源的耗时更短了)。对于网站来说，缓存是达到高性能的重要组成部分。缓存需要合理配置，因为并不是所有资源都是永久不变的：重要的是对一个资源的缓存应截止到其下一次发生改变（即不能缓存过期的资源）。

缓存的种类有很多,其大致可归为两类：私有与共享缓存。共享缓存存储的响应能够被多个用户使用。私有缓存只能用于单独用户。本文将主要介绍浏览器与代理缓存，除此之外还有网关缓存、CDN、反向代理缓存和负载均衡器等部署在服务器上，为站点和 web 应用提供更好的稳定性、性能和扩展性。

(私有)浏览器缓存

私有缓存只能用于单独用户。你可能已经见过浏览器设置中的“缓存”选项。浏览器缓存拥有用户通过 [HTTP](https://developer.mozilla.org/en-US/docs/Web/HTTP) 下载的所有文档。这些缓存为浏览过的文档提供向后/向前导航，保存网页，查看源码等功能，可以避免再次向服务器发起多余的请求。它同样可以提供缓存内容的离线浏览。

(共享)代理缓存

共享缓存可以被多个用户使用。例如，ISP 或你所在的公司可能会架设一个 web 代理来作为本地网络基础的一部分提供给用户。这样热门的资源就会被重复使用，减少网络拥堵与延迟。

**缓存操作的目标[Edit](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Caching_FAQ$edit" \l "%E7%BC%93%E5%AD%98%E6%93%8D%E4%BD%9C%E7%9A%84%E7%9B%AE%E6%A0%87)**

虽然 HTTP 缓存不是必须的，但重用缓存的资源通常是必要的。然而常见的 HTTP 缓存只能存储 [GET](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/GET) 响应，对于其他类型的响应则无能为力。缓存的关键主要包括request method和目标URI（一般只有GET请求才会被缓存）。 普遍的缓存案例:

* 一个检索请求的成功响应: 对于 [GET](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/GET)请求，响应状态码为：[200](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Status/200" \o "状态码 200 OK 表明请求已经成功. 默认情况下状态码为200的响应可以被缓存。)，则表示为成功。一个包含例如HTML文档，图片，或者文件的响应.
* 不变的重定向: 响应状态码：[301](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Status/301" \o "HTTP 301 永久重定向 说明请求的资源已经被移动到了由 Location 头部指定的url上，是固定的不会再改变。搜索引擎会根据该响应修正。).
* 错误响应: 响应状态码：[404](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Status/404" \o "状态码 404 Not Found 代表客户端错误，指的是服务器端无法找到所请求的资源。返回该响应的链接通常称为坏链（broken link）或死链（dead link），它们会导向链接出错处理(link rot)页面。) 的一个页面.
* 不完全的响应: 响应状态码 [206](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Status/206)，只返回局部的信息.
* 除了 [GET](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/GET) 请求外，如果匹配到作为一个已被定义的cache键名的响应.

针对一些特定的请求，也可以通过关键字区分多个存储的不同响应以组成缓存的内容。具体参考[下文](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Caching_FAQ#Varying_responses)关于 [Vary](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Vary) 的信息 .

**缓存控制[Edit](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Caching_FAQ$edit" \l "%E7%BC%93%E5%AD%98%E6%8E%A7%E5%88%B6)**

Cache-control 头

HTTP/1.1定义的 [Cache-Control](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Cache-Control) 头用来区分对缓存机制的支持情况， 请求头和响应头都支持这个属性。通过它提供的不同的值来定义缓存策略。

**禁止进行缓存**

缓存中不得存储任何关于客户端请求和服务端响应的内容。每次由客户端发起的请求都会下载完整的响应内容。

Cache-Control: no-store

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

**强制确认缓存**

如下头部定义，此方式下，每次有请求发出时，缓存会将此请求发到服务器（译者注：该请求应该会带有与本地缓存相关的验证字段），服务器端会验证请求中所描述的缓存是否过期，若未过期（注：实际就是返回304），则缓存才使用本地缓存副本。

Cache-Control: no-cache

**私有缓存和公共缓存**

"public" 指令表示该响应可以被任何中间人（译者注：比如中间代理、CDN等）缓存。若指定了"public"，则一些通常不被中间人缓存的页面（译者注：因为默认是private）（比如 带有HTTP验证信息（帐号密码）的页面 或 某些特定影响状态码的页面），将会被其缓存。

而 "private" 则表示该响应是专用于某单个用户的，中间人不能缓存此响应，该响应只能应用于浏览器私有缓存中。

Cache-Control: private

Cache-Control: public

**缓存过期机制**

过期机制中，最重要的指令是 "max-age=<seconds>"，表示资源能够被缓存（保持新鲜）的最大时间。相对[Expires](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Expires)而言，max-age是距离请求发起的时间的秒数。针对应用中那些不会改变的文件，通常可以手动设置一定的时长以保证缓存有效，例如图片、css、js等静态资源。

详情看下文关于[缓存有效性](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Caching_FAQ#Freshness)的内容。

Cache-Control: max-age=31536000

**缓存验证确认**

当使用了 "must-revalidate" 指令，那就意味着缓存在考虑使用一个陈旧的资源时，必须先验证它的状态，已过期的缓存将不被使用。详情看下文关于[缓存校验](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Caching_FAQ" \l "Cache_validation)的内容。

Cache-Control: must-revalidate

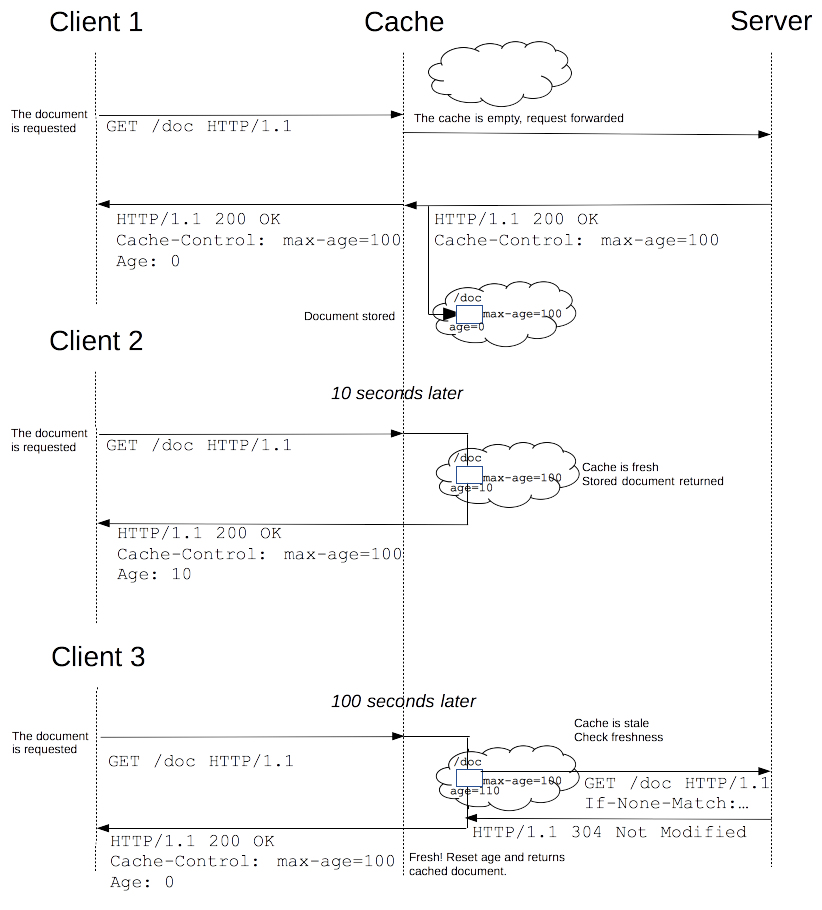
Pragma 头

[Pragma](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Pragma) 是HTTP/1.0标准中定义的一个header属性，请求中包含Pragma的效果跟在头信息中定义Cache-Control: no-cache相同，但是HTTP的响应头不支持这个属性，所以它不能拿来完全替代HTTP/1.1中定义的Cache-control头。通常定义Pragma以向后兼容基于HTTP/1.0的客户端。

**新鲜度**[**Edit**](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Caching_FAQ$edit#%E6%96%B0%E9%B2%9C%E5%BA%A6)

理论上来讲，当一个资源被缓存存储后，该资源应该可以被永久存储在缓存中。由于缓存只有有限的空间用于存储资源副本，所以缓存会定期地将一些副本删除，这个过程叫做缓存驱逐。另一方面，当服务器上面的资源进行了更新，那么缓存中的对应资源也应该被更新，由于HTTP是C/S模式的协议，服务器更新一个资源时，不可能直接通知客户端及其缓存，所以双方必须为该资源约定一个过期时间，在该过期时间之前，该资源（缓存副本）就是新鲜的，当过了过期时间后，该资源（缓存副本）则变为陈旧的*。*驱逐算法用于将陈旧的资源（缓存副本）替换为新鲜的，注意，一个陈旧的资源（缓存副本）是不会直接被清除或忽略的，当客户端发起一个请求时，缓存检索到已有一个对应的陈旧资源（缓存副本），则缓存会先将此请求附加一个[If-None-Match](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/If-None-Match" \o "The If-None-Match HTTP request header makes the request conditional. For GET and HEAD methods, the server will send back the requested resource, with a 200 status, only if doesn't have a ETag matching the given one. For other methods, the request will be processed only if the eventually existing resource's ETag doesn't match any of the values listed.)头，然后发给目标服务器，以此来检查该资源副本是否是依然还是算新鲜的，若服务器返回了 [304](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/304) (Not Modified)（该响应不会有带有实体信息），则表示此资源副本是新鲜的，这样一来，可以节省一些带宽。若服务器通过 If-None-Match 或 If-Modified-Since判断后发现已过期，那么会带有该资源的实体内容返回。

下面是上述缓存处理过程的一个图示：



对于含有特定头信息的请求，会去计算缓存寿命。比如Cache-control: max-age=N的请求头，相应的缓存的寿命就是N。通常情况下，对于不含这个属性的请求则会去查看是否包含[Expires](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Expires)属性，通过比较Expires的值和头里面[Date](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Date)属性的值来判断是否缓存还有效。如果max-age和expires属性都没有，找找头里的[Last-Modified](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Last-Modified)信息。如果有，缓存的寿命就等于头里面Date的值减去Last-Modified的值除以10（注：根据rfc2626其实也就是乘以10%）。

缓存失效时间计算公式如下：

expirationTime = responseTime + freshnessLifetime - currentAge

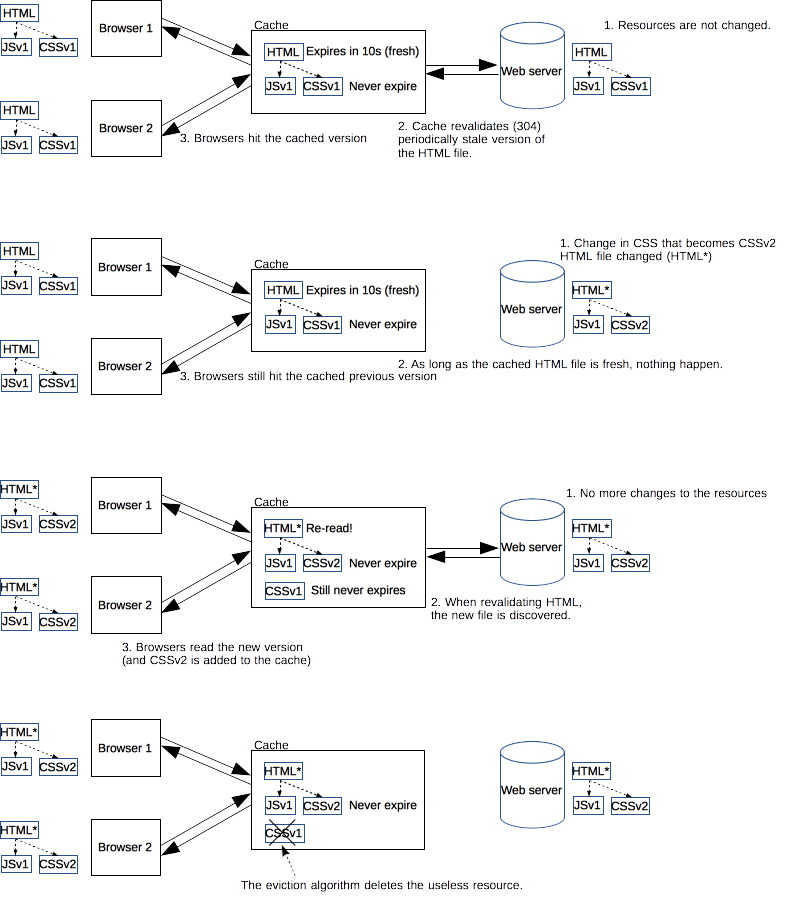
上式中，responseTime 表示浏览器接收到此响应的那个时间点。

加速资源

更多地利用缓存资源，可以提高网站的性能和相应速度。为了优化缓存，过期时间设置得尽量长是一种很好的策略。对于定期或者频繁更新的资源，这么做是比较稳妥的，但是对于那些长期不更新的资源会有点问题。这些固定的资源在一定时间内受益于这种长期保持的缓存策略，但一旦要更新就会很困难。特指网页上引入的一些js/css文件，当它们变动时需要尽快更新线上资源。

web开发者发明了一种 Steve Sounders 称作加速（译者注：revving）的技术[[1]](https://www.stevesouders.com/blog/2008/08/23/revving-filenames-dont-use-querystring/)。不频繁更新的文件会使用特定的命名方式：在URL后面（通常是文件名后面）会加上版本号。加上版本号后的资源就被视作一个完全新的独立的资源，同时拥有一年甚至更长的缓存过期时长。但是这么做也存在一个弊端，所有引用这个资源的地方都需要更新链接。web开发者们通常会采用自动化构建工具在实际工作中完成这些琐碎的工作。当低频更新的资源（js/css）变动了，只用在高频变动的资源文件（html）里做入口的改动。

这种方法还有一个好处：同时更新两个缓存资源不会造成部分缓存先更新而引起新旧文件内容不一致。对于互相有依赖关系的css和js文件，避免这种不一致性是非常重要的。



加在加速文件后面的版本号不一定是一个正式的版本号字符串，如1.1.3这样或者其他固定自增的版本数。它可以是任何防止缓存碰撞的标记例如hash或者时间戳。

**缓存验证**[**Edit**](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Caching_FAQ$edit#%E7%BC%93%E5%AD%98%E9%AA%8C%E8%AF%81)

用户点击刷新按钮时会开始缓存验证。如果缓存的响应头信息里含有"Cache-control: must-revalidate”的定义，在浏览的过程中也会触发缓存验证。另外，在浏览器偏好设置里设置Advanced->Cache为强制验证缓存也能达到相同的效果。

当缓存的文档过期后，需要进行缓存验证或者重新获取资源。只有在服务器返回强校验器或者弱校验器时才会进行验证。

ETags

作为缓存的一种强校验器，[ETag](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/ETag" \o "ETag HTTP响应头是资源的特定版本的标识符。这可以让缓存更高效，并节省带宽，因为如果内容没有改变，Web服务器不需要发送完整的响应。而如果内容发生了变化，使用ETag有助于防止资源的同时更新相互覆盖（\“空中碰撞\”）。) 响应头是一个对用户代理(User Agent, 下面简称UA)不透明（译者注：UA 无需理解，只需要按规定使用即可）的值。对于像浏览器这样的HTTP UA，不知道ETag代表什么，不能预测它的值是多少。如果资源请求的响应头里含有ETag, 客户端可以在后续的请求的头中带上 [If-None-Match](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/If-None-Match) 头来验证缓存。

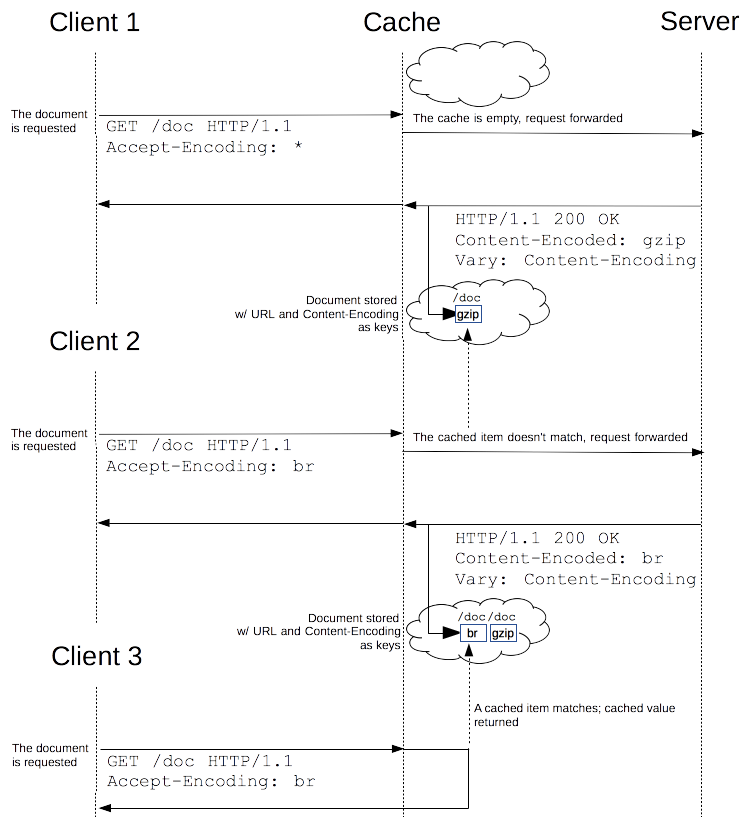
[Last-Modified](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Last-Modified) 响应头可以作为一种弱校验器。说它弱是因为它只能精确到一秒。如果响应头里含有这个信息，客户端可以在后续的请求中带上 [If-Modified-Since](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/If-Modified-Since) 来验证缓存。

当向服务端发起缓存校验的请求时，服务端会返回 [200](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Status/200) ok表示返回正常的结果或者 [304](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Status/304) Not Modified(不返回body)表示浏览器可以使用本地缓存文件。304的响应头也可以同时更新缓存文档的过期时间。

**带Vary头的响应[Edit](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Caching_FAQ$edit" \l "%E5%B8%A6Vary%E5%A4%B4%E7%9A%84%E5%93%8D%E5%BA%94)**

[Vary](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Vary) HTTP 响应头决定了对于后续的请求头，如何判断是请求一个新的资源还是使用缓存的文件。

当缓存服务器收到一个请求，只有当前的请求和原始（缓存）的请求头跟缓存的响应头里的Vary都匹配，才能使用缓存的响应。



使用vary头有利于内容服务的动态多样性。例如，使用Vary: User-Agent头，缓存服务器需要通过UA判断是否使用缓存的页面。如果需要区分移动端和桌面端的展示内容，利用这种方式就能避免在不同的终端展示错误的布局。另外，它可以帮助google或者其他搜索引擎更好地发现页面的移动版本，并且告诉搜索引擎没有引入[Cloaking](https://en.wikipedia.org/wiki/Cloaking)。

Vary: User-Agent

因为移动版和桌面的客户端的请求头中的User-Agent不同， 缓存服务器不会错误地把移动端的内容输出到桌面端到用户。

**参考**[**Edit**](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Caching_FAQ$edit#%E5%8F%82%E8%80%83)

* [RFC 7234: Hypertext Transfer Protocol (HTTP/1.1): Caching](https://tools.ietf.org/html/rfc7234)
* [Caching Tutorial – Mark Nottingham](https://www.mnot.net/cache_docs)
* [HTTP caching – Ilya Grigorik](https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching)
* [RedBot](https://redbot.org/), 检查缓存相关的 HTTP 头的工具.