

23 | 渲染流水线：CSS如何影响首次加载时的白屏时间？

2019-09-26 李兵

浏览器工作原理与实践

[进入课程 >](#)



讲述：李兵

时长 09:50 大小 13.51M



在[上一篇文章](#)中我们详细介绍了 DOM 的生成过程，并结合具体例子分析了 JavaScript 是如何阻塞 DOM 生成的。那本文我们就继续深入聊聊渲染流水线中的 CSS。因为 CSS 是页面中非常重要的资源，它决定了页面最终显示出来的效果，并影响着用户对整个网站的第一体验。所以，搞清楚浏览器中的 CSS 是怎么工作的很有必要，只有理解了 CSS 是如何工作的，你才能更加深刻地理解如何去优化页面。

本文我们先站在渲染流水线的视角来介绍 CSS 是如何工作的，然后通过 CSS 的工作流程来分析性能瓶颈，最后再来讨论如何减少首次加载时的白屏时间。

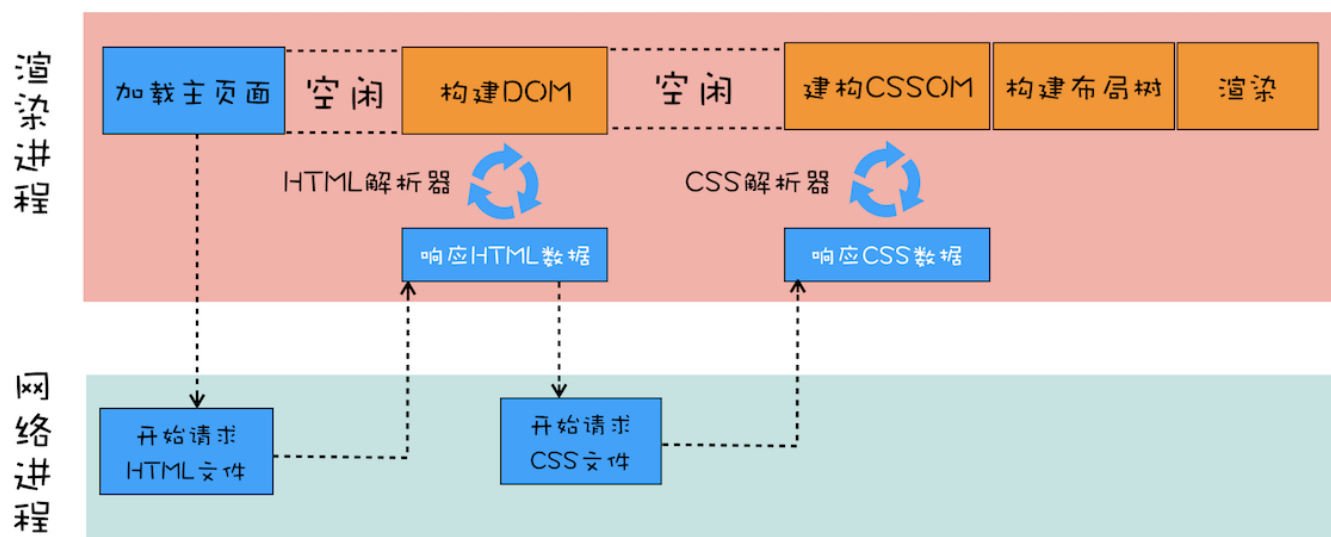
渲染流水线视角下的 CSS

我们先结合下面代码来看看最简单的渲染流程：

```
1 //theme.css
2 div{
3     color : coral;
4     background-color:black
5 }
```

```
1 <html>
2 <head>
3     <link href="theme.css" rel="stylesheet">
4 </head>
5 <body>
6     <div>geekbang com</div>
7 </body>
8 </html>
```

这两段代码分别由 CSS 文件和 HTML 文件构成，我们来分析下打开这段 HTML 文件时的渲染流水线，你可以先参考下面这张渲染流水线示意图：



含有 CSS 的页面渲染流水线

下面我们结合上图来分析这个页面文件的渲染流水线。

首先是发起主页面的请求，这个发起请求方可能是渲染进程，也有可能是浏览器进程，发起的请求被送到网络进程中去执行。网络进程接收到返回的 HTML 数据之后，将其发送给渲染进程，渲染进程会解析 HTML 数据并构建 DOM。这里你需要特别注意下，请求 HTML 数据和构建 DOM 中间有一段空闲时间，这个空闲时间有可能成为页面渲染的瓶颈。

[上一篇文章](#)中我们提到过，当渲染进程接收 HTML 文件字节流时，会先开启一个**预解析线程**，如果遇到 JavaScript 文件或者 CSS 文件，那么预解析线程会提前下载这些数据。对于上面的代码，预解析线程会解析出来一个外部的 theme.css 文件，并发起 theme.css 的下载。这里也有一个空闲时间需要你注意一下，就是在 DOM 构建结束之后、theme.css 文件还未下载完成的这段时间内，渲染流水线无事可做，因为下一步是合成布局树，而合成布局树需要 CSSOM 和 DOM，所以这里需要等待 CSS 加载结束并解析成 CSSOM。

那渲染流水线为什么需要 CSSOM 呢？

和 HTML 一样，渲染引擎也是无法直接理解 CSS 文件内容的，所以需要将其解析成渲染引擎能够理解的结构，这个结构就是 CSSOM。和 DOM 一样，CSSOM 也具有两个作用，**第一个是提供给 JavaScript 操作样式表的能力，第二个是为布局树的合成提供基础的样式信息**。这个 CSSOM 体现在 DOM 中就是 `document.styleSheets`。具体结构你可以去查阅相关资料，这里我就不过多介绍了，你知道 CSSOM 的两个作用是怎样的就行了。

有了 DOM 和 CSSOM，接下来就可以合成布局树了，我们在前面 [《05 | 渲染流程（上）：HTML、CSS 和 JavaScript 文件，是如何变成页面的？》](#) 这篇文章中讲解过布局树的构造过程，这里咱们再简单回顾下。等 DOM 和 CSSOM 都构建好之后，渲染引擎就会构造布局树。布局树的结构基本上就是复制 DOM 树的结构，不同之处在于 DOM 树中那些不需要显示的元素会被过滤掉，如 `display:none` 属性的元素、`head` 标签、`script` 标签等。复制好基本的布局树结构之后，渲染引擎会为对应的 DOM 元素选择对应的样式信息，这个过程就是**样式计算**。样式计算完成之后，渲染引擎还需要计算布局树中每个元素对应的几何位置，这个过程就是**计算布局**。通过样式计算和计算布局就完成了最终布局树的构建。再之后，就该进行后续的绘制操作了。

这就是在渲染过程中涉及到 CSS 的一些主要流程。

了解了这些之后，我们再来看看稍微复杂一点场景，还是看下面这段 HTML 代码：

```

1 //theme.css
2 div{
3     color : coral;
4     background-color:black
5 }

```

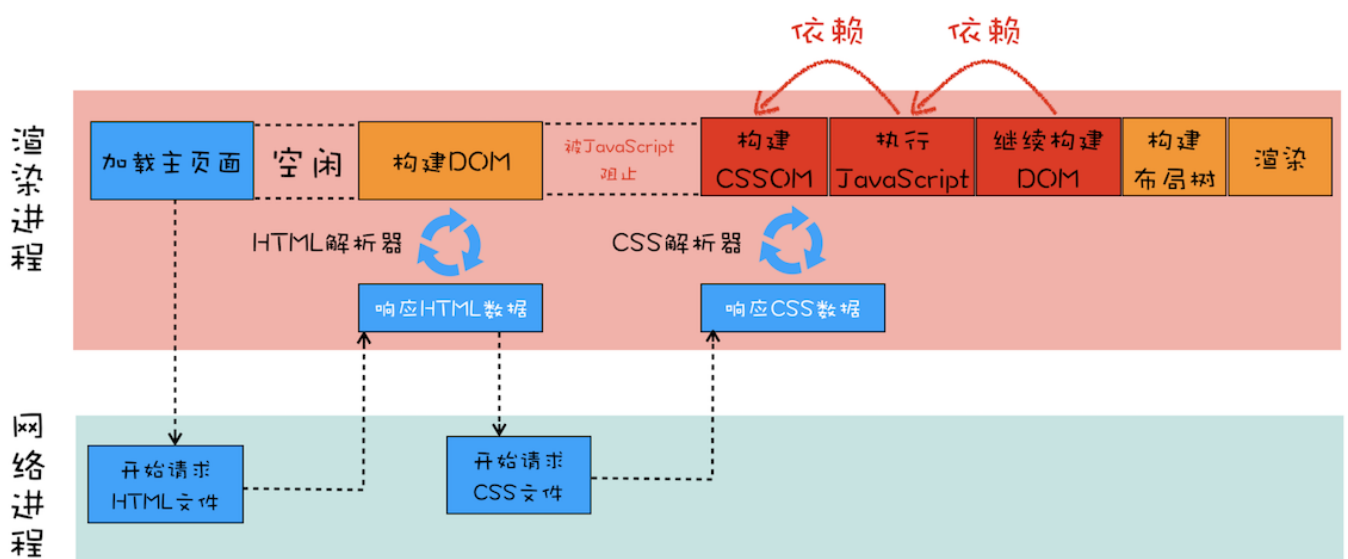
复制代码

```

1 <html>
2 <head>
3     <link href="theme.css" rel="stylesheet">
4 </head>
5 <body>
6     <div>geekbang com</div>
7     <script>
8         console.log('time.geekbang.org')
9     </script>
10    <div>geekbang com</div>
11 </body>
12 </html>

```

这段代码是我在开头代码的基础之上做了一点小修改，在 body 标签内部加了一个简单的 JavaScript。有了 JavaScript，渲染流水线就有点不一样了，可以参考下面这张渲染流水线图：




含有 JavaScript 和 CSS 的页面渲染流水线


那我们就结合这张图来分析含有外部 CSS 文件和 JavaScript 代码的页面渲染流水线，[上篇文章](#)中我们提到过在解析 DOM 的过程中，如果遇到了 JavaScript 脚本，那么需要先暂停 DOM 解析去执行 JavaScript，因为 JavaScript 有可能会修改当前状态下的 DOM。

不过在执行 JavaScript 脚本之前，如果页面中包含了外部 CSS 文件的引用，或者通过 style 标签内置了 CSS 内容，那么渲染引擎还需要将这些内容转换为 CSSOM，因为 JavaScript 有修改 CSSOM 的能力，所以在执行 JavaScript 之前，还需要依赖 CSSOM。也就是说 CSS 在部分情况下也会阻塞 DOM 的生成。


我们再来看看更加复杂一点的情况，如果在 body 中被包含的是 JavaScript 外部引用文件，Demo 代码如下所示：

 复制代码

```
1 //theme.css
2 div{
3     color : coral;
4     background-color:black
5 }
```

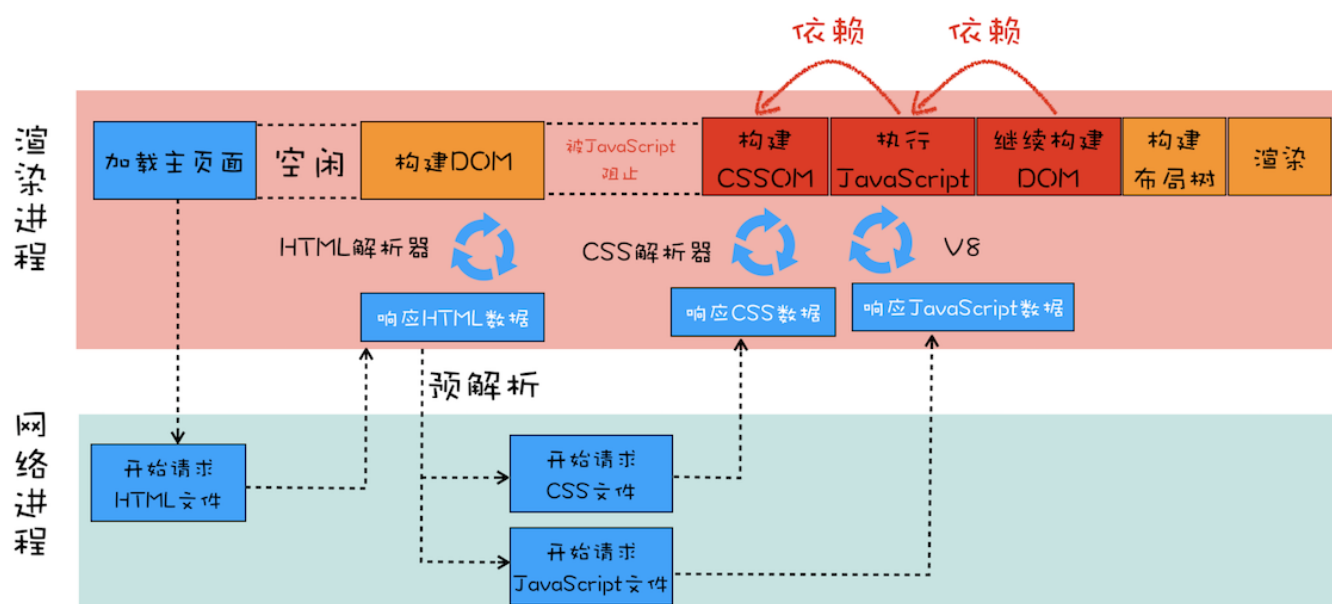
 复制代码

```
1 //foo.js
2 console.log('time.geekbang.org')
```

 复制代码

```
1 <html>
2 <head>
3     <link href="theme.css" rel="stylesheet">
4 </head>
5 <body>
6     <div>geekbang com</div>
7     <script src='foo.js'></script>
8     <div>geekbang com</div>
9 </body>
10 </html>
```


从上面代码可以看出来，HTML 文件中包含了 CSS 的外部引用和 JavaScript 外部文件，那它们的渲染流水线是怎样的呢？可参考下图：



含有 JavaScript 文件和 CSS 文件页面的渲染流水线

从图中可以看出来，在接收到 HTML 数据之后的预解析过程中，HTML 预解析器识别出来了有 CSS 文件和 JavaScript 文件需要下载，然后就同时发起这两个文件的下载请求，需要注意的是，这两个文件的下载过程是重叠的，所以下载时间按照最久的那个文件来算。

后面的流水线就和前面是一样的了，不管 CSS 文件和 JavaScript 文件谁先到达，都要先等到 CSS 文件下载完成并生成 CSSOM，然后再执行 JavaScript 脚本，最后再继续构建 DOM，构建布局树，绘制页面。

影响页面展示的因素以及优化策略

前面我们为什么要花这么多文字来分析渲染流水线呢？主要原因就是**渲染流水线影响到了首次页面展示的速度，而首次页面展示的速度又直接影响到了用户体验**，所以我们分析渲染流水线的目的就是为了找出一些影响到首屏展示的因素，然后再基于这些因素做一些针对性的调整。

那么接下来我们就来看看从发起 URL 请求开始，到首次显示页面的内容，在视觉上经历的三个阶段。

第一个阶段，等请求发出去之后，到提交数据阶段，这时页面展示出来的还是之前页面的内容。关于提交数据你可以参考前面[《04 | 导航流程：从输入 URL 到页面展示，这中间发生了什么？》](#)这篇文章。

第二个阶段，提交数据之后渲染进程会创建一个空白页面，我们通常把这段时间称为**解析白屏**，并等待 CSS 文件和 JavaScript 文件的加载完成，生成 CSSOM 和 DOM，然后合成布局树，最后还要经过一系列的步骤准备首次渲染。

第三个阶段，等首次渲染完成之后，就开始进入完整页面的生成阶段了，然后页面会一点点被绘制出来。

影响第一个阶段的因素主要是网络或者是服务器处理这块儿，前面文章中我们已经讲过了，这里我们就不再继续分析了。至于第三个阶段，我们会在后续文章中分析，所以这里也不做介绍了。

现在我们重点关注第二个阶段，这个阶段的主要问题是白屏时间，如果白屏时间过久，就会影响到用户体验。为了缩短白屏时间，我们来挨个分析这个阶段的主要任务，包括了解析 HTML、下载 CSS、下载 JavaScript、生成 CSSOM、执行 JavaScript、生成布局树、绘制页面一系列操作。

通常情况下的瓶颈主要体现在**下载 CSS 文件、下载 JavaScript 文件和执行 JavaScript**。

所以要想缩短白屏时长，可以有以下策略：

通过内联 JavaScript、内联 CSS 来移除这两种类型的文件下载，这样获取到 HTML 文件之后就可以直接开始渲染流程了。

但并不是所有的场合都适合内联，那么还可以尽量减少文件大小，比如通过 webpack 等工具移除一些不必要的注释，并压缩 JavaScript 文件。

还可以将一些不需要在解析 HTML 阶段使用的 JavaScript 标记上 sync 或者 defer。

对于大的 CSS 文件，可以通过媒体查询属性，将其拆分为多个不同用途的 CSS 文件，这样只有在特定的场景下才会加载特定的 CSS 文件。

通过以上策略就能缩短白屏展示的时长了，不过在实际项目中，总是存在各种各样的情况，这些策略并不能随心所欲地去引用，所以还需要结合实际情况来调整最佳方案。

总结


好了，今天就介绍到这里，下面我来总结下今天的内容。

我们首先介绍了 CSS 在渲染流水线中的位置，以及 CSS 是如何影响到渲染流程的；接下来我们通过渲染流水线分析了从发出请求到页面首次绘制的三个阶段；最后重点介绍了第二个白屏阶段以及优化该阶段的一些策略。

通过今天的内容我们可以知道虽然 JavaScript 和 CSS 给我们带来了极大的便利，不过也对页面的渲染带来了很多的限制，所以我们要关注资源加载速度，需要小心翼翼地处理各种资源之间的关联关系。

思考时间

今天留给你的思考题是：当你横屏方向拿着一个手机时，打开一个页面，观察下面几种资源的加载方式，你认为哪几种会阻塞页面渲染？为什么？

 复制代码

```
1 1:<script src="foo.js" type="text/javascript"></script>
2 2:<script defer src="foo.js" type="text/javascript"></script>
3 3:<script sync src="foo.js" type="text/javascript"></script>
4 4:<link rel="stylesheet" type="text/css" href="foo.css" />
5 5:<link rel="stylesheet" type="text/css" href="foo.css" media="screen"/>
6 6:<link rel="stylesheet" type="text/css" href="foo.css" media="print" />
7 7:<link rel="stylesheet" type="text/css" href="foo.css" media="orientation:landscape" /
8 8:<link rel="stylesheet" type="text/css" href="foo.css" media="orientation:portrait" />
```

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

浏览器工作原理与实践

>>> 透过浏览器看懂前端本质

李兵

前盛大创新院高级研究员



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 22 | DOM树：JavaScript是如何影响DOM树构建的？

下一篇 24 | 分层和合成机制：为什么CSS动画比JavaScript高效？

精选留言 (6)

写留言



Angus

2019-09-26

第1条：下载JavaScript文件并执行同步代码，会阻塞页面渲染

第2条：defer异步下载JavaScript文件，会在HTML解析完成之后执行，不会阻塞页面渲染

第3条：sync异步下载JavaScript文件，下载完成之后会立即执行，有可能会阻塞页面渲染

第4条：下载CSS文件，可能阻塞页面渲染

第5条：media属性用于区分设备，screen表示用于有屏幕的设备，无法用于打印机、3D...

展开 ∨



7



Geek_159de7

2019-09-27

老师，浏览器是不是有一个尽量快速渲染页面的机制呢？比如说，有时候打开一个网页很慢，后面慢慢地显示了样式错乱的页面，这明显是css没有加载构建完成，但是还是看到有

页面出来了，只是样式有点乱。老师，能解释一下浏览器在这个过程中的一些优化措施吗？

展开 ▾



1



Marvin

2019-09-27

因为cssom构造优先，所以先考虑css加载：没有media配置和配置了screen、landscape，3个css文件会命中，也就是4、5、7。js文件分加载和执行两部分，其中defer和async两种属性配置之后都会与dom树构造无关，也就是与渲染无关，其他的情况都会阻塞渲染，也就是1、3。总结：1、3、4、5、7会阻塞渲染。我查阅了h5规范，没有sync属性显式配置的定义，所以sync是无效属性，和1效果一致。网上有文章称，sync和defer同为...

展开 ▾



HB

2019-09-26

有个疑问，DOM tree 和 render tree 构建是同时进行的还是DOM tree和 cssom都构建完成后才构建render tree呢？看how browser works 有些没看清楚

展开 ▾



柒月

2019-09-26

1 4吧 一样的资源应该只加载一次吧

展开 ▾



mfist

2019-09-26

首先说下几个属性的含义：defer：异步下载js文件，等待domcontentloaded事件后解析执行，sync猜应该是async，异步下载js文件，下载完成后立即解析执行，media='screen' 电脑屏幕场景，media='print' 打印场景，orientation:landscape输出设备可见区域高度小于宽度；portrait 剩余场景。

阻塞页面渲染的有1 4 5 8。

展开 ▾



