## 计算机的设计初衷就是把一件事件简单化，提高我们的生产力。在我们日常前端工作，跟浏览器打交道是必不可少的，既然要跟浏览器沟通，我们就要去了解它渲染机制，我相信这会对我们日后的工作有所帮助。

## 做一件事情总要知道理由吧，好吧下面先说一下学习浏览器渲染机制的目的吧。

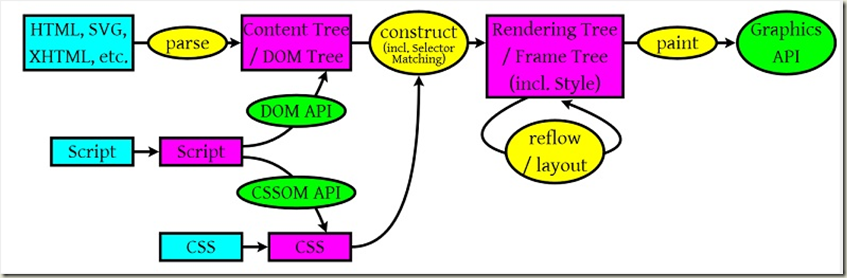
为什么要了解浏览器加载、解析、渲染这个过程？

了解浏览器如何进行加载，我们可以在引用外部样式文件，外部js时，将他们放到合适的位置，使浏览器以最快的速度将文件加载完毕。

了解浏览器如何进行解析，我们可以在构建DOM结构，组织css选择器时，选择最优的写法，提高浏览器的解析速率。

了解浏览器如何进行渲染，明白渲染的过程，我们在设置元素属性，编写js文件时，可以减少”重绘“”重新布局“的消耗。

这三个过程在实际进行的时候又不是完全独立，而是会有交叉。会造成一边加载，一边解析，一边渲染的工作现象。

浏览器是怎样渲染一个页面的？

上面这个图中，我们可以看到那么几个事：

1）浏览器会解析三个东西：

* 一个是HTML/SVG/XHTML，事实上，Webkit有三个C++的类对应这三类文档。解析这三种文件会产生一个DOM Tree。
* CSS，解析CSS会产生CSS规则树。
* Javascript，脚本，主要是通过DOM API和CSSOM API来操作DOM Tree和CSS Rule Tree.

2）解析完成后，浏览器引擎会通过DOM Tree 和 CSS Rule Tree 来构造 Rendering Tree。注意：

* Rendering Tree 渲染树并不等同于DOM树，因为一些像Header或display:none的东西就没必要放在渲染树中了。
* CSS 的 Rule Tree主要是为了完成匹配并把CSS Rule附加上Rendering Tree上的每个Element。也就是DOM结点。也就是所谓的Frame。
* 然后，计算每个Frame（也就是每个Element）的位置，这又叫layout和reflow过程。

3）最后通过调用操作系统Native GUI的API绘制。

我们从浏览器渲染页面的大概过程开始说起：

由从服务器接收到的 HTML 形成 DOM（文档对象模型）。

样式被加载和解析，形成 CSSOM（CSS 对象模型）。

紧接着 DOM 和 CSSOM 创建了一个渲染树，这个渲染树是一些被渲染对象的集合（ Webkit 分别叫它们”renderer”和”render object”，而在Gecko 引擎中叫”frame”）。除了不可见的元素（比如 head 标签和一些有 display:none 属性的元素），渲染树映射了 DOM 的结构。在渲染树中，每一个文本字符串都被当做一个独立的 renderer。每个渲染对象都包含了与之对应的计算过样式的DOM 对象（或者一个文本块）。换句话说，渲染树描述了 DOM 的直观的表现形式。

对每个渲染元素来说，它的坐标是经过计算的，这被叫做“布局(layout)”。浏览器使用一种只需要一次处理的“流方法”来布局所有元素（tables需要多次处理）。

最后，将布局显示在浏览器窗口中，这个过程叫做“绘制(painting)”。

实际的优化建议

汇总了一些有用的信息，我建议以下几点：

创建合法的 HTML 和 CSS ，别忘了制定文件编码，Style 应该写在 head 标签中，script 标签应该加载 body 标签结束的位置。

试着简化和优化 CSS 选择器（这个优化点被大多数使用 CSS 预处理器的开发者忽略了）。将嵌套层数控制在最小。以下是 CSS 选择器的性能排行（从最快的开始）：

ID选择器：#id

class选择器： .class

标签: div

相邻的兄弟元素：a + i

父元素选择器： ul > li

通配符选择器： \*

伪类和伪元素： a:hover ，你应该记住浏览器处理选择器是从右向左的，这也就是为什么最右面的选择器会更快——#id或.class。