CENTRO UNIVERSITARIO FELIPE CARRILLO PUERTO

ADA 8: Investigación eventos de JavaScript

8vo Cuatrimestre

Por

Canul Ceh Adriana Guadalupe

Trabajo presentado como cumplimiento de los requisitos de la materia de: Programación de Sitios web

Maestro: Erik Romero Lara

Mérida, Yucatán, 27 de marzo 2025

Eventos de JavaScript

¿Qué es un evento?

Un evento no es más que una "notificación" de que una característica relevante acaba de suceder. Algunos ejemplos de ello son:

- **Click:** Se ha hecho click de ratón sobre un elemento de la página.
- **Keydown:** Pulsación de una tecla específica del teclado.
- **Play:** Reproducción de un archivo de audio/video.
- **Wheel:** Scroll con la rueda del ratón sobre un elemento de la página.
- **Beforeprint:** El usuario ha activado la opción "Imprimir página".

El objetivo como desarrolladores es preparar nuestro código para que cuando ocurra un determinado evento, se lleve a cabo una funcionalidad asociada. Este proceso de llama escuchar eventos, de esta forma se puede preparar la página o aplicación para que cuando ocurran ciertos eventos, reaccionen a ellos.

¿Por qué utilizarlo?

Utilizar eventos en JavaScript tiene muchísimas ventajas, ya que no permite organizar nuestro código de una forma más lógica y clara, pero una de las principales ventas es que desacoplamos nuestro código. Básicamente nos permiten evitar dependencias entre partes de códigos. Con los eventos simplemente se escucha si ocurre un cierto evento, y si ocurre, ejecutamos una acción; sin embargo, no importa quién emite ese evento o donde sucede, lo importante es que ha sucedido el evento.

Formas de manejar eventos

La forma más fácil de trabajar con eventos JavaScript es mediante atributos de una etiqueta HTML. Sin embargo, aunque es la más sencilla, también es la menos recomendable, pero es bueno empezar desde ahí.

Ejemplo sin JavaScript:

```
<button>Saludar</button>
```

En el navegador aparecerá un botón con el texto de “Saludar”. Sin embargo, si le dan click, no realizará ninguna acción ni tendrá funcionamiento, esto ocurre porque por defecto el botón no tiene funcionalidad, por lo tanto, hay me dársela mediante JavaScript.

Ejemplo con JavaScript:

```
<button onClick="alert('Hello!')">Saludar</button>
```

Organizando la funcionalidad

El valor del atributo onClick llevará la funcionalidad en cuestión que queremos ejecutar cuando se produzca el evento. En nuestro ejemplo anterior, hemos colocado un alert(), pero lo habitual es que necesitemos ejecutar un fragmento de código más extenso, por lo que lo ideal sería meter todo ese código en una función, y en lugar del alert(), ejecutar dicha función:

```
<script>
  function doTask() {
    alert("Hello!");
  }
</script>
<button onClick="doTask()">Saludar</button>
```

Ahora el código está mejor organizado, sin embargo, no es muy habitual tener bloques <script> de código Javascript en nuestro HTML, sino que lo habitual suele ser externalizarlo en ficheros .js para dividir y organizar mejor nuestro código. Así quedaría el código:

```
<script src="tasks.js"></script>

<button onClick="doTask()">Saludar</button>
```

Ahora se tiene el código JavaScript en un fichero externo tasks.js, y en el atributo onClick llamamos a una función doTask(), que es la que tendremos que implementar en el tasks.js.

Sin embargo, ahora aparece un nuevo problema que quizás aún no sea muy evidente. En nuestro <button> estamos haciendo referencia a una función doTask() que, aparentemente, confiaremos en que se encuentra declarada dentro del fichero tasks.js.

Esto podría convertirse en un problema, si posteriormente, o dentro de cierto tiempo, nos encontramos modificando código en el fichero tasks.js y le cambiamos el nombre a la función doTask(), ya que podríamos olvidar que hay una llamada a una función Javascript en uno (o varios) ficheros .html. Se dice que el código HTML está acoplado al código Javascript (depende uno del otro). Por esta razón, suele ser buena práctica no incluir llamadas a funciones Javascript en nuestro código .html, sino que es mejor hacerlo desde el fichero externo .js, localizando los elementos del DOM con un .querySelector() o similar.

Eventos desde JavaScript

Existen varias formas de gestionar eventos sin necesidad de hacerlo desde nuestro .html mediante un atributo en línea. Principalmente, son tres formas:

- ✓ **addEventListener:** La forma preferida de trabajar con eventos desde Javascript es utilizar el evento .addEventListener() sobre nuestro elemento particular, de esta forma podremos escuchar el evento indicado de forma alternativa a usar el onClick:
    - Buscar el elemento en el DOM.
    - Escuchar el evento en el elemento.
    - Asociar la función al evento del listener.

```
<button>Saludar</button>

<script>
const button = document.querySelector("button");

button.addEventListener("click", function () {
  alert("Hello!");
});
</script>
```

Se observa que previamente, en lugar de añadir el atributo onClick al <button>, lo hemos localizado mediante querySelector(). Esto podríamos hacerlo mediante una clase, pero en este ejemplo lo hemos hecho directamente con el botón, para simplificar.

- ✓ **BOM (Browser Object Model):** es un mecanismo tradicional de los navegadores para acceder a ciertos elementos a través de propiedades Javascript. La idea es la misma que vimos al utilizar listeners, sólo que en esta ocasión haremos uso de una propiedad Javascript, a la que le asignaremos la función con el código asociado:

```
<button>Saludar</button>

<script>

const button = document.querySelector("button");

button.onclick = function() {

  alert("Hello!");

}

</script>
```

Lo que se hace es asignar una función con el código deseado en la propiedad .onclick del elemento <button>. Esta es una propiedad especial que tienen todos los elementos del DOM y que se ejecutan cuando sucede el evento en cuestión.

- ✓ DOM: Se utiliza “setAttribute”, lo que se hace es añadir un atributo onClick en el <button> de nuestro HTML, solo que lo hacemos a través de la API de Javascript:

```
<button>Saludar</button>

<script>
const button = document.querySelector("button");

const doTask = () => alert("Hello!");
button.setAttribute("onclick", "doTask()");
</script>
```

Si se llegará a inspeccionar el HTML del navegador, se puede notar que el método .setAttribute() lo que ha hecho es añadir el atributo onClick con el valor indicado en el HTML.

Metodo .addEvenListe

El método .addEventListener() nos permite empezar a escuchar un evento concreto, y en el caso de que ocurra, ejecutar la función asociada. De forma opcional, se le puede pasar un tercer parámetro con ciertas opciones.

- ✓ Arrow functions en .addEventListener(): Ejemplo

```
const button = document.querySelector("button");
button.addEventListener("click", function() {
  alert("Hello!");
});
```

En lugar de tener la función definida fuera, la utilizamos en el propio .addEventListener() y nos ahorramos ponerle un nombre, es decir, utilizamos una función anónima. Si prefieres utilizar las funciones flecha de Javascript, quedaría incluso más legible:

```
const button = document.querySelector("button");
button.addEventListener("click", () => {
  alert("Hello!");
});
```

De hecho, cuando tenemos un código muy corto, podemos incluso ahorrarnos las llaves en las arrow function y escribirlo todo en una sola línea

- ✓ Multiples listeners: El método .addEventListener() permite asociar múltiples funciones a un mismo evento, algo que, aunque no es imposible, es menos sencillo e intuitivo en las modalidades de gestionar eventos.
    - Action: muestra un mensaje de saludo.
    - Toggle: añade o quita el color rojo del botón.

```
<button>Saludar</button>

<style>
.red { background: red }
</style>

<script>
const button = document.querySelector("button");
const action = () => alert("Hello!");
const toggle = () => button.classList.toggle("red");

button.addEventListener("click", action);        // Hello message
button.addEventListener("click", toggle);        // Add/remove red CSS
</script>
```

Al pulsar el botón se efectuan ambas acciones (ya que hay dos listeners en escucha). Primero veremos el mensaje de alerta, y luego, el color rojo se añade o se quita.

Referencias

https://lenguajejs.com/eventos/control/objeto-event/

https://developer.mozilla.org/es/docs/Learn_web_development/Core/Scripting/Events