24 de octubre de 2022

# Estilos y clases

Antes de profundizar en cómo JavaScript maneja las clases y los estilos, hay una regla importante. Aunque es lo suficientemente obvio, aún tenemos que mencionarlo.

Por lo general, hay dos formas de dar estilo a un elemento:

1. Crear una clase `css` y agregarla: `<div class="...">`
2. Escribir las propiedades directamente en `style`: `<div style="...">`.

JavaScript puede modificar ambos, clases y las propiedades de `style`.

Nosotros deberíamos preferir las clases `css` en lugar de `style`. Este último solo debe usarse si las clases "no pueden manejarlo".

Por ejemplo, `style` es aceptable si nosotros calculamos las coordenadas de un elemento dinámicamente y queremos establecer estas desde JavaScript, así:

```
let top = /* cálculos complejos */;
let left = /* cálculos complejos */;

elem.style.left = left; // ej. '123px', calculado en tiempo de ejecución
elem.style.top = top; // ej. '456px'
```

Para otros casos como convertir un texto en rojo, agregar un icono de fondo. Escribir eso en CSS y luego agregar la clase (JavaScript puede hacer eso), es más flexible y más fácil de mantener.

## className y classList

Cambiar una clase es una de las acciones más utilizadas.

En la antigüedad, había una limitación en JavaScript: una palabra reservada como `"class"` no podía ser una propiedad de un objeto. Esa limitación no existe ahora, pero en ese momento era imposible tener una propiedad `"class"`, como `elem.class`.

Entonces para clases de similares propiedades, `"className"` fue introducido: el `elem.className` corresponde al atributo `"class"`.

Por ejemplo:

```
<body class="main page">
  <script>
```

```
    alert(document.body.className); // página principal
  </script>
</body>
```

Si asignamos algo a `elem.className`, reemplaza toda la cadena de clases. A veces es lo que necesitamos, pero a menudo queremos agregar o eliminar una sola clase.

Hay otra propiedad para eso: `elem.classList`.

El `elem.classList` es un objeto especial con métodos para agregar, eliminar y alternar (`add/remove/toggle`) una sola clase.

Por ejemplo:

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

    alert(document.body.className); // clase "article" de la página principal
  </script>
</body>
```

Entonces podemos trabajar con ambos: todas las clases como una cadena usando `className` o con clases individuales usando `classList`. Lo que elijamos depende de nuestras necesidades.

Métodos de `classList`:

- `elem.classList.add/remove("class")` – agrega o remueve la clase.
- `elem.classList.toggle("class")` – agrega la clase si no existe, si no la remueve.
- `elem.classList.contains("class")` – verifica si tiene la clase dada, devuelve `true/false`.

Además, `classList` es iterable, entonces podemos listar todas las clases con `for..of`, así:

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

## style de un elemento

La propiedad `elem.style` es un objeto que corresponde a lo escrito en el atributo `"style"`. Establecer `elem.style.width="100px"` funciona igual que sí tuviéramos en el atributo `style` una cadena con `width:100px`.

Para propiedades de varias palabras se usa `camelCase`:

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

Por ejemplo:

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

> **Propiedades prefijadas**
>
> Propiedades con prefijos del navegador como `-moz-border-radius`, `-webkit-border-radius` también siguen la misma regla: un guion significa mayúscula.
>
> Por ejemplo:
>
> ```
> button.style.MozBorderRadius = '5px';
> button.style.WebkitBorderRadius = '5px';
> ```

## Reseteando la propiedad style

A veces queremos asignar una propiedad de estilo y luego removerla.

Por ejemplo, para ocultar un elemento, podemos establecer `elem.style.display = "none"`.

Luego, más tarde, es posible que queramos remover `style.display` como si no estuviera establecido. En lugar de `delete elem.style.display` deberíamos asignarle una cadena vacía: `elem.style.display = ""`.

```
// si ejecutamos este código, el <body> parpadeará
document.body.style.display = "none"; // ocultar

setTimeout(() => document.body.style.display = "", 1000); // volverá a lo norma
```

Si establecemos `style.display` como una cadena vacía, entonces el navegador aplica clases y estilos CSS incorporados normalmente por el navegador, como si no existiera tal `style.display`.

También hay un método especial para eso, `elem.style.removeProperty('style property')`. Así, podemos quitar una propiedad:

```
document.body.style.background = 'red'; //establece background a rojo

setTimeout(() => document.body.style.removeProperty('background'), 1000); // qu
```

### Reescribir todo usando `style.cssText`

Normalmente, podemos usar `style.*` para asignar propiedades de estilo individuales. No podemos establecer todo el estilo como `div.style="color: red; width: 100px"`, porque `div.style` es un objeto y es solo de lectura.

Para establecer todo el estilo como una cadena, hay una propiedad especial: `style.cssText`:

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

<script>
  // podemos establecer estilos especiales con banderas como "important"
  div.style.cssText=`color: red !important;
    background-color: yellow;
    width: 100px;
    text-align: center;
  `;

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

Esta propiedad es rara vez usada, porque tal asignación remueve todo los estilos: no agrega estilos sino que los reemplaza en su totalidad. Ocasionalmente podría eliminar algo necesario. Pero podemos usarlo de manera segura para nuevos elementos, cuando sabemos que no vamos a eliminar un estilo existente.

Lo mismo se puede lograr estableciendo un atributo: `div.setAttribute('style', 'color: red...')`.

## Cuidado con las unidades CSS

No olvidar agregar las unidades CSS a los valores.

Por ejemplo, nosotros no debemos establecer `elem.style.top` a `10`, sino más bien a `10px`. De lo contrario no funcionaría:

```
<body>
  <script>
    // ¡no funciona!
    document.body.style.margin = 20;
    alert(document.body.style.margin); // '' (cadena vacía, la asignación es ig

    // ahora agregamos la unidad CSS (px) y esta sí funciona
    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>
```

Tenga en cuenta: el navegador "desempaqueta" la propiedad `style.margin` en las últimas lineas e infiere `style.marginLeft` y `style.marginTop` de eso.

## Estilos calculados: getComputedStyle

Entonces, modificar un estilo es fácil. ¿Pero cómo *leerlo*?

Por ejemplo, queremos saber el tamaño, los márgenes, el color de un elemento. ¿Cómo hacerlo?

**La propiedad `style` solo opera en el valor del atributo `"style"` , sin ninguna cascada de `css` .**

Entonces no podemos leer ninguna clase CSS usando `elem.style` .

Por ejemplo, aquí `style` no ve el margen:

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

  El texto en rojo
  <script>
    alert(document.body.style.color); // vacío
    alert(document.body.style.marginTop); // vacío
  </script>
</body>
```

Pero si necesitamos incrementar el margen a `20px` ? vamos el querer el valor de la misma.

Hay otro método para eso: `getComputedStyle` .

La sintaxis es:

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

**element**

Elemento del cual se va a leer el valor.

**pseudo**

Un pseudo-elemento es requerido, por ejemplo `::before` . Una cadena vacía o sin argumento significa el elemento mismo.

El resultado es un objeto con estilos, como `elem.style` , pero ahora con respecto a todas las clases CSS.

Por ejemplo:

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

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

    // ahora podemos leer los márgenes y el color de ahí

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

</body>
```

### Valores calculado y resueltos

Hay dos conceptos en CSS:

1. Un estilo *calculado* es el valor final de aplicar todas las reglas y herencias CSS, como resultado de la cascada CSS. Puede parecer `height:1em` o `font-size:125%`.
2. Un estilo *resuelto* es la que finalmente se aplica al elemento. Valores como `1em` o `125%` son relativos. El navegador toma el valor calculado y hace que todas las unidades sean fijas y absolutas, por ejemplo: `height:20px` o `font-size:16px`. Para las propiedades de geometría los valores resueltos pueden tener un punto flotante, como `width:50.5px`.

Hace mucho tiempo `getComputedStyle` fue creado para obtener los valores calculados, pero los valores resueltos son muchos más convenientes, y el estándar cambió.

Así que hoy en día `getComputedStyle` en realidad devuelve el valor resuelto de la propiedad, usualmente en `px` para geometría.

### El método `getComputedStyle` requiere el nombre completo de la propiedad

Siempre deberíamos preguntar por la propiedad exacta que queremos, como `paddingLeft` o `marginTop` o `borderTopWidth`. De lo contrario, no se garantiza el resultado correcto.

Por ejemplo, si hay propiedades `paddingLeft/paddingTop`, entonces ¿qué deberíamos obtener de `getComputedStyle(elem).padding`? ¿Nada, o tal vez un valor "generado" de los paddings? No hay una regla estándar aquí.

### ¡Los estilos aplicados a los enlaces `:visited` están ocultos!

Los enlaces visitados pueden ser coloreados usando la pseudo-clase `:visited` de CSS.

Pero `getComputedStyle` no da acceso a ese color, porque de lo contrario una página cualquiera podría averiguar si el usuario visitó un enlace creándolo en la página y verificar los estilos.

JavaScript no puede ver los estilos aplicados por `:visited`. También hay una limitación en CSS que prohíbe la aplicación de estilos de cambio de geometría en `:visited`. Eso es para garantizar que no haya forma para que una página maligna pruebe si un enlace fue visitado y vulnere la privacidad.

## Resumen

Para manejar clases, hay dos propiedades del DOM:

- `className` – el valor de la cadena, perfecto para manejar todo el conjunto de clases.
- `classList` – el objeto con los métodos: `add/remove/toggle/contains`, perfecto para clases individuales.

Para cambiar los estilos:

- La propiedad `style` es un objeto con los estilos en `camelcase`. Leer y escribir tiene el mismo significado que modificar propiedades individuales en el atributo `"style"`. Para ver cómo aplicar `important` y otras cosas raras, hay una lista de métodos en MDN.
- La propiedad `style.cssText` corresponde a todo el atributo `"style"`, la cadena completa de estilos.

Para leer los estilos resueltos (con respecto a todas las clases, después de que se aplica todo el `css` y se calculan los valores finales):

- El método `getComputedStyle(elem, [pseudo])` retorna el objeto de estilo con ellos (solo lectura).

## Tareas

### Crear una notificación

importancia: 5

Escribir una función `showNotification(options)` que cree una notificación: `<div class="notification">` con el contenido dado. La notificación debería desaparecer automáticamente despues de 1.5 segundos.

Las opciones son:

```
// muestra un elemento con el texto "Hello" cerca de la parte superior de la ve
showNotification({
  top: 10, // 10px desde la parte superior de la ventana (por defecto es 0px)
  right: 10, // 10px desde el borde derecho de la ventana (por defecto es 0px)
  html: "Hello!", // el HTML de la notificación

```

```
7    className: "welcome" // una clase adicional para el "div" (opcional)
   });
```

Demo en nueva ventana

Usar posicionamiento CSS para mostrar el elemento en las coordenadas (top/right) dadas. El documento tiene los estilos necesarios.

Abrir un entorno controlado para la tarea.

solución

Lección anterior

Próxima lección

Compartir

Mapa del Tutorial

## Comentarios

- Si tiene sugerencias sobre qué mejorar, por favor enviar una propuesta de GitHub o una solicitud de extracción en lugar de comentar.
- Si no puede entender algo en el artículo, por favor explique.
- Para insertar algunas palabras de código, use la etiqueta `<code>`, para varias líneas – envolverlas en la etiqueta `<pre>`, para más de 10 líneas – utilice una entorno controlado (sandbox) (plnkr, jsbin, codepen...)

acerca del proyecto contáctenos