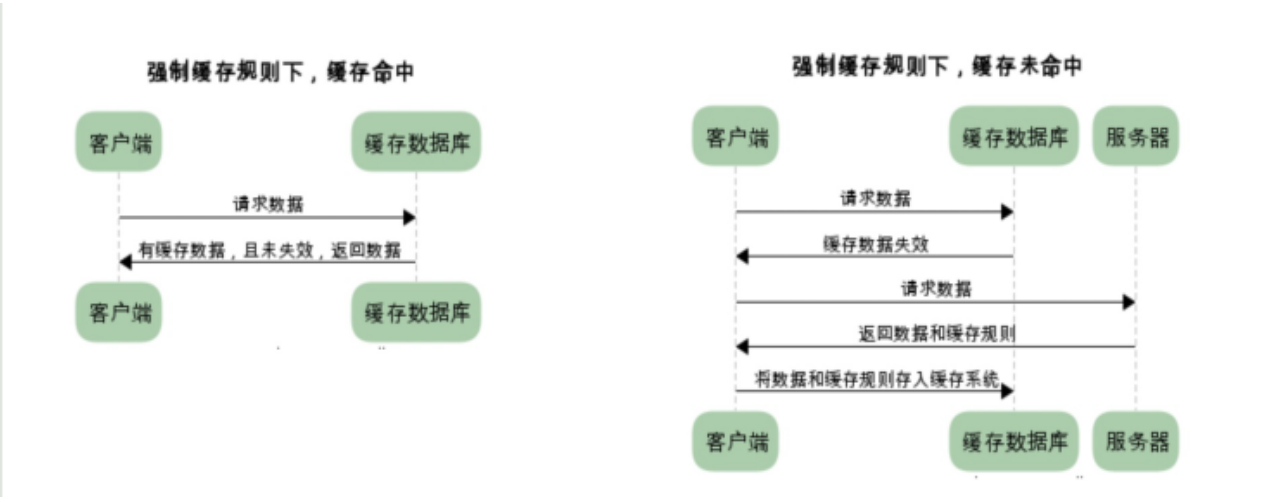


Http相关学习总结

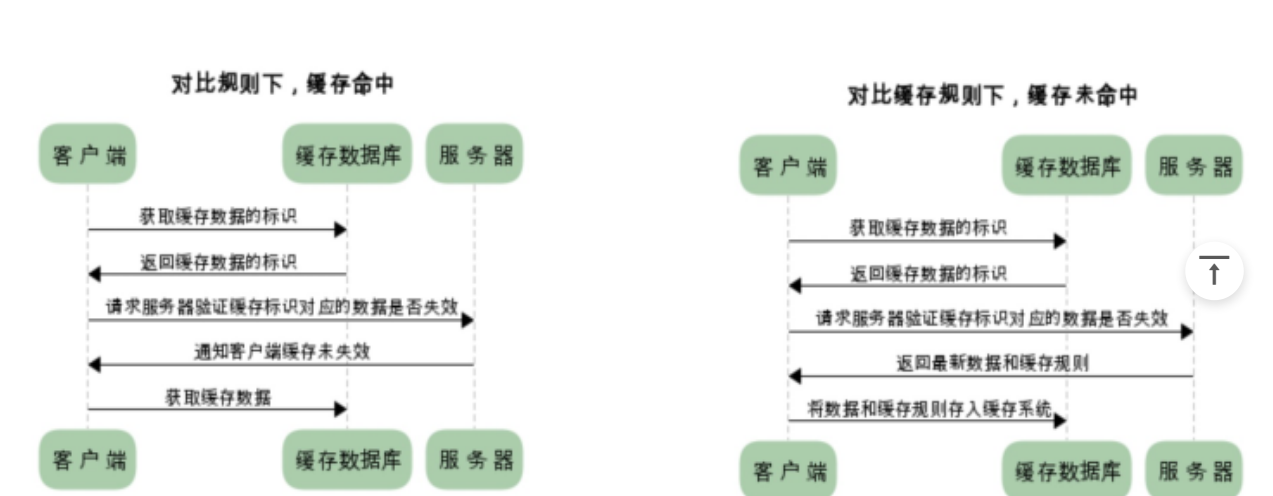
缓存

与缓存相关的规则信息，均包含在HTTP报文的首部（header）中。
HTTP报文就是浏览器和服务器间通信时发送及响应的数据块。
HTTP缓存有多种规则，根据是否需要重新向服务器发起请求来分类，分为两大类(强制缓存，对比缓存)

存在缓存数据，强制缓存



存在缓存数据，对比缓存



基于**对比缓存**的流程下，不管是否使用缓存，都需要向服务器发送请求。
两类缓存规则可以同时存在，**强制缓存**优先级高于**对比缓存**，即当执行**强制缓存**的规则时，如果缓存生效，直接使用缓存，不再执行**对比缓存**规则。

对比缓存生效时，状态码为304，并且报文大小和请求时间大大减少。原因是，服务端在进行标识比较后，只返回header部分，通过状态码通知客户端使用缓存，不再需要将报文主体部分返回给客户端。

缓存失效规则

在没有缓存数据的时候，浏览器向服务器请求数据时，服务器会将数据和缓存规则一并返回，**缓存规则信息包含在响应header中**。

对于强制缓存来说，响应header中会有两个字段来标明失效规则（**Expires/Cache-Control**）。

Expires

Expires的值为服务端返回的到期时间，即下一次请求时，请求时间小于服务端返回的到期时间，直接使用缓存数据。（它是HTTP 1.0的东西，其作用基本忽略，另一个问题是，到期时间是由服务端生成的，但是**客户端时间可能跟服务端时间有误差**，这就会导致缓存命中的误差。所以HTTP 1.1 的版本，使用**Cache-Control**替代。）

Cache-Control

Cache-Control 是最重要的规则。常见的取值有private、public、no-cache、max-age、no-store，默认为private。

| | |
|--------------|--|
| private: | 客户端可以缓存 |
| public: | 客户端和代理服务器都可缓存（前端的同学，可以认为public和private是一样的） |
| max-age=xxx: | 缓存的内容将在 xxx 秒后失效 |
| no-cache: | 需要使用 对比缓存 来验证缓存数据（后面介绍） |
| no-store: | 所有内容都不会缓存， 强制缓存 ， 对比缓存 都不会触发（对于前端开发来说，缓存越多越好，so...基本上和它说886） |

缓存标识



Last-Modified / If-Modified-Since

Last-Modified:

服务器在响应请求时，告诉浏览器资源的最后修改时间。

If-Modified-Since:

再次请求服务器时，通过此字段通知服务器上次请求时，服务器返回的资源最后修改时间。服务器收到请求后发现头If-Modified-Since 则与被请求资源的最后修改时间进行比对。若资源的最后修改时间大于If-Modified-Since，说明资源又被改动过，则响应整片资源内容，返回状态码200；若资源的最后修改时间小于或等于If-Modified-Since，说明资源无新修改，则响应HTTP 304，告知浏览器继续使用所保存的cache。

Etag / If-None-Match

Etag:

服务器响应请求时，告诉浏览器当前资源在服务器的唯一标识（生成规则由服务器决定）。

If-None-Match:


再次请求服务器时，通过此字段通知服务器客户端缓存数据的唯一标识。服务器收到请求后发现头If-None-Match 则与被请求资源的唯一标识进行比对，不同，说明资源又被改动过，则响应整片资源内容，返回状态码200；相同，说明资源无新修改，则响应HTTP 304，告知浏览器继续使用所保存的cache。

总结

对于强制缓存，服务器通知浏览器一个缓存时间，在缓存时间内，下次请求，直接用缓存，不在时间内，执行比较缓存策略。

对于比较缓存，将缓存信息中的Etag和Last-Modified通过请求发送给服务器，由服务器校验，返回304状态码时，浏览器直接使用缓存。

url从输入到页面渲染完成的过程

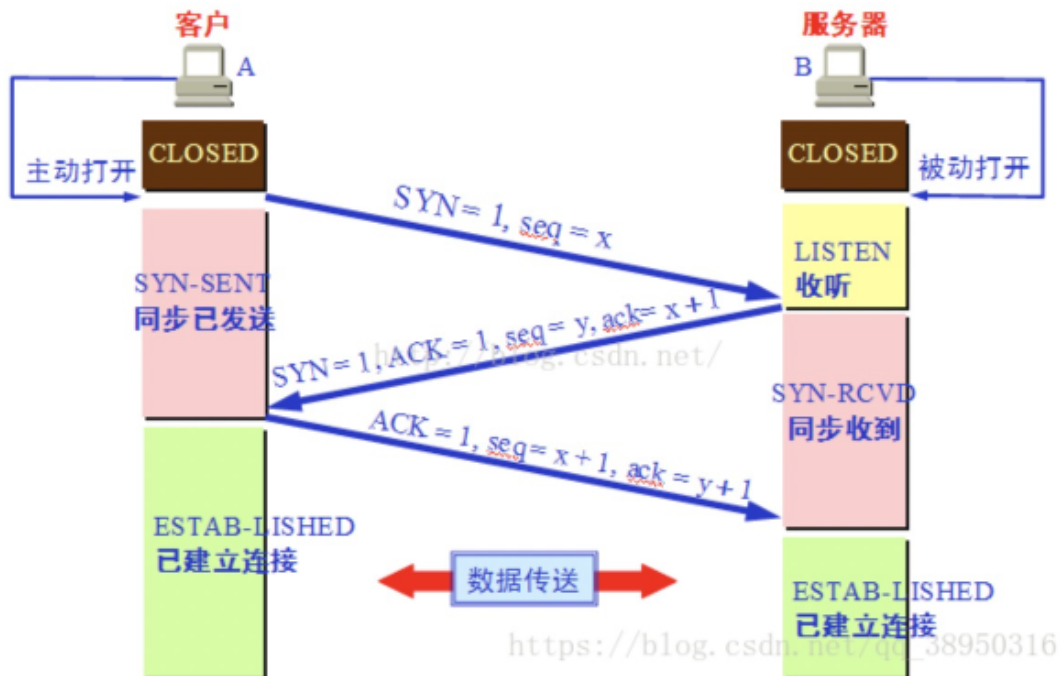
- 用户输入URL（判断搜索还是URL，搜索则使用浏览器默认搜索引擎合成新的URL）
- 浏览器解析URL解析出主机名
- 浏览器通过主机名换成服务器IP地址（浏览器先查找本地DNS缓存列表，若无，再像浏览器默认DNS服务器发送查询请求，并缓存
- 浏览器解析端口号
- 建立一条与目标浏览器的TCP链接（三次握手）
- 浏览器发送HTTP请求
- 服务器返回HTTP响应
- 浏览器接收响应结果关闭连接（四次挥手）若服务器建立长连接，添加keep-alive响应头字段来保p可复用
- 浏览器解析渲染页面（webkit为例）
 - a. 解析HTML构建DOM树
 - b. 解析CSS渲染DOM树，渲染完成之后，开始布局绘制到屏幕上

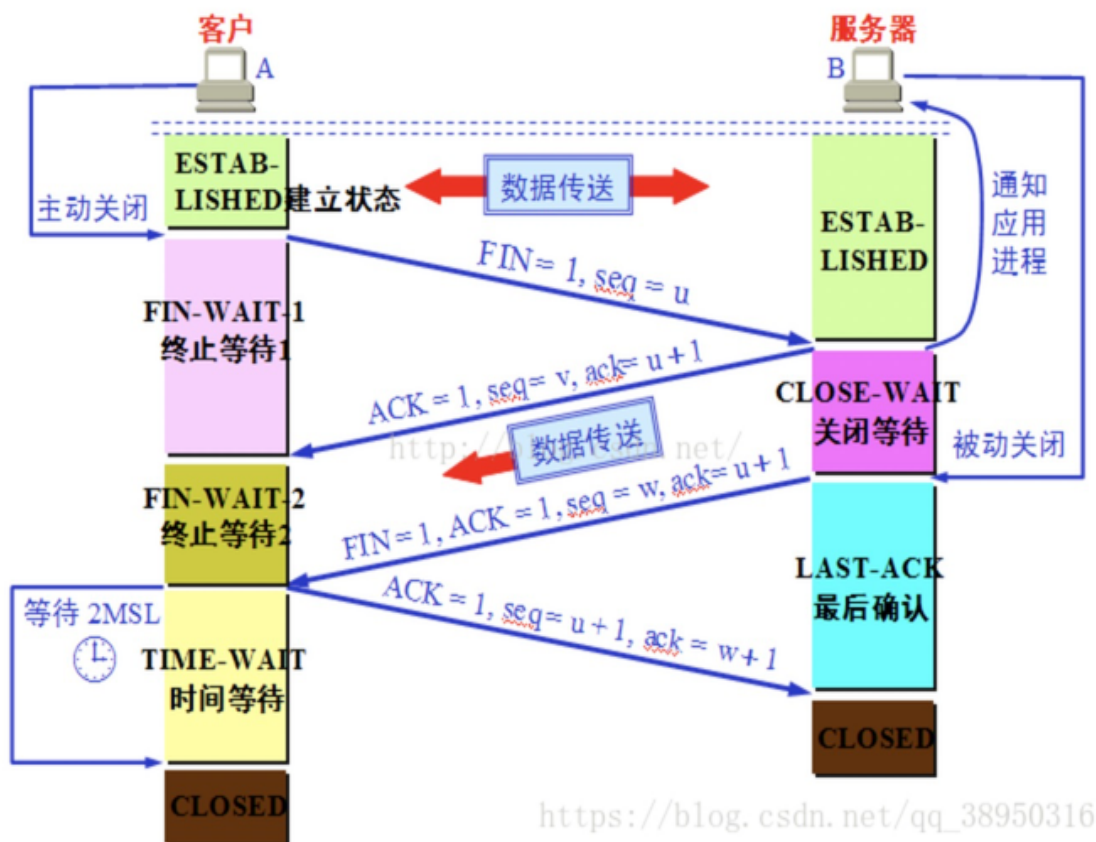
DOM节点中的各个元素都是以盒模型的形式存在，这些都需要浏览器去计算其位置和大小等，这个过程称reflow (回流)。当盒模型的位置,大小以及其他属性，如颜色,字体,等确定下来之后，浏览器便开始绘制内

容，这个过程称为 repain (重绘)。页面在首次加载时必然会经历 reflow 和 repain 。reflow 和 repain 过程是非常消耗性能的，它会破坏用户体验，有时会造成页面卡顿。所以我们应该尽可能少的减少 reflow 和 repain 。

c. 解析js

当文档解析过程中遇到js，会停止HTML的解析，先解析js，等到js加载解析完毕才继续解析HTML。因为JS可能会改动DOM和CSS，所以继续解析会造成资源浪费。





https://blog.csdn.net/qq_38950316

Http与Https

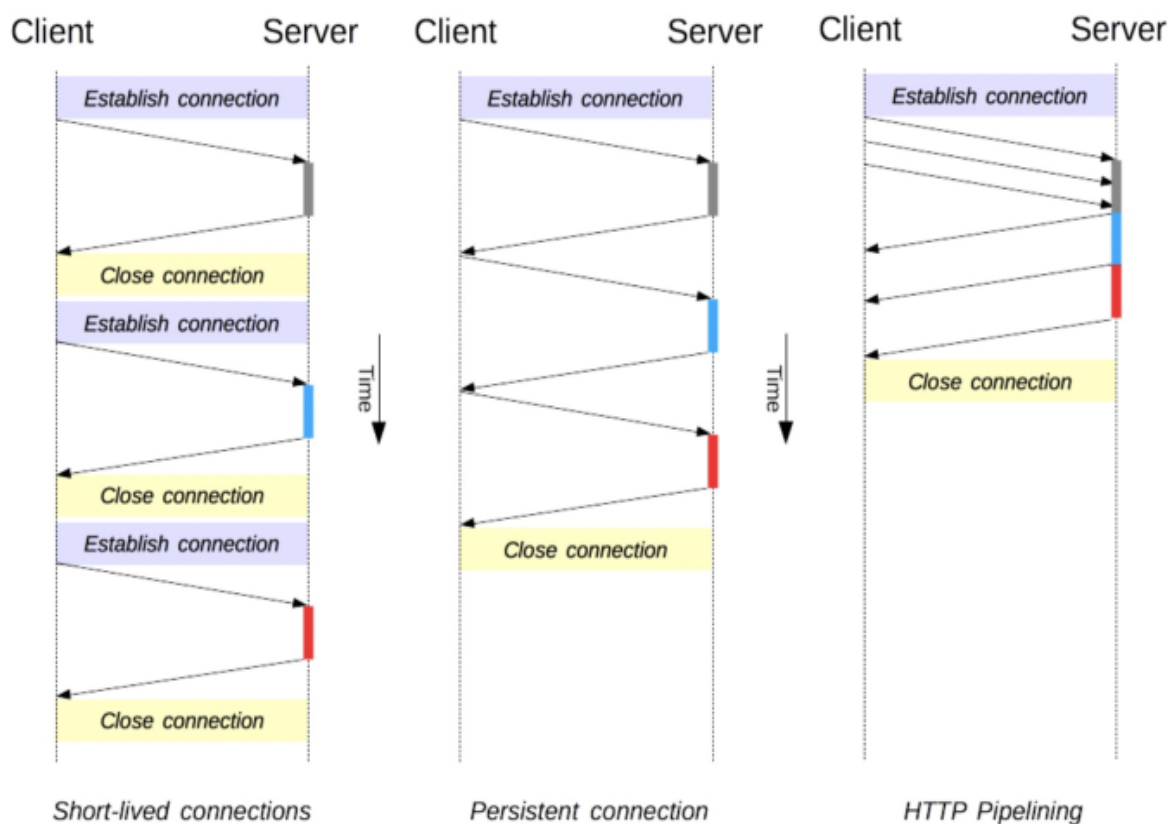
HTTP特点:

1. 无状态: 协议对客户端没有状态存储, 对事物处理没有“记忆”能力, 比如访问一个网站需要反复进行登录操作
2. 无连接: HTTP/1.1之前, 由于无状态特点, 每次请求需要通过TCP三次握手四次挥手, 和服务端重新建立连接。比如某个客户机在短时间多次请求同一个资源, 服务器并不能区别是否已经响应过用户的请求, 所以每次需要重新响应请求, 需要耗费不必要的时间和流量。
3. 基于请求和响应: 基本的特性, 由客户端发起请求, 服务端响应
4. 简单快速、灵活
5. 通信使用明文、请求和响应不会对通信方进行确认、无法保护数据的完整性

https特点

- 基于HTTP协议, 通过SSL或TLS提供加密处理数据、验证对方身份以及数据完整性保护。
- 收方能够证实发送方的真实身份;
- 发送方事后不能否认所发送过的报文;
- 收方或非法者不能伪造、篡改报文。

HTTP1.0, HTTP1.1, HTTP2



1. 短连接与长连接

当浏览器访问一个包含多张图片的 HTML 页面时，除了请求访问的 HTML 页面资源，还会请求图片资源。如果每进行一次 HTTP 通信就要新建一个 TCP 连接，那么开销会很大。

长连接只需要建立一次 TCP 连接就能进行多次 HTTP 通信。

- 从 HTTP/1.1 开始默认是长连接的，如果要断开连接，需要由客户端或者服务器端提出断开，使用 `Connection : close;`
- 在 HTTP/1.1 之前默认是短连接的，如果需要使用长连接，则使用 `Connection : Keep-Alive`。

2. 流水线

默认情况下，HTTP 请求是按顺序发出的，下一个请求只有在当前请求收到响应之后才会被发出。由于受到网络延迟和带宽的限制，在下一个请求被发送到服务器之前，可能需要等待很长时间。

流水线是在同一条长连接上连续发出请求，而不用等待响应返回，这样可以减少延迟。

Cookies

cookie 是服务器发送给浏览器的一小部分数据，浏览器可存储它，并将下一个请求发回给同一个服务器，cookie 判断请求是否来自同一个浏览器。

cookie 曾用于一般的客户端存储，但现在推荐使用现代存储 API（Web 存储 API，indexDB【存储大量数据，在 web worker 中可用】）；

why?

- cookie 随每个请求一起发送，因此可能会恶化性能。

- 每特定域名下数量有限。
- 存储量小（4KB）。【web storage 5MB】
- 需自己封装获取，设置，删除的方法。【web storage有setItem、getItem、removeItem、clear等方法】

属性

Expires, Max-Age

expires：特定日期，max-age：指定长度；

Domain, path

domain：指定允许接收的域名（及其子域名）若未指定，默认当前域名，不包含子域名，path：指定路径（及其子路径）；

secure, httpOnly

secure HTTPS协议通过加密请求发送到服务器（http无法使用），httpOnly：cookie只被发送到服务器；

Http 状态码

| 状态码 | 类别 | 含义 |
|-----|------------------------|---------------|
| 1XX | Informational（信息性状态码） | 接收的请求正在处理 |
| 2XX | Success（成功状态码） | 请求正常处理完毕 |
| 3XX | Redirection（重定向状态码） | 需要进行附加操作以完成请求 |
| 4XX | Client Error（客户端错误状态码） | 服务器无法处理请求 |
| 5XX | Server Error（服务器错误状态码） | 服务器处理请求出错 |



- 2** 成功状态码
 - 200 OK 请求成功
 - 201 已创建
 - 202 已接受
 - 203 非授权信息
 - 204 无内容
 - 205 重置内容
 - 206 部分内容
- 3** 重定向
 - 300 多种选择
 - 301 Moved Permanently 永久重定向
 - 302 Found 临时重定向
 - 303 查看其它地址
 - 304 Not Modified 未修改
 - 305 使用代理
- 4** 客户端错误
 - 400 Bad Request 请求的语法错误
 - 401 Unauthorized 要求身份验证
 - 403 Forbidden 服务器拒绝执行该请求
 - 404 Not Found 未找到资源
- 5** 服务器错误
 - 500 Internal Server Error 服务器内部错误
 - 501 服务器不支持该功能，无法完成请求
 - 502 Bad Gateway 服务器作为网关服务器执行请求时，从远程服务器接收到了无效的响应
 - 503 Service Unavailable 系统维护
 - 504 Gateway Time-out 超时

跨域

当一个请求url的**协议、域名、端口**三者之间任意一个与当前页面url不同即为跨域。

跨域出于浏览器的同源策略限制。同源策略（Sameoriginpolicy）是一种约定，它是浏览器最核心也最基本的安全功能，如果缺少了同源策略，则浏览器的正常功能可能都会受到影响。可以说Web是构建在同源策略基础之上的，浏览器只是针对同源策略的一种实现。同源策略会阻止一个域的javascript脚本和另外一个域的内容进行交互。所谓同源（即指在同一个域）就是两个页面具有相同的协议（protocol），主机（host）和端口号（port）。

解决跨域

1.CORS

CORS是跨域资源分享（Cross-Origin Resource Sharing）的缩写。它是 W3C 标准，属于跨源 AJAX 请求的根本解决方法。

1.普通跨域请求：只需服务器端设置Access-Control-Allow-Origin

2.带cookie跨域请求：前后端都需要进行设置。 `withCredentials=true`

2.jsonp

JSONP 是服务器与客户端跨源通信的常用方法。最大特点就是简单适用，兼容性好（兼容低版本IE），缺点是只支持get请求，不支持post请求。

核心思想：网页通过添加一个<script>元素，向服务器请求 JSON 数据，服务器收到请求后，将数据放在一个指定名字的回调函数的参数位置传回来。

3.主域相同，子域不同时，可通过设置document.domain解决无法读取非同源网页的 Cookie 问题

