EBook Gratis

APRENDIZAJE DOM

Free unaffiliated eBook created from
Stack Overflow contributors.

#dom

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: dom

It is an unofficial and free DOM ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official DOM.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con DOM

Observaciones

El DOM, o Modelo de objetos de documento, es la API utilizada por los navegadores web y otras aplicaciones para acceder a los contenidos de un documento HTML.

El DOM representa la estructura como un árbol, los nodos pueden contener nodos secundarios, los nodos sin hijos son dichos nodos de hoja.

Con él, se puede manipular la estructura y las propiedades del documento y sus partes constituyentes.

Los temas principales incluyen la búsqueda de elementos, el acceso a la información de estilo y la animación.

La mayoría del trabajo con el DOM se realiza utilizando el lenguaje JavaScript , pero la API está abierta a cualquier idioma.

Versiones

W3C DOM

Versión	Fecha de lanzamiento
1	1998-10-01
2 (núcleo)	2000-11-13
3 (Core)	2004-04-07
4	2013-11-07

Selectores de nivel de API

Versión	Fecha de lanzamiento
1	2013-02-21

Examples

Recuperando elementos html existentes

Una de las tareas más comunes es recuperar un elemento existente del DOM para manipularlo. Más comúnmente, estos métodos se ejecutan en un `document` , porque es el nodo raíz, pero todos estos métodos funcionan en cualquier elemento HTML en el árbol. Solo devolverán hijos del nodo en el que se ejecuta.

Recuperar por ID

```
var element = document.getElementById("logo");
```

`element` contendrá el (único) elemento que tiene su atributo `id` establecido en "logo", o contiene `null` si no existe tal elemento. Si existen varios elementos con este ID, el documento no es válido y puede pasar cualquier cosa.

Recuperar por nombre de etiqueta

```
var elements = document.getElementsByTagName("a");
```

`elements` contendrán una `HTMLCollection` *vivo* (un objeto similar a una matriz) de todas las etiquetas de enlace en el documento. Esta colección está sincronizada con el DOM, por lo que los cambios realizados en el DOM se reflejan en esta colección. La colección proporciona acceso aleatorio y tiene una longitud.

```
var element = elements[0];
//Alternative
element = elements.item(0);
```

> `element` contiene el primer elemento de enlace HTML encontrado, o `null` si el índice está fuera de límites

```
var length = elements.length;
```

> `length` es igual al número de elementos de enlace HTML que se encuentran actualmente en la lista. Este número puede cambiar cuando se cambia el DOM.

Recuperar por clase

```
var elements = document.getElementsByClassName("recipe");
```

`elements` contendrán una `HTMLCollection` *vivo* (un objeto similar a una matriz) de todos los elementos donde su atributo de `class` incluye "receta". Esta colección está sincronizada con el DOM, por lo que los cambios realizados en el DOM se reflejan en esta colección. La colección proporciona acceso aleatorio y tiene una longitud.

```
var element = elements[0];
//Alternative
```

```
element = elements.item(0);
```

`element` contiene el primer elemento HTML encontrado con esta clase. Si no hay tales elementos, el `element` tiene el valor `undefined` en el primer ejemplo y `null` en el segundo ejemplo.

```
var length = elements.length;
```

`length` es igual al número de elementos HTML que actualmente tienen la clase "receta". Este número puede cambiar cuando se cambia el DOM.

Recuperar por nombre

```
var elements = document.getElementsByName("zipcode");
```

`elements` contendrán un `NodeList` *vivo* (un objeto similar a una matriz) de todos los elementos con su atributo de `name` establecido en "zipcode". Esta colección está sincronizada con el DOM, por lo que los cambios realizados en el DOM se reflejan en esta colección. La colección proporciona acceso aleatorio y tiene una longitud.

```
var element = elements[0];
//Alternative
element = elements.item(0);
```

`element` contiene el primer elemento HTML encontrado con este nombre.

```
var length = elements.length;
```

`length` es igual a la cantidad de elementos HTML que actualmente tienen "código postal" como atributo de nombre. Este número puede cambiar cuando se cambia el DOM.

Empezando

El DOM (Document Object Model) es la interfaz de programación para documentos HTML y XML, define la estructura lógica de los documentos y la forma en que se accede y se manipula a un documento.

Los principales implementadores de la API DOM son los navegadores web. Las especificaciones están estandarizadas por los grupos W3C y WHATWG , y el modelo de objeto especifica el modelo lógico para la interfaz de programación.

La representación de la estructura de DOM se asemeja a una vista en forma de árbol, donde cada nodo es un objeto que representa una parte del marcado, dependiendo del tipo en que cada elemento también hereda funciones específicas y compartidas.

Se eligió el nombre "Modelo de objetos de documento" porque es un "modelo de objeto" en el

sentido de diseño orientado a objetos tradicional: los documentos se modelan utilizando objetos, y el modelo abarca no solo la estructura de un documento, sino también el comportamiento de un documento y los objetos de los que se compone. En otras palabras, tomando el diagrama HTML de ejemplo, los nodos no representan una estructura de datos, representan objetos, que tienen funciones e identidad. Como modelo de objeto, el modelo de objeto de documento identifica:

- Las interfaces y objetos utilizados para representar y manipular un documento.
- Semántica de estas interfaces y objetos, incluidos tanto el comportamiento como los atributos.
- Las relaciones y colaboraciones entre estas interfaces y objetos.

Espera a que se cargue el DOM

Use `DOMContentLoaded` cuando el código `<script>` que interactúa con DOM se incluye en la sección `<head>` . Si no se `DOMContentLoaded` dentro de la `DOMContentLoaded` llamada `DOMContentLoaded` , el código arrojará errores como

> No se puede leer algo de `null`

```
document.addEventListener('DOMContentLoaded', function(event) {
    // Code that interacts with DOM
});
```

https://html.spec.whatwg.org/multipage/syntax.html#the-end

Alternativa a `DOMContentLoaded`

Una alternativa (adecuada para **IE8**)

```
// Alternative to DOMContentLoaded
document.onreadystatechange = function() {
    if (document.readyState === "interactive") {
        // initialize your DOM manipulation code here
    }
}
```

https://developer.mozilla.org/en/docs/Web/API/Document/readyState

Usar innerHTML

HTML

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

JS

```
document.getElementById('app').innerHTML = '<p>Some text</p>'
```

y ahora el HTML se ve así

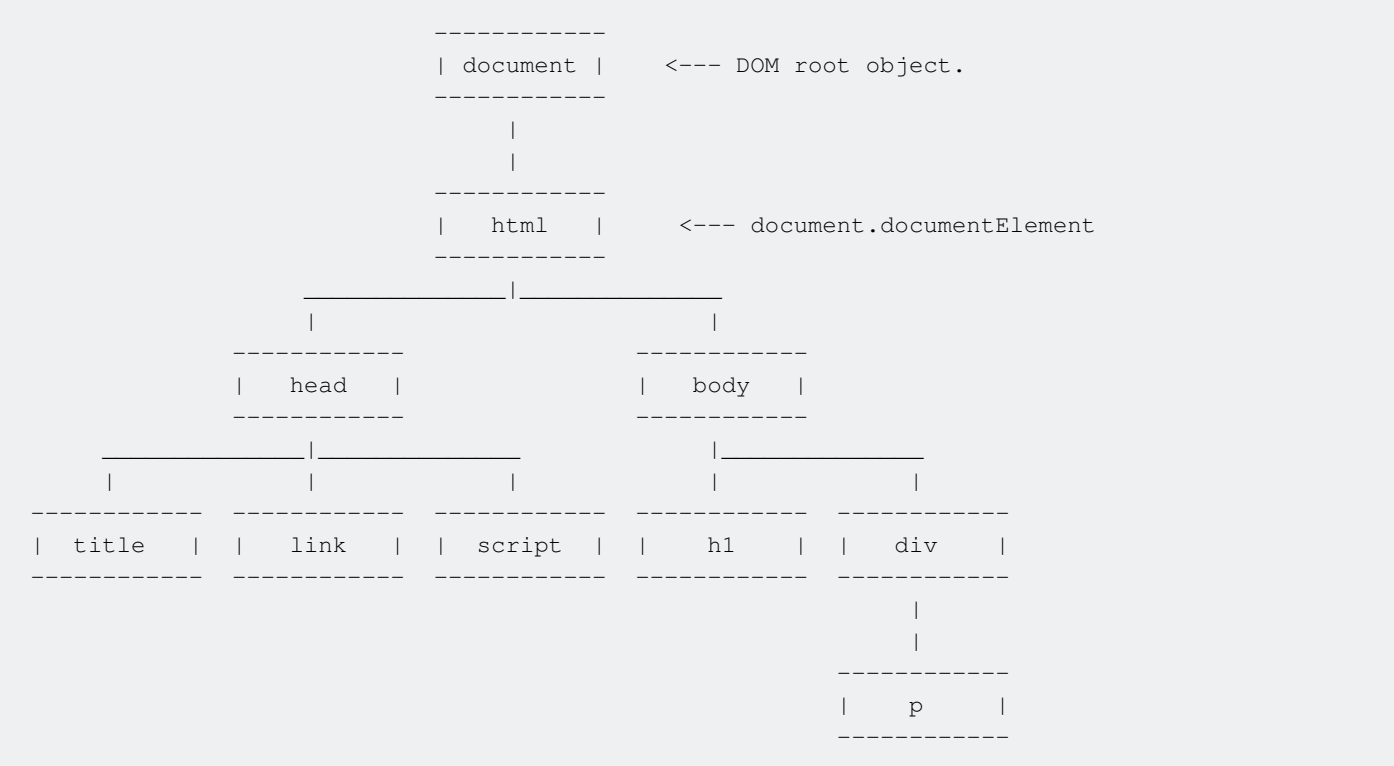
```
<div id="app">
    <p>Some text</p>
</div>
```

Marcado HTML

ejemplo de entrada:

```
<html>
  <head>
    <title>the title</title>
    <link href='css/app.css' type='text/css' rel='stylesheet'>
    <script src='js/app.js'></script>
  </head>
  <body>
    <h1>header</h1>
    <div>
      <p>hello!</p>
    </div>
  </body>
</html>
```

Salida del elemento DOM:



Todos los elementos anteriores se heredan de la interfaz HTMLElement y se personalizan dependiendo de la etiqueta específica

Lea Empezando con DOM en línea: https://riptutorial.com/es/dom/topic/2584/empezando-con-dom

Capítulo 2: Eventos

Parámetros

Parámetro	Descripción
tipo	`String` define el nombre del evento a escuchar.
oyente	`Function` dispara cuando ocurre el evento.
opciones	`Boolean` para establecer la captura, si `Object` puede establecer las siguientes propiedades en él, observe que la opción de objeto es débilmente compatible.
1. *captura*	Un valor booleano que indica que los eventos de este tipo se enviarán al oyente registrado antes de enviarse a cualquier EventTarget debajo de él en el árbol DOM.
2. *una vez*	Un valor booleano que indica que el oyente debe invocarse a lo sumo una vez después de agregarse. Si es verdad, el oyente se eliminaría automáticamente cuando se invoca.
3. *pasivo*	Un valor booleano que indica que el oyente nunca llamará a preventDefault (). Si lo hace, el agente de usuario debe ignorarlo y generar una advertencia de consola.

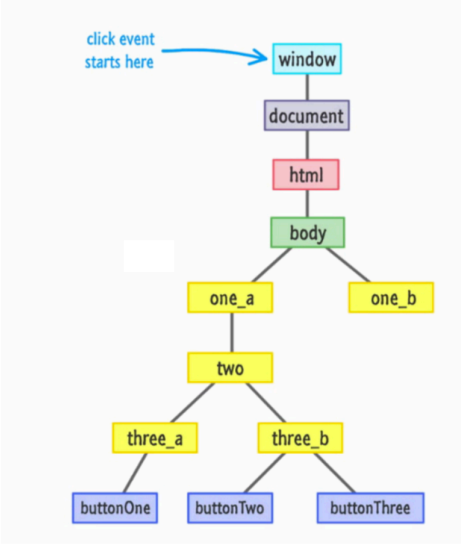
Observaciones

Origen de los acontecimientos

Los eventos no comienzan con lo que activó el evento (por ejemplo, un botón).

En lugar

Toca cada elemento en su camino e informa a cada elemento que un evento está sucediendo. Los eventos también regresan una vez que llegan a su destino, informando nuevamente a los elementos de su ocurrencia.

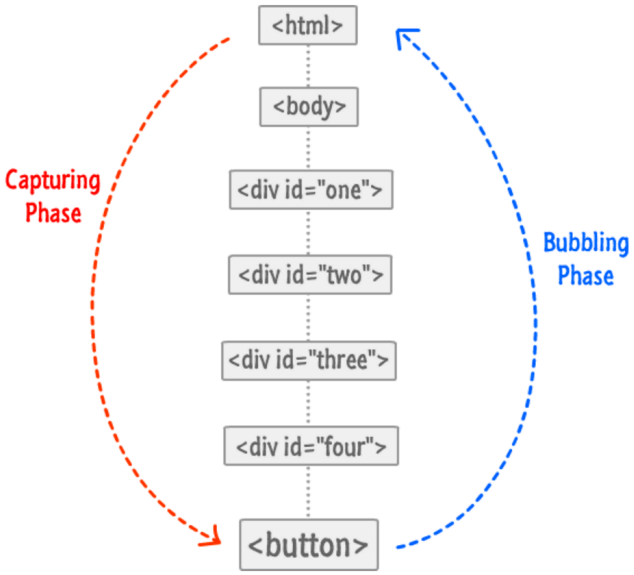


Capturando y burbujeando

Como aprendimos, los eventos comienzan desde la parte superior del árbol DOM, informan a cada nodo en su camino hacia su destino, luego regresan cuando llega a su destino, e informan a cada elemento que toca sobre su ocurrencia.

> Los eventos que pasan por el árbol DOM están en la **fase de captura** , los eventos que suben por el árbol DOM están en la **fase de propagación** .

Por defecto, los eventos se escuchan en la fase de propagación. Para cambiar esto, puede especificar en qué fase se escucha el evento especificando el tercer parámetro en la función addEventListener. (Ejemplo de código en la sección de *captura*)



Examples

Introducción

Definición:

> En computación, un evento es una acción u ocurrencia reconocida por el software que puede ser manejada por el software. Los eventos informáticos pueden ser generados o activados por el sistema, por el usuario o de otras maneras. *Fuente de definición*

Los eventos HTML son "cosas" que suceden a los elementos HTML. JavaScript puede "reaccionar" en estos eventos. a través de `Event Listeners` . Además, los eventos personalizados pueden activarse utilizando `dispatchEvent` . Pero esto es solo una introducción, ¡así que comencemos!

Escucha de eventos básicos

Para escuchar eventos, llame a `target.addEventListener(type, listener);`

```
function loadImage() {
  console.log('image code here!');
}
var myButton = document.querySelector('#my-button');
myButton.addEventListener('click', loadImage);
```

Esto activará loadImage cada vez que se haga clic en `my-button` .

Los detectores de eventos se pueden adjuntar a cualquier nodo en el árbol DOM. para ver una lista completa de todos los eventos activados de forma nativa en el navegador: haga clic aquí Enlace MDN para ver la lista completa de eventos

Eliminar oyentes de eventos

El método removeEventListener () elimina los controladores de eventos que se han asociado con el método addEventListener ():

```
element.removeEventListener("mousemove", myFunction);
```

Todo (nombre de evento, función y opciones) en el `removeEventListener` debe coincidir con el conjunto al agregar el detector de eventos al elemento.

.bind con removeListener

el uso de `.bind` en la función al agregar un detector de eventos evitará que la función se elimine, para eliminar realmente el eventListener que puede escribir:

```
function onEvent() {
   console.log(this.name);
}

var bindingOnEvent = onEvent.bind(this);

document.addEventListener('click', bindingOnEvent);

...

document.removeEventListener('click', bindingOnEvent);
```

escuchar un evento solo una vez

Hasta que `once` opción sea ampliamente compatible, debemos eliminar manualmente la escucha par una vez que el evento se active por primera vez.

Este pequeño ayudante nos ayudará a lograr esto:

```
Object.prototype.listenOnce = Object.prototype.listenOnce ||
  function listenOnce(eventName, eventHandler, options) {
      var target = this;
      target.addEventListener(eventName, function(e) {
          eventHandler(e);
          target.removeEventListener(eventName, eventHandler, options);
      }, options);
  }

var target = document.querySelector('#parent');
target.listenOnce("click", clickFunction, false);
```

** No es una buena práctica adjuntar funciones al prototipo de Objeto, por lo tanto, puede eliminar la primera línea de este código y agregarle un objetivo como primer parámetro.*

Esperando que el documento se cargue

Uno de los eventos más utilizados es esperar a que el documento se haya cargado, incluidos los archivos de script y las imágenes. El evento de `load` en el `document` se utiliza para esto.

```
document.addEventListener('load', function() {
   console.log("Everything has now loaded!");
});
```

En ocasiones, intenta acceder a un objeto DOM antes de que se cargue, lo que genera punteros nulos. Estos son realmente difíciles de depurar. Para evitar esto, use el evento `DOMContentLoaded` `document` en su lugar. `DOMContentLoaded` garantiza que el contenido HTML se haya cargado e inicializado sin esperar otros recursos externos.

```
document.addEventListener('DOMContentLoaded', function() {
   console.log("The document contents are now available!");
});
```

Objeto de evento

Para acceder al objeto de evento, incluya un parámetro de `event` en la función de devolución de llamada del detector de eventos:

```
var foo = document.getElementById("foo");
foo.addEventListener("click", onClick);

function onClick(event) {
  // the `event` parameter is the event object
  // e.g. `event.type` would be "click" in this case
};
```

e.stopPropagation ();

HTML:

```
<div id="parent">
   <div id="child"></div>
</div>
```

Javascript:

```
var parent = document.querySelector('#parent');
var child = document.querySelector('#child');

child.addEventListener('click', function(e) {
   e.stopPropagation();
   alert('child clicked!');
});
```

```
parent.addEventListener('click', function(e) {
   alert('parent clicked!');
});
```

ya que el niño detiene la propagación del evento, y los eventos se escuchan durante la fase de propagación, haciendo clic en el niño solo activará al niño. sin detener la propagación se activarán ambos eventos.

e.preventDefault ();

El método `event.preventDefault()` detiene la acción predeterminada de un elemento para que no ocurra.

Por ejemplo:

- Evitar que un botón de envío envíe un formulario
- Evita que un enlace siga la URL

```
var allAnchorTags = document.querySelector('a');

allAnchorTags.addEventListener('click', function(e){
    e.preventDefault();
    console.log('anchor tags are useless now! *evil laugh*');
});
```

e.target vs e.currentTarget

> e.currentTarget Identifica el destino actual para el evento, ya que el evento atraviesa el DOM. Siempre hace referencia al elemento al que se ha vinculado el controlador de eventos, en contraposición a event.target, que identifica el elemento en el que se produjo el evento.

en otras palabras

`e.target` devolverá lo que desencadena el dispatcher de eventos para desencadenar

`e.currentTarget` devolverá lo que asignó a su oyente.

HTML:

```
<body>
   <button id="my-button"></button>
</body>
```

Javascript:

```
var body = document.body;
body.addEventListener( 'click', function(e) {
    console.log('e.target', e.target);
    console.log('e.currentTarget', e.currentTarget);
});
```

si haces clic en `my-button` ,

- **e.target** será `my-button`
- **e.currentTarget** será `body`

Evento de burbujas y captura

Los eventos activados en elementos DOM no solo afectan el elemento al que se dirigen. Cualquiera de los ancestros del objetivo en el DOM también puede tener la oportunidad de reaccionar ante el evento. Considere el siguiente documento:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
</head>
<body>
  <p id="paragraph">
    <span id="text">Hello World</span>
  </p>
</body>
</html>
```

Si solo agregamos escuchas a cada elemento sin ninguna opción, entonces activamos un clic en el intervalo ...

```
document.body.addEventListener('click', function(event) {
  console.log("Body clicked!");
});
window.paragraph.addEventListener('click', function(event) {
  console.log("Paragraph clicked!");
});
window.text.addEventListener('click', function(event) {
  console.log("Text clicked!");
});

window.text.click();
```

... entonces el evento se **propagará** a través de cada antepasado, activando cada controlador de clic en el camino:

```
Text clicked!
Paragraph clicked!
Body clicked!
```

Si desea que uno de sus manejadores impida que el evento `event.stopPropagation()` más manejadores, puede llamar al método `event.stopPropagation()` . Por ejemplo, si reemplazamos

nuestro segundo controlador de eventos con esto:

```
window.paragraph.addEventListener('click', function(event) {
  console.log("Paragraph clicked, and that's it!");
  event.stopPropagation();
});
```

Veríamos la siguiente salida, con el manejador de `click` `body` nunca activado:

```
Text clicked!
Paragraph clicked, and that's it!
```

Finalmente, tenemos la opción de agregar detectores de eventos que se activan durante la " **captura** " en lugar de burbujear. Antes de que un evento brote de un elemento a través de sus antepasados, primero se "captura" hasta el elemento a través de sus antepasados. Se agrega una escucha de captura especificando `true` o `{capture: true}` como el tercer argumento opcional para `addEventListener` . Si agregamos los siguientes oyentes a nuestro primer ejemplo anterior:

```
document.body.addEventListener('click', function(event) {
  console.log("Body click captured!");
}, true);
window.paragraph.addEventListener('click', function(event) {
  console.log("Paragraph click captured!");
}, true);
window.text.addEventListener('click', function(event) {
  console.log("Text click captured!");
}, true);
```

Obtendremos la siguiente salida:

```
Body click captured!
Paragraph click captured!
Text click captured!
Text clicked!
Paragraph clicked!
Body clicked!
```

Por defecto, los eventos se escuchan en la fase de propagación. Para cambiar esto, puede especificar en qué fase se escucha el evento especificando el tercer parámetro en la función addEventListener. (Para aprender sobre la captura y el burbujeo, verifique los *comentarios*)

```
element.addEventListener(eventName, eventHandler, useCapture)
```

useCapture: `true` significa escuchar el evento cuando se está ejecutando en el árbol DOM. `false` significa escuchar el evento mientras sube el árbol DOM.

```
window.addEventListener("click", function(){alert('1: on bubble')}, false);
window.addEventListener("click", function(){alert('2: on capture')}, true);
```

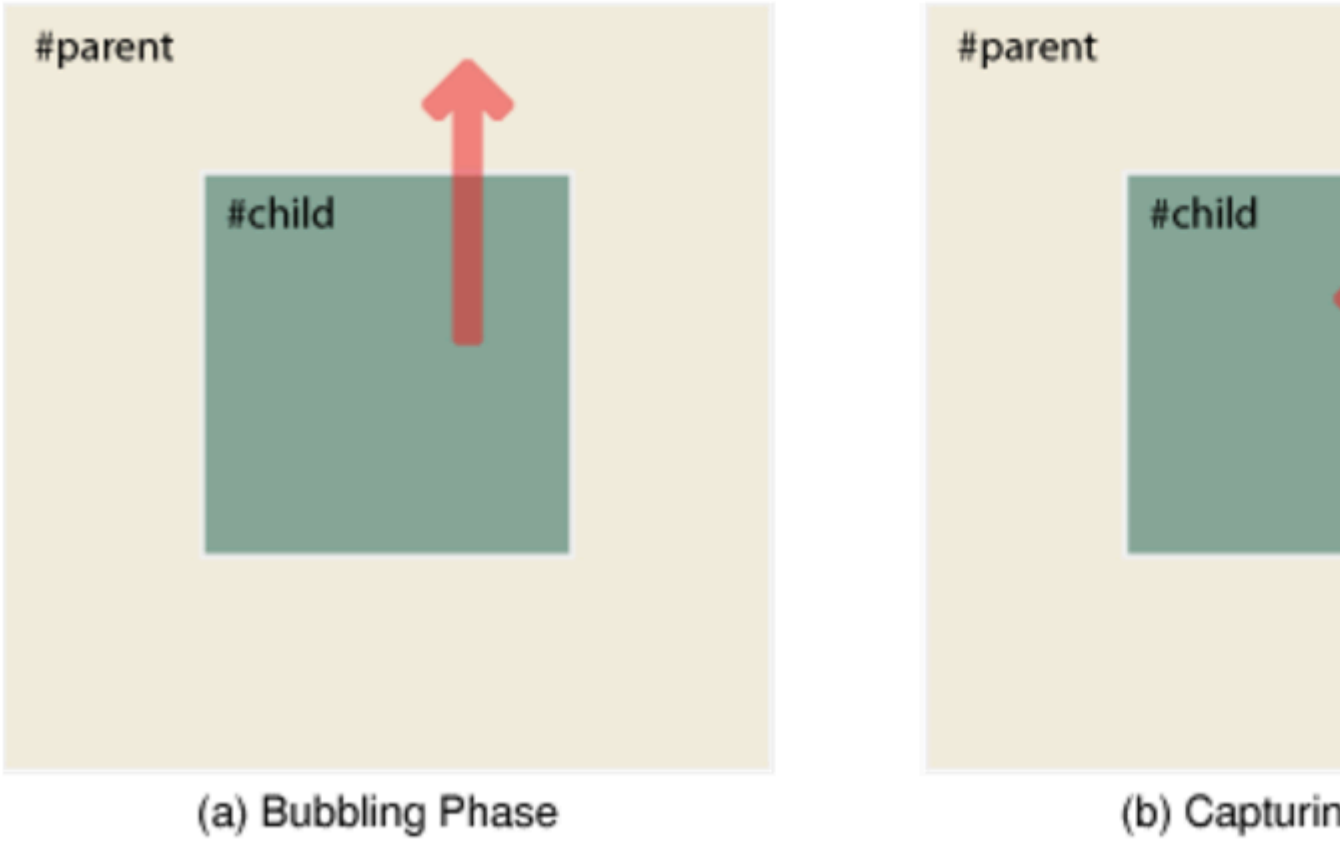
Los cuadros de alerta aparecerán en este orden:

- 2: en la captura
- 1: en burbuja

Casos de uso en el mundo real.

El Evento de captura se enviará antes del Evento de burbuja, por lo que puede asegurarse de que un evento se escuche primero si lo escucha en su fase de captura.

Si está escuchando un evento de clic en un elemento padre y otro en su hijo, puede escuchar primero al niño o al padre, según cómo cambie el parámetro useCapture.



en el burbujeo, el evento hijo se llama primero, en la captura, el padre primero

HTML:

```
<div id="parent">
   <div id="child"></div>
</div>
```

Javascript:

```
child.addEventListener('click', function(e) {
   alert('child clicked!');
});

parent.addEventListener('click', function(e) {
   alert('parent clicked!');
}, true);
```

Si se establece fiel al evento primario, el EventListener activará primero el escucha principal.

Combinado con e.stopPropagation () puede evitar que el evento active al oyente de eventos secundario o al padre. (más sobre eso en el siguiente ejemplo)

Delegación de eventos

La delegación de eventos es un proceso que nos permite evitar agregar escuchas de eventos a nodos específicos; en su lugar, el detector de eventos se agrega al nodo principal. Este mecanismo utiliza la propagación / propagación de eventos para controlar un evento en un elemento / nodo de nivel superior en el DOM en lugar de utilizar el elemento en el que se originó el evento. Por ejemplo, creo que necesitamos agregar eventos para los siguientes elementos de la lista:

```
<ul id="container">
    <li id="item-1" class="new">Item 1</li>
    <li id="item-2">Item 2</li>
    <li id="item-3">Item 3</li>
</ul>
```

Necesitamos agregar manejadores de `click` y, básicamente, podemos agregar oyentes a cada elemento utilizando un bucle, pero imaginemos que queremos agregar elementos dinámicamente. Por lo tanto, registramos todos los controladores de eventos cuando se carga el DOM y después de que el DOM se inicializa y registra todos los controladores de eventos para cada elemento, el elemento recién insertado en el `UL` anterior no responderá al hacer clic porque ese elemento no estaba presente en el DOM Cuando hayamos registrado los oyentes de click click.

Entonces, para superar este problema, podemos aprovechar la delegación de eventos. Lo que significa que, en lugar de registrar los oyentes en cada uno de los elementos `li` , podemos vincular al oyente del evento a su elemento `UL` primario, por ejemplo:

```
document.getElementById("container").addEventListener("click", function(e) {
    console.log("List item " e.target.id, " was clicked!");
});
```

Dado que, el evento se propaga (burbujea hacia arriba) de forma predeterminada, luego, al hacer clic en cualquier elemento `LI` , el elemento `UL` activará el mismo evento. En este caso, podemos usar el parámetro `e` en la función, que es en realidad el objeto de evento y contiene información útil sobre el evento, incluido el elemento original, que inició el evento. Entonces, por ejemplo, podemos usar algo como lo siguiente:

```
document.getElementById("container").addEventListener("click", function(e) {
```

```
    // If UL itself then no action is require
    if(e.target.nodeName == 'UL') return false;

    if(e.target.classList.contains('new')) {
        console.log("List item " e.target.id, " was clicked and it's new!");
    }
});
```

Entonces, es obvio que `e` (objeto de evento) nos permite examinar el elemento fuente (e.target) y podemos inyectar fácilmente nuevos elementos a la `UL` después de cargar el DOM y el único controlador de eventos delegado manejará todos los eventos de clic. dentro del `UL` primario, que también consume menos memoria porque declaramos una sola función para todos los elementos.

Activación de eventos personalizados.

La API CustomEvent permite a los desarrolladores crear eventos personalizados y activarlos en nodos DOM, pasando datos a lo largo del camino.

```
event = new CustomEvent(typeArg, customEventInit);
```

typeArg - DOMString que representa el nombre del evento.

customEventInit: son parámetros opcionales (que se pasarán como `e` en el siguiente ejemplo).

Puede adjuntar `eventListeners` al `document` o *cualquier* elemento HTML.

Una vez que se haya agregado un evento personalizado y se haya vinculado al elemento (o documento), es posible que desee dispararlo manualmente desde javascript.

```
document.addEventListener("event-name", function(e) {
  console.log(e.detail); // logs custom object passed from the event.
});

var event = new CustomEvent("event-name", { "param-name": "param-value" });
document.dispatchEvent(event);
```

Lea Eventos en línea: https://riptutorial.com/es/dom/topic/5388/eventos

Capítulo 3: Manipulando atributos

Observaciones

Los atributos son un tipo específico de objeto en la API DOM. En versiones anteriores de la API DOM, se heredaban del tipo de `Node` , pero esto se cambió en la versión 4.

En los ejemplos que se refieren al `dataset` de `dataset` , los "navegadores modernos" excluyen específicamente las versiones de Internet Explorer de menos de 11. Visite caniuse.com para obtener información más actualizada.

Examples

Obteniendo un atributo

Algunos atributos son directamente accesibles como propiedades del elemento (por ejemplo, `alt` , `href` , `id` , `title` y `value`).

```
var a = document.querySelector("a"),
   url = a.href;
```

Se puede acceder a otros atributos, incluidos los atributos de datos, de la siguiente manera:

```
var a = document.querySelector("a"),
   tooltip = a.getAttribute("aria-label");
```

A los atributos de los datos también se puede acceder usando un `dataset` (navegadores modernos)

```
// <a href="#" data-tracking-number="ABC-123">Widget</a>
var a = document.querySelector("a"),
  tracker = a.dataset.trackingNumber;
```

Estableciendo un atributo

Algunos atributos son directamente accesibles como propiedades del elemento (por ejemplo, alt, href, id, title y value).

```
document.querySelector("a").href = "#top";
```

Otros atributos, incluidos los atributos de datos, se pueden configurar de la siguiente manera:

```
document.querySelector("a").setAttribute("aria-label", "I like turtles");
```

Los atributos de los datos también se pueden configurar utilizando un conjunto de datos

(navegadores modernos)

```
var a = document.querySelector("a");
a.dataset.test = "123";
a.dataset['test-2'] = "456";
```

resultados en

```
<a href="#" data-test="123" data-test-2="456">Widget</a>
```

Eliminando un atributo

Para eliminar un atributo, incluyendo propiedades directamente accesibles

```
document.querySelector("a").removeAttribute("title");
```

Los atributos de los datos también se pueden eliminar de la siguiente manera (navegadores modernos):

```
// remove "data-foo" attribute
delete document.querySelector("a").dataset.foo;
```

Lea Manipulando atributos en línea: https://riptutorial.com/es/dom/topic/5236/manipulando-atributos

Capítulo 4: Manipulando Elementos

Examples

Elementos de clonacion

Un elemento se puede clonar invocando el método `cloneNode` . Si el primer parámetro pasado a `cloneNode` es `true` , los hijos del original también se clonarán.

```
var original = document.getElementsByTagName("li")[0];
var clone = original.cloneNode(true);
```

Añadiendo un elemento

En este ejemplo, creamos un nuevo elemento de lista con el texto "nuevo texto", y seleccionamos la primera lista desordenada y su primer elemento de lista.

```
let newElement = document.createElement("li");
newElement.innerHTML = "new text";

let parentElement = document.querySelector("ul");
let nextSibling = parentElement.querySelector("li");
```

Al insertar un elemento, lo hacemos *debajo* del elemento primario, y justo antes de un elemento secundario particular de ese elemento principal.

```
parentElement.insertBefore(newElement, nextSibling);
```

> El nuevo elemento se inserta debajo de `parentElement` y justo antes de `nextSibling` .

Cuando uno quiere insertar un elemento como el último elemento secundario de `parentElement` , el segundo argumento puede ser `null` .

```
parentElement.insertBefore(newElement, null);
```

> El nuevo elemento se inserta bajo `parentElement` como el último elemento secundario.

En su lugar, `appendChild()` se puede usar para simplemente agregar el hijo a los hijos del nodo padre.

```
parentElement.appendChild(newElement);
```

> El nuevo elemento se inserta bajo `parentElement` como el último elemento secundario.

Reemplazo de un elemento

En este ejemplo, creamos un nuevo elemento de lista con el texto "nuevo texto", y seleccionamos la primera lista desordenada y su primer elemento de lista.

```
let newElement = document.createElement("li");
newElement.innerHTML = "new text";

let parentElement = document.querySelector("ul");
let nextSibling = parentElement.querySelector("li");
```

Para reemplazar un elemento, usamos `replaceChild` :

```
parentElement.replaceChild(newElement, nextSibling);
```

`nextSibling` se elimina del DOM. En su lugar está ahora el elemento `newElement` .

Eliminando un elemento

Se puede eliminar un elemento llamando a `remove()` en él. Alternativamente, uno puede llamar a `removeChild()` en su padre. `removeChild()` tiene mejor soporte de navegador que `remove()` .

```
element.remove();
```

`element` , y todos sus childnodes, se eliminan del DOM.

```
parentElement.removeChild(element);
```

`element` , y todos sus childnodes, se eliminan del DOM.

En cualquier caso, se puede insertar este nodo en el DOM en un momento posterior, siempre y cuando todavía haya referencias a este nodo.

Añadir y Preponer métodos

JavaScript ahora tiene los métodos Append y Prepend que estaban presentes en jQuery

La principal ventaja de `append` y `prepend` es a diferencia de `appendChild` e `insertBefore` , puede tomar cualquier número de argumentos, ya sea elemento HTML o texto sin formato (que se convertirá en nodos de texto).

Para añadir dig 1 div, 1 nodo de texto y 1 tramo

```
document.body.append(document.createElement('div'),"Hello
world",document.createElement('span'))
```

Esto cambiará la página a la siguiente estructura

```
<body>
      .....(other elements)
      <div></div>
```

```
      "Hello World"
      <span></span>
</body>
```

Anteponer lo mismo en cuerpo.

Utilizar

```
document.body.prepend(document.createElement('div'),"Hello
world",document.createElement('span'))
```

Esto cambiará la página a la siguiente estructura

```
<body>
      <div></div>
      "Hello World"
      <span></span>
      .....(other elements)
</body>
```

Tenga en cuenta que los soportes del navegador son

Chrome 54+
Firefox 49+
Opera 39+

Lea más en MDN

Adjuntar

Prepend

Lea Manipulando Elementos en línea: https://riptutorial.com/es/dom/topic/5200/manipulando-elementos

Capítulo 5: Manipulando una lista de clases de CSS

Examples

Añadiendo una clase

Los navegadores modernos proporcionan un objeto `classList` para facilitar la manipulación del atributo de clase del elemento. Los navegadores más antiguos requieren la manipulación directa de la propiedad `className` del elemento.

W3C DOM 4

Un método simple para agregar una clase a un elemento es agregarlo al final de la propiedad `className` . Esto no evitará nombres de clase duplicados, y los espacios **deben** incluirse entre los nombres de clase.

```
document.getElementById("link1").className += " foo";
document.getElementById("link2").className += " foo bar";
```

Para elementos múltiples, deberá agregar los nombres de clase dentro de un bucle

```
var els = document.getElementsByClassName("foo"),
  indx = els.length;
while (indx--) {
  els[indx].className += " bar baz";
}
```

W3C DOM 4

Se puede agregar un solo nombre de clase como una cadena. Para agregar varios nombres de clase, use el operador de propagación de ES6:

```
document.querySelector("#link1").classList.add("foo");
document.querySelector("#link2").classList.add(...['foo', 'bar']);
```

Para elementos múltiples, deberá agregar los nombres de clase dentro de un bucle

```
document.querySelectorAll(".foo").forEach(el => {
  el.classList.add(...['bar', 'baz']);
});
```

Eliminando una clase

Los navegadores modernos proporcionan un objeto `classList` para facilitar la manipulación del atributo de clase del elemento. Los navegadores más antiguos requieren la manipulación directa

de la propiedad `className` del elemento.

* Los nombres de clase de nota no se almacenan en la propiedad del elemento en ningún orden particular

W3C DOM 4

Eliminar una clase de un elemento requiere un poco de manipulación de la propiedad `className` .

```
var toRemove = "bar",
  el = document.getElementById("link1");
el.className = el.className.replace(new RegExp("\\b" + toRemove + "\\b", "g"), "").trim();
```

Eliminar varios nombres de clase requeriría un bucle. Los ejemplos restantes utilizarán una función para aislar el trabajo.

```
function removeClass(el, name) {
  name = name.split(/\s+/);
  var index = name.length,
    classes = el.className;
  while (index--) {
    classes = classes.replace(new RegExp("\\b" + name[index] + "\\b", "g"), "").trim();
  }
  el.className = classes;
}
var el = document.getElementById("link1");
removeClass(el, "bar baz");
```

Múltiples elementos con múltiples nombres de clase para eliminar requerirían dos bucles

```
function removeClass(els, name) {
  name = name.split(/\s+/);
  var regex, len,
    index = name.length;
  while (index--) {
    regex = new RegExp("\\b" + name[index] + "\\b", "g");
    len = els.length;
    while (len--) {
      els[len].className = els[len].className.replace(regex, "").trim();
    }
  }
}
var els = document.getElementsByTagName("a");
removeClass(els, "bar baz");
```

W3C DOM 4

Un solo nombre de clase puede ser eliminado como una cadena. Para eliminar varios nombres de clase, use el operador de propagación de ES6:

```
document.querySelector("#link1").classList.remove("foo");
document.querySelector("#link2").classList.remove(...['foo', 'bar']);
```

Para varios elementos, deberá eliminar los nombres de clase dentro de un bucle

```
document.querySelectorAll(".foo").forEach(el => {
  el.classList.remove(...['bar', 'baz']);
});
```

Pruebas para una clase

Los navegadores modernos proporcionan un objeto `classList` para facilitar la manipulación del atributo de clase del elemento. Los navegadores más antiguos requieren la manipulación directa de la propiedad `className` del elemento.

* Los nombres de clase de nota no se almacenan en la propiedad del elemento en ningún orden particular

W3C DOM 4

Probar si un elemento contiene una clase requiere un poco de manipulación de la propiedad `className` . Este ejemplo utiliza un método de matriz para probar la clase.

```
function hasClass(el, name) {
  var classes = (el && el.className || "").split(/\s+/);
  return classes.indexOf(name) > -1;
}
var el = document.getElementById("link1");
console.log(hasClass(el, "foo"));
```

La prueba de varios nombres de clase requeriría un bucle.

```
function hasClass(el, name) {
  name = name.split(/[\s.]+/);
  var hasClass = true,
    classes = (el && el.className || "").split(/\s+/),
    index = name.length;
  while (index--) {
    hasClass = hasClass && classes.indexOf(name[index]) > -1;
  }
  return hasClass;
}
var el = document.getElementById("link1");
console.log(hasClass(el, "foo"));
```

En lugar de usar `.indexOf()` , también puede considerar usar una expresión regular.

```
function hasClass(el, name) {
  return new RegExp("\\b" + name+ "\\b").test(el.className);
}
var el = document.getElementById("link1");
console.log(hasClass(el, "foo"));
```

W3C DOM 4

La prueba de un solo nombre de clase se realiza de la siguiente manera:

```
var hasClass = document.querySelector("#link1").classList.contains("foo");
```

Para múltiples nombres de clase, es más fácil usar `matches` . Tenga en cuenta el uso del selector de clase; El selector puede ser cualquier selector de cadena válido (id, atributo, pseudo-clases, etc.).

```
var hasClass = document.querySelector("#link1").matches('.foo.bar');
var hasClass = document.querySelector("#link2").matches('a.bar[href]');
```

Lea Manipulando una lista de clases de CSS en línea:
https://riptutorial.com/es/dom/topic/5865/manipulando-una-lista-de-clases-de-css

Capítulo 6: Recuperando elementos

Examples

Por identificación

```
document.getElementById('uniqueID')
```

recuperará

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

Mientras exista un elemento con la ID dada, `document.getElementById` devolverá solo ese elemento. De lo contrario, se devolverá `null` .

Nota: las identificaciones deben ser únicas. Varios elementos no pueden tener la misma ID.

Por nombre de clase

```
document.getElementsByClassName('class-name')
```

recuperará

```
<a class="class-name">Any</a>
<b class="class-name">tag</b>
<div class="class-name an-extra-class">with that class.</div>
```

Si ningún elemento existente contiene la clase dada, se devolverá una colección vacía.

Ejemplo:

```
<p class="my-class">I will be matched</p>
<p class="my-class another-class">So will I</p>
<p class="something-else">I won't</p>
```

```
var myClassElements = document.getElementByClassName('my-class');
console.log(myClassElements.length); // 2
var nonExistentClassElements = document.getElementByClassName('nope');
console.log(nonExistentClassElements.length); // 0
```

Por nombre de etiqueta

```
document.getElementsByTagName('b')
```

recuperará

```
<b>All</b>
<b>of</b>
<b>the b elements.</b>
```

Si no existen elementos con el nombre de etiqueta dado, se devolverá una colección vacía.

Por CSS Selector

Considere seguir el código html

```
<ul>
  <li id="one" class="main">Item 1</li>
  <li id="two" class="main">Item 2</li>
  <li id="three" class="main">Item 3</li>
  <li id="four">Item 4</li>
</ul>
```

El siguiente árbol dom se construirá basándose en el código html anterior

```
                      ul

                      |

     |          |          |          |

    li         li         li         li
     |          |          |          |
Item 1        Item 2    Item 3    Item 4
```

Podemos seleccionar elementos del árbol DOM con la ayuda de selectores de CSS. Esto es posible por medio de dos métodos de javascript: `querySelector()` y `querySelectorAll()` .

El método **querySelector ()** devuelve el primer elemento que coincide con el selector css dado desde el DOM.

```
document.querySelector('li.main')
```

devuelve el primer elemento `li` quien es la clase `main`

```
document.querySelector('#two')
```

devuelve el elemento con id `two`

NOTA: Si no se encuentra ningún elemento se devuelve un `null` . Si la cadena del selector contiene un pseudo-elemento CSS, el retorno será `null` .

El método **querySelectorAll ()** devuelve todos los elementos que coinciden con el selector css dado desde el DOM.

```
document.querySelectorAll('li.main')
```

devuelve una lista de nodos que contiene todos los elementos `li` cuya clase es `main` .

NOTA : Si no se encuentra ningún elemento, se devuelve una lista de nodos vacía. Si la cadena de selectores contiene un pseudo-elemento CSS, la elementList devuelta estará vacía

Selectores de consultas

En los navegadores modernos [1] , es posible usar el selector tipo CSS para consultar los elementos de un documento, de la misma forma que sizzle.js (utilizado por jQuery).

querySelector

Devuelve el primer `Element` en el documento que coincide con la consulta. Si no hay coincidencia, devuelve `null` .

```
// gets the element whose id="some-id"
var el1 = document.querySelector('#some-id');

// gets the first element in the document containing "class-name" in attribute class
var el2 = document.querySelector('.class-name');

// gets the first anchor element in the document
var el2 = document.querySelector('a');

// gets the first anchor element inside a section element in the document
var el2 = document.querySelector('section a');
```

querySelectorAll

Devuelve una lista de `NodeList` contiene todos los elementos del documento que coinciden con la consulta. Si ninguno coincide, devuelve un `NodeList` vacío.

```
// gets all elements in the document containing "class-name" in attribute class
var el2 = document.querySelectorAll('.class-name');

// gets all anchor elements in the document
var el2 = document.querySelectorAll('a');

// gets all anchor elements inside any section element in the document
var el2 = document.querySelectorAll('section a');
```

Lea Recuperando elementos en línea: https://riptutorial.com/es/dom/topic/2658/recuperando-elementos

Capítulo 7: Travesía

Examples

Árbol caminando

`TreeWalker` es una interfaz similar a un generador que hace que el filtrado recursivo de nodos en un árbol DOM sea fácil y eficiente.

El siguiente código concatena el valor de todos los nodos de `Text` en la página e imprime el resultado.

```
let parentNode = document.body;
let treeWalker = document.createTreeWalker(parentNode, NodeFilter.SHOW_TEXT);

let text = "";
while (treeWalker.nextNode())
    text += treeWalker.currentNode.nodeValue;

console.log(text); // all text in the page, concatenated
```

La función `.createTreeWalker` tiene una firma de

```
createTreeWalker(root, whatToShow, filter, entityReferenceExpansion)
```

Parámetro	Detalles
raíz	El nodo 'raíz' cuyo subárbol debe ser recorrido
que hacer	Opcional, sin signo largo que designa qué tipos de nodos mostrar. Ver NodeFilter para más información.
filtrar	Opcional, un objeto con un método acceptNode para determinar si un nodo, después de pasar la comprobación de whatToShow debe considerarse
entityReferenceExpansion	Obsoleto y opcional, es una bandera booleana que indica si al descartar una EntityReference todo su subárbol debe descartarse al mismo tiempo.

Iterando sobre nodos

La interfaz NodeIterator proporciona métodos para iterar sobre nodos en un árbol DOM.

Dado un documento como este:

```
<html>
```

```
<body>
  <section class="main">
    <ul>
      <li>List Item</li>
      <li>List Item</li>
      <li>List Item</li>
      <li>List Item</li>
    </ul>
  </section>
</body>
</html>
```

Uno podría imaginar un iterador para obtener los elementos `<li>` :

```
let root = document.body;
let whatToShow = NodeFilter.SHOW_ELEMENT | NodeFilter.SHOW_TEXT;
let filter = (node) => node.nodeName.toLowerCase() === 'li' ?
  NodeFilter.FILTER_ACCEPT :
  NodeFilter.FILTER_REJECT;
let iterator = document.createNodeIterator(root, whatToShow, filter);
var node;
while (node = iterator.nextNode()) {
  console.log(node);
}
```

Ejemplo adaptado del ejemplo proporcionado por los Colaboradores de Mozilla de la `document.createNodeIterator()` en la Red de Desarrolladores de Mozilla, con licencia CC-by-SA 2.5 .

Esto registrará algo como:

```
<li>List Item</li>
<li>List Item</li>
<li>List Item</li>
<li>List Item</li>
```

Tenga en cuenta que esto es similar a la interfaz de árbol de TreeWalker, pero proporciona solo las `nextNode()` y `previousNode()` .

Lea Travesía en línea: https://riptutorial.com/es/dom/topic/5261/travesia

Capítulo 8: Usando estilos CSS

Observaciones

Las interfaces detalladas en este documento se introdujeron en DOM Level 2 Style , que salió aproximadamente al mismo tiempo que DOM Level 2 Core y, por lo tanto, se considera "parte de DOM versión 2".

Examples

Leyendo y cambiando estilos en línea.

Estilo en linea

Puede manipular el estilo CSS en línea de un elemento HTML simplemente leyendo o editando su propiedad de `style` .

Supongamos el siguiente elemento:

```
<div id="element_id" style="color:blue;width:200px;">abc</div>
```

Con este JavaScript aplicado:

```
var element = document.getElementById('element_id');

// read the color
console.log(element.style.color); // blue

//Set the color to red
element.style.color = 'red';

//To remove a property, set it to null
element.style.width = null;
element.style.height = null;
```

Sin embargo, si `width: 200px;` se establecieron en una hoja de estilo CSS externa, `element.style.width = null` no tendría ningún efecto. En este caso, para restablecer el estilo, tendría que configurarlo en `initial` : `element.style.width = 'initial'` .

Leyendo y cambiando estilos de una hoja de estilo

`element.style` solo lee las propiedades CSS establecidas en línea, como un atributo de elemento. Sin embargo, los estilos a menudo se establecen en una hoja de estilo externa. Se puede acceder al estilo real de un elemento con `window.getComputedStyle(element)` . Esta función devuelve un objeto que contiene el valor real calculado de todos los estilos.

Similar al ejemplo de lectura y cambio de estilos en línea, pero ahora los estilos están en una hoja

de estilo:

```
<div id="element_id">abc</div>
<style type="text/css">
    #element_id {
        color:blue;
        width:200px;
    }
</style>
```

JavaScript:

```
var element = document.getElementById('element_id');

// read the color
console.log(element.style.color); // '' -- empty string
console.log(window.getComputedStyle(element).color); // rgb(0, 0, 255)

// read the width, reset it, then read it again
console.log(window.getComputedStyle(element).width); // 200px
element.style.width = 'initial';
console.log(window.getComputedStyle(element).width); // 885px (for example)
```

Lea Usando estilos CSS en línea: https://riptutorial.com/es/dom/topic/5595/usando-estilos-css

Creditos

S. No	Capítulos	Contributors
1	Empezando con DOM	Blackus, Community, D.J., Dr. J. Testington, Henrique Barcelos, Jonas S, Leon Byford, maioman, Mike McCaughan, Mikhail, mnoronha, Mottie, Noushad PP, Roko C. Buljan, rvighne, Scimonster, Shog9, Sumurai8, Tushar
2	Eventos	Bamieh, Ian, Jeremy Banks, kamoroso94, Matas Vaitkevicius, Mike McCaughan, Mottie, Rap, The Alpha, Thriggle, zer00ne
3	Manipulando atributos	Mike McCaughan, Mottie
4	Manipulando Elementos	Mike McCaughan, mnoronha, Sagar V, Sumurai8
5	Manipulando una lista de clases de CSS	Mike McCaughan, Mottie, Shog9
6	Recuperando elementos	geeksal, Henrique Barcelos, maioman, Mike C, Mike McCaughan
7	Travesía	Jonas S, Mike McCaughan, rvighne
8	Usando estilos CSS	Blackus, Mike McCaughan, Scimonster