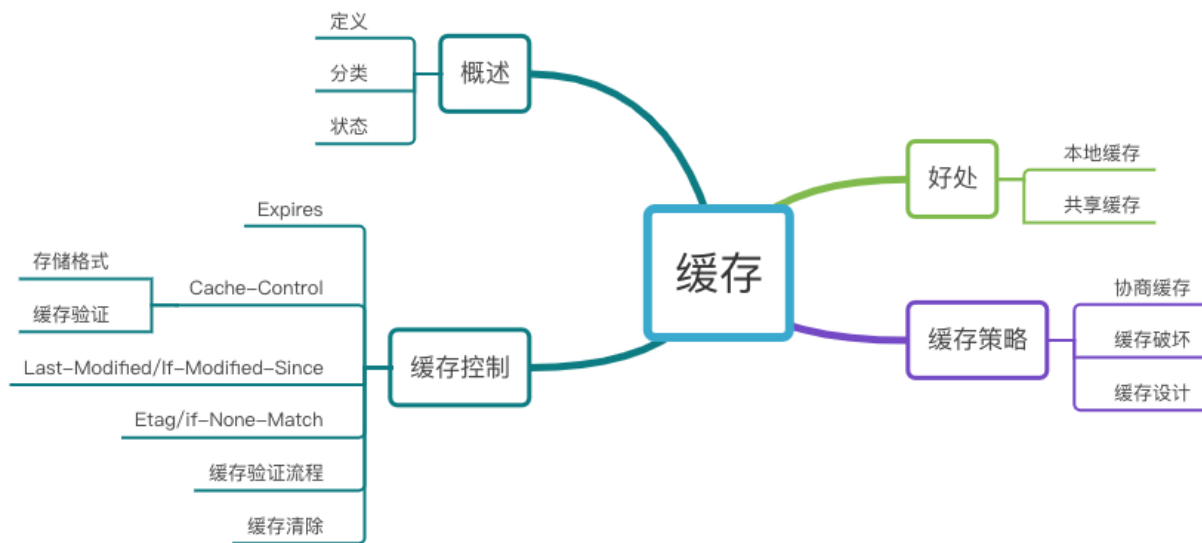


缓存优化数据及原理分析

文章内容

- 缓存概述
- 缓存的好处
- 缓存控制
- 缓存策略



缓存概述

缓存即在本地存储数据的过程，将获取来的数据资源存储在指定的地方，以便将来可以更快地进行访问。而浏览器（HTTP）缓存是我们目前使用最多也是最常见的缓存类型。HTTP 缓存存储与请求关联的响应，并将存储的响应复用于后续请求。

响应的可复用性可以给网站带来极大的性能提升，但同时可能会带来灾难性的后果，所以正确掌握缓存使用是至关重要。

缓存的分类

根据不同的标准，对缓存有着不同的分类

1、在 [HTTP Caching](#) 标准中，有两种不同类型的缓存：**私有缓存**和**共享缓存**。

私有缓存：缓存与用户代理所绑定，不与其他用户代理共享。简单说响应存储在客户端，不会被其他客户端所访问。

共享缓存：共享缓存位于客户端和服务端之间，可以存储能在用户之间共享的响应。共享缓存可以进一步细分为**代理缓存**和**托管缓存**。

2、按照是否需要去服务器重新验证分为：**强缓存和协商缓存**。

强缓存：如果命中缓存则直接使用缓存，不需要去服务端验证资源的有效性。

协商缓存：在使用缓存资源前，必须去服务端重新验证资源的有效性，如果新鲜则使用缓存，不新鲜则返回新的资源响应。

缓存的状态

缓存的状态分为两种：**fresh** 和 **stale**。fresh表示缓存的响应还是新鲜的也就是有效，可以被重复使用，而stale则表示缓存过时。

缓存状态是fresh还是stale取决于缓存响应的**age**，age表示响应报文创建时到现在所经历的时间，通俗说就是响应的年龄。由当前时间与响应首部的Date字段的差值决定。最后与max-age比较，小于则表示新鲜，否则表示资源过时。

缓存的好处

1. 无需发起网络请求，通过复用缓存资源，减少了客户端等待时间，加快页面渲染。
2. 减少了网络流量，同时也缓解了服务端压力。

本地缓存

研究表明，一秒钟的延迟会导致转化率下降7%，页面浏览量下降11%，客户满意度下降16%。

因此资源加载的越快延迟越低，能很大程度上做到挽留客户，避免客户流失。

下面是一些项目的本地抽样测试的测试数据，从数据上更直观的说明使用缓存所带来的性能优化。

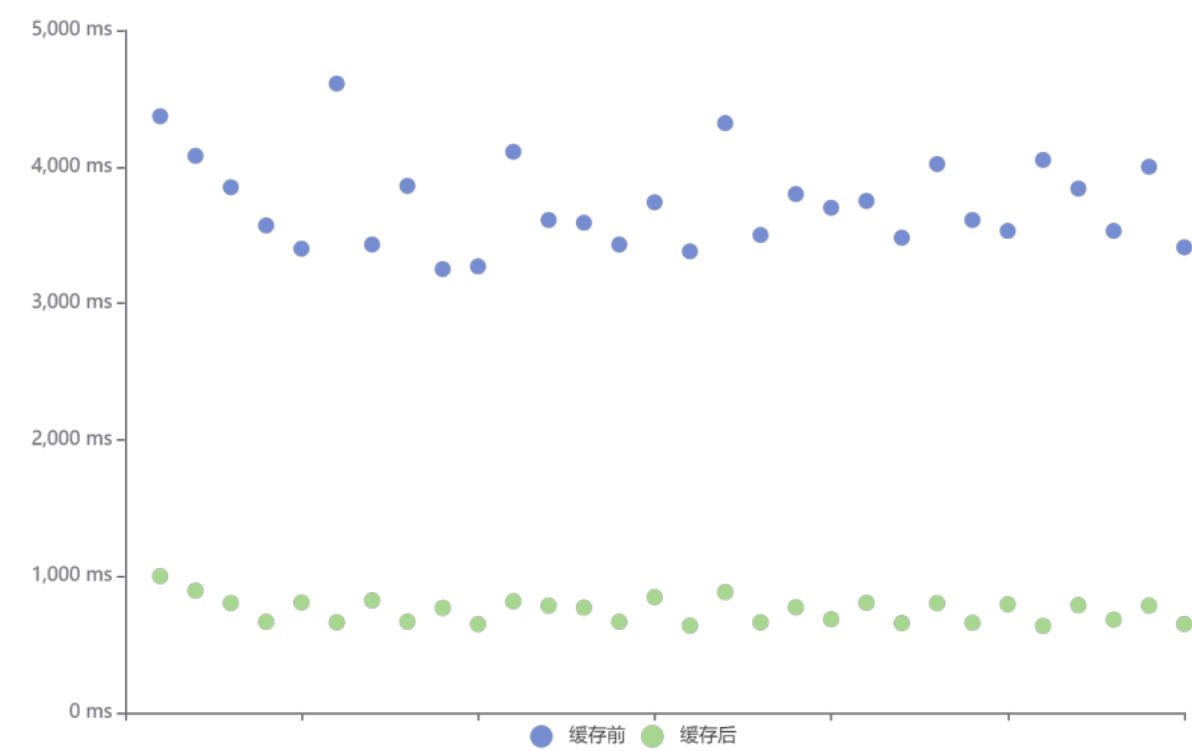
首先，进行了公司实体项目中信在没有使用缓存前和使用缓存后，文件资源的加载时间对比，如下：

测试参数：

测试环境	本地（chrome浏览器）
资源文件大小	19.2MB
网络带宽	100Mbit/s
服务器距离	中信服务器

数据散点图：

响应速度



图表分析：

检测对象	检测点数	性能(毫秒)		
		均值	最好	最差
禁用缓存	30	4378	3250	4610
启用缓存	30	749.83	635	1000

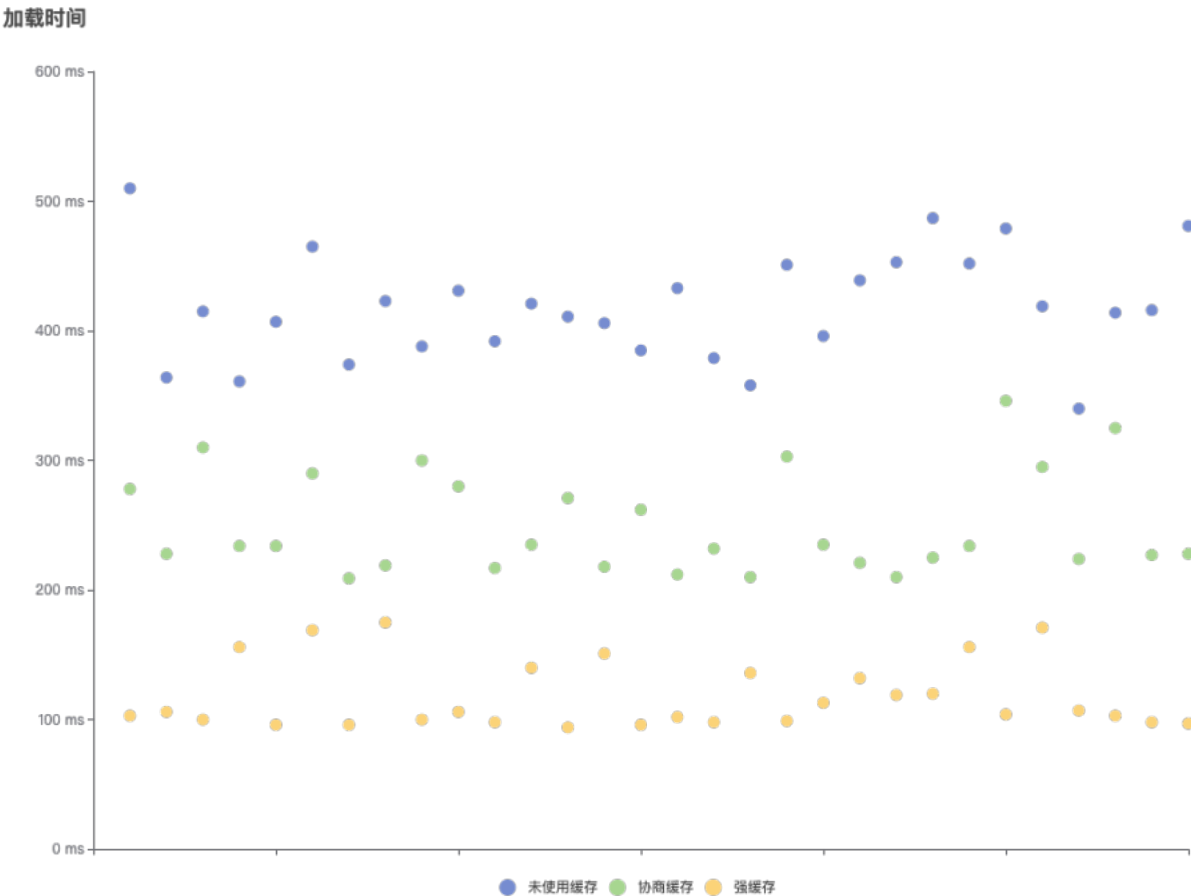
以上可以看出未使用缓存和使用缓存之间还是有着非常明显的差距的，未使用缓存资源的加载时间在3~5s之间，使用缓存后时间缩短至1s以内。

然后下面是我自己写的一个demo，主要用来测试未启用缓存、协商缓存和强缓存之间文件资源加载时间对比，如下：

测试参数：

测试环境	本地（chrome浏览器）
资源文件大小	1.1MB
网络带宽	100Mbit/s
服务器距离	本地服务器

数据散点图：



图表分析：

检测对象	检测点数	性能(毫秒)		
		均值	最好	最差
禁用缓存	30	418.33	340	510
协商缓存	30	250.4	209	346
强缓存	30	118	94	175

强缓存所用时间最少，协商缓存次之最后是未启用缓存。

而加载时间差距也会随着服务器距离、传输的资源大小和网络带宽大小不同而不同。一般来说离服务器越远、资源文件越大带宽越小，他们之间的差距也会越来越大。

共享缓存

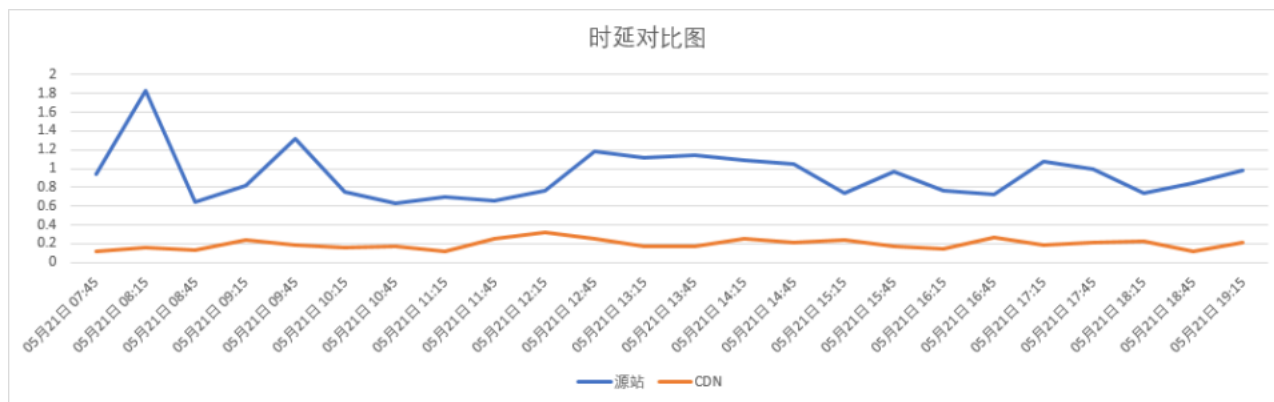
以上都是本地缓存的测试数据，接下来看共享缓存所带来性能优化。

共享缓存分为代理缓存和托管缓存，它们的本质上都是减少用户与服务器之间的物理距离，从而缩短资源的加载时间来达到加速的效果，这里将简单介绍目前广泛使用的托管缓存CDN。

内容分发网络（Content Delivery Network，CDN），是在现有 Internet 中增加的一层新的网络架构，由遍布全球的高性能加速节点构成。这些高性能的服务节点都会按照一定的缓存策略存储您的业务内容，当您的用户向您的某一业务内容发起请求时，请求会被调度至最接近用户的服务节点，直接由服务节点快速响应，有效降低用户访问延迟，提升可用性。

因为CDN无法进行本地测试，所以借鉴了腾讯云在CDN模块中做的性能指标测试数据如下：

单位：s



图表分析

检测对象	检测点数	性能（秒）		
		均值	最好	最差
CDN	2235	0.196	0.117	0.313
源站	2177	0.933	0.635	1.827

所有的测试数据和环境相关的可以去[腾讯云CDN测试数据](#)详细查看。

从上不难看出使用CDN缓存加速，资源加载的速度也是得到了很大的提升。但是由于共享缓存是位于用户与源站之间的，不管这样资源的加载效率都低于本地缓存。

所以从资源的加载时间来看：本地缓存 > 共享缓存 > 源站。而强缓存 > 协商缓存。

而本地缓存和共享缓存是互无相关的，搭配使用可以最大程度降低页面加载延迟，从而获得良好的用户体验。

离线访问

缓存还有一大好处是在没有网络的情况下依旧能进行页面访问，想要实现离线访问还需要借助 service-worker 技术。

Service workers 本质上充当 Web 应用程序、浏览器与网络（可用时）之间的代理服务器。这个 API 旨在创建有效的离线体验，它会拦截网络请求并根据网络是否可用来采取适当的动作、更新来自服务器的资源。

因为Service workers充当中间代理服务器，所以可以完全按照我们自己的缓存要求去存储响应资源副本，因此能够做到离线访问的目的。

缓存控制

1、Expires

在HTTP1.0标准中，缓存是由Expires响应首部控制。

Expires 首部使用明确的时间而不是通过指定经过的时间来指定缓存的生命周期。

```
expires: Sat, 19 Nov 2022 19:14:18 GMT
```

当当前时间不超过expires指定时间则表示新鲜，否则表示不新鲜过时。

但是expires时间格式难以解析，并且可以通过修改本地时间来绕过缓存或者命中缓存。

2、Cache-Control

Cache-Control是HTTP1.1标准中新增的用来进行缓存控制的通用首部，相比较于expires，cache-control提供了更为细致的缓存控制手段。

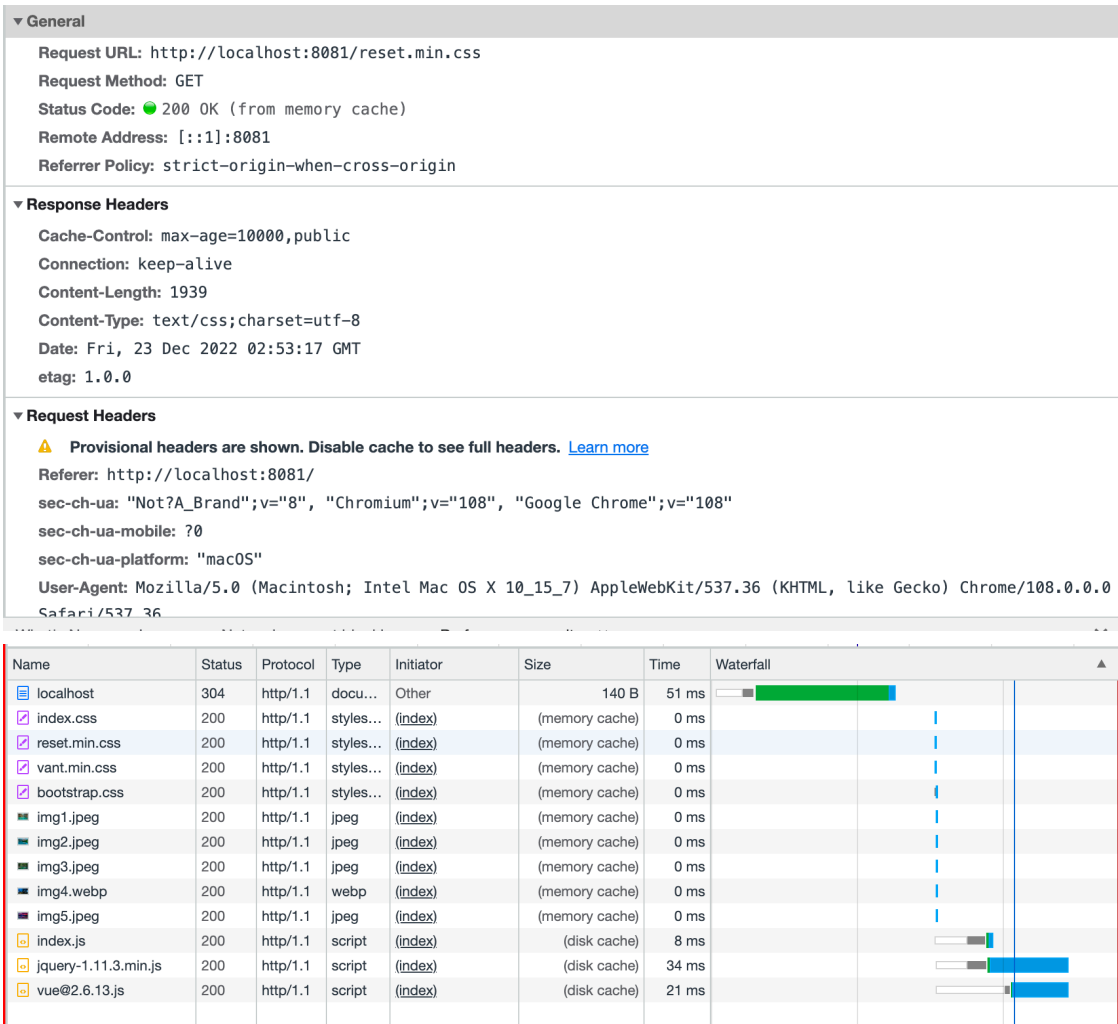
Cache-Control通过不同的指令来定义缓存策略，一些指令如下表：

指令	含义	响应头	请求头
max-age	缓存的生命周期，响应报文生成时在指定时间内缓存是新鲜的，单位s。		
s-maxage	共享缓存的生命周期，当设置了s-maxage，max-age将失效，只对共享缓存生效。		
no-cache	在使用缓存之前必须去服务器重新验证资源的有效性。		
no-store	客户端和中间缓存不进行任何缓存。		
public	共享缓存，资源可以被用户代理和中间缓存进行缓存。		
private	私有缓存，资源只能被用户代理所缓存。		
must-revalidation	当缓存过期时，必须去服务端验证资源的有效性，在验证资源的有效性之前不允许使用缓存。		
max-stale	允许的最大过期时间，单位s。允许资源过期后，但是过期时间在max-stale时间内，复用缓存资源。		
min-fresh	要求保证缓存资源最小时间新鲜度，希望服务器返回的max-age不小于min-fresh。		

max-age和no-cache是cache-control比较常用，也是比较特殊的两个指令，因为它们既可以作为请求头也可以作为响应头。

max-age

作为响应头时表示缓存的生命周期，响应报文生成时在指定时间内缓存是新鲜的。



max-age指定了10000s，所以接下来10000s内缓存都是有效的，因此可以看到再次发起请求时都是读取本地缓存，而不是发起网络请求去服务器获取。

作为请求头时表示复用 age 小于或等于max-age的缓存，例如请求中的 max-age=0 指令指定“重用 age 为 0 或更少的响应”，然后请求通过 If-None-Match 和 If-Modified-Since 进行验证。请求中指定 max-age=0 也被认为重新加载，因为max-age等于0是不会使用缓存。

[illegible]

上图请求头中指定了max-age等于0，所以尽管缓存还没有过期都会去重新加载，加载过程中会在服务端验证资源的有效性，如果有效则返回304。

no-cache

作为响应头表示协商缓存，每次使用缓存之前都需要去服务端验证资源的有效性。

```
▼ General
Request URL: http://localhost:8081/bootstrap.css
Request Method: GET
Status Code: 304 Not Modified
Remote Address: [::1]:8081
Referrer Policy: strict-origin-when-cross-origin

▼ Response Headers
View source
Cache-Control: no-cache
Connection: keep-alive
Date: Fri, 23 Dec 2022 03:04:20 GMT
etag: 1.0.0
```

如果资源有效则返回304， 否则返回200并携带最新的资源实体。

作为请求头表示强制重新加载，意味着不使用缓存，强制去服务器获取资源。


▼ **Request Headers** [View source](#)

Accept: text/css,*/*;q=0.1
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Cache-Control: no-cache
Connection: keep-alive
Host: localhost:8081
Pragma: no-cache
Referer: http://localhost:8081/
sec-ch-ua: "Not?A_Brand";v="8", "Chromium";v="108", "Google Chrome";v="108"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Sec-Fetch-Dest: style

当chrome浏览器禁用缓存或者硬性重新加载时，浏览器会默认添加上cache-control和pragma为no-cache表示强制重新加载，不再使用缓存。

缓存的存储格式

区分缓存响应的方式本质上是基于它们的 URL，类似下表：



URL	Response
https://example.com/index.html	<!doctype html>...
https://example.com/style.css	body { ...
https://example.com/script.js	function main () { ...

在请求的url和返回的响应之间建立映射关系，以方便通过请求url读取缓存资源。


但是相同的url不一定返回相同的响应内容，因为HTTP标准中还存在内容协商机制，来自服务器的响应可能取决于 Accept、Accept-Language 和 Accept-Encoding 请求首部的值。

例如，对于带有 Accept-Language: en 请求首部并已缓存的英语内容，不希望再对具有 Accept-Language: ja 请求首部的请求重用该缓存响应。在这种情况下，可以通过在 Vary 响应首部的值中添加“Accept-Language”，根据语言单独缓存响应。

Vary 是一个 HTTP 响应头部，它被服务器用来表明在 [content negotiation algorithm](#)（内容协商算法）中选择一个资源代表的时候应该使用哪些头部信息（headers）。

```
Vary: Accept-Language;
```

当响应的首部字段中包含Vary时，就变成下表形式：



URL	Accept-Language	Response
https://example.com/index.html	ja-JP	<!doctype html>...
https://example.com/index.html	en-US	<!doctype html>...
https://example.com/style.css	ja-JP	body { ...
https://example.com/script.js	ja-JP	function main () { ...

可以看到映射关系不再简单的是url到响应，还需要加上Accept-Language，只有url和Accept-Language的值同时满足才能获取到相应的缓存资源。

缓存验证

通过映射关系获取到缓存资源后，还需要去验证资源的有效性。

HTTP缓存是基于age的验证策略，通过当前时间与报文的生成时间Date计算age，然后与max-age进行比较判断当前缓存资源是否新鲜。

举例：

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 22:22:22 GMT
Last-Modified: Tue, 22 Feb 2022 22:00:00 GMT
Cache-Control: max-age=3600
<!doctype html>
...
```

表示缓存资源在2022/02/22 22:22:22之后一个小时内是有效的。

当缓存的响应过期时也不会立即被丢弃。因为HTTP 有一种机制，在本地验证资源过期后，会发请求去询问服务器当前缓存资源是否依旧有效，如果有效则响应 304 Not Modified，不返回响应实体。如果无效则响应200 OK，返回最新的响应资源，更新本地缓存。

去服务端验证通过使用包含 If-Modified-Since 或 If-None-Match 请求首部的条件请求完成的。

3、Last-Modified/If-Modified-Since

Last-Modified响应首部，表示资源文件最后的修改时间。

```
Last-Modified: Tue, 22 Feb 2022 22:00:00 GMT
```

当资源过期后，发送请求到服务端验证时会携带If-modified-since请求首部，该字段的值就是缓存响应的Last-Modified值，到服务端进行比较后判断文件是否被修改，以此判断当前缓存资源是否依旧有效。

如果有效则服务器返回304 Not Modified。否则返回200 OK和资源的最新版本响应。

缺点：

1. 时间格式难以解析。
2. 分布式服务器难以同步文件更新时间。
3. 存在文件内容不变但是最后修改时间变更情况，例如进行了文件编辑，但是没有进行任何内容修改，文件的最后修改时间也会发生变化。

基于以上提出了Etag解决方案。

4、Etag/if-None-Match

Etag响应首部是资源文件的标识，可以按照服务器要求设置任意值，例如正文内容的hash值和版本号。

当正文内容发生改变时，需要生成新的Etag值来表示资源文件的变更。

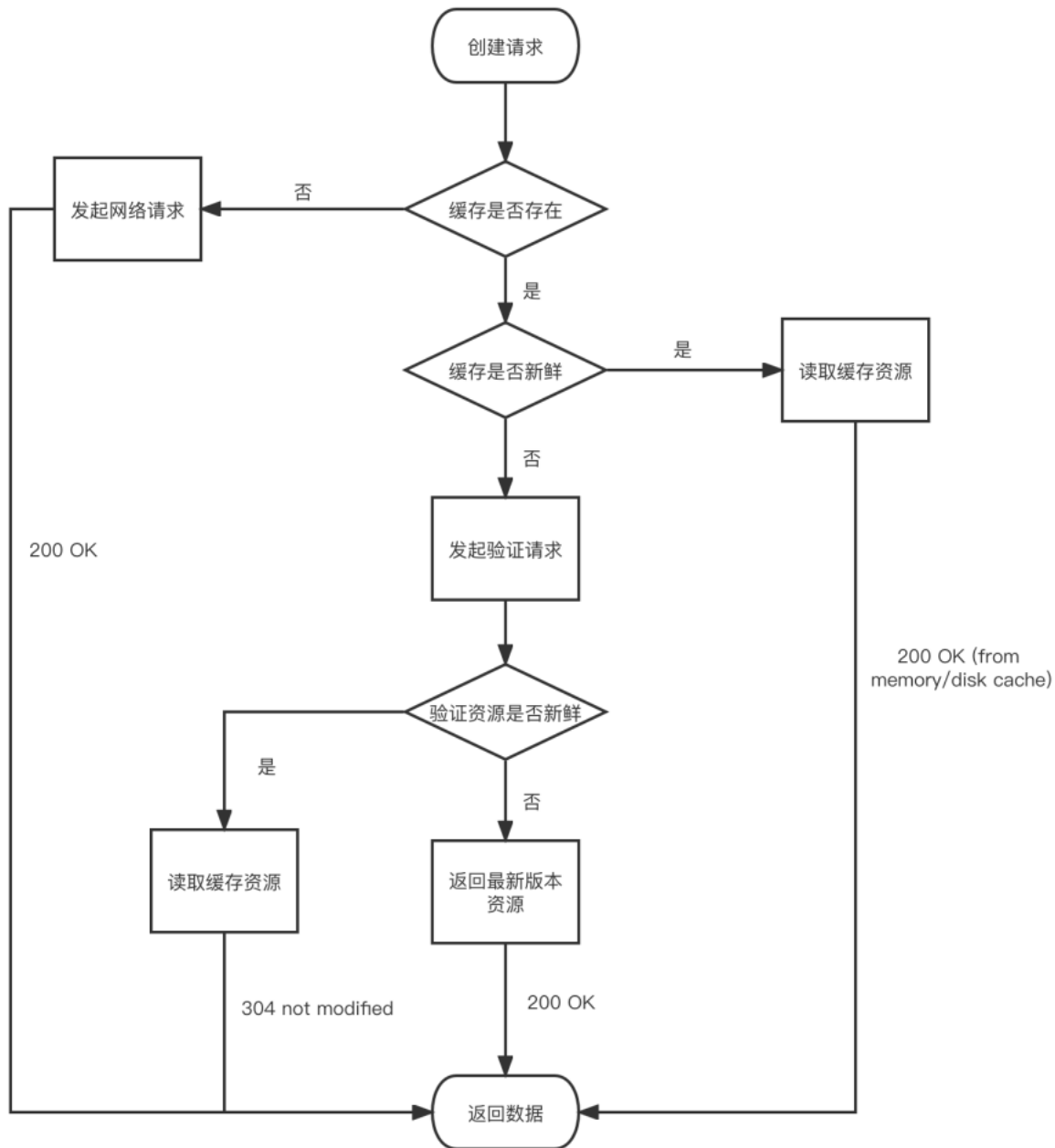
```
etag: W/"1547c-FBWJ3DKddf7pJH/C6fOMIXWhAu0"
```

与Last-Modified一样，会在资源过期需要到服务器重新验证时起作用，验证请求会携带着if-None-Match请求首部，值为缓存的Etag值，到服务器进行比较验证，判断当前缓存是否依旧有效。

在重新验证请求中，如果同时包含etag和last-modified值时，以etag值优先。

如果单从缓存的角度考虑。last-modified是不必要的，但是last-modified只是单纯的一个字段用来表示资源文件的修改时间，除了用做缓存处理之外还有着其他意义则需要加上。

5、缓存验证流程



1. 创建资源请求对象
2. 获取请求url与vary定义的相关字段，判断缓存是否存在，不存在则发起网络请求获取资源。
3. 如果存在则需要判断缓存是否新鲜，获取请求中与缓存相关字段cache-control与expires等，基于age的验证策略判断缓存是否新鲜。如果新鲜则复用响应并将请求状态改为200 OK (from memory/disk cache)。
4. 如果不新鲜则需要发送该请求到服务端重新验证资源的有效性，获取缓存中的last-modified和etag值赋予if-modified-since和if-none-match字段并添加到请求头中。

5. 如果服务端验证资源依旧有效，则返回304 Not Modified，不返回响应实体，客户端复用缓存响应。如果资源过期则返回200 OK，并且返回最新资源响应。

6、缓存清除

如果服务器资源更新，但是因为缓存的缘故请求到达不了服务器而导致用户访问的还是一些过时资源版本。

这时希望能够清除缓存从而去服务器获取最新的资源版本，但是HTTP标准中没有定义删除缓存的方法，所以缓存只能人为的清除。

针对于用户的本地缓存，可以手动清除浏览器缓存或者禁用缓存来达到访问服务器资源的目的。

而共享缓存例如CDN一般都会提供操作面板允许对缓存进行刷新等处理。

因此，针对用户的本地缓存需要一个合适的缓存策略，以保证用户每次拿到的资源都是最新的版本。

缓存策略

协商缓存

```
Cache-Control: no-cache;
```

or

```
Cache-Control: max-age=0,must-revalidation;
```

上面两条指令的作用是一样的，在使用缓存资源前，强制去服务器重新验证资源的有效性，只有资源有效才允许使用缓存资源。

因此，如果所有的静态资源都使用协商缓存，在使用缓存之前都去服务端验证下，那么用户每次访问的资源都会是最新版本。避免了上述存在的缓存问题。

都使用协商缓存的策略是存在一些问题的。例如存在一些静态资源可能托管到服务器以后就不会在发生变更，针对这些资源每一次都需要发请求去服务器重新验证资源的有效性就会浪费性能，减缓资源加载与页面渲染。

所以需要有一个允许使用本地缓存，但是在资源更新时能保证使用的是最新版本资源的缓存设计。

缓存破坏

缓存破坏的意思是指通过改变请求的url，来达到绕过缓存向服务器发起请求的目的。

以上可以知道缓存是以请求的url和vary定义的相关字段作为映射进行查找的，所以改变请求的url，可以达到不命中缓存，从而向服务器发起请求的目的。适用于所有子资源。

缓存设计

因此利用缓存破坏的思想，针对静态子资源例如css、js和img等等资源可以给一个很大的max-age值，使得资源被缓存的更长时间。

```
Cache-Control: max-age=31536000;
```

如上资源被缓存一年，在缓存的期间项目进行了版本迭代，资源发生了更新。那么请求url也需要进行更新，保证获取到的是最新的服务器资源。

可以使用带有版本号或者文件hash值的请求路径，如下：

version in filename

bundle.v123.js

version in query

bundle.js?v=123

hash in filename

bundle.YsAlAAAA-QG4G6kCMAMBAAAAAAoK.js

hash in query

bundle.js?v=YsAlAAAA-QG4G6kCMAMBAAAAAAoK

在项目打包时，可以给所有资源文件路径加上当前项目版本或者根据文件内容生成的hash值。不管怎样都能保证用户获取到的是最新的资源。

缓存破坏适用于子资源而不适合主资源HTML，因为子资源是受我们控制的，而HTML是用户在地址栏输入的url进行访问的，我们不能控制用户的行为，所以针对主资源只能使用协商缓存方法保证资源的有效性。

总结：

1. 主资源HTML使用协商缓存策略
2. HTML下的子资源使用强缓存与缓存破坏的缓存策略