Eventos del DOM

Flujo de eventos

Como sabemos, las etiquetas HTML pueden estar anidadas. Si tenemos por ejemplo un elemento con un botón dentro, podemos asignar una función manejadora del clic en el botón, otra función manejadora del clic en el y otra en la página completa. De esta forma **un solo evento (clic) provocará la respuesta de tres elementos de la página.**

Cuando ocurre un evento en un elemento destino, el navegador chequeará si existe un manejador de eventos asociado a ese evento en el elemento, después lo comprobará en el padre, después en el padre del padre... Así en el ejemplo anterior, si se produce el clic en el botón, el navegador chequeará botón para ver si tiene un manejador de eventos asociado al clic, después chequeará el elemento , después al elemento padre del , y así sucesivamente.

Cuando esta fase de recorrer el árbol de nodos "hacia arriba" acaba, el evento atravesará de nuevo el DOM hasta su elemento original, chequeando de nuevo si hay algún manejador de eventos asociado a cada elemento. Cuando termina esta segunda fase, termina el evento.

La fase en que se recorre el árbol DOM hacia arriba, se llama fase **bubbling**, y la fase en que el evento vuelve hacia abajo hasta el elemento destino se llama fase de **captura**. Se puede especificar la fase en la cual queremos que se ejecute el manejador de eventos. Por ejemplo, si queremos que el evento asociado a un clic de ratón en el elemento padre se ejecute antes que el evento asociado al mismo clic en el elemento hijo, debemos especificar que el manejador en el padre se ejecute en la fase bubbling y en el hijo en la fase de captura, o bien ambos en la fase captura.

Diferentes eventos

En el DOM tenemos un número enorme de eventos que se pueden agrupar en 6 grupos básicos:

- Eventos de ratón: click, mousedown(pulsar sin soltar un butón), mouseup (soltar el botón que estaba pulsado), mousemove (mover el ratón), mouseover (el ratón entra en el elemento -pasa por encima-), mouseout (el ratón sale del elemento).
- Eventos de teclado: keypress, keydown y keyup. Keypress sólo ocurre con las teclas que producen entrada de caracteres y la tecla “delete”, por lo tanto es lo más útil para saber el carácter que se pulsó en el teclado.
- Eventos de objeto: load (para el objeto windows por ejemplo), error, resize, scroll, etc.
- Eventos de formulario: select, change, submit, reset, focus, etc.
- Eventos de interfaz de usuario: focusin y focusout.
- Eventos DOM o de mutación: se originan cuando se produce un cambio en la estructura DOM de la página.

Asignar manejadores de eventos

addEventListener() es un método que toma tres argumentos: el tipo de evento (click, keypress, etc), el argumento listener (código que se va a ejecutar cuando ocurra el evento), y opcionalmente un parámetro booleano que indica la fase en la que se ejecutará el código. Si el tercer parámetro es true se ejecutará durante la fase de *captura*, y si es **false (valor por defecto)** se ejecutará durante la de *bubling* (la primera, la que va del elemento más interno al más externo).

```
var miParrafo = document.getElementById("par");
function manejadorEventos(event) {
	console.log("Has hecho click");
}
miParrafo.addEventListener("click", manejadorEventos, false);
```

Cuando se haga clic en el párrafo, aparecerá una alerta.
Se puede asignar más de un manejador de eventos a un solo evento. Se ejecutarán en el orden en que fueron asignados. Por ejemplo:

```
miParrafo.addEventListener("click", primerManejadorEventos, false);
miParrafo.addEventListener("click", segundoManejadorEventos, false);
```

Liberar manejadores de eventos

Se usa el siguiente método:
removeEventListener() que toma tres parámetros: el tipo de evento, la función a eliminar (si no existe no hará nada) y un booleano que indica la fase. No hay forma de liberar manejadores de eventos definidos con una función anónima en línea.

```
var miParrafo = document.getElementById("par");
function manejadorEventos(event) {
	console.log("Desenlazando el manejador de eventos");
	miParrafo.removeEventListener("click", manejadorEventos, false);
}
miParrafo.addEventListener("click", manejadorEventos, false);
```

En el ejemplo anterior, se ejecutará la manejadora cuando se haga clic en el párrafo y esa manejadora se liberará a sí misma.
Si se asocia una función a un flujo de eventos determinado, esa función sólo se puede liberar en el mismo tipo de flujo de eventos. En el siguiente ejemplo la última instrucción no tiene ningún efecto:

```
function muestraMensaje() {
	alert("Has pulsado el ratón");
}
var elDiv = document.getElementById("div_principal");
elDiv.addEventListener("click", muestraMensaje, false);

// Más adelante se decide desasociar la función al evento
elDiv.removeEventListener("click", muestraMensaje, true);
```

El objeto Event

Normalmente, la función que procesa el evento necesita información relativa al evento producido: la tecla que se ha pulsado, la posición del ratón, el elemento que ha producido el evento, etc.
El objeto event es un objeto que se crea automáticamente cuando se produce un evento y que se destruye de forma automática cuando se han ejecutado todas las funciones manejadoras asignadas al evento. Si se necesita, se puede tener acceso a él en un script.

```
elDiv.addEventListener("click", manejadora, false);
...
function manejadora(elEvento) {
	//elEvento es el nombre que tendrá mi objeto Event
}
```

```
elDiv.addEventListener("click", function(elEvento) {
			//elEvento es el nombre que tendrá mi objeto Event
		}, false);
```

En la función se define un parámetro para poder referirnos a él, pero no hay que pasárselo, se crea automáticamente.

Contexto de ejecución de eventos

Cuando un manejador de evento se ejecuta, el navegador le pasa como parámetro un **objeto Event** que contiene algunas propiedades que proporcionan detalles sobre el evento. Este objeto también tiene algunos métodos interesantes para modificar el comportamiento de la propagación de eventos. Diferentes clases de eventos tienen distintos tipos de objetos evento asociados.

Propiedades y métodos

Se enumeran a continuación los más notables:

evento.clientX, evento.clientY: Coordenadas de la posición del ratón respecto del área visible de la ventana (en eventos asociados al ratón)
evento.offsetX, evento.offsetY: Coordenadas de la posición del ratón respecto al elemento destino (por ejemplo un botón)
evento.keyCode: Indica el código ASCII de la tecla pulsada (en eventos asociados al teclado)
evento.target: Un puntero al elemento del DOM que dispara el evento (por ejemplo al elemento en el que hace clic el usuario), aunque el manejador de eventos esté asignado a otro elemento que lo contenga (manejador de eventos asignado a un elemento ancestro del elemento en el que se ha hecho clic).
evento.currentTarget: Un puntero al elemento del DOM al que se asignó el manejador de eventos. Por ejemplo si asignamos un manejador de eventos en un nodo <p> que contiene un elemento <a> que a su vez contiene un , entonces evento.currentTarget se refiere a <p> independientemente de que el usuario haga clic en o en <a> para desencadenar el evento. En algunos casos esto será "this".

Ejemplos en c9: T3-eventosRaton
T3-ejemploFAQs/ejFAQs1.html
T3-ejemploFAQS/Ejfaqs2.html

evento.eventPhase: Un entero que indica la fase en la que está el evento: 1 para capture, 2 para target, 3 para bubbling.
evento.type: Una cadena indicando el tipo del evento ("click", "keypress", etc.).
evento.relatedTarget: Se utiliza en algunos eventos específicos para apuntar al elemento relacionado con el elemento donde se originó el evento. En el evento *mouseover* indica el elemento que el cursor acaba de dejar (de donde viene el ratón), en el evento *mouseout* indica el elemento donde acaba de entrar el cursor.
evento.stopPropagation(): Detiene la propagación de eventos . Si se ha asignado más de un manejador a este elemento para este evento, se ejecutarán los manejadores que queden.
evento.stopImmediatePropagation(): Parecido stopPropagation(), pero detiene también los manejadores de eventos que queden por ejecutar para este elemento.
evento.preventDefault(): Se emplea para cancelar la acción predefinida del evento. Por ejemplo, si se asigna un manejador al evento clic de una etiqueta <a> y ponemos este método dentro del manejador, impedirá que se siga el link. Recordad que la acción por defecto si se hace clic en un link es cargar la página o ir al sitio especificado por el atributo *href* del link. De igual forma, la acción por defecto si hacemos clic en un botón submit o reset en un formulario es enviar o borrar los datos en el formulario. Para evitarlo, se hace uso del método preventDefault(). Por ejemplo:

```
var manejadorEvento = function (evento) {
       evento.preventDefault();
}
```

En el siguiente ejemplo vemos cómo funciona la propiedad evento.relatedTarget:

.html

```
<html>
	<head>
		<title>Eventos de ratón</title>
	    <meta charset="utf-8">
	<link rel="stylesheet" type="text/css" href="eventosRaton.css" />
	<script src="eventosRaton.js"></script>
	</head>
	<body>
		<div id="exterior">
			<div id="arriba"></div>
			<div id="abajo"></div>
		</div>
	</body>
</html>
```

.css

```
div > div {
    height: 128px;
    width: 128px;
}
#arriba {
    background-color: red;
}
#abajo {
    background-color: blue;
}
```

.js

```
window.onload=inicio;
function inicio() {
    document.getElementById("arriba").addEventListener("mouseover", manejadorOver);
    document.getElementById("arriba").addEventListener("mouseout", manejadorOut);
    document.getElementById("abajo").addEventListener("mouseover", manejadorOver);
    document.getElementById("abajo").addEventListener("mouseout", manejadorOut);
}

function manejadorOut(event) {
    alert("salió de " + evento.target.id + " hacia " + evento.relatedTarget.id);
}

function manejadorOver(event) {
  alert("entró en " + evento.target.id + " desde " + evento.relatedTarget.id);
}
```

En el siguiente ejemplo asignamos un manejador de eventos al evento *clic* en cada elemento *li*, cuando se hace clic en un elemento li, dicho elemento se mueve a otra lista <ul>. Podemos hacerlo de otra forma aprovechando el hecho de que los eventos "brotan" a través del DOM usando una manera llamada *delegación de eventos* que veremos en el siguiente apartado.

```
		<h3>Rescata gatitos</h3>
		<p>Haz click para llevar cada uno a su cesta</p>
		<ul id="gatitos">
			<li>Rowly</li>
			<li>Fred</li>
			<li>Mittens</li>
			<li>Lenore</li>
		</ul>
		<ul class="cesta"></ul>
		<script>
			var cesta = document.querySelector(".cesta"),
				gatitos = document.querySelectorAll("li"),
				numeroGatitos = kittens.length,
				i;
			for(i = 0; i < numeroGatitos; i++) {
				gatitos[i].addEventListener("click", function(event) {
					cesta.appendChild(evento.target);
				}, false);
			}
		</script>
```

Delegación de eventos

Cuando en el ejemplo anterior se hace click en un elmento <li>, también se está haciendo click en <ul>, <body> y <html>. Se dice que el evento se propaga a través de todos esos elementos.

La delegación de eventos es una forma de permitir que un manejador de eventos sea asignado a un elemento que está más arriba en el árbol del DOM que el elemento destino original. De esta forma se reduce el número de manejadores de eventos y se consigue mayor eficiencia. El ejemplo anterior, rehecho con delegación de eventos: en lugar de aplicar un manejador a cada elemento li, se delega el manejo de evento al elemento que lo contiene <ul>.

```
var cesta = document.querySelector(".cesta"),
gatitos = document.getElementById("kittens"); //ul
gatitos.addEventListener("click", function(event) {
        cesta.appendChild(evento.target);
}, false);
```

Ejercicio 1:

> Hacer el ejemplo FAQs con delegación de eventos

Eventos de tiempo

Permiten ejecutar funciones después de un periodo de tiempo especificado.

Ejecutar una función después de un tiempo una vez

El primer tipo de temporizador llama a su función sólo una vez. Para crearlo se usa el método setTimeout. Su primer parámetro es la función a ejecutar. El segundo parámetro es el número de milisegundos que hay que esperar antes de que se llame a la función.

setTimeout (funcion, espera)

Para cancelar el temporizador usamos el método clearTimeout.

clearTimeout(temporizador)

```
temporizador = setTimeout(ocultarMensaje, 5000);
function ocultarMensaje(){
        ….
        clearTimeout(temporizador);
}
```

Un ejemplo con closure:

```
(function() {
    var ejemploTiempo = (function() {
        //ejemploTiempo es la función que hay en el return pero las variables son persistentes
        //Las siguientes variables van a conservar su valor entre llamadas a la función de retorno (clausura)
        // y son variables privadas para la función ejemploTiempo
        var suma = 0;
        return function(llamada) {
            window.setTimeout(function() {
                suma = suma + 1;
                console.log('llamada: '+llamada);
                console.log('suma vale: '+suma);
            }, (Math.random() * 10000));
        };
    })(); //esto hace que la variable ejemploTiempo sea el resultado de la ejecución de la funcion anónima
    ejemploTiempo('a');
    ejemploTiempo('b');
}());
```

Ejecutar una función periódicamente

Llama a la función repetidas veces. Para crear el temporizador usamos el método setInterval. Su primer parámetro es la función a llamar, el segundo parámetro es el intervalo de tiempo transcurrido entre llamadas. Para cancelar este tipo de temporizador se usa clearInterval.

setInterval (funcion, intervalo)

clearInterval (temporizador)

```
var contador = 0;
var temporizador = setInterval(
        function (){
                contador++;
                document.getElementById("contador").innerText=contador;
        },
        1000 );
}
```

(Ejemplos: FAQ's, timers, y slide_show)
(Ejemplo eventosTiempo que incluye una closure)

Accesibilidad y manejadores de eventos

Si queremos que nuestras páginas sean "accesibles" de manera que faciliten el acceso de las personas con discapacidad, debemos seguir unas pautas a la hora de desarrollar nuestra web. Una página accesible los sería tanto para una persona con discapacidad, como para cualquier otra persona que se encuentre bajo circunstancias externas que dificulten su acceso a la información (ruidos externos, pantallas con visibilidad reducida, etc.).

En el siguiente enlace podéis leer más sobre el tema:

https://www.w3c.es/Divulgacion/accesibilidad

En lo relativo a accesibilidad y manejadores de eventos podemos decir lo siguiente:

Evento ***focus***: se dispara cuando el cursor se sitúa en un elemento de un formulario, o cuando el usuario utiliza el tabulador para situarse en dicho elemento.

Evento ***blur***: se dispara cuando el cursor abandona un elemento de un formulario o el usuario utiliza el tabulador para salir de dicho elemento.

Ambos, Focus y Blur son manejadores de eventos independientes del dispositivo, pudiendose ejecutar con el ratón, el teclado u otra tecnología de asistencia.

Evento ***click***: el manejador de este evento se usa con links navegables o controles de formularios. La mayoría de navegadores disparan este evento al pulsar la tecla Enter cuando el link o el control tienen el foco. En este caso el manejador del evento click es independiente del dispositivo.

Evento ***dblClick*** no tiene un manejador independiente del dispositivo equivalente.

https://webaim.org/techniques/javascript/eventhandlers

(Ejemplos: FAQ's accesibilidad)

Pasar parámetros a un manejador de eventos

Ejemplo:

```
var primerParrafo = document.getElementById("primerParrafo");
    var numero = 3;
primerParrafo.addEventListener("click", function(evento) {
        miManejadorEvento(evento, numero)
    }, false);
function miManejadorEvento(evento, numero) {
    alert("¡Has hecho click en " + evento.target.id + "! " + numero);
}
```

Ejercicio 2:

Desarrolla una aplicación parecida a la aplicación FAQs que se ha mostrado de ejemplo pero que despliegue una lista de nombres de paisajes naturales en cada link.

Los link estarán agrupados en "playas" y "montañas". En principio aparecerán sólo las cabeceras de los grupos con el signo + en la parte izquierda. Si el usuario hace clic en uno de los grupos (playas o montañas), se desplegará una lista de nombres de paisajes del tipo correspondiente.

Si el usuario hace clic en uno de los nombres desplegados aparecerá una imagen del paisaje asociado a la derecha de la lista.

Además, cada vez que el usuario haga clic en un grupo con un signo más o menos delante, la imagen del paisaje desaparecerá.

Asegúrate de cancelar la acción por defecto de los links.

Ejercicio 3:

Realiza una página que saque una serie de diapositivas a partir de unos ficheros con fotos. El pase de diapositivas será automático cada 2 segundos y aparecerá además de la foto el título de la misma.

Puedes añadir las funcionalidades que se te ocurran.

Bibliografía:

JavaScript Programmer's Reference
Ed: Apress
Aut: Thomas Valentine and Jonathan Reid
Murach JavaScript and jQuery
Ed: Mike Murach & Associates, Inc.
Aut: Zak Ruvalcaba and Mike Murach