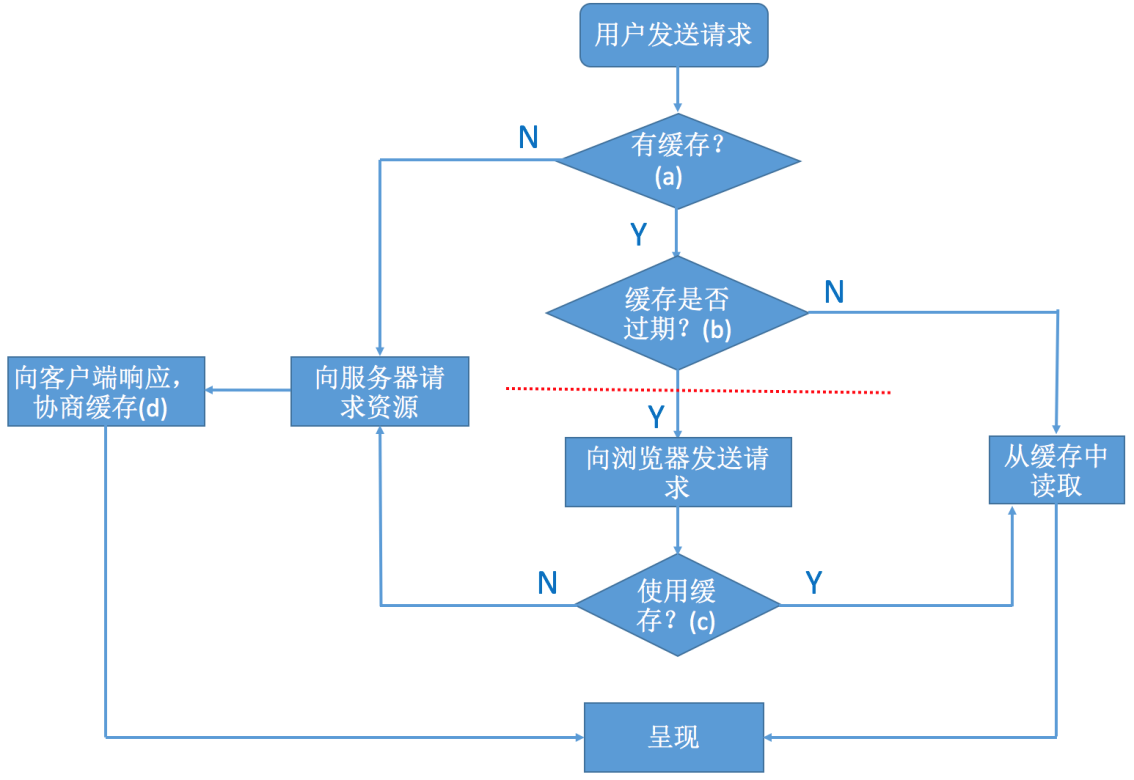
# [彻底理解浏览器缓存机制](http://www.cnblogs.com/shixiaomiao1122/p/7591556.html)

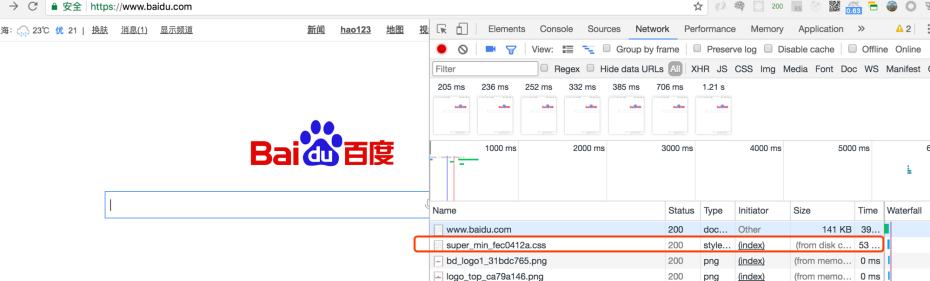
  
先看上图，如果对图中的(a)(b)(c)(d)四个过程的处理方式都很清楚了，那么请不用再看本文了。

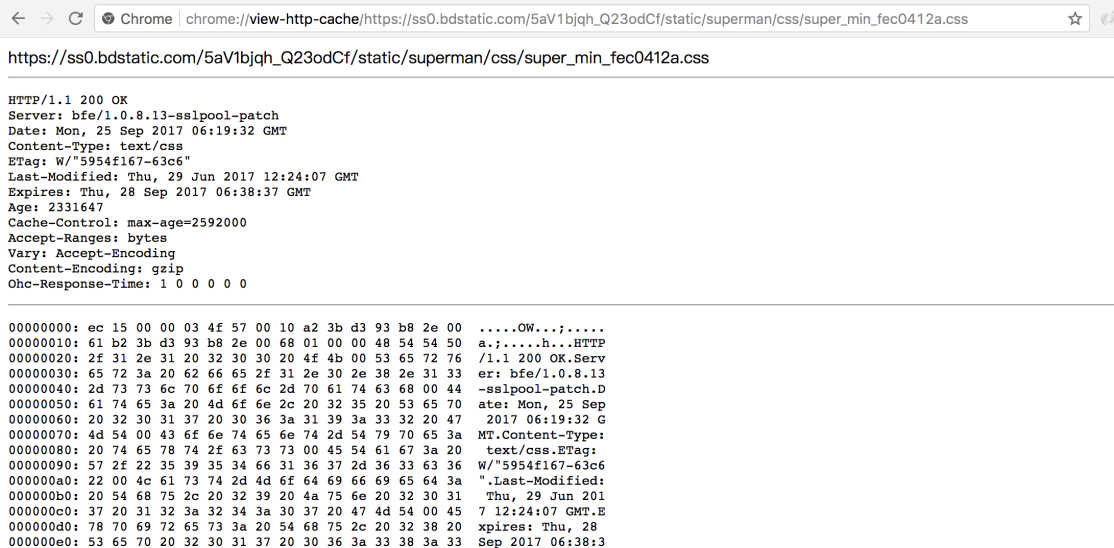
## 两个概念

* 强缓存  
  用户发送的请求，直接从客户端缓存中获取，不发送请求到服务器，不与服务器发生交互行为。
* 协商缓存  
  用户发送的请求，发送到服务器后，由服务器判定是否从缓存中获取资源。
* 两者共同点：客户端获得的数据最后都是从客户端缓存中获得。
* 两者的区别：从名字就可以看出，强缓存不与服务器交互，而协商缓存则需要与服务器交互。

## 四个过程详解

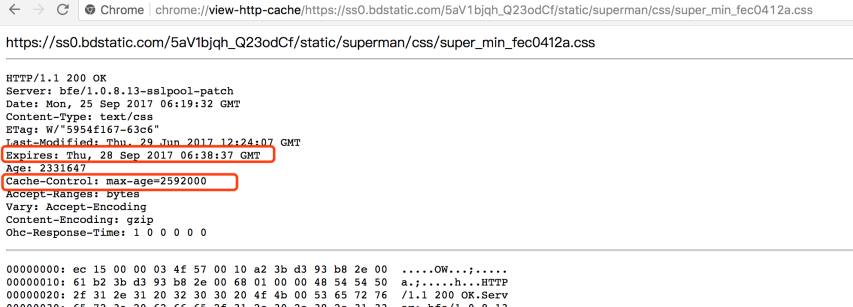
#### （a）浏览器判定是否有缓存

先理解个概念，所谓“客户端缓存”就是指用户设备中本地资源。不同浏览器缓存文件的地址也不尽相同。  
我们以chrome为例来查看下浏览器缓存文件的地址，  
1）首先在chrome中输入：chrome://chrome-urls/，看到一堆列表,里面隐藏了许多浏览器的奥秘，有兴趣的可以自己深扒。  
2）找到 chrome://cache（当然也可以直接输入这个地址）  
为了验证缓存，我们打开百度，打开开发者模式，去掉disable-cache选项  


从上图中可以看到，第一个从缓存中取的文件是： <https://ss0.bdstatic.com/5aV1bjqh_Q23odCf/static/superman/css/super_min_fec0412a.css>  
然后回到chrome://cache 页面，找到它，并点击进去，可以看到：  
  
回到问题，浏览器怎么判定是否有缓存，就可以转化为浏览器去读取本地放缓存的地方（注：不同浏览器不同系统都会不同）是否有该对应的请求啦。

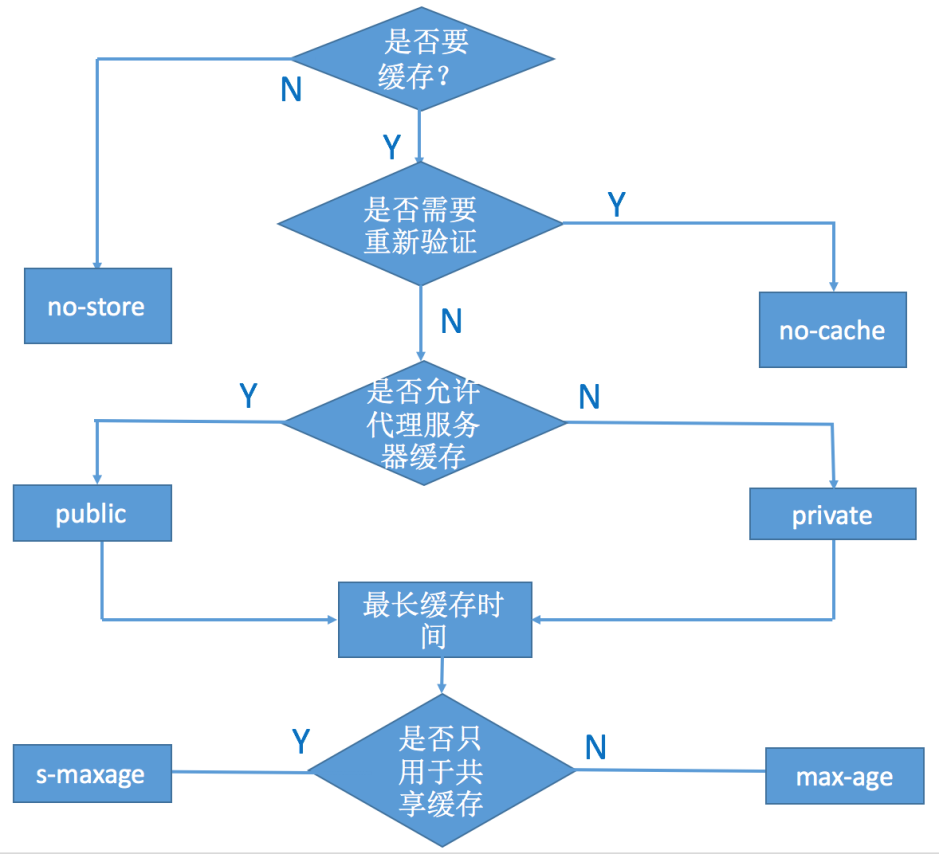
#### 总结来说就是个查找文件是否存在的问题。

#### （b）缓存是否过期

  
我们再以这张图为例，这张图中表明，客户端保留了一个服务器端的response header。  
里面的Date字段表明此次缓存时服务器的时间。  
**里面有两个字段：expires 、Cache-Control**

* expires  
  Http1.0 中的标准，表明过期时间，注意此处的时间都是指的是服务器的时间。  
  可以看到过期时间被设定为了：Thu, 28 Sep 2017 06:38:37 GMT

#### 存在的问题：服务器时间与客户端时间的不一致，就会导致缓存跟期待效果出现偏差。

* Cache-Control  
  Http1.1 中的标准，可以看成是 expires 的补充。使用的是相对时间的概念。  
  简单介绍下Cache-Control的属性设置。  
  1）max-age: 设置缓存的最大的有效时间，单位为秒（s）。max-age会覆盖掉Expires  
  2) s-maxage: 只用于共享缓存，比如CDN缓存（s -> share）。与max-age 的区别是：max-age用于普通缓存，  
  而s-maxage用于代理缓存。如果存在s-maxage,则会覆盖max-age 和 Expires.  
  3) public：响应会被缓存，并且在多用户间共享。默认是public。  
  4) private: 响应只作为私有的缓存，不能在用户间共享。如果要求HTTP认证，响应会自动设置为private。  
  5）no-cache: 指定不缓存响应，表明资源不进行缓存。但是设置了no-cache之后并不代表浏览器不缓存，而是在缓存前要向服务器确认资源是否被更改。因此有的时候只设置no-cache防止缓存还是不够保险，还可以加上private指令，将过期时间设为过去的时间。  
  6）no-store: 绝对禁止缓存。  
  7）must-revalidate: 如果页面过期，则去服务器进行获取。  
  设置cache-control 的规则可以参见下图：  
  

所以判断缓存是否过期步骤是：  
1） **查看是否有cache-control 的max-age / s-maxage ,** 如果有，**则用服务器时间date值 + max-age/s-maxage 的秒数计算出新的过期时间，将当前时间与过期时间进行比较，判断是否过期**2）查看是否有cache-control 的max-age / s-maxage，如果没有，**则用expires 作为过期时间比较**  


#### 总结：（b）过程执行完后，如果判定为未过期，则使用客户端缓存。那么就是属于“强缓存”。

#### （c）跟服务器协商是否使用缓存

  
到这一步的时候，浏览器会向服务器发送请求，同时如果上一次的缓存中有Last-modified 和 Etag 字段，  
浏览器将在request header 中加入If-Modified-Since（对应于Last-modified）， 和If-None-Match（对应于Etag）。

* Last-modified: 表明请求的资源上次的修改时间。
* If-Modified-Since：客户端保留的资源上次的修改时间。
* Etag：资源的内容标识。（不唯一，通常为文件的md5或者一段hash值，只要保证写入和验证时的方法一致即可）
* If-None-Match： 客户端保留的资源内容标识。

##### ⚠️：

1） 分布式系统尽量关闭Etag，因为每台机器生成的Etag都不一样。  
2）分布式系统里多台机器间文件的Last-Modified必须一致，以免负载均衡不同导致对比失败。

通常情况下，如果同时发送 If-None-Match 、If-Modified-Since字段，服务器只要比较etag 的内容即可，当然具体处理方式，看服务器的约定规则。

#### （d）协商缓存

**在这个阶段，服务器一般会将Cache-control、expires 、last-modified、date、etag 等字段在response header 中返回，便于下次缓存。**当然具体的场景，也是看服务器的约定规则设定。

### 缓存的不同来源

⚠️： 这个问题暂时没有找到非常满意的、清楚的回答。

#### from disk cache

从磁盘中获取缓存资源，等待下次访问时不需要重新下载资源，而直接从磁盘中获取。它的直接操作对象为CurlCacheManager。

#### from memory cache

从内存中获取资源，等待下次访问时不需要重新下载资源，而直接从内存中获取。Webkit早已支持memoryCache。  
目前Webkit资源分成两类，一类是主资源，比如HTML页面，或者下载项，一类是派生资源，比如HTML页面中内嵌的图片或者脚本链接，分别对应代码中两个类：　　　　MainResourceLoader和SubresourceLoader。虽然Webkit支持memoryCache，但是也只是针对派生资源，它对应的类为CachedResource，用于保存原始数据（比如CSS，JS等），以及解码过的图片数据。

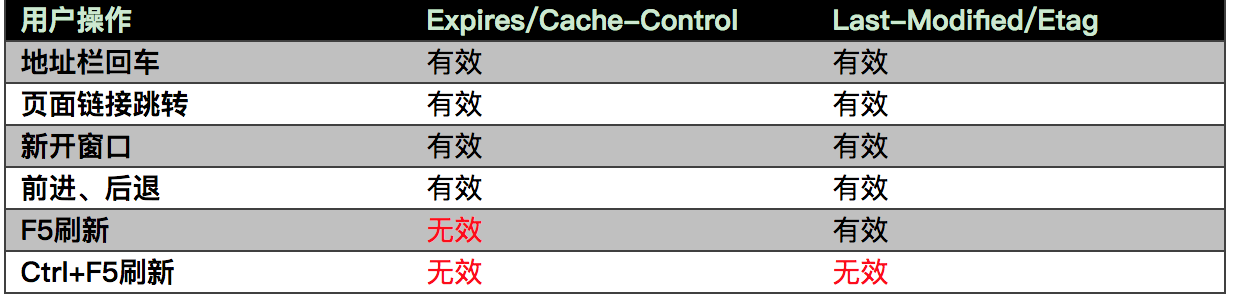
#### 区别

当退出进程时，内存中的数据会被清空，而磁盘的数据不会，所以，当下次再进入该进程时，该进程仍可以从diskCache中获得数据，而memoryCache则不行。

#### 相似

diskCache与memoryCache相似之处就是也只能存储一些派生类资源文件。它的存储形式为一个index.dat文件，记录存储数据的url，然后再分别存储该url的response信息和content内容。Response信息最大作用就是用于判断服务器上该url的content内容是否被修改。

### 用户行为

最后附上一张，用户行为影响浏览器的缓存行为。  


### 总结

自此可以将本文开头的流程图理解清楚。

对于页面中静态资源（html/js/css/img/webfont），理想中的效果：

1. **页面以最快的速度获取到所有必须静态资源，渲染飞快；**
2. **服务器上静态资源未更新时再次访问不请求服务器；**
3. **服务器上静态资源更新时请求服务器最新资源，加载又飞快。**

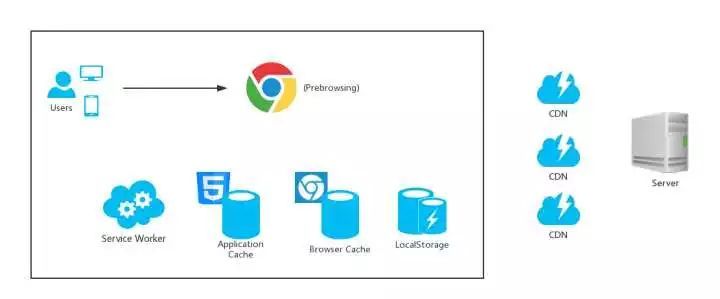
总结下来也就是2个指标：

* **静态资源加载速度**
* **页面渲染速度**

静态资源加载速度引出了我们今天的主题，因为最直接的方式就是将**静态资源进行缓存。**页面渲染速度建立在资源加载速度之上，但不同资源类型的加载顺序和时机也会对其产生影响，所以也留给了我们更多的优化空间。

当然除了速度，缓存还有另外2大功效，**减少用户请求的带宽和减少服务器压力。**

先用一张图来概括下本文中将会涉及到的内容。



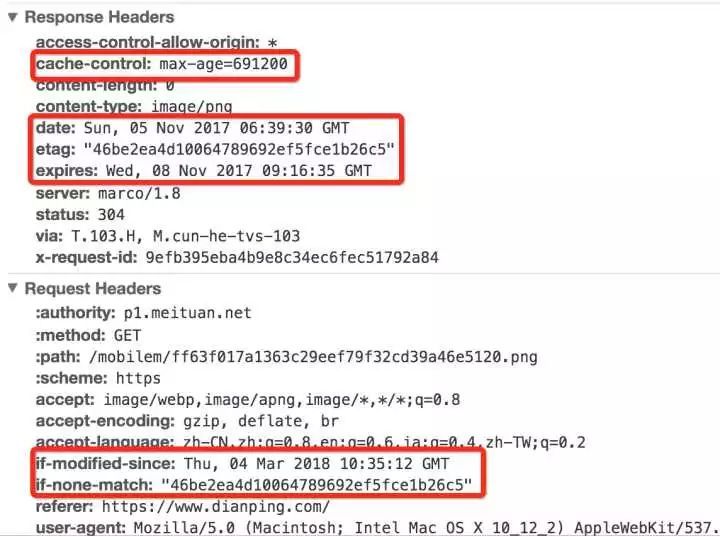
常见缓存类型

1、浏览器缓存

对于前端而言，这可能是我们最容易忽略的缓存类型，原因在于大部分设置都在服务器运维层面上进行，不属于前端开发的维护范围。但**静态资源的内容更新时机其实前端是最清楚的**，如果能在理解浏览器缓存策略的基础上合理配置效果最佳。

浏览器缓存策略一般通过资源的Response Header来定义，html文件在很早之前的规范里也可以通过Meta标签的http-equiv来定义。

一个Response Header示例：



可在w3的官方文档中查看所有HTTP Response Header字段的定义，跟缓存相关的主要有上图中被圈出来的几个：

* Cache-Control：
  + public：响应被缓存，并且在多用户间共享。
  + private：默认值，响应只能够作为私有的缓存(e.g., 在一个浏览器中)，不能再用户间共享；
  + no-cache：响应不会被缓存,而是实时向服务器端请求资源。
  + **max-age：数值，单位是秒，从请求时间开始到过期时间之间的秒数。基于请求时间（Date字段）的相对时间间隔，而不是绝对过期时间；**

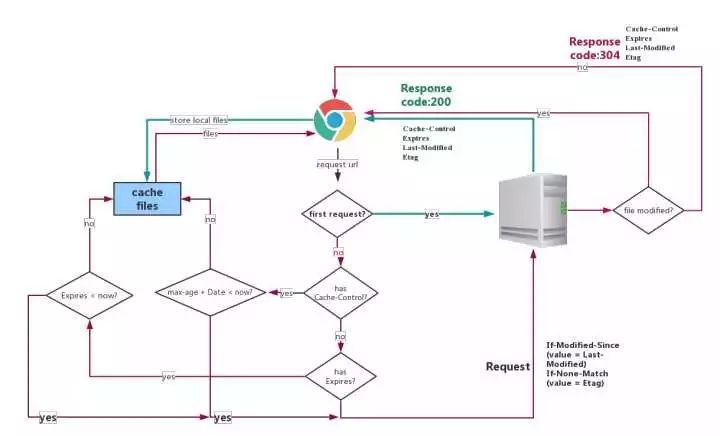
**注：HTTP/1.0 没有实现 Cache-Control，所以为了兼容HTTP/1.0出现了Pragma字段。**

* Pragma: 只有一个用法Pragma: no-cache，它和Cache-Control:no-cache作用一模一样。（Cache-Control: no-cache是http 1.1才提供的， 因此Pragma: no-cache可以使no-cache应用到http 1.0 和http 1.1。）
  + **Expires：指定了在浏览器上缓冲存储的页距过期还有多少时间，等同Cache-control中的max-age的效果，如果同时存在，则被Cache-Control的max-age覆盖。若把其值设置为0，则表示页面立即过期。并且若此属性在页面当中被设置了多次，则取其最小值。**

注：这个规则允许源服务器，对于一个给定响应，向 HTTP/1.1（或之后）缓存比 HTTP/1.0 提供一个更长的过期时间。

* Date：生成消息的具体时间和日期；
* Last-Modified/If-Modified-Since：本地文件在服务器上的最后一次修改时间。缓存过期时把浏览器端缓存页面的最后修改时间发送到服务器去，服务器会把这个时间与服务器上实际文件的最后修改时间进行对比，如果时间一致，那么返回304，客户端就直接使用本地缓存文件。
* Etag/If-None-Match：(EntityTags)是URL的tag，用来标示URL对象是否改变，一般为资源实体的哈希值。和Last-Modified类似，如果服务器验证资源的ETag没有改变（该资源没有更新），将返回一个304状态告诉客户端使用本地缓存文件。Etag的优先级高于Last-Modified，Etag主要为了解决 Last-Modified无法解决的一些问题。
  + 文件也许会周期性的更改，但是他的内容并不改变，不希望客户端重新get；
  + If-Modified-Since能检查到的粒度是s级；
  + 某些服务器不能精确的得到文件的最后修改时间。

缓存策略执行过程



**本地缓存过期后，**浏览器会向服务器发送请求，request中会携带以下两个字段：

* If-Modified-Since：值为之前response中Last-Modified；
* If-None-Match：值为之前response中Etag（如果存在的话）；

其中在图右侧的**“file modified?”**判断中，服务器会读取请求头这两个值，**判断出客户端缓存的资源是否最新，**如果**是的话服务器就会返回HTTP/304 Not Modified响应头，但没有响应体。客户端收到304响应后,就会从缓存中读取对应的资源；否则返回HTTP/200和响应体。**