目录

20 重绘与回流的相爱相杀

更新时间：2019-09-02 09:57:51

> 没有智慧的头脑，就象没有腊烛的灯笼。
>
> ——列夫·托尔斯泰

重绘(Repaint)和回流(Reflow)是我们在性能优化的时候经常会听到的两个词，这两个词与渲染流程息息相关的。前面我们在渲染原理与性能优化这一小节当中我们提到浏览器把 HTML 解析 DOM，CSS 解析成 CSSOM，然后DOM 和 CSSOM 一起附着合成 Render Tree。

然后浏览器就可以根据 Render Tree 绘制具体的页面，当 Render Tree 当中有元素的大小、布局、结构等发生改变时，就会触发回流，每个页面在首次加载的时候都需要进行至少一次回流；当 Render Tree 有元素属性需要更新，这些属性只是影响元素的外观、风格，那么则会触发重绘。

触发重绘与回流的具体操作

触发重绘的相关属性

影响重绘的一般就是和它的外观息息相关的一些属性，这里我单独简单罗列一下这些属性，大家无需记忆，只要在大脑中有个概念即可，知道什么属性会引发重绘，如下：

- color、background 相关属性(如：background-color、background-image 等)
- outline 相关属性(outline-color、outline-width)、text-decoration
- border-radius、visibility、box-shadow

触发回流的操作

触发回流的操作无非就是改变元素的位置和尺寸等等，这里我们也总结了一些常见的会触发回流的操作，如下：

- 页面初次渲染

- 浏览器窗口大小变化
- 添加或者删除 DOM 元素
- 激活 CSS 伪类
- 改变字体

如上就是一些常见的回流操作，这里要注意回流与重绘相比开销更大一些，而且回流一定会引发重绘，而重绘不一定会引起回流，这就是重绘与回流的联系，我们可以根据它们之间的联系对其概念加深记忆和理解。

如何减少重绘与回流

上面我们已经介绍了触发重绘和回流的一些操作，由于这些操作对性能的影响都是非常大的，那么我们最好的办法就是尽量减少这些操作，具体的方法我们介绍如下：

- 如果大家接触前端足够早，那么一定听说过表格(table)布局，在之前还没有 flex 和 grid 这么先进的布局方法，所以前端早期都是使用表格布局，表格有一个非常大的缺陷，那就是它完成整个布局需要进行多次计算，这就会导致大量的回流操作，所以我们现在基本上已经放弃了表格布局，大家在日常开发当中也要避免使用表格布局方法。

- 大家都知道减少重绘与回流最好的办法就是减少触发它们操作的次数，其实针对这个浏览器内部也做了优化操作，浏览器会自己有维护一个队列，然后把所有会触发重绘以及回流的操作放到这个队列当中， 当队列当中的操作达到一定数量，那么就进行一次批处理，这样就原来多次触发重绘和回流的操作就只变成一次了，自然效率大大提高。

  这里我补充一句学习过 Java 的同学肯定都知道 JVM 虚拟机这个概念，其中 JVM 当中有一个非常著名的算法那就是标注清楚垃圾回收算法，这个算法的核心就是将多次类似的操作合并成一次操作，很显然这个与浏览器的这个优化是非常类似的，所以大家学习的时候也可以进行类比，很多东西在原理上面都是相通的，我们虽然是前端开发，但是眼界不能仅仅局限于前端的一亩三分地，其他领域我们也要进行相关的学习和了解。

- 我们在日常的开发当中既会对元素进行写操作，也会对元素进行读操作，这里涉及到的优化就是进行读写分离，看如下一段代码：

```
div.style.width = '10px';
div.style.height = '10px';
div.style.right = '10px';
div.style.left = '10px';
console.log(div.offsetWidth);
console.log(div.offsetHeight);
console.log(div.offsetRight);
console.log(div.offsetLeft);
```

  如上我们把写操作集中到最上面，读操作集中到最下面，这样我们从始至终就只触发了一次回流操作，而不是四次，后面的读操作并不会触发回流，当然一处这样的操作对性能的提升可能帮助有限，但是如果我们在整个操作当中都是按照这个规范进行，相信相关性能可得到大幅度的提升。

- 前面我们提到浏览器内部会自己维护一个队列，把一次一次的操作进行批量处理提高性能。但是有一些实时属性并不会遵循这个策略，比如： `scrollWidth` 、 `scrollHeight` 、 `scrollTop` 、 `scrollLeft` 、 `getBoundingClientRect()` 等等。如果我们需要这些实时属性

们还是看一段代码，如下：

```
// 不缓存
for(let i = 0;i < 10;i++) {
  div.style.left = div.offsetLeft + 10 + 'px';
        div.style.right = div.offsetRight + 10 + 'px';
}
// 缓存
var curLeft = div.offsetLeft;
var curRight = div.offsetRight;
for(let i = 0;i < 10;i++) {
  div.style.left = curLeft + 10 + 'px';
        div.style.right = curRight + 10 + 'px';
}
```

如上，我们没有使用缓存方法之前，我们每次都需要获取这些实时属性，这样就需要浏览器每次都需要进行计算，性能非常低。如果我们使用变量将其先缓存起来，这样我们只需要计算一次即可。

- 修改样式的时候使用类名的方式一次性进行修改，在实际开发当中我们尽量使用类名方式集中的对样式进行修改，而不是使用 style 的方式一个一个赋予相关元素的样式属性。

- 对一些涉及到复杂动画的元素，在不影响相关布局的情况下，尽量将其 position 属性设置为 absolute 或者fixed，因为这样的话我们不会影响其他元素的相关布局，只会自己进行重新绘制，相比较不使用，性能消耗更低。

扩展部分

上面部分我们讲了一些常用的减少重绘与回流的方法，其中有提到在运用高级动画的时候，我们可以将元素的position属性设置为absolute或者fixed，下面我们就具体说一下这背后的原因。

要说明白这个问题还是离不开渲染原理，在渲染原理与性能优化小节当中，我们介绍了浏览器最后是根据Render Tree来进行页面的绘制，其实这不是最全面的介绍，这里我把整个完成流程的图重新绘制了一下，如下：

之所以在渲染部分我们没有介绍 Render Layer 下半部分的内容，是因为我觉得把下班部分的内容放到重绘与回流这一小节当中理解起来更加容易一些，相对来说结合实例记忆才会更加深刻。上面提到把 position 设置为absolute 或者 fixed，其实就是把 Render Tree 上面的这个元素提升到了 Render Layer，提升到 Render Layer 之后，这个时候就不会影响其他元素也进行回流操作，而只会自己进行回流操作，那么这就是背后真正的原因。

> 件加速(GPU加速)来进行优化，比如说一些移动元素的操作我们可以使用 translate3d() 方法来完成，使用这个方法之后，我们就触发了硬件加速，由于硬件加速使用的是 GPU 绘制，所以速度更快，而且不会引起回流和重绘，性能也可以得到较大的提升。

小结

这一小节我们介绍了回流与重绘的相关概念以及相关的优化方法，在扩展部分还为大家介绍一些操作的详细原因，相信如果在面试的时候，遇到类似的问题，如果你能将扩展部分的内容回答出来，相信对你的整体表现可以加分不少，所以这里推荐学有余力的同学可以牢记整个渲染流程。

← 19 不容小觑的 DOM 性能优化 21 Event Loop 力挽狂澜 →

精选留言 1

欢迎在这里发表留言，作者筛选后可公开显示

sih

render tree后面的流程能不能详细讲下？看了有点疑惑例如为什么render layout后分了两条分支?

0 回复 2019-09-18

千学不如一看，千看不如一练