如何理解JS的异步？

JS是一门单线程的语言，这是因为它运行在浏览器的渲染主线程中，而渲染主线程只有一个。

渲染主线程承担着诸多的工作，如渲染页面、解析HTML、解析CSS、执行JS等

如果使用同步的方式，就极有可能导致主线程产生阻塞，从而导致消息队列中的很多其他任务无法得到执行。

所以浏览器使用异步的方式来避免。具体做法是当某些任务发生时，比如计时器、网络、事件监听。，主线程将任务交给其他线程去处理，自身立即结束任务的执行，转而执行后续代码。当其他线程完成时，将实现传递的回调函数包装成任务，加入到消息队列的末尾排队，等待主线程调度执行。

在这种异步模式下，浏览器永不阻塞，从而最大限度的保证了单线程的流畅运行。

阐述一下JS的事件循环

事件循环又叫消息循环，是浏览器渲染主线程的工作方式

在谷歌源码中，它开启一个不会结束的for循环，每次循环从消息队列中取出第一个任务执行，而其他线程只需要在合适的时候将任务加入到队列末尾即可。

过去把消息队列简单分为宏队列和微队列，这种说法现在已经无法满足复杂的浏览器环境，现在的W3C的解释是，每个任务有不同的类型，同类型的任务必须在同一个队列，不同的任务可以属于不同的队列。不同任务队列有不同的优先级，在一次事件循环中，由浏览器自行决定取哪一个队列的任务。但浏览器必须有一个微队列，微队列的任务一定具有最高的优先级，必须优先调度执行。

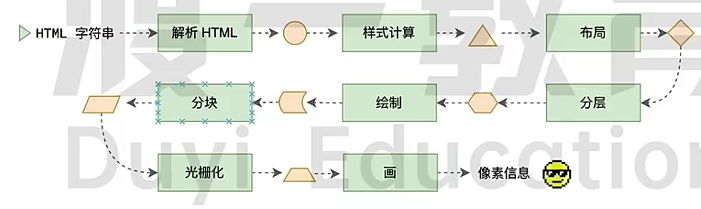
JS中的计时器能做到精确计时吗？为什么？

不行。

1. 计算机硬件没有原子钟，无法做到精确计时
2. 操作系统的计时函数本身就有少量的偏差
3. 按照W3C的标准，浏览器实现计时器时，如果嵌套层级超过5层，则会带有4ms的最少时间，这样在计时时间少于4ms时带来了误差
4. 受事件循环的影响，计时器的回调函数只能在主线程空闲时运行，因此带来偏差

浏览器是如何渲染页面的？ 内容必须在行盒中、行盒和块盒不能相邻

当浏览器的网络线程收到HTML文档后，会产生一个渲染任务，并将其传递给渲染主线程的消息队列，在事件循环的机制下，渲染主线程取出消息队列中的渲染任务，开启渲染流程。



渲染的第一步：解析HTML（DOM树、CSSOM树）——Parse HTML

浏览器在开始解析之前，会启动一个预解析线程，当解析过程中遇到CSS，同步解析CSS，不影响HTML的解析过程；当解析过程中遇到JS，会暂停解析HTML，先解析好JS，再继续解析HTML（JS代码在执行过程中可能会修改DOM树）

渲染的第二步：样式计算——Style

主线程遍历所有的DOM树，依次为树中的每个节点计算出最终的样式。预设值会变成绝对值，例如颜色转换（red=>rgb(255, 0, 0)）、像素转换（em=>px）

渲染的第三步：布局-布局树（Layout树）——Layout

布局阶段会遍历DOM树的每一个节点，计算每个节点的几何信息（节点的宽高、相对包含块的位置）

大多数情况下DOM树和布局树是不对应的。

Display:none的节点没有几何信息，不会生成到布局树

伪元素选择器——DOM树中不存在伪元素节点，但有几何信息，会生成到布局树中

匿名行盒、匿名块盒等也会导致DOM树和布局树不对应

渲染的第四步：分层——Layer

主线程会对整个布局树进行一个分层。当某一个层发生改变后，只会对该层进行处理，不影响其他的层，可以提升效率。

滚动条、堆叠上下文、transform、opacity等会影响分层，也可通过will-change属性来进行过分层效果。

渲染的第五步：绘制——Paint

为每一层生成如何绘制的指令，完成绘制后，主线程将每个图层的绘制信息提交给合成线程，剩余工作将由合成线程完成。

渲染主线程的工作结束，后续的操作交给其他线程

渲染的第六步：分块——Tiling

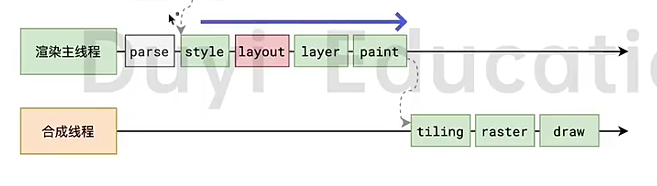
将每一层都分成多个小块，分块的工作交给多个线程同时进行

渲染的第七步：光栅化——Raster

光栅化是将每个块变成位图；优先处理靠近视口区域的块。在GPU进程中进行

渲染的第八步：画——Draw

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



什么是reflow?

重新计算layout树（布局树）

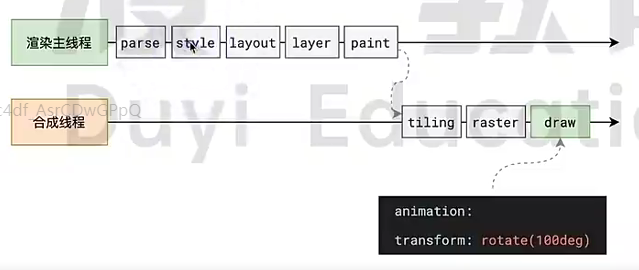
当进行了会影响布局树的操作后，需要重新计算布局树

什么是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transfrom效率高？



Transform不会影响布局，也不会影响绘制指令，只会改变draw，而draw是在合成线程上进行，所以对主线程没有影响