# 原文链接http://kb.cnblogs.com/page/169820/

# [浏览器的重绘与重排](http://kb.cnblogs.com/page/169820/)

　　在项目的交互或视觉评审中，前端同学常常会对一些交互效果质疑，提出这样做不好那样做不好。主要原因是这些效果通常会产生一系列的浏览器重绘(redraw)和重排(reflow)，需要付出高昂的性能代价。那么，什么是浏览器的重绘和重排呢？二者何时发生以及如何权衡？如何在具体的开发过程中将重绘和重排引发的性能问题考虑进去？本文期待可以部分解释以上三个问题。

　　浏览器从下载文档到显示页面的过程是个复杂的过程，这里包含了重绘和重排。各家浏览器引擎的工作原理略有差别，但也有一定规则。简单讲，通常在文档初次加载时，浏览器引擎会解析HTML文档来构建DOM树，之后根据DOM元素的几何属性构建一棵用于渲染的树。渲染树的每个节点都有大小和边距等属性，类似于盒子模型（由于隐藏元素不需要显示，渲染树中并不包含DOM树中隐藏的元素）。当渲染树构建完成后，浏览器就可以将元素放置到正确的位置了，再根据渲染树节点的样式属性绘制出页面。由于浏览器的流布局，对渲染树的计算通常只需要遍历一次就可以完成。但table及其内部元素除外，它可能需要多次计算才能确定好其在渲染树中节点的属性，通常要花3倍于同等元素的时间。这也是为什么我们要避免使用table做布局的一个原因。

　　重绘是一个元素外观的改变所触发的浏览器行为，例如改变visibility、outline、背景色等属性。浏览器会根据元素的新属性重新绘制，使元素呈现新的外观。重绘不会带来重新布局，并不一定伴随重排。

　　重排是更明显的一种改变，可以理解为渲染树需要重新计算。下面是常见的触发重排的操作：

　　1. DOM元素的几何属性变化

　　当DOM元素的几何属性变化时，渲染树中的相关节点就会失效，浏览器会根据DOM元素的变化重新构建渲染树中失效的节点。之后，会根据新的渲染树重新绘制这部分页面。而且，当前元素的重排也许会带来相关元素的重排。例如，容器节点的渲染树改变时，会触发子节点的重新计算，也会触发其后续兄弟节点的重排，祖先节点需要重新计算子节点的尺寸也会产生重排。最后，每个元素都将发生重绘。可见，重排一定会引起浏览器的重绘，一个元素的重排通常会带来一系列的反应，甚至触发整个文档的重排和重绘，性能代价是高昂的。

　　2. DOM树的结构变化

　　当DOM树的结构变化时，例如节点的增减、移动等，也会触发重排。浏览器引擎布局的过程，类似于树的前序遍历，是一个从上到下从左到右的过程。通常在这个过程中，当前元素不会再影响其前面已经遍历过的元素。所以，如果在body最前面插入一个元素，会导致整个文档的重新渲染，而在其后插入一个元素，则不会影响到前面的元素。

　　3. 获取某些属性

　　浏览器引擎可能会针对重排做了优化。比如Opera，它会等到有足够数量的变化发生，或者等到一定的时间，或者等一个线程结束，再一起处理，这样就只发生一次重排。但除了渲染树的直接变化，当获取一些属性时，浏览器为取得正确的值也会触发重排。这样就使得浏览器的优化失效了。这些属性包括：offsetTop、offsetLeft、 offsetWidth、offsetHeight、scrollTop、scrollLeft、scrollWidth、scrollHeight、clientTop、clientLeft、clientWidth、clientHeight、getComputedStyle() (currentStyle in IE)。所以，在多次使用这些值时应进行缓存。

　　此外，改变元素的一些样式，调整浏览器窗口大小等等也都将触发重排。

　　开发中，比较好的实践是尽量减少重排次数和缩小重排的影响范围。例如：

　　1. 将多次改变样式属性的操作合并成一次操作。例如，

　　JS:

var changeDiv = document.getElementById(‘changeDiv’);

changeDiv.style.color = ‘#093′;

changeDiv.style.background = ‘#eee’;

changeDiv.style.height = ’200px’;

　　可以合并为：  
　　CSS:

div.changeDiv {

background: #eee;

color: #093;

height: 200px;

}

　　JS:

document.getElementById(‘changeDiv’).className = ‘changeDiv’;

　　2. 将需要多次重排的元素，position属性设为absolute或fixed，这样此元素就脱离了文档流，它的变化不会影响到其他元素。例如有动画效果的元素就最好设置为绝对定位。

　　3. 在内存中多次操作节点，完成后再添加到文档中去。例如要异步获取表格数据，渲染到页面。可以先取得数据后在内存中构建整个表格的html片段，再一次性添加到文档中去，而不是循环添加每一行。

　　4. 由于display属性为none的元素不在渲染树中，对隐藏的元素操作不会引发其他元素的重排。如果要对一个元素进行复杂的操作时，可以先隐藏它，操作完成后再显示。这样只在隐藏和显示时触发2次重排。

　　5. 在需要经常获取那些引起浏览器重排的属性值时，要缓存到变量。