浏览器的解析渲染原理以及js， css 阻塞问题的分析

2018年01月25日 13:15:27 悬炫321 阅读数：98 更多

最近在重新翻阅js教程这本书，在翻阅第二章的时候，发现之前遗留的问题，现在也没怎么搞明白，那就是script标签里面的defer以及async属性到底有什么用？样的

□使用defer属性可以让脚本在文档完全呈现之后再执行。延迟脚本总是按照指定它们的顺序执行。

□使用async属性可以表示当前脚本不必等待其他脚本，也不必阻塞文档呈现。不能保证异步脚本按照它们在页面中出现的顺序执行。

当时看完，我心中有两个问题：

1. 此处的呈现是什么意思，dom已经渲染完毕了吗？可是经我测试以后，无论是带有defer还是带有async属性的script标签都是会阻塞页面页面渲染的
2. 浏览器到底是怎么加载，解析，渲染页面的呢？（这里的加载其实就是下载相应的资源）

带着这两个问题，我查阅大量的资料和文献，做出了如下的一些分析与总结：

浏览器大致的解析渲染流程如下

1.首先当用户输入一个URL的时候，浏览器就会发送一个请求，请求URL对应的资源,请求成功的话，浏览器会收到一个html文件。

2.然后浏览器的HTML解析器会对这个文件自上而下开始解析，尝试去构建一棵完整的Dom树。

3.在构建**DOM**树的时候，当遇到**JS**元素时，**HTML**解析器就会将控制权转让给**JavaScript**引擎线程，该线程会阻断**HTML**解析器的运行，当**js**加载并且执行完毕后，**JavaScript**引擎线程会将控制器，让其去继续构建**dom**树；当遇到**css**元素时，浏览器会开 启一个异步请求线程，在该线程上，浏览器会去请求相应的**css**文件，并且根据该文件去构建**cssDom**树(也叫**css rule**)，该线程会阻塞 **Ja**（即 **css** 后 面的 **js** 模块的解析会在 **css** 解析完毕后执行），，但是不会阻塞 dom 树的构建。具体案例可以参考 https://www.cnblogs.com/chenjg/p/7126822.html（async，defer这两个属性说白了就是用来控制js的执行于

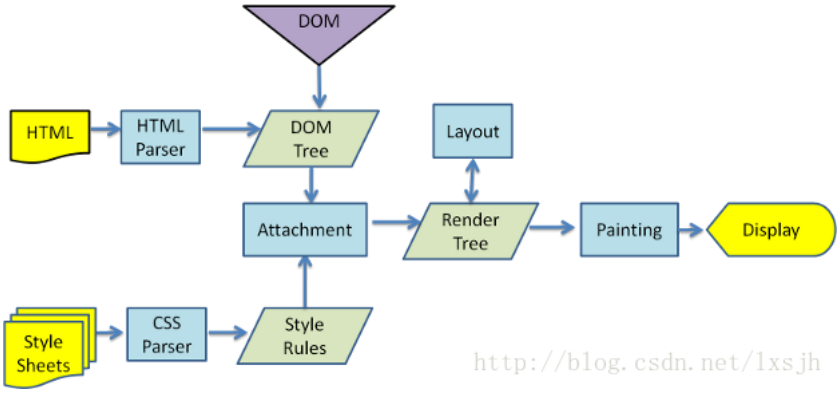
4.DOM树构建完之后，浏览器把DOM树中的一些不可视元素去掉，然后与CSSOM合成一棵render树。

5.接着浏览器根据这棵render树，计算出各个节点(元素)在屏幕的位置。这个过程叫做layout，输出的是一棵layout树。

7.最后浏览器根据这棵layout树，将页面渲染到屏幕上去。

总结一下就是

接收html以构建dom树和cssdom树 ->合并dom树和cssdom树-> 构建render树 -> 布局render树 -> 绘制render树（下图中的attachment就是dom树和cssdom



这里需要注意一点，在现在浏览器中，为了减缓渲染被阻塞的情况，现代的浏览器都使用了猜测预加载。当解析被阻塞的时候，浏览器会有一个轻量级的HTML（或CS anner）继续在文档中扫描，查找那些将来可能能够用到的资源文件的url，在渲染器使用它们之前将其下载下来，并且下载是可以并行进行的，并行的上限一般为6。

既然Dom树完全生成好后页面才能渲染出来，浏览器又必须读完全部HTML才能生成完整的Dom树，如果不考虑js要对元素进行处理的情况，script标签不放在body样，因为dom树的生成需要整个文档解析完毕。

其实现代浏览器为了更好的用户体验,渲染引擎将尝试尽快在屏幕上显示的内容。它不会等到所有HTML解析之前开始构建和布局渲染树。部分的内容将被解析并显示器能够渲染不完整的dom树和cssom，尽快的减少白屏的时间。假如我们将js放在header，js将阻塞解析dom，dom的内容会影响到First Paint，导致First Paint延后。所以说我面，以减少First Paint的时间，但是不会减少DOMContentLoaded被触发的时间。当外联的JS代码和CSS代码还没从服务器传到浏览器的时候，这个时候如果DOM树上有可视元通常会选择在这个时候，将一些内容提前渲染到屏幕上来。

好，现在再来回答一下开始的问题：1 .书中的呈现指的是dom树的构建 2.浏览器的渲染机制就是上面说的这些 ；有关于async于defer属性更详细的对比，请参考ht ault.com/q/1010000000640869

户体验。

完整的回答是：

浏览器大致的解析渲染流程如下

1. 首先当用户输入一个URL的时候，浏览器就会发送一个请求，请求URL对应的资源,请求成功的话，浏览器会收到一个html文件。

2.然后浏览器的HTML解析器会对这个文件自上而下开始解析，尝试去构建一棵完整的Dom树。

3.在构建DOM树的时候，当遇到JS元素时，HTML解析器就会将控制权转让给JS解析器，浏览器会开启JavaScript引擎线程，该线程会阻断HTML解析器的 进程时，没有其他资源会被继续加载与解析，dom树的构建与渲染都会被阻塞；当遇到css元素时，HTML解析器就换将控制权转让给css解 析器，浏览器会开启一个异步请程上，浏览器会去请求相应的css文件，并且根据该文件去构建cssDom树(也叫css rule)，该线程会阻塞 JavaScript引擎线程（即css 模块的解析会在c行），但是不会阻塞dom树的构建。具体案例可以参考 https://www.cnblogs.com/chenjg/p/7126822.ht c，defer这两个属来控制js的执行开始时间的）

4.DOM树构建完之后，浏览器把DOM树中的一些不可视元素去掉，然后与CSSOM合成一棵render树。

5.接着浏览器根据这棵render树，计算出各个节点(元素)在屏幕的位置。这个过程叫做layout，输出的是一棵layout树。

7.最后浏览器根据这棵layout树，将页面渲染到屏幕上去。

但是有两点需要注意一下：

1.为了减缓渲染被阻塞的情况，现代的浏览器都使用了猜测预加载。当解析被阻塞的时候，浏览器会有一个轻量级的HTML（或CSS）扫描器（scanner）继续在文档中扫描可能能够用到的资源文件的url，在渲染器使用它们之前将其下载下来，并且下载是可以并行进行的，并行的上限一般为6。

2.其实现代浏览器为了更好的用户体验,渲染引擎将尝试尽快在屏幕上显示的内容。它不会等到所有HTML解析之前开始构建和布局渲染树。部分的内容将被解析并显示。也够渲染不完整的dom树和cssom，尽快的减少白屏的时间。

所以，如果将css文件放在头部的话，浏览器部分渲染的时候，cssDom树还未构建呢，构建之后的话，要对之前的就行重新渲染。还有如果将js文件文件放置于顶部的话，的构建，浏览器无法进行部分渲染。

最后补充两个名词的解释： （1）Reflow（回流）：浏览器要花时间去渲染，当它发现了某个部分发生了变化影响了布局，那就需要倒回去重新渲染。

（2）Repaint（重绘）：如果只是改变了某个元素的背景颜色，文字颜色等，不影响元素周围或内部布局的属性，将只会引起浏览器的repaint，重画某一部分。Reflow要比Repaint更花费时间，也就更影响性能。所以在写代码的时候，要尽量避免过多的Reflow。

第一次写博客，可能存在诸多不足，还望指正！