每个前端开发者都要理解网页渲染

> 渲染应该从最开始当页面布局被定义时就进行优化，样式和脚本在页面渲染中扮演着非常重要的角色。专业人员知道一些技巧以避免一些性能问题。

浏览器是怎样渲染一个页面的？

我们从浏览器渲染页面的大概过程开始说起：

1. 由从服务器接收到的 HTML 形成 DOM（文档对象模型）。
2. 样式被加载和解析，形成 CSSOM（CSS 对象模型）。
3. 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。
4. 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）。
5. 最后，将布局显示在浏览器窗口中，这个过程叫做“绘制(painting)”。

重绘

当在页面上修改了一些不需要改变定位的样式的时候（比如background-color,border-color,visibility)，浏览器只会将新的样式重新绘制给元素（这就叫一次“重绘”或者“重新定义样式”）。

重排

当页面上的改变影响了文档内容、结构或者元素定位时，就会发生重排（或称“重新布局”）。重排通常由以下改变触发：

1. DOM 操作（如元素增、删、改或者改变元素顺序）。
2. 内容的改变，包括 Form 表单中文字的变化。
3. 计算或改变 CSS 属性。
4. 增加或删除一个样式表。
5. 改变”class”属性。
6. 浏览器窗口的操作（改变大小、滚动窗口）。
7. 激活伪类（如:hover状态）。

浏览器如何优化渲染？

浏览器尽最大努力限制重排的过程仅覆盖已更改的元素的区域。举个例子，一个 position 为 absolue 或 fixed 的元素的大小变化只影响它自身和子孙元素，而对一个 position 为 static 的元素做同样的操作就会引起所有它后面元素的重排。

另一个优化就是当运行一段Jjavascript 代码的时候，浏览器会将一些修改缓存起来，然后当代码执行的时候，一次性的将这些修改执行。举例来说，这段代码会触发一次重绘和一次重排：

```
var $body = $('body');
$body.css('padding', '1px');
// 重排，重绘
$body.css('color', 'red');
// 重绘
$body.css('margin', '2px');
// 重排，重绘
// 实际上只有一次重排和重绘被执行。
```

如上面所说，访问一个元素的属性会进行一次强制重排。如果我们给上面的代码加上一行读取元素属性的代码，这个情况就会出现：

```
var $body = $('body');
$body.css('padding', '1px');
$body.css('padding');
//这里读取了一次元素的属性，一次强制重排就会发生。
$body.css('color', 'red');
$body.css('margin', '2px');
```

上面这段代码的结果就是，进行了两次重排。因此，为了提高性能，你应该将读取元素属性的代码组织在一起。

有一种情况是必须触发一次强制重排的。例如：给元素改变同一个属性两次（比如margin-left），一开始设置100px，没有动画，然后通过动画的形式将值改为50px。具体可以看例子，当然，我在这里会讲更多的细节。

我们从一个有transition的CSS class开始：

```
.has-transition {
   -webkit-transition: margin-left 1s ease-out;
      -moz-transition: margin-left 1s ease-out;
        -o-transition: margin-left 1s ease-out;
           transition: margin-left 1s ease-out;
}
```

然后进行实现：

```
//我们的元素默认有"has-transition"属性
var $targetElem = $('#targetElemId');

//删除包含transition的class
$targetElem.removeClass('has-transition');

// 当包含transition的class已经没了的时候，改变元素属性
$targetElem.css('margin-left', 100);

// 再将包含transition的class添加回来
$targetElem.addClass('has-transition');

// 改变元素属性
$targetElem.css('margin-left', 50);
```

上面的实现没有按照期望的运行。所有的修改都被浏览器缓存了，只在上面这段代码的最后才会执行。我们需要的是一次强制重排，我们可以通过进行以下修改来实现：

```
//删除包含transition的class
$(this).removeClass('has-transition');

// 改变元素属性
$(this).css('margin-left', 100);

//触发一次强制重排，从而使变化了的class或属性能够立即执行。
$(this)[0].offsetHeight;
// offsetHeight仅仅是个例子，其他的属性也可以奏效。

// 再将包含transition的class添加回来
$(this).addClass('has-transition');

// 改变元素属性
$(this).css('margin-left', 50);
```

现在这段代码如我们所期望的运行了。

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）：
    1. ID选择器：#id
    2. class选择器： .class
    3. 标签: div
    4. 相邻的兄弟元素：a + i
    5. 父元素选择器： ul > li
    6. 通配符选择器： *
    7. 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。

```
div * {...} // bad
.list li {...} // bad
.list-item {...} // good
#list .list-item {...} // good
```

8. 在你的脚本中，尽可能的减少 DOM 的操作。把所有东西都缓存起来，包括属性和对象（如果它可被重复使用）。进行复杂的操作的时候，最好操作一个“离线”的元素（“离线”元素的意思是与 DOM 对象分开、仅存在内存中的元素），然后将这个元素插入到 DOM 中。
9. 如果你使用 jQuery，遵循jQuery 选择器最佳实践
10. 要改变元素的样式，修改“class”属性是最高效的方式之一。你要改变 DOM 树的层次越深，这一条就越高效（这也有助于将表现和逻辑分开）。
11. 尽可能的只对 position 为 absolute 或 fix 的元素做动画。
12. 当滚动时禁用一些复杂的 :hover 动画是一个很好的主意（例如，给 body 标签加一个 no-hover 的 class）