Objeto «event»

Generalmente, los manejadores de eventos necesitan información adicional para procesar las tareas que tienen que realizar. Si una función procesa, por ejemplo, el evento clic, lo más probable es que necesite conocer la posición en la que estaba el ratón en el momento de realizar el click; aunque esto quizás tampoco sea muy habitual, a no ser que estemos programando alguna utilidad de tipo gráfico.

Lo que sí es más común es tener información adicional en los eventos del teclado. Por ejemplo, cuando pulsamos una tecla nos interesa saber cuál ha sido la tecla pulsada, o si tenemos alguna tecla especial pulsada como Alt, Control, etc.

Para gestionar toda esa información disponemos del objeto Event, el cual nos permitirá acceder a esas propiedades adicionales que se generan en los eventos.

Como siempre, los navegadores gestionan de forma diferente los objetos event. Por ejemplo, en las versiones antiguas de Internet Explorer, el objeto event forma parte del objeto window, mientras que en otros navegadores como Firefox, Chrome, etc., se accede al objeto event a través de un parámetro, que escribiremos en la función que gestionará el evento.

```
function manejadorEventos(elEvento) {
  var evento = elEvento;
}
```

Si se quiere programar una aplicación que funcione correctamente en todos los navegadores, es necesario obtener el objeto event de forma correcta según cada navegador. El siguiente código muestra la forma correcta de obtener el objeto event en cualquier navegador:

```
function manejadorEventos(elEvento) {
  var evento = elEvento || window.event;
}
```

Una vez obtenido el objeto `event`, ya se puede acceder a toda la información relacionada con el evento, que depende del tipo de evento producido.

Principales propiedades del objeto event

altKey, ctrlKey, metaKey, shiftKey	Valor booleano que indica si están presionadas alguna de las teclas Alt, Ctrl, Meta o Shift en el momento del evento
Bubbles	Valor booleano que indica si el evento burbujea o no
Cancelable	Valor booleano que indica si el evento se puede cancelar
clientX, clientY	Devuelve las coordenadas de la posición del ratón en el momento del evento respecto a la ventana activa
offsetX, offsetY	Coordenadas del cursor dentro del elemento HTML que haya provocado el evento.
screenX, screenY	Devuelve las coordenadas del ratón relativas a la pantalla dónde se disparó el evento
currentTarget	El elemento al que se asignó el evento. Por ejemplo si tenemos un evento de click en un divA que contiene un hijo divB. Si hacemos click en divB,

	currentTarget referenciará a divA (el elemento donde se asignó el evento) mientras que target devoverá divB, el elemento donde ocurrió el evento
target	El elemento dónde se originó el evento, que puede diferir del elemento que tenga asignado el evento. Véase currentTarget.
eventPhase	Un valor integer que indica la fase del evento que está siendo procesada. Fase de captura(1), en destino(2) o fase de burbujeo(3)
relatedTarget	En un evento de "`mouseover`" indica el nodo que ha abandonado el ratón. En un evento de "`mouseout`" indica el nodo hacia el que se ha movido el ratón
timestamp	Devuelve la hora (en milisegundos desde epoch) a la que se creó el evento. Por ejemplo cuando se presionó una tecla. No todos los eventos devuelven timestamp
Type	Una cadena de texto que indica el tipo de evento "`click`", "`mouseout`","`mouseover`", etc
key	Devuelve la tecla que se ha pulsado Un solo carácter ("A", "a", "4", "+", "$") Varios caracteres ("F1", "Enter", "HOME", "BLOQ MAYÚS")
button	Devuelve que botón del ratón se ha pulsado. Normalmente se usa con el evento onmousedown (0: izquierdo, 1:medio, 2:derecho)
cancelBubble`->OBSOLETO, usar stopPropagation()`	Esta propiedad es modificable por el usuario. Su valor es `false` cuando un evento se produce, lo que significa que el ámbito es global (ver explicación más abajo). Si el usuario pone esta propiedad a `true` limita el ámbito.

Lista de métodos del objeto event

preventDefault()	Cancela cualquier acción asociada por defecto a un evento
stopPropagation()	Evita que un evento burbujee. Por ejemplo si tenemos un divA que contiene un divB hijo. Cuando asignamos un evento de click a divA, si hacemos click en divB, por defecto se dispararía también el evento en divA en la fase de burbujeo. Para evitar esto se puede llamar a stopPropagation() en divB. Para ello creamos un evento de click en divB y le hacemos stopPropagation()

Ámbito de un evento :

Cuando un evento ocurre en un elemento HTML, ocurre igualmente en los elementos que le engloban. Si - por ejemplo - un elemento `<DIV>` contiene un elemento `<IMG>`, cuando el usuario hace clic sobre la imagen, ocurre un evento `click` que puede ser

tratado a nivel del elemento <IMG> como también a nivel del <DIV> que le contiene, y, por supuesto, a nivel <BODY> que engloba a ambos.
El evento es tratado en primer lugar por el elemento donde se desencadena. Existirán, pues, - en nuestro ejemplo - los tratamientos consecutivos ligados a <IMG>, <DIV> y <BODY>. Podríamos denominar a esto como «*escalada o ámbito*» del evento. Se podría llamar también «*burbujeo*».

Ejemplo :

El procedimiento siguiente puede ser aplicado a los diversos eventos originados sobre los elementos HTML. Mostrará ciertas características del evento y permitirá verificar su ámbito.

```
<script>
function controlEvent(even)
{
   evt=window.event || even;
   var info = "";
   info += "Elemento tratado = " + evt.currentTarget + "/" +
evt.currentTarget.id + "\n";
   info += "Elemento desencadenante = ";
   info += evt.target.id + "\n";
   info += "Nombre del evento = " + evt.type + "\n";
   info += "Coordenadas del ratón : ";
   info += "pantalla=" + evt.screenX + "," + evt.screenY;
   info += " - ventana=" + evt.clientX + "," + evt.clientY;
   info += " - elemento HTML=" + evt.offsetX + "," + evt.offsetY +
"\n";
   alert(info);
}
</script>
<BODY onclick="controlEvent(event);">
<DIV id=div1 onclick="controlEvent(event);">
   <DIV id=div2 onclick="controlEvent(event);"
      style="position:absolute; left:200px; top:100px;">
      <INPUT id=boton1 onclick="controlEvent(event);"
         type=button value="Botón 1 (onclick)">
   </DIV>
   <DIV id=div3 onmousemove="controlEvent(event);"
      style="position:absolute; left:300px; top:200px;">
      <INPUT id=boton2 onmousemove="controlEvent(event);"
         type=button value="Botón 2 (onmousemove)">
   </DIV>
</DIV>
</BODY>
```

Colocando la línea inferior, en lugar de la correspondiente, en el código de aquí arriba podrás comprobar el efecto de modificar a `true` la propiedad `cancelBubble`. Se limita el ámbito a la etiqueta <INPUT>.

```
<INPUT id=boton1 onclick="event.cancelBubble=true;
controlEvent(event);"
type=button value="Botón 1 (onclick)">
```

ó usando el método **stopPropagation**:

```
<INPUT id=boton1
onclick="event.stopPropagation();controlEvent(event);"
         type=button value="Botón 1 (onclick)">
```

Eventos del teclado en JavaScript

Keyboard Events (w3schools.com)

Uno de los eventos más complicados de gestionar en JavaScript son los eventos de teclado, debido a que suele haber bastantes incompatibilidades entre navegadores, teclados, idiomas, etc.
En el proceso de pulsación de una tecla se generan tres eventos seguidos: keydown, keypress y keyup y además disponemos de dos tipos de teclas: las especiales (Shift, Alt, AltGr, Enter, etc.) y las teclas normales, que contienen letras, números, y símbolos.

Con la incorporación de la propiedad **Key** se ha simplificado su manejo, siendo la propiedad recomendada a utilizar.

Keypress sólo se produce al pulsar caracteres imprimibles(excluye Control, ALT,..) y se recomienda no utilizar, utilizar el evento onkeydown en su lugar para detectar si un usuario pulsa una tecla.

Para convertir el código de un carácter al carácter que representa la tecla que se ha pulsado, se utiliza la función **String.fromCharCode()** y para obtener el código ascii de un carácter **String.CharCodeAt()**