

1. Use efficient CSS selectors: 使用高效的 CSS 选择符

关于高效的 CSS 选择符的叙述，很多书籍和网站很多博客都有相关的说明，尤其《[Writing Efficient CSS for Use in the Mozilla UI](#)》提出了很多的建议以及讲解了 CSS 引擎解析样式规则的原理。在那本《高性能网站建设进阶指南》中也独立出一章节来讲解高效的 CSS 选择符。

当浏览器开始下载 HTML 内容并解析 HTML 文档的时候，就会生成一个 DOM 节点树。而 CSS 选择符的匹配就是在这个节点树中查询符合条件的元素，并添加样式。当浏览器在解析 CSS 样式表的时候，它会解析样式表中的每一条样式规则，而不管最后样式规则是否会匹配到元素。

并且，CSS 引擎是从右往左进行解析规则的，所以这里就有几点建议可以用来优化 CSS 选择符了：

- 第一、避免最右端的选择符选择过多的 DOM 元素。比如全局选择符“*”，标记符 `(.hide-scrollbars * {...})`，这些在 CSS 引擎解析该规则的时候最开始会选择 DOM 节点树中太多的节点，这样再往左查询的时候，造成查询所用的时间就更长了。所以，右边的选择符匹配的元素越少越好。
- 第二、对于子选择符或者说是邻近选择符 (child-selector: `ul > li > a`) 为什么低效，Google Page Speed 做了如下解释：因为浏览器在解析一个选择符其中一步的时候，比如上面的 `a`，首先它要确认它的父元素中存在 `li` 元素，同时还要判断 `li` 元素是否是 `a` 的直接父元素，这就需要两个判断操作，相比于后代选择符 (`ul li a`) 来说，它就多花费了一倍的性能消耗来执行每一步的匹配。
- 第三、避免一些累赘的选择符。比如在 `class` 类中增加了标签 (`div.sidebar`)，ID 也同样。当然了，如果你想要通过这种方式来增加选择符的权重，这个就另当别论了。
- 第四、对于 `:hover` 选择符的选择。`:hover` 选择符在 IE7 以上的版本的标准模式中都是良好的，但是一旦回退到怪癖模式，就跟 IE6 一个样了，只认超链接元素。所以总是声明标准的 DOCTYPE，可以避免很多不必要的问题发生。
- 第五、从 CSS 引擎的解析原理出发，选择符越简短越好，越明确越好，确保右边的规则不会选择过多的 DOM 元素。同时，可以的话选择 `class` 来代替后代选择符。

2. Specify a character set early: 尽早的声明 character.

这点在《[页面中 charset 的 meta 声明将如何影响页面加载性能](#)》这篇文章中有详细的介绍，这里就不多说了。

3. Specify image dimensions: 声明图片的尺寸

很多网站页面中对图片都没有明确的声明图片 `img` 的尺寸（即：设置 `img` 标签的 `height` 和 `width` 属性为图片的原始大小，或者在 CSS 中声明，不过最好还是在 `img` 标签中直接声明）。明确的在 `img` 标签中声明图片的尺寸可以加快浏览器对图片的渲染和布局，从而避免不必要的回流 (reflow) 和重新渲染 (repaint)。当浏览器布局页面的时候，它需要循环检查那些元素是可以置换的，比如图片。如果浏览器预先知道了一个可以置换的元素（比如图片）的尺寸，那么就可以使得一些不可置换的元素跟这个可置换元素正常的布局，即使图片还没有加载下来，浏览器就可以开始正常的渲染、布局整个页面。

但是，如果没有预先声明的尺寸，或者声明的尺寸跟元素的实际尺寸不符，那么一旦这些元素下载下来的时候，浏览器就需要一次回流 (reflow) 和重新渲染 (repaint) 的过程。

因此，为了避免不必要的回流和重新渲染的操作，预先声明图片的宽高，并且这个宽高需要跟图片的实际大小相符，其他可置换元素也一样的原理。

4. Put CSS in the document head: 将 CSS 文件放到 head 里声明

将行内样式和 CSS 文件都放到 `head` 部位来声明，可以使得页面能及时渲染，也能避免不必要的“白屏”和“无样式内容的闪烁”的问题的产生。

同时还有一点需要说明的是：当使用 `@import` 来加载外部样式表的时候，也可能出现上面提到的“白屏”和“无样式内容的闪烁”的问题，因为采用 `@import` 加载样式表的时候，加载顺序是不定的，它有可能使

得样式表在最后才加载进来。还有避免样式表后面紧跟 script 标签所声明的行内脚本。

5. Put Javascript in the document bottom: 将脚本文件放到文档的末尾

这个主要是考虑到正常通过 script 标签加载 Javascript 文件的时候，会阻塞后面一切资源的加载。所以尽可能的将 Javascript 文件都放到文档的末尾来加载，就不会影响页面的渲染进程，但是它仍然对 onload 事件造成延迟，直到文件下载并执行完毕。《<http://www.ilovejs.net/lab/loadjs/>》这个页面展示了动态加载 Javascript 文件的方式，可以有效的解决这个延迟问题。

同时，Javascript 文件和样式表文件要按照合理的顺序来加载，避免阻塞样式表加载的问题。

6. Avoid CSS expressions: 避免使用 CSS expression

关于 css 的 expression，应用的是比较少的，但是不能说它没有被使用。Css 的 expression 会明显的降低页面渲染的性能，避免使用它会改善 IE 用户的体验。

CSS expression 是在 IE5 的时候加进 IE 中的，使用它可以使得 Javascript 可以通过一些事件动态的操作 CSS 中的某些属性，从而来改变 document 的某些属性。它在 IE5 到 IE7 版本之间受支持，IE8 声称不再受支持，同时其他浏览器都不支持。

但是使用 CSS expression 会造成非常糟糕的性能问题。因为在一些普通细小的操作都可能使得 expression 重复执行，比如：窗口缩放、鼠标移动等等，反正是触发了 reflow 的因素都会触发 expression，这对 IE 用户来说，是很恶心的性能问题，严重的阻碍用户跟页面的交互操作，恐怖的就是使得 IE 崩溃了。

7. Combine external Javascript and CSS: 合并外部的 Javascript 和 CSS 文件

这个主要是从最小化 HTTP 请求这个优化准则来说的，将几个 Javascript 文件或者几个 CSS 文件合并成一个文件，再结合 Gzip 压缩，来优化资源的加载时间。可以浏览这篇文章的说明：《[Javascript,CSS Minify](#)》。

8. Avoid bad requests: 避免请求无效的链接

当我们点击了一个超链接，而这里链接的目的地却是无效的，这个不仅造成了恶心的 404 状态吗，而且还严重影响了用户体验。通过返回 404 状态的时候都是比较消耗时间的，因为服务器需要查询整个站点是否有这个链接所执行的资源，到最终重定向到一个 404 页面。而用户等待了些时间最后得来的，却是一个 404 Not Found 页面，很恶心的事情。

因此，当一个站点越来越大，更新快的时候，适时的检查页面的链接是否失效并及时的修正这些无效的链接，这对提高用户体验以及避免页面加载一些无用的字节都是有用的。

9. Minimize redirects: 最少化重定向的数量

重定向在某些情况下是必须的，将用户从一个页面重定向到另外一个页面。但是重定向会触发额外的 HTTP 请求和响应的操作，并且增加了往返时间的延迟，而这个额外的 HTTP 请求和响应就是 301 和 302 状态码了。从重定向的时候服务器会返回一个 301 或者 302 状态码来确定此次请求是重定向，之后再发送请求到服务器请求另外一个页面来加载页面内容和资源。

这里我需要说明一点的是我们平时都可能不太在意的细节---URL，我们知道有些 URL 中有斜杠“/”，而有些 URL 中不包含，比如：<http://www.example.com/sport> 和 <http://www.example.com/sport/>，这两个 URL 中的区别仅仅是最末尾的一个斜杠，但是请求这两个 URL 时，第一个链接会产生一个重定向，因为服务器不知道请求的这个是文件还是目录，当判断是目录的时候，就会在 URL 的末尾加上一个斜杠(这就造成重定向)，之后就会查询该目录下的 index 文件名的文件来响应请求；而第二个链接就会使得服务器直接省略重定向的操作，请求该目录下的 index 类型的文件。

通过，还可以几点说明：如果需要重定向几个页面的话，比如 A—B—C，但是如果直接 A—C 重定向就能实现功能的话，就尽量减少重定向的次数。

通常我们从跟一个页面重定向到另外一个页面的方式是采用 Javascript 的 `location` 或者在 `head` 里通过 `meta` 的方式来实现,但是这样都会造成重定向造成的性能问题。对于上面提到的 URL 方面不注意造成的重定向,有时候可以通过在服务器端 `rewrite` 的方式来解决。这就可以使得从一个 URL 映射到另外一个 URL,这个操作是服务器自动完成的,不会出发重定向的问题。