缓存

HTTP缓存可以分为两种分为强制缓存，协商缓存。

①强制缓存判断HTTP首部字段：Expires ，cache-control。

expire是个过期时间(绝对时间，服务器时间)。所以当浏览器再次向服务器请求资源的时候，浏览器就会先对比本地时间和expires的保质期，如果本地时间小于expires的保质期(还没失效)，那么就直接去缓存中取这个资源。

但是该方法存在一个问题：服务器时间与客户端时间可能不一致。因此该字段已经很少使用。

cache-control中的max-age保存一个相对时间。例如Cache-Control: max-age = 484200，表示浏览器收到文件后，缓存在484200s内均有效。**如果同时存在cache-control和Expires，浏览器总是优先使用cache-control。**

关于Cache-Control的另一个参数public，告诉浏览器是否可以被代理服务器缓存，如果我们只想浏览器缓存，默认设置private便可

Cache-Control还有一个为代理浏览器服务的参数叫做s-maxage，如果两者同时出现且s-maxage未过期，则向代理服务器请求其缓存内容。在大型项目中，架构会依赖各种代理服务器，所以我们不得不去考虑代理服务器的缓存问题

强缓存会比协商缓存的优先级要高，这是因为协商缓存也会向服务器发送http请求的！

②协商缓存(对比缓存)通过HTTP的last-modified，Etag字段进行判断。

Last-Modified是第一次请求资源时，服务器返回的字段，表示最后一次更新的时间。下一次浏览器请求资源时就发送If-Modified-Since的时间戳字段,它的值正是上一次请求时Last-Modified的值。

服务器用本地Last-modified时间与接收到的If-Modified-Since时间比较，如果不一致则认为缓存已过期并返回新资源给浏览器(200状态码和最新的修改日期)；如果一致则发送304状态码，让浏览器继续使用缓存。

出现的问题:

如果我们编辑了文件，但是并没有对文件内容做修改

=>Last-Modified时间戳也会变

=>不该重新请求的时候也去重新请求了

当我们修改文件的手速巨快

=>If-Modified-Since感知不到这个改动

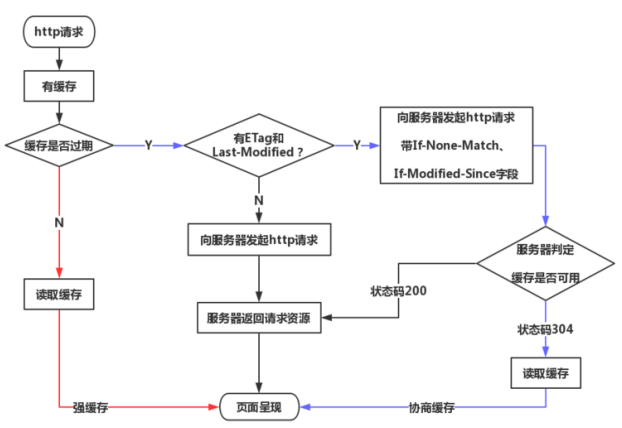
=>该重新请求的时候不重新请求

=====>etag作为Last-Modified的补充就出现了。

etag和Last-Modified类似，当我们首次请求时，会在响应头里获取到一个最初的标识字符串，这个是服务器算出的hash值，比last-modified更准确

等到用户下一次请求的时候，请求头里会带上一个名为If-None-Match的值给服务器来比对，这个值就是上一次服务器返回的 etag 的值

服务器会判断Etag是否发生变化，如果变化则返回新资源，否则返回304。



**讲讲304缓存的原理**

服务器首先产生ETag，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。

304是HTTP状态码，服务器用来标识这个文件没修改，不返回内容，浏览器在接收到个状态码后，会使用浏览器已缓存的文件

客户端请求一个页面（A）。服务器返回页面A，并在给A加上一个ETag。 客户端展现该页面，并将页面连同ETag一起缓存。 客户再次请求页面A，并将上次请求时服务器返回的ETag一起传递给服务器。 服务器检查该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（未修改——Not Modified）和一个空的响应体。

当浏览器要请求资源时

①调用 Service Worker 的 fetch 事件响应

②查看 memory cache

③查看 disk cache。这里又细分：

如果有强制缓存且未失效，则使用强制缓存，不请求服务器。这时的状态码全部是 200

如果有强制缓存但已失效，使用对比缓存，比较后确定 304 还是 200

1. 发送网络请求，等待网络响应
2. 把响应内容存入 disk cache (如果 HTTP 头信息配置可以存的话)
3. 把响应内容 的引用 存入 memory cache (无视 HTTP 头信息的配置)
4. 把响应内容存入 Service Worker 的 Cache Storage (如果 Service Worker 的脚本调用了 cache.put())

**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；

而前端缓存则可以在剩下的两步：“请求”和“响应”中进行。在“请求”步骤中，浏览器也可以通过存储结果的方式直接使用资源，直接省去了发送请求；而“响应”步骤需要浏览器和服务器共同配合，通过减少响应内容来缩短传输时间。

浏览器发出的所有HTTP请求会首先路由到浏览器缓存,已确认是否缓存可用于请求的有效响应。如果有匹配的响应,则从缓存中读取响应,这样就避免了网路延迟和传送产生的流量费用

**更新或废弃缓存的响应**

如何才能实现客户端缓存和快速更新,你可以在资源内容发生变化时,更改它的网址,强制用户下载新响应。通常情况下,可以通过在文件名中嵌入文件的指纹或版本号来实现

HTML被标记为no-cache

这意味着浏览器再每次请求时都始终重新验证文档,并在内容变化时获取最新版本。此外再HTML标记内,再CSS和javascript中嵌入指纹,如果这些文件的内容发生变化,网页的HTML也会随之改变,并会下载HTML响应的新副本

允许浏览器和中间缓存(例如CDN)缓存CSS,并将CSS设置为1年后到期,因为再文件名中嵌入了文件的指纹,CSS更新时网址也会随之变化

JavaScript同样设置为1年后到期,但标记为private,这或许是因为它包含的某些用户私人数据是CDN不应缓存的。

图像缓存时不包含版本或唯一指纹,并设置为一天后到期

