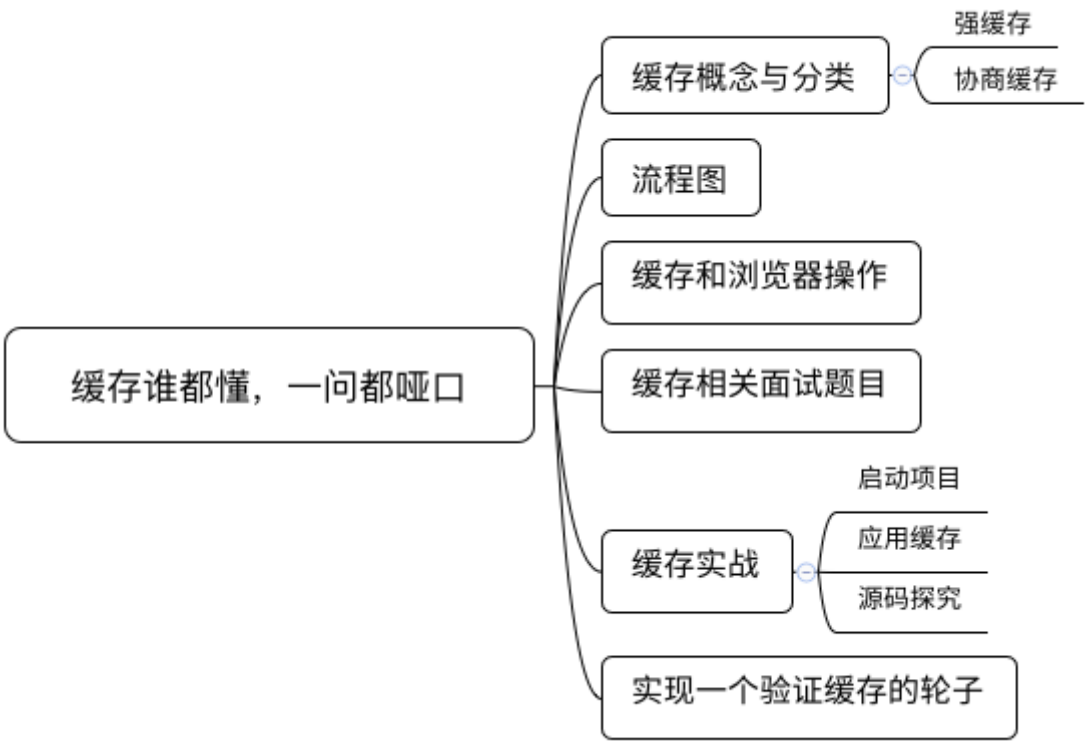
缓存谁都懂，一问都哑口

缓存是网络世界中非常重要的一环，也是解决性能问题最常用的手段之一。说起缓存这个概念，貌似谁都可以说上两句，但又不能完全面面俱到；你可能听说过 etag 或者 if-modified-since 这样的头部，可是并不能梳理好所有这些头部的关系；你可能观察过某个网站或者请求的缓存策略，但是并没有亲自设计并应用个缓存机制；你可能在面试中被问起，在实际开发中踩过坑。

我们将用两节课，彻底梳理缓存知识的方方面面，亲自动手配置尝试，打消那些似懂非懂。主要内容如下：



缓存概念与分类

其实缓存是一个很「大」的概念，尤其 Web 缓存分为很多种。比如：

数据库缓存

（代理）服务器缓存

CDN 缓存

浏览器缓存

甚至一个函数的执行结果都可以进行缓存。而我们要分析的就是 HTTP 缓存，或者浏览器缓存

HTTP 缓存的官方概念：

HTTP 缓存（或 Web 缓存）是用于临时存储（缓存）Web 文档（如 HTML 页面和图像），以减少服务器延迟的一种信息技术。HTTP 缓存系统会保存下通过这套系统的文档的副本；如果满足某些条件，则可以由缓存满足后续请求。HTTP 缓存系统既可以指设备，也可以指计算机程序。

《HTTP 权威指南》一书中，这样介绍到缓存：

在前端开发中，性能一直都是被大家所重视的一点，然而判断一个网站的性能最直观的就是看网页打开的速度。其中提高网页反应速度的一个方式就是使用缓存。一个优秀的缓存策略可以缩短网页请求资源的距离，减少延迟，并且由于缓存文件可以重复利用，还可以减少带宽，降低网络负荷。那么下面我们就来看看服务器端缓存的原理。

目前网络应用中很少有不接入缓存的案例。缓存之所以这么重要，是因为它能带来非常多的好处：

使得网页加载和呈现速度更快

减少了不必要的的数据传输，因而节省网络流量和带宽

在上一步的基础上，服务器的负担因此减少

事实上，前两点非常好理解，合理地使用缓存，能够最大限度地读取和利用本地已有的静态资源，减少了数据传输，加快了网页应用的呈现。对于第三点，可能一两个用户的访问对于减小服务器的负担没有明显效果。但请设想高并发的场景，使用缓存对于减小服务器压力非常有帮助。

对于浏览器缓存的分类，分类方式有很多，按缓存位置分类，我们有：

memory cache

disk cache

Service Worker 等

浏览器的资源缓存分为 from disk cache 和 from memory cache 两类。当首次访问网页时，资源文件被缓存在内存中，同时也会在本地磁盘中保留一份副本。当用户刷新页面，如果缓存的资源没有过期，那么直接从内存中读取并加载。当用户关闭页面后，当前页面缓存在内存中的资源被清空。当用户再一次访问页面时，如果资源文件的缓存没有过期，那么将从本地磁盘进行加载并再次缓存到内存之中。

关于 from disk cache 和 from memory cache 的区别：

When you visit a URL in Chrome, the HTML and the other assets(like images) on the page are stored locally in a memory and a disk cache. Chrome will use the memory cache first because it is much faster, but it will also store the page in a disk cache in case you quit your browser or it crashes, because the disk cache is persistent.

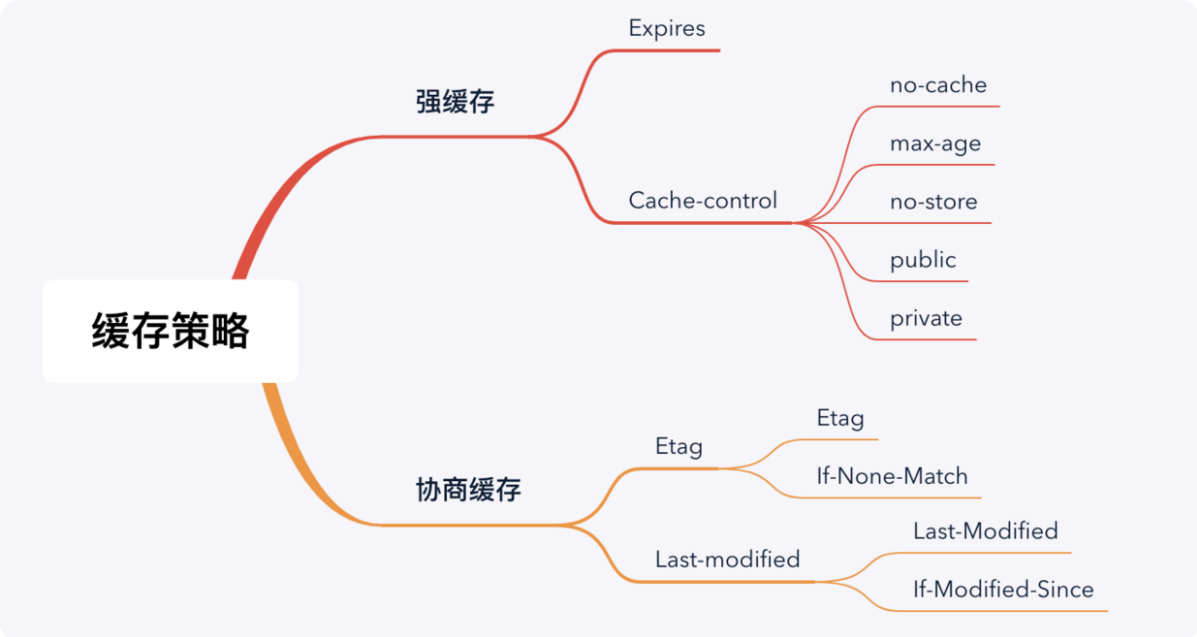
翻译：

当您访问 chrome 中的 URL 时，页面上的 HTML 和其他资产（如图像）将本地存储在内存和磁盘缓存中。Chrome 将首先使用内存缓存，因为它的速度快得多，但它也会将页面存储在磁盘缓存中，以防您退出浏览器或它崩溃，因为磁盘缓存是持久的。

如果按失效策略分类，我们有：

强缓存

协商缓存



缓存策略是理解缓存的最重要一环，我们这节课重点了解一下强缓存和协商缓存。说到底缓存最重要的核心就是解决**什么时候使用缓存，什么时候更新缓存**的问题。

强缓存

强缓存是指客户端在第一次请求后，有效时间内不会再去请求服务器，而是直接使用缓存数据。

那么这个过程，就涉及到一个缓存有效时间的判断。在有效时间判断上，HTTP 1.0 和 HTTP 1.1 是有所不同的。

HTTP 1.0 版本规定响应头字段 Expires，它对应一个未来的时间戳。客户端第一次请求之后，服务端下发 Expires 响应头字段，当客户端再次需要请求时，先会对比当前时间和 Expires 头中设置的时间。如果当前时间早于 Expires 时间，那么直接使用缓存数据；反之，需要再次发送请求，更新数据。

响应头如：

```
Expires:Tue, 13 May 2020 09:33:34 GMT
```

上述 Expires 信息告诉浏览器：在 2020.05.13 号之前,可以直接使用该文本的缓存副本。

Expires 为负数，那么就等同于 no-cache，正数或零同 max-age 的表意是相同的。

但是使用 Expires 响应头存在一些小的瑕疵，比如：

可能会因为服务器和客户端的 GMT 时间不同，出现偏差

如果修改了本地时间，那么客户端端日期可能不准确

写法太复杂，字符串多个空格，少个字母，都会导致非法属性从而设置失效

在 HTTP 1.1 版本中，服务端使用 Cache-control 这个响应头，这个头部更加强大，它具有多个不同值：

private：表示私有缓存，不能被共有缓存代理服务器缓存，不能在用户间共享，可被用户的浏览器缓存。

public：表示共有缓存，可被代理服务器缓存，比如 CDN，允许多用户间共享

max-age：值以秒为单位，表示缓存的内容会在该值后过期

no-cache：需要使用协商缓存，协商缓存的内容我们后面介绍。注意这个字段并不表示不使用缓存

no-store：所有内容都不会被缓存

must-revalidate：告诉浏览器，你这必须再次验证检查信息是否过期, 返回的代号就不是 200 而是 304 了

关于 Cache-control 的取值，还有其他情况比如 s-maxage，proxy-revalidate 等，以及 HTTP 1.0 的 Pragma，由于比较少用或已经过气，我们不再过多介绍。

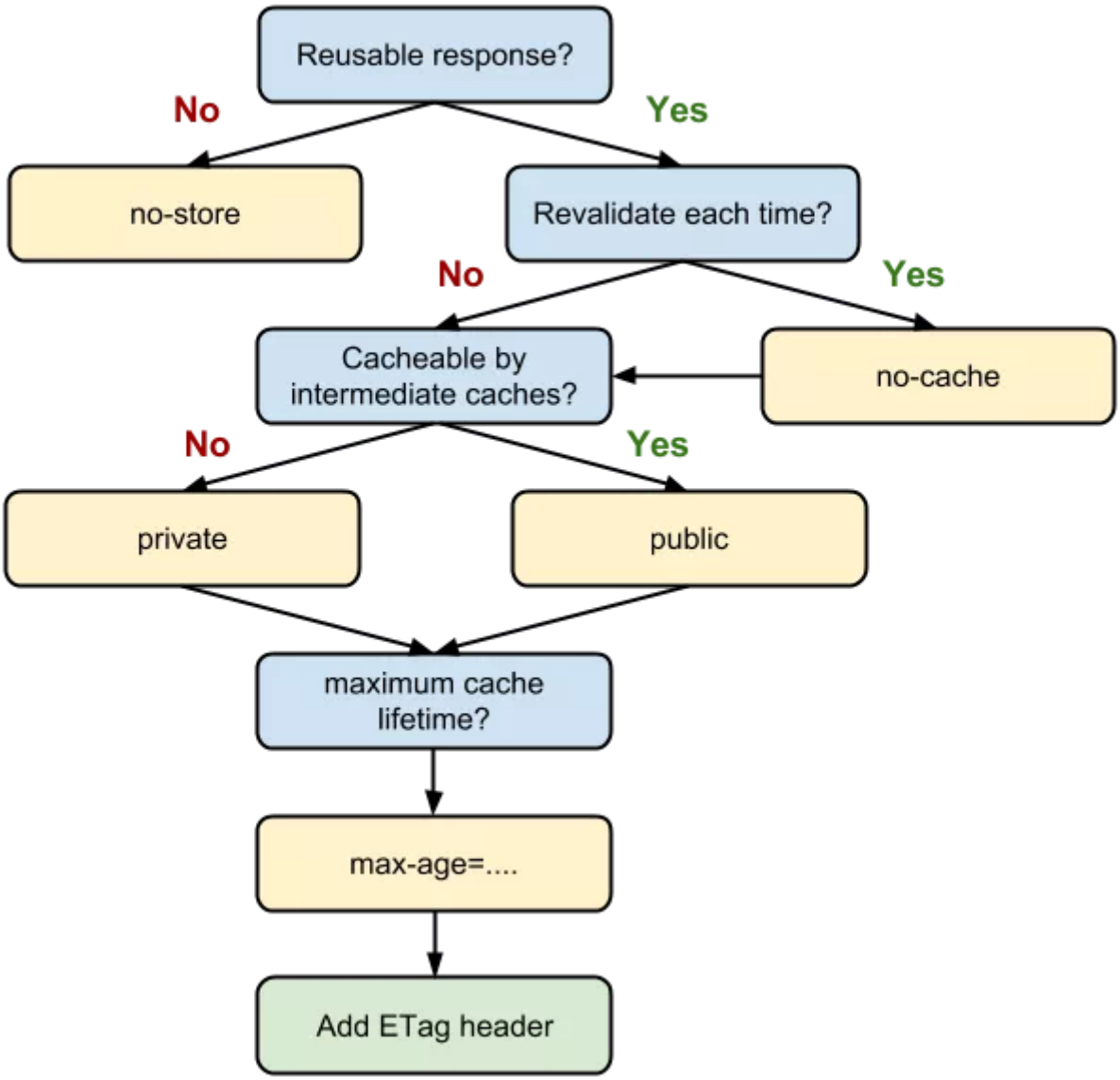
我们看这样的 Cache-control 设置：

```
//Response Headers
Cache-Control:private, max-age=0, must-revalidate
```

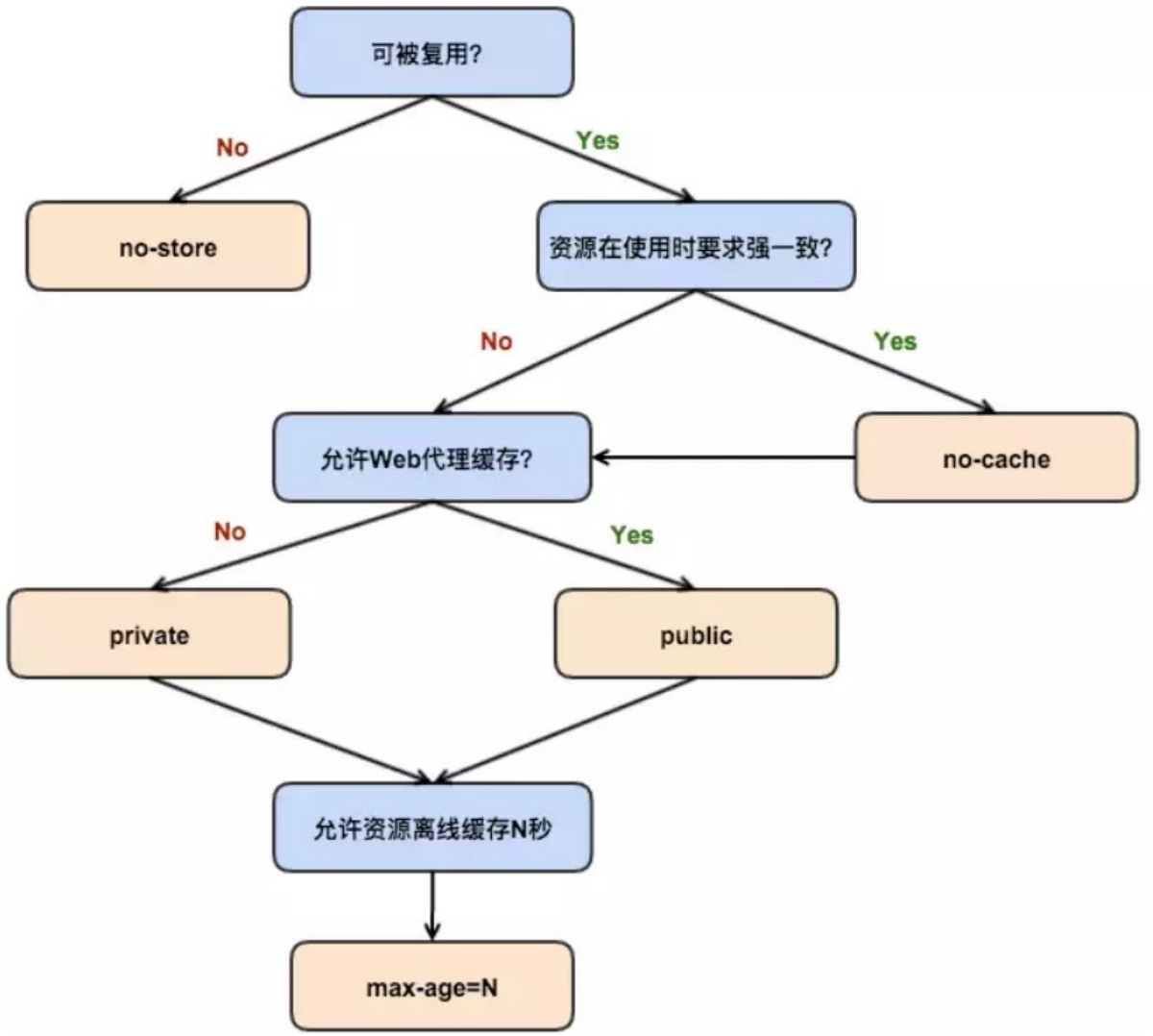
它表示：该资源只能被浏览器缓存，而不能被代理缓存。max-age 标识为 0，说明该缓存资源立即过期，must-revalidate 告诉浏览器，需要验证文件是否过期，接下来可能会使用协商缓存进行判断。

HTTP 规定，如果 Cache-control 的 max-age 和 Expires 同时出现，那么 max-age 的优先级更高，他会默认覆盖掉 expires。

关于 Cache-control 取值总结，我们可以参考 Google developer 的一个图示：


Reusable response?
No
Yes
no-store
Revalidate each time?
No
Yes
Cacheable by intermediate caches?
no-cache
No
Yes
private
public
maximum cache lifetime?
max-age=....
Add ETag header

对于上图的翻译图：



协商缓存

我们进一步思考，强缓存判断的实质上是缓存资源是否超出某个时间或者某个时间段。很多情况是超出了这个时间或时间段，但是资源并没有更新。从优化的角度来说，我们真正应该关心的是服务器端文件是否已经发生了变化。此时我们需要用到协商缓存策略。

那如何做到知晓「服务器端文件是否已经发生了变化」了呢？回到强缓存上，强缓存关于是否使用缓存的决断完全是由浏览器作出的，单一的浏览器是不可能知道「服务器端文件是否已经发生了变化」的。那么协商缓存需要将是否使用缓存的决定权交给服务端，因此协商缓存还是需要一次网络请求的。

协商缓存过程：在浏览器端，当对某个资源的请求没有命中强缓存时，浏览器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。

现在问题就到服务端如何判断资源有没有过期上了。服务端掌握着最新的资源，那么为了做对比，它需要知道客户端的资源信息。根据 HTTP 协议，这个决断是根据【Last-Modified，If-Modified-Since】和【ETag、If-None-Match】这两对 header 来作出的。

我们先来看【Last-Modified，If-Modified-Since】 这一对 header 主导的协商缓存过程：

浏览器第一次请求资源，服务端在返回资源的响应头中加入 Last-Modified 字段，这个字段表示这个资源在服务器上的最近修改时间

```
Last-Modified: Tue, 12 Jan 2019 09:08:53 GMT
```

浏览器收到响应，并记录 Last-Modified 这个响应头的值为 T

当浏览器再次向服务端请求该资源时，请求头加上 If-Modified-Since 的 header，这个 If-Modified-Since 的值正是上一次请求该资源时，后端返回的 Last-Modified 响应头值 T

服务端再次收到请求，根据请求头 If-Modified-Since 的值 T，判断相关资源是否在 T 时间后有变化；如果没有变化则返回 304 Not Modified，且并不返回资源内容，浏览器使用资源缓存值；如果有变化，则正常返回资源内容，且更新 Last-Modified 响应头内容

我们思考这种基于时间的判断方式和 HTTP 1.0 的 Expires 的问题类似，如果客户端的时间不准确，就会导致判断不可靠；同时 Last-Modified 标注的最后修改只能精确到秒级，如果某些文件在 1 秒钟以内，被修改多次的话，它将不能准确标注文件的修改时间；也要考虑到，一些文件也许会周期性的更改，但是他的内容并不改变，仅仅改变的修改时间，这时候使用 Last-Modified 就不是很合适了。为了弥补这种小缺陷，就有了 【ETag、If-None-Match】这一对 header 头来进行协商缓存的判断。

我们来看 【ETag、If-None-Match】这一对 header 主导的协商缓存过程：

浏览器第一次请求资源，服务端在返回资源的响应头中加入 Etag，Etag 能够弥补 Last-Modified 的问题，因为 Etag 的生成过程类似文件 hash 值，Etag

是一个字符串，不同文件内容对应不同的 Etag 值

```
//response Headers
ETag:"751F63A30AB5F98F855D1D90D217B356"
```

浏览器收到响应，记录 Etag 这个响应头的值为 E

浏览器再次跟服务器请求这个资源时，在请求头上加上 If-None-Match，值为 Etag 这个响应头的值 E

服务端再次收到请求，根据请求头 If-None-Match 的值 E，根据资源生成一个新的 ETag，对比 E 和新的 Etag：如果两值相同，则说明资源没有变化，返回 304 Not Modified，同时携带着新的 ETag 响应头；如果两值不同，就正常返回资源内容，这时也更新 ETag 响应头

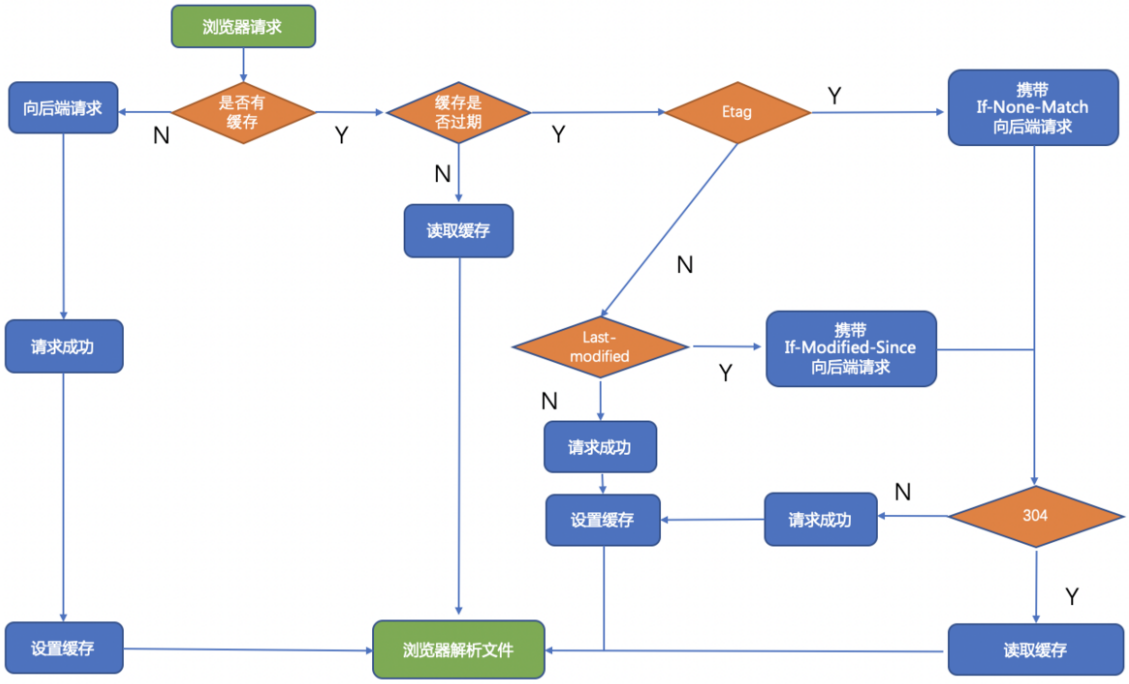
浏览器收到 304 的响应后，就会从缓存中加载资源

这里需要重点说明一下的是 Etag 的生成策略，实际上规范并没有强制说明，这就取决于各大厂商或平台的自主实现方式了：Apache 中，ETag 的值，默认是对文件的索引节（INode），大小（Size）和最后修改时间（MTime）进行混淆后得到的；MDN 使用 wiki 内容的十六进制数字的哈希值。

另外一个需要注意的细节是：Etag 优先级比 Last-Modified 高，如果他们组合出现在请求头当中，我们会优先采用 Etag 策略。同时 Etag 也有自己的问题：相同的资源，在两台服务器产生的 Etag 是不是相同的，所以对于使用服务器集群来处理请求的网站来说， Etag 的匹配概率会大幅降低。所在在这种情况下，使用 Etag 来处理缓存，反而会有更大的开销。

流程图

由上述内容我们开出：为了使缓存策略更加可靠，灵活，HTTP 1.0 版本 和 HTTP 1.1 版本的缓存策略一直是在渐进增强的。这也意味着 HTTP 1.0 版本 和 HTTP 1.1 版本关于缓存的特性可以同时使用，强制缓存和协商缓存也会同时使用。当然他们在混合使用时有优先级的限制，我们通过下面这个流程图来做一个总结：



根据这个流程，我们该如何合理应用缓存呢？一般来说：

优先级上：Cache-Control > Expires > ETag > Last-Modified

强制缓存优先级最高，并且资源的改动在缓存有效期内浏览器都不会发送请求，因此强制缓存的使用适用于大型且不易修改的的资源文件，例如第三方 CSS、JS 文件或图片资源。如果更加灵活的话，我们也可以为文件名加上 hash 进行版本的区分。

协商缓存灵活性高，适用于数据的缓存，根据上述知识的介绍，采用 Etag 标识进行对比灵活度最高，也最为可靠。对于数据的缓存，我们可以重点考虑存入内存中，因为内存加载速最快，并且数据体积小。

总结

这一讲我们梳理了缓存知识体系，实际上缓存并不难理解，只要搞清楚什么时候使用缓存这个关键问题，并以此问题为核心，结合 HTTP 协议关于缓存的发展变革，就很容易掌握理论知识。

下一讲，我们将集中总结常见的缓存面试考察点，并结合实战来巩固知识。

点击查看下一节

缓存谁都懂，一问都哑口