UNIDAD 4. Interacción con el usuario: eventos y formularios

Desarrollo Web en Entorno Cliente

2º DAW

Contenidos

Definición de eventos

Introducción

En JavaScript se puede definir código que se ejecutará cuando se produzca un determinado evento. Los **eventos** se disparan cuando el usuario realiza alguna acción sobre la página web, como, por ejemplo: pulsar una tecla `keypress`, pulsar y soltar el botón del ratón `click`, pasar el ratón por encima de un elemento `mouseover`, finalizar la carga de la página `load`, etc. y tendrán funciones asociadas que permiten atender al evento.

Las acciones de un usuario pueden dar lugar a una sucesión de eventos. Por ejemplo, al pulsar sobre un botón de tipo `<input type="submit">` se desencadenan los eventos `mousedown`, `click`, `mouseup`, `submit`.

Más formalmente: un **evento** indica la ocurrencia de un hecho que es atendido por un manejador. Un **manejador de un evento** es un bloque de código o función asociada al evento que se ejecuta al ocurrir este.

Un programa JavaScript se guía por eventos (*event driven*). El script inicial debe configurar los primeros manejadores y después solo se ejecutarán los manejadores de los eventos que ocurren.

La gestión de eventos ha resultado ser una fuente de incompatibilidades entre los navegadores web, por lo que hay varios modos o modelos de asociar código a los eventos:

- **Modelo de eventos en línea**: se añade como un atributo en la etiqueta HTML: `<a href="…" onClick=acción/>`
- **Modelo de eventos tradicional**: los eventos son una propiedad de los elementos: `elementoDOM.onclick = acción;`
- **Modelo de eventos de Microsoft** (IE8 y anteriores) IE11 y posteriores no soportan este modelo: utiliza `elementoDOM.attachEvent(evento, acción);`
- **Modelo avanzado de registro de eventos del W3C**: utiliza `addEventListener();`

Hoy en día **se recomienda usar solamente el modelo de eventos del W3C**, que se creó para estandarizar el manejo de eventos y, por lo tanto, eliminar la incompatibilidad con navegadores.

Modelo de eventos en línea

Este modelo se aprende a utilizar por su sencillez y necesidad de adaptar páginas web antiguas, pero **no es nada recomendable ya que NO separa HTML y JavaScript**.

Los elementos **HTML** tienen **atributos** que asocian eventos, por ejemplo: `onclick` (clic), `ondbclick` (doble clic). El nombre de esos atributos es `on` + nombre del evento. Como hemos visto, para el evento `click`, existe la propiedad `onclick`.

El **valor** que tendrán esos atributos asociados a eventos será código JavaScript, que se ejecutará al ocurrir el evento, es decir, será el **manejador del evento**:

- Pueden ser sentencias JavaScript:
  Ejemplo:
```
<input type="button" value="Haz clic y verás" onclick="console.lo
g('Gracias por hacer clic');">
```

- Se puede utilizar la palabra reservada `this`, que hace referencia al objeto DOM del elemento al que está asociado el evento:
  Ejemplo:
```
<div onmouseover="this.style.borderColor='silver'"
onmouseout="this.style.borderColor='black'">...</div>
```

- Pueden ser llamadas a funciones:
  Ejemplo:
```
<input type="button" value="Haz clic y verás" onclick="muestraMen
saje();">

function muestraMensaje() {
  console.log('Gracias por pinchar');
}
```
  En las funciones externas no es posible utilizar la variable `this` como en el caso anterior. Por eso es necesario pasar la variable `this` como parámetro a la función manejadora.
  Ejemplo:
```
<h3 onclick="cambiar(this)">
    Pulsa aquí para ver qué lo que ocurre
</h3>
<script>
function cambiar(elem) {
    elem.innerHTML = "JavaScript en atributo HTML y función exter
na";
}
</script>
```

- Se puede evitar que se ejecute la acción por defecto devolviendo `false`:
  Ej:
```
<a href="http://www.google.com" onclick="alert('Vamos a Google');
return false;">Pulsa aquí para ver qué se ejecuta</a>
```

Evento onload

Un evento especial es el evento `onload` del elemento `body`: ocurre cuando se ha cargado todo el documento HTML.

Modelo de eventos tradicional

Este modelo se creó para poder separar el código HTML y JavaScript, pero actualmente **no se recomienda su uso**. Lo veremos para identificarlo y si fuera necesario, adaptar páginas web antiguas al siguiente modelo que veremos.

Se basa en utilizar el **DOM** (modelo de objetos del documento). Cuando se carga una página web, se crea una estructura de objetos, denominada DOM. Cada elemento HTML se representa en dicha estructura, dependiendo del tipo de elemento, tendrá unas propiedades u otras.

Algunas de las propiedades (en realidad son métodos de los objetos), permiten enlazar funciones que se disparen cuando se produzcan eventos sobre los elementos del DOM. El nombre de la propiedad es `on` + nombre del evento.

Asociar el evento a su manejador
Sintaxis:

```
document.getElementById("…").onclick = función;
```

Es importante recordar que no se escriben paréntesis después del nombre de la función: si se escribieran, se ejecutaría la función. En este modelo, las funciones que definimos para responder a los eventos, se conocen como **manejadores de eventos semánticos**.

El primer ejemplo que vimos en el modelo de eventos en línea sería del siguiente modo:

```
function muestraMensaje() {
  console.log('Gracias por hacer clic');
}
document.getElementById("boton").onclick = muestraMensaje;

<input id="boton" type="button" value="Haz clic y verás">
```

Hemos de tener en cuenta que, con este modelo, al igual que con el anterior, solo se puede asignar una función a un evento. Si se asignaran dos funciones, solo se tendrá en cuenta la última que se asigne.

Evento onload
Un inconveniente de este método es que los manejadores se asignan mediante los métodos del objeto DOM, que solamente se pueden utilizar después de que la página se ha cargado completamente. Por tanto, para que la asignación de los manejadores no resulte errónea, es necesario asegurarse de que la página ya se ha cargado. Una forma de conseguirlo es utilizar el evento onload del objeto window.

Ejemplo:

```
window.onload = function() {
    document.getElementById("boton").onclick = muestraMensaje;
}
```

Eliminar el manejador de un evento
Se puede desenlazar una función de un evento. Basta con asignar el valor `null`.
Ejemplo:

```
document.getElementById("boton").onclick = null;
```

Modelo de eventos del W3C

Este es el modelo de eventos que vamos a estudiar más en profundidad, ya que es el que **es el más recomendable, y, por lo tanto, el que debemos utilizar**.

El flujo de eventos

Antes de estudiar el modo de asignar los manejadores de eventos, tenemos que entender el concepto **flujo de eventos** (*event flow*) de los navegadores. El flujo de eventos permite que varios elementos diferentes puedan responder a un mismo evento.

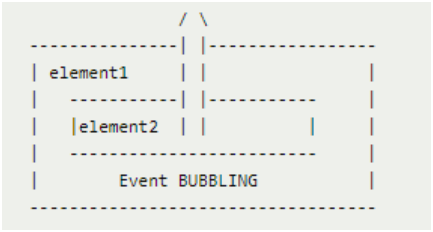
Por ejemplo: si se hace clic sobre un botón, los elementos HTML dentro de los cuales está el botón (body, div, etc.) también pueden responder al evento clic.

El orden en el que se ejecuten los eventos es lo que se conoce como flujo de eventos.

Event bubbling

Modelo desarrollado en los navegadores de Microsoft (IE).

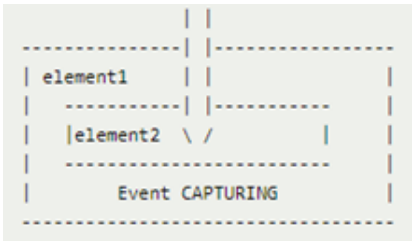
En este modelo de flujo de eventos, el orden que se sigue es desde el elemento más específico al menos específico.



Event capturing

Modelo desarrollado por Netscape.

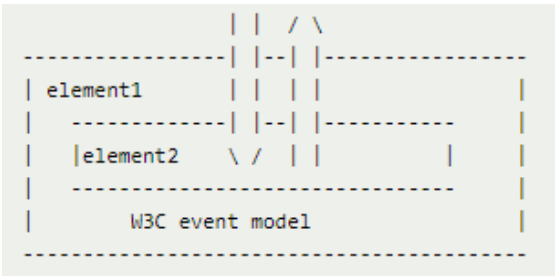
Es el mecanismo contrario al *event bubbling*: el orden es del menos específico al más específico.



Modelo del W3C

El W3C tomó una postura intermedia. Todo evento que tiene lugar en el modelo de eventos del W3C, es capturado primero hasta que alcanza el elemento destino (*capturing*), y después se transmite hacia arriba (*bubbing*).

El desarrollador web puede elegir si registrar el manejador de eventos en la fase de *capturing* o de *bubbling*.



Añadir un manejador de eventos: .addEventListener()

Sintaxis:

```
elemento.addEventListener("evento", miFunción [, useCapture]);
```

Parámetros:

- *evento*: "click", "mouseover", etc.
- *miFunción*: puede ser el nombre de una función, o una función anónima (pero no una llamada a una función).
- *useCapture*: opcional; es una variable booleana; el valor por defecto es *false*.
  - Si es true, el manejador se emplea en la fase de *capture*.
  - Si es false, se asocia a la fase de *bubbing*.

Notas:

- Es posible asignar varios manejadores de eventos distintos a un evento sobre un elemento. En este modelo se puede, en los dos modelos anteriores, no.
- Uso de **`this`** dentro del manejador de eventos: es una referencia al elemento para el cual se está ejecutando el manejador. Tiene el mismo valor que la propiedad `currentTarget` del objeto de tipo evento que se pasa al manejador.

Eliminar un manejador de eventos: .removeEventListener()

Sintaxis:

```
elemento.removeEventListener("evento", miFunción [, useCapture]);
```

Parámetros:

Los mismos que para addEventListener().

Notas:

- Si se asocia una función a un flujo de eventos, esa función solo se puede desasociar en el mismo flujo de eventos (es decir, si en el `addEventListener()` el tercer parámetro es false, en el `removeEventListener()` también deberá ser false).

Ejemplo:

```
function muestraMensaje() {
    console.log("Has pulsado el ratón");
}
let d = document.getElementById("divPrincipal");
d.addEventListener("click", muestraMensaje, false);

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

Eventos DOMContentLoaded y load

Como hemos ido viendo crear nuestros scripts, nos interesa que el código se ejecute cuando se termine de cargar la página completamente, o al menos, el árbol DOM.

Para ello, podemos utilizar el evento `DOMContentLoaded`. En `miFuncion` estará todo el código que queremos que se ejecute cuando se termine de carga el DOM.

```
document.addEventListener("DOMContentLoaded", miFuncion);
```

El evento **`DOMContentLoaded`** indica cuando finaliza la carga de la página, aunque los demás recursos (imágenes, ficheros JavaScript o CSS, etc.) no hayan cargado todavía. El evento **`load`** ocurre en cambio cuando se ha cargado la página y todos sus recursos, por lo tanto, `DOMContentLoaded` ocurre antes que `load` y es preferible utilizarlo.

Manejar la información de un evento

El objeto ***event*** se crea automáticamente cuando se produce un evento y 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.

```
document.getElementById("…").addEventListener("evento", miFuncion);

function miFuncion(e){
  // e es un objeto de tipo event
}
```

COMPATIBILIDAD con IE

En IE, *event* no se pasa como parámetro al manejador de eventos, sino que es una propiedad del objeto *window*.

Una forma de conseguir la compatibilidad en los manejadores sería:

```
document.getElementById("…").addEventListener("evento", miFuncion);

function miFuncion(e){
  let evento = e || window.event;
  //si e es null, se asigna window.event a la variable evento
}
```

Propiedades de event

Algunas Propiedades y métodos en los navegadores que siguen los estándares:

Propiedad	Tipo	Contenido
type	Cadena de texto	El nombre del evento.
timeStamp	Número	La fecha y hora en la que se ha producido el evento.
target	Element	El elemento que origina el evento.
currentTarget	Element	El elemento que es el objetivo del evento.
cancelable	Boolean	Indica si el evento se puede cancelar.
bubbles	Boolean	Indica si el evento pertenece al flujo de eventos de *bubbling*.
cancelBubble	Boolean	Indica si se ha detenido el flujo de eventos de tipo *bubbling.*

Método	Devuelve	Descripción
preventDefault()	`undefined`	Se emplea para cancelar la acción predefinida del evento.
stopPropagation()	`undefined`	Impide que se siga propagando el evento en las fases de *capturing* y *bubbling*.

Ejemplos:

1. Obtener el elemento que origina el evento:
   ```
   elEvento.target
   ```

COMPATIBILIDAD con IE
`Event.srcElement` es un alias (implementado en Internet Explorer) de Event.target, que es soportado por algunos navegadores por motivos de compatibilidad.

2. Obtener el tipo de evento:
```
elEvento.type
```

```
const procesaEvento = (elEvento)=> {
  if(elEvento.type === "click") {
    console.log("Has pulsado el ratón.");
  }
  else if(elEvento.type === "mouseover") {
    console.log("Has movido el ratón.");
  }
}

d.addEventListener("click", procesaEvento);
d.addEventListener("mouseover", procesaEvento);
```

3. Detener completamente la propagación del evento hacia arriba (*bubbling*) o hacia abajo (*capturing*):
```
elEvento.stopPropagation();
```

4. Impedir que se complete el comportamiento por defecto de un evento:
```
elEvento.preventDefault();
```

Eventos del ratón

Estos eventos están definidos para todos los elementos HTML.

Evento	Se produce
`click`	Al pulsar el botón izquierdo del ratón. También se produce cuando el foco de la aplicación está situado en un botón y se pulsa la tecla `ENTER`.
`dblclick`	Al pulsar dos veces el botón izquierdo del ratón.
`mousedown`	Al pulsar cualquier botón del ratón.
`mouseup`	Al soltar cualquier botón del ratón que haya sido pulsado.
`mouseover`	El ratón “entra” en el elemento.
`mouseout`	El ratón “sale” del elemento.
`mousemove`	Se produce (de forma continua) cuando el puntero del ratón se encuentra sobre un elemento.

Notas:

- Cuando se pulsa un botón del ratón, la secuencia de eventos que se produce es la siguiente:
  `mousedown >> mouseup >> click`
- Por tanto, la secuencia de eventos necesaria para llegar al doble clic llega a ser tan compleja como la siguiente:

```
mousedown >> mouseup >> click >> mousedown >> mouseup >> click
>> dblclick
```

Propiedades de objeto *event* específicas de los eventos de ratón

Propiedad	Tipo	Contenido
button (*)	Número entero	El botón del ratón que ha sido pulsado. Posibles valores: 0. Ningún botón pulsado 1. Se ha pulsado el botón izquierdo 2. Se ha pulsado el botón derecho 3. A la vez el botón izquierdo y el derecho 4. Se ha pulsado el botón central 5. A la vez el botón izquierdo y el central 6. A la vez el botón derecho y el central 7. Se pulsan a la vez los 3 botones
clientX	Número entero	Coordenada X de la posición del ratón respecto del área visible de la ventana.
clientY	Número entero	Coordenada Y de la posición del ratón respecto del área visible de la ventana.
detail	Número entero	El número de veces que se han pulsado los botones del ratón.
pageX	Número entero	Coordenada X de la posición del ratón respecto de la página.
pageY	Número entero	Coordenada Y de la posición del ratón respecto de la página.
screenX	Número entero	Coordenada X de la posición del ratón respecto de la pantalla completa.
screenY	Número entero	Coordenada Y de la posición del ratón respecto de la pantalla completa.

(*) `button`: solo en los eventos `mousedown`, `mousemove`, `mouseout`, `mouseover` y `mouseup`

Eventos del teclado

Estos eventos se producen sobre los elementos de formulario y <body>.

Evento	Se produce
keydown	Al pulsar cualquier tecla. También se produce de forma continua si se mantiene pulsada la tecla.
keypress	Al pulsar una tecla correspondiente a un carácter alfanumérico (no se tienen en cuenta telas como `SHIFT`, `ALT`, etc.). También se produce de forma continua si se mantiene pulsada la tecla.
keyup	Al soltar cualquier tecla pulsada.

Propiedades del objeto *event* específicas de los eventos de teclado

Propiedad	Tipo	Contenido
altKey	Boolean	*true* si se ha pulsado la tecla `ALT` y *false* en otro caso.
ctrlKey	Boolean	*true* si se ha pulsado la tecla `CTRL` y *false* en otro caso.
shiftKey	Boolean	*true* si se ha pulsado la tecla `SHIFT` y *false* en otro caso.
code	String	Código de la tecla física pulsada. Ej: "KeyY", "Key1", "NumPad1", "KeyA", "KeyA"… Ej: "AltLeft", "ShiftRight", "ArrowDown"…
key	String	Valor de la tecla pulsada. Ej. "Y", "1", "a", "A". Ej: "Alt", "Shift", "ArrowDown"…

Notas:

- 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
  - Si la tecla corresponde a un carácter alfanumérico:
    `keydown >> keypress >> keydown >> keypress >> … >> keyup`
  - En otro tipo de teclas:
    `keydown >> keydown >> … >> keyup`

Ejemplo: impedir que en el <`textarea`> se introduzca ningún carácter

```
<textarea onkeypress="return false;"></textarea>
```

Ejemplo: limitar el número de caracteres en un textarea

```
function limita(maximoCaracteres) {
  let elemento = document.getElementById("texto");
  if(elemento.value.length >= maximoCaracteres ) {
    return false;
  } else {
    return true;
  }
}
<textarea id="texto" onkeypress="return limita(100);"></textarea>
```

Eventos HTML

Evento	Descripción
load	Se produce en el objeto `window` cuando la página y todos sus recursos, se cargan por completo. En el elemento <`img`> cuando se carga por completo la imagen.
DOMContentLoaded	Se produce cuando finaliza la carga de la página, aunque los demás recursos (imágenes, ficheros JavaScript o CSS, etc.) no hayan cargado todavía, pero el árbol DOM sí.

unload	Se produce en el objeto `window` cuando la página desaparece por completo (al cerrar la ventana del navegador, por ejemplo). En el elemento `object` cuando desaparece el objeto.
abort	Se produce en un elemento `object` cuando el usuario detiene la descarga del elemento antes de que haya terminado.
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 y en el elemento `object` cuando el elemento no se carga correctamente.
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	Se produce 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.

Notas:

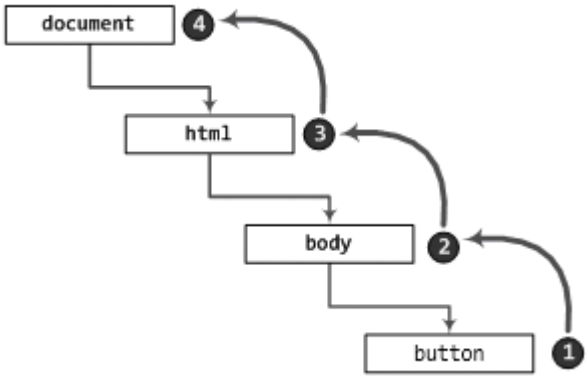
- Uno de los eventos más utilizados es el evento `DOMContentLoaded`, ya que todas las acciones que se realizan accediendo al DOM requieren que la página esté cargada por completo y, por tanto, el árbol DOM se haya construido completamente.
- Si lo que necesitamos es que no solo se cargue la página, sino que además lo hagan todos sus recursos (imágenes, ficheros JavaScript o CSS, etc.), utilizaremos `load`.
- `DOMContentLoaded` ocurre antes que `load` y es preferible utilizarlo.
- El elemento <`body`> define las propiedades `scrollLeft` y `scrollTop` que se pueden emplear junto con el evento `scroll`.

Eventos delegados: *bubbling*

El *bubbling* nos permite programar los eventos de forma delegada, lo que simplifica y clarifica mucho el código, especialmente cuando tenemos varios manejadores de eventos asociados al mismo evento, pero generados por diferentes elementos HTML.

Ejemplo: evento `click`

Cuando nosotros hacemos clic sobre un botón, se mira si hay un manejador asociado a ese elemento HTML, si hay, se atiende y finaliza, si no hay, se pasa al nivel siguiente, en este caso sería el `body`, se mira si hay un manejador asociado a `body`, si hay si atiende y si no, se pasa al siguiente nivel, así hasta llegar a `document`.
Lo que hacemos es manejar los eventos en `document` directamente, hacemos un único manejador para todos los eventos `click` y trabajamos con la información del elemento HTML.



Para saber cuál ha sido el elemento HTML de todos los posibles que pueden lanzar el evento, e invocar la acción correspondiente, tenemos el siguiente método del DOM:

Método	**Devuelve**	**Descripción**
`elemento.matches(`*`selectorCSS`*`)`	`Boolean`	Devuelve true si el elemento tiene asociado el *selectorCSS*, en caso contrario, devuelve *false*.

Ejemplo:

```
document.addEventListener("click", ev => {
    if (ev.target.matches("#n1")) vaciar();
    else if (ev.target.matches("#b1")) cuadrado();
});
```