UD3 - DOM

DAW2 - DEWC

Document Object Model

El DOM es una interfaz de programación en los navegadores para manipular la página.

El navegador lee el código HTML y construye la estructura jerárquica del árbol.

A través de JavaScript podemos manipular dinámicamente su contenido.

Objetos principales:

- window => this – ámbito principal
- document => página HTML
- Browser Object Model (BOM) => APIs adicionales

BOM

navigator
- navigator.geolocation

screen

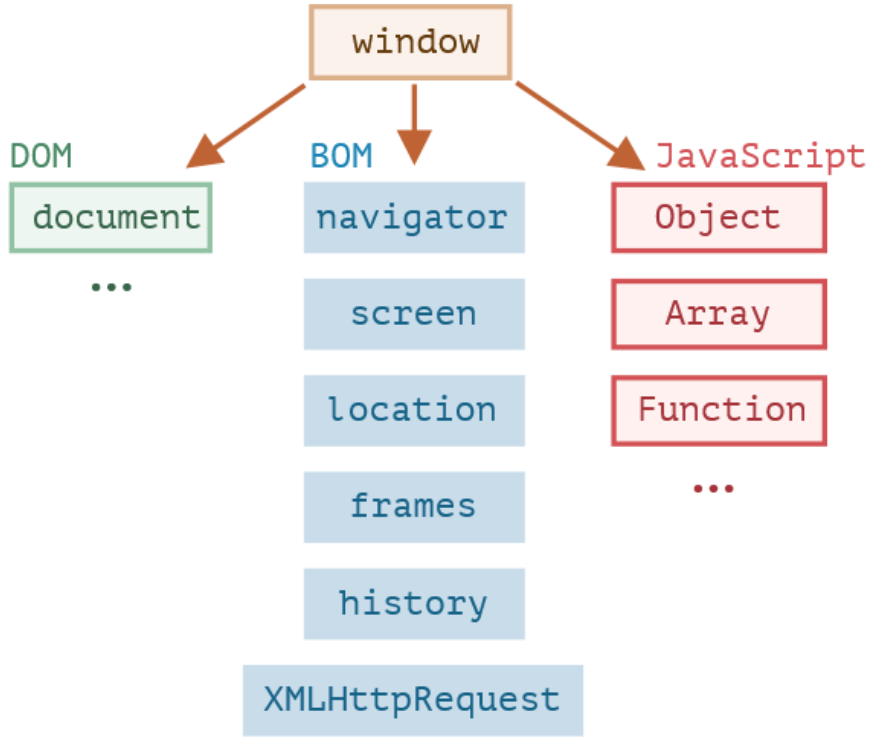
location
- location.href

history
- history.go()

localStorage

sessionStorage

etc...



Tipos de nodos

ELEMENT_NODE: ,<h1>,<p>,etc…

ATTRIBUTE_NODE: class, id, etc…

TEXT_NODE: texto plano

DOCUMENT_NODE: document

DOCUMENT_FRAGMENT_NODE: contenedor independiente

Colecciones

- HTMLCollection => colección “viva”
- NodeList => consultas de selección estáticas, propiedad childNodes “viva”

Navegación entre nodos

Propiedades para navegar

- parentNode
- childNodes (todos), children (sólo elementHTML)
- firstChild y lastChild
- firstElementChild y lastElementChild
- nextSibling y previousSibling
- nextElementSibling y previousSibling

Búsqueda de nodos

Podemos buscar a partir de “**document**” o de un “**elementHTML**”

Métodos

- getElementById()
- getElementsByClassName()
- getElementsByTagName()
- querySelector()
- querySelectorAll()
- closest()

Ejemplo de búsqueda

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <title>Búsqueda DOM</title>
</head>
<body>
    <div class="contenedor">
        <span class="texto">Hola</span>
    </div>
</body>
</html>
```

```javascript
// Obtener el elemento por su clase
const span = document.querySelector('.texto');

// Buscar el contenedor más cercano hacia arriba en el árbol
const contenedor = span.closest('.contenedor');
console.log(contenedor); // <div class="contenedor">
```

Creación de nodos

Crear un nodo no significa que se incluya en el árbol automáticamente

Principales métodos

- document.createElement(‘etiqueta HTML’)
- document.createTextNode(‘texto plano’)
- document.createComment(‘texto comentario’)
- document.createDocumentFragment()

```
// Crear un nuevo elemento <p>
const nuevoParrafo = document.createElement('p');
nuevoParrafo.textContent = 'Este es un párrafo nuevo.';

// Crear un nodo de texto
const texto = document.createTextNode('Este es un texto sin etiqueta.');
document.body.appendChild(texto);

// Crear un comentario
const comentario = document.createComment('Este es un comentario.');
document.body.appendChild(comentario);
```

Manipulación de nodos

contenedor.appendChild(nuevoNodo)

contenedor.insertBefore(nuevoNodo, referencia)

contenedor.prepend(nuevoNodo)

contenedor.replaceChild(nuevoNodo, viejoNodo)

contenedor.removeChild(borrarNodo)

```html
<body>
    <div id="contenedor">
        <p class="referencia">JavaScript mola</p>
    </div>
</body>
```

```javascript
const nuevoParrafo = document.createElement('h1');
nuevoParrafo.textContent = 'Mensaje importante';

// Seleccionar el contenedor
const contenedor = document.getElementById('contenedor');

// Insertar el nuevo título antes del párrafo de referencia
const referencia = document.querySelector('.referencia');
contenedor.insertBefore(nuevoParrafo, referencia);
```

Manipulación de atributos

Sólo los elementos HTML cuentan con atributos

Propiedad “attributes”

Métodos

- setAttribute(‘nombre_atr’, ‘valor’)
- getAttribute(‘nombre_atr’)
- nuevoParrafo.removeAttribute('class')

Manejo de clases - class

La propiedad en html “class” se convierte la colección “classList” en JS

Métodos

- elemento.classList.add('clase')
- elemento.classList.remove('clase')
- elemento.classList.toggle('clase')
- elemento.classList.contains('clase')
- elemento.classList.replace('claseActual', 'claseNueva')

OJO: “className” es una cadena con las clases en crudo

```
boton.addEventListener('click', () => {
    boton.classList.toggle('activo');
});
```

Atributos personalizados – data-*

Las propiedades “data-*” en html se convierte la colección “**dataset**” en JS

Los nombres se transforman. data-entidad=“alumno” => dataset.entidad=‘alumno’

Los guiones en nombres compuestos se transforman en notación “camelCase”

Métodos

- element.dataset.nombreAtributo
- element.dataset.nombreAtributo = valor
- element.removeAttribute(‘nombreAtributo’)

Asignación de Eventos

Atributo HTML (onclick).

- Directamente en el HTML, no es lo ideal.

Asignación directa (element.onclick)

- En un objeto JS a través de su atributo “onclick”
- SOBRESCRIBE OTROS MANEJADORES

Método “addEventListener()”

- Permite definir más de un manejador.
- Facilita la eliminación de manejadores específicos.

Visto lo anterior que funcione eliminar con **“removeEventListener()”** depende de nuestro código

Propiedades objeto event

Comunes

- evento.target: Elemento que origina el evento.
- evento.currentTarget: Elemento en el que se asigno el manejador (contenedor del disparador)
- evento.type: Tipo de evento (lo que va a la derecha del “on”)

Recuerda que en función del tipo de evento es posible que aparezcan propiedades especificas, no es lo mismo el teclado, que el ratón, que la ventana…

Delegación de eventos

Podemos asignar un evento a un contenedor en vez de asignar el evento a cada elemento hijo.

Esta técnica mejora el rendimiento y simplifica el código.

```html
<ul id="lista">
    <li>Elemento 1</li>
    <li>Elemento 2</li>
    <li>Elemento 3</li>
</ul>
```

```html
<script>
    const lista = document.getElementById('lista');
    lista.addEventListener('click', (event) => {
        if (event.target.tagName === 'LI') {
            alert(`Has clicado en: ${event.target.textContent}`);
        }
    });
</script>
```

Flujo de eventos

Podemos decidir como se notifican los eventos a todos los elementos relacionados con él.

Capturing (fase de captura)

- El evento se propaga desde la ventana hacia el origen del evento.
- Se activa con el tercer parámetro de addEventListener a true.

Bubbling (fase de burbuja)

- El evento se propaga desde el origen del evento hacia la ventana.
- Este es el comportamiento por defecto.

Gestión del evento.

- event.preventDefault()
- event.stopPropagation()
- event.stopInmediatePropagation()

Conclusiones

Hace falta comprender la estructura del DOM.

Necesitamos comprender los tipos de nodos y los eventos para manipularlos.

Importancia de las propiedades **classList** y **dataset**

Diferenciar las técnicas de asignación de eventos

Asignación por delegación

Obtener información del objeto evento => ¡¡¡cuidadín!!! no he desarrollado los eventos específicos en la presentación.

Preguntas