浏览器缓存机制

一、什么是缓存

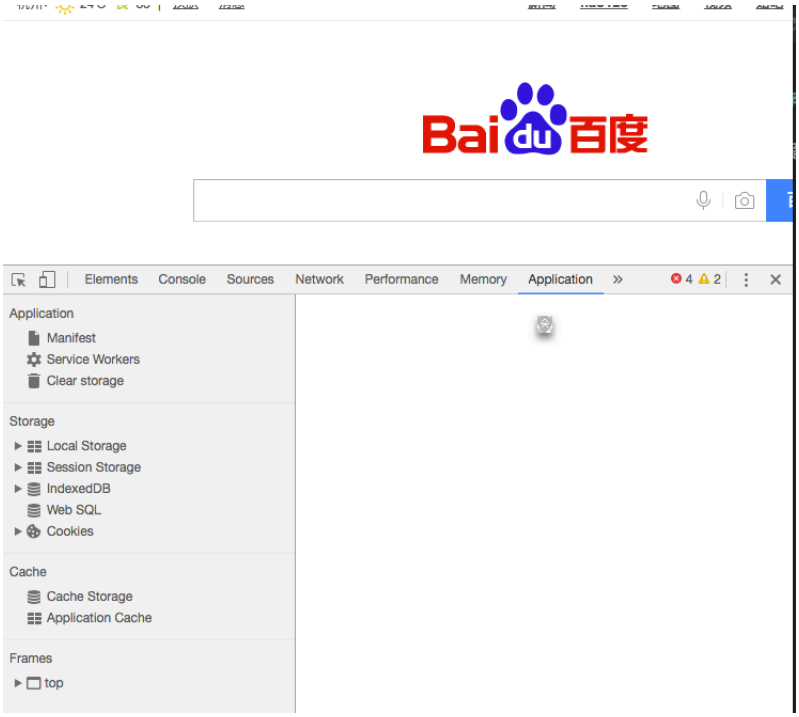
将一些已经请求过的Web资源拷贝成副本，保存在浏览器中。如html页面,图片，js，数据等。
当下次请求同一个url地址的时候，如果网页没有更新，就会直接使用本地缓存的网页，不会再次下载页面。

二、为什么使用缓存（有什么好处？）

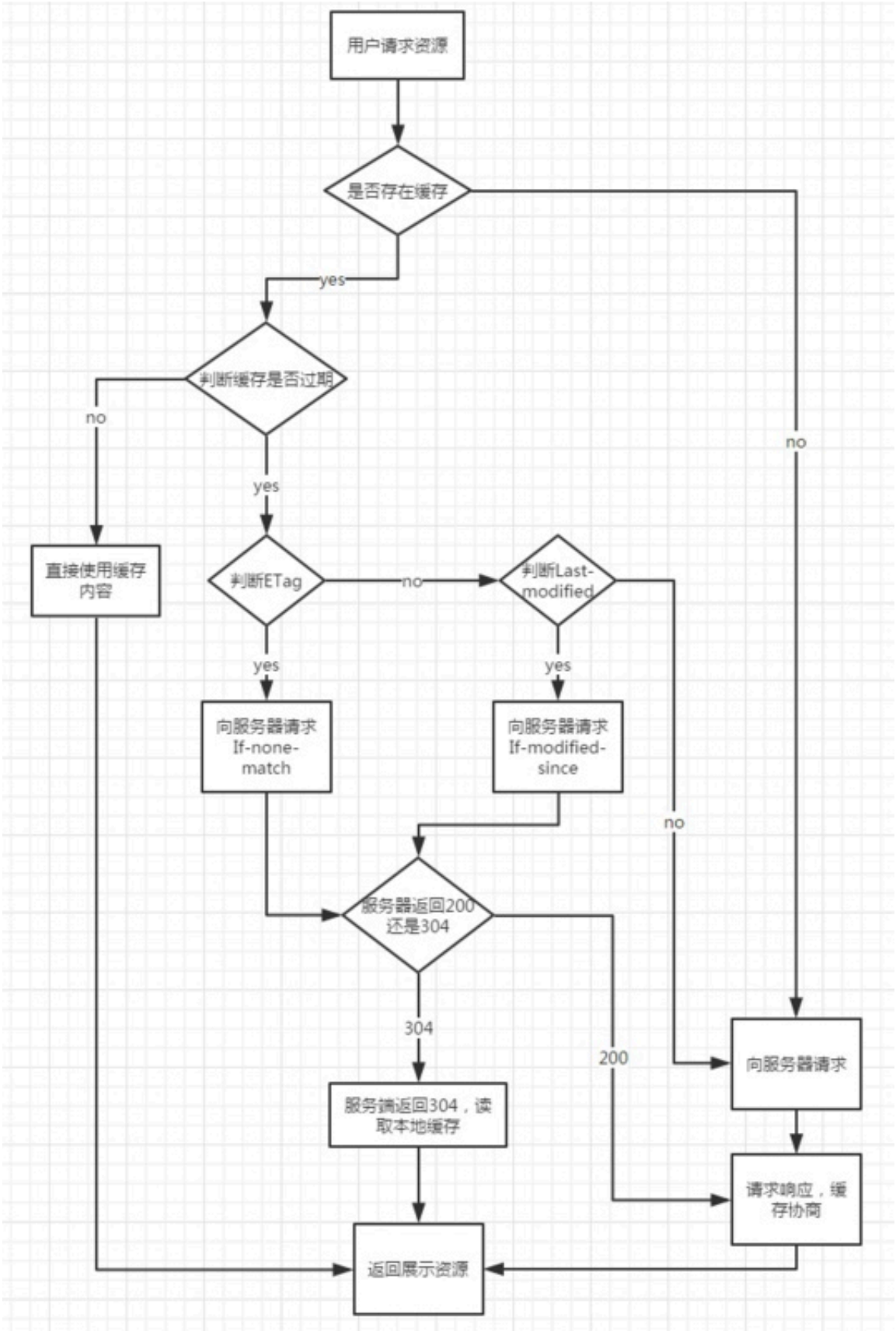
对用户：减少网络延迟，加快打开页面的速度，达到更好的体验；减少网络带宽的消耗，产生较小的网络流量；
对网络运营者：减少对服务器的请求，降低服务器的压力，同样，也能减少带宽，降低运营成本。

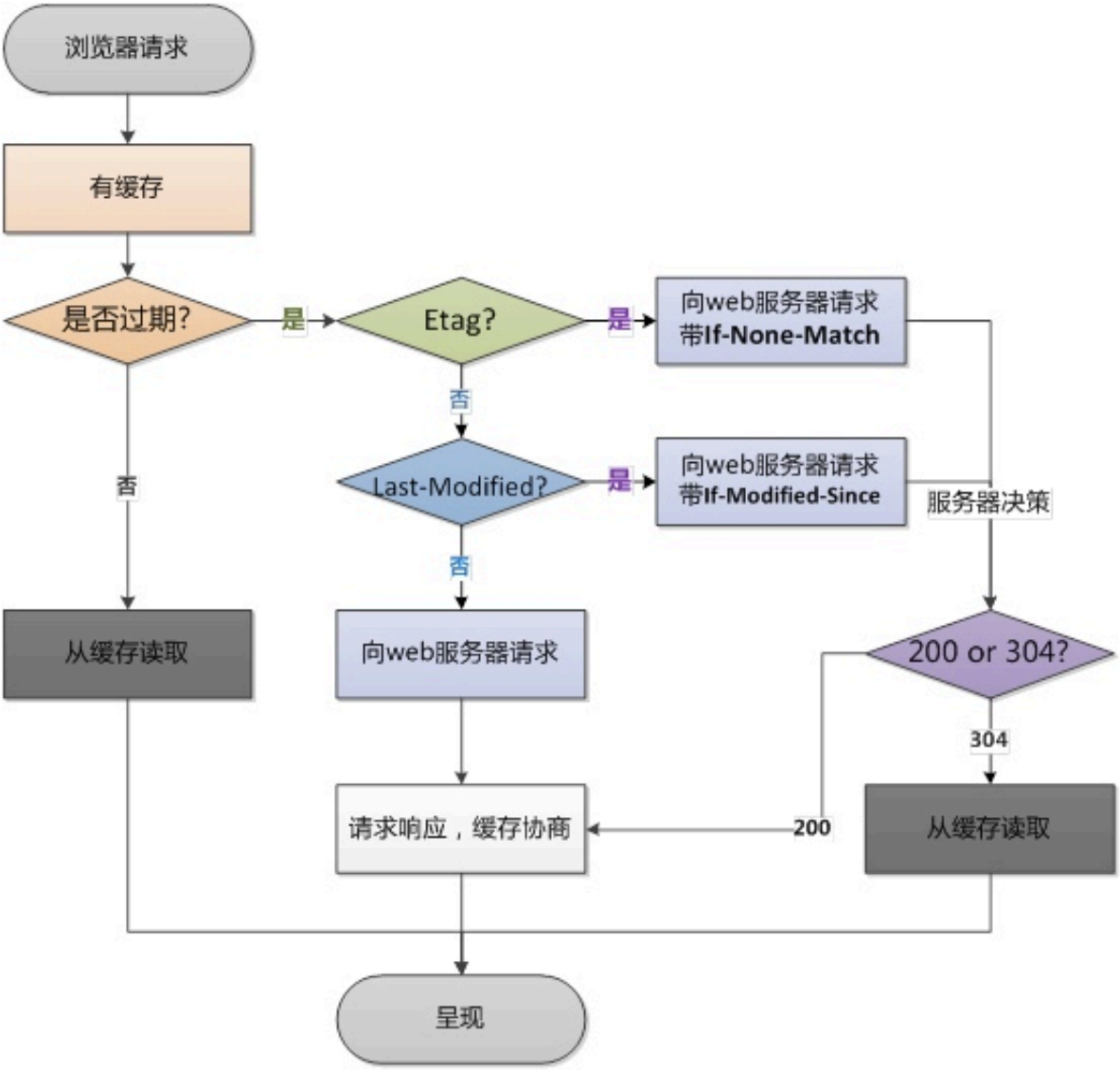
三、浏览器处理缓存的常见情况

浏览器询问服务器缓存是否有效，服务器返回 304 指示浏览器使用缓存。
资源仍然处于有效期时，浏览器会直接使用磁盘缓存（在刷新时稍有不同，见下文）。



chrome控制台下的Frames即展示的是浏览器的http文件级缓存。
http缓存是基于HTTP协议的浏览器文件级缓存机制。


用户请求资源
是否存在缓存
yes
no
判断缓存是否过期
no
yes
直接使用缓存内容
判断ETag
no
判断Last-modified
yes
yes
向服务器请求If-none-match
向服务器请求If-modified-since
no
服务器返回200还是304
304
200
向服务器请求
服务端返回304，读取本地缓存
请求响应，缓存协商
返回展示资源



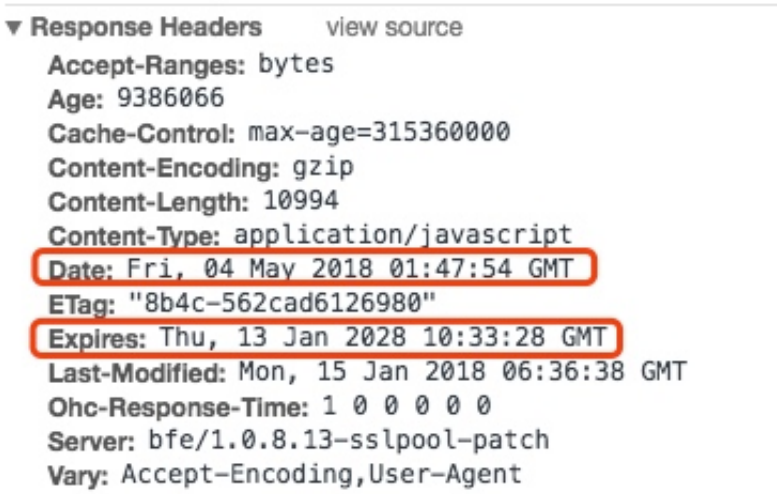
四、浏览器如何控制缓存?

（1）使用HTML Meta 标签

（2）使用缓存有关的HTTP消息报头

主要是四种方式--http协议定义了五个可以用来控制浏览器缓存的HTTP头:
1. Expires
2. LastModified
3. CacheControl
4. ETag(entity tag)(Validation)
5. Pragma

1、Expires+过期时间



Expires是Web服务器响应消息头字段。在响应http请求时告诉浏览器在过期时间前浏览器可以直接从浏览器缓存取数据，而无需再次请求。

```
在2028-1-13 10:33:28之前，能使用缓存文件，请求日期是 2018-05-04 01:47:54
```

不过Expires 是HTTP 1.0的东西，现在默认浏览器均默认使用HTTP 1.1，所以它的作用基本忽略。

2、Cache-control策略（重点关注）

```
浏览器缓存里，Cache-Control是金字塔顶尖的规则，它藐视一切其他设置，只要其他设置与其抵触，一律覆盖之.

不仅如此，它还是一个复合规则，包含多种值，横跨 存储策略，过期策略 两种，同时在请求头和响应头都可设置.
```

作用同Expires。如果同时出现，优先级高于Expires。Cache-Control的选择更多，设置更细致。

值可以是public、private、no-cache、no- store、no-transform、must-revalidate、proxy-revalidate、max-age

各个消息中的指令含义如下：

> “

- *Public 表明响应可以被任何对象（包括：发送请求的客户端，代理服务器，等等）缓存。*
- *Private 表明响应只能被单个用户缓存，不能作为共享缓存（即代理服务器不能缓存它）。资源仅被客户端缓存, 代理服务器不缓存*
- *no-cache 相当于max-age:0,must-revalidate。即资源被缓存, 但是缓存立刻过期, 同时下次访问时强制验证资源有效性；*
- *no-store 这个才是响应不被缓存的意思*
- *max-age 指示客户机可以接收生存期不大于指定时间（以秒为单位）的响应。*
- *min-fresh 缓存的资源至少要保持指定时间的新鲜期*
- *max-stale 指定时间内, 即使缓存过时, 资源依然有效*
- *max-age 缓存资源, 但是在指定时间(单位为秒)后缓存过期*

```
import http from 'http'

let server = http.createServer((req, res) => {
  res.setHeader('Cache-Control', 'public, max-age=86400')
  res.end('harttle.land')
})

server.listen(3333)
```

例子：

当max-age 与 max-stale 和 min-fresh 同时使用时, 它们的设置相互之间独立生效, 最为保守的缓存策略总是有效. 这意味着, 如果max-age=10 days, max-stale=2 days, min-fresh=3 days, 那么:

```
根据max-age的设置，覆盖原缓存周期，缓存资源将在4月15日失效(5+10=15);

根据max-stale的设置，缓存过期后两天依然有效，此时响应将返回110(Response is stale)状态码，缓存资源将在4月14日失效(12+2=14);

根据min-fresh的设置，至少要留有3天的新鲜期，缓存资源将在4月9日失效(12-3=9);
```

由于客户端总是采用最保守的缓存策略, 因此, 4月9日后, 对于该资源的请求将重新向服务器发起验证.

注意：Cache-control和Expires都是强缓存

如果`Expires`, `Cache-Control: max-age`, 或 `Cache-Control:s-maxage`都没有在响应头中出现, 并且也没有其它缓存的设置, 那么浏览器默认会采用一个启发式的算法, 通常会取响应头的`Date_value - Last-Modified_value`值的10%作为缓存时间.

3、ETag/If-None-Match

```
实体标签，服务器资源的唯一标识符.
```

ETag 优先级比 Last-Modified 高.
Etag响应头字段表示资源的版本，浏览器在发送请求时会带 If-None-Match 头字段，来询问服务器该版本是否仍然可用。如果服务器发现该版本仍然是最新的，就可以返回 304 状态码指示继续使用缓存。（所以是请求两次。第一次浏览器发送get请求给服务器端，服务器端返回ETag和状态码200。）

第一次请求时：

```
1.客户端发起HTTP GET 请求一个资源;

2.服务器处理请求，返回资源，包括Http Etag和状态码200
```

后面每次请求时：

1.客户端发起 HTTP GET 请求一个文件，请求中包括一个If-None-Match头，内容就是第一次请求时服务器返回的Etag的值

2.服务器判断接收到的Etag和计算出来的Etag是否匹配。若匹配，返回304状态码，客户端继续使用本地的缓存。若不匹配，返回资源和新的ETag。

用node实现：

```
let server = http.createServer((req, res) => {
  console.log(req.url, req.headers['if-none-match'])
  if (req.headers['if-none-match']) {
    // 检查文件版本
    res.statusCode = 304
    res.end()
  }
  else {
    res.setHeader('Etag', '00000000')
    res.end('harttle.land')
  }
})

server.listen(3333)
```

4、Last-Modified/If-Modified-Since

用于标记请求资源的最后一次修改时间，Last-Modified HTTP 响应头也用来标识资源的有效性。

与 Etag 不同的是**使用修改时间**而不是**实体标签**。对应的请求头字段为If-Modified-Since

Last-Modified/If-Modified-Since要配合Cache-Control使用。

lLast-Modified：标示这个响应资源的最后修改时间。web服务器在响应请求时，告诉浏览器资源的最后修改时间。

lIf-Modified-Since：

当资源过期时（使用Cache-Control标识的max-age），发现资源具有`Last-Modified`声明，则再次向web服务器请求时带上头If-Modified-Since，表示请求时间。

web服务器收到请求后发现有头If-Modified-Since则与被请求资源的最后修改时间进行比对。若最后修改时间较新，说明资源又被改动过，则响应整片资源内容（写在响应消息包体内），HTTP 200；若最后修改时间较旧，说明资源无新修改，则响应HTTP 304 (无需包体，节省浏览)，告知浏览器继续使用所保存的cache。

```
import http from 'http'

let server = http.createServer((req, res) => {
  console.log(req.url, req.headers['if-modified-since'])
  if (req.headers['if-modified-since']) {
    // 检查时间戳
    res.statusCode = 304
    res.end()
  }
  else {
    res.setHeader('Last-Modified', new Date().toString())
    res.end('harttle.land')
  }
})

server.listen(3333)
```

既生Last-Modified何生Etag?

你可能会觉得使用Last-Modified已经足以让浏览器知道本地的缓存副本是否足够新，为什么还需要Etag（实体标识）呢？HTTP1.1中Etag的出现主要是为了解决几个Last-Modified比较难解决的问题：

1. 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
2. 如果某些文件会被定期生成，当有时内容并没有任何变化，但Last-Modified却改变了，导致文件没法使用缓存
3. 有可能存在服务器没有准确获取文件修改时间，或者与代理服务器时间不一致等情形

Etag是服务器自动生成或者由开发者生成的对应资源在服务器端的唯一标识符，能够更加准确的控制缓存。Last-Modified与ETag是可以一起使用的，服务器会优先验证ETag，一致的情况下，才会继续比对Last-Modified，最后才决定是否返回304。

5、Pragma策略

Pragma: no-cache：跟Cache-Control: no-cache相同，Pragma: no-cache兼容http 1.0 ，Cache-Control: no-cache是http 1.1提供的。

因此，Pragma: no-cache可以应用到http 1.0 和http 1.1，
而`Cache-Control: no-cache`只能应用于**http 1.1**.

五、多缓存控制手段

· Last-Modified/ETag与Cache-Control/Expires

配置Last-Modified/ETag的情况下，浏览器再次访问统一URI的资源，还是会发送请求到服务器询问文件是否已经修改，如果没有，服务器会只发送一个304回给浏览器，告诉浏览器直接从自己本地的缓存取数据；如果修改过那就整个数据重新发给浏览器；

Cache-Control/Expires则不同，如果检测到本地的缓存还是有效的时间范围内，浏览器直接使用本地副本，不会发送任何请求。

两者一起使用时，Cache-Control/Expires的优先级要高于Last-Modified/ETag。

即当本地副本根据Cache-Control/Expires发现还在有效期内时，则不会再次发送请求去服务器询问修改时间

（Last-Modified）或实体标识（Etag）了。

一般情况下，使用Cache-Control/Expires会配合Last-Modified/ETag一起使用，因为即使服务器设置缓存时间，当用户点击“刷新”按钮时，浏览器会忽略缓存继续向服务器发送请求，这时Last-Modified/ETag将能够很好利用304，从而减少响应开销。

不能缓存的请求：

当然并不是所有请求都能被缓存，无法被浏览器缓存的请求如下：

1. HTTP信息头中包含Cache-Control:no-cache，pragma:no-cache（HTTP1.0），或Cache-Control:max-age=0等告诉浏览器不用缓存的请求

2. 需要根据Cookie，认证信息等决定输入内容的动态请求是不能被缓存的

3. 经过HTTPS安全加密的请求（有人也经过测试发现，ie其实在头部加入Cache-Control：max-age信息，firefox在头部加入Cache-Control:Public之后，能够对HTTPS的资源进行缓存，参考《HTTPS的七个误解》）

4. POST请求无法被缓存

5. HTTP响应头中不包含Last-Modified/Etag，也不包含Cache-Control/Expires的请求无法被缓存

参考1
参考2

六、协商缓存和强缓存

强缓存：没过期，直接用本地缓存
协商缓存：过期了，重新向服务器要来的资源

浏览器在请求某一资源时，会先获取该资源缓存的header信息，判断是否命中强缓存（cache-control和expires信息）；
若命中直接从缓存中获取资源信息，包括缓存header信息；
本次请求根本就不会与服务器进行通信；

如果没有命中强缓存，浏览器会发送请求到服务器，请求会携带第一次请求返回的有关缓存的header字段信息（Last-Modified/If-Modified-Since和Etag/If-None-Match），由服务器根据请求中的相关header信息来比对结果是否协商缓存命中；
若命中，则服务器返回新的响应header信息更新缓存中的对应header信息，但是并不返回资源内容，它会告知浏览器可以直接从缓存获取；否则返回最新的资源内容

localStorage 和 sessionStorage的区别

Web Storage 定义了两种用于存储数据的对象：sessionStorage 和 localStorage。前者严格用于在一个浏览器会话中存储数据，因为数据在浏览器关闭后会立即删除；后者用于跨会话持久化数据并遵循跨域安全策略。

cookie 和 session

cookie：HTTP Cookie，是绑定在特定的域名下的，每次向特定域名发送请求，都会默认发送cookie。

大小一般是**4096B**。

cookie存在浏览器中，最大只能保存4K数据，不安全

session存在服务器中，不能独立（先读取cookie再读取session），较安全

cookie和session往往是一起合作的，cookie的值是`sessionid`（相当于【钥匙】），据

此去找到对应的session（相当于用【钥匙】开锁），这样可以拿到真正的用户名和密码。

通过给cookie设置有效期，实现浏览器在有效时间内无需二次登录。

在登陆页面，用户登陆了，此时，服务端会生成一个session，session中有对于用户的信息（如用户名、密码等），然后会有一个`sessionid`（相当于是服务端的这个session对应的key），然后服务端在登录页面中写入cookie，值就是:`sessionid=xxx`，然后浏览器本地就有这个cookie了，以后访问同域名下的页面时，自动带上cookie，自动检验，在有效时间内无需二次登陆。