12 | 浏览器：一个浏览器是如何工作的（阶段三）

winter 2019-02-14

00:00 11:00

讲述：winter 大小：10.08M

大家好，我是 winter。

在上一节课中，我已经讲了浏览器的 DOM 构建过程，但是这个构建的 DOM，实际上信息是不全的，它只有节点和属性，不包含任何的样式信息。

我们这一节课就来讲讲：浏览器是如何把 CSS 规则应用到节点上，并给这棵朴素的 DOM 树添加上 CSS 属性的。

首先 CSS 选择器这个名称，可能会给你带来一定的误解，觉得好像 CSS 规则是 DOM 树构建好了以后，再进行选择并给它添加样式的。实际上，这个过程并不是这样的。

我们回忆一下我们在浏览器第一节课讲的内容，浏览器会尽量流式处理整个过程。我们上一节课构建 DOM 的过程是：从父到子，从先到后，一个一个节点构造，并且挂载到 DOM 树上的，那么这个过程中，我们是否能同步把 CSS 属性计算出来呢？

答案是肯定的。

在这个过程中，我们依次拿到上一步构造好的元素，去检查它匹配到了哪些规则，再根据规则的优先级，做覆盖和调整。所以，从这个角度看，所谓的选择器，应该被理解成“匹配器”才更合适。

我在 CSS 语法部分，已经总结了选择器的各种符号，这里再把它列出来，我们回顾一下：

- 空格: 后代，选中它的子节点和所有子节点的后代节点。
- >: 子代，选中它的子节点。
- +：直接后继选择器，选中它的下一个相邻节点。
- ~：后继，选中它之后所有的相邻节点。
- ||：列，选中表格中的一列。

关于选择器的知识，我会在 CSS 的部分继续讲解。这里我们主要介绍浏览器是如何实现这些规则的。

不知道你有没有发现，这里的选择器有个特点，那就是选择器的出现顺序，必定跟构建 DOM 树的顺序一致。这是一个 CSS 设计的原则，即保证选择器在 DOM 树构建到当前节点时，已经可以准确判断是否匹配，不需要后续节点信息。

也就是说，未来也不可能会出现“父元素选择器”这种东西，因为父元素选择器要求根据当前节点的子节点，来判断当前节点是否被选中，而父节点会先于子节点构建。

理解了 CSS 构建的大概过程，我们下面来看看具体的操作。

首先，我们必须把 CSS 规则做一下处理。作为一门语言，CSS 需要先经过词法分析和语法分析，变成计算机能够理解的结构。

这部分具体的做法属于编译原理的内容，这里就不做赘述了。我们这里假设 CSS 已经被解析成了一棵可用的抽象语法树。

我们在之前的 CSS 课程中已经介绍过 compound-selector 的概念，一个 compound-selector 是检查一个元素的规则，而一个复合型选择器，则是由数个 compound-selector 通过前面讲的符号连接起来的。

后代选择器 “空格”

我们先来分析一下后代选择器，我们来一起看一个例子：

复制代码

```
a#b .cls {
    width: 100px;
}

```

可以把一个 CSS 选择器按照 compound-selector 来拆成数段，每当满足一段条件的时候，就前进一段。

比如，在上面的例子中，当我们找到了匹配 a#b 的元素时，我们才会开始检查它所有的子代是否匹配 .cls。

除了前进一段的情况，我们还需要处理后退的情况，比如，我们这样一段代码：

复制代码

```
<a id=b>
    <span>1<span>
    <span class=cls>2<span>
</a>
<span class=cls>3<span>

```

当遇到 时，必须使得规则 a#b .cls 回退一步，这样第三个 span 才不会被选中。后代选择器的作用范围是父节点的所有子节点，因此规则是在匹配到本标签的结束标签时回退。

后继选择器“ ~ ”

接下来我们看下后继选择器，跟后代选择器不同的地方是，后继选择器只作用于一层，我们来看一个例子：

复制代码

```
.cls~* {
    border:solid 1px green;
}
<div>
<span>1<span>
<span class=cls>2<span>
<span>
    3
    <span>4</span>
<span>
<span>5</span>
</div>

```

这里 .cls 选中了 span 2 然后 span 3 是它的后继，但是 span 3 的子节点 span 4 并不应该被选中，而 span 5 也是它的后继，因此应该被选中。

按照 DOM 树的构造顺序，4 在 3 和 5 中间，我们就没有办法像前面讲的后代选择器一样通过激活或者关闭规则来实现匹配。

但是这里有个非常方便的思路，就是给选择器的激活，带上一个条件：父元素。

注意，这里后继选择器，当前半段的 .cls 匹配成功时，后续 * 所匹配的所有元素的父元素都已经确定了（后继节点和当前节点父元素相同是充分必要条件）。在我们的例子中，那个 div 就是后继节点的父元素。

子代选择器“ >”

我们继续看，子代选择器是如何实现的。

实际上，有了前面讲的父元素这个约束思路，我们很容易实现子代选择器。区别仅仅是拿当前节点作为父元素，还是拿当前节点的父元素作为父元素。

复制代码

```
div>.cls {
    border:solid 1px green;
}
<div>
<span>1<span>
<span class=cls>2<span>
<span>
    3
    <span>4</span>
<span>
<span>5</span>
</div>

```

我们看这段代码，当 DOM 树构造到 div 时，匹配了 CSS 规则的第一段，因为是子代选择器，我们激活后面的 .cls 选择条件，并且指定父元素必须是当前 div。于是后续的构建 DOM 树构建过程中，span 2 就被选中了。

直接后继选择器“ +”

直接后继选择器的实现是上述中最为简单的了，因为它只对唯一一个元素生效，所以不需要像前面几种一样反复激活和关闭规则。

一个最简单的思路是，我们可以把它当作检查元素自身的选择器来处理。即我们把 #id+.cls 都当做检查某一个元素的选择器。

另外的一种思路是：给后继选择器加上一个 flag，使它匹配一次后失效。你可以尝试一下，告诉我结果。

列选择器“ || ”

列选择器比较特别，它是专门针对表格的选择器，跟表格的模型建立相关，我们这里不详细讲了。

其它

我们不要忘记，CSS 选择器还支持逗号分隔，表示“或”的关系。这里最简单的实现是把逗号视为两条规则的一种简易写法。

比如

复制代码

```
a#b, .cls {

}

```

我们当作两条规则来处理：

复制代码

```
a#b {

}

```

复制代码

```
.cls {

}

```

还有一个情况，就是选择器可能有重合，这样，我们可以使用树形结构来进行一些合并，来提高效率：

复制代码

```
#a .cls {

}

#a span {

}
#a>span {

}

```

这里实际上可以把选择器构造成一棵树：

需要注意的是，这里的树，必须要带上连接符。

结语

这一节我们讲解了 CSS 计算的过程。CSS 计算是把 CSS 规则应用到 DOM 树上，为 DOM 结构添加显示相关属性的过程。在这一节中，我们主要介绍了选择器的几种复合结构应该如何实现。

在这一步骤之后，我们得到了一棵带有 CSS 属性的树，为我们后续继续显式打下了基础。

最后留一个问题，你认为 CSS 语法解析成什么结构，最适合我们进行 CSS 计算。