Youtic's
Lic. Gabriel Chaldú

DOM - Document Object Model

En JavaScript, el término "DOM" significa "Document Object Model" (Modelo de Objetos del Documento). El DOM es una representación de la estructura de un documento HTML o XML que permite a los programadores acceder y manipular los elementos de la página web de manera dinámica. El DOM se crea automáticamente cuando se carga una página web en un navegador y proporciona una interfaz de programación que permite a los desarrolladores web interactuar con los elementos de la página, como elementos HTML, atributos, texto y más.

El DOM organiza los elementos de la página web en una estructura de árbol, donde cada elemento HTML es representado como un nodo en el árbol. Estos nodos pueden ser elementos HTML, atributos, texto u otros tipos de datos relacionados con el documento. Los programadores pueden utilizar JavaScript para acceder a estos nodos, cambiar su contenido, estilo o posición en la página, y responder a eventos del usuario, como clics y pulsaciones de teclas.

El uso del DOM es fundamental en el desarrollo web moderno, ya que permite la creación de aplicaciones web interactivas y dinámicas al permitir la manipulación y actualización de elementos de la página sin necesidad de recargarla por completo.

Document

El objeto `document` en el DOM (Modelo de Objetos del Documento) es un objeto central que representa todo el documento HTML actual en un navegador web. Es un punto de entrada esencial para interactuar con la estructura y el contenido de una página web desde JavaScript. Algunos de los métodos y atributos más utilizados asociados con el objeto `document` incluyen:

Métodos más utilizados:

1. **getElementById(id)**: Este método permite acceder a un elemento HTML por su atributo `id`. Retorna una referencia al elemento encontrado o `null` si no se encuentra ningún elemento con ese `id`.

2. **getElementsByClassName(className):** Este método devuelve una lista de elementos que tienen una clase específica. Puedes acceder a los elementos individuales de la lista por índice

3. **getElementsByTagName(tagName):** Devuelve una lista de elementos con un nombre de etiqueta HTML específico (por ejemplo, 'div', 'p', 'a', etc.).

4. **querySelector(selector):** Permite seleccionar un elemento utilizando un selector CSS. Devuelve el primer elemento que coincide con el selector.

5. **querySelectorAll(selector):** Similar a `querySelector`, pero devuelve una lista de todos los elementos que coinciden con el selector.

Youtic's
Lic. Gabriel Chaldú

Atributos más utilizados:

1. **document.title:** Permite acceder al título de la página y modificarlo.

2. **document.body**: Proporciona acceso al elemento `<body>` del documento, que es donde generalmente se coloca el contenido principal de la página.

3. **document.cookie**: Permite leer y escribir cookies del navegador.

4. **document.documentElement**: Representa el elemento raíz del documento, que es `<html>`.

5. **document.head**: Proporciona acceso al elemento `<head>` del documento, que se utiliza para incluir metadatos y enlaces a hojas de estilo y scripts.

Estos son solo algunos ejemplos de los métodos y atributos más comunes del objeto `document`. El DOM ofrece muchas más funcionalidades para interactuar con elementos, manipular contenido y responder a eventos en una página web.

```
var elemento = document.getElementById('miElemento');
var elementos = document.getElementsByClassName('miClase');
var parrafos = document.getElementsByTagName('p');
var elemento = document.querySelector('#miId');
var elementos = document.querySelectorAll('.miClase');
document.title = 'Nuevo Título';
var body = document.body;
document.cookie = "nombre=valor";
var html = document.documentElement;
var head = document.head;
```

document.forms

La propiedad `document.forms` en JavaScript es una colección que representa todos los formularios (`<form>`) en un documento HTML. Esta propiedad te permite acceder a los formularios de una página web y manipularlos dinámicamente. `document.forms` devuelve un objeto `HTMLCollection`, que es similar a un array y contiene referencias a los formularios en orden de aparición en el documento HTML.

Aquí hay algunos ejemplos de cómo puedes utilizar `document.forms`:

1. Acceder a un formulario específico por índice o nombre:

Puedes acceder a un formulario específico en la colección utilizando su índice numérico o su atributo `name`:

```
var primerFormulario = document.forms[0]; // Accede al primer formulario en la página

var formularioPorNombre = document.forms["miFormulario"]; // Accede a un formulario por su nombre
```

2. Acceder a elementos de formulario dentro de un formulario:

Una vez que tienes una referencia a un formulario, puedes acceder a sus elementos de formulario, como campos de entrada, botones y selectores:

```
var miFormulario = document.forms["miFormulario"];

var campoTexto = miFormulario.elements["miCampoTexto"];

var botonEnviar = miFormulario.elements["enviarBoton"];
```

3. Iterar a través de todos los formularios en la página:

Puedes recorrer todos los formularios en la página utilizando un bucle `for`:

```
var todosLosFormularios = document.forms;

for (var i = 0; i < todosLosFormularios.length; i++) {
    console.log(todosLosFormularios[i].name);
}
```

4. Validar formularios:

`document.forms` es útil para recopilar datos de formularios y realizar validaciones antes de enviarlos al servidor. Puedes acceder a los valores de los campos de entrada y realizar comprobaciones personalizadas.

```
var formulario = document.forms["miFormulario"];

var campoNombre = formulario.elements["nombre"];
if (campoNombre.value === "") {
    alert("Por favor, ingrese su nombre.");
    return false; // Evita que el formulario se envíe si el campo está vacío
}
```

Youtic's
Lic. Gabriel Chaldú

Modificar elementos HTML

Aquí tienes ejemplos de cómo modificar atributos de algunos elementos HTML comunes utilizando JavaScript y el DOM:

1. Modificar el contenido de un elemento de texto (por ejemplo, un párrafo):

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de Modificación de Atributos</title>
</head>
<body>
    <p id="miParrafo">Este es un párrafo.</p>

    <script>
        // Modificar el contenido del párrafo
        var parrafo = document.getElementById('miParrafo');
        parrafo.textContent = 'Este es el nuevo contenido del párrafo.';
    </script>
</body>
</html>
```

2. Modificar el atributo "src" de una imagen:

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de Modificación de Atributos</title>
</head>
<body>
    <img id="miImagen" src="imagen1.jpg" alt="Imagen 1">

    <script>
        // Modificar el atributo "src" de la imagen
        var imagen = document.getElementById('miImagen');
        imagen.src = 'imagen2.jpg';
    </script>
</body>
</html>
```

3. Modificar el atributo "href" de un enlace:

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de Modificación de Atributos</title>
</head>
<body>
    <a id="miEnlace" href="https://www.ejemplo.com">Enlace de Ejemplo</a>

    <script>
        // Modificar el atributo "href" del enlace
        var enlace = document.getElementById('miEnlace');
        enlace.href = 'https://www.nuevovinculo.com';
    </script>
</body>
</html>
```

4. Modificar el atributo "class" de un elemento:

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de Modificación de Atributos</title>
    <style>
        .rojo {
            color: red;
        }
    </style>
</head>
<body>
    <p id="miParrafo" class="rojo">Este es un párrafo con clase.</p>

    <script>
        // Modificar el atributo "class" del párrafo
        var parrafo = document.getElementById('miParrafo');
        parrafo.className = 'azul'; // Cambia la clase a "azul"
    </script>
</body>
</html>
```

Estos ejemplos muestran cómo puedes utilizar JavaScript para modificar atributos de elementos HTML comunes, como el contenido de un párrafo, la fuente de una imagen, el enlace de un hipervínculo y la clase de un elemento. Al modificar estos atributos, puedes actualizar dinámicamente el contenido y el estilo de tu página web en respuesta a eventos o acciones del usuario.

Youtic's
Lic. Gabriel Chaldú

El "traversing" del DOM

El "traversing" del DOM (navegación en el DOM) es un término que se utiliza en programación web para referirse a la acción de recorrer y navegar por la estructura jerárquica de un documento HTML utilizando JavaScript. En otras palabras, se trata de moverse hacia arriba o hacia abajo en el árbol del DOM para acceder a elementos HTML específicos o sus relaciones.

El "traversing" del DOM es útil para acceder a elementos relacionados, como elementos secundarios, elementos hermanos, padres o hijos de un elemento específico. Esto es especialmente útil cuando deseas interactuar con elementos HTML específicos o realizar manipulaciones en función de la relación entre elementos en una página web.

Aquí hay algunos ejemplos de operaciones de "traversing" comunes en el DOM:

1. Acceder a elementos hijos:

Puedes usar propiedades como `childNodes`, `firstChild`, `lastChild`, o métodos como `querySelector` para acceder a los elementos secundarios de un elemento padre.

```
var elementoPadre = document.getElementById('miDiv');
var primerHijo = elementoPadre.firstChild;
```

2. Acceder a elementos hermanos:

Puedes utilizar propiedades como `nextSibling` y `previousSibling` para acceder a elementos hermanos (elementos que comparten el mismo padre).

```
var elementoActual = document.getElementById('miElemento');
var siguienteHermano = elementoActual.nextSibling;
var hermanoAnterior = elementoActual.previousSibling;
```

3. Acceder al padre de un elemento:

Puedes utilizar la propiedad `parentNode` para acceder al elemento padre de un elemento específico.

```
var elementoHijo = document.getElementById('miHijo');
var padreDelHijo = elementoHijo.parentNode;
```

4. Recorrer elementos en una lista:

Puedes recorrer una lista de elementos (por ejemplo, elementos de una lista `<ul>`) utilizando un bucle `for` y accediendo a cada elemento por su índice.

```
var lista = document.getElementsByTagName('li');
for (var i = 0; i < lista.length; i++) {
    console.log(lista[i].textContent);
}
```

5. Acceder a elementos a través de relaciones de clase o etiquetas:

Puedes utilizar métodos como `getElementsByClassName` y `getElementsByTagName` para acceder a elementos en función de sus clases o etiquetas.

```
var elementosConClase = document.getElementsByClassName('miClase');
var elementosDiv = document.getElementsByTagName('div');
```

El "traversing" del DOM es esencial para interactuar con elementos específicos en una página web y realizar tareas de manipulación dinámica, como agregar, eliminar o modificar contenido en función de la estructura del documento HTML. Estas operaciones son fundamentales en el desarrollo web para crear experiencias interactivas y dinámicas para los usuarios.

`appendChild` y `removeChild`

Las funciones `appendChild` y `removeChild` son métodos del DOM (Modelo de Objetos del Documento) en JavaScript que se utilizan para agregar y eliminar elementos HTML en el árbol de un documento. Estos métodos son especialmente útiles para la manipulación dinámica de elementos en una página web.

1. `appendChild`:

El método `appendChild` se utiliza para agregar un nodo (elemento) como hijo de otro nodo. Generalmente, se usa para agregar un elemento HTML a otro, como agregar un párrafo a un div o un elemento de lista a una lista desordenada (ul).

Ejemplo de uso:

```
// Crear un nuevo párrafo
var nuevoParrafo = document.createElement('p');
nuevoParrafo.textContent = 'Este es un nuevo párrafo.';

// Obtener el elemento div existente
var contenedorDiv = document.getElementById('miDiv');

// Agregar el párrafo como hijo del div
contenedorDiv.appendChild(nuevoParrafo);
```

En este ejemplo, se crea un nuevo párrafo, se obtiene un div existente por su id y se agrega el párrafo como un hijo del div utilizando `appendChild`.

2. `removeChild`:

El método `removeChild` se utiliza para eliminar un nodo hijo de un nodo padre. Se pasa como argumento el nodo hijo que se desea eliminar.

Ejemplo de uso:

```
 // Obtener el elemento que se desea eliminar
var elementoEliminar = document.getElementById('elementoAEliminar');

// Obtener el padre del elemento
var padre = elementoEliminar.parentNode;

// Eliminar el elemento hijo del padre
padre.removeChild(elementoEliminar);
```

En este ejemplo, se obtiene un elemento por su id, se accede a su padre utilizando `parentNode`, y luego se elimina el elemento hijo del padre utilizando `removeChild`.

Estos métodos son fundamentales para la manipulación del DOM en tiempo real y son comunes en la creación y modificación de contenido en una página web dinámica. Pueden ser utilizados para agregar, mover o eliminar elementos en función de la interacción del usuario o de otras condiciones de la aplicación web.

onClick

El evento `onClick` es un evento de JavaScript que se dispara cuando se hace clic en un elemento HTML. Para usar este evento en un elemento seleccionado, primero debes seleccionar el elemento en JavaScript y luego agregar un manejador de eventos `onClick` a ese elemento.

Aquí hay un ejemplo de cómo hacerlo:

Supongamos que tienes un botón en tu HTML con un `id` "miBoton" y deseas que ocurra algo cuando se hace clic en ese botón:

html

```
<button id="miBoton">Haz clic en mí</button>
```

Puedes seleccionar el botón en JavaScript y agregar un manejador de eventos `onClick` de la siguiente manera:

```
// Seleccionar el botón por su id
var boton = document.getElementById('miBoton');
// Agregar un manejador de eventos onClick
boton.onclick = function() {
    // Aquí puedes realizar acciones cuando se hace clic en el botón
    alert('¡Hiciste clic en el botón!');
};
```

En este ejemplo, seleccionamos el botón por su `id` usando `getElementById` y luego agregamos un manejador de eventos `onClick` al botón. Cuando el botón se hace clic, la función anónima dentro del manejador de eventos se ejecutará, y en este caso, muestra una alerta.

También puedes usar la forma más moderna de agregar manejadores de eventos utilizando `addEventListener`:

```
// Seleccionar el botón por su id
var boton = document.getElementById('miBoton');
// Agregar un manejador de eventos onClick usando addEventListener
boton.addEventListener('click', function() {
    // Aquí puedes realizar acciones cuando se hace clic en el botón
    alert('¡Hiciste clic en el botón!');
});
```

Esta forma permite agregar múltiples manejadores de eventos a un elemento sin sobrescribir los existentes y es más flexible en situaciones donde necesitas gestionar varios eventos en el mismo elemento.