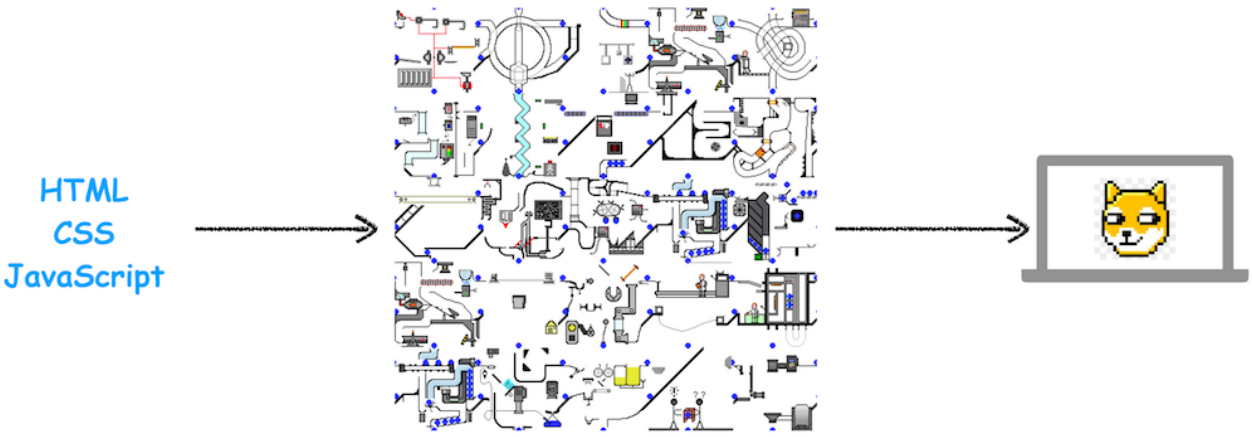
05-渲染流程（上）：HTML、CSS和JavaScript，是如何变成页面的?

在上一篇文章中我们介绍了导航相关的流程，那导航被提交后又会怎么样呢？就进入了渲染阶段。这个阶段很重要，了解其相关流程能让你“看透”页面是如何工作的，有了这些知识，你可以解决一系列相关的问题，比如能熟练使用开发者工具，因为能够理解开发者工具里面大部分项目的含义，能优化页面卡顿问题，使用JavaScript优化动画流程，通过优化样式表来防止强制同步布局，等等。

既然它的功能这么强大，那么今天，我们就来好好聊聊**渲染流程**。

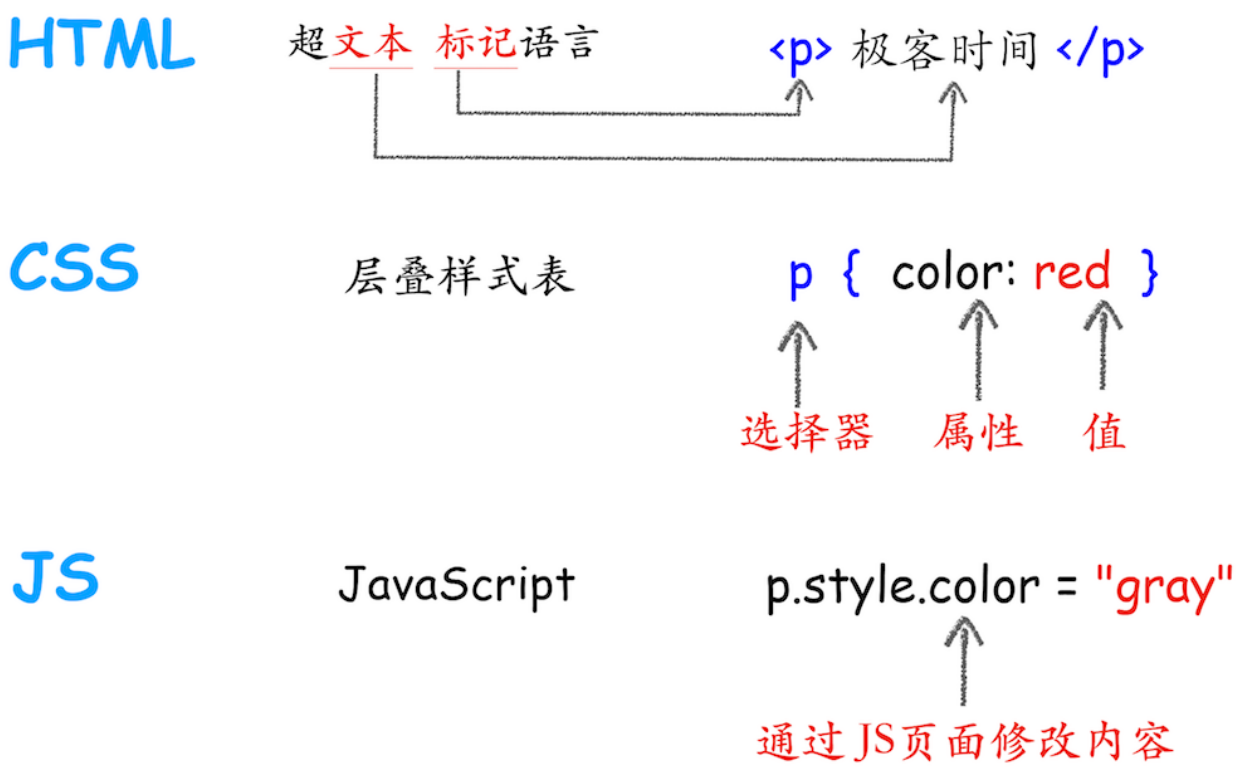
通常，我们编写好HTML、CSS、JavaScript等文件，经过浏览器就会显示出漂亮的页面（如下图所示），但是你知道它们是如何转化成页面的吗？这背后的原理，估计很多人都答不上来。



渲染流程示意图

从图中可以看出，左边输入的是HTML、CSS、JavaScript数据，这些数据经过中间渲染模块的处理，最终输出为屏幕上的像素。

这中间的**渲染模块**就是我们今天要讨论的主题。为了能更好地理解下文，你可以先结合下图快速抓住HTML、CSS和JavaScript的含义：



HTML、CSS和JavaScript关系图

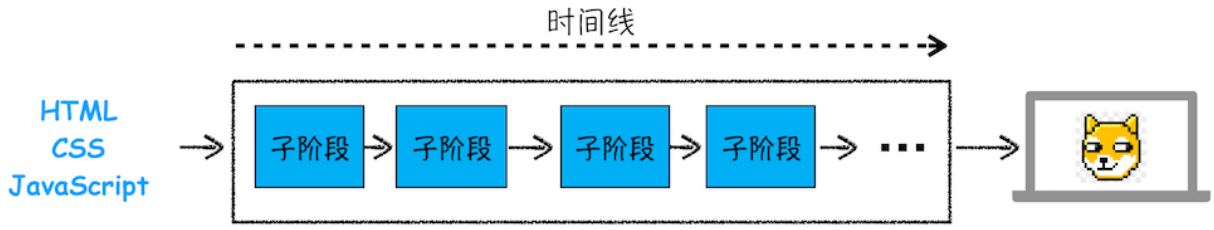
从上图可以看出，**HTML的内容是由标记和文本组成**。标记也称为**标签**，每个标签都有它自己的语意，浏览器会根据标签的语意来正确展示HTML内容。比如上面的<p>标签是告诉浏览器在这里的内容需要创建一个新段落，中间的文本就是段落中需要显示的内容。

如果需要改变HTML的字体颜色、大小等信息，就需要用到CSS。CSS又称为**层叠样式表，是由选择器和属性组成**，比如图中的p选择器，它会把HTML里面<p>标签的内容选择出来，然后再把选择器的属性值应用到<p>标签内容上。选择器里面有个color属性，它的值是red，这是告诉渲染引擎把<p>标签的内容显示为红色。

至于**JavaScript（简称为JS），使用它可以使网页的内容“动”起来**，比如上图中，可以通过JavaScript来修改CSS样式值，从而达到修改文本颜色的目的。

搞清楚HTML、CSS和JavaScript的含义后，那么接下来我们就正式开始分析渲染模块了。

由于渲染机制过于复杂，所以渲染模块在执行过程中会被划分为很多子阶段，输入的HTML经过这些子阶段，最后输出像素。我们把这样的一个处理流程叫做**渲染流水线**，其大致流程如下图所示：



渲染流水线示意图

按照渲染的时间顺序，流水线可分为如下几个子阶段：构建DOM树、样式计算、布局阶段、分层、绘制、分块、光栅化和合成。内容比较多，我会用两篇文章来为你详细讲解这各个子阶段。接下来，在介绍每个阶段的过程中，你应该重点关注以下三点内容：

开始每个子阶段都有其**输入的内容**；

然后每个子阶段有其**处理过程**；

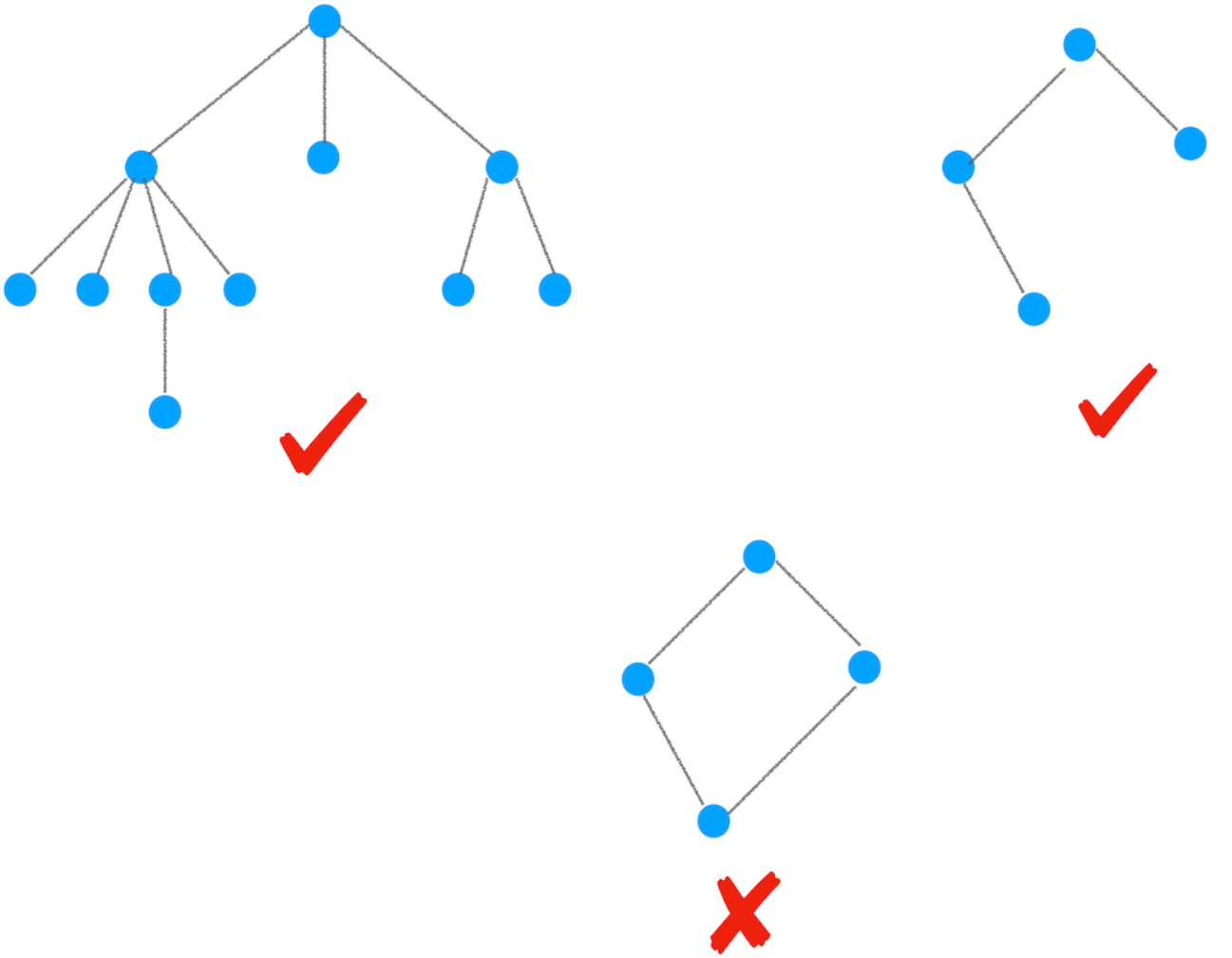
最终每个子阶段会生成**输出内容**。

理解了这三部分内容，能让你更加清晰地理解每个子阶段。

构建DOM树

为什么要构建DOM树呢？**这是因为浏览器无法直接理解和使用HTML，所以需要将HTML转换为浏览器能够理解的结构——DOM树。**

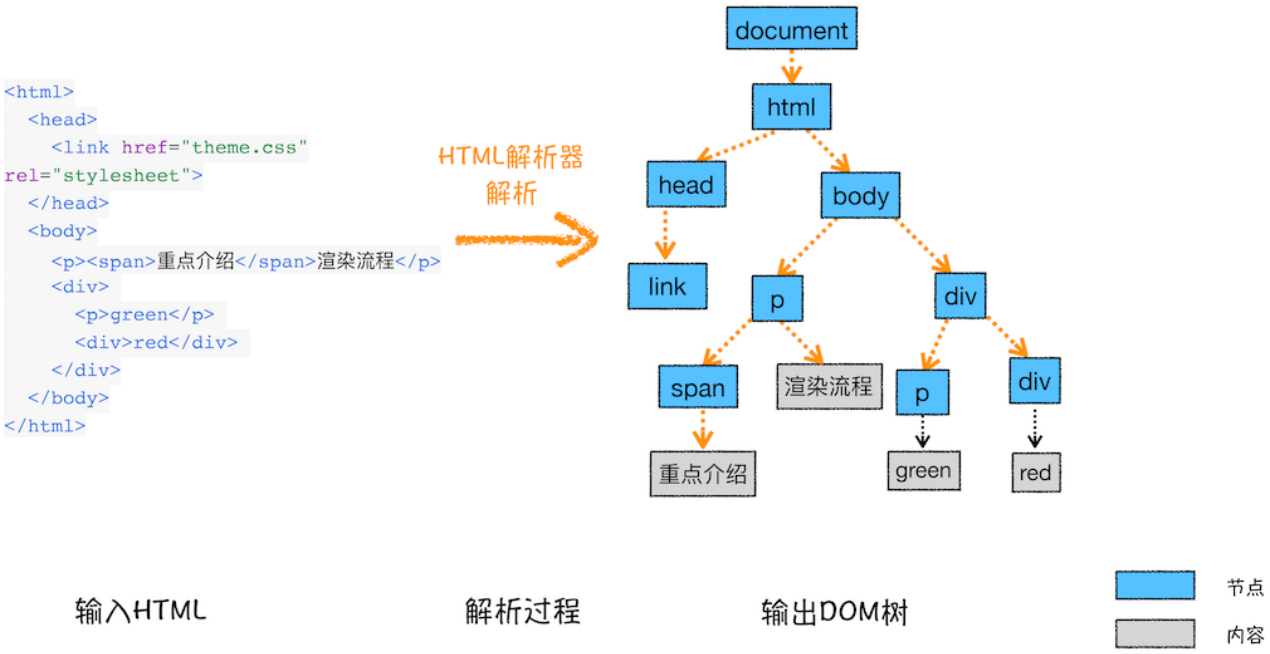
这里我们还需要简单介绍下什么是**树结构**，为了更直观地理解，你可以参考下面我画的几个树结构：



树结构示意图

从图中可以看出，树这种结构非常像我们现实生活中的“树”，其中每个点我们称为**节点**，相连的节点称为**父子节点**。树结构在浏览器中的应用还是比较多的，比如下面我们要介绍的渲染流程，就在频繁地使用树结构。

接下来咱们还是言归正传，来看看DOM树的构建过程，你可以参考下图：



DOM树构建过程示意图

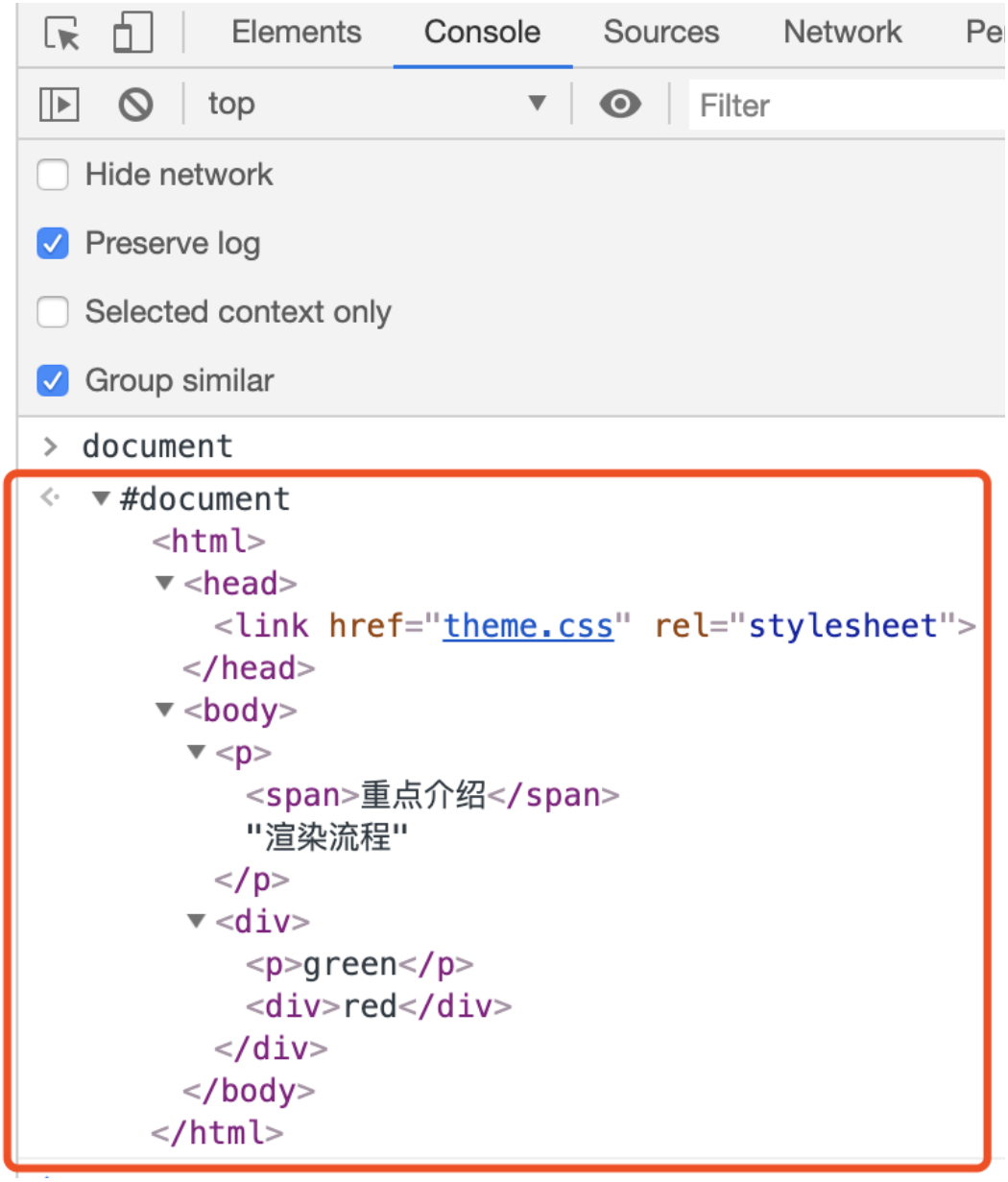
从图中可以看出，构建DOM树的**输入内容**是一个非常简单的HTML文件，然后经由HTML解析器解析，最终输出树状结构的DOM。

为了更加直观地理解DOM树，你可以打开Chrome的“开发者工具”，选择“Console”标签来打开控制台，然后在控制台里面输入“document”后回车，这样你就能看到一个完整的DOM树结构，如下图所示：

渲染流程

green

red



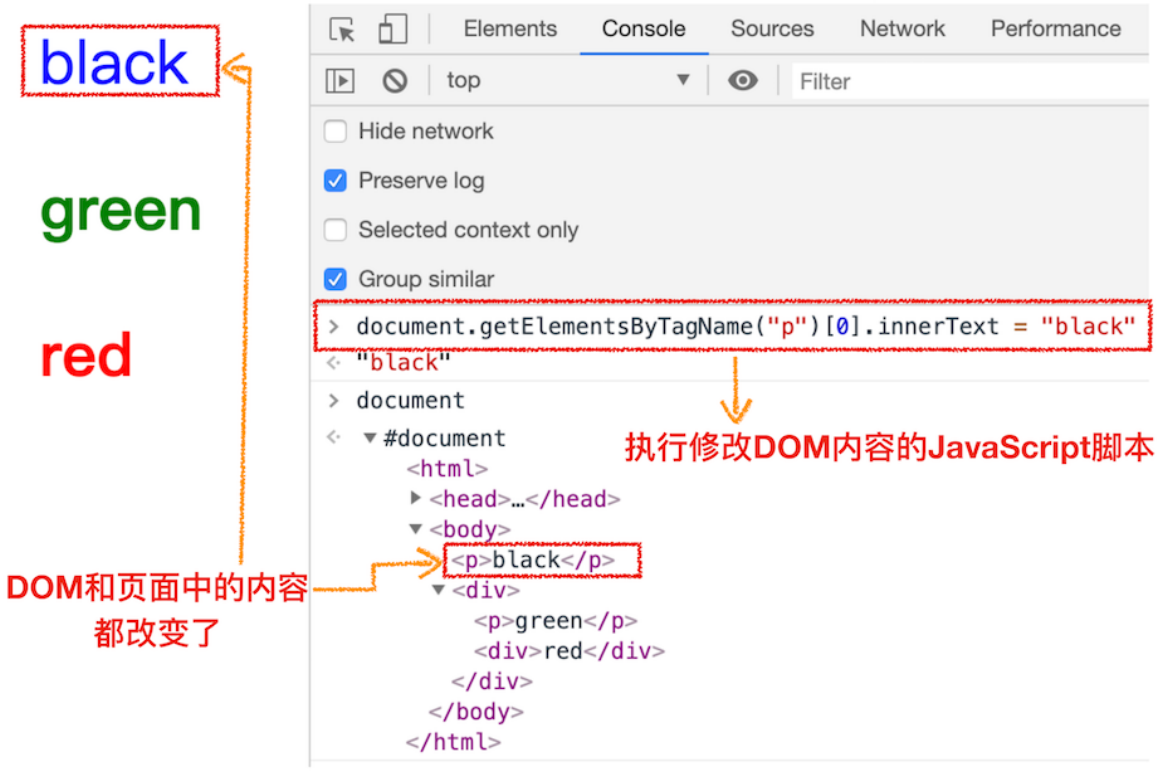
DOM可视化

图中的document就是DOM结构，你可以看到，DOM和HTML内容几乎是一样的，但是和HTML不同的是，DOM是保存在内存中树状结构，可以通过JavaScript来查询或修改其内容。

那下面就来看看如何通过JavaScript来修改DOM的内容，在控制台中输入：

```
document.getElementsByTagName("p")[0].innerText = "black"
```

这行代码的作用是把第一个<p>标签的内容修改为black，具体执行结果你可以参考下图：



通过JavaScript修改DOM

从图中可以看出，在执行了一段修改第一个<p>标签的JavaScript代码后，DOM的第一个p节点的内容成功被修改，同时页面中的内容也被修改了。

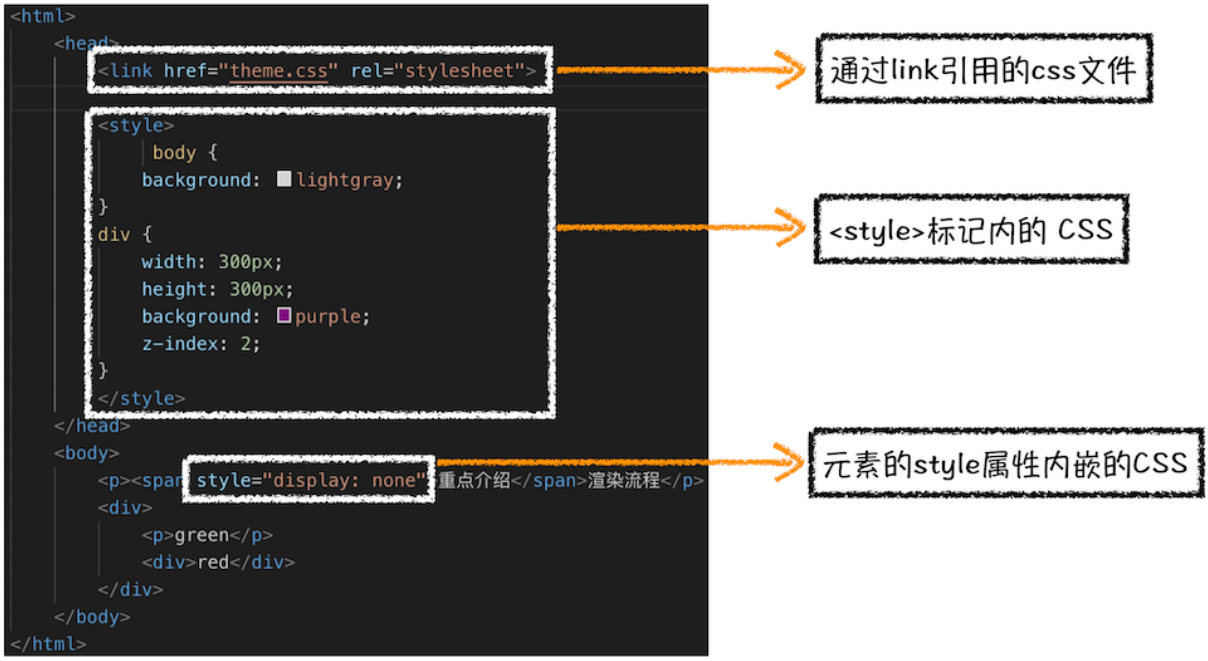
好了，现在我们已经生成DOM树了，但是DOM节点的样式我们依然不知道，要让DOM节点拥有正确的样式，这就需要样式计算了。

样式计算（Recalculate Style）

样式计算的目的是为了计算出DOM节点中每个元素的具体样式，这个阶段大体可分为三步来完成。

1. 把CSS转换为浏览器能够理解的结构

那CSS样式的来源主要有哪些呢？你可以先参考下图：



HTML加载CSS的三种方式

从图中可以看出，CSS样式来源主要有三种：

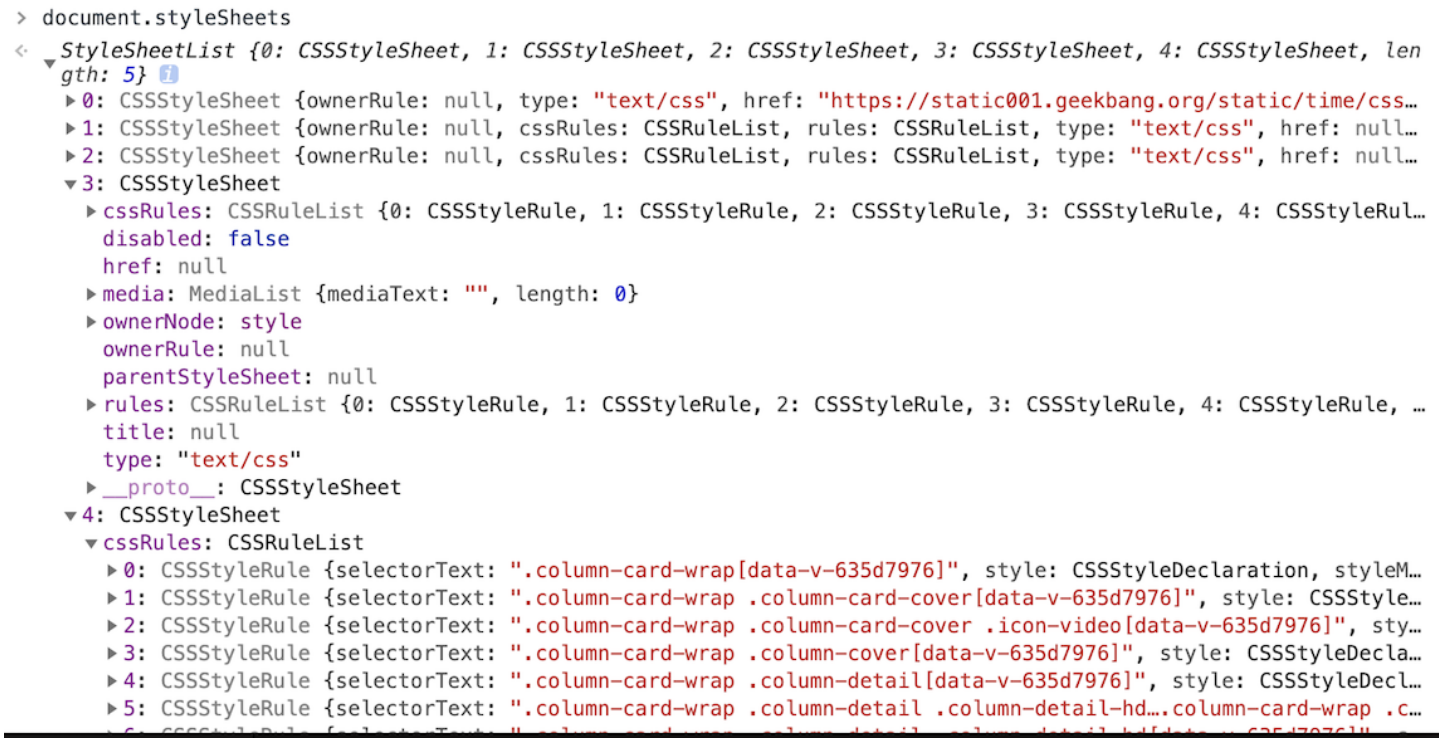
通过link引用的外部CSS文件

<style>标记内的 CSS

元素的style属性内嵌的CSS

和HTML文件一样，浏览器也是无法直接理解这些纯文本的CSS样式，所以**当渲染引擎接收到CSS文本时，会执行一个转换操作，将CSS文本转换为浏览器可以理解的结构——styleSheets**。

为了加深理解，你可以在Chrome控制台中查看其结构，只需要在控制台中输入document.styleSheets，然后就看到如下图所示的结构：



styleSheets

从图中可以看出，这个样式表包含了很多种样式，已经把那三种来源的样式都包含进去了。当然样式表的具体结构不是我们今天讨论的重点，你只需要知道渲染引擎会把获取到的CSS文本全部转换为styleSheets结构中的数据，并且该结构同时具备了查询和修改功能，这会为后面的样式操作提供基础。

2. 转换样式表中的属性值，使其标准化

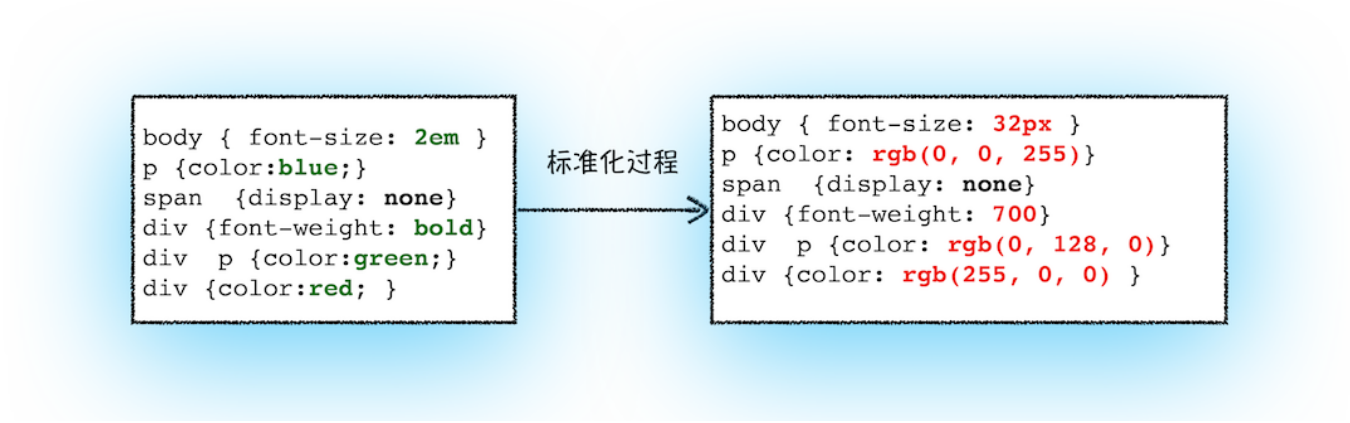
现在我们已经把现有的CSS文本转化为浏览器可以理解的结构了，那么**接下来就要对其进行属性值的标准化操作**。

要理解什么是属性值标准化，你可以看下面这样一段CSS文本：

```
body { font-size: 2em }
p {color:blue;}
span  {display: none}
div {font-weight: bold}
div  p {color:green;}
div {color:red; }
```

可以看到上面的CSS文本中有很多属性值，如2em、blue、bold，这些类型数值不容易被渲染引擎理解，所以**需要将所有值转换为渲染引擎容易理解的、标准化的计算值**，这个过程就是属性值标准化。

那标准化后的属性值是什么样子的？



标准化属性值

从图中可以看到，2em被解析成了32px，red被解析成了rgb(255,0,0)，bold被解析成了700……

3. 计算出DOM树中每个节点的具体样式

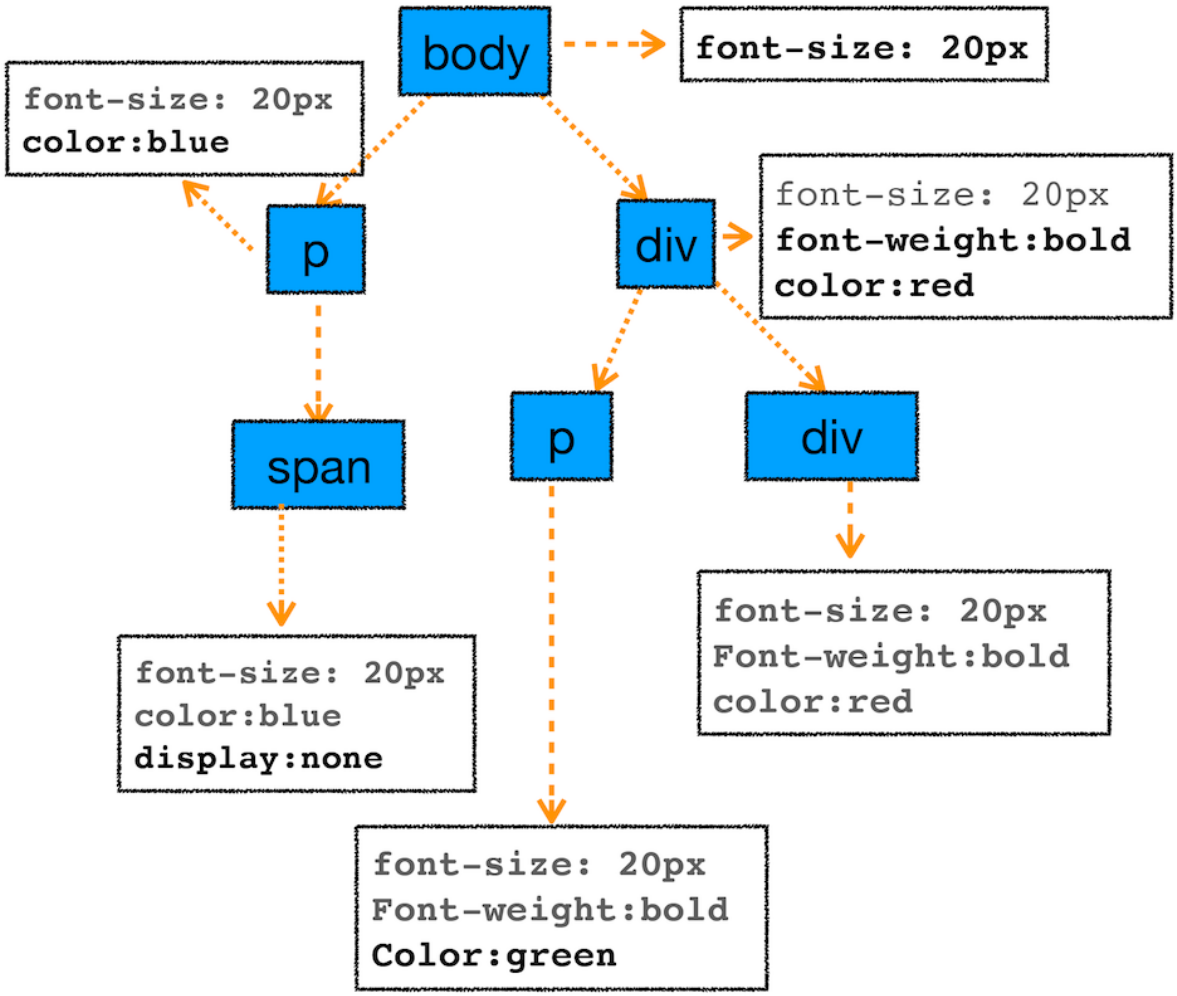
现在样式的属性已被标准化了，接下来就需要计算DOM树中每个节点的样式属性了，如何计算呢？

这就涉及到CSS的继承规则和层叠规则了。

首先是CSS继承。**CSS继承就是每个DOM节点都包含有父节点的样式**。这么说可能有点抽象，我们可以结合具体例子，看下面这样一张样式表是如何应用到DOM节点上的。

```
body { font-size: 20px }
p {color:blue;}
span  {display: none}
div {font-weight: bold;color:red}
div  p {color:green;}
```

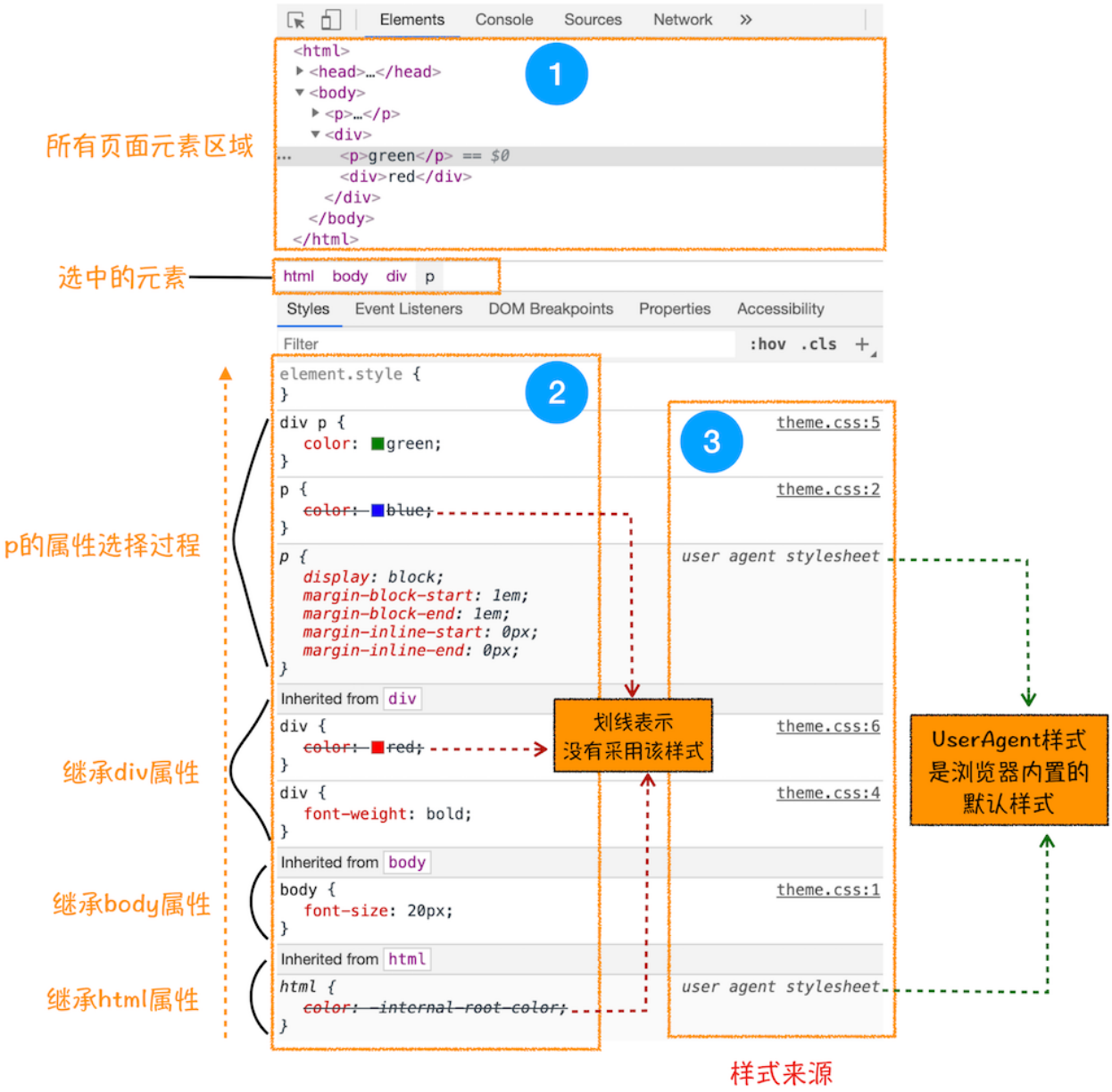
这张样式表最终应用到DOM节点的效果如下图所示：



计算后DOM的样式

从图中可以看出，所有子节点都继承了父节点样式。比如body节点的font-size属性是20，那body节点下面的所有节点的font-size都等于20。

为了加深你对CSS继承的理解，你可以打开Chrome的“开发者工具”，选择第一个“element”标签，再选择“style”子标签，你会看到如下界面：



样式的继承过程界面

这个界面展示的信息很丰富，大致可描述为如下。

首先，可以选择要查看的**元素的样式（位于图中的区域2中）**，在图中的第1个区域中点击对应的元素元素，就可以了下面的区域查看该元素的样式了。比如这里我们选择的元素是<p>标签，位于html.body.div.这个路径下面。

其次，可以从**样式来源（位于图中的区域3中）**中查看样式的具体来源信息，看看是来源于样式文件，还是来源于UserAgent样式表。**这里需要特别提下UserAgent样式，它是浏览器提供的一组默认样式，如果你不提供任何样式，默认使用的就是UserAgent样式**。

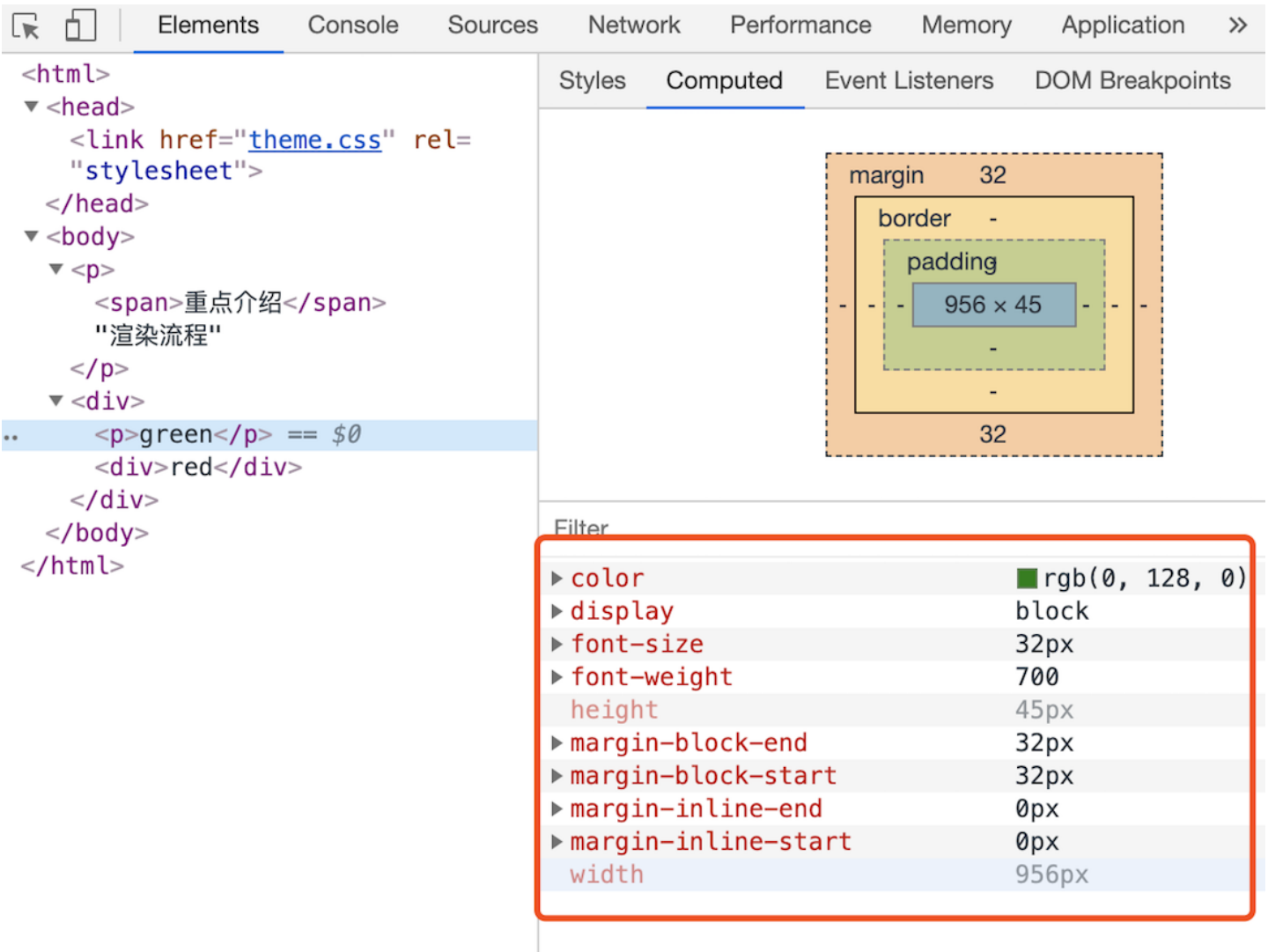
最后，可以通过区域2和区域3来查看样式继承的具体过程。

以上就是CSS继承的一些特性，样式计算过程中，会根据DOM节点的继承关系来合理计算节点样式。

样式计算过程中的第二个规则是样式层叠。**层叠是CSS的一个基本特征，它是一个定义了如何合并来自多个源的属性值的算法。它在CSS处于核心地位，CSS的全称“层叠样式表”正是强调了这一点。**关于层叠的具体规则这里就不做过多介绍了，网上资料也非常多，你可以自行搜索学习。

总之，样式计算阶段的目的是为了计算出DOM节点中每个元素的具体样式，在计算过程中需要遵守CSS的继承和层叠两个规则。这个阶段最终输出的内容是每个DOM节点的样式，并被保存在ComputedStyle的结构内。

如果你想了解每个DOM元素最终的计算样式，可以打开Chrome的“开发者工具”，选择第一个“element”标签，然后再选择“Computed”子标签，如下图所示：



DOM元素最终计算的样式

上图红色方框中显示了html.body.div.p标签的ComputedStyle的值。你想要查看哪个元素，点击左边对应的标签就可以了。

布局阶段

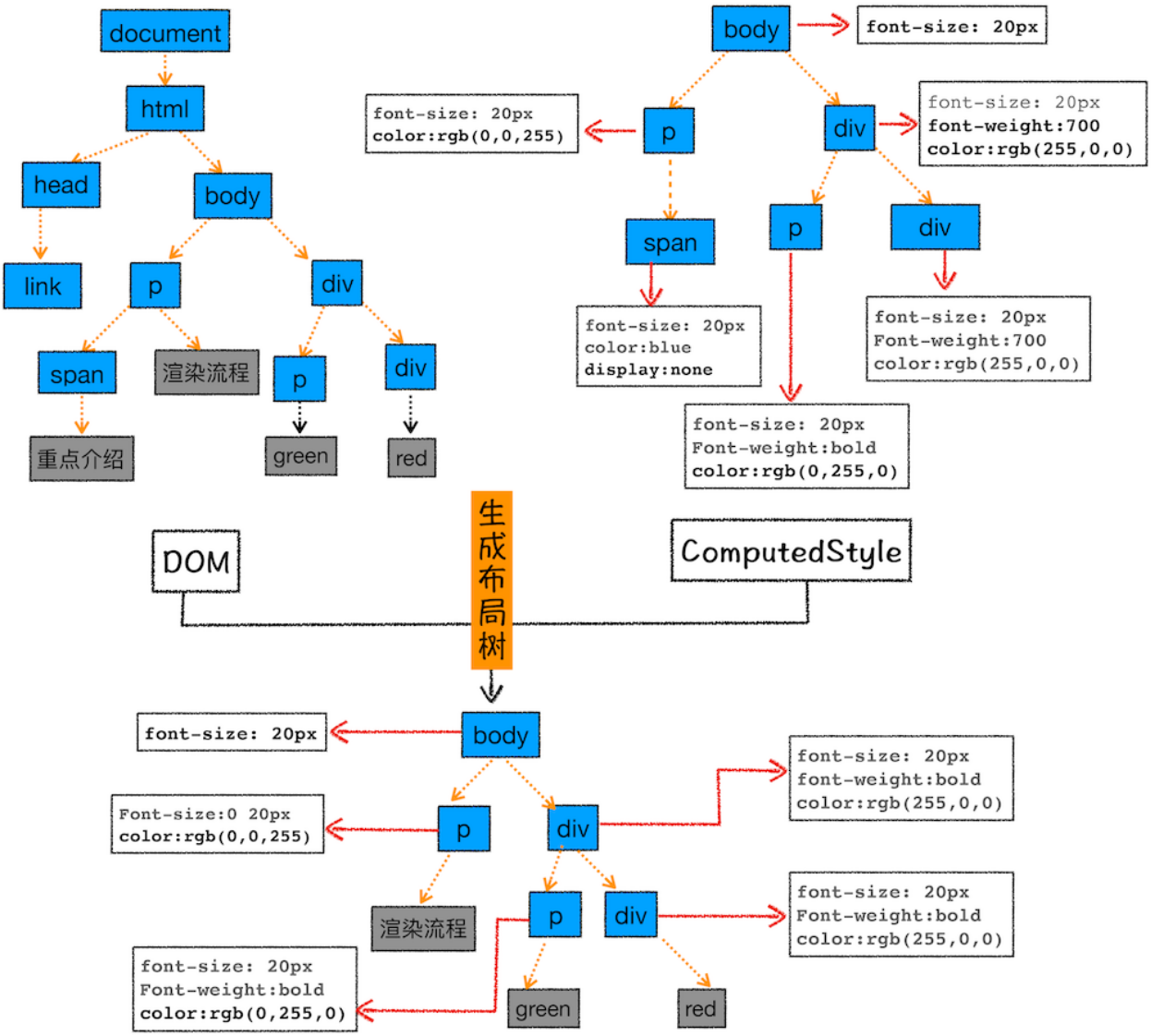
现在，我们有DOM树和DOM树中元素的样式，但这还不足以显示页面，因为我们还不知道DOM元素的几何位置信息。**那么接下来就需要计算出DOM树中可见元素的几何位置，我们把这个计算过程叫做布局**。

Chrome在布局阶段需要完成两个任务：创建布局树和布局计算。

1. 创建布局树

你可能注意到了DOM树还含有很多不可见的元素，比如head标签，还有使用了display:none属性的元素。所以**在显示之前，我们还要额外地构建一棵只包含可见元素布局树**。

我们结合下图来看看布局树的构造过程：



布局树构造过程示意图

从上图可以看出，DOM树中所有不可见的节点都没有包含到布局树中。

为了构建布局树，浏览器大体上完成了下面这些工作：

遍历DOM树中的所有可见节点，并把这些节点加到布局中；

而不可见的节点会被布局树忽略掉，如head标签下面的全部内容，再比如body.p.span这个元素，因为它的属性包含dispaly:none，所以这个元素也没有被包进布局树。

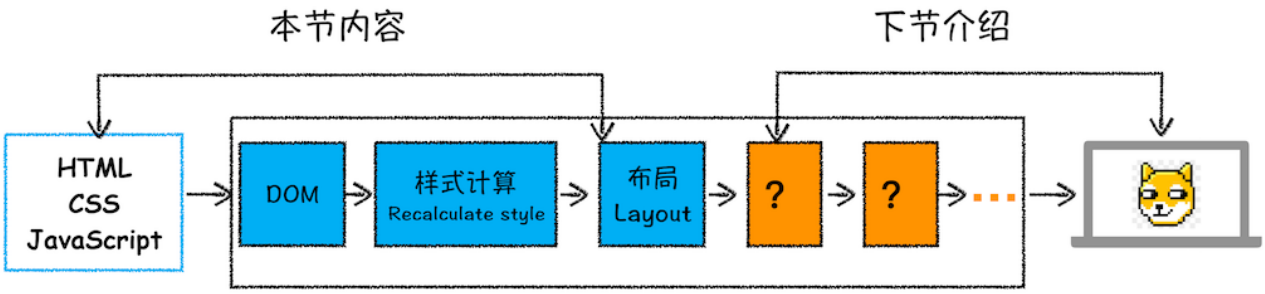
2. 布局计算

现在我们有了一棵完整的布局树。那么接下来，就要计算布局树节点的坐标位置了。布局的计算过程非常复杂，我们这里先跳过不讲，等到后面章节中我再做详细的介绍。

在执行布局操作的时候，会把布局运算的结果重新写回布局树中，所以布局树既是输入内容也是输出内容，这是布局阶段一个不合理的地方，因为在布局阶段并没有清晰地将输入内容和输出内容区分开来。针对这个问题，Chrome团队正在重构布局代码，下一代布局系统叫LayoutNG，试图更清晰地分离输入和输出，从而让新设计的布局算法更加简单。

总结

好了，今天正文就到这里，我画了下面这张比较完整的渲染流水线，你可以结合这张图来回顾下今天的内容。



渲染流水线图

从图中可以看出，本节内容我们介绍了渲染流程的前三个阶段：DOM生成、样式计算和布局。要点可大致总结为如下：

浏览器不能直接理解HTML数据，所以第一步需要将其转换为浏览器能够理解的DOM树结构；

生成DOM树后，还需要根据CSS样式表，来计算出DOM树所有节点的样式；

最后计算DOM元素的布局信息，使其都保存在布局树中。

到这里我们的每个节点都拥有了自己的样式和布局信息，那么后面几个阶段就要利用这些信息去展示页面了，由于篇幅限制，剩下的这些阶段我会在下一篇文章中介绍。

思考时间

最后，给你留个思考题：如果下载CSS文件阻塞了，会阻塞DOM树的合成吗？会阻塞页面的显示吗？

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。

精选留言：

mfist 2019-08-15 06:30:18
关于下载css文件阻塞的问题，我理解
1 不会阻塞dom树的构建，原因Html转化为dom树的过程，发现文件请求会交给网络进程去请求对应文件，渲染进程继续解析Html。
2 会阻塞页面的显示，当计算样式的时候需要等待css文件的资源进行层叠样式。资源阻塞了，会进行等待，直到网络超时，network直接报出相应错误，渲染进程继续层叠样式计算

[3赞]

William 2019-08-15 01:33:33
请问老师，为什么没有清晰地将输入内容和输出内容区分开来不好，我们平时编码过程中，应该尽量做到将输入内容和输出内容区分开来吗？ [2赞]

作者回复2019-08-15 09:19:11

分开来，结构会更加清晰，目前布局操作都是在主线程执行执行的，如果将布局的输入结构和输出结构分开来，那么可以在另外一个线程上执行布局操作，解析完把结果提交给主线程，这样会减轻主线程的压力。

所将输入结构和输出结构分开，后续就可以更好地重构渲染模块的代码了！

这也是Chrome渲染团队目前在做的一件事。

Angus 2019-08-15 20:10:24
这节讲的有些过于省略了，好多东西没有深入去讲。我记得是DOM树和CSSOM树并行构建合成渲染树。从这个角度来说，不会阻塞DOM树的构建，但是会阻塞页面显示，因为页面显示需要完整的渲染树去完成布局计算。 [1赞]

作者回复2019-08-15 21:35:36
和DOM不一样，在源码里面并没有CSSOM这个词，你说的CSSOM 应该是就是styleSheets，这个styleSheets是能直观感受的到的。

渲染树也是16年之前的东西了，现在的代码完全重构了，你可以把LayoutTree看成是渲染树，不过和之前的渲染树还是有一些差别的。

袋袋 2019-08-15 22:53:53
不阻塞dom合成，也不阻塞页面渲染，页面还是会生成，只不过没有样式而已，别忘了标签是有语义化的

Been 2019-08-15 17:24:45
老师，渲染进程的工作原理您是从哪知道的，看浏览器的源码吗？ 有链接吗来一个

作者回复2019-08-15 20:25:57
这个链接有一些参考资料你可以参考下：https://time.geekbang.org/column/article/116572

潮汐 2019-08-15 16:56:21
对留言顶部第一条基本赞成。
但有个疑问：css文件的下载是在网络进程中进行，成功或失败，都是在通知准备渲染进程时已经确定了的吧，还是说渲染过程中会并行下载文件。如果是前者，应该没有阻碍的问题，最多延迟进入准备渲染阶段，也相当于阻塞了页面加载；如果是后者，猜测会阻碍布局树的生成。
提个建议：如果上一节课的问题有标准的对与错答案，下节课开头，老师能不能给一个解答或提及。

作者回复2019-08-15 21:37:09
我尽量在留言区回答问题吧，在正式文章里面回答问题会暂用比较多的时间。

另外CSS和JS下载都是异步的，也就是在解析DOM的过程中下载的。

这块内容我会在后面的页面模块做详细介绍。

许童童 2019-08-15 11:31:05
如果下载 CSS 文件阻塞了，会阻塞 DOM 树的合成吗？会阻塞页面的显示吗？
不会阻塞DOM 树的合成，但会阻塞页面的显示。
DOM 树和CSSOM树是并行生成的，两个都完成后才进行布局树的生成，但如果期间有JS文件，那就需要等待JS文件加载并执行完成，JS也需要等待CSSOM树生成，因为JS可能操作DOM树和CSSOM树。

ytd 2019-08-15 07:58:11
1，不会阻塞dom树生成，因为dom树只要把html下载下来后就可以生成了
2，会阻塞页面显示，浏览器需要等待下载样式表文件合成样式表，进行后面的样式计算。
但是实际观察chrome浏览器加载页面，即便某个样式文件因为网络错误不能下载，页面最终也会显示，是不是样式计算和后续的布局是一个反复的过程？即，先用浏览器默认样式和style标签内样式、内联样式合成并布局显示页面，等下载好外部样式表再次合成并布局。不知道这样理解对不对？另外，如果用户通过操作修改了样式，是不是合成和布局也需要重新进行？

William 2019-08-15 01:37:20
思考题。不会，CSS阻塞了，DOM树照样能正常解析和渲染。猜测浏览器机制，会优先渲染DOM到页面上。平时网络不好时遇到过。

XWL 2019-08-15 01:27:00
应该不会阻塞，link加载CSS样式本身是异步进行的，所以并不会影响浏览器继续解析之后的DOM的标签，最后由CSS树和DOM树合成render树，然后由render树渲染成页面，所以CSS的下载不阻塞DOM树，但阻塞着最后页面的渲染。
这是我的理解，有错误请指出。。。
另外，老师为什么不讲讲回流和重绘

作者回复2019-08-15 09:00:19
重绘和重排这些概念会在06节介绍，要等渲染流水线介绍完。

leitong 2019-08-15 00:53:41

DOM合成不受影响，但是肯定会阻塞页面的显示