CSS 操作

目录
HTML 元素的 style 属性
CSSStyleDeclaration 接口
简介
CSSStyleDeclaration 实例属性
CSSStyleDeclaration 实例方法
CSS 模块的侦测
CSS 对象
CSS.escape()
CSS.supports()
window.getComputedStyle()
CSS 伪元素
StyleSheet 接口
概述
实例属性
实例方法
实例：添加样式表
CSSRuleList 接口
CSSRule 接口
概述
CSSRule 实例的属性
CSSStyleRule 接口
CSSMediaRule 接口
window.matchMedia()
基本用法
MediaQueryList 接口的实例属性
MediaQueryList 接口的实例方法

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。

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

上面的代码相当于下面的 HTML 代码。

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

`style` 不仅可以使用字符串读写，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。

```
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。

```
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。

```
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。

```
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。

```
// 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。

```
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。

```
// 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。

```
// 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。

```
// 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。

```
// 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

CSS

- important CSS !important

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

myDiv

CSS

CSS CSS
“CSS ”

style

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

CSS
undefined

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

max-width maximum-width

CSS style

```
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。

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

这种侦测方法可以写成一个函数。

```
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。

```
<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。

```
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。

```
// 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。

```
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。

```
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 :before :after :first-line :first-letter

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

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

height styleObj CSSStyleDeclaration CSSStyleDeclaration

- CSSStyleDeclaration CSS px rgb(#, #, #) rgba(#, #, #, #)
- CSS margin marginLeft marginTop font font-size
- CSS styleObj['z-index'] CSS styleObj.zIndex
- CSSStyleDeclaration cssText

CSS

CSS CSS DOM :before :after content

HTML

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

CSS :before

```
#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。

```
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。

```
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。

```
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。

```
// 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。

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。

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。

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。

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

6、StyleSheet.parentStyleSheet

CSS 的 `@import` 规则可以引入其他样式表。`StyleSheet.parentStyleSheet` 属性返回包含当前样式表的那张样式表。如果当前样式表是顶层样式表，则返回 `null`。

```
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，也就是 `<link>` 或 `<style>` 所在的节点。如果不存在这样的节点，返回 `null`。

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

8、CSSStyleSheet.cssRules

`CSSStyleSheet.cssRules` 属性返回一个类似数组的对象，`CSSRuleList` 实例。由于可以通过索引来读取每条 CSS 规则，并且该对象的 `cssText` 属性可以返回整个样式表的内容，因此可以通过这个属性来遍历一个样式表的所有规则。

```
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; }"
```

上面是 CSS 样式表的写法，`style` 属性也可以用来设置 CSS 规则。

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

9、CSSStyleSheet.ownerRule

有些样式表是通过 `@import` 规则插入的，`ownerRule` 属性返回一个 `CSSRule` 实例，代表 `@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。

```
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。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。

```
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。

```
// 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);
```

□□□□□□□□□□□□□□□□□□□□ `<link>`□□□□□□ `href`□□□□□□□□□□ URL□

```
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 □□

CSSRuleList □□□□□□□□□□□□□□□□□ CSS □□□□□□□ CSSRule □□□

□□ CSSRuleList □□□□□□□□ `StyleSheet.cssRules`□□□

```
// HTML □□□□
// <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 □□□□□□□□□□□ CSSRule □□□□□□□ `rules.item(index)`□□ `rules[index]`□□□
CSS □□□□□□□ `rules.length`□□□□□□□□□□□□

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

□□□□□□□□□□□□□□□ CSSRuleList □□□□□□□□□□□□□ StyleSheet □□□□□□
`StyleSheet.insertRule()`□ `StyleSheet.deleteRule()`□□□

CSSRule □□

□□

□□ CSS □□□□□□□□ CSS □□□□□□□□□□□□□□□□ CSS □□□

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

JavaScript □□ CSSRule □□□□ CSS □□□□□□□ CSSRuleList □□□ `StyleSheet.cssRules` □□□ CSSRule □□□

```
// HTML □□□□
// <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 □□□□

□ 1□ CSSRule.cssText

`CSSRule.cssText` □□□□□□□□□□□□□□□□□□□□□□□□

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

□□□□□□□□ `@import` □□□□□□□ `cssText` □□□□ `@import 'url'` □

□ 2□ CSSRule.parentStyleSheet

`CSSRule.parentStyleSheet` □□□□□□□□□□□□□□□□□□ StyleSheet □□□□□□□□□□□□□□□

```
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。

// 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。

```
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。

```
// 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 □□□ `cssText`□□□□□□□□□□□□□□□□□□□□

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

CSSMediaRule □□

□□□□ CSS □□□ `@media`□□□□□□□□□ CSSRule □□□□□□□ CSSMediaRule □□□

□□□□□□□ `media`□□□ `conditionText`□□□□□□□□□ `@media`□□□□□□□□ MediaList □□□□□□□□ `@media`□□□□□□□□

```
// HTML □□□□
// <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 `Media Query`条件语句，转换成一个 MediaQueryList 实例。

```
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。

```
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。

```
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。

```
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。

```
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。

```
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 600像素以上变为以下，另一种是从 600 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。

```
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。

版本号 #2
由 壳壳虫 创建于 2018-04-15 10:48:00 UTC
由 壳壳虫 更新于 2024-04-25 15:02:47 UTC