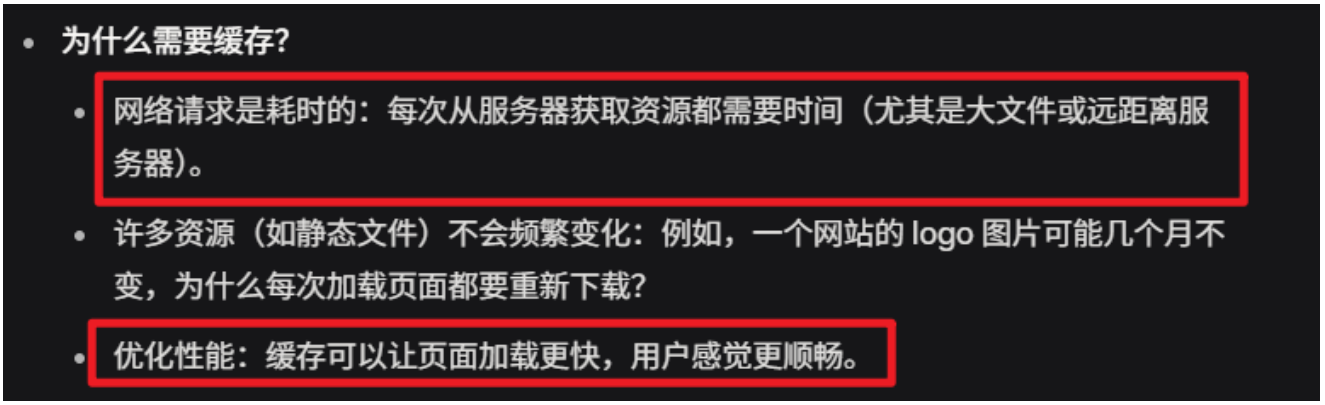
HTTP缓存机制

什么是HTTP缓存

HTTP 缓存是一种机制，用于存储 HTTP 请求的响应数据（如 HTML、CSS、JS、图片等资源），以便在后续请求相同资源时，**直接从缓存中读取**，而不需要重新从服务器获取。**这可以节省网络带宽、减少延迟，并降低服务器负载。**



HTTP 缓存主要发生在浏览器端（浏览器缓存），但也可以在代理服务器（如 CDN 或代理缓存）上发生。作为前端开发者，你主要关注浏览器缓存。

缓存分类

HTTP 缓存主要分为两大类：**私有缓存**和**共享缓存。**这两个类别是根据缓存的存储位置和使用范围来划分的。

私有缓存

(Private Cache)

私有缓存只服务于单个用户，最典型的例子就是你提到的**浏览器缓存。**

内存缓存 (Memory Cache)：这是一种短期、快速的缓存。它将资源直接存储在**计算机内存**中，**访问速度极快。当你在浏览器中来回切换页面或者重新加载时，如果资源还在内存里**，浏览器会直接从内存中读取。它的生命周期很短，通常在当前页面关闭后就会被清空。

磁盘缓存 (Disk Cache)：这是一种**长期、持久的缓存。**它将资源存储在**计算机的硬盘**上，即使浏览器关闭、电脑重启，这些缓存文件通常也依然存在。**当浏览器需要某个资源时，会先检查磁盘缓存，如果找到了并且没有过期，就可以直接使用**，这样可以避免再次向服务器发送请求。

这两种缓存协同工作，浏览器会根据不同的情况决定使用哪一种。比如，**对于短期内频繁访问的资源，浏览器更倾向于使用内存缓存以获得最快的速度**；而对于不经常访问但希望保留的资源，则会存储在磁盘缓存中。

共享缓存

(Shared Cache)

共享缓存服务于多个用户，它可以被所有用户访问。你提到的**代理缓存**和 **CDN 缓存**都属于这一类。

代理缓存 (Proxy Cache)：代**理服务器位于用户（浏览器）和源服务器之间。**当多个用户请求同一个资源时，代理服务器会把这个资源缓存下来。这样一来，**后续的请求就可以直接从代理服务器获取**，而不需要每次都去源服务器拉取。

CDN (内容分发网络) 缓存：CDN 是一种分布式网络系统，**它在全球各地部署了大量的边缘服务器。当用户请求资源时，CDN 会将请求导向距离用户最近的服务器节点。**这个节点会缓存源服务器上的内容。因此，CDN 缓存是代理缓存的更高级和更专业的应用，它显著提升了网站的访问速度和稳定性。

小结

HTTP 缓存是一种**客户端缓存**，它的核心思想是：**客户端（浏览器或代理）在发出 HTTP 请求后，将响应内容存储在本地**，以便下次访问时可以直接使用。**这种缓存由 HTTP 协议本身定义和控制**，主要通过响应头（如 `Cache-Control`、`Expires`、`ETag`、`Last-Modified`）来管理。

而**服务端缓存**是**服务器内部的缓存机制。**例如，**当服务器需要处理一个复杂的数据库查询时，它可以将查询结果暂时存放在 Redis 或 Memcached 等内存数据库中。**这样，下次有相同的查询时，服务器就可以直接从内存中读取结果，而不需要再次访问数据库。这个过程发生在服务器内部，对用户（客户端）是不可见的，它旨在提高服务器自身的处理效率。

简而言之，HTTP 缓存旨在减少**网络请求**，而服务端缓存旨在减少**服务器的内部处理开销。**

缓存策略

HTTP 缓存分为两种策略：**强缓存（Expiration Model）和协商缓存（Validation Model）。它们通过 HTTP 响应头和请求头来控制。**

强缓存

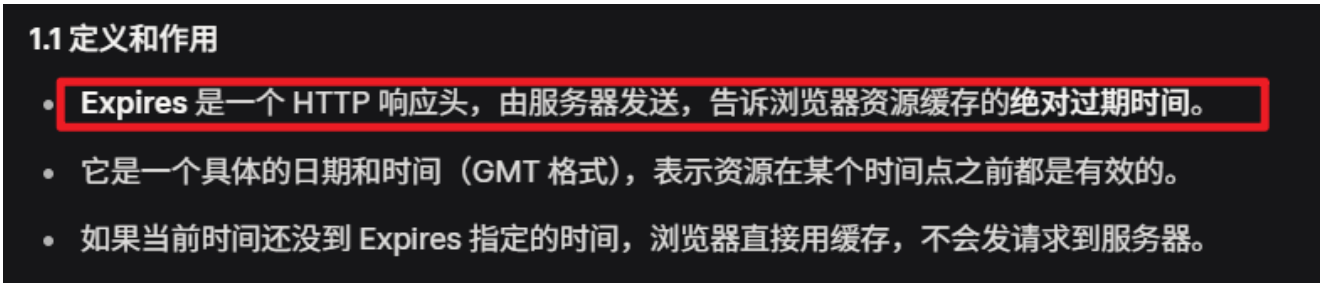
强缓存：浏览器直接从缓存中读取资源，**不需要向服务器验证。**适用于资源很少变化的场景。

强缓存的核心思想是：**告诉浏览器一个资源在什么时候会过期。**只要在这个有效期内，**浏览器就不用向服务器发起任何请求，直接使用本地缓存的文件。**

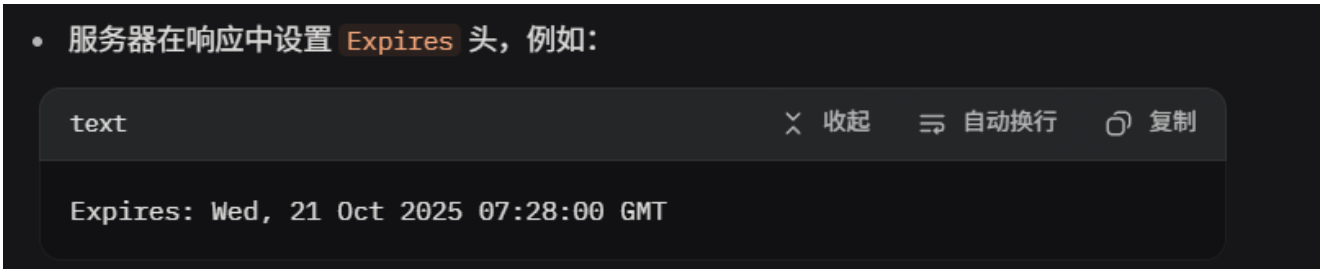
强缓存主要由两个HTTP头控制 ： （ Expires 头 Cache-Control 头）

Expires 头

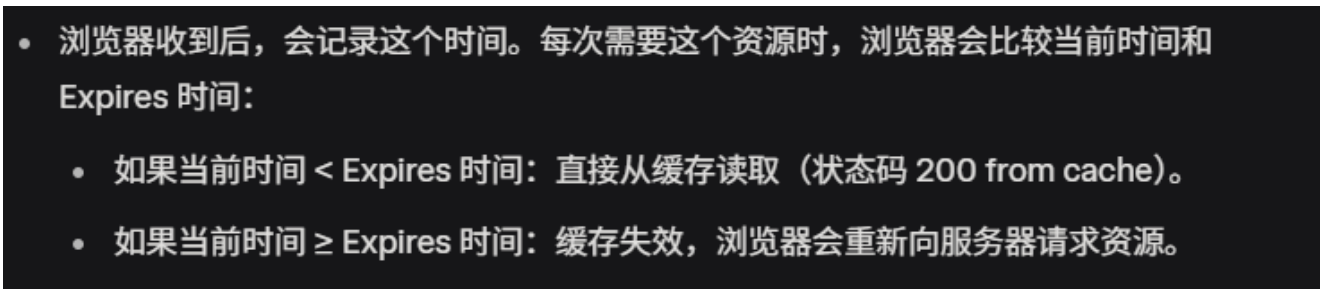
1.定义和作用



2.工作原理



这表示资源在 **2025 年 10 月 21 日 07:28:00（格林威治时间）**之前有效



浏览器收到之后进行记录，**每次读取资源的时候，与当下时间进行比较**

3.缺点

- **依赖客户端时间**：如果用户电脑时间不准（比如手动改了时间），可能导致缓存错误。比如，电脑时间被设到未来，缓存可能被误认为已过期。
- **不够灵活**：只能设置一个固定的过期时间，不能细粒度控制缓存行为。
- 因为这些问题，现代开发中 Expires 已经被 **Cache-Control** 取代，但仍然兼容使用。

示例

假设服务器返回一个图片资源的响应：

```text
HTTP/1.1 200 OK
Content-Type: image/png
Expires: Wed, 21 Oct 2025 07:28:00 GMT
```

Cache-Control 头

1.定义和作用

Cache-Control 是 HTTP/1.1 引入的响应头，功能更强大、更灵活，是现在强缓存的主要控制方式。

它通过设置一系列指令（directives），告诉浏览器如何缓存资源、缓存多久、**是否可以被代理服务器缓存等。**

优先级高于 Expires：如果响应中同时有 Expires 和 Cache-Control，浏览器会**优先使用 Cache-Control。**

2.工作原理

Cache-Control 使用键值对或指令，例如 **max-age=3600，表示资源缓存的有效时间（以秒为单位）。**

浏览器根据这些指令决定是否使用缓存。例如：

- 如果 max-age=3600（1小时），浏览器在收到资源后的 1 小时内直接用缓存。
- 过期后，浏览器会重新请求服务器。

3.常见指令

- `max-age=<seconds>`：资源缓存的有效时间（秒）。例如，`max-age=86400` 表示缓存 1 天（24小时）。
- `no-cache`：不使用强缓存，但可以用协商缓存（后面会讲到，初学者暂时不用深入）。
- `no-store`：完全禁用缓存，每次都必须从服务器获取（用于敏感数据，如银行信息）。
- `public`：允许浏览器和代理服务器（如 CDN）缓存资源。
- `private`：只允许浏览器缓存，代理服务器不能缓存（适合用户特定数据）。
- `must-revalidate`：缓存过期后，必须向服务器验证，不能用过期的缓存。

4.缺点

配置稍复杂，需要理解不同指令的含义。

如果设置不当（比如 max-age 太长），可能导致用户获取旧资源。

5.示例

```
HTTP/1.1 200 OK
Content-Type: text/css
Cache-Control: max-age=86400, public
```

- 浏览器缓存这个 CSS 文件。
- 在接下来的 24 小时（86400秒）内，浏览器直接从缓存读取。
- 代理服务器（如 CDN）也可以缓存，因为有 `public`。
- 24 小时后，缓存失效，浏览器会重新请求。

小结

特性	Expires	Cache-Control
定义	指定绝对过期时间（日期）	指定相对有效时间（秒）或其他指令
格式	GMT 时间，如 "Wed, 21 Oct 2025 07:28:00 GMT"	指令，如 `max-age=3600, public`
优先级	低于 Cache-Control	高于 Expires
依赖	客户端时间，可能不准	不依赖客户端时间，基于收到响应的时间
灵活性	单一，功能有限	多种指令，灵活性高
现代使用	较少用，HTTP/1.0 遗留	现代标准，广泛使用

协商缓存

协商缓存：浏**览器先向服务器询问资源是否变化**，如果没变则使用缓存；如果变了则重新下载。适用于资源可能变化的场景。

当强缓存过期后，浏览器并不会立刻放弃本地文件，而是会尝试和**服务器“协商”**，看看文件有没有更新。这个过程就是协商缓存。它通过在**请求头**中携带**本地文件的标识，让服务器来判断文件是否需要更新。**

协商缓存主要靠两个组控制头：**Last-Modified/If-Modified-Since**（基于时间）和 **ETag/If-None-Match**（基于内容指纹）。**它们是成对的：服务器用响应头设置，浏览器用请求头询问。**

（注意这个两个请求头是浏览器带给服务器的）

Last-Modified/If-Modified-Since

请求流程

1.**初次请求**

浏览器请求一个资源，**比如** `https://example.com/logo.png`

服务器返回**last-Modified**(响应头)

浏览器保存：

- **文件内容**
- **响应头的** `Last-Modified` **时间戳**

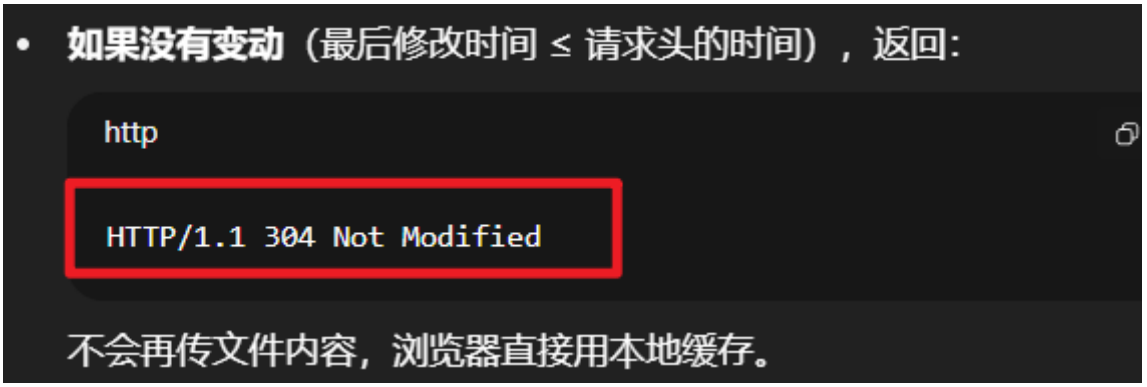
2.再次请求

浏览器请求这个资源时，会在 **请求头** 里加上：

这句话的意思是：**“服务器啊，我这边有一个 2024-08-21 07:28:00 的副本，请问它还是最新的吗？”**

3.服务器响应

没有变动-返回304

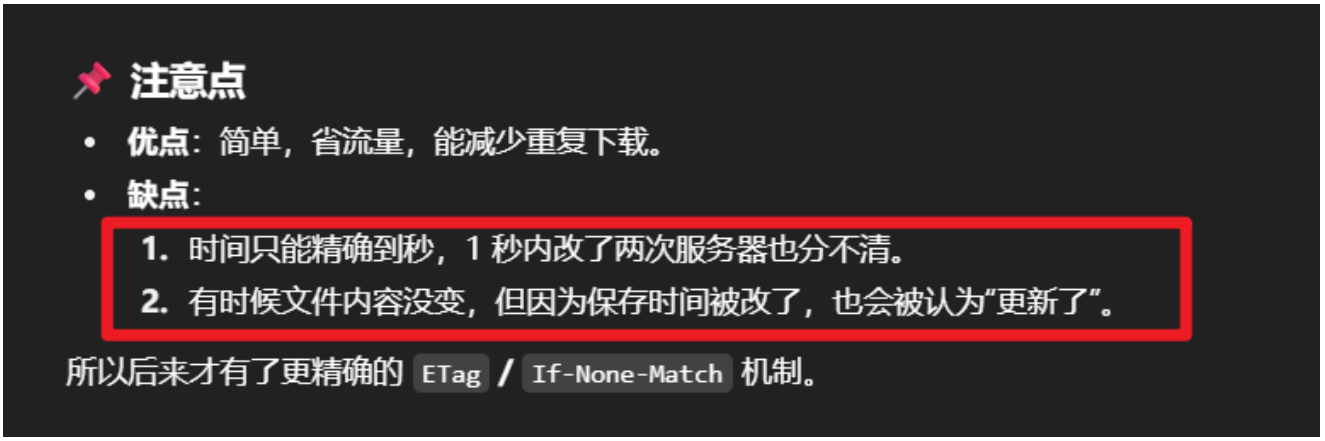


变动了


如果变动了（最后修改时间 > 请求头的时间），返回:
http
HTTP/1.1 200 OK
Last-Modified: Wed, 28 Aug 2024 09:00:00 GMT
同时带上最新的资源内容，浏览器替换本地缓存。

4.流程


总结流程图
mathematica
第一次请求 → 服务器返回资源 + Last-Modified 时间
↓
浏览器保存资源 + 时间戳
↓
第二次请求 → 浏览器带上 If-Modified-Since 时间
↓
服务器检查:
├ 文件没改 → 返回 304（用缓存）
└ 文件改过 → 返回 200（新资源 + 新 Last-Modified）


注意点
优点：简单，省流量，能减少重复下载。
缺点:
1. 时间只能精确到秒，1 秒内改了两次服务器也分不清。
2. 有时候文件内容没变，但因为保存时间被改了，也会被认为"更新了"。
所以后来才有了更精确的 ETag / If-None-Match 机制。

ETag/If-None-Match

1.什么是ETag

ETag（Entity Tag） 是服务器为某个资源生成的一个**唯一标识（类似“指纹”或者文件版本号）。**

它常常是资源内容的 **哈希值**、文件内容的 **摘要**，或者是数据库版本号。

每次资源内容改变，服务器就会给它生成一个新的 ETag。

2.初次请求

（注意：`Cache-Control: no-cache` 表示虽然要走缓存，但每次都要问一下服务器资源是否更新）

3.再次请求

浏览器再次请求资源时，会带上上次记录的 **ETag 值**，放在请求头 `If-None-Match` 里：

```sql
GET /style.css
If-None-Match: "v1-asdf123"
```

4.服务器处理

- 如果资源的 ETag 还是 `"v1-asdf123"`，说明内容没变，就返回：

```mathematica
HTTP/1.1 304 Not Modified
```

(不再返回内容，浏览器直接用本地缓存)

- 如果资源 ETag 改了，比如 `"v2-xyz456"`，说明文件被改动过，服务器就返回新的资源和新的 ETag：

```vbnet
HTTP/1.1 200 OK
ETag: "v2-xyz456"
Content: ...（新内容）
```

小结

🔹 ETag vs Last-Modified 的区别

特点	Last-Modified	ETag
对比方式	通过文件的最后修改时间	通过资源内容生成的唯一标识
精确度	秒级（改动太快可能不准确）	字节级（内容改1个字符都算不同）
局限性	文件时间可能没变但内容变了； 或者时间变了但内容没变	生成 ETag 需要服务器额外计算（可能耗性能）
常见用途	静态资源（图片、CSS、JS）	动态资源（接口数据、页面片段）

🔑 总结（适合初学者记忆）

- **Last-Modified / If-Modified-Since**：靠 **时间戳** 比对（粗粒度）
- **ETag / If-None-Match**：靠 **唯一标识** 比对（细粒度，更精确）
- 实际上，很多服务器会 **两个机制一起用**，提高兼容性。

协商缓存流程

🔷 协商缓存完整流程图

```sql

第一次请求资源
浏览器 → GET /resource
服务器 → 200 OK + 资源内容
          Last-Modified: <时间戳>
          ETag: "<唯一标识>"
浏览器缓存资源 + 记录 Last-Modified 和 ETag

第二次请求资源（协商缓存）
浏览器 → GET /resource
          If-Modified-Since: <上次 Last-Modified>
          If-None-Match: "<上次 ETag>"

服务器检查：
┌──────────────────────────────────────┐
| 资源内容没变?                          |
| ├─ 时间没变 AND ETag没变 → 304 Not Modified |
| |   浏览器继续用缓存                    |
| └─ 否 → 200 OK + 新资源 + 新 Last-Modified + 新 ETag
|     浏览器替换缓存
└──────────────────────────────────────┘
```

两种控制头一起用

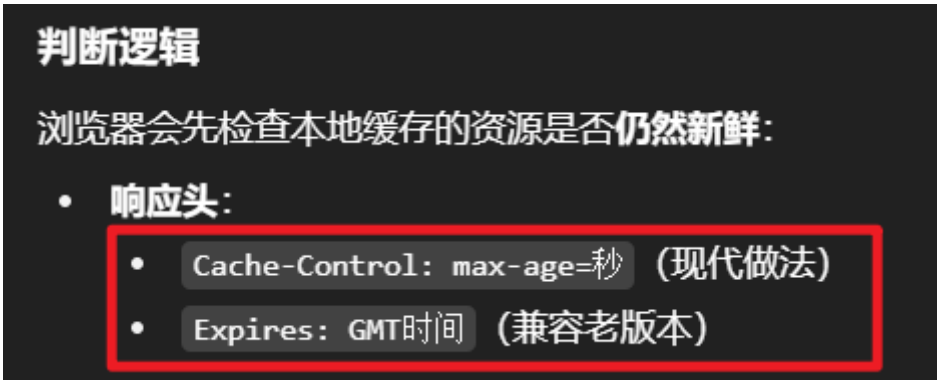
🔑 流程关键点总结

1. **第一次请求**：浏览器缓存资源，并记录 `Last-Modified` 和 `ETag`。
2. **第二次请求**：浏览器带上两个头 `If-Modified-Since` + `If-None-Match` 去问服务器。
3. **服务器判断**：
   - 如果资源没变 → 304 → 浏览器直接用缓存。
   - 如果资源变了 → 200 + 新资源 → 浏览器更新缓存。
4. **ETag 更精确**，Last-Modified 依赖时间戳，可能有误判。
5. **常见做法**：两者一起用，既能保证兼容性，又能提高缓存命中率。

缓存执行流程

浏览器进行缓存判断的流程是这样的

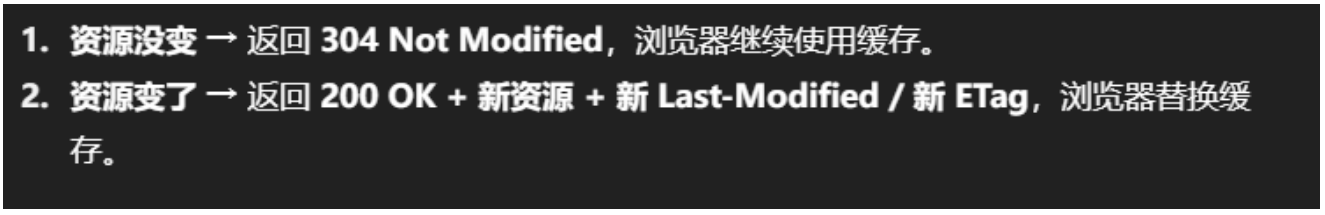
1.**检查强缓存**：首先，浏览器会检查 `Cache-Control` 和 `Expires`。如果文件在有效期内，直接使用本地缓存，不向服务器发请求。



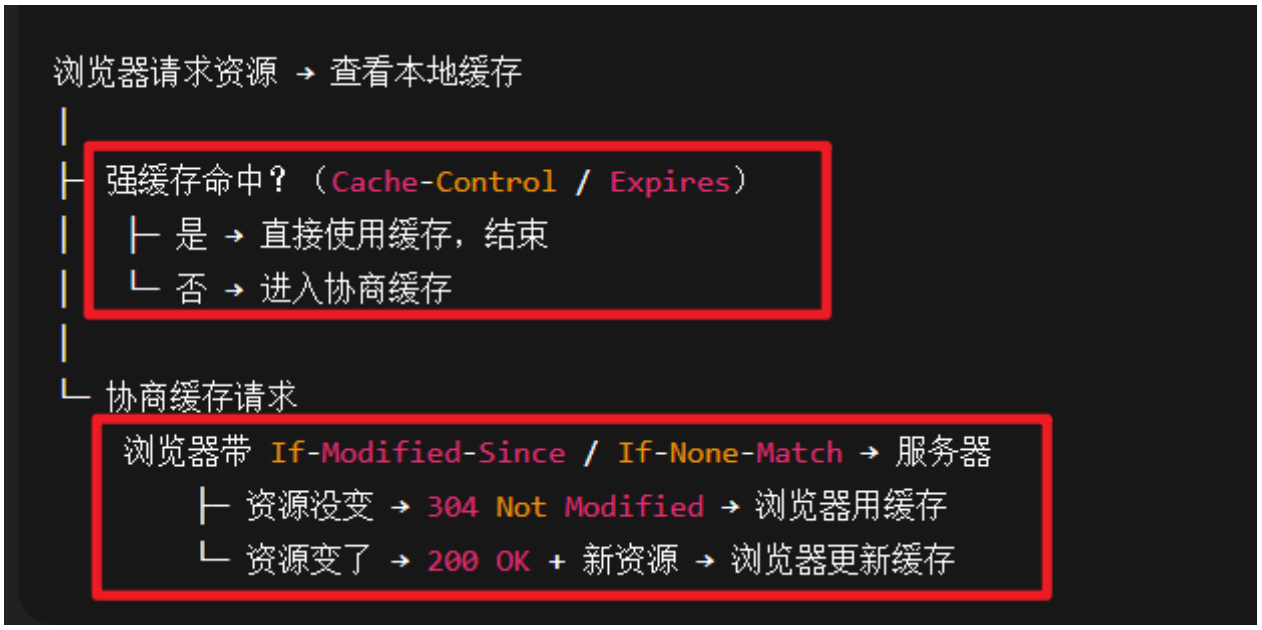
2.**检查协商缓存**：如果强缓存过期了，**浏览器会携带 `If-None-Match` 或 `If-Modified-Since` 向服务器发请求。**

3.**服务器响应**：

- 如果服务器判断文件没更新，返回 **304**，浏览器使用本地缓存。
- 如果服务器判断文件已更新，返回 **200** 和新文件，浏览器下载新文件并更新缓存。



整个流程



小结

1. **强缓存优先**，如果命中，浏览器完全不发请求，性能最好。
2. **协商缓存**是兜底方案，当资源可能过期时才会使用。
3. 强缓存和协商缓存可以结合使用：
   - 静态资源（JS/CSS/图片） → 强缓存 + `immutable`
   - HTML/接口数据 → 协商缓存（ETag / Last-Modified）
4. 浏览器 DevTools Network 面板：
   - `200 (from memory cache)` / `200 (from disk cache)` → 强缓存命中
   - `304 Not Modified` → 协商缓存命中
   - `200 OK` → 缓存失效，回源获取新资源

拓展

CDN缓存

1.什么是 CDN 缓存

- **CDN（Content Delivery Network）** 是一种分布式缓存网络，把你的资源（JS、CSS、图片、视频、HTML 页面等）放到离用户最近的 **边缘节点**。
- **核心目的**：加速访问、减轻源站压力、提高可用性。

> 浏览器缓存只在用户本地生效，CDN 缓存则是在**网络中间层**生效。

2.基本流程

```
用户请求资源 → CDN 节点
|
├─ 节点有缓存且没过期 → 直接返回缓存资源（加速）
|
└─ 节点没有缓存或过期 → 请求源站服务器 → 返回资源 → CDN 缓存 → 返回用户
```

- 用户请求资源时，如果 CDN 节点已经有缓存，用户不需要访问你的源服务器，速度快。
- CDN 节点也会根据 HTTP 响应头来判断缓存策略（Cache-Control / Expires / ETag / Last-Modified）。

3.区别

🌟 3. CDN 缓存与浏览器缓存的区别

特点	浏览器缓存	CDN 缓存
缓存位置	用户本地（内存 / 磁盘）	CDN 边缘节点（网络中间层）
作用	减少浏览器请求网络	减少请求到源服务器，加速全球访问
缓存控制	HTTP 响应头 + Cache API	HTTP 响应头 + CDN 配置策略
生命周期	由浏览器判断	由 CDN 节点判断，可以独立设置

4.缓存控制

CDN 缓存主要靠两类信息：

1. **源站 HTTP 响应头**
   - `Cache-Control` / `Expires` / `ETag` / `Last-Modified`
     CDN 会尊重这些头信息决定缓存时长。
2. **CDN 自身策略**
   - 可以配置强制缓存、忽略源站头部、设置刷新规则。

```
Cache-Control: public, max-age=3600
```

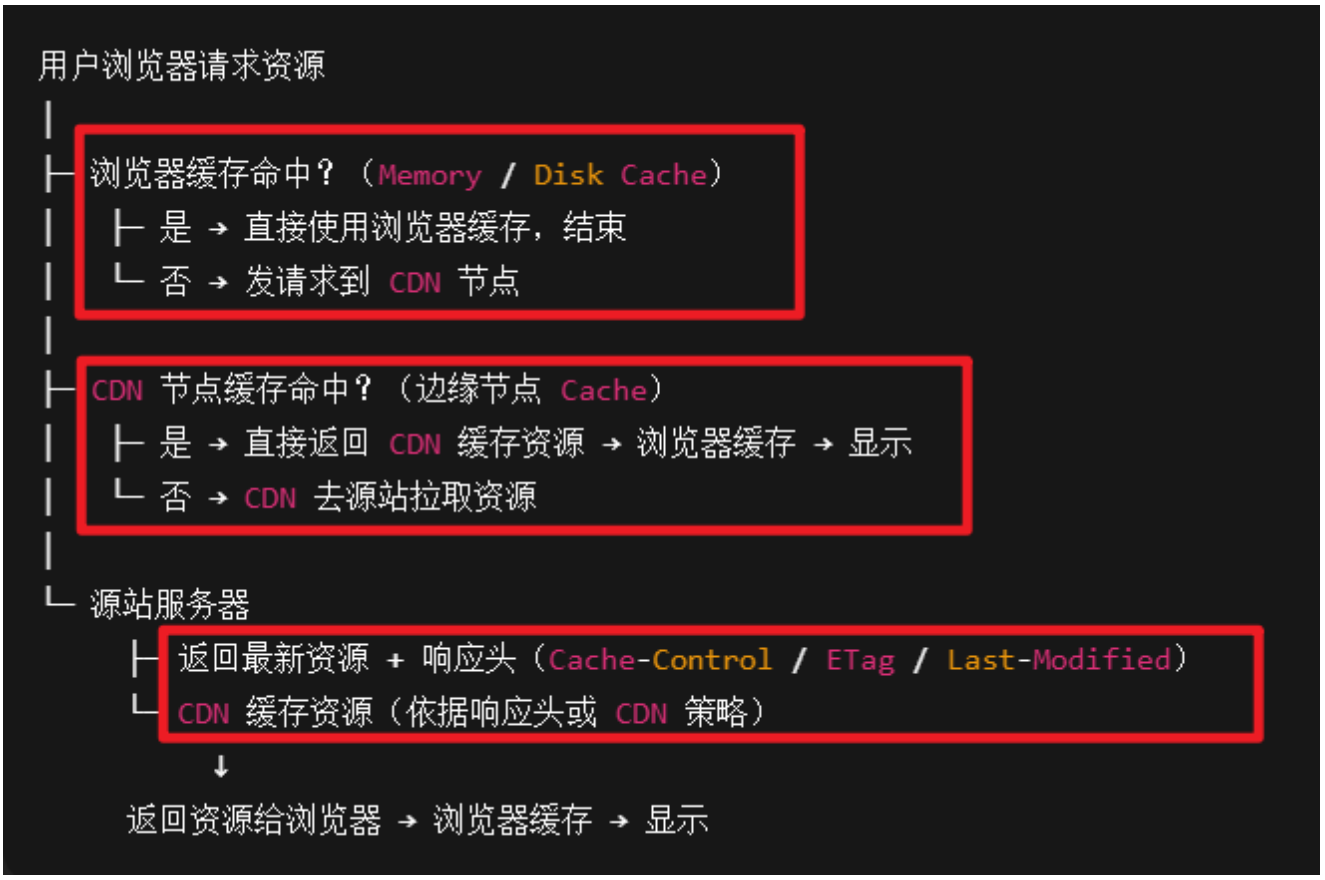
- CDN 节点会缓存资源 1 小时（3600 秒）
- 如果用户再次请求，同样先查 CDN 节点 → 命中则直接返回缓存

5.刷新机制

- **主动刷新**：管理员手动刷新某个资源或整个目录（Purging / Invalidation）
- **被动刷新**：资源过期后，CDN 会去源站拉取最新内容
- **强制刷新**：修改 URL（加 hash / query 参数），浏览器和 CDN 都认为是新资源

前端常用方式：静态资源文件名带 hash（如 `app.abc123.js`），更新资源时文件名变，CDN 和浏览器缓存都自动更新。

6.流程



7.小结

1. **优先级**：
   - 浏览器缓存 > CDN 缓存 > 源站
   - 每一层都可能命中缓存，减少网络请求或源站压力。
2. **缓存控制来源**：
   - **浏览器缓存**：HTTP 响应头（Cache-Control / ETag / Last-Modified）
   - **CDN 缓存**：HTTP 响应头 + CDN 配置策略
3. **刷新策略**：
   - **CDN缓存过期** → 被动拉源站
   - **强制刷新** → 手动 Purge 或 URL 改 hash
4. **协作效果**：
   - 多层缓存协作，可以实现全球访问加速、减少回源次数，提高性能。