1 de septiembre de 2022

Atributos y propiedades

Cuando el navegador carga la página, "lee" (o "parser"(analiza en inglés")) el HTML y genera objetos DOM a partir de él. Para los nodos de elementos, la mayoría de los atributos HTML estándar se convierten automáticamente en propiedades de los objetos DOM.

Por ejemplo, si la etiqueta es `<body id="page">` , entonces el objeto DOM tiene `body.id="page"` .

¡Pero el mapeo de propiedades y atributos no es uno a uno! En este capítulo, prestaremos atención para separar estas dos nociones, para ver cómo trabajar con ellos, cuándo son iguales y cuándo son diferentes.

Propiedades DOM

Ya hemos visto propiedades DOM integradas. Hay muchas. Pero técnicamente nadie nos limita, y si no hay suficientes, podemos agregar las nuestras.

Los nodos DOM son objetos JavaScript normales. Podemos alterarlos.

Por ejemplo, creemos una nueva propiedad en `document.body` :

```
1  document.body.myData = {
2    name: 'Cesar',
3    title: 'Emperador'
4  };
5
6  alert(document.body.myData.title); // Emperador
```

También podemos agregar un método:

```
1  document.body.sayTagName = function() {
2    alert(this.tagName);
3  };
4
5  document.body.sayTagName(); // BODY (el valor de 'this' en el método es documer
```

También podemos modificar prototipos incorporados como `Element.prototype` y agregar nuevos métodos a todos los elementos:

```
1  Element.prototype.sayHi = function() {
```

```
  alert(`Hola, yo soy ${this.tagName}`);
};

document.documentElement.sayHi(); // Hola, yo soy HTML
document.body.sayHi(); // Hola, yo soy BODY
```

Por lo tanto, las propiedades y métodos DOM se comportan igual que los objetos JavaScript normales:

- Pueden tener cualquier valor.
- Distingue entre mayúsculas y minúsculas (escribir `elem.nodeType`, no es lo mismo que `elem.NoDeTyPe`).

Atributos HTML

En HTML, las etiquetas pueden tener atributos. Cuando el navegador analiza el HTML para crear objetos DOM para etiquetas, reconoce los atributos *estándar* y crea propiedades DOM a partir de ellos.

Entonces, cuando un elemento tiene `id` u otro atributo *estándar*, se crea la propiedad correspondiente. Pero eso no sucede si el atributo no es estándar.

Por ejemplo:

```
<body id="test" something="non-standard">
  <script>
    alert(document.body.id); // prueba
    // el atributo no estándar no produce una propiedad
    alert(document.body.something); // undefined
  </script>
</body>
```

Tenga en cuenta que un atributo estándar para un elemento puede ser desconocido para otro. Por ejemplo, `"type"` es estándar para `<input>` (HTMLInputElement), pero no para `<body>` (HTMLBodyElement). Los atributos estándar se describen en la especificación para la clase del elemento correspondiente.

Aquí podemos ver esto:

```
<body id="body" type="...">
  <input id="input" type="text">
  <script>
    alert(input.type); // text
    alert(body.type); // undefined: Propiedad DOM no creada, porque no es está
  </script>
</body>
```

Entonces, si un atributo no es estándar, no habrá una propiedad DOM para él. ¿Hay alguna manera de acceder a tales atributos?

Claro. Todos los atributos son accesibles usando los siguientes métodos:

- `elem.hasAttribute(nombre)` – comprueba si existe.
- `elem.getAttribute(nombre)` – obtiene el valor.
- `elem.setAttribute(nombre, valor)` – establece el valor.
- `elem.removeAttribute(nombre)` – elimina el atributo.

Estos métodos funcionan exactamente con lo que está escrito en HTML.

También se pueden leer todos los atributos usando `elem.attributes` : una colección de objetos que pertenecen a una clase integrada Attr, con propiedades `nombre` y `valor` .

Aquí hay una demostración de la lectura de una propiedad no estándar:

```
<body something="non-standard">
  <script>
    alert(document.body.getAttribute('something')); // no estándar
  </script>
</body>
```

Los atributos HTML tienen las siguientes características:

- Su nombre no distingue entre mayúsculas y minúsculas (`id` es igual a `ID`).
- Sus valores son siempre strings.

Aquí hay una demostración extendida de cómo trabajar con atributos:

```
<body>
  <div id="elem" about="Elephant"></div>

  <script>
    alert( elem.getAttribute('About') ); // (1) 'Elephant', leyendo

    elem.setAttribute('Test', 123); // (2), escribiendo

    alert( elem.outerHTML ); // (3), ver si el atributo está en HTML (sí)

    for (let attr of elem.attributes) { // (4) listar todo
      alert( `${attr.name} = ${attr.value}` );
    }
  </script>
</body>
```

Tenga en cuenta:

1. `getAttribute ('About)` – la primera letra está en mayúscula aquí, y en HTML todo está en minúscula. Pero eso no importa: los nombres de los atributos no distinguen entre mayúsculas y minúsculas.
2. Podemos asignar cualquier cosa a un atributo, pero se convierte en un string. Así que aquí tenemos `"123"` como valor.
3. Todos los atributos, incluidos los que configuramos, son visibles en `outerHTML` .

4. La colección `attributes` es iterable y tiene todos los atributos del elemento (estándar y no estándar) como objetos con propiedades `name` y `value`.

Sincronización de propiedad y atributo

Cuando cambia un atributo estándar, la propiedad correspondiente se actualiza automáticamente, y (con algunas excepciones) viceversa.

En el ejemplo a continuación, `id` se modifica como un atributo, y podemos ver que la propiedad también es cambiada. Y luego lo mismo al revés:

```
<input>

<script>
  let input = document.querySelector('input');

  // atributo -> propiedad
  input.setAttribute('id', 'id');
  alert(input.id); // id (actualizado)

  // propiedad -> atributo
  input.id = 'newId';
  alert(input.getAttribute('id')); // newId (actualizado)
</script>
```

Pero hay exclusiones, por ejemplo, `input.value` se sincroniza solo del atributo a la propiedad (atributo → propiedad), pero no de regreso:

```
<input>

<script>
  let input = document.querySelector('input');

  // atributo -> propiedad
  input.setAttribute('value', 'text');
  alert(input.value); // text

  // NO propiedad -> atributo
  input.value = 'newValue';
  alert(input.getAttribute('value')); // text (¡no actualizado!)
</script>
```

En el ejemplo anterior:

- Cambiar el atributo `value` actualiza la propiedad.
- Pero el cambio de propiedad no afecta al atributo.

Esa “característica” en realidad puede ser útil, porque las acciones del usuario pueden conducir a cambios de `value`, y luego, si queremos recuperar el valor “original” de HTML, está en el atributo.

Las propiedades DOM tienen tipo

Las propiedades DOM no siempre son strings. Por ejemplo, la propiedad `input.checked` (para casillas de verificación) es un booleano:

```
<input id="input" type="checkbox" checked> checkbox

<script>
  alert(input.getAttribute('checked')); // el valor del atributo es: string va
  alert(input.checked); // el valor de la propiedad es: true
</script>
```

Hay otros ejemplos. El atributo `style` es un string, pero la propiedad `style` es un objeto:

```
<div id="div" style="color:red;font-size:120%">Hola</div>

<script>
  // string
  alert(div.getAttribute('style')); // color:red;font-size:120%

  // object
  alert(div.style); // [object CSSStyleDeclaration]
  alert(div.style.color); // red
</script>
```

La mayoría de las propiedades son strings.

Muy raramente, incluso si un tipo de propiedad DOM es un string, puede diferir del atributo. Por ejemplo, la propiedad DOM `href` siempre es una URL *completa*, incluso si el atributo contiene una URL relativa o solo un `#hash`.

Aquí hay un ejemplo:

```
<a id="a" href="#hola">link</a>
<script>
  // atributo
  alert(a.getAttribute('href')); // #hola

  // propiedad
  alert(a.href ); // URL completa de http://site.com/page#hola
</script>
```

Si necesitamos el valor de `href` o cualquier otro atributo exactamente como está escrito en el HTML, podemos usar `getAttribute`.

Atributos no estándar, dataset

Cuando escribimos HTML, usamos muchos atributos estándar. Pero, ¿qué pasa con los no personalizados y personalizados? Primero, veamos si son útiles o no. ¿Para qué?

A veces, los atributos no estándar se utilizan para pasar datos personalizados de HTML a JavaScript, o para "marcar" elementos HTML para JavaScript.

Como esto:

```
<!-- marque el div para mostrar "nombre" aquí -->
<div show-info="nombre"></div>
<!-- y "edad" aquí -->
<div show-info="edad"></div>

<script>
 // el código encuentra un elemento con la marca y muestra lo que se solicita
  let user = {
    nombre: "Pete",
    edad: 25
  };

  for(let div of document.querySelectorAll('[show-info]')) {
    // inserta la información correspondiente en el campo
    let field = div.getAttribute('show-info');
    div.innerHTML = user[field]; // primero Pete en "nombre", luego 25 en "edad"
  }
</script>
```

También se pueden usar para diseñar un elemento.

Por ejemplo, aquí para el estado del pedido se usa el atributo `order-state`:

```
<style>
  /* los estilos se basan en el atributo personalizado "order-state" */
  .order[order-state="nuevo"] {
    color: green;
  }

  .order[order-state="pendiente"] {
    color: blue;
  }

  .order[order-state="cancelado"] {
    color: red;
  }
```

```
</style>

<div class="order" order-state="nuevo">
  Un nuevo pedido.
</div>

<div class="order" order-state="pendiente">
  Un pedido pendiente.
</div>

<div class="order" order-state="cancelado">
  Un pedido cancelado
</div>
```

¿Por qué sería preferible usar un atributo a tener clases como `.order-state-new`, `.order-state-pending`, `.order-state-canceled`?

Porque un atributo es más conveniente de administrar. El estado se puede cambiar tan fácil como:

```
// un poco más simple que eliminar/agregar clases
div.setAttribute('order-state', 'canceled');
```

Pero puede haber un posible problema con los atributos personalizados. ¿Qué sucede si usamos un atributo no estándar para nuestros propósitos y luego el estándar lo introduce y hace que haga algo? El lenguaje HTML está vivo, crece y cada vez hay más atributos que aparecen para satisfacer las necesidades de los desarrolladores. Puede haber efectos inesperados en tal caso.

Para evitar conflictos, existen atributos data-*.

Todos los atributos que comienzan con "data-" están reservados para el uso de los programadores. Están disponibles en la propiedad `dataset`.

Por ejemplo, si un `elem` tiene un atributo llamado `"data-about"`, está disponible como `elem.dataset.about`.

Como esto:

```
<body data-about="Elefante">
<script>
  alert(document.body.dataset.about); // Elefante
</script>
```

Los atributos de varias palabras como `data-order-state` se convierten en camel-case: `dataset.orderState`

Aquí hay un ejemplo reescrito de "estado del pedido":

```
<style>
  .order[data-order-state="nuevo"] {
    color: green;
```

```
  }

  .order[data-order-state="pendiente"] {
    color: blue;
  }

  .order[data-order-state="cancelado"] {
    color: red;
  }
</style>

<div id="order" class="order" data-order-state="nuevo">
  Una nueva orden.
</div>

<script>
  // leer
  alert(order.dataset.orderState); // nuevo

  // modificar
  order.dataset.orderState = "pendiente"; // (*)
</script>
```

El uso de los atributos `data- *` es una forma válida y segura de pasar datos personalizados.

Tenga en cuenta que no solo podemos leer, sino también modificar los atributos de datos. Luego, CSS actualiza la vista en consecuencia: en el ejemplo anterior, la última línea `(*)` cambia el color a azul.

Resumen

- Atributos: es lo que está escrito en HTML.
- Propiedades: es lo que hay en los objetos DOM.

Una pequeña comparación:

	Propiedades	Atributos
Tipo	Cualquier valor, las propiedades estándar tienen tipos descritos en la especificación	Un string
Nombre	El nombre distingue entre mayúsculas y minúsculas	El nombre no distingue entre mayúsculas y minúsculas

Los métodos para trabajar con atributos son:

- `elem.hasAttribute(nombre)` – para comprobar si existe.
- `elem.getAttribute(nombre)` – para obtener el valor.
- `elem.setAttribute(nombre, valor)` – para dar un valor.
- `elem.removeAttribute(nombre)` – para eliminar el atributo.
- `elem.attributes` es una colección de todos los atributos.

Para la mayoría de las situaciones, es preferible usar las propiedades DOM. Deberíamos referirnos a los atributos solo cuando las propiedades DOM no nos convienen, cuando necesitamos exactamente atributos, por ejemplo:

- Necesitamos un atributo no estándar. Pero si comienza con `data-`, entonces deberíamos usar `dataset`.
- Queremos leer el valor "como está escrito" en HTML. El valor de la propiedad DOM puede ser diferente, por ejemplo, la propiedad `href` siempre es una URL completa, y es posible que queramos obtener el valor "original ".

Tareas

Obtén en atributo

importancia: 5

Escribe el código para obtener el atributo `data-widget-name` del documento y leer su valor.

```
<!DOCTYPE html>
<html>
<body>

  <div data-widget-name="menu">Elige el genero</div>

  <script>
    /* Tu código */
  </script>
</body>
</html>
```

solución

Haz los enlaces externos naranjas

importancia: 3

Haz todos los enlaces externos de color orange alterando su propiedad `style`.

Un link es externo si:

- Su `href` tiene `://`
- Pero no comienza con `http://internal.com`.

Ejemplo:

```
<a name="list">the list</a>
<ul>
  <li><a href="http://google.com">http://google.com</a></li>
  <li><a href="/tutorial">/tutorial.html</a></li>
  <li><a href="local/path">local/path</a></li>
```

```
  <li><a href="ftp://ftp.com/my.zip">ftp://ftp.com/my.zip</a></li>
  <li><a href="http://nodejs.org">http://nodejs.org</a></li>
  <li><a href="http://internal.com/test">http://internal.com/test</a></li>
</ul>

<script>
  // establecer un estilo para un enlace
  let link = document.querySelector('a');
  link.style.color = 'orange';
</script>
```

El resultado podría ser:

La lista:

- http://google.com
- /tutorial.html
- local/path
- ftp://ftp.com/my.zip
- http://nodejs.org
- http://internal.com/test

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