JS: Eventos

Antonio Espín Herranz

Contenidos

- Eventos
  - Oyentes de eventos
  - Funciones de callback

Eventos

- **Eventos**: acciones que pueden ser detectadas por JavaScript, permiten interactuar al usuario con la página.
- Ejemplos:
  - Click de ratón
  - Envío de un formulario HTML
  - Pasar el ratón sobre una sección de un menú
  - Cargar una página en el Navegador.

- Cada elemento de una página Web dispone de una serie de eventos que pueden invocar funciones JavaScript
- Se insertan como atributos de tags HTML
  - Ejemplo:

```
<input type="text" size="30" id="email"
       onchange="checkEmail()">
```

Eventos

- **Eventos**: acciones que pueden ser detectadas por JavaScript, permiten interactuar al usuario con la página.
- Ejemplos:
  - Click de ratón
  - Envío de un formulario HTML
  - Pasar el ratón sobre una sección de un menú
  - Cargar una página en el Navegador.

- Cada elemento de una página Web dispone de una serie de eventos que pueden invocar funciones JavaScript
- Se insertan como atributos de tags HTML
  - Ejemplo:

```
<input type="text" size="30" id="email"
       onchange="checkEmail()">
```

Eventos

- Normalmente vamos a asignar una función a un evento.
- Estas funciones recibe el nombre de manejadores de eventos.
- También podemos incrustar código directamente en la etiqueta del evento.

```
<input type="button" id="btn" name="btn"
  value="hacer click" onclick="alert('Click')" />
```

Modelo básico de Eventos

Evento	Descripción	Elementos para los que está definido
onblur	Deseleccionar el elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onchange	Deseleccionar un elemento que se ha modificado	<input>, <select>, <textarea>
onclick	Pinchar y soltar el ratón	Todos los elementos
ondblclick	Pinchar dos veces seguidas con el ratón	Todos los elementos
onfocus	Seleccionar un elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onkeydown	Pulsar una tecla (sin soltar)	Elementos de formulario y <body>
onkeypress	Pulsar una tecla	Elementos de formulario y <body>
onkeyup	Soltar una tecla pulsada	Elementos de formulario y <body>
onload	La página se ha cargado completamente	<body>
onmousedown	Pulsar (sin soltar) un botón del ratón	Todos los elementos
onmousemove	Mover el ratón	Todos los elementos

Modelo básico de Eventos II

Evento	Descripción	Elementos para los que está definido
onmouseout	El ratón *"sale"* del elemento (pasa por encima de otro elemento)	Todos los elementos
onmouseover	El ratón *"entra"* en el elemento (pasa por encima del elemento)	Todos los elementos
onmouseup	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
onreset	Inicializar el formulario (borrar todos sus datos)	<form>
onresize	Se ha modificado el tamaño de la ventana del navegador	<body>
onselect	Seleccionar un texto	<input>, <textarea>
onsubmit	Enviar el formulario	<form>
onunload	Se abandona la página (por ejemplo al cerrar el navegador)	<body>

Manejadores de Eventos

- Los manejadores de Eventos se pueden aplicar de 3 formas:
  - Manejadores como atributos de los elementos HTML.
  - Manejadores como funciones JavaScript externas.
  - Manejadores *"semánticos"*.

  - La primera forma no es muy aconsejable (ensucia el código).

Manejadores como att de xHTML

- Se trata del método más sencillo y a la vez *menos profesional* de indicar el código JavaScript que se debe ejecutar cuando se produzca un evento.

- En este caso, el código se incluye en un atributo del propio elemento XHTML.
  - <input type="button" value="Pinchame y verás" onclick="alert('Gracias por pinchar');" />

Utilización de this

- Es interesante cuando dentro del evento necesitamos acceder a las propiedades del elemento que ha producido el evento.

- Si no utilizamos **this**, tendremos que estar capturando el control (por ejemplo mediante su id con el método getElementById(unId);

Ejemplo

```
<body>
    <div id="contenidos" style="width:150px; height:60px;
      border:thin solid silver"
      onmouseover="this.style.borderColor='black';"
      onmouseout="this.style.borderColor='silver';">
        Sección de contenidos...
    </div>
</body>
```

Manejadores como funciones externas

- El ejemplo anterior es útil cuando tenemos a lo sumo una instrucción que aplicar, de todas formas el código queda mas estructurado si lo sacamos fuera: fichero.js con sus funciones manejadoras.

- Definimos una función en js y desde el control de xHTML la llamamos en el evento correspondiente.

- Puede ocurrir que dentro de la función manejadora queramos acceder a las propiedades del control, para ello, podemos:

  – Podemos capturar el control y realizar alguna acción sobre él. Esto implica **getElementById**();

  – O podemos pasar el control a la función manejadora, haciendo uso de **this** en la llamada al evento.

Ejemplo

```
<html>

<head>
    <title>Página sin título</title>
    <script type="text/javascript">

        function mueveRaton(capa){
            capa.style.border="1px solid red";
        }

    </script>
</head>

<body>
    <div id="capa" onmouseover="mueveRaton(this)"><p>Esta es una capa de
    prueba</p></div>
    <input type="button" name="btn" value="on"
    onclick="this.value=(this.value=='on')?'off':'on'" />
</body>
</html>
```

Manejadores Semánticos

- Es un paso mas es dejar el código mas claro y consiste el definir el control como siempre pero no definir el evento.

- Mediante javascript vamos a definir la función manejadora del evento (igual que antes) y también vamos a asignar la función mediante javascript.

Ejemplo

```
<html>
<head>
	<title>Página sin título</title>
	<script type="text/javascript">

		function mueveRaton(){
			document.getElementById('capa').style.border="1px solid red";
		}

		document.getElementById("capa").onmouseover=mueveRaton;

	</script>
</head>

<body>
	<div id="capa"><p>Esta es una capa de prueba</p></div>

</body>
</html>
```

Ojo! → Sin paréntesis:

mueveRaton**()**

¿Hay mas problemas?

La forma correcta de asignar eventos

```
<html>
<head>
	<title>Página sin título</title>
	<script type="text/javascript">

		window.onload = function() {
			// Uso de funciones anónimas:
			document.getElementById("capa").onmouseover=mueveRaton;
		}


		function mueveRaton(){
			document.getElementById('capa').style.border="1px solid red";
		}

	</script>
</head>

<body>
	<div id="capa"><p>Esta es una capa de prueba</p></div>
</body>
</html>
```

Mas sobre Eventos

- Con el modelo básico es compatible con todos los navegadores, lo básico sigue el estándar de W3C.
- El problema es cuando queremos detectar la tecla pulsada, la posición del ratón, etc.
- Aquí las diferencias entre los navegadores se incrementan.
  - En **IE** podemos acceder a la información del evento mediante **window.event. En Opera también está definido.**
  - En cambio **Mozilla** pasa un objeto **event** a los manejadores de eventos.
    - Este parámetro se define en la función manejadora.

Ejemplo

```
<html>
<head>
	<title>Página sin título</title>
	<script type="text/javascript">

		window.onload = function() {
		  document.getElementById("capa").onclick=mueveRaton;
		}

		function mueveRaton(event){
		  var obj, visor;
		  // Chequear según el Navegador, siempre comprobamos si disponemos, del objeto:
		  // Igual haremos con Ajax.

			if (window.event) // IE, Opera
				obj = window.event;
			else // Mozilla
				obj = event;

			visor = document.getElementById("capa2");
			visor.innerHTML = "<h1>" + obj.clientX + ", " + obj.clientY + "</h1>";
		}
	</script>
</head>
<body>
	<div id="capa" style="background-color:#333333;height:80px"></div>
	<div id="capa2" style="border:1px solid black; height:80px; text-align:center"></div>
</body>
</html>
```

Propiedades del Evento

- Aquí sigue habiendo discrepancias cada navegador nombra de forma distinta a las propiedades de este objeto.

- En el caso de las coordenadas del ratón coinciden. clientX / clientY

Propiedades del Evento

IE	MOZILLA	DESCRIP.
altKey	altKey	Propiedad boolean que indica si la tecla alt estaba pulsada.
cancelBubble	stopPropagation	Cancela el evento por los ancestro del árbol DOM.
clientX / clientY	clientX/clientY	Coordenadas relativas al evento, dentro del área que delimita el control.
ctrlKey	ctrlkey	Idem de alt pero con la tecla de control.
fromElement	relatedTarget	Para eventos de ratón, es el elemento desde el que partió el cursor del ratón.
keyCode	keyCode	Código de la Tecla pulsada, 0 para eventos de ratón.
returnValue	preventDefault	Usado para evitar que se ejecute la acción por defecto del evento.

Propiedades del Evento

IE	MOZILLA	DESCRIP.
screenX	screenY	Posición relativa a la pantalla
shiftKey	shiftKey	Boolean que indica si la tecla Shift estaba pulsada.
srcElement	target	El elemento que provoco el evento.
toElement	currentTarget	Para eventos de ratón, es el elemento al que el ratón se dirigió cuando terminó el mismo.
type	type	Devuelve el tipo del elemento.

Métodos de Event

- Métodos para añadir / eliminar eventos en los dos navegadores.

Método de Internet Explorer	Método de Mozilla
attachEvent(tipoEvento, referenciaFuncion)	addEventListener(tipoEvento, referenciaFuncion, usarCaptura)
detachEvent(tipoEvento, referenciaFuncion)	removeEventListener(tipoEvento, referenciaFuncion, usarCaptura)

Origen de los Eventos

- En el DOM se considera que un evento se origina en el exterior de la pagina Web y se propaga de alguna manera hasta los elementos internos de la pagina.

- Un posible ejemplo de esta propagación es:
  - EVENTO → Ventana → Document → HTML → BODY → DIV → DESTINO
  - RESPUESTA → DIV → BODY → HTML → Document → Window → EVENTO

- Siguiendo esta idea, se establecen tres etapas, **captura** la cual es cuando el evento se esta trasladando a su destino. **Blanco** que es cuando llega al blanco, o sea que llega a su destino. Este destino es el objeto en el cual se va a crear una reaccion a este evento. Finalmente la etapa de **burbujeo** que es cuando el evento "regresa" a su posición original.

Métodos de Event

- Tenemos que indicar el tipo de evento y la función que se encarga de manejar ese evento.
- Además de haber métodos distintos entre IE y Mozilla a la hora de indicar el tipo evento también cambia.
  - IE → “onmouseover”, “onclick”.
  - Mozilla → “mouseover”, “click”.

Manejadores de Eventos en HTML5

- El método **addEventListener** es el estándar dentro de la especificación de HTML5.

  - **Tiene 3 atributos:**
    - Nombre del evento.
    - La función que será ejecutada.
    - Un valor boolean (falso o verdadero). Para indicar como un evento será disparado en elementos superpuestos.

Ejemplo

- Con false: El flujo de eventos es: botón → capa → body
- Con true: body → capa → botón.
- Normalmente será con el valor **false**, es el comportamiento esperado.

```
function alerta() {
        alert(this);
}

window.onload = function() {
    var elBody = document.getElementsByTagName('body')[0];
    var elDiv = document.getElementById('elDiv');
    var elButton = document.getElementById('elButton');
    elBody.addEventListener('click', alerta, false);
    elDiv.addEventListener('click', alerta, false);
    elButton.addEventListener('click', alerta, false);
}
```

Cortar la propagación

- Cuando tenemos registrados eventos en varios elementos que se contienen unos a otros y se definen eventos en todos, por ejemplo:
- **button → div → body**
- Se pueden cancelar con el objeto **Event** y el método **stopPropagation**().

Ejemplo

- En HTML

```
<body>
    <div>
        <button type="button" …>
    </div>
</body>
```

  - Definimos el evento click a nivel de los 3 elementos: body, div y button

JavaScript

```
addEventListener("load", () => {
        let capa = document.getElementById("capa");
        capa.addEventListener("click", (e) => {
          console.log("click en capa");
          e.stopPropagation();  // Corta el evento y no se propaga al body
        });

        let boton = document.getElementById("boton");
        boton.addEventListener("click", (e) => {
          console.log("click boton");
          console.log(e.target);
        });
      });

      addEventListener("click", () => {
        console.log("click en body");
});
```

Cancelar un evento

- Un evento se **puede cancelar** con el método **preventDefault** aplicado al objeto event:
    - Anular el comportamiento por defecto.

- **Ejemplo:**

```
let drop = document.getElementById("dropbox");
// Eliminar el comportamiento por defecto:
drop.addEventListener("dragenter", (e) => {
  e.preventDefault();
});
```