

浏览器CSSOM：如何获取一个元素的准确位置

winter 2019-03-16



你好，我是 winter。

在前面的课程中，我们已经学习了 DOM 相关的 API，狭义的 DOM API 仅仅包含 DOM 树形结构相关的内容。今天，我们再来学习一类新的 API：CSSOM。

我想，你在最初接触浏览器 API 的时候，应该都有跟我类似的想法：“好想要 `element.width`、`element.height` 这样的 API 啊”。

这样的 API 可以直接获取元素的显示相关信息，它们是非常符合人的第一印象直觉的设计，但是，偏偏 DOM API 中没有这样的内容。

随着学习的深入，我才知道，这样的设计是有背后的逻辑的，正如 HTML 和 CSS 分别承担了语义和表现的分工，DOM 和 CSSOM 也有语义和表现的分工。

DOM 中的所有的属性都是用来表现语义的属性，CSSOM 的则都是表现的属性，`width` 和 `height` 这类显示相关的属性，都属于我们今天要讲的 CSSOM。

顾名思义，CSSOM 是 CSS 的对象模型，在 W3C 标准中，它包含两个部分：描述样式表和规则等 CSS 的模型部分（CSSOM），和跟元素视图相关的 View 部分（CSSOM View）。

在实际使用中，CSSOM View 比 CSSOM 更常用一些，因为我们很少需要用代码去动态地管理样式表。

在今天的文章中，我来分别为你介绍这两部分的 API。

CSSOM

首先我们来介绍下 CSS 中样式表的模型，也就是 CSSOM 的本体。

我们通常创建样式表也都是使用 HTML 标签来做到的，我们用 style 标签和 link 标签创建样式表，例如：

```
1 <style title="Hello">
2   a {
3     color:red;
4   }
5 </style>
6 <link rel="stylesheet" title="x" href="data:text/css,p%7Bcolor:blue%7D">
```

 复制代码

我们创建好样式表后，还有可能要对它进行一些操作。如果我们以 DOM 的角度去理解的话，这些标签在 DOM 中是一个节点，它们有节点的内容、属性，这两个标签中，CSS 代码有的在属性、有的在子节点。这两个标签也遵循 DOM 节点的操作规则，所以可以使用 DOM API 去访问。

但是，这样做的后果是我们需要去写很多分支逻辑，并且，要想解析 CSS 代码结构也不是一件简单的事情，所以，这种情况下，我们直接使用 CSSOM API 去操作它们生成的样式表，这是一个更好的选择。

我们首先了解一下 CSSOM API 的基本用法，一般来说，我们需要先获取文档中所有的样式表：

```
1 document.styleSheets
```

document 的 styleSheets 属性表示文档中的所有样式表，这是一个只读的列表，我们可以用方括号运算符下标访问样式表，也可以使用 item 方法来访问，它有 length 属性表示文档中的样式表数量。

样式表只能使用 style 标签或者 link 标签创建（对 XML 来说，还可以使用，咱们暂且不表）。

我们虽然无法用 CSSOM API 来创建样式表，但是我们可以修改样式表中的内容。

```
1 document.styleSheets[0].insertRule("p { color:pink; }", 0)
2 document.styleSheets[0].removeRule(0)
```

更进一步，我们可以获取样式表中特定的规则（Rule），并且对它进行一定的操作，具体来说，就是使用它的 cssRules 属性来实现：

```
1 document.styleSheets[0].cssRules
```

这里取到的规则列表，同样是支持 item、length 和下标运算。

不过，这里的 Rules 可就没那么简单了，它可能是 CSS 的 at-rule，也可能是普通的样式规则。不同的 rule 类型，具有不同的属性。

我们在 CSS 语法部分，已经为你整理过 at-rule 的完整列表，多数 at-rule 都对应着一个 rule 类型：

CSSStyleRule

CSSCharsetRule

CSSImportRule

CSSMediaRule

CSSFontFaceRule

CSSPageRule

CSSNamespaceRule

CSSKeyframesRule

CSSKeyframeRule

CSSSupportsRule

具体的规则支持的属性，建议你可以用到的时候，再去查阅 MDN 或者 W3C 的文档，在我们的文章中，仅为你详细介绍最常用的 CSSStyleRule。


CSSStyleRule 有两个属性：selectorText 和 style，分别表示一个规则的选择器部分和样式部分。

selector 部分是一个字符串，这里显然偷懒了没有设计进一步的选择器模型，我们按照选择器语法设置即可。

style 部分是一个样式表，它跟我们元素的 style 属性是一样的类型，所以我们可以像修改内联样式一样，直接改变属性修改规则中的具体 CSS 属性定义，也可以使用 cssText 这样的工具属性。

此外，CSSOM 还提供了一个非常重要的方法，来获取一个元素最终经过 CSS 计算得到的属性：

```
1 window.getComputedStyle(elt, pseudoElt);
```

 复制代码

其中第一个参数就是我们要获取属性的元素，第二个参数是可选的，用于选择伪元素。

好了，到此为止，我们可以使用 CSSOM API 自由地修改页面已经生效的样式表了。接下来，我们来一起关注一下视图的问题。

CSSOM View

CSSOM View 这一部分的 API，可以视为 DOM API 的扩展，它在原本的 Element 接口上，添加了显示相关的功能，这些功能，又可以分成三个部分：窗口部分，滚动部分和布局部分，下面我来分别带你了解一下。

窗口 API

窗口 API 用于操作浏览器窗口的位置、尺寸等。

`moveTo(x, y)` 窗口移动到屏幕的特定坐标；


`moveBy(x, y)` 窗口移动特定距离；

`resizeTo(x, y)` 改变窗口大小到特定尺寸；

`resizeBy(x, y)` 改变窗口大小特定尺寸。

此外，窗口 API 还规定了 `window.open()` 的第三个参数：

```
1 window.open("about:blank", "_blank", "width=100,height=100,left=100,right=100")
```

 复制代码

一些浏览器出于安全考虑没有实现，也不适用于移动端浏览器，这部分你仅需简单了解即可。下面我们来了解一下滚动 API。

滚动 API

要想理解滚动，首先我们必须建立一个概念，在 PC 时代，浏览器可视区域的滚动和内部元素的滚动关系是比较模糊的，但是在移动端越来越重要的今天，两者必须分开看待，两者的性能和行为都有区别。

视口滚动 API

可视区域（视口）滚动行为由 window 对象上的一组 API 控制，我们先来了解一下：

`scrollX` 是视口的属性，表示 X 方向上的当前滚动距离，有别名 `pageXOffset`；


`scrollY` 是视口的属性，表示 Y 方向上的当前滚动距离，有别名 `pageYOffset`；

`scroll(x, y)` 使得页面滚动到特定的位置，有别名 `scrollTo`，支持传入配置型参数 `{top, left}`；

`scrollBy(x, y)` 使得页面滚动特定的距离，支持传入配置型参数 `{top, left}`。

通过这些属性和方法，我们可以读取视口的滚动位置和操纵视口滚动。不过，要想监听视口滚动事件，我们需要在 `document` 对象上绑定事件监听函数：

```
1 document.addEventListener("scroll", function(event){
2     //.....
3 })
```

 复制代码

视口滚动 API 是页面的顶层容器的滚动，大部分移动端浏览器都会采用一些性能优化，它和元素滚动不完全一样，请大家一定建立这个区分的意识。

元素滚动 API

接下来我们来认识一下元素滚动 API，在 `Element` 类（参见 DOM 部分），为了支持滚动，加入了以下 API。

`scrollTop` 元素的属性，表示 Y 方向上的当前滚动距离。

`scrollLeft` 元素的属性，表示 X 方向上的当前滚动距离。

`scrollWidth` 元素的属性，表示元素内部的滚动内容的宽度，一般来说会大于等于元素宽度。

`scrollHeight` 元素的属性，表示元素内部的滚动内容的高度，一般来说会大于等于元素高度。


`scroll(x, y)` 使得元素滚动到特定的位置，有别名 `scrollTo`，支持传入配置型参数 `{top, left}`。

`scrollBy(x, y)` 使得元素滚动到特定的位置，支持传入配置型参数 `{top, left}`。

`scrollIntoView(arg)` 滚动元素所在的父元素，使得元素滚动到可见区域，可以通过 `arg` 来指定滚到中间、开始或者就近。

除此之外，可滚动的元素也支持 `scroll` 事件，我们在元素上监听它的事件即可：

```
1 element.addEventListener("scroll", function(event){
2     //.....
```

 复制代码

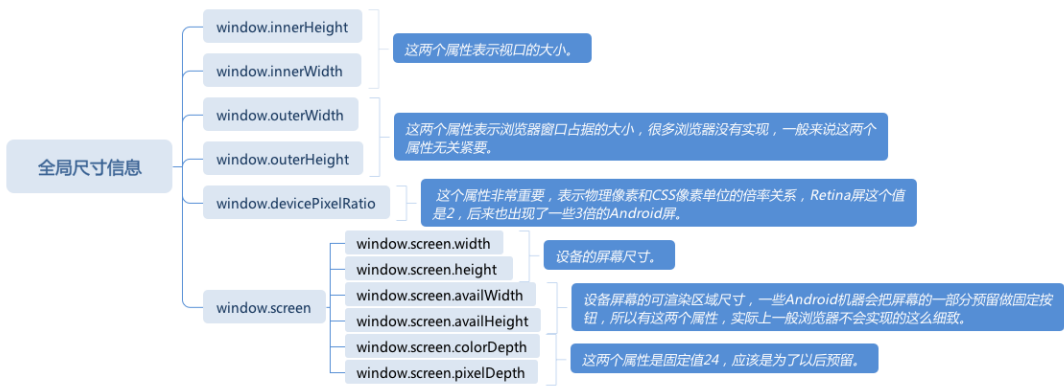
这里你需要注意一点，元素部分的 API 设计与视口滚动命名风格上略有差异，你在使用的時候不要记混。

布局 API

最后我们来介绍一下布局 API，这是整个 CSSOM 中最常用到的部分，我们同样要分成全局 API 和元素上的 API。

全局尺寸信息

window 对象上提供了一些全局的尺寸信息，它是通过属性来提供的，我们一起来了解一下这些属性。



window.innerHeight, window.innerWidth 这两个属性表示视口的大小。

window.outerWidth, window.outerHeight 这两个属性表示浏览器窗口占据的大小，很多浏览器没有实现，一般来说这两个属性无关紧要。

window.devicePixelRatio 这个属性非常重要，表示物理像素和 CSS 像素单位的倍率关系，Retina 屏这个值是 2，后来也出现了一些 3 倍的 Android 屏。

window.screen （屏幕尺寸相关的信息）

- window.screen.width, window.screen.height 设备的屏幕尺寸。
- window.screen.availWidth, window.screen.availHeight 设备屏幕的可渲染区域尺寸，一些 Android 机器会把屏幕的一部分预留做固定按钮，所以有这两个属性，实际上一概浏览器不会实现的这么细致。

- `window.screen.colorDepth`, `window.screen.pixelDepth` 这两个属性是固定值 24, 应该是为了以后预留。

虽然 `window` 有这么多相关信息, 在我看来, 我们主要使用的是 `innerHeight`、`innerWidth` 和 `devicePixelRatio` 三个属性, 因为我们前端开发工作只需要跟视口打交道, 其它信息大概了解即可。

元素的布局信息

最后我们来到了本节课一开始提到的问题, 我们是否能够取到一个元素的宽 (`width`) 和高 (`height`) 呢?

实际上, 我们首先应该从脑中消除“元素有宽高”这样的概念, 我们课程中已经多次提到了, 有些元素可能产生多个盒, 事实上, 只有盒有宽和高, 元素是没有的。

所以我们获取宽高的对象应该是“盒”, 于是 CSSOM View 为 `Element` 类添加了两个方法:

```
getClientRects();  
  
getBoundingClientRect()。
```

`getClientRects` 会返回一个列表, 里面包含元素对应的每一个盒所占据的客户端矩形区域, 这里每一个矩形区域可以用 `x`, `y`, `width`, `height` 来获取它的位置和尺寸。

`getBoundingClientRect`, 这个 API 的设计更接近我们脑海中的元素盒的概念, 它返回元素对应的所有盒的包裹的矩形区域, 需要注意, 这个 API 获取的区域会包括当 `overflow` 为 `visible` 时的子元素区域。

根据实际的精确度需要, 我们可以选择何时使用这两个 API。

这两个 API 获取的矩形区域都是相对于视口的坐标, 这意味着, 这些区域都是受滚动影响的。

如果我们要获取相对坐标, 或者包含滚动区域的坐标, 需要一点小技巧:


```
1 var offsetX = document.documentElement.getBoundingClientRect().x - element.get
```

如这段代码所示，我们只需要获取文档跟节点的位置，再相减即可得到它们的坐标。

这两个 API 的兼容性非常好，定义又非常清晰，建议你如果是用 JavaScript 实现视觉效果时，尽量使用这两个 API。

结语

今天我们一起学习了 CSSOM 这一类型的 API。我们首先就说到了，就像 HTML 和 CSS 分别承担了语义和表现的分工，DOM 和 CSSOM 也有语义和表现的分工。

CSSOM 是 CSS 的对象模型，在 W3C 标准中，它包含两个部分：描述样式表和规则等 CSS 的模型部分（CSSOM），和跟元素视图相关的 View 部分（CSSOM View）。

最后留给你一个问题，写好欢迎留言来讨论，请找一个网页，用我们今天讲的 API，把页面上的所有盒的轮廓画到一个 canvas 元素上。

猜你喜欢

Vue 开发实战

从 0 开始搭建大型 Vue 项目

戳此试读



唐金州
一点资讯前端技术专家
Ant Design Vue 作者

精选留言(12)



阿成

Look via gist: <https://gist.github.com/aimergenge/2bcf41ac4c4d2586e48ccd5cec5c9768>

```
void function () {
  const canvas = document.createElement('canvas')
```

```
canvas.width = document.documentElement.offsetWidth
canvas.height = document.documentElement.offsetHeight
```

```
canvas.style.position = 'absolute'
canvas.style.left = '0'
canvas.style.right = '0'
canvas.style.top = '0'
canvas.style.bottom = '0'
canvas.style.zIndex = '99999'
```

```
document.body.appendChild(canvas)
```

```
const ctx = canvas.getContext('2d')
draw(ctx, getAllRects())
```

```
function draw (ctx, rects) {
  let i = 0
  ctx.strokeStyle = 'red'
  window.requestAnimationFrame(_draw)
```

```
function _draw () {
  let {x, y, width, height} = rects[i++]
  ctx.strokeRect(x, y, width, height)
  if (i < rects.length) {
    window.requestAnimationFrame(_draw)
  } else {
    console.log('%cDONE', 'background-color: green; color: white; padding: 0.3em 0.5em;')
  }
}
}
```

```
function getAllRects () {
  const allElements = document.querySelectorAll('*')
  const rects = []
  const {x: htmlX, y: htmlY} = document.documentElement.getBoundingClientRect()
  allElements.forEach(element => {
    const eachElRects = Array.from(element.getClientRects()).filter(rect => {
      return rect.width || rect.height
    }).map(rect => {
      return {
        x: rect.x - htmlX,
        y: rect.y - htmlY,
        width: rect.width,
        height: rect.height
      }
    })
  })
}
```

```
}  
})  
  rects.push(...eachElRects)  
})  
  return rects  
}  
}()
```

2019-03-16



40



welkin

希望作者能讲一下虚拟dom
还有浏览器的重绘和重排
以及性能优化，跨域的常用操作(希望细致一点)
包括一些漏洞和攻击，比如xss，sql注入
还有一些技术栈，和一些对于前端需要了解的方案，比如离线方案等

2019-03-25



1



9



痕近痕远

请问老师，如何解决UI自动化测试，定位标签显示元素不可见的问题

2019-03-17



3



热心网友好宅 🐱

一直忍着没问，哪来这么多猫片 🤔

2019-04-25



1



周飞

```
<body>  
  <canvas id="rect"></canvas>  
  <script type="text/javascript">  
    const canvas = document.getElementById('rect');  
    canvas.width =document.documentElement.getBoundingClientRect().width;  
    canvas.height = document.documentElement.getBoundingClientRect().height;  
    canvas.style.position="absolute";  
    canvas.style.top=0;
```

```
canvas.style.left=0;
canvas.style.border='1px solid red';
const ctx = canvas.getContext('2d');
function travalDOM(root){
  if(root.tagName && root.tagName !=='text' && root.tagName !=='canvas'){
    const startX = root.getBoundingClientRect().x;
    const startY = root.getBoundingClientRect().y;
    const width = root.getBoundingClientRect().width;
    const height = root.getBoundingClientRect().height;
    ctx.beginPath();
    ctx.lineWidth="1";
    ctx.strokeStyle="blue";
    ctx.rect(startX,startY,width,height);
    ctx.stroke();
  }
  root.childNodes.forEach(node=>{
    travalDOM(node);
  });
}
travalDOM(document);
</script>
</body>
```

2019-04-07



1



Russell

emm~~ 我又读了一遍文档，发现了对我来说很关键词，“狭义的”。那我现在的理解是酱紫的。广义的理解，就是BOM+DOM，CSSOM是DOM扩展的一部分；如果狭义地认为DOM就是树形结构的话，就可以分出来DOM、CSSOM两部分内容了。我这样想对么？

2019-04-03



1



宋宋

前面讲浏览器渲染时有讲到，CSS经过词法分析和语法分析被解析成一颗抽象语法树。这个抽象语法树和CSSOM有什么关联么？因为很多文章都讲CSS经过词法分析和语法分析被解析成CSSOM，感觉很疑惑。

2019-03-16



1



pcxpcx_

冲冲冲

2020-03-22



西伯利亚雪橇犬

IE的edge版本scroll事件，事件中一个元素进行定位，有残影，谷歌就是平滑移动

2020-02-17



。 。 。

display:inline;的元素会不会产生盒？

作者回复: 会，而且会产生多个盒

2019-10-09



Russell

不对，我觉得我这么理解不对。。。

2019-04-03



Russell

这个咋换行啊。。。 不好意思，老师好，我想咨询浏览器API的种类。 我可以认为是，DOM，BOM，CSSOM这几类么？

作者回复: 这是几个大类，还有好多游离的

2019-04-03

