

CSS 操作

CSS 与 JavaScript 是两个有着明确分工的领域，前者负责页面的视觉效果，后者负责与用户的行为互动。但是，它们毕竟同属网页开发的前端，因此不可避免有着交叉和互相配合。本章介绍如何通过 JavaScript 操作 CSS。

HTML 元素的 style 属性

操作 CSS 样式最简单的方法，就是使用网页元素节点的`getAttribute()`方法、`setAttribute()`方法和`removeAttribute()`方法，直接读写或删除网页元素的`style`属性。

```
```javascript
div.setAttribute(
 'style',
 'background-color:red;' + 'border:1px solid black;'
);
```
```

上面的代码相当于下面的 HTML 代码。

```
```html
<div style="background-color:red; border:1px solid black;" />
```
```

`style`不仅可以使使用字符串读写，它本身还是一个对象，部署了`CSSStyleDeclaration`接口（详见下面的介绍），可以直接读写个别属性。

```
```javascript
e.style.fontSize = '18px';
e.style.color = 'black';
```
```

CSSStyleDeclaration 接口

简介

CSSStyleDeclaration 接口用来操作元素的样式。三个地方部署了这个接口。

- 元素节点的`style`属性（`Element.style`）
- `CSSStyle`实例的`style`属性
- `window.getComputedStyle()`的返回值

CSSStyleDeclaration 接口可以直接读写 CSS 的样式属性，不过，连词号需要变成骆驼拼写法。

```
```javascript
var divStyle = document.querySelector('div').style;
```

```
divStyle.backgroundColor = 'red';
divStyle.border = '1px solid black';
divStyle.width = '100px';
divStyle.height = '100px';
divStyle.fontSize = '10em';
```

```
divStyle.backgroundColor // red
divStyle.border // 1px solid black
divStyle.height // 100px
divStyle.width // 100px
```

```

上面代码中，`style`属性的值是一个 `CSSStyleDeclaration` 实例。这个对象所包含的属性与 CSS 规则一一对应，但是名字需要改写，比如`background-color`写成`backgroundColor`。改写的规则是将横杠从 CSS 属性名中去除，然后将横杠后的第一个字母大写。如果 CSS 属性名是 JavaScript 保留字，则规则名之前需要加上字符串`css`，比如`float`写成`cssFloat`。

注意，该对象的属性值都是字符串，设置时必须包括单位，但是不含规则结尾的分号。比如，`divStyle.width`不能写为`100`，而要写为`100px`。

另外，`Element.style`返回的只是行内样式，并不是该元素的全部样式。通过样式表设置的样式，或者从父元素继承的样式，无法通过这个属性得到。元素的全部样式要通过`window.getComputedStyle()`得到。

CSSStyleDeclaration 实例属性

** (1) CSSStyleDeclaration.cssText**

`CSSStyleDeclaration.cssText`属性用来读写当前规则的所有样式声明文本。

```
```javascript
var divStyle = document.querySelector('div').style;

divStyle.cssText = 'background-color: red;'
 + 'border: 1px solid black;'
 + 'height: 100px;'
 + 'width: 100px;';
```
```

注意，`cssText`的属性值不用改写 CSS 属性名。

删除一个元素的所有行内样式，最简便的方法就是设置`cssText`为空字符串。

```
```javascript
divStyle.cssText = '';
```
```

** (2) CSSStyleDeclaration.length**

`CSSStyleDeclaration.length`属性返回一个整数值，表示当前规则包含多少条样式声明。

```
```javascript
// HTML 代码如下
// <div id="myDiv"
// style="height: 1px;width: 100%;background-color: #CA1;"
// ></div>
var myDiv = document.getElementById('myDiv');
var divStyle = myDiv.style;
divStyle.length // 3
```
```

上面代码中，`myDiv`元素的行内样式共包含3条样式规则。

**** (3) CSSStyleDeclaration.parentRule****

`CSSStyleDeclaration.parentRule`属性返回当前规则所属的那个样式块（CSSRule 实例）。如果不存在所属的样式块，该属性返回`null`。

该属性只读，且只在使用 CSSRule 接口时有意义。

```
```javascript
var declaration = document.styleSheets[0].rules[0].style;
declaration.parentRule === document.styleSheets[0].rules[0]
// true
```
```

CSSStyleDeclaration 实例方法

**** (1) CSSStyleDeclaration.getPropertyPriority()****

`CSSStyleDeclaration.getPropertyPriority`方法接受 CSS 样式的属性名作为参数，返回一个字符串，表示有没有设置`important`优先级。如果有就返回`important`，否则返回空字符串。

```
```javascript
// HTML 代码为
// <div id="myDiv" style="margin: 10px!important; color: red;">
var style = document.getElementById('myDiv').style;
style.margin // "10px"
style.getPropertyPriority('margin') // "important"
style.getPropertyPriority('color') // ""
```
```

上面代码中，`margin`属性有`important`优先级，`color`属性没有。

**** (2) CSSStyleDeclaration.getPropertyValue()****

`CSSStyleDeclaration.getPropertyValue`方法接受 CSS 样式属性名作为参数，返回一个字符串，表示该属性的属性值。

```
```javascript
// HTML 代码为
// <div id="myDiv" style="margin: 10px!important; color: red;" />
var style = document.getElementById('myDiv').style;
style.margin // "10px"
style.getPropertyValue("margin") // "10px"
```
```

**** (3) CSSStyleDeclaration.item()**

`CSSStyleDeclaration.item`方法接受一个整数值作为参数，返回该位置的 CSS 属性名。

```
```javascript
// HTML 代码为
// <div id="myDiv" style="color: red; background-color: white;" />
var style = document.getElementById('myDiv').style;
style.item(0) // "color"
style.item(1) // "background-color"
```
```

上面代码中，`0`号位置的 CSS 属性名是`color`，`1`号位置的 CSS 属性名是`background-color`。

如果没有提供参数，这个方法会报错。如果参数值超过实际的属性数目，这个方法返回一个空字符串值。

**** (4) CSSStyleDeclaration.removeProperty()**

`CSSStyleDeclaration.removeProperty`方法接受一个属性名作为参数，在 CSS 规则里面移除这个属性，返回这个属性原来的值。

```
```javascript
// HTML 代码为
// <div id="myDiv" style="color: red; background-color: white;">
// 111
// </div>
var style = document.getElementById('myDiv').style;
style.removeProperty('color') // 'red'
// HTML 代码变为
// <div id="myDiv" style="background-color: white;">
```
```

上面代码中，删除`color`属性以后，字体颜色从红色变成默认颜色。

**** (5) CSSStyleDeclaration.setProperty()**

`cssStyleDeclaration.setProperty`方法用来设置新的 CSS 属性。该方法没有返回值。

该方法可以接受三个参数。

- 第一个参数：属性名，该参数是必需的。
- 第二个参数：属性值，该参数可选。如果省略，则参数值默认为空字符串。
- 第三个参数：优先级，该参数可选。如果设置，唯一的合法值是`important`，表示 CSS 规则里面的`!important`。

```
```javascript
// HTML 代码为
// <div id="myDiv" style="color: red; background-color: white;">
// 111
// </div>
var style = document.getElementById('myDiv').style;
style.setProperty('border', '1px solid blue');
```
```

上面代码执行后，`myDiv`元素就会出现蓝色的边框。

CSS 模块的侦测

CSS 的规格发展太快，新的模块层出不穷。不同浏览器的不同版本，对 CSS 模块的支持情况都不一样。有时候，需要知道当前浏览器是否支持某个模块，这就叫做“CSS模块的侦测”。

一个比较普遍适用的方法是，判断元素的`style`对象的某个属性值是否为字符串。

```
```javascript
typeof element.style.animationName === 'string';
typeof element.style.transform === 'string';
```
```

如果该 CSS 属性确实存在，会返回一个字符串。即使该属性实际上并未设置，也会返回一个空字符串。如果该属性不存在，则会返回`undefined`。

```
```javascript
document.body.style['maxWidth'] // ""
document.body.style['maximumWidth'] // undefined
```
```

上面代码说明，这个浏览器支持`max-width`属性，但是不支持`maximum-width`属性。

注意，不管 CSS 属性名的写法带不带连词线，`style`属性上都能反映出该属性是否存在。

```
```javascript
document.body.style['backgroundColor'] // ""
document.body.style['background-color'] // ""
```
```

另外，使用的时候，需要把不同浏览器的 CSS 前缀也考虑进去。

```
```javascript
var content = document.getElementById('content');
typeof content.style['webkitAnimation'] === 'string'
```
```

这种侦测方法可以写成一个函数。

```
```javascript
function isPropertySupported(property) {
 if (property in document.body.style) return true;
 var prefixes = ['Moz', 'Webkit', 'O', 'ms', 'Khtml'];
 var prefProperty = property.charAt(0).toUpperCase() + property.substr(1);

 for(var i = 0; i < prefixes.length; i++){
 if((prefixes[i] + prefProperty) in document.body.style) return true;
 }

 return false;
}

isPropertySupported('background-clip')
// true
```
```

CSS 对象

浏览器原生提供 CSS 对象，为 JavaScript 操作 CSS 提供一些工具方法。

这个对象目前有两个静态方法。

CSS.escape()

`CSS.escape` 方法用于转义 CSS 选择器里面的特殊字符。

```
```html
<div id="foo#bar">
```
```

上面代码中，该元素的 `id` 属性包含一个 `#` 号，该字符在 CSS 选择器里面有特殊含义。不能直接写成 `document.querySelector('#foo#bar')`，只能写成 `document.querySelector('#foo\\#bar')`。这里必须使用双斜杠的原因是，单引号字符串本身会转义一次斜杠。

`CSS.escape` 方法就用来转义那些特殊字符。

```
```javascript
document.querySelector('#' + CSS.escape('foo#bar'))
```
```

```
### CSS.supports()
```

`CSS.supports`方法返回一个布尔值，表示当前环境是否支持某一句 CSS 规则。

它的参数有两种写法，一种是第一个参数是属性名，第二个参数是属性值；另一种是整个参数就是一行完整的 CSS 语句。

```
```javascript
// 第一种写法
CSS.supports('transform-origin', '5px') // true

// 第二种写法
CSS.supports('display: table-cell') // true
```
```

注意，第二种写法的参数结尾不能带有分号，否则结果不准确。

```
```javascript
CSS.supports('display: table-cell;') // false
```
```

```
## window.getComputedStyle()
```

行内样式 (inline style) 具有最高的优先级，改变行内样式，通常会立即反映出来。但是，网页元素最终的样式是综合各种规则计算出来的。因此，如果想得到元素实际的样式，只读取行内样式是不够的，需要得到浏览器最终计算出来的样式规则。

`window.getComputedStyle`方法，就用来返回浏览器计算后得到的最终规则。它接受一个节点对象作为参数，返回一个 `CSSStyleDeclaration` 实例，包含了指定节点的最终样式信息。所谓“最终样式信息”，指的是各种 CSS 规则叠加后的结果。

```
```javascript
var div = document.querySelector('div');
var styleObj = window.getComputedStyle(div);
styleObj.backgroundColor
```
```

上面代码中，得到的背景色就是 `div` 元素真正的背景色。

注意，`CSSStyleDeclaration` 实例是一个活的对象，任何对于样式的修改，会实时反映到这个实例上面。另外，这个实例是只读的。

`getComputedStyle`方法还可以接受第二个参数，表示当前元素的伪元素（比如 `:before`、`:after`、`:first-line`、`:first-letter` 等）。

```
```javascript
var result = window.getComputedStyle(div, ':before');
```

```
...
```

下面的例子是如何获取元素的高度。

```
```javascript
var elem = document.getElementById('elem-container');
var styleObj = window.getComputedStyle(elem, null)
var height = styleObj.height;
// 等同于
var height = styleObj['height'];
var height = styleObj.getPropertyValue('height');
```
```

上面代码得到的`height`属性，是浏览器最终渲染出来的高度，比其他方法得到的高度更可靠。由于`styleObj`是`CSSStyleDeclaration`实例，所以可以使用各种`CSSStyleDeclaration`的实例属性和方法。

有几点需要注意。

- `CSSStyleDeclaration`实例返回的CSS值都是绝对单位。比如，长度都是像素单位（返回值包括`px`后缀），颜色是`rgb(#, #, #)`或`rgba(#, #, #, #)`格式。
- CSS规则的简写形式无效。比如，想读取`margin`属性的值，不能直接读，只能读`marginLeft`、`marginTop`等属性；再比如，`font`属性也是不能直接读的，只能读`font-size`等单个属性。
- 如果读取CSS原始的属性名，要用方括号运算符，比如`styleObj['z-index']`；如果读取骆驼拼写法的CSS属性名，可以直接读取`styleObj.zIndex`。
- 该方法返回的`CSSStyleDeclaration`实例的`cssText`属性无效，返回`undefined`。

## ## CSS 伪元素

CSS伪元素是通过CSS向DOM添加的元素，主要是通过`:before`和`:after`选择器生成，然后用`content`属性指定伪元素的内容。

下面是一段HTML代码。

```
```html
<div id="test">Test content</div>
```
```

CSS添加伪元素`:before`的写法如下。

```
```css
#test:before {
  content: 'Before ';
  color: #FF0;
}
```
```



节点元素的`style`对象无法读写伪元素的样式，这时就要用到`window.getComputedStyle()`。JavaScript 获取伪元素，可以使用下面的方法。

```
```javascript
var test = document.querySelector('#test');

var result = window.getComputedStyle(test, ':before').content;
var color = window.getComputedStyle(test, ':before').color;
```
```

此外，也可以使用 `CSSStyleDeclaration` 实例的`getPropertyValue`方法，获取伪元素的属性。

```
```javascript
var result = window.getComputedStyle(test, ':before')
    .getPropertyValue('content');
var color = window.getComputedStyle(test, ':before')
    .getPropertyValue('color');
```
```

## ## StyleSheet 接口

### ### 概述

`StyleSheet`接口代表网页的一张样式表，包括`<link>`元素加载的样式表和`<style>`元素内嵌的样式表。

`document`对象的`styleSheets`属性，可以返回当前页面的所有`StyleSheet`实例（即所有样式表）。它是一个类似数组的对象。

```
```javascript
var sheets = document.styleSheets;
var sheet = document.styleSheets[0];
sheet instanceof StyleSheet // true
```
```

如果是`<style>`元素嵌入的样式表，还有另一种获取`StyleSheet`实例的方法，就是这个节点元素的`sheet`属性。

```
```javascript
// HTML 代码为 <style id="myStyle"></style>
var myStyleSheet = document.getElementById('myStyle').sheet;
myStyleSheet instanceof StyleSheet // true
```
```

严格地说，`StyleSheet`接口不仅包括网页样式表，还包括 XML 文档的样式表。所以，它有一个子类`CSSStyleSheet`表示网页的 CSS 样式表。我们在网页里面拿到的样式表实例，实际上是

`CSSStyleSheet`的实例。这个子接口继承了`StyleSheet`的所有属性和方法，并且定义了几个自己的属性，下面把这两个接口放在一起介绍。

### ### 实例属性

`StyleSheet`实例有以下属性。

#### \*\* (1) StyleSheet.disabled\*\*

`StyleSheet.disabled`返回一个布尔值，表示该样式表是否处于禁用状态。手动设置`disabled`属性为`true`，等同于在`<link>`元素里面，将这张样式表设为`alternate stylesheet`，即该样式表将不会生效。

注意，`disabled`属性只能在 JavaScript 脚本中设置，不能在 HTML 语句中设置。

#### \*\* (2) Stylesheet.href\*\*

`Stylesheet.href`返回样式表的网址。对于内嵌样式表，该属性返回`null`。该属性只读。

```
```javascript
document.styleSheets[0].href
```
```

#### \*\* (3) StyleSheet.media\*\*

`StyleSheet.media`属性返回一个类似数组的对象（`MediaList`实例），成员是表示适用媒介的字符串。表示当前样式表是用于屏幕（screen），还是用于打印（print）或手持设备（handheld），或各种媒介都适用（all）。该属性只读，默认值是`screen`。

```
```javascript
document.styleSheets[0].media.mediaText
// "all"
```
```

`MediaList`实例的`appendMedium`方法，用于增加媒介；`deleteMedium`方法用于删除媒介。

```
```javascript
document.styleSheets[0].media.appendMedium('handheld');
document.styleSheets[0].media.deleteMedium('print');
```
```

#### \*\* (4) StyleSheet.title\*\*

`StyleSheet.title`属性返回样式表的`title`属性。

#### \*\* (5) StyleSheet.type\*\*

`StyleSheet.type` 属性返回样式表的 `type` 属性，通常是 `text/css`。

```
```javascript
document.styleSheets[0].type // "text/css"
```
```

#### \*\* (6) `StyleSheet.parentStyleSheet` \*\*

CSS 的 `@import` 命令允许在样式表中加载其他样式表。`StyleSheet.parentStyleSheet` 属性返回包含了当前样式表的那张样式表。如果当前样式表是顶层样式表，则该属性返回 `null`。

```
```javascript
if (stylesheet.parentStyleSheet) {
  sheet = stylesheet.parentStyleSheet;
} else {
  sheet = stylesheet;
}
```
```

#### \*\* (7) `StyleSheet.ownerNode` \*\*

`StyleSheet.ownerNode` 属性返回 `StyleSheet` 对象所在的 DOM 节点，通常是 `<link>` 或 `<style>`。对于那些由其他样式表引用的样式表，该属性为 `null`。

```
```javascript
// HTML代码为
// <link rel="StyleSheet" href="example.css" type="text/css" />
document.styleSheets[0].ownerNode // [object HTMLLinkElement]
```
```

#### \*\* (8) `CSSStyleSheet.cssRules` \*\*

`CSSStyleSheet.cssRules` 属性指向一个类似数组的对象（`CSSRuleList` 实例），里面每一个成员就是当前样式表的一条 CSS 规则。使用该规则的 `cssText` 属性，可以得到 CSS 规则对应的字符串。

```
```javascript
var sheet = document.querySelector('#styleElement').sheet;

sheet.cssRules[0].cssText
// "body { background-color: red; margin: 20px; }"

sheet.cssRules[1].cssText
// "p { line-height: 1.4em; color: blue; }"
```
```

每条 CSS 规则还有一个 `style` 属性，指向一个对象，用来读写具体的 CSS 命令。

```
```javascript
```

```
cssStyleSheet.cssRules[0].style.color = 'red';
cssStyleSheet.cssRules[1].style.color = 'purple';
```
```

## **\*\* (9) CSSStyleSheet.ownerRule\*\***

有些样式表是通过`@import`规则输入的，它的`ownerRule`属性会返回一个`CSSRule`实例，代表那行`@import`规则。如果当前样式表不是通过`@import`引入的，`ownerRule`属性返回`null`。

### **### 实例方法**

## **\*\* (1) CSSStyleSheet.insertRule()\*\***

`CSSStyleSheet.insertRule`方法用于在当前样式表的插入一个新的 CSS 规则。

```
```javascript
var sheet = document.querySelector('#styleElement').sheet;
sheet.insertRule('#block { color: white }', 0);
sheet.insertRule('p { color: red }', 1);
```
```

该方法可以接受两个参数，第一个参数是表示 CSS 规则的字符串，这里只能有一条规则，否则会报错。第二个参数是该规则在样式表的插入位置（从0开始），该参数可选，默认为0（即默认插在样式表的头部）。注意，如果插入位置大于现有规则的数目，会报错。

该方法的返回值是新插入规则的位置序号。

注意，浏览器对脚本在样式表里面插入规则有很多[限制](https://drafts.csswg.org/cssom/#insert-a-css-rule)。所以，这个方法最好放在`try...catch`里使用。

## **\*\* (2) CSSStyleSheet.deleteRule()\*\***

`CSSStyleSheet.deleteRule`方法用来在样式表里面移除一条规则，它的参数是该条规则在`cssRules`对象中的位置。该方法没有返回值。

```
```javascript
document.styleSheets[0].deleteRule(1);
```
```

### **## 实例：添加样式表**

网页添加样式表有两种方式。一种是添加一张内置样式表，即在文档中添加一个`<style>`节点。

```
```javascript
// 写法一
var style = document.createElement('style');
style.setAttribute('media', 'screen');
```

```
style.innerHTML = 'body{color:red}';
document.head.appendChild(style);
```

// 写法二

```
var style = (function () {
  var style = document.createElement('style');
  document.head.appendChild(style);
  return style;
})();
style.sheet.insertRule('.foo{color:red;}', 0);
````
```

另一种是添加外部样式表，即在文档中添加一个`<link>`节点，然后将`href`属性指向外部样式表的 URL。

```
````javascript
var linkElm = document.createElement('link');
linkElm.setAttribute('rel', 'stylesheet');
linkElm.setAttribute('type', 'text/css');
linkElm.setAttribute('href', 'reset-min.css');

document.head.appendChild(linkElm);
````
```

## ## CSSRuleList 接口

CSSRuleList 接口是一个类似数组的对象，表示一组 CSS 规则，成员都是 CSSRule 实例。

获取 CSSRuleList 实例，一般是通过`StyleSheet.cssRules`属性。

```
````javascript
// HTML 代码如下
// <style id="myStyle">
//   h1 { color: red; }
//   p { color: blue; }
// </style>
var myStyleSheet = document.getElementById('myStyle').sheet;
var crl = myStyleSheet.cssRules;
crl instanceof CSSRuleList // true
````
```

CSSRuleList 实例里面，每一条规则（CSSRule 实例）可以通过`rules.item(index)`或者`rules[index]`拿到。CSS 规则的条数通过`rules.length`拿到。还是用上面的例子。

```
````javascript
crl[0] instanceof CSSRule // true
crl.length // 2
````
```

注意，添加规则和删除规则不能在 `CSSRuleList` 实例操作，而要在它的父元素 `StyleSheet` 实例上，通过 `StyleSheet.insertRule()` 和 `StyleSheet.deleteRule()` 操作。

## ## CSSRule 接口

### ### 概述

一条 CSS 规则包括两个部分：CSS 选择器和样式声明。下面就是一条典型的 CSS 规则。

```
```css
.myClass {
  color: red;
  background-color: yellow;
}
```

JavaScript 通过 `CSSRule` 接口操作 CSS 规则。一般通过 `CSSRuleList` 接口（`StyleSheet.cssRules`）获取 `CSSRule` 实例。

```
```javascript
// HTML 代码如下
// <style id="myStyle">
// .myClass {
// color: red;
// background-color: yellow;
// }
// </style>
var myStyleSheet = document.getElementById('myStyle').sheet;
var ruleList = myStyleSheet.cssRules;
var rule = ruleList[0];
rule instanceof CSSRule // true
```
```

CSSRule 实例的属性

** (1) CSSRule.cssText **

`CSSRule.cssText` 属性返回当前规则的文本，还是使用上面的例子。

```
```javascript
rule.cssText
// ".myClass { color: red; background-color: yellow; }"
```
```

如果规则是加载（`@import`）其他样式表，`cssText` 属性返回 `@import 'url'`。

** (2) CSSRule.parentStyleSheet **

`CSSRule.parentStyleSheet`属性返回当前规则所在的样式表对象（`StyleSheet` 实例），还是使用上面的例子。

```
``javascript
rule.parentStyleSheet === myStyleSheet // true
``
```

** (3) CSSRule.parentRule**

`CSSRule.parentRule`属性返回包含当前规则的父规则，如果不存在父规则（即当前规则是顶层规则），则返回`null`。

父规则最常见的情况是，当前规则包含在`@media`规则代码块之中。

```
``javascript
// HTML 代码如下
// <style id="myStyle">
//   @supports (display: flex) {
//     @media screen and (min-width: 900px) {
//       article {
//         display: flex;
//       }
//     }
//   }
// }
// </style>
var myStyleSheet = document.getElementById('myStyle').sheet;
var ruleList = myStyleSheet.cssRules;

var rule0 = ruleList[0];
rule0.cssText
// "@supports (display: flex) {
//   @media screen and (min-width: 900px) {
//     article { display: flex; }
//   }
// }"

// 由于这条规则内嵌其他规则，
// 所以它有 cssRules 属性，且该属性是 CSSRuleList 实例
rule0.cssRules instanceof CSSRuleList // true

var rule1 = rule0.cssRules[0];
rule1.cssText
// "@media screen and (min-width: 900px) {
//   article { display: flex; }
// }"

var rule2 = rule1.cssRules[0];
rule2.cssText
// "article { display: flex; }"
```

```
rule1.parentRule === rule0 // true
rule2.parentRule === rule1 // true
```
```

#### **\*\* (4) CSSRule.type\*\***

`CSSRule.type`属性返回一个整数值，表示当前规则的类型。

最常见的类型有以下几种。

- 1: 普通样式规则 (CSSStyleRule 实例)
- 3: `@import`规则
- 4: `@media`规则 (CSSMediaRule 实例)
- 5: `@font-face`规则

#### **### CSSStyleRule 接口**

如果一条 CSS 规则是普通的样式规则 (不含特殊的 CSS 命令)，那么除了 CSSRule 接口，它还部署了 CSSStyleRule 接口。

CSSStyleRule 接口有以下两个属性。

#### **\*\* (1) CSSStyleRule.selectorText\*\***

`CSSStyleRule.selectorText`属性返回当前规则的选择器。

```
```javascript
var stylesheet = document.styleSheets[0];
stylesheet.cssRules[0].selectorText // ".myClass"
```
```

注意，这个属性是可写的。

#### **\*\* (2) CSSStyleRule.style\*\***

`CSSStyleRule.style`属性返回一个对象 (CSSStyleDeclaration 实例)，代表当前规则的样式声明，也就是选择器后面的大括号里面的部分。

```
```javascript
// HTML 代码为
// <style id="myStyle">
//   p { color: red; }
// </style>
var styleSheet = document.getElementById('myStyle').sheet;
styleSheet.cssRules[0].style instanceof CSSStyleDeclaration
// true
```
```



CSSStyleDeclaration 实例的`cssText`属性，可以返回所有样式声明，格式为字符串。

```
```javascript
styleSheet.cssRules[0].style.cssText
// "color: red;"
styleSheet.cssRules[0].selectorText
// "p"
```
```

### ### CSSMediaRule 接口

如果一条 CSS 规则是`@media`代码块，那么它除了 CSSRule 接口，还部署了 CSSMediaRule 接口。

该接口主要提供`media`属性和`conditionText`属性。前者返回代表`@media`规则的一个对象（MediaList 实例），后者返回`@media`规则的生效条件。

```
```javascript
// HTML 代码如下
// <style id="myStyle">
//   @media screen and (min-width: 900px) {
//     article { display: flex; }
//   }
// </style>
var styleSheet = document.getElementById('myStyle').sheet;
styleSheet.cssRules[0] instanceof CSSMediaRule
// true

styleSheet.cssRules[0].media
// {
//   0: "screen and (min-width: 900px)",
//   appendMedium: function,
//   deleteMedium: function,
//   item: function,
//   length: 1,
//   mediaText: "screen and (min-width: 900px)"
// }

styleSheet.cssRules[0].conditionText
// "screen and (min-width: 900px)"
```
```

### ## window.matchMedia()

#### ### 基本用法

`window.matchMedia`方法用来将 CSS 的[MediaQuery]([https://developer.mozilla.org/en-US/docs/DOM/Using\\_media\\_queries\\_from\\_code](https://developer.mozilla.org/en-US/docs/DOM/Using_media_queries_from_code))条件语句，转换成一个 MediaQueryList 实例。

```
```javascript
var mdl = window.matchMedia('(min-width: 400px)');
mdl instanceof MediaQueryList // true
```
```

上面代码中，变量`mdl`就是`mediaQueryList`的实例。

注意，如果参数不是有效的`MediaQuery`条件语句，`window.matchMedia`不会报错，依然返回一个`MediaQueryList`实例。

```
```javascript
window.matchMedia('bad string') instanceof MediaQueryList // true
```
```

### ### MediaQueryList 接口的实例属性

MediaQueryList 实例有三个属性。

#### \*\* (1) MediaQueryList.media\*\*

`MediaQueryList.media`属性返回一个字符串，表示对应的`MediaQuery`条件语句。

```
```javascript
var mql = window.matchMedia('(min-width: 400px)');
mql.media // "(min-width: 400px)"
```
```

#### \*\* (2) MediaQueryList.matches\*\*

`MediaQueryList.matches`属性返回一个布尔值，表示当前页面是否符合指定的`MediaQuery`条件语句。

```
```javascript
if (window.matchMedia('(min-width: 400px)').matches) {
  /* 当前视口不小于 400 像素 */
} else {
  /* 当前视口小于 400 像素 */
}
```
```

下面的例子根据`mediaQuery`是否匹配当前环境，加载相应的 CSS 样式表。

```
```javascript
var result = window.matchMedia("(max-width: 700px)");

if (result.matches){
  var linkElm = document.createElement('link');
  linkElm.setAttribute('rel', 'stylesheet');
  linkElm.setAttribute('type', 'text/css');
}
```

```

linkElm.setAttribute('href', 'small.css');

document.head.appendChild(linkElm);
}
...

```

** (3) MediaQueryList.onChange **

如果 MediaQuery 条件语句的适配环境发生变化，会触发`change`事件。

`MediaQueryList.onChange`属性用来指定`change`事件的监听函数。该函数的参数是`change`事件对象（MediaQueryListEvent 实例），该对象与 MediaQueryList 实例类似，也有`media`和`matches`属性。

```

```javascript
var mql = window.matchMedia('(max-width: 600px)');

mql.onChange = function(e) {
 if (e.matches) {
 /* 视口不超过 600 像素 */
 } else {
 /* 视口超过 600 像素 */
 }
}
...

```

上面代码中，`change`事件发生后，存在两种可能。一种是显示宽度从700像素以上变为以下，另一种是从700像素以下变为以上，所以在监听函数内部要判断一下当前是哪一种情况。

### ### MediaQueryList 接口的实例方法

MediaQueryList 实例有两个方法`MediaQueryList.addListener()`和`MediaQueryList.removeListener()`，用来为`change`事件添加或撤销监听函数。

```

```javascript
var mql = window.matchMedia('(max-width: 600px)');

```

```

// 指定监听函数
mql.addListener(mqCallback);

```

```

// 撤销监听函数
mql.removeListener(mqCallback);

```

```

function mqCallback(e) {
  if (e.matches) {
    /* 视口不超过 600 像素 */
  } else {
    /* 视口超过 600 像素 */
  }
}

```

```
}
```

注意，`MediaQueryList.removeListener()`方法不能撤销`MediaQueryList.onchange`属性指定的监听函数。