重绘和回流

浏览器呈现页面的处理过程

1. 解析文档中的HTML代码构建成DOM树。

DOM树和html标签完全一一对应，包括display:none隐藏，还有用JS动态添加的元素等。

2. 把所有样式解析成样式结构体。

在解析的过程中会去掉浏览器不能识别的样式，比如IE会去掉-moz开头的样式，FireFox会去掉_开头的样式。

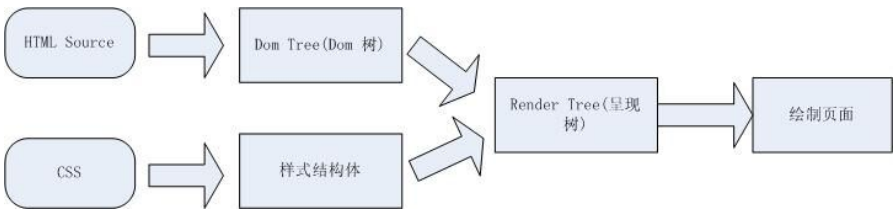
3、DOM 树和样式结构体组合后构建渲染树。

渲染树会忽略掉不用于呈现（不需要渲染）的元素，比如head、display:none的元素等。

设置了visibility:hidden的节点还是会包含到渲染树中，因为visibility:hidden 会影响布局，会占有空间。

渲染树中的每一个节点都存储有对应的css属性。

4. 当渲染树构建完成后，浏览器就可以根据渲染树将元素放置到页面上正确的位置了，再根据渲染树节点的样式属性绘制出页面。



重绘(repaint)

重绘是指一个元素外观的改变所触发的浏览器行为。部分节点需要更新，但是没有影响其他元素的布局。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。

发生重绘的情况

改变visibility、outline、字体颜色、背景颜色等属性。

回流/重排(reflow)

回流是渲染树的结构发生变化（元素的尺寸，位置，隐藏等）而需要重新计算。浏览器会使渲染树中受到影响的部分失效，并重新构造渲染树。

每个页面至少需要一次回流，就是在页面第一次加载的时候。

回流必将引起重绘，而重绘不一定会引起回流。回流比重绘的代价要更高。

发生回流的情况

1. 页面渲染初始化
2. 浏览器窗口尺寸改变——resize事件触发
3. DOM树的结构变化，例如节点的增减、移动、隐藏(display:none)等
4. 元素位置改变
5. 元素尺寸改变——边距、填充、边框、宽度和高度
6. 内容改变——比如用户在input框中输入文字
7. 样式改变——设置 style 属性的值
8. 改变字体大小
9. 激活 CSS 伪类，比如 :hover

10. fixed定位的元素，在拖动滚动条的时候

11. 浏览器引擎会针对重排做优化，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：

a. offsetTop, offsetLeft, offsetWidth, offsetHeight
b. scrollTop/Left/Width/Height
c. clientTop/Left/Width/Height
d. width,height
e. 调用了getComputedStyle(), 或者 IE的 currentStyle

减少回流方法

1. 避免使用CSS的JavaScript表达式，将多次改变样式属性的操作合并成一次操作：

JS

```
var changeDiv = document.getElementById( ‘changeDiv’ );
changeDiv.style.color = ‘#093’ ;
changeDiv.style.background = ‘#eee’ ;
changeDiv.style.height = ’ 200px’ ;
```

优化：

CSS

```
div.changeDiv {
background: #eee;
color: #093;
height: 200px;
}
```

JS

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

2.减少dom操作

3. 将需要多次重排的元素（例如有动画效果的元素），position 属性设为 absolute 或 fixed，这样此元素就脱离了文档流，它的变化不影响其他元素的布局，只会导致重绘。

4. 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。

5. 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。

6. 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。

7. 避免使用table布局，是因为可能会因为表格最后一个单元格的内容过宽而导致纵列大小完全改变，即使是一些小的变化也将导致表格(table)中的所有其他节点回流。

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。