事件循环

浏览器进程模型

进程

程序运行需要有自己的内存空间

每个应用至少有一个进程，进程之间相互独立

线程

进程中，运行代码的就是线程

一个进程至少有一个线程，称为主线程

浏览器有哪些进程和线程

多进程：

1. 浏览器进程：界面展示，用户交互，子进程管理
2. 网络进程：加载网络资源
3. 渲染进程：启动后开启一个渲染主线程，负责执行HTML，CSS，JS

默认情况，浏览器为每一个标签页开启一个新的渲染进程

多线程

渲染主线程的工作

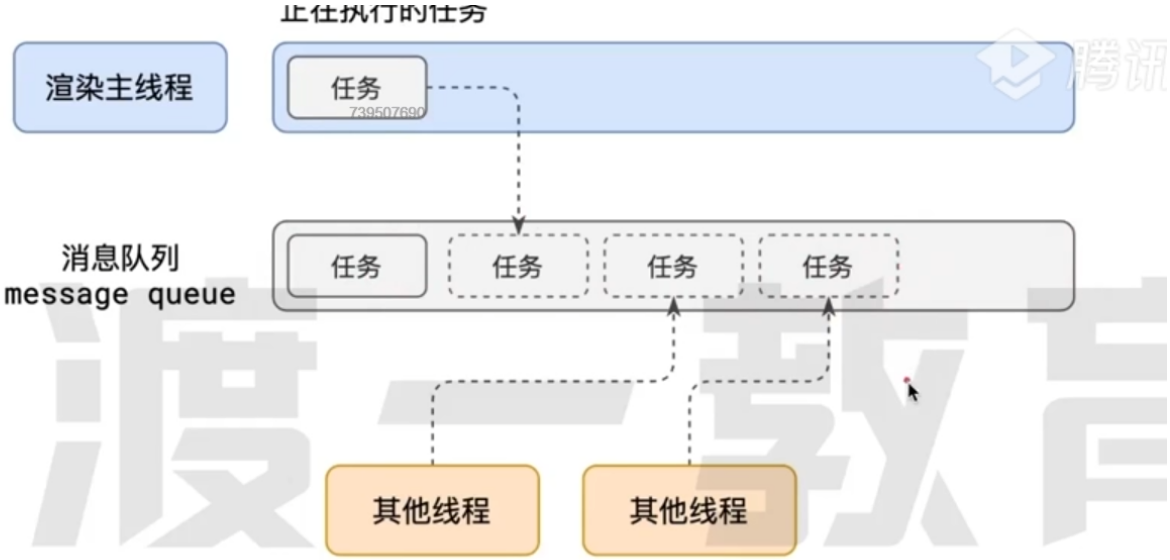
解析HTML，CSS

计算样式：em，百分比，层叠规则

布局

执行全局JS代码

主线程调度任务



排队

渲染主线程进行死循环，每一次循环检查消息队列是否有任务存在，有就取出第一个任务执行，执行完进行下一次循环，没有则进入休眠状态

其他线程和其他进程的线程都可以向消息队列添加任务

异步

计时开始就表示这个任务结束了，获取下一个任务

优先级

消息队列中有优先级

浏览器准备好一个微队列，微队列中的任务优先所有其他任务执行

目前chrome包含的队列：

- 延时队列：存放计时器到达后的回调，优先级中
- 交互队列：用户操作后产生的事务处理任务，优先级高

- 微队列：最快执行的任务，优先级最高，Promise

```
// 直接将一个函数放入微队列
Promise.resolve.then(函数)
```

JS中计时器能否做到精确计时

1.调用的是操作系统的计时函数，有误差

2.事件循环的影响，延时队列优先级低，带来偏差

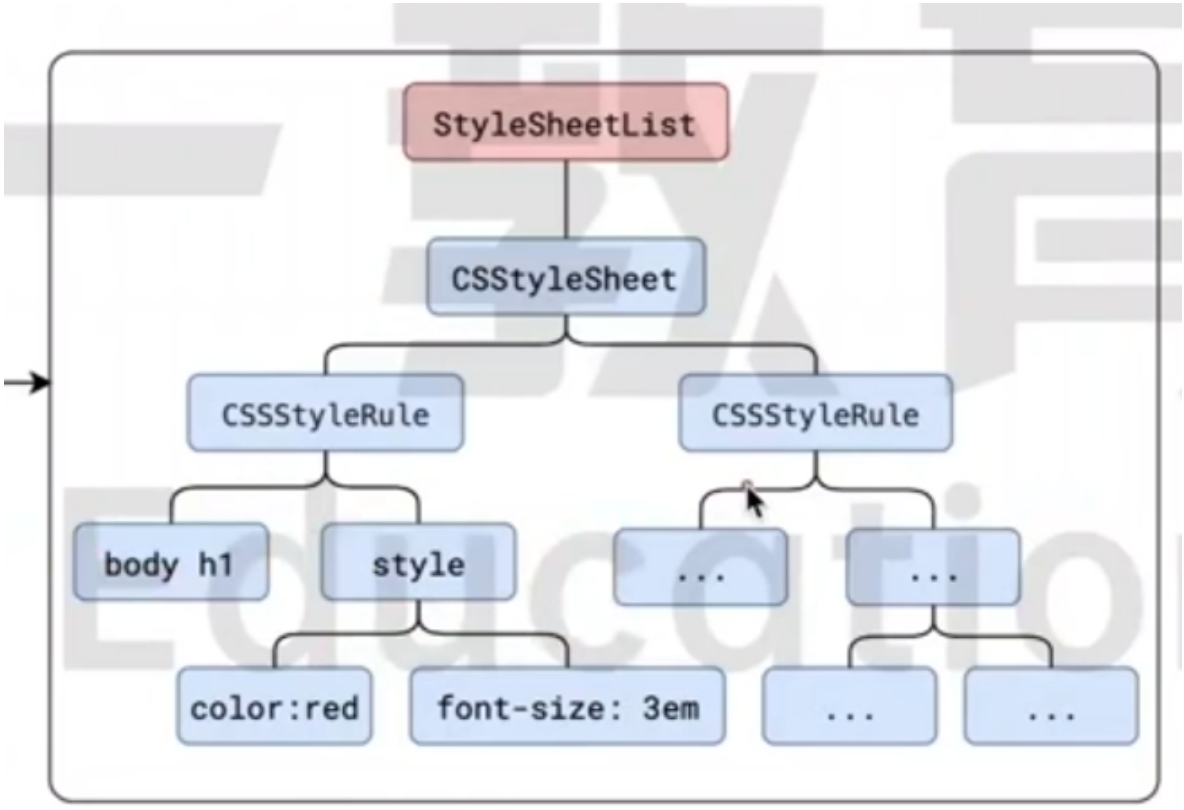
3.w3c规定当计时器嵌套超过五级时，间隔时间被增加到四毫米

浏览器渲染原理

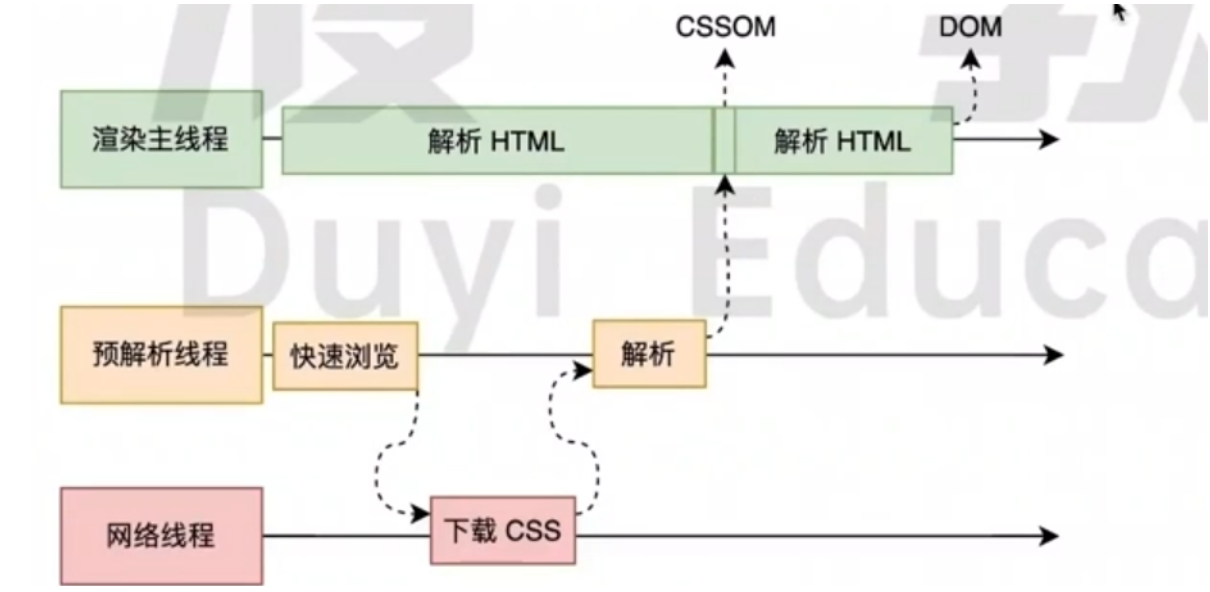
网络进程收到HTML字符串后，产生一个渲染任务，并传递给渲染主线程的消息队列，

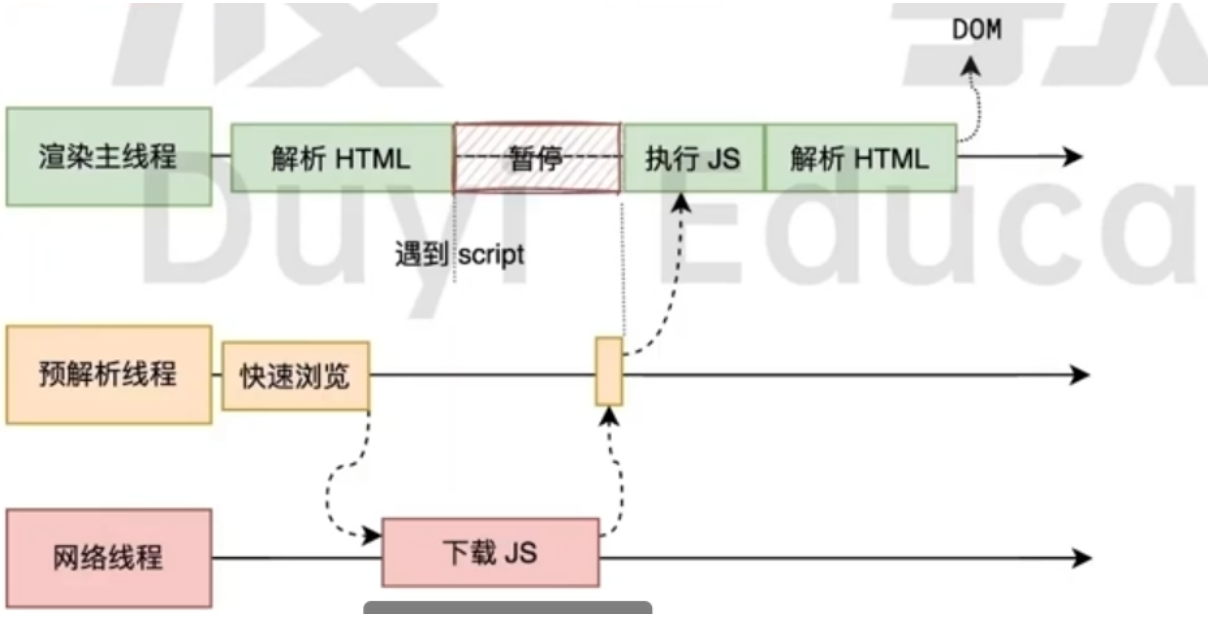
解析HTML

1. 生成DOM树，CSSOM树



2. 浏览器会启动预解析器率先下载和解析JS，CSS。遇到link，如果css还没解析好，主线程会继续解析后续的HTML，遇到script，会停止解析HTML，等待JS文件下载好，JS可能会操作DOM树，DOM树的渲染必须暂停





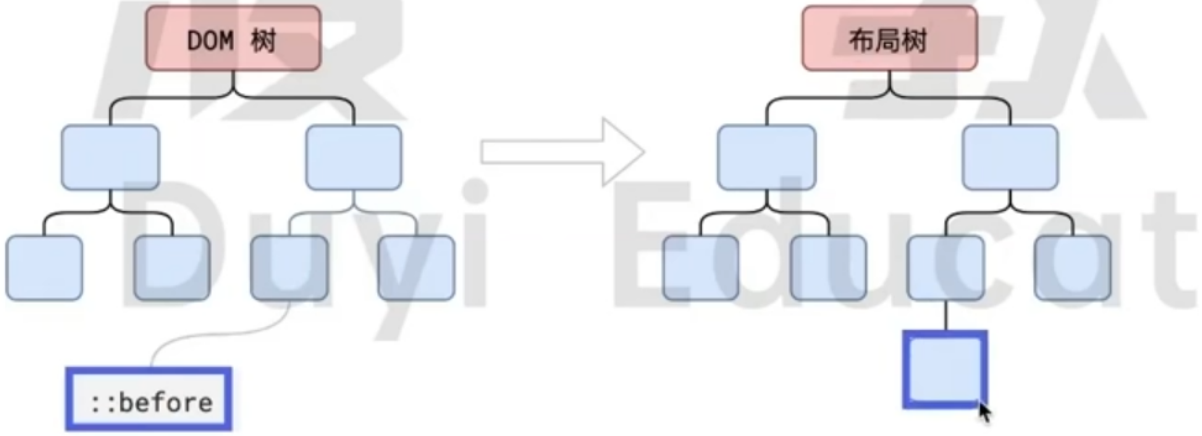
样式计算

主线程遍历DOM树，为每一个节点计算出最终样式。

将em变为px，

布局

遍历DOM树的每一个节点，计算每个节点的几何信息

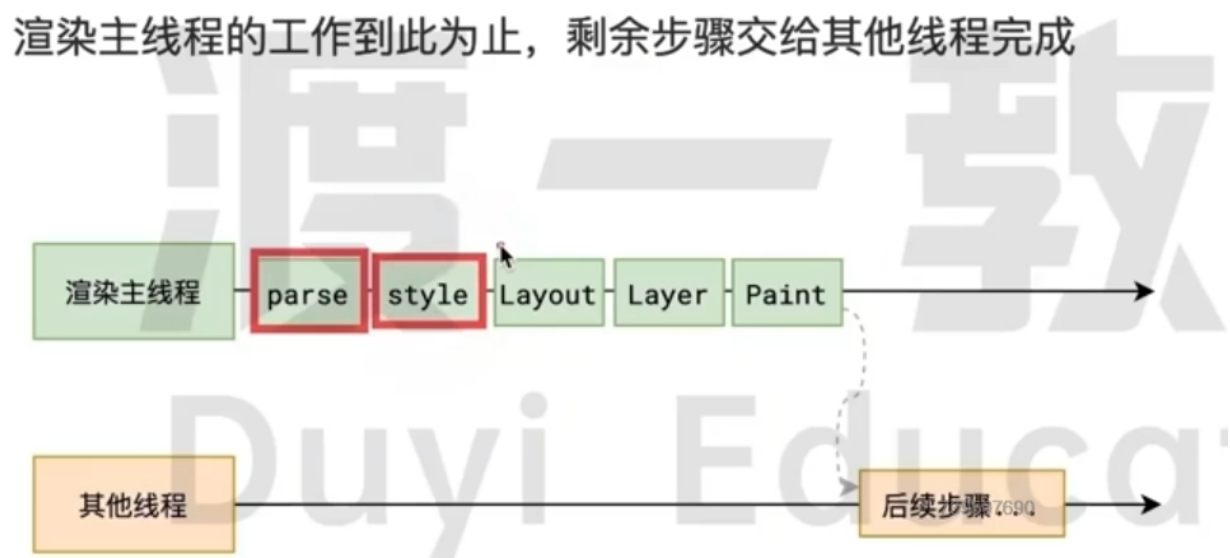


分层

层叠上下文

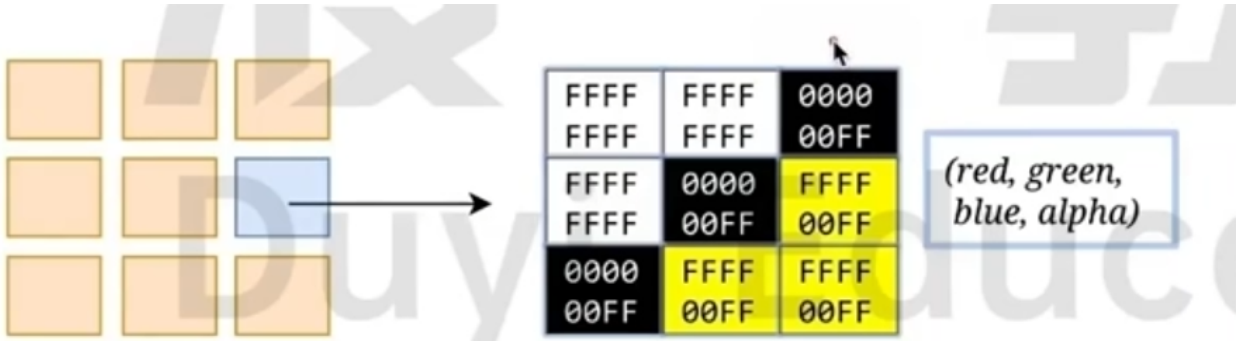
绘制

为每一层生成一个绘制指令



完成绘制后，主线程将每个图层的绘制信息提交到合成线程

合成线程分块后，提交到GPU进程进行光栅化



画

合成进程拿到每一个位图后生成一个个指引信息，指引会标识每一个位图应该画到哪个位置。

transform发生在合成线程，与渲染主线程无关，所以效率高

重绘

改动了可见样式，重新绘制

重排

重新计算布局树

浏览器会合并多次导致布局树反复计算的操作