10 页面渲染机制（二）之渲染树的构建及渲染过程

更新时间：2019-06-21 17:26:09

> 每个人的生命都是一只小船，理想是小船的风帆。
>
> ——张海迪

上一节讲了 DOM 树和 CSSOM 树的构建，为页面的渲染准备好了材料。这一节我们来介绍渲染树的构建以及后续的渲染过程。这一节我们还会介绍一下在渲染过程中会出现的重排和重绘这两种特殊情况。

渲染树（Render Tree）的构建

在 DOM 树和 CSSOM 树都渲染完成以后，就会进入渲染树的构建工作。渲染树就是对 DOM 树和 CSSOM 树的结合，得到一个可以知道每个节点会应用什么样式的数据结构。这个结合的过程大体上是遍历整个 DOM 树，然后在 CSSOM 树里查询到匹配的样式。但在不同浏览器里这个过程也不太一样，在 Chrome 里会在每个节点上使用 attach() 方法，把 CSSOM 树的节点挂在 DOM 树上作为渲染树。然而在 Firefox 里，会单独构造一个新的结构，用来连接 DOM 树和 CSSOM 树的映射关系。它们内部的实现方式有所不同，但它们构造出来的渲染树是有很多共同点的。渲染树会有以下的特点：

1. 渲染树的根是 HTML 节点。

在 Google Web Fundamentals 这个文档中，渲染树的根节点是 body，但实际上 HTML 节点上的样式也是可以显示在页面上的，所以我觉得渲染树也应该是由 HTML 节点开始，但是 head 标签里的内容和显示没有关系，所以渲染树中可以没有 head 标签的部分。

2. 渲染树和 DOM 树 的结构并不完全一致。

渲染树里会把所有不可见的元素忽略掉，所以如果是 DOM 树中的节点有 “display: none;” 属性的节点以及它的子节点，最终都不会出现在渲染树中。但是具有 “visibility: hidden;” 样式的元素会出现在渲染树中，因为具有这个样式的元素是需要占位的，只不过不需要显示出来。

之前的例子中，我们可以看到 .header 元素对应的属性里包含 “display:none;” 样式的，所以它最终会被渲染树忽略。

3. 样式优先级关系。

同一个 DOM 节点可能会匹配到多个 CSSOM 节点，而最终的表现由哪个 CSS 规则来确定，就是样式优先级的问题了。当一个 DOM 元素受到多条样式控制的时候，样式的优先级顺序应该是

```
内联样式 > ID选择器 > 类选择器 > 标签选择器 > 通用选择器 > 继承样式 > 浏览器默认样式
```

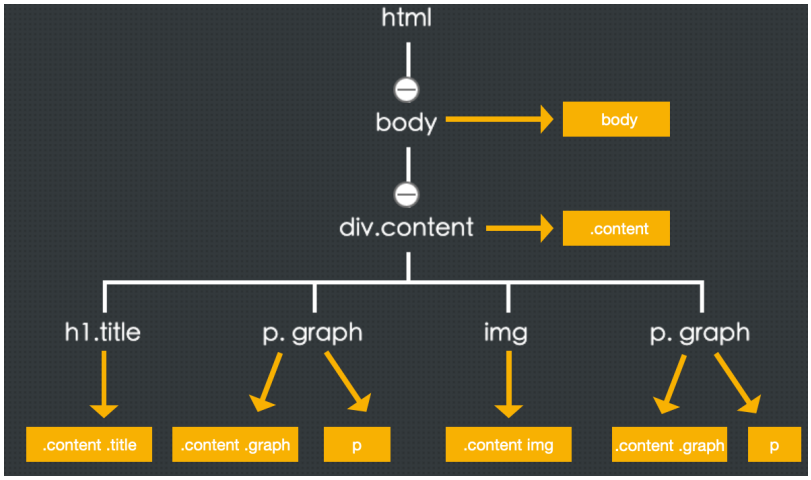
在有同类型的浏览器的时候，还有一套计算方法，给不同的选择器都赋了一个权重值。当考察优先级的时候，直接用公式计算整条选择器的权重作为该样式的优先级。比如：

- 内联样式的权重是1000。
- ID 选择器里样式的权重是100。
- 类选择器、属性选择器和伪类选择器里样式的权重是10。
- 标签选择器里样式的权重是1。
- 通用选择器直接忽略。

那么在计算的时候，假设一个选择器里有 a 个权重值是100的、b 个权重值是10的和 c 个权重值是1的选择器。那么这个选择器的权重值就 a*100 + b*10 + c。

> **Tips：**
> 这个计算公式的形式就是这样，但有几点要注意：
> 1、这个计算模型仅供理解样式优先级关系，不能代表浏览器里真实的计算方法。
> 2、权重值的计算不能越级，比如选择器A 只有1个 ID 选择器，权重就是100；选择器B 用了20个类选择器，权重值是200。这个时候如果两个选择器对应的样式作用在同一个 DOM 节点上，那么还是选择器A会生效，因为它的选择器级别更高。
> 3、如果两个选择器 A 和 B 是同级别选择器，并且最终计算的权重值也相同，那么这两个选择器谁在后面谁优先级高。

参考上面渲染树的特点，由之前的 DOM 树和 CSSOM 树就可以构建出来一棵如下图的渲染树：



这棵渲染树中的节点是和 DOM 树中的节点对应的，而黄色框部分的内容就是从 CSSOM 树中查找出来的。从上图里可以看出来，这棵渲染树中去掉了 head 标签里的内容，也去掉了有 “display:none;” 样式的 .header 元素及其子元素。而渲染树里面的 p.graph 元素会同时对应两条样式。这就是一棵由 DOM 树和 CSSOM 树结合而来的渲染树。

> **Tips:**
> 1、渲染树的构建过程中，会遍历 DOM 树中的可见节点，然后在 CSSOM 树中查找每个节点匹配的样式，最后通过组合这些可见节点以及和它们相匹配的样式就可以构建出一棵渲染树（带有“visibility: hidden;”属性的元素不可见，但会在页面中占位，所以会出现在渲染树中）。这里在查找的时候，出于效率的考虑，会从 CSSOM 树的叶子节点开始查找，对应在 CSS 选择器上也就是从选择器的最右侧向左查找。这就是在 2-4 讲解后代选择器时提到使用“.page .article p”会有效率问题的原因，这个选择器中会最先在 CSSOM 的所有叶子节点里查找 p 标签，这种标签类的选择器会很多且没有索引，会造成查找效率低下。不建议使用标签选择器和通配选择器的原因也是这个。
>
> 2、在 2-4 讲解兄弟选择器的时候，说过兄弟选择器为什么只能向后寻找兄弟元素。这是因为在生成渲染树的时候会遍历 DOM 节点来生成渲染树的节点，当遇到兄弟选择器的时候，它前面的兄弟元素在渲染树上的节点已经生成完毕，而它后面的兄弟节点还没有生成。这时候如果再回头去改前面兄弟节点的那就麻烦了，整个遍历的规则都要变化，而后面兄弟节点在生成的时候把兄弟选择器的影响加进去就可以。所以这就是为什么兄弟选择器只能向后寻找兄弟元素，而没提供向前寻找的方式。

布局（Layout）

经过上面的步骤，生成了一棵渲染树，这棵树就是我们展示页面的关键。通过计算渲染树上每个节点的样式，就能得出来每个元素所占空间的大小和位置。当有了所有元素的大小和位置后，就可以在浏览器的页面区域里去绘制元素的边框了。这个过程就是布局，英文中会用 Layout 这个词来描述。

绘制（Paint）

经过布局，每个元素的位置和大小就有了，经过最后绘制这一步，就可以把样式可视化的展现在屏幕上了。在绘制的过程中，浏览器会调用图形处理器，逐层逐块的把所有计算好位置和样式的元素都绘制出来。

当绘制工作结束，我们的页面就终于展示在浏览器上了。

重排（Reflow）与重绘（Repaint）

最后还要讲两个概念，重排（Reflow）与重绘（Repaint）。渲染树是动态构建的，DOM 节点和 CSS 节点的改动都可能会造成渲染树的重建。渲染树的改动就会造成重排或者重绘，下面我们来介绍这两个概念，以及它们都是在什么情况下会被触发。

1、重排。

当我们在 DOM 树中新增、删除了元素，或者是改变了某些元素的大小、位置、布局方式等，在这个时候渲染树里这个有改动的节点和它会影响的节点，都要重新计算。在改动发生时，要重新经历 DOM 的改动、 CSSOM 树的构建、渲染树的构建、布局和绘制整个流程，这个过程就叫做“重排”，也有的叫做“回流”。

以刚才代码中隐藏的 .header 元素为例，假如我们通过 JS 把它的 “display:none;” 属性去掉，那么它就要显示在屏幕中。这种情况下会经历下面的过程

- DOM 树没有变化。
- CSSOM 树中这个样式节点里的 display 属性没有了。
- 渲染树中的变化就比较大了，因为之前 “display:none;” 的元素没有出现在渲染树中，所以这个时候渲染树就要再重新结合 DOM 树和 CSSOM 树，把 .header 这个元素和它的子元素都加到渲染树中来。
- 布局的过程也会有不小的花销，需要给新加进来的 .header 元素找到位置，然后再把后面影响到的所有元素的大小和位置都重新计算一遍。这样得到一个新的布局值。
- 最后就是按着新的布局，把 .header 和受它影响的元素都重新绘制一遍，这个页面的改动就生效了。

2、重绘

重绘是当我们改变元素的字体颜色、背景色等外观元素的时候，并不会改变它的大小和位置，也不会影响到其他元素的布局，这个时候就没有必要再重新构建渲染树了。浏览器会直接对元素的样式重新绘制，这个过程就叫做“重绘”。

我们还以上面的代码为例，假如我们想对 .content 元素加一个 “color: black;” 的样式。这个时候就会经历以下的过程：

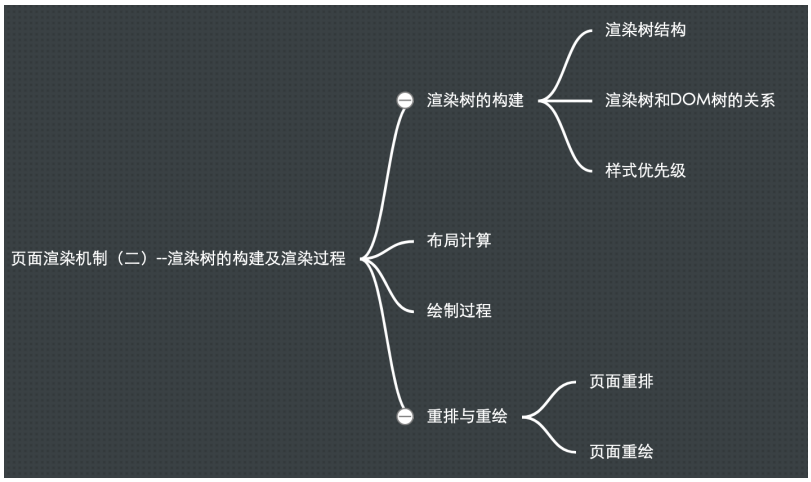
- DOM 树没有变化。
- CSSOM 树中 .content 对应的节点加入一条 “color: black;” 的样式。
- Color 属性的改变不会造成渲染树结构的变化，所以会在现有的渲染树中找出 .content 元素，给它加上 “color: black;” 的样式。
- 因为存在样式继承机制，所以浏览器还会找到 .content 元素的子元素，如果有可以继承的节点，那么也要给这些节点加上 “color: black;” 的样式，这个例子中就会在 h1.title、p.graph 元素上都加入 “color: black;” 的样式。
- 不涉及位置变动，布局过程直接忽略。
- 对 .content 元素及其子元素占用的块重新绘制。

这就是重排和重绘的概念，相对来说重排操作的消耗会比较大，所以在操作中尽量少的造成页面的重排。

> **Tips:**
> 为了减少重排，可以通过几种方式优化：
> 1、不要逐项的更改样式，可以把需要改动的样式收集到一块，用一次操作改变。
> 2、可以使用 class 的变动代替样式的改变，也能达到第1条的效果。
> 3、不要循环操作 DOM，循环的结果也要缓存起来，最后用一次操作来完成。
> 4、需要频繁改动的元素（比如动画）尽量使用绝对定位，脱离文档流的元素会减少对后面元素的影响。
> 5、在条件允许的情况下尽量使用 CSS3 动画，它可以调用 GPU 执行渲染。

小结

这一节我们讲了渲染树的构建、布局和绘制这些过程，在最后还介绍了重排和重绘的概念。这两节渲染的过程会稍微难理解一些，但要知道实际的渲染过程比这还要复杂的多，这里讲的还只是一个模型。本节的内容结构如下：



页面的渲染过程也是面试时比较常见的问题，另外面试官还会通过其他问题来测试面试者对渲染过程的理解。比如：

- 为什么 CSS 要放在 HTML 中靠前的 head 标签中，而 JS 最好放在页面的最后。这么放对首屏时间和整个页面的加载时间都有什么影响？
- 为什么不推荐使用通配选择器和标签选择器，它们的效率为什么会低？
- 为什么会有样式优先级的问题出现？
- 减少重排的方式有哪些？为什么要减少重排？
- “display: none;” 和 “visibility: hidden;” 有什么区别？
- …

面试时候遇到的这些问题，其实都是可以从页面渲染机制的角度来回答的，如果能从原理上给出答案一定是加分的。小伙伴们可以反复的去理解下这些知识点，这部分内容死记硬背是没有用的，一定要深入地理解。有什么问题也欢迎同学们来评论区留言讨论，我们一起思考、学习、进步。

09 页面渲染机制（一）之 DOM 树和 CSSOM 树的构建
11 对比分析 CSS屏幕适配方案

精选留言 3

欢迎在这里发表留言，作者筛选后可公开显示

Jonny邹

也就是说 样式的渲染是从cssom树反向遍历的吗？

0 回复 2019-06-24

作者 **Rosen** 回复 **Jonny邹**

是的，从css的根节点开始查起的话，不符合的路径能在第一时间就排除，可以加快查询的速度

回复 2019-06-26 15:23:41

Jonny邹 回复
Jonny邹

原来如此

回复 2019-06-26 16:17:04

王军校0207

写的非常好，对于我这新手来说，真的是醍醐灌顶啊

3 回复 2019-06-22

慕斯2162977

太有帮助了，最近正在想优化页面加载，多谢了老师

2 回复 2019-06-21