educación IT
CODE YOUR FUTURE

Bootcamp Java Developer

Fase 2 - Java Web Developer

Módulo 19

API EventTarget

API EventTarget

Es una interfaz de JavaScript para la gestión de eventos.

Sirve como interfaz estándar para cualquier objeto suceptible de disparar eventos (no solo eventos HTML).

Miembros de EventTarget

```
EventTarget.addEventListener(event:string, callback:function, propagation:boolean?)
```

Agrega una función al disparo de un evento.

```
EventTarget.removeEventListener(event:string, callback:function, options:object?, useCapture:boolean?)z
```

Remueve una función del disparo de un evento.

```
EventTarget.dispatchEvent(event:Event)
```

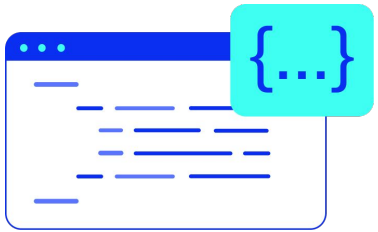
Dispara un evento, ejecutando todas las funciones asociadas.

Usar EventTarget con el DOM

En el ejemplo del siguiente slide, añadimos una función por cada vez que se agregue un nombre para el saludo.

De esta forma, al hacer click en el botón, se ejecutarán todas las funciones asociadas.

Esto es muy útil para desacoplar y dividir el programa.



```
// HTML
<div>
    <h2>Saludador</h2>
    <div>
        <input type="text" id="nombreActual" placeholder="Introduce tu nombre" />
        <button id="agregarNombre">Registrar nombre</button>
    </div>
    <button id="mostrador">Saludar</button>
</div>

// JS
const nombreActual = document.getElementById('nombreActual');
const agregarNombre = document.getElementById('agregarNombre');
const mostrador = document.getElementById('mostrador');

agregarNombre.addEventListener('click', function()
{
    mostrador.addEventListener('click', function()
    {
        const val = nombreActual.value;
        alert('Hola '+val);
    });
});

```

Crear mi propio EventTarget

En la pantalla siguiente, usamos **`EventTarget`** para dar al usuario del objeto *Carrito* la posibilidad de incluir código luego de que se agregue o quite un ítem.

Esto es especialmente útil en aplicaciones en las que el código fuente sea susceptible de ser cambiado, o en las que se deba dar al usuario la capacidad de modificar el código de una aplicación en un solo lugar.

```
const Carrito = {

    eventTarget: new EventTarget(),

    items: [],

    agregar: (item) => {
        Carrito.items.push(item);
        Carrito.eventTarget.dispatchEvent(new Event('item-agregado'))
    },

    quitar: (item) => {
        Carrito.items = Carrito.items.filter((actual) => item != actual);
        Carrito.eventTarget.dispatchEvent(new Event('item-borrado'))
    }

}


Carrito.eventTarget.addEventListener('item-agregado', function()
{
    alert('Nuevo item creado');
});

agregar(23);
```

Fases

La fase de un evento se define como **la forma en que ese evento se propaga a través del DOM**. Por defecto, todos los eventos vienen con una **fase predeterminada** pero podemos optar por cambiarlo de fase o bien cancelar cualquiera de las dos si no las necesitamos.

Las fases pueden ser:

- **`Bubbling:`** Ejecuta el **`handler`** del elemento **`target`** (el mismo que dispara el evento) y luego intenta disparar los **`handlers`** que haya registrados en sus nodos padres directos hasta llegar al elemento raíz, es decir, el HTML(DOM).
- **`Capturing:`** Es la fase inversa a **`Bubbling`**. Dispara los **`handlers`** registrados del mismo tipo de evento. Arranca por el elemento raíz y baja hasta el **`target`**.

Por defecto los eventos ya se disparan en la fase **Bubbling** pero podemos cambiar de fase usando el último parámetro opcional del **addEventListener**.

```
<button id=”btn”>Clickeame</button>
const btn = document.getElementById(‘btn’)
document.addEventListener("click",function(){
    console.log("click DOM")
},true)
btn.addEventListener("click",function(){
    console.log("click boton")
},true)
```

De esta manera, si se recarga la página y se hace clic en el botón, podremos notar que primero se dispara el clic registrado para el DOM, es decir la etiqueta HTML y luego el del `target`, es decir el elemento que disparó originalmente todo el evento de click: Nuestro botón.

A menos que no se estén usando las fases para construir alguna funcionalidad en particular, podremos siempre optar por cancelar la fase. Para ello, es necesario usar el objeto *Evento* construido por la misma arquitectura del lenguaje y enviado al **`handler`** del evento como primer parámetro. **`Event`** cuenta con dos funciones muy importantes:

- **`stopPropagation()`**: detiene la propagación.
- **`preventDefault()`**: cancela los comportamientos por defecto.

¡Sigamos trabajando!