Programación con JavaScript I

Manipulación del DOM

Introducción

Los programas JavaScript del lado del cliente utilizan un modelo de programación asíncrono basado en eventos. En este estilo de programación, el navegador web genera un evento cada vez que le sucede algo interesante al documento o navegador o a algún elemento u objeto asociado a él.

Por ejemplo, el navegador web genera un evento cuando termina de cargar un documento, cuando el usuario mueve el mouse sobre un hipervínculo o cuando el usuario presiona una tecla. Si una aplicación de JavaScript se preocupa por un tipo particular de evento, puede registrar una o más funciones para invocarlas cuando ocurran eventos de ese tipo. Ten en cuenta que esto no es exclusivo de la programación web, todas las aplicaciones con interfaces gráficas de usuario están diseñadas de esta manera: se sientan a esperar que interactúen con ellas (es decir, esperan que ocurran eventos) y luego responden.

Comencemos esta experiencia educativa con algunas definiciones importantes que ayudan a explicar ese modelo de evento.

Explicación

Un evento en JavaScript, según lo define Kantor (2022), es una notificación de que algo "interesante" acaba de ocurrir. Los eventos, por tanto, demuestran una interacción entre las aplicaciones y el usuario. Dichas interacciones pueden ser muy variadas: cuando el usuario pulsa el mouse sobre algún elemento de la página o alguna tecla en específico, reproduce un archivo de audio, etcétera. Sin embargo, para que un evento se produzca, no siempre es necesario que intervenga el usuario, existen algunos eventos que se pueden ejecutar al cargar la página.

Un evento por sí mismo carece de utilidad hasta que los desarrolladores preparan el código de la página web para que, cuando ocurra algún evento, se ejecute el código con la funcionalidad indicada. Este código que se desarrolla para cada evento se conoce como **manejador de eventos** (event handlers) y en JavaScript existen varias formas de indicarlos:

a) Mediante atributos HTML
Esta es la forma más sencilla de trabajar con eventos en JavaScript, sin embargo, también es la menos profesional. Aquí el código se incluye directamente en el atributo del elemento HTML. Considera el caso en el que se desea que se dispare un evento al presionar un botón:

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

Entonces, cuando se presiona el botón, el código que se encuentra dentro del onClick se va a ejecutar. Un atributo HTML jamás va a ser un lugar adecuado para escribir mucho código JavaScript. Para solucionarlo, se recomienda crear una función y llamarla desde el onclick, por ejemplo:

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

Es importante mencionar que los atributos HTML no son case sensitive, es decir, no distinguen entre mayúsculas y minúsculas, por lo que podemos utilizar **onclick** u **onClick** indistintamente.

Explicación

Aunque lo anterior parece solucionar el problema, pueden presentarse dos inconvenientes: el primero es que no es recomendable incluir código JavaScript dentro del código HTML. Esto se puede resolver fácilmente separando el código JavaScript del HTML:

```
<script src="tareas.js"></script>
<button onClick="saludar()">Saludar</button>
```

Por otra parte, si el código se incluyera en un archivo por separado o si por alguna razón se le cambiase el nombre a la función que estamos invocando, se vería afectada la funcionalidad y no sería muy perceptible.

b Mediante propiedades JavaScript (DOM)

Por lo anterior, LenguajeJS (s.f.) sugiere como buena práctica, por un lado, no incluir llamadas a funciones JavaScript desde el html y, por otro lado, localizar los elementos del DOM con un .querySelector() desde un archivo .js externo. Utilizando el mismo ejemplo del saludo, con la propiedad JavaScript, el código se vería de la siguiente manera:

```
<button>Saludar</button>

<script>
const button = document.querySelector("button");
button.onclick = function() {
alert("Hola!");
}
</script>
```

De esta manera, en lugar de añadir el onclick al botón, se localiza mediante el querySelector() y luego se asigna una función con el código deseado en la propiedad .onClick del elemento.

Otra manera de hacerlo es utilizando el método .setAttribute(), en el cual se le asigna el atributo y el valor asignado. Con esta forma, el código queda de la siguiente manera:

Explicación

```
<button>Saludar</button>


<script>
const button = document.querySelector("button”);
const saludar = () => alert("Hola!");
button.setAttribute("onclick", "saludar()”);
</script>
```

Así se crea el html mediante la API del DOM de JavaScript.

c) Mediante addEventListener()

Esta es la forma más recomendable, debido a que con **addEventListener** se pueden añadir, eliminar y crear comportamientos especiales más fácilmente, además de que se le pueden asignar múltiples handlers a un solo evento.

Para utilizar .**addEventListener** es necesario enviarle como parámetros el nombre del evento, por ejemplo, **“click”**, la función handler y, opcionalmente, algún objeto adicional. Para el ejemplo utilizado, quedaría de la siguiente manera:

```
<button>Saludar</button>

<script>
const button = document.querySelector("button");
const saludar = () => alert("Hola!");
button.addEventListener("click", saludar);
</script>
```

Explicación

Asignando múltiples eventos:

```
<button>Saludar</button>

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

<script>
```

```
const button = document.querySelector("button");
const saludar = () => alert("Hola!");
const toggle = () => button.classList.toggle("red");

button.addEventListener("click", saludar); // Hola
button.addEventListener("click", toggle); // Agrega/Remueve el color red CSS
</script>
```

Así como existe un método para agregar eventos, también existe un método **.removeEventListener** para eliminar un **handler.**

```
<button>Saludar</button>


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


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


button.addEventListener("click", saludar); // Agregar listener
button.addEventListener("click", toggle); // Toggle red CSS
button.removeEventListener("click", saludar); // Remueve listener
</script>
```

Explicación

El objeto event

Con la finalidad de manejar de forma más eficiente un evento, es necesario saber algunos otros detalles de lo que está pasando, por ejemplo, quizás en un evento click nos interese saber con qué botón del mouse se hizo. Haverbeke (2019) explica que el navegador crea un objeto del evento, coloca los detalles dentro y posteriormente se los pasa como argumento al handler. Por ejemplo, si queremos implementar el ejemplo mencionado:

```
<button>Haz clic de la manera que gustes</button>

<script>
  let button = document.querySelector("button");
  button.addEventListener("mousedown", event => {
    if (event.button == 0) {
  console.log("Boton Izquierdo");
    } else if (event.button == 1) {
  console.log("Boton Medio");
     } else if (event.button == 2) {
  console.log("Boton Derecho");
    }
    console.log(event);

  });
</script>
```

Explicación

Al imprimir el contenido del evento, se vería algo parecido a esto:

En JavaScript has observado que existen diferentes tipos de eventos, hasta este momento solo hemos visto eventos con mouse, pero también existen otro tipo de eventos que podemos resumir en cuatro grupos:

Explicación

Eventos de mouse: estos se llevan a cabo cuando el usuario los usa para interactuar con los elementos de alguna página. Algunos de ellos son los siguientes:

Evento	Descripción
click	Se produce cuando se pulsa el botón izquierdo del mouse sobre un elemento.
dblclick	Se produce cuando se pulsa dos veces el botón izquierdo del mouse.
mousedown / mouseup	Se produce cuando se pulsa/suelta cualquier botón del mouse sobre un elemento.
mouseover / mouseout	Se produce cuando el puntero del mouse se encuentra dentro/fuera de un elemento y el usuario mueve el puntero.
mousemove	Se produce cuando el mouse se encuentra sobre un elemento.

Eventos de teclado: estos son similares a los eventos del mouse, pero usando el teclado como dispositivo de entrada. Por ejemplo:

Evento	Descripción
keydown / keyup	Se produce cuando se pulsa/suelta cualquier tecla sobre un elemento.

Eventos HTML: se llevan a cabo cuando las ventanas del navegador sufren cambios. A continuación, se presentan algunos de ellos:

Evento	Descripción
load	Se produce en el objeto window cuando la página se carga por completo.
unload	Se produce en el objeto window cuando la página desaparece por completo.
error	Se produce en el objeto window cuando se produce un error de JavaScript.
select	Se produce cuando se seleccionan varios caracteres de un cuadro de texto.
submit	Se produce cuando se pulsa sobre un botón de tipo submit en un formulario.
focus	Se produce en cualquier elemento cuando se obtiene el foco.

Explicación

Eventos DOM: también conocidos como eventos de mutación, estos se llevan a cabo cuando el DOM sufre cambios. Algunos de ellos son los siguientes:

Evento	Descripción
DOMSubtreeModified	Se produce cuando se añaden o eliminan nodos en el subárbol de un documento.
DOMNodeInserted /	Se produce cuando se añade/elimina un nodo como hijo de otro nodo.
DOMContentLoaded	Se produce cuando el DOM está completamente construido.
DOMNodeRemovedFromDocument / DOMNodeInsertedIntoDocument	Se produce cuando se añade/elimina un nodo del documento.

Cierre

En esta experiencia educativa aprendiste acerca de los eventos, que las funciones definidas para cada evento se denominan manejadores de evento y que existen tres maneras de asignar manejadores: por atributos HTML, por propiedades DOM y por los métodos **addEventListtener / removeEventListener.**

Además, conociste algunas buenas prácticas y pros y contras de utilizar cualquiera de las opciones indicadas para asignar manejadores.

Por último, descubriste que existen cuatro diferentes tipos de eventos y los principales de cada una de las categorías.

Referencias bibliográficas

- Haverbeke, M. (2018). *ELOQUENT JAVASCRIPT* (3ª ed.). Estados Unidos: No Starch Press.
- Kantor, I. (2022). *The Modern JavaScript Tutorial.* Recuperado de https://javascript.info/
- LenguajeJS. (s.f.). *¿Qué son los eventos?* Recuperado de https://lenguajejs.com/javascript/eventos/que-son-eventos/

Para saber más

Lecturas

Para conocer más acerca de eventos, te sugerimos leer lo siguiente:

- Kumar, A. (2021). *DOM events in JavaScript*. Recuperado de https://anil-pace.medium.com/dom-events-in-javascript-6ff696e798f8
- MDN. (s.f.). *Referencia de Eventos*. Recuperado de https://developer.mozilla.org/es/docs/Web/Events

Videos

- freeCodeCamp.org. (2021, 21 de junio). *JavaScript Programming - Full Course* [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=jS4aFq5-91M&t=26244s
- Jonmircha. (2020, 15 de mayo). *Curso JavaScript: 71. DOM: Modificando Elementos (Cool Style) - #jonmircha* [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=tqqw2b1kasg
- Jonmircha. (2020, 18 de mayo). *Curso JavaScript: 72. DOM: Manejadores de Eventos - #jonmircha* [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=IQchmLGDXgU
- Jonmircha. (2020, 19 de mayo). *Curso JavaScript: 73. DOM: Eventos con Parámetros y Remover Eventos - #jonmircha* [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=2oHVjLrnRmY

Checkpoints

Asegúrate de:

- Comprender el funcionamiento de los eventos del DOM.
- Entender el uso de los diferentes métodos para implementar manejadores de eventos.
- Identificar los principales eventos de cada una de las cuatro categorías.

Requerimientos técnicos

- Computadora con acceso a Internet.
- Editor de texto.
- Permisos de administrador y/o Git instalado previamente.

Prework

- Haber realizado la actividad guiada del tema 7, de preferencia durante la sesión.
- Leer detenidamente y comprender el material explicado en el tema.
- Practicar todos los ejemplos que se describen previamente en el tema.
- Revisar cada uno de los recursos adicionales que se proponen en este tema.