JavaScript高级

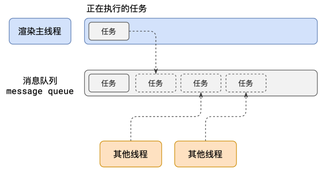
事件循环

浏览器是一个多线程多进程的应用程序

主要进程

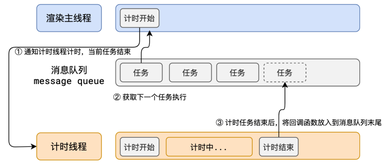
- 浏览器进程：主要负责界面显示、用户交互、子进程管理等。浏览器进程内部会启动多个线程处理不同的任务
- 网络进程：负责加载网络资源。网络进程内部会启动多个线程来处理不同的网络任务。
- 渲染进程：渲染进程启动后，会开启一个**渲染主线程**，主线程负责执行 HTML、CSS、JS 代码。
  - 默认情况下，浏览器会为每个标签页开启一个新的渲染进程，以保证不同的标签页之间不相互影响。

渲染主线程的工作方式：**消息队列**



- 在最开始的时候，渲染主线程会进入一个无限循环
- 每一次循环会检查消息队列中是否有任务存在。如果有，就取出第一个任务执行，执行完一个后进入下一次循环；如果没有，则进入休眠状态。
- 其他所有线程（包括其他进程的线程）可以随时向消息队列添加任务。新任务会加到消息队列的末尾。在添加新任务时，如果主线程是休眠状态，则会将其唤醒以继续循环拿取任务

渲染主线程如何实现JavaScript异步



消息队列是有优先级的

w3c标准

- 每个任务都有一个任务类型，同一个类型的任务必须在一个队列，不同类型的任务可以分属于不同的队列。在一次事件循环中，浏览器可以根据实际情况从不同的队列中取出任务执行。
- 浏览器必须准备好一个微队列，微队列中的任务优先所有其他任务执行

在目前 chrome 的实现中，至少包含了下面的队列

- 延时队列：用于存放计时器到达后的回调任务，优先级「中」
- 交互队列：用于存放用户操作后产生的事件处理任务，优先级「高」
- 微队列：用户存放需要最快执行的任务，优先级「最高」

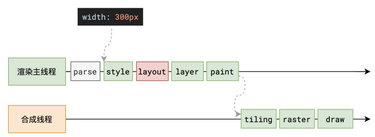
```
// 立即把一个函数添加到微队列
Promise.resolve().then(函数)
```

浏览器渲染原理

浏览器渲染流程是一个pipline

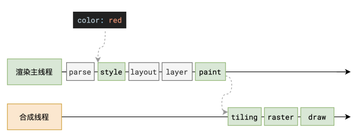
1. 解析HTML：将请求到的HTML解析成DOM，将CSS解析成CSSOM
   - 在解析HTML的时候遇到CSS怎么办？为了提高解析效率，浏览器会启动一个预解析器率先下载和解析 CSS
   - HTML 解析过程中遇到 JS 代码怎么办？渲染主线程遇到 JS 时必须暂停一切行为，等待下载执行完后才能继续 Js有可能会改变DOM
2. 样式计算：将DOM书和CSSOM树结合起来，计算DOM树上每个节点的样式 每个节点的样式 = 浏览器默认样式 + 开发者自定义样式 + 用户自定义样式
3. 布局计算：根据带样式的DOM，计算Layout Layout树中仅有需要显示的DOM节点 浏览器会添加节点（例如匿名块/行盒），删除不需要显示的节点
4. 分层：将需要显示的页面分为多层 这样更新页面中某个元素时，只需要更新包含该元素的层，而不是整个页面
   - 优化浏览器效率
   - 滚动条、堆叠上下文、transform、opacity 等样式都会或多或少的影响分层结果，也可以通过`will-change`属性更大程度的影响分层结果。
5. 绘制：为每一层生成绘制指令
6. 分块：将每一层划分为多个小区域，分别绘制，优先绘制需要显示的部分 分块的工作是交给多个线程同时进行的
7. 光栅化：GPU将指令绘制成待显示像素画面
8. 画：合成线程计算出每个位图在屏幕上的位置，交给 GPU 进行最终呈现

渲染主线程



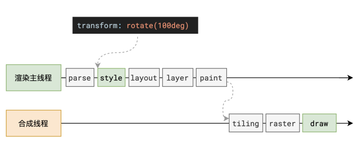
reflow

- 当进行了会影响布局树的操作后，需要重新计算布局树，会引发 reflow
- 为了避免连续的多次操作导致布局树反复计算，浏览器会合并这些操作，当 JS 代码全部完成后再进行统一计算。所以，改动属性造成的 reflow 是异步完成的
  - 也同样因为如此，当 JS 获取布局属性时，就可能造成无法获取到最新的布局信息
  - 浏览器在反复权衡下，最终决定获取属性立即reflow



repaint

- repaint 的本质就是重新根据分层信息计算了绘制指令
- 当改动了可见样式后，就需要重新计算，会引发 repaint



为什么 transform 的效率高?

- 因为 transform 既不会影响布局也不会影响绘制指令，它影响的只是渲染流程的最后一个「draw」阶段
- 由于 draw 阶段在合成线程中，所以 transform 的变化几乎不会影响渲染主线程。反之，渲染主线程无论如何忙碌，也不会影响 transform 的变化。