14 | 浏览器：一个浏览器是如何工作的？（阶段五）

winter 2019-02-19

你好，我是 winter。我们的浏览器系列已经进行到最后一篇。

在之前的几篇文章中，我们已经经历了把 URL 变成字符流，把字符流变成词（token）流，把词（token）流构造成 DOM 树，把不含样式信息的 DOM 树应用 CSS 规则，变成包含样式信息的 DOM 树，并且根据样式信息，计算了每个元素的位置和大小。

那么，我们最后的步骤，就是根据这些样式信息和大小信息，为每个元素在内存中渲染它的图形，并且把它绘制到对应的位置。

渲染

首先我们来谈谈渲染这个词，渲染也是个外来词，它是英文词 render 的翻译，render 这个词在英文里面，有“导致”“变成”的意思，也有“粉刷墙壁”的意思。

在计算机图形学领域里，英文 render 这个词是一个简写，它是特指把模型变成位图的过程。我们把 render 翻译成“渲染”，是个非常有意思的翻译，中文里“渲染”这个词是一种绘画技法，是指沾清水把墨涂开的意思。

所以，render 翻译成“渲染”，我认为是非常高明的，对 render 这个过程，用国画的渲染手法来概括，是颇有神似的。

我们现在的一些框架，也会把“从数据变成 HTML 代码的过程”称为 render，其实我觉得这是非常具有误导性的，我个人是非常不喜欢这种命名方式，当然了，所谓“文无第一”，在自然语言的范围里，我们很难彻底否定这种用法的合理性。

不过，在本篇文章中，我们可以约定一下，本文中出现的“渲染”一词，统一指的是它在图形学的意义，也就是把模型变成位图的过程。

这里的位图就是在内存里建立一张二维表格，把一张图片的每个像素对应的颜色保存进去（位图信息也是 DOM 树中占据浏览器内存最多的信息，我们在做内存占用优化时，主要就是考虑这一部分）。

浏览器中渲染这个过程，就是把每一个元素对应的盒变成位图。这里的元素包括 HTML 元素和伪元素，一个元素可能对应多个盒（比如 inline 元素，可能会分成多行）。每一个盒对应着一张位图。

这个渲染过程是非常复杂的，但是总体来说，可以分成两个大类：图形和文字。

盒的背景、边框、SVG 元素、阴影等特性，都是需要绘制的图形类。这就像我们实现 HTTP 协议必须要基于 TCP 库一样，这一部分，我们需要一个底层库来支持。

一般的操作系统会提供一个底层库，比如在 Android 中，有大名鼎鼎的 Skia，而 Windows 平台则有 GDI，一般的浏览器会做一个兼容层来处理掉平台差异。

这些盒的特性如何绘制，每一个都有对应的标准规定，而每一个的实现都可以作为一个独立的课题来研究，当年圆角 + 虚线边框，可是难倒了各个浏览器的工程师。考虑到这些知识互相都比较独立，对前端工程师来说也不是特别重要的细节，我们这里就不详细探究了。

盒中的文字，也需要用底层库来支持，叫做字体库。字体库提供读取字体文件的基本能力，它能根据字符的码点抽取出字形。

字形分为像素字形和矢量字形两种。通常的字体，会在 6px 8px 等小尺寸提供像素字形，比较大的尺寸则提供矢量字形。矢量字形本身就需要经过渲染才能继续渲染到元素的位图上去。目前最常用的字体库是 Freetype，这是一个 C++ 编写的开源的字体库。

在最普遍的情况下，渲染过程生成的位图尺寸跟它在上一步排版时占据的尺寸相同。

但是理想和现实是有差距的，很多属性会影响渲染位图的大小，比如阴影，它可能非常巨大，或者渲染到非常遥远的位置，所以为了优化，浏览器实际的实现中会把阴影作为一个独立的盒来处理。

注意，我们这里讲的渲染过程，是不会把子元素绘制到渲染的位图上的，这样，当父子元素的相对位置发生变化时，可以保证渲染的结果能够最大程度被缓存，减少重新渲染。

合成

合成是英文术语 compositing 的翻译，这个过程实际上是一个性能考量，它并非实现浏览器的必要一环。

我们上一小节中讲到，渲染过程不会把子元素渲染到位图上面，合成的过程，就是为一些元素创建一个“合成后的位图”（我们把它称为合成层），把一部分子元素渲染到合成的位图上面。

看到这句话，我想你一定会问问题，到底是为哪些元素创建合成后的位图，把哪些子元素渲染到合成的位图上面呢？

这就是我们要讲的合成的策略。我们前面讲了，合成是一个性能考量，那么合成的目标就是提高性能，根据这个目标，我们建立的原则就是最大限度减少绘制次数原则。

我们举一个极端的例子。如果我们把所有元素都进行合成，比如我们为根元素 html 创建一个合成后的位图，把所有子元素都进行合成，那么会发生什么呢？

那就是，一旦我们用 JavaScript 或者别的什么方式，改变了任何一个 CSS 属性，这份合成后的位图就失效了，我们需要重新绘制所有的元素。

那么如果我们所有的元素都不合成，会怎样呢？结果就是，相当于每次我们都必须要重新绘制所有的元素，这也不是对性能友好的选择。

那么好的合成策略是什么呢，好的合成策略是“猜测”可能变化的元素，把它排除到合成之外。

我们来举个例子：