现代浏览器网页渲染原理

最开始的时候。当浏览器拿到文件的时候，会根据html生成html树(DOM)。根据CSS生成CSS树（CSSOM）。CSSOM和DOM合并会生成一个rendering tree。

接下来浏览器开始对rendering tree进行渲染流程。

**渲染路径：**

JavaScript/CSS/Web Animation API->计算样式->布局->绘制->复合

1. Javascript/CSS/Web Animation API会触发视觉变化
2. 浏览器会进行样式计算，计算出所有元素最终的形态。
3. 根据元素的样式（大小，位置），将它们布局到页面上。
4. 将元素画到页面上，包括元素，阴影。但是他们会画到不同的图层上（为了提高效率）。换句话说，绘制是像素化每个节点的过程。
5. 最后组合到一起形成一张图。

布局阶段，浏览器会计算每个节点精确的位置和大小。

回流（影响布局）的操作：

* 添加/删除元素
* Display:none
* 移动元素位置
* 修改浏览器大小，字体大小
* Client-width

**布局抖动：**

获取OffsetTop/left/top值时，浏览器会进行一个读取操作，会触发一个布局的重新计算（强制回流）。

然后写入width时，又会触发一个回流。

解决方案：

使用不会触发布局流程的方法，比如动画的translate。

读写分离，批量读，批量写(Fast DOM)。

将页面拆分成图层，图层之间绘制图片，互不影响。

以下操作只触发复合

transform:translate

transform:scale

transform:rotate

Opacity.

浏览器会自动拆图层。图层里面的变化只触发样式重新计算，跳过布局，直接跳到最后一步复合。

图层数量太多也会影响性能。

**如何避免重绘**

把图片单独提取到一个图层（will-change:transform），使用transform和opacity。

**高频事件处理函数，要进行防抖**

所谓高频事件处理函数就是一帧内会触发多次响应，比如滚动鼠标移动。一帧16毫秒。一秒60帧。

可以使用requestAnimationFrame这个函数来进行处理

rAF的触发是在JS事件触发之后，布局流程之前。也就是说在布局和绘制发生之前触发函数。 JS会自动进行raf的调度，会尽量在每一次绘制之前，触发raf。

用window.rrequestAnimationFrame()进行包裹。

window.requestAnimationFrame() 告诉浏览器——你希望执行一个动画，并且要求浏览器在下次重绘之前调用指定的回调函数更新动画。该方法需要传入一个回调函数作为参数，该回调函数会在浏览器下一次重绘之前执行

**React时间调度实现**

react-fiber将dom批量任务拆解成多个小任务，再使用时间调度去完成。

实际上是用requestAnimationFrame去模拟requestIdleCallback（rIC）

requestIdleCallback希望解决的痛点是，希望浏览器在一帧之内，如果有空余时间，去执行一些其他操作。 但是这个兼容性不好，所以React通过requestAnimationFrame去模拟这个函数。

requestIdleCallback发生在一帧的最后。但是我们还是要把空闲时间留给用户进行交互，至少留50ms。

requestAnimationFrame有一个超时时间。但是react里面的rAF自己实现了一个源码，可以实现后台运行。 并且里面实际上是根据优先级来维护了一个队列。