6th November 2019

Styles and classes

Before we get into JavaScript's ways of dealing with styles and classes – here's an important rule. Hopefully it's obvious enough, but we still have to mention it.

There are generally two ways to style an element:

1. Create a class in CSS and add it: `<div class="...">`
2. Write properties directly into `style`: `<div style="...">`.

JavaScript can modify both classes and `style` properties.

We should always prefer CSS classes to `style`. The latter should only be used if classes "can't handle it".

For example, `style` is acceptable if we calculate coordinates of an element dynamically and want to set them from JavaScript, like this:

```
let top = /* complex calculations */;
let left = /* complex calculations */;

elem.style.left = left; // e.g '123px', calculated at run-time
elem.style.top = top; // e.g '456px'
```

For other cases, like making the text red, adding a background icon – describe that in CSS and then add the class (JavaScript can do that). That's more flexible and easier to support.

className and classList

Changing a class is one of the most often used actions in scripts.

In the ancient time, there was a limitation in JavaScript: a reserved word like `"class"` could not be an object property. That limitation does not exist now, but at that time it was impossible to have a `"class"` property, like `elem.class`.

So for classes the similar-looking property `"className"` was introduced: the `elem.className` corresponds to the `"class"` attribute.

For instance:

```
<body class="main page">
  <script>
    alert(document.body.className); // main page
  </script>
</body>
```

If we assign something to `elem.className` , it replaces the whole string of classes. Sometimes that's what we need, but often we want to add/remove a single class.

There's another property for that: `elem.classList` .

The `elem.classList` is a special object with methods to `add/remove/toggle` a single class.

For instance:

```
<body class="main page">
  <script>
    // add a class
    document.body.classList.add('article');

    alert(document.body.className); // main page article
  </script>
</body>
```

So we can operate both on the full class string using `className` or on individual classes using `classList` . What we choose depends on our needs.

Methods of `classList` :

- `elem.classList.add/remove("class")` – adds/removes the class.
- `elem.classList.toggle("class")` – adds the class if it doesn't exist, otherwise removes it.
- `elem.classList.contains("class")` – checks for the given class, returns `true/false` .

Besides, `classList` is iterable, so we can list all classes with `for..of` , like this:

```
<body class="main page">
  <script>
    for (let name of document.body.classList) {
      alert(name); // main, and then page
    }
  </script>
</body>
```

Element style

The property `elem.style` is an object that corresponds to what's written in the `"style"` attribute. Setting `elem.style.width="100px"` works the same as if we had in the attribute `style` a string `width:100px` .

For multi-word property the camelCase is used:

```
background-color  => elem.style.backgroundColor
z-index           => elem.style.zIndex
border-left-width => elem.style.borderLeftWidth
```

For instance:

```
document.body.style.backgroundColor = prompt('background color?', 'green');
```

Prefixed properties

Browser-prefixed properties like `-moz-border-radius`, `-webkit-border-radius` also follow the same rule: a dash means upper case.

For instance:

```
button.style.MozBorderRadius = '5px';
button.style.WebkitBorderRadius = '5px';
```

Resetting the style property

Sometimes we want to assign a style property, and later remove it.

For instance, to hide an element, we can set `elem.style.display = "none"`.

Then later we may want to remove the `style.display` as if it were not set. Instead of `delete elem.style.display` we should assign an empty string to it: `elem.style.display = ""`.

```
// if we run this code, the <body> will blink
document.body.style.display = "none"; // hide

setTimeout(() => document.body.style.display = "", 1000); // back to normal
```

If we set `style.display` to an empty string, then the browser applies CSS classes and its built-in styles normally, as if there were no such `style.display` property at all.

Full rewrite with `style.cssText`

Normally, we use `style.*` to assign individual style properties. We can't set the full style like `div.style="color: red; width: 100px"`, because `div.style` is an object, and it's read-only.

To set the full style as a string, there's a special property `style.cssText`:

```
<div id="div">Button</div>

<script>
  // we can set special style flags like "important" here
  div.style.cssText=`color: red !important;
    background-color: yellow;
    width: 100px;
    text-align: center;
  `;

  alert(div.style.cssText);
</script>
```

This property is rarely used, because such assignment removes all existing styles: it does not add, but replaces them. May occasionally delete something needed. But we can safely use it for new elements, when we know we won't delete an existing style.

The same can be accomplished by setting an attribute: `div.setAttribute('style', 'color: red...')`.

Mind the units

Don't forget to add CSS units to values.

For instance, we should not set `elem.style.top` to `10`, but rather to `10px`. Otherwise it wouldn't work:

```
<body>
  <script>
    // doesn't work!
    document.body.style.margin = 20;
    alert(document.body.style.margin); // '' (empty string, the assignment is

    // now add the CSS unit (px) - and it works
    document.body.style.margin = '20px';
    alert(document.body.style.margin); // 20px

    alert(document.body.style.marginTop); // 20px
    alert(document.body.style.marginLeft); // 20px
  </script>
</body>
```

Please note: the browser "unpacks" the property `style.margin` in the last lines and infers `style.marginLeft` and `style.marginTop` from it.

Computed styles: getComputedStyle

So, modifying a style is easy. But how to *read* it?

For instance, we want to know the size, margins, the color of an element. How to do it?

The `style` property operates only on the value of the `"style"` attribute, without any CSS cascade.

So we can't read anything that comes from CSS classes using `elem.style`.

For instance, here `style` doesn't see the margin:

```
<head>
  <style> body { color: red; margin: 5px } </style>
</head>
<body>

  The red text
  <script>
    alert(document.body.style.color); // empty
    alert(document.body.style.marginTop); // empty
  </script>
</body>
```

…But what if we need, say, to increase the margin by `20px`? We would want the current value of it.

There's another method for that: `getComputedStyle`.

The syntax is:

```
getComputedStyle(element, [pseudo])
```

element

Element to read the value for.

pseudo

A pseudo-element if required, for instance `::before`. An empty string or no argument means the element itself.

The result is an object with styles, like `elem.style`, but now with respect to all CSS classes.

For instance:

```
<head>
  <style> body { color: red; margin: 5px } </style>
</head>
<body>

  <script>
    let computedStyle = getComputedStyle(document.body);

    // now we can read the margin and the color from it

    alert( computedStyle.marginTop ); // 5px
    alert( computedStyle.color ); // rgb(255, 0, 0)
  </script>

</body>
```

Computed and resolved values

There are two concepts in CSS:

1. A *computed* style value is the value after all CSS rules and CSS inheritance is applied, as the result of the CSS cascade. It can look like `height:1em` or `font-size:125%`.
2. A *resolved* style value is the one finally applied to the element. Values like `1em` or `125%` are relative. The browser takes the computed value and makes all units fixed and absolute, for instance: `height:20px` or `font-size:16px`. For geometry properties resolved values may have a floating point, like `width:50.5px`.

A long time ago `getComputedStyle` was created to get computed values, but it turned out that resolved values are much more convenient, and the standard changed.

So nowadays `getComputedStyle` actually returns the resolved value of the property, usually in `px` for geometry.

`getComputedStyle` requires the full property name

We should always ask for the exact property that we want, like `paddingLeft` or `marginTop` or `borderTopWidth`. Otherwise the correct result is not guaranteed.

For instance, if there are properties `paddingLeft/paddingTop`, then what should we get for `getComputedStyle(elem).padding`? Nothing, or maybe a “generated” value from known paddings? There’s no standard rule here.

There are other inconsistencies. As an example, some browsers (Chrome) show `10px` in the document below, and some of them (Firefox) – do not:

```
<style>
  body {
    margin: 10px;
  }
</style>
<script>
  let style = getComputedStyle(document.body);
  alert(style.margin); // empty string in Firefox
</script>
```

Styles applied to `:visited` links are hidden!

Visited links may be colored using `:visited` CSS pseudoclass.

But `getComputedStyle` does not give access to that color, because otherwise an arbitrary page could find out whether the user visited a link by creating it on the page and checking the styles.

JavaScript may not see the styles applied by `:visited`. And also, there's a limitation in CSS that forbids applying geometry-changing styles in `:visited`. That's to guarantee that there's no side way for an evil page to test if a link was visited and hence to break the privacy.

Summary

To manage classes, there are two DOM properties:

- `className` – the string value, good to manage the whole set of classes.
- `classList` – the object with methods `add/remove/toggle/contains`, good for individual classes.

To change the styles:

- The `style` property is an object with camelCased styles. Reading and writing to it has the same meaning as modifying individual properties in the `"style"` attribute. To see how to apply `important` and other rare stuff – there's a list of methods at MDN.
- The `style.cssText` property corresponds to the whole `"style"` attribute, the full string of styles.

To read the resolved styles (with respect to all classes, after all CSS is applied and final values are calculated):

- The `getComputedStyle(elem, [pseudo])` returns the style-like object with them. Read-only.

Tasks

Create a notification

importance: 5

Write a function `showNotification(options)` that creates a notification: `<div class="notification">` with the given content. The notification should automatically disappear after 1.5 seconds.

The options are:

```
// shows an element with the text "Hello" near the right-top of the window
showNotification({
  top: 10, // 10px from the top of the window (by default 0px)
  right: 10, // 10px from the right edge of the window (by default 0px)
  html: "Hello!", // the HTML of notification
  className: "welcome" // an additional class for the div (optional)
});
```

Demo in new window

Use CSS positioning to show the element at given top/right coordinates. The source document has the necessary styles.

Open a sandbox for the task.

solution

Previous lesson Next lesson

Share Tutorial map

Comments

- If you have suggestions what to improve - please submit a GitHub issue or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>`, for more than 10 lines – use a sandbox (plnkr, JSBin, codepen…)