Javascript - DOM y Eventos

Programación Web I

Comisión Miércoles Noche

Prof. Damián Spizzirri

Prof. Esteban David Alaníz

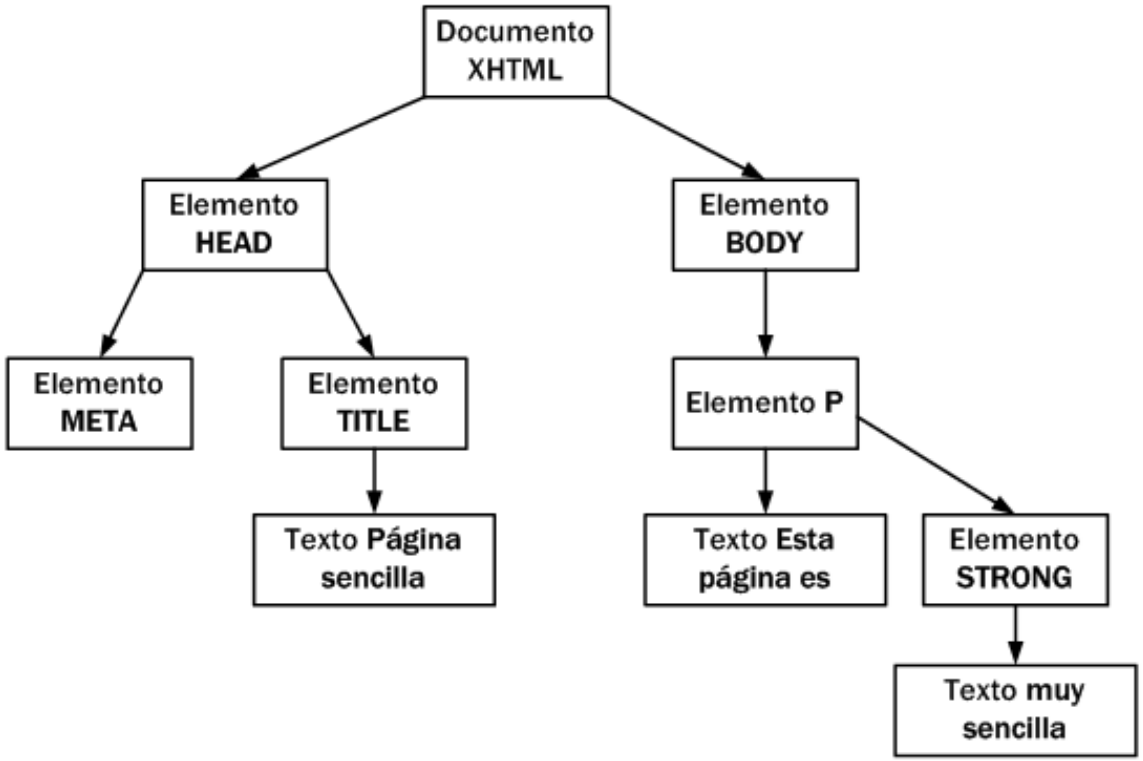
Comisión Viernes Mañana

Prof. Gabriel Panik

Prof. Javier Barraza

Document Object Model (DOM)

```
<!DOCTYPE html>
<html>
<head>
          <meta charset=”UTF-8”>
     <title>Página sencilla</title>
</head>
<body>
     <p>Esta página es <strong> muy sencilla </strong></p>
</body>
</html>
```



Tipos de Nodos

DOM define 12 tipos de nodos, aunque las páginas HTML habituales se pueden manipular manejando solamente cuatro o cinco tipos de nodos:

1. **Document**
   Nodo raíz del que derivan todos los demás nodos del árbol.

2. **Element**
   Representa cada una de las etiquetas HTML. Se trata del único nodo que puede contener atributos y el único del que pueden derivar otros nodos.

3. **Attr**
   Se define un nodo de este tipo para representar cada uno de los atributos de las etiquetas HTML, es decir, uno por cada par atributo=valor.

4. **Text**
   Nodo que contiene el texto encerrado por una etiqueta HTML.

5. **Comment**
   Representa los comentarios incluidos en la página HTML.

Los otros tipos de nodos existentes que no se van a considerar
son DocumentType, CDataSection, DocumentFragment,Entity, EntityReference, ProcessingInstruction y Notation.

DOM - Tipo de Acceso a Nodos

Tipo de Acceso a los nodos

Una vez construido automáticamente el árbol **completo** de nodos DOM, ya es posible utilizar las funciones DOM para acceder de forma directa a cualquier nodo del árbol. Es decir, su consulta, modificación y su eliminación solamente es posible después de que la página HTML se cargue por completo.

DOM proporciona dos métodos alternativos para acceder a un nodo específico:

1. Acceso a través de sus nodos padre
2. Acceso directo.

Las funciones que proporciona DOM para acceder a un nodo a través de sus nodos padre consisten en acceder al nodo raíz de la página y después a sus nodos hijos y a los nodos hijos de esos hijos y así sucesivamente hasta el último nodo de la rama terminada por el nodo buscado. Sin embargo, cuando se quiere acceder a un nodo específico, es mucho más rápido acceder directamente a ese nodo y no llegar hasta él descendiendo a través de todos sus nodos padre.

DOM - Acceso Directo a los nodos

1) getElementsByTagName(nombreEtiqueta)
La función obtiene todos los elementos de la página HTML cuya etiqueta sea igual que el parámetro que se le pasa a la función.

```
var parrafos = document.getElementsByTagName("p");
```

El valor devuelto es un array de nodos DOM, no un array de cadenas de texto o un array de objetos normales

2) getElementsByName()
Se buscan los elementos cuyo atributo “name” sea igual al parámetro proporcionado.

DOM - Acceso Directo a los nodos

Acceso directo a los nodos

3) getElementById()

Devuelve el elemento HTML cuyo atributo id coincide con el parámetro indicado en la función. Como el atributo id debe ser único para cada elemento de una misma página, la función devuelve únicamente el nodo deseado.

```
var nombre = document.getElementById("nombre");
```

Es tan importante y tan utilizada en todas las aplicaciones web, que casi todos los ejemplos y ejercicios que siguen la utilizan constantemente.

Query Selector

```
//Selector Query Selector, trae el primer elemento
let selector=document.querySelector(".texto");
```

https://developer.mozilla.org/es/docs/Web/API/Document/querySelector

```
//Query selector all, trae todos los nodos
let selectores=document.querySelectorAll(".texto");
```

https://developer.mozilla.org/es/docs/Web/API/Document/querySelectorAll

Query Selector

```
//Selector Query Selector, trae el primer elemento
let selector=document.querySelector(".texto");
```

Notas

- Devuelve null si no se encuentran coincidencias, de lo contrario, retorna el primer elemento encontrado.
- Si el selector coincide con un ID y este ID es usado erróneamente varias veces en el documento, devuelve el primer elemento encontrado.
- Lanza una excepción de tipo SYNTAX_ERR si el grupo de selectores especificado no es válido.
- querySelector() se introdujo en la API Selectors.
- La cadena de caracteres que se pasa como argumento a querySelector debe seguir la sintaxis CSS.
- Las Pseudo-clases CSS nunca devolverán elementos, tal y como está especificado en la API Selectors.
- Para que coincidan ID's o selectores que no siguen la sintáxis CSS (usando inapropiadamente dos puntos o un espacio por ejemplo), se debe 'escapar' el carácter con una barra invertida (\). Como la barra invertida es un carácter de 'escape' en JavaScript, si estás indicando una cadena de caracteres literal, debes 'escaparla' dos veces (una para la cadena de caracteres JavaScript y otra para el querySelector):

https://developer.mozilla.org/es/docs/Web/API/Document/querySelector

DOM - Creación de Nodos

Creación de elementos HTML simples

Por este motivo, crear y añadir a la página un nuevo elemento HTML sencillo consta de cuatro pasos diferentes:

1. Creación de un nodo de tipo Element que represente al elemento.
2. Creación de un nodo de tipo Text que represente el contenido del elemento.
3. Añadir el nodo Text como nodo hijo del nodo Element.
4. Añadir el nodo Element a la página, en forma de nodo hijo del nodo correspondiente al lugar en el que se quiere insertar el elemento.

El proceso de creación de nuevos nodos puede llegar a ser tedioso, ya que implica la utilización de tres funciones DOM:

1. createElement(etiqueta): crea un nodo de tipo Element que representa al elemento HTML cuya etiqueta se pasa como parámetro.
2. createTextNode(contenido): crea un nodo de tipo Text que almacena el contenido textual de los elementos HTML.
3. nodoPadre.appendChild(nodoHijo): añade un nodo como hijo de otro nodo. Se debe utilizar al menos dos veces con los nodos habituales: en primer lugar se añade el nodo Text como hijo del nodo Element y a continuación se añade el nodo Element como hijo de algún nodo de la página.

DOM - Creación y Eliminación de Nodos

```
// Crear nodo de tipo Element
var parrafo = document.createElement("p");

// Crear nodo de tipo Text
var contenido = document.createTextNode("Hola Mundo!");

// Añadir el nodo Text como hijo del nodo Element
parrafo.appendChild(contenido);

// Añadir el nodo Element como hijo de la pagina
document.body.appendChild(parrafo);
```

DOM - Creación y Eliminación de Nodos

Creación y Eliminación de nodos

Eliminación

Solamente es necesario utilizar la función removeChild().

```
var parrafo = document.getElementById("provisional");
parrafo.parentNode.removeChild(parrafo);


<p id="provisional">...</p>
```

La función removeChild() requiere como parámetro el nodo que se va a eliminar. Además, esta función debe ser invocada desde el elemento padre de ese nodo que se quiere eliminar.

Así, para eliminar un nodo de una página HTML se invoca a la función removeChild() desde el valor parentNode del nodo que se quiere eliminar. Cuando se elimina un nodo, también se eliminan automáticamente todos los nodos hijos que tenga, por lo que no es necesario borrar manualmente cada nodo hijo.

DOM - Creación y Eliminación de Nodos

Acceso directo a los atributos HTML

Mediante DOM, es posible acceder de forma sencilla a todos los atributos HTML y todas las propiedades CSS de cualquier elemento de la página.

Los atributos HTML de los elementos de la página se transforman automáticamente en propiedades de los nodos. Para acceder a su valor, simplemente se indica el nombre del atributo HTML detrás del nombre del nodo.

```
var enlace = document.getElementById("enlace");

alert(enlace.href); // muestra http://www...com

<a id="enlace" href="http://www...com">Enlace</a>
```

Las propiedades CSS no son tan fáciles de obtener como los atributos HTML. Para obtener el valor de cualquier propiedad CSS del nodo, se debe utilizar el atributo style.

```
var imagen = document.getElementById("imagen");

alert(imagen.style.margin);

<img id="imagen" style="margin:0; border:0;" src="logo.png" />
```

DOM - Acceso directo a los atributos HTML

Acceso directo a los atributos HTML

La transformación del nombre de las propiedades CSS compuestas consiste en eliminar todos los guiones medios (-) y escribir en mayúscula la letra siguiente a cada guión medio.

- font-weight se transforma en fontWeight
- line-height se transforma en lineHeight
- border-top-style se transforma en borderTopStyle
- list-style-image se transforma en listStyleImage

El único atributo HTML que no tiene el mismo nombre en HTML y en las propiedades DOM es el atributo “class”. Como la palabra class está reservada por JavaScript, no es posible utilizarla para acceder al atributo class del elemento HTML. En su lugar, DOM utiliza el nombre className para acceder al atributo class de HTML

DOM - Acceso directo a los atributos HTML

https://www.w3schools.com/jsref/met_node_clonenode.asp

https://www.w3schools.com/jsref/met_node_removechild.asp

EVENTOS de Teclado

EVENTO	DESCRIPCIÓN	ELEMENTOS QUE PUEDEN TENER ESTE EVENTO
keyup	dejar de teclear	TODOS
keypress	presionar una tecla	TODOS
keydown	presionar una tecla sin soltar	TODOS

ACCEDER A UN ELEMENTO

```
<!--HTML-->
<h1 id="titulo">

//javascript
document.getElementById("titulo")

document.querySelector("#titulo")
```

addEventListener

Le agrega un manejador de eventos a un elemento. Es un escuchador.

Sintaxis:

```
element.addEventListener(event, function);
```

Ejemplo:

```
const boton= document.getElementById("boton");

boton.addEventListener("click", function(){
   ...acciones
});
```

Eventos

```
let campoNombre=
document.getElementById("nombre");

campoNombre.addEventListener("keyup",(e)=>{
    //acciones
});
```

Value

```
<!--HTML-->
<input type="text" id="nombre"
name="nombre">

//javascript
let nombre=
document.getElementById("nombre").value;
```

INNERHTML

```
//javascript
document.getElementById("mensaje")
.innerHTML="texto";
```

EVENTOS de FORMULARIOS

EVENTO	DESCRIPCIÓN	ELEMENTOS QUE PUEDEN TENER ESTE EVENTO
focus	seleccionar un elemento	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`
blur	deseleccionar un elemento	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`
submit	enviar un formulario	`<form>`
reset	al resetear un formulario	`<form>`
change	deseleccionar un elemento que fue modificado	`<input>`, `<select>`, `<textarea>`

Modificar Estilos

```
//javascript
  document.getElementById("titulo")
  .style.propiedadCss= "valor";

  document.getElementById("titulo")
  .style.display= "none";

  document.getElementById("titulo")
  .style.backgroundColor= "#000";
```

Trabajo con clases

```
//agrega una clase
element.classList.add("clase");

//elimina una clase
element.classList.remove("clase");

//cambia entre un estado y otro
element.classList.toggle("clase");
```

Trabajo con clases

```
//devuelve si contiene
element.classList.contain("clase");
```

https://developer.mozilla.org/en-US/docs/Web/API/Element/classList

```
//pisa las clases por esta clase
element.className="clase";
```

EVENTOS DE PÁGINA

EVENTO	DESCRIPCIÓN	ELEMENTOS QUE PUEDEN TENER ESTE EVENTO
load	Al cargar la página	`<body>`
unload	Al abandonar la página	`<body>`
resize	al achicar o agrandar la ventana del navegador	`<body>`

Eventos de Mouse

EVENTO	DESCRIPCIÓN	ELEMENTOS QUE PUEDEN TENER ESTE EVENTO
click	al hacer click	TODOS
dblclick	al hacer doble click	TODOS
mouseover	al pasar el mouse	TODOS
mousedown	mientras tengo presionado el mouse	TODOS
mouseup	cuando suelto el mouse	TODOS
mousemove	cuando muevo el mouse	TODOS

Eventos de Mouse

```
const btn=
document.querySelector("#btn");

btn.addEventListener("click",
()=>{
        //acciones a ejecutar
});

<button id="btn"></button>
```

Expresiones Regulares

```
expresionRegular.test(valorAEvaluar);
```

```
(regex).test(cadenaAEvaluar)
```

Session Storage

```
Guardar:
localStorage.setItem
('nombreVariable',valor);

Obtener:
localStorage.getItem
('nombreVariable');
```

Session Storage

borrar:

```
sessionStorage.removeItem("item")
```

Array de Objetos JSON

```
const productos = [
        {
         nombre: "Jabon",
         cantidad: 2,
         precio: 165
    },
    {
         nombre: "Lavandina",
         cantidad: 1,
         precio: 50
    }
];
```

Lleva coma si no es el último objeto. Cada objeto va entre {} y cada atributo de objeto lleva el formato atributo: valor

Agregar objeto a array

```
const producto1 = {
    nombre: "Detergente",
    cantidad: 3,
    precio: 130
}

productos.push(producto1);
// agrega un objeto al array
```

Lleva coma si no es el último atributo. El formato es atributo: valor

Convertir a string y parsear

LocalStorage, almacena datos de tipo string.
Si queremos almacenar un json, tenemos que convertirlo a string al guardarlo en localStorage y parsearlo para recuperarlo como JSON

```
sessionStorage.setItem("productos",
JSON.stringify(productos));

const productosObtenidos =
JSON.parse(sessionStorage.getItem("productos"));
```

Filtrar Objetos Array.prototype.filter()

```
const buscador = document.querySelector("#buscador");

buscador.addEventListener("keyup",()=>{

    const consulta=buscador.value;

    const res = cursos.filter(curso =>curso.titulo.indexOf(consulta)>-1);

    console.log(res);

});
```

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

Intervalos

Intervalo, ejecutar unas instrucciones cada x cantidad de tiempo.

```
setInterval(función,tiempo en milisegundos);

setInterval(function(){
   funcion();
},1000);
```

Frenar un intervalo

```
clearInterval(nombreIntervalo);
```

Tiempo definido

```
setTimeout(function(){
  funcion()
},1000);
```

Promesas

Un objeto que representa la terminación o el fracaso de una operación asíncrona.

Permite que ciertas operaciones se desencadenen después de que una operación anterior haya terminado.

Tiene su propio manejo de errores.

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Using_promises

Promesas

```
new Promise((resolver, rechazar) => {
    console.log('Inicial');

    resolver();
})
.then(() => {
    throw new Error('Algo falló');

    console.log('Haz esto');
})
.catch(() => {
    console.log('Haz aquello');
})
.then(() => {
    console.log('Haz esto sin que importe lo que sucedió
antes');
});
```