# 一、本地存储

**Web Storage：**

一种类似cookie的本地存储方式

|  |  |  |  |
| --- | --- | --- | --- |
|  | cookie | localStorage | sessionStorage |
| 生命周期 | 根据set-cookie头设置其内容、 | 永久的，只要是在某个浏览器内存储，只要不主动删除，就永远可以访问到；无法跨域； | 同源的窗口中始终存在（即使刷新）。浏览器窗口关闭后销毁，同时独立的打开同一个窗口同一个页面，sessionStorage也是不一样的。 |
| 存储大小 | 4KB | 5MB | 5MB |
| 存储类型 | 字符串 | 字符串 | 字符串 |
| 存储位置 | 本地 | 本地 | 本地 |
| 获取方式 | document.cookie | window.localStorage | window.sessionStorage |

**Cookie、Session、Token:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Cookie** | **Session** | **Token** |
| **过程** | 1、用户登录后，服务器端返回头中设置Set-Cookie：cookieId，date等  2、浏览器按照返回头生成Cookie  3、剩下的看程序员的计划 | 1、用户登录后，服务器给用户一个sessionid（一般通过cookie传送），自己存下session，两者对应；  2、用户获得sessionid，存于cookie  3、用户再此登录时携带cookie中的sessionid  4、服务器验证发现sessionid与session对应，免登录（或其他） | 1、用户登录后，服务端为其设置userid，并加密生成一个签名，userid+签名=token全部发给用户  2、用户得到token，下次请求携带token（userid+签名）  3、服务端将userid加密后，与签名比较，相同就验证成功。 |
| **大小** | 4KB | Sessionid 4KB  Session 服务端决定 | 1、请求头Authorization中传送（chrome250kb)  2、url传送(至少的为2KB) |
| **类型** | 字符串 | Sessionid同Cookie  Session在服务端，自定义 | 字符串 |
| **时间** | set-cookie中设置  不设置默认浏览器未关闭 | 服务端决定set-cookie | 将时间戳放入token，验证时取出与当前时间判断 |
| **跨域** | 各网站不同，不可跨域共享 | **？** | **？** |
| **优势** |  |  |  |
| **缺点** | CSRF | 多服务器不共享session  服务其存过多session，压力过大 |  |

js设置cookie（chrome不支持本地设置cookie，只支持远程document中的cookie设置）：

（1）创建cookie

document.cookie="username=John Doe; expires=Thu, 18 Dec 2043 12:00:00 GMT";

（2）读取cookie

document.cookie="username=John Doe";

（3）修改cookie

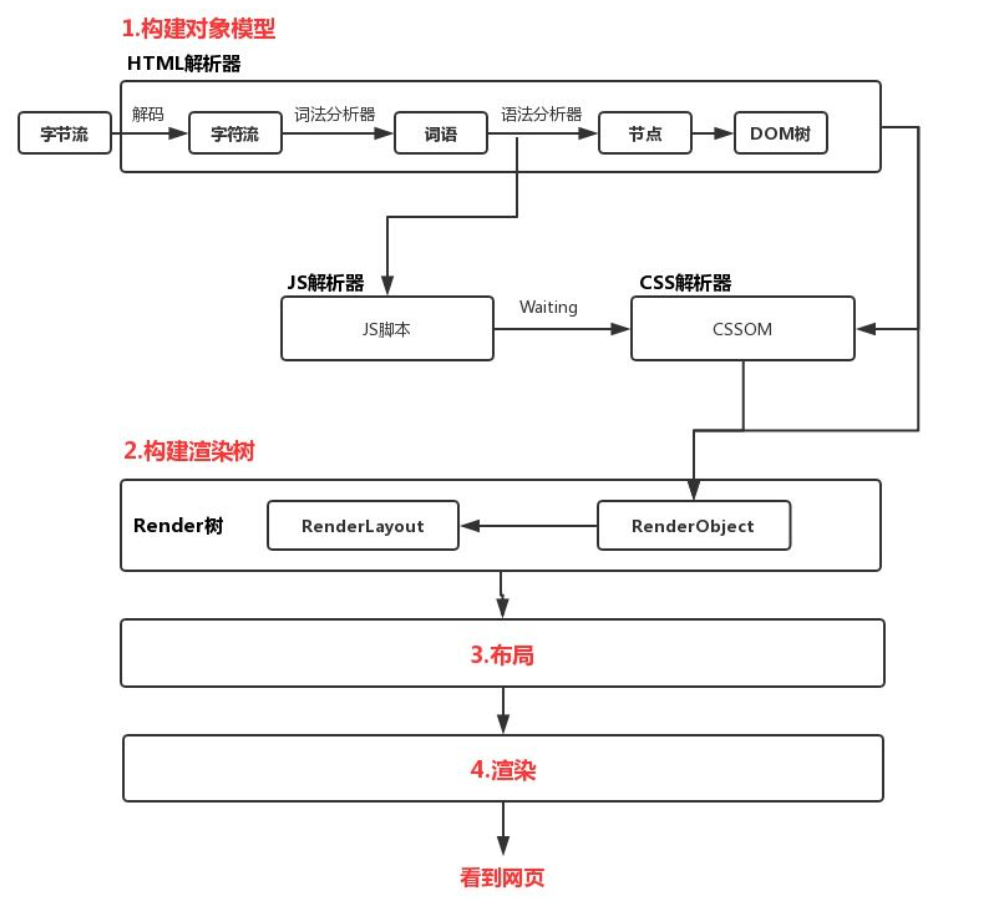
document.cookie="username=John Smith; expires=Thu, 18 Dec 2043 12:00:00 GMT";

（4）删除cookie

document.cookie="username=John Smith; expires=Thu, 18 Dec 2043 11:00:00 GMT";（时间设为之前即可）

# 二、浏览器渲染

总流程：



## 1、DOMTree构建·

（1）解析：将HTML原始字节码，通过设置的charset编码，转换成相应字符。

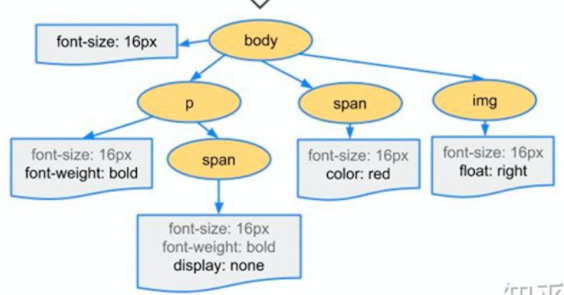
（2）词法分析：将字符串解析成各种标签（开始标签、结束标签、文本标签），此时

（3）构建DOM树：将各种标签生成为nodes，并构建DOM树（深度遍历下，边解析标签边生成node）

（如果遇到script/link，并且没设置defer/async的话，会去下载并执行其中的代码，如果其中想要操作DOM元素，会报错）

## 2、CSSOMTree构建：

总体流程和DOM相同：

（注意树中对p span与span的区分）

(如果js代码试图在CSSOMTree渲染未完成前进行修改css，浏览器会暂停脚本和DOM构建，直到CSSOM构建完成）

## 3、渲染树构建

DOMTree和CSSOMTree共同构建出渲染树render tree

渲染树根据会根据节点的属性（非根节点、透明、等）取构建Layer tree

Layer tree构建成我们所见到的画面

# 三、浏览器缓存

由上而下寻找：

1、Service Worker

2、Memory Cache

内存中的缓存，相同src的img只会被请求一次。禁止：Cache-control: no-store

3、Disk Cache

4、网络请求

# 四、性能优化

async/defer

# 五、路由跳转