Programación

Eventos

Decimoquinta semana

Febrero 2022

Cruz García, Iago

Introducción

En el tema anterior, vimos que el DOM permite recoger y manejar los eventos de la página web. Entendemos por evento las acciones de los usuarios en los elementos del HTML: botones, cuadros de texto, selectores de opciones…

Este tema vamos a trabajar el manejo de eventos básico, cómo eliminar eventos y evitar que se propaguen sus funciones.

Recomendación: A partir de ahora es necesario que a la hora de realizar ejercicios nos acostumbremos a buscar información en la documentación oficial. En caso de no poder resolver las dudas con la documentación ofrecida, el siguiente paso será preguntar las dudas en el foro de clase.

Documentación oficial de JavaScript

Foro de la asignatura

Eventos

Lista de disparadores

Eventos

Los eventos se componen de dos partes, el disparador o 'trigger' y el manejador o 'listener'.

Hay que aclarar una cosa antes de continuar aunque es probable que os hayáis dado cuenta a estas alturas. Hasta ahora, el código JavaScript se ejecutaba de arriba a abajo y de izquierda a derecha sin esperas. Sin embargo, los eventos rompen esta dinámica, pues no tienen un orden lógico de ejecución (se puede clickar un botón y luego hacer scroll en la página o al revés).

Lo que ocurre es que un evento asigna un pedazo de código a una acción concreta, por lo que es **muy importante** saber exactamente qué es lo que ocurrirá tras la acción.

Disparador

El hacer click en un elemento o hacer scroll por la página son acciones casuales que cualquier usuario realiza a diario. Para una máquina, no son más que meros cambios de contexto o acciones inocuas, pero para un desarrollador es una oportunidad. Estas acciones las conocemos como **disparadores,** que permiten indicar a nuestro programa que algo sucedió o está sucediendo. Se puede asociar con símiles de la vida real: cuando un semáforo se pone en verde realmente no es un cambio físico o un cambio de estado del ambiente, sin embargo la sociedad está condicionada para

entender que el verde significa 'seguir' o 'pasar'. Ocurre exactamente lo mismo con los elementos del HTML.

Cuando un botón se presiona, envía una señal. Podremos recoger esa señal y decidir qué hacer con ello gracias a los **manejadores**.

Existen multitud de señales y verlas todas sería tedioso. A lo largo del tema veremos algunos ejemplos, pero la lista completa la podéis encontrar en la introducción de este tema. Algunos de los más usados son: load, scroll, click, mouseup, focus, keydown, submit o input.

Manejador

Para poder gestionar un evento, poder realizar acciones cuando un usuario haga click en un botón, escriba un texto, presione una tecla, etc. necesitamos invocar el manejador. Para ello tenemos dos formas de hacerlo.

HTML

Desde el propio HTML se puede utilizar el atributo del disparador necesario seguido de la función en el código JS que debe accionar:

index.html

```
<body>
	<h1 id="cambio">Este texto debe cambiar</h1>
	<button onclick="cambiaTexto()">Pulsa aquí</button>
</body>
<footer>
	<script src="main.js"></script>
</footer>
```

main.js

```
function cambiaTexto() {
	document.getElementById("cambio").innerHTML = "Se cambió el texto"
}
```

Como vemos en *index.html*, en el botón el atributo *onclick* hacemos la llamada al método en *main.js,* por lo que al pulsarlo, se cambiará el texto.

En este enlace de W3C Schools encontrareis una larga lista de manejadores para HTML pero no debería utilizarse este método debido a la poca legibilidad que otorga y lo difícil de seguir la secuencia.

JavaScript

Si recogemos un elemento de nuestro documento (con getElementById, por ejemplo), podremos asociar el manejador utilizando '.addEventListener(...)'. Este método nos permitirá utilizar todas las ventajas de JavaScript y además modificar si queremos que desaparezca el evento, si debe propagarse, utilizar el parámetro de eventos…

index.html

```
<body>
        <h1 id="cambio">Este texto debe cambiar</h1>
        <button id="boton">Pulsa aquí</button>
</body>
<footer>
        <script src="main.js"></script>
</footer>
```

main.js

```
var boton = document.getElementById("boton");
boton.addEventListener('click', () => {
        var texto = document.getElementById("cambio");
        texto.innerHTML = "Se cambió el texto"
});
```

Como vemos, la construcción difiere del ejemplo de HTML. Para empezar, *addEventListener(...)* requiere como mínimo dos parámetros:

- El disparador, en este caso 'click'.
- La función que ocurre al dispararse el evento.

Pero esta forma de asignar los eventos todavía no es óptima todavía, nos queda un paso más.

index.html

```
<body>
	<h1 id="cambio">Este texto debe cambiar</h1>
	<button id=”boton”>Pulsa aquí</button>
</body>
<footer>
	<script src="main.js"></script>
</footer>
```

main.js

```
function __main__() {
	var boton = document.getElementById("boton");
	boton.addEventListener('click', cambioTexto);
}

function cambioTexto() {
	var texto = document.getElementById("cambio");
	texto.innerHTML = "Se cambió el texto";
}

__main__();
```

Como se puede observar, en vez de hacer una función anónima (funciones de un solo uso sin nombre como en el ejemplo anterior), es posible utilizar una función nombrada.

Pero aún encontramos un problema. ¿Cómo puede recibir parámetros este método? No es tan simple como introducir los parámetros entre paréntesis como de costumbre, pero con algo de imaginación…

index.html

```
<body>
	<h1 id="cambio">Este texto debe cambiar</h1>
	<button id=”boton”>Pulsa aquí</button>
</body>
<footer>
	<script src="main.js"></script>
</footer>
```

main.js

```
function __main__() {
	var boton = document.getElementById("boton");
	boton.addEventListener('click', (evt) => {
	cambioTexto(evt);
	});

}

function cambioTexto(evt) {
	console.log(evt.currentTarget.tagName);
	var texto = document.getElementById("cambio");
	texto.innerHTML = "Se cambió el texto";
}

__main__();
```

Utilizando la función anónima y llamando desde ella a la función *cambioTexto(evt)* sí que nos permite utilizar parámetros.

Variables de evento

Las variables de evento nos permite obtener información sobre el tipo de evento que se ejecuta, donde se ejecuta, posición del click del ratón, etc. Tenemos dos variables disponibles:

- El propio evento, que debe declararse como un parámetro en la función anónima. El nombre se lo daremos nosotros.

main.js

```
function __main__() {
        var boton = document.getElementById("boton");
        boton.addEventListener('click', (evento)=>{
          evento.currentTarget.tagname;
        });
}

function cambioTexto() {
        var texto = document.getElementById("cambio");
        texto.innerHTML = "Se cambió el texto";
}

__main__();
```

- La variable de entorno *this*, que será el objeto que disparó el evento. En el caso del botón, *this* nos permitirá acceder a dicho objeto y a sus atributos.

main.js

```
function __main__() {
        var boton = document.getElementById("boton");
        boton.addEventListener('click', (evento)=>{
          evento.currentTarget.tagname;
        });
}

function cambioTexto(evt) {
        this.innerHTML = "Esto es una prueba de eventos"
        var texto = document.getElementById("cambio");
        texto.innerHTML = "Se cambió el texto";
}

__main__();
```

Eliminar el manejador

Para eliminar el manejador de eventos de un elemento, tendremos que utilizar *'.removeEventListener(evento, función)'* donde *evento* debe ser el mismo disparador y *función* la misma que asignamos al agregarlo.

index.html

```
<body>
        <h1 id="cambio">Este texto debe cambiar</h1>
        <button id="boton">Pulsa aquí</button>
</body>
<footer>
        <script src="main.js"></script>
</footer>
```

main.js

```
function __main__() {
	var boton = document.getElementById("boton");
	boton.addEventListener('click', cambioTexto);
	boton.removeEventListener('click', cambioTexto);
}

function cambioTexto() {
	var texto = document.getElementById("cambio");
	texto.innerHTML = "Se cambió el texto";
}

__main__();
```

Este ejemplo es simple, pero comprobareis que al hacer click en el botón, el mensaje ya no cambiará.

Evitar propagación

Uno de los grandes problemas de los eventos en JavaScript es que al añadir un manejador de eventos en una etiqueta anidada, es decir, con etiqueta/s 'padre/s', es que se propaga dicho manejador al contenedor. Este problema es bastante concreto y se puede evitar simplemente escribiendo 'variableDeEvento.stopPropagation()'. En el siguiente enlace podéis leer en que casos sería útil esta estructura.