

[Code 码农网](#)

- [码农网](#)
- [码农文章](#)
- [码农社区](#)
- [码农教程](#)
- [码农网分类](#)
 - [码农软件](#)
 - [码农书籍](#)
 - [码农日报](#)
 - [码农工具](#)
- [码农网登录](#)
- [码农网导航](#)
 - [关于码农网](#)
 - [联系码农网](#)
 - [码农网导航](#)
- [码农软件](#)
- [码农书籍](#)
- [码农日报](#)
- [码农工具](#)

初探CSS对象模型（CSSOM）

栏目: [CSS](#) · 发布时间: [1年前](#)

来源: www.w3cplus.com

内容简介：今年花了不少的时间在学习DOM相关的知识，经过这段时间的学习，可以通过一些JavaScript的API操作和处理Web页面上的HTML元素。在Web中除了DOM之外还有另外一个对象模型：既然我们要探讨CSSOM是什么？那就很有必要先了解它是一个什么东东？大致的意思就是：

本文转载自：<https://www.w3cplus.com/javascript/cssom-css-typed-om.html>，本站转载出于传递更多信息之目的，版权归原作者或者来源机构所有。

今年花了不少的时间在学习DOM相关的知识，经过这段时间的学习，可以通过一些 [JavaScript](#) 的API操作和处理Web页面上的 [HTML](#) 元素。在Web中除了DOM之外还有另外一个对象模型：**CSS对象模型（即CSSOM）**。或许你已经在项目中已经用过了，只不过没有意识到这一点而以。今天这篇文章中，我们主要来一起探讨有关于CSSOM相关的特性。

CSSOM是什么？

既然我们要探讨CSSOM是什么？那就很有必要先了解它是一个什么东东？[MDN上对CSSOM的描述](#)是这样的：

The CSS Object Model is a set of APIs allowing the manipulation of CSS **from JavaScript**. **It is** much like the DOM, but **for** the CSS rather

大致的意思就是：CSSOM是一组允许JavaScript操作CSS的API。它非常类似于DOM，但是用于CSS而不是HTML。它允许用户动态读取和修改CSS样式。

[CSSOM在W3C规范中有一个独立的模块](#)，对于我们学习CSSOM还是很有帮助的，但相较于MDN而来，更难于阅读和理解。

为了更好的理解CSSOM是什么？我们先来看一个简单的示例。

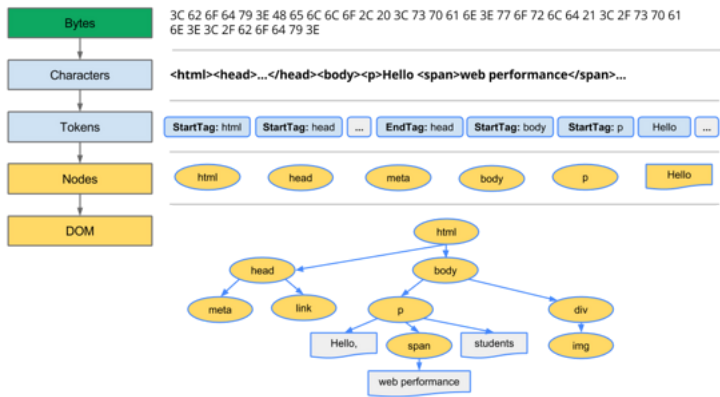
```
<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <link href="style.css" rel="stylesheet">
    <title>Critical Path</title>
  </head>
  <body>
    <p>Hello <span>web performance</span> students!</p>
    <div></div>
  </body>
</html>

// style.css
body {
  font-size: 16px
}
p {
  font-weight: bold
}
span {
  color: red
}
p span {
  display: none
}
img {
```

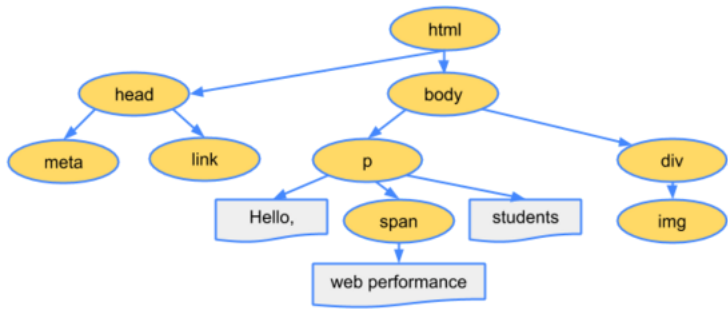


```
float: right;
}
```

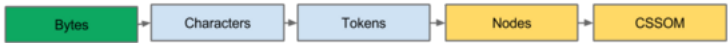
这是一个非常简单的Web页面，“包含了一些文本和一幅图片”。浏览器处理这个页面的过程如下：



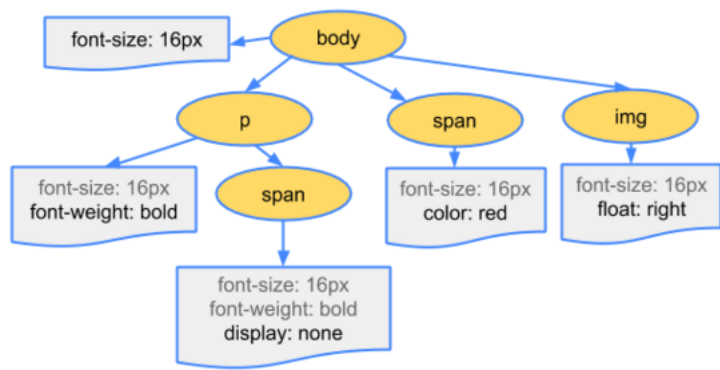
根据前面所学，其对应的DOM结构建如下：



对于Web的样式，其处理HTML有点类似，需要将收到的CSS规则转换成某种浏览器能够理解和处理的东西。因此，我们会重复HTML过程，只不过是CSS而不是HTML：



CSS字节转换成字符，接着转换成令牌和节点，最后链接到一个CSSOM的树结构中：



是不是看上去和DOM结构树类似呀。那么CSSOM为何具有树结构呢？为页面上的任何对象计算最后一组样式时，浏览器都会先从适用于该节点的最通用规则开始，比如，如果该节点是 `body` 元素的子元素，则应用所有 `body` 样式，然后通过应用更具体的规则（这里将会运用 [CSS层级相关的管理规则](#)）以递归方式优化计算的样式。

上面的示例就很形象的介绍CSSOM。

注意，上图显示的树并非是一颗完整的CSSOM树，它只显示了我们决定在样式表中替换的样式。

事实上这一过程是相当复杂的过程，在这里不做过多的介绍，如果你感兴趣的话，可以阅读下面两篇文章：

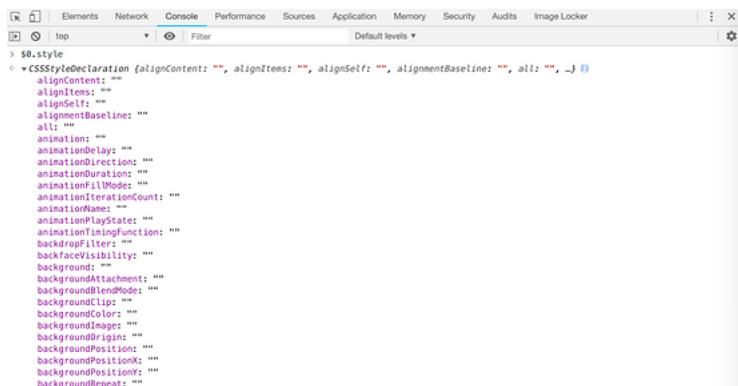
- 浏览器的渲染原理简介
- 浏览器的渲染：过程与原理
- 浏览器内核、JS引擎、页面呈现原理及其优化
- 探究CSS解析原理
- 浏览器的工作原理：新式网络浏览器幕后揭秘
- 浏览器原理
- 理解关键的渲染路径
- 关键渲染路径



但这一切都并不重要，重要的是我们可以通过这篇文章来学习CSSOM一些常见的特性，有利于我们更好的掌握CSSOM相关的特性和API所起的相关作用。

使用 `ele.style` 设置元素行内样式

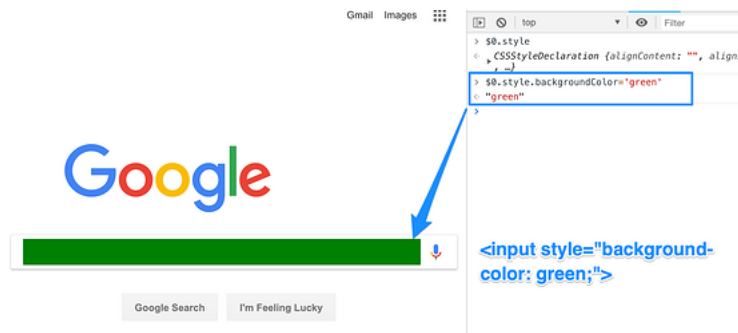
在Web开发中，我们有的时候需要动态的控制HTML元素的样式，对于这样的场景，大多数都是通过JavaScript的API来控制HTML的 `style` 属性。面对这样的场景是使用 `ele.style` 这个API来控制 `style` 对象。我们可以通过在浏览器的控制台中，输入 `$0.style` 可以输出对应元素的 `style` 所对应的属性：



比如我们要修改表单元素 `input` 的背景颜色，我们可以这么做：

```
$0.style.backgroundColor='green'
```

这样元素 `input` 自动加下了 `style` 属性，而且值为 `background-color: green`。同时表单的背景颜色变成了 `green`：



`$0` 是浏览器调试器中的一个技巧，指定是选择中的元素。在实际使用的时候，可以通过JavaScript选择器相关的API来获取你想要的DOM元素。最为常见的就是使用 `getElement*` 和 `querySelector*` API，有关于这方面更为详细的介绍，可以阅读DOM系列中的《[getElement* 和 querySelector*](#)》一文。

也就是说，我们可以使用相同的格式添加或更改页面上任何对象的CSS：`ele.style.propertyName`，其中 `ele` 指的是DOM元素，`propertyName` 指的是希望给 `ele` 元素要添加的样式属性（记住，带有 `-` 中划线的CSS属性需要改用驼峰形式，比如上面示例中的 `background-color` 属性要写成 `backgroundColor`）。

注意，在动态设置 `float` 属性时，需要使用 `cssFloat`，这是因为 `float` 是JavaScript中的一个关键词。这个有点类似于 `getAttribute()` 给HTML元素设置 `for` 属性时，需要使用 `htmlFor`。

这种方式是使用JavaScript给DOM元素设置样式最简单的方法。但是以这种方式给DOM元素设置样式有一个最大的局限性：只能给DOM元素添加内联样式。同样的，如果我们想获取一个DOM元素的内联样式中某个属性的值时，也可以采用这种方式：

```
$0.style.backgroundColor // => green
```

当然，通过上面方式获取DOM元素内联样式对应属性的值时，有个前提条件，那就是该元素定义了该内联样式。如果未指定（定义）该样式，那么将不会返回任何值：

```
$0.style.color // => ""
```

在 [CSS Houdini](#) 中的 [CSSOM](#)，我们可以使用 `.attributeStyleMap` 属性来替代 `ele.style`。可以使用 `ele.attributeStyleMap.set(property, value)` 来设置元素内联样式：

```
$0.attributeStyleMap.set('background-color', 'green')
```

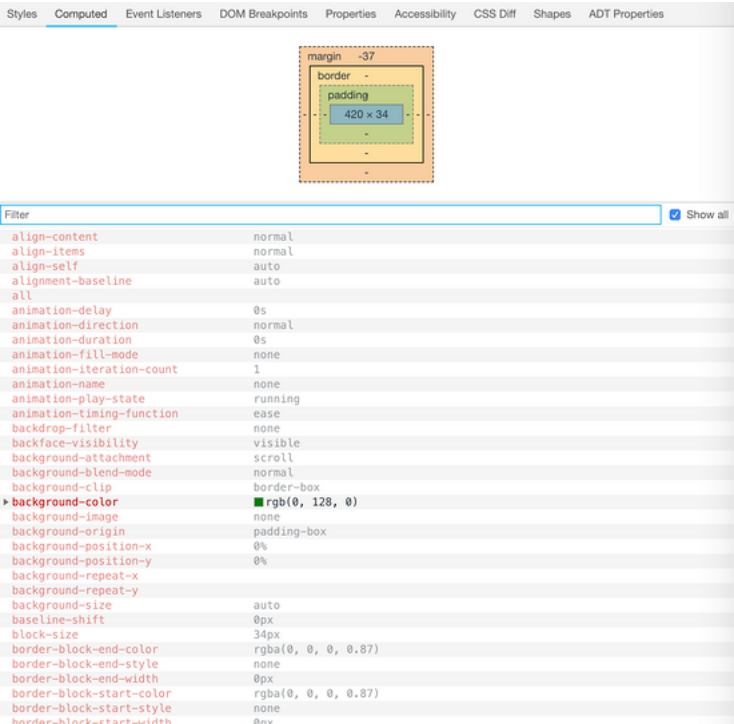
其得到的效果和 `ele.style.property = value` 等同的效果。另外，`.attributeStyleMap` 类似于Map对象，所以它们支持对象常有的一些方法，比如 `get`、`set`、`keys`、`values` 和 `entry` 等。这样让我们的工作也变得更为灵活：

```
$0.attributeStyleMap.set('background-color', 'green') // 设置background-color的值为green
$0.attributeStyleMap.get('background-color').value === 'green' // => false
$0.attributeStyleMap.has('background-color') // => true
$0.attributeStyleMap.delete('background-color') // => 删除background-color
$0.attributeStyleMap.clear() // => 删除所有样式
```



获取计算样式

我们可以使用 `window.getComputedStyle()` 方法获取元素上任何CSS的计算值。



在浏览器的 **Computed** 一项中，我们可以查看到任何元素具有的可计算的样式。如上图所示。那么我们可以通过 `window.getComputedStyle()` 方法获取相应的计算样式，比如像下面这样：

```
window.getComputedStyle($0).backgroundColor // => "rgb(0, 128, 0)"
```

上面只是获取了其中一个计算样式值。除了上述的方式，我们还可以通过其他的方式来获取，比如：

```
window.getComputedStyle(el).backgroundColor;
window.getComputedStyle(el)['background-color'];
window.getComputedStyle(el).getPropertyValue('background-color');
```

而在新CSSOM中有一个新的API，可以让我们获取计算值。比如：

```
el.computedStyleMap().get('opacity').value // => 0.5
```

注意， `window.getComputedStyle()` 和 `ele.computedStyleMap()` 的差别是，前者返回的是解析值，而后值返回计算值。类如，如果你的样式中有一个这样的值， `width: 50%`，那么在 **Typed OM** 中将保留百分值（ `width: 50%` ）;而CSSOM中返回的是解析值（ `width: 200px` ）。

上面示例中， `window.getComputedStyle()` 方法只传了一个参数，对于普通元素可以省略第二个参数，或者显示的传一个 `null` 值：

```
window.getComputedStyle(ele, null).property;
```

其实，它有一个小细节，它允许你检索伪元素的样式信息：

```
window.getComputedStyle(ele, '::before').property;
```

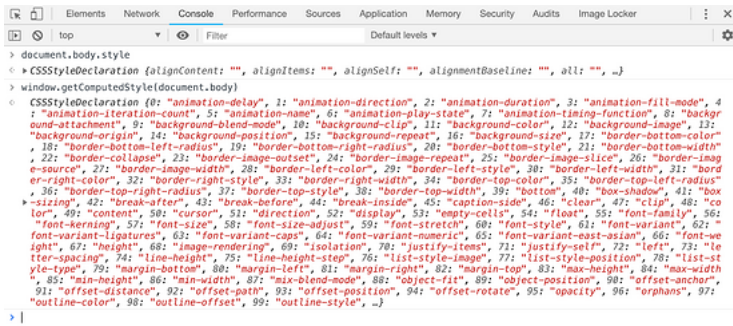
CSSStyleDeclaration 相关API

通过前面的内容我们知道如何通过 `style` 对象或使用 `getComputedStyle()` 访问样式属性，这两个其实是 `CSSStyleDeclaration` 接口。也就是说，我们可以像下面这样将 `body` 元素上返回一个 `CSSStyleDeclaration` 对象：

```
document.body.style;
window.getComputedStyle(document.body);
```

我们可以在浏览器控制台中看到上面的命令将会输出的内容：





这两者有点不同，前者其实是前面介绍的 `ele.style`，它可以获取和设置元素CSS属性的值，只不过只是给元素添加内联样式；但 `window.getComputedStyle(ele)` 获取的是只读值。

`CSSStyleDeclaration` 有几个常用的方法：

- `setProperty()`：给一个声明了CSS样式的对象设置一个新的值
- `getPropertyValue()`：用来获取CSS属性的值
- `item()`：通过下标从 `CSSStyleDeclaration` 返回一个CSS属性值
- `getPropertyPriority()`：根据传入的CSS属性，返回一个 `DOMString` 来表示该属性的权重（优先级）
- `removeProperty()`：移除 `style` 对象的一个属性

接下来分别看这几个方法是如何使用的。

setProperty()

该方法可以给CSS的属性设置一个新的值。可以像下面这样使用：

```
ele.style.setProperty(property, value, priority)
```

其中 `property` 指的是CSS属性，`value` 设置的属性的值，`priority` 允许设置CSS的权重，即 `!important`。比如下面这个示例：

```
$0.style.setProperty('color', 'red')

window.getComputedStyle($0).color // => "rgb(255, 0, 0)"
```

getPropertyValue()

该方法可以用来获取CSS属性的值，比如像下面这样：

```
$0.style.getPropertyValue('color') // => "red"
```

使用该方法时，如果 `getComputedStyle` 没有给元素指定属性时，它将返回一个空字符串：

```
$0.style.getPropertyValue('background-color') // => ""
```

item()

在 `CSSStyleDeclaration` 的 `item()` 方式可以让我们通过下标从 `CSSStyleDeclaration` 返回一个CSS属性值。其使用格式：

```
ele.style.item(index)
```

其中 `index` 是需要查找节点的索引，索引下标从 0 开始。如果我们要获取元素行内样式中所有的属性时可以通过下面的方式遍历出来：

```
for(let i = 0; i < $0.style.length; i++) {
  console.log($0.style.item(i))
}

// => clear
// => position
// => zoom
```

这里有一个小细节，`item()` 方法只要传入参数，这个方法就不会抛出异常，当传入的下标越界时会返回空字符串，当未传入参数时会抛出一个 `TypeError`。



getPropertyPriority()

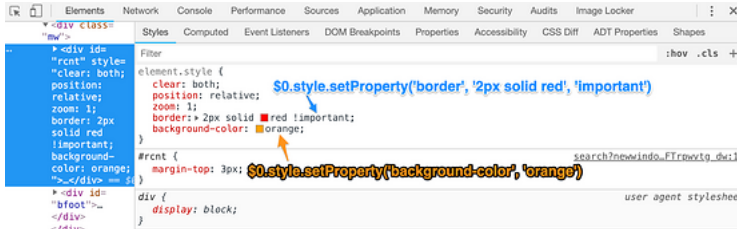
`getPropertyPriority()` 方法是一个很有意思的方法。这个方法会根据传入的CSS属性，返回一个 `DOMString` 来表示该属性的优先级。如果有的话，则返回 `important`；如果不存在的话，返回空字符串。

在介绍 `style.setProperty()` 方法的时候，我们在给其传参数的时候，第三个参数就可以指定属性的优先级。或者在原有的CSS中带有 `!important` 时，该方法也会返回 `important` 字符串。比如下面这个小示例：

```
$0.style.setProperty('border', '2px solid red', 'important')
$0.style.setProperty('background-color', 'orange')

$0.style.getPropertyPriority('border')           // => "important"
$0.style.getPropertyPriority('background-color') // => ""
```

上面的示例中，第一行代码和第二行代码使用了 `ele.style.setProperty()` 方式给元素分别设置了 `border` 和 `background-color` 两个属性，不同之处是，第一个传了第三个参数 `priority`（即 `"important"`）。这个参数就相当于在给属性值后面附加了 `!important` 关键字。



在用 `!important` 设置属性之后，使用 `ele.style.getPropertyPriority()` 方法检查该属性的优先级。前面也提到过了，如果元素的 `style` 中的属性带有 `!important` 值，也可以使用该方法进行检查。

这里有一个小细节需要注意，如果内联样式中的简写属性，比如 `margin` 属性值带有 `!important` 关键词，如果我们使用 `ele.style.getPropertyPriority()` 在检查简写属性或简写的属性的时，都将返回 `important` 的值。比如下面的代码：

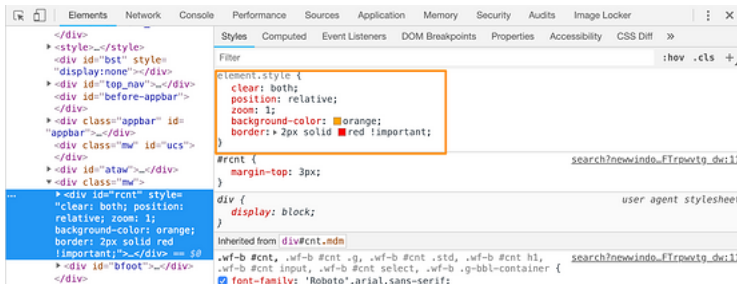
```
$0.style.getPropertyPriority('margin')           // => "important"
$0.style.getPropertyPriority('margin-top')       // => "important"
$0.style.getPropertyPriority('margin-right')     // => "important"
$0.style.getPropertyPriority('margin-bottom')    // => "important"
$0.style.getPropertyPriority('margin-left')      // => "important"
```

removeProperty()

该方法可以移除 `style` 对象的一个属性：

```
$0.style.removeProperty('margin') // => ""
$0.style.getPropertyValue('margin') // => ""
```

这个时候，DOM元素中 `style` 里的 `margin` 属性被移除了，比如下图所示的结果：



CSSStyleSheet接口

前面我们所聊的内容大部分都是关于元素内联样式（通常局限性较大）和计算样式（通常很有用，但过于具体）。接下来要聊的 `CSSStyleSheet` 相关的API是一个更有用的API，它允许检索具有可读和可写值的样式表，而不仅仅是内联样式表。简单地说，该接口代表一个单一的CSS样式表。

在写Web页面的时候，我们一直都提倡将页面的样式规则放入到一个单一（或多个）样式文件中，或者 `<style>` 标签中。这两种方式写样式都会包含一组CSS规则。每条CSS规则可以通过与之相关联的对象进行操作，这个关联对象实现了 `CSSStyleRule` 接口，而 `CSSStyleRule` 反过来实现了 `CSSRule`。`CSSStyleSheet` 允许你检测与修改和它相关联的的样式表，包括样式表的规则列表。

实际上，`CSSStyleSheet` 也实现了更为通用的 `StyleSheet` 接口。实现一个 `document` 的样式表的 `CSSStyleSheet` 列表可以过 `document.styleSheet` 属性获取(这个 `document` 通过外联样式表或内嵌的 `style` 元素定义样式)。

比如，我们可以使用下面的方式来查看一个页面（文档）中有多少样式表：

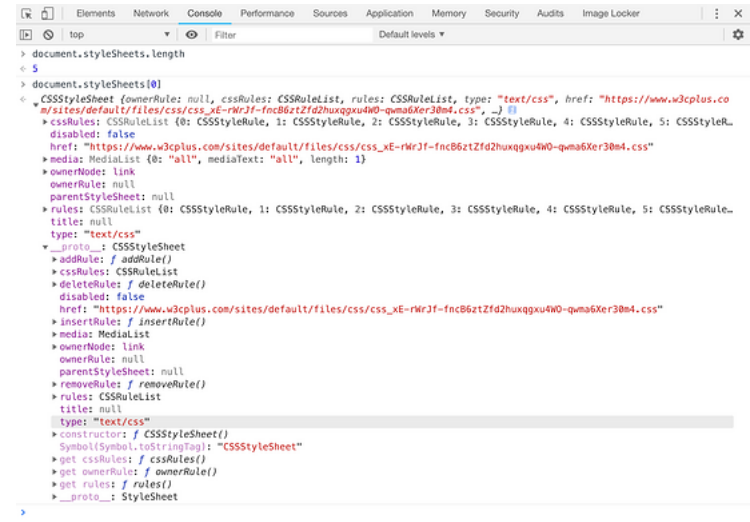
```
document.styleSheets.length // => 5
```

上面代码查询出W3cplus网站总共用了多少个CSS样式表（样式文件）：


```
<title>w3plus 网站导航, 打造前端精品教程</title>
<link type="text/css" rel="stylesheet" href="https://www.w3cplus.com/sites/default/files/css/
css_xE-RW3f-fncB6ztZfd2huxagxu4W0-qma6Xer3Bm4.css" media="all">
<link type="text/css" rel="stylesheet" href="https://www.w3cplus.com/sites/default/files/css/
css_bu6h2Zcct1-h0100tRLB8F8aBuz-sf5CnH-AUw1Sa8.css" media="all">
<link type="text/css" rel="stylesheet" href="https://www.w3cplus.com/sites/default/files/css/
css_rJUBd7Z5fGtLMSCo8MLf3X-CyF5v0lJ5MdgavSYZf.css" media="all">
<link type="text/css" rel="stylesheet" href="https://www.w3cplus.com/sites/default/files/css/
css_0ee0p8HufJ3L-UTYVWf3SMU0f9PpYpekFW_oB5J3eR0.css" media="all">
<script src="https://www.w3cplus.com/sites/default/files/blogs/2018/1807/zhi12-colorbox.js">
</script>
<script type="text/javascript" src="//cdn.jsdelivr.net/npm/jquery@1.12.2/jquery.min.js">
</script>
<link rel="stylesheet" type="text/css" href="https://www.w3cplus.com/sites/default/files/blogs/
2018/1807/zhi12/colorbox_style.css">
</head>
<body class="html front logged-in one-sidebar sidebar-second page-node featured admin-menu" style=
```

```
@media screen and (max-
width: 768px) {
  css_0ee0p8HufJ3L_oB5J3eR0-
  .main-wrap {
    margin-right: 0;
  }
  css_0ee0p8HufJ3L_oB5J3eR0-
  .main-wrap {
    margin-right: 360px;
  }
  css_0ee0p8HufJ3L_oB5J3eR0-
  .main-wrap {
    min-height: 140px;
  }
  css_0ee0p8HufJ3L_oB5J3eR0-
  *::before, *::after {
    box-sizing: inherit;
  }
}
```

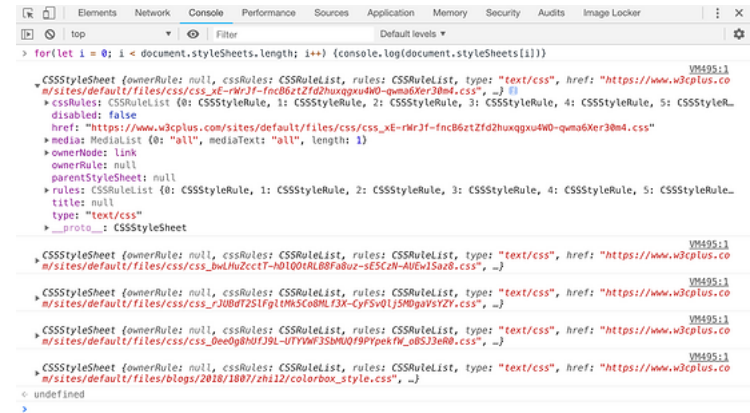
同样的，我们可以使用下标索引用文档中的任何样式表，比如：



```
> document.styleSheets.length
5
> document.styleSheets[0]
CSSStyleSheet {ownerRule: null, cssRules: CSSRuleList, rules: CSSRuleList, type: "text/css", href: "https://www.w3cplus.com/sites/default/files/css/css_xE-RW3f-fncB6ztZfd2huxagxu4W0-qma6Xer3Bm4.css", ...}
  cssRules: CSSRuleList (0: CSSStyleRule, 1: CSSStyleRule, 2: CSSStyleRule, 3: CSSStyleRule, 4: CSSStyleRule, 5: CSSStyleRule, ...), disabled: false, href: "https://www.w3cplus.com/sites/default/files/css/css_xE-RW3f-fncB6ztZfd2huxagxu4W0-qma6Xer3Bm4.css", ownerNode: Link, ...
  rules: CSSRuleList (0: CSSStyleRule, 1: CSSStyleRule, 2: CSSStyleRule, 3: CSSStyleRule, 4: CSSStyleRule, 5: CSSStyleRule, ...), title: null, type: "text/css"
  __proto__: CSSStyleSheet
    addRule(): f addRule()
    cssRules: CSSRuleList
    deleteRule(): f deleteRule()
    disabled: false
    href: "https://www.w3cplus.com/sites/default/files/css/css_xE-RW3f-fncB6ztZfd2huxagxu4W0-qma6Xer3Bm4.css"
    insertRule(): f insertRule()
    media: MediaList
    ownerNode: Link
    ownerRule: null
    parentStyleSheet: null
    removeRule(): f removeRule()
    rules: CSSRuleList
    title: null
    type: "text/css"
    constructor: f CSSStyleSheet()
    Symbol(Symbol.toStringTag): "CSSStyleSheet"
    get cssRules(): f cssRules()
    get ownerRule(): f ownerRule()
    get rules(): f rules()
    __proto__: StyleSheet
```

我们也可遍历出来所有运用到的样式表的相关信息：

```
for(let i = 0; i < document.styleSheets.length; i++) {
  console.log(document.styleSheets[i])
}
```



```
> for(let i = 0; i < document.styleSheets.length; i++) { console.log(document.styleSheets[i]) }
CSSStyleSheet {ownerRule: null, cssRules: CSSRuleList, rules: CSSRuleList, type: "text/css", href: "https://www.w3cplus.com/sites/default/files/css/css_xE-RW3f-fncB6ztZfd2huxagxu4W0-qma6Xer3Bm4.css", ...} VM4495:1
CSSStyleSheet {ownerRule: null, cssRules: CSSRuleList, rules: CSSRuleList, type: "text/css", href: "https://www.w3cplus.com/sites/default/files/css/css_bu6h2Zcct1-h0100tRLB8F8aBuz-sf5CnH-AUw1Sa8.css", ...} VM4495:1
CSSStyleSheet {ownerRule: null, cssRules: CSSRuleList, rules: CSSRuleList, type: "text/css", href: "https://www.w3cplus.com/sites/default/files/css/css_rJUBd7Z5fGtLMSCo8MLf3X-CyF5v0lJ5MdgavSYZf.css", ...} VM4495:1
CSSStyleSheet {ownerRule: null, cssRules: CSSRuleList, rules: CSSRuleList, type: "text/css", href: "https://www.w3cplus.com/sites/default/files/css/css_0ee0p8HufJ3L-UTYVWf3SMU0f9PpYpekFW_oB5J3eR0.css", ...} VM4495:1
CSSStyleSheet {ownerRule: null, cssRules: CSSRuleList, rules: CSSRuleList, type: "text/css", href: "https://www.w3cplus.com/sites/default/files/blogs/2018/1807/zhi12/colorbox_style.css", ...} VM4495:1
undefined
```

在上面两个截图中，我们都可以看到 `cssRules` 和 `ownerRule` 两个属性：

- **cssRules**：返回样式表中CSS规则的 `CSSRuleList` 对象
- **ownerRule**：如果一个样式表示通过 `@import` 规则引入 `document` 的，则 `ownerRule` 将返回那个 `CSSImportRule` 对象，否则返回 `null`

其中 `cssRules` 属性是较为有用的。此属性提供样式表中包含的所有CSS规则（包括声明块、`at-rules` 和媒体查询等）的列表。

(index)	selectorText	style	styleMap	type	cssText	parentStyleSheet	Value
0	"#autocomplete"	CSSStyleDeclaration	StylePropertyMap	1	"#autocomplete { border: 1px solid; ...	null	CSSStyleSheet
1	"#autocomplete ul"	CSSStyleDeclaration	StylePropertyMap	1	"#autocomplete ul { list-style-type: none; ...	null	CSSStyleSheet
2	"#autocomplete li"	CSSStyleDeclaration	StylePropertyMap	1	"#autocomplete li { background-color: #fff; ...	null	CSSStyleSheet
3	"html.js input.form-autocomplete"	CSSStyleDeclaration	StylePropertyMap	1	"html.js input.form-autocomplete { ...	null	CSSStyleSheet
4	"html.js input.throbbing"	CSSStyleDeclaration	StylePropertyMap	1	"html.js input.throbbing { background-color: #fff; ...	null	CSSStyleSheet
5	"html.js fieldset.collapsed"	CSSStyleDeclaration	StylePropertyMap	1	"html.js fieldset.collapsed { border: 1px solid #ccc; ...	null	CSSStyleSheet
6	"html.js fieldset.collapsed.fieldset-wrapper"	CSSStyleDeclaration	StylePropertyMap	1	"html.js fieldset.collapsed.fieldset-wrapper { ...	null	CSSStyleSheet
7	"fieldset.collapsible"	CSSStyleDeclaration	StylePropertyMap	1	"fieldset.collapsible { position: relative; ...	null	CSSStyleSheet
8	"form-textarea-wrapper"	CSSStyleDeclaration	StylePropertyMap	1	"form-textarea-wrapper textarea { ...	null	CSSStyleSheet
9	".form-textarea-wrapper"	CSSStyleDeclaration	StylePropertyMap	1	".form-textarea-wrapper { ...	null	CSSStyleSheet
10	".resizable-textarea.grippie"	CSSStyleDeclaration	StylePropertyMap	1	".resizable-textarea.grippie { background-color: #fff; ...	null	CSSStyleSheet
11	"body.drag"	CSSStyleDeclaration	StylePropertyMap	1	"body.drag { cursor: move; ...	null	CSSStyleSheet
12	".draggable a.tabledrag-handle"	CSSStyleDeclaration	StylePropertyMap	1	".draggable a.tabledrag-handle { color: #000; ...	null	CSSStyleSheet
13	"a.tabledrag-handle"	CSSStyleDeclaration	StylePropertyMap	1	"a.tabledrag-handle { text-decoration: underline; ...	null	CSSStyleSheet
14	"a.tabledrag-handle.handle"	CSSStyleDeclaration	StylePropertyMap	1	"a.tabledrag-handle.handle { background-color: #fff; ...	null	CSSStyleSheet
15	"a.tabledrag-handle.handle:hover"	CSSStyleDeclaration	StylePropertyMap	1	"a.tabledrag-handle.handle:hover { background-color: #000; ...	null	CSSStyleSheet
16	"div.indentation"	CSSStyleDeclaration	StylePropertyMap	1	"div.indentation { float: left; width: 20px; ...	null	CSSStyleSheet
17	"div.tree-child"	CSSStyleDeclaration	StylePropertyMap	1	"div.tree-child { background-color: #fff; ...	null	CSSStyleSheet
18	"div.tree-child-last"	CSSStyleDeclaration	StylePropertyMap	1	"div.tree-child-last { background-color: #fff; ...	null	CSSStyleSheet
19	"div.tree-child-horizontal"	CSSStyleDeclaration	StylePropertyMap	1	"div.tree-child-horizontal { background-color: #fff; ...	null	CSSStyleSheet
20	".tabledrag-toggle-weight-wrapper"	CSSStyleDeclaration	StylePropertyMap	1	".tabledrag-toggle-weight-wrapper { background-color: #fff; ...	null	CSSStyleSheet
21	"table.sticky-header"	CSSStyleDeclaration	StylePropertyMap	1	"table.sticky-header { background-color: #fff; ...	null	CSSStyleSheet
22	".progress .bar"	CSSStyleDeclaration	StylePropertyMap	1	".progress .bar { background-color: #000; ...	null	CSSStyleSheet
23	".progress .filled"	CSSStyleDeclaration	StylePropertyMap	1	".progress .filled { background-color: #000; ...	null	CSSStyleSheet
24	".progress .percentage"	CSSStyleDeclaration	StylePropertyMap	1	".progress .percentage { float: right; width: 50px; ...	null	CSSStyleSheet
25	"#ajax-progress"	CSSStyleDeclaration	StylePropertyMap	1	"#ajax-progress { display: inline-block; width: 100px; ...	null	CSSStyleSheet
26	".ajax-progress.throbber"	CSSStyleDeclaration	StylePropertyMap	1	".ajax-progress.throbber { background-color: #000; ...	null	CSSStyleSheet
27	"#ajax-progress.message"	CSSStyleDeclaration	StylePropertyMap	1	"#ajax-progress.message { padding-left: 5px; ...	null	CSSStyleSheet
28	"tr.ajax-progress.throbber"	CSSStyleDeclaration	StylePropertyMap	1	"tr.ajax-progress.throbber { margin: 0; ...	null	CSSStyleSheet
29	"#ajax-progress-bar"	CSSStyleDeclaration	StylePropertyMap	1	"#ajax-progress-bar { width: 100px; height: 10px; ...	null	CSSStyleSheet
30	".container-inline"	CSSStyleDeclaration	StylePropertyMap	1	".container-inline { display: inline-block; vertical-align: top; ...	null	CSSStyleSheet
31	".container-inline.fieldset-wrapper"	CSSStyleDeclaration	StylePropertyMap	1	".container-inline.fieldset-wrapper { border: 1px solid #ccc; ...	null	CSSStyleSheet
32	".nowrap"	CSSStyleDeclaration	StylePropertyMap	1	".nowrap { white-space: nowrap; ...	null	CSSStyleSheet
33	"html.js .js-hide"	CSSStyleDeclaration	StylePropertyMap	1	"html.js .js-hide { display: none; ...	null	CSSStyleSheet
34	".element-hidden"	CSSStyleDeclaration	StylePropertyMap	1	".element-hidden { display: none; ...	null	CSSStyleSheet
35	".element-invisible"	CSSStyleDeclaration	StylePropertyMap	1	".element-invisible { clip: rect(1px 1px 1px 1px); ...	null	CSSStyleSheet
36	".element-invisible.element-focusable"	CSSStyleDeclaration	StylePropertyMap	1	".element-invisible.element-focusable { outline: 1px solid #000; ...	null	CSSStyleSheet
37	".clearfix:after"	CSSStyleDeclaration	StylePropertyMap	1	".clearfix:after { content: ' '; display: block; clear: both; height: 0; ...	null	CSSStyleSheet
38	"#html .clearfix"	CSSStyleDeclaration	StylePropertyMap	1	"#html .clearfix { height: 1px; ...	null	CSSStyleSheet
39	".first-child"	CSSStyleDeclaration	StylePropertyMap	1	".first-child { html { clear: both; ...	null	CSSStyleSheet
40	"ul.menu"	CSSStyleDeclaration	StylePropertyMap	1	"ul.menu { border: none; list-style-type: none; ...	null	CSSStyleSheet
41	"ul.menu li"	CSSStyleDeclaration	StylePropertyMap	1	"ul.menu li { margin: 0px 0px 0px 0px; ...	null	CSSStyleSheet
42	"ul li.expanded"	CSSStyleDeclaration	StylePropertyMap	1	"ul li.expanded { list-style-type: none; ...	null	CSSStyleSheet
43	"ul li.collapsed"	CSSStyleDeclaration	StylePropertyMap	1	"ul li.collapsed { list-style-type: none; ...	null	CSSStyleSheet

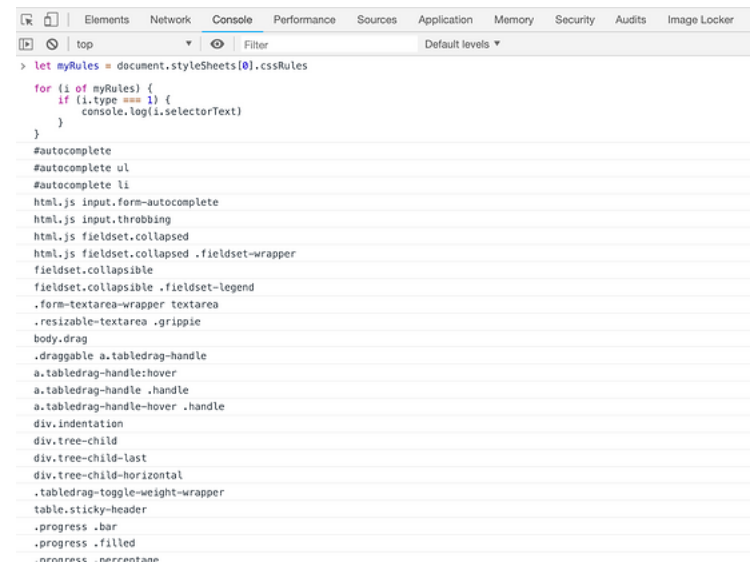
在这个示例中，总共有 116 个CSS规则。

在接下来的部分中，我们将详细介绍如何使用这个API来操作和读取外部样式表中的样式。比如我们要把第一个 .css 文件中所有选择器打印出来，我们就可以像下面这样做：

```
let myRules = document.styleSheets[0].cssRules

for (i of myRules) {
  if (i.type === 1) {
    console.log(i.selectorText)
  }
}
```

打印出来的结果类似下图这样：



在上面的代码中需要注意两件事。首先，把第一个样式表中的 cssRules 对象赋值给一个变量缓存起来，然后使用 for... of 循环来循环该对象中的所有规则，检查每个规则的类型。在这种情况下，我们需要的规则类型(type)是 1，它表示 STYLE_RULE 常量。其他常量包括 IMPORT_RULE (对应的 type = 3)、MEDIA_RULE (对应的 type=4)和 KEYFRAMES_RULE (对应的 type=7)。更多的类型如下图所示，[也可以在MDN上查阅](#)：

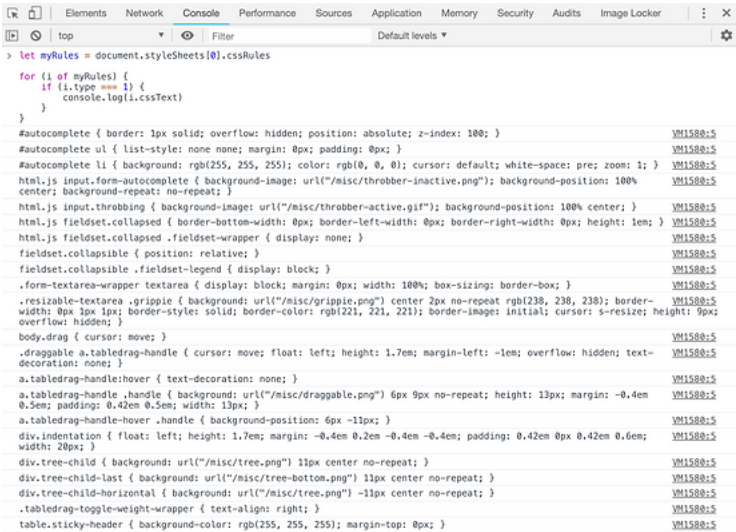
Type	Value	Rule-specific interface
CSSRule.STYLE_RULE	1	CSSStyleRule
CSSRule.MEDIA_RULE	4	CSSMediaRule
CSSRule.FONT_FACE_RULE	5	CSSFontFaceRule
CSSRule.PAGE_RULE	6	CSSPageRule
CSSRule.IMPORT_RULE	3	CSSImportRule : ❸IDL: nsIDOMCSSImportRule
CSSRule.CHARSET_RULE	2	CSSCharsetRule ❹
CSSRule.UNKNOWN_RULE	0	CSSUnknownRule ❺
CSSRule.KEYFRAMES_RULE	7	CSSKeyframesRule [1] ❶
CSSRule.KEYFRAME_RULE	8	CSSKeyframeRule [1] ❷
Reserved for future use	9	Should be used to define color profiles in the future
CSSRule.NAMESPACE_RULE	10	CSSNamespaceRule ❸
CSSRule.COUNTER_STYLE_RULE	11	CSSCounterStyleRule ❹
CSSRule.SUPPORTS_RULE	12	CSSSupportsRule
CSSRule.DOCUMENT_RULE	13	CSSDocumentRule ❶
CSSRule.FONT_FEATURE_VALUES_RULE	14	CSSFontFeatureValuesRule
CSSRule.VIEWPORT_RULE	15	CSSViewportRule ❷
CSSRule.REGION_STYLE_RULE	16	CSSRegionStyleRule ❸

同样的，我们可以使用类似的方法打印出 @media 和 @keyframes 里面相关的信息。也可以以类似方式打印出类似 selectorText 相关的信息，比如 style、styleMap 和 cssText 等。比如：

```
let myRules = document.styleSheets[0].cssRules

for (i of myRules) {
  if (i.type === 1) {
    console.log(i.cssText)
  }
}
```

打印出来的结果类似下图：



在 CSSStyleSheet 接口中除了上面提到的两个常见属性之外，还有两个方法，允许你从样式表中添加或删除整个规则。

- insertRule：向样式表中插入一条新规则
- deleteRule：从当前样式表对象中删除指定的样式规则

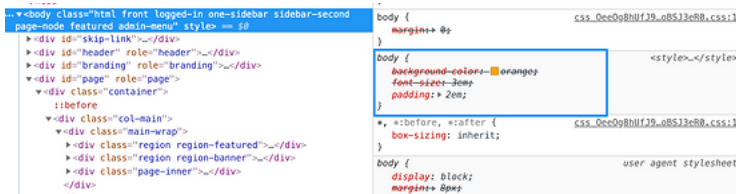
比如我们要给第一个样式表中添加一条新的样式规则：

```
let firstStylesheet = document.styleSheets[0]
console.log(firstStylesheet.cssRules.length) // => 116

firstStylesheet.insertRule(`
body {
  background-color: orange;
  font-size: 3em;
  padding: 2em;
},
firstStylesheet.cssRules.length
`)
```

这个时候在样式表中添加了下面的样式：





我也可以通过下面的代码，来验证：

```
for (i of firstStylesheet.cssRules) {
  if (i.type === 1) {
    console.log(i.cssText)
  }
}
```



其 `cssRules` 的 `length` 值由 116 变成 117：

```
console.log(firstStylesheet.cssRules.length) // => 117
```

`stylesheet.insertRule()` 方法接受两值参数：

```
rule
index
```

注意，对于普通样式规则来说，要插入的字符串应该包含选择器和样式声明。对于 `@` 规则来说，要插入的字符串应该包含 `@` 标识符和样式规则的内容。另外，`index` 未设置的话，则默认为 0，新添加的 `rule` 将会插入到样式表的最前面，如果 `index` 索引值恰好大于 `cssRules.length`，将会抛出一个错误。

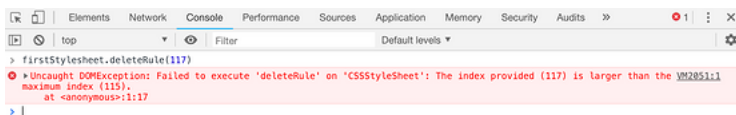
`deleteRule()` 方法相对来说更为容易，它只接受一个参数 `index`。`index` 就是一个数字，用来指定样式规则的位置。作为参数传入的所选 `index` 必须小于 `cssRules.length`，否则将抛出错误。比如我们现在要删除刚才新增加的样式规则：

```
body {
  background-color: orange;
  font-size: 3em;
  padding: 2em;
}
```

我们就可以像下面这样来删除这条规则：

```
firstStylesheet.deleteRule(116)
```

对应的样式规则就删除了。如果把 116 换成 117 就会报错：



CSSOM的未来

在介绍 `ele.style` 这个API的时候，简单的提到过，[CSS Houdini](#) 中提到了新的CSSOM（即 [CSS Typed OM](#)）。新的CSSOM相关的API能提供更大的优势。有关于这方面的介绍，可以阅读Google开发者文档中 [@Eric Bidelman](#) 写的 [博文](#)。

总结

通过JavaScript中的相关API来操作CSS样式表肯定不是每个项目中都会用到的。但文章中提到的一些API的的确确可以帮助我们实现一些复杂交互。因此，掌握这些API是很有必要的，同时能加强我们处理业务的能力。

扩展阅读

- [An Introduction and Guide to the CSS Object Model](#)
- [Working with the new CSS Typed Object Model](#)
- [CSS Object Model](#)
- [更高效、更安全地操作 CSSOM：CSS Typed OM](#)
- [CSSOM视图模式\(CSSOM View Module\)相关整理](#)

以上就是本文的全部内容，希望对大家的学习有所帮助，也希望大家多多支持 [码农网](#)



关注我们，获取更多IT资讯^_^

为你推荐:

- [Chrome 66 Beta 发布: CSS 类型对象模型变化](#)
- [更高效、更安全地操作 CSSOM: CSS Typed OM](#)
- [CSS - 盒模型](#)
- [初探CSS 选择器Level 4](#)

相关软件推荐:

- [分布式对象图 NetworkObjects](#)
- [面向对象的WebAPI框架 xxi-hex](#)
- [基于 Moya 和 SwiftyJSON 的快速解析模型工具 MoyaMapper](#)
- [小型 tcp 堆栈 picoTcp](#)
- [创建 Atomic CSS 工具 Atomizer](#)

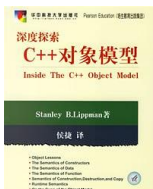
[查看所有标签](#)

本站部分资源来源于网络，本站转载出于传递更多信息之目的，版权归原作者或者来源机构所有，如转载稿涉及版权问题，请[联系我们](#)。

热门标签

[erlang模型](#) [javascript对象](#) [cssom](#) [php面向对象](#) [perl 面向对象](#) [javascript浏览器对象](#) [css](#) [groovy 泛型](#) [css-calc](#) [css-counter](#) [css-content](#) [css-filters](#) [css-grid](#) [yaml-css](#) [css-modules](#) [css-reflections](#) [paper-css](#) [css-centering](#) [css-classes](#) [css-layout](#) [nginx模块](#) [css-frameworks](#) [css-sprites](#) [css-import](#) [pure-css](#) [css-hyphens](#) [css-preprocessor](#) [css-rem](#) [css-columns](#) [css-combinators](#) [css-flexbox](#) [css-grids](#) [css-height](#) [jquery-css](#) [tachyons-css](#) [css-cascade](#) [nginx 模块](#) [perl 模块](#) [oracle 模拟](#) [hibernate 一对一](#) [print-css](#) [html css](#) [css 3.0](#) [css 3](#) [unsemantic-css](#) [normalize-css](#) [webflow-css](#) [kube-css](#) [toast-css](#) [compass-css](#)

码农书籍



深度探索C++对象模型

[美] Stanley B. Lippman / 侯捷 / 华中科技大学出版社 / 2001-5 / 54.00元

这本书探索“对象导向程序所支持的C++对象模型”下的程序行为。对于“对象导向性质之基础实现技术”以及“各种性质背后的隐含利益交换”提供一个清楚的认识。检验由程序变形所带来的效率冲击。提供丰富的程序范例、图片，以及对象导向观念和底层对象模型之间的效率测量。一起来看看[《深度探索C++对象模型》](#)这本书的介绍吧!

码农工具



CSS 压缩/解压工具

在线压缩/解压 CSS 代码



正则表达式在线测试

正则表达式在线测试



HEX HSV 转换工具

HEX HSV 互换工具

- New
 - 文章
 - 话题
 - 教程

- · [对2019年各个国家0_day漏洞使用情况的介绍](#)
- · [利好以太坊的 tBTC 采用究竟要多久?](#)
- · [DCEP、区块链技术应用落地之道](#)
- · [全球首个基于容器的云原生5G网络即将推出，Kube-OVN参与其中](#)
- · [本地内核调试神器：livekd 使用总结](#)
- · [复杂业务如何保证Flutter的高性能高流畅度?](#)

- 阅读排行

榜

- 月
- 周
- 日

- · [使用V2Ray实现科学爱国 — Chrarcadia](#)
- · [AI 换脸](#)
- · [没有美区的Apple ID 下载 Potatso Lite 的超简单办法 \(ShadowRocket的完美替代\)](#)
- · [如何确定ARIMA模型中参数p、d、q](#)
- · [Python GUI教程 \(十六\) : 在PyQt5中美化和装扮图形界面](#)
- · [PHP preg_match\(\) 函数](#)

关注 码农网 公众号



码农网最新文章

[对2019年各个国家0_day漏洞使用情况的介绍](#)

[利好以太坊的 tBTC 采用究竟要多久?](#)

↑

↓

[DCEP、区块链技术应用落地之道](#)

[全球首个基于容器的云原生5G网络即将推出，Kube-OVN参与其中](#)

[本地内核调试神器：livekd 使用总结](#)

码农网最新帖子

[Kong 发布 API 协作设计工具 Insomnia Designer](#)

[使用 Wayland 在 Android 10 上运行 KDE Plasma 5](#)

[Oracle 提交补丁，可使 Linux 内核引导提速 6%–49%](#)

[微软开源 Rust/WinRT，方便使用 Rust 构建 Windows 应用](#)

[2020年5月03日 程序员老黄历，宜:拒绝996](#)

码农网关键词

[码农网](#) [码农](#) [程序员](#) [码农教程](#) [码农社区](#) [码农工具](#) [码农日报](#) [码农头条](#) [码农网论坛](#) [码农网源码](#) [码农网官网](#)