《前端高手进阶》

朱德龙 前中兴软创主任工程师

— 拉勾教育出品 —

第15讲：如何让浏览器更快地加载网络资源？

浏览器加载网络资源的速度

想要加快浏览器加载网络资源的速度，可以通过减少响应内容大小

比如使用 gzip 算法压缩响应体内容和 HTTP/2 的压缩头部功能

另一种更通用也更为重要的技术就是**使用缓存**

浏览器加载网络资源的速度

Name	Status	Type	Size	Time
localhost	200	document	761 B	24 ms
index.css	200	stylesheet	258 B	120 ms
%E4%BA%86%E4%B8%8D%E8%B5%B7%E7%9A%8...	200	png	1.1 MB	444 ms
index.js	200	script	548 B	15 ms

http://localhost:1234/

Name	Status	Type	Size	Time
localhost	304	document	180 B	19 ms
index.css	200	stylesheet	(memory cac...	0 ms
%E4%BA%86%E4%B8%8D%E8%B5%B7%E7%9A%8...	200	png	(memory cac...	0 ms
index.js	200	script	(memory cac...	0 ms

http://localhost:1234/

HTTP 缓存

使用缓存最大的问题往往不在于将资源缓存在什么位置

或者如何读写资源

而在于如何保证缓存与实际资源一致的同时

提高缓存的命中率

也就是说尽可能地让浏览器从缓存中获取资源

但同时又要保证被使用的缓存与服务端最新的资源保持一致

HTTP 支持的缓存策略有两种：**强制缓存**和**协商缓存**

强制缓存

强制缓存是在浏览器加载资源的时候

先直接从缓存中查找请求结果

如果不存在该缓存结果，则直接向服务端发起请求

强制缓存

1.Expires

HTTP/1.0 中可以使用响应头部字段 Expires 来设置缓存时间

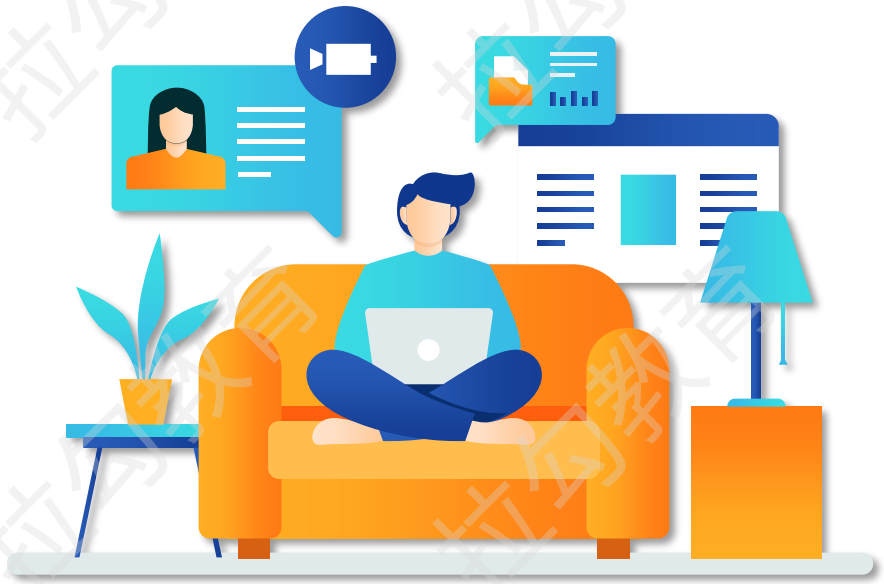
它对应一个未来的时间戳

客户端第一次请求时，服务端会在响应头部添加 Expires 字段

当浏览器再次发送请求时，先会对比当前时间和 Expires 对应的时间

如果当前时间早于 Expires 时间，那么直接使用缓存

反之，需要再次发送请求



强制缓存

1.Expires

```
Expires: Sat, 10 Oct 2020 00:00:00 GMT
```

强制缓存

2.Cache-Control

- **no-cache**，表示使用协商缓存，即每次使用缓存前必须向服务端确认缓存资源是否更新
- **no-store**，禁止浏览器以及所有中间缓存存储响应内容
- **public**，公有缓存，表示可以被代理服务器缓存，可以被多个用户共享
- **private**，私有缓存，不能被代理服务器缓存，不可以被多个用户共享
- **max-age**，以秒为单位的数值，表示缓存的有效时间
- **must-revalidate**，当缓存过期时，需要去服务端校验缓存的有效性

强制缓存

2.Cache-Control

```
cache-control: public, max-age=31536000
```

强制缓存

2.Cache-Control

```
<meta http-equiv="expires" content="Wed, 20 Jun 2021 22:33:00 GMT"
```

协商缓存

协商缓存的更新策略是不再指定缓存的有效时间了

而是浏览器直接发送请求到服务端进行确认缓存是否更新

如果请求响应返回的 HTTP 状态为 304

则表示缓存仍然有效

控制缓存的**难题**就是从浏览器端转移到了服务端

协商缓存

1. Last-Modified 和 If-Modified-Since

服务端要判断缓存有没有过期，只能将双方的资源进行对比

若浏览器直接把资源文件发送给服务端进行比对的话

网络开销太大，而且也会失去缓存的意义

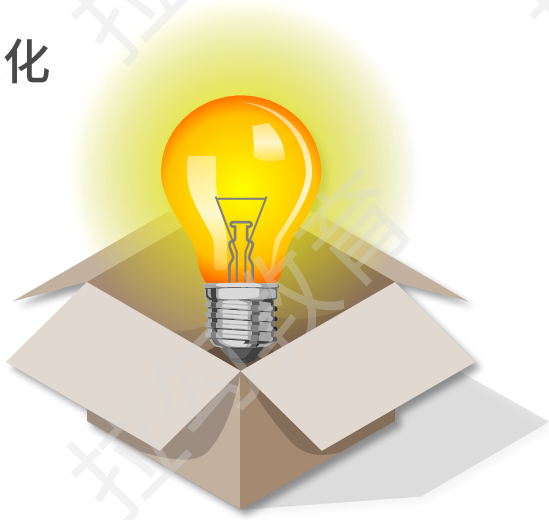
通过响应头部字段 **Last-Modified** 和请求头部字段 **If-Modified-Since**

比对双方资源的修改时间

协商缓存

1. Last-Modified 和 If-Modified-Since

- 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
  该字段表示这个资源在服务端上的最近修改时间
- 当浏览器再次向服务端请求该资源时，请求头部带上之前服务端返回的修改时间
  这个请求头叫 If-Modified-Since
- 服务端再次收到请求，根据请求头 If-Modified-Since 的值，判断相关资源是否有变化
  如果没有则返回 304 Not Modified，并且不返回资源内容，浏览器使用资源缓存值
  否则正常返回资源内容，且更新 Last-Modified 响应头内容



协商缓存

1. Last-Modified 和 If-Modified-Since

精度问题

Last-Modified 的时间精度为秒，如果在 1 秒内发生修改，那么缓存判断可能会失效

准度问题

如果一个文件被修改，然后又被还原，内容并没有发生变化

在这种情况下，浏览器的缓存还可以继续使用

但因为修改时间发生变化，也会重新返回重复的内容

2. ETag 和 If-None-Match

为了解决精度问题和准度问题

HTTP 提供了另一种不依赖于修改时间

而依赖于文件哈希值的精确判断缓存的方式

那就是**响应头部字段 ETag** 和**请求头部字段 If-None-Match**

2. ETag 和 If-None-Match

- 浏览器第一次请求资源，服务端在返响应头中加入 Etag 字段
  Etag 字段值为该资源的哈希值
- 当浏览器再次跟服务端请求这个资源时，在请求头上加上 If-None-Match
  值为之前响应头部字段 ETag 的值
- 服务端再次收到请求，将请求头 If-None-Match 字段的值和响应资源的哈希值进行比对
  如果两个值相同，则说明资源没有变化，返回 304 Not Modified
  否则就正常返回资源内容
  无论是否发生变化，都会将计算出的哈希值放入响应头部的 ETag 字段中

2. ETag 和 If-None-Match

计算成本

生成哈希值相对于读取文件修改时间而言是一个开销比较大的操作

尤其是对于大文件而言

如果要精确计算则需读取完整的文件内容

如果从性能方面考虑，只读取文件部分内容，又容易判断出错

2. ETag 和 If-None-Match

计算误差

HTTP 并没有规定哈希值的计算方法

所以不同服务端可能会采用不同的哈希值计算方式

这样带来的问题是

同一个资源在两台服务端产生的 Etag 可能是不相同的

所以对于使用服务器集群来处理请求的网站来说

使用 Etag 的缓存命中率会有所降低

ServiceWorker

拉勾教育
互联网人实战大学

ServiceWorker 是浏览器在后台独立于网页运行的脚本

也可以这样理解，它是浏览器和服务端之间的代理服务器

ServiceWorker 非常强大

可以实现包括推送通知和后台同步等功能

更多功能还在进一步扩展，但其最主要的功能是**实现离线缓存**

使用限制

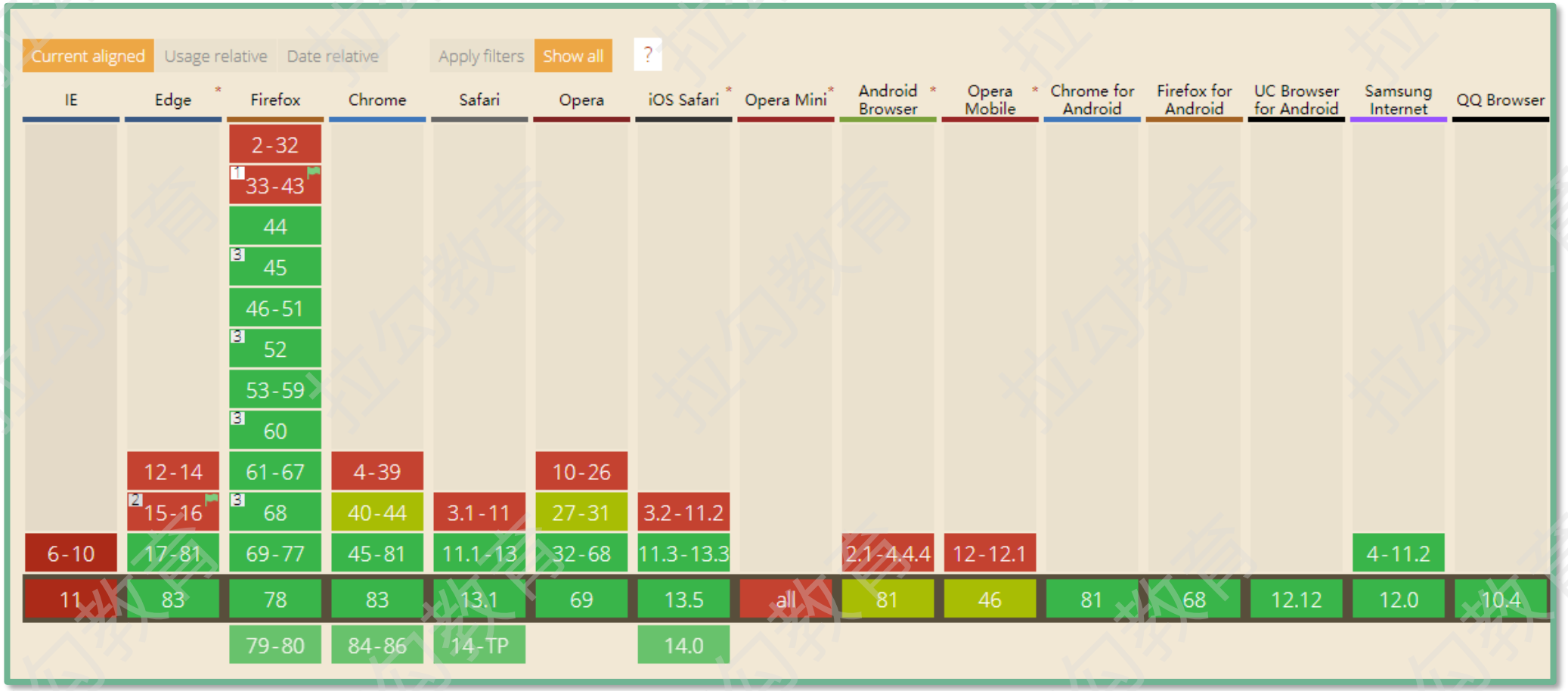
- 在 ServiceWorker 中无法直接访问 DOM

  但可以通过 postMessage 接口发送的消息来与其控制的页面进行通信
- ServiceWorker 只能在本地环境下或 HTTPS 网站中使用
- ServiceWorker 有作用域的限制

  一个 ServiceWorker 脚本只能作用于当前路径及其子路径
- 由于 ServiceWorker 属于实验性功能

  所以兼容性方面会存在一些问题

使用限制


Current aligned
Usage relative
Date relative
Apply filters
Show all
?
IE
Edge *
Firefox
Chrome
Safari
Opera
iOS Safari *
Opera Mini *
Android Browser *
Opera Mobile *
Chrome for Android
Firefox for Android
UC Browser for Android
Samsung Internet
QQ Browser
2-32
1 33-43
44
3 45
46-51
3 52
53-59
3 60
12-14
61-67
4-39
10-26
2 15-16
3 68
40-44
3.1-11
27-31
3.2-11.2
6-10
17-81
69-77
45-81
11.1-13
32-68
11.3-13.3
2.1-4.4.4
12-12.1
4-11.2
11
83
78
83
13.1
69
13.5
all
81
46
81
68
12.12
12.0
10.4
79-80
84-86
14-TP
14.0

使用方法


```
if ('serviceWorker' in window.navigator) {
  window.navigator.serviceWorker
    .register('./sw.js')
    .then(console.log)
    .catch(console.error)
} else {
  console.warn('浏览器不支持 ServiceWorker!')
```

使用方法


```
const CACHE_NAME = 'ws'
let preloadUrls = ['/index.css']


self.addEventListener('install', function (event) {
 event.waitUntil(
  caches.open(CACHE_NAME)
  .then(function (cache) {
   return cache.addAll(preloadUrls);
  })
 );
});
```

使用方法


```
self.addEventListener('fetch', function (event) {
  event.respondWith(
    caches.match(event.request)
    .then(function (response) {
      if (response) {
        return response;
      }
      return caches.open(CACHE_NAME).then(function (cache) {
        const path = event.request.url.replace(self.location.origin, '')
        return cache.add(path)
      })
      .catch(e => console.error(e))
```

使用方法

```
.then(function (response) {
 if (response) {
  return response;
 }
 return caches.open(CACHE_NAME).then(function (cache) {
   const path = event.request.url.replace(self.location.origin, '')
   return cache.add(path)
  })
  .catch(e => console.error(e))
 })
);
})
```

总结

缓存是**解决性能问题**的重要手段

使用缓存除了能让浏览器更快地加载网络资源之外

还可以**节省网络流量和带宽**，以及**减少服务端的负担**

总结

介绍了 HTTP 缓存策略及 ServiceWorker

HTTP 缓存可以分为强制缓存和协商缓存

强制缓存就是在缓存有效期内直接使用浏览器缓存

协商缓存则需要先询问服务端资源是否发生改变

如果未改变再使用浏览器缓存

ServiceWorker 可以用来实现离线缓存

主要实现原理是拦截浏览器请求并返回缓存的资源文件

总结

如果要让浏览器不缓存资源，你有哪些实现方式？

Next：第16讲：《浏览器同源策略与跨域方案详解》

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容

L / A / G / O / U