PROGRAMACIÓN CON JAVASCRIPT

EVENTOS

Contenido

1. Introducción

Cuando un usuario interactúa con una aplicación, se producen una serie de eventos (de teclado, ratón, etc.) que nuestro código debería gestionar de forma adecuada. Hay muchos eventos que pueden ser capturados y procesados. Veremos algunos de los más comunes.

1.1. Eventos de la página

Estos eventos son producidos en el documento HTML. Normalmente afectan al elemento body.

load → Este evento se lanza cuando el documento HTML ha terminado de cargarse. Es útil para realizar acciones que requieran que el DOM haya sido completamente cargado (como consultar o modificar el DOM).

unload → Ocurre cuando el documento es destruido, por ejemplo, después de cerrar la pestaña donde la página estaba cargada.

beforeunload → Ocurre justo antes de cerrar la página. Por defecto, un mensaje pregunta al usuario si usted desea realmente salir de la página, pero hay otras acciones que pueden ser ejecutadas.

resize → Este evento se lanza cuando el tamaño del documento cambia (normalmente se usa si la ventana se redimensiona)

1.2. Eventos del teclado

keydown → El usuario presiona una tecla. Si la tecla se mantiene pulsada durante un tiempo, este evento se generará de forma repetida.

keyup → Se lanza cuando el usuario deja de presionar la tecla.

keypress → Más o menos lo mismo que keydown. Acción de pulsar y levantar.

1.3. Eventos del ratón

click → Este evento ocurre cuando el usuario pulsa un elemento (presiona y levanta el dedo del botón → mousedown + mouseup). También normalmente se lanza cuando un evento táctil de toque (tapón) es recibido.

dblclick → Se lanza cuando se hace un doble clic sobre el elemento

mousedown → Este evento ocurre cuando el usuario presiona un botón del ratón

mouseup → Este evento ocurre cuando el usuario levanta el dedo del botón del ratón

mouseenter → Se lanza cuando el puntero del ratón entra en un elemento

mouseleave → Se lanza cuando el puntero del ratón sale de un elemento

mousemove → Este evento se llama repetidamente cuando el puntero de un ratón se mueve mientras está dentro de un elemento

1.4. Eventos táctiles

Touchstart, touchend, touchmove, touchcancel

1.5. Eventos de formulario

foco → Este evento se ejecuta cuando un elemento (no sólo un elemento de un formulario) tiene el foco (es seleccionado o está activo).

blur → Se ejecuta cuando un elemento pierde el foco.

change → Se ejecuta cuando el contenido, selección o estado de la casilla de selección de un elemento cambia (sólo <input>, <select>, y <textarea>)

input → Este evento se produce cuando el valor de un elemento <input> o <textarea> cambia.

select → Este evento se lanza cuando el usuario selecciona un texto de un <input> o <textarea>.

submit → Se ejecuta cuando un formulario es enviado (el envío puede ser cancelado).

2. Gestor de eventos

Hay muchas formas de asignar un código o función a un determinado evento.

Veremos las dos formas posibles (para ayudar a entender código hecho por otros), pero lo recomendado (y la forma válida para este curso) es usar evento listeners.

2.1. Manejo de eventos clásico (menos recomendado)

Lo primero, podemos poner código JavaScript (o llamar a una función) en un atributo de un elemento HTML. Estos atributos se nombran como los eventos, pero con el prefijo 'on' (clic → onclick). Ejemplo:

```
<!DOCTYPE>
<html>
	<head>
		<title>Ejemplo JS</title>
	</head>
	<body>
		<div id="div1">
			<p>
				<input type="texto" onclick="alerta('¡Me has pulsado!')" />
			</p>
		</div>
		<script src="./ejemplo1.js"></script>
	</body>
</html>
```

Si llamamos a una función, podemos pasarle parámetros (si es necesario). La palabra reservada this y event se pueden usar para pasar el elemento HTML (afectado por el evento) y/o el objeto con información del evento a la función.

```
Archivo: ejemplo1.html

<input type="text" id="input1" onclick="inputClick(this, event)" />

Archivo: ejemplo1.js

function inputClick(element, event) {
   // Mostrará "Un evento click ha sido detectado en  #input1"
   alert("Un evento" + event.type + " ha sido detectado en #" + element.id);
}
```

1 Ejemplo código JS. onclick

Podemos añadir un manejador (función) de evento desde código, accediendo a la propiedad correspondiente (onclick, onfocus, …), o asignarles un valor nulo si queremos dejar de escuchar algún evento.

```
let input = document.getElementById("input1");
input.onclick = function(event) {
   // Dentro de esta función, 'this' se refiere al elemento
   alert("Un evento " + event.type + " ha sido detectado en " + this.id);
}
```

2 Ejemplo código JS. onclick II

2.2. Event listeners (recomendado)

El método para manejar eventos explicado arriba tiene algunas desventajas, por ejemplo, no se pueden gestionar varias funciones manejadoras para un mismo evento.

Para añadir un evento listenero, usemos el método addEventListener sobre el elemento. Este método recibe al menos dos parámetros. El nombre del evento (una cadena) y un manejador (función anónima o nombre de una función existente).

```
let input = document.getElementById("input1");
input.addEventListener('click', function(event) {
   alert("Un evento " + event.type + " ha sido detectado en " + this.id);
});
```

3 Ejemplo código JS. Event listener

Podemos añadir tantos manejadores como queramos. Sin embargo, si queremos eliminar un manejador, debemos indicar qué función estamos eliminando.

```
let inputClick = function(event) {
        console.log("Un evento " event.type + " ha sido detectado en " + this.id);
};
let inputClick2 = function(event) {
        console.log("Yo soy otro manejador para el evento click!");
};
let input = document.getElementById("input1");
// Añadimos ambos manejadores. Al hacer clic, se ejecutarían ambos por orden
input.addEventListener('click', inputClick);
input.addEventListener('click', inputClick2);
// Así es como se elimina el manejador de un evento
input.removeEventListener('click', inputClick);
input.removeEventListener('click', inputClick2);
```

2.3. Objeto del evento

El objeto del evento es creado por JavaScript y pasado al manejador como parámetro.
Este objeto tiene algunas propiedades generales (independientemente del tipo de evento) y otras propiedades específicas (por ejemplo, un evento del ratón tiene las coordenadas del puntero, etc.).

Éstas son algunas propiedades generales que tienen todos los eventos:

target → El elemento que lanza el evento (si fue tomado, etc...).

type → El nombre del evento: 'click', 'keypress', …

cancelable → Devuelve true o false. Si el evento se puede cancelar significa que llamando a event.preventDefault() se puede anular la acción por defecto (El envío de un formulario, el clic de un link, etc…).

bubbles → Devuelve cierto o falso dependiendo de si el evento se está propagando

preventDefault() → Este método previene el comportamiento por defecto (cargar una página cuando se pulsa un enlace, el envío de un formulario, etc.)

stopPropagation() → Previene la propagación del evento.

stopImmediatePropagation() → Si el evento tiene más de un manejador, se llama a este método para prevenir la ejecución del resto de manejadores.

Dependiendo del tipo de evento, el objeto tendrá distintas propiedades:

MouseEvent

button → Devuelve el botón del ratón que lo ha tomado (0: botón izquierdo, 1: la rueda del ratón, 2: botón derecho).

clientX, clientY → Coordenadas relativas del ratón en la ventana del navegador cuando el evento fue lanzado.

pageX, pageY → Coordenadas relativas del documento HTML, si se ha realizado algún tipo de desplazamiento (scroll), éste será añadido (usando clientX, clientY no se añade)

screenX, screenY → Coordenadas absolutas del ratón en la pantalla.

detail → Indica cuántas veces el botón del ratón ha sido tomado (un clic, doble, o triple clic).

KeyboardEvent

key → Devuelve el nombre de la tecla tomada.

keyCode → Devuelve el código del carácter Unicode en el evento keypress, keyup o keydown.

altKey, ctrlKey, shiftKey, metaKey → Devuelven si las teclas “alt”, “control”, “shift” o “meta” deben sido pulsadas durante el evento (Bastante útil para las combinaciones de teclas como ctrl+c).
El objeto MouseEvent también tiene estas propiedades.

3. Propagación de eventos (bubbling)

Muchas veces, existen elementos en una web que se solapan con otros elementos (están contenidos dentro). Por ejemplo, si pulsamos sobre un párrafo que está contenido en un elemento , ¿el evento se ejecutará en el párrafo, en el o en ambos? ¿Cuál se ejecuta primero?

Por ejemplo, veremos qué ocurre con estos dos elementos (un elemento dentro de otro) cuando hacemos clic en ellos.

Archivo: ejemplo1.html

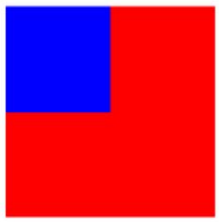
```
<div id="div1" style="background-color: red; width: 200px; height: 200px;">
  <div id="div2" style="background-color: blue; width: 100px; height: 100px;"></div>
</div>
```

Archivo: ejemplo1.js

```
let divClick = function(event) {
  console.log("Has pulsado: " + this.id);
};

let div1 = document.getElementById("div1");
let div2 = document.getElementById("div2");

div1.addEventListener('click', divClick);
div2.addEventListener('click', divClick);
```



4 Ejemplo código HTML y JS

Si hacemos clic sobre el elemento rojo , se imprimirá sólo “Has pulsado: div1”. Sin embargo, si pulsamos sobre el elemento azul (el cual está dentro del elemento rojo) imprimirá ambos mensajes (#div2 primero). En conclusión, por defecto el elemento que está al frente (normalmente un elemento hijo) recibe el evento primero y entonces pasa a ejecutar los manejadores que contiene.

Normalmente, la propagación de eventos va de padres a hijos, pero podemos cambiar este proceso añadiendo un tercer parámetro al método addEventListener y establecerlo en true.

Veremos otro ejemplo:

Archivo: ejemplo1.html

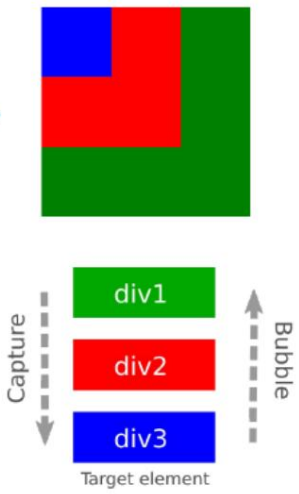
```
<div id="div1" style="background-color: green; width: 150px; height: 150px;">
    <div id="div2" style="background-color: red; width: 100px; height: 100px;">
        <div id="div3" style="background-color: blue; width: 50px; height: 50px;"></div>
    </div>
</div>
```

Archivo: ejemplo1.js

```
let divClick = function(event) {
  // eventPhase: 1 -> capture, 2 -> target (objetivo), 3 -> bubble
  console.log("Has pulsado: " + this.id + ". Fase: " + event.eventPhase);
};

let div1 = document.getElementById("div1");
let div2 = document.getElementById("div2");
let div3 = document.getElementById("div3");

div1.addEventListener('click', divClick);
div2.addEventListener('click', divClick);
div3.addEventListener('click', divClick);
```



5 Ejemplo propagación de eventos

Por defecto, cuando pulsamos el azul (div3), imprime:

Has pulsado: div3. Fase: 2 → Target element

Has pulsado: div2. Fase: 3 → Bubbling

Has pulsado: div1. Fase: 3 → Bubbling

Si se establece un tercer parámetro en true, se imprimirá:

Has pulsado: div1. Fase: 1 → Propagation

Has pulsado: div2. Fase: 1 → Propagation

Has pulsado: div3. Fase: 2 → Target element

Podemos llamar al método stopPropagation en el evento, no continuará la propagación (si es en la fase de captura) o en (la fase de propagación o bubbling).

```
let divClick = function(event) {
   // eventPhase: 1 -> capture, 2 -> target (clicked), 3 -> bubble
   console.log("Has pulsado: " + this.id + ". Fase: " + event.eventPhase);
   event.stopPropagation();
};
```

6 Ejemplo stopPropagation

Ahora, cuando hacemos clic el elemento imprimirá sólo un mensaje, “Has pulsado: div3. Fase: 2”, si el tercer argumento no se ha establecido (o se ha puesto a false), o “Has pulsado: div1. Fase: 1” si el tercer argumento se ha marcado en true (el elemento padre previene a los hijos de recibirlo en este caso).

4. Obtener valores de un formulario

Para capturar el envío de un formulario en JavaScript, debemos capturar el evento sumido de éste. Es más recomendable esto que usar el evento clic en un botón, ya que al presionar la tecla entero también se envía el formulario, por ejemplo.

Además, debemos anular el comportamiento por defecto del evento (recargar la página), ya que si no, no funcionará.

```
const form = document.getElementById('formulario');

form.addEventListener('submit', e => {
   e.preventDefault();
   ...
});
```

7 preventDefault en sumit

Podemos obtener fácilmente los valores de un formulario, obteniendo una referencia al mismo y accediendo a los input a través de su nombre (atributo name).

Campo de texto

```
<input type="texto" id="número" name="número">
form.número.value
```

Checkbox

En ese caso, bajo el mismo nombre tendremos una colección de inputs. Cada input tiene un atributo checked (booleano) que indica si está marcado y su atributo value con el correspondiente valor. Podemos filtrar los marcados y obtener un array con sus valores.

```
<input type="checkbox" id="deporte" name="aficiones" value="deporte"> <label
for="deporte">Deporte</label>
<input type="checkbox" id="viajar" name="aficiones" value="viajar"> <label
for="viajar">Viajar</label> <input
type="checkbox" id="comer" name= "aficiones" value="comer"> <label
for="comer">Comer</label> const
aficiones = Array.from(form.aficiones) .filter(input
        => input.checked) .map(input
        => input .value);
```

Radio

Aunque en este caso también tendremos una colección de inputs, como sólo se puede elegir uno, podemos obtener el valor seleccionado directamente (value).

```
<input type="radio" id="rojo" name="color" value="rojo" checked> <label
for="rojo">Rojo</label> <input
type="radio" id="verde" name ="color" value="verde"> <label
for="verde">Verde</label> <input
type="radio" id="azul" name="color" value="azul">

<label for="azul">Azul</label>

form.color.value
```

File

En este tipo de inputs, existe un array llamado filas, donde podremos acceder a la información del archivo (o archivos) seleccionados. No podremos acceder a la ruta, por ejemplo.

```
<input type="file" id="fichero" name="fichero">
if(formulario.fichero.files.length) { // Si hemos seleccionado un archivo
        const archivo = formulario.fichero.files[0];
        console.log(`Archivo: ${fichero.name}, tipo: ${fichero.type}, tamaño: ${fichero.size}bytes`);
}
```

Además, podremos utilizar clases como FileReader para leer el contenido del archivo, y en este caso devolverlo en formato base64.

```
if(fichero.type.startsWith('image')) { let
        reader = new FileReader();
        reader.readAsDataURL(fichero); // Leer en base64
        reader.addEventListener('load', e => {
                // Ver la imagen en un <img> en HTML
                document.getElementById("imgPreview").src = reader.result;
        });
}
```

5. Enlaces de interés

Eventos: https://www.w3schools.com/jsref/dom_obj_event.asp

6. Bibliografía

Bernal Mayordomo, Arturo. Curso Programación con JavaScript. Cefire 2020

W3Schools - JavaScript Tutorial