LENGUAJES DE MARCAS Y SISTEMAS DE GESTIÓN DE LA INFORMACIÓN

4. Eventos en Javascript

Profesor: Alejandro Amat Reina

ÍNDICE

1. Eventos en Javascript

En la programación tradicional, las aplicaciones se ejecutan secuencialmente de principio a fin para producir sus resultados. Sin embargo, en la actualidad el modelo predominante es el de la programación basada en eventos. Los scripts y programas esperan sin realizar ninguna tarea hasta que se produzca un evento. Una vez producido, ejecutan alguna tarea asociada a la aparición de ese evento, y cuando concluye el script o programa vuelve al estado de espera.

Los eventos de JavaScript permiten la interacción entre las aplicaciones JavaScript y los usuarios. Cada vez que se pulsa un botón, se produce un evento. Cada vez que se pulsa una tecla, también se produce un evento. No obstante, para que se produzca un evento no es obligatorio que intervenga el usuario, ya que, por ejemplo, cada vez que se carga una página, también se produce un evento.

1.1. Tipos de Eventos

Cada elemento HTML tiene definida su propia lista de posibles eventos que se le pueden asignar. Un mismo tipo de evento (por ejemplo, pinchar el botón izquierdo del ratón) puede estar definido para varios elementos HTML y un mismo elemento HTML puede tener asociados diferentes eventos.

Las siguientes tablas resumen los eventos más importantes definidos por JavaScript:

- Eventos de la página:

Evento	**Descripción**
`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
`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 quiere 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)

- Eventos del teclado:

Evento	**Descripción**
`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.

- Eventos del ratón:

Evento	Descripción
`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 (tap) es recibido
`dblclick`	Se lanza cuando se hace un doble click 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

- Eventos touch:

Evento	Descripción
`touchstart`	Se lanza cuando se detecta un toque en la pantalla táctil
`touchend`	Se lanza cuando se deja de pulsar la pantalla táctil
`touchmove`	Se lanza cuando un dedo es desplazado a través de la pantalla
`touchcancel`	Este evento ocurre cuando se interrumpe un evento táctil

- Eventos de formulario:

Evento	Descripción
`focus`	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 del checkbox 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 una cancelación)

1.2. Manejadores de eventos

En la programación orientada a eventos, las aplicaciones esperan a que se produzcan los eventos. Una vez que se produce un evento, la aplicación responde ejecutando cierto código especialmente preparado. Este tipo de código se denomina "manejadores de eventos" (del inglés "event handlers") y las funciones externas que se definen para responder a los eventos se suelen denominar "funciones manejadoras".

A continuación veremos tres formas de asignar un manejador de evento para un elemento HTML. Las dos primeras se ven para ayudar a entender código antiguo o hecho por otros, pero la forma correcta de trabajar, y la que usaremos en clase es la tercera forma (event listeners).

- **Manejadores de eventos como atributos HTML**: La forma más sencilla (y menos recomendable) de incluir un manejador de evento, es incluir un atributo HTML con el mismo nombre del evento que se quiere procesar, con la `palabra on delante. El siguiente ejemplo muestra un mensaje cuando el usuario pincha en el botón:

```
<input type="button" value="pínchame" onclick="alert('Gracias');">
```

  En este caso, como se quiere mostrar un mensaje cuando se pincha con el ratón sobre un botón, el evento es onclick. El contenido del atributo es una cadena de texto que contiene todas las instrucciones JavaScript que se ejecutan cuando se produce el evento. Este es un método sencillo, pero poco aconsejable para tratar con los eventos en JavaScript.

- **Manejadores de eventos semánticos**: Utilizar los atributos HTML para añadir manejadores de eventos tiene un grave inconveniente: "ensucian" el código HTML de la página.

  Como es conocido, al crear páginas web se recomienda separar los contenidos (HTML) de la presentación (CSS). En lo posible, también se recomienda separar los contenidos (HTML) de la programación (JavaScript). Mezclar JavaScript y HTML complica excesivamente el código fuente de la página, dificulta su mantenimiento y reduce la semántica del documento final producido.

  Afortunadamente, existen métodos alternativos para definir los manejadores de eventos de JavaScript. Esta técnica consiste en:

  1. Crear una función de JavaScript encargada de manejar el evento.
  2. Asignar la función a un evento concreto del elemento HTML mediante DOM.

  Así, el ejemplo anterior se puede transformar en:

```
<input id="pinchable" type="button" value="pínchame"
       onclick="alert('gracias por pinchar');">
```

```
document.getElementById('pinchable').onclick = function () {
    alert('Gracias');
};
```

  El código HTML resultante es muy "limpio", ya que no se mezcla con el código JavaScript, pero hemos de recordar que para poder utilizar las funciones del DOM el árbol debe estar totalmente construido, por lo tanto, debemos asegurarnos de que la asignación de los eventos a los distintos elementos la realizamos en un lugar que nos asegure que se ha efectuado la carga completa de la página. El lugar idóneo sería en el evento onload de la ventana. Así, el ejemplo anterior sería menos incorrecto de la siguiente forma:

```
'use strict';
(function() {
    window.onload = function () {
        document.getElementById('pinchable').onclick = function () {
            alert('Gracias');
        };
    };
})();
```

Cualquiera de estos tres modelos funciona correctamente en todos los navegadores disponibles en la actualidad.

- **Manejadores de eventos con Event listeners**: Como hemos comentado, el método anterior, no es del todo correcto, ya que tiene algunas desventajas. Por ejemplo, no se pueden gestionar (añadir, quitar) en el tiempo varias funciones manejadoras para un mismo evento.

  Si queremos disponer de esa posibilidad, debemos usar el método addEventListener sobre el elemento al cual le queremos asignar un manejador. Este método recibe al menos dos parámetros. El nombre del evento (fíjate que ahora no hay que poner el on delante del nombre del evento) y un manejador (función anónima o nombre de una función existente). Veamos el ejemplo anterior utilizando este método:

```
'use strict';
(function() {
    window.addEventListener('load', function () {
        document.getElementById('pinchable').addEventListener('click', function () {
            alert('Gracias');
        });
    });
})();
```

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("Gracias");
};
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 cómo se elimina el manejador de un evento
input.removeEventListener('click', inputClick);
input.removeEventListener('click', inputClick2);
```

1.3. El objeto event

Cuando se produce un evento, no es suficiente con asignarle una función responsable de procesar ese evento. 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 el mecanismo definido por los navegadores para proporcionar toda esa información. Se trata de 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 asignadas al evento.

El estándar DOM especifica que el objeto event es el único parámetro que se debe pasar a las funciones encargadas de procesar los eventos. Por tanto, en los navegadores que siguen los estándares, se puede acceder al objeto event pasándoselo a la función manejadora:

```
'use strict';
(function() {
    window.addEventListener('load', function () {
        document.getElementById('pinchable').addEventListener('click', function (event) {
            alert("Un evento " + event.type + " ha sido detectado en " + this.id);
        });
    });
})();
```

1.4. Propiedades y métodos del objeto event

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, un evento de teclado tiene la tecla pulsada, etc.).

A continuación veremos una tabla que nos muestra las propiedades y métodos más importantes existentes para el objeto event.

Propiedad	**Devuelve**	**Descripción**
`bubbles`	Boolean	Indica si el evento pertenece al flujo de eventos de *bubbling*
`cancelable`	Bolean	Indica si el evento se puede cancelar
`cancelBubble`	Boolean	Indica si se ha detenido el flujo de eventos de tipo *bubbling*
`charcode`	Número entero	El código unicode del carácter correspondiente a la tecla pulsada
`currentTarget`	Element	El elemento que es el objetivo del evento
`detail`	Número entero	El número de veces que se han pulsado los botones del ratón
`eventPhase`	Número entero	La fase a la que pertenece el evento: 0 – Fase capturing 1 – En el elemento destino 2 – Fase bubbling
`isChar`	Boolean	Indica si la tecla pulsada corresponde a un carácter
`keyCode`	Número entero	Indica el código numérico de la tecla pulsada
`metaKey`	Número entero	Devuelve true si se ha pulsado la tecla META y false en otro caso
`pageX, pageY`	Número entero	Coordenada X o Y de la posición del ratón respecto de la página

`preventDefault()`	Función	Se emplea para cancelar la acción predefinida del evento
`relatedTarget`	Element	El elemento que es el objetivo secundario del evento (relacionado con los eventos de ratón)
`stopPropagation()`	Función	Se emplea para detener el flujo de eventos de tipo *bubbling*
`target`	Element	El elemento que origina el evento
`timeStamp`	Número	La fecha y hora en la que se ha producido el evento
`button`	Número	Devuelve el botón del ratón que lo ha pulsado (0: botón izquierdo, 1: la rueda del ratón, 2: botón derecho).
`type`	String	El nombre del evento: 'click', 'keypress', etc.
`stopInmediatePropagation`	Función	Si el evento tiene más de un manejador, se llama a este método para prevenir la ejecución del resto de manejadores

1.5. Tipos de eventos

Evidentemente, cada propiedad tendrá o no sentido en función del evento en el que nos encontremos, por ejemplo, no tendrá sentido que utilicemos la propiedad `button` en un evento de tipo `keyPress` que se refiere al teclado, del mismo modo que no tiene sentido acceder a la propiedad `keyCode` en un evento de tipo `mousemove` que se refiere al ratón.

La lista completa de eventos que se pueden generar en un navegador se puede dividir en cuatro grandes grupos.

1.5.1. Eventos de ratón

Se originan cuando el usuario emplea el ratón para realizar algunas acciones (`clic`, `dblclick`, `mousedown`, `mouseout`, `mouseover`, `mouseup`, `mousemove`). Todos los elementos de las páginas soportan estos eventos.

Podemos encontrar en el objeto event las siguientes propiedades referidas a estos eventos:

- Las coordenadas del ratón (todas las coordenadas diferentes relativas a los distintos elementos).
- La propiedad `type`
- La propiedad `target`
- Las propiedades `shiftKey`, `ctrlKey`, `altKey` y `metaKey`
- La propiedad `button` (sólo en los eventos `mousedown`, `mousemove`, `mouseout`, `mouseover` y `mouseup`)

Los eventos `mouseover` y `mouseout` tienen una propiedad adicional denominada `relatedTarget`. En el evento `mouseout`, `relatedTarget` apunta al elemento al que se ha movido el ratón. En el evento `mouseover`, `relatedTarget` apunta al elemento desde el que se ha movido el puntero del ratón.

1.5.2. Eventos de teclado

Se originan cuando el usuario pulsa sobre cualquier tecla de su teclado (`keydown, keypress, keyup`).

El objeto event contiene las siguientes propiedades para los eventos de teclado:

- La propiedad `keyCode`
- La propiedad `charCode`
- La propiedad `target`
- Las propiedades `shiftKey, ctrlKey, altKey y metaKey`

Cuando se pulsa una tecla correspondiente a un carácter alfanumérico, se produce la siguiente secuencia de eventos: `keydown, keypress, keyup`. Cuando se pulsa otro tipo de tecla, se produce la siguiente secuencia de eventos: `keydown, keyup`. Si se mantiene pulsada la tecla, en el primer caso se repiten de forma continua los eventos `keydown y keypress` y en el segundo caso, se repite el evento `keydown` de forma continua.

1.5.3. Eventos HTML

Se originan cuando se producen cambios en la ventana del navegador o cuando se producen ciertas interacciones entre el cliente y el servidor. Podemos encontrar los siguientes eventos html:

- `load`: Se produce en el objeto `window` cuando la página se carga por completo. En el elemento `<img>` cuando se carga por completo la imagen.
- `unload`: Se produce en el objeto `window` cuando la página desaparece por completo (al cerrar la ventana del navegador por ejemplo).
- `error`: Se produce en el objeto `window` cuando se produce un error de javaScript. En el elemento `<img>` cuando la imagen no se ha podido cargar por completo.
- `select`: Se produce cuando se seleccionan varios caracteres de un cuadro de texto (`<input> y <textarea>`).
- `change`: Se produce cuando un cuadro de texto (`<input> y <textarea>`) pierde el foco y su contenido ha variado. También se produce cuando varía el valor de un elemento `<select>`.
- `submit`: Se produce cuando se pulsa sobre un botón de tipo `submit (<input type="submit">`).
- `reset`: Se produce cuando se pulsa sobre un botón de tipo reset (`<input type="reset">`).
- `resize`: Se produce en el objeto `window` cuando se redimensiona la ventana del navegador scroll y en cualquier elemento que tenga una barra de scroll, cuando el usuario la utiliza. El elemento `<body>` contiene la barra de scroll de la página completa.
- `focus`: Se produce en cualquier elemento (incluido el objeto `window`) cuando el elemento obtiene el foco.

- `blur`: Se produce en cualquier elemento (incluido el objeto `window`) cuando el elemento pierde el foco.

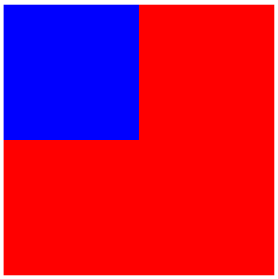
2. Propagación de eventos (bubbling)

Muchas veces, hay 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 ejecutará primero?.

Vamos a ver qué ocurre con un elemento dentro de otro cuando hacemos clic en ellos.

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

```
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);
```

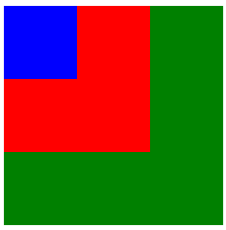


Si hacemos click sobre el elemento rojo , se imprimirá solo “Has pulsado: div1”. Sin embargo, si pulsamos sobre el elemento azul (el cual está dentro del elemento rojo) imprimirá ambos mensajes (#div2 primero). En conclusión, por defecto el elemento el cual está al frente recibe el evento primero y pasa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

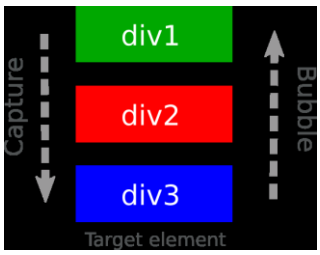
Normalmente, la propagación de eventos va de padres a hijos, pero podemos cambiar este proceso añadiendo un tercer parámetro al método addEventListener y establecerlo a true.

Vamos a ver otro ejemplo:

```
<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>
```



```
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);
```



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

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 a 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();
};
```

Ahora, cuando hacemos click el elemento imprimirá solo 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 a true (el elemento padre previene a los hijos de recibirlo en este caso).

3. Ejercicios

En este tema hemos visto cómo podemos interactuar con el usuario por medio de los eventos.

A continuación, plantearemos una serie de ejercicios para mejorar nuestro carrito de la compra y practicar los conceptos vistos en el tema.

3.1. Ejercicio 1

Si en el tema anterior añadíamos el primer artículo de la lista al carrito al cargar la página, ahora no haremos nada al cargar la página. En lugar de esto, añadiremos un artículo al carrito cuando el usuario haga doble click sobre él, siguiendo las instrucciones que se indicaron en el tema anterior.

A tener en cuenta:

- Debes añadir un manejador para el evento load de la página.
- En este manejador de evento, debes obtener todos los artículos de la lista y añadirle, a cada uno de ellos, un manejador para el evento dblclick.
- En el manejador del evento dblclick haremos los mismo que hicimos en el ejercicio del tema anterior, pero utilizando el ítem sobre el que se haya hecho dblclick.
- Tienes que comprobar si quedan artículos disponibles. El stock debe ser mayor que 0, si no es así no se podrá introducir el artículo en el carrito.
- De momento sólo se verán en el carrito los cuatro primeros artículos añadidos, pero más adelante solucionaremos este problema.

3.2. Ejercicio 2

Para eliminar un producto del carrito se debe pulsar sobre el enlace que se ha creado al introducirlo, por lo tanto, tendremos que asociar un manejador de evento al enlace delete que creamos dinámicamente. Esto lo haremos con el evento click visto durante el tema.

Al pulsar sobre el enlace debes hacer lo siguiente:

- Obtén el id del elemento padre, es decir, el artículo que hay dentro del carrito, y a partir de éste, obtén el id de la lista de artículos a comprar. Como esto lo haces desde el evento click del enlace, puedes acceder a él mediante this[1], y al artículo del carrito mediante this.parentNode. Una vez obtenido el id del artículo del carrito que quieres eliminar, puedes obtener el de la lista de artículos eliminando la letra c que añadiste al insertarlo al carrito.
- A partir del id obtenido incrementa el stock disponible del artículo que se elimina del carrito. Si el stock estaba a cero elimina la clase agotado del mismo.

[1] OJO! Cuando usamos arrow functions no disponemos del objeto this, por lo tanto, estas no son aconsejables como manejadoras de eventos

- Actualiza el total de productos comprados restándole uno.
- Actualiza el precio total de la compra restándole el precio del artículo eliminado.
- Elimina el artículo del carro de la compra.
- Como la acción por defecto de un enlace al hacer click sobre él es navegar a otra página, debes evitar este comportamiento. Para hacerlo llama al método preventDefault del objeto event.

3.3. Ejercicio 3

Al pulsar sobre el botón btn_clear debes vaciar el carrito, pero no sólo eso, también tienes que actualizar los stocks disponibles de los artículos, el número de artículos comprados y el precio total de la compra.

Esto puede parecer engorroso, pero si aprovechas las funcionalidades que ya tienes implementadas, puedes hacerlo de una forma muy sencilla.

¿Cómo vaciarías el carrito si no existiera el botón vaciar y tuvieras que hacerlo de forma manual? Tendrías que pulsar con el ratón sobre el botón delete de todos los artículos del carrito, ¿y esto actualizaría todos los elementos que hemos comentado? Sí. Pues ya está, lo que tienes que hacer al pulsar el botón vaciar, es lanzar por código un evento de click del ratón sobre el botón delete de todos los artículos del carrito.

Simplemente, selecciona los elementos delete y lanza el evento correspondiente llamando al método .click().