Programador JavaScript Avanzado

Unidad 2: Eventos y Formularios

Contactos: tecnologia@elearning-total.com
Web: www.elearning-total.com

Indice

Unidad 2: Eventos y Formularios

Objetivos

Que el alumno logre:

- Conocer e implementar la Propagación de Eventos en JS.

Propagación de eventos

La propagación de eventos en JavaScript se refiere a cómo se transmiten los eventos desde el elemento que lo desencadena hasta los elementos secundarios dentro de la estructura del DOM (Document Object Model).

Cuando se produce un evento en un elemento, ese evento puede propagarse hacia arriba y hacia abajo en la jerarquía de elementos del DOM.

Existen dos tipos de propagación de eventos: la **propagación de eventos ascendente** (bubbling) y la **propagación de eventos descendente** (capturing). Estos dos tipos de propagación ocurren en diferentes fases del flujo del evento.

Propagación de eventos ascendente (bubbling)

En la propagación de eventos ascendente, el evento se dispara en el elemento objetivo y luego se propaga hacia los elementos padre en la jerarquía del DOM. Es decir, el evento se maneja primero en el elemento objetivo y luego en sus elementos padre.

Por ejemplo, si tenemos una estructura de elementos anidados como este:

```
<div id="nivel1">
  <div id="nivel2">
    <div id="nivel3">
    </div>
  </div>
</div>
```

Y tenemos un controlador de eventos en el elemento "nivel3":

```
const nivel3 = document.getElementById('nivel3')

nivel3.addEventListener('click', function(event) {
  console.log('Evento clic en Nivel 3')
});
```

Al hacer clic en el elemento "nivel3", el evento se propaga hacia arriba en la jerarquía del DOM y se ejecutan los controladores de eventos en los elementos "nivel2" y "nivel1" en ese orden, si están definidos.

Propagación de eventos descendente (capturing)

En la propagación de eventos descendente, el evento se dispara en el elemento padre más externo y luego se propaga hacia los elementos secundarios en la jerarquía del DOM. Es decir, el evento se maneja primero en el elemento padre y luego en los elementos secundarios.

Para capturar la fase de propagación descendente, puedes usar el método addEventListener con el tercer parámetro establecido en true:

```
const nivel1 = document.getElementById('nivel1')

nivel1.addEventListener('click', function(event) {
  console.log('Evento clic en nivel1')
}, true)
```

En este caso, cuando se hace clic en el elemento "nivel3", el evento se ejecuta primero en el controlador de eventos del elemento "nivel1" y luego en el controlador de eventos del elemento "nivel2" y "nivel3", si están definidos.

Es importante tener en cuenta que por defecto, la propagación de eventos en JavaScript sigue el modelo de propagación de eventos ascendente (bubbling). Puedes detener la propagación de eventos en cualquier momento utilizando el método stopPropagation() del objeto de evento.

```
const nivel3 = document.getElementById('nivel3')

nivel3.addEventListener('click', function(event) {
  event.stopPropagation()
  console.log('Evento clic en Nivel 3')
})
```

Esto evitará que el evento se propague más allá del elemento "nivel3".

Más sobre la Propagación de Eventos.

Todos los callbacks que registramos tienden a propagarse a través del DOM o el BOM o donde estemos registrando el evento. En muchos casos no vamos a querer que esto suceda porque puede generar comportamientos erráticos dentro de nuestro programa, como por ejemplo, ejecutando callbacks de una manera que no teníamos prevista.

También existen casos en donde la Propagación de Eventos es beneficiosa para nuestro programa, por ejemplo, en el caso que estemos trabajando con elementos dinámicos. Es decir, elementos que hayan sido escritos directamente en nuestro HTML, sino que los generemos de forma dinámica con JavaScript.

Vamos a crear un elemento botón en nuestro HTML:

```
<button id="estatico">Botón creado en HTML</button>
```

Desde nuestro archivo .js, le vamos a asignar a ese botón un evento que al hacer clic, cree un nuevo botón y agregarlo a nuestro HTML:

```
let btnHTML = document.querySelector("#estatico")

estatico.addEventListener("click", function(){
    let btnJS = document.createElement('button')
    btnJS.innerText = 'Botón creado desde JS'
    document.body.appendChild(btnJS)
})
```

¿Qué pasaría si quisiéramos agregar un evento a ese nuevo botón?

Para esto, vamos a asignarle un id:

```
let btnHTML = document.querySelector("#estatico")

estatico.addEventListener("click", function(){
    let btnJS = document.createElement('button')
    btnJS.innerText = 'Botón creado desde JS'
    btnJS.id = 'dinamico'
    document.body.appendChild(btnJS)
})
```

Y hacemos un proceso similar al que realizamos con el primer botón:

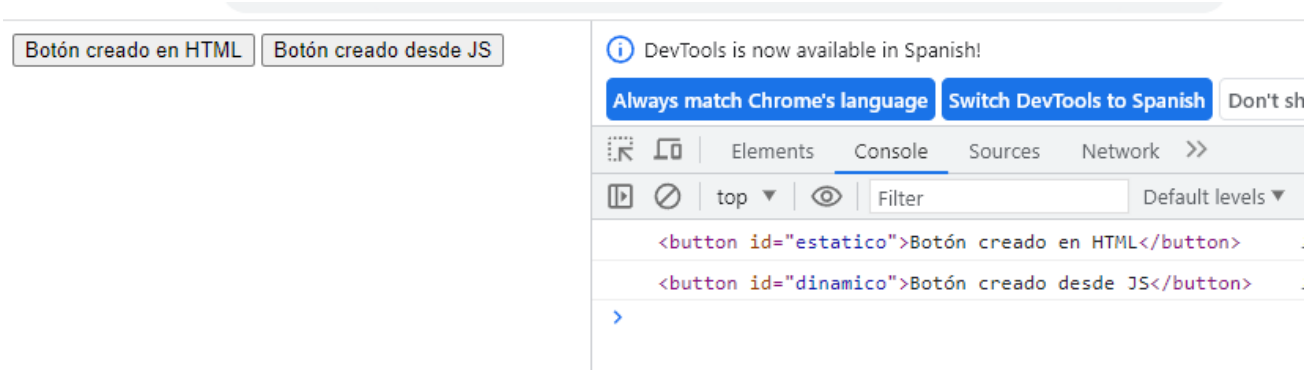
```
let btnJS = document.getElementById("dinamico")
   btnJS.addEventListener("click", function(){

})
```

Al ejecutar esto en nuestro navegador veremos que nos da un error en donde indica que la propiedad addEventListener no se puede leer de algo nulo. Al no tener el elemento en HTML, no existe en el DOM. Para esto, vamos a usar la propagación de eventos, asignando el evento que queremos registrar a un elemento que si exista en el DOM. Se lo vamos a asignar, por ejemplo, al objeto document. Vamos a acceder al objeto evento (e) :

```
document.addEventListener("click", function(e){
    console.log(e.target)
} )
```

Ahora, si hacemos clic en el botón dinámico, nos va a aparecer en consola ese botón.



El e.target es un objeto que referencia a elementos de HTML, también podemos acceder a sus propiedades, por ejemplo:

```
document.addEventListener("click", function(e){
    console.log(e.target.id)
    if(e.target.id== "dinamico") {
        console.log("Soy el botón dinamico creado con JS")
    }
} )
```

De esta forma podemos agregar la funcionalidad que queríamos, pero que no podíamos porque el elemento no estaba en el DOM.

La propagación de eventos nos permite, en Javascript, asignar eventos a elementos creados de forma dinámica.

Resumen

En esta Unidad…

Trabajamos con Formularios y Eventos

En la próxima Unidad…

Trabajaremos con AJAX